Inteligencia Artificial

Procesamiento, Evaluación, Optimización y Sobreajuste

Procesamiento

@mihifidem
If you want to master something, teach it.

04_ 2.Procesamiento. Preprocesado

2 Preprocesamiento de datos para clasificación

1. Data Cleaning
2. Análisis exploratorio de datos
3. Las variables dummy

04_ 2.Procesamiento. Preprocesado

"La Inteligencia Artificial, la profunda, es el futuro de la computación. Las máquinas capaces de aprender y decidir por sí mismas es lo que nos depara el futuro.."

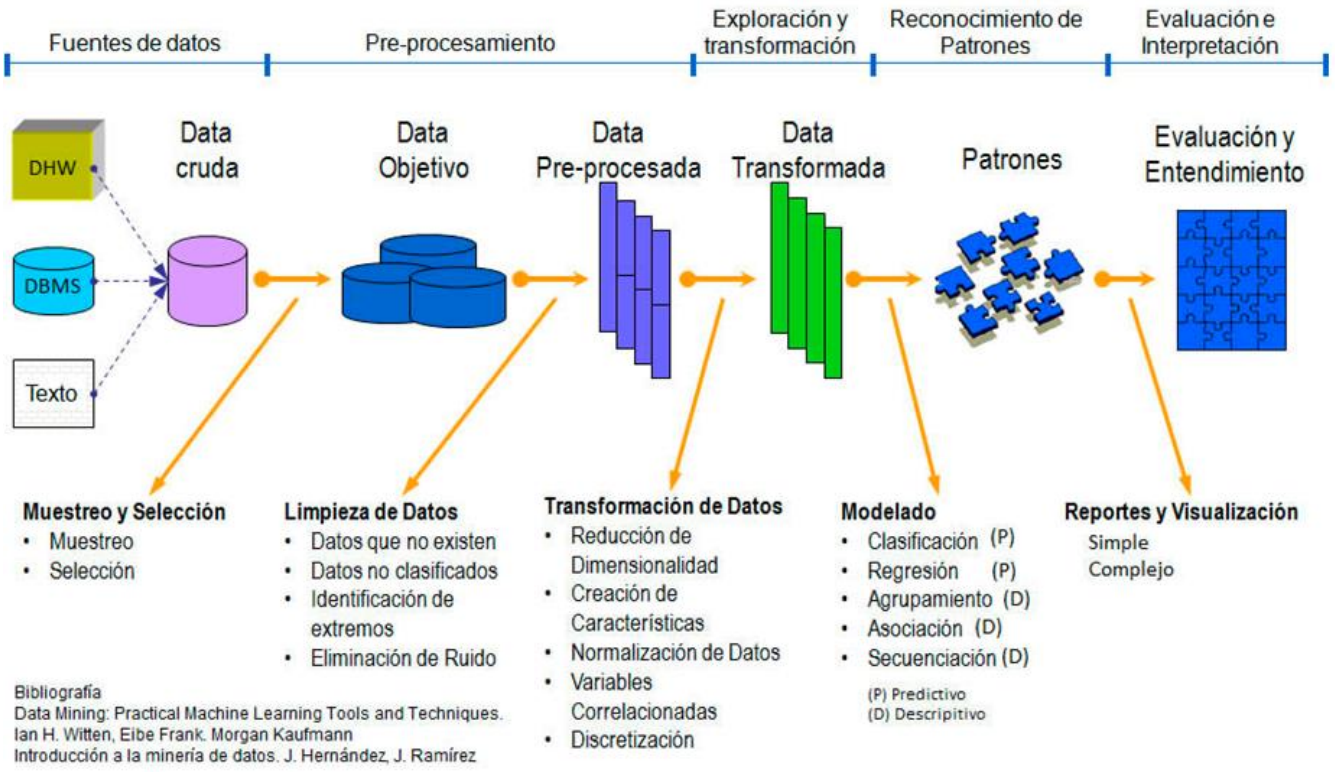
- Jensen Huang

Jensen Huang es el cofundador, presidente y CEO de NVIDIA Corporation, una empresa líder en la creación de procesadores gráficos (GPUs) que son ampliamente utilizados en aplicaciones de Inteligencia Artificial. Su cita destaca la creencia de que el futuro de la informática reside en máquinas que sean capaces de aprender y tomar decisiones autónomas, esencialmente la base de la IA.

04_ 2.Procesamiento. Preprocesado

4.2. Preprocesamiento de datos para clasificación

El preprocesamiento de datos es un paso crítico en el proceso de Machine Learning y es especialmente importante para la clasificación. Aquí te menciono algunos pasos que normalmente se llevan a cabo en el preprocesamiento de datos:



Limpieza de datos: Este paso incluye la identificación y corrección de errores en los datos, eliminando o imputando los valores faltantes. Los métodos de imputación comunes incluyen el uso de medidas de tendencia central como la media o la mediana, o utilizando algoritmos de aprendizaje automático para predecir los valores faltantes.

Transformación de los datos: Muchos algoritmos de Machine Learning se desempeñan mejor cuando los datos son de una naturaleza específica. Por ejemplo, algunos algoritmos requieren que todas las características sean numéricas, mientras que otros pueden requerir que los datos se encuentren en una escala específica (por ejemplo, 0-1).

Extracción y selección de características: Esto implica la selección de las características más relevantes para la clasificación. A veces, puede haber características que no aportan mucha información para la clasificación y pueden eliminarse. Otros métodos pueden incluir la transformación de variables para crear nuevas características que puedan proporcionar más información para el modelo de clasificación.

04_ 2.Procesamiento. Preprocesado

División de los datos: Por lo general, se divide el conjunto de datos en conjuntos de entrenamiento y prueba para poder evaluar el rendimiento del modelo. A veces también se puede tener un conjunto de validación.

Balanceo de clases: En muchos problemas de clasificación, se pueden tener clases desequilibradas (es decir, una clase tiene muchas más muestras que la otra). En estos casos, puede ser útil aplicar técnicas de balanceo de clases para evitar que el modelo esté sesgado hacia la clase más frecuente.

Estos son algunos de los pasos más comunes, pero el preprocesamiento de datos puede variar dependiendo del problema específico y del conjunto de datos con el que estés trabajando.

4.2.1. Data Cleaning

La limpieza de datos, o "data cleaning", es una parte esencial del proceso de análisis de datos. Aquí te menciono algunos de los pasos más comunes que se llevan a cabo en este proceso:

Manejo de valores faltantes: Los valores faltantes pueden ser un problema en muchos conjuntos de datos. Dependiendo de la naturaleza de los datos y del problema, puedes optar por eliminar las filas o columnas con valores faltantes, o puedes imputar los valores faltantes con un valor predeterminado, la media, mediana, o incluso un valor estimado por un modelo de aprendizaje automático.

El manejo de valores faltantes es una parte importante del preprocesamiento de datos en cualquier proyecto de análisis de datos o machine learning. Los datos pueden faltar por una variedad de razones: tal vez no se recogieron, tal vez se perdieron durante el procesamiento, etc. No importa la razón, es importante manejar estos valores faltantes de manera adecuada para evitar errores o sesgos en los resultados del análisis. Aquí te dejo algunos métodos comunes para manejar los valores faltantes:

- **Eliminación**: Si la cantidad de valores faltantes en una columna o fila es demasiado grande, puede ser mejor eliminar esa columna o fila por completo. Sin embargo, este método puede resultar en la pérdida de información importante, por lo que se debe usar con precaución.
- **Imputación por la media, mediana o moda**: En este método, los valores faltantes de una columna son reemplazados por la media, mediana o moda de esa columna. Es una solución simple pero puede no ser la mejor opción si los datos son sesgados o si los valores faltantes no son aleatorios.
- **Imputación por regresión:** Este método implica utilizar las demás variables en los datos para predecir el valor faltante. Puede ser más preciso que la imputación por la media, mediana o moda, pero también puede ser más computacionalmente costoso.
- **Imputación múltiple**: Este método es similar a la imputación por regresión, pero en lugar de rellenar cada valor faltante con un solo valor, se rellena con un conjunto de valores posibles, lo que permite capturar la incertidumbre acerca del valor correcto a imputar.

- **Uso de algoritmos robustos a los valores faltantes**: Algunos algoritmos de machine learning pueden manejar los valores faltantes sin necesidad de imputación. Por ejemplo, los árboles de decisión pueden manejar los valores faltantes mediante el uso de nodos que permiten la bifurcación en caso de un valor faltante.

4.2.2. Análisis Exploratorio de Datos para clasificación

El Análisis Exploratorio de Datos (EDA por sus siglas en inglés) es un paso importante en el preprocesamiento de los datos para cualquier tipo de tarea de aprendizaje automático, incluida la clasificación. Esta fase implica examinar los datos de cerca para entender su estructura, relaciones, patrones y anomalías antes de proceder a la modelación. Aquí hay algunas técnicas comunes utilizadas en EDA:

04_ 2.Procesamiento. Preprocesado

1. **Resumen estadístico:** Describe las estadísticas básicas de los datos como la media, la mediana, el rango, la desviación estándar, etc.

```
import pandas as pd

# Supongamos que 'data' es nuestro DataFrame
data = pd.DataFrame({
   'edad': [25, 32, 35, 19, 45, 33, 42, 20, 28, 30],
   'horas_semana': [10, 9, 12, 15, 9, 7, 8, 11, 12, 9],
   'peliculas_mes': [5, 6, 7, 10, 6, 5, 6, 9, 7, 6]
})

print(data.describe())
```

```
            edad  horas_semana  peliculas_mes
count  10.000000     10.000000      10.000000
mean   30.900000     10.200000       6.700000
std     8.491172      2.347576       1.636392
min    19.000000      7.000000       5.000000
25%    25.750000      9.000000       6.000000
50%    31.000000      9.500000       6.000000
75%    34.500000     11.750000       7.000000
max    45.000000     15.000000      10.000000
```

En este resumen estadístico, obtenemos información útil sobre nuestros datos:

- **count** nos da el número total de entradas en cada columna.
- **mean** nos da el promedio de las entradas en cada columna.
- **std** nos da la desviación estándar, que indica cómo se distribuyen los datos alrededor de la media.
- **min** y **max** nos dan los valores mínimo y máximo respectivamente.
- **25%, 50% y 75% son los percentiles**, lo que significa que el 25%, el 50% (que es la mediana) y el 75% de las edades están por debajo de estos valores respectivamente.

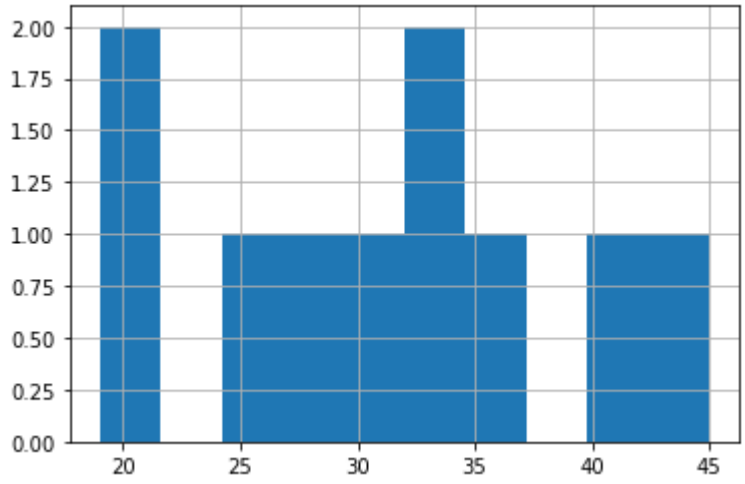
04_ 2.Procesamiento. Preprocesado

2. **Visualización de datos**: Histogramas, gráficos de barras, gráficos de dispersión, gráficos de caja, gráficos de correlación, etc., pueden ayudar a entender la distribución y las relaciones entre las variables.

Para esto, vamos a usar las bibliotecas Matplotlib y Seaborn, dos de las bibliotecas de visualización de datos más populares en Python.

Histogramas: Muestran la distribución de una variable numérica.

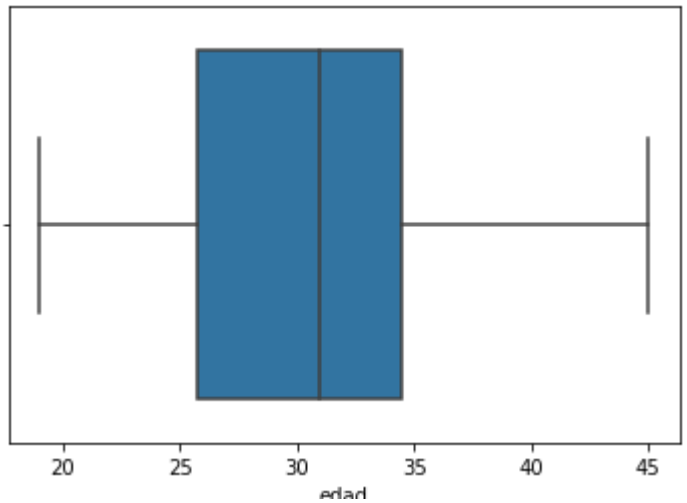
```
import matplotlib.pyplot as plt

data['edad'].hist(bins=10)
plt.show()
```



04_ 2.Procesamiento. Preprocesado

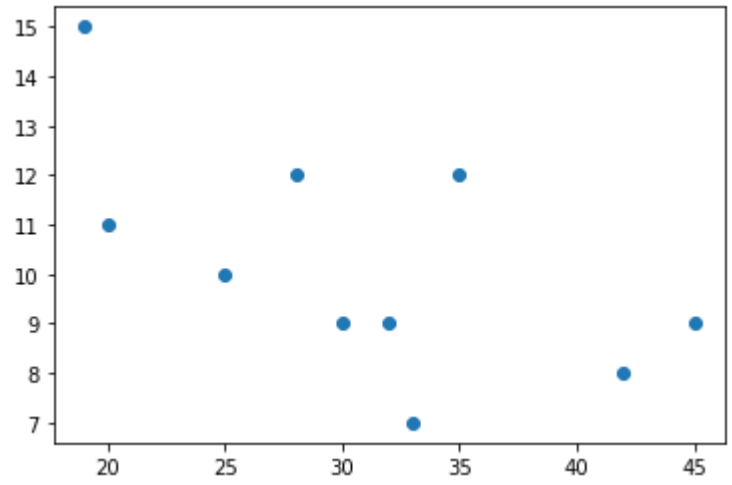
Diagramas de caja (box plots): Estos gráficos muestran la mediana, los cuartiles y los valores atípicos de una variable numérica.

```
import seaborn as sns

sns.boxplot(x=data['edad'])
plt.show()
```



Diagramas de dispersión (scatter plots): Estos gráficos muestran la relación entre dos variables numéricas.

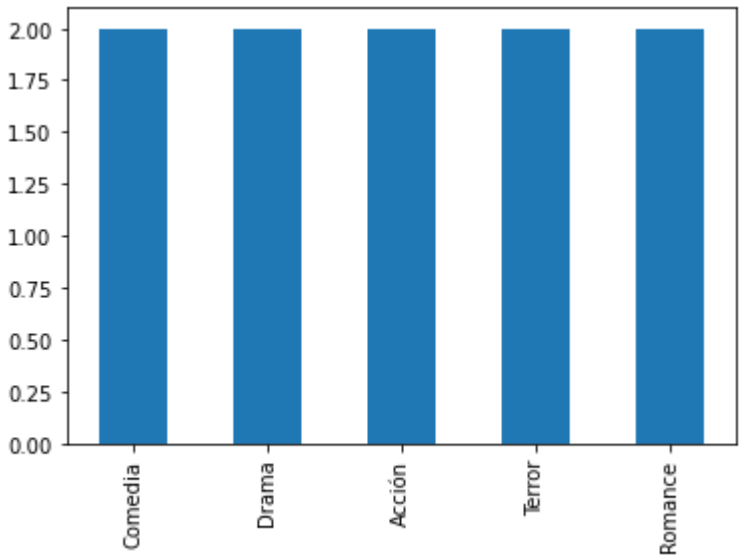
```
plt.scatter(data['edad'], data['horas_semana'])
plt.show()
```



04_ 2.Procesamiento. Preprocesado

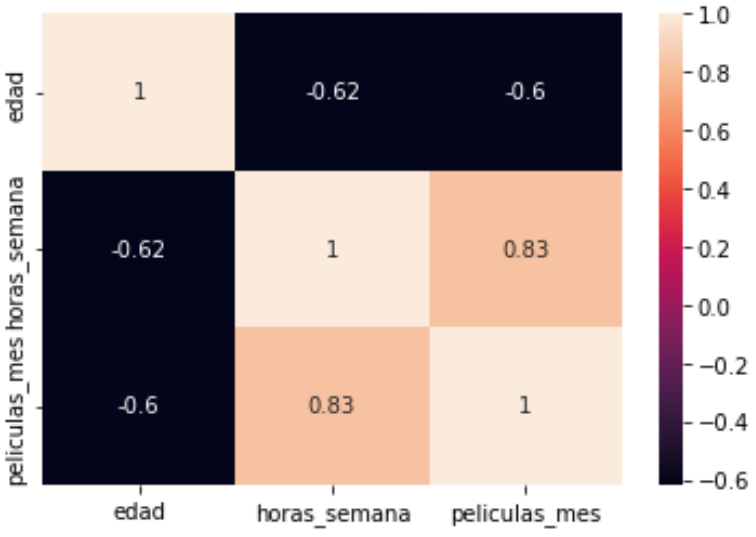
Diagramas de barras: Son útiles para comparar categorías de una variable categórica.

```
data['genero_preferido'] = ['Comedia', 'Drama', 'Acción', 'Terror', 'Comedia', 'Romance', 'Acción', 'Drama', 'Terror', 'Romance']

data['genero_preferido'].value_counts().plot(kind='bar')
plt.show()
```



Mapas de calor (heatmaps): Son útiles para visualizar la correlación entre las variables.

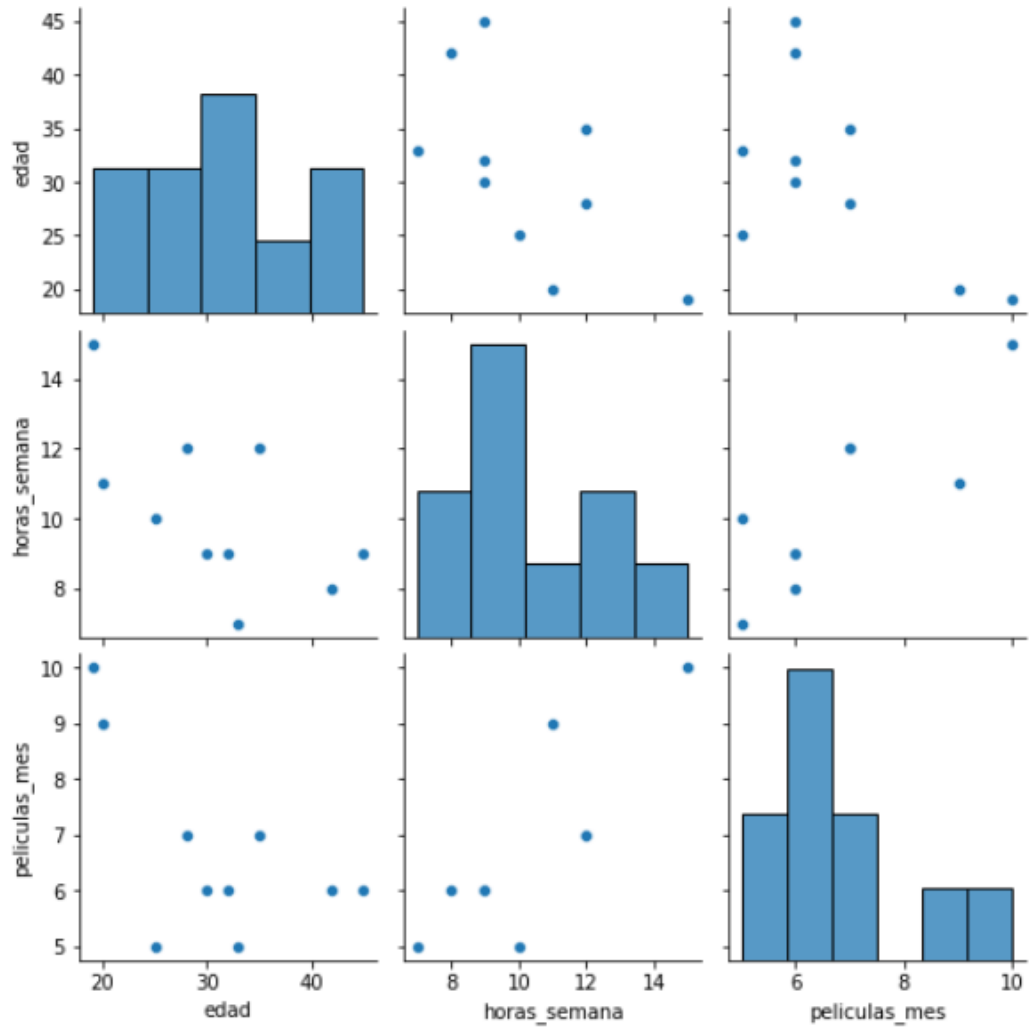
```
genre_mapping = {'Comedia': 0, 'Drama': 1, 'Acción': 2, 'Terror': 3, 'Romance': 4}  # Define the mapping for all genres

data['genero_preferido'] = data['genero_preferido'].map(genre_mapping)  # Convert the 'Genre' column to numeric representation

sns.heatmap(data.corr(), annot=True)
plt.show()
```



04_ 2.Procesamiento. Preprocesado

Pair plots: Un pair plot permite visualizar pares de variables en un formato de matriz de gráficos de dispersión.

```
sns.pairplot(data)
plt.show()
```



Es importante notar que la elección de gráficos depende en gran medida del tipo de datos y de las preguntas que estamos tratando de responder. Los gráficos mencionados anteriormente son sólo algunos ejemplos de los muchos tipos de visualizaciones disponibles.

04_ 2.Procesamiento. Preprocesado

3. **Análisis de variables categóricas**: En caso de clasificación, es útil entender la distribución de las clases. ¿Están las clases balanceadas? Si no, puede que necesitemos estrategias para manejar el desbalance de clases.

El análisis de variables categóricas implica el estudio de datos que pueden ser categorizados en grupos o categorías. Estas categorías pueden ser ordinales (con un orden específico, como "bajo", "medio", "alto") o nominales (sin un orden específico, como "rojo", "azul", "verde").

Aquí te dejo algunos ejemplos de cómo se podría analizar las variables categóricas:

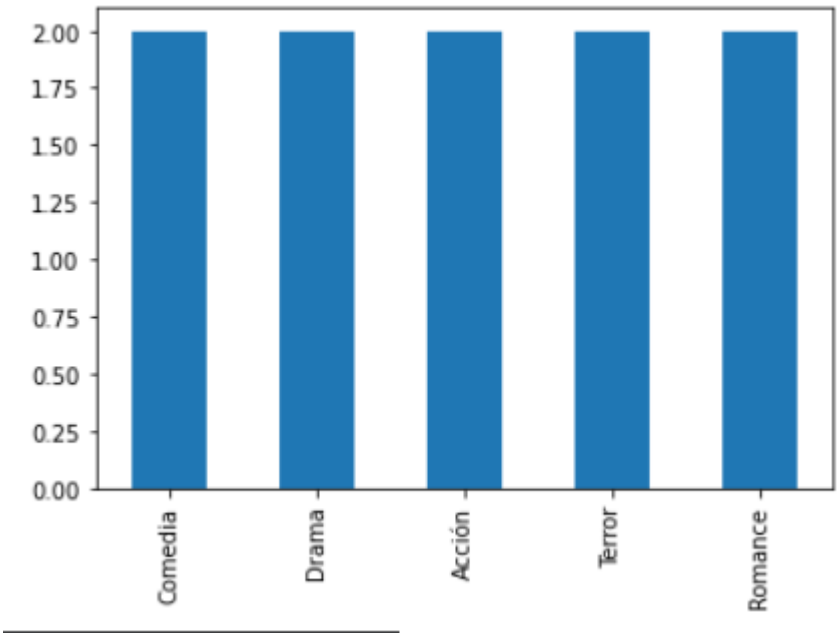
- **Frecuencia de las categorías**: Esto implica contar cuántas observaciones hay en cada categoría. En Python, se puede hacer usando el método value_counts().

```
data['genero_preferido'].value_counts()
```

```
Comedia    2
Drama      2
Acción     2
Terror     2
Romance    2
Name: genero_preferido, dtype: int64
```

- **Gráficos de barras**: Son una forma efectiva de visualizar las frecuencias de las categorías.

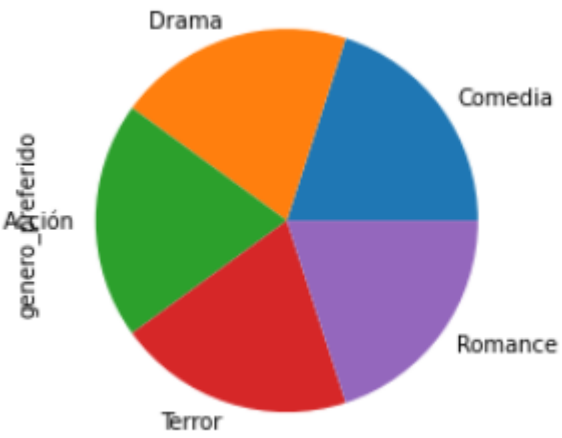
```
data['genero_preferido'].value_counts().plot(kind='bar')
```



04_ 2.Procesamiento. Preprocesado

- **Diagramas de tarta**: Otro método visual para mostrar las proporciones de las categorías.

```
data[‘genero_preferido’].value_counts().plot(kind=’pie’)
```



- **Tablas de contingencia**: Son útiles para entender la relación entre dos variables categóricas.

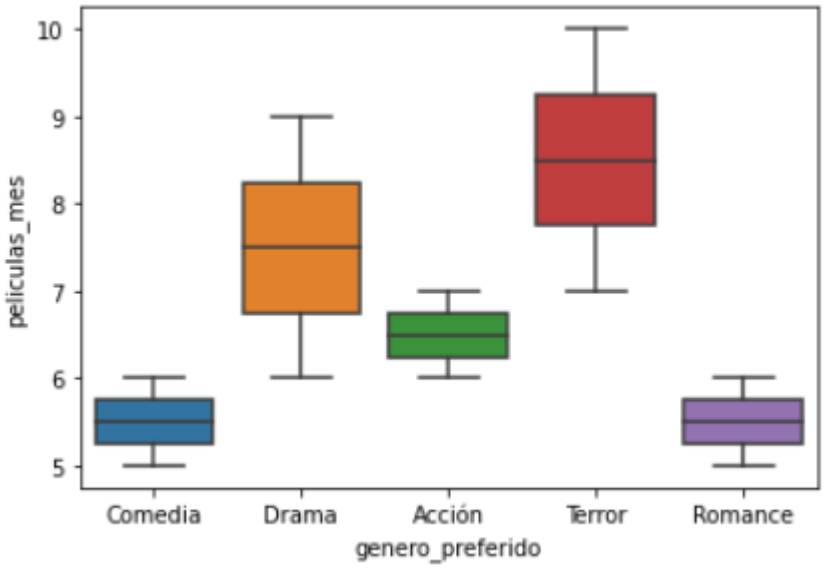
```
data['ciudad'] = ['Madrid', 'Barcelona', 'Valencia', 'Sevilla', 'Zaragoza', 'Málaga', 'Murcia', 'Palma', 'Bilbao', 'Alicante']

pd.crosstab(data['genero_preferido'], data['ciudad'])
```

ciudad	Alicante	Barcelona	Bilbao	Madrid	Murcia	Málaga	Palma	Sevilla	Valencia	Zaragoza
genero_preferido										
Acción	0	0	0	0	1	0	0	0	1	0
Comedia	0	0	0	1	0	0	0	0	0	1
Drama	0	1	0	0	0	0	1	0	0	0
Romance	1	0	0	0	0	1	0	0	0	0
Terror	0	0	1	0	0	0	0	1	0	0

04_ 2.Procesamiento. Preprocesado

- **Gráfico de cajas**: Para entender la relación entre una variable categórica y una variable numérica, un gráfico de cajas puede ser útil.

```
sns.boxplot(x=data['genero_preferido'], y=data['peliculas_mes'])
```



- **Codificación de variables categóricas**: Para usar variables categóricas en muchos algoritmos de aprendizaje automático, necesitamos codificarlas como números. Esto se puede hacer usando varias técnicas como la codificación one-hot, la codificación ordinal, entre otras.

```
pd.get_dummies(data['genero_preferido']) # One-hot encoding
```

Estas son solo algunas formas de analizar variables categóricas. La elección de la técnica dependerá del problema específico que estés tratando de resolver.

04_ 2.Procesamiento. Preprocesado

4. **Identificación de valores atípicos**: Los valores atípicos pueden tener un gran impacto en el rendimiento de los modelos de clasificación. Por lo tanto, debemos identificar y decidir cómo manejar estos valores.

Los valores atípicos, también conocidos como outliers, son puntos de datos que se desvían significativamente de los otros puntos de datos en el mismo conjunto. Pueden ser causados por errores de medición, errores en los datos, o pueden ser indicativos de una variación natural en el conjunto de datos. Los valores atípicos pueden tener un gran efecto en el análisis estadístico y los modelos de aprendizaje automático, y por lo tanto, es importante detectarlos y decidir cómo manejarlos.

Existen diversas técnicas para identificar los valores atípicos:

- **Método del rango intercuartílico (IQR):** El rango intercuartil es la medida de variabilidad, basada en la división de un conjunto de datos en cuartiles. Los cuartiles dividen un rango en cuatro partes iguales, o sea, el IQR es el rango entre el primer cuartil (25%) y el tercer cuartil (75%).
-

```
#añadimos dos registros con edad atípica (120, -15)
data.loc[len(data.index)] = [120, 8, 7, 'Acción', 'Barcelona']
data.loc[len(data.index)] = [-15, 8, 7, 'Acción', 'Madrid']
data.tail()
```

	edad	horas_semana	peliculas_mes	ciudad	genero_preferido
7	20	11	9	Palma	Drama
8	28	12	7	Bilbao	Terror
9	30	9	6	Alicante	Romance
10	120	8	7	Acción	Barcelona
11	-15	8	7	Acción	Madrid

04_ 2.Procesamiento. Preprocesado

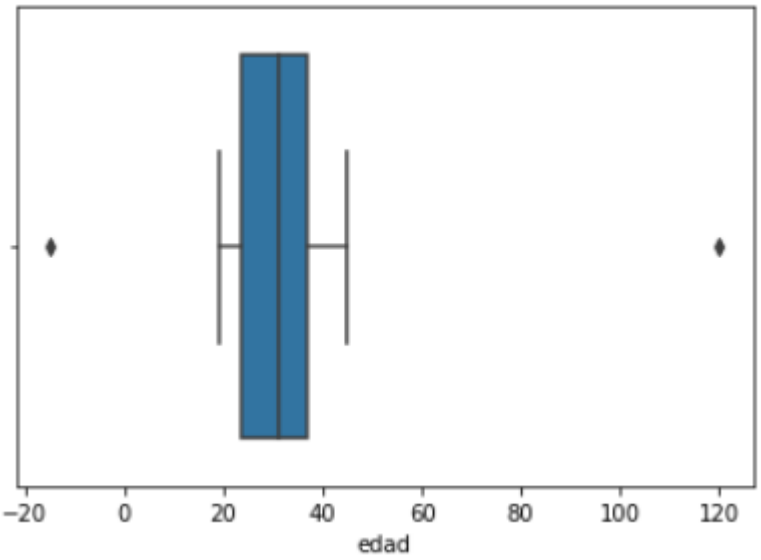
```
# Calcular IQR
Q1 = data['edad'].quantile(0.25)
Q3 = data['edad'].quantile(0.75)
IQR = Q3 - Q1

# Definir los límites para los valores atípicos
filtro = (data['edad'] >= Q1 - 1.5 * IQR) & (data['edad'] <= Q3 + 1.5 *IQR)

# Filtrar los valores que no son atípicos
data_filtrada = data.loc[filtro]

print("Datos originales:")
print(data)

print("\nDatos sin atípicos:")
print(data_filtrada)
```

```
Datos originales:
    edad  horas_semana  peliculas_mes     ciudad genero_preferido
0     25            10              5     Madrid          Comedia
1     32             9              6  Barcelona           Drama
2     35            12              7   Valencia          Acción
3     19            15             10    Sevilla          Terror
4     45             9              6   Zaragoza          Comedia
5     33             7              5     Málaga          Romance
6     42             8              6     Murcia          Acción
7     20            11              9      Palma           Drama
8     28            12              7     Bilbao          Terror
9     30             9              6   Alicante          Romance
10   120             8              7     Acción        Barcelona
11   -15             8              7     Acción           Madrid

Datos sin atípicos:
   edad  horas_semana  peliculas_mes     ciudad genero_preferido
0    25            10              5     Madrid          Comedia
1    32             9              6  Barcelona           Drama
2    35            12              7   Valencia          Acción
3    19            15             10    Sevilla          Terror
4    45             9              6   Zaragoza          Comedia
5    33             7              5     Málaga          Romance
6    42             8              6     Murcia          Acción
7    20            11              9      Palma           Drama
8    28            12              7     Bilbao          Terror
9    30             9              6   Alicante          Romance
```

04_ 2.Procesamiento. Preprocesado

- **Diagrama de caja (Box plot):** Este es un gráfico que se basa en los cuartiles y muestra la distribución de un conjunto de datos junto con los valores atípicos. Los puntos de datos que se encuentran fuera de las "bigotes" del gráfico son los valores atípicos.

```
import seaborn as sns

sns.boxplot(x=data['edad'])
```



- **Z-Score**: El valor Z, también conocido como puntuación Z o Z-score, es una medida estadística que describe la posición de una observación en relación con la media de un grupo de datos. Es una medida de cuántas desviaciones estándar está un dato del promedio.

  El valor Z se calcula utilizando la siguiente fórmula:

  $Z = (X - \mu) / \sigma$

  Donde:

  Z es el Z-score,
  X es el valor de la observación,
  μ es la media de la población o muestra,
  σ es la desviación estándar de la población o muestra.
  Un valor Z de 0 indica que la observación es igual a la media. Valores positivos de Z indican que la observación está por encima de la media, y valores negativos que está por debajo de la media.
-

04_ 2.Procesamiento. Preprocesado

En el contexto de la detección de valores atípicos, a menudo se considera que un valor es un atípico si su puntuación Z es superior a 3 o inferior a -3. Esto se basa en la regla empírica (o regla del 68-95-99.7) en la estadística, que dice que en una distribución normal, aproximadamente el 99.7% de los datos caerán dentro de tres desviaciones estándar de la media.

```
from scipy import stats
import numpy as np
#Añadimos un valor extemo
data.loc[len(data.index)] = [4000, 8, 7, 'Acción', 'Madrid']
z_scores = np.abs(stats.zscore(data['edad']))
outliers = data[(z_scores > 3)]
print(z_scores)
outliers
```

```
0     0.183573
1     0.175699
2     0.172325
3     0.190322
4     0.161076
5     0.174574
6     0.164451
7     0.189197
8     0.180198
9     0.177949
10    0.076714
11    0.228566
12    0.380419
13    0.380419
14    0.605386
15    0.256687
16    0.661627
17    0.238240
18    0.166700
19    4.287642
Name: edad, dtype: float64
```

	edad	horas_semana	peliculas_mes	genero_preferido	ciudad
19	4000	8	7	Acción	Madrid

04_ 2.Procesamiento. Preprocesado

5. **Verificación de la calidad de los datos**: Esto implica revisar si hay datos faltantes, inconsistencias, errores, duplicados, etc.

La verificación de la calidad de los datos es un paso crucial en cualquier proceso de aprendizaje automático o análisis de datos. Un conjunto de datos de alta calidad puede mejorar significativamente la capacidad de un modelo para aprender y hacer predicciones precisas.

Aquí están algunos pasos que puedes seguir para verificar la calidad de tus datos:

- **Verificar la consistencia de los datos:** Asegúrate de que los datos sean consistentes en todo el conjunto de datos. Por ejemplo, verifica que las unidades de medida sean las mismas en todo el conjunto de datos.

  Comprobaciones que puedes realizar para verificar la consistencia de tus datos:

  - Verificar la integridad referencial: Si tienes varias tablas relacionadas entre sí, debes asegurarte de que las claves primarias y foráneas están correctamente establecidas y que no hay valores faltantes en estas columnas.
  - Comprobar el rango de valores: Para ciertas columnas numéricas, puede haber un rango válido de valores. Por ejemplo, la edad de una persona no puede ser un número negativo. Puedes utilizar operaciones de enmascaramiento para verificar si todos los valores de una columna están dentro del rango deseado.
  - Verificar la unicidad: Algunas columnas pueden requerir valores únicos, como un identificador de usuario o un correo electrónico. Puedes verificar esto utilizando la función duplicated() en Pandas.
  - Verificar el formato: Algunas columnas pueden requerir un formato específico. Por ejemplo, una columna de fecha puede requerir que todas las fechas estén en el formato "AAAA-MM-DD".
  - Verificar la completitud: Verificar si hay valores faltantes en tus datos utilizando la función isnull() en Pandas.

```
# Comprobar el rango de valores
print("Valores de edad fuera de rango:", data['edad'].apply(lambda x: x < 0 or x > 120).any())
# Comprobar la unicidad
print("Ciudades duplicadas:", data['ciudad'].duplicated().any())
```

```
Valores de edad fuera de rango: True
Ciudades duplicadas: True
```

04_ 2.Procesamiento. Preprocesado

- **Verificar la precisión de los datos**: Asegúrate de que los datos sean precisos. Por ejemplo, si tienes datos sobre el peso de las personas, sería improbable que el peso sea negativo.

  Formas de verificar la precisión de los datos:

  - Comprobaciones de Lógica: Puedes realizar comprobaciones lógicas básicas. Por ejemplo, si tienes una columna que registra las fechas de nacimiento y otra que registra la edad de las personas, puedes verificar si ambas son consistentes.
  - Recolección de Datos Duplicados: Verifica si hay duplicados en tus datos. A veces, los mismos datos se pueden recopilar varias veces por error.
  - Comprobaciones contra una fuente autorizada: Si tienes acceso a una fuente de datos autorizada o confiable, puedes comparar tus datos con esta fuente para verificar la precisión.
  - Verificación por muestreo o revisión manual: Aunque puede ser una tarea muy laboriosa, la revisión manual de los datos puede proporcionar una forma de verificar la precisión de los datos, especialmente si la cantidad de datos no es muy grande.
  - Comprobaciones de rango: Verifica si los datos caen dentro de un rango esperado razonable. Por ejemplo, la altura humana normalmente no excede los 2,5 metros.
  - Validación de patrones: Si los datos deben seguir un cierto patrón o formato, puedes verificar su precisión comprobando este patrón. Por ejemplo, puedes verificar si los números de teléfono tienen la longitud correcta y solo contienen números.

  Es importante señalar que garantizar la precisión de los datos es un paso importante antes de cualquier análisis de datos o modelado de Machine Learning, ya que los errores en los datos pueden llevar a conclusiones erróneas o a un rendimiento deficiente del modelo.

04_ 2.Procesamiento. Preprocesado

- **Verificar la integridad de los datos**: Asegúrate de que los datos sean completos y no falten datos esenciales. Los datos faltantes pueden causar problemas a la hora de realizar el análisis o de entrenar el modelo.

  Formas de verificar la integridad de los datos:

  - Integridad referencial: Esta es una propiedad fundamental en las bases de datos relacionales y garantiza que las relaciones entre las tablas permanezcan consistentes. Específicamente, se asegura que cualquier valor de clave foránea coincida con un valor de clave primaria en la tabla a la que hace referencia.
  - Verificar la completitud: Esto implica verificar que no falten datos y que todos los registros esperados estén presentes. Por ejemplo, si estás recopilando datos de una encuesta, querrías asegurarte de que tienes respuestas para todas las encuestas enviadas.
  - Validaciones de rango, longitud y tipo de datos: Esto se refiere a verificar si los datos cumplen con ciertas restricciones, como estar dentro de un cierto rango, tener una cierta longitud o ser de un tipo específico.
  - Verificar la consistencia de datos: Esto implica verificar que los datos sean coherentes en diferentes lugares. Por ejemplo, si tienes la misma información en dos lugares diferentes, querrías asegurarte de que sean iguales.
  - Usar controles de suma: Esta es una técnica que puede usarse para verificar la integridad de los datos. Implica sumar los datos y comparar el resultado con un valor conocido o precalculado.
  - Usar hash de datos: Una función de hash puede utilizarse para garantizar que los datos no hayan sido alterados. Si los datos se alteran, la salida de la función de hash cambiará.

  Recordar que, al igual que con la precisión de los datos, garantizar la integridad de los datos es vital para obtener resultados de análisis válidos y de alta calidad.

04_ 2.Procesamiento. Preprocesado

- **Verificar la relevancia de los datos**: Asegúrate de que los datos sean relevantes para el problema que estás tratando de resolver. Por ejemplo, si estás tratando de predecir el precio de las viviendas, los datos sobre el clima podrían no ser relevantes.

  Consideraciones para verificar la relevancia de los datos:

  - Relación con la pregunta de investigación o los objetivos de negocio: Los datos que recopilas y utilizas deben ser relevantes para tu pregunta de investigación o los objetivos de tu negocio. Si estás construyendo un modelo de predicción para prever ventas futuras, por ejemplo, necesitarás datos históricos de ventas, precios, información sobre productos, etc.
  - Pertinencia temporal: Los datos deben ser relevantes para el período de tiempo que estás estudiando. Si estás analizando tendencias recientes, los datos de hace diez años pueden no ser relevantes.
  - Pertinencia geográfica: Si estás analizando datos a nivel de una ciudad o país específico, asegúrate de que los datos sean relevantes para esa ubicación geográfica.
  - Pertinencia demográfica: Si estás estudiando un grupo demográfico específico, necesitarás datos que sean relevantes para ese grupo. Por ejemplo, si estás investigando comportamientos de compra de millennials, necesitarás datos específicos sobre ese grupo demográfico.
  - Completitud: Asegúrate de tener todos los datos que necesitas para responder a tu pregunta de investigación o cumplir con tu objetivo de negocio. Si te faltan datos relevantes, es posible que necesites recopilar más datos o reconsiderar tu enfoque.

  Al verificar la relevancia de los datos, estarás un paso más cerca de asegurarte de que tus resultados sean válidos y significativos.

04_ 2.Procesamiento. Preprocesado

- **Verificar la unicidad de los datos**: Asegúrate de que no hay duplicados en tus datos. Los datos duplicados pueden causar problemas y sesgos en el análisis y el modelo.

```
#introducimos valores repetidos
data.loc[len(data.index)] = [35, 7, 7, 'Terror', 'Soria']
data.loc[len(data.index)] = [35, 7, 7, 'Terror', 'Soria']

data.loc[len(data.index)] = [30, 5, 5, 'Drama', 'Soria']
data.loc[len(data.index)] = [30, 5, 5, 'Drama', 'Soria']


#introducimos valores faltantes
data.loc[len(data.index)] = [np.nan, 7, 7, 'Drama','Soria']
data.loc[len(data.index)] = [44, np.nan, 7, 'Drama', 'Soria']

# Verificar duplicados
print("Número de filas duplicadas: ", data.duplicated().sum())
# Verificar datos faltantes
print("Número de valores faltantes: ", data.isnull().sum().sum())
# Resumen estadístico para verificar precisión y consistencia
data.describe()
```

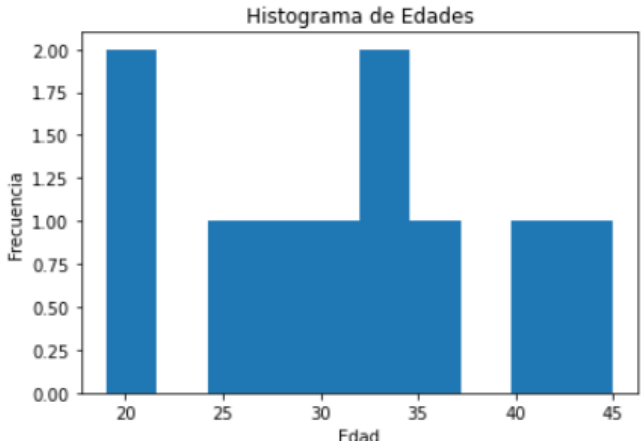
```
Número de filas duplicadas:  2
Número de valores faltantes:  2
```

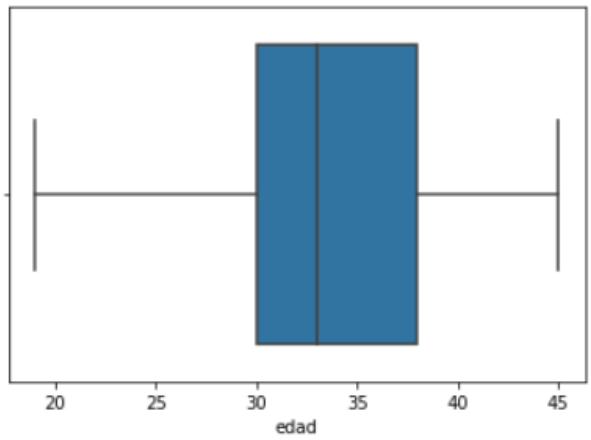
	edad	horas_semana	peliculas_mes
count	20.000000	20.000000	21.000000
mean	233.500000	8.550000	6.666667
std	886.861498	2.459675	1.238278
min	-15.000000	5.000000	5.000000
25%	29.500000	7.000000	6.000000
50%	34.000000	8.000000	7.000000
75%	42.500000	9.250000	7.000000
max	4000.000000	15.000000	10.000000

04_ 2.Procesamiento. Preprocesado

- **Verificar la distribución de los datos**: La distribución de los datos puede proporcionar una gran cantidad de información sobre la calidad de los datos. Los histogramas y los gráficos de caja son dos herramientas útiles para visualizar la distribución de los datos.

```
import seaborn as sns

plt.hist(data_filtrada['edad'])

plt.xlabel('Edad')

plt.ylabel('Frecuencia')

plt.title('Histograma de Edades')

plt.show()# Boxplot

sns.boxplot(x=data['columna'])
```





Estos son solo algunos ejemplos de cómo puedes verificar la calidad de tus datos. Dependiendo de tus necesidades y del tipo de datos que estés manejando, es posible que debas realizar más comprobaciones o diferentes tipos de comprobaciones.

04_ 2.Procesamiento. Preprocesado

6. **Correlación y análisis de independencia**: Es importante entender cómo las características se correlacionan entre sí y con la variable objetivo.

La correlación y el análisis de independencia son dos conceptos fundamentales en estadística y ciencia de datos. Ambos se utilizan para entender las relaciones entre variables.

Correlación: Mide la relación lineal entre dos variables. Una correlación positiva significa que las variables tienden a aumentar o disminuir juntas, mientras que una correlación negativa significa que cuando una variable aumenta, la otra tiende a disminuir.

El coeficiente de correlación de Pearson es una medida común de la correlación lineal. Varía entre -1 y +1, donde -1 indica una correlación negativa perfecta, +1 indica una correlación positiva perfecta, y 0 indica ninguna correlación.

Aquí hay un ejemplo de cómo calcular la correlación en Python usando pandas:

```
data_filtrada= data_filtrada.drop('ciudad', axis=1)



correlation_matrix = data_filtrada.corr()

print(correlation_matrix)
```

```
                     edad  horas_semana  peliculas_mes  genero_preferido
edad             1.000000     -0.617604      -0.602140         -0.166782
horas_semana    -0.617604      1.000000       0.827212          0.031750
peliculas_mes   -0.602140      0.827212       1.000000          0.091098
genero_preferido -0.166782     0.031750       0.091098          1.000000
```

La variable edad tiene una correlación negativa moderada con horas_semana (-0.618) y peliculas_mes (-0.602). Esto sugiere que a medida que la edad aumenta, tiende a haber una disminución en las horas semanales dedicadas a ver películas y en la cantidad de películas vistas al mes.

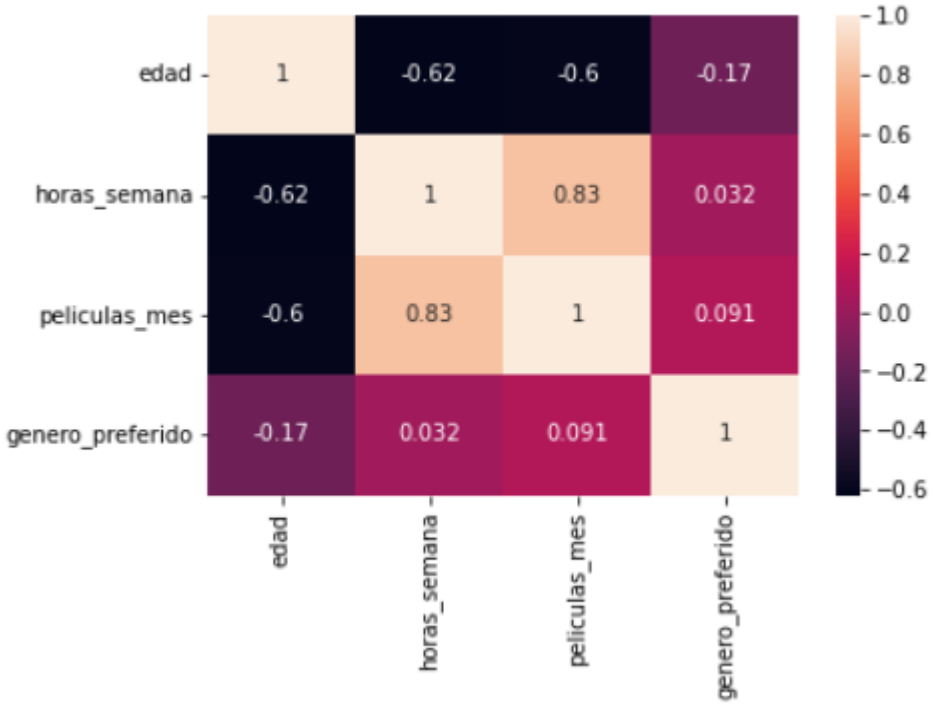
La variable horas_semana muestra una correlación positiva fuerte con peliculas_mes (0.827). Esto indica que a medida que aumentan las horas semanales dedicadas a ver películas, también tiende a aumentar la cantidad de películas vistas al mes.

La variable genero_preferido tiene una correlación débil positiva con peliculas_mes (0.091). Esto sugiere que hay una ligera tendencia de que el género preferido influya ligeramente en la cantidad de películas vistas al mes.

04_ 2.Procesamiento. Preprocesado

Esta función devuelve una matriz de correlación, que puedes visualizar con un mapa de calor:

```
import seaborn as sns

sns.heatmap(correlation_matrix, annot=True)
```



04_ 2.Procesamiento. Preprocesado

Análisis de independencia: Se refiere a la falta de relación entre dos variables. Dos variables son independientes si el cambio en una no afecta al cambio en la otra.

En términos de estadística, si dos variables son independientes, entonces la probabilidad conjunta de ellas es el producto de sus probabilidades individuales.

Una forma común de verificar la independencia de dos variables categóricas es utilizando el test de Chi-cuadrado.

Aquí está cómo hacerlo en Python usando scipy:

```
from scipy.stats import chi2_contingency
df = data_filtrada
contingency_table = pd.crosstab(df['edad'], df['horas_semana'])
chi2, p, dof, expected = chi2_contingency(contingency_table)
p
```

```
0.2673366001622652
```

Si el valor de p es inferior a 0.05 (o cualquier otro umbral que hayas establecido), puedes rechazar la hipótesis nula de que las variables son independientes.

Recuerda que la correlación no implica causalidad. Aunque dos variables pueden estar correlacionadas, eso no significa necesariamente que una cause la otra.

Es importante tener en cuenta que el análisis exploratorio de datos es un proceso iterativo y creativo que puede involucrar la adición de nuevas variables, la transformación de variables existentes, la eliminación de variables innecesarias, etc.

4.2.3. Manejo de datos faltantes

Es importante tener en cuenta que no existe una única solución correcta para el manejo de los valores faltantes. La mejor opción dependerá de la naturaleza de tus datos y del problema específico que estés tratando de resolver.

Corrección de errores de formato: Los datos a menudo vienen en diferentes formatos y pueden requerir cierta cantidad de limpieza para ser útiles. Por ejemplo, puede ser necesario convertir las fechas a un formato estándar, o los números pueden necesitar ser convertidos de cadenas de texto a números.

Formas en las que puedes corregir errores de formato en tus datos:

- **Corrigiendo valores inconsistentes**: Los valores inconsistentes son un problema común en los datos. Por ejemplo, la misma entidad puede representarse de diferentes formas en diferentes filas de los datos (por ejemplo, "EE. UU.", "USA", "Estados Unidos" pueden referirse al mismo país). Una solución es estandarizar estos valores utilizando un formato uniforme.
- **Corrigiendo formatos de fecha y hora**: Las fechas y horas pueden representarse de muchas maneras diferentes, lo que puede crear inconsistencias en tus datos. Puedes utilizar bibliotecas como pandas en Python para convertir estas fechas y horas a un formato estándar.
- **Corrigiendo tipos de datos incorrectos**: A veces, los datos pueden tener el tipo incorrecto. Por ejemplo, un número podría estar representado como una cadena de texto. Puedes utilizar funciones en pandas como to_numeric() o astype() para convertir estos datos al tipo correcto.
- **Corrigiendo datos mal codificados:** A veces, los datos pueden estar mal codificados, lo que puede causar que los caracteres no se muestren correctamente. Puedes corregir esto cambiando la codificación de los datos.

Por ejemplo, si tienes un DataFrame de pandas df y quieres corregir el tipo de dato de una columna, podrías hacerlo así:

```
# Convertir la columna 'edad' a tipo entero

df['edad'] = df['edad'].astype(int)
```

Siempre es importante explorar y entender tus datos antes de hacer estas correcciones, ya que cada conjunto de datos tendrá sus propios problemas únicos.

04_ 2.Procesamiento. Preprocesado

Identificación y manejo de outliers: Los outliers pueden distorsionar el modelo de aprendizaje automático y producir resultados poco confiables. Identificar y manejar correctamente los outliers es un paso importante en la limpieza de datos.

El manejo de outliers puede hacerse de varias formas:

- **Eliminación**: Si estás seguro de que los outliers son errores en los datos o si son muy pocos, puedes optar por eliminar estas filas de tu conjunto de datos.
- **Imputación**: Si piensas que los outliers podrían ser el resultado de errores de medición, puedes optar por imputar estos valores con la media, la mediana, el modo o utilizar algún método de imputación más sofisticado.
- **Transformación**: Algunas veces, aplicar una transformación a los datos puede hacer que los outliers sean menos extremos. Algunas transformaciones comunes incluyen transformaciones logarítmicas, cuadradas, raíz cuadrada, etc.

Recuerda, el tratamiento de los outliers debe hacerse con precaución, ya que los outliers legítimos pueden ser datos valiosos y eliminarlos podría resultar en la pérdida de información.

04_ 2.Procesamiento. Preprocesado

Vamos a hacer un ejemplo utilizando el método Z-Score y el método del rango intercuartílico (IQR) con la biblioteca pandas de Python.

```
import pandas as pd
from scipy import stats
import numpy as np

# Supongamos que 'df' es nuestro DataFrame
df = pd.DataFrame({
   'edad': [25, 32, 35, 19, 45, 33, 42, 20, 28, 130],  # Hay un outlier (130)
})

# Z-Score
df['z_score'] = np.abs(stats.zscore(df['edad']))
outliers_z_score = df[df['z_score'] > 3]
print('Outliers según Z-Score:')
print(outliers_z_score)

# IQR
Q1 = df['edad'].quantile(0.25)
Q3 = df['edad'].quantile(0.75)
IQR = Q3 - Q1
outliers_iqr = df[(df['edad'] < (Q1 - 1.5 * IQR)) | (df['edad'] > (Q3 + 1.5 * IQR))]
print('\nOutliers según IQR:')
print(outliers_iqr)

# Puedes manejar los outliers ahora. Por ejemplo, puedes decidir eliminar las filas con outliers:
df_clean_z_score = df[df['z_score'] <= 3]
df_clean_iqr = df[(df['edad'] >= (Q1 - 1.5 * IQR)) & (df['edad'] <= (Q3 + 1.5 * IQR))]

print('\nDataFrame limpio (Z-Score):')
print(df_clean_z_score)

print('\nDataFrame limpio (IQR):')
print(df_clean_iqr)
```

En este ejemplo, estamos creando un DataFrame simple con una columna de edades, donde 130 es claramente un valor atípico.

Para el método Z-Score, calculamos el Z-Score de cada edad y luego seleccionamos las edades donde el Z-Score es mayor que 3 como outliers.

04_ 2.Procesamiento. Preprocesado

Para el método IQR, calculamos el primer y tercer cuartil, así como el rango intercuartílico, y luego seleccionamos las edades que están por debajo de Q1 - 1.5IQR o por encima de Q3 + 1.5IQR como outliers.

Finalmente, eliminamos las filas con outliers para obtener un DataFrame "limpio".

Este es solo un ejemplo, en la práctica, deberías pensar cuidadosamente antes de decidir cómo manejar los outliers.

04_ 2.Procesamiento. Preprocesado

Eliminación de duplicados: Los registros duplicados pueden sesgar los análisis, por lo que es importante eliminarlos.

```
import pandas as pd

# Creamos un DataFrame con algunos datos duplicados
df = pd.DataFrame({
   'Nombre': ['Ana', 'Luis', 'Juan', 'Ana', 'Luis', 'Juan', 'Ana'],
   'Edad': [24, 30, 29, 24, 30, 29, 25],
})

print("DataFrame original:\n")
print(df)

# Usamos drop_duplicates para eliminar duplicados. Por defecto, considera todas las columnas.
df_sin_duplicados = df.drop_duplicates()

print("\nDataFrame sin duplicados:\n")
print(df_sin_duplicados)
```

En este ejemplo, la función drop_duplicates() elimina las filas que son completamente iguales. Si quieres considerar solo algunas columnas para identificar duplicados, puedes pasar esas columnas como una lista al método, por ejemplo, df.drop_duplicates(['Nombre']) eliminará todas las filas donde el nombre sea igual, independientemente de las otras columnas.

Si quieres mantener la última aparición de los datos duplicados en lugar de la primera (que es el comportamiento por defecto), puedes pasar el argumento keep='last' al método, por ejemplo, df.drop_duplicates(keep='last').

04_ 2.Procesamiento. Preprocesado

Normalización: En algunos casos, es útil normalizar los datos para que tengan un rango específico o una media y una varianza estándar. Esto es especialmente útil para algunos modelos de aprendizaje automático.

Hay varios métodos de normalización, entre los que se incluyen:

- **Escalado Min-Max**: Este método de escalado lleva todos los valores en el rango entre 0 y 1. La fórmula general para un valor min-max de un punto x es:

- X_norm = (X - X_min) / (X_max - X_min)

- **Escalado Z-score o estandarización**: Esta técnica considera la distribución de los datos y transforma los datos en términos de la desviación estándar. La fórmula general para la estandarización de un punto x es:

- X_standard = (X - mean(X)) / std_dev(X)

- **Escalado por la norma L1:** En este método, los datos se escalan de tal manera que la suma absoluta de cada fila sea 1. Es útil cuando se desea que los valores de la fila sumen 1.

- **Escalado por la norma L2**: Similar al escalado L1, pero en lugar de que la suma sea 1, la suma de los cuadrados de los valores será 1.

04_ 2.Procesamiento. Preprocesado

Aquí tienes un ejemplo de cómo normalizar los datos en pandas:

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Creamos un DataFrame
df = pd.DataFrame({
   'A': [14.00, 90.20, 90.95, 96.27, 91.21],
   'B': [103.02, 107.26, 110.35, 114.23, 114.68],
   'C': ['rojo', 'verde', 'azul', 'amarillo', 'blanco']
})

# Creamos un scaler
scaler = MinMaxScaler()

# Ajustamos y transformamos las columnas numéricas
df[['A', 'B']] = scaler.fit_transform(df[['A', 'B']])

print(df)
```

En este ejemplo, usamos la clase MinMaxScaler de la biblioteca scikit-learn para normalizar las columnas 'A' y 'B' de nuestro DataFrame entre 0 y 1.

04_ 2.Procesamiento. Preprocesado

Codificación de variables categóricas: Muchos modelos de aprendizaje automático requieren que las entradas sean numéricas. Si tienes variables categóricas en tus datos, puede ser necesario codificarlas como variables dummy (también conocidas como variables binarias) para que puedan ser usadas en estos modelos.

Recuerda que la limpieza de datos es un proceso que depende mucho de los datos específicos con los que estés trabajando y del problema que estés intentando resolver. Por lo tanto, estos son solo algunos ejemplos de los tipos de tareas que podrías realizar durante este proceso.

Ejemplos

Supongamos que tenemos un DataFrame llamado df con valores faltantes.

```
import pandas as pd
import numpy as np

# Crear un DataFrame con valores faltantes
data = {'A': [1, 2, np.nan], 'B': [5, np.nan, np.nan], 'C': [1, 2, 3]}
df = pd.DataFrame(data)

print(df)
```

```
    A    B    C
0  1.0  5.0    1
1  2.0  NaN    2
2  NaN  NaN    3
```

04_ 2.Procesamiento. Preprocesado

1. **Eliminación de valores faltantes:**

Puedes eliminar las filas con valores faltantes usando el método dropna().

```
df.dropna()
```

2. **Imputación por la media, mediana o moda:**

Puedes reemplazar los valores faltantes con la media, mediana o moda de la columna correspondiente usando el método fillna().

```
# Reemplazar con la media
df.fillna(value=df.mean())
```

```
# Reemplazar con la mediana
df.fillna(value=df.median())
# Reemplazar con la moda
df.fillna(value=df.mode().iloc[0])
```

3. **Imputación por regresión:**

Para un ejemplo simple de imputación por regresión, puedes usar SimpleImputer de sklearn.impute con strategy='constant'.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=0)
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

4. **Imputación múltiple:**

sklearn.impute tiene una clase IterativeImputer que puede ser utilizada para imputación múltiple.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imp = IterativeImputer(max_iter=10, random_state=0)
df_imputed = pd.DataFrame(imp.fit_transform(df), columns=df.columns)
```

4.2.4. Las variables Dummy

Las variables dummy o variables indicadoras son variables binarias creadas para representar una variable con dos o más categorías. Son una herramienta muy útil en análisis de regresión, ya que permiten incluir variables categóricas en modelos que requieren que las variables sean numéricas.

El proceso para crear variables dummy a partir de una variable categórica consiste en crear una nueva variable binaria para cada categoría de la variable categórica. Por ejemplo, si tuviéramos una variable "color" con las categorías "rojo", "verde" y "azul", crearíamos tres nuevas variables: "color_rojo", "color_verde" y "color_azul". Cada una de estas variables tomaría el valor 1 cuando el color correspondiente sea el correcto y 0 en cualquier otro caso.

En Python, esto se puede hacer fácilmente usando la función get_dummies() de pandas. Por ejemplo:

```
import pandas as pd

# Creamos un DataFrame con una variable categórica
df = pd.DataFrame({
    'color': ['rojo', 'verde', 'azul', 'verde', 'rojo', 'azul', 'azul']
})

# Creamos las variables dummy para la variable 'color'
df_dummies = pd.get_dummies(df, columns=['color'])

print(df_dummies)
```

En este ejemplo, la función get_dummies() crea tres nuevas variables ('color_azul', 'color_rojo' y 'color_verde') y elimina la variable original 'color'.