

Tabla de contenido

0. INTRODUCCIÓN	2
Contexto socioeconómico del Sistema de Salud en EEUU.....	2
1. DEFINICIÓN DEL PROBLEMA Y OBJETIVOS	3
2. EXPLORACIÓN INICIAL DEL DATASET.....	5
3. LIMPIEZA Y PROCESAMIENTO DE DATOS.....	12
4. ANÁLISIS EXPLORATORIO DE DATOS (EDA).....	22
Estadísticas descriptivas	22
Distribución de variables numéricas	23
Distribución de variables categóricas	24
5. IDENTIFICACIÓN DE PATRONES Y ANÁLISIS DE VARIABLES.....	27
Análisis de variables Categóricas vs Género (Chi2).....	31
Análisis de variables Cuantitativas vs Género (t-test).....	32
Análisis de variables Categóricas entre ellas (Chi2)	33
6. IDENTIFICACIÓN DE INSIGHTS	40
7. PREDICCIÓN DE MODELOS	52
TÉCNICAS DE APRENDIZAJE SUPERVISADO	53
Modelado Predictivo de una Variable Numérica	54
REGRESIÓN LINEAL	54
ALGORITMO RANDOMFOREST	55
NEURAL NETWORK (RED NEURONAL).....	58
GRADIENT BOSTING (XGBOOST)	59
TÉCNICA DE MACHINE LEARNING ENSEMBLE.....	60
TÉCNICAS DE APRENDIZAJE NO SUPERVISADO Supervisado	64
Modelado predictivo de una variable categórica.....	64
CLUSTERING (K-MODES).....	64
PCA APLICADO A LOS CLUSTERS GENERADOS POR K-MODES	78
7. DISCUSIÓN	84
8. CONCLUSIONES.....	86
9. RECOMENDACIONES	88

0. INTRODUCCIÓN

En el dinámico campo de la atención médica, los datos juegan un papel esencial en la promoción de innovaciones y en la mejora de los resultados para los pacientes. Las grandes cantidades de datos que se generan a diario ofrecen información invaluable para las agencias aseguradoras en aquellos sistemas sanitarios que basan su sistema en medios privados.

Contexto socioeconómico del Sistema de Salud en EEUU

El sistema de salud de Estados Unidos se caracteriza por su naturaleza fragmentada, no universal y altamente dependiente del sector privado. Su desarrollo histórico ha estado marcado por transiciones entre modelos puramente privados y la implementación parcial de programas públicos. Inicialmente, el acceso a la atención médica era un privilegio restringido a quienes podían costearlo directamente. Durante la Gran Depresión, surgieron los primeros seguros hospitalarios como **Blue Cross**, y en la década de 1960 se implementaron **Medicare** (para adultos mayores) y **Medicaid** (para personas de bajos ingresos), dando lugar a un sistema mixto.

Aunque hubo intentos de establecer cobertura universal, como la fallida reforma de salud promovida por la administración Clinton en 1993, fue recientemente en 2010 con la aprobación de la Ley del Cuidado de Salud a Bajo Precio (**Affordable Care Act – ACA**, Obamacare) que se amplió la cobertura mediante subsidios y la expansión de Medicaid. Esta ley no ha sido derogada en la actualidad, pero si se enfrenta a serios intentos de legislación legislativa.

Hoy, el sistema combina seguros privados, generalmente obtenidos por empleo, y programas públicos como Medicare y Medicaid. Sin embargo, la cobertura no es universal, existen marcadas desigualdades en el acceso, especialmente para personas con bajos ingresos o sin seguro, y los costos se mantienen entre los más altos del mundo. A pesar de los avances, el sistema sigue sin garantizar una atención equitativa y accesible para toda la población.

1. DEFINICIÓN DEL PROBLEMA Y OBJETIVOS

- **Propósito:** ¿Por qué se analiza este dataset?

El dataset se analizará para predecir los costos médicos o monto de la facturación (Billing Amount) en función de una serie de variables predictoras o dependientes. Se analizan, por ejemplo, variables como la edad, condición médica o el tiempo de hospitalización y el hospital para predecir el coste de ingreso de cada paciente.

- **Preguntas clave o “insights”:** ¿Qué se busca entender o descubrir?

1. Estadística descriptiva

- ¿Cuál es la distribución de condiciones médicas entre pacientes?
- ¿Cuál es la edad promedio de los pacientes por condición médica?
- ¿Hay diferencias entre géneros (o tipo de admisión hospitalaria) en la frecuencia de enfermedades?
- ¿Cuál es la frecuencia de facturación por género, aseguradora, tipo de admisión o medicación?
- ¿Cuáles son los hospitales (o los médicos) con más pacientes o con mayores montos de facturación?
- ¿Cuál es la reincidencia de pacientes por aseguradora?

2. Análisis temporal

- ¿Cuál es la duración promedio de una estancia hospitalaria por género o por aseguradora?
- ¿Cuál es el número de hospitalizaciones en función del número de ingresos y de su edad (por ejemplo en pacientes mayores de 65 años)?
- ¿Cuál es la reincidencia de hospitalizaciones por aseguradora?

3. Predicción y Machine Learning

- ¿Podemos estimar el monto de facturación médica de un paciente antes de su admisión?
- ¿Podemos predecir el monto de facturación de un paciente basándonos en su edad, género, aseguradora o tipo de admisión?

- ¿Podemos predecir los días de hospitalización de un paciente basándonos en su edad, género, aseguradora o tipo de admisión?
- ¿Podemos predecir la condición médica de un paciente basándonos en edad, género, tipo de sangre u otros factores?
- ¿Qué factores influyen más en el resultado de los exámenes médicos (Test Results)?

4. Financiero

- ¿Qué condiciones médicas están asociadas con mayores costos?
- ¿Cuál es la variabilidad en el costo entre aseguradoras?
- ¿Qué médicos están asociados con las facturaciones más altas?
- ¿Cuál es la variabilidad del costo en función de los días de estancia?

5. Operaciones del hospital

- ¿Qué tipo de admisión (urgente, emergencia, etc.) es más común?
- ¿Hay relación entre el número de habitación y la duración o el costo del tratamiento?
- ¿Qué medicamentos se usan más frecuentemente por condición médica?
- **Contexto del dataset:** ¿Quién lo recolectó? ¿Cómo se obtuvo?

Estos datos aprovechan la biblioteca de python faker reflejando la estructura que se encuentra comúnmente en el registro de atención médica en algún otro país. También se quiere resaltar que este conjunto de datos es solo para fines educativos, es totalmente sintético y no contiene datos reales del paciente.

<https://www.kaggle.com/datasets/prasad22/healthcare-dataset?resource=download>

2. EXPLORACIÓN INICIAL DEL DATASET

- **Descripción del dataset:** Número de registros, columnas, tipos de variables.

La base de datos cuenta con 55.500 registros de pacientes y 15 campos, siendo Billing Amount la variable objetivo y quedando 14 variables para el análisis. Se eliminaron columnas sin valor predictivo y solo se conservaron las variables numéricas, categóricas y datetime relevantes descritas a continuación.

categóricas	numéricas	DateTime	No Relevantes
			Name
	Age (discreta)		
Gender			
BoodType			
Medical_Condition			
		Data_of_Admission	
Doctor			
Hospital			
Insurance_Provider			
Room_Number			
AdmissionType			
		Discharge Date	
Medication			
Test_Results			

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Name              55500 non-null   object 
 1   Age               36865 non-null   float64
 2   Gender            36865 non-null   object 
 3   Blood Type        36865 non-null   object 
 4   Medical Condition 36865 non-null   object 
 5   Date of Admission 36865 non-null   object 
 6   Doctor            36865 non-null   object 
 7   Hospital           36865 non-null   object 
 8   Insurance Provider 36865 non-null   object 
 9   Billing Amount     36865 non-null   float64
 10  Room Number        36865 non-null   float64
 11  Admission Type    36865 non-null   object 
 12  Discharge Date     36865 non-null   object 
 13  Medication          36865 non-null   object 
 14  Test Results        36865 non-null   object 
dtypes: float64(3), object(12)
memory usage: 6.4+ MB
```

Tipo de variable en función de:

Variable	Naturaleza del Dato	Nivel de Medición	Posición en la Investigación
Age	Cuantitativa discreta	Cuantitativa de razón	Independiente
Gender	Cualitativa dicotómica	-	Independiente
Blood Type	Cualitativa politómica	-	Independiente
Medical condition	Cualitativa politómica	-	Independiente
Doctor	Cualitativa politómica	-	Independiente
Hospital	Cualitativa politómica	-	Independiente
Insurance Provider	Cualitativa politómica	-	Independiente
Room Number	Cualitativa politómica	-	Independiente
Admission Type	Cualitativa politómica	-	Independiente
Medication	Cualitativa politómica	-	Independiente
Test Results	Cualitativa politómica	-	Independiente
Billing Amount	-	-	Dependiente

- **Diccionario de datos:** Definiciones de cada campo.

- **Name** (Nombre): Nombres de los pacientes ingresados en el hospital y asociado con la historia clínica.
- **Age** (Edad): Edad de los pacientes en el momento del ingreso, expresa años.
- **Gender** (Sexo): Género de los pacientes, se trabajará con masculino y femenino
- **Blood Type** (Tipo de Sangre): Tipo de sangre de los pacientes, que puede ser uno de los tipos de sangre más comunes (por ejemplo, "A+", "O-", etc).
- **Medical Condition** (Condición médica): Descripción de la condición médica o diagnóstico por el cual el paciente es admitido, es decir, especifica la afección médica primaria o el diagnóstico asociado con el paciente, como "Diabetes", "Hipertensión", "Asma" y más.
- **Data of Admission** (Fecha de ingreso/alta): Fecha en que el paciente ingresó en el hospital
- **Doctor** (Médico): Nombre o identificador del médico responsable de la atención del paciente.
- **Hospital** (Hospital): Nombre o identificador del hospital donde se encuentra ingresado el paciente
- **Insurance Provider** (Proveedor de Seguros): Nombre del proveedor de seguros que cubre los gastos médicos del paciente, que puede ser una de varias opciones, incluidas "Aetna", "Blue Cross", "Cigna", "UnitedHealthcare" y "Medicare".
- **Billing Amount** (Monto facturado): Monto facturado al paciente o a su proveedor de seguros por los servicios médicos recibidos durante la estancia en el hospital. Es decir, la cantidad de dinero facturada por los servicios de atención médica del paciente durante su ingreso. Esto se expresa como un número flotante.
- **Room number** (Habitación): Identificador de la habitación donde se aloja el paciente.
- **Admission Type** (Tipo de admisión): Tipo de admisión que puede ser "Emergencia", "Electiva" o "Urgente", reflejando las circunstancias de la admisión.
- **Discharge Date** (Fecha de alta): La fecha en la que el paciente fue dado de alta del centro de atención médica, en función de la fecha de ingreso y un número aleatorio de días dentro de un rango realista.
- **Medication** (Medicación): Lista de medicamentos recetados o administrados al paciente durante su hospitalización. Algunos ejemplos son "Aspirina", "Ibuprofeno", "Penicilina", "Paracetamol" y "Lipitor"

- **Test Results** (Resultado de pruebas): Resultados de las pruebas médicas realizadas al paciente durante su estancia hospitalaria. Los valores posibles incluyen "Normal", "Anormal" o "No concluyente", lo que indica el resultado de la prueba.

- **Manejo de datos faltantes y duplicados.**

	df.isnull().sum()	round((df.isnull().sum() / len(df)) * 100, 1)
	0	0
Name	0	0.0
Age	18635	33.6
Gender	18635	33.6
Blood Type	18635	33.6
Medical Condition	18635	33.6
Date of Admission	18635	33.6
Doctor	18635	33.6
Hospital	18635	33.6
Insurance Provider	18635	33.6
Billing Amount	18635	33.6
Room Number	18635	33.6
Admission Type	18635	33.6
Discharge Date	18635	33.6
Medication	18635	33.6
Test Results	18635	33.6

Se eliminan 18635 columnas debido a que se observan columnas faltantes en todas las variables excepto Name. Por lo tanto, pasaremos a tener un dataset compuesto por 36865 registros.

No se registraron valores duplicados.

```
duplicados = df.duplicated().sum() #Contar el número total de filas duplicadas (exceptuando
print(f'El número de duplicados es {duplicados}')
# tenemos 369 duplicados

El número de duplicados es 0
```

- **Registros de datos inconsistentes**

- **Comprobar el formato de fecha no válido**

```
# Comprobar si los formatos de fecha son iguales
import warnings
import re
columnas_fecha = ['Date of Admission', 'Discharge Date']
for col in columnas_fecha:
    print(f"\nFormatos únicos en la columna: {col}")
    formatos = df[col].astype(str).apply(lambda x: re.sub(r'\d', 'D', x))
    print(formatos.value_counts())
warnings.filterwarnings('ignore')

\ Formatos únicos en la columna: Date of Admission
Date of Admission
DDDD-DD-DD    36865
nan            18635
Name: count, dtype: int64
\ Formatos únicos en la columna: Discharge Date
Discharge Date
DDDD-DD-DD    36865
nan            18635
Name: count, dtype: int64
```

```
df['Date of Admission'] = pd.to_datetime(df['Date of Admission'], errors='coerce')
df['Discharge Date'] = pd.to_datetime(df['Discharge Date'], errors='coerce')

# Encuentra inconsistencias lógicas: alta antes de ingreso
inconsistencias = df[df['Discharge Date'] < df['Date of Admission']]

# Mostrar resultados
print(f"Registros con fechas inconsistentes: {len(inconsistencias)}")
print(inconsistencias[['Date of Admission', 'Discharge Date']])
```

Python

```
Registros con fechas inconsistentes: 0
Empty DataFrame
Columns: [Date of Admission, Discharge Date]
Index: []
```

- **Comprobar la inconsistencia lógica de fechas, que la fecha de alta no sea mayor a la de ingreso.**

```
df['Date of Admission'] = pd.to_datetime(df['Date of Admission'], errors='coerce')
df['Discharge Date'] = pd.to_datetime(df['Discharge Date'], errors='coerce')

# Encuentra inconsistencias lógicas: alta antes de ingreso
inconsistencias = df[df['Discharge Date'] < df['Date of Admission']]

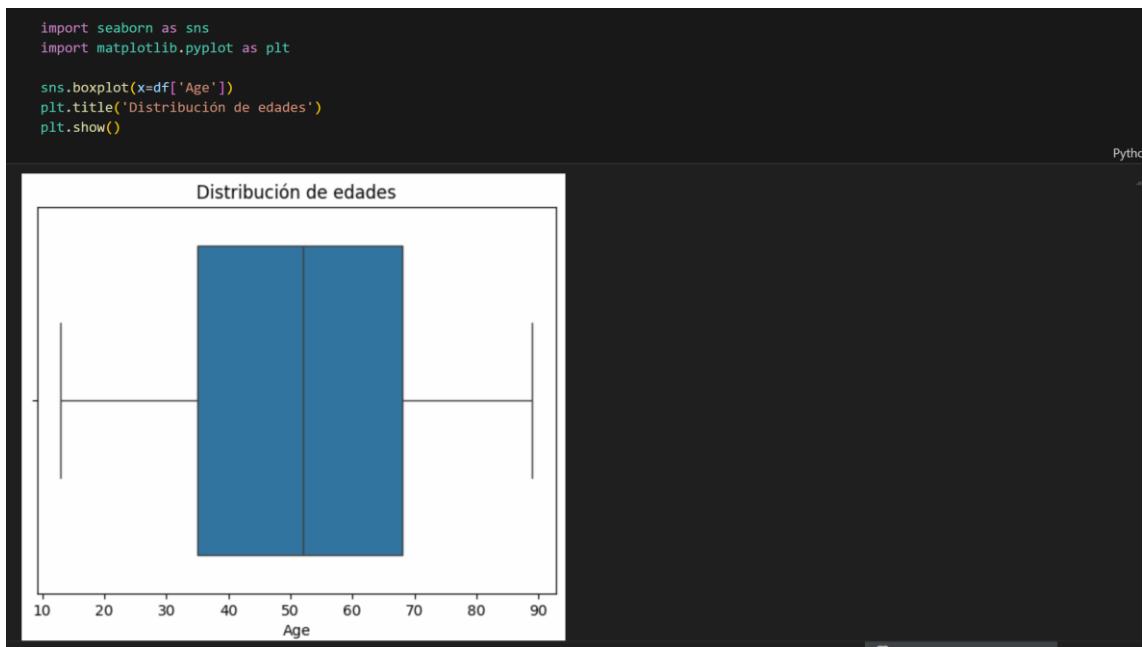
# Mostrar resultados
print(f"Registros con fechas inconsistentes: {len(inconsistencias)}")
print(inconsistencias[['Date of Admission', 'Discharge Date']])
```

Python

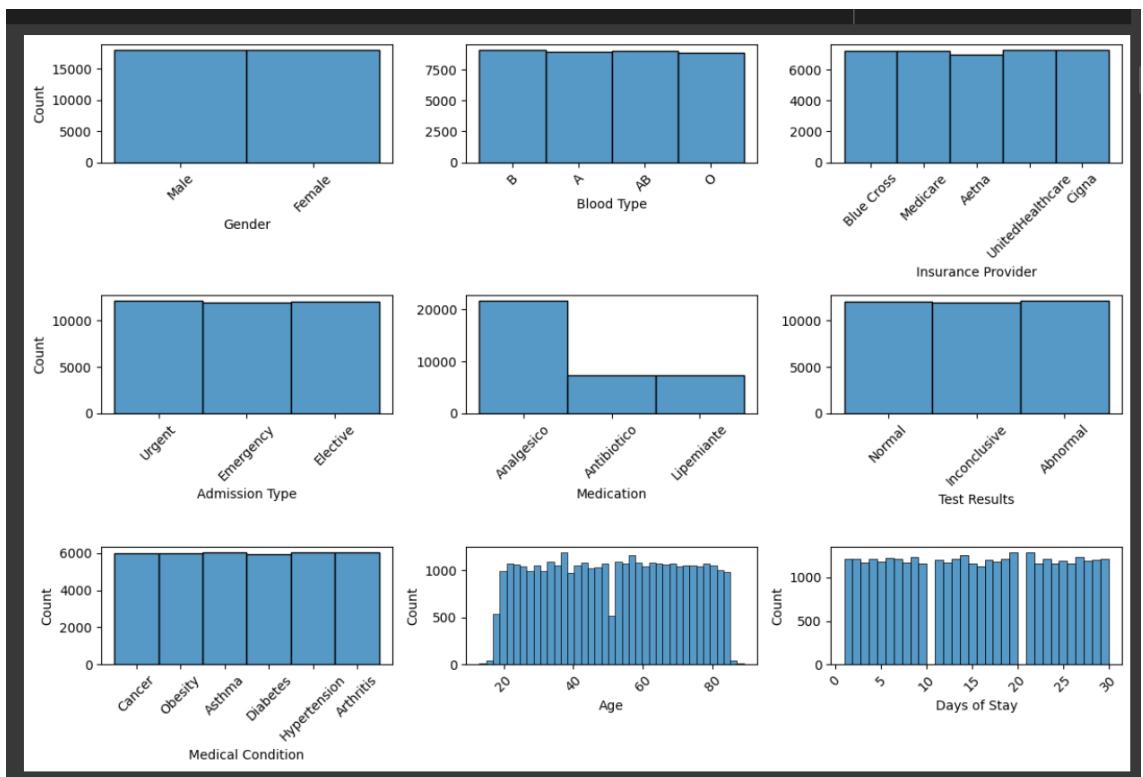
```
Registros con fechas inconsistentes: 0
Empty DataFrame
Columns: [Date of Admission, Discharge Date]
Index: []
```

- **Identificación de posibles outliers.**

Nuestro dataset original no contenía outliers



Visualización del dataset original:



Observamos que el dataset que se está utilizando en este estudio es altamente sintético, lo que significa que ha sido generado artificialmente y no refleja con precisión la complejidad ni la variabilidad de los datos clínicos reales. Esta

característica limita significativamente el alcance de los análisis posibles, ya que no se observan comportamientos ni relaciones propias de contextos reales.

- **Modificación artificial del dataset original.**

Dado que los datos carecen de profundidad y riqueza informativa, nos vemos en la necesidad de complementarlos con registros balanceados que imiten patrones estadísticos verosímiles. Esta integración de datos similares a los reales permitirá mantener la validez metodológica del estudio y asegurar que las conclusiones obtenidas sean más representativas. En este sentido, trabajar únicamente con información sintética podría introducir sesgos importantes, razón por la cual es imprescindible enriquecer el conjunto con ejemplos que se acerquen a escenarios clínicos auténticos.

A partir de este punto del estudio se utilizará el dataset modificado, llamado mi_data.csv.

```
1 data.to_csv('mi_data.csv', index=False)
```

3. LIMPIEZA Y PROCESAMIENTO DE DATOS

Dataset “mi_data.csv”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name              55500 non-null   object  
 1   Age               55500 non-null   int64  
 2   Gender            55500 non-null   object  
 3   Blood Type        55500 non-null   object  
 4   Medical Condition 55500 non-null   object  
 5   Date of Admission 55500 non-null   object  
 6   Doctor            55500 non-null   object  
 7   Hospital           55500 non-null   object  
 8   Insurance Provider 55500 non-null   object  
 9   Billing Amount     55500 non-null   float64 
 10  Room Number       55500 non-null   int64  
 11  Admission Type   55500 non-null   object  
 12  Discharge Date    55500 non-null   object  
 13  Medication         55500 non-null   object  
 14  Test Results       55500 non-null   object  
dtypes: float64(1), int64(2), object(12)
memory usage: 6.4+ MB
```

- Conversión de tipos de datos (fechas, números, categorías).

```
import warnings
import re # Comprobar si los formatos de fecha son iguales
columnas_fecha = ['Date of Admission', 'Discharge Date']
for col in columnas_fecha:
    print(f"\nFormatos únicos en la columna: {col}")
    formatos = data[col].astype(str).apply(lambda x: re.sub(r'\d', 'D', x))
    print(formatos.value_counts())
warnings.filterwarnings('ignore')
data['Date of Admission'] = pd.to_datetime(data['Date of Admission'], errors='coerce')
data['Discharge Date'] = pd.to_datetime(data['Discharge Date'], errors='coerce')

inconsistencias = data[data['Discharge Date'] < data['Date of Admission']] # Encuentra inconsistencias lógicas
print(f"\nRegistros con fechas inconsistentes: {len(inconsistencias)}") # Mostrar resultados
print(inconsistencias[['Date of Admission', 'Discharge Date']])

\ Formatos únicos en la columna: Date of Admission
Date of Admission
DDDD-DD-DD      55392
Name: count, dtype: int64
\ Formatos únicos en la columna: Discharge Date
Discharge Date
DDDD-DD-DD      55392
Name: count, dtype: int64

Registros con fechas inconsistentes: 0
Empty DataFrame
Columns: [Date of Admission, Discharge Date]
Index: []
```

No se encontró ninguna inconsistencia entre las fechas de admisión y salida, ya que todas las entradas mantienen una secuencia lógica y coherente. Además, ambas variables presentan un formato homogéneo, lo que facilita su análisis y procesamiento.

- **Tratamiento de valores nulos y duplicados**

Se recomprobó que este nuevo dataset modificado artificialmente no contiene nulos ni duplicados.

```
1 df.isnull().sum()

Name          0
Age           0
Gender         0
Blood Type    0
Medical Condition 0
Date of Admission 0
Doctor          0
Hospital        0
Insurance Provider 0
Billing Amount  0
Room Number    0
Admission Type 0
Discharge Date 0
Medication      0
Test Results    0
dtype: int64

duplicados_totales = df_unicos[df_unicos.duplicated(keep=False)].sum() #comprobamos que no quedan duplicados
print(" Filas duplicadas en general:")
print(f'El numero de duplicados en el dataframe es: \n{duplicados_totales}')

Filas duplicadas en general:
El numero de duplicados en el dataframe es:
Name          0
Age           0
Gender         0
Blood Type    0
Medical Condition 0
Date of Admission 0
Doctor          0
Hospital        0
Insurance Provider 0
Billing Amount  0.0
Room Number    0
Admission Type 0
Discharge Date 0
Medication      0
Test Results    0
dtype: object
```

- **Agrupamiento de variables categóricas.**

```
print(data['Gender'].unique())
print(data['Blood Type'].unique())
print(data['Medical Condition'].unique())

print(data['Doctor'].unique())
print(data['Hospital'].unique())

print(data['Insurance Provider'].unique())
print(data['Admission Type'].unique())
print(data['Medication'].unique())
print(data['Test Results'].unique())

['Male' 'Female']
['B-' 'A+' 'A-' 'O+' 'AB+' 'AB-' 'B+' 'O-']
['Arthritis' 'Diabetes' 'Obesity' 'Asthma' 'Hypertension' 'Cancer']
['Matthew Smith' 'Samantha Davies' 'Tiffany Mitchell' ... 'Deborah Sutton'
 'Mary Bartlett' 'Alec May']
['Sons and Miller' 'Kim Inc' 'Cook PLC' ... 'Guzman Jones and Graves,' 
 'and Williams, Brown McKenzie' 'Moreno Murphy, Griffith and']
['Blue Cross' 'Medicare' 'Aetna' 'UnitedHealthcare' 'Cigna']
['Elective' 'Emergency' 'Urgent']
['Aspirin' 'Paracetamol' 'Lipitor' 'Ibuprofen' 'Penicillin']
['Abnormal' 'Inconclusive' 'Normal']
```

Blood Type:

```
grupo_sanguineo_map = { # Diccionario para mapear cada tipo a un grupo
    'A+': 'A',
    'A-': 'A',
    'B+': 'B',
    'B-': 'B',
    'AB+': 'AB',
    'AB-': 'AB',
    'O+': 'O',
    'O-': 'O'
}
data['Blood Type'] = data['Blood Type'].map(grupo_sanguineo_map) # sustituir la columna Blood Type por la
```

```

conteo_BT = data['Blood Type'].value_counts()
print(conteo_BT)

Blood Type
A      13892
B      13869
AB     13866
O      13765
Name: count, dtype: int64

```

Medication:

```

medication_map = { # Diccionario para mapear cada tipo a un grupo
    'Aspirin': 'Analgesico',
    'Ibuprofen': 'Analgesico',
    'Paracetamol': 'Analgesico',
    'Penicillin': 'Antibiotico',
    'Lipitor': 'Lipemiantante',
}
data['Medication'] = data['Medication'].map(medication_map) # sustituir la columna Medication por la nueva
data

conteo_M = data['Medication'].value_counts()
print(conteo_M)

Medication
Analgesico    41632
Lipemiantante 8248
Antibiotico   5512
Name: count, dtype: int64

```

Tras realizar las distintas agrupaciones, cada campo presenta los siguientes atributos resultantes:

Variable	Valores	Agrupación	
Gender	2	-	
Blood Type	8	4	A, B, AB, O
Medical condition	6	-	
Doctor y Hospital	Tienen muchos valores únicos	-	
Insurance provider	5	-	
Admission Type	3	-	
Medication	5	3	Analgésico, Lipemiantante, Antibiótico
Test Results	3		Test Results

```

print(data['Gender'].unique())
print(data['Blood Type'].unique())
print(data['Medical Condition'].unique())

print(data['Doctor'].unique())
print(data['Hospital'].unique())

print(data['Insurance Provider'].unique())
print(data['Admission Type'].unique())
print(data['Medication'].unique())
print(data['Test Results'].unique())

['Male' 'Female']
['B' 'A' 'O' 'AB']
['Arthritis' 'Diabetes' 'Obesity' 'Asthma' 'Hypertension' 'Cancer']
['Matthew Smith' 'Samantha Davies' 'Tiffany Mitchell' ... 'Deborah Sutton'
 'Mary Bartlett' 'Alec May']
['Sony and Miller' 'Kim Inc' 'Cook PLC' ... 'Guzman Jones and Graves,' 
 'and Williams, Brown McKenzie' 'Moreno Murphy, Griffith and']
['Blue Cross' 'Medicare' 'Aetna' 'UnitedHealthcare' 'Cigna']
['Elective' 'Emergency' 'Urgent']
['Analgesico' 'Lipemante' 'Antibiotico']
['Abnormal' 'Inconclusive' 'Normal']

```

- **Filtrado de valores negativos correspondientes a una devolución a la aseguradora médica**

Se excluyen los valores negativos de Billing Amount, ya que representan devoluciones de dinero al usuario. Nuestro objetivo es predecir el monto de la factura, y estos valores distorsionan el análisis al no aportar información relevante.

```

data=df_unicos
data.describe()

      Age  Billing Amount  Room Number
count  55500.000000  55500.000000  55500.000000
mean   39.622468  25539.316097  301.134829
std    12.848535  14211.454431  115.243069
min    13.000000  -2008.492140  101.000000
25%   30.000000  13241.224652  202.000000
50%   39.000000  25538.069376  302.000000
75%   48.000000  37820.508436  401.000000
max   84.000000  52764.276736  500.000000

```

```

data_infzero = data[data['Billing Amount']<0]
data_infzero
# es un numero de registros poco relevante y lo eliminamos (0.18% o 67/36127)

```

```

1 data.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55392 entries, 0 to 55391
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Name              55392 non-null   object  
 1   Age               55392 non-null   int64  
 2   Gender            55392 non-null   object  
 3   Blood Type        55392 non-null   object  
 4   Medical Condition 55392 non-null   object  
 5   Date of Admission 55392 non-null   datetime64[ns]
 6   Doctor            55392 non-null   object  
 7   Hospital           55392 non-null   object  
 8   Insurance Provider 55392 non-null   object  
 9   Billing Amount     55392 non-null   float64 
10  Room Number        55392 non-null   int64  
11  Admission Type    55392 non-null   object  
12  Discharge Date    55392 non-null   datetime64[ns]
13  Medication          55392 non-null   object  
14  Test Results        55392 non-null   object  
15  Days of Stay       55392 non-null   int64  
16  AD_Mes             55392 non-null   category
17  AD_DiaSemana       55392 non-null   category
18  AD_Trimestre       55392 non-null   category
19  AD_Anio            55392 non-null   category
...
26  Num_cronicas       55392 non-null   int64  
27  Num_Cronicas        55392 non-null   int64  
dtypes: category(9), datetime64[ns](2), float64(1), int64(6), object(10)
memory usage: 8.5+ MB

```

- **Outliers**

Se identificaron y eliminaron los outliers de las variables 'Age' y 'Billing Amount' utilizando el método del rango intercuartílico (IQR). Esto permitió conservar únicamente los valores dentro de un rango estadísticamente razonable, mejorando la calidad del análisis.

```

1 # Comprobamos que hay outliers usando método IQR:
2 quant_vars = ['Age', 'Billing Amount']
3 for var in quant_vars:
4
5     q1, q3 = np.percentile(data[var], [25, 75])
6     iqr=q3-q1
7     lim_inferior = q1 - 1.5 * iqr
8     lim_superior = q3 + 1.5 * iqr
9     print(f"\nEn la variable {var}")
10    print(f"Q1:{q1}, Q3:{q3}, IQR: {iqr}")
11    print(f"Limite inferior: {lim_inferior}")
12    print(f"Limite superior: {lim_superior}")
13
14    outliers_iqr_var= data[var][(data[var] < lim_inferior) | (data[var] > lim_superior)]
15    print(f"\nOutliers using IQR method: \n{outliers_iqr_var}")
✓ 0.0s

```

```

En la variable Age
Q1:30.0, Q3:48.0, IQR: 18.0
Límite inferior: 3.0
Límite superior: 75.0

Outliers using IQR method:
67      77
2074     77
2791     77
2825     77
2910     76
...
53318    76
53642    76
53812    76
54840    76
55185    79
Name: Age, Length: 155, dtype: int64

En la variable Billing Amount
Q1:13297.478681528502, Q3:37849.210062242, IQR: 24551.731380713492
Límite inferior: -23530.118389541734
Límite superior: 74676.80713331224

Outliers using IQR method:
Series([], Name: Billing Amount, dtype: float64)

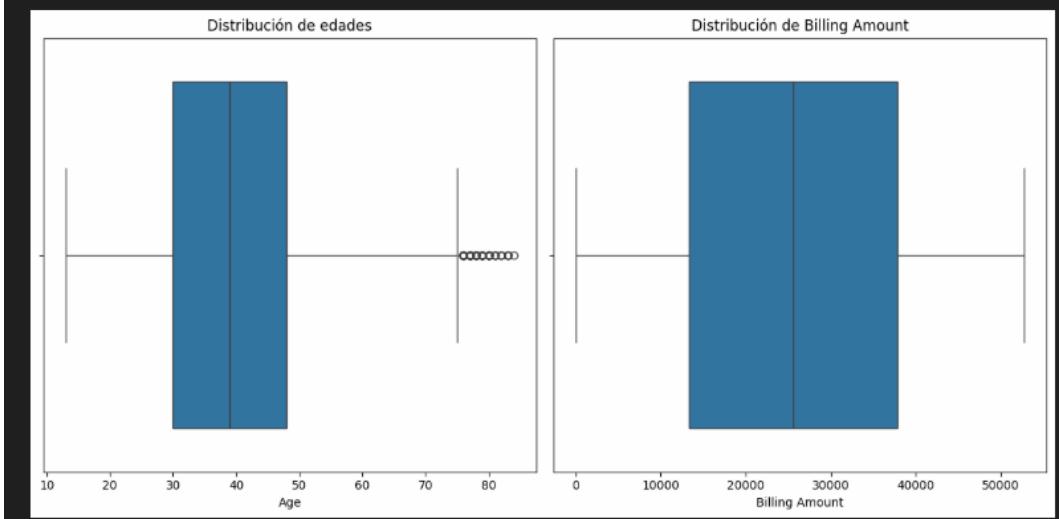
```

```

1 fig, axs = plt.subplots(1, 2, figsize=(12, 6))
2 sns.boxplot(x=data['Age'], ax=axs[0])
3 axs[0].set_title('Distribución de edades')
4
5 sns.boxplot(x=data['Billing Amount'], ax=axs[1])
6 axs[1].set_title('Distribución de Billing Amount')
7
8 plt.tight_layout()
9 plt.show()
10 #No hay outliers para

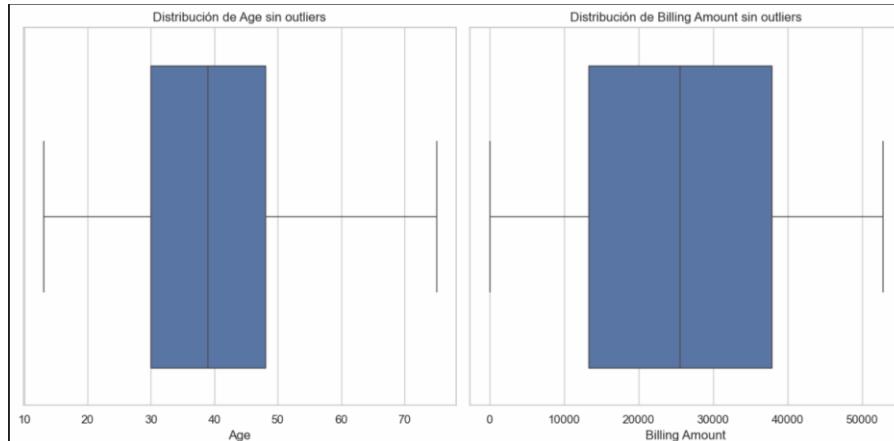
```

Python



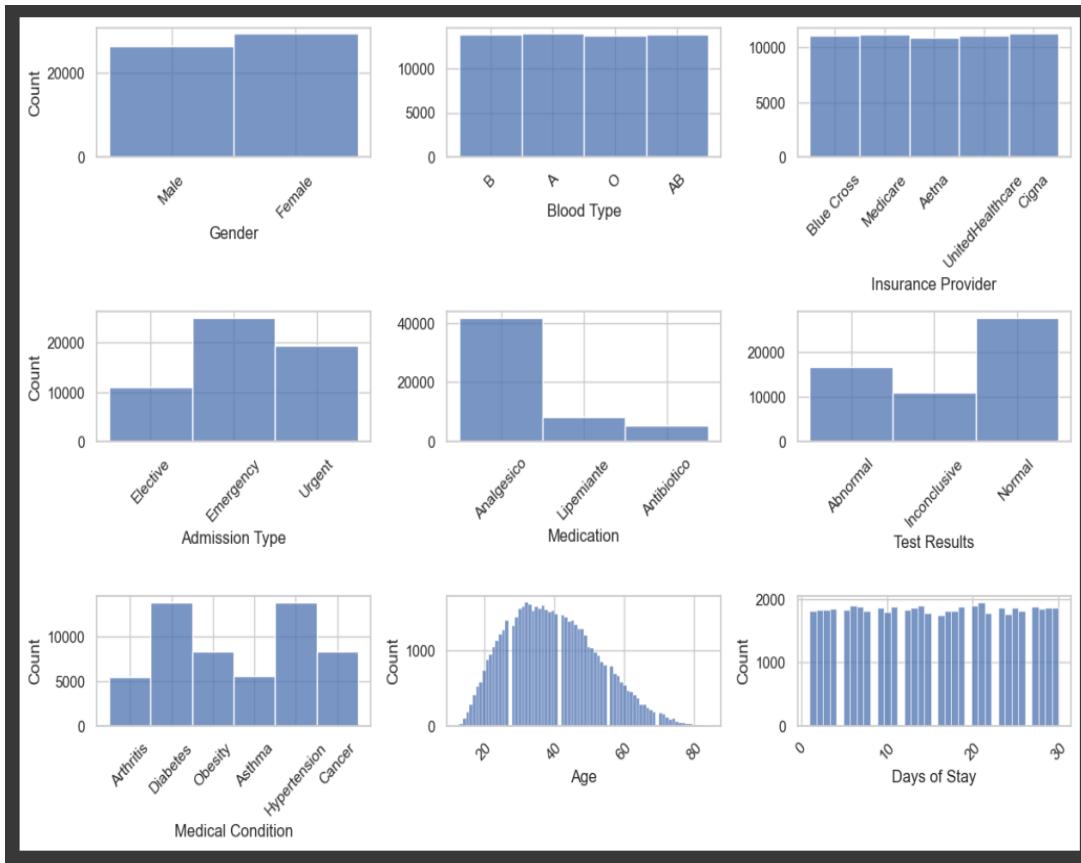
Eliminamos los outliers

```
1 quant_vars = ['Age', 'Billing Amount']
2
3 # Crear filtro para conservar filas sin outliers en ambas variables
4 filtro_sin_outliers = np.ones(len(data), dtype=bool)
5
6 for var in quant_vars:
7     q1, q3 = np.percentile(data[var], [25, 75])
8     iqr = q3 - q1
9     lim_inferior = q1 - 1.5 * iqr
10    lim_superior = q3 + 1.5 * iqr
11
12    filtro_sin_outliers &= (data[var] >= lim_inferior) & (data[var] <= lim_superior)
13
14 # Filtrar datos sin outliers
15 data_sin_outliers = data[filtro_sin_outliers]
16 data_sin_outliers = data #para consistencia de la propagación del código
17
18 # Graficar boxplots sin outliers
19 fig, axs = plt.subplots(1, 2, figsize=(12, 6))
20
21 for i, var in enumerate(quant_vars):
22     sns.boxplot(x=data_sin_outliers[var], ax=axs[i])
23     axs[i].set_title(f'Distribución de {var} sin outliers')
24
25 plt.tight_layout()
26 plt.show()
✓ 0.6s
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 55237 entries, 0 to 55391
Data columns (total 28 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name             55237 non-null   object 
 1   Age              55237 non-null   int64  
 2   Gender            55237 non-null   object 
 3   Blood Type        55237 non-null   object 
 4   Medical Condition 55237 non-null   object 
 5   Date of Admission 55237 non-null   datetime64[ns]
 6   Doctor            55237 non-null   object 
 7   Hospital           55237 non-null   object 
 8   Insurance Provider 55237 non-null   object 
 9   Billing Amount     55237 non-null   float64
 10  Room Number       55237 non-null   int64  
 11  Admission Type    55237 non-null   object 
 12  Discharge Date    55237 non-null   datetime64[ns]
 13  Medication          55237 non-null   object 
 14  Test Results        55237 non-null   object 
 15  Days of Stay        55237 non-null   int64  
 16  AD Mes              55237 non-null   category
 17  AD_DiaSemana        55237 non-null   category
 18  AD_Trimestre        55237 non-null   category
 19  AD_Año              55237 non-null   category
 ...
 26  Num_cronicas        55237 non-null   int64  
 27  Num_Cronicas         55237 non-null   int64  
dtypes: category(9), datetime64[ns](2), float64(1), int64(6), object(10)
memory usage: 8.9+ MB
```

Visualizaciones del DataFrame modificado



- **Transformación de las variables: creación de “Days of Stay”**

Calculamos los días de estancia en el hospital a partir de Data of Admision y Discharge data para cada paciente, posteriormente creamos un nuevo DataFrame para time y categorización de las fechas.

```
data['Days of Stay']=(data['Discharge Date']-data['Date of Admission']).dt.days
data = data.reset_index(drop=True) #el indice se reinicia (no afecta directamente al análisis pero puede causar confusión si haces
#operaciones que dependan de indices consecutivos (como .iloc).
data.info()
```

```
data['Date of Admission'] = pd.to_datetime(data['Date of Admission'], errors='coerce')

data['AD_Mes'] = data['Date of Admission'].dt.month.astype('category')
data['AD_DiaSemana'] = data['Date of Admission'].dt.day_name().astype('category')
data['AD_Trimestre'] = data['Date of Admission'].dt.quarter.astype('category')
data['AD_Año'] = data['Date of Admission'].dt.year.astype('category')

data['Discharge Date'] = pd.to_datetime(data['Discharge Date'], errors='coerce')

data['DD_Mes'] = data['Discharge Date'].dt.month.astype('category')
data['DD_DiaSemana'] = data['Discharge Date'].dt.day_name().astype('category')
data['DD_Trimestre'] = data['Discharge Date'].dt.quarter.astype('category')
data['DD_Año'] = data['Discharge Date'].dt.year.astype('category')
```

```

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55392 entries, 0 to 55391
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Name              55392 non-null   object  
 1   Age               55392 non-null   int64  
 2   Gender            55392 non-null   object  
 3   Blood Type        55392 non-null   object  
 4   Medical Condition 55392 non-null   object  
 5   Date of Admission 55392 non-null   datetime64[ns]
 6   Doctor            55392 non-null   object  
 7   Hospital           55392 non-null   object  
 8   Insurance Provider 55392 non-null   object  
 9   Billing Amount     55392 non-null   float64 
 10  Room Number       55392 non-null   int64  
 11  Admission Type   55392 non-null   object  
 12  Discharge Date   55392 non-null   datetime64[ns]
 13  Medication         55392 non-null   object  
 14  Test Results       55392 non-null   object  
 15  Days of Stay      55392 non-null   int64  
 16  AD_Mes             55392 non-null   category
 17  AD_DiaSemana      55392 non-null   category
 18  AD_Trimestre      55392 non-null   category
 19  AD_Año             55392 non-null   category
 20  DD_Mes             55392 non-null   category
 21  DD_DiaSemana      55392 non-null   category
 22  DD_Trimestre      55392 non-null   category
 23  DD_Año             55392 non-null   category
dtypes: category(8), datetime64[ns](2), float64(1), int64(3), object(10)
memory usage: 7.2+ MB

```

- Creación de los tres DataFrames en función del tipo de variable para el análisis exploratorio de datos

```

1 df_num=data[['Age','Billing Amount', 'Days of Stay']]
2 df_num.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55392 entries, 0 to 55391
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Age               55392 non-null   int64  
 1   Billing Amount    55392 non-null   float64 
 2   Days of Stay     55392 non-null   int64  
dtypes: float64(1), int64(2)
memory usage: 1.3 MB

```

```
1 df_cat=data[['Gender','Blood Type','Medical Condition','Doctor','Hospital',
2 [ ]|['Insurance Provider', 'Admission Type', 'Medication', 'Test Results']]
3 df_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55392 entries, 0 to 55391
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Gender            55392 non-null   object  
 1   Blood Type        55392 non-null   object  
 2   Medical Condition 55392 non-null   object  
 3   Doctor            55392 non-null   object  
 4   Hospital           55392 non-null   object  
 5   Insurance Provider 55392 non-null   object  
 6   Admission Type    55392 non-null   object  
 7   Medication         55392 non-null   object  
 8   Test Results       55392 non-null   object  
dtypes: object(9)
memory usage: 3.8+ MB
```

```
1 df_time = data[['Date of Admission','AD_Mes', 'AD_DiaSemana', 'AD_Trimestre', 'AD_Año','AD_Mes', 'DD_DiaSemana',
2 [ ]| 'DD_Trimestre', 'DD_Año','Discharge Date', 'Days of Stay', 'Admission Type']]
3 df_time.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55392 entries, 0 to 55391
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype    
--- 
 0   Date of Admission 55392 non-null   datetime64[ns]
 1   AD_Mes             55392 non-null   category 
 2   AD_DiaSemana       55392 non-null   category 
 3   AD_Trimestre       55392 non-null   category 
 4   AD_Año              55392 non-null   category 
 5   AD_Mes             55392 non-null   category 
 6   DD_DiaSemana       55392 non-null   category 
 7   DD_Trimestre       55392 non-null   category 
 8   DD_Año              55392 non-null   category 
 9   Discharge Date     55392 non-null   datetime64[ns]
 10  Days of Stay        55392 non-null   int64    
 11  Admission Type     55392 non-null   object  
dtypes: category(8), datetime64[ns](2), int64(1), object(1)
memory usage: 2.1+ MB
```

4. ANÁLISIS EXPLORATORIO DE DATOS (EDA)

Estadísticas descriptivas

```
1 quant_vars = ['Age', 'Days of Stay', 'Billing Amount']
2 for var in quant_vars:
3     media = df_num[var].mean()
4     devstd= df_num[var].std()
5     mediana = df_num[var].median()
6     varianza = df_num[var].var()
7     moda= df_num[var].mode()[0]# Pandas tiene el método .mode() que devuelve una Serie (porque puede haber
8     #más de una moda). Por eso, para imprimirla como número necesitas seleccionar la primera moda
9     print(f'La variable: {var}, tiene una media de {media:.0f}')
10    print(f'La variable: {var}, tiene una desviación estandard de {devstd:.0f}')
11    print(f'La variable: {var}, tiene una mediana de {mediana:.0f}')
12    print(f'La variable: {var}, tiene una varianza de {varianza:.0f}')
13    print(f'La variable: {var}, tiene una moda de {moda:.0f}')
7]
```

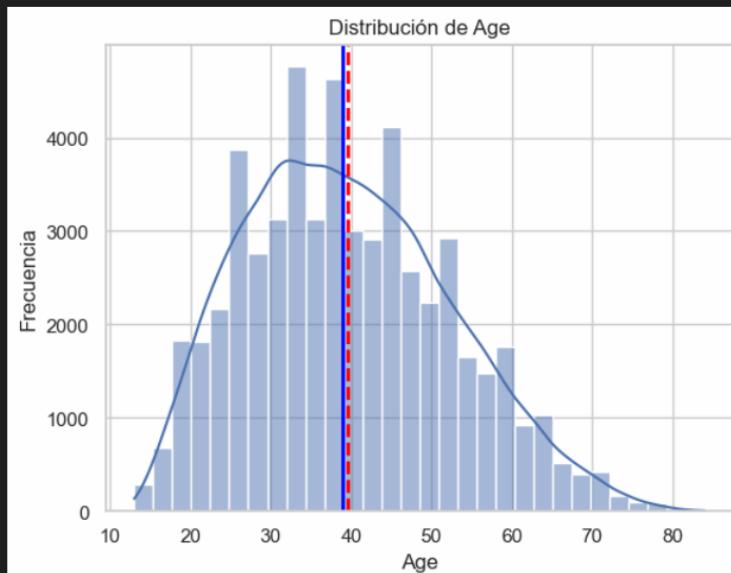
```
La variable: Age, tiene una media de 40
La variable: Age, tiene una desviación estandard de 13
La variable: Age, tiene una mediana de 39
La variable: Age, tiene una varianza de 165
La variable: Age, tiene una moda de 32
La variable: Days of Stay, tiene una media de 16
La variable: Days of Stay, tiene una desviación estandard de 9
La variable: Days of Stay, tiene una mediana de 15
La variable: Days of Stay, tiene una varianza de 75
La variable: Days of Stay, tiene una moda de 21
La variable: Billing Amount, tiene una media de 25590
La variable: Billing Amount, tiene una desviación estandard de 14179
La variable: Billing Amount, tiene una mediana de 25574
La variable: Billing Amount, tiene una varianza de 201034329
La variable: Billing Amount, tiene una moda de 69
```

Con estos valores de media y mediana supondríamos una distribución normal, pero analizaremos la distribución.

Distribución de variables numéricas

El análisis de variables muestra que Age, Days of Stay y Billing Amount no tienen una distribución normal, y tal como se muestra a continuación:

```
1 mean_age = df_num['Age'].mean()
2 median_age = df_num['Age'].median()
3 sns.histplot(data=df_num, x='Age', kde=True, bins=30)
4 plt.axvline(mean_age, color='red', linestyle='--', linewidth=2, label=f'Media: {mean_age:.1f}')
5 plt.axvline(median_age, color='blue', linestyle='-', linewidth=2, label=f'Mediana: {median_age:.1f}')
6 plt.title('Distribución de Age')
7 plt.xlabel('Age')
8 plt.ylabel('Frecuencia')
9 plt.show()
10
```



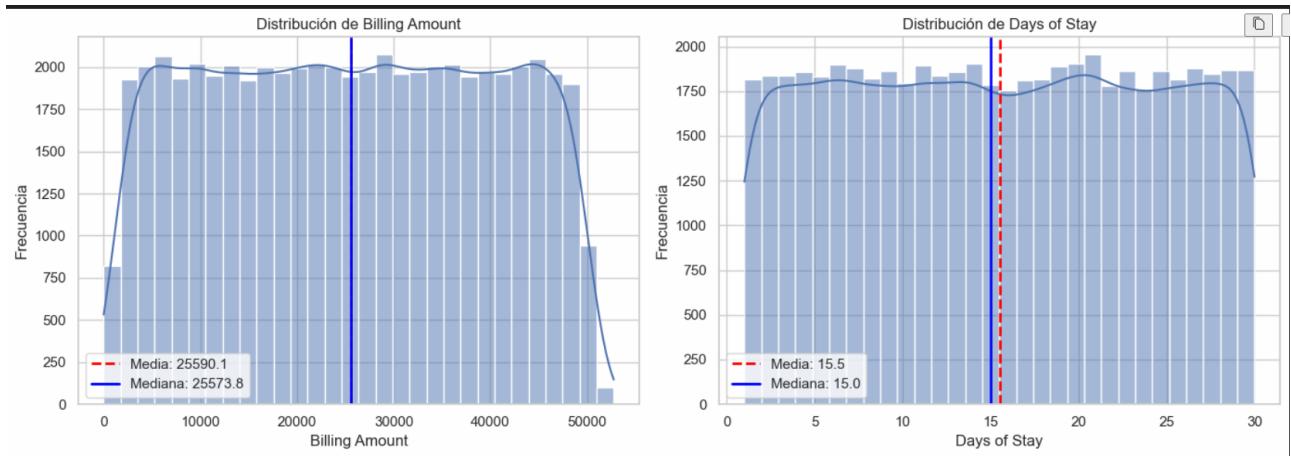
```
mean_billing = df_num['Billing Amount'].mean()
median_billing = df_num['Billing Amount'].median()

mean_days = df_num['Days of Stay'].mean()
median_days = df_num['Days of Stay'].median()

fig, axes = plt.subplots(1, 2, figsize=(14, 5))
# Histograma + KDE de Billing Amount
sns.histplot(data=df_num, x='Billing Amount', kde=True, bins=30, ax=axes[0])
axes[0].axvline(mean_billing, color='red', linestyle='--', linewidth=2, label=f'Media: {mean_billing:.1f}')
axes[0].axvline(median_billing, color='blue', linestyle='-', linewidth=2, label=f'Mediana: {median_billing:.1f}')
axes[0].set_title('Distribución de Billing Amount')
axes[0].set_xlabel('Billing Amount')
axes[0].set_ylabel('Frecuencia')
axes[0].legend()

# Histograma + KDE de Days of Stay
sns.histplot(data=df_num, x='Days of Stay', kde=True, bins=30, ax=axes[1])
axes[1].axvline(mean_days, color='red', linestyle='--', linewidth=2, label=f'Media: {mean_days:.1f}')
axes[1].axvline(median_days, color='blue', linestyle='-', linewidth=2, label=f'Mediana: {median_days:.1f}')
axes[1].set_title('Distribución de Days of Stay')
axes[1].set_xlabel('Days of Stay')
axes[1].set_ylabel('Frecuencia')
axes[1].legend()

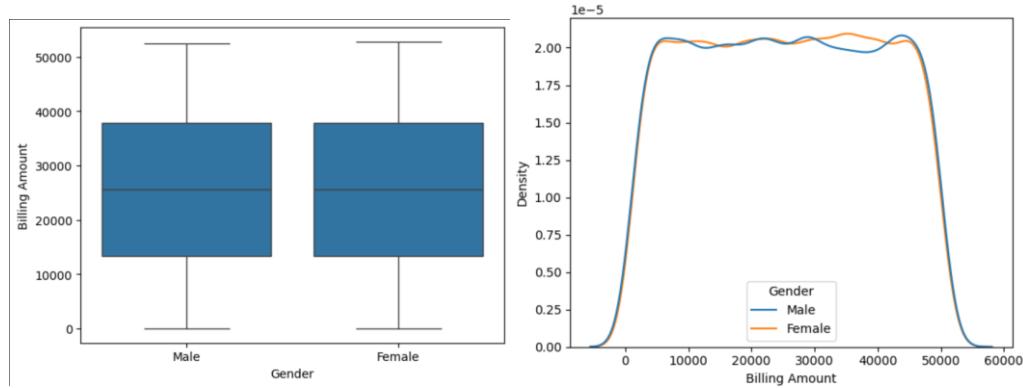
plt.tight_layout()
plt.show()
```



Distribución de variables categóricas

Cuando analizamos las variables categóricas observamos que todas no están balanceadas y tampoco tienen una distribución normal como muestran los kde.plots

GENDER:

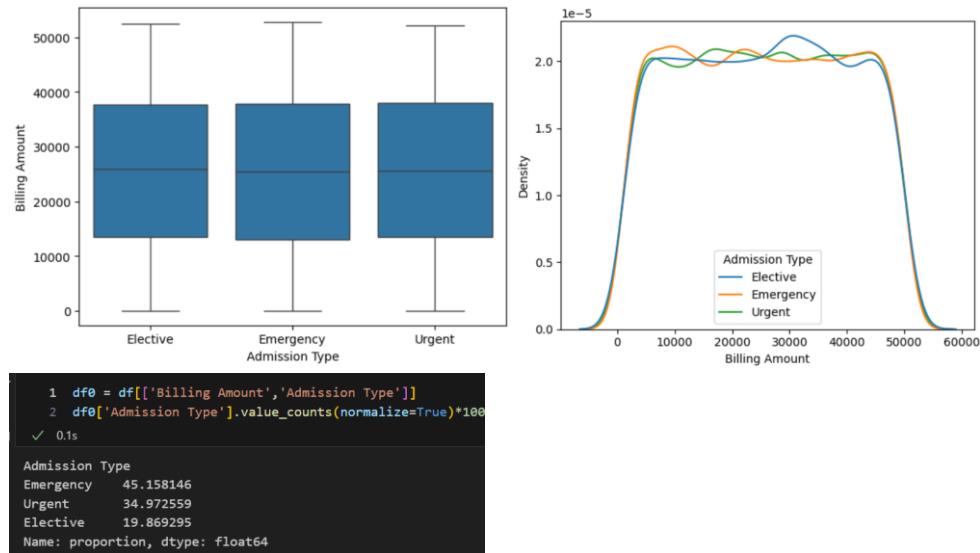


```

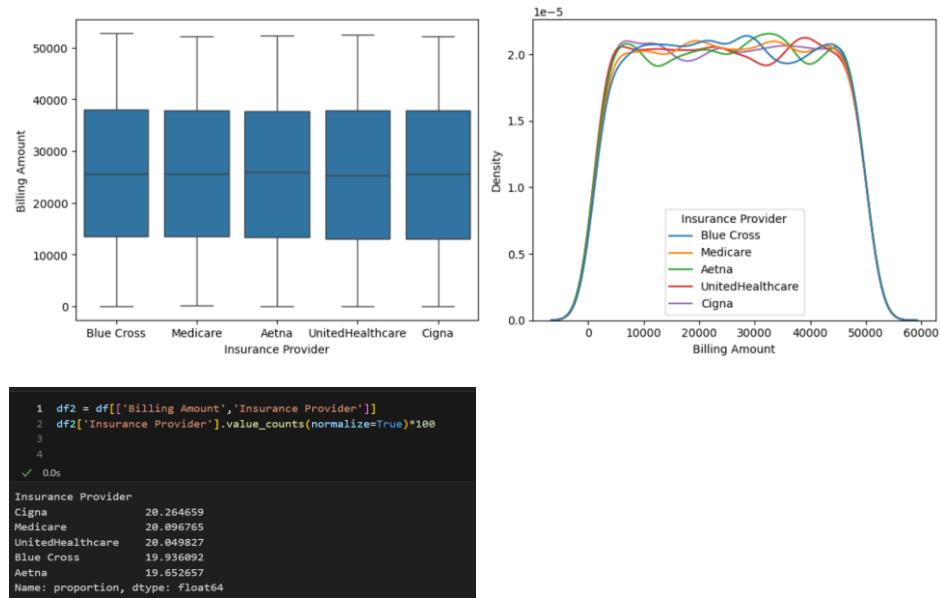
1 df1 = df[['Billing Amount','Gender']]
2 df1['Gender'].value_counts(normalize=True)*100
3
4 0.0s
5
6 Gender
7 Female 52.691724
8 Male 47.308276
9 Name: proportion, dtype: float64

```

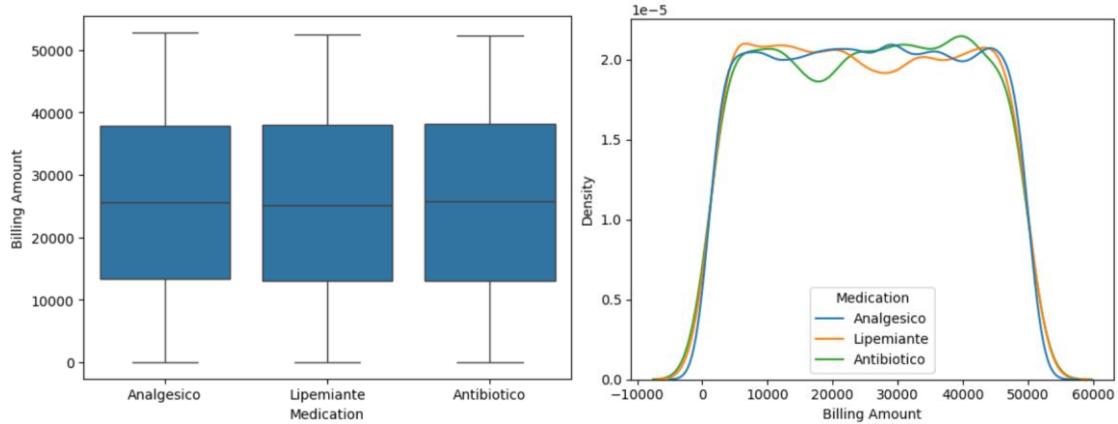
ADMISSION TYPE:



INSURANCE PROVIDER:



MEDICATION:

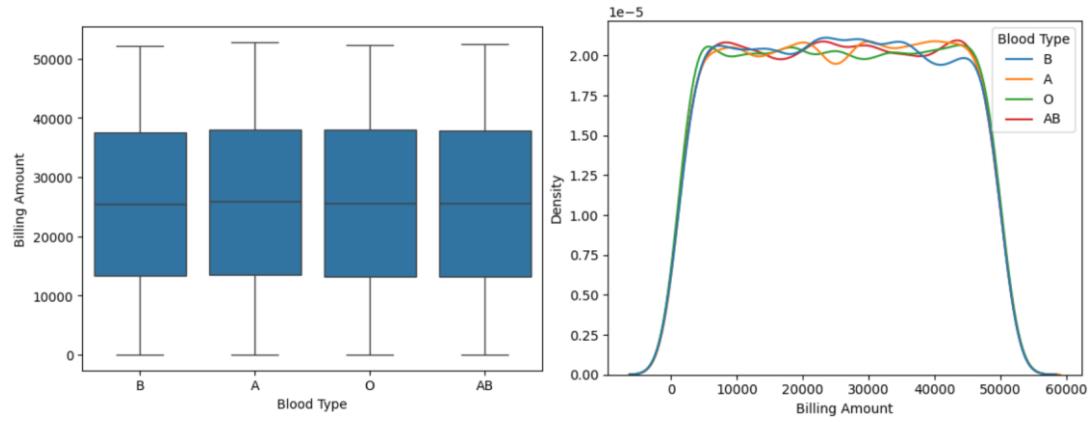


```

1 df3 = df[['Billing Amount','Medication']]
2 df3['Medication'].value_counts(normalize=True)*100
3
4
✓ 0.0s
Medication
Analgesico      75.158868
Lipemiente      14.890237
Antibiotico      9.950895
Name: proportion, dtype: float64

```

BLOOD TYPE:



```

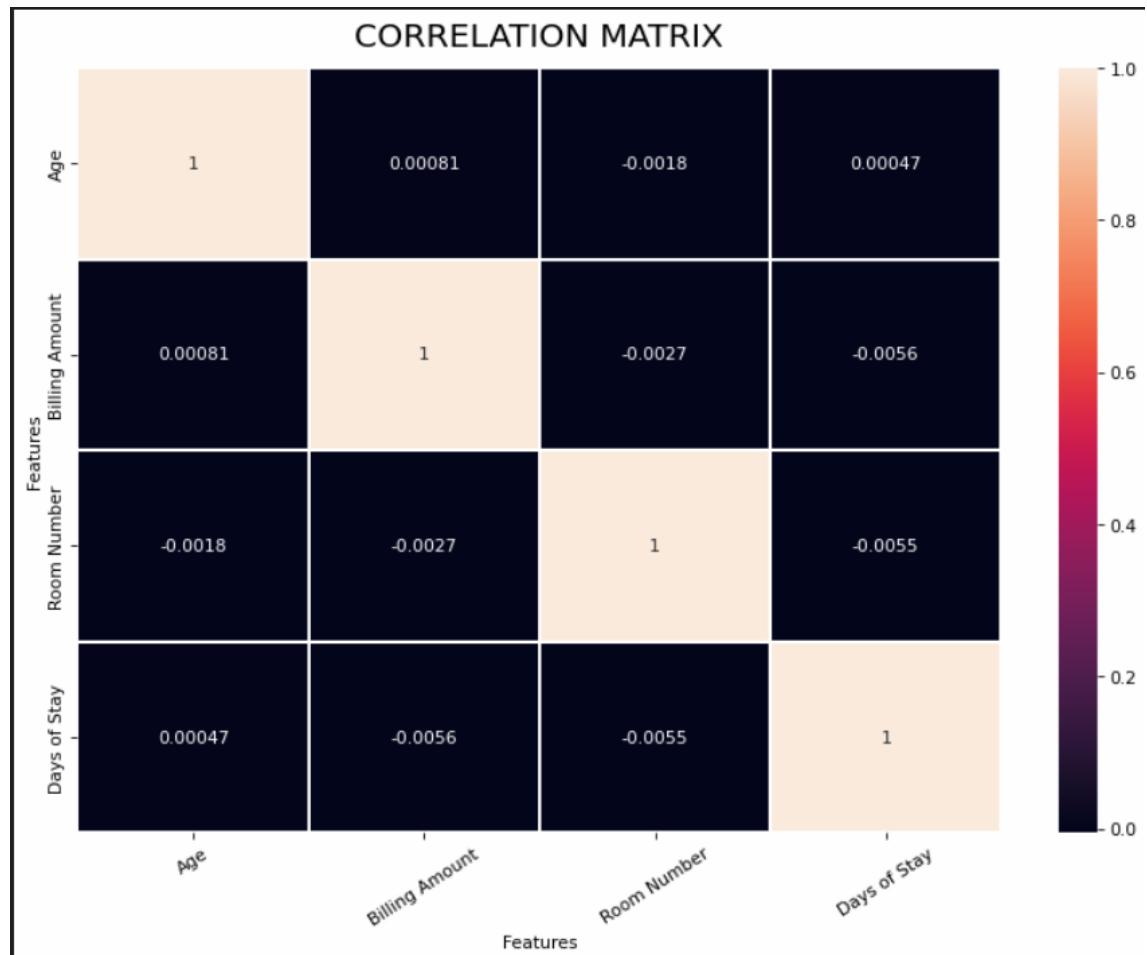
1 df4 = df[['Billing Amount','Blood Type']]
2 df4['Blood Type'].value_counts(normalize=True)*100
✓ 0.0s
Blood Type
A      25.079434
B      25.037912
AB     25.032496
O      24.850159
Name: proportion, dtype: float64

```

5. IDENTIFICACIÓN DE PATRONES Y ANÁLISIS DE VARIABLES

- Correlaciones entre variables cuantitativas.

```
❶ 1 correlation = data.corr(numeric_only=True) # Visualizacion de la matriz de correlación de los datos crudos (data)
❷ 2 plt.figure(figsize=(12,8), dpi=77)
❸ 3 sns.heatmap(correlation, linecolor='white', linewidths=0.1, annot=True)
❹ 4 plt.title('Correlation Matrix'.upper(), size=19, pad=13)
❺ 5 plt.xlabel('Features')
❻ 6 plt.ylabel('Features')
❼ 7 plt.xticks(rotation=33)
❼ 8 plt.show()
✓ 0.4s
```



- Análisis entre variables categóricas y numéricas

```

1 var_cat = ['Gender','Blood Type','Medical Condition','Insurance Provider', 'Admission Type', 'Medication', 'Test Results']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_num['Billing Amount'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el monto de facturación\n')
9     else:
10        print('→ No hay una relación significativa con el monto de facturación\n')
✓ 8.4s

```

```

... Variable: Gender
Chi2: 49890.0545, p-value: 0.5155
→ No hay una relación significativa con el monto de facturación

Variable: Blood Type
Chi2: 166176.0000, p-value: 0.0000
→ Hay una relación significativa con el monto de facturación

Variable: Medical Condition
Chi2: 249708.5500, p-value: 0.3917
→ No hay una relación significativa con el monto de facturación

Variable: Insurance Provider
Chi2: 221568.0000, p-value: 0.0000
→ Hay una relación significativa con el monto de facturación

Variable: Admission Type
Chi2: 99788.6045, p-value: 0.5149
→ No hay una relación significativa con el monto de facturación

Variable: Medication
Chi2: 99802.2130, p-value: 0.5028
→ No hay una relación significativa con el monto de facturación

Variable: Test Results
Chi2: 99775.9394, p-value: 0.5262
→ No hay una relación significativa con el monto de facturación

```

```

1 var_cat = ['Gender','Blood Type','Medical Condition','Insurance Provider', 'Admission Type', 'Medication', 'Test Results']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_num['Age'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el monto de facturación\n')
9     else:
10        print('→ No hay una relación significativa con el monto de facturación\n')
✓ 0.3s

```

```

Variable: Gender
Chi2: 88.5771, p-value: 0.0773
→ No hay una relación significativa con el monto de facturación

Variable: Blood Type
Chi2: 209.3953, p-value: 0.5570
→ No hay una relación significativa con el monto de facturación

Variable: Medical Condition
Chi2: 358.4244, p-value: 0.4392
→ No hay una relación significativa con el monto de facturación

Variable: Insurance Provider
Chi2: 266.9493, p-value: 0.7587
→ No hay una relación significativa con el monto de facturación

Variable: Admission Type
Chi2: 153.8427, p-value: 0.2347
→ No hay una relación significativa con el monto de facturación

Variable: Medication
Chi2: 147.1142, p-value: 0.3672
→ No hay una relación significativa con el monto de facturación

Variable: Test Results
Chi2: 135.9775, p-value: 0.6266
→ No hay una relación significativa con el monto de facturación

```

```

1 var_cat = ['Gender','Blood Type','Medical Condition','Insurance Provider', 'Admission Type', 'Medication', 'Test Results']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_num['Days of Stay'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el monto de facturación\n')
9     else:
10         print('→ No hay una relación significativa con el monto de facturación\n')
✓ 0:1s

```

```
Variable: Gender
Chi2: 18.2779, p-value: 0.9384
→ No hay una relación significativa con el monto de facturación

Variable: Blood Type
Chi2: 143.1436, p-value: 0.0001
→ Hay una relación significativa con el monto de facturación

Variable: Medical Condition
Chi2: 132.1997, p-value: 0.7690
→ No hay una relación significativa con el monto de facturación

Variable: Insurance Provider
Chi2: 151.4309, p-value: 0.0151
→ Hay una relación significativa con el monto de facturación

Variable: Admission Type
Chi2: 64.1110, p-value: 0.2709
→ No hay una relación significativa con el monto de facturación

Variable: Medication
Chi2: 50.8217, p-value: 0.7368
→ No hay una relación significativa con el monto de facturación

Variable: Test Results
Chi2: 63.5007, p-value: 0.2888
→ No hay una relación significativa con el monto de facturación
```

Únicamente se observa una relación entre la aseguradora y el monto de facturación y con los días de estancia

Análisis de variables Categóricas vs Género (Chi2).

Debido a que no hemos encontrado diferencias significativas entre el Billing Amount y las variables categóricas, decidimos estudiar TODAS estas variables con respecto al GÉNERO. **Se descartan Doctor y Hospital puesto que son variables con una elevada variabilidad de métricas.**

```
ENCODING O CATEGORIZACIÓN DEL GÉNERO
```

```
1 df_num2=data[['Gender', 'Age', 'Billing Amount', 'Days of Stay']]
2 df_num2['Gender']=df_num2['Gender'].map({'Male': 0, 'Female': 1})
3 df_num2.head()
✓ 0.0s
```

	Gender	Age	Billing Amount	Days of Stay
0	0	45	18856.281306	2
1	0	38	33643.327287	6
2	0	49	27955.096079	15
3	1	29	37909.782410	30
4	1	62	14238.317814	20

```
var_cat = ['Blood Type', 'Medical Condition', 'Insurance Provider', 'Admission Type', 'Medication', 'Test Re
for var in var_cat:
    contingency_table = pd.crosstab(df_cat['Gender'], df_cat[var])
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    print(f'Variable: {var}')
    print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
    if p < 0.05:
        print('→ Hay una relación significativa con el género\n')
    else:
        print('→ No hay una relación significativa con el género\n')

Variable: Blood Type
Chi2: 9.5383, p-value: 0.0229
→ Hay una relación significativa con el género

Variable: Medical Condition
Chi2: 10.3242, p-value: 0.0666
→ No hay una relación significativa con el género

Variable: Insurance Provider
Chi2: 4.3636, p-value: 0.3590
→ No hay una relación significativa con el género

Variable: Admission Type
Chi2: 4.4566, p-value: 0.1077
→ No hay una relación significativa con el género

Variable: Medication
Chi2: 4.6428, p-value: 0.0981
→ No hay una relación significativa con el género

Variable: Test Results
Chi2: 0.1699, p-value: 0.9186
→ No hay una relación significativa con el género

Variable: Doctor
Chi2: 40345.2403, p-value: 0.4014
→ No hay una relación significativa con el género

Variable: Hospital
Chi2: 39711.1196, p-value: 0.6415
→ No hay una relación significativa con el género
```

A partir del análisis de chi-cuadrado que se ha realizado entre el género y varias variables categóricas, sólo "Blood Type" mostró una relación estadísticamente significativa ($p < 0.05$), lo que sugiere que la distribución de tipos de sangre varía según el género. El resto de las variables analizadas, incluyendo condiciones médicas, proveedores de seguros, tipo de admisión y resultados de pruebas, no mostraron diferencias significativas entre hombres y mujeres. Incluso variables con chi-cuadrados altos como "Doctor" y "Hospital" no fueron significativas, probablemente debido a un gran número de categorías o distribución no uniforme. Esto indica que, en general, el género no influye en la mayoría de las características clínicas o administrativas analizadas en este conjunto de datos.

Análisis de variables Cuantitativas vs Género (t-test)

```

quant_vars = ['Age', 'Billing Amount', 'Days of Stay']
# Dividir en dos grupos por género
grupo1 = df_num2[df_num2['Gender'] == 'Male']
grupo2 = df_num2[df_num2['Gender'] == 'Female']

# Aplicar t-test
for var in quant_vars:
    t_stat, p_val = ttest_ind(grupo1[var], grupo2[var], equal_var=False)
    print(f"Variable: {var}")
    print(f"T-student: {t_stat:.3f}, p-valor: {p_val:.3f}")

    if p_val < 0.05:
        print("→ Hay diferencia estadísticamente significativa entre los dos géneros.\n")
    else:
        print("→ No hay diferencia significativa entre los dos géneros.\n")

Variable: Age
T-student: nan, p-valor: nan
→ No hay diferencia significativa entre los dos géneros.

Variable: Billing Amount
T-student: nan, p-valor: nan
→ No hay diferencia significativa entre los dos géneros.

Variable: Days of Stay
T-student: nan, p-valor: nan
→ No hay diferencia significativa entre los dos géneros.

```

El resultado de aplicar el test para comparar las variables cuantitativa Edad, Monto Facturado, y Días de Estancia entre hombres y mujeres, se obtiene como resultado NaN(no definido), indicando un valor muy pequeño, tanto en el estadístico t como en el p_value, esto nos da a entender que no hay diferencia significativa.

Análisis de variables Categóricas entre ellas (Chi2)

Únicamente se ha observado relación con Doctor y Hospital, pero tienen una gran variabilidad.

Blood Type

```
1 var_cat = ['Medical Condition', 'Insurance Provider', 'Admission Type', 'Medication', 'Test Results', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Blood Type'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')
11
12
13 Variable: Medical Condition
14 Chi2: 11.8924, p-value: 0.6872
15 → No hay una relación significativa con el grupo sanguíneo

16 Variable: Insurance Provider
17 Chi2: 11.9203, p-value: 0.4521
18 → No hay una relación significativa con el grupo sanguíneo

19 Variable: Admission Type
20 Chi2: 4.7180, p-value: 0.5805
21 → No hay una relación significativa con el grupo sanguíneo

22 Variable: Medication
23 Chi2: 6.0400, p-value: 0.4187
24 → No hay una relación significativa con el grupo sanguíneo

25 Variable: Test Results
26 Chi2: 3.5897, p-value: 0.7320
27 → No hay una relación significativa con el grupo sanguíneo

28 Variable: Doctor
29 Chi2: 134735.2902, p-value: 0.0000
30 → Hay una relación significativa con el grupo sanguíneo

31 Variable: Hospital
32 Chi2: 132931.8840, p-value: 0.0000
33 → Hay una relación significativa con el grupo sanguíneo
```

Medical condition

```
1 var_cat = ['Blood Type', 'Insurance Provider', 'Admission Type', 'Medication', 'Test Results', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Medical Condition'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')
```

Variable: Blood Type

Chi2: 11.8924, p-value: 0.6872

→ No hay una relación significativa con el grupo sanguíneo

Variable: Insurance Provider

Chi2: 21.0228, p-value: 0.3958

→ No hay una relación significativa con el grupo sanguíneo

Variable: Admission Type

Chi2: 8.0903, p-value: 0.6200

→ No hay una relación significativa con el grupo sanguíneo

Variable: Medication

Chi2: 2.5169, p-value: 0.9906

→ No hay una relación significativa con el grupo sanguíneo

Variable: Test Results

Chi2: 14.9976, p-value: 0.1321

→ No hay una relación significativa con el grupo sanguíneo

Variable: Doctor

Chi2: 201542.9433, p-value: 0.3953

→ No hay una relación significativa con el grupo sanguíneo

Variable: Hospital

Chi2: 199094.0466, p-value: 0.4844

→ No hay una relación significativa con el grupo sanguíneo

Insurance Provider

```
1 var_cat = ['Blood Type', 'Medical Condition', 'Admission Type', 'Medication', 'Test Results', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Insurance Provider'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')
```

Variable: Blood Type

Chi2: 11.9203, p-value: 0.4521

→ No hay una relación significativa con el grupo sanguíneo

Variable: Medical Condition

Chi2: 21.0228, p-value: 0.3958

→ No hay una relación significativa con el grupo sanguíneo

Variable: Admission Type

Chi2: 7.2199, p-value: 0.5131

→ No hay una relación significativa con el grupo sanguíneo

Variable: Medication

Chi2: 14.4042, p-value: 0.0718

→ No hay una relación significativa con el grupo sanguíneo

Variable: Test Results

Chi2: 8.1507, p-value: 0.4189

→ No hay una relación significativa con el grupo sanguíneo

Variable: Doctor

Chi2: 179802.1662, p-value: 0.0000

→ Hay una relación significativa con el grupo sanguíneo

Variable: Hospital

Chi2: 177013.9258, p-value: 0.0000

→ Hay una relación significativa con el grupo sanguíneo

Admission Type

```
1 var_cat = ['Blood Type', 'Medical Condition', 'Insurance Provider', 'Medication', 'Test Results', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Admission Type'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')
```

```

Variable: Blood Type
Chi2: 4.7180, p-value: 0.5805
→ No hay una relación significativa con el grupo sanguíneo

Variable: Medical Condition
Chi2: 8.0903, p-value: 0.6200
→ No hay una relación significativa con el grupo sanguíneo

Variable: Insurance Provider
Chi2: 7.2199, p-value: 0.5131
→ No hay una relación significativa con el grupo sanguíneo

Variable: Medication
Chi2: 5.7472, p-value: 0.2188
→ No hay una relación significativa con el grupo sanguíneo

Variable: Test Results
Chi2: 7.5405, p-value: 0.1099
→ No hay una relación significativa con el grupo sanguíneo

Variable: Doctor
Chi2: 80774.3235, p-value: 0.2877
→ No hay una relación significativa con el grupo sanguíneo

Variable: Hospital
Chi2: 79514.5349, p-value: 0.6113
→ No hay una relación significativa con el grupo sanguíneo

```

Medication

```

1 var_cat = ['Blood Type', 'Medical Condition', 'Insurance Provider', 'Admission Type', 'Test Results', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Medication'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')

```

```

Variable: Blood Type
Chi2: 6.0400, p-value: 0.4187
→ No hay una relación significativa con el grupo sanguíneo

Variable: Medical Condition
Chi2: 2.5169, p-value: 0.9906
→ No hay una relación significativa con el grupo sanguíneo

Variable: Insurance Provider
Chi2: 14.4042, p-value: 0.0718
→ No hay una relación significativa con el grupo sanguíneo

Variable: Admission Type
Chi2: 5.7472, p-value: 0.2188
→ No hay una relación significativa con el grupo sanguíneo

Variable: Test Results
Chi2: 8.1584, p-value: 0.0859
→ No hay una relación significativa con el grupo sanguíneo

Variable: Doctor
Chi2: 80663.8169, p-value: 0.3878
→ No hay una relación significativa con el grupo sanguíneo

Variable: Hospital
Chi2: 79785.2613, p-value: 0.3462
→ No hay una relación significativa con el grupo sanguíneo

```

Test Results

```

1 var_cat = ['Blood Type', 'Medical Condition', 'Insurance Provider', 'Admission Type', 'Medication', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Test Results'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')

```

```

Variable: Blood Type
Chi2: 3.5897, p-value: 0.7320
→ No hay una relación significativa con el grupo sanguíneo

Variable: Medical Condition
Chi2: 14.9976, p-value: 0.1321
→ No hay una relación significativa con el grupo sanguíneo

Variable: Insurance Provider
Chi2: 8.1507, p-value: 0.4189
→ No hay una relación significativa con el grupo sanguíneo

Variable: Admission Type
Chi2: 7.5405, p-value: 0.1099
→ No hay una relación significativa con el grupo sanguíneo

Variable: Medication
Chi2: 8.1584, p-value: 0.0859
→ No hay una relación significativa con el grupo sanguíneo

Variable: Doctor
Chi2: 80593.9151, p-value: 0.4558
→ No hay una relación significativa con el grupo sanguíneo

Variable: Hospital
Chi2: 79768.2045, p-value: 0.3621
→ No hay una relación significativa con el grupo sanguíneo

```

Blood Type

```

1 var_cat = ['Medical Condition', 'Insurance Provider', 'Admission Type', 'Medication', 'Test Results', 'Doctor', 'Hospital']
2 for var in var_cat:
3     contingency_table = pd.crosstab(df_cat['Blood Type'], df_cat[var])
4     chi2, p, dof, expected = chi2_contingency(contingency_table)
5     print(f'Variable: {var}')
6     print(f'Chi2: {chi2:.4f}, p-value: {p:.4f}')
7     if p < 0.05:
8         print('→ Hay una relación significativa con el grupo sanguíneo\n')
9     else:
10        print('→ No hay una relación significativa con el grupo sanguíneo\n')

```

Variable: Medical Condition
Chi2: 11.8924, p-value: 0.6872
→ No hay una relación significativa con el grupo sanguíneo

Variable: Insurance Provider
Chi2: 11.9203, p-value: 0.4521
→ No hay una relación significativa con el grupo sanguíneo

Variable: Admission Type
Chi2: 4.7180, p-value: 0.5805
→ No hay una relación significativa con el grupo sanguíneo

Variable: Medication
Chi2: 6.0400, p-value: 0.4187
→ No hay una relación significativa con el grupo sanguíneo

Variable: Test Results
Chi2: 3.5897, p-value: 0.7320
→ No hay una relación significativa con el grupo sanguíneo

Variable: Doctor
Chi2: 134735.2902, p-value: 0.0000
→ Hay una relación significativa con el grupo sanguíneo

Variable: Hospital
Chi2: 132931.8840, p-value: 0.0000
→ Hay una relación significativa con el grupo sanguíneo

6. IDENTIFICACIÓN DE INSIGHTS

1.- Cálculo de la duración de la estancia, saber cuánto tiempo han estado hospitalizados. Incluir insurance provider (aseguradora).

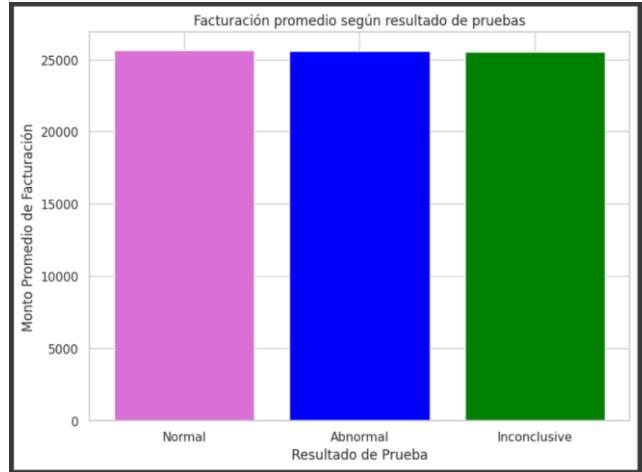
```
df1 = data.groupby('Insurance Provider')['Days of Stay'].mean().reset_index()
print("\nDuración promedio de estancia por aseguradora:")
print(df1)
df1 = df1.sort_values('Days of Stay', ascending=False)
plt.figure(figsize=(8, 6))
plt.bar(df1['Insurance Provider'], df1['Days of Stay'], color= ('mediumseagreen', 'red', 'black', 'grey', 'yellow'))
plt.xlabel('Aseguradora')
plt.ylabel('Duración promedio de estancia (días)')
plt.title('Duración promedio de estancia hospitalaria por aseguradora')
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```

```
Duración promedio de estancia por aseguradora:
Insurance Provider Days of Stay
0      Aetna      15.441576
1    Blue Cross    15.520964
2      Cigna      15.492027
3    Medicare      15.627740
4 UnitedHealthcare  15.460202
```



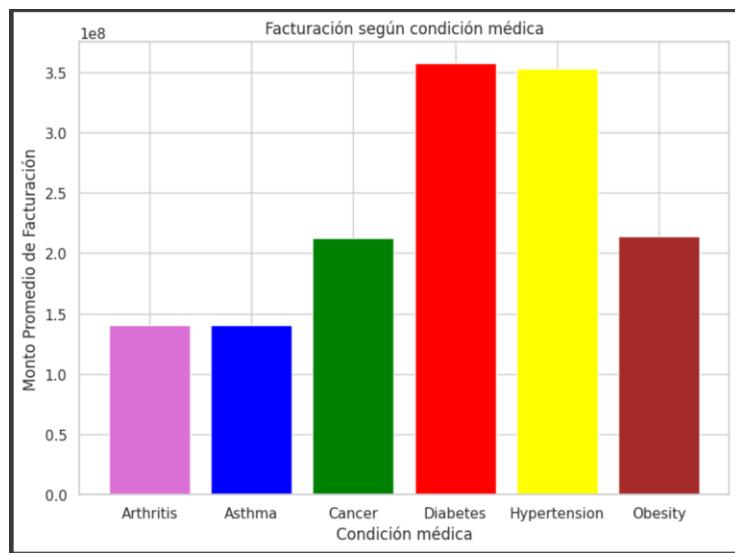
2.- Monto de la facturación vs. el resultado de pruebas

```
df2 = data
# Agrupa por resultado de prueba y calcula la facturación media (o suma)
facturacion_por_resultado = df2.groupby('Test Results')['Billing Amount'].mean().reset_index()
# Ordena por nivel de facturación (opcional)
facturacion_por_resultado = facturacion_por_resultado.sort_values('Billing Amount', ascending=False)
# Gráfica
plt.figure(figsize=(8, 6))
plt.bar(facturacion_por_resultado['Test Results'], facturacion_por_resultado['Billing Amount'], color=['orchid', 'blue', 'green'])
plt.xlabel('Resultado de Prueba')
plt.ylabel('Monto Promedio de Facturación')
plt.title('Facturación promedio según resultado de pruebas')
plt.tight_layout()
plt.show()
```



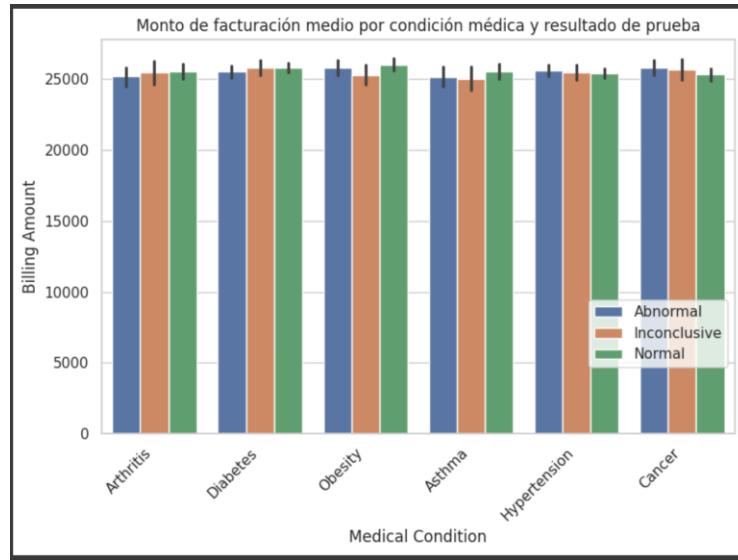
3.- Monto de facturación vs. condición médica/tipo de enfermedad

```
df3=data.groupby('Medical Condition')['Billing Amount'].sum().reset_index()
plt.figure(figsize=(8, 6))
plt.bar(df3['Medical Condition'], df3['Billing Amount'], color=['orchid', 'blue', 'green', 'red', 'yellow', 'brown'])
plt.xlabel('Condición médica')
plt.ylabel('Monto Promedio de Facturación')
plt.title('Facturación según condición médica')
plt.tight_layout()
plt.show()
```



4.- Relación entre los resultados de las pruebas médicas y la condición médica con el monto de facturación.

```
df4=data.groupby(['Medical Condition','Test Results'])['Billing Amount'].mean().reset_index()
plt.figure(figsize=(8, 6))
sns.barplot(x='Medical Condition', y='Billing Amount', hue='Test Results', data=df2)
plt.title('Monto de facturación medio por condición médica y resultado de prueba')
plt.xticks(rotation=45, ha='right')
plt.legend(loc='lower right', bbox_to_anchor=(1, 0.15))
plt.tight_layout()
plt.show()
```



5.- ¿Qué hospital tiene más monto de facturación?

```
# 5.- ¿Qué hospital tiene más monto de facturación?
df5=data
# Agrupamos por hospital y sumamos la facturación
facturacion_por_hospital = df5.groupby('Hospital')[['Billing Amount']].sum().reset_index()
# Encuentramos el hospital con mayor facturación
hospital_top = facturacion_por_hospital.loc[facturacion_por_hospital['Billing Amount'].idxmax()]
print("El hospital con más monto de facturación es:")
print(hospital_top)

El hospital con más monto de facturación es:
Hospital      Johnson PLC
Billing Amount 1084202.692965
Name: 14844, dtype: object
```

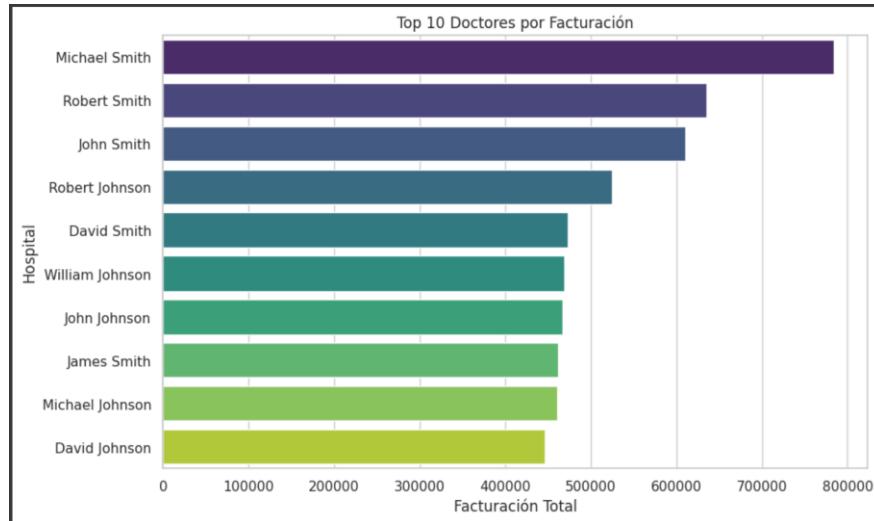
6.- ¿% de pacientes con admisión de emergencia?

```
# 6.- % de pacientes con admisión de emergencia?
df6 = data[data['Admission Type']=='Emergency']
porcentage= len(df6)/len(data)*100
print(f'El % pacientes que ingresan por emergencias es:{porcentage:.1f}')
```

El % pacientes que ingresan por emergencias es:45.2

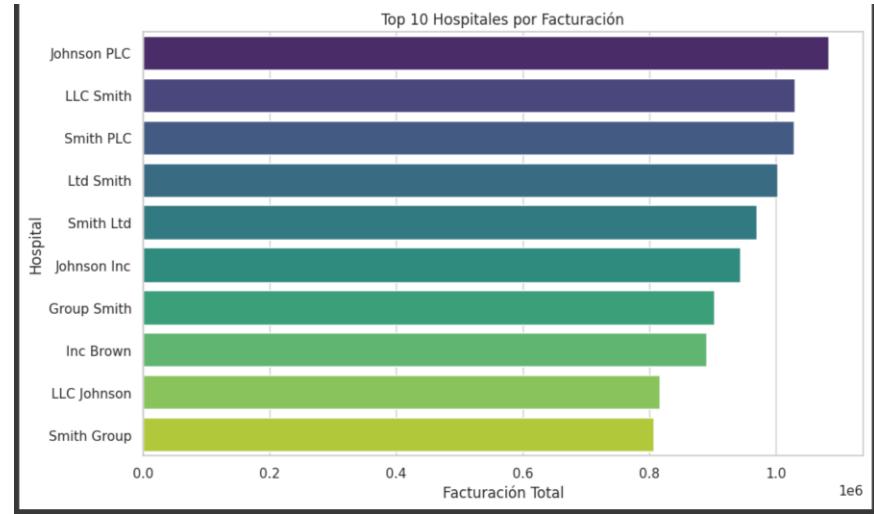
7.- Ranking de doctores con mayor facturación para el hospital.

```
df7 = data.groupby('Doctor')['Billing Amount'].sum().sort_values(ascending=False).reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(df7.head(10), x='Billing Amount', y='Doctor', palette='viridis')
plt.title('Top 10 Doctores por Facturación')
plt.xlabel('Facturación Total')
plt.ylabel('Hospital')
plt.tight_layout()
plt.show()
```



8.- Ranking de hospitales por facturación y doctores que manejan múltiples condiciones

```
df8 = data.groupby('Hospital')['Billing Amount'].sum().sort_values(ascending=False).reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(data=df8.head(10), x='Billing Amount', y='Hospital', palette='viridis')
plt.title('Top 10 Hospitales por Facturación')
plt.xlabel('Facturación Total')
plt.ylabel('Hospital')
plt.tight_layout()
plt.show()
```



9.- ¿Número de estancia en el hospital por género? Número de estancia promedio en el hospital por género?

```
#9y 10 juntos
# --- Datos para el gráfico 1: Número total de estancias ---
df9_1 = data
estancias_por_genero = df9_1['Gender'].value_counts().reset_index()
estancias_por_genero.columns = ['Gender', 'Número de Estancias']

# --- Datos para el gráfico 2: Estancia promedio ---
df9_2 = data.groupby('Gender')['Days of Stay'].mean().reset_index()

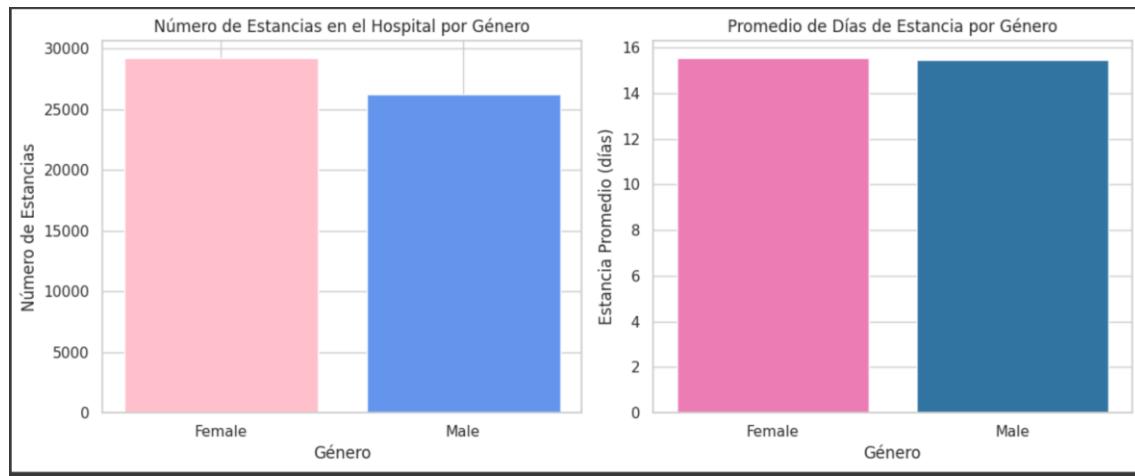
# --- Colores personalizados ---
custom_palette = {'Male': '#1f77b4', 'Female': '#ff69b4'}

# --- Crear figura con 2 subplots horizontales ---
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# --- Subplot 1: Número total de estancias ---
axes[0].bar(estancias_por_genero['Gender'], estancias_por_genero['Número de Estancias'],
            color=['pink', 'cornflowerblue'])
axes[0].set_title('Número de Estancias en el Hospital por Género')
axes[0].set_xlabel('Género')
axes[0].set_ylabel('Número de Estancias')

# --- Subplot 2: Estancia promedio ---
sns.barplot(data=df9_2, x='Gender', y='Days of Stay', palette=custom_palette, ax=axes[1])
axes[1].set_title('Promedio de Días de Estancia por Género')
axes[1].set_xlabel('Género')
axes[1].set_ylabel('Estancia Promedio (días)')

# --- Ajustes finales ---
plt.tight_layout()
plt.show()
```



10.-Incremento de enfermedades en función de la edad.

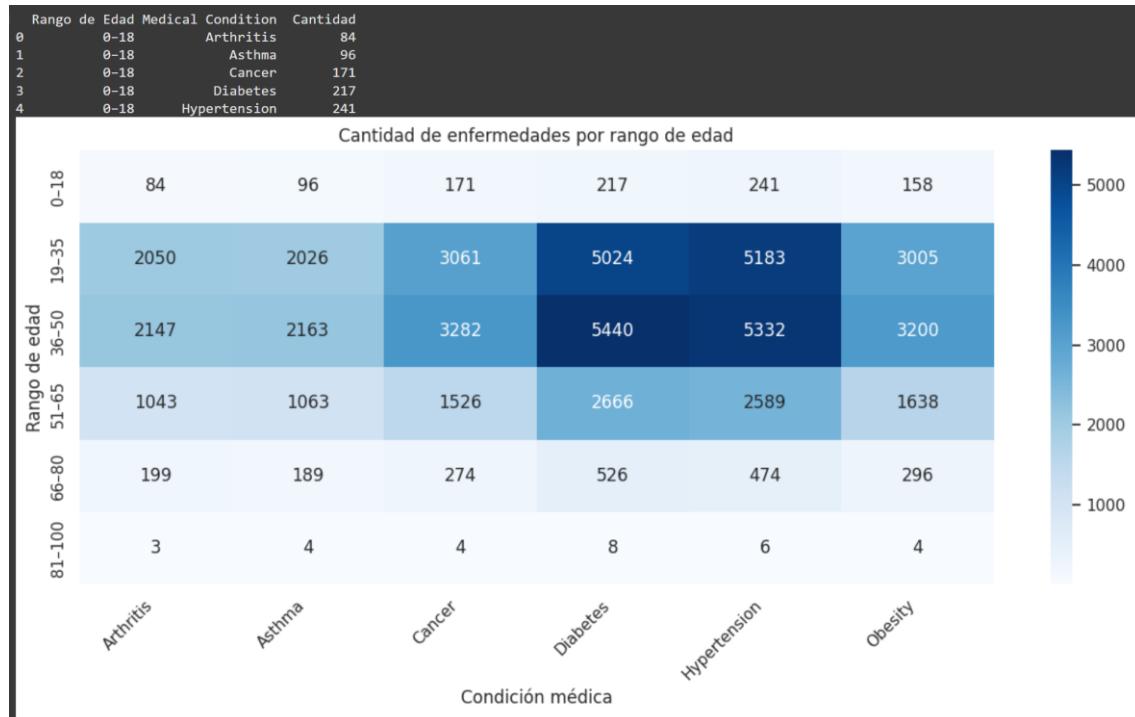
```

bins = [0, 18, 35, 50, 65, 80, 100]# Crear rangos de edad
labels = ['0-18', '19-35', '36-50', '51-65', '66-80', '81-100']
data['Rango de Edad'] = pd.cut(data['Age'], bins=bins, labels=labels, right=False)

# Agrupar por rango de edad y enfermedad, y contar los casos
df10= data.groupby(['Rango de Edad', 'Medical Condition']).size().reset_index(name='Cantidad')
print(df10.head())
# Pivotear la tabla para el heatmap
pivot = df10.pivot(index='Rango de Edad', columns='Medical Condition', values='Cantidad').fillna(0)

plt.figure(figsize=(12, 6))
sns.heatmap(pivot, annot=True, fmt=".0f", cmap='Blues')
plt.title('Cantidad de enfermedades por rango de edad')
plt.xlabel('Condición médica')
plt.ylabel('Rango de edad')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

```



11.- Análisis de enfermedades según su género

```

# 11.- Incremento de enfermedades según su género
df11 = data.copy()
df11['AD_Año'] = pd.to_datetime(df11['Date of Admission']).dt.year

# Agrupar por año, enfermedad y género
enfermedad_genero_año = df11.groupby(['AD_Año', 'Medical Condition', 'Gender']).size().reset_index(name='Casos')

# Lista de enfermedades a graficar
enfermedades = ['Diabetes', 'Cancer', 'Arthritis', 'Obesity', 'Asthma']

# Crear figura con 5x2 subplots
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14, 12))
axes = axes.flatten() # Aplanamos para fácil indexación

# --- GRÁFICOS 1-5: Evolución por enfermedad y género ---
for idx, enfermedad in enumerate(enfermedades):
    df_sub = enfermedad_genero_año[enfermedad_genero_año['Medical Condition'] == enfermedad]
    for genero in df_sub['Gender'].unique():
        datos = df_sub[df_sub['Gender'] == genero]
        axes[idx].plot(datos['AD_Año'], datos['Casos'], marker='o', label=genero)

    axes[idx].set_title(f'{enfermedad} por Género')
    axes[idx].set_xlabel('Año')
    axes[idx].set_ylabel('Casos')
    axes[idx].legend()

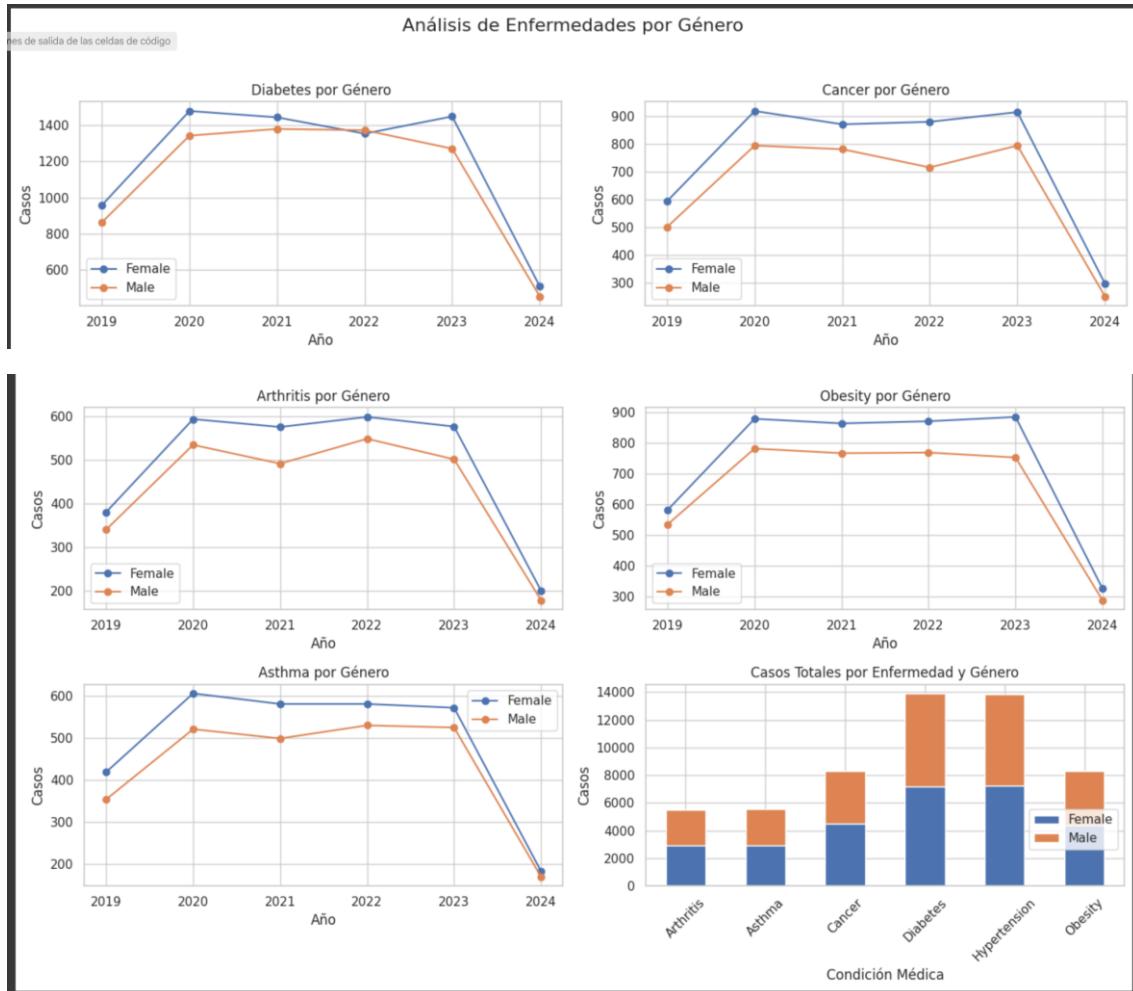
# --- GRÁFICO 6: Barras apiladas global ---
enfermedad_genero = df11.groupby(['Medical Condition', 'Gender']).size().unstack(fill_value=0)

# Usamos el último subplot para el gráfico de barras apiladas
enfermedad_genero.plot(kind='bar', stacked=True, ax=axes[5])
axes[5].set_title('Casos Totales por Enfermedad y Género')
axes[5].set_xlabel('Condición Médica')
axes[5].set_ylabel('Casos')
axes[5].tick_params(axis='x', rotation=45)

# Título general de la figura
plt.suptitle('Análisis de Enfermedades por Género', fontsize=16)
plt.legend(loc='lower right', bbox_to_anchor=(1, 0.15))

# Ajustes de diseño
plt.tight_layout(rect=[0, 0, 1, 0.98])

```



12.- Identificación de recaídas, por cáncer, vs aseguradora

```
# Filtrar solo pacientes con cáncer
df12 = data[data['Medical Condition'].str.contains('cancer', case=False, na=False)].copy()

# Ordenar por paciente y fecha de admisión
df12= df12.sort_values(['Name', 'Date of Admission'])

# Contar el número de ingresos por paciente (para cáncer)
recaidas = df12.groupby('Name').size().reset_index(name='Num_Ingresos')

# Filtrar pacientes con más de un ingreso = recaidas
pacientes_recaidas = recaidas[recaidas['Num_Ingresos'] > 1]

print(f"Número de pacientes con recaídas por cáncer: {len(pacientes_recaidas)}")

# Ahora juntar con aseguradora
recaidas_con_aseguradora = df12[df12['Name'].isin(pacientes_recaidas['Name'])]

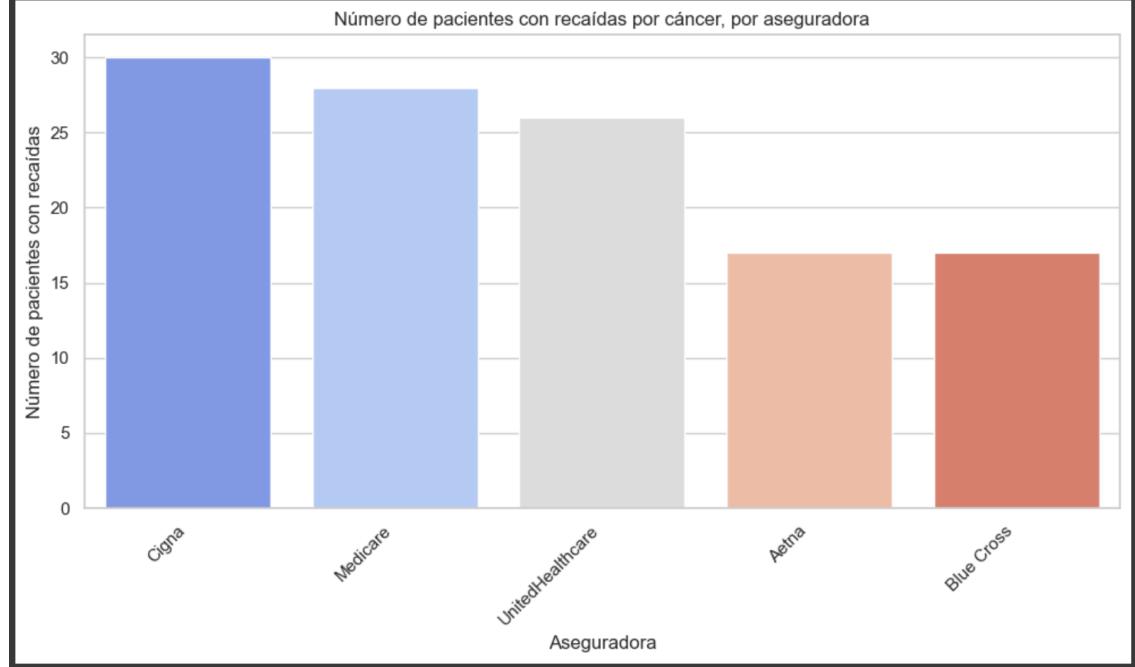
# Contar recaídas por aseguradora (número de pacientes únicos con recaídas)
recaidas_por_aseguradora = recaidas_con_aseguradora.groupby('Insurance Provider')[['Name']].nunique().reset_index(name='Pacientes con recaídas')

print(recaidas_por_aseguradora)

plt.figure(figsize=(10,6))
sns.barplot(data=recaidas_por_aseguradora.sort_values('Pacientes con recaídas', ascending=False),
            x='Insurance Provider', y='Pacientes con recaídas', palette='coolwarm')

plt.title('Número de pacientes con recaídas por cáncer, por aseguradora')
plt.xlabel('Aseguradora')
plt.ylabel('Número de pacientes con recaídas')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
Número de pacientes con recaídas por cáncer: 117
    Insurance Provider  Pacientes con recaídas
0           Aetna           17
1       Blue Cross          17
2         Cigna            30
3      Medicare           28
4  UnitedHealthcare        26
```



13.- Porcentaje de doctores por aseguradora

```
df13 = data
conteo = df13.groupby(['Doctor', 'Insurance Provider']).size().reset_index(name='Num_Analisis')
conteo['Total_por_Doctor'] = conteo.groupby('Doctor')['Num_Analisis'].transform('sum')
conteo['Porc_por_Doctor'] = (conteo['Num_Analisis'] / conteo['Total_por_Doctor']) * 100
pivot = conteo.pivot(index='Doctor', columns='Insurance Provider', values='Porc_por_Doctor').fillna(0)
print("Tanto por ciento de análisis por doctor y aseguradora:")
print(pivot)
```

```
Tanto por ciento de análisis por doctor y aseguradora:
Insurance Provider  Aetna  Blue Cross  Cigna  Medicare  UnitedHealthcare
Doctor
Aaron Acevedo      0.0      0.0      0.0    100.0      0.0
Aaron Adams        0.0      0.0      0.0    100.0      0.0
Aaron Aguilar      0.0    100.0      0.0      0.0      0.0
Aaron Alexander    0.0      0.0    100.0      0.0      0.0
Aaron Anderson     0.0      0.0      0.0    100.0      0.0
...
Zoe Khan           0.0      0.0      0.0      0.0    100.0
Zoe Knight         0.0    100.0      0.0      0.0      0.0
Zoe Nichols        0.0      0.0      0.0    100.0      0.0
Zoe Roberts        0.0      0.0      0.0      0.0    100.0
Zoe Wallace        0.0    100.0      0.0      0.0      0.0

[40276 rows x 5 columns]
```

14.- ¿Qué aseguradora tiene más facturación total?

```
# Agrupamos por aseguradora y suma la facturación total de cada una de ellas
df14 = data
facturacion_por_aseguradora = df14.groupby('Insurance Provider')['Billing Amount'].sum().round(2).reset_index()
# Encuentramos la aseguradora con mayor facturación
aseguradora_top = facturacion_por_aseguradora.loc[facturacion_por_aseguradora['Billing Amount'].idxmax()]
print(f"La aseguradora con mayor facturación total es: \n{aseguradora_top}")

La aseguradora con mayor facturación total es:
Insurance Provider      Cigna
Billing Amount          287151707.24
Name: 2, dtype: object
```

ANÁLISIS DE COHORTES

15.- Las hospitalizaciones en pacientes mayores de 65 años con más de dos enfermedades crónicas."

```
df15= data
df15['Num_cronicas'] = df15['Medical Condition'].apply(lambda x: len(str(x).split(',')))
# Filtramos hospitalizaciones en pacientes mayores de 65 años y más de 2 enfermedades crónicas
condicion = (df15['Age'] > 65)
hospitalizaciones_mayores2cronicas = df15[condicion]
# Número total de hospitalizaciones
total_hospitalizaciones = len(df15)
# Número de hospitalizaciones en pacientes mayores de 65 con más de 2 crónicas
num_concentradas = len(hospitalizaciones_mayores2cronicas)
# Porcentaje
porcentaje = (num_concentradas / total_hospitalizaciones) * 100
print(f"Total de hospitalizaciones: {total_hospitalizaciones}")
print(f"Hospitalizaciones en mayores de 65 años : {num_concentradas}")
print(f"Porcentaje: {porcentaje:.2f}%")

Total de hospitalizaciones: 55392
Hospitalizaciones en mayores de 65 años : 1715
Porcentaje: 3.10%
```

16 .- Las hospitalizaciones en pacientes mayores de 65 años con más de dos enfermedades crónicas vs al resto.

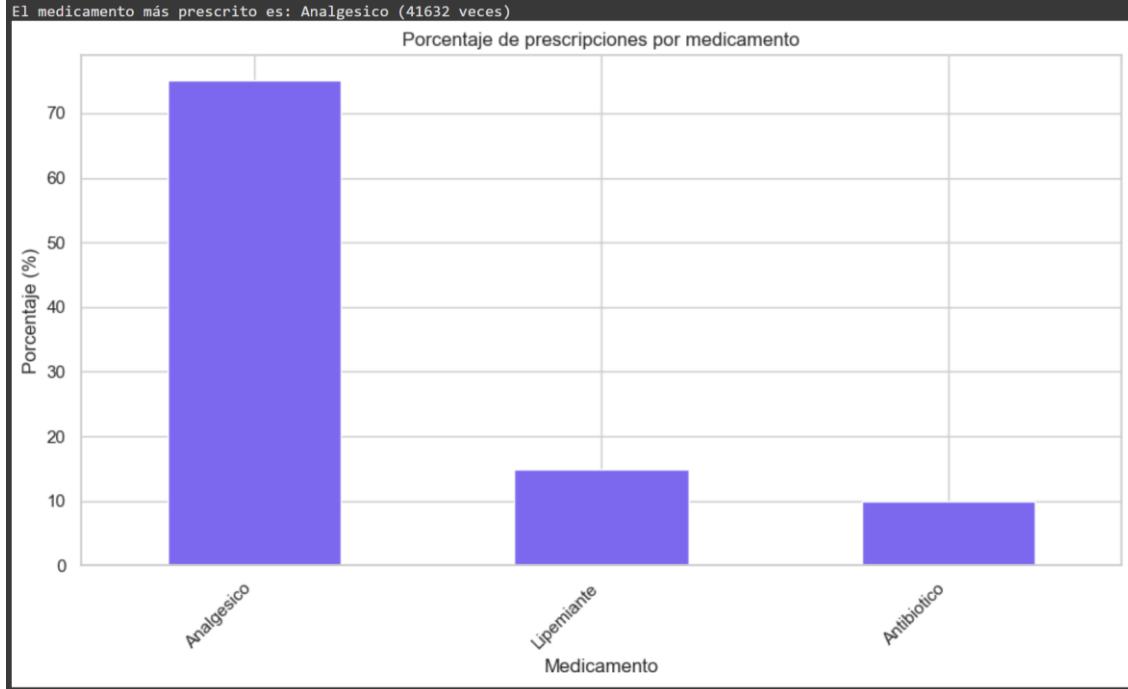
```
df16= data
df16["Num_Cronicas"] = df[["Medical Condition"]].apply(lambda x: len(str(x).split(",")))

# Definir cohortes
cohorte_mayores_cronicos = df16[(df16["Age"] > 65) & (df16["Num_Cronicas"] > 2)]
cohorte_restante = df16[(df16["Age"] > 65) & (df16["Num_Cronicas"] > 2)]
```

Hospitalizaciones cohortes mayores de 65 con más de 2 crónicas: 6
Hospitalizaciones restantes: 55392

17.- Qué medicamento se prescribe más?

```
df16 = data
medicamento_mas_prescrito = df16['Medication'].value_counts().idxmax()# Calculamos el medicamento más prescrito
cantidad_prescripciones = df16['Medication'].value_counts().max()
print(f"El medicamento más prescrito es: {medicamento_mas_prescrito} ({cantidad_prescripciones} veces)")
# Calculamos el porcentaje de cada medicación
porcentaje_meds = df16['Medication'].value_counts(normalize=True) * 100
# Graficamos el porcentaje como gráfico de barras
plt.figure(figsize=(10,6))
porcentaje_meds.plot(kind='bar', color= 'mediumslateblue')
plt.ylabel('Porcentaje (%)')
plt.xlabel('Medicamento')
plt.title('Porcentaje de prescripciones por medicamento')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



18.- Las hospitalizaciones en pacientes mayores de 65 años en función del número de ingresos

```

df_agg = df16.groupby('Name').agg({
    'Name': 'count',
    'Age': 'first',
}).rename(columns={'Name': 'num_ingeresos'}).reset_index()

df_menor65 = df_agg[df_agg['Age'] < 65]
df_mayor65 = df_agg[df_agg['Age'] >= 65]

def porcentaje_sobre_ingeresos(df):
    conteo = df['num_ingeresos'].value_counts().sort_index()
    total_ingeresos = (conteo.index * conteo).sum()
    porcentajes = (conteo.index * conteo) / total_ingeresos * 100
    return porcentajes

porc_ingeresos_menor65 = porcentaje_sobre_ingeresos(df_menor65)
porc_ingeresos_mayor65 = porcentaje_sobre_ingeresos(df_mayor65)

fig, axes = plt.subplots(2, 2, figsize=(14,10))

# Gráfico barras + markers + labels para menores de 65
ax = axes[0,0]
x = np.arange(len(porc_ingeresos_menor65))
bars = ax.bar(x, porc_ingeresos_menor65.values, color='mediumseagreen')
ax.plot(x, porc_ingeresos_menor65.values, 'o', color='black', markersize=8)

for i, v in enumerate(porc_ingeresos_menor65.values):
    ax.text(x[i], v + 0.5, f'{v:.3f}%', ha='center', va='bottom', fontsize=9)

ax.set_xticks(x)
ax.set_xticklabels(porc_ingeresos_menor65.index)
ax.set_title('Distribución ingresos (% sobre total ingresos) <65 años')
ax.set_xlabel('Número de ingresos')
ax.set_ylabel('Porcentaje (%)')

# Gráfico barras + markers + labels para mayores de 65
ax = axes[0,1]
x = np.arange(len(porc_ingeresos_mayor65))
bars = ax.bar(x, porc_ingeresos_mayor65.values, color='coral')
ax.plot(x, porc_ingeresos_mayor65.values, 'o', color='black', markersize=8)

for i, v in enumerate(porc_ingeresos_mayor65.values):
    ax.text(x[i], v + 0.5, f'{v:.3f}%', ha='center', va='bottom', fontsize=9)

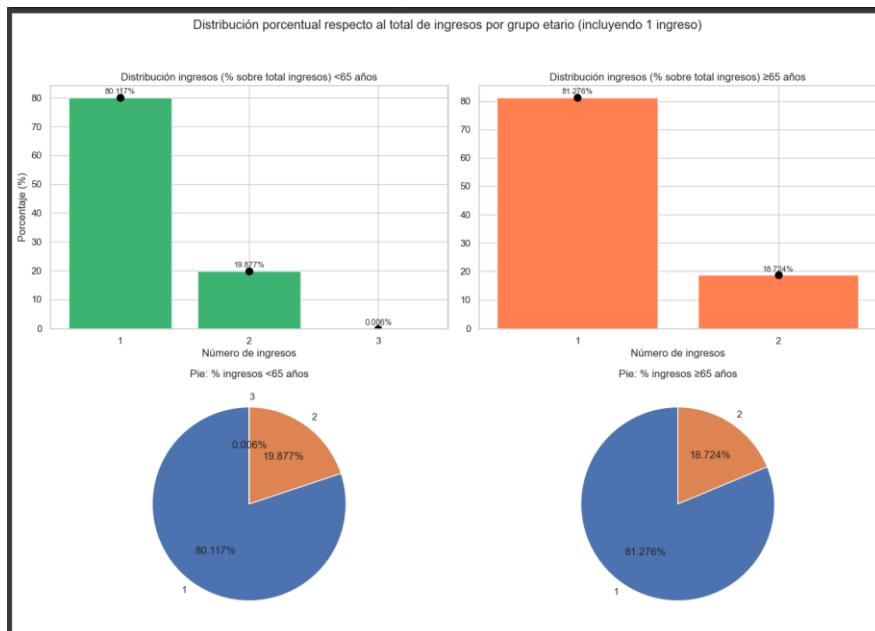
ax.set_xticks(x)
ax.set_xticklabels(porc_ingeresos_mayor65.index)
ax.set_title('Distribución ingresos (% sobre total ingresos) ≥65 años')
ax.set_xlabel('Número de ingresos')

# Pie charts
porc_ingeresos_menor65.plot(kind='pie', autopct='%1.3f%%', startangle=90, ax=axes[1,0], legend=False)
axes[1,0].set_ylabel('')
axes[1,0].set_title('Pie: % ingresos <65 años')

porc_ingeresos_mayor65.plot(kind='pie', autopct='%1.3f%%', startangle=90, ax=axes[1,1], legend=False)
axes[1,1].set_ylabel('')
axes[1,1].set_title('Pie: % ingresos ≥65 años')

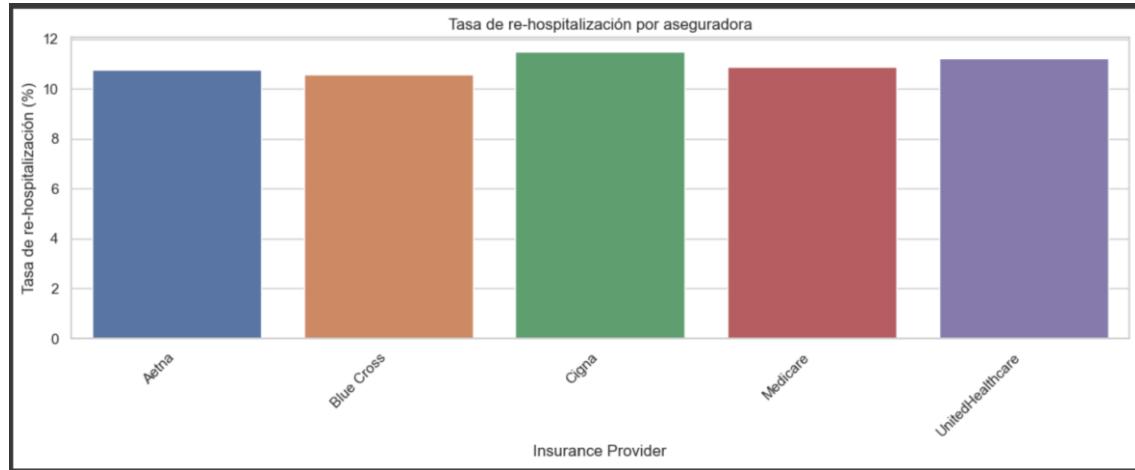
plt.suptitle('Distribución porcentual respecto al total de ingresos por grupo etario (incluyendo 1 ingreso)')
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

```



19.- Agrupar por paciente y aseguradora para contar hospitalizaciones únicas:

```
df18=data
rehospitalizaciones = df18.groupby(['Name', 'Insurance Provider']).size().reset_index(name='Num_Hospitalizaciones')
# Identifica para cada aseguradora cuántos pacientes tuvieron más de una hospitalización
rehosp_x_aseg = rehospitalizaciones.groupby('Insurance Provider').apply(lambda x: (x['Num_Hospitalizaciones'] > 1).sum()).reset_index(name='Pacientes_rehospitalizados')
# Para comparar también cuántos pacientes únicos tiene cada aseguradora:
total_pacientes = rehospitalizaciones.groupby('Insurance Provider')[['Name']].nunique().reset_index(name='Pacientes_totales')
# Fusionar para calcular tasa de rehospitalización
df_plot = pd.merge(rehosp_x_aseg, total_pacientes, on='Insurance Provider')
df_plot['Tasa_rehospitalización'] = df_plot['Pacientes_rehospitalizados'] / df_plot['Pacientes_totales'] * 100
#
#Gráfico de barras: frecuencia absoluta y tasa (%)
plt.figure(figsize=(12,5))
sns.barplot(x='Insurance Provider', y='Tasa_rehospitalización', data=df_plot, palette='deep')
plt.ylabel('Tasa de re-hospitalización (%)')
plt.title('Tasa de re-hospitalización por aseguradora')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



20.- Reincidencia de hospitalización por aseguradora

```
# 19.- Reincidencia de hospitalización por aseguradora
df19= data
rehospitalizaciones = df19.groupby(['Name', 'Insurance Provider']).size().reset_index(name='Num_Hospitalizaciones')
#Ver cuántos pacientes tienen más de una hospitalización por aseguradora
rehosp_x_aseg = rehospitalizaciones.groupby('Insurance Provider').apply(lambda x: (x['Num_Hospitalizaciones'] > 1).sum())
print("Pacientes con rehospitalización por aseguradora:")
print(rehosp_x_aseg)

#
Pacientes con rehospitalización por aseguradora:
Insurance Provider
Aetna          1060
Blue Cross     1056
Cigna          1156
Medicare       1094
UnitedHealthcare 1122
dtype: int64
```

7. PREDICCIÓN DE MODELOS

Tras realizar los primeros modelos de regresión y Random Forest, observamos la necesidad de balancear el conjunto de datos para evitar sesgos en el aprendizaje. Esto se debe a que los datos sintéticos presentaban un sesgo significativo en la mayoría de las variables.

```
1 df = pd.read_csv('mi_data.csv')# Cargar el archivo original
2 print("Distribución original:") # Verificar la distribución original (opcional)
3 print(df[['Test Results', 'Gender','Admission Type']].value_counts())
4 # Encontrar la cantidad mínima de muestras entre las clases
5 min_count = df[['Test Results', 'Gender','Admission Type']].value_counts().min()
6 # Realizar el undersampling para balancear
7 df_balanced = df.groupby(['Test Results', 'Gender','Admission Type']).sample(n=min_count, random_state=42)
8 # Verificar la nueva distribución (opcional)
9 print("\nDistribución balanceada:")
0 print(df_balanced[['Test Results', 'Gender','Admission Type']].value_counts())
1 # Guardar el nuevo dataset en un archivo CSV
2 df_balanced.to_csv('mi_data_balanceado.csv', index=False)
3
4 print("\nArchivo guardado como 'mi_data_balanceado.csv'")
```

```
● 1 df_balanced.info()
2 #18396 filas

<class 'pandas.core.frame.DataFrame'>
Index: 18396 entries, 13991 to 36889
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name              18396 non-null   object 
 1   Age               18396 non-null   int64  
 2   Gender            18396 non-null   object 
 3   Blood Type        18396 non-null   object 
 4   Medical Condition 18396 non-null   object 
 5   Date of Admission 18396 non-null   object 
 6   Doctor            18396 non-null   object 
 7   Hospital           18396 non-null   object 
 8   Insurance Provider 18396 non-null   object 
 9   Billing Amount     18396 non-null   float64
 10  Room Number        18396 non-null   int64  
 11  Admission Type     18396 non-null   object 
 12  Discharge Date     18396 non-null   object 
 13  Medication          18396 non-null   object 
 14  Test Results        18396 non-null   object 
 15  Days of Stay        18396 non-null   int64  
 16  AD_Mes              18396 non-null   int64  
 17  AD_DiaSemana        18396 non-null   object 
 18  AD_Trimestre        18396 non-null   int64  
 19  AD_Año              18396 non-null   int64  
 ...
 22  DD_Trimestre        18396 non-null   int64  
 23  DD_Año              18396 non-null   int64  
dtypes: float64(1), int64(9), object(14)
memory usage: 3.5+ MB
```

Todos los modelos siguientes se hicieron con este dataset mi_data_balanceado

TÉCNICAS DE APRENDIZAJE SUPERVISADO

En este punto se va a evaluar la capacidad de generalización de los modelos, para ello, se entrenaron distintos algoritmos de aprendizaje supervisado, en concreto se ha utilizado regresión lineal, regresión de árbol (Decisión Tree), Random Forest y gradient Boosting (XGBoosting), midiendo el rendimiento de cada modelo mediante las siguientes métricas error cuadrático medio (RMSE), y el coeficiente de determinación (R^2), eligiendo el modelo con mejor capacidad predictiva sobre los datos de prueba.

Tipo de Aprendizaje	Modelo	Descripción breve	Tarea principal
SUPERVISADO	Regresión Lineal	Modelo simple para predecir valores continuos	Regresión
	Regresión Logística	Clasificación binaria (sí/no, 0/1)	Clasificación
	Árboles de Decisión	Divide los datos según reglas en forma de árbol	Clasificación / Regresión
	Random Forest	Conjunto de árboles para mejorar precisión y reducir sobreajuste	Clasificación / Regresión
	Gradient Boosting (XGBoost, etc)	Modelo por ensamble que mejora errores gradualmente	Clasificación / Regresión
	Redes Neuronales	Modelos flexibles y potentes para tareas complejas	Clasificación / Regresión

Modelado Predictivo de una Variable Numérica

REGRESIÓN LINEAL

```
X = df[['Age', 'Days of Stay', 'Admission Type', 'Gender', 'Insurance Provider', 'Medication']]  
X = pd.get_dummies(X) # Codificación one-hot para variables categóricas  
y = df['Billing Amount']  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
model = LinearRegression()  
model.fit(X_train, y_train)  
  
y_pred = model.predict(X_test)  
  
# Evaluate the model  
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))  
print("R2 Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 194410751.12432227  
R2 Score: -0.0008445455789141132
```

El MSE obtenido es de casi 194 millones es debido a la escala puesto que la variable es monto de facturación de las aseguradoras en los Estados Unidos. Aunque el MSE no es excesivamente alto comparado con la escala de la variable objetivos. La R^2 de -0.00084 indicando que el modelo no mejora nada respecto a simplemente predecir el valor promedio de la variable objetivo. De hecho, predice ligeramente peor.

Comparado con Random Forest, el Random Forest obtenido tiene R^2 de -0.21, peor que la regresión lineal. Eso sugiere que ninguno de los modelos está funcionando bien, y el problema está en los datos (no en el algoritmo).

```
X = df[['Age', 'Days of Stay', 'Admission Type', 'Gender', 'Insurance Provider', 'Medication']]  
X = pd.get_dummies(X) # Codificación one-hot para variables categóricas  
y = df['Billing Amount']  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
# Crear y entrenar el modelo Random Forest  
rf_rg = RandomForestRegressor(n_estimators=150, max_depth=26, random_state=42)  
rf_rg.fit(X_train, y_train)  
  
# Predecir en el conjunto de prueba  
y_pred = rf_rg.predict(X_test)  
# Mètriques de regressió  
mae = mean_absolute_error(y_test, y_pred)  
mse = mean_squared_error(y_test, y_pred)  
rmse = np.sqrt(mse)  
r2 = r2_score(y_test, y_pred)  
  
# Mostrar resultats  
print("MAE (Error absolut mitjà):", round(mae, 2))  
print("MSE (Error quadràtic mitjà):", round(mse, 2))  
print("RMSE (Arrel quadrada del MSE):", round(rmse, 2))  
print("R2 (Coeficient de determinació):", round(r2, 2))
```

ALGORITMO RANDOMFOREST

```
X = df[['Age','Days of Stay','Admission Type','Gender', 'Insurance Provider', 'Medication' ]]
X = pd.get_dummies(X) # Codificación one-hot para variables categóricas
y = df['Billing Amount']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear y entrenar el modelo Random Forest
rf_rg = RandomForestRegressor(n_estimators=150, max_depth=26, random_state=42)
rf_rg.fit(X_train, y_train)

# Predecir en el conjunto de prueba
y_pred = rf_rg.predict(X_test)
# Métricas de regresión
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Mostrar resultados
print("MAE (Error absolut mitjà):", round(mae, 2))
print("MSE (Error quadràtic mitjà):", round(mse, 2))
print("RMSE (Arrel quadrada del MSE):", round(rmse, 2))
print("R² (Coeficient de determinació):", round(r2, 2))
```

```
MAE (Error absolut mitjà): 12903.07
MSE (Error quadràtic mitjà): 234122724.8
RMSE (Arrel quadrada del MSE): 15301.07
R² (Coeficient de determinació): -0.21
```

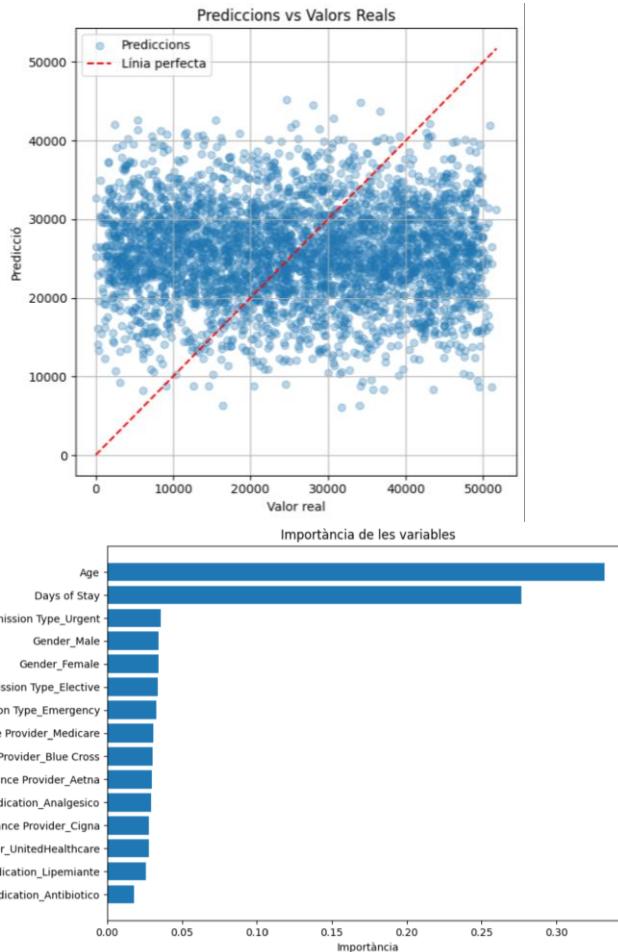
El valor de R^2 negativo indica que el modelo está prediciendo peor que una línea base constante, es decir, peor que simplemente predecir el valor medio de la variable objetivo para todos los casos.

Aunque el MAE y RMSE no son catastróficos por sí solos, el R^2 negativo sugiere que el modelo no va a capturar la relación entre las variables.

```
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred, alpha=0.3, label="Prediccions")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label="Línia perfecta")
plt.xlabel("Valor real")
plt.ylabel("Predicció")
plt.title("Prediccions vs Valors Reals")
plt.legend() # Afegim llegenda
plt.grid(True)
plt.tight_layout()
plt.show()

importances = rf_rg.feature_importances_
features = X.columns
sorted_idx = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.barh(features[sorted_idx], importances[sorted_idx])
plt.xlabel("Importància")
plt.title("Importància de les variables")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



```

# Ara pots usar RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score

X = df[['Age', 'Days of Stay', 'Admission Type', 'Gender', 'Insurance Provider', 'Medication']]
X = pd.get_dummies(X)
y = df['Billing Class']

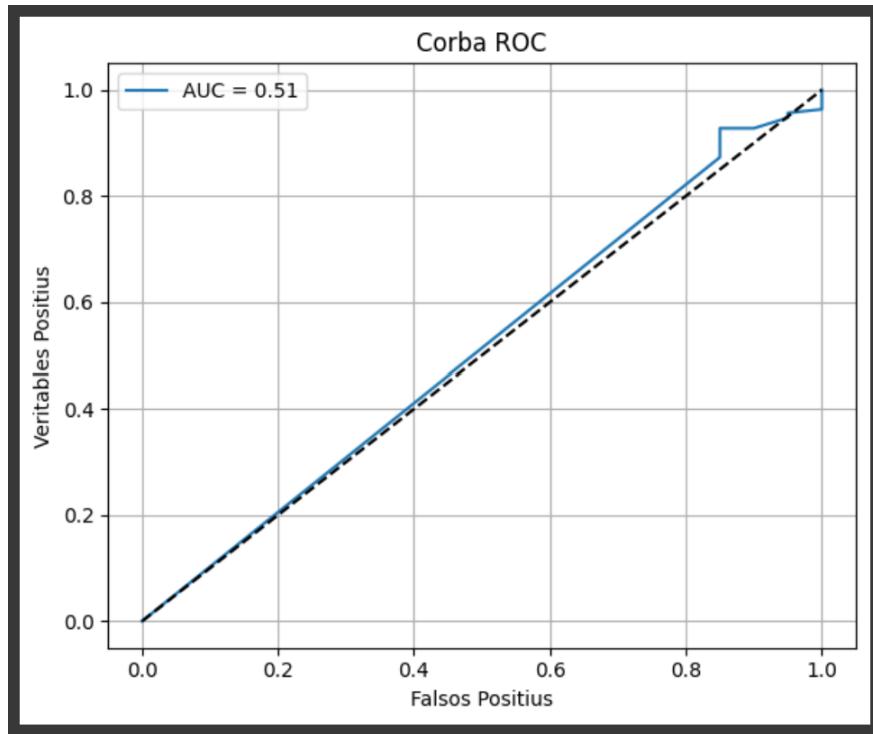
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Corba ROC
y_prob = clf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

# Plot
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("Falsos Positius")
plt.ylabel("Veritables Positius")
plt.title("Corba ROC")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```



NEURAL NETWORK (RED NEURONAL)

```
df=pd.read_csv('mi_data_balanceado.csv')
X = df[['Age','Days of Stay','Admission Type','Gender', 'Insurance Provider', 'Medication' ]]
X = pd.get_dummies(X) # Codificación one-hot para variables categóricas
y = df['Billing Amount']

X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.2,random_state=777)
scaler = StandardScaler()
scaler.fit(X_train)
X_train , X_test = scaler.transform(X_train),scaler.transform(X_test)
rna = MLPRegressor(
    solver='adam',
    alpha=0.001,
    hidden_layer_sizes=(1000 , )
)
modelo = rna.fit(X_train , y_train)
pred_rna = modelo.predict(X_test)
print(f"R2: {r2_score(y_test , pred_rna)}")
print(f"MAE: {mean_absolute_error(y_test , pred_rna)}")
for i , (w,b) in enumerate(zip(rna.coefs_ , rna.intercepts_)):
    print(f"Capa{i} -> capa{i+1}")
    print(f"Peso : {w.shape}")
    print(f"Bias : {b.shape}")
```

```
R2: -0.005093339458617452
MAE: 12211.184434815937
Capa0 -> capa1
Peso : (15, 1000)
Bias : (1000,)
Capa1 -> capa2
```

El modelo de red neuronal mostró un rendimiento muy limitado en la tarea de predicción del monto de facturación, con un coeficiente de determinación $R^2 = -0.0051$, lo que indica que el modelo no logró captar una relación útil entre las variables predictoras y la variable objetivo, rindiendo apenas por debajo de una predicción basada en la media. A pesar de un error absoluto medio (MAE) de aproximadamente 12,211 dólares, que podría considerarse aceptable dependiendo del rango de facturación que tenemos, la falta de capacidad explicativa evidencia que el modelo no generaliza bien.

GRADIENT BOOSTING (XGBOOST)

```
X = df[['Age', 'Days of Stay','Admission Type','Gender', 'Insurance Provider', 'Medication']] # Eliminar la columna objectiu de les predictors
X = pd.get_dummies(X) # Codificació one-hot para variables categòriques
y = df['Billing Amount'] # # Definir la variable objectiu;Corregido a una serie (no lista)

X_train, X_test, y_train, y_test = train_test_split(X , y , random_state=777)
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('xgb', XGBRegressor( 'reg:squarederror' , random_state = 444))
])
param_grid = {
    'xgb_n_estimators':[100,200],
    'xgb_max_depth':[3,5],
    'xgb_learning_rate':[0.01 , 0.1],
    'xgb_subsample':[0.8,1],
    'xgb_colsample_bytree':[0.8,1]
}
cv = KFold(n_splits=5 , shuffle=True , random_state=444)
grid = GridSearchCV(pipeline , param_grid , cv = cv , scoring='neg_mean_squared_error')
grid.fit(X_train,y_train)
print(f"Mejor parametro :{grid.best_params_}")
y_pred = grid.predict(X_test)
print(f"MAE : {mean_absolute_error(y_test , y_pred)}")
print(f"MSE : {mean_squared_error(y_test , y_pred)}")
print(f"RMSE : {root_mean_squared_error(y_test , y_pred)}")
print(f"R2 : {r2_score(y_test , y_pred)})")
```

```
Mejor parametro :{'xgb_colsample_bytree': 0.8, 'xgb_learning_rate': 0.01, 'xgb_max_depth': 3, 'xgb_n_estimators': 100, 'xgb_subsample': 0.8}
MAE : 12227.222538237505
MSE : 199517821.1795032
RMSE : 14125.877740653436
R2 : -0.0013820993887629918
```

```
{
    'xgb_colsample_bytree': 0.8,
    'xgb_learning_rate': 0.01,
    'xgb_max_depth': 3,
    'xgb_n_estimators': 100,
    'xgb_subsample': 0.8
}
```

Los hiperparametros indican un modelo suavemente regularizado, de aprendizaje lento y arboles poco profundos con xg_max_depth =3 con un submuestreo subsample = 0.8 y colsample_bytree = 0.8 para evitar overfitting.

La R² sigue siendo negativa con un valor -0.00138, al igual que en los otros algoritmos el modelo no está capturando una relación útil entre la variable objeto y las variables independientes, el modelo predice peor que simplemente usar la media.

MAE ≈ \$12,227 y RMSE de \$14,125 que son aceptable puesto que los montos de facturación están en el orden de millones, ese error puede ser aceptable en magnitud absoluta, pero como ya se ha comentado R² sugiere que el modelo no está explicando variabilidad, solo predice un valor promedio sin ajustar a los datos.

TÉCNICA DE MACHINE LEARNING ENSEMBLE.

El siguiente paso que se decide relativo a los resultados obtenidos es realizar un ensamblado (ENSEMBLE), técnica de machine learning donde se combinan varios modelos para mejorar el rendimiento predictivo respecto a cualquier modelo individual. Buscamos mejorar la predicción, reducir el sesgo mejorando las estimaciones, etc.

En nuestro caso, se combina la Regresión lineal, la red neuronal y XGBoosting, se descarta el Random forest por tener R2 más alta, utilizando Averaging para regresión puesto que nuestra variable es numérica.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold, GridSearchCV, cross_val_score

from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import VotingRegressor

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')

df=pd.read_csv('mi_data_balanceado.csv')
X = df[['Age', 'Days of Stay', 'Admission Type', 'Gender', 'Insurance Provider', 'Medication' ]]
X = pd.get_dummies(X) # Codificación one-hot para variables categóricas
y = df['Billing Amount']
X_train, X_test, y_train, y_test = train_test_split(X , y , test_size=0.2 , random_state=777)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test) #X_test_scaled = scaler.transform(X_test)           # Usa esos valores en el test
                                         #X_train_scaled = scaler.fit_transform(X_train) # Aprende media y std del train
kfold = KFold(n_splits=5 , shuffle=True , random_state=777)
lr = LinearRegression()
nn = MLPRegressor(max_iter=1000 , random_state=44)
xg = XGBRegressor(random_state=999)
param_grid_lr = {
    'fit_intercept':[True , False]
}
param_grid_xg = {
    'xgb_n_estimators':[100,200],
    'xgb_max_depth':[3,5],
    'xgb_learning_rate':[0.01 , 0.1],
    'xgb_subsample':[0.8,1],
    'xgb_colsample_bytree':[0.8,1]
}
param_grid_nn = {
    'hidden_layer_sizes': [(50,), (100,), (100, 50)],
    'learning_rate_init': [0.001, 0.01],
    'alpha': [0.0001, 0.001]
}
grid_lr = GridSearchCV(lr , param_grid_lr , cv = kfold , scoring='neg_mean_squared_error' , n_jobs=-1)
grid_nn = GridSearchCV(nn , param_grid_nn , cv = kfold , scoring='neg_mean_squared_error' , n_jobs=-1)
grid_xg = GridSearchCV(xg , param_grid_xg , cv = kfold , scoring='neg_mean_squared_error' , n_jobs=-1)
grid_lr.fit(X_train,y_train)
grid_nn.fit(X_train_scaled,y_train)
grid_xg.fit(X_train,y_train)
best_lr = grid_lr.best_estimator_
best_nn = grid_nn.best_estimator_
best_xg = grid_xg.best_estimator_
```

```

1 print(f"Regresion lineal: {grid_lr.best_params_}")
2 print(f"Red neuronal: {grid_nn.best_params_}")
3 print(f"XGBoost: {grid_xg.best_params_}")

Regresion lineal: {'fit_intercept': True}
Red neuronal: {'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.001}
XGBoost: {'xgb__colsample_bytree': 0.8, 'xgb__learning_rate': 0.01, 'xgb__max_depth': 3, 'xgb__n_estimators': 100, 'xgb__subsample': 0.8}

1 ensemble = VotingRegressor(
2     estimators=[
3         ('lr', best_lr),
4         ('nn', best_nn),
5         ('xg', best_xg)
6     ]
7 )

1 def evaluar_modelo(model, X_train, y_train, X_test, y_test, model_name, use_scale=False):
2     def adjusted_r2_score(y_true, y_pred, n_features):
3         r2 = r2_score(y_true, y_pred)
4         n = len(y_true)
5         adj_r2 = 1 - (1-r2) * (n-1) / (n - n_features -1)
6         return adj_r2
7
8     X_train_eval = X_train_scaled if use_scale else X_train
9     X_test_eval = X_test_scaled if use_scale else X_test
10
11     mse_scores = -cross_val_score(model, X_train_eval, y_train, cv=kfold, scoring='neg_mean_squared_error')
12     r2_scores = cross_val_score(model, X_train_eval, y_train, cv=kfold, scoring='r2')
13
14     n = X_train_eval.shape[0]
15     p = X_train_eval.shape[1]
16     adj_r2_scores = [1 - (1-r2) * (n-1) / (n-p-1) for r2 in r2_scores]
17
18     print(f"\n{model_name} (Validacion Cruzada en Entrenamiento) : ")
19     print(f"  MSE (mean ± std): {np.mean(mse_scores):.4f} ± {np.std(mse_scores):.4f}")
20     print(f"  R² (mean ± std): {np.mean(r2_scores):.4f} ± {np.std(r2_scores):.4f}")
21     print(f"  R² ajustado (mean ± std): {np.mean(adj_r2_scores):.4f} ± {np.std(adj_r2_scores):.4f}")
22
23     model.fit(X_train_eval, y_train)
24     y_pred = model.predict(X_test_eval)
25     mse_test = mean_squared_error(y_test, y_pred)
26     r2_test = r2_score(y_test, y_pred)
27     adj_r2_test = adjusted_r2_score(y_test, y_pred, X_train_eval.shape[1])
28
29     print(f"\n{model_name} (Conjunto prueba) : ")
30     print(f"  MSE: {mse_test:.4f}")
31     print(f"  R²: {r2_test:.4f}")
32     print(f"  R² ajustado: {adj_r2_test:.4f}")

1 evaluar_modelo(best_lr, X_train, y_train, X_test, y_test, 'Lr (optimizada)')
2 evaluar_modelo(best_nn, X_train_scaled, y_train, X_test_scaled, y_test, 'NN (optimizada)', use_scale=True)
3 evaluar_modelo(best_xg, X_train, y_train, X_test, y_test, 'XG (optimizada)')
4 evaluar_modelo(ensemble, X_train_scaled, y_train, X_test_scaled, y_test, 'Ensamblado (RL + NN + XG)', use_scale=True)

```

```

Lr (optimizada) (Validacion Cruzada en Entrenamiento) :
MSE (mean ± std): 3215.1729 ± 498.3043
R2 (mean ± std): 0.4607 ± 0.0728
R2 ajustado (mean ± std): 0.4449 ± 0.0749

Lr (optimizada) (Conjunto prueba) :
MSE: 2395.5688
R2: 0.5628
R2 ajustado: 0.5068

RF (optimizada) (Validacion Cruzada en Entrenamiento) :
MSE (mean ± std): 3243.7355 ± 218.0227
R2 (mean ± std): 0.4531 ± 0.0515
R2 ajustado (mean ± std): 0.4371 ± 0.0530

RF (optimizada) (Conjunto prueba) :
MSE: 3302.7597
R2: 0.3973
R2 ajustado: 0.3200

NN (optimizada) (Validacion Cruzada en Entrenamiento) :
MSE (mean ± std): 3335.0058 ± 733.9493
R2 (mean ± std): 0.4401 ± 0.1191
R2 ajustado (mean ± std): 0.4238 ± 0.1226
...
Ensamblado (RL + RF + NN) (Conjunto prueba) :
MSE: 2517.8636
R2: 0.5405
R2 ajustado: 0.4816

```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

```

1 esemble.fit(X_train_scaled , y_train)



lr
nn
xg
LinearRegression ?
MLPRegressor ?
XGBRegressor ?


```

```

1 y_pred = esemble.predict(X_test_scaled[0].reshape(1,-1))
2 print(f"y_pred = {y_pred[0]:.4f}")
3 print(f"y_test = {y_test.iloc[0]:.4f}") # Accede por posición, no por etiqueta
4

```

```

y_pred = 23373.8224
y_test = 13692.4049

```

```

1 # Diccionario para guardar métricas
2 resultados = {}
3 # Lista de modelos y nombres
4 modelos = [
5     ("Regresión Lineal", best_lr, X_test),
6     ("Red Neuronal", best_nn, X_test_scaled),
7     ("XGBoost", best_xg, X_test),
8     ("Ensamblado", ensamble, X_test_scaled)
9 ]
10 # Calcular métricas de cada modelo
11 for nombre, modelo, X_test_eval in modelos:
12     y_pred = modelo.predict(X_test_eval)
13     mse = mean_squared_error(y_test, y_pred)
14     r2 = r2_score(y_test, y_pred)
15     n = len(y_test)
16     p = X_test_eval.shape[1]
17     r2_adj = 1 - (1 - r2) * (n - 1) / (n - p - 1)
18
19     resultados[nombre] = {
20         "MSE": mse,
21         "R2": r2,
22         "R2 Ajustado": r2_adj
23     }
24 # Convertir a DataFrame y ordenar por R2
25 df_resultados = pd.DataFrame(resultados).T
26 df_resultados = df_resultados.sort_values(by="R2", ascending=False)
27 print("\n Comparación final de modelos:")
28 print(df_resultados.round(4))

```

Comparación final de modelos:

	MSE	R ²	R ² Ajustado
Regresión Lineal	1.990740e+08	-0.0033	-0.0074
Red Neuronal	1.992851e+08	-0.0044	-0.0085
Ensamblado	2.018962e+08	-0.0176	-0.0217
XGBoost	2.201454e+08	-0.1095	-0.1141

TÉCNICAS DE APRENDIZAJE NO SUPERVISADO

Supervisado

Los modelos no supervisados son aquellos que no utilizan una variable objetivo (target) para el entrenamiento. Es decir, trabajan únicamente con los datos de entrada (features), sin saber de antemano cuál debería ser la salida. Se utilizan principalmente para descubrir patrones ocultos, estructuras o agrupaciones dentro de los datos.

Tipo de Aprendizaje	Modelo	Descripción breve	Tarea principal
NO SUPERVISADO	K-Means	Agrupa datos en k grupos según similitud	Clustering
	PCA (Análisis de Componentes Principales)	Reduce dimensiones conservando varianza	Reducción de Dimensión

Modelado predictivo de una variable categórica

CLUSTERING (K-MODES)

Una de las técnicas más empleadas en este ámbito es el clustering, cuyo objetivo es agrupar observaciones con características similares en subconjuntos llamados clústeres o segmentos. A través de esta metodología, es posible identificar patrones ocultos, estructuras internas o perfiles representativos dentro de la muestra analizada.

Aplicar clustering resulta especialmente valioso cuando se busca entender la diversidad dentro de un conjunto heterogéneo de datos. Por ejemplo, permite segmentar pacientes según su admisión hospitalaria, clasificar pacientes por perfil de atención o por aseguradora.

K-Modes es un algoritmo de clustering utilizado específicamente para datos categóricos (en lugar de datos numéricos como en K-Means). Es una variante del K-Means, pero adaptada para trabajar con características categóricas, utilizando un criterio de distancia diferente. Mientras K-Means utiliza la media para calcular los centroides de los clusters, lo cual solo funciona con datos numéricos, K-Modes utiliza la moda (el valor más frecuente) para definir el centro de cada cluster. Además, el algoritmo K-Modes minimiza la disimilitud entre las observaciones dentro de cada cluster. La disimilitud se calcula usando el distanciamiento de Hamming, que mide cuántas veces las categorías son diferentes entre las observaciones.

De las 5 variables categóricas elegidas se hacen grupos dos a dos y se eligen los que a nuestro criterio tienen más sentido para la clasificación:

```

1 cat=['Admission Type', 'Gender', 'Insurance Provider', 'Medication', 'Blood Type']
2 comb = set()
3 for i in cat:
4     for j in cat:
5         if i < j:
6             comb.add((i,j))
7
8 comb

{('Admission Type', 'Blood Type'),
 ('Admission Type', 'Gender'),
 ('Admission Type', 'Insurance Provider'),
 ('Admission Type', 'Medication'),
 ('Blood Type', 'Gender'),
 ('Blood Type', 'Insurance Provider'),
 ('Blood Type', 'Medication'),
 ('Gender', 'Insurance Provider'),
 ('Gender', 'Medication'),
 ('Insurance Provider', 'Medication')}

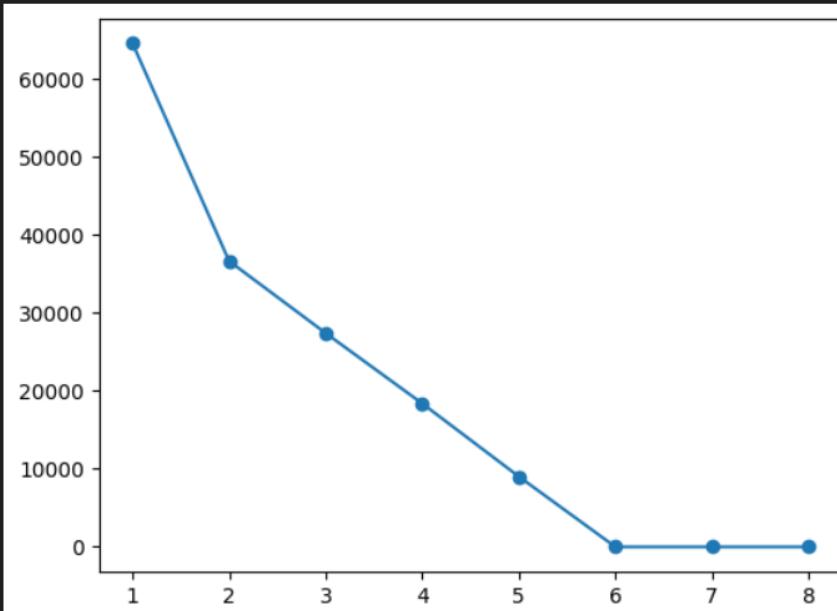
```

- Admission Type-Gender

```

1 costes = []
2 df11 = df[['Admission Type' , 'Gender']]
3 num_clust = range(1,9)
4 for k in num_clust:
5     km = KModes(n_clusters= k , init='Huang' , n_init=5 , verbose=0)
6     km.fit(df11)
7     costes.append(km.cost_)
8
9 plt.plot(num_clust , costes , marker='o')
10 plt.show()

```



```

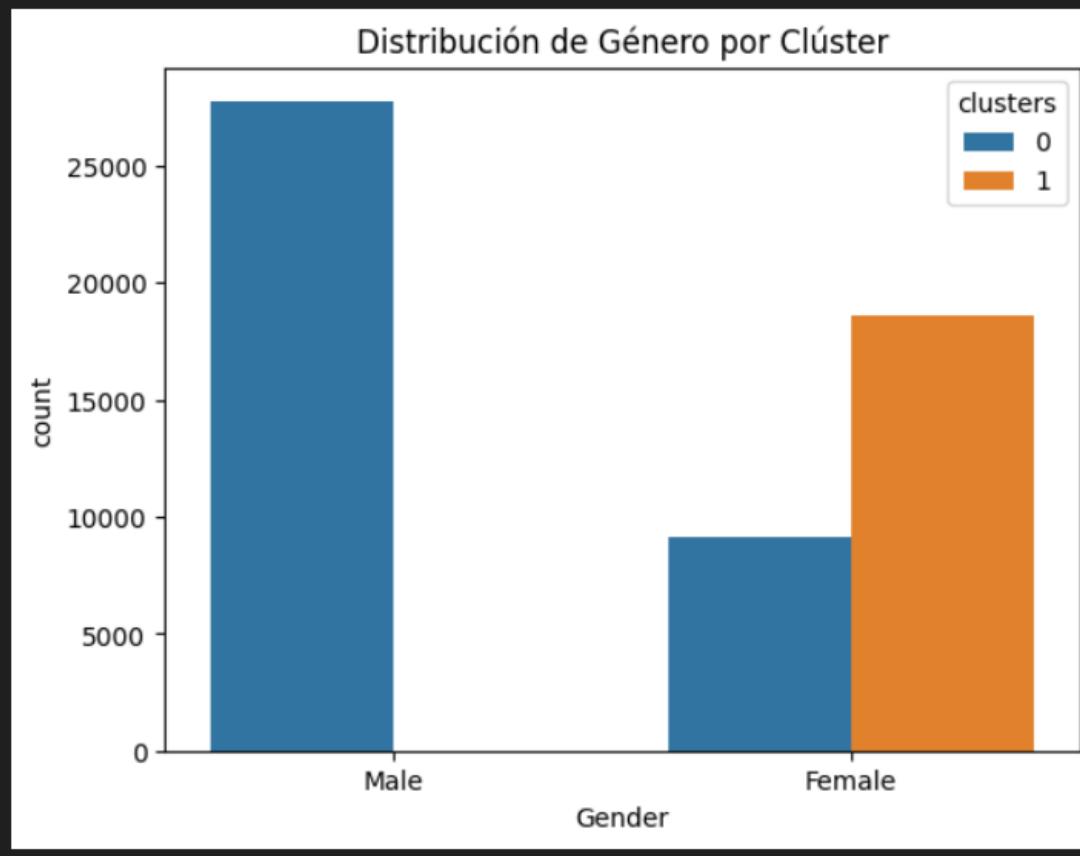
1 #Probamos 2 clusters
2 num = 2
3 km = KModes(n_clusters= num , init='Huang' , n_init=5 , verbose = 1)
4 clusters = km.fit_predict(df11)
5 df11['clusters'] = clusters
6 df11.groupby('clusters').value_counts()

Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 37101.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 18525, cost: 36658.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 9281, cost: 36658.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 36658.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 37111.0
Best run was number 2

clusters  Admission Type  Gender
0         Urgent      Male    9468
          Elective    Male    9281
          Urgent     Female   9108
          Emergency  Male    9025
1         Elective    Female  9374
          Emergency  Female  9244

```

```
1 sns.countplot(data=df11, x='Gender', hue='clusters')
2 plt.title('Distribución de Género por Clúster')
3 plt.show()
```

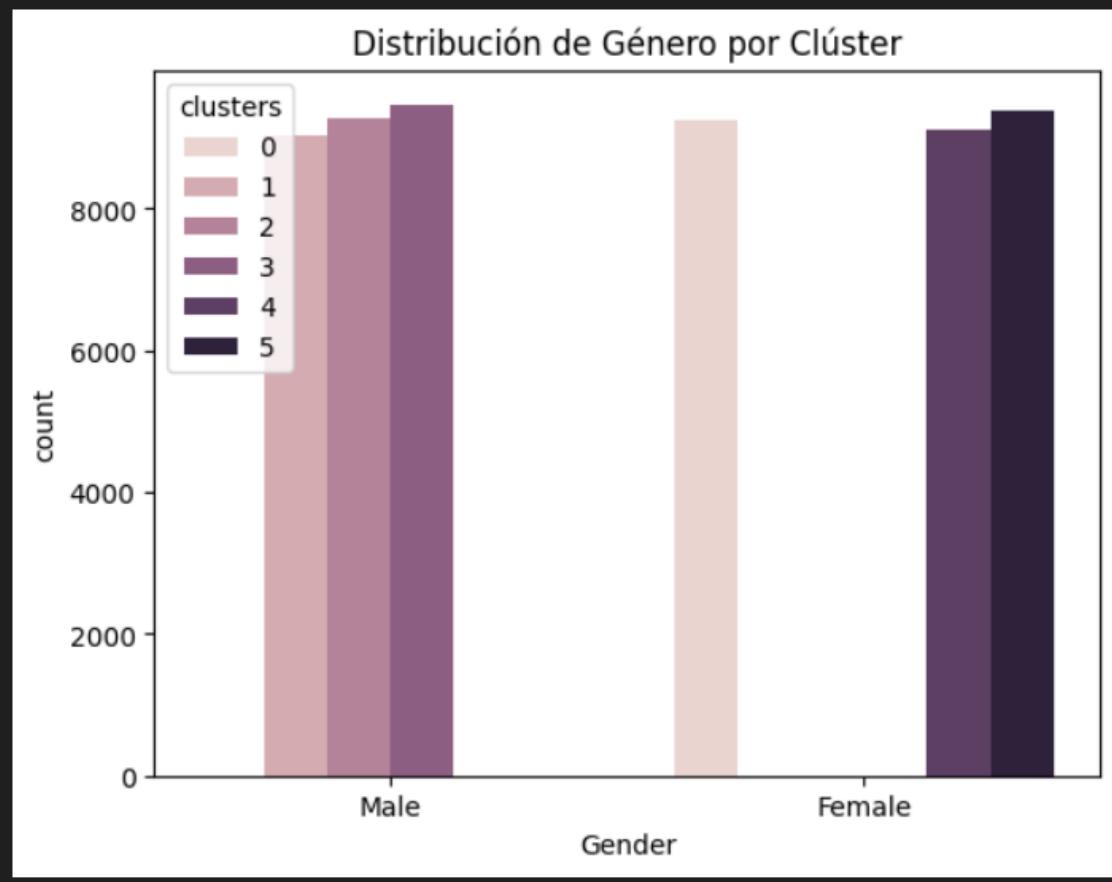


```
1 #Probamos 6 clusters
2 num = 6
3 km = KModes(n_clusters= num , init='Huang' , n_init=5 , verbose = 1)
4 clusters = km.fit_predict(df11)
5
6 df11['clusters'] = clusters
7
✓ 1.8s
```

Init: initializing centroids
Init: initializing clusters
Starting iterations...

```
1 sns.countplot(data=df11, x='Gender', hue='clusters')
2 plt.title('Distribución de Género por Clúster')
3 plt.show()
```

⌚



```
1 df11.groupby('clusters').value_counts()
```

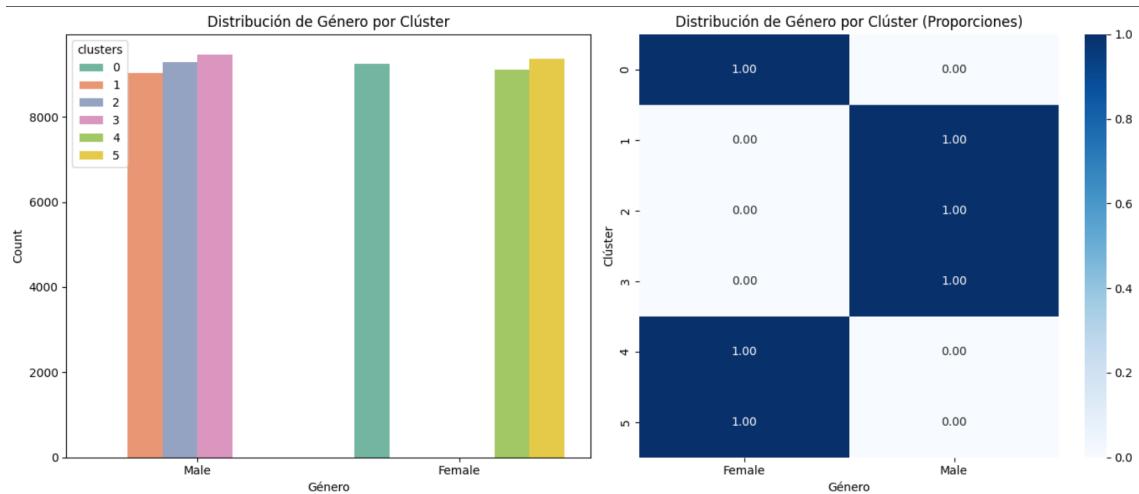
⌚

```
clusters  Admission Type  Gender
0          Emergency      Female  9244
1          Emergency      Male    9025
2          Elective        Male    9281
3          Urgent          Male    9468
4          Urgent          Female  9108
5          Elective        Female  9374
Name: count, dtype: int64
```

```

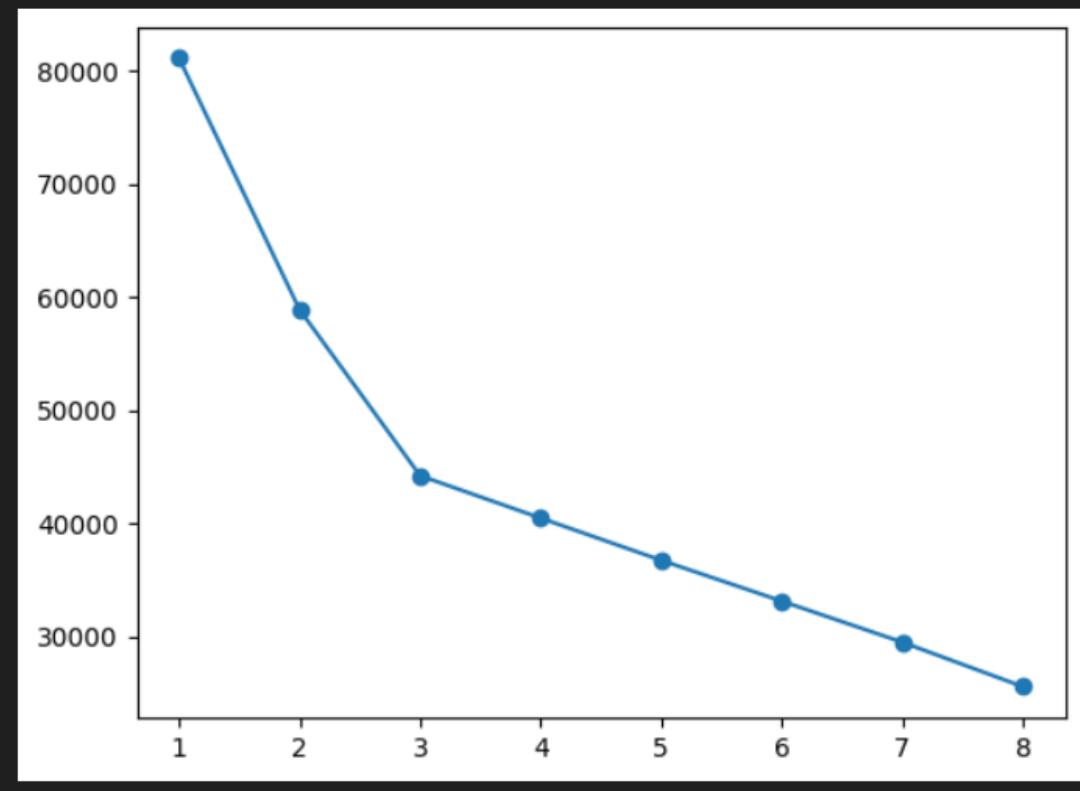
1 #VISUALIZACION CONJUNTA
2 fig, axes = plt.subplots(1, 2, figsize=(14, 6)) # Gráfico combinado
3
4 sns.countplot(data=df11, x='Gender', hue='clusters', ax=axes[0], palette='Set2')
5 axes[0].set_title('Distribución de Género por Clúster')
6 axes[0].set_xlabel('Género')
7 axes[0].set_ylabel('Count')
8
9 cross_tab = pd.crosstab(df11['clusters'], df11['Gender'], normalize='index')
10 sns.heatmap(cross_tab, annot=True, cmap='Blues', fmt=".2f", ax=axes[1])
11 axes[1].set_title('Distribución de Género por Clúster (Proporciones)')
12 axes[1].set_xlabel('Género')
13 axes[1].set_ylabel('Clúster')
14
15 plt.tight_layout()
16 plt.show()

```



- Admission Type-Insurance Provider

```
1 costes = []
2 df12 = df[['Admission Type' , 'Insurance Provider']]
3 num_clust = range(1,9)
4 for k in num_clust:
5     km = KModes(n_clusters= k , init='Huang' , n_init=5 , verbose=0)
6     km.fit(df12)
7     costes.append(km.cost_)
8
9 plt.plot(num_clust , costes , marker='o')
10 plt.show()
```



```

1 num = 3
2 km = KModes(n_clusters= num , init='Huang' , n_init=5 , verbose = 1)
3 clusters = km.fit_predict(df12)
4
5 df12['clusters2'] = clusters
✓ 14.7s

Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 14772, cost: 44316.0
Run 1, iteration: 2/100, moves: 3673, cost: 44316.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 3709, cost: 44323.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 18203, cost: 44376.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 51549.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 14656, cost: 51421.0
Best run was number 1

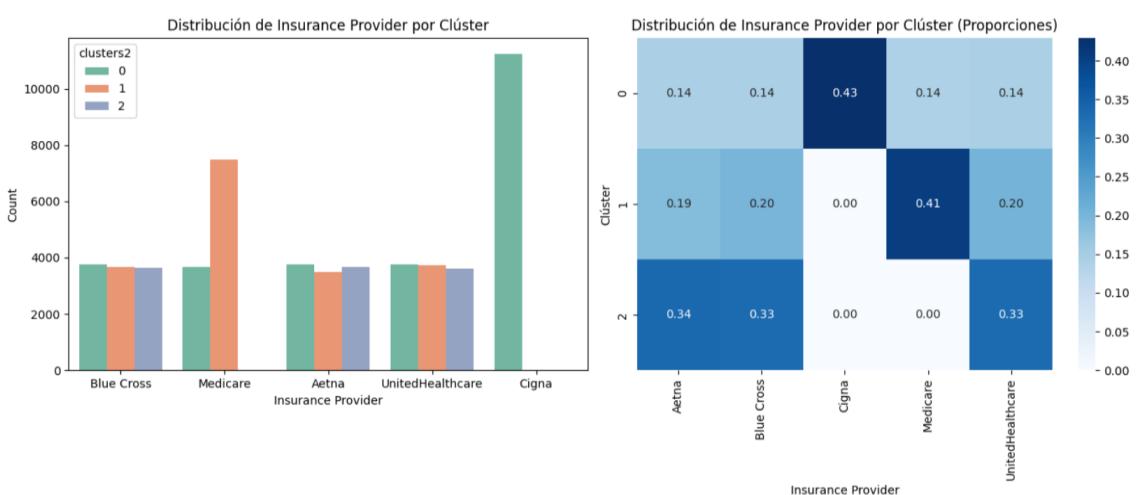
```

```

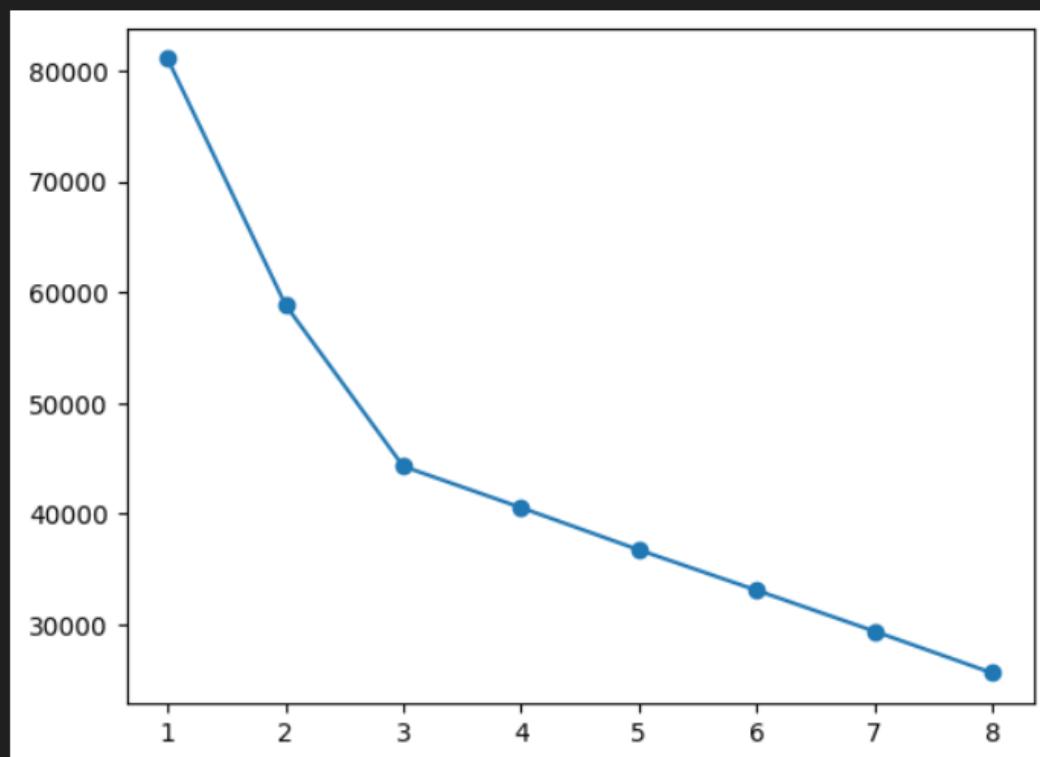
1 df12.groupby('clusters2').value_counts()
✓ 0.0s

clusters2  Admission Type  Insurance Provider
0          Urgent          Cigna          3863
           Elective        UnitedHealthcare  3766
                           Aetna          3754
                           Blue Cross      3747
                           Cigna          3709
                           Medicare        3679
           Emergency        Cigna          3677
1          Urgent          Medicare        3800
                           UnitedHealthcare  3746
                           Blue Cross      3683
           Emergency        Medicare        3675
           Urgent           Aetna          3484
2          Emergency        Aetna          3675
           Emergency        Aetna          3629
                           Blue Cross      3629
                           UnitedHealthcare  3613
Name: count, dtype: int64

```

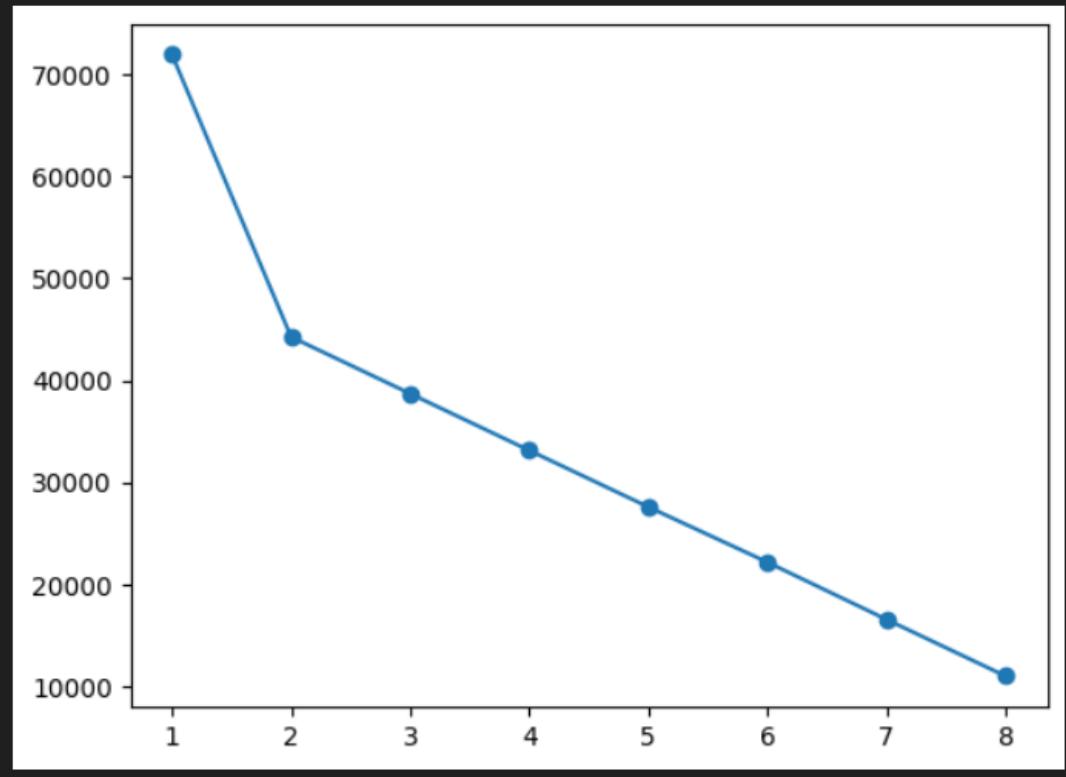


```
1 costes = []
2 df12 = df[['Admission Type' , 'Insurance Provider']]
3 num_clust = range(1,9)
4 for k in num_clust:
5     km = KModes(n_clusters= k , init='Huang' , n_init=5 , verbose=0)
6     km.fit(df12)
7     costes.append(km.cost_)
8
9 plt.plot(num_clust , costes , marker='o')
10 plt.show()
✓ 2m 57.4s
```



- Insurance Provider-Gender

```
1 costes = []
2 df13 = df[['Insurance Provider', 'Gender']]
3 num_clust = range(1,9)
4 for k in num_clust:
5     km = KModes(n_clusters= k , init='Huang' , n_init=5 , verbose=0)
6     km.fit(df13)
7     costes.append(km.cost_)
8
9 plt.plot(num_clust , costes , marker='o')
10 plt.show()
```



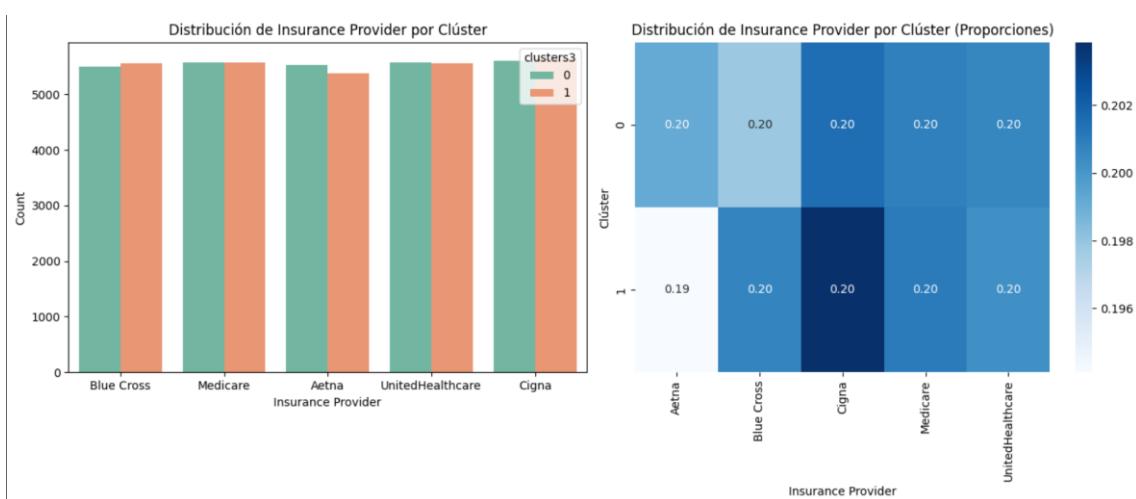
```
1 num = 2
2 km = KModes(n_clusters= num , init='Huang' , n_init=5 , verbose = 1)
3 clusters = km.fit_predict(df13)
4
5 df13['clusters3'] = clusters

Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 60852.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 44269.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 60852.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 22003, cost: 44328.0
Run 4, iteration: 2/100, moves: 0, cost: 44328.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 44251.0
Best run was number 5
```

```
1 df13.groupby('clusters3').value_counts()
```

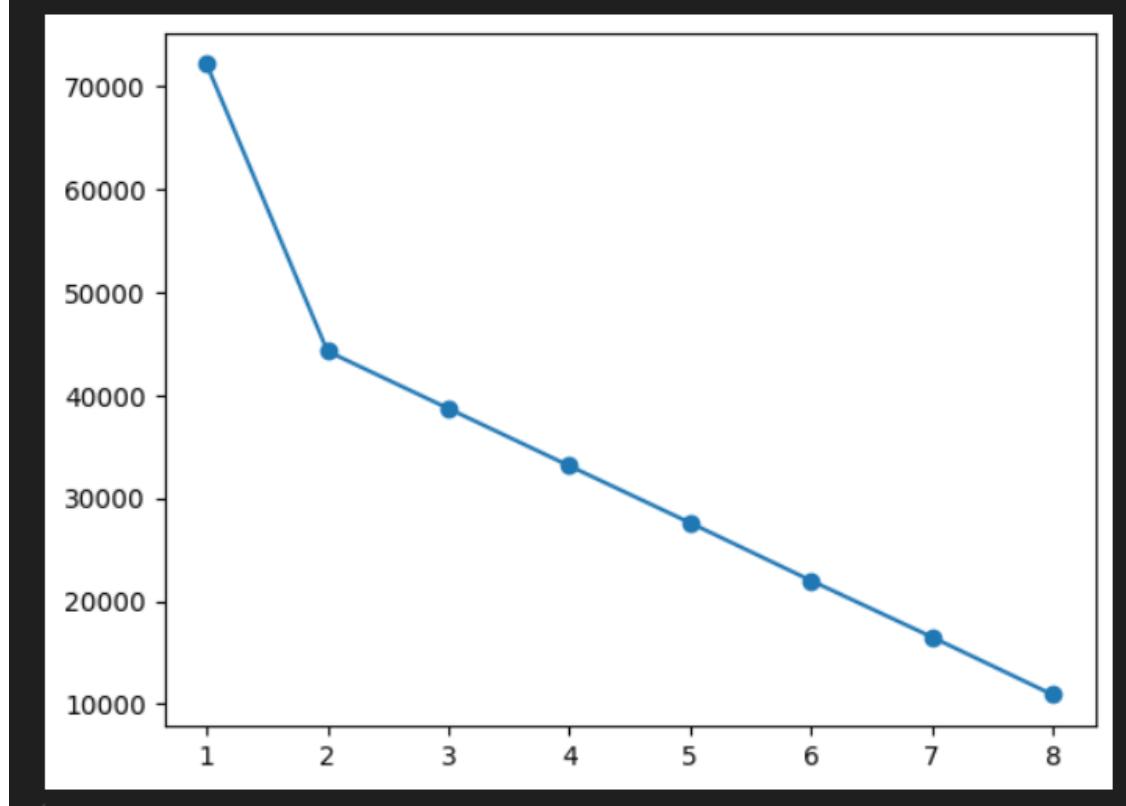
clusters3	Insurance Provider	Gender	Count
0	Cigna	Male	5597
	Medicare	Male	5579
	UnitedHealthcare	Male	5573
	Aetna	Male	5531
	Blue Cross	Male	5494
1	Cigna	Female	5652
	Medicare	Female	5575
	Blue Cross	Female	5565
	UnitedHealthcare	Female	5552
	Aetna	Female	5382

Name: count, dtype: int64



- Medication-Gender

```
1 costes = []
2 df14 = df[['Medication', 'Gender']]
3 num_clust = range(1,9)
4 for k in num_clust:
5     km = KModes(n_clusters= k , init='Huang' , n_init=5 , verbose=0)
6     km.fit(df14)
7     costes.append(km.cost_)
8
9 plt.plot(num_clust , costes , marker='o')
10 plt.show()
```



```

1 num = 2
2 km = KModes(n_clusters= num , init='Huang' , n_init=5 , verbose = 1)
3 clusters = km.fit_predict(df14)
4
5 df14['clusters4'] = clusters

Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 60959.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 61018.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 21978, cost: 44310.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 44435.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 44344.0
Best run was number 3

```

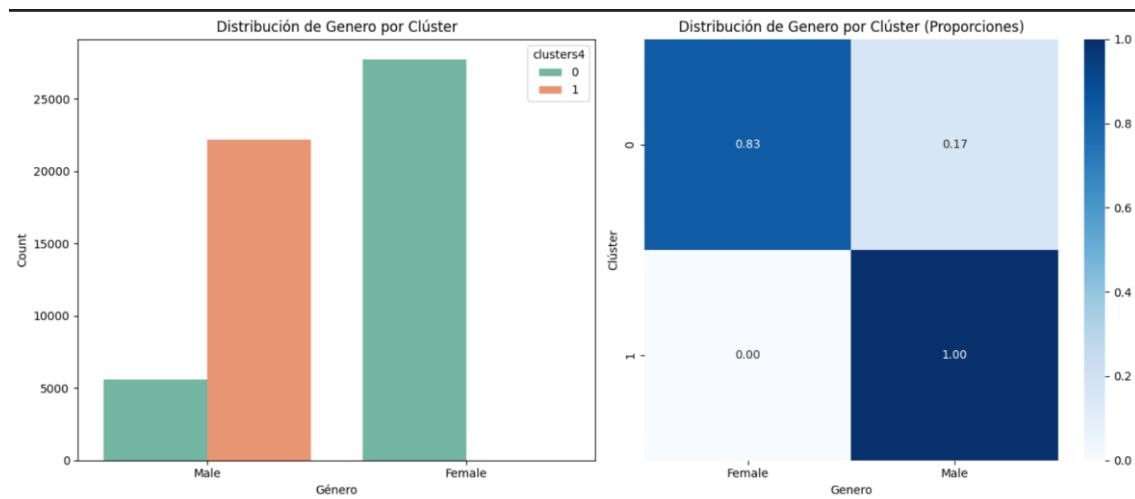
```

1 df14.groupby('clusters4').value_counts()


```

clusters4	Medication	Gender	Count
0	Aspirin	Female	5622
	Ibuprofen	Female	5593
	Lipitor	Male	5589
		Female	5551
	Penicillin	Female	5528
1	Paracetamol	Female	5432
	Paracetamol	Male	5639
	Penicillin	Male	5540
	Ibuprofen	Male	5534
Aspirin	Male	5472	

Name: count, dtype: int64



PCA APPLICADO A LOS CLUSTERS GENERADOS POR K-MODES

El Análisis de Componentes Principales (PCA) es una técnica de reducción de dimensionalidad que permite visualizar datos complejos en dos o tres dimensiones. Cuando se aplica a los resultados de un algoritmo de clustering como K-Means, PCA facilita la interpretación visual de los grupos generados. Al proyectar los datos en las componentes principales, se pueden observar mejor las separaciones entre los clusters. Esto permite verificar si los grupos encontrados por K-Means están bien definidos y diferenciados. Además, PCA ayuda a detectar posibles solapamientos o outliers que podrían afectar la calidad del clustering.

```
# 1. Filtrado y preparación
df["Gender"] = df["Gender"].map({"Male": 0, "Female": 1})

# 2. Variables predictoras (sin la columna objetivo)
X = df[['Age', 'Days of Stay', 'Admission Type', 'Test Results', 'Insurance Provider', 'Medication']]

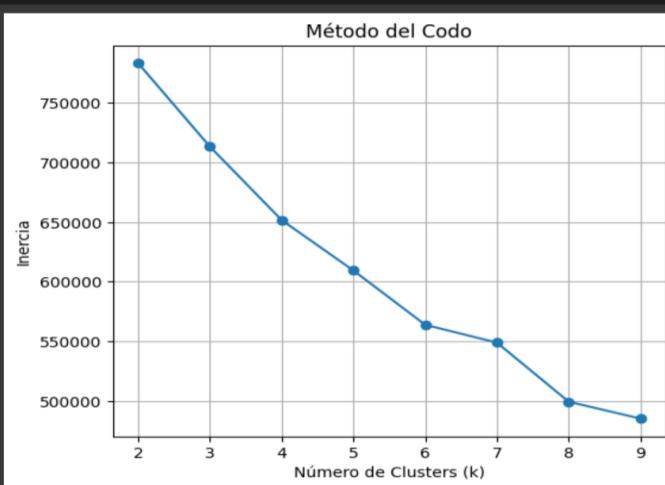
X = pd.get_dummies(X)

# 3. Escalar variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# 4. Método del codo
inertias = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(6, 5))
plt.plot(K_range, inertias, marker='o')
plt.title('Método del Codo')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Inercia')
plt.grid(True)
plt.show()
```



Se aplicó la técnica del codo para determinar el valor óptimo de K, pero no mostró un punto claro de inflexión. Por ello, se utilizó el método del coeficiente de silueta, el cual indicó que el valor más adecuado es K = 6.

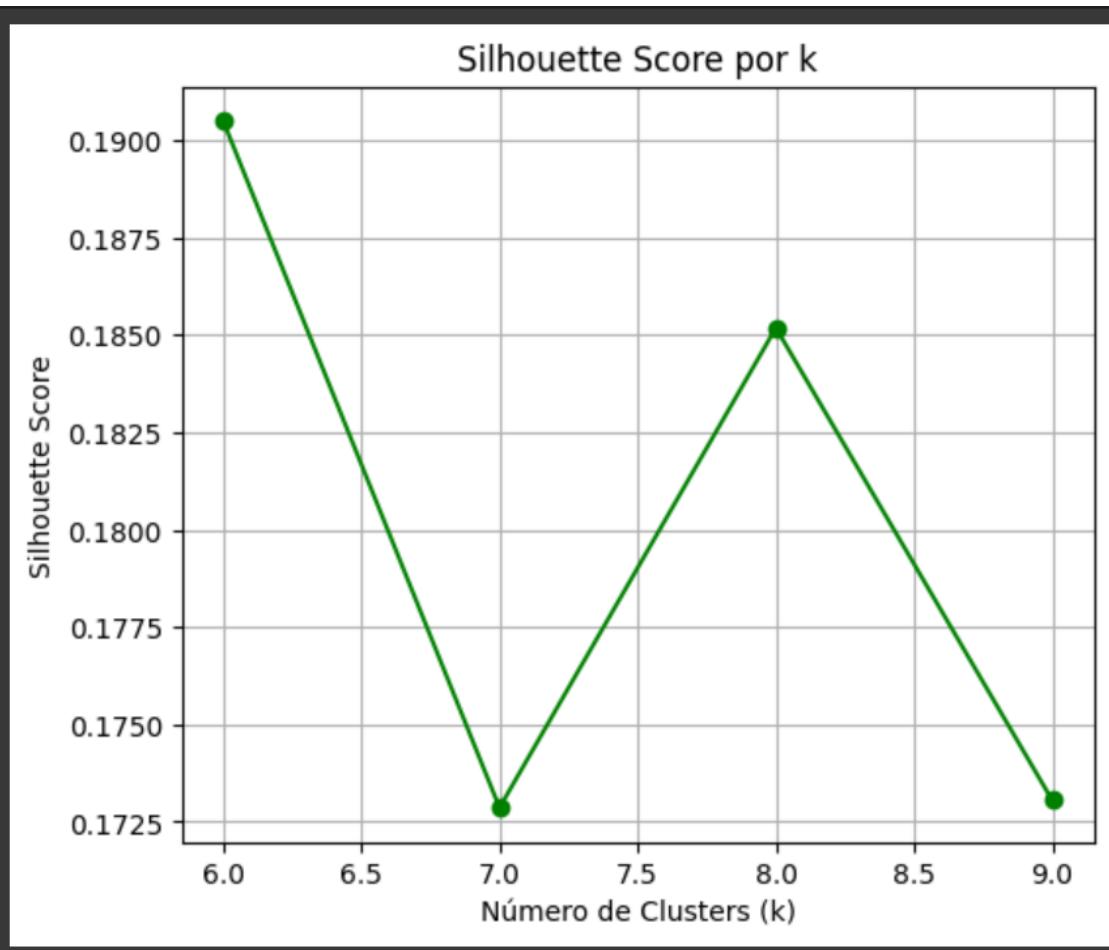
```

# 5. Método de silhouette score
silhouette_scores = []
K_range = range(6, 10) #porque veiamos 6 en el método del codo

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    clusters = kmeans.fit_predict(X_scaled)
    silhouette_scores.append(silhouette_score(X_scaled, clusters))

plt.figure(figsize=(6, 5))
plt.plot(K_range, silhouette_scores, marker='o', color='green')
plt.title('Silhouette Score por k')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()

```



```

# 6. Elegir un k (manual basandonos en las gráficas)
k = 6 # de acuerdo con los dos métodos
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)

# 7. Evaluar agrupamiento con silhouette score
sil_score = silhouette_score(X_scaled, clusters)
print(f"Silhouette Score final para k={k}: {sil_score:.2f}")

Silhouette Score final para k=6: 0.19

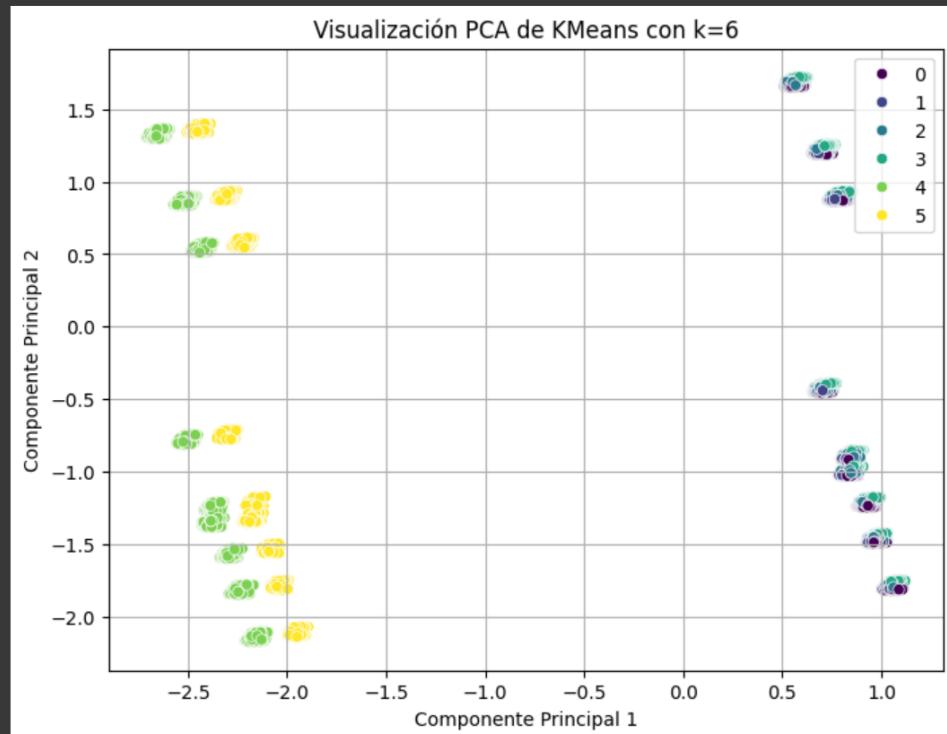
```

```

# 8. Visualización con PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=clusters, palette='viridis')
plt.title(f'Visualización PCA de KMeans con k={k}')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.grid(True)
plt.show()

```



```

df[ 'Cluster' ] = clusters

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55392 entries, 0 to 55391
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  --  
 0   Name              55392 non-null   object  
 1   Age               55392 non-null   int64  
 2   Gender            55392 non-null   int64  
 3   Blood Type        55392 non-null   object  
 4   Medical Condition 55392 non-null   object  
 5   Date of Admission 55392 non-null   object  
 6   Doctor            55392 non-null   object  
 7   Hospital           55392 non-null   object  
 8   Insurance Provider 55392 non-null   object  
 9   Billing Amount     55392 non-null   float64 
 10  Room Number        55392 non-null   int64  
 11  Admission Type    55392 non-null   object  
 12  Discharge Date    55392 non-null   object  
 13  Medication          55392 non-null   object  
 14  Test Results        55392 non-null   object  
 15  Days of Stay        55392 non-null   int64  
 16  AD_Mes             55392 non-null   int64  
 17  AD_DiaSemana       55392 non-null   object  
 18  AD_Trimestre       55392 non-null   int64  
 19  AD_Anio            55392 non-null   int64  
 20  DD_Mes             55392 non-null   int64  
 21  DD_DiaSemana       55392 non-null   object  
 22  DD_Trimestre       55392 non-null   int64  
 23  DD_Anio            55392 non-null   int64  
 24  Cluster             55392 non-null   int32  
dtypes: float64(1), int32(1), int64(10), object(13)
memory usage: 10.4+ MB

```

```
#Agrupar por Cluster y revisar medias de variables:  
df.groupby('Cluster')[['Age', 'Days of Stay']].mean()
```

Cluster	Age	Days of Stay
0	39.544894	15.525323
1	39.791227	15.607622
2	39.738270	15.394805
3	39.582494	15.453357
4	39.506790	15.549952
5	39.341255	15.697569

A partir de las medias agrupadas por cluster, se pueden extraer las siguientes ideas:

Los valores medios de edad y días de estancia son muy similares entre los seis clusters. La edad promedio se mantiene alrededor de los 39.5 años en todos los grupos. La diferencia en los días de estancia también es mínima, variando entre 15.39 y 15.70 días. Esto indica que estas dos variables no explican claramente la segmentación. Es probable que otras variables hayan tenido mayor peso en la formación de los clusters. Se recomienda analizar otras características o aplicar visualizaciones para interpretar mejor los grupos.

```
#Ver proporciones de género por clúster:  
pd.crosstab(df['Gender'], df['Cluster'], normalize='columns')
```

Cluster	0	1	2	3	4	5
Gender						
0	0.464239	0.478907	0.474178	0.47458	0.480238	0.461538
1	0.535761	0.521093	0.525822	0.52542	0.519762	0.538462

Las proporciones de género son bastante equilibradas en todos los clusters. En cada grupo, el porcentaje de mujeres (valor 1) se mantiene ligeramente por encima del 50%. El cluster 5 presenta la mayor proporción de mujeres, con un 53.85%, mientras que el cluster 0 tiene la menor, con un 53.58%. Estas pequeñas variaciones sugieren que el género no fue un factor determinante en la segmentación. En general, no se observan diferencias significativas por género entre los clusters.

```
#Ver proporciones de tipo de admisión por clúster:  
pd.crosstab(df['Admission Type'], df['Cluster'], normalize='columns')
```

Admission Type	Cluster	0	1	2	3	4	5
Elective	0	0.202823	0.197148	0.199673	0.195564	0.197987	0.197569
Emergency	1	0.448820	0.453619	0.446086	0.451679	0.454413	0.464804
Urgent	2	0.348357	0.349233	0.354241	0.352758	0.347599	0.337627

Las proporciones de tipo de admisión son similares entre los diferentes clusters. En todos los grupos, la mayoría de los pacientes fueron admitidos por emergencia, con valores cercanos al 45%. Las admisiones urgentes representan alrededor del 35%, con ligeras variaciones entre clusters. Las admisiones electivas son las menos frecuentes, con proporciones cercanas al 20%. En conjunto, no se observan diferencias relevantes en el tipo de admisión que expliquen la segmentación en clusters.

Por último, dado que las variables analizadas hasta ahora (edad, días de estancia, género y tipo de admisión) muestran muy poca variabilidad entre clusters, eso sugiere que los grupos creados por K-Means no están claramente diferenciados en estas dimensiones. Por lo tanto, usar estos clusters como etiquetas para entrenar un modelo de predicción supervisado no sería recomendable, al menos con las variables actuales.

7. DISCUSIÓN

Este estudio analiza un conjunto de datos sintéticos del sistema de salud con el objetivo de predecir el monto de facturación médica, basado en diversas variables clínicas y administrativas. En el análisis, se observó que solo algunas variables, como la aseguradora y los días de estancia hospitalaria, mostraron una leve relación con los costos, mientras que otras, como el género, tipo de admisión o condición médica, no tuvieron un impacto significativo.

Las técnicas estadísticas aplicadas, tales como el test Chi-cuadrado (Chi2) y los modelos de regresión, confirmaron que muchas de las variables no guardaban una relación estadísticamente significativa con el monto de facturación. Esta observación resalta la limitación de las variables disponibles para realizar predicciones precisas.

Una de las principales limitaciones del estudio fue la naturaleza sintética de los datos, lo cual puede haber afectado negativamente la precisión de los modelos predictivos, ya que estos datos probablemente no reflejan la complejidad ni la variabilidad de los datos reales. Los algoritmos de machine learning probados, incluyendo regresión lineal, Random Forest, redes neuronales y XGBoost, arrojaron resultados muy pobres, con valores negativos de R^2 , lo que sugiere que los modelos no mejorarían las predicciones más allá de un simple cálculo de promedio.

Adicionalmente, los análisis de clústeres y las técnicas de reducción dimensional no lograron identificar segmentos bien diferenciados entre los pacientes, lo que refleja una falta de estructura en los datos que pudiera permitir una segmentación efectiva.

A pesar de estas limitaciones, el estudio sigue un enfoque metodológico robusto que incluye una exhaustiva limpieza de los datos, un análisis exploratorio (EDA), visualización de patrones y pruebas con modelos tanto supervisados como no supervisados. Asimismo, se enfatiza el esfuerzo por abordar el desequilibrio en las variables, incorporando datos adicionales más representativos (mi_data) y

creando un conjunto balanceado (mi_data_balanceado), con la esperanza de obtener predicciones más precisas y equilibradas.

Al comparar los resultados de este estudio con los de investigaciones basadas en bases de datos reales, como MIMIC-III, se encontró que los resultados obtenidos en este estudio fueron significativamente más bajos en términos de R^2 . No obstante, el análisis ofrece un ejercicio académico valioso y proporciona recomendaciones clave para mejorar la calidad de los datos y las estrategias de modelado en investigaciones futuras.

Desde una perspectiva práctica, los resultados confirman que las variables presentes en el conjunto de datos no son suficientes para predecir el monto de facturación hospitalaria de manera precisa. Las fortalezas del estudio residen en su revisión metodológica completa y en el uso de diversas técnicas de machine learning, lo que asegura la robustez del proceso. Sin embargo, la principal limitación sigue siendo la naturaleza sintética de los datos, que carecen de la variabilidad y complejidad de los datos clínicos reales.

8. CONCLUSIONES

- **Metodología completa:** El estudio incluyó un proceso metodológico exhaustivo que abarca la limpieza de datos, codificación, detección de outliers, análisis estadístico, modelos predictivos y clustering. Aunque el valor predictivo clínico es limitado en este contexto sintético, el ejercicio resulta valioso como una base para la creación de prototipos o para su aplicación en sistemas reales. La calidad del conjunto de datos es un factor determinante para la efectividad de los modelos de Machine Learning en el ámbito de la salud.
- **Ánalysis exhaustivo de relaciones entre variables:** Se realizó un análisis detallado de las relaciones entre diferentes tipos de variables, como categóricas versus numéricas, categóricas versus categóricas, entre otras. Este análisis reveló importantes patrones que ayudan a comprender cómo interactúan las variables entre sí.
- **Sesgo en los datos:** La distribución desequilibrada de algunas categorías, como tipo de sangre o proveedor de seguros, introdujo sesgos en el entrenamiento de los modelos, lo que afectó la precisión de las predicciones y generó resultados no representativos de la realidad.
- **Uso de diferentes técnicas de Machine Learning:** Se probaron varias técnicas de Machine Learning, tanto supervisadas como no supervisadas, con el fin de identificar patrones y construir modelos predictivos. Sin embargo, la naturaleza sintética de los datos impidió que los modelos reflejaran completamente la complejidad de los datos clínicos reales, afectando negativamente los resultados obtenidos.
- **Naturaleza sintética de los datos:** Aunque los datos fueron modificados para reflejar una mayor realismo, siguen sin capturar la complejidad inherente a los datos clínicos reales, lo que limita la capacidad de los modelos para identificar patrones precisos y útiles.
- **Variables irrelevantes:** Las variables con alta cardinalidad, como “Doctor” y “Hospital”, fueron consideradas irrelevantes para el análisis, ya que no proporcionaron agrupaciones informativas consistentes y complicaron la interpretación de los resultados.
- **Factores asociados al monto de facturación:** Se identificaron relaciones entre algunas variables y el monto de facturación. En particular, se observó que el tipo de aseguradora y los días de estancia hospitalaria tenían cierta correlación con los costos. Por otro lado, no se encontraron diferencias significativas entre géneros en cuanto a las variables clínicas ni en los costos de facturación. Además, se detectaron hospitales y médicos asociados a facturaciones más altas, así como enfermedades más costosas dependiendo de la

condición médica de los pacientes.

- **Predominancia de admisiones de emergencia:** La mayoría de las admisiones fueron de emergencia, lo que podría indicar un patrón relevante en los costos de facturación que debe ser explorado más a fondo en investigaciones futuras.

- **Modelado predictivo limitado:** Todos los modelos de regresión (lineal, Random Forest, redes neuronales y XGBoost) mostraron un R^2 negativo, lo que sugiere que ningún modelo fue capaz de explicar adecuadamente la variabilidad en el costo de facturación. Este resultado resalta las dificultades inherentes a trabajar con datos sintéticos y la necesidad de datos más complejos y representativos para mejorar las predicciones.

En conclusión, aunque el estudio presenta varias limitaciones debido a la naturaleza sintética de los datos, ofrece una base para futuras investigaciones en la predicción de costos en el ámbito de la salud y subraya la importancia de contar con datos de calidad para obtener resultados robustos y aplicables.

9. RECOMENDACIONES

Finalmente, se sugieren pasos futuros para mejorar las predicciones de costos y resultados. Entre estos, se destaca la incorporación de datos más completos sobre la gravedad clínica, las complicaciones y comorbilidades, así como la realización de validaciones externas con bases de datos reales, lo que podría ayudar a afinar y optimizar los algoritmos predictivos para obtener resultados más precisos y aplicables en la práctica clínica.