PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGISTER EN DATA SCIENCE

TEMA:

CARACTERIZACIÓN DE DEFUNCIONES HOSPITALARIAS POR PROBLEMAS CARDIACOS EN ECUADOR, USANDO TÉCNICAS DE CIENCIA DE DATOS

AUTOR:

JIMMY BYRON GAVILANEZ OCAMPO

DIRECTOR:

MIGUEL DIMITRI ORTIZ NAVARRETE

QUITO, MARZO 2025

DEDICATORIA

A Dios por haberme puesto la fuerza, dedicación y empeño para cumplir mis objetivos profesionales, y por permitirme hacer este sueño realidad.

A mi familia porque son el pilar fundamental y fuente de inspiración para mi superación personal.

A mis padres porque gracias a su ejemplo y enseñanza que con esfuerzo, trabajo, constancia y dedicación todo es posible.

A mis hijas Danna y Amelia por su amor y cariño y como fuente de su inspiración, para el cumplimiento de sus futuros objetivos.

AGRADECIMIENTO

A Dios por iluminarme, protegerme y centrarme todos los días de mi vida.

Agradezco a mis padres por sus consejos y no dudar de mis capacidades, e inspirarme para llegar más lejos.

Al Máster Miguel Ortiz, director de este trabajo de titulación por su apoyo, conocimientos, y su paciencia lo cual ha sido muy importante para poder culminar este proyecto.

RESUMEN

La finalidad de este proyecto es aplicar técnicas de ciencia de datos que permitan analizar el comportamiento de las defunciones hospitalarias por problemas cardiacos en Ecuador mediante el uso de series temporales y técnicas de clusterización. La información para realizar este análisis se obtuvo de los egresos hospitalarios recopilada y proporcionada por el INEC en formato csv, para la preparación de la data y elaboración de los respectivos modelos se utilizó la metodología CRISP-DM con su respectivo proceso.

Para la aplicación de las series temporales se usó el método de Holt con tendencia, el método multiplicativo de Holt Winters, ARIMA y SARIMA, además se emplea la técnica de clusterización k modes, para identificar patrones, además en series temporales se usó algunas técnicas de evaluación como el MSE, RMSE, MAE y MAPE, mientras que para la técnica de k-modes se utilizó el Silhouette Score.

Los algoritmos fueron desarrollados en Python en Jupyter Notebook, a partir de la evaluación de los modelos realizados se concluyó que el mejor método para predecir las defunciones por problemas cardiacos es el método multiplicativo de Holt Winters, además el análisis de clusterización con Silhouette Score se determinó que el mejor número de clústeres es 3 los cuales comprenden cada uno con sus características considerables como los grupos de edad, el género, y demás características demográficas. Este análisis sirve de insumo para las autoridades sanitarias del país para diseñar programas de salud pública centrados en la prevención, detección temprana y manejo de afecciones relacionadas con las defunciones por problemas cardiacos, además, las entidades encargadas de esto podrían explorar métodos como el análisis de cohortes, redes neuronales o modelos de predicción basados en aprendizaje automático, permitiendo una respuesta más precisa a las problemáticas de salud en Ecuador.

PALABRAS CLAVE: Ciencia de datos, CRISP-DM, K-Modes, series temporales, Silhouette Score, aprendizaje supervisado, aprendizaje no supervisado.

ABSTRACT

The purpose of this project is to apply data science techniques that allow analyzing the behavior of hospital deaths due to heart problems in Ecuador through the use of time series and clustering techniques. The information to carry out this analysis was obtained from hospital discharges compiled and provided by INEC in csv format, for the preparation of the data and elaboration of the respective models the CRISP-DM methodology was used with its respective process.

For the application of the time series, the Holt method with tendency, the multiplicative method of Holt Winters, ARIMA and SARIMA were used, in addition the k modes clustering technique is used, to identify patterns, in addition in time series some evaluation techniques such as MSE, RMSE, MAE and MAPE were used, while for the k-modes technique the Silhouette Score was used.

The algorithms were developed in Python in Jupyter Notebook, from the evaluation of the models carried out it was concluded that the best method to predict deaths due to heart problems is the multiplicative method of Holt Winters, in addition to the clustering analysis with Silhouette Score it was determined that the best number of clusters is 3 which comprise each one with its considerable characteristics such as age groups, gender, and other demographic characteristics. This analysis serves as an input for the country's health authorities to design public health programs focused on the prevention, early detection and management of conditions related to deaths due to heart problems, in addition, the entities in charge of this could explore methods such as cohort analysis, neural networks or prediction models based on machine learning. allowing a more precise response to health problems in Ecuador.

KEY WORDS: Data science, CRISP-DM, K-Modes, time series, Silhouette Score, supervised learning, unsupervised learning.

INDICE DE CONTENIDO

CAPÍTULO 1 INTRODUCCIÓN 1
1.2 Planteamiento del problema 2
1.3 Justificación 3
1.4 Objetivos 4
1.4.1 Objetivo General 4
1.4.2 Objetivos Específicos 4
1.5 Alcance 4
CAPÍTULO 2 MARCO TEÓRICO Y CONCEPTUAL 5
2.1 Antecedentes o marco referencial 5
2.2 Importancia de los datos de defunciones hospitalarias por problemas cardiacos 7
2.3 Sistemas de estadísticas vitales 8
2.4 Políticas Públicas de Salud 9
2.5 Ciencia de datos en la salud 10
2.6 Marco Conceptual 12
2.6.1 Machine Learning 12
Categorías de machine learning 13
2.6.2 Aprendizaje Supervisado 13
2.6.3 El aprendizaje No Supervisado 14
2.6.4 Aprendizaje por refuerzo 14
2.6.5 Clustering 14
2.6.6 K means 15
2.6.7 Clustering Jerárquico 16
2.6.8 Series de Tiempo 16
2.6.8.1 Métodos y Técnicas de Pronóstico de Series Temporales 17

2.6.8.2 Métodos Simples de Pronóstico de Series Temporales: 17
2.6.8.3 Técnicas de Suavizado Exponencial:..... 17
2.7 CRIPS DM..... 19
CAPÍTULO 3 MARCO METODOLÓGICO 22
3.1 Marco Metodológico de investigación 22
3.2 Método..... 22
3.3 Marco Metodológico de Ciencia de Datos 23
CAPÍTULO 4 RESULTADOS 24
4.1 Aplicación de métodos y técnicas de pronóstico de series de tiempo para predecir el comportamiento de las defunciones Hospitalarias por problemas cardiacos..... 24
4.1.1 Comprensión del Negocio 24
4.1.1.1 Contexto..... 24
4.1.1.2 Definición de los Objetivos del Negocio 24
4.1.1.3 Criterios de Rendimiento 25
4.1.1.4 Evaluación de la Situación..... 25
4.1.2 Determinación de los Objetivos de Minería de Datos 25
4.1.2.1 Objetivos de los Modelos de Predicción de Series de Tiempo 25
4.1.2.2 Criterios de rendimiento..... 26
4.1.2.3 Plan del proyecto..... 26
4.1.3 Compresión de los Datos 26
4.1.3.1 Recopilación de Datos Iniciales 27
4.1.3.2 Descripción de Datos 27
4.1.3.3 Exploración de datos..... 29
4.1.3.4 Verificación de la Calidad de los Datos..... 33
4.1.4 Preparación de los Datos..... 34
4.1.5 Modelado 34
4.1.5.1 Selección de Técnicas de Modelado 35

4.1.5.2 Modelado de Supuestos 35
4.1.5.3 Diseño de Comprobación 36
4.1.5.4 Generación de los Modelos 36
4.1.6 Evaluación de los Modelos 48
4.2 Aplicación de técnicas de clusterización en las defunciones Hospitalarias por problemas cardiacos 48
4.2.1 Comprensión del Negocio 48
4.2.1.1 Contexto 48
4.2.1.2 Definición de los Objetivos del Negocio 48
4.2.1.3 Criterios de Rendimiento 49
4.2.1.4 Evaluación de la Situación 49
4.2.2 Determinación de los Objetivos de Minería de Datos 49
4.2.2.1 Objetivos Aplicados a Clustering 49
4.2.2.2 Criterios de Rendimiento 49
4.2.2.1 Plan de Proyecto 49
4.2.3 Comprensión de los Datos 50
4.2.3.1 Recopilación de Datos Iniciales 50
4.2.3.2 Descripción de los Datos 50
4.2.3.3 Exploración de Datos 51
4.2.3.4 Verificación de la Calidad de los Datos 54
4.2.4 Preparación de los Datos 55
4.2.5 Modelado 59
4.2.5.1 Selección de Técnicas de Modelado 59
4.2.5.2 Modelado de supuestos 59
4.2.5.3 Diseño de Comprobación 59
4.2.5.4 Generación de Modelos 59
4.2.6 Evaluación de los Modelos 64

CAPITULO 5 ANALISIS DE RESULTADOS 67

5.1 Aplicación de métodos y técnicas de pronóstico de series de tiempo para predecir el comportamiento de las defunciones Hospitalarias por problemas cardiacos..... 67

5.1.1. Despliegue 67

5.2. Aplicación de técnicas de clusterización en las defunciones Hospitalarias por problemas cardiacos..... 69

5.2.1 Despliegue 70

CAPÍTULO 6 CONCLUSIONES Y RECOMENDACIONES 71

6.1 Conclusiones..... 71

6.2 Recomendaciones 73

INDICE DE FIGURAS

Figura 1:Metodología de aplicación de un modelo de aprendizaje automático 10
Figura 2: Proceso CRIS DM 19
Figura 3: Verificación de valores nulos..... 33
Figura 4: Código de 80/20..... 34
Figura 5: Código para eliminación de Outliers 37
Figura 6: Librerías de descomposición..... 38
Figura 7: División de la data en entrenamiento y evaluación..... 40
Figura 8: Método de suavizado exponencial de Holt con tendencia. 40
Figura 9: Método multiplicativo Holt Winter con tendencia y estacionalidad..... 42
Figura 10: Ejecución de código para crear la función para la prueba de Dickey-Fuller. 43
Figura 11: Salida de la prueba de Dickey-Fuller. 44
Figura 12: Código modelo ARIMA..... 44
Figura 13:Código de parametrización modelo SARIMA..... 45
Figura 14: Selección de mejor modelo..... 46
Figura 15: Código de ajuste al mejor modelo 47
Figura 16: Código para invertir las transformaciones, modelo SARIMA..... 47
Figura 17: Resultados de la evaluacion de las métricas de los métodos aplicados 48
Figura 18: Verificación de tipos de datos..... 51
Figura 19:Verificación de valores nulos..... 54
Figura 20: Verificación de calidad de los datos 55
Figura 21: selección de variables de interés. 55
Figura 22: Código de recodificación 56
Figura 23: Código de estructuración de fecha..... 56
Figura 24: Código para eliminar valores vacíos..... 57
Figura 25: Código de recodificación de edad..... 58
Figura 26: Código de verificación del tipo de dato 58
Figura 27: Código de modificación de tipo de dato 59
Figura 28: Librerias de KModes..... 60
Figura 29: Código de determinación del número optimo de clústeres..... 60
Figura 30: Código para clustering..... 61
Figura 31: Código para insertar etiquetas..... 61
Figura 32: Código para importar librerías..... 62

Figura 33: Código para obtener una muestra y modificar el tipo de datos..... 62
Figura 34: Código para obtener la distancia de Gower y calcular el Silhouette Score .. 62
Figura 35: Código para seleccionar nuevas variables 63
Figura 36: Código para determinar el número óptimo de clústeres 63
Figura 37: Código para transformar las variables categóricas a dummies..... 65
Figura 38: Código para inicializar y ajustar el objeto MCA y para crear las coordenadas 65
Figura 39: Código modelo Holt-Winters Multiplicativo..... 67
Figura 40: Código de ajuste del Modelo Multiplicativo 68
Figura 41: Código de predicción a 36 meses..... 68

INDICE DE GRAFICOS

Gráfico 1: Total de Defunciones por Año 29
Gráfico 2: Total de defunciones Cardiaca por Género 30
Gráfico 3: Principales Causas de Defunciones por Problemas Cardiacos..... 31
Gráfico 4: Total de defunciones Cardiacas por Entidad..... 32
Gráfico 5: Total de Defunciones Cardiacas por Grupos de Edad 32
Gráfico 6: Evolución Anual de Defunciones por Enfermedades Cardiacas..... 36
Gráfico 7: Boxplot de defunciones por problemas cardiacos..... 36
Gráfico 8:Boxplot de defunciones por problemas cardiacos sin dato atípico 37
Gráfico 9: Evolución Anual de Defunciones por Enfermedades Cardiacas sin dato atípico 38
Gráfico 10: Descomposición aditiva, causa de muerte enfermedades isquémicas del corazón..... 39
Gráfico 11: Descomposición multiplicativa, causa de muerte enfermedades isquémicas del corazón..... 39
Gráfico 12: Pronóstico exponential Smoothing Holt-Winters: 41
Gráfico 13: Pronóstico multiplicativo de Holt Winters..... 42
Gráfico 14: Pronostico ARIMA 45
Gráfico 15: Pronóstico SARIMA 47
Gráfico 16: Defunciones Cardiacas por Provincia 51
Gráfico 17: Defunciones Cardiacas por Área..... 52
Gráfico 18:Defunciones Cardiacas por subtitulo Cie10..... 53
Gráfico 19: Histograma de Edad 54
Gráfico 20: Método Elbow para determinar el número óptimo de clusters 60
Gráfico 21: Distribución de los clústeres a nivel del género del fallecido por problemas cardiacas 63
Gráfico *22*: Método Elbow para determinar el número óptimo de clústeres..... 64
Gráfico 23: Visualización de los clústeres 66
Gráfico 24: Pronóstico modelo Holt-Winters Multiplicativo, valores futuros..... 69

INDICE DE TABLAS

Tabla 1: Plan del proyecto de fin de titulación, series de tiempo 26
Tabla 2: Campos que componen la base de datos de egresos hospitalarias 27
Tabla 3: Plan del proyecto de fin titulación clustering 49

CAPÍTULO 1 INTRODUCCIÓN

El presente estudio se fundamenta en el amplio universo de datos médicos para afrontar una preocupación de vital importancia, las defunciones hospitalarias ligadas con problemas cardíacos. El auge tecnológico y la disponibilidad de datos están en constante expansión, por esta razón surge la posibilidad de explotar el poder de la ciencia de datos para transformar la comprensión y la gestión de las defunciones por problemas cardíacos.

La aparición de la ciencia de datos promete la solución de problemas con la sobrecarga de información, y con la multitud de tipologías de datos desde registros médicos electrónicos, historias clínicas digitales, información de ingresos, egresos y defunciones hospitalarias, y datos de diagnóstico por medio de imágenes incluso información genómica y factores ambientales. Al aprovechar estos lagos de datos (data lake's), se puede obtener una visión más completa de las defunciones por problemas cardíacos, identificando características, patrones, correlaciones y factores de riesgo que podrían pasar desapercibidos en investigaciones tradicionales.

Las enfermedades por problemas cardiacos presentan múltiples factores de riesgos y ciertas particularidades pueden ser controlables y otros no pueden ser controladas, los pacientes con elevado riesgo cardiovascular se identifican mediante la evaluación de las lesiones asintomáticas de los órganos, y otras afecciones que se pueden determinar mediante exámenes médicos.

Los diversos factores de riesgo más importantes de las enfermedades cardiovasculares asociadas a las defunciones por este tipo de problemas, según la Organización Panamericana de Salud (OPS) (2013) en su estudio "El consumo de tabaco y la hipertensión aumentan riesgo de muerte por enfermedad cardiovascular" menciona que son problemas alimenticios, inactividad física, adicciones de consumo de tabaco y alcohol y los efectos pueden presentarse después de cierto tiempo como manifestaciones de hipertensión, sobrepeso, y existe mayor probabilidad de producir ataque cardiaco, accidente cerebrovascular, insuficiencia cardiaca y otras complicaciones.

En este sentido, el análisis, caracterización y pronóstico de las defunciones hospitalarias por problemas cardiacos en Ecuador, son herramientas fundamentales en el sector de la salud, para la toma de decisiones informadas en base a indicadores, vinculadas al desarrollo, asignación de recursos económicos para el sistema de salud y para establecer políticas públicas en pro del desarrollo del sector salud (Giraldo et al, 2017).

1.2 Planteamiento del problema

Según la OPS (2019) menciona que, en el año 2019, 2 millones de personas murieron a causa de enfermedades por problemas cardiacos, de la misma manera la tasa de mortalidad disminuyó de 203.3 defunciones por 100.000 habitantes a 137 defunciones por 100.000 habitantes en el 2019. Las afecciones principales fueron 76.3 muertes por 100.000 habitantes, otras enfermedades circulatorias 14.8, enfermedad cardiaca hipertensiva 10.6 etc.

En el siglo XXI, las defunciones por problemas cardiacos en los países desarrollados comenzaron a disminuir, mientras que en los países en vías de desarrollo aún persiste su ascenso en el presente siglo. Este escenario propició investigaciones que exploraron los complejos procesos que incluyeron la utilización de grandes volúmenes de información, lo que permitió el desarrollo de estrategias de análisis de datos y la implementación de estrategias preventivas para toda la población.

Las defunciones hospitalarias por problemas del corazón representan un problema de salud pública importante en el Ecuador. A pesar de los avances en el diagnóstico y tratamiento, estas defunciones siguen siendo una preocupación debido a su impacto en la salud pública de la población y en los recursos de atención médica.

Según el INEC (2020) las enfermedades cardíacas son extremadamente comunes y afectan a una gran parte de la población en los países desarrollados, y en mayor medida en los países en desarrollo, como es el caso de Ecuador. Además, durante el período de 2020 ocuparon el primer lugar como causa de mortalidad general. (INEC, Registro de Defunciones Generales, 2020)

Las defunciones hospitalarias por problemas del corazón tienen comportamientos diferentes dependiendo de factores que influyen directamente por ejemplo la edad, el peso etnia, sexo, etc. Por lo tanto, surge la exigencia de usar algoritmos de ciencia de datos capaces de adaptarse a la necesidad de caracterizar comportamientos dependiendo de las variables de análisis y clústeres ya formados de estas.

Actualmente, es necesario tener datos actualizados que caractericen las defunciones por problemas isquémicos cardiacos u otros que afecten directamente al corazón dependiendo de la diversidad de variables de estudio y análisis.

1.3 Justificación

Las defunciones por enfermedades cardiacas constituyen problemáticas de salud pública en la sociedad moderna, según el INEC (2022) menciona que es la principal causa de muerte en Ecuador con un 15% siendo el valor más alto de mortalidad en el año 2022.

Dado el alto porcentaje del perfil de mortalidad de las defunciones hospitalarias por problemas cardíacos en Ecuador es relevante el desarrollo de la investigación, partiendo desde el estudio de la afección (CIE10), diagnóstico apropiado, el manejo de datos hospitalarios y el correcto uso de los mismos, existe la necesidad de utilizar algoritmos de machine learning capaces de identificar y caracterizar el comportamiento con un enfoque en la prevención e identificación temprana de las enfermedades cardíacas.

La utilización y elección de métodos y técnicas de machine learning posibilitará la creación de modelos establecidos en datos previos, con el propósito de pronosticar la tendencia de las defunciones por problemas cardíacos, comprender sus factores causales e identificar tendencias significativas, esta información es esencial en la actualidad para la elaboración de políticas públicas y programas de salud más eficaces, adaptados a las demandas específicas de la población.

La predicción mediante modelos se ha vuelto un campo de estudio crucial, proporcionando fundamentos para la formulación de políticas públicas dirigidas al sistema de salud. Estos modelos posibilitan la creación de herramientas analíticas y de seguimiento de las tendencias de mortalidad (OPS, 2025). El análisis cuantitativo de las defunciones por problemas cardíacos y sus determinantes juega un papel esencial en la evaluación del impacto en las condiciones sanitarias a nivel nacional (Castañeda & González, 2014).

El resultado de este estudio beneficiará al sector salud de Ecuador por medio de la aplicación de las técnicas de ciencias de datos permiten visibilizar la disminución o incremento de las defunciones hospitalarias por problemas cardíacos y exponen escenarios que pueden desencadenarse a corto o largo plazo. Adicionalmente se logra entender riesgos de los problemas cardíacos, y entender su comportamiento de manera que se pueda tener información disponible para preparar procedimientos de desarrollo de políticas públicas y programas de salud más efectivos, que reflejan la disminución de esta problemática en la población.

1.4 Objetivos

1.4.1 Objetivo General

Analizar el comportamiento de las defunciones hospitalarias por problemas cardiacos en Ecuador, mediante el uso de técnicas de ciencia de datos.

1.4.2 Objetivos Específicos

- Analizar la evolución y las causas de defunciones Hospitalarias por problemas cardiacos en Ecuador.
- Aplicar métodos y técnicas de pronóstico de series de tiempo para determinar cuál es el más efectivo para predecir el comportamiento de las defunciones Hospitalarias por problemas cardiacos en Ecuador.
- Caracterizar los principales clústeres determinantes de defunciones Hospitalarias por problemas cardiacos en Ecuador, usando técnicas aprendizaje no supervisado.

1.5 Alcance

Para este estudio se dispone de los datos de egresos hospitalarios, correspondientes a los años 2010 hasta el 2023. Los diagnósticos médicos están catalogados con código CIE-10 lo que corresponde a la Clasificación Internacional de Enfermedades 10ª Edición, de las cuales se seleccionó solo las correspondientes a defunciones por problemas cardiacos además mediante el uso de técnicas de machine learning se utilizará métodos de aprendizaje supervisado y no supervisado para identificar patrones en los datos de defunción hospitalaria con el propósito de seleccionar el método más efectivo para predecir, además se usó CRIS DM para llevar a cabo este análisis en todas sus fases que envuelven desde el entendimiento del problema, entendimiento de los datos, preparación de los datos, modelado, evaluación y despliegue, consecutivamente, se analizarán los resultados obtenidos y se formularán conclusiones y recomendaciones basadas en estos hallazgos del análisis.

CAPÍTULO 2 MARCO TEÓRICO Y CONCEPTUAL

2.1 Antecedentes o marco referencial

Según estudios nacionales eh internacionales mencionan que las enfermedades relacionadas con problemas del corazón son la principal causa de mortalidad en el mundo, por esta razón es relevante contar con estudios que mencionan las caracterizaciones de los pacientes fallecidos para plantear a la población tratamientos y cuidados especiales, especialmente en pacientes de alto riesgo de enfermedades por problemas cardiovasculares.

En este sentido, Gonzales (2018) menciona que para seguir reduciendo la mortalidad por problemas cardiovasculares es necesario centrarse en diversos factores, así como en el desarrollo de la tecnología médica y la transición de las poblaciones urbanas a estilos de vida menos arriesgados. La expansión de los servicios de seguro médico es un factor importante, ya que los países desarrollados están aumentando la cobertura mientras que los servicios de seguro médico se están estableciendo y experimentando descensos (Levy 1984). Dado que las tecnologías desarrolladas en aquella época están disponibles hoy en nuestro país, la principal recomendación de este estudio es que, además de las campañas dirigidas a cambiar los hábitos alimentarios y otros comportamientos de riesgo, también se preste atención a las infraestructuras urbanas que afectan al atractivo. Los paseos por la ciudad y los centros sanitarios de atención terciaria, así como la vigilancia del medio ambiente, serán de gran ayuda.

Uno de los retos fundamentales a los que se enfrentan los analistas del sector sanitario es el procesamiento de grandes cantidades de información procedente de los registros de mortalidad. La comprensión de estos datos contribuye directamente a la resolución y evaluación de programas sanitarios, la realización de estimaciones y previsiones, el desarrollo de intervenciones orientadas a políticas de salud pública, la aplicación de diversas técnicas de aprendizaje automático y series temporales que producen indicadores precisos. (Wang et al., 2022)

Según la Organización Panamericana de la Salud (OPS, 2023) en marzo de 2020 se realizó en Ecuador un estudio dirigido a personas con riesgo de enfermedad cardiovascular. El estudio recogió datos de 2.231 personas de entre 18 y 69 años, cuyo

análisis mostró que el 30% de los adultos de entre 40 y 69 años corrían riesgo de sufrir enfermedades cardiovasculares.

Son diversas las características manifiestas en las defunciones por problemas del corazón, en este sentido Salinas (2022) menciona que los pacientes con presencia de afecciones iniciales como Covid son la primera causa de defunciones por problemas cardiacos en el año 2022.

Según Mora (2022) en su investigación Proyecciones de la ciencia de datos en la cirugía cardíaca, menciona que la pandemia de Covid 19 ha puesto de manifiesto varios aspectos positivos y negativos del análisis y el procesamiento de grandes volúmenes de datos, y es muy amplia la necesidad de desarrollar métricas y análisis de datos en tiempo real o en tiempo mínimo en el sector de la salud, la aplicación del aprendizaje automático en la salud es muy grande, pero aún queda un gran vacío por explorar, analizar y descubrir, en este estudio, se utilizaron algoritmos de agrupación no supervisados en un conjunto de 656 pacientes quirúrgicos, con la ayuda de la IA. Es asombroso cómo, a partir de las ideas sencillas tomando como referencia las formuladas por Alan Turing para resolver problemas complejos en la primera mitad del siglo XX, hemos evolucionado hasta las poderosas y sofisticadas herramientas tecnológicas que utilizamos hoy en día, estas herramientas no sólo son omnipresentes, sino que ofrecen soluciones que repercuten en todos los ámbitos de la sociedad, siendo la medicina en general y la cirugía cardiaca en particular las áreas más demandantes.

Según Bierrenbach et all (2019) en su estudio Redistribución de las muertes por insuficiencia cardíaca mediante dos métodos: vinculación de los registros hospitalarios con los datos de los certificados de defunción y los datos de múltiples causas de muerte, menciona que se emplearon dos enfoques de reasignación en los datos de Brasil, entre 2008 y 2012, para individuos de 55 años en adelante, en el método de causas múltiples de fallecimiento, las defunciones por insuficiencia cardíaca se redistribuyeron según la proporción de causas implícitas halladas en muertes relacionadas que presentaban la insuficiencia cardíaca como causa intermedia, desde otra perspectiva, en el método de datos de hospitalización, las defunciones por insuficiencia cardíaca se reasignaron basándose en la información del registro de hospitalización de los fallecidos, existió 123,269 (3,7%) muertes por insuficiencia cardíaca, y el método de causas múltiples de fallecimiento redistribuyó el 25,3% a enfermedades cardíacas y renales hipertensivas, el

22,6% a enfermedades coronarias y el 9,6% a diabetes, al contrario un total de 41,324 muertes por insuficiencia cardíaca se asociaron con los registros de hospitalización y en el 45,8% de los registros de hospitalización correspondientes, la insuficiencia cardíaca fue listada como diagnóstico principal y no existió redistribución para este grupo, para lo cual el método de datos de hospitalización redistribuyó el 21,2% al grupo con otras enfermedades no cardíacas, el 6,5% a infecciones de las vías respiratorias inferiores y el 9,3% a otros códigos.

Según el estudio de Lopez (2022) realizado en la parroquia Izamba, provincia de Tungurahua, Ecuador, se han identificado varios factores de riesgo cardiovascular en adultos jóvenes, entre ellos factores no modificables como el sexo masculino y los antecedentes familiares de enfermedad cardiovascular precoz, así como factores modificables directos como la hipertensión, la diabetes mellitus de tipo 2 y el hipercolesterolemia, e indirectos como la obesidad y el sedentarismo. El estudio subraya la importancia de aplicar estrategias de intervención para modificar estos factores y prevenir las enfermedades cardiovasculares en la población joven.

2.2 Importancia de los datos de defunciones hospitalarias por problemas cardiacos

Según un Feitosa 2020, se observó una disminución significativa del 2,3% en la tasa de mortalidad por insuficiencia cardíaca en Brasil durante un período de 21 años, el análisis abarcó 1.242.014 defunciones notificadas por insuficiencia cardíaca en el país. Sin embargo, al considerar las dimensiones continentales de Brasil, es necesario investigar estas tendencias en áreas regionales para obtener una comprensión más detallada de la distribución espacial de la mortalidad por insuficiencia cardíaca.

Según la OPS (2020) el comportamiento de enfermedades cardiovasculares en todo el mundo está aumentando en los países en vías de desarrollo, causando más de 16 millones de muertes, el 80% de las cuales se origina en países con ingresos bajos y medios, la actividad física constante reduce el riesgo de sufrir mortalidad cardiovascular, además representa el 1% y 3% de los costos sanitarios. En este sentido según el INEC en el año 2017 las enfermedades isquémicas del corazón son la principal causa de muerte, con 7.404.

Según Núñez (2018) en su estudio Mortalidad por enfermedades isquémicas del corazón en Ecuador, 2001-2016: estudio de tendencias, las enfermedades isquémicas del corazón (EIC) son la principal causa de mortalidad y de años de vida potencialmente perdidos

(AVPP) en países desarrollados y en desarrollo, afectando por igual a hombres y mujeres, en el año 2016, se registraron a nivel mundial 9,48 millones de muertes debido a EIC, lo que constituyó más del 85,1% de todas las muertes por enfermedades cardiovasculares, junto con las enfermedades cerebrovasculares. En países como Colombia, Chile, Venezuela, Argentina, Bolivia, Brasil y Paraguay, las EIC ocupan el primer lugar como causa de mortalidad, reportando un porcentaje de 21,6%, 20,7%, 44,5%, 7,0%, 43%, 22,6%, 48,3%, respectivamente para el período 2005-2016, en Ecuador, entre 2001 y 2016, las tasas de mortalidad por enfermedades isquémicas del corazón (EIC) mostraron una tendencia ascendente. El análisis realizado de regresión reveló dos periodos importantes: un primer periodo con una tendencia a la disminución 2001-2012, seguido por un periodo notable de aumento 2012-2016.

Los datos de defunciones hospitalarias proporcionan una visión clara del impacto de las enfermedades cardíacas, permitiendo evaluar la carga de enfermedad y detectar tendencias en la mortalidad, asimismo los datos también facilitan la planificación de recursos hospitalarios y el ajuste de estrategias de salud basadas en evidencias actualizadas, lo que contribuye a una respuesta más eficaz frente a problemas cardíacos, en fin, estos datos son básicos para mejorar la atención sanitaria, optimizar políticas y reducir las defunciones relacionadas con enfermedades por problemas cardiacos. (Lozano et all. 2023)

2.3 Sistemas de estadísticas vitales

La principal fuente de información son los sistemas de salud de camas y egresos hospitalarios recopiladas por el INEC esta entidad realiza un análisis y procesamiento de los egresos hospitalarios incluyendo altas y defunciones, en todas las instituciones de salud, tanto públicas como privadas, a nivel nacional. Las estadísticas de salud representan la única fuente confiable de datos sobre morbilidad y mortalidad, y tienen la ventaja de ser diagnósticos verificados, ya que todos los casos son evaluados por profesionales médicos y profesionales estadísticos certificados en la CIE10 quedan documentados en las historias clínicas. (Brito, 2024)

Las entidades responsables de entregar la información de egresos hospitalarios al INEC pertenecen específicamente al Ministerio de salud pública, IESS, ISSFA, ISSPOL y de la red privada complementaria, y esta información debe cumplir con las siguientes

características de calidad, debe ser precisa, exhaustiva, oportuna y continua sobre los hechos vitales. (INEC, 2024)

En resumen, las leyes y regulaciones en varios países del mundo aseguran que las enfermedades se registren a través de informes estadísticos de salud emitidos por profesionales médicos, que, además del diagnóstico de la enfermedad, brindan información adicional sobre la causa de la muerte y brindan información muy relevante de egresos hospitalarios (INEC, 2024).

2.4 Políticas Públicas de Salud

Los datos de egresos hospitalarios se han convertido en una herramienta crucial para evaluar la salud de la población, y más aún los datos de defunciones hospitalarias por problemas cardiacos, entender la calidad de vida de las personas y las condiciones de acceso a los servicios de salud, estos datos son fundamentales para desarrollar políticas que aborden las necesidades de la comunidad y apunten a disminuir las desigualdades en el acceso a servicios de salud pública (Organización Panamericana de la Salud, 2017).

Cuando las enfermedades por problemas cardíacos empezaron a convertirse en una causa significativa del universo de defunciones, las autoridades del campo salud de los países más desarrollados, preocupadas por los altos costos asociados a esta problemática, decidieron implementar una serie de estrategias mediante campañas informativas con el objetivo de reducir el consumo excesivo de alimentos sobre todo las comidas chatarra y, de esta manera, disminuir la prevalencia de estos trastornos.

Según el estudio realizado por Aguilera, (2017) concluye que el análisis de la variabilidad genética ofrecerá información valiosa sobre los mecanismos moleculares que influyen en cómo diferentes individuos responden a una misma dieta, adicionalmente se podría desarrollar una herramienta preventiva eficaz, es decir, una serie de marcadores genéticos que permitirían anticipar el éxito o el fracaso de una intervención dietética en función de las características genéticas de cada persona. Así, no hay una dieta universal, es decir esta debe ajustarse a las particularidades individuales y a los resultados que se desean obtener, la información de las defunciones cardiacas sirve de base para el desarrollo de políticas públicas que permitan abordar las necesidades en el sector de la salud, siendo el Estado, el actor principal y gestor de las políticas públicas de salud, que ayuden a prevenir este tipo de enfermedades.

2.5 Ciencia de datos en la salud

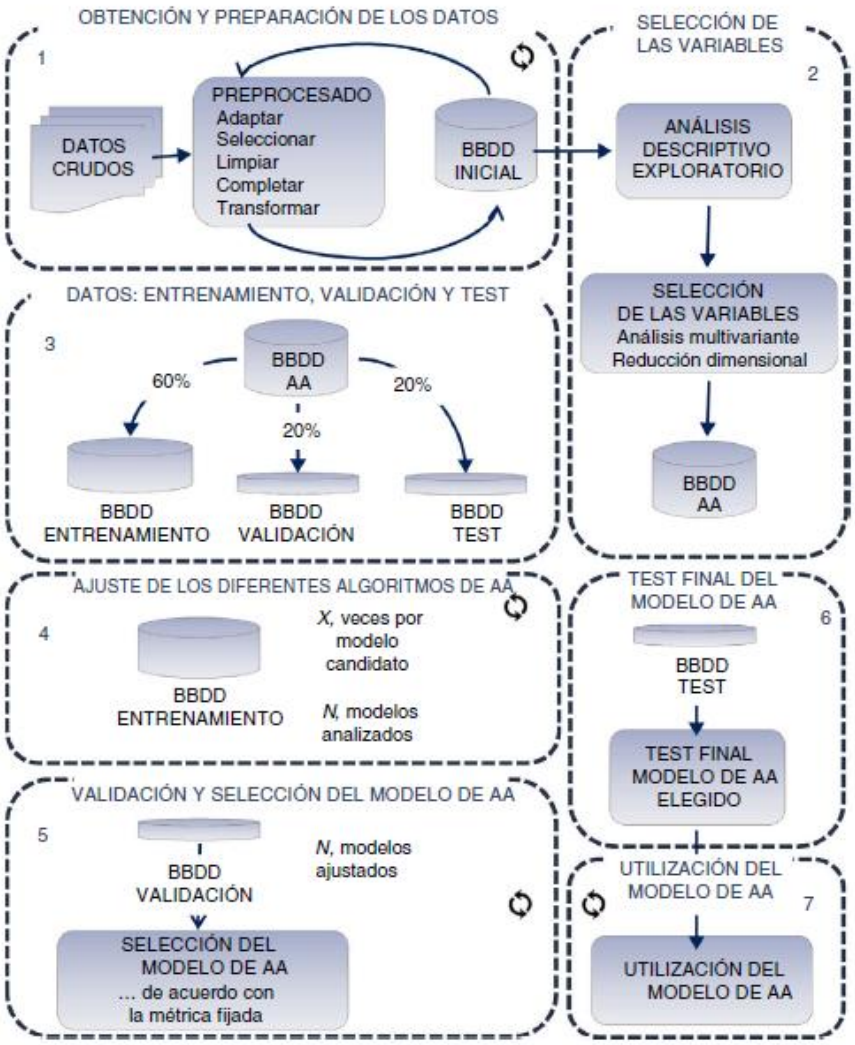
Con el avance de las tecnologías de la información y la comunicación (TIC), se están generando y almacenando constantemente datos en diversas áreas de la sociedad, en este contexto según Rodriguez (2025) en su investigación "La Revolución del Big Data: Impacto y Tendencias en el Mundo Actual" concluye que esta propensión ha dado lugar al término Big Data (BD), que se refiere a volúmenes de datos inmensos y complejos que no pueden ser gestionados de manera efectiva mediante las técnicas, tecnologías y herramientas cotidianas, sin embargo, el tamaño por sí solo no define el BD; también son importantes la velocidad, la heterogeneidad y la variedad de los datos.

Los términos ciencia de datos, Big Data, inteligencia artificial (IA) y algoritmos avanzados (AA), aunque pertenecen a la misma área de estudio relacionada con el análisis y extracción de información a partir de datos, tienen significados distintos, a pesar de esto, en ocasiones estos conceptos se usan de manera confusa, en este contexto el termino Big Data fue introducido por primera vez en el año 2005 por R. Magoulas, quien lo definió como un volumen extremadamente grande de datos que supera las capacidades de los sistemas tradicionales de almacenamiento y procesamiento. Según IBM (2024) Big Data se centra en siete características principales: volumen, velocidad, variedad, veracidad, validez, volatilidad y valor de los datos también conocidas como las 7 v del Big Data. En otras palabras, Big Data se refiere a la generación de información a alta velocidad, garantizando la integridad de los datos y manejando una amplia gama de tipos y formatos de información, como texto, imágenes, vídeos, provenientes de distintos sistemas y proveedores. (IBM, 2024)

Según Dorado-Díaz et al. (2019) en su investigación "Aplicaciones de la inteligencia artificial en cardiología: el futuro ya está aquí" menciona que el campo de la cardiología, la aplicación de Big Data puede ser eficaz en colaboraciones a nivel nacional, europeo o mundial, en las que las instituciones hospitalarias pongan en común y compartan datos para crear grandes repositorios. Los grandes repositorios serán de mucha ayuda para la práctica clínica diaria, facilitando la creación de protocolos, el diagnóstico estandarizado y precoz de enfermedades, la aplicación de técnicas para realizar predicciones de la progresión de las afecciones y la planificación del proceso de tratamiento de los pacientes. La ciencia de los datos es un campo que abarca muchos aspectos de la extracción de conocimientos a partir de los datos, lo que significa que utiliza métodos, procesos y

sistemas científicos para comprender mejor los datos, a menudo mediante la aplicación de técnicas innovadoras como la inteligencia artificial (IA). Aunque IA y aprendizaje automático (AA) se utilizan a menudo indistintamente, no son sinónimos. La Inteligencia Artificial es un término muy extenso dentro de esta sublevación tecnológica que forman parte tanto el aprendizaje automático como la programación automatizada (PA).

Figura 1:Metodología de aplicación de un modelo de aprendizaje automático



Fuente: (Dorado-Díaz et al, 2019)

En el panel derecho de la Figura 1 se muestra el interés creciente en estos conceptos a lo largo de los últimos tres años, con un interés cada vez mayor en el aprendizaje automático, la IA, la ciencia de datos y la programación automatizada. Según Dorado-Díaz (2019) en su estudio Aplicaciones de la inteligencia artificial en cardiología menciona que: En España, el sur de Europa y América del Sur, el término big data sigue siendo el más relevante; en Estados Unidos, Canadá, el Reino Unido, el norte de Europa y Australia,

machine learning (aprendizaje automático) es el más destacado; mientras que, en China y Japón, el término más relevante es deep learning (aprendizaje profundo).

En la atención en los sistemas de salud, la evolución de las TIC desde un enfoque centrado en las afecciones a otro centrado en el paciente ha dado lugar a grandes cantidades de información. Además de las fuentes tradicionales, como la historia clínica electrónica de la HCU, los historiales médicos personales y las investigaciones auxiliares, como las pruebas de imagen y de laboratorio, se han introducido nuevas tecnologías que aportan datos adicionales. Estas nuevas tecnologías de la información abarcan, entre otras, las ciencias de procesos, la telemedicina, los biosensores y los rastreadores de actividad física. En este contexto, el término informática y analíticas sanitarias se utiliza para describir las TIC y los análisis de datos necesarios para utilizar eficazmente esta información en la atención sanitaria. (Rosa & Frutos, 2022)

La ciencia de datos cobra cada vez más relevancia cuando se trata de procesar y hacer comprensible este amplio y complejo data lake o conjunto de información, a un nivel básico además puede definirse como un campo interdisciplinario emergente, dentro de los cuales está formado por la estadística, la informática, la computación, la comunicación, la gestión y la sociología para analizar datos y su contexto. Este análisis permite convertir los datos en conocimientos, decisiones y acciones mediante una metodología específica y un enfoque determinado.

2.6 Marco Conceptual

2.6.1 Machine Learning

El machine learning, es el campo de estudio que otorga a las computadoras la capacidad de aprender sin necesidad de programación manifiesta, es decir los sistemas aprenden a partir de determinados datos u información de entrenamiento para automatizar la creación de modelos analíticos y resolver tareas y su propósito es enseñar a las máquinas a manejar datos de manera eficiente (Mahesh, 2018; Janiesch et al., 2021). Esta tecnología es crucial en el ámbito de la ciencia de datos, ya que, mediante métodos estadísticos, los algoritmos se entrenan para realizar clasificaciones o predicciones, así como para descubrir información relevante en proyectos de minería de datos (IBM, s.f.). En fin, el machine learning tiene como objetivo lograr que las máquinas o sistemas aprendan de manera autónoma, y este proceso puede clasificarse en tres tipos principales de algoritmos, que se describen a continuación.

El machine learning es esencial en el área de la ciencia de datos debido a que los algoritmos se entrenan para llevar a cabo tareas de clasificación o predicción, así como para identificar información valiosa en proyectos de minería de datos (IBM, s.f.). En otras palabras, el aprendizaje automático busca permitir que las máquinas o sistemas aprendan de manera autónoma.

En este proyecto, es fundamental entender y comprender la aplicación del machine learning en el campo de la salud, ya que este enfoque puede aportar múltiples beneficios, ya que busca cómo la tecnología puede mejorar los procesos y resultados en el ámbito sanitario, especialmente en áreas como la predicción de enfermedades y la personalización de tratamientos, es relevante conocer los tres tipos de algoritmos de machine learning que se emplearán en el estudio, los cuales se detallan a continuación, para comprender su funcionamiento y cómo cada uno contribuirá al desarrollo de este trabajo de titulación.

Categorías de machine learning

Según para qué sirven y cómo se pueden usar, los algoritmos de Machine Learning se clasifican en 3 tipos, siendo los dos primeros los más usuales: aprendizaje supervisado, aprendizaje no supervisado, y finalmente, el aprendizaje por refuerzo.

2.6.2 Aprendizaje Supervisado

Su característica principal es el uso de un conjunto de datos de entrenamiento que contiene entradas de datos etiquetadas, es decir se distinguen dos tipos de problemas, la regresión, donde se predice un valor numérico, y la clasificación, donde la predicción resulta en una afiliación a una clase categórica (Janiesch et al., 2021).

Entre las formas más conocidas del aprendizaje supervisado está la regresión lineal, la regresión logística y diversos algoritmos para aprender por sí solos, como los árboles de decisión y las máquinas de vectores de soporte (IBM - United States, 2025).

Para este proyecto, es relevante tener en cuenta las técnicas de aprendizaje supervisado que se puedan utilizar, dependiendo de la calidad y disponibilidad de los datos e información disponible como la aplicación de estas técnicas en el análisis de defunciones por problemas cardiacos, permitirá identificar patrones y factores de riesgo relevantes, lo que contribuirá significativamente a la caracterización precisa de este tipo de defunciones, esta metodología no solo mejorará la comprensión de las causas implícitas, sino que también facilitará herramientas para desarrollar predicciones más efectivas y estrategias

de prevención dentro del ámbito de la enfermedades por problemas cardiacos.

2.6.3 El aprendizaje No Supervisado

Se utiliza cuando los datos no están etiquetados, por lo que se ordena reconociendo similitudes o diferencias, es decir los algoritmos buscan relaciones naturales y agrupaciones sin necesidad de obtener una respuesta específica, algunos ejemplos de estos algoritmos son k-means y agrupamiento de maximización de expectativas o de máxima verosimilitud. (IBM - United States, 2025)

Es primordial para este proyecto comprender el aprendizaje no supervisado, ya que, debido a la naturaleza de los datos, se emplearán técnicas propias de este enfoque, estas técnicas son particularmente útiles para analizar el comportamiento de las defunciones relacionadas con problemas cardiacos, especialmente porque la mayoría de los datos disponibles son de tipo categórico y la aplicación de estas metodologías permitirá identificar patrones ocultos y agrupaciones relevantes dentro de los datos, lo que contribuirá a una caracterización más precisa y detallada de las defunciones cardiacas en este trabajo de titulación.

2.6.4 Aprendizaje por refuerzo

En contraste con el aprendizaje supervisado y no supervisado, este algoritmo se basa en recompensas y penalizaciones durante el proceso de resolución del problema de investigación, su enfoque consiste en desarrollar la estrategia óptima para maximizar el rendimiento a lo largo del tiempo.

Es importante para este proyecto evaluar la viabilidad de aplicar el aprendizaje por refuerzo, un enfoque en el que un agente autónomo aprende a realizar una tarea mediante prueba y error, sin la necesidad de guía directa por parte de una persona, en el contexto de este trabajo de titulación sobre la caracterización de defunciones cardiacas, la aplicación del aprendizaje por refuerzo podría resultar provechoso para optimizar procesos de diagnóstico y predicción, permitiendo que el sistema mejore sus resultados a medida que interactúa con los datos y ajusta sus decisiones con base en la retroalimentación obtenida, y hay la posibilidad de que esto ayude a identificar patrones o relaciones complejas en los datos de manera más eficaz.

2.6.5 Clustering

Según Niansheng Tang (2022) menciona que la agrupación es un proceso cuyo objetivo es dividir los datos en grupos similares, para medir la similitud de los datos se utilizan

diferentes formas de distancias como la distancia euclidiana, la distancia de Manhattan, Mahalanobis y así sucesivamente. La representación de los datos mediante un conjunto de conglomerados da lugar a la pérdida de características, pero se consigue una simplificación. En resumen, esta técnica desempeña un papel muy importante en el proceso de minería de datos y es responsable del análisis exploratorio de datos, la recuperación de información y las aplicaciones de minería de textos en el diagnóstico médico, etc. (Tang, 2022)

Para este proyecto, es importante conocer los métodos de clustering porque permiten agrupar datos similares para facilitar la caracterización de las muertes por cardiopatías. Mediante la aplicación de estos métodos, se pueden identificar patrones y grupos de muertes con características comunes, lo que permite una comprensión más clara de los factores asociados a estas muertes y, en este sentido, la agrupación ayudará a segmentar eficazmente los datos, mejorando la capacidad de analizar y comprender las causas subyacentes de las muertes por cardiopatías.

2.6.6 K means

Según Marrero et al (2021) menciona que el método k-means tiene como objetivo principal la agrupación de elementos en función de las propiedades que comparten, es decir, los grupos formados tienen un alto nivel de homogeneidad interna y marcadas diferencias con otros grupos, en este contexto, al crear agrupaciones homogéneas facilita la descripción de taxonomías, la reducción de la complejidad de los datos y la identificación de relaciones este método se utiliza más comúnmente como procedimiento de partición en grupos en función de la media o promedio ponderado de sus puntos es decir, su centro de gravedad, que sólo puede aplicarse a datos numéricos.

En el contexto de esta tesis sobre la caracterización de las muertes cardíacas, se utilizará K-means para agrupar las muertes con características similares, como factores de riesgo, edad, sexo y otras variables relevantes. De este modo, se identificarán patrones, tendencias y comportamientos comunes en las muertes por cardiopatías, lo que ayudará a segmentar los datos de forma eficaz, proporcionando una imagen clara de los grupos de pacientes más vulnerables. La aplicación de K-means beneficiará al proyecto al identificar subgrupos específicos en las muertes por cardiopatías, lo que permitirá mejorar las estrategias de prevención y tratamiento de estas enfermedades.

2.6.7 Clustering Jerárquico

Es un método de data mining para agrupar datos, este procedimiento identifica grupos respectivamente homogéneos de casos, basándose en las características de análisis del problema de investigación, por medio de un algoritmo que inicia con cada variable en un clúster diferente y combinándolos hasta dejarlos solo uno, es importante analizar las variables ordinarias o elegir diversas estandarizaciones, las medidas de distancias y similaridades se crean mediante las proximidades.

Es fundamental para este proyecto entender cómo se aplica el Clustering Jerárquico, una técnica de clusterización que organiza los datos en una estructura jerárquica, permitiendo formar grupos a diferentes niveles de granularidad, en este contexto la caracterización de defunciones cardiacas, el Clustering Jerárquico se utiliza para identificar relaciones entre variables y agrupaciones de defunciones por problemas cardiacos según diversas características, como factores de riesgo, condiciones de salud preexistentes, y datos demográficos, esta técnica admitirá la creación de una jerarquía de clústeres que proporciona una visión más minuciosa de cómo se relacionan los diversos grupos, lo que facilita el análisis de tendencias y la identificación de subgrupos relevantes, la utilización de esta técnica beneficiará el proyecto al proporcionar una estructura flexible para explorar las variaciones en las defunciones cardiacas, mejorando la precisión en la identificación de factores clave asociados a estas.

2.6.8 Series de Tiempo

Una serie de tiempo es una secuencia ordenada de datos recopilados a lo largo de un periodo de tiempo (Daza & García, 2021), el análisis de series temporales tiene como principal objetivo explicar la variabilidad observada en sucesos pasadas, identificando patrones específicos para pronosticar el comportamiento futuro de una variable. Según Perez (2022), este proceso involucra la identificación de ciertos componentes como la tendencia, la estacionalidad y el ruido, y la aplicación de modelos estadísticos para realizar diversas predicciones basadas en datos históricos. (Perez, 2022)

Una serie temporal se puede caracterizar de acuerdo con sus componentes:

a. Tendencia: Está formado por el componente de largo plazo que establece la base de crecimiento o decrecimiento de la serie, en caso de que la serie sea estacionaria, su media y varianza permanecen estables es decir constantes.

b. Estacionalidad: Hace referencia al comportamiento de una serie en un período específico, además pueden presentar patrones que se repiten de un período a otro.

c. Ciclos: Hacen referencia a desviaciones de las tendencias subyacente causada por diversos factores generalmente externos, los cuales son distintos de la estacionalidad y el tiempo y duración de los ciclos no necesariamente es constante.

d. Aleatoriedad: Hace referencia a las fluctuaciones impredecibles o no periódicas que se ocultan en los datos, las mismas variaciones no siguen un patrón específico y son el resultado de factores fortuitos o aleatorios. (Codificandobits, 2025)

2.6.8.1 Métodos y Técnicas de Pronóstico de Series Temporales

Los métodos de proyección histórica o de series de tiempo utilizan un análisis minucioso de los patrones en los datos a lo largo del tiempo, extrapolando estos patrones hacia el futuro, este proceso se objeta únicamente en los valores pasados de la variable objetivo (Barría, 2022).

Existen diferentes métodos que se utilizan en el análisis de series temporales, que comprenden desde enfoques muy simples hasta enfoques más complejos.

2.6.8.2 Métodos Simples de Pronóstico de Series Temporales:

Este tipo de métodos puede incluir el método ingenuo Naive en el que se utiliza la media simple, la media móvil y el método Naive para pronosticar series temporales que no tienen tendencia ni estacionalidad y se caracterizan por su estabilidad, mientras que la media simple utiliza la media de todas las observaciones para pronosticar (Pathak, 2021), finalmente el método de la media móvil utiliza la media de los k valores de datos más recientes de la serie temporal (García, 2021).

2.6.8.3 Técnicas de Suavizado Exponencial:

La técnica de suavización exponencial es el resultado del promedio ponderado de los valores de una serie temporal pasada para pronosticar valores futuros (Villarreal, 2016), estos métodos generan pronósticos confiables rápidamente y para una amplia gama de series temporales, lo cual es una gran ventaja (Pathak, 2021). Existen tres técnicas, suavizado exponencial simple, método Holt con tendencia y el método Holt Winters.

El método de suavizado exponencial simple realiza las predicciones objetándose en un valor del período anterior, siendo adecuado cuando la serie no presenta una tendencia constante ni estacionalidad, en otro sentido, el método Holt realiza las predicciones en

función de la tendencia de la serie temporal, siendo beneficioso cuando se observa una tendencia lineal, pero sin estacionalidad, y por último, el método de Winters de suavizado exponencial triple modela el nivel general de la serie, así como la tendencia y la estacionalidad, siendo especialmente efectivo cuando la serie muestra tanto una tendencia como patrones estacionales (FasterCapital, 2023).

El método Winters, puede ser aplicado de maneta aditiva o multiplicativa, el enfoque aditivo es recomendado para las series con una tendencia lineal y con un patrón estacional que no está relacionado con el nivel de la serie, mientras que el modelo multiplicativo es más apropiado para series con tendencia lineal y con un patrón estacional que esta influenciado por el nivel de la serie (Florencia, 2018).

Métodos Autoregresivos: En los métodos autorregresivos, la técnica de regresión se utiliza para pronosticar observaciones futuras, utilizando una combinación lineal de observaciones pasadas, pero para ello la serie temporal debe seguir los supuestos de estacionalidad y autocorrelación (Pathak, 2021).

Método de Regresión Automática (AR): En este enfoque el proceso se representa como la suma ponderada de las observaciones pasadas, es decir para modelar y predecir relaciones lineales entre variables dependientes y sus rezagos (Ríos & Hurtado, 2008).

Método de Media Móvil (MA): En estos modelos la predicción se realiza en función de los errores de pronóstico pasados en un modelo similar a una regresión, es decir se enfoca en los errores o residuos de los valores pasados (Pathak, 2021).

Método de Media Móvil de Regresión Automática (ARMA): En este modelo, el pronóstico se realiza en función de las observaciones pasadas y de los valores actuales y rezagados **(Pypro, 2025)**

Media Móvil Integrada Regresiva Automática (ARIMA): Es un modelo econométrico ampliamente utilizado para el análisis y pronóstico de series temporales, este método incorpora un valor como una combinación lineal de datos previos y errores aleatorios, permitiendo modelar patrones y tendencias en datos secuenciales, especialmente cuando la serie temporal a modelar no es estacionaria a causa de una clara tendencia **(Noble, 2024)**

Media Móvil Integrada Autoregresiva Estacional (SARIMA): SARIMA es un método de ARIMA, pero tiene un componente adicional de parametrización de estacionalidad

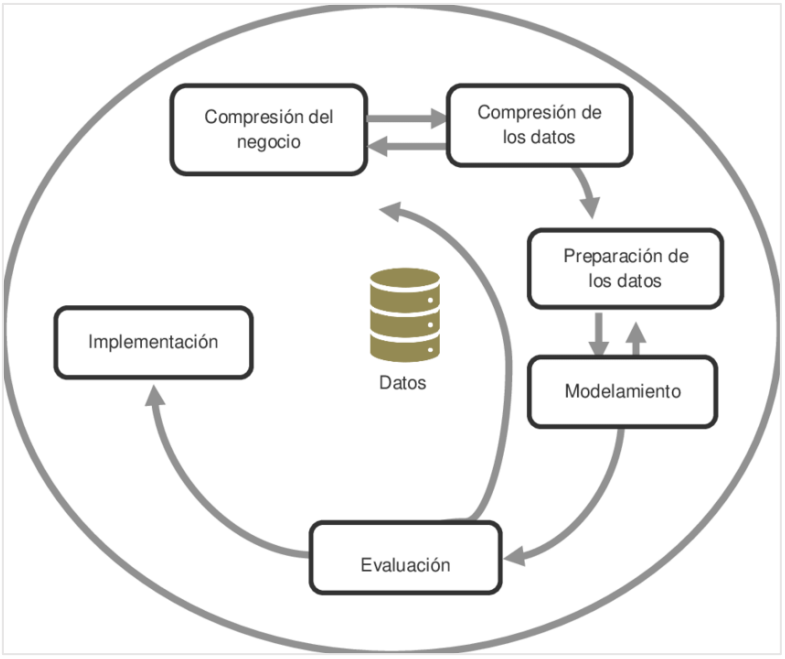
(Pathak, 2021), pueden manejar datos con patrones estacionales y los parámetros asociados permiten capturar las fluctuaciones estacionales en los datos (Subramanian et al., 2023).

Es relevante para este proyecto comprender cual es la aplicación del análisis de series temporales, una técnica que permite estudiar patrones y tendencias a lo largo del tiempo, el análisis de series temporales se utilizará para examinar las fluctuaciones y cambios en las tasas de defunciones por problemas cardiacos a lo largo del tiempo, identificando posibles estacionalidades, picos o descensos significativos relacionados con factores como condiciones de salud públicas, intervenciones médicas y cambios en los estilos de vida, este método ayudara a detectar patrones temporales que pueden ser esenciales para predecir futuros brotes o tendencias en las defunciones cardiacas, la utilización de series temporales beneficiará el proyecto al permitir una comprensión más profunda de la evolución de las defunciones cardiacas y contribuirá a la planificación de estrategias de prevención y respuesta más efectivas.

2.7 CRIPS DM

Para llevar a cabo esta investigación, es necesario utilizar un marco o metodología que ofrezca un enfoque estandarizado para identificar patrones o características en los conjuntos de datos. Por ello, se ha optado por emplear la metodología CRISP-DM.

Figura 2: Proceso CRIS DM



Fuente: (IBM, 2025)

CRISP-DM representa a Cross Industry Standard Process for Data Mining es un modelo abierto que describe las etapas de un proyecto de ciencia de datos, este proceso es ampliamente utilizado en el ámbito de la ciencia de datos porque permite adaptar cualquier tema o estudio relacionado con la minería de datos, este modelo CRISP-DM define seis fases como se observa en la figura 2 en el desarrollo de un proyecto de minería de datos, estas fases incluyen la comprensión del negocio, la comprensión de los datos, la preparación de los datos, el modelado, la evaluación y, finalmente, el despliegue.

Aunque el modelo CRISP-DM establece un orden para las fases, este no es necesariamente rígido, es decir, se permite iterar de manera recursiva entre las distintas etapas, por lo tanto, esta capacidad de recurrencia en CRISP-DM permite refinar cada fase durante la ejecución del proyecto, en consecuencia, tras obtener los resultados, es posible repetir las fases tantas veces como sea necesario hasta alcanzar los resultados deseados en el proyecto de minería de datos. A continuación, se describen cada una de las fases del modelo CRISP-DM:

a. Comprensión del Negocio (Business Understanding)

Esta fase inicial se centra en comprender los objetivos y necesidades desde una perspectiva empresarial y por consecuencia evalúa la situación actual y convierte los objetivos del negocio en un problema específico de minería de datos, estableciendo las metas para el análisis, en esta etapa se elabora un plan preliminar para el desarrollo del proyecto.

b. Comprensión de Datos (Data Understanding)

Esta fase comprende la recopilación y el análisis preliminar de los datos, además se lleva a cabo una descripción de los datos, evaluando aspectos como la cantidad, los tipos de valores y la codificación utilizada, adicionalmente se revisa la calidad de los datos, identificando problemas como datos faltantes, errores en los metadatos y discrepancias en la codificación, y por último, se elabora un resumen que puede incluir estadísticas básicas, tablas, visualizaciones e inferencias que ayudan a formular premisas o hipótesis iniciales.

c. Preparación de Datos (Data Preparation)

Esta fase abarca los aspectos cruciales de la minería de datos y, a menudo, las tareas asociadas pueden consumir entre el 50% y el 70% del tiempo y esfuerzo en un proyecto

de ciencia de datos, y es necesario usar el tiempo adecuado en las fases anteriores esto puede reducir la complejidad y los costos indirectos, la preparación de los datos suele realizarse varias veces y sin un orden fijo, e incluye tareas como la agregación de registros, el formateo de datos, la integración de conjuntos, la creación de nuevos atributos, la clasificación de datos, y la eliminación o sustitución de valores.

d. Modelado (Modeling)

Esta fase tiene como objetivo desarrollar y seleccionar el modelo que mejor se ajuste a los objetivos y requisitos del negocio, es decir el proceso de modelado suele llevarse a cabo en múltiples iteraciones y los científicos de datos o analistas prueban diversos algoritmos inicialmente con parámetros predeterminados, y luego ajustan estos parámetros para optimizar el rendimiento, dependiendo de los resultados, las métricas de evaluación y el comportamiento del modelo, es posible que se necesite regresar a la fase anterior para modificar la selección de variables o ajustar la limpieza de datos.

e. Evaluación (Evaluation)

Esta fase permite verificar si los modelos son técnicamente correctos y eficientes según los criterios definidos en las fases anteriores, además se debe evaluar los resultados obtenidos en relación con los criterios establecidos al inicio del proyecto, es relevante asegurarse de que la organización o el negocio pueda utilizar el modelo y los resultados, que comúnmente se denominan descubrimientos, en resumen es importante revisar el proceso realizado, interpretar los resultados y confirmar que el modelo cumple con los objetivos establecidos.

f. Despliegue (Deployment)

Esta fase se encarga de utilizar el conocimiento adquirido para implementar mejoras en la organización o el negocio, principalmente, incluye actividades de planificación y supervisión del despliegue de los resultados, determinando el plan de implementación y los requisitos básicos, adicionalmente documentan las actividades realizadas durante el proyecto de minería de datos, dependiendo de los requerimientos de la fase de despliegue, se definen los pasos para la implementación, la integración con otros sistemas, los requisitos de hardware y software, el plan de contingencia, las soluciones para problemas comunes y la documentación técnica asociada al proyecto de minería de datos.

CAPÍTULO 3 MARCO METODOLÓGICO

3.1 Marco Metodológico de investigación

La fuente de la información fue extraída de los Egresos Hospitalarios del Instituto Nacional de Estadística y Censos (INEC), entidad encargada de recopilar datos sobre los egresos hospitalarios a nivel nacional, abarcando todas las instituciones de salud del país. Se consideró el período comprendido entre 2010 - 2023, debido a la disponibilidad de los datos para esos años. La información original estaba en formato SAV (archivos de SPSS), por lo que se procedió a transformar todas las variables a formato CSV para facilitar su manejo y análisis.

El procesamiento de los datos y la realización de los joins entre las diferentes fuentes se llevó a cabo utilizando Python en el entorno de Jupyter Notebook, durante este proceso, se realizaron ajustes en la cantidad de variables para asegurar una correcta integración de los datos, garantizando que las variables estuvieran alineadas y listas para ser combinadas de manera coherente, y este tratamiento permitió generar un conjunto de datos homogéneo y apto para el análisis posterior se realizó un análisis profundo de la base de datos de egresos hospitalarios con algoritmos y técnicas de clasificación.

3.2 Método

El presente estudio según su enfoque es cuantitativo de acuerdo con la obtención de datos de una base es secundaria, según el tipo de variables estudiadas es descriptivo, analítico, inferencial.

Para el primer objetivo en analizar la evolución y las causas de defunciones Hospitalarias por problemas cardiacos en Ecuador, usando técnicas de machine learning. Este análisis se llevó a cabo usando el proceso CRISP-DM. En particular, en la parte de la descripción y exploración de datos de la serie temporal, nos enfocaremos en la evolución de las defunciones.

Para el segundo objetivo en aplicar métodos y técnicas de pronóstico de series de tiempo para determinar cuál es el más efectivo para predecir el comportamiento de las defunciones Hospitalarias por problemas cardiacos en Ecuador, se realizó un análisis de series de tiempo con el Método Suavizado Exponencial de Holt con Tendencia, Método Multiplicativo de Holt Winters con Tendencia y Estacionalidad, ARIMA y SARIMA y además se evaluo el mejor método usando

Para el tercer objetivo en caracterizar los principales clústeres determinantes de defunciones Hospitalarias por problemas cardiacos en Ecuador, usando técnicas aprendizaje no supervisado, se realizó un análisis de clusterización y un análisis factorial de correspondencias múltiples.

3.3 Marco Metodológico de Ciencia de Datos

Dentro del contexto de la metodología de la ciencia de datos, se utilizó el enfoque CRISP-DM este es un modelo ampliamente reconocido que ofrece un marco estructurado y eficiente para llevar a cabo proyectos de análisis de datos.

Como ya se explicó en el apartado 2.7 el primer paso de este enfoque es el Entendimiento del Negocio, en esta fase, el objetivo principal es obtener una visión clara y detallada de los requisitos y metas del proyecto el siguiente paso es la Comprensión de los Datos, en esta fase, se llevó a cabo la integración de los datos, los cuales estaban separados por año, utilizando Jupyter Notebook en Python durante este proceso, se entendió los datos, lo que es crucial para evaluar su calidad y verificar si la información disponible es adecuada para responder a las preguntas planteadas, y a continuación, se llevó a cabo la Preparación de los Datos, esta fase incluye una serie de actividades, que van desde la limpieza de los datos hasta la integración de información proveniente de diversas fuentes, en caso de ser necesario después la siguiente fase es el Modelado, en la cual se aplicó diversas técnicas de ciencia de datos y algoritmos para explorar varios modelos y elegir el más adecuado. Este proceso puede incluir la prueba de diferentes enfoques y la realización de ajustes con el fin de optimizar el rendimiento del modelo, y la fase de Evaluación sigue al modelado, en esta etapa, se evaluó el modelo seleccionado para garantizar que cumpla de manera efectiva con los objetivos del negocio y, por último, se procedió a la etapa de Despliegue del modelo. En esta fase, el modelo se aplica a datos nuevos relacionados con las defunciones cardiacas para generar insights, y los resultados se presentan a las partes interesadas. Este paso garantiza que los conocimientos obtenidos se conviertan en acciones efectivas y valiosas para abordar el problema de salud.

CAPÍTULO 4 RESULTADOS

4.1 Aplicación de métodos y técnicas de pronóstico de series de tiempo para predecir el comportamiento de las defunciones Hospitalarias por problemas cardiacos

4.1.1 Comprensión del Negocio

Determinación de los Objetivos Comerciales

4.1.1.1 Contexto

Como se planteó en el Capítulo 1 el propósito del presente trabajo de investigación aplicada es identificar clústeres y patrones que mayormente contribuyen a los casos de las defunciones hospitalarias relacionadas con problemas cardíacos, de forma que se logre caracterizar los eventos más recurrentes, proporcionando información relevante al MSP para que se promuevan programas o políticas adecuadas en la prevención de enfermedades por problemas cardiacos.

El análisis de las defunciones por enfermedades cardiacas es fundamental porque se pueden diseñar e implementar programas y políticas públicas más efectivas, orientadas a las principales patologías que afectan la salud cardiaca de la población, en este contexto proactivo facilita la identificación de factores de riesgo, mejora la prevención y optimiza los tratamientos, y adicionalmente, permite tomar decisiones estratégicas basadas en datos reales, garantizando una respuesta más eficiente y adaptada a las necesidades de la población, efectivamente, el desarrollo de modelos predictivos de mortalidad cardiaca se convierte en una herramienta esencial para fortalecer la gestión pública en el ámbito de la salud, asegurando una intervención más oportuna y centrada en los sectores más vulnerables.

4.1.1.2 Definición de los Objetivos del Negocio

Desarrollar un modelo predictivo para anticipar las principales causas de muerte, particularmente las enfermedades por problemas cardiacos, en Ecuador es una medida esencial para mejorar la planificación y ejecución de políticas públicas en el sector salud, en este sentido que las enfermedades cardíacas son una de las principales causas de mortalidad en el país, contar con una herramienta predictiva permitiría al Estado identificar con precisión las áreas de mayor riesgo y las poblaciones más vulnerables. Esto posibilitaría la implementación de intervenciones focalizadas, como programas de prevención, diagnóstico temprano, y tratamientos adecuados, ajustados a las características específicas de cada región y grupo demográfico.

4.1.1.3 Criterios de Rendimiento

Obtener una precisión efectiva en los modelos de series de tiempo para predecir las principales causas de muerte cardiovasculares, es primordial para desarrollar un sistema de alerta temprana que permita al Estado tomar decisiones informadas, en este contexto, la precisión en las predicciones es crucial, ya que garantiza que los recursos y las intervenciones se asignen de manera eficiente, abordando con anticipación las tendencias y picos en la mortalidad cardiaca, usando modelos de series de tiempo, es posible identificar patrones estacionales, ciclos y tendencias a largo plazo que contribuyen a la mortalidad por problemas cardíacos, permitiendo prever posibles aumentos en los casos de manera anticipada.

4.1.1.4 Evaluación de la Situación

El Instituto Nacional de Estadística y Censos (INEC), como organismo rector y coordinador del Sistema Estadístico Nacional, tiene un papel fundamental en la recopilación y publicación de datos sobre la salud en Ecuador en este caso de egresos hospitalarios, cada año, el INEC publica el Registro Estadístico de egresos hospitalarios, recoge información sobre todos los egresos hospitalarios ocurridas e inscritas en el territorio nacional, esta valiosa base de datos se alimenta de registros procedentes de diversas fuentes oficiales, como los registros de egresos del MSP, de las entidades privadas, del IESS, ISSFA, ISSPOL y entidades complementarias etc.

Para este proyecto, no existen restricciones legales que limiten el acceso a la información utilizada, ya que los datos provienen de fuentes públicas y están adecuadamente anonimizados, garantizando el respeto a las leyes de la seguridad de la información es decir la privacidad y la confidencialidad de los individuos, por otro lado la disponibilidad y el acceso a estos datos son fundamentales para la creación de modelos predictivos de defunciones cardiacas, lo que permitirá identificar patrones y tendencias que son clave para la toma de decisiones en el ámbito de la salud pública.

4.1.2 Determinación de los Objetivos de Minería de Datos

4.1.2.1 Objetivos de los Modelos de Predicción de Series de Tiempo

Desarrollar modelos de pronóstico de series de tiempo para predecir las tres principales causas de defunciones cardiacas en el Ecuador.

Utilizar métricas de evaluación para seleccionar el modelo más efectivo y preciso.

4.1.2.2 Criterios de rendimiento.

El desarrollo del presente proyecto de investigación los parámetros de evaluación utilizados para los métodos de pronóstico de series temporales son: el MAPE (Mean Absolute Percentage Error), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) y Mean Absolute Error (MAE).

4.1.2.3 Plan del proyecto

Tabla 1: Plan del proyecto de fin de titulación, series de tiempo

Fase	Duración	Recursos	Riesgos
Comprensión del negocio	1 semana	Responsable del proyecto	
Comprensión de los datos	2 semanas	Acceso a datos históricos sobre egresos hospitalarios	
Preparación de los datos	3 semanas		Excesivos datos faltantes
Modelado	4 Semanas	Plataforma para el modelado (Jupyter Notebook)	Dificultad para encontrar el modelo adecuado
Evaluación	1 Semana		

En la tabla 1 se desarrolla la planificación del plan del modelo a realizar el cual contiene la fase, la duración, los recursos, los riesgos del desarrollo de las series temporales, y sus diferentes etapas que se va a desarrollar.

4.1.3 Compresión de los Datos

Los datos de las defunciones cardiacas fueron extraídos de la información de egresos hospitalarios cuyos datos están publicados por el Instituto Nacional de Estadística y Censos, en formato SPSS y csv cargados en Jupyter Notebook, para su revisión y procesamiento, el periodo comprende desde el año 2010 hasta el año 2023.

Antes de iniciar con la preparación de los datos, es importante acceder a los datos primarios o datos sin procesar que tenemos de egresos hospitalarios los cuales contienen las defunciones por problemas cardiacos, en el Ecuador en el periodo 2010-2023 y

explorarlos con ayuda de tablas, gráficos que permitan determinar la calidad de datos a priori obtenidos del INEC, inclusive tener un entendimiento inicial de sus características.

4.1.3.1 Recopilación de Datos Iniciales

La información necesaria se descargó de la página web del Instituto Nacional de Estadística y Censos INEC, se utilizó la información de egresos hospitalarios del período 2010-2023, a través del método cuantitativo se medirán las variables, se realizará el análisis de factores relacionados con problemas isquémicos del Corazón con la utilización de la CIE 10 a través de técnicas estadísticas y de data mining y clustering, con el propósito de estudiar las variables demográficas, sociales y de atenciones hospitalarias.

4.1.3.2 Descripción de Datos

Se analizaron variables demográficas, epidemiológicas, radiológicas, de laboratorio y terapéuticas, así como de mortalidad durante el ingreso o reingreso el tipo de defunción si es mayor a 48 horas o menor a 48 horas.

La variable dependiente es la mortalidad por problemas del corazón mientras que las independientes son las Variables clínicas CIE 10 de mortalidad o morbilidad, la edad, el sexo, el peso etc.

Tabla 2: Campos que componen la base de datos de egresos hospitalarias

Categoría	Campos
prov_ubi	Provincia de ubicación del establecimiento de salud
cant_ubi	Cantón de ubicación del establecimiento de salud
parr_ubi	Parroquia de ubicación del establecimiento de salud
area_ubi	Area de ubicación del establecimiento de salud
Clase	Clase del establecimiento de salud
Tipo	Tipo de establecimiento de salud
Entidad	Entidad a la que pertenece el establecimiento de salud
Sector	Sector al que pertenece el establecimieto de salud
mes_inv	Mes de registro/investigación
Sexo	Sexo del paciente
cod_edad	Condición de la edad del paciente
Edad	Edad del paciente
Etnia	Definición étnica del paciente
prov_res	Provincia de residencia habitual del paciente
cant_res	Cantón de residencia habitual del paciente
parr_res	Parroquia de residencia habitual del paciente
area_res	Área de residencia habitual del paciente

anio_ingr	Año de ingreso
mes_ingr	Mes de ingreso
dia_ingr	Día de ingreso
fecha_ingr	Fecha de ingreso
anio_egr	Año del egreso
mes_egr	Mes egreso
dia_egr	Día egreso
fecha_egr	Fecha de egreso
dia_estad	Días estada
con_egrpa	Condición del egreso
esp_egrpa	Especialidad del egreso
cau_cie10	Causa de lista internacional detallada a 4 dígitos de la CIE-10
causa3	Causa de lista internacional detallada a 3 dígitos de la CIE-10
cap221rx	Capitulo lista 221
cau221rx	Lista especial de 221 grupos
cau298rx	Lista de causas 298

La data de egresos hospitalarios de defunciones cardiacas en el Ecuador consta de 20.820 registros y en la tabla 2 se observa la existencia de 33 variables las cuales son: provincia de ubicación, cantón de ubicación, parroquia de ubicación, clase de establecimiento de salud, tipo de establecimiento de salud, entidad a la que pertenece el establecimiento de salud, sector al que pertenece el establecimiento de salud, mes de registro/investigación, sexo del paciente, condición de la edad del paciente, edad del paciente, definición étnica del paciente, provincia de residencia habitual del paciente, cantón de residencia habitual del paciente, parroquia de residencia habitual del paciente, área de residencia habitual del paciente, año de ingreso, mes de ingreso, día de ingreso, fecha de egreso, año del egreso, mes egreso, día egreso, fecha de egreso, para el caso de días estada se refiere a la estancia de un paciente en el hospital es decir la diferencia entre la fecha egreso e ingreso, y condición del egreso se refiere a las categorías en las cuales un paciente egreso del hospital y son 3 (1 = vivo, 2 =defunción menor a 49 horas y 3 =defunción mayor a 48 horas en este proyecto se usó las condiciones 2 y 3), especialidad del egreso es el área de internación hospitalaria en que se dio la defunción en este, causa de lista internacional detallada a 4 dígitos de la CIE-10 hace referencia al catálogo internacional de codificación de enfermedades entre las cuales solo se determinaron las que corresponden a defunciones cardiacas, causa de lista internacional detallada a 3 dígitos de la CIE-10 al igual proviene

del catálogo internacional de la CIE 10 y es el nombre de los capítulos principales , capitulo lista 221 pertenece a los códigos de los subtítulos de la codificación de la CIE 10, lista especial de 221 grupos, lista de causas hace referencia a la descripción de la afección de acuerdo al catálogo internacional de la CIE 10.

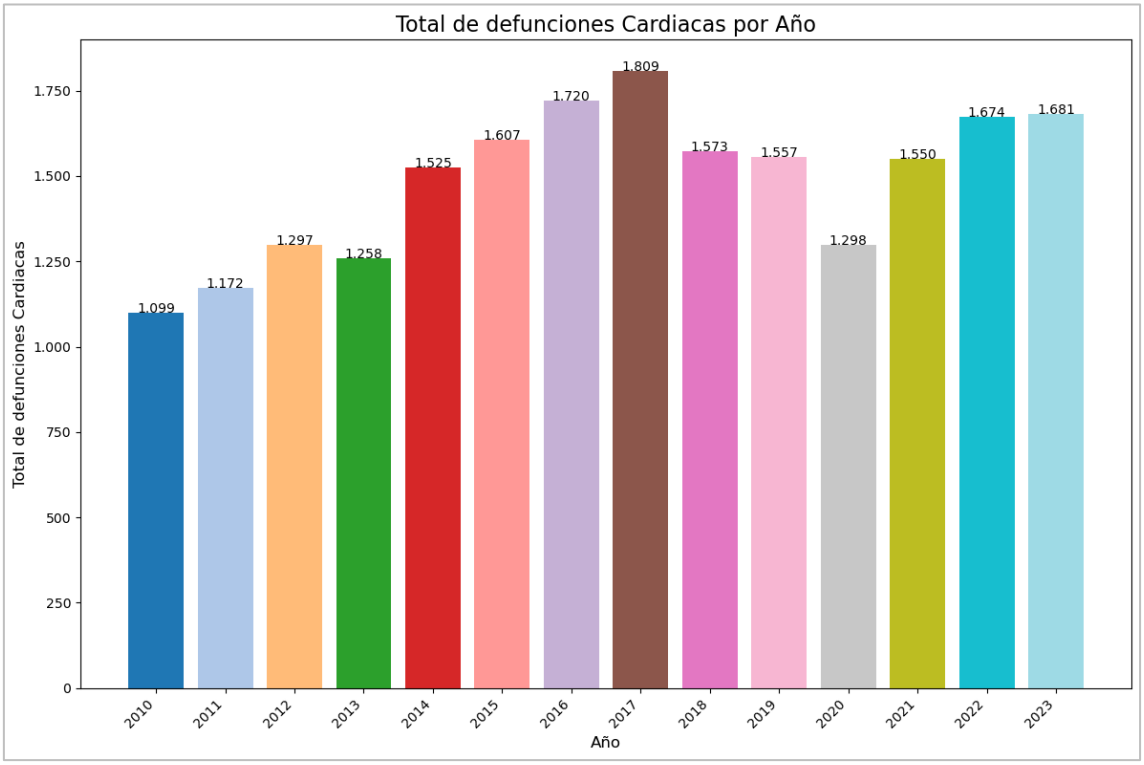
Para crear modelos predictivos de series temporales univariantes, es relevante incorporar el factor temporal, en este tema, este factor se define por el día, el mes y el año en que ocurrió el fallecimiento.

4.1.3.3 Exploración de datos

El archivo que contiene los datos de problemas cardiacos tiene 119.669 registros y el archivo que contiene las defunciones del caso tiene 20.820 registros, después de la integración de los datos en el archivo definitivo para el estudio se tiene 20.820 registros.

Se realiza un análisis gráfico de la distribución de las variables categóricas: AÑO, ETNIA, GENERO, ESTADO CIVIL y DIAGNÓSTICOS CIE10.

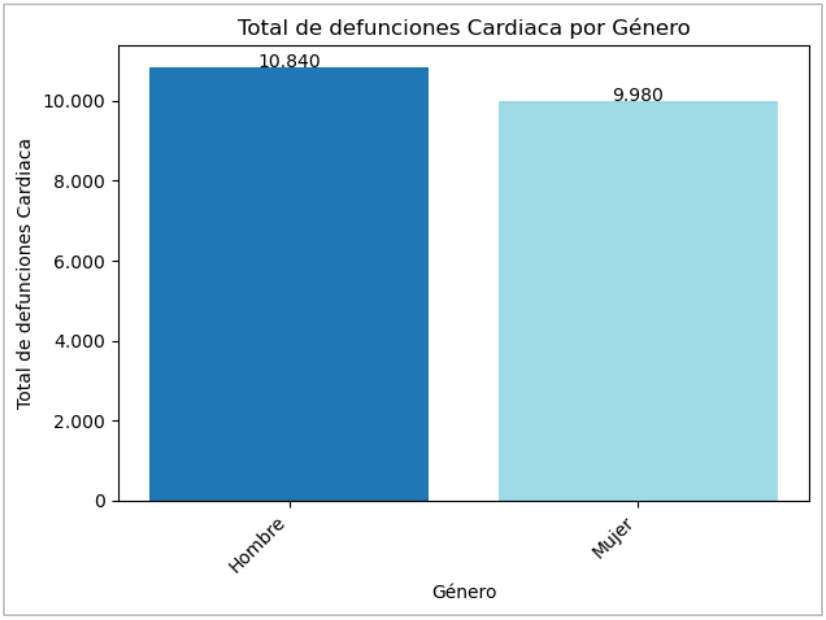
Gráfico 1: Total de Defunciones por Año



En el Gráfico 1, que muestra la distribución anual de defunciones por problemas cardíacos, se observan fluctuaciones significativas en el número de muertes a lo largo de los años en el año 2010 se registran 1.099 defunciones cardiacas mientras que en 2015,

se registraron 1.607 defunciones, cifra que se incrementó en 2017, año en el que se alcanzó el máximo histórico con la mayor cantidad de defunciones por problemas cardíacos, en el año 2020, se produjo una disminución notable, con 1.298 defunciones, la cifra más baja del período analizado, es decir este descenso podría estar relacionado con factores como las restricciones de movilidad y cambios en las tendencias de atención médica debido a la pandemia de COVID-19, y por último, en el año 2023, el número de defunciones aumentó nuevamente, alcanzando 1,681 casos, lo que refleja un repunte en las muertes por problemas cardíacos, es decir este patrón sugiere que, aunque los esfuerzos para controlar las enfermedades por problemas cardiacos han tenido algunos efectos, siguen existiendo factores que contribuyen a la persistencia y, en algunos casos, al aumento de las defunciones cardíacas a lo largo del tiempo.

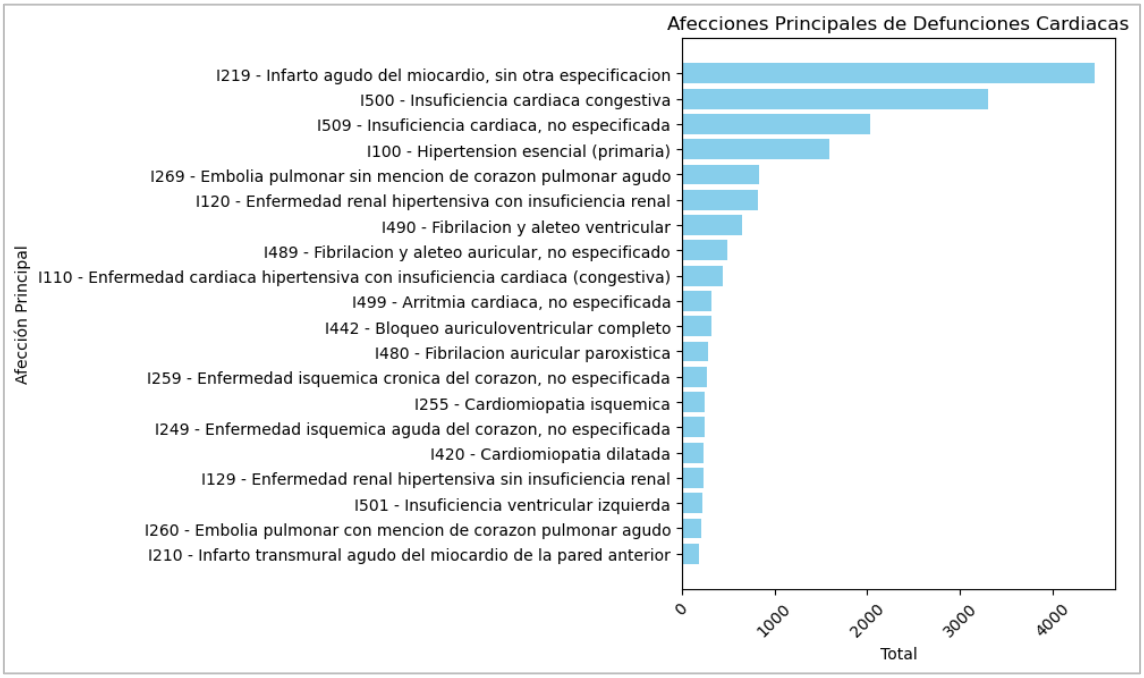
Gráfico 2: Total de defunciones Cardiaca por Género



En el grafico 2, se observa que las defunciones cardíacas por género, existe una prevalencia significativa del género masculino en comparación con el femenino, esta tendencia puede explicarse por diversos factores biológicos, sociales y de estilo de vida que afectan la salud cardiovascular de los hombres, además el comportamiento natural de las defunciones cardíacas en hombres se debe en parte a la mayor exposición a factores de riesgo como el tabaquismo, el consumo elevado de alcohol, la obesidad, la hipertensión y el sedentarismo, que son más comunes en esta población y además, los hombres suelen desarrollar enfermedades cardiovasculares a una edad más temprana que las mujeres, lo que contribuye a un mayor número de defunciones cardíacas en este grupo, no obstante

de que las mujeres también están expuestas a riesgos cardiovasculares, estos factores no son tan prevalentes o tienen un impacto menos inmediato que en los hombres, lo que puede explicar la disparidad observada en los datos.

Gráfico 3: Principales Causas de Defunciones por Problemas Cardiacos



Según el grafico 3, en Ecuador se registra una cantidad significativa de defunciones debido a problemas cardíacos, siendo el infarto agudo de miocardio la causa más frecuente es decir este trastorno ocurre cuando el flujo sanguíneo hacia el corazón se bloquea o se ve fuertemente restringido, generalmente por la acumulación de grasa, colesterol y otras sustancias en las arterias coronarias, lo que provoca una obstrucción, seguido por del infarto agudo de miocardio, la segunda causa más común de defunciones cardíacas es la insuficiencia cardíaca congestiva, una condición en la que el corazón no puede bombear sangre de manera eficiente y la causa menos frecuente es la fibrilación auricular paroxística, un tipo de arritmia caracterizada por episodios de latidos irregulares y rápidos en las aurículas del corazón, este comportamiento refleja las principales afecciones cardíacas que afectan a la población de Ecuador y subraya la importancia de abordar los factores de riesgo asociados con estas condiciones.

En el grafico 4 se observa que la entidad con el mayor número de defunciones cardíacas registradas es el Ministerio de Salud Pública del Ecuador, con un total de 9.357 casos, por otro lado, las entidades municipales representan el menor número, con tan solo 12 defunciones registradas, y es importante destacar también que las entidades privadas

reportan 5.576 defunciones, seguidas por el IESS, con 3.525 en fin estos datos reflejan las variaciones significativas en la distribución de las defunciones cardíacas entre diferentes tipos de instituciones en el país.

Gráfico 4: Total de defunciones Cardiacas por Entidad

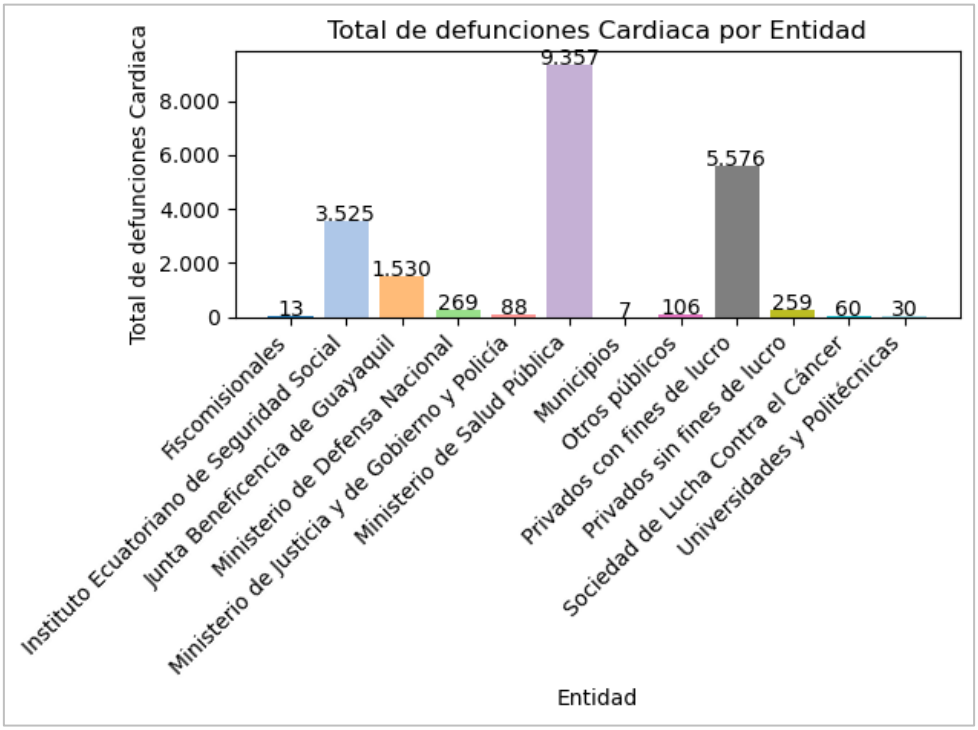
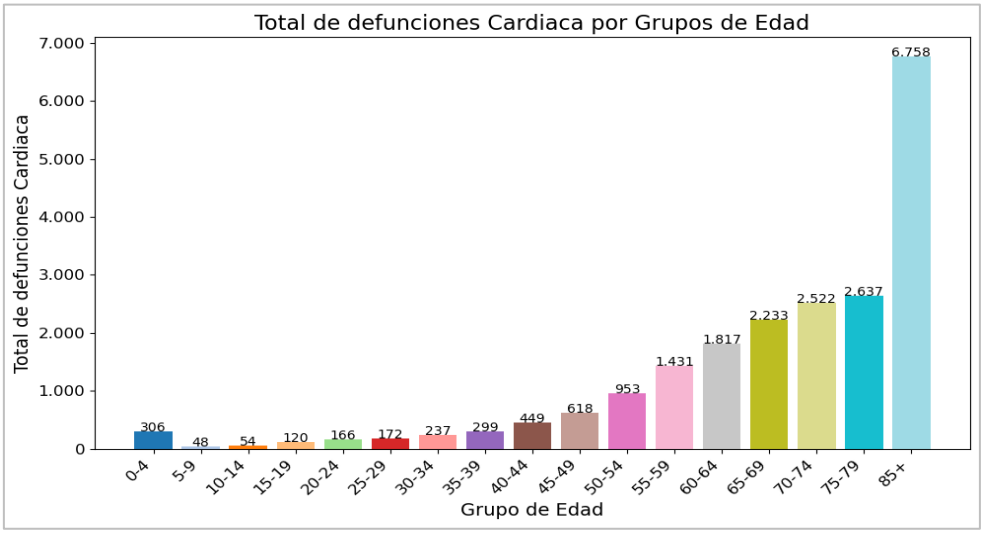


Gráfico 5: Total de Defunciones Cardiacas por Grupos de Edad



Según el grafico número 5, que muestra el total de defunciones cardiacas por grupos de edad, se puede observar una tendencia clara de aumento en el número de muertes a medida que avanza la edad, en los primeros grupos de edad, como el de 0 a 4 años existen 306 casos, y el de 5 a 9 años de edad 48 casos, las muertes son considerablemente bajas,

lo cual refleja la rareza de las enfermedades por problemas cardiacos en la infancia, aunque pueden estar asociadas a malformaciones congénitas o enfermedades graves, al avanzar en la adolescencia y juventud, como en el grupo de 10 a 14 años existen 54 casos, las muertes siguen siendo bajas, pero empiezan a reflejar condiciones más complejas o enfermedades hereditarias y a partir de los 30 a 34 años, se nota un aumento significativo con 237 casos, y en los grupos posteriores, como los de 40 a 44 años 449 casos y 50 a 54 años existen 953 casos, se observa un crecimiento progresivo es decir esto refleja la creciente influencia de factores de riesgo, como hipertensión, diabetes, tabaquismo y sedentarismo, que empiezan a afectar la salud cardiovascular en la edad adulta media., por consiguiente a partir de los 60 a 64 años hay 1.817 casos, las defunciones por problemas cardiacos aumentan de manera pronunciada, siendo los factores de riesgo acumulados a lo largo de la vida los principales responsables, y en este contexto el incremento más notable se presenta en los grupos de 75 a 79 años con 2.637 casos y 85 años y más con 6.758 casos, donde las defunciones alcanzan cifras mucho más altas, esto refleja cómo el envejecimiento y las morbilidades asociadas al paso de los años aumentan considerablemente el riesgo de enfermedades cardiacas, que se convierten en la principal causa de muerte en las personas mayores.

4.1.3.4 Verificación de la Calidad de los Datos

Figura 3: Verificación de valores nulos

```
prov_ubi                0
cant_ubi                0
parr_ubi                0
area_ubi                0
clase                   0
tipo                    0
entidad                 0
sector                  0
mes_inv                 0
sexo                    0
cod_edad                0
edad                    0
etnia                   0
prov_res                0
cant_res                0
parr_res                0
area_res                0
anio_ingr               0
mes_ingr                0
dia_ingr                0
fecha_ingr              0
anio_egr                0
mes_egr                 0
dia_egr                 0
fecha_egr               0
dia_estad               0
con_egrpa               0
esp_egrpa               0
cau_cie10               0
causa3                  0
cap221rx                0
cau221rx                0
cau298rx                0
etnia_recodificada      0
sexo_recodificado       0
Codc10                  0
AFECCIÓN_PRINCIPAL      0
Categoria 3 Subtitulos  0
descrp                  0
entidad_recodificado    0
edad_c                  0
rango_edad_c            0
des_3d                  0
dia                     0
fecha_def               0
dtype: int64
```

En la figura 3, se verifica que no existe la presencia de valores nulos en las variables de interés.

4.1.4 Preparación de los Datos

Seleccionar los Datos

Para la construcción de los modelos predictivos de series temporales, son necesarias las siguientes variables:

Año de egreso (año_egr): Corresponde al año en el cual se dio el egreso y la defunción.

Mes de egreso (mes_egr): Corresponde al mes en el cual aucedio el egreso y la defunción.

Afección principal (AFECCIÓN_PRINCIPAL): Corresponde a la lista de defunciones codificadas de la CIE 10 de la edición del 2018.

Se aplica el teorema de Pareto para determinar las causas de muerte cardiaca más críticas, se calcula el 80%, esto se detalla en la figura 4.

Figura 4: Código de 80/20

```
total_causas= conteo_t
ochenta_por_ciento= int(0.8*conteo_t)
ochenta_por_ciento

121
```

121 causas de defunción cardiaca son demasiadas para realizar el presente análisis, por lo cual nos enfocaremos en agrupar todas las causas como una solo de defunciones por problemas cardiacos.

Además, se crea una variable de tiempo, para el efecto se extrae la variable que corresponde al día de la defunción cardiaca, con la finalidad de contar con una fecha tipo d/m/a, y finalmente se crea la variable temporal que corresponde al año, mes y día de defunción.

4.1.5 Modelado

Existen varios métodos y enfoques para construir modelos de series de tiempo univariadas, entre los cuales se incluyen: métodos básicos de pronóstico, técnicas de suavizado exponencial y modelos autorregresivos.

4.1.5.1 Selección de Técnicas de Modelado

Teniendo en cuenta los beneficios de cada una de las técnicas y métodos de series de tiempo y analizando el tipo de problema a resolver, se han escogido cuatro en particular, dentro de los métodos de suavizamiento exponencial, el método Holt con tendencia y el método multiplicativo de Holt-Winters con estacionalidad y tendencia; y dentro de los métodos autorregresivos, se han seleccionado el ARIMA y el SARIMA.

Para aplicar los métodos seleccionados, se cuenta con una cantidad adecuada de datos temporales que facilitan la captura de las tendencias de las defunciones cardiacas a lo largo del tiempo, es decir estos datos se dividen en dos conjuntos uno de entrenamiento y otro de evaluación, con el fin de validar y estimar el desempeño de los modelos.

4.1.5.2 Modelado de Supuestos

Holt-Winters con estacionalidad y tendencia, son apropiados para identificar tendencias y comportamientos en los datos de series temporales debido a su capacidad para adaptarse a cambios en la serie a lo largo del tiempo, adicionalmente estos métodos resultan útiles cuando los datos presentan una tendencia lineal o estacional, ya que pueden capturar variaciones a corto y largo plazo y el método de Holt, es adecuado cuando se observa una tendencia constante, mientras que el de Holt-Winters, al incorporar estacionalidad, permite modelar comportamientos más complejos que incluyen ciclos repetitivos a lo largo del tiempo, como los efectos estacionales.

Por otro lado, los métodos autorregresivos, en particular ARIMA y SARIMA, son robustos frente a valores atípicos, lo que los hace muy eficaces en el tratamiento de datos ruidosos o que contienen datos anómalos, es decir estos métodos son especialmente adecuados para series temporales que muestran patrones de dependencia en el tiempo, es decir, donde el valor de la serie en un momento dado depende de sus valores anteriores, y ARIMA se adapta bien a series sin estacionalidad, mientras que SARIMA, al incluir un componente estacional, es idóneo para capturar tanto las dependencias temporales como los efectos estacionales en los datos, es decir los dos métodos son poderosos para predecir comportamientos futuros de series complejas, ya que no solo identifican tendencias y estacionalidades, sino que también ajustan sus predicciones teniendo en cuenta la autocorrelación de los datos.

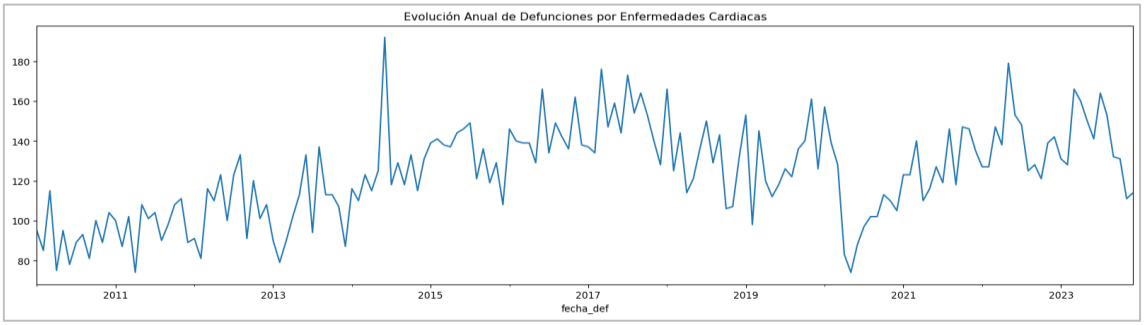
4.1.5.3 Diseño de Comprobación

En este contexto el conjunto de datos se divide en dos segmentos, uno para entrenamiento y otro para evaluación, además se prueban diversos modelos, entre ellos ARIMA, SARIMA, Holt con tendencia y Holt-Winters multiplicativo, además el rendimiento de cada modelo se analiza utilizando métricas específicas, tales como el Error Cuadrático Medio (MSE), la Raíz del Error Cuadrático Medio (RMSE), el Error Absoluto Medio (MAE) y el Error Porcentual Absoluto Medio (MAPE).

4.1.5.4 Generación de los Modelos

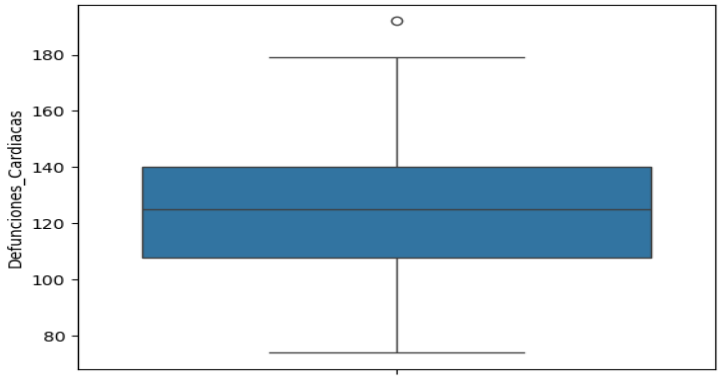
Causa de Muerte por problemas cardiacos

Gráfico 6: Evolución Anual de Defunciones por Enfermedades Cardiacas



En el gráfico 6 se puede observar la evolución de las defunciones por enfermedades cardíacas, destacándose algunos valores atípicos que reflejan fluctuaciones inusuales en determinados períodos, este comportamiento podría indicar factores específicos que afectan la mortalidad durante esos intervalos.

Gráfico 7: Boxplot de defunciones por problemas cardiacos



En el grafico 7 se presenta un diagrama de caja y bigote, el mismo permite visualizar la distribución de los datos, confirmando la presencia de un valor atípico que se encuentra

claramente fuera del rango esperado es decir el valor podría ser muestra de un comportamiento anómalo o una variación inesperada en los datos.

Figura 5: Código para eliminación de Outliers

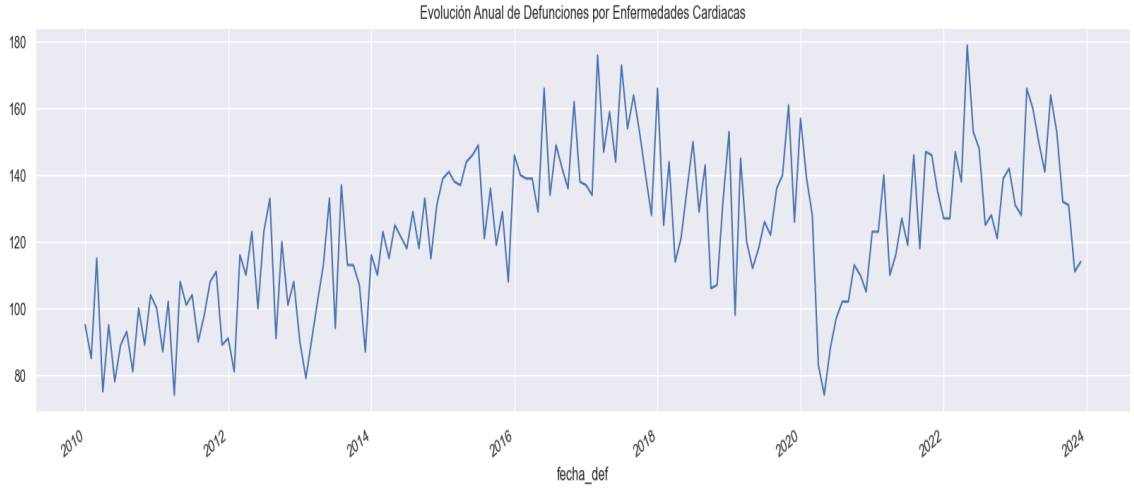
```
# Calcula los cuartiles de la columna 'Defunciones_Cardiacas'
Q1 = data1['Defunciones_Cardiacas'].quantile(0.25)
Q3 = data1['Defunciones_Cardiacas'].quantile(0.75)

# Calcula el rango intercuartil (IQR)
IQR = Q3 - Q1

# Calcula los límites superior e inferior
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR

# Filtra las filas que no son outliers en 'Defunciones_Cardiacas'
data1 = data1[(data1['Defunciones_Cardiacas'] >= limite_inferior) &
                        (data1['Defunciones_Cardiacas'] <= limite_superior)]
```

En los gráficos generados, se pueden identificar ciertos valores atípicos que requieren tratamiento, para esto, se utiliza la técnica del rango intercuartílico, que permite la identificación y tratamiento de los datos atípicos a través de la distancia intercuartílica, definida por el espacio entre el primer cuartil (Q1) y el tercer cuartil (Q3) de la distribución de los datos, además el umbral de 1.5 se elige de manera estándar, ya que es un valor usualmente utilizado para identificar valores atípicos dentro de esta técnica, y para este propósito, se genera la función detallada en la Figura 5.

Gráfico 8:Boxplot de defunciones por problemas cardiacos sin dato atípico

En el grafico 8 se observa la eliminación del dato atípico

Gráfico 9: Evolución Anual de Defunciones por Enfermedades Cardiacas sin dato atípico



En el grafico 9 se observa la corrida de los datos sin datos atípicos.

En los gráficos 8 y 9, se observa la eliminación de ciertos valores atípicos, una medida destinada a mejorar la eficacia de los modelos predictivos al entrenar y evaluar los datos con un proceder más afín a una distribución normal, además la exclusión de estos valores atípicos ayuda a reducir posibles distorsiones que podrían afectar negativamente la capacidad predictiva de los modelos.

Figura 6: Librerías de descomposición

```
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
from statsmodels.tsa.seasonal import seasonal_decompose
import seaborn as sns
sns.set()
```

```
pip install python-dateutil
```

```
from dateutil.parser import parse
```

En la figura 6 se importa Statsmodels la cual permite explorar y descomponer series temporales de manera efectiva, además la función seasonal_decompose se emplea para descomponer la serie en componentes estacionales, de tendencia y residuos, adicionalmente, se utilizan herramientas de diagnóstico como tsaplots y stattools para evaluar la autocorrelación y la estacionalidad de los datos, lo que facilita la identificación de patrones y la evaluación de la estacionariedad de la serie temporal.

Gráfico 10: Descomposición aditiva, causa de muerte enfermedades isquémicas del corazón.

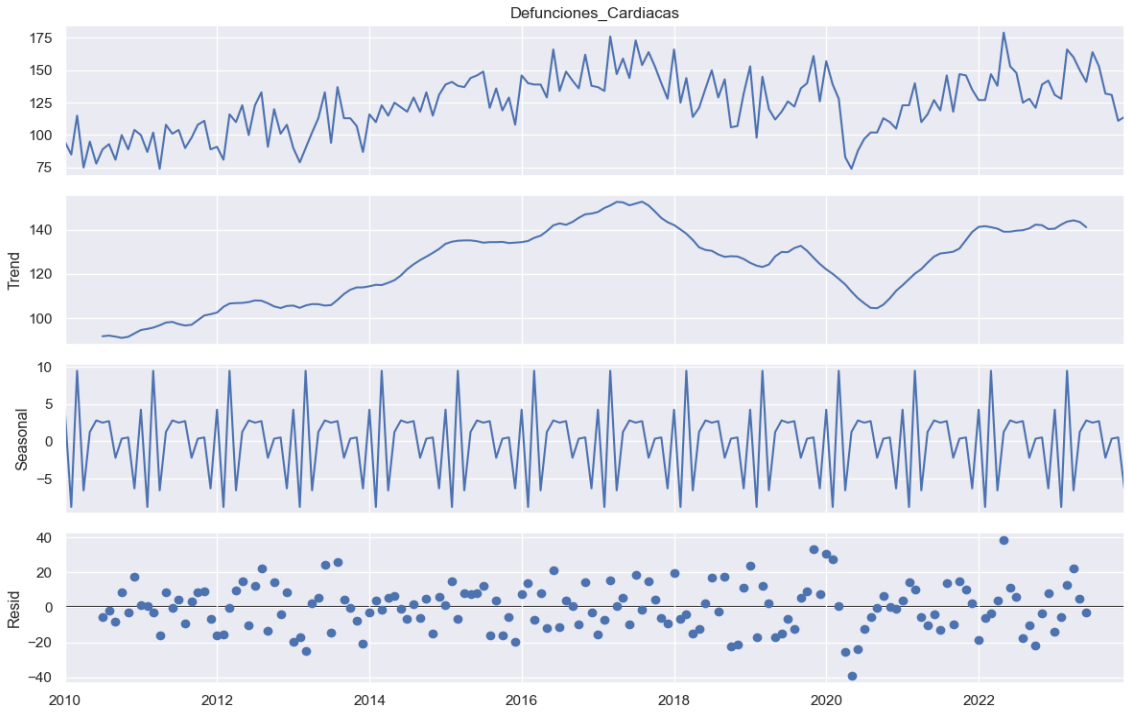
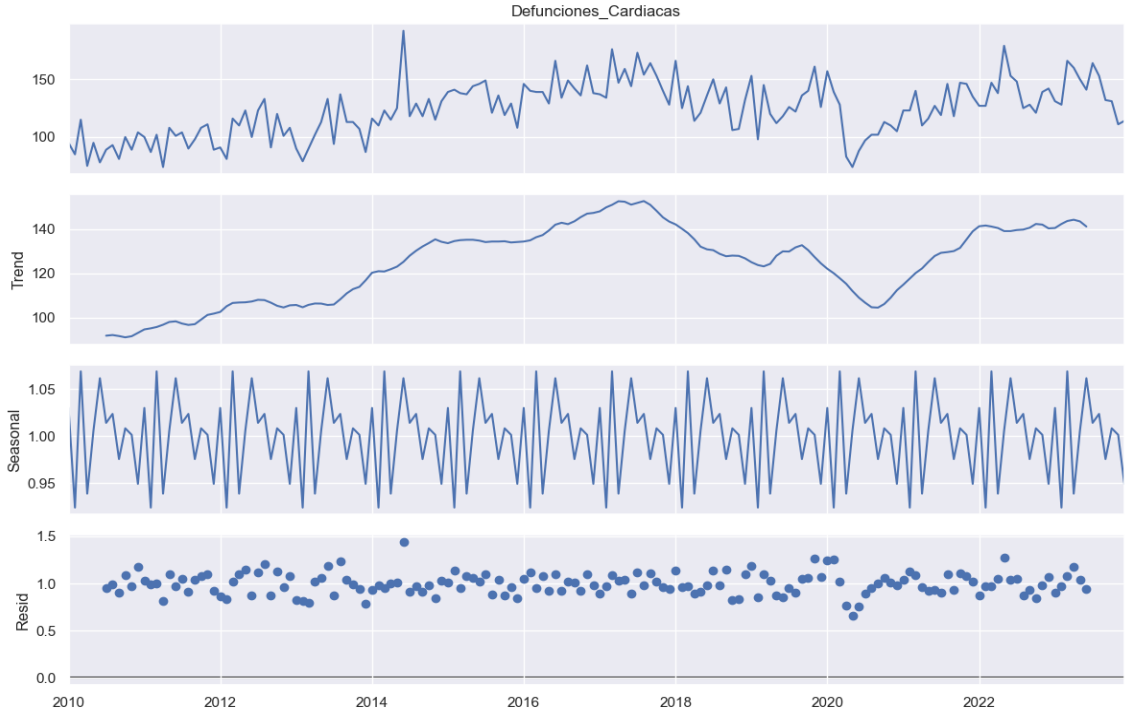


Gráfico 11: Descomposición multiplicativa, causa de muerte enfermedades isquémicas del corazón.



En los gráficos 10 y 11 se lleva a cabo la descomposición aditiva y multiplicativa, métodos empleados para descomponer la serie temporal en sus componentes principales,

como la tendencia, la estacionalidad y los residuos, a modo que se pueda apreciar en los gráficos, la tendencia de las defunciones por problemas cardiacos muestra un incremento continuo a lo largo del tiempo y adicionalmente, se pueden identificar picos en meses específicos del año, lo que indica la presencia de un patrón estacional en los datos.

Figura 7: División de la data en entrenamiento y evaluación.

```
total_l= len(data1)
train_len=round(total_l*0.8)
train_len
train=data1[0:train_len]
test=data1[train_len : ]
```

En la figura 7 se observa que el presente código divide los datos en dos conjuntos de datos uno de entrenamiento y otro de prueba, primero, calcula el tamaño total de los datos total_l y luego determina el tamaño del conjunto de entrenamiento como el 80% de los datos train_len, consecutivamente, se crea el conjunto de entrenamiento con los primeros train_len elementos de data1, y el conjunto de prueba con el resto de los elementos, esta parte es comúnmente utilizada en aprendizaje automático para el entrenamiento de un modelo con una parte de los datos y evaluar su desempeño con el conjunto de prueba.

a. Método Suavizado Exponencial de Holt con Tendencia

Para aplicar el método de suavizado exponencial de Holt con componente de tendencia, se delimitan los parámetros del modelo, como el número de períodos estacionales igual a 12 meses, el tipo de tendencia que sea aditiva y la ausencia de estacionalidad en los datos, puesto que este método se enfoca principalmente en la tendencia, el proceso se ilustra en la figura 8, mientras que las predicciones se muestran en el gráfico 12.

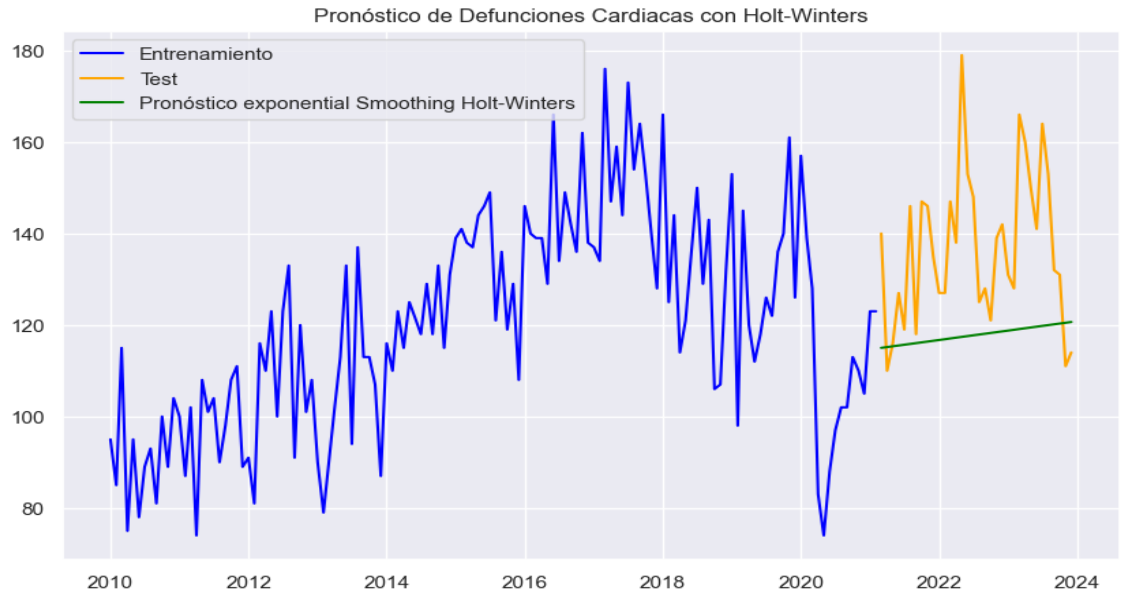
Figura 8: Método de suavizado exponencial de Holt con tendencia.

```
# Modelo Holt-Winters con tendencia aditiva y sin componente estacional
model = ExponentialSmoothing(
    np.asarray(train['Defunciones_Cardiacas']),
    seasonal_periods=12,  # Periodos estacionales (ajusta según tu caso)
    trend='additive',     # Tendencia aditiva
    seasonal=None         # Sin componente estacional
)

# Ajustar el modelo
model_fit = model.fit(optimized=True)

# Imprimir los parámetros ajustados del modelo
print(model_fit.params)

# Realizar el pronóstico para el conjunto de prueba
y_hat_holt = test.copy()  # Copiar los datos de test
y_hat_holt['holt_forecast'] = model_fit.forecast(len(test))  # Pronóstico para el periodo de test
```

En la figura 8 se observa que el código implementa un modelo de suavizado exponencial de Holt para realizar pronósticos de una serie temporal, en primer lugar, se crea el modelo

utilizando los datos de entrenamiento (train['Defunciones_Cardiacas']), delimitando el número de períodos estacionales es 12 y asumiendo un patrón anual, se usa una tendencia aditiva, y no se incluye componente estacional (seasonal=None), después de esto, el modelo se ajusta a los datos de entrenamiento utilizando el método fit con optimización de los parámetros, tras ajustar el modelo, se imprimen los parámetros estimados, en resumen, se realiza un pronóstico sobre el conjunto de prueba y se almacena en una nueva columna holt_forecast en el dataframe y_hat_holt y el pronóstico cubre la misma longitud que los datos de prueba.

Gráfico 12: Pronóstico exponential Smoothing Holt-Winters:



Como se puede observar en el gráfico 13, a partir del año 2021 se presenta una ligera tendencia creciente en los datos, lo que manifiesta un comportamiento sostenido de aumento a lo largo del tiempo, en este contexto, el pronóstico realizado con el método de suavizado exponencial Holt-Winters también muestra un crecimiento progresivo, siguiendo la misma dirección, esta tendencia sugiere que, aunque el incremento no es drástico, existe una tendencia estable que se mantiene en el horizonte temporal proyectado por el modelo.

b. Método Multiplicativo de Holt Winters con Tendencia y Estacionalidad

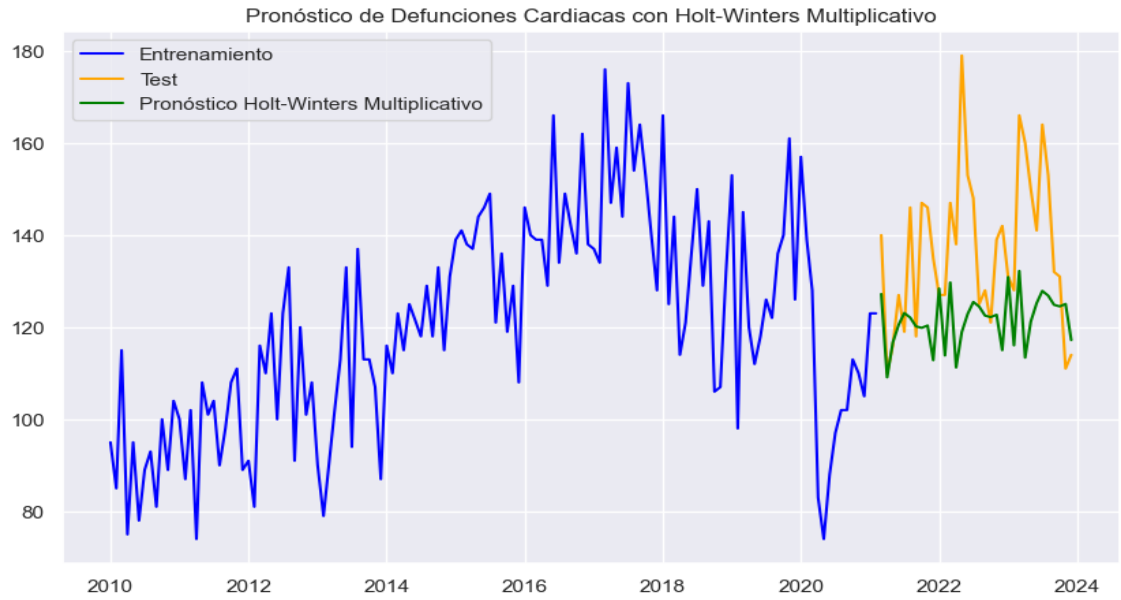
Para el método multiplicativo de Holt-Winters con componente de tendencia y estacionalidad, se delimitan los parámetros del modelo, como el número de períodos estacionales iguales a 12, el tipo de tendencia aditiva y la estacionalidad multiplicativa.

Figura 9: Método multiplicativo Holt Winter con tendencia y estacionalidad.

```python
# Modelo Holt-Winters con tendencia y estacionalidad multiplicativa
model_multiplicative = ExponentialSmoothing(
    np.asarray(train['Defunciones_Cardiacas']),
    seasonal_periods=12,  # Ajusta según el número de periodos estacionales (12 para datos mensuales)
    trend='additive',      # Tendencia aditiva (puedes probar 'multiplicative' si deseas)
    seasonal='multiplicative'  # Componente estacional multiplicativo
)

# Ajustar el modelo
model_multiplicative_fit = model_multiplicative.fit(optimized=True)

# Imprimir los parámetros ajustados del modelo
print(model_multiplicative_fit.params)

# Realizar el pronóstico para el conjunto de prueba
y_hat_multiplicative = test.copy()  # Copiar los datos de test
y_hat_multiplicative['holt_forecast_multiplicative'] = model_multiplicative_fit.forecast(len(test))  # Pronóstico
```

En la figura 9 se observa que el código implementa un modelo de suavizado exponencial de Holt-Winters con componente multiplicativo para predecir la serie temporal de Defunciones_Cardiacas, en primer lugar, se define el modelo, especificando que los datos de entrada provienen del conjunto de entrenamiento (train['Defunciones_Cardiacas']), con un número de períodos estacionales de 12 es decir para datos mensuales, una tendencia aditiva y una estacionalidad multiplicativa, después, el modelo se ajusta a los datos de entrenamiento utilizando el método fit con cierta optimización de los parámetros, tras el ajuste, se imprimen los parámetros obtenidos, en fin, se realiza un pronóstico para el conjunto de prueba (test), y el pronóstico generado se guarda en una nueva columna holt_forecast_multiplicative del dataframe y_hat_multiplicative.

Gráfico 13: Pronóstico multiplicativo de Holt Winters.



En el grafico 13 a partir del año 2021, el gráfico muestra una tendencia moderadamente creciente en los datos, con un ligero aumento en los valores hasta alcanzar el año 2023,

esta tendencia sugiere una estabilización o un crecimiento sostenido, aunque no de manera inesperada, en este sentido la tendencia indica que, si bien el incremento es gradual, persiste una inclinación ascendente en el período observado, lo cual podría reflejar cambios subyacentes en los factores que afectan las variables representadas.

c. Método ARIMA

Para utilizar métodos autorregresivos como ARIMA o SARIMA, es primordial cerciorarse de que la serie temporal sea estacionaria, es decir se debe a que una de las premisas clave de estos modelos es la suposición de que las propiedades estadísticas de la serie, como la media y la varianza, se mantienen constantes a lo largo del tiempo, de lo contrario si la serie no es estacionaria, es necesario aplicar transformaciones o técnicas adicionales, como la diferenciación Box Cox, para lograr la estacionariedad antes de aplicar estos métodos, sin esta condición, los resultados del modelo pueden ser inexactos o engañosos, lo que afectaría su capacidad predictiva.

Figura 10: Ejecución de código para crear la función para la prueba de Dickey-Fuller.

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Paso 1: Verificar la estacionariedad de la serie
def test_stationarity(series):
    result = adfuller(series)
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    if result[1] <= 0.05:
        print("La serie es estacionaria")
    else:
        print("La serie no es estacionaria")

test_stationarity(train['Defunciones_Cardiacas'])
```

En la figura 10 se observa que el código realiza una prueba de estacionariedad en una serie temporal utilizando la prueba de Dickey-Fuller aumentado (ADF) de la librería statsmodels, en primer lugar, importa las funciones necesarias para realizar la prueba (adfuller), así como herramientas para visualizar las funciones de autocorrelación de la serie temporal plot_acf y plot_pacf, aunque estas últimas no se utilizan en este fragmento de código, además la función test_stationarity toma como entrada una serie temporal y aplica la prueba ADF a dicha serie, por consiguiente, imprime el valor de la estadística ADF y el valor p, y si el valor p es menor o igual a 0.05, se concluye que la serie es

estacionaria, de lo contrario, se considera no estacionaria. Finalmente, la función es ejecutada sobre la columna Defunciones_Cardiacas del DataFrame train.

Figura 11: Salida de la prueba de Dickey-Fuller.

```
ADF Statistic: -3.0451219339832236
p-value: 0.030890083575817216
La serie es estacionaria
```

En la figura 11 se observa que la salida de la prueba de Dickey-Fuller aumentado (ADF) muestra una estadística ADF de -3.0451 y un valor p de 0.0309, dado que el valor p es menor que el umbral de significancia comúnmente utilizado de 0.05, se rechaza la hipótesis nula de que la serie temporal tiene una raíz unitaria lo que revelaría que es serie no estacionaria, por esta razón, podemos concluir que la serie temporal es estacionaria, esto implica que las características estadísticas de la serie, como la media y la varianza, permanecen estables a lo largo del tiempo.

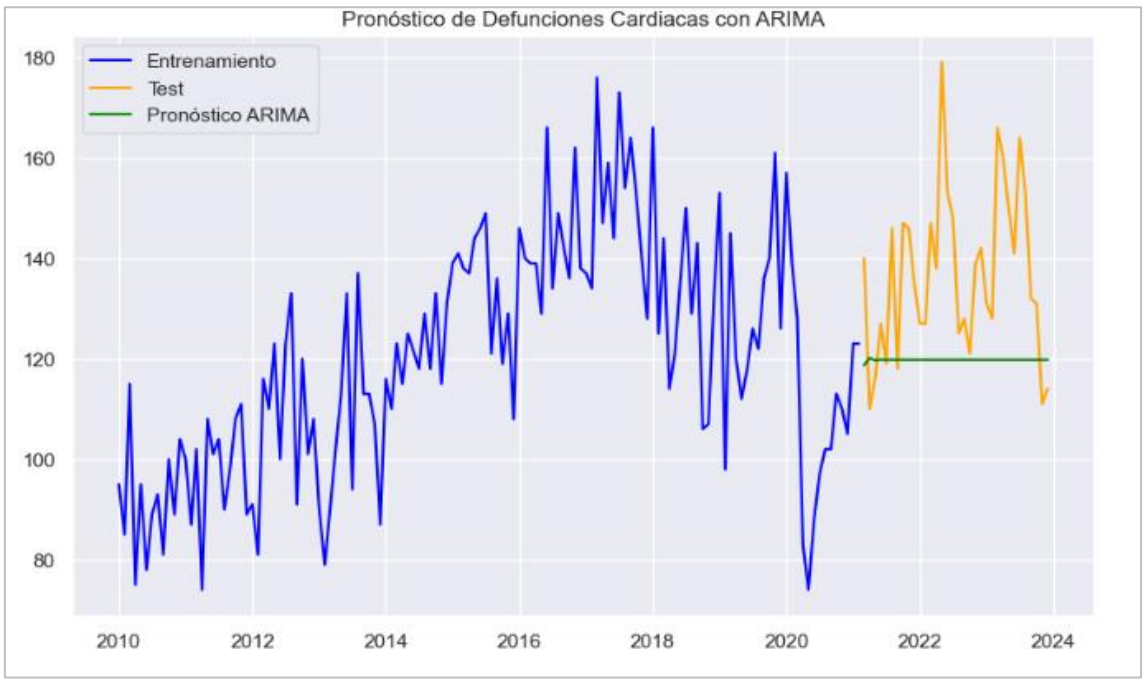
Figura 12: Código modelo ARIMA.

```
# Verificar la estacionariedad de la serie de entrenamiento
test_stationarity(train['Defunciones_Cardiacas'])

# Si la serie no es estacionaria, podemos diferenciarla:
# Diferenciamos si es necesario (d > 0)
train_diff = train['Defunciones_Cardiacas'].diff().dropna()

# Paso 2: Encontrar los parámetros p, d, q usando auto_arima
# Si no se ha determinado manualmente, podemos utilizar esta función
model_auto_arima = auto_arima(train['Defunciones_Cardiacas'], seasonal=False, stepwise=True, trace=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1119.739, Time=0.28 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1179.069, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1125.056, Time=0.11 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1124.147, Time=0.08 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1177.084, Time=0.03 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1123.377, Time=0.22 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1118.242, Time=0.26 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1124.212, Time=0.12 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1125.772, Time=0.15 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1119.699, Time=0.31 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1124.935, Time=0.22 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=1121.698, Time=0.48 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=1116.626, Time=0.13 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=1122.307, Time=0.07 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=1123.849, Time=0.06 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=1118.065, Time=0.21 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=1118.111, Time=0.22 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=1123.128, Time=0.06 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=1121.508, Time=0.15 sec
 ARIMA(3,1,0)(0,0,0)[0]             : AIC=1123.029, Time=0.09 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=1119.991, Time=0.52 sec

Best model:  ARIMA(2,1,1)(0,0,0)[0]
Total fit time: 3.829 seconds
```

En la figura 12 se observa que el procedimiento de búsqueda paso a paso para seleccionar el modelo ARIMA más adecuado minimizando el AIC es decir el Criterio de Información de Akaike resultó en varios modelos evaluados, e indica que el mejor modelo encontrado fue el ARIMA con parámetros (2,1,1)(0,0,0)[0], con el valor más bajo de AIC (1116.626), lo que sugiere que es el modelo más eficiente para ajustar la serie temporal según este criterio, y el tiempo total de ajuste de los modelos fue de 3.829 segundos.

Gráfico 14: Pronostico ARIMA



c. Método SARIMA

Un modelo SARIMA contiene factores estacionales autorregresivos (P), diferenciación estacional (D) y promedio móvil estacional (Q), es decir, el modelo SARIMA contiene factores estacionales y no estacionales.

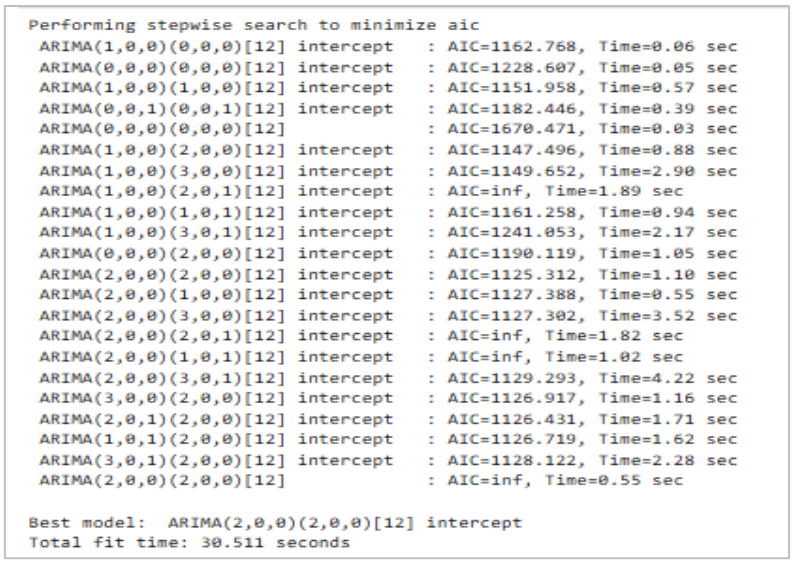
Figura 13:Código de parametrización modelo SARIMA

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

from pmdarima import auto_arima

# Ajustando un modelo SARIMA
model_auto_sarima = auto_arima(train['Defunciones_Cardiacas'], seasonal=True, m=12, stepwise=True, trace=True,
                               start_p= 1,d=0, start_q= 0, max_p = 3, max_d = 2,max_q= 3,
                              start_P=0,D=0,start_Q=0,max_P =3,max_Q=3,max_D=2, suppress_warnings=True,
                               error_action='ignore')
```

En la figura 13 se realiza la parametrización y la búsqueda automática para encontrar el mejor modelo SARIMA.

- trace=True: Señala detalles durante el proceso de búsqueda del modelo, como los modelos que se están probando y las métricas de ajuste correspondientes.
- error_action='ignore': Establece que, en caso de que ocurra un error durante el ajuste del modelo, se debe omitir el error y continuar con el proceso.
- start_p=0, d=0, start_q=0, max_p=3, max_d=2, max_q=3: Estos parámetros definen los rangos de búsqueda para los componentes no estacionales (p, d, q) dentro del modelo SARIMA.
- m=12: Menciona que la estacionalidad es de tipo mensual.
- start_P=0, D=0, start_Q=0, max_P=3, max_Q=3, max_D=2: Determina los parámetros de los componentes estacionales (P, D, Q) dentro del modelo SARIMA.
- suppress_warnings=True: Evita que se muestren advertencias durante el ajuste del modelo.
- stepwise=False: Indica que se realizará una búsqueda exhaustiva, en lugar de una búsqueda secuencial, para encontrar el modelo SARIMA óptimo.
- seasonal=True: Especifica que se ajustará un modelo SARIMA con estacionalidad.

Figura 14: Selección de mejor modelo



En la figura 14 se observa que el proceso de búsqueda paso a paso busca minimizar el AIC es decir el Criterio de Información de Akaike para encontrar el mejor modelo ARIMA, el modelo con el menor AIC es ARIMA(2,0,0)(2,0,0)[12] con un AIC de

1125.312, que se seleccionó como el mejor modelo después de realizado el proceso de comparar múltiples combinaciones.

Figura 15: Código de ajuste al mejor modelo

```
model_sarima = sm.tsa.SARIMAX(train['Defunciones_Cardiacas'],
                              order=(2, 0, 0),  # Reemplazar con los valores adecuados
                              seasonal_order=(2,0, 0,12),  # Reemplazar con los valores adecuados
                              enforce_stationarity=False,
                              enforce_invertibility=False)

sarimax_fit= model_sarima.fit()
# Ajustar el modelo
result_sarima = model_sarima.fit()

# Ver los resultados
print(result_sarima.summary())
```

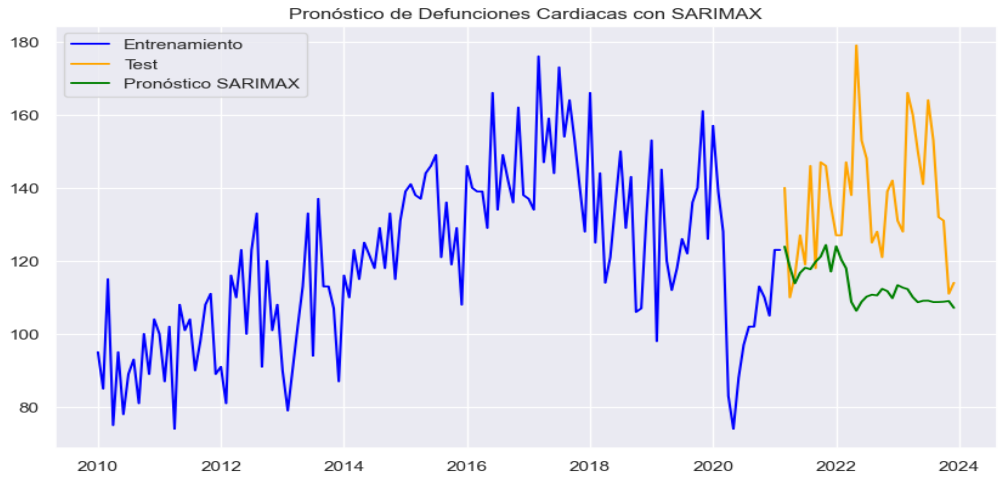
En la figura 15 se está ajustando un modelo SARIMA a los datos de Defunciones_Cardiacas, con parámetros no estacionales (2, 0, 0) y estacionales (2, 0, 0, 12), donde enforce_stationarity=False y enforce_invertibility=False permiten flexibilidad en la estación y la invertibilidad del modelo, el modelo se ajusta y guarda el resultado en result_sarima.

Figura 16: Código para invertir las transformaciones, modelo SARIMA

```
# Realizar pronósticos para el conjunto de prueba
y_hat_sarimax = test.copy()
y_hat_sarimax['sarimax_forecast'] = sarimax_fit.forecast(len(test))

# Mostrar las predicciones
print(y_hat_sarimax[['Defunciones_Cardiacas', 'sarimax_forecast']])
```

En la figura 16 se observa que el código realiza pronósticos con el modelo SARIMAX ajustado para el conjunto de prueba y muestra las predicciones junto con los valores reales de Defunciones_Cardiacas.

Gráfico 15: Pronóstico SARIMA



4.1.6 Evaluación de los Modelos

Figura 17: Resultados de la evaluacion de las métricas de los métodos aplicados

Modelo	MSE	RMSE	MAE	MAPE (%)
ARIMA	709.060328	26.628187	21.816781	14.877987
SARIMAX	888.118490	29.801317	24.091816	16.425162
Holt-Winters Multiplicativo	492.629467	22.195258	16.998431	11.513475
Holt-Winters sin Estacionalidad	629.554993	25.090934	20.449136	13.946976

En la figura 17 se observa que el mejor modelo es Holt-Winters Multiplicativo, ya que tiene los valores más bajos en las métricas clave:

MSE: 492.63

RMSE: 22.20

MAE: 16.99

MAPE (%): 11.51%

Este modelo tiene un mejor rendimiento en comparación con los otros, ya que presenta los errores más bajos en todas las métricas.

4.2 Aplicación de técnicas de clusterización en las defunciones Hospitalarias por problemas cardiacos

4.2.1 Comprensión del Negocio

Determinación de los Objetivos Comerciales

4.2.1.1 Contexto

El estudio de los patrones de agrupación de defunciones correspondiente a problemas cardiacos en Ecuador es fundamental para generar datos clave que orienten el diseño de políticas sanitarias específicas, esto permite identificar a los sectores de la población más vulnerables, lo que a su vez facilita la distribución adecuada de recursos y la puesta en marcha de estrategias de prevención y manejo de enfermedades por problemas cardiacos.

4.2.1.2 Definición de los Objetivos del Negocio

Detectar patrones de agrupación vinculados a las defunciones por problemas cardiacos en Ecuador, con el objetivo de ofrecer información valiosa que impulse el desarrollo de políticas de salud orientadas a las características demográficas y efectivas.

4.2.1.3 Criterios de Rendimiento

Detectar patrones relevantes que resalten las características distintivas de cada grupo demográfico.

4.2.1.4 Evaluación de la Situación

Mediante los datos del registro estadístico de Egresos Hospitalarios, se pretende detectar patrones y tendencias determinantes en cada grupo demográfico, en este contexto, la comparación entre segmentos resulta fundamental para valorar el impacto de las decisiones que serán tomadas por las entidades hospitalarias públicas y privadas del país.

Algunos de los principales riesgos que podrían manifestarse durante la ejecución de este proyecto incluyen la selección inadecuada del número de clusters, la presencia de valores faltantes o incorrectos que podrían afectar la calidad de los resultados, y la interpretación subjetiva de los resultados, que puede depender de la perspectiva del analista.

La interdependencia entre las variables categóricas puede influir en la exactitud del modelo, adicionalmente, puede haber dificultades al evaluar los resultados, ya que las métricas no son aplicables en todos los contextos.

4.2.2 Determinación de los Objetivos de Minería de Datos

4.2.2.1 Objetivos Aplicados a Clustering

Aplicar técnicas de clustering con el objetivo de descubrir patrones, correlaciones y tendencias significativas en la mortalidad en el Ecuador.

4.2.2.2 Criterios de Rendimiento

Para evaluar el rendimiento de la técnica de clustering se utilizará la métrica de Silhouette Score, un valor más alto indica una mejor calidad de los conjuntos de datos de cada grupo, donde los objetos están más cerca de los miembros de su propio grupo y más alejados de los miembros de otros grupos.

4.2.2.1 Plan de Proyecto

En la tabla 3 se desarrolla la planificación del plan del modelo a realizar el cual contiene la fase, la duración, los recursos, los riesgos del desarrollo de la clusterización, y sus diferentes etapas que se va a desarrollar.

Tabla 3: Plan del proyecto de fin titulación clustering

Fase	Duración	Recursos	Riesgos
Comprensión del negocio	1 semana	Responsable del proyecto	
Comprensión de los datos	2 semanas	Acceso a datos históricos sobre egresos hospitalarios	
Preparación de los datos	3 semanas		Excesivos datos faltantes
Modelado	4 Semanas	Plataforma para el modelado (Jupyter Notebook)	Dificultad para encontrar el modelo adecuado
Evaluación	1 Semana		

4.2.3 Comprensión de los Datos

4.2.3.1 Recopilación de Datos Iniciales

La información acerca de las defunciones por enfermedades cardíacas en Ecuador fue proporcionada por el Instituto Nacional de Estadística y Censos en formato SPSS, estos registros de datos fueron descargados, importados y analizados en Jupyter Notebook con el propósito de ser procesados, adicionalmente el intervalo de tiempo abarca desde el año 2010 hasta 2023.

4.2.3.2 Descripción de los Datos

El archivo que contiene los datos de problemas cardiacos tiene 119.669 registros y el archivo que contiene las defunciones del caso tiene 20.820 registros, después de esto se realizó la integración de los datos en el archivo definitivo para el estudio se tiene 20.820 registros.

Como se puede observar en la figura 18, la mayor parte de los tipos de datos son categóricos,

Figura 18: Verificación de tipos de datos

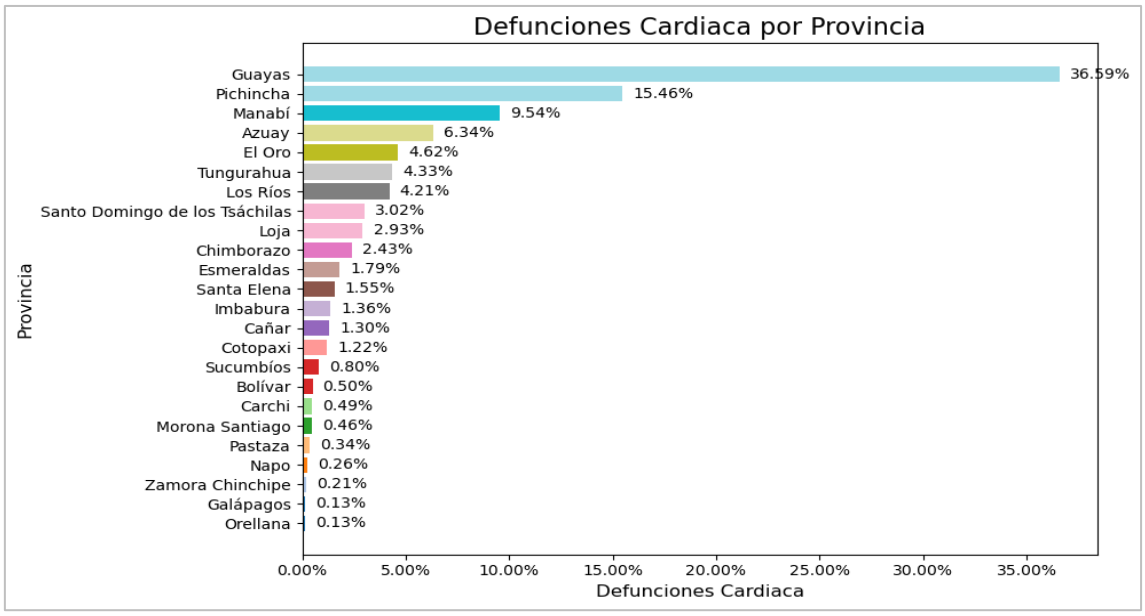
prov_ubi	object
cant_ubi	int64
parr_ubi	int64
area_ubi	object
clase	object
tipo	object
entidad	int64
sector	int64
mes_inv	int64
sexo	int64
cod_edad	int64
edad	int64
etnia	int64
prov_res	int64
cant_res	int64
parr_res	int64
area_res	int64
anio_ingr	int64
mes_ingr	int64
dia_ingr	int64
fecha_ingr	object
anio_egr	int64
mes_egr	int64
dia_egr	int64
fecha_egr	object

dia_estad	int64
con_egrpa	int64
esp_egrpa	int64
cau_cie10	object
causa3	object
cap221rx	int64
cau221rx	int64
cau298rx	int64
etnia_recodificada	object
sexo_recodificado	object
Codc10	object
AFECCIÓN_PRINCIPAL	object
Categoria 3 Subtitulos	object
descrp	object
entidad_recodificado	object
edad_c	float64
rango_edad_c	object
des_3d	object
dia	int64
fecha_def	object
dtype: object	

4.2.3.3 Exploración de Datos

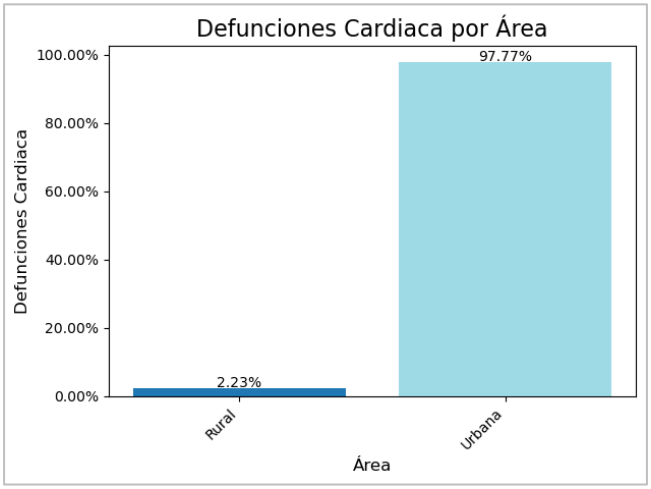
En esta etapa se llevó a cabo una exploración inicial de los datos y las variables de las defunciones por problemas cardiacos que se utilizó para la construcción del modelo de agrupamiento, además, se realizará un análisis estadístico fundamental y se generarán visualizaciones para entender el comportamiento de cada variable y su contribución al modelo.

Gráfico 16: Defunciones Cardiacas por Provincia



En el grafico 16 se enfatiza la provincia de Guayas lidera con un significativo 36.59% de las defunciones cardíacas, lo que representa su alta densidad poblacional y su principal rol como centro económico y urbano del país, concentrando la mayor parte de la atención médica y los factores de riesgo asociados a enfermedades por problemas cardiacos, le sigue Pichincha con un 15.46%, lo que también es considerable dado que incluye la capital del Ecuador, Quito, una ciudad con una población considerable y un entorno urbano que podría estar relacionado con un estilo de vida sedentario y factores de riesgo como la alimentación y el estrés, al contrario, Manabí con un 9.54% y Azuay con un 6.34% representan porcentajes más bajos, lo que podría reflejar una menor urbanización y un mayor dominio de zonas rurales donde las enfermedades por problemas cardiacos, podrían estar menos documentadas debido al acceso limitado a servicios de salud. A diferencia de, las provincias con menores porcentajes, como Napo con un 0.26%, Zamora-Chinchipe con un 0.21%, Galápagos con un 0.13% y Orellana con un 0.13%, presentan cifras mucho menores, lo que podría deberse a la densidad poblacional ya que es mucho menor, o un estilo de vida más activo en zonas rurales o una menor prevalencia de factores de riesgo, este comportamiento resalta la necesidad de políticas de salud diferenciadas según las condiciones de cada provincia, con especial atención a las áreas urbanas con mayores tasas de mortalidad por problemas cardiacos.

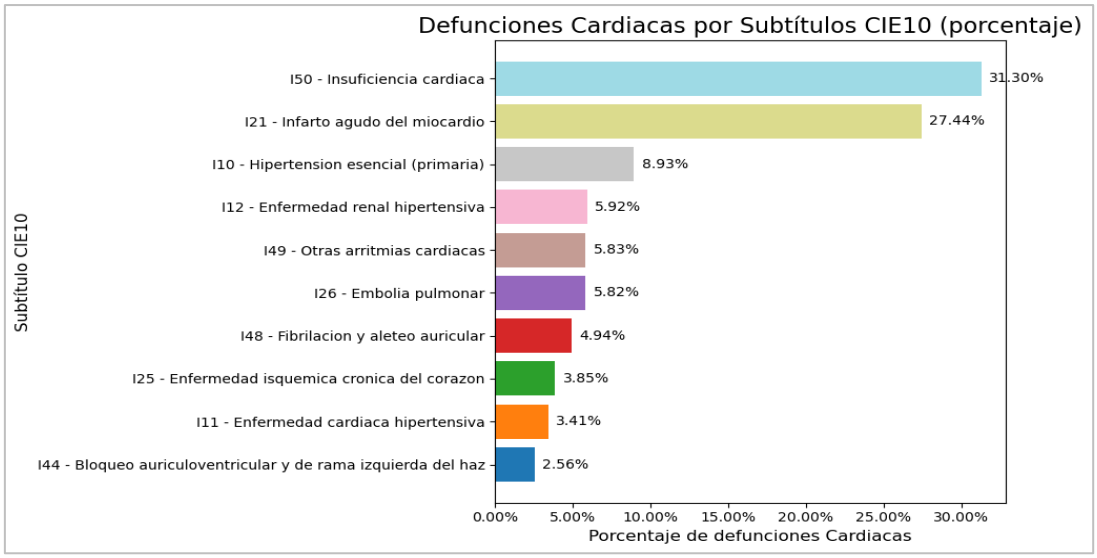
Gráfico 17: Defunciones Cardiacas por Área



En el Gráfico 17 se observa que la gran mayoría de las defunciones por problemas cardíacos se concentran en el área urbana, representando un 97.77%, lo que refleja la prevalencia de factores de riesgo asociados a un estilo de vida urbano, como la mala alimentación, el sedentarismo y el estrés, consecutivamente el 2.23% de las defunciones ocurren en el área rural, lo que podría atribuirse a una menor urbanización, un estilo de

vida más activo y posiblemente un acceso limitado a servicios de salud y problemas con el registro de la información es decir la presencia de subregistro.

Gráfico 18:Defunciones Cardiacas por subtitulo Cie10

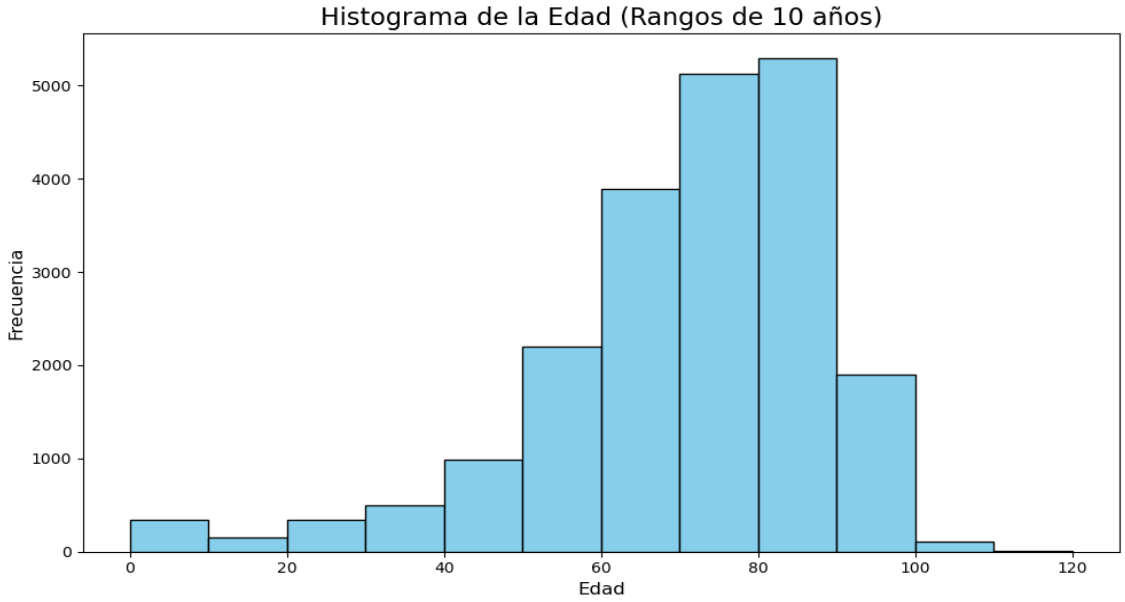


El Gráfico 18 se observa la distribución de las defunciones cardíacas según los subtítulos de la CIE-10, destacando que la insuficiencia cardíaca I50 es la principal causa, con un 31.30% de las muertes, seguida por el infarto agudo de miocardio I21 con un 27.44% es decir estas dos condiciones reflejan la alta prevalencia de enfermedades por problemas cardiacos en la data estudiada, influenciadas por factores de riesgo como la hipertensión, el sedentarismo y la obesidad, por otro lado la hipertensión esencial I10, que contribuye con el 8.93% de las defunciones, es otro factor importante, ya que es una condición crónica que, si no se maneja adecuadamente, puede derivar en complicaciones graves como el infarto y la insuficiencia cardíaca, en este sentido la enfermedad cardiaca hipertensiva I11 y el bloqueo auricular I44 tienen una menor incidencia, con 3.41% y 2.56% de las muertes, correspondientemente. Este comportamiento subraya la importancia de la prevención y el manejo adecuado de las enfermedades por problemas cardiacos, especialmente la hipertensión y las condiciones subyacentes, la concientización y la educación en salud y el acceso a atención médica oportuna son clave para reducir la mortalidad y mejorar la calidad de vida de los pacientes, destacando la necesidad de políticas públicas enfocadas en la prevención de enfermedades.

En el gráfico 19 se puede apreciar que la mayoría de los fallecimientos se concentra en el rango de edad de entre 50 y 100 años de edad, además se destaca una presencia significativa de muertes en el grupo de cero a diez años, a partir de este rango de edad, el

número de defunciones por problemas cardiacos crece de manera progresiva hasta llegar a su pico alrededor de los 100 años.

Gráfico 19: Histograma de Edad



4.2.3.4 Verificación de la Calidad de los Datos

Figura 19:Verificación de valores nulos

prov_ubi	0
cant_ubi	0
parr_ubi	0
area_ubi	0
clase	0
tipo	0
entidad	0
sector	0
mes_inv	0
sexo	0
cod_edad	0
edad	0
etnia	0
prov_res	0
cant_res	0
parr_res	0
area_res	0
anio_ingr	0
mes_ingr	0
dia_ingr	0
fecha_ingr	0
anio_egr	0
mes_egr	0
dia_egr	0
fecha_egr	0

dia_estad	0
con_egrpa	0
esp_egrpa	0
cau_cie10	0
causa3	0
cap221rx	0
cau221rx	0
cau298rx	0
etnia_recodificada	0
sexo_recodificado	0
Codc10	0
AFECCIÓN_PRINCIPAL	0
Categoria 3 Subtitulos	0
descrp	0
entidad_recodificado	0
edad_c	0
rango_edad_c	0
des_3d	0
dia	0
fecha_def	0
dtype: int64	

En la figura 19, se verifica que no existen valores nulos debido al tratamiento inicial de los datos que se realizó.

Figura 20: Verificación de calidad de los datos

```
    data_clustering['prov_ubi'].unique()
```

```
array(['Guayas', 'Loja', 'Manabi', 'Bolivar', 'Azuay', 'Los Ríos',
       'El Oro', 'Chimborazo', 'Santo Domingo de los Tsáchilas',
       'Pichincha', 'Orellana', 'Cotopaxi', 'Napo', 'Imbabura',
       'Tungurahua', 'Esmeraldas', 'Santa Elena', 'Pastaza', 'Galápagos',
       'Cañar', 'Zamora Chinchipe', 'Carchi', 'Sucumbíos',
       'Morona Santiago'], dtype=object)
```

```
data_clustering['area_ubi'].unique()
```

```
array(['Urbana', 'Rural'], dtype=object)
```

```
data_clustering['sexo_recodificado'].unique()
```

```
array(['Mujer', 'Hombre'], dtype=object)
```

```
data_clustering['etnia_recodificada'].unique()
```

```
array(['Ignorado/a', 'Mestizo/a', 'Otro/a', 'Montubio/a', 'Indígena',
       'Blanco/a', 'Negro/a', 'Afroecuatoriano/a Afrodescendiente',
       'Mulato/a'], dtype=object)
```

En la figura 20 se observa el código para obtener la categoría de las variables que se van usar en el respectivo análisis.

4.2.4 Preparación de los Datos

Selección de Datos

Para seleccionar los atributos, se considera el enfoque de clustering basado en k modes el cual requiere únicamente de variables categóricas, el resto de las variables no se consideran relevantes para el presente análisis.

Figura 21: selección de variables de interés.

```
Colums_select= ['area_ubi','prov_ubi','sexo_recodificado','etnia_recodificada','fecha_def','edad_c','entidad_recodificado','tipo']
data_clustering= egresos_final[Colums_select]
data_clustering.head()
```

	area_ubi	prov_ubi	sexo_recodificado	etnia_recodificada	fecha_def	edad_c	entidad_recodificado	tipo
0	Urbana	Guayas	Mujer	Ignorado/a	2023-07-01	57.0	Privados con fines de lucro	Sin Tipo (Hospitales Básicos)
1	Urbana	Loja	Mujer	Mestizo/a	2023-03-01	84.0	Privados con fines de lucro	Agudo
2	Urbana	Guayas	Hombre	Ignorado/a	2023-01-01	46.0	Junta Beneficencia de Guayaquil	Agudo
3	Urbana	Manabí	Hombre	Mestizo/a	2023-01-01	62.0	Privados con fines de lucro	Agudo
4	Urbana	Guayas	Hombre	Mestizo/a	2023-01-01	79.0	Ministerio de Salud Pública	Agudo

En la figura 21 se observa el código para seleccionar solo las variables que se van a utilizar en el análisis clúster.

Limpieza de datos

Es relevante y necesario destacar que antes de llegar a esta etapa se realiza la limpieza de los datos lo cual comprende la eliminación de datos como exterior de la provincia de fallecimiento además la recodificación por la necesidad de las diferentes variables a utilizar, como es el caso de prov_ubi y de fecha_def.

Figura 22: Código de recodificación

```
# Aplicar la recodificación inversa con lambda y apply
egresos_final['prov_ubi'] = egresos_final['prov_ubi'].apply(
    lambda x: 'Azuay' if x == '1' else
              'Bolívar' if x == '2' else
              'Cañar' if x == '3' else
              'Carchi' if x == '4' else
              'Cotopaxi' if x == '5' else
              'Chimborazo' if x == '6' else
              'El Oro' if x == '7' else
              'Esmeraldas' if x == '8' else
              'Guayas' if x == '9' else
              'Imbabura' if x == '10' else
              'Loja' if x == '11' else
              'Los Ríos' if x == '12' else
              'Manabí' if x == '13' else
              'Morona Santiago' if x == '14' else
              'Napo' if x == '15' else
              'Pastaza' if x == '16' else
              'Pichincha' if x == '17' else
              'Tungurahua' if x == '18' else
              'Zamora Chinchipe' if x == '19' else
              'Galápagos' if x == '20' else
              'Sucumbíos' if x == '21' else
              'Orellana' if x == '22' else
              'Santo Domingo de los Tsáchilas' if x == '23' else
              'Santa Elena' if x == '24' else x
)
```

En la figura 22 se observa que el código utiliza la función apply() junto con una expresión lambda para recodificar los valores numéricos de la columna 'prov_ubi' del DataFrame egresos_final, en función del valor numérico, se asigna el nombre correspondiente de la provincia ecuatoriana, si el valor no coincide con ninguno de los casos especificados, se deja sin cambios.

Figura 23: Código de estructuración de fecha

```
egresos_final['dia'] =1
```

```
egresos_final['fecha_def'] = egresos_final['anio_egr'].astype(str) + '-' + egresos_final['mes_egr'].astype(str)+ '-' + egresos_final['dia'].astype(str)
```

```
egresos_final['fecha_def']
```

```
0         2023-7-1
1         2023-3-1
2         2023-1-1
3         2023-1-1
4         2023-1-1
          ...
20815    2010-10-1
20816    2010-11-1
20817     2010-8-1
20818     2010-8-1
20819     2010-2-1
Name: fecha_def, Length: 20820, dtype: object
```

El código de la figura 23 crea una nueva columna llamada 'fecha_def' en el DataFrame egresos_final, concatenando las columnas anio_egr, mes_egr y día en formato de cadena

la misma que convierte cada una de estas columnas a tipo str y las une con un guion entre ellas, formando una fecha en formato 'YYYY-MM-DD'.

Figura 24: Código para eliminar valores vacíos

```
data_clustering.dropna()
data_clustering.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20820 entries, 0 to 20819
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   area_ubi              20820 non-null  object
 1   prov_ubi              20820 non-null  object
 2   sexo_recodificado     20820 non-null  object
 3   etnia_recodificada    20820 non-null  object
 4   fecha_def             20820 non-null  object
 5   edad_c                20820 non-null  float64
 6   entidad_recodificado  20820 non-null  object
 7   tipo                  20820 non-null  object
dtypes: float64(1), object(7)
memory usage: 1.3+ MB
```

En la figura 24 se observa que el código tiene como objetivo eliminar las filas con valores nulos del Data Frame data_clustering utilizando el método dropna(), asegurando que solo queden registros completos, seguido de esto, se utiliza el método info() para obtener información detallada sobre el DataFrame, como el número de filas, columnas y el tipo de datos de cada columna.

Construcción de Rango de edad

Para analizar la edad correctamente se proceden a generar rangos para edad para cada registro:

- Niños (0-12 años)
- Adolescentes (13-19 años)
- Adultos jóvenes (20-39 años)
- Adultos de mediana edad (40-59 años)
- Adultos de mayor edad (60 años +)

La construcción de estos rangos de edad permite obtener una mejor visualización en el desarrollo del modelo, además la eliminación de posibles valores atípicos en la edad, centrando únicamente en los grupos desarrollados.

Figura 25: Código de recodificación de edad

```
import pandas as pd

# Definir los rangos de edad
bins = [0, 12, 19, 39, 59, float('inf')]  # Definir los límites de los rangos
labels = ['Niños', 'Adolescentes', 'Adultos jóvenes',
          'Adultos de mediana edad', 'Adultos de mayor edad']

# Crear una nueva columna 'rango_edad' en data_clustering asignando los rangos de edad
data_clustering['rango_edad'] = pd.cut(data_clustering['edad_c'], bins=bins, labels=labels, right=True)

# Mostrar el resultado
print(data_clustering[['edad_c', 'rango_edad']].head())
```

```
   edad_c              rango_edad
0    57.0  Adultos de mediana edad
1    84.0    Adultos de mayor edad
2    46.0  Adultos de mediana edad
3    62.0    Adultos de mayor edad
4    79.0    Adultos de mayor edad
```

En la figura 25 se observa que el código define los rangos de edad necesarios para el análisis y los asigna a la columna rango_edad en el DataFrame data_clustering usando la función pd.cut(), dichos rangos son, niños, adolescentes, adultos jóvenes, adultos de mediana edad y adultos mayores y por consiguiente, se elimina la columna edad_c y se muestra el DataFrame resultante.

Formateo de Datos

Figura 26: Código de verificación del tipo de dato

```
data_clustering.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20820 entries, 0 to 20819
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   area_ubi              20820 non-null  object
 1   prov_ubi              20820 non-null  object
 2   sexo_recodificado     20820 non-null  object
 3   etnia_recodificada    20820 non-null  object
 4   fecha_def             20820 non-null  object
 5   entidad_recodificado  20820 non-null  object
 6   tipo                  20820 non-null  object
 7   rango_edad            20820 non-null  category
dtypes: category(1), object(7)
memory usage: 1.1+ MB
```

En la figura 26 se observa la presencia de variables las cuales contienes datos de tipo que no corresponde adecuadamente con la información que contienen, esto indica que algunas variables están registradas con tipos de datos incorrectos, lo que podría generar inconsistencias en el análisis y dificultar su interpretación precisa.

Figura 27: Código de modificación de tipo de dato

```
data_clustering['area_ubi']=data_clustering['area_ubi'].astype('category')
data_clustering['prov_ubi']=data_clustering['prov_ubi'].astype('category')
data_clustering['sexo_recodificado']=data_clustering['sexo_recodificado'].astype('category')
data_clustering['etnia_recodificada']=data_clustering['etnia_recodificada'].astype('category')
data_clustering['fecha_def']=data_clustering['fecha_def'].astype('category')
data_clustering['entidad_recodificado']=data_clustering['entidad_recodificado'].astype('category')
data_clustering['tipo']=data_clustering['tipo'].astype('category')
```

En la figura 27 se observa que el código convierte varias columnas del DataFrame en el tipo de dato category, al realizarlo, este proceso tiene como objetivo optimizar el uso de la memoria y se mejora la eficiencia en el procesamiento de datos categóricos, lo que facilita su análisis y operaciones, como el agrupamiento y la codificación de variables.

4.2.5 Modelado

4.2.5.1 Selección de Técnicas de Modelado

Con el objetivo de descubrir patrones, correlaciones y tendencias significativas en las muertes por enfermedades cardiacas en Ecuador, y considerando que la mayoría de los datos son de carácter categórico, es necesario utilizar el algoritmo k-modes, este algoritmo está diseñado específicamente para trabajar con variables categóricas, lo que permite realizar un análisis más efectivo y obtener resultados significativos a partir de este tipo de información.

4.2.5.2 Modelado de supuestos

Para la aplicación del algoritmo k-modes en el análisis de muertes por enfermedades cardiacas en Ecuador, se describen los supuestos bajo los cuales el algoritmo funcionará de manera efectiva, como pueden ser la naturaleza categórica de los datos, la independencia de las observaciones y la necesidad de un número adecuado de clústeres.

4.2.5.3 Diseño de Comprobación

En este paso es necesario evaluar la efectividad de la técnica de agrupamiento utilizada, para esto se utilizó la métrica Silhouette Score, que mide la calidad de los clústeres generados.

4.2.5.4 Generación de Modelos

En primer lugar, se procede a la identificación eh instalación del algoritmo k-modes en el entorno de trabajo Jupiter Notebook, seguido de esto, se importan las bibliotecas necesarias para la ejecución del algoritmo k-modes.

Figura 28: Librerias de KModes

```
import pandas as pd
from kmodes.kmodes import KModes
```
✓ 1.0s

En la figura 28 se ejecuta el código de instalación de la librería de KModes esta librería proporciona las funciones y herramientas esenciales para realizar el proceso de clusterización de datos categóricos de manera eficiente.

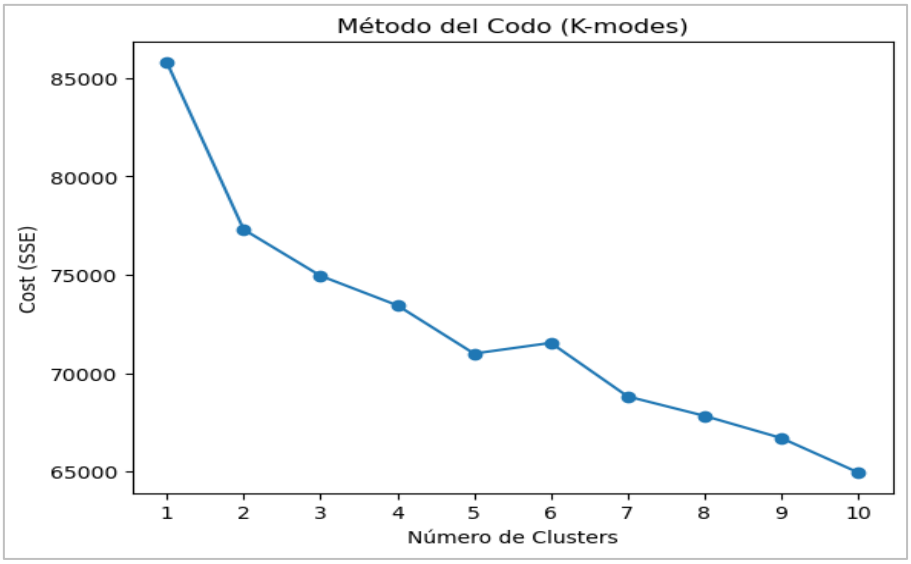
En la figura 29 se observa que el código ajusta un modelo de clustering usando el algoritmo k-modes para diferentes valores de k en este caso de 1 a 10 clústeres, una por cada iteración, el modelo se ajusta a los datos es decir en la data data_clustering y calcula el costo del clustering y adicionalmente los costos resultantes se almacenan en una lista llamada costos.

Figura 29: Código de determinación del número optimo de clústeres

```
costos = []

# Probar diferentes valores de k (número de clusters)
for k in range(1, 11):  # Probar de 1 a 10 clusters
    k_modes = KModes(n_clusters=k, init='Cao', n_init=5, verbose=1)
    k_modes.fit(data_clustering)  # Ajustar el modelo
    costos.append(k_modes.cost_)
```

Los centroides se inician utilizando el método Huang, además se define cuántas veces se ejecutará el algoritmo K-modes con diferentes centroides iniciales, y se ajusta el nivel de verbosidad para mostrar detalles sobre el proceso de agrupamiento y se establece una semilla aleatoria.

Gráfico 20: Método Elbow para determinar el número óptimo de clusters



En el gráfico 20, se observa un leve codo en el clúster 3 y otro en el clúster 5, de igual manera, se observa que a partir del clúster 6, el costo disminuye más lentamente, en este sentido se probarán estos valores específicos para luego evaluar los resultados obtenidos.

En la figura 30 se inserta y se parametriza el modelo con base de k=3, se inicializa con el método Huang y realiza 10 inicializaciones para mejorar la convergencia, por su parte la función fit_predict() asigna cada punto de datos a uno de los clústeres y devuelve las etiquetas de los clústeres a las que pertenecen los elementos en la data.

Figura 30: Código para clustering

```
kmode2= KModes(n_clusters=3,init="Huang",n_init=10,verbose=1,random_state=0)
clusters2=kmode2.fit_predict(data_clustering)
clusters2
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 4441, cost: 45916.0
Run 1, iteration: 2/100, moves: 788, cost: 45916.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 6058, cost: 51466.0
Run 2, iteration: 2/100, moves: 1112, cost: 51466.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 4461, cost: 50458.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 3571, cost: 59279.0
Run 4, iteration: 2/100, moves: 506, cost: 59279.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 4485, cost: 50789.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 6, iteration: 1/100, moves: 3179, cost: 52006.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 7, iteration: 1/100, moves: 2189, cost: 45916.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 8, iteration: 1/100, moves: 3987, cost: 50615.0
Run 8, iteration: 2/100, moves: 513, cost: 50615.0
```

Figura 31: Código para insertar etiquetas

```
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
```

En la figura 31 se insertan las etiquetas además se crea una copia del DataFrame data_clustering y la asigna el nombre a la data datc, seguido de esto, se inserta una nueva columna llamada cluster_labels en la posición 0 y al principio del DataFrame utilizando los valores contenidos en los clústeres, que corresponden a las etiquetas de los clústeres

asignadas en el paso previo, y el argumento True asegura que, si ya existe una columna con ese nombre, se reemplazará, respectivamente esto permite agregar la información de los clústeres al DataFrame original de manera organizada.

Figura 32: Código para importar librerías

```
!pip install gower
!pip install scikit-learn
import gower
from sklearn.metrics import silhouette_score
```

En la figura 32 se observa que se instala e importa bibliotecas esenciales para realizar clustering y evaluar su rendimiento, en primer lugar, instala gower, que permite calcular distancias entre datos mixtos es decir numéricos y categóricos, después se instala scikit-learn, una librería primordial para machine learning y aprendizaje automático en Python, después de esto, se importa gower para usar sus funciones y silhouette_score de scikit-learn, una métrica que evalúa la calidad del clustering al medir la cohesión dentro de los clústeres y la separación entre ellos, estas herramientas son clave para aplicar y evaluar técnicas de agrupamiento de datos.

Figura 33: Código para obtener una muestra y modificar el tipo de datos

```
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
```

En la figura 33 debido al alto costo computacional al calcular el Silhouette Score, se opta por trabajar con una muestra de 9.000 registros, además se establece una semilla aleatoria para asegurar la reproducibilidad de los resultados, y previamente se transforman los valores en cadenas de texto.

Figura 34: Código para obtener la distancia de Gower y calcular el Silhouette Score

```
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))

# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')

# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Obteniendo el resultado del Siluette Score= 0.4103 para 3 clúster mientras que:

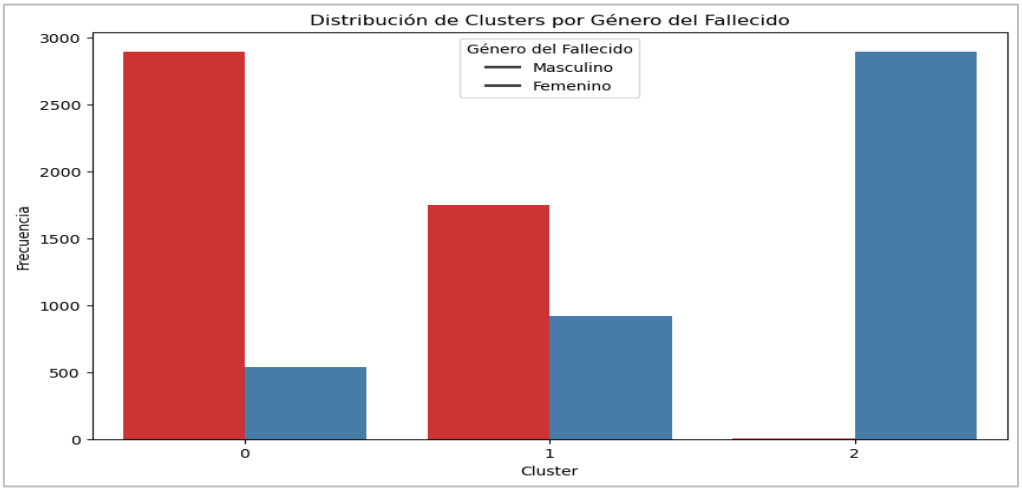
Clúster 4= 0.4041

Clúster 5= 0.3973

Clúster 6= 0.4031

Por lo tanto, según los resultados obtenidos, se observa que el valor máximo del Silhouette Score se alcanza al utilizar 3 clústeres, y por consiguiente es necesario analizar si estos resultados si son válidos y si puede ser interpretado de manera adecuada.

Gráfico 21: Distribución de los clústeres a nivel del género del fallecido por problemas cardiacas



El gráfico 21 muestra que, en los 3 clústeres, los datos se dividen en dos grupos en los clústeres 0 y 1, diferenciando entre hombres y mujeres, mientras que en el clúster 2 solo se agrupan hombres. Por lo tanto, estos resultados no tienen una interpretación válida, lo que lleva a la decisión de reducir el número de variables consideradas para el análisis.

Figura 35: Código para seleccionar nuevas variables

```
Colums_select= ['area_ubi','prov_ubi','etnia_recodificada','edad_c','entidad_recodificado','tipo','Codc10']
data_clustering= egresos_final[Colums_select]
data_clustering.head()
```

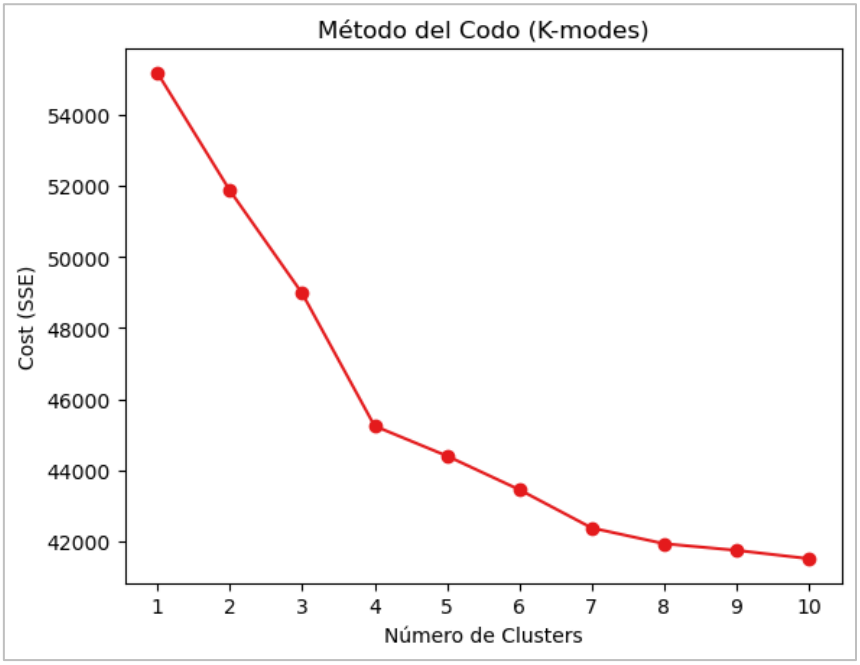
En la figura 35, se selecciona el área de ubicación, la provincia de ubicación, la etnia, la edad, la entidad, el tipo y la causa de muerte.

Figura 36: Código para determinar el número óptimo de clústeres

```
costos = []

# Probar diferentes valores de k (número de clusters)
for k in range(1, 11):  # Probar de 1 a 10 clusters
    k_modes = KModes(n_clusters=k, init='Cao', n_init=5, verbose=1)
    k_modes.fit(data_clustering)  # Ajustar el modelo
    costos.append(k_modes.cost_)
```

En la figura 36 se determina el numero optimo de clústeres, para ello, prueba con 1 a 10 clústeres y calcula el costo, una medida de qué tan similares son los elementos dentro de cada clúster para cada número, guardando estos costos en una lista.

Gráfico 22: Método Elbow para determinar el número óptimo de clústeres



En el gráfico 22 se observa que existe un ligero codo en el clúster número 4, y otro en el clúster número 4 posterior al clúster número 6 aumenta el coste, por ende, se procederá a probar diferentes números de clústeres.

4.2.6 Evaluación de los Modelos

Después de completar el proceso de entrenamiento de los modelos en cada clúster, se obtienen los resultados correspondientes al Silhouette Score.

Clúster 3= 0.5022

Clúster 4= 0.4796

Clúster 5= 0.4801

En base a los resultados el clúster número 3 obtiene el valor más alto para el Average Silhouette Score.

Para representar el clúster, se reduce la dimensionalidad del conjunto de datos a dos variables mediante el uso de Análisis de Correspondencias Múltiples, el MCA se utiliza para analizar y representar visualmente datos categóricos, facilitando la identificación de patrones y relaciones entre las categorías de distintas variables.

Figura 37: Código para transformar las variables categóricas a dummies

```
import pandas as pd

# Aplicar pd.get_dummies para las columnas que quieres convertir a variables dummies
datc = pd.get_dummies(datc, columns=['area_ubi', 'prov_ubi', 'etnia_recodificada',
                                     'entidad_recodificado', 'Codc10', 'tipo', 'rango_edad'],
                      drop_first=False)

# Verifica el resultado
datc.head()
```

En la figura 37, primero se codifican las categorías de las variables, luego se calculan las distancias para evaluar la similitud o diferencia entre ellas, después se aplica la descomposición en valores singulares para reducir la dimensionalidad de los datos, y finalmente se generan los gráficos que permiten observar las distancias.

Además se procedió a instalar la librería prince con el siguiente código !pip install prince import prince la cual es necesaria para el proceso MCA.

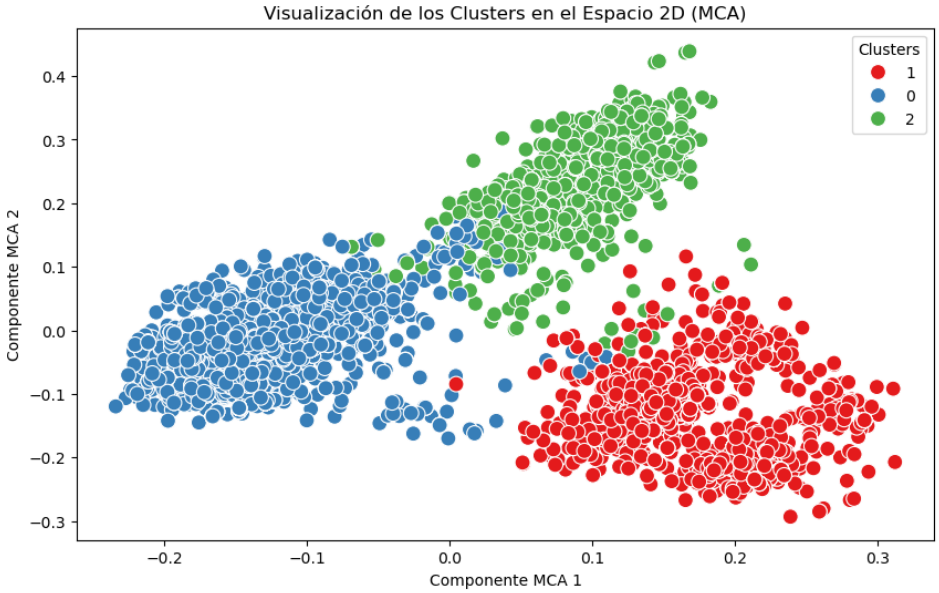
Figura 38: Código para inicializar y ajustar el objeto MCA y para crear las coordenadas

```
mca= prince.MCA(n_components=2)
mca=mca.fit(datc)
mca_coordinates=mca.transform(datc)
```

```
# Supongamos que 'cluster_labels' es la columna que contiene las etiquetas de los clusters
# Añadir las coordenadas al DataFrame para facilitar la visualización
datc['MCA_1'] = mca_coordinates[0]
datc['MCA_2'] = mca_coordinates[1]

# Visualización en un gráfico de dispersión
plt.figure(figsize=(10, 6))

# Utilizar seaborn para crear un gráfico de dispersión con clusters
sns.scatterplot(x='MCA_1', y='MCA_2', hue='cluster_labels', data=datc, palette='Set1', s=100)
```

En la figura 38 se crea el objeto MCA y se definen dos componentes principales, después de esto, se adapta a los datos al objeto MCA, lo que permite calcular las coordenadas correspondientes a las categorías y, por último, se realiza la transformación del conjunto de datos utilizando el modelo MCA ajustado, lo mismo que genera las coordenadas de cada registro en los nuevos ejes establecidos por los componentes principales.

Gráfico 23: Visualización de los clústeres



En el gráfico 23 se observa que los diferentes clústeres están ubicados muy cerca entre sí, lo que sugiere que las muestras dentro de cada grupo comparten similitudes o características comunes es decir la proximidad podría ser una indicación de que los registros presentan patrones o comportamientos similares, además de que el hecho de que los clústeres no se separen considerablemente también podría indicar que las diferencias entre ellos no son tan marcadas.

CAPITULO 5 ANALISIS DE RESULTADOS

5.1 Aplicación de métodos y técnicas de pronóstico de series de tiempo para predecir el comportamiento de las defunciones Hospitalarias por problemas cardiacos

En base a las métricas utilizadas, el metodo Holt-Winters Multiplicativo se destaca como el más preciso para predecir las defunciones por problemas cardiacos. Este método presenta los menores errores entre los modelos evaluados, lo que evidencia una mayor exactitud en sus predicciones. A diferencia del estudio de Jiménez (2024), en el que se analiza el comportamiento de la mortalidad en Ecuador mediante el uso de machine learning, se concluye que el método SARIMA es el más adecuado para predecir las defunciones.

Los gráficos generados para comparar las predicciones con los datos de validación revelan que el método Holt-Winters Multiplicativo es capaz de capturar eficazmente las tendencias y los períodos de estacionalidad. Sin embargo, se observa que las predicciones tienden a estar por debajo de los valores reales utilizados para la evaluación.

5.1.1. Despliegue

Para el desarrollo de las predicciones de las defunciones por problemas cardiacos se utiliza el modelo autorregresivo Holt-Winters Multiplicativo, el cual posee las mejores métricas de evaluación con respecto al resto de modelos.

Figura 39: Código modelo Holt-Winters Multiplicativo

```
# Modelo Holt-Winters con tendencia y estacionalidad multiplicativa
model_multiplicative = ExponentialSmoothing(
    np.asarray(data1['Defunciones_Cardiacas']),  # Datos de entrenamiento
    seasonal_periods=12,  # Ajusta según el número de periodos estacionales (12 para datos mensuales)
    trend='additive',      # Tendencia aditiva (puedes probar 'multiplicative' si deseas)
    seasonal='multiplicative'  # Componente estacional multiplicativo
)
```

En la Figura 39, se visualiza claramente la ejecución del modelo Holt-Winters multiplicativo, en el gráfico 24, la tendencia aditiva y las variaciones estacionales multiplicativas se reflejan de manera clara y precisa, permitiendo una mejor comprensión de cómo el modelo ajusta las predicciones a los datos históricos y cómo captura tanto la tendencia de largo plazo como los patrones estacionales recurrentes.

En la figura 40 se observa que el modelo Holt-Winters Multiplicativo ajusta los parámetros para predecir las defunciones cardíacas teniendo en cuenta tanto la tendencia respectivamente estable, como la estacionalidad con variabilidad moderada, pero sin dar

demasiada importancia a cambios drásticos en estos componentes, la tendencia y las fluctuaciones estacionales se ajustan suavemente, y la predicción se inicia con un nivel inicial de aproximadamente 59.89, con una ligera tendencia ascendente al inicio.

Figura 40: Código de ajuste del Modelo Multiplicativo

```
# Ajustar el modelo
model_multiplicative_fit = model_multiplicative.fit(optimized=True)

# Imprimir los parámetros ajustados del modelo
print("Parámetros del modelo Holt-Winters Multiplicativo:")
print(model_multiplicative_fit.params)
```

```
Parámetros del modelo Holt-Winters Multiplicativo:
{'smoothing_level': 0.39482703844420197, 'smoothing_trend': 4.577788764133791e-05, 'smoothing_seasonal': 0.00013854996422399258, 'damping_trend': nan,
'initial_level': 59.88864563832528, 'initial_trend': 0.13133036239914897, 'initial_seasons': array([1.59972279, 1.44263235, 1.67335564, 1.45563685, 1.56
39127 ,
       1.57117658, 1.60526704, 1.58028875, 1.51772617, 1.53112524,
       1.53183408, 1.45376395]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

Figura 41: Código de predicción a 36 meses

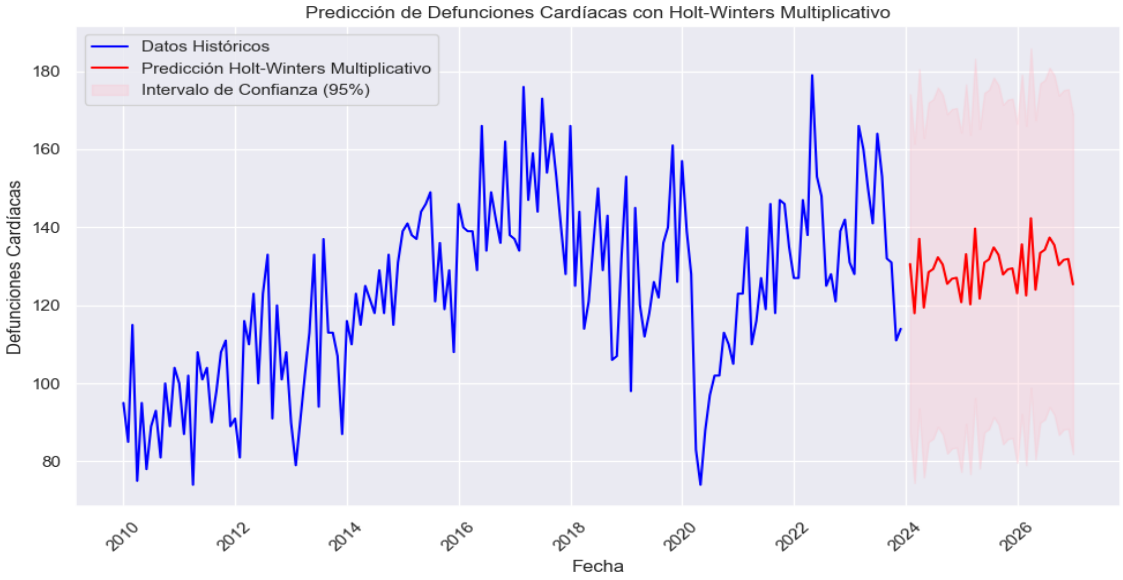
```
# Realizar predicciones a futuro (ejemplo: pronóstico para los próximos 36 meses)
forecast_steps = 36  # Número de pasos a predecir
future_forecast = model_multiplicative_fit.forecast(forecast_steps)

# Crear rango de fechas para las predicciones futuras
last_data1_date = data1.index[-1]  # Última fecha de entrenamiento
```

```
# Verificar si el último mes de datos es diciembre
if last_data1_date.month == 12:
    # Si es diciembre, establece la predicción a partir de enero del siguiente año
    start_date = pd.Timestamp(year=last_data1_date.year + 1, month=1, day=1)
else:
    # Si no es diciembre, simplemente pasa al siguiente mes
    start_date = last_data1_date + pd.Timedelta(days=1)

# Crear el rango de fechas de las predicciones futuras
forecast_index = pd.date_range(start=start_date, periods=forecast_steps, freq='M')  # Fechas futuras

# Mostrar las predicciones a futuro
predicted_df = pd.DataFrame({
    'Fecha': forecast_index,
    'Defunciones Predichas': future_forecast
})

print(predicted_df)
```

```
        Fecha  Defunciones Predichas
0  2024-01-31             130.605066
1  2024-02-29             117.969220
2  2024-03-31             137.060308
3  2024-04-30             119.416429
4  2024-05-31             128.505840
5  2024-06-30             129.306480
```

En la figura 41 se observa que el código realiza un pronóstico de las defunciones cardíacas para los próximos 36 meses, ajustando el inicio de las predicciones dependiendo de si el último mes de los datos históricos es diciembre o no, si el último mes es diciembre, el pronóstico comienza en enero del siguiente año, de lo contrario, se inicia en el siguiente mes. Posteriormente, se generan las fechas futuras y se presentan junto con las predicciones de defunciones en un DataFrame.

Gráfico 24: Pronóstico modelo Holt-Winters Multiplicativo, valores futuros.



5.2. Aplicación de técnicas de clusterización en las defunciones Hospitalarias por problemas cardiacos

Los resultados de la clusterización revelan tres grupos distintos en el conjunto de datos.

Clúster 0

El Clúster 0 se caracteriza por una fuerte representación de adultos de mediana edad y adultos mayores, la mayor concentración de este grupo se encuentra ubicada en las provincias de Guayas y Pichincha, seguidas por Manabí y Los Ríos, en cuanto al tipo de atención médica, hay fuerte tendencia hacia los tipos de servicio agudos y Hospitales Básicos sin especialidad, lo que puede indicar que las personas de este grupo necesitan atención médica inmediata o básica pero los hospitales no tienen respuesta y especialidad en cardiología por ser básicos, también, aunque la presencia de niños y adolescentes es baja, hay algo de representación en el rango de edad de niños y adolescentes, en fin este clúster indica la presencia de individuos de mediana y avanzada edad, con un mayor uso de servicios de salud de tipo urgente, agudos y en hospitales básicos, caracterizado por residencia de personas áreas rurales y en la provincia de Los Ríos.

Clúster 1

En el Clúster 1, predomina la combinación de personas adultas mayores con residencia en áreas urbanas, los valores más altos se encuentran en las variables mencionadas, lo que sugiere que el grupo está formado principalmente por adultos mayores que residen en

áreas urbanas, la concentración se mantiene en las provincias de Guayas y Pichincha, pero también es notable la presencia de Manabí y Azuay, al igual que en el Clúster 0, las personas en este grupo tienden a necesitar atención de tipo inmediata es decir aguda y en Hospitales Básicos sin especialidad, lo que indica que las condiciones médicas no son necesariamente crónicas, sino más bien urgentes y no especializadas es decir estas clínicas que brindan este tipo de emergencias no tienen la especialidad de cardiología, la representación de los rangos de edad más jóvenes es mínima, con una leve presencia de adultos jóvenes y adolescentes, en resumen, este clúster está compuesto principalmente por adultos mayores con residencia en área urbana con necesidades de atención de salud básica y urgente, y en clínicas sin especialidad caracterizado por la ubicación de la provincia de Azuay.

Clúster 2

El Clúster 2 presenta una estructura similar al Clúster 1, con una alta presencia en el área urbana y rango de edad de adultos de mayor edad, pero con una diferencia importante: una mayor proporción de tipo clínicas generales sin especialidad, lo que sugiere que este grupo recibe atención en clínicas generales más que en hospitales especializados, este grupo además tiene una presencia destacada en Guayas y Pichincha, pero también se destaca la presencia de Manabí, aunque la mayoría de las personas en este grupo son adultos mayores, hay una proporción más significativa de adultos de mediana edad y jóvenes adultos que en los otros dos clústeres analizados, y la presencia de niños y adolescentes sigue siendo baja pero existe, en fin, este clúster incluye principalmente adultos mayores con residencia en áreas urbanos, y adicionalmente tiene una representación de otras edades y una inclinación hacia la atención médica en clínicas generales.

5.2.1 Despliegue

Los resultados del proceso de análisis de clústeres realizado presentan tres grupos diferentes en el conjunto de datos, hecho que puede utilizarse para generar una nueva base de datos actualizada que podría generar el nuevo desarrollo de modelos predictivos, y en relación a las características y tendencias de defunciones por problemas cardiacos, los gobiernos podrán tomar las mejores decisiones informadas para diseñar, implementar y efectuar políticas públicas más eficaces y dirigidas a la reducción de las defunciones por problemas cardiacos de todos los grupos demográficos.

CAPÍTULO 6 CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

La aplicación de métodos de series de tiempo a las defunciones por problemas cardiacos en Ecuador, junto con la evaluación y selección de modelos de pronóstico, ha permitido alcanzar los objetivos establecidos, se realizó el análisis de la evolución identificando tendencias y periodos estacionales, además, mediante la técnica de agrupación k-modes, se han identificado patrones en grupos específicos de la data analizada, lo que aporta una comprensión más detallada de los factores que influyen en las defunciones cardiacas.

En función de la aplicación de los métodos de predicción mediante series temporales y la metodología CRISP-DM, se concluye que la evolución de las defunciones hospitalarias por problemas cardíacos presenta fluctuaciones significativas a lo largo de los años, el análisis revela que, aunque hubo un aumento general en las muertes, especialmente en los años 2015 y 2017, se registró una disminución importante en el año 2020, posiblemente influenciada por la pandemia de COVID-19, sin embargo, en el año 2023, el número de defunciones volvió a repuntar, alcanzando cifras más altas, este comportamiento propone que los esfuerzos para controlar las enfermedades cardiovasculares han tenido algunos efectos, pero siguen existiendo factores que perturban las defunciones cardiacas, como el estilo de vida y la prevalencia de enfermedades comórbidas.

El análisis también muestra una mayor prevalencia de defunciones cardíacas en hombres, lo cual puede explicarse por factores biológicos y de comportamiento de riesgo más comunes en este grupo, adicionalmente, se identifica que el infarto agudo de miocardio es la principal causa de muerte, seguido por insuficiencia cardíaca congestiva, lo que resalta la importancia de desarrollar estrategias de prevención más efectivas, asimismo, se observan variaciones en la distribución de las defunciones según la entidad encargada de reportarlas, con el Ministerio de Salud Pública liderando los registros, finalmente, el análisis confirma una conducta de incremento de muertes cardíacas con la edad, reflejando cómo el envejecimiento y los factores de riesgo acumulados a lo largo de la vida inciden de manera significativa en la mortalidad cardiovascular, siendo una elección acertada la metodología CRIS DM ofrece un marco organizado que orienta cada fase del proceso de del análisis de las defunciones por problemas cardiacos, desde la exploración inicial de los datos hasta la evaluación final de los modelos obtenidos, gracias a CRISP-DM, cada etapa del proyecto fue abordada de manera clara y con un objetivo bien establecido.

En función de los métodos aplicados de predicción de series de tiempo como suavizamiento exponencial Holt con tendencia, método multiplicativo de Holt Winters con tendencia y estacionalidad, ARIMA y SARIMA se concluye que, para las defunciones por problemas cardiacos, el método, método multiplicativo de Holt Winters con tendencia y estacionalidad es el más preciso.

Es decir, el modelo multiplicativo de Holt-Winters, incorpora tanto tendencia como estacionalidad, este modelo ha permitido captar con mayor precisión los patrones estacionales y las fluctuaciones de los datos, lo que mejora sustancialmente la capacidad de pronóstico a corto, mediano y largo plazo, de esta manera, se obtiene un instrumento valioso para la planificación y toma de decisiones en el ámbito de la salud pública.

Los tres clústeres comparten características clave, como la presencia dominante de adultos mayores urbanos y la baja representación de rangos de edad más jóvenes, cada uno refleja diferencias importantes en cuanto a las preferencias por servicios médicos, el Clúster 0 se destaca por caracterizarse en el servicio recibido en hospitales básicos, ubicados en la provincia de Los Ríos, el Clúster 1 se centra en la atención de tipo aguda con una ubicación en la provincia del Azuay, y el Clúster 2 muestra una inclinación por las clínicas generales, en todos los casos, las provincias más densamente pobladas, como Guayas y Pichincha, siguen siendo los principales focos de concentración, lo que podría estar relacionado con las altas concentraciones de población con residencia en área urbana que requieren atención médica constante, la baja presencia de niños y adolescentes en todos los clústeres también indica que las necesidades de atención en estos rangos de edad son considerablemente menores en comparación con las de los adultos.

Los grupos identificados en los clústeres muestran diferencias clave en términos de edad, causas de muerte cardiaca y características demográficas, en particular, la provincia del Guayas emerge como una región sustancial en cuanto a la alta tasa de defunciones por problemas cardiacos, lo que sugiere una concentración regional de este tipo de muertes, adicionalmente, se observa que el género masculino es el más afectado por estas afecciones, lo que resalta la importancia de enfocar esfuerzos preventivos en este grupo, estos hallazgos destacan la necesidad urgente de desarrollar políticas de salud pública orientadas a la prevención y tratamiento de enfermedades por problemas cardiacos, con un enfoque específico en las regiones más afectadas.

6.2 Recomendaciones

Llevar a cabo un seguimiento constante de las tendencias de defunciones por problemas cardiacos en Ecuador mediante el uso de análisis de series de tiempo, lo que permitirá identificar de manera temprana cambios relevantes en los patrones y facilitará la planificación de acciones gubernamentales, así como la implementación de políticas y programas de salud preventiva.

Es fundamental llevar a cabo una validación periódica de los modelos de predicción utilizando datos actualizados, con el fin de asegurar la precisión y confiabilidad de las proyecciones a medida que evolucionan las condiciones y factores que influyen en las defunciones por problemas cardiacos, al mismo tiempo, se debe integrar de manera eficaz los resultados de estas predicciones en la planificación y el diseño de políticas de salud pública, esta información consigue ser esencial para la correcta asignación de recursos, el desarrollo y la implementación de programas preventivos, y la toma de decisiones informadas, para optimizar aún más la efectividad de las intervenciones, se recomienda establecer procedimientos de retroalimentación continua entre los datos de defunciones cardiacas y las políticas implementadas, permitiendo ajustar y optimizar las estrategias de salud en tiempo real, respondiendo de manera más eficiente a los cambios en los patrones de salud pública.

Aunque el modelo multiplicativo de Holt-Winters demostró ser el más preciso, sería valioso explorar otros enfoques y técnicas de series de tiempo, como RNN, SVM, Random Forest, modelos de descomposición, entre otros, igualmente se podría considerar la utilización de modelos que integren variables exógenas que impacten en cada una de las afecciones Cie10, lo que permitiría identificar factores clave que influyen en los patrones de defunciones cardiacas, estos modelos alternativos podrían proporcionar información valiosa para la formulación y ejecución de políticas y programas de salud pública más efectivos y adaptados al orden natural dinámico.

Dado que los clústeres identificados revelan patrones específicos relacionados con género, provincia de fallecimiento, rango de edad y causa de muerte, es esencial implementar políticas públicas dirigidas al desarrollo de programas de intervención que consideren las necesidades específicas de estos grupos, en particular, se deben diseñar programas de salud pública centrados en la prevención, detección temprana y manejo de afecciones relacionadas con las defunciones por problemas cardiacos, especialmente en

la provincia del Guayas en zonas urbanas y rurales, además, las entidades encargadas de esto podrían explorar métodos como el análisis de cohortes, redes neuronales o modelos de predicción basados en aprendizaje automático u otros, para personalizar y optimizar estas intervenciones, permitiendo una respuesta más precisa a las problemáticas de salud en Ecuador.

REFERENCIAS

Aguilera, A. (2017). El costo-beneficio como herramienta de decisión en la inversión en actividades científicas. Universidad de la Habana, Cuba, 23.

Aretio, L. G. (2019). Necesidad de una educación digital en un mundo digital. Obtenido de UNED: https://redined.educacion.gob.es/xmlui/bitstream/handle/11162/190710/Necesidad.pdf?sequence=1&isAllowed=y

Barría. (2022). Modelos de Series de Tiempo para Predecir el Número de Casos de Variantes Dominantes del SARS-COV-2 Durante las Olas Epidémicas en Chile. Revista Politécnica, 17-26.

Brito, P. (01 de 07 de 2024). INEC. Obtenido de INEC: https://www.ecuadorencifras.gob.ec/documentos/web-inec/Estadisticas_Sociales/Camas_Egresos_Hospitalarios/2023/Boletin_tecnico_ECEH_2023.pdf

Codificandobits. (16 de 02 de 2025). Codificandobits. Obtenido de Codificandobits: https://codificandobits.com/blog/guia-analisis-de-series-de-tiempo/

Dorado-Díaz et al. (2019). Aplicaciones de la inteligencia artificial en cardiología. Revista Española de Cardiología, 11.

Edsrobotics. (07 de 07 de 2021). Edsrobotics. Obtenido de Edsrobotics: https://www.edsrobotics.com/blog/realidad-aumentada-que-es/

FasterCapital. (01 de 01 de 2023). FasterCapital. Obtenido de FasterCapital: https://fastercapital.com/es/tema/t%C3%A9cnica-de-suavizado-exponencial-simple.html

Feitosa. (01 de 01 de 2020). National Library of Medicine. Obtenido de National Library of Medicine: https://pmc.ncbi.nlm.nih.gov/articles/PMC8959057/

García, C. L. (2021). Análisis de series temporales con R Time series analysis with R. Universidad de Almería, 1-59. Obtenido de https://repositorio.ual.es/bitstream/handle/10835/17675/LOPEZ%20GARCIA%2c%20CELINA.pdf?sequence=1&isAllowed=y

IBM - United States. (15 de 02 de 2025). IBM - United States. Obtenido de IBM - United States: https://www.ibm.com/mx-es/topics/supervised-learning

IBM. (18 de Noviembre de 2024). IBM. Obtenido de IBM: https://www.ibm.com/mx-es/think/topics/big-data

IBM. (24 de 01 de 2025). IBM. Obtenido de IBM: https://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/15.0/es/CRISP-DM.pdf

INEC. (01 de 07 de 2024). INEC. Obtenido de INEC: https://www.ecuadorencifras.gob.ec/documentos/web-inec/Estadisticas_Sociales/Camas_Egresos_Hospitalarios/2023/Boletin_tecnico_ECEH_2023.pdf

Jhoan, C. (2020). EDUCACIÓN UNIVERSITARIA: TRANSICIÓN Y DISRUPCIÓN DIGITAL. Gicos, 130-140.

Koeman, J., David Rockil, & Gernot , s. (01 de 08 de 2019). McKinsey Company. Obtenido de McKinsey Company: https://maestriasenlinea.puce.edu.ec/pluginfile.php/390456/mod_resource/content/1/an-insurance-company-transforms-itself-by-putting-technology-first.pdf

Lopez. (2022). Factores de riesgo de enfermedades cardiovasculares en adultos jóvenes. Universidad Tecnica de Manabi, 1-17.

Lozano, R., Razo, C., & Montoya, A. (2023). La carga de la enfermedad, lesiones y factores de riesgo en México. Actualización del período 1990-2021. Scielo, 10.

Marrero, L. C.-S.-V. (2021). Uso de algoritmo K-means para clasificar perfiles de clientes con datos de medidores inteligentes de consumo eléctrico: Un caso de estudio. Revista chilena de ingeniería, 778-787.

Michael, J. (2019). Scaling and accelerating a digital. McKinsey's Sydney, 10.

Noble, J. (02 de 05 de 2024). IBM. Obtenido de IBM: https://www.ibm.com/mx-es/think/topics/arima-model

OPS. (16 de 05 de 2023). OPS. Obtenido de OPS: https://www.paho.org/es/noticias/16-5-2023-informe-ecuador-mejorando-salud-cardiovascular-desde-comunidades-locales-hasta

OPS. (15 de 02 de 2025). IRIS PAHO Home. Obtenido de IRIS PAHO Home: IRIS PAHO Home

Perez, M. A. (22 de 06 de 2022). Rpubs. Obtenido de Rpubs: https://rpubs.com/ca_ademir/series_temporales

Pypro. (01 de 01 de 2025). Pypro. Obtenido de Pypro: https://www.pypro.mx/app/curso/analisis-de-series-de-tiempo-con-python/modelos-arma-y-arima

Rodriguez, A. (22 de Enero de 2025). DIGITALWEEK.ES. Obtenido de DIGITALWEEK.ES: https://digitalweek.es/la-revolucion-del-big-data-impacto-y-tendencias-en-el-mundo-actual/

Rosa, J., & Frutos, E. (2022). Ciencia de datos en salud: desafíos y oportunidades en América Latina. Revista Médica Clínica Las Condes, 591-597.

Sierra, Y. (01 de 02 de 2021). Blog Lemontech. Obtenido de Blog Lemontech.

Tang, N. (2022). Data analytics. London: IntechOpen.

Torres, A. d. (01 de 07 de 2023). BUSINESS. Obtenido de BUSINESS: https://www.esic.edu/rethink/marketing-y-comunicacion/ventajas-desventajas-tecnologia-blockchain-c

ANEXOS

ANEXO 1. SERIES TEMPORALES

In [ ]:

```
import pandas as pd
# Leer el archivo CSV
egresos_final = pd.read_csv("D:/NO ELIMINAR/Desktop/PUCE/TESIS/egresos_final.csv")
# Mostrar las primeras filas para verificar que se haya leído correctamente
egresos_final
```

In [ ]:

```
import pandas as pd
# Supongamos que 'egresos_final' es tu DataFrame
# Agrupar por 'AFECCIÓN_PRINCIPAL' y contar las ocurrencias
conteo_afectaciones = egresos_final['des_3d'].value_counts().reset_index()

# Renombrar las columnas para mayor claridad
conteo_afectaciones.columns = ['des_3d', 'Total']

# Ordenar de mayor a menor
conteo_afectaciones = conteo_afectaciones.sort_values(by='Total', ascending=False)

# Aplicar formato de miles con punto
conteo_afectaciones['Total'] = conteo_afectaciones['Total'].apply(lambda x:
'{:,.0f}'.format(x).replace(',', '.'))

# Mostrar las 20 principales
Total_afectaciones = conteo_afectaciones.head(20)

# Mostrar el resultado
Total_afectaciones
```

In [ ]:

```
    egresos_final['dia'] =1
    egresos_final['fecha_def'] = egresos_final['anio_egr'].astype(str) + '-' +
egresos_final['mes_egr'].astype(str)+ '-' + egresos_final['dia'].astype(str)
```

egresos_final['fecha_def']

In [ ]:

egresos_final['fecha_def'].unique()

In [ ]:

egresos_final['fecha_def'] = pd.to_datetime(egresos_final['fecha_def'])

In [ ]:

defunciones_cardiacas = egresos_final[['fecha_def', 'des_3d']]

defunciones_cardiacas

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
# Agrupar por fecha y contar las ocurrencias
total_afectaciones = defunciones_cardiacas.groupby('fecha_def').size().reset_index(name='Defunciones por problemas cardiacos')

# Ordenar por cantidad de defunciones de forma descendente
total_afectaciones = total_afectaciones.sort_values('fecha_def', ascending=False)


# Asegurarse de que 'fecha_def' sea del tipo datetime
total_afectaciones['fecha_def'] = pd.to_datetime(total_afectaciones['fecha_def'])

# Agrupar por año y contar las ocurrencias de defunciones por problemas cardiacos
total_afectaciones['año'] = total_afectaciones['fecha_def'].dt.year
total_afectaciones_anual = total_afectaciones.groupby('año')['Defunciones por problemas cardiacos'].sum().reset_index()

# Crear el gráfico de línea
plt.figure(figsize=(10, 6))
plt.plot(total_afectaciones_anual['año'], total_afectaciones_anual['Defunciones por problemas cardiacos'], marker='o', color='b', label='Defunciones por problemas cardiacos')

# Personalizar el gráfico
```

```
plt.title('Evolución Anual de Defunciones por Enfermedades Cardiacas')
plt.xlabel('Año')
plt.ylabel('Número de Defunciones')
plt.grid(True)
plt.xticks(total_afectaciones_anual['año'], rotation=45)
plt.legend()

# Mostrar el gráfico
plt.tight_layout()
plt.show()
```

In [ ]:

```
import pandas as pd

# Supongamos que ya tienes el DataFrame `df` con los datos cargados.
# Si no, puedes cargar los datos desde un CSV o una base de datos.

# Agrupar por la columna 'fecha_def' y contar las ocurrencias
data1 = defunciones_cardiacas.groupby('fecha_def').size().reset_index(name='Defunciones_Cardiacas')

# Ordenar por la fecha
data1 = data1.sort_values(by='fecha_def')

# Mostrar el resultado
data1
```

Tratamiento de outliers

In [ ]:

```
import pandas as pd
# Calcula los cuartiles de la columna 'Defunciones_Cardiacas'
Q1 = data1['Defunciones_Cardiacas'].quantile(0.25)
Q3 = data1['Defunciones_Cardiacas'].quantile(0.75)

# Calcula el rango intercuartil (IQR)
```

```
IQR = Q3 - Q1

# Calcula los límites superior e inferior
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR

# Filtra las filas que no son outliers en 'Defunciones_Cardiacas'
data1 = data1[(data1['Defunciones_Cardiacas'] >= limite_inferior) &
              (data1['Defunciones_Cardiacas'] <= limite_superior)]

# Imprime el DataFrame resultante
print(data1)
In [ ]:
import seaborn as sns
In [ ]:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Verificar las columnas del DataFrame
print(data1.columns)



# Crear el boxplot
sns.boxplot(y=data1['Defunciones_Cardiacas'])
plt.show()
In [ ]:
print(data1.columns)
In [ ]:
data1
In [ ]:
```

```
import matplotlib.pyplot as plt

data1.date = pd.to_datetime(data1.fecha_def, dayfirst = True)

data1.set_index("fecha_def", inplace = True)

sns.boxplot(y=data1['Defunciones_Cardiacas'])

plt.show()
```

In [ ]:

```
data1.Defunciones_Cardiacas.plot(figsize=(20,5), title = "Evolución Anual de Defunciones por Enfermedades Cardiacas")

plt.show()
```

In [ ]:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.graphics.tsaplots as sgt

import statsmodels.tsa.stattools as sts

from statsmodels.tsa.seasonal import seasonal_decompose

import seaborn as sns

sns.set()
```

In [ ]:

```
pip install python-dateutil
```

In [ ]:

```
from dateutil.parser import parse
```

In [ ]:

```
sts.adfuller(data1['Defunciones_Cardiacas'])
```

Seasonality

In [ ]:

```
data1.index = pd.to_datetime(data1.index)  # Asegúrate de que el índice sea un DatetimeIndex

data1 = data1.asfreq('MS')  # Establece la frecuencia como mensual
```

In [ ]:

```
print(data1.index.freq)  # Esto muestra la frecuencia del índice
```

In [ ]:

```
print(data1.index.freq)  # Esto muestra la frecuencia del índice
```

In [ ]:

```
print(f"Datos restantes: {data1.shape[0]}")
```

In [ ]:

```
data1['Defunciones_Cardiacas']
```

In [ ]:

```
print(data1['Defunciones_Cardiacas'].isna().sum())  # Esto muestra cuántos valores faltantes hay
```

In [ ]:

```
# Rellenar valores faltantes con interpolación
data1['Defunciones_Cardiacas'] = data1['Defunciones_Cardiacas'].interpolate(method='linear')
```

In [ ]:

```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
plt.rc("figure", figsize=(12,8))
# Descomposición aditiva
s_dec_additive = seasonal_decompose(data1['Defunciones_Cardiacas'], model="additive")

# Graficar la descomposición
s_dec_additive.plot()
plt.show()
```

In [ ]:

```
plt.rc("figure", figsize=(12,8))
s_dec_multiplicative = seasonal_decompose(data1.Defunciones_Cardiacas, model = "multiplicative")
s_dec_multiplicative.plot()
plt.show()
```

In [ ]:

```
total_l= len(data1)
train_len=round(total_l*0.8)
train_len
train=data1[0:train_len]
test=data1[train_len : ]
```

In [ ]:

```
# arrays and dataframes
```

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

In [ ]:

```
fitted_model = ExponentialSmoothing(train['Defunciones_Cardiacas'],trend='mul',seasonal='mul',seasonal_periods=12).fit()
```

In [ ]:

```
test_predictions = fitted_model.forecast(36).rename('Test Forecast')
```

In [ ]:

```
test_predictions[:10]
```

In [ ]:

```
train['Defunciones_Cardiacas'].plot(legend=True,label='TRAIN')

test['Defunciones_Cardiacas'].plot(legend=True,label='TEST',figsize=(12,8))

plt.title('Train and Test Data');
```

Transformación box cox

In [ ]:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.stats import boxcox


# Suponiendo que 'train' es tu DataFrame y 'Defunciones_Cardiacas' es la columna de datos que quieres transformar


# Asegúrate de que los datos sean positivos, ya que Box-Cox no puede trabajar con datos negativos o cero

data = train['Defunciones_Cardiacas']


# Aplicar la transformación Box-Cox

transformed_data, lambda_value = boxcox(data)


# Imprimir el valor de lambda
```

```
print(f"Valor de lambda: {lambda_value}")

# Crear un gráfico para comparar los datos originales y los transformados
plt.figure(figsize=(10, 6))

# Gráfico de los datos originales
plt.subplot(1, 2, 1)
plt.plot(data, label='Datos Originales', color='blue')
plt.title('Datos Originales')
plt.xlabel('Tiempo')
plt.ylabel('Defunciones Cardiacas')

# Gráfico de los datos transformados
plt.subplot(1, 2, 2)
plt.plot(transformed_data, label='Datos Transformados (Box-Cox)', color='green')
plt.title('Datos Transformados con Box-Cox')
plt.xlabel('Tiempo')
plt.ylabel('Defunciones Cardiacas (Transformados)')

# Mostrar las leyendas y el gráfico
plt.tight_layout()
plt.show()
```

Box cox y diferenciación

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import boxcox

# Asegúrate de tener tus datos de entrenamiento
# Suponiendo que 'train' es tu DataFrame y 'Defunciones_Cardiacas' es la columna de datos que quieres transformar
```

```
# Paso 1: Aplicar la transformación Box-Cox
data = train['Defunciones_Cardiacas']

# Asegúrate de que los datos sean positivos para la transformación Box-Cox
data_positive = data + abs(data.min()) + 1  # Sumar el valor absoluto del mínimo + 1

# Aplicar la transformación Box-Cox
transformed_data, lambda_value = boxcox(data_positive)

# Paso 2: Aplicar la diferenciación para hacer los datos estacionarios
# Diferenciación de primer orden (puedes aumentar el número si es necesario)
transformed_data_diff = np.diff(transformed_data)

# Mostrar el valor de lambda de la transformación Box-Cox
print(f"Valor de lambda: {lambda_value}")

# Crear un gráfico para comparar los datos originales, transformados y diferenciados
plt.figure(figsize=(12, 8))

# Gráfico de los datos originales
plt.subplot(3, 1, 1)
plt.plot(data, label='Datos Originales', color='blue')
plt.title('Datos Originales')
plt.xlabel('Tiempo')
plt.ylabel('Defunciones Cardiacas')

# Gráfico de los datos transformados con Box-Cox
plt.subplot(3, 1, 2)
plt.plot(transformed_data, label='Datos Transformados (Box-Cox)', color='green')
plt.title('Datos Transformados con Box-Cox')
plt.xlabel('Tiempo')
plt.ylabel('Defunciones Cardiacas (Transformados)')
```

```
# Gráfico de los datos diferenciados
plt.subplot(3, 1, 3)
plt.plot(transformed_data_diff, label='Datos Diferenciados', color='red')
plt.title('Datos Diferenciados')
plt.xlabel('Tiempo')
plt.ylabel('Defunciones Cardiacas Diferenciadas')

# Ajuste de la disposición de los gráficos
plt.tight_layout()
plt.show()
```

In [ ]:

```
from statsmodels.tsa.stattools import adfuller

# Comprobar la estacionariedad de los datos diferenciados
result = adfuller(transformed_data_diff)
print(f'Estadístico ADF: {result[0]}')
print(f'p-valor: {result[1]}')
if result[1] <= 0.05:
    print("La serie diferenciada es estacionaria")
else:
    print("La serie diferenciada no es estacionaria")
```

Método Suavizado Exponencial de Holt con Tendencia

In [ ]:

```
import numpy as np
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Asegúrate de tener tus datos de entrenamiento y prueba
# Suponiendo que 'train' y 'test' ya están definidos previamente.

# Modelo Holt-Winters con tendencia aditiva y sin componente estacional
```

```
model = ExponentialSmoothing(
    np.asarray(train['Defunciones_Cardiacas']),
    seasonal_periods=12,  # Periodos estacionales (ajusta según tu caso)
    trend='additive',     # Tendencia aditiva
    seasonal=None         # Sin componente estacional
)

# Ajustar el modelo
model_fit = model.fit(optimized=True)

# Imprimir los parámetros ajustados del modelo
print(model_fit.params)

# Realizar el pronóstico para el conjunto de prueba
y_hat_holt = test.copy()  # Copiar los datos de test
y_hat_holt['holt_forecast'] = model_fit.forecast(len(test))  # Pronóstico para el periodo de test

# Mostrar las predicciones
print(y_hat_holt[['Defunciones_Cardiacas', 'holt_forecast']])
In [ ]:
import matplotlib.pyplot as plt

# Graficar las series reales y las predicciones
plt.figure(figsize=(10, 6))
plt.plot(train['Defunciones_Cardiacas'], label='Entrenamiento', color='blue')
plt.plot(test['Defunciones_Cardiacas'], label='Test', color='orange')
plt.plot(y_hat_holt['holt_forecast'], label='Pronóstico exponential Smoothing Holt-Winters',
color='green')
plt.legend(loc='best')
plt.title('Pronóstico de Defunciones Cardiacas con Holt-Winters')
plt.show()
In [ ]:
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
import numpy as np

# Obtener las predicciones del modelo Holt-Winters sin componente estacional
y_true_holt = test['Defunciones_Cardiacas']  # Valores reales de test
y_pred_holt = y_hat_holt['holt_forecast']  # Predicciones del modelo Holt-Winters

# 1. MSE (Mean Squared Error)
mse_holt = mean_squared_error(y_true_holt, y_pred_holt)
print(f'MSE (Holt-Winters sin estacionalidad): {mse_holt}')

# 2. RMSE (Root Mean Squared Error)
rmse_holt = np.sqrt(mse_holt)
print(f'RMSE (Holt-Winters sin estacionalidad): {rmse_holt}')

# 3. MAE (Mean Absolute Error)
mae_holt = mean_absolute_error(y_true_holt, y_pred_holt)
print(f'MAE (Holt-Winters sin estacionalidad): {mae_holt}')

# 4. MAPE (Mean Absolute Percentage Error)
mape_holt = np.mean(np.abs((y_true_holt - y_pred_holt) / y_true_holt)) * 100
print(f'MAPE (Holt-Winters sin estacionalidad): {mape_holt}%')
In [ ]:
```

Método Multiplicativo de Holt Winters con Tendencia y Estacionalidad

```
In [ ]:
import numpy as np
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import matplotlib.pyplot as plt

# Asegúrate de tener tus datos de entrenamiento y prueba
# Suponiendo que 'train' y 'test' ya están definidos previamente.
```

```
# Modelo Holt-Winters con tendencia y estacionalidad multiplicativa
model_multiplicative = ExponentialSmoothing(
    np.asarray(train['Defunciones_Cardiacas']),
    seasonal_periods=12,  # Ajusta según el número de periodos estacionales (12 para datos mensuales)
    trend='additive',      # Tendencia aditiva (puedes probar 'multiplicative' si deseas)
    seasonal='multiplicative'  # Componente estacional multiplicativo
)

# Ajustar el modelo
model_multiplicative_fit = model_multiplicative.fit(optimized=True)

# Imprimir los parámetros ajustados del modelo
print(model_multiplicative_fit.params)

# Realizar el pronóstico para el conjunto de prueba
y_hat_multiplicative = test.copy()  # Copiar los datos de test
y_hat_multiplicative['holt_forecast_multiplicative'] = model_multiplicative_fit.forecast(len(test))  # Pronóstico

# Mostrar las predicciones
print(y_hat_multiplicative[['Defunciones_Cardiacas', 'holt_forecast_multiplicative']])

# Graficar las series reales y las predicciones
plt.figure(figsize=(10, 6))
plt.plot(train['Defunciones_Cardiacas'], label='Entrenamiento', color='blue')
plt.plot(test['Defunciones_Cardiacas'], label='Test', color='orange')
plt.plot(y_hat_multiplicative['holt_forecast_multiplicative'], label='Pronóstico Holt-Winters Multiplicativo', color='green')
plt.legend(loc='best')
plt.title('Pronóstico de Defunciones Cardiacas con Holt-Winters Multiplicativo')
plt.show()
```

In [ ]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Obtener las predicciones del modelo Holt-Winters multiplicativo
y_true_hw = test['Defunciones_Cardiacas']  # Valores reales de test
y_pred_hw = y_hat_multiplicative['holt_forecast_multiplicative']  # Predicciones del modelo Holt-Winters

# 1. MSE (Mean Squared Error)
mse_hw = mean_squared_error(y_true_hw, y_pred_hw)
print(f'MSE (Holt-Winters Multiplicativo): {mse_hw}')

# 2. RMSE (Root Mean Squared Error)
rmse_hw = np.sqrt(mse_hw)
print(f'RMSE (Holt-Winters Multiplicativo): {rmse_hw}')

# 3. MAE (Mean Absolute Error)
mae_hw = mean_absolute_error(y_true_hw, y_pred_hw)
print(f'MAE (Holt-Winters Multiplicativo): {mae_hw}')

# 4. MAPE (Mean Absolute Percentage Error)
mape_hw = np.mean(np.abs((y_true_hw - y_pred_hw) / y_true_hw)) * 100
print(f'MAPE (Holt-Winters Multiplicativo): {mape_hw}%')
```

METODO ARIMA

In [ ]:

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Paso 1: Verificar la estacionariedad de la serie
def test_stationarity(series):
    result = adfuller(series)
```

```
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    if result[1] <= 0.05:
        print("La serie es estacionaria")
    else:
        print("La serie no es estacionaria")

test_stationarity(train['Defunciones_Cardiacas'])

# Si no es estacionaria, puedes diferenciarla
train_diff = train['Defunciones_Cardiacas'].diff().dropna()

# Paso 2: Graficar ACF y PACF para determinar p y q
plot_acf(train_diff)
plot_pacf(train_diff)
plt.show()
In [ ]:
from statsmodels.tsa.arima.model import ARIMA

# Definir p, d, q manualmente (por ejemplo, p=1, d=1, q=1)
p, d, q = 2, 1, 1

# Ajustar el modelo ARIMA
arima_model = ARIMA(train['Defunciones_Cardiacas'], order=(p, d, q))
arima_fit = arima_model.fit()

# Imprimir el resumen del modelo ajustado
print(arima_fit.summary())

# Pronóstico
y_hat_arima = test.copy()
y_hat_arima['arima_forecast'] = arima_fit.forecast(len(test))
```

```
# Mostrar las predicciones
print(y_hat_arima[['Defunciones_Cardiacas', 'arima_forecast']])

# Graficar los resultados
plt.figure(figsize=(10, 6))
plt.plot(train['Defunciones_Cardiacas'], label='Entrenamiento', color='blue')
plt.plot(test['Defunciones_Cardiacas'], label='Test', color='orange')
plt.plot(y_hat_arima['arima_forecast'], label='Pronóstico ARIMA', color='green')
plt.legend(loc='best')
plt.title('Pronóstico de Defunciones Cardiacas con ARIMA')
plt.show()
In [ ]:
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from pmdarima import auto_arima  # Para determinar automáticamente los parámetros p, d, q

# Asegúrate de tener tus datos de entrenamiento y prueba
# Suponiendo que 'train' y 'test' ya están definidos previamente.

# Paso 1: Verificar la estacionariedad de la serie (con prueba de Dickey-Fuller)
def test_stationarity(series):
    result = adfuller(series)
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    if result[1] <= 0.05:
        print("La serie es estacionaria")
    else:
        print("La serie no es estacionaria")
```

In [ ]:

```
# Verificar la estacionariedad de la serie de entrenamiento
test_stationarity(train['Defunciones_Cardiacas'])


# Si la serie no es estacionaria, podemos diferenciarla:
# Diferenciamos si es necesario (d > 0)
train_diff = train['Defunciones_Cardiacas'].diff().dropna()
# Suprimir FutureWarnings específicos de sklearn
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)


# Paso 2: Encontrar los parámetros p, d, q usando auto_arima
# Si no se ha determinado manualmente, podemos utilizar esta función
model_auto_arima = auto_arima(train['Defunciones_Cardiacas'], seasonal=True,
stepwise=False,
                              trace=True,start_p= 1, start_q= 1, max_p = 3,max_q=
3,suppress_warnings=True,
                              error_action='ignore')
```

In [ ]:

```
# Paso 3: Ajustar el modelo ARIMA (usando los parámetros optimizados por auto_arima)
p, d, q = model_auto_arima.order  # Obtener los parámetros p, d, q
arima_model = ARIMA(train['Defunciones_Cardiacas'], order=(2, 1, 1))
arima_fit = arima_model.fit()


# Ver los parámetros del modelo ajustado
print(arima_fit.summary())


# Paso 4: Realizar pronósticos para el conjunto de prueba
y_hat_arima = test.copy()
y_hat_arima['arima_forecast'] = arima_fit.forecast(len(test))
```

In [ ]:

```
# Mostrar las predicciones
print(y_hat_arima[['Defunciones_Cardiacas', 'arima_forecast']])
```

```
# Paso 5: Graficar los resultados
plt.figure(figsize=(10, 6))
plt.plot(train['Defunciones_Cardiacas'], label='Entrenamiento', color='blue')
plt.plot(test['Defunciones_Cardiacas'], label='Test', color='orange')
plt.plot(y_hat_arima['arima_forecast'], label='Pronóstico ARIMA', color='green')
plt.legend(loc='best')
plt.title('Pronóstico de Defunciones Cardiacas con ARIMA')
plt.show()
In [ ]:
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Obtener las predicciones del modelo ARIMA
y_true_arima = test['Defunciones_Cardiacas']  # Valores reales de test
y_pred_arima = y_hat_arima['arima_forecast']  # Predicciones del modelo ARIMA

# 1. MSE (Mean Squared Error)
mse_arima = mean_squared_error(y_true_arima, y_pred_arima)
print(f'MSE (ARIMA): {mse_arima}')

# 2. RMSE (Root Mean Squared Error)
rmse_arima = np.sqrt(mse_arima)
print(f'RMSE (ARIMA): {rmse_arima}')

# 3. MAE (Mean Absolute Error)
mae_arima = mean_absolute_error(y_true_arima, y_pred_arima)
print(f'MAE (ARIMA): {mae_arima}')

# 4. MAPE (Mean Absolute Percentage Error)
mape_arima = np.mean(np.abs((y_true_arima - y_pred_arima) / y_true_arima)) * 100
print(f'MAPE (ARIMA): {mape_arima}%')
```

In [ ]:

SARIMAX

In [ ]:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [ ]:

```
from pmdarima import auto_arima


# Suprimir FutureWarnings específicos de sklearn
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# Ajustar el modelo SARIMA con los parámetros predeterminados
#model_auto_sarima = auto_arima(train['Defunciones_Cardiacas'], seasonal=True, m=12,
#                               stepwise=False, trace=True, suppress_warnings=True,
#                               error_action='ignore')

#model_auto_sarima = auto_arima(train['Defunciones_Cardiacas'], seasonal=True, m=12,
stepwise=True, trace=True)
from pmdarima import auto_arima

# Ajustando un modelo SARIMA
model_auto_sarima = auto_arima(train['Defunciones_Cardiacas'], seasonal=True, m=12,
stepwise=True, trace=True,
                               start_p= 1,d=0, start_q= 0, max_p = 3, max_d = 2,max_q= 3,
                              start_P=0,D=0,start_Q=0,max_P =3,max_Q=3,max_D=2,
suppress_warnings=True,
                               error_action='ignore')
```

In [ ]:

```
import statsmodels.api as sm


# Definir el modelo SARIMA con parámetros (p, d, q) y estacionales (P, D, Q, m)
```

```
model_sarima = sm.tsa.SARIMAX(train['Defunciones_Cardiacas'],
                              order=(2, 0, 0),  # Reemplazar con los valores adecuados
                              seasonal_order=(2,0, 0,12),  # Reemplazar con los valores adecuados
                              enforce_stationarity=False,
                              enforce_invertibility=False)

sarimax_fit= model_sarima.fit()
# Ajustar el modelo
result_sarima = model_sarima.fit()
In [ ]:
# Realizar pronósticos para el conjunto de prueba
y_hat_sarimax = test.copy()
y_hat_sarimax['sarimax_forecast'] = sarimax_fit.forecast(len(test))

# Mostrar las predicciones
print(y_hat_sarimax[['Defunciones_Cardiacas', 'sarimax_forecast']])
In [ ]:
# Graficar los resultados
plt.figure(figsize=(10, 6))
plt.plot(train['Defunciones_Cardiacas'], label='Entrenamiento', color='blue')
plt.plot(test['Defunciones_Cardiacas'], label='Test', color='orange')
plt.plot(y_hat_sarimax['sarimax_forecast'], label='Pronóstico SARIMAX', color='green')
plt.legend(loc='best')
plt.title('Pronóstico de Defunciones Cardiacas con SARIMAX')
plt.show()
In [ ]:
# Obtener las predicciones de SARIMAX
y_true_sarimax = test['Defunciones_Cardiacas']  # Valores reales de test
y_pred_sarimax = y_hat_sarimax['sarimax_forecast']  # Predicciones del modelo SARIMAX

# 1. MSE (Mean Squared Error)
mse_sarimax = mean_squared_error(y_true_sarimax, y_pred_sarimax)
```

```
print(f'MSE (SARIMAX): {mse_sarimax}')

# 2. RMSE (Root Mean Squared Error)
rmse_sarimax = np.sqrt(mse_sarimax)
print(f'RMSE (SARIMAX): {rmse_sarimax}')

# 3. MAE (Mean Absolute Error)
mae_sarimax = mean_absolute_error(y_true_sarimax, y_pred_sarimax)
print(f'MAE (SARIMAX): {mae_sarimax}')

# 4. MAPE (Mean Absolute Percentage Error)
mape_sarimax = np.mean(np.abs((y_true_sarimax - y_pred_sarimax) / y_true_sarimax)) * 100
print(f'MAPE (SARIMAX): {mape_sarimax}%')
```

EVALUACIÓN DE LOS MODELOS

In [ ]:

```
import pandas as pd

# Diccionario con los resultados de las métricas de cada modelo
metrics_dict = {
    'Modelo': ['ARIMA', 'SARIMAX', 'Holt-Winters Multiplicativo', 'Holt-Winters sin Estacionalidad'],
    'MSE': [mse_arima, mse_sarimax, mse_hw, mse_holt],
    'RMSE': [rmse_arima, rmse_sarimax, rmse_hw, rmse_holt],
    'MAE': [mae_arima, mae_sarimax, mae_hw, mae_holt],
    'MAPE (%)': [mape_arima, mape_sarimax, mape_hw, mape_holt]
}

# Crear DataFrame de la matriz de comparación
comparison_matrix = pd.DataFrame(metrics_dict)

# Mostrar la matriz
comparison_matrix
```

PREDICCIÓN DE SARIMAX A FUTURO

In [ ]:

```
# Ajustar el modelo SARIMAX
# Supongamos que la estacionalidad es anual (12 meses), por lo que s=12
model = SARIMAX(data1['Defunciones_Cardiacas'] ,
                order=(2, 1, 1),  # ARIMA(p,d,q)
                seasonal_order=(2, 1, 1, 12),  # SARIMAX(P,D,Q,s)
                enforce_stationarity=False,
                enforce_invertibility=False)

# Ajustar el modelo
result = model.fit()

# Resumen del modelo
print(result.summary())
```

In [ ]:

```
# Realizar las predicciones a futuro (por ejemplo, los próximos 36 meses)
forecast_steps = 36  # Número de pasos a predecir
forecast = result.get_forecast(steps=forecast_steps)

# Obtener el intervalo de confianza
forecast_ci = forecast.conf_int()

# Crear un rango de fechas para las predicciones
last_train_date = data1.index[-1]  # Última fecha de entrenamiento
forecast_index = pd.date_range(start=last_train_date + pd.Timedelta(days=1),
periods=forecast_steps, freq='M')
```

In [ ]:

```
# Mostrar las predicciones
predicted_values = forecast.predicted_mean

# Crear DataFrame con las predicciones y las fechas
predicted_df = pd.DataFrame({
    'Fecha': forecast_index,
```

```
    'Defunciones Predichas': predicted_values
})

print(predicted_df)
```

In [ ]:

```
# Graficar los resultados
plt.figure(figsize=(10, 6))
plt.plot(data1.index, data1['Defunciones_Cardiacas'], label='Datos Históricos', color='blue')
plt.plot(forecast_index, predicted_values, color='red', label='Predicción SARIMAX')

# Rellenar el área entre los intervalos de confianza
plt.fill_between(forecast_index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink', alpha=0.3)

plt.title('Predicción de Defunciones Cardíacas con SARIMAX')
plt.xlabel('Fecha')
plt.ylabel('Defunciones Cardíacas')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Ver los valores predichos (predicted_mean)
predicted_values = forecast.predicted_mean

# Mostrar los valores predichos junto con las fechas
predicted_dates = forecast_index
predicted_df = pd.DataFrame({
    'Fecha': predicted_dates,
    'Defunciones Predichas': predicted_values
})
```

```
# Mostrar el DataFrame con las predicciones
print(predicted_df)
```

Predicción a futuro Multiplicativo

In [ ]:

```
import numpy as np
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import matplotlib.pyplot as plt

# Asegúrate de tener tus datos de entrenamiento y prueba
# Suponiendo que 'train' y 'test' ya están definidos previamente.

# Modelo Holt-Winters con tendencia y estacionalidad multiplicativa
model_multiplicative = ExponentialSmoothing(
    np.asarray(data1['Defunciones_Cardiacas']),  # Datos de entrenamiento
    seasonal_periods=12,  # Ajusta según el número de periodos estacionales (12 para datos mensuales)
    trend='additive',      # Tendencia aditiva (puedes probar 'multiplicative' si deseas)
    seasonal='multiplicative'  # Componente estacional multiplicativo
)

# Ajustar el modelo
model_multiplicative_fit = model_multiplicative.fit(optimized=True)

# Imprimir los parámetros ajustados del modelo
print("Parámetros del modelo Holt-Winters Multiplicativo:")
print(model_multiplicative_fit.params)
```

In [ ]:

```
# Realizar predicciones a futuro (ejemplo: pronóstico para los próximos 36 meses)
forecast_steps = 36  # Número de pasos a predecir
future_forecast = model_multiplicative_fit.forecast(forecast_steps)

# Crear rango de fechas para las predicciones futuras
```

```
last_data1_date = data1.index[-1]  # Última fecha de entrenamiento

In [ ]:

# Verificar si el último mes de datos es diciembre

if last_data1_date.month == 12:

    # Si es diciembre, establece la predicción a partir de enero del siguiente año

    start_date = pd.Timestamp(year=last_data1_date.year + 1, month=1, day=1)

else:

    # Si no es diciembre, simplemente pasa al siguiente mes

    start_date = last_data1_date + pd.Timedelta(days=1)


# Crear el rango de fechas de las predicciones futuras

forecast_index = pd.date_range(start=start_date, periods=forecast_steps, freq='M')  # Fechas futuras


# Mostrar las predicciones a futuro

predicted_df = pd.DataFrame({

    'Fecha': forecast_index,

    'Defunciones Predichas': future_forecast

})


print(predicted_df)

In [ ]:

# Calcular el error cuadrático medio (RMSE) de las predicciones anteriores (esto es solo una aproximación)

# Suponiendo que 'y_hat_multiplicative' contiene las predicciones sobre el conjunto de prueba.

rmse = np.sqrt(((y_hat_multiplicative['holt_forecast_multiplicative'] - test['Defunciones_Cardiacas']) ** 2).mean())


# Establecer el intervalo de confianza (aproximado) usando el RMSE

# Aquí asumimos que el intervalo de confianza será de +/- 1.96 veces el RMSE (para un 95% de intervalo de confianza)

upper_bound = future_forecast + 1.96 * rmse

lower_bound = future_forecast - 1.96 * rmse
```

```
# Graficar los resultados
plt.figure(figsize=(10, 6))

# Graficar los datos históricos de entrenamiento
plt.plot(data1.index, data1['Defunciones_Cardiacas'], label='Datos Históricos', color='blue')

# Graficar las predicciones generadas por el modelo Holt-Winters
plt.plot(forecast_index, future_forecast, color='red', label='Predicción Holt-Winters
Multiplicativo')

# Rellenar el área entre los límites superior e inferior (simulación de intervalo de confianza)
plt.fill_between(forecast_index, lower_bound, upper_bound, color='pink', alpha=0.3,
label='Intervalo de Confianza (95%)')

# Título y etiquetas
plt.title('Predicción de Defunciones Cardíacas con Holt-Winters Multiplicativo')
plt.xlabel('Fecha')
plt.ylabel('Defunciones Cardíacas')

# Agregar leyenda, cuadrícula y ajustes
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()

# Mostrar el gráfico
plt.show()
In [ ]:
# Definir el número de divisiones para la validación cruzada
n_splits = 5
tscv = TimeSeriesSplit(n_splits=n_splits)

# Listas para almacenar los errores y predicciones
```

```
errors = []
predictions = []

# Iterar sobre cada división
for train_index, test_index in tscv.split(data1):
    train, test = data1.iloc[train_index], data1.iloc[test_index]

    # Crear el modelo Holt-Winters con tendencia aditiva y estacionalidad multiplicativa
    model = ExponentialSmoothing(
        np.asarray(train['Defunciones_Cardiacas']),
        seasonal_periods=12,  # Ajusta según el número de periodos estacionales (12 para datos mensuales)
        trend='additive',
        seasonal='multiplicative'
    )

    # Ajustar el modelo
    model_fit = model.fit(optimized=True)

    # Realizar predicciones sobre el conjunto de prueba
    y_hat = model_fit.forecast(len(test))

    # Almacenar las predicciones y calcular el error
    predictions.extend(y_hat)
    rmse = np.sqrt(mean_squared_error(test['Defunciones_Cardiacas'], y_hat))
    errors.append(rmse)

# Mostrar el error promedio y la desviación estándar
mean_error = np.mean(errors)
std_error = np.std(errors)
print(f'Error promedio (RMSE): {mean_error:.2f}')
print(f'Desviación estándar del error: {std_error:.2f}')
```

ANEXO 2. RESULTADO SERIE TEMPORAL

```
print(predicted_df)
```

	Fecha	Defunciones Predichas
0	2024-01-31	130.605066
1	2024-02-29	117.969220
2	2024-03-31	137.060308
3	2024-04-30	119.416429
4	2024-05-31	128.505840
5	2024-06-30	129.306480
6	2024-07-31	132.322359
7	2024-08-31	130.470623
8	2024-09-30	125.505778
9	2024-10-31	126.821257
10	2024-11-30	127.074769
11	2024-12-31	120.792277
12	2025-01-31	133.125696
13	2025-02-28	120.242327
14	2025-03-31	139.697041
15	2025-04-30	121.710056
16	2025-05-31	130.970103
17	2025-06-30	131.782140
18	2025-07-31	134.851725
19	2025-08-31	132.960626
20	2025-09-30	127.897224
21	2025-10-31	129.233938
22	2025-11-30	129.488447
23	2025-12-31	123.082998
24	2026-01-31	135.646326
25	2026-02-28	122.515434
26	2026-03-31	142.333773
27	2026-04-30	124.003683
28	2026-05-31	133.434365
29	2026-06-30	134.257800
30	2026-07-31	137.381090
31	2026-08-31	135.450628
32	2026-09-30	130.288670
33	2026-10-31	131.646619
34	2026-11-30	131.902124
35	2026-12-31	125.373719

ANEXO 3. CLUSTERIZACIÓN DE DATOS

In [ ]:

```
import pandas as pd


# Leer el archivo CSV
egresos_final = pd.read_csv("D:/NO ELIMINAR/Desktop/PUCE/TESIS/egresos_final.csv")


# Mostrar las primeras filas para verificar que se haya leído correctamente
egresos_final
```

In [ ]:

```
egresos_final['prov_ubi'] = egresos_final['prov_ubi'].astype(str)
```

In [ ]:

```
egresos_final['prov_ubi'].unique()
```

In [ ]:

```
# Aplicar la recodificación inversa con lambda y apply
egresos_final['prov_ubi'] = egresos_final['prov_ubi'].apply(
   lambda x: 'Azuay' if x == '1' else
            'Bolívar' if x == '2' else
            'Cañar' if x == '3' else
            'Carchi' if x == '4' else
            'Cotopaxi' if x == '5' else
            'Chimborazo' if x == '6' else
            'El Oro' if x == '7' else
            'Esmeraldas' if x == '8' else
            'Guayas' if x == '9' else
            'Imbabura' if x == '10' else
            'Loja' if x == '11' else
            'Los Ríos' if x == '12' else
            'Manabí' if x == '13' else
            'Morona Santiago' if x == '14' else
            'Napo' if x == '15' else
            'Pastaza' if x == '16' else
```

```
        'Pichincha' if x == '17' else
        'Tungurahua' if x == '18' else
        'Zamora Chinchipe' if x == '19' else
        'Galápagos' if x == '20' else
        'Sucumbíos' if x == '21' else
        'Orellana' if x == '22' else
        'Santo Domingo de los Tsáchilas' if x == '23' else
        'Santa Elena' if x == '24' else x
)
In [ ]:
# Mostrar el DataFrame actualizado
egresos_final['prov_ubi'].unique()
In [ ]:
egresos_final
In [ ]:
egresos_final['area_ubi'] = egresos_final['area_ubi'].astype(str)
In [ ]:
# Diccionario de mapeo
# Verificar los valores originales
print(egresos_final['area_ubi'].unique())

# Asegúrate de que los valores en la columna sean enteros
egresos_final['area_ubi'] = egresos_final['area_ubi'].astype(int)

# Diccionario de mapeo
mapa_area_ubi = {
    1: 'Urbana',
    2: 'Rural'
}

# Función lambda para aplicar el mapeo
egresos_final['area_ubi'] = egresos_final['area_ubi'].apply(
```

```
    lambda x: mapa_area_ubi.get(x, x)  # Si x no está en el diccionario, mantener el valor original
)

# Verificar los valores recodificados
print(egresos_final['area_ubi'].unique())
In [ ]:
egresos_final['area_ubi'].unique()
In [ ]:
# Diccionario original
mapa_clase = {
    'Hospital básico': 1,
    'Hospital general': 2,
    'Infectología': 3,
    'Gineco-Obstétrico': 4,
    'Pediátrico': 5,
    'Psiquiátrico y Sanatorio de Alcohólicos': 6,
    'Oncológico': 8,
    'Hospital Neumológico (Antituberculoso)':9,
    'Geriátrico': 10,
    'Hospital de especialidades': 11,
    'Clínica general (sin especialidad)': 12,
    'Gineco-Obstétrica': 13,
    'Traumatología': 15,
    'Psiquiátrica': 16,
    'Otras clínicas especializadas': 17,
    'Establecimientos del día que prestan internacion hospitalaria': 17,
    'Otros establecimientos sin internación (Cruz Roja, Planif. F)':32
}

# Invertir el diccionario
mapa_clase_invertido = {v: k for k, v in mapa_clase.items()}
```

```
# Recodificación inversa: Función lambda para convertir números en nombres
egresos_final['clase'] = egresos_final['clase'].apply(
    lambda x: mapa_clase_invertido.get(x, x)  # Si no está en el diccionario, mantiene el valor original
)

# Verificar los valores recodificados
print(egresos_final['clase'].unique())
In [ ]:
egresos_final['tipo'].astype
In [ ]:
print(egresos_final['tipo'].unique())
In [ ]:
egresos_final['tipo'] = egresos_final['tipo'].astype(int)
In [ ]:
# Diccionario que mapea los códigos numéricos a sus respectivos nombres
mapa_tipo = {
    1: 'Agudo',
    2: 'Crónico',
    3: 'Clínicas Generales sin especialidad',
    4: 'Establecimientos SIN internación',
    5: 'Sin Tipo (Hospitales Básicos)'
}

# Recodificar la columna 'tipo' usando el diccionario
egresos_final['tipo'] = egresos_final['tipo'].map(mapa_tipo).fillna('Desconocido')

# Verificar los resultados
print(egresos_final['tipo'].unique())
In [ ]:
egresos_final.dtypes
In [ ]:
```

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np

# Calcular el total por etnia recodificada
totales_por_provincia = egresos_final.groupby('prov_ubi')['causa3'].count()

# Ordenar los totales de mayor a menor
totales_por_provincia = totales_por_provincia.sort_values(ascending=True)

# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(totales_por_provincia))(np.linspace(0, 1,
len(totales_por_provincia)))

# Crear el gráfico de barras horizontales con colores personalizados
plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico
plt.barh(totales_por_provincia.index, totales_por_provincia.values, color=colores)

# Agregar título y etiquetas
plt.title('Total de defunciones Cardiaca por Provincia', fontsize=16)
plt.xlabel('Total de defunciones Cardiaca', fontsize=12)
plt.ylabel('Provincia', fontsize=12)

# Función para formatear los números con puntos como separadores de miles
formatter = ticker.FuncFormatter(lambda x, pos: f'{int(x):,}'.replace(',', '.') if x.is_integer() else
f'{x:,.2f}'.replace(',', '.'))

# Aplicar el formateador a las etiquetas de los valores en el eje X
plt.gca().xaxis.set_major_formatter(formatter)

# Mostrar los totales en las barras con puntos como separadores de miles
for i, v in enumerate(totales_por_provincia.values):
    plt.text(v + 0.1, i, f'{int(v):,}'.replace(',', '.'), va='center', fontsize=10)
```

```
# Mejorar la legibilidad de las etiquetas del eje Y si es necesario
plt.tight_layout()

# Mostrar el gráfico
plt.show()
In [ ]:
# Calcular el total por provincia
totales_por_provincia = egresos_final.groupby('prov_ubi')['causa3'].count()

# Ordenar los totales de mayor a menor
totales_por_provincia = totales_por_provincia.sort_values(ascending=True)

# Calcular el total global
total_global = totales_por_provincia.sum()

# Convertir a porcentajes
porcentajes_por_provincia = (totales_por_provincia / total_global) * 100

# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(totales_por_provincia))(np.linspace(0, 1,
len(totales_por_provincia)))

# Crear el gráfico de barras horizontales con colores personalizados
plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico
plt.barh(porcentajes_por_provincia.index, porcentajes_por_provincia.values, color=colores)

# Agregar título y etiquetas
plt.title('Defunciones Cardiaca por Provincia', fontsize=16)
plt.xlabel('Defunciones Cardiaca', fontsize=12)
plt.ylabel('Provincia', fontsize=12)

# Función para formatear los números con puntos como separadores de miles
```

```
formatter = ticker.FuncFormatter(lambda x, pos: f'{x:.2f}%')

# Aplicar el formateador a las etiquetas de los valores en el eje X
plt.gca().xaxis.set_major_formatter(formatter)

# Mostrar los porcentajes en las barras
for i, v in enumerate(porcentajes_por_provincia.values):
    plt.text(v + 0.5, i, f'{v:.2f}%', va='center', fontsize=10)

# Mejorar la legibilidad de las etiquetas del eje Y si es necesario
plt.tight_layout()

# Mostrar el gráfico
plt.show()
In [ ]:
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker

# Calcular el total por etnia recodificada
totales_por_des_cie3d = egresos_final.groupby('des_3d')['causa3'].count()

# Ordenar los totales de mayor a menor
totales_por_des_cie3d = totales_por_des_cie3d.sort_values(ascending=True)

# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(totales_por_des_cie3d))(np.linspace(0, 1,
len(totales_por_des_cie3d)))

# Crear el gráfico de barras horizontales con colores personalizados
plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico
plt.barh(totales_por_des_cie3d.index, totales_por_des_cie3d.values, color=colores)
```

```
# Agregar título y etiquetas
plt.title('Total de defunciones Cardiaca por Subtitulos CIE10', fontsize=16)
plt.xlabel('Total de defunciones Cardiaca', fontsize=12)
plt.ylabel('Provincia', fontsize=12)

# Función para formatear los números con puntos como separadores de miles
formatter = ticker.FuncFormatter(lambda x, pos: f'{int(x):,}'.replace(',', '.') if x.is_integer() else f'{x:,.2f}'.replace(',', '.'))

# Aplicar el formateador a las etiquetas de los valores en el eje X
plt.gca().xaxis.set_major_formatter(formatter)

# Mostrar los totales en las barras con puntos como separadores de miles
for i, v in enumerate(totales_por_des_cie3d.values):
    plt.text(v + 0.1, i, f'{int(v):,}'.replace(',', '.'), va='center', fontsize=10)

# Mejorar la legibilidad de las etiquetas del eje Y si es necesario
plt.tight_layout()

# Mostrar el gráfico
plt.show()
In [ ]:
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker

# Calcular el total por etnia recodificada
totales_por_des_cie3d = egresos_final.groupby('des_3d')['causa3'].count()

# Ordenar los totales de mayor a menor
totales_por_des_cie3d = totales_por_des_cie3d.sort_values(ascending=False)

# Seleccionar solo las 10 principales causas
```

```
totales_por_des_cie3d_top_10 = totales_por_des_cie3d.head(10)


# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)

colores = plt.cm.get_cmap('tab20', len(totales_por_des_cie3d_top_10))(np.linspace(0, 1, len(totales_por_des_cie3d_top_10)))


totales_por_des_cie3d_top_10 = totales_por_des_cie3d_top_10.sort_values(ascending=True)


# Crear el gráfico de barras horizontales con colores personalizados

plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico

plt.barh(totales_por_des_cie3d_top_10.index, totales_por_des_cie3d_top_10.values, color=colores)


# Agregar título y etiquetas

plt.title('Defunciones Cardiacas por Subtítulos CIE10', fontsize=16)

plt.xlabel('Total de defunciones Cardiacas', fontsize=12)

plt.ylabel('Subtítulo CIE10', fontsize=12)


# Función para formatear los números con puntos como separadores de miles

formatter = ticker.FuncFormatter(lambda x, pos: f'{int(x):,}'.replace(',', '.') if x.is_integer() else f'{x:,.2f}'.replace(',', '.'))


# Aplicar el formateador a las etiquetas de los valores en el eje X

plt.gca().xaxis.set_major_formatter(formatter)


# Mostrar los totales en las barras con puntos como separadores de miles

for i, v in enumerate(totales_por_des_cie3d_top_10.values):

    plt.text(v + 0.1, i, f'{int(v):,}'.replace(',', '.'), va='center', fontsize=10)


# Mejorar la legibilidad de las etiquetas del eje Y si es necesario

plt.tight_layout()


# Mostrar el gráfico
```

```
plt.show()
```

In [ ]:

```
# Calcular el total por subtítulo CIE10
totales_por_des_cie3d = egresos_final.groupby('des_3d')['causa3'].count()

# Ordenar los totales de mayor a menor
totales_por_des_cie3d = totales_por_des_cie3d.sort_values(ascending=False)

# Seleccionar solo las 10 principales causas
totales_por_des_cie3d_top_10 = totales_por_des_cie3d.head(10)

# Calcular el total global
total_global_cie3d = totales_por_des_cie3d_top_10.sum()

# Convertir a porcentajes
porcentajes_por_des_cie3d_top_10 = (totales_por_des_cie3d_top_10 / total_global_cie3d) * 100

# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(totales_por_des_cie3d_top_10))(np.linspace(0, 1, len(totales_por_des_cie3d_top_10)))

# Ordenar los totales de menor a mayor
porcentajes_por_des_cie3d_top_10 = porcentajes_por_des_cie3d_top_10.sort_values(ascending=True)

# Crear el gráfico de barras horizontales con colores personalizados
plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico
plt.barh(porcentajes_por_des_cie3d_top_10.index, porcentajes_por_des_cie3d_top_10.values, color=colores)

# Agregar título y etiquetas
plt.title('Defunciones Cardiacas por Subtítulos CIE10 (porcentaje)', fontsize=16)
plt.xlabel('Porcentaje de defunciones Cardiacas', fontsize=12)
```

```
plt.ylabel('Subtítulo CIE10', fontsize=12)

# Función para formatear los números como porcentaje
formatter = ticker.FuncFormatter(lambda x, pos: f'{x:.2f}%')

# Aplicar el formateador a las etiquetas de los valores en el eje X
plt.gca().xaxis.set_major_formatter(formatter)

# Mostrar los porcentajes en las barras
for i, v in enumerate(porcentajes_por_des_cie3d_top_10.values):
    plt.text(v + 0.5, i, f'{v:.2f}%', va='center', fontsize=10)

# Mejorar la legibilidad de las etiquetas del eje Y si es necesario
plt.tight_layout()

# Mostrar el gráfico
plt.show()
In [ ]:
import matplotlib.pyplot as plt

# Definir los límites de los bins en intervalos de 10 años
bins = range(0, int(egresos_final['edad_c'].max()) + 10, 10)

# Crear el histograma con los bins definidos
plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico
plt.hist(egresos_final['edad_c'], bins=bins, color='skyblue', edgecolor='black')

# Agregar título y etiquetas
plt.title('Histograma de la Edad (Rangos de 10 años)', fontsize=16)
plt.xlabel('Edad', fontsize=12)
plt.ylabel('Frecuencia', fontsize=12)
```

```
# Mostrar el gráfico
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Verificar si hay valores nulos en el DataFrame
valores_nulos = egresos_final.isnull().sum()
# Mostrar las columnas con valores nulos
print(valores_nulos[valores_nulos > 0])
```

In [ ]:

```
# Verificar los tipos de datos de las columnas
tipos_de_datos = egresos_final.dtypes
print(tipos_de_datos)
```

In [ ]:

```
# Verificar si hay filas duplicadas
duplicados = egresos_final.duplicated().sum()
# Mostrar la cantidad de duplicados
print(f'Número de filas duplicadas: {duplicados}')
```

In [ ]:

```
# Detectar valores atípicos en una columna específica (por ejemplo, 'edad_c')
Q1 = egresos_final['edad_c'].quantile(0.25)
Q3 = egresos_final['edad_c'].quantile(0.75)
IQR = Q3 - Q1
# Definir el rango para los valores normales
rango_inferior = Q1 - 1.5 * IQR
rango_superior = Q3 + 1.5 * IQR
# Filtrar valores atípicos
outliers = egresos_final[(egresos_final['edad_c'] < rango_inferior) | (egresos_final['edad_c'] >
rango_superior)]
# Mostrar valores atípicos
print(outliers)
```

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
import matplotlib.ticker as ticker

# Calcular el total por 'clase'

totales_por_clase = egresos_final.groupby('clase')['causa3'].count()

# Ordenar los totales de mayor a menor

totales_por_clase = totales_por_clase.sort_values(ascending=True)

# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)

colores = plt.cm.get_cmap('tab20', len(totales_por_clase))(np.linspace(0, 1,
len(totales_por_clase)))

# Crear el gráfico de barras horizontales con colores personalizados

plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico

plt.barh(totales_por_clase.index, totales_por_clase.values, color=colores)

# Agregar título y etiquetas

plt.title('Total de Defunciones Cardiaca por Clase', fontsize=16)

plt.xlabel('Total de Defunciones Cardiaca', fontsize=12)

plt.ylabel('Clase', fontsize=12)

# Función para formatear los números con puntos como separadores de miles

formatter = ticker.FuncFormatter(lambda x, pos: f'{int(x):,}'.replace(',', '.') if x.is_integer() else
f'{x:,.2f}'.replace(',', '.'))

# Aplicar el formateador a las etiquetas de los valores en el eje X

plt.gca().xaxis.set_major_formatter(formatter)

# Mostrar los totales en las barras con puntos como separadores de miles

for i, v in enumerate(totales_por_clase.values):

    plt.text(v + 0.1, i, f'{int(v):,}'.replace(',', '.'), va='center', fontsize=10)

# Mejorar la legibilidad de las etiquetas del eje Y si es necesario

plt.tight_layout()

# Mostrar el gráfico

plt.show()

In [ ]:

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.ticker as ticker

# Calcular el total por 'tipo'

totales_por_tipo = egresos_final.groupby('tipo')['causa3'].count()
```

```
# Ordenar los totales de mayor a menor
totales_por_tipo = totales_por_tipo.sort_values(ascending=True)
# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(totales_por_tipo))(np.linspace(0, 1, len(totales_por_tipo)))
# Crear el gráfico de barras horizontales con colores personalizados
plt.figure(figsize=(10, 6))  # Ajustar el tamaño del gráfico
plt.barh(totales_por_tipo.index, totales_por_tipo.values, color=colores)
# Agregar título y etiquetas
plt.title('Total de Defunciones Cardiaca por Tipo', fontsize=16)
plt.xlabel('Total de Defunciones Cardiaca', fontsize=12)
plt.ylabel('Tipo', fontsize=12)

# Función para formatear los números con puntos como separadores de miles
formatter = ticker.FuncFormatter(lambda x, pos: f'{int(x):,}'.replace(',', '.') if x.is_integer() else f'{x:,.2f}'.replace(',', '.'))
# Aplicar el formateador a las etiquetas de los valores en el eje X
plt.gca().xaxis.set_major_formatter(formatter)
# Mostrar los totales en las barras con puntos como separadores de miles
for i, v in enumerate(totales_por_tipo.values):
    plt.text(v + 0.1, i, f'{int(v):,}'.replace(',', '.'), va='center', fontsize=10)
# Mejorar la legibilidad de las etiquetas del eje Y si es necesario
plt.tight_layout()
# Mostrar el gráfico
plt.show()
In [ ]:
# Calcular el total por género recodificado
totales_por_area = egresos_final.groupby('area_ubi')['causa3'].count()
# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(totales_por_area))(np.linspace(0, 1, len(totales_por_area)))
# Crear el gráfico de barras con colores personalizados
plt.bar(totales_por_area.index, totales_por_area.values, color=colores)
```

```
# Agregar título y etiquetas
plt.title('Total de defunciones Cardiaca por Area')
plt.xlabel('Género')
plt.ylabel('Total de defunciones Cardiaca')
# Función para formatear los números con puntos como separadores de miles
formatter = ticker.FuncFormatter(lambda x, pos: f'{int(x):,}'.replace(',', '.') if x.is_integer() else f'{x:,.2f}'.replace(',', '.'))
# Aplicar el formateador a las etiquetas de los valores en el eje Y
plt.gca().yaxis.set_major_formatter(formatter)
# Mostrar los totales en las barras con puntos como separadores de miles
for i, v in enumerate(totales_por_area.values):
    plt.text(i, v + 0.1, f'{int(v):,}'.replace(',', '.'), ha='center')
# Mejorar la legibilidad de las etiquetas del eje X si es necesario
plt.xticks(rotation=45, ha='right')
# Mostrar el gráfico
plt.tight_layout()
plt.show()
In [ ]:
# Calcular el total por área
totales_por_area = egresos_final.groupby('area_ubi')['causa3'].count()
# Calcular el total global
total_global_area = totales_por_area.sum()
# Convertir a porcentajes
porcentajes_por_area = (totales_por_area / total_global_area) * 100
# Crear una lista de colores diferentes (puedes personalizarla o usar un colormap)
colores = plt.cm.get_cmap('tab20', len(porcentajes_por_area))(np.linspace(0, 1, len(porcentajes_por_area)))
# Crear el gráfico de barras con colores personalizados
plt.bar(porcentajes_por_area.index, porcentajes_por_area.values, color=colores)
# Agregar título y etiquetas
plt.title('Defunciones Cardiaca por Área', fontsize=16)
plt.xlabel('Área', fontsize=12)
plt.ylabel('Defunciones Cardiaca', fontsize=12)
```

```
# Función para formatear los números como porcentaje
formatter = ticker.FuncFormatter(lambda x, pos: f'{x:.2f}%')
# Aplicar el formateador a las etiquetas de los valores en el eje Y
plt.gca().yaxis.set_major_formatter(formatter)
# Mostrar los porcentajes en las barras
for i, v in enumerate(porcentajes_por_area.values):
    plt.text(i, v + 0.5, f'{v:.2f}%', ha='center', fontsize=10)
# Mejorar la legibilidad de las etiquetas del eje X si es necesario
plt.xticks(rotation=45, ha='right')
# Mejorar la presentación
plt.tight_layout()
# Mostrar el gráfico
plt.show()
```

Clustering

In [ ]:

```
Colums_select=
['area_ubi','prov_ubi','sexo_recodificado','etnia_recodificada','fecha_def','edad_c','entidad_recodificado','tipo','Codc10']
data_clustering= egresos_final[Colums_select]
data_clustering.head()
```

In [ ]:

```
egresos_final['prov_ubi'].unique()
```

In [ ]:

```
import pandas as pd
# Definir los rangos de edad
bins = [0, 12, 19, 39, 59, float('inf')]  # Definir los límites de los rangos
labels = ['Niños', 'Adolescentes', 'Adultos jóvenes',
          'Adultos de mediana edad', 'Adultos de mayor edad']
# Crear una nueva columna 'rango_edad' en data_clustering asignando los rangos de edad
data_clustering['rango_edad'] = pd.cut(data_clustering['edad_c'], bins=bins, labels=labels, right=True)
# Mostrar el resultado
print(data_clustering[['edad_c', 'rango_edad']].head())
```

In [ ]:

```
data_clustering.dropna()
```

```
data_clustering.info()
```

In [ ]:

```
data_clustering = data_clustering.drop('edad_c', axis=1)
```

In [ ]:

```
data_clustering['area_ubi']=data_clustering['area_ubi'].astype('category')
```

```
data_clustering['prov_ubi']=data_clustering['prov_ubi'].astype('category')
```

```
data_clustering['sexo_recodificado']=data_clustering['sexo_recodificado'].astype('category')
```

```
data_clustering['etnia_recodificada']=data_clustering['etnia_recodificada'].astype('category')
```

```
data_clustering['fecha_def']=data_clustering['fecha_def'].astype('category')
```

```
data_clustering['entidad_recodificado']=data_clustering['entidad_recodificado'].astype('category
')
```

```
data_clustering['tipo']=data_clustering['tipo'].astype('category')
```

```
data_clustering['Codc10']=data_clustering['Codc10'].astype('category')
```

In [ ]:

```
import pandas as pd
```

```
from kmodes.kmodes import KModes
```

In [ ]:

```
costos = []


# Probar diferentes valores de k (número de clusters)
for k in range(1, 11):  # Probar de 1 a 10 clusters
    k_modes = KModes(n_clusters=k, init='Cao', n_init=5, verbose=1)
    k_modes.fit(data_clustering)  # Ajustar el modelo
    costos.append(k_modes.cost_)
```

In [ ]:

```
# Graficar el SSE en función del número de clusters
plt.plot(range(1, 11), costos, marker='o')
plt.title('Método del Codo (K-modes)')
plt.xlabel('Número de Clusters')
plt.ylabel('Cost (SSE)')
plt.xticks(range(1, 11))
```

```
plt.show()
```

In [ ]:

```
!pip install gower
!pip install scikit-learn
import gower
from sklearn.metrics import silhouette_score
```

Cluster 3

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=3, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['sexo_recodificado'] = label_encoder.fit_transform(datc['sexo_recodificado'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['fecha_def'] = label_encoder.fit_transform(datc['fecha_def'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
```

```
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Supongamos que la tabla de contingencia ya está lista
contingency_table = pd.crosstab(datc['sexo_recodificado'], datc['cluster_labels'])
# Convertir la tabla de contingencia en un formato largo para graficar
contingency_table_long = contingency_table.reset_index().melt(id_vars='sexo_recodificado',
                                                               var_name='Cluster',
                                                               value_name='Frecuencia')
# Crear el gráfico de barras
plt.figure(figsize=(10, 6))
# Utilizar un palet de colores personalizado para asegurar que se asignen colores distintos
sns.set_palette("Set1")  # Cambia a otro palet si prefieres otro
# Graficar la distribución de frecuencias con un gráfico de barras
sns.barplot(x='Cluster', y='Frecuencia', hue='sexo_recodificado', data=contingency_table_long)
# Personalizar título y etiquetas
plt.title('Distribución de Clusters por Género del Fallecido')
plt.xlabel('Cluster')
plt.ylabel('Frecuencia')
# Asegurarse de que la leyenda tenga etiquetas claras y colores asociados
plt.legend(title='Género del Fallecido', labels=['Masculino', 'Femenino'])
# Mostrar el gráfico
```

```
plt.show()
```

Cluster 4

```
In [ ]:
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=4, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['sexo_recodificado'] = label_encoder.fit_transform(datc['sexo_recodificado'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['fecha_def'] = label_encoder.fit_transform(datc['fecha_def'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
```

```
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Cluster 5

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=5, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['sexo_recodificado'] = label_encoder.fit_transform(datc['sexo_recodificado'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['fecha_def'] = label_encoder.fit_transform(datc['fecha_def'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
```

```
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Cluster 6

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=6, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder

# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['sexo_recodificado'] = label_encoder.fit_transform(datc['sexo_recodificado'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
```

```
datc['fecha_def'] = label_encoder.fit_transform(datc['fecha_def'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Cluster 7

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=7, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
```

```
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['sexo_recodificado'] = label_encoder.fit_transform(datc['sexo_recodificado'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['fecha_def'] = label_encoder.fit_transform(datc['fecha_def'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

In [ ]:

POR ESTAR DIVIDIENDO LOS DATOS SE ELIGEN MENOS VARIABLES

In [ ]:

```
Colums_select=
['area_ubi','prov_ubi','etnia_recodificada','edad_c','entidad_recodificado','tipo','Codc10']
data_clustering= egresos_final[Colums_select]
data_clustering.head()
```

In [ ]:

```
import pandas as pd
# Definir los rangos de edad
bins = [0, 12, 19, 39, 59, float('inf')]  # Definir los límites de los rangos
labels = ['Niños', 'Adolescentes', 'Adultos jóvenes',
        'Adultos de mediana edad', 'Adultos de mayor edad']
# Crear una nueva columna 'rango_edad' en data_clustering asignando los rangos de edad
data_clustering['rango_edad'] = pd.cut(data_clustering['edad_c'], bins=bins, labels=labels,
right=True)
# Mostrar el resultado
print(data_clustering[['edad_c', 'rango_edad']].head())
```

In [ ]:

```
data_clustering = data_clustering.drop('edad_c', axis=1)
```

In [ ]:

```
data_clustering['area_ubi']=data_clustering['area_ubi'].astype('category')
data_clustering['prov_ubi']=data_clustering['prov_ubi'].astype('category')
data_clustering['etnia_recodificada']=data_clustering['etnia_recodificada'].astype('category')
data_clustering['entidad_recodificado']=data_clustering['entidad_recodificado'].astype('category')
data_clustering['tipo']=data_clustering['tipo'].astype('category')
data_clustering['Codc10']=data_clustering['Codc10'].astype('category')
```

In [ ]:

```
costos = []
# Probar diferentes valores de k (número de clusters)
for k in range(1, 11):  # Probar de 1 a 10 clusters
    k_modes = KModes(n_clusters=k, init='Cao', n_init=5, verbose=1)
    k_modes.fit(data_clustering)  # Ajustar el modelo
    costos.append(k_modes.cost_)
```

In [ ]:

```
# Graficar el SSE en función del número de clusters
plt.plot(range(1, 11), costos, marker='o')
plt.title('Método del Codo (K-modes)')
plt.xlabel('Número de Clusters')
plt.ylabel('Cost (SSE)')
plt.xticks(range(1, 11))
plt.show()
```

realizamos el proceso de nuevo

Cluster 3

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=3, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
```

```
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

MCA

In [ ]:

```
import pandas as pd
# Aplicar pd.get_dummies para las columnas que quieres convertir a variables dummies
datc = pd.get_dummies(datc, columns=['area_ubi', 'prov_ubi', 'etnia_recodificada',
                                     'entidad_recodificado', 'Codc10', 'tipo', 'rango_edad'],
                      drop_first=False)
```

```
# Verifica el resultado
datc.head()
In [ ]:
column_names = list(datc.columns)
column_names
In [ ]:
!pip install prince
import prince
In [ ]:
mca= prince.MCA(n_components=2)
mca=mca.fit(datc)
mca_coordinates=mca.transform(datc)
In [ ]:
# Supongamos que 'cluster_labels' es la columna que contiene las etiquetas de los clusters
# Añadir las coordenadas al DataFrame para facilitar la visualización
datc['MCA_1'] = mca_coordinates[0]
datc['MCA_2'] = mca_coordinates[1]
# Visualización en un gráfico de dispersión
plt.figure(figsize=(10, 6))
# Utilizar seaborn para crear un gráfico de dispersión con clusters
sns.scatterplot(x='MCA_1', y='MCA_2', hue='cluster_labels', data=datc, palette='Set1', s=100)
# Añadir título y etiquetas
plt.title('Visualización de los Clusters en el Espacio 2D (MCA)')
plt.xlabel('Componente MCA 1')
plt.ylabel('Componente MCA 2')
plt.legend(title='Clusters')
# Mostrar el gráfico
plt.show()
In [ ]:
import pandas as pd
# Agrupar por cluster_labels y calcular la suma de las columnas binarias
cluster_summary = datc.groupby('cluster_labels').agg('sum').reset_index()
```

```
# Mostrar las primeras filas del resumen
cluster_summary
```

In [ ]:

```
# Crear un resumen de texto
for cluster in cluster_summary['cluster_labels']:
    print(f"Análisis para el Cluster {cluster}:")
    cluster_data = cluster_summary[cluster_summary['cluster_labels'] == cluster].drop('cluster_labels', axis=1)
        # Mostrar las variables más frecuentes en el cluster
    for column in cluster_data.columns:
        freq = cluster_data[column].values[0]
        print(f"{column}: {freq}")
    print("\n" + "-"*50 + "\n")
```

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Asegúrate de que las coordenadas del MCA ya estén calculadas y agregadas al DataFrame
datc['MCA_1'] = mca_coordinates[0]
datc['MCA_2'] = mca_coordinates[1]
# Crear un gráfico de dispersión de los clusters
plt.figure(figsize=(10, 6))
# Utilizar seaborn para crear un gráfico de dispersión con clusters
sns.scatterplot(x='MCA_1', y='MCA_2', hue='cluster_labels', data=datc, palette='Set1', s=100)
# Añadir título y etiquetas
plt.title('Visualización de los Clusters en el Espacio 2D (MCA)')
plt.xlabel('Componente MCA 1')
plt.ylabel('Componente MCA 2')
plt.legend(title='Clusters')
# Mostrar el gráfico
plt.show()
```

Cluster 4

In [ ]:

```
# Paso 2: Aplicar K-modes
```

```
kmodes = KModes(n_clusters=4, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])

# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))

# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Cluster 5

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=5, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
```

```
print("Silhouette Score:", silhouette)
```

Cluster 6

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=6, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder

# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
```

```
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Cluster 7

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=7, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
from sklearn.preprocessing import LabelEncoder
# Supongamos que 'column_name' es tu columna categórica, como 'cluster_labels'
label_encoder = LabelEncoder()
# Convertir las columnas categóricas a números
datc['area_ubi'] = label_encoder.fit_transform(datc['area_ubi'])
datc['prov_ubi'] = label_encoder.fit_transform(datc['prov_ubi'])
datc['etnia_recodificada'] = label_encoder.fit_transform(datc['etnia_recodificada'])
datc['entidad_recodificado'] = label_encoder.fit_transform(datc['entidad_recodificado'])
datc['Codc10'] = label_encoder.fit_transform(datc['Codc10'])
datc['tipo'] = label_encoder.fit_transform(datc['tipo'])
datc['rango_edad'] = label_encoder.fit_transform(datc['rango_edad'])
# Crear una matriz de distancias entre todos los puntos (sin la columna 'cluster_labels')
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
```

```
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

Tercer cluster

In [ ]:

```
# Paso 2: Aplicar K-modes
kmodes = KModes(n_clusters=3, init='Huang', n_init=10, random_state=42)
clusters = kmodes.fit_predict(data_clustering)
# Añadir los resultados de los clusters al DataFrame original
data_clustering['cluster'] = clusters
```

In [ ]:

```
# Paso 3: Calcular el Silhouette Score utilizando la distancia de Hamming
# El Silhouette Score para K-modes se puede calcular utilizando la distancia de Hamming
# Calcular la matriz de distancia de Hamming entre las observaciones
def hamming_distance(a, b):
    return np.sum(a != b) / len(a)
```

In [ ]:

```
datc=data_clustering.copy()
datc.insert(0,"cluster_labels",clusters, True)
```

In [ ]:

```
datc=datc.applymap(str)
datc=datc.sample(n=9000,random_state=1)
```

In [ ]:

```
import gower
from sklearn.metrics import silhouette_score
# Convertir las columnas categóricas a tipo 'str' para evitar problemas con CategoricalDtype
datc = datc.apply(lambda x: x.astype(str) if x.dtype.name == 'category' else x)
# Crear la matriz de distancias utilizando gower
distances = gower.gower_matrix(datc.drop("cluster_labels", axis=1))
# Calcular el Silhouette Score utilizando la matriz de distancias y las etiquetas de los clusters
silhouette = silhouette_score(distances, datc['cluster_labels'], metric='precomputed')
```

```
# Imprimir el Silhouette Score
print("Silhouette Score:", silhouette)
```

In [ ]:

```
import pandas as pd
# Aplicar pd.get_dummies para las columnas seleccionadas
datc = pd.get_dummies(datc, columns=['area_ubi', 'prov_ubi', 'etnia_recodificada',
                                     'entidad_recodificado', 'Codc10', 'tipo', 'rango_edad'],
                      drop_first=False)
# Verifica el resultado para asegurar que los encabezados sean los correctos
datc
```

In [ ]:

```
# Supongamos que 'cluster_labels' es la columna que contiene las etiquetas de los clusters
# Añadir las coordenadas al DataFrame para facilitar la visualización
datc['MCA_1'] = mca_coordinates[0]
datc['MCA_2'] = mca_coordinates[1]
# Visualización en un gráfico de dispersión
plt.figure(figsize=(10, 6))
# Utilizar seaborn para crear un gráfico de dispersión con clusters
sns.scatterplot(x='MCA_1', y='MCA_2', hue='cluster_labels', data=datc, palette='Set1', s=100)
# Añadir título y etiquetas
plt.title('Visualización de los Clusters en el Espacio 2D (MCA)')
plt.xlabel('Componente MCA 1')
plt.ylabel('Componente MCA 2')
plt.legend(title='Clusters')
# Mostrar el gráfico
plt.show()
```

In [ ]:

```
import pandas as pd
# Agrupar por cluster_labels y calcular la suma de las columnas binarias
cluster_summary = datc.groupby('cluster_labels').agg('sum').reset_index()
# Mostrar las primeras filas del resumen
cluster_summary
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Asegúrate de que 'mca_coordinates' sea un array de numpy (2, N)
# Si 'mca_coordinates' es un DataFrame, conviértelo a un array de numpy:
mca_coordinates = np.array(mca_coordinates)
# Crear una figura
plt.figure(figsize=(10, 8))
# Graficar las observaciones en el plano cartesiano
sns.scatterplot(x='MCA_1', y='MCA_2', hue='cluster_labels', data=datc, palette='Set1', s=100)
# Graficar las flechas para cada variable (en base a las coordenadas MCA)
# Solo graficamos las primeras dos componentes, ya que solo tenemos 2 en 'mca_coordinates'
for i, column in enumerate(datc.columns[:-2]):  # Excluyendo las columnas 'MCA_1' y 'MCA_2'
    if i < mca_coordinates.shape[1]:  # Asegurarse de que no estamos excediendo el número de componentes
        plt.quiver(0, 0, mca_coordinates[0, i], mca_coordinates[1, i], angles='xy', scale_units='xy', scale=1, color='black')
        plt.text(mca_coordinates[0, i] * 1.1, mca_coordinates[1, i] * 1.1, column, color='black', fontsize=12)
# Añadir título y etiquetas
plt.title('Gráfico SPAD: Observaciones y Variables en el Espacio 2D (MCA)', fontsize=14)
plt.xlabel('Componente MCA 1')
plt.ylabel('Componente MCA 2')

# Mostrar la leyenda para los clusters
plt.legend(title='Clusters')
# Mostrar el gráfico
plt.grid(True)
plt.show()
```

In [ ]:

```
import pandas as pd
# Agrupar por 'cluster_labels' y sumar las columnas categóricas
```

```
cluster_summary = datc.groupby('cluster_labels').agg('sum').reset_index()

# Listamos las columnas categóricas para analizar

categorical_columns = [col for col in cluster_summary.columns if 'area_ubi' in col or 'prov_ubi' in col or 'tipo' in col or 'rango_edad' in col]

# Para almacenar los resúmenes de cada clúster

cluster_variable_summary = {}

# Iteramos sobre los clústeres

for cluster in cluster_summary['cluster_labels']:
    cluster_variable_summary[cluster] = []
        # Filtrar las filas correspondientes a un clúster específico
    cluster_data = cluster_summary[cluster_summary['cluster_labels'] == cluster]
        # Recorremos las columnas categóricas para cada clúster
    for column in categorical_columns:
        # Obtenemos las frecuencias de cada categoría (True/False o 1/0)
        category_counts = cluster_data[column].values[0]
            # Obtenemos la categoría más frecuente
        most_frequent_category = 'True' if category_counts > 0 else 'False'
            # Calculamos el valor de prueba (por ejemplo, la frecuencia de la categoría)
        test_value = category_counts
            # Generamos el histograma de barras
        hist_bar = '*' * (category_counts // 10)  # Ajustamos la escala del histograma
            # Guardamos los resultados
        cluster_variable_summary[cluster].append({
            'Variable label': column,
            'Caracteristic category': most_frequent_category,
            'Test-value': test_value,
            'Histogram': hist_bar
        })
        # Convertimos la lista a un DataFrame y ordenamos de mayor a menor según el test-value
    cluster_variable_summary[cluster] = pd.DataFrame(cluster_variable_summary[cluster])
    cluster_variable_summary[cluster] =
cluster_variable_summary[cluster].sort_values(by='Test-value', ascending=False)

# Mostrar el resumen de las variables para cada clúster ordenado
```

```
for cluster in cluster_variable_summary:
    print(f"Group: CLUSTER {cluster} (Count: {len(cluster_summary[cluster_summary['cluster_labels'] == cluster])} - Percentage: {100 * len(cluster_summary[cluster_summary['cluster_labels'] == cluster]) / len(datc):.2f})")
    print(cluster_variable_summary[cluster])
    print("\n")
```

In [ ]:

```
import pandas as pd

# Agrupar por 'cluster_labels' y sumar las columnas categóricas
cluster_summary = datc.groupby('cluster_labels').agg('sum').reset_index()


# Listamos las columnas categóricas para analizar
categorical_columns = [col for col in cluster_summary.columns if 'area_ubi' in col or 'prov_ubi' in col or 'tipo' in col or 'rango_edad' in col]
# Para almacenar los resúmenes de cada clúster
cluster_variable_summary = {}
# Iteramos sobre los clústeres
for cluster in cluster_summary['cluster_labels'].unique():  # Cambié para que itere sobre todos los clústeres únicos
    cluster_variable_summary[cluster] = []
    # Filtrar las filas correspondientes a un clúster específico
    cluster_data = cluster_summary[cluster_summary['cluster_labels'] == cluster]
    # Recorremos las columnas categóricas para cada clúster
    for column in categorical_columns:
        # Obtenemos las frecuencias de cada categoría (True/False o 1/0)
        category_counts = cluster_data[column].values[0]
            # Obtenemos la categoría más frecuente
        most_frequent_category = 'True' if category_counts > 0 else 'False'
            # Calculamos el valor de prueba (por ejemplo, la frecuencia de la categoría)
        test_value = category_counts
            # Generamos el histograma de barras
        hist_bar = '*' * (category_counts // 10)  # Ajustamos la escala del histograma
            # Guardamos los resultados
        cluster_variable_summary[cluster].append({
```

```
            'Variable label': column,
            'Caracteristic category': most_frequent_category,
            'Test-value': test_value,
            'Histogram': hist_bar
        })
        # Convertimos la lista a un DataFrame y ordenamos de mayor a menor según el test-value
    cluster_variable_summary[cluster] = pd.DataFrame(cluster_variable_summary[cluster])
    cluster_variable_summary[cluster] = cluster_variable_summary[cluster].sort_values(by='Test-value', ascending=False)
# Mostrar el resumen de las variables para cada clúster en formato tabular, con un formato plano
for cluster in cluster_variable_summary:
    # Imprimir el encabezado con el número de clúster y su porcentaje
    cluster_count = len(cluster_summary[cluster_summary['cluster_labels'] == cluster])
    total_count = len(datc)
    percentage = (cluster_count / total_count) * 100
    print(f"Group: CLUSTER {cluster} (Count: {cluster_count} - Percentage: {percentage:.2f})")
        # Limitar la longitud de la columna 'Histogram' para evitar que se desborde
    cluster_variable_summary[cluster]['Histogram'] = cluster_variable_summary[cluster]['Histogram'].apply(lambda x: x[:50])  # Limitar a 50 caracteres
        # Convertimos la columna 'Histogram' a formato de cadena para asegurarnos de que se alinee a la izquierda
    cluster_variable_summary[cluster]['Histogram'] = cluster_variable_summary[cluster]['Histogram'].astype(str)
        # Mostramos la tabla con las columnas alineadas correctamente
    print(cluster_variable_summary[cluster].to_string(index=True, header=True, justify='left'))
    print("\n")
In [ ]:
pip install kmodes scikit-learn
In [ ]:
 In [ ]:
from kmodes.kmodes import KModes
from sklearn.model_selection import KFold
```

```
from sklearn.metrics import adjusted_rand_score
import numpy as np
In [ ]:
# Dividir en K particiones (en este caso 5 pliegues)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Inicializar lista para almacenar las métricas de validación
ari_scores = []
# Realizar validación cruzada
for train_index, test_index in kf.split(data_clustering):
    # Dividir los datos en train y test
    X_train, X_test = data_clustering.iloc[train_index], data_clustering.iloc[test_index]
    In [ ]:
from sklearn.metrics import adjusted_rand_score
from kmodes.kmodes import KModes
from sklearn.model_selection import KFold
import numpy as np
# Número de clusters y la inicialización de K-modes
n_clusters = 3
kmodes = KModes(n_clusters=n_clusters, init='Huang', n_init=10, random_state=42)
# Preparar los datos de entrada (asegúrate de que data_clustering esté limpio y listo)
# data_clustering = ... # Tus datos
# labels_true = ... # Las etiquetas reales de los clústeres, si las tienes

# Asegúrate de definir las etiquetas reales
# Por ejemplo, si tus etiquetas verdaderas están en una columna llamada "cluster_labels" de tu dataframe:
labels_true = data_clustering['cluster_labels']  # Esto es solo un ejemplo
# Dividir en K particiones (en este caso 5 pliegues)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Inicializar lista para almacenar las métricas de validación
ari_scores = []
# Realizar validación cruzada
for train_index, test_index in kf.split(data_clustering):
```

```
    # Dividir los datos en train y test
    X_train, X_test = data_clustering.iloc[train_index], data_clustering.iloc[test_index]
    labels_true_test = labels_true.iloc[test_index]  # Las etiquetas reales para los datos de prueba
        # Entrenar el modelo K-modes
    kmodes.fit(X_train)
        # Realizar predicciones en el conjunto de test
    clusters_test = kmodes.predict(X_test)
        # Calcular el Adjusted Rand Index (ARI) usando las etiquetas reales
    ari_score = adjusted_rand_score(labels_true_test, clusters_test)
    # Almacenar la métrica
    ari_scores.append(ari_score)
# Promediar las puntuaciones de ARI de cada pliegue
mean_ari_score = np.mean(ari_scores)
print(f"Adjusted Rand Index Promedio: {mean_ari_score:.4f}")
```

ANEXO 4. RESULTADO CLUSTER

Group: CLUSTER 0 (Count: 1 - Percentage: 0.01)

	Variable label	Caracteristic category	Test-value	Histogram
1	area_ubi_Urbana	True	2100	**********
26	tipo_Agudo	True	1685	**********
33	rango_edad_Adultos de mediana edad	True	1286	**********
11	prov_ubi_Guayas	True	802	**********
32	rango_edad_Adultos de mayor edad	True	518	**********
30	tipo_Sin Tipo (Hospitales Básicos)	True	379	********
20	prov_ubi_Pichincha	True	279	*******
15	prov_ubi_Manabí	True	222	******
34	rango_edad_Adultos jóvenes	True	216	******
14	prov_ubi_Los Ríos	True	188	*****
35	rango_edad_Niños	True	103	**
2	prov_ubi_Azuay	True	102	**
8	prov_ubi_El Oro	True	91	**
24	prov_ubi_Tungurahua	True	83	**
13	prov_ubi_Loja	True	72	**
22	prov_ubi_Santo Domingo de los Tsáchilas	True	54	*
9	prov_ubi_Esmeraldas	True	51	*
0	area_ubi_Rural	True	50	*
28	tipo_Crónico	True	45	*
27	tipo_Clínicas Generales sin especialidad	True	41	*
21	prov_ubi_Santa Elena	True	40	*
6	prov_ubi_Chimborazo	True	31	*
23	prov_ubi_Sucumbíos	True	31	*
31	rango_edad_Adolescentes	True	27	*
12	prov_ubi_Imbabura	True	21	*
5	prov_ubi_Cañar	True	21	*
7	prov_ubi_Cotopaxi	True	16	*
16	prov_ubi_Morona Santiago	True	13	*
4	prov_ubi_Carchi	True	9	
17	prov_ubi_Napo	True	6	
19	prov_ubi_Pastaza	True	6	
25	prov_ubi_Zamora Chinchipe	True	5	
10	prov_ubi_Galápagos	True	3	
3	prov_ubi_Bolívar	True	2	
18	prov_ubi_Orellana	True	2	
29	tipo_Establecimientos SIN internación	False	0	

Group: CLUSTER 1 (Count: 1 - Percentage: 0.01)

	Variable label	Caracteristic category	Test-value	Histogram
32	rango_edad_Adultos de mayor edad	True	4246	**********
1	area_ubi_Urbana	True	4230	**********
26	tipo_Agudo	True	3724	**********
11	prov_ubi_Guayas	True	1435	**********
20	prov_ubi_Pichincha	True	655	**********
30	tipo_Sin Tipo (Hospitales Básicos)	True	526	**********
15	prov_ubi_Manabí	True	331	********
2	prov_ubi_Azuay	True	314	********
24	prov_ubi_Tungurahua	True	276	*******
8	prov_ubi_El Oro	True	185	*****
6	prov_ubi_Chimborazo	True	159	****
13	prov_ubi_Loja	True	154	****
0	area_ubi_Rural	True	118	***
22	prov_ubi_Santo Domingo de los Tsáchilas	True	113	***
14	prov_ubi_Los Ríos	True	103	**
28	tipo_Crónico	True	95	**
5	prov_ubi_Cañar	True	92	**
9	prov_ubi_Esmeraldas	True	84	**
12	prov_ubi_Imbabura	True	82	**
7	prov_ubi_Cotopaxi	True	80	**
21	prov_ubi_Santa Elena	True	65	*
34	rango_edad_Adultos jóvenes	True	63	*
23	prov_ubi_Sucumbíos	True	43	*
3	prov_ubi_Bolívar	True	39	*
4	prov_ubi_Carchi	True	35	*
19	prov_ubi_Pastaza	True	27	*
35	rango_edad_Niños	True	24	*
25	prov_ubi_Zamora Chinchipe	True	20	*
17	prov_ubi_Napo	True	19	
16	prov_ubi_Morona Santiago	True	19	
31	rango_edad_Adolescentes	True	12	
10	prov_ubi_Galápagos	True	9	
18	prov_ubi_Orellana	True	9	
33	rango_edad_Adultos de mediana edad	True	3	
27	tipo_Clínicas Generales sin especialidad	True	2	
29	tipo_Establecimientos SIN internación	True	1	

Group: CLUSTER 2 (Count: 1 - Percentage: 0.01)

	Variable label	Caracteristic category	Test-value	Histogram
1	area_ubi_Urbana	True	2455	**********
32	rango_edad_Adultos de mayor edad	True	2296	**********
26	tipo_Agudo	True	1435	**********
11	prov_ubi_Guayas	True	1046	**********
27	tipo_Clínicas Generales sin especialidad	True	890	**********
20	prov_ubi_Pichincha	True	510	**********
15	prov_ubi_Manabí	True	291	**********
8	prov_ubi_El Oro	True	151	**********
30	tipo_Sin Tipo (Hospitales Básicos)	True	137	**********
2	prov_ubi_Azuay	True	136	**********
34	rango_edad_Adultos jóvenes	True	88	********
14	prov_ubi_Los Ríos	True	84	********
33	rango_edad_Adultos de mediana edad	True	81	********
22	prov_ubi_Santo Domingo de los Tsáchilas	True	81	********
13	prov_ubi_Loja	True	56	******
0	area_ubi_Rural	True	47	*****
21	prov_ubi_Santa Elena	True	41	*****
28	tipo_Crónico	True	40	*****
6	prov_ubi_Chimborazo	True	30	****
35	rango_edad_Niños	True	28	***
24	prov_ubi_Tungurahua	True	25	**
9	prov_ubi_Esmeraldas	True	13	**
5	prov_ubi_Cañar	True	12	*
12	prov_ubi_Imbabura	True	11	*
7	prov_ubi_Cotopaxi	True	9	
31	rango_edad_Adolescentes	True	9	
23	prov_ubi_Sucumbíos	True	3	
19	prov_ubi_Pastaza	True	2	
3	prov_ubi_Bolívar	True	1	
17	prov_ubi_Napo	False	0	
16	prov_ubi_Morona Santiago	False	0	
25	prov_ubi_Zamora Chinchipe	False	0	
10	prov_ubi_Galápagos	False	0	
29	tipo_Establecimientos SIN internación	False	0	
4	prov_ubi_Carchi	False	0	
18	prov_ubi_Orellana	False	0