Manipulación de datos con pandas

Maya tiene hojas de cálculo enormes y desordenadas, y necesita limpiarlas y analizarlas.

Ben le presenta **Pandas**, una biblioteca de Python que facilita la manipulación y análisis de datos.

Pandas se compara con una **navaja suiza** o una **lupa de detective de datos**, que ayuda a descubrir patrones ocultos.

Ejemplos de lo que Pandas puede hacer:

- Filtrar datos (ej. ventas de diciembre).
- Calcular métricas (ej. gasto promedio por cliente).
- Detectar tendencias y patrones.
- Crear **visualizaciones claras y atractivas**.

Aunque Maya es novata, Ben le asegura que Pandas es fácil de aprender: como un baile con pocos pasos.

Idea central: **Los datos son arcilla, y Pandas es la herramienta para moldearlos en algo útil y valioso**

Dataframes de pandas: Conceptos básicos

Qué es un DataFrame

- Un **DataFrame** es una estructura bidimensional de Pandas, similar a una hoja de cálculo, organizada en **filas** (observaciones) y **columnas** (variables).
- Puede contener distintos tipos de datos: números, texto, fechas u objetos complejos.
- Componentes clave:

1. **Datos**: la información real almacenada.
2. **Índice**: identifica de forma única cada fila, facilitando el acceso eficiente.
3. **Columnas**: representan variables, cada una con un nombre y tipo de datos.

Operaciones básicas con DataFrames

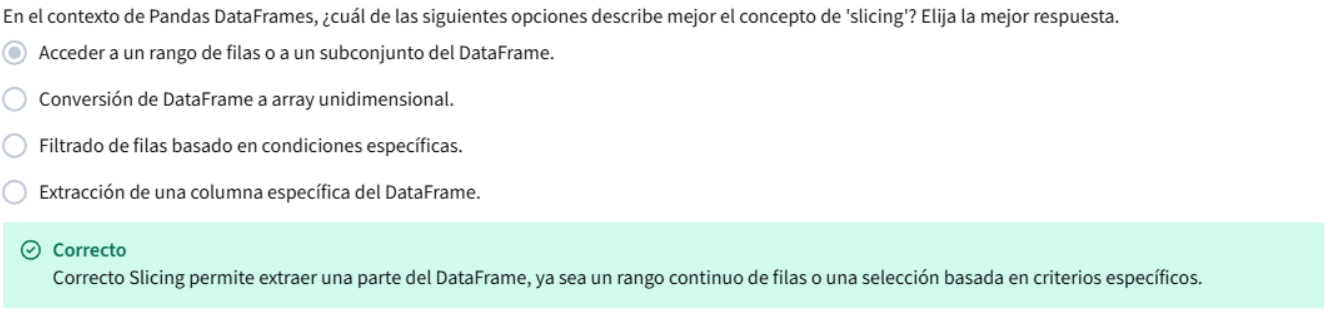
1. **Indexación**
    - Permite acceder a elementos específicos por etiqueta (`.loc`) o por posición (`.iloc`).
    - Útil para recuperar o actualizar datos concretos rápidamente.
    - Ejemplo: obtener detalles de un pedido específico en un DataFrame de ventas.
2. **Segmentación (slicing)**
    - Selecciona un rango de filas o columnas.
    - Ideal para extraer subconjuntos de datos para análisis o visualización.
    - Ejemplo: aislar los precios de acciones de un año específico de un DataFrame financiero.
3. **Filtrado**
    - Selecciona filas que cumplan condiciones específicas.
    - Muy útil para limpieza de datos o centrarse en valores relevantes.
    - Ejemplo: identificar clientes con baja satisfacción en una encuesta.

Conclusión

- Estas tres operaciones son esenciales para manipular y analizar datos de manera eficiente en Pandas.

- Con la práctica, se pueden dominar muchas más funciones avanzadas para trabajar con datos complejos.
- Clave: **experimentar, explorar y aplicar Pandas a problemas del mundo real** para ganar seguridad y fluidez.

Pregunta



Indexación en Pandas

La **indexación** en Pandas permite localizar y recuperar datos dentro de un DataFrame de manera rápida y precisa. Es esencial para manejar grandes conjuntos de datos, realizar filtrados, agregaciones y manipulaciones complejas de manera eficiente.

Tipos principales de indexación

1. **Indexación basada en etiquetas: `.loc`**
    - Accede a datos usando las **etiquetas de filas y columnas**.
    - Muy útil cuando los nombres de filas o columnas son significativos.
    - Puede trabajar con **una etiqueta** o un **rango de etiquetas**.

Ejemplo:

```
df.loc['customer_123', 'purchase_amount']
```

- Esto obtiene el importe de la compra del cliente con ID `'customer_123'`.

2. **Indexación basada en posición: `.iloc`**

- Accede a datos usando **posiciones enteras** (comenzando desde 0).
- Ideal cuando las etiquetas no son intuitivas o faltan.
- También admite **cortes** para seleccionar rangos.

Ejemplo:

```
df.iloc[5, 2]
```

- Esto obtiene el valor de la tercera columna de la sexta fila.

Indexación booleana (basada en máscaras)

- Filtra filas que cumplen condiciones específicas usando valores **True/False**.
- Permite combinar múltiples condiciones con operadores lógicos: `&` (y), `|` (o), `~` (no).

Ejemplo: seleccionar clientes con membresía Platino y más de 10 compras:

```
df[(df['membership_level'] == 'Platinum') &
(df['number_of_purchases'] > 10)]
```

- Esto devuelve un DataFrame con solo los clientes que cumplen ambas condiciones.

Resumen de uso

- `.loc`: acceso **por etiquetas** → intuitivo y legible.
- `.iloc`: acceso **por posición** → flexible cuando no hay etiquetas útiles.
- **Indexación booleana**: filtra datos según **condiciones lógicas**, ideal para análisis complejo.

Otros métodos de indexación en Pandas

1. **Acceso rápido a valores individuales: `.at` y `.iat`**

- `.at`: basado en **etiquetas**.
- `.iat`: basado en **posiciones enteras**.
- Optimizados para **velocidad**, útiles cuando solo se necesita leer o modificar un único valor.

Ejemplo:

```
df.at['customer_123', 'membership_level'] = 'Gold'
```

2. **Establecer y modificar valores**
    - Permite actualizar valores existentes o asignar nuevos en ubicaciones específicas.
    - Facilita corregir errores, imputar datos faltantes o crear nuevas columnas directamente.
3. **Indexación multinivel (jerárquica)**
    - Para datos con jerarquía natural (ej. ventas por año → mes → producto).
    - Permite crear múltiples niveles de etiquetas en filas o columnas.
    - Facilita **agrupaciones, agregaciones y pivotaciones** a través de múltiples niveles.
4. **Método `query()`**
    - Sintaxis similar a **SQL** para consultas en DataFrames.
    - Alternativa más legible a la indexación booleana, especialmente para condiciones complejas.

Ejemplo:

```
df.query("membership_level == 'Platinum' & number_of_purchases > 10")
```

Casos de uso prácticos

Análisis de ventas

Calcular ventas totales de un producto en una región específica:

```
total_sales = df.loc[(df['Product'] == 'Product A') & (df['Region'] == 'North')]['Sales'].sum()
```

1.

Extracción de información de clientes

Obtener datos de contacto de un cliente:

```
customer_info = df.loc[df['Name'] == 'John Doe', ['Email', 'Phone']]
```

2.

Mejores prácticas de indexación

- **Elegir el método adecuado**:
    - `.loc` → por etiquetas.
    - `.iloc` → por posición.
    - Indexación booleana → filtrar por condiciones.
    - `.at` / `.iat` → acceder rápidamente a valores individuales.
- **Encadenamiento de operaciones**:
  Permite escribir código más conciso y eficiente evitando variables intermedias.
- **Priorizar la claridad sobre la complejidad**:
  Mantener el código legible y documentado.
- **Gestionar datos faltantes**:
  Usar `.fillna()` o `.dropna()` para evitar errores y mantener resultados fiables.
- **Aprovechar la indexación multinivel**:
  Organiza datos jerárquicos de manera intuitiva y facilita operaciones complejas.

Conclusión

La indexación en Pandas es **clave para el análisis de datos**. Combinando `.loc`, `.iloc`, `.at`, `.iat`, indexación booleana, multinivel y `query()`, puedes **navegar, seleccionar y filtrar datos eficientemente**, desbloqueando información valiosa y mejorando la toma de decisiones basada en datos.

Demostración: Carga e inspección de conjuntos de datos con pandas (contiene código)

Introducción a Pandas

- **Pandas** es una biblioteca fundamental de Python para **análisis y manipulación de datos**.
- Permite **cargar, limpiar, transformar y analizar datos** de manera eficiente, ya sea desde CSV, hojas de cálculo o bases de datos SQL.
- Es especialmente útil cuando los datos son **desordenados, incompletos o complejos**.

Conceptos clave

- Los datos se organizan en **filas** (observaciones) y **columnas** (atributos o características).
- Cada columna puede considerarse una **dimensión**, útil para análisis multidimensional.
- Pandas organiza estos datos en una **estructura llamada DataFrame**, similar a una hoja de cálculo pero mucho más potente.

Primeros pasos: cargar y explorar datos

Importar Pandas

```
import pandas as pd
```

1.

Cargar un archivo CSV

```
df = pd.read_csv('cars.csv')
```

2.
    - `df` es la variable que contiene el DataFrame.
    - `read_csv` convierte los datos del archivo en un formato estructurado listo para analizar.

3. **Exploración inicial**

Visualizar las primeras filas:

```
df.head()
```

-

Información general del DataFrame:

```
df.info()
```

- Muestra nombres de columnas, tipos de datos y valores faltantes.

Estadísticas descriptivas de columnas numéricas:

```
df.describe()
```

- Proporciona recuento, media, desviación estándar y cuartiles.

Conclusión

- Cargar datos y explorarlos con `head()`, `info()` y `describe()` son los **primeros pasos esenciales** en cualquier análisis de datos con Pandas.

- Pandas ofrece muchas otras funciones para **limpiar, transformar y visualizar datos**, desbloqueando su verdadero potencial a medida que se explora más profundamente.

Exploración de las transformaciones de datos

Transformaciones de datos en Pandas

Pandas permite **manipular y transformar datos** para extraer información valiosa y tomar decisiones informadas. Las transformaciones comunes incluyen **fusión, concatenación, ordenación, agrupación y agregación**.

1. Combinar conjuntos de datos

- **Fusión (`pd.merge()`)**: combina DataFrames basándose en columnas o índices comunes. **JOINS DE SQL**
    - Tipos de fusión: **inner, outer, left, right**.
    - Ejemplo: unir datos demográficos con preferencias de producto.
- **Concatenación (`pd.concat()`)**: apila DataFrames por filas o columnas.
    - Se puede especificar el eje y cómo manejar los índices.

2. Ordenar datos

- `df.sort_values()`: ordena un DataFrame por una o más columnas.
    - Se puede elegir **ascendente o descendente**.
    - Ejemplos: ordenar comentarios por fecha o características de diseño por popularidad.

3. Agrupación y agregación

- `df.groupby()`: divide los datos en grupos según una o varias columnas.
- **Agregación (`df.agg()`)**: aplica funciones de cálculo sobre grupos o columnas.
    - Funciones comunes: **mean, sum, count, min, max**.
    - Ejemplo: calcular el tiempo promedio de permanencia de usuarios por grupo de edad.

Buenas prácticas

1. **Definir objetivos claros** antes de transformar los datos.
2. **Documentar los pasos** para reproducir resultados.
3. **Validar resultados** para detectar errores temprano.
4. **Experimentar e iterar** para dominar las transformaciones y explorar nuevas perspectivas.

Conclusión

- Las transformaciones de datos son esenciales para **analizar, descubrir patrones y validar algoritmos**.
- Dominar funciones como `merge`, `concat`, `sort_values`, `groupby` y `agg` permite a los desarrolladores de Python **extraer insights significativos y tomar decisiones basadas en datos**.

Pregunta

¿Cuál describe mejor el concepto de agrupación en el contexto de las transformaciones de datos utilizando pandas? Seleccione la mejor respuesta.

- (●) Dividir los datos en grupos basados en columnas específicas y luego realizar cálculos en estos grupos.
- ( ) Combinación de dos o más marcos de datos basados en columnas o índices comunes.
- ( ) Disposición de los datos en un orden específico para identificar patrones y valores atípicos.
- ( ) Apilar varios marcos de datos uno encima de otro, ya sea por filas o por columnas.

✓ Correcto
Correcto La agrupación consiste en dividir los datos en subconjuntos basados en características compartidas, lo que permite realizar análisis y cálculos posteriores dentro de cada grupo.

Pandas: Hoja de trucos esencial

1. Creación de DataFrames

Un **DataFrame** es una estructura de datos de Pandas que organiza información en **filas y columnas**, similar a una hoja de cálculo.

Formas comunes de crear DataFrames:

1. **Desde diccionarios**

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 28]}
df = pd.DataFrame(data)
```

- Claves → nombres de columnas
- Valores → datos de cada columna

2. **Desde listas de listas**

```
data = [['Alice', 25], ['Bob', 30], ['Charlie', 28]]
df = pd.DataFrame(data, columns=['Name', 'Age'])
```

- Cada lista interior → una fila
- Argumento `columns` → nombres de columnas

3. **Desde archivos CSV**

```
df = pd.read_csv('data.csv')
```

- Pandas también puede importar datos de Excel (`read_excel`), JSON (`read_json`) o SQL (`read_sql`).

2. Inspección de DataFrames

Pandas ofrece funciones para **explorar y entender tus datos**:

- `df.head()` → primeras 5 filas (por defecto)
- `df.tail()` → últimas 5 filas (por defecto)
- `df.shape` → dimensiones del DataFrame `(filas, columnas)`
- `df.info()` → resumen conciso: nombres de columnas, tipos de datos, valores faltantes
- `df.describe()` → estadísticas descriptivas de columnas numéricas: recuento, media, desviación estándar, mínimos, máximos y cuartiles

Conclusión

- Los DataFrames son la **herramienta central de Pandas** para organizar datos.
- Con Pandas, puedes **crear, importar y explorar datos** de manera eficiente antes de analizarlos o transformarlos.

1. Selección de datos

- **Columnas:**

Una columna → devuelve una **Serie**:

```
df['Age']
```

  -

Varias columnas → devuelve un **DataFrame**:

```
df[['Name', 'Age']]
```

  -
- **Filas:**

Por etiquetas (`.loc`):

```
df.loc[0]  # fila con etiqueta 0
```

-

Por posición (`.iloc`):

```
df.iloc[0]  # primera fila
```

-

2. Filtrado de datos

Indexación booleana: filtra filas según condiciones:

```
df[df['Age'] > 25]
```

-

Múltiples condiciones con `.query()` (sintaxis tipo SQL):

```
df.query('Age > 25 and Name == "Bob"')
```

-

3. Gestión de datos faltantes

Detectar valores faltantes (`NaN`):

```
df.isnull()          # todo el DataFrame
df['Age'].isnull()   # columna específica
```

-

Eliminar filas con valores faltantes:

```
df.dropna()
```

-

Rellenar valores faltantes:

```
df.fillna(0)                                # con un valor fijo
```

```
df['Age'].fillna(df['Age'].mean(), inplace=True)  # con la media de la columna
```

-
- Nota: imputar con la media conserva la media de la columna, pero puede subestimar la varianza. Existen otras técnicas más sofisticadas como imputación por mediana o regresión.

Conclusión

- Pandas permite **seleccionar, filtrar y gestionar datos faltantes** de manera eficiente.
- La práctica con conjuntos de datos reales es clave para dominar estas herramientas y extraer información valiosa.

Demostración: Transformaciones de datos con pandas

Manipulación de datos con Pandas

1. Importación y preparación de datos

- Importa Pandas y carga tus datos en un **DataFrame**:

```
import pandas as pd
df = pd.read_csv('archivo.csv')
```

- `DataFrame` → similar a una hoja de cálculo organizada, diseñada para análisis de datos.
- **`df.head()`** → vista rápida de las primeras filas para inspeccionar el contenido y estructura.

2. Transformaciones comunes

1. **Ordenar datos (`sort_values`)**
    - Permite resaltar los valores más importantes:

```
df.sort_values('Ventas', ascending=False)
```

2.
3. **Agrupar datos (`groupby`)**
    - Agrupa por columnas y aplica agregaciones:

```
df.groupby('Región')['Ventas'].sum()
```

4.
5. **Agregación (`agg`)**
    - Calcula métricas como promedio, suma, máximo, mínimo:

```
df['Ventas'].mean()
```

6.
7. **Filtrar datos**
    - Selecciona subconjuntos según condiciones:

```
df[df['Ventas'] > 1000]
```

8.
9. **Aplicar funciones personalizadas (`apply`)**
    - Modifica o transforma columnas con funciones:

```
df['Ventas_descuento'] = df['Ventas'].apply(lambda x: x*0.9)
```

10.
11. **Pivotar datos (`pivot`)**
    - Reorganiza filas y columnas para comparar categorías:

```
df.pivot(index='Región', columns='Producto', values='Ventas')
```

12.

3. Manejo de valores faltantes

- **Rellenar valores faltantes (`fillna`)**:

```
df.fillna(0)
df['Ventas'].fillna(df['Ventas'].mean(), inplace=True)
```

- **Eliminar filas con valores faltantes (`dropna`)**:

```
df.dropna()
```

4. Combinar DataFrames

- **Concatenar (`concat`)** → apila DataFrames vertical u horizontalmente:

```
pd.concat([df1, df2])
```

- **Fusionar (`merge`)** → combina DataFrames según columnas compartidas:

```
pd.merge(df1, df2, on='ID')
```

Conclusión

- Pandas es una herramienta **indispensable para manipulación de datos en Python**.
- Permite **ordenar, agrupar, agregar, filtrar, aplicar funciones, pivotar y manejar valores faltantes**.
- Combina DataFrames fácilmente mediante **concatenación y fusión**.
- La práctica constante es clave para dominar todas estas transformaciones y extraer información valiosa de tus datos.

Transforming Data with Pandas

```
# Import the necessary packages:
import pandas as pd

# Extract the csv file for this assay:
df = pd.read_csv("sales_data.csv")
print(df.head())
```

```
         Date    Product Region  Sales
0  2024-09-01  Widget A  North   1500
1  2024-09-01  Widget B  South   1200
2  2024-09-02  Widget A  North   1600
3  2024-09-02  Widget C   East   1100
4  2024-09-03  Widget B  South   1300
```

Sort the DataFrame

```
df_sorted = df.sort_values(by="Sales", ascending=False)
print(df_sorted.head())
```

```
         Date    Product Region  Sales
2  2024-09-02  Widget A  North   1600
8  2024-09-05  Widget A  South   1550
0  2024-09-01  Widget A  North   1500
5  2024-09-03  Widget A   West   1400
4  2024-09-03  Widget B  South   1300
```

Group the DataFrame

```
df_grouped = df.groupby("Region")["Sales"].sum()
print(df_grouped)
```

```
Region
East     2250
North    4350
South    4050
West     2600
Name: Sales, dtype: int64
```

Aggregate the DataFrame

```
mean_sales = df["Sales"].mean()
print(mean_sales)
```

```
1325.0
```

Filter the DataFrame

```
df_filtered = df[df["Sales"] > 1000]
print(df_filtered.head())
```

```
         Date   Product Region  Sales
0  2024-09-01  Widget A  North   1500
1  2024-09-01  Widget B  South   1200
2  2024-09-02  Widget A  North   1600
3  2024-09-02  Widget C   East   1100
4  2024-09-03  Widget B  South   1300
```

Discount the DataFrame

```
def calculate_discount(sales):
  return sales * 0.9

df["Discounted_Sales"] = df["Sales"].apply(calculate_discount)
print(df.head())
```

```
         Date   Product Region  Sales  Discounted_Sales
0  2024-09-01  Widget A  North   1500            1350.0
1  2024-09-01  Widget B  South   1200            1080.0
2  2024-09-02  Widget A  North   1600            1440.0
3  2024-09-02  Widget C   East   1100             990.0
4  2024-09-03  Widget B  South   1300            1170.0
```

Pivot the DataFrame

```
df_pivot = df.pivot_table(index="Region", columns="Product", values="Sales")
print(df_pivot)
```

```
Product  Widget A  Widget B  Widget C
Region
East          NaN       NaN    1125.0
North      1550.0    1250.0       NaN
South      1550.0    1250.0       NaN
West       1400.0       NaN    1200.0
```

DataFrame with Missing Values

```
import numpy as np

# Sample DataFrame with missing values (replace with your actual data)
df_with_missing_values = pd.DataFrame({'A': [1, 2, np.nan], 'B': [5, np.nan, np.nan], 'C': [1, 2, 3]})

print(df_with_missing_values)
```

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

Impute the Missing Values

```
df_filled = df_with_missing_values.fillna(0)
print(df_filled)
```

```
     A    B  C
0  1.0  5.0  1
1  2.0  0.0  2
2  0.0  0.0  3
```

Drop Rows with Missing Values

```
df_dropped = df_with_missing_values.dropna()
print(df_dropped)
```

```
     A    B  C
0  1.0  5.0  1
```

Concatenate the Data

```
#Combining DataFrames is a common necessity when working with multiple data sources.

# Sample DataFrames (replace with your actual data)
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

df_concatenated = pd.concat([df1, df2])
print(df_concatenated)
```

```
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
```

Merging the DataFrames

```
# Sample DataFrames with a common column (replace with your actual data)
df1 = pd.DataFrame({'key': ['K0', 'K1', 'K2'], 'A': ['A0', 'A1', 'A2'], 'B': ['B0', 'B1', 'B2']})
df2 = pd.DataFrame({'key': ['K0', 'K1', 'K3'], 'C': ['C0', 'C1', 'C3'], 'D': ['D0', 'D1', 'D3']})

df_merged = pd.merge(df1, df2, on='key')
print(df_merged)
```

```
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
```

Pregunta

Escenario: Usted tiene un DataFrame de pandas llamado datos_ventas [SAY: Sales Data] que contiene columnas para "Producto," "Región," y "Ventas." Quiere identificar el producto más vendido en cada región.

Pregunta Stem: ¿Qué operaciones de Pandas utilizaría para conseguirlo? Seleccione la mejor respuesta.

- ○ Agrupe el DataFrame por "Producto" y calcule la suma de "Ventas" de cada producto.
- ○ Ordena el DataFrame por "Ventas" en orden descendente.
- ◉ Agrupe el DataFrame por "Región" y luego ordene cada grupo por "Ventas" en orden descendente.
- ○ Filtra el DataFrame para incluir sólo las filas con el valor más alto de "Ventas".

✓ Correcto
Correcto Esta es la forma más eficaz de identificar el producto más vendido en cada región. La agrupación permite analizar las ventas dentro de cada región, y la clasificación ayuda a identificar el producto más vendido.

Actividad: Carga e inspección de datos con pandas

Paso 1: Cargar el conjunto de datos en un DataFrame de pandas

El primer paso consiste en utilizar pandas, una potente librería de Python diseñada para trabajar con datos. Piense en las bibliotecas como colecciones de código pre-escrito que proporcionan herramientas y funciones útiles, ahorrándole tener que escribir todo desde cero. Comenzará importando la librería pandas y cargando la información de los clientes desde un Archivo CSV a un DataFrame pandas, un formato estructurado que facilita la manipulación y el análisis eficiente de los datos. Piense en un DataFrame como una tabla de gran alcance, similar a una hoja de cálculo, que hace que sea fácil de organizar, manipular y analizar sus datos en Python.

- Importar `pandas` con el alias `pd`
- Utilice `.read_csv()` para cargar el archivo 'datos_cliente_50.csv' en un DataFrame llamado `customer_data`.
- Ejecute la celda.

Paso 2: Compruebe las dimensiones de los datos

Ahora que ya tiene sus datos cargados, es hora de obtener una visión rápida de su tamaño. Utilizarás el atributo `.shape` para determinar el número de filas y columnas de tu DataFrame, proporcionando una idea de la escala del conjunto de datos.

- Se ha proporcionado el código para imprimir la forma del DataFrame.
- Ejecute la celda para ver el número de filas y columnas de sus datos.

Paso 3: Vista Previa de los Datos - Obtenga un Vistazo Rápido de los Datos

A continuación, echará un vistazo a las primeras filas de su DataFrame utilizando la función `df.head()`. Las funciones son bloques de código reutilizables que realizan tareas específicas. En este caso, la función `df.head()` mostrará las cinco primeras filas, permitiéndole ver los nombres de las columnas y los tipos de valores que contienen. Este primer vistazo le ayudará a familiarizarse con la estructura y el contenido de los datos.

- Utilizando una sentencia `print`, imprima el DataFrame `customer_data` utilizando `df.head()`.
- Ejecute la celda y observe la salida para familiarizarse con sus datos.

Paso 4: Obtenga un Resumen de los Datos

En este paso, profundizará un poco más en la estructura de sus datos utilizando otra función, `df.info()`. Esta función proporciona un resumen conciso de cada columna, incluyendo su nombre, el tipo de datos que contiene (por ejemplo, números, texto) y si falta algún valor. Esta información es crucial para comprender cómo trabajar con cada columna de forma eficaz y detectar posibles problemas de calidad de los datos.

- Utilizando una sentencia `print`, imprima el DataFrame `customer_data` utilizando `df.info()`.
- Ejecute la celda para ver la información detallada sobre las columnas de su DataFrame.

Paso 5: Resumir datos numéricos

Para las columnas que contienen datos numéricos, puede obtener rápidamente estadísticas esenciales como la media, el mínimo, el máximo, etc. La función `df.describe()` genera automáticamente estos estadísticos descriptivos, ofreciendo una valiosa información sobre la distribución y las tendencias centrales de sus datos numéricos.

- Utilizando una sentencia `print`, imprima el DataFrame `customer_data` utilizando `df.describe()`.
- Ejecute la celda para mostrar los estadísticos de resumen.

Paso 6: Resumir Datos numéricos

En este paso, se centrará en la columna `['age']` y calculará dos métricas clave: la edad media y la mediana de edad de sus clientes. Para ello, utilizará las funciones `.mean()` y `.median()`, respectivamente, aplicadas a la columna `['age']`. Estas métricas proporcionan diferentes perspectivas sobre la edad típica dentro de su base de clientes. `['age']`: Los corchetes `[]` se utilizan para la selección de columnas en pandas. Dentro de los corchetes, se especifica el nombre de la columna que se desea extraer, que en este caso es 'edad'.

- Utilice la función `.mean()` en la columna `['age']` del DataFrame `customer_data`. Los corchetes `[]` se utilizan para la selección de columnas en pandas. Dentro de los corchetes se especifica el nombre de la columna que se quiere extraer, que en este caso es 'edad'. Guarde los resultados en la variable `mean_age`, que se le ha proporcionado.
- Utilice la función `.median()` en la columna `['age']` del DataFrame `customer_data`. Los corchetes `[]` se utilizan para la selección de columnas en pandas. Dentro de los corchetes, especifique el nombre de la columna que desea extraer, que en este caso es 'edad'. Guarde los resultados en la variable `median_age`, que se le ha proporcionado.
- Ejecuta la celda para ver la edad media y mediana de tus clientes.

1. ¿Cuál de las siguientes funciones se utiliza para cargar datos de un Archivo CSV en un DataFrame de Pandas? Seleccione la mejor respuesta.

- ○ `pd.load_csv()`
- ◉ `pd.read_csv()`
- ○ `pd.read_table()`
- ○ `df.load_csv()`

Correcto
Correcto Esta es la función principal para cargar datos CSV en un DataFrame.

2. ¿Qué indica el atributo `.shape` de un DataFrame? Seleccione la mejor respuesta.

- ○ Número total de valores del DataFrame
- ○ Tipos de datos de cada columna
- ◉ El número de filas y columnas
- ○ El uso de memoria del DataFrame

Correcto
Correcto Devuelve una tupla con el número de filas primero, y luego el número de columnas.

3. ¿Qué función proporciona un resumen conciso de un DataFrame, incluidos los nombres de las columnas, los tipos de datos y el recuento de valores no nulos? Seleccione la mejor respuesta.

- (•) `df.info()`
- ( ) `df.head()`
- ( ) `df.describe()`
- ( ) `df.summary()`

Correcto
Correcto Esta es la función que le da una visión estructural de su DataFrame

4. ¿Qué hace la función `df.describe()`? Seleccione la mejor respuesta.

- ( ) Calcula la media y la mediana de todas las columnas
- (•) Genera estadísticas descriptivas para las columnas numéricas del DataFrame
- ( ) Imprime los nombres de las columnas y sus tipos de datos
- ( ) Muestra las 5 primeras filas del DataFrame

Correcto
Correcto Esta función proporciona un rápido resumen estadístico de sus datos numéricos

Cuestionario: Pandas, tu centro de manipulación de datos

1. Acaba de cargar un conjunto de datos en un DataFrame de pandas llamado `df`. Desea obtener una visión rápida del DataFrame, incluyendo los nombres de las columnas, los tipos de datos de cada columna y el número de valores no nulos. ¿Qué función de Pandas utilizarías para este propósito? Seleccione la mejor respuesta.

   - ○ `describe()`
   - ◉ `info()`
   - ○ `shape`
   - ○ `head()`

   Correcto
   Correcto La función `info()` proporciona un resumen conciso del DataFrame, incluidos los nombres de las columnas, los tipos de datos y los valores no nulos.

2. Usted tiene dos Pandas DataFrames, `df1` y `df2`, que contienen información relacionada sobre los clientes. Necesita combinar estos DataFrames en un único DataFrame basado en una columna común llamada 'CustomerID'. ¿Qué función de Pandas utilizarías para esta operación? Seleccione la mejor respuesta.

   - ○ `sort_values()`
   - ◉ `merge()`
   - ○ `groupby()`
   - ○ `concat()`

   Correcto
   Correcto La función `merge()` es la herramienta adecuada para combinar DataFrames basados en una o más columnas o índices comunes, que es exactamente lo que necesita hacer en este escenario utilizando la columna 'CustomerID'.

3. Estás trabajando con un DataFrame en Pandas y necesitas realizar algunas operaciones básicas sobre él. ¿Cuál de las siguientes operaciones puede realizar utilizando la indexación y segmentación básica de DataFrame? Seleccione todo lo que corresponda.

- [ ] Eliminación de filas en función de una condición
- [x] Acceso a un valor específico mediante etiquetas de filas y columnas

Correcto
Correcto Puede utilizar loc o at para acceder a valores específicos basados en etiquetas.

- [x] Modificación de los valores de determinadas celdas

Correcto
Correcto Puede cambiar los valores de celdas específicas utilizando .loc o .iloc .

- [x] Selección de un rango de filas y columnas

Correcto
Correcto Tanto iloc como loc pueden utilizarse para seleccionar rangos de filas y columnas.

4. Usted tiene un DataFrame que contiene datos de clientes, incluyendo sus nombres, edades e historial de compras. Quiere ordenar este DataFrame alfabéticamente por los apellidos de los clientes. ¿Qué método de Pandas utilizaría para realizar esta ordenación? Seleccione la mejor respuesta.

- (•) ordenar_valores()
- ( ) concat()
- ( ) groupby()
- ( ) sort_index()

Correcto
¡Correcto! El método sort_values() se utiliza para ordenar un DataFrame por sus columnas en Pandas.

5. Está trabajando con un conjunto de datos que contiene datos de ventas de una tienda minorista. Quiere calcular las ventas totales de cada categoría de producto. ¿Cuál de las siguientes funciones de Pandas puede utilizar para realizar esta agregación? Seleccione todas las que correspondan.

- [ ] concat()
- [x] agregar()

Correcto
Correcto La función aggregate() se utiliza para aplicar funciones de agregación sobre datos agrupados.

- [x] groupby()

Correcto
Correcto La función groupby() se utiliza para agrupar datos y aplicar funciones de agregación.

- [ ] fusionar()

6. Estás trabajando con un DataFrame en Pandas y necesitas extraer las tres primeras filas para su posterior análisis. ¿Cómo lo conseguiría utilizando el método iloc, que se basa en la indexación de enteros? Seleccione la mejor respuesta.

- ( ) df.iloc[1:3]
- ( ) df.iloc[3]
- ( ) df.iloc[0:2]
- (•) df.iloc[:3]

Correcto
Correcto! df.iloc[:3] selecciona las tres primeras filas utilizando indexación basada en posiciones

7. Trabaja con un gran conjunto de datos y necesita realizar operaciones complejas como filtrar, ordenar y agregar datos. Quiere elegir una herramienta que esté específicamente diseñada para la manipulación y el análisis eficiente de datos. ¿Por qué elegirías Pandas en lugar de las listas y diccionarios tradicionales de Python para esta tarea? Seleccione la mejor respuesta.

- (●) Es más rápido y eficaz para las operaciones de datos a gran escala.
- ( ) Es la única biblioteca disponible para la manipulación de datos en Python.
- ( ) Genera automáticamente visualizaciones.
- ( ) Su sintaxis es más sencilla.

Correcto
Correcto! Pandas está optimizado para operaciones de datos a gran escala, por lo que es más rápido y eficiente que las listas y diccionarios tradicionales de Python.

8. Está trabajando con un gran conjunto de datos en Pandas y desea inspeccionar rápidamente las primeras filas para obtener una comprensión inicial de su estructura y contenido. ¿Qué función utilizarías para mostrar estas filas? Seleccione la mejor respuesta.

- (●) cabeza()
- ( ) describir()
- ( ) info()
- ( ) primero()

Correcto
Correcto La función head() se utiliza para mostrar las primeras filas de un DataFrame.

Análisis exploratorio de datos (EDA)

Sarah y su compañero se sienten abrumados por la cantidad de datos —demográficos, ventas y análisis web— sin saber cómo interpretarlos. Entonces, deciden **enfocar el análisis como si fueran detectives**:

- En lugar de ver solo números, buscarán **patrones, anomalías y relaciones ocultas** dentro de los datos.
- Este enfoque se llama **Análisis Exploratorio de Datos (EDA)**, el **primer paso de toda investigación de datos**, donde se explora y se comprende el conjunto antes de sacar conclusiones.

Usarán **Python como su kit de herramientas de detective**, con **bibliotecas como Pandas y Matplotlib**, para:

- **Visualizar los datos** mediante gráficos.
- **Detectar tendencias, valores atípicos y correlaciones.**
- **Descubrir la historia oculta detrás de los números.**

Finalmente, entienden que el EDA no trata solo de cálculos, sino de **interpretar y revelar los secretos que los datos esconden**, como resolver un misterio o una búsqueda del tesoro.

La importancia de la limpieza de datos

La **limpieza de datos** es un paso esencial en el desarrollo con Python, ya que permite transformar datos desordenados, incompletos o inexactos en información **estructurada, clara y confiable**.

Por qué es crucial

- Los **datos sin limpiar** pueden generar errores, decisiones equivocadas y pérdida de recursos.
- Los **datos limpios** proporcionan una base sólida para análisis precisos y decisiones informadas.
- Ayudan a **identificar patrones, tendencias e ineficiencias**, optimizando el rendimiento del código y las aplicaciones.

Beneficios principales

1. **Mayor precisión y fiabilidad** en los análisis al corregir o imputar valores faltantes.
2. **Identificación y resolución de problemas** en el rendimiento o la lógica de las aplicaciones.
3. **Desarrollo de funciones más efectivas**, alineadas con el comportamiento real de los usuarios.
4. **Toma de decisiones basada en datos**, desde la planificación hasta la implementación.

5. **Mejor comunicación y colaboración** en el equipo al trabajar con datos consistentes y confiables.

Buenas prácticas para limpiar datos

1. **Conocer los datos**: entender su origen, tipo y limitaciones.
2. **Gestionar valores faltantes**: eliminarlos, imputarlos o mantenerlos según el caso.
3. **Corregir errores e inconsistencias**: tipografías, formatos y duplicados.
4. **Transformar y agregar datos**: crear variables nuevas, agrupar o resumir información.
5. **Validar y verificar**: asegurarse de que los datos finales sean precisos, completos y coherentes.

En resumen, **la limpieza de datos no es solo una tarea técnica**, sino una etapa estratégica que **garantiza la calidad del desarrollo, la fiabilidad del análisis y la eficacia de las aplicaciones en Python**.

Pregunta

¿Cuál es la principal ventaja de utilizar datos limpios en el desarrollo con Python? Selecciona la mejor respuesta.

- ○ Corrige automáticamente los errores del código.
- ◉ Permite la Toma de decisiones basada en datos a lo largo de todo el proceso de desarrollo.
- ○ Elimina la necesidad de seguir analizando los datos.
- ○ Garantiza la optimización del código.

Correcto
¡Correcto! El vídeo destaca cómo los datos limpios informan las decisiones desde el desarrollo hasta la implantación, garantizando que las soluciones se basan en datos precisos y fiables.

Tácticas esenciales para la manipulación de datos

La **manipulación de datos** es una habilidad clave para los desarrolladores de Python, ya que permite transformar datos crudos y desordenados en información estructurada y útil. Bibliotecas como **Pandas**, **NumPy** y **Scikit-learn** son herramientas fundamentales, siendo

Pandas la más usada gracias a sus estructuras **DataFrame** (tablas) y **Series** (columnas individuales).

1. Limpieza y preprocesamiento de datos

Los datos reales suelen ser imperfectos: contienen **valores faltantes, incoherencias, duplicados y valores atípicos**.

- **Valores faltantes:**
  - `fillna()` → rellenar con la media, mediana u otro valor.
  - `dropna()` → eliminar filas o columnas con datos faltantes.
- **Incoherencias:**
  - Diferentes formatos de fecha, unidades o mayúsculas.
  - `astype()` → convertir tipos de datos.
  - Expresiones regulares → estandarizar formatos.
- **Duplicados:**
  - `drop_duplicates()` → eliminar filas repetidas.
- **Valores atípicos:**
  - Detectar mediante estadísticas o visualizaciones.
  - Corregir o transformar para evitar sesgos en el análisis.

2. Remodelación y transformación de datos

Cambiar la estructura del DataFrame para análisis o visualización:

- **`pivot_table()`** → crear tablas dinámicas y resumir datos por categorías.
- **`melt()`** → convertir columnas en filas (de formato ancho a largo).

- **`stack()` / `unstack()`** → reestructurar datos entre formatos anchos y largos, útil para análisis multidimensionales.

3. Filtrado y creación de subconjuntos

Permite seleccionar partes específicas de los datos:

- **Indexación booleana:** aplicar condiciones (`>`, `<`, `==`, etc.) para filtrar filas.
- **`loc[]`** → selección por etiquetas (nombres de columnas o índices).
- **`iloc[]`** → selección por posición (números de fila o columna).
- **`query()`** → filtrar usando sintaxis similar a SQL (más legible para condiciones complejas).

4. Agregación e integración

Consiste en resumir y calcular estadísticas sobre grupos de datos:

- **`groupby()`** → agrupa datos por una o más columnas.
    - Funciones comunes: `sum()`, `mean()`, `count()`, `min()`, `max()`, etc.
- Permite crear **estadísticas descriptivas** y analizar tendencias por categorías.
- Se pueden aplicar **funciones personalizadas** con `lambda` o funciones definidas por el usuario para cálculos avanzados.

En resumen

La manipulación de datos en Python implica:

1. **Limpiar y preparar** los datos.
2. **Reestructurarlos y transformarlos** según las necesidades del análisis.
3. **Filtrar y seleccionar** información relevante.

4. **Agrupar y resumir** para obtener conclusiones claras.

Unir y Fusionar DataFrames

- **Objetivo:** combinar datos de diferentes fuentes en un solo conjunto coherente.
- `merge()`: combina DataFrames usando **columnas o índices comunes**, similar a las uniones de SQL.
    - Tipos de unión:
        - **Inner join:** mantiene solo las filas con coincidencias en ambos DataFrames.
        - **Outer join:** incluye todas las filas, rellenando los valores faltantes con NaN.
        - **Left join / Right join:** conserva todas las filas del DataFrame izquierdo o derecho.
- `join()`: une DataFrames basándose **en los índices**; útil con índices jerárquicos o etiquetas de fila.
- Es crucial definir correctamente el **tipo de unión**, las **columnas o índices clave**, y cómo gestionar **duplicados o conflictos** para evitar pérdida o duplicación de datos.

Técnicas avanzadas

1. Análisis de series temporales

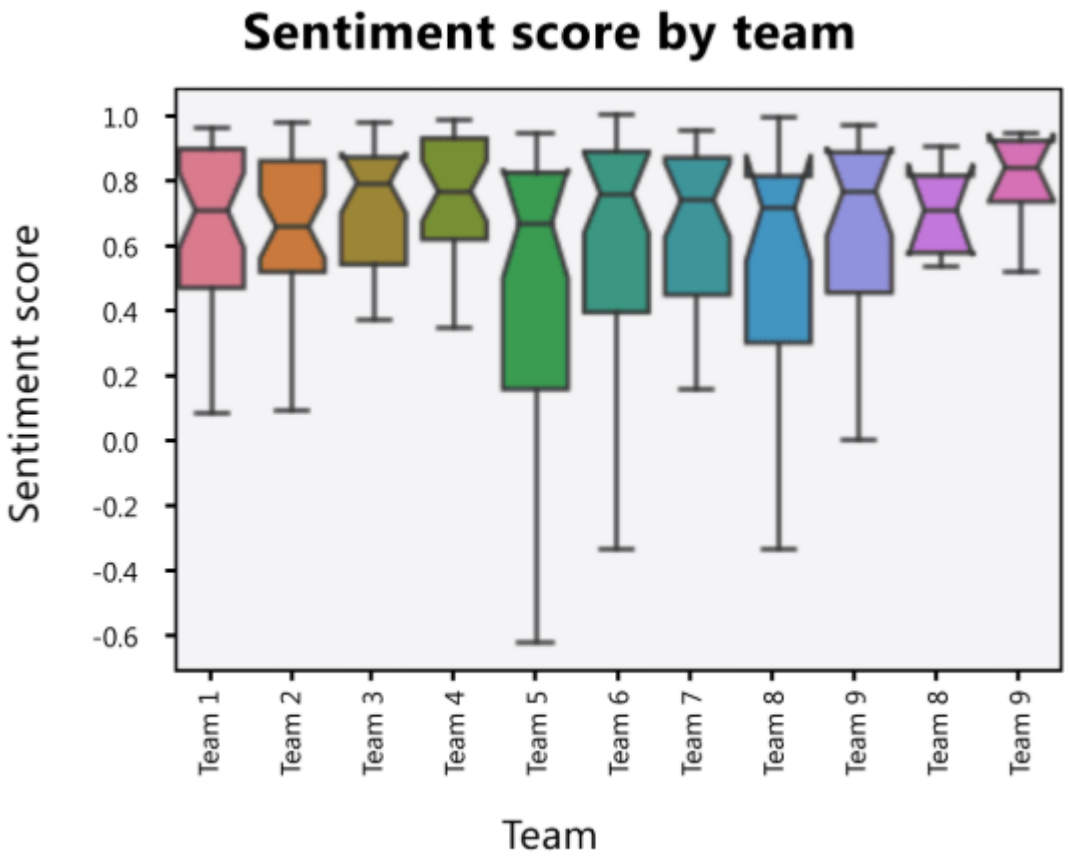
Ideal para datos con componente temporal (precios, ventas, sensores).
Pandas incluye herramientas especializadas para:

- **Indexación fecha/hora:** facilita el acceso y manipulación de datos con timestamps.
- **Remuestreo (`resample`)**: cambia la frecuencia (ej. de diaria a mensual).
- **Ventanas móviles (`rolling`)**: permite analizar tendencias o promedios a lo largo del tiempo.

2. Procesamiento de texto y NLP

Para datos textuales como reseñas o redes sociales.
Bibliotecas clave: **NLTK** y **spaCy**.
Técnicas principales:

- **Tokenización:** dividir texto en palabras o frases.
- **Stemming:** reducir palabras a su raíz.
- **Lematización:** obtener la forma base (diccionario).
- **Análisis de sentimientos:** identificar tono emocional.
- **Modelado temático:** descubrir temas en textos.
  Convierte texto no estructurado en datos organizados para análisis o modelos predictivos.



3. Integración con Aprendizaje Automático

Una vez limpios y transformados los datos, pueden usarse directamente en modelos de **Machine Learning** con **Scikit-learn**.
Permite tareas como:

- **Predicción**, **clasificación** y **agrupación**.
  La compatibilidad directa con **Pandas DataFrames** simplifica el flujo de trabajo y evita conversiones complejas.

4. Visualización de datos

La **visualización** es clave para interpretar los resultados y detectar patrones ocultos.
Bibliotecas recomendadas:

- **Matplotlib:** gráficos personalizados y flexibles.

- **Apache Superset:** visualizaciones interactivas y paneles analíticos.
  Las gráficas ayudan a **comunicar hallazgos** y **validar resultados analíticos**.

Reflexión final

Aunque Python ofrece muchas herramientas para manipular datos, el **pensamiento crítico** sigue siendo esencial.
Aplicar métodos sin comprender los datos puede generar interpretaciones erróneas.
Dominar estas técnicas —desde unir y limpiar hasta visualizar y modelar— convierte la manipulación de datos en una **habilidad esencial para cualquier desarrollador de Python**.

Identificación y tratamiento de datos faltantes

En el mundo real, los datos suelen estar incompletos. Las **lagunas o valores faltantes** pueden deberse a:

- Errores humanos o fallos en la entrada de datos.
- Sensores defectuosos.
- Preguntas omitidas en encuestas.

Si no se manejan correctamente, los datos faltantes pueden generar:

- **Errores en el código.**
- **Cálculos incorrectos** (promedios o sumas erróneas).
- **Modelos de aprendizaje automático sesgados** o poco precisos.

Identificación de datos faltantes

A veces los valores faltantes son **visibles** (celdas vacías o `NaN`), pero otras veces están **ocultos**:

- Se representan con valores ficticios como `999` o `"desconocido"`.
- Para detectarlos, es necesario **examinar los datos y su documentación**, buscando valores anómalos o patrones inusuales.

Elegir la estrategia adecuada

No existe una única solución.
Depende de:

- **Cuántos datos faltan.**
- **Qué tipo de datos faltan.**
- **El propósito del análisis.**

Por ejemplo:

- Si faltan pocos valores, puede **eliminar esas filas**.
- Si faltan muchos y no al azar, eliminarlas puede **sesgar los resultados**.
- En modelos de machine learning, la falta de datos puede **afectar gravemente al rendimiento**.

Técnicas comunes para manejar valores faltantes

1. Eliminación

- Usa `df.dropna()` para quitar filas o columnas con valores faltantes.
- Adecuada si los datos faltantes son pocos y aleatorios.
- Riesgo: pérdida de información y sesgo en los resultados.

2. Imputación

Rellenar los valores faltantes con estimaciones o cálculos:

- **Media, mediana o moda** → métodos simples y rápidos.

- **Regresión** → predice valores faltantes usando otras variables.
- **Imputación múltiple** → crea varios conjuntos de datos con diferentes valores plausibles.

3. **Creación de categorías especiales**
   - Para **datos categóricos**, puede añadirse una categoría `"Desconocido"` o `"Sin dato"`.
   - Preserva la información sobre qué valores estaban ausentes.

4. **Algoritmos que manejan datos faltantes**
   - Algunos modelos de **machine learning** (como árboles de decisión) pueden tratar valores faltantes sin necesidad de imputación explícita.

5. **Recopilación de más datos**
   - Ideal, aunque no siempre viable.
   - Reduce incertidumbre y mejora la calidad general del análisis.

Conclusión

Manejar los valores faltantes es un paso **crítico en la limpieza de datos**.
Elegir la estrategia correcta garantiza:

- **Integridad y confiabilidad** de los datos.
- **Análisis precisos y sin sesgos.**
- **Modelos más sólidos y decisiones de diseño bien fundamentadas.**

Pregunta

¿Cuál de los siguientes es un ejemplo de una forma sutil de representar los datos que faltan en un conjunto de datos? Elija la mejor respuesta.

- ○ Un valor nulo en una base de datos
- ○ Aparece un mensaje de error al acceder a los datos
- ◉ Un valor como "999" para la edad
- ○ Una celda en blanco en una hoja de cálculo

✓ Correcto
Correcto Es una forma sutil de representar los datos que faltan, ya que a primera vista pueden parecer datos válidos.

Tratamiento de valores duplicados

Los **valores duplicados** son registros idénticos dentro de un conjunto de datos.
Pueden ser:

- **Filas completas repetidas**, o
- **Valores duplicados en una columna específica.**

Causas comunes

- **Errores humanos** (entrada doble de información).
- **Combinación de fuentes de datos** (fusión de bases de datos con registros repetidos).
- **Errores de software o importación.**

Por qué son un problema

- Distorsionan el análisis estadístico (por ejemplo, promedios o conteos incorrectos).
- Generan **informes inexactos y decisiones erróneas.**
- Ocupan **espacio innecesario** y pueden ralentizar el procesamiento.

Cómo gestionar los duplicados en Python con pandas

Importar pandas y cargar los datos

```
import pandas as pd
datos = pd.read_csv("tus_datos.csv")
```

1.

Detectar duplicados

```
duplicados = datos.duplicated()
```

2. Esto devuelve un valor booleano para cada fila indicando si está duplicada.

Visualizar las filas duplicadas

```
filas_duplicadas = datos[datos.duplicated()]
print(filas_duplicadas)
```

3.

Eliminar duplicados

```
datos = datos.drop_duplicates()
```

4.
    - `keep='first'` → conserva la primera aparición (predeterminado).
    - `keep='last'` → conserva la última.
    - `keep=False` → elimina **todas** las filas duplicadas.

Cómo prevenir duplicados en el futuro

1. **Mejorar los procesos de entrada de datos**
    - Capacitar al personal.
    - Usar formularios o plantillas estandarizadas.
    - Implementar sistemas de **doble entrada y verificación.**
2. **Añadir comprobaciones de validación**

- Reglas que impidan registrar el mismo dato dos veces.
- Alertas que **marquen posibles duplicados.**

3. **Usar identificadores únicos**
    - Asignar **ID de cliente o claves primarias** automáticas.
    - Garantizar que cada registro sea único en la base de datos.

Conclusión

Detectar y eliminar duplicados es esencial para:

- **Asegurar la precisión del análisis.**
- **Evitar sesgos o errores en los resultados.**
- **Optimizar el almacenamiento y la eficiencia.**

En resumen: **datos limpios = decisiones confiables**.
Dominar estas técnicas con **pandas** te convierte en un analista de datos más sólido y profesional.

Causas habituales de la falta de datos

La **falta de datos** es un problema frecuente que puede afectar la **precisión y fiabilidad del análisis**. Comprender sus causas permite diseñar estrategias efectivas para prevenirla o corregirla.

1. Errores de introducción de datos y omisiones

- El **error humano** es una de las principales causas: omitir preguntas en encuestas, errores tipográficos o datos mal ingresados.
- Estos errores generan lagunas que pueden **sesgar el análisis** y llevar a conclusiones inexactas.

Cómo mitigarlo:

- Aplicar **validación de datos** (restricciones de tipo, rangos permitidos, campos obligatorios).
- Dar **instrucciones claras** a quienes recopilan los datos.
- Usar **métodos de doble entrada** para verificar la exactitud.

2. Fallos técnicos y de equipos automatizados

- En sistemas automatizados (sensores, dispositivos de control), los **fallos de hardware o software** pueden dejar períodos sin registro.
- Esto impide identificar tendencias o detectar anomalías con precisión.

Prevención:

- **Mantenimiento y calibración** regular de los equipos.
- Uso de **sensores redundantes** como respaldo.
- Implementar **mecanismos de gestión de errores** en el software para detectar y corregir fallos rápidamente.

3. Falta de respuesta en encuestas y cuestionarios

- Los participantes pueden **no responder** preguntas (por ser sensibles o extensas) o **abandonar la encuesta**, generando sesgo en los datos.
- Esto puede hacer que los resultados **no representen a toda la población.**

Soluciones:

- Diseñar **encuestas claras y breves**, con una interfaz fácil de usar.
- Garantizar **confidencialidad** y ofrecer **incentivos**.
- Analizar los **patrones de no respuesta** para detectar posibles sesgos.

4. Problemas de integración y fusión de datos

- Al combinar datos de múltiples fuentes, pueden faltar valores por **incompatibilidades de formato, identificadores o definiciones.**

- Ejemplo: una base usa ID de cliente, otra correo electrónico → se pierden coincidencias.

Prevención:

- **Perfilar los datos** antes de la integración (conocer su estructura y calidad).

- Aplicar **estrategias de limpieza y transformación.**

- Usar **fuentes de referencia externas** para resolver discordancias.

5. Eliminación o anonimización por privacidad

- Las regulaciones (como **GDPR**) obligan a eliminar o anonimizar datos sensibles.

- Esto protege la privacidad, pero puede limitar el análisis de tendencias o estudios longitudinales.

Equilibrio entre privacidad y utilidad:

- Aplicar **enmascaramiento de datos** para mantener la estructura sin revelar información personal.

- Usar **datos sintéticos** o **privacidad diferencial** como alternativas seguras.

Conclusión

La falta de datos puede deberse a **errores humanos, fallos técnicos, baja participación, integración deficiente o restricciones de privacidad**.
Para minimizar su impacto:

- Implementa **buenas prácticas de validación, mantenimiento, diseño de encuestas y gobernanza de datos.**

- Busca siempre **equilibrar la calidad del análisis con la protección de la privacidad.**

Cómo abordar la falta de datos con Pandas

La **falta de datos** es un problema común en el análisis de datos. **Pandas**, una de las bibliotecas más potentes de Python para el manejo de datos, ofrece diversas herramientas para **detectar, eliminar, imputar y analizar** los valores perdidos, además de tratar **valores atípicos** y **tipos de datos inconsistentes**.

1. Identificación de valores perdidos

- Se utilizan las funciones `isnull()` y `notnull()` para crear máscaras booleanas que muestran qué valores están ausentes.

- `isnull().sum()` permite ver el número de valores perdidos por columna.

2. Eliminación de valores perdidos

- Cuando la cantidad de datos faltantes es pequeña y aleatoria, se pueden eliminar las filas o columnas incompletas con `dropna()`.

- Esta función ofrece flexibilidad para decidir si eliminar filas, columnas o establecer umbrales de tolerancia.

3. Imputación (relleno) de valores faltantes

- En lugar de eliminar datos, se pueden **reemplazar los valores perdidos por estimaciones**:

    - Rellenar con la **media** o la **mediana** de la columna (`fillna()` junto con `mean()` o `median()`).

    - También pueden usarse métodos más avanzados (regresiones o modelos de ML).

4. Detección y manejo de valores atípicos

- Los valores atípicos (outliers) pueden distorsionar el análisis.
- Pueden identificarse con funciones como `describe()`, `quantile()` o visualizaciones (por ejemplo, boxplots).
- Una estrategia común es **limitar** (capar) los valores a un umbral con `clip()`.

5. Conversión de tipos de datos

- Asegurar la coherencia de tipos es crucial.
- Pandas permite convertir columnas fácilmente con:
    - `astype()`
    - `to_numeric()`
    - `to_datetime()`

Estas funciones también permiten manejar errores de conversión y mantener el conjunto de datos limpio.

6. Análisis Exploratorio de Datos (EDA)

- Con herramientas como `describe()` y `groupby(), info()`, pandas facilita obtener estadísticas descriptivas, analizar patrones y descubrir posibles relaciones.
- Este paso es esencial para decidir la mejor estrategia ante los datos faltantes.

Ejemplo práctico en código (con explicación paso a paso)

```
import pandas as pd
import numpy as np
```

```
# Crear un DataFrame con valores perdidos
data = {'Name': ['Alice', 'Bob', np.nan, 'David'],
        'Age': [25, 30, np.nan, 35],
        'City': ['New York', np.nan, 'London', 'Paris']}
df = pd.DataFrame(data)

# 1. Identificar valores perdidos
print("Missing value counts per column:\n", df.isnull().sum())

# 2. Eliminar filas con valores perdidos
df_dropped = df.dropna()
print("\nDataFrame after dropping rows with any missing value:\n",
df_dropped)

# 3. Imputar valores faltantes con la media
df_filled_mean = df.fillna(df.mean(numeric_only=True))
print("\nDataFrame after filling missing 'Age' with mean:\n",
df_filled_mean)

# 3. Imputar valores faltantes con la mediana
df_filled_median = df.fillna(df.median(numeric_only=True))
print("\nDataFrame after filling missing 'Age' with median:\n",
df_filled_median)

# 4. Manejo de valores atípicos: limitar (capar) a 40
df['Age_capped'] = df['Age'].clip(upper=40)
print("\nDataFrame with 'Age' capped at 40:\n", df)

# 5. Conversión de tipos de datos
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
print("\nData types after conversion:\n", df.dtypes)

# 6. Análisis exploratorio de datos (EDA)
print("\nDescriptive statistics:\n", df.describe())

# Agrupar por ciudad y calcular la edad promedio
grouped_data = df.groupby('City')['Age'].mean()
print("\nAverage Age by City:\n", grouped_data)
```

Salida resumida del código

- Se detectan valores perdidos en las columnas `Name`, `Age` y `City`.
- Tras aplicar `dropna()`, solo quedan las filas completas (Alice y David).
- Los valores numéricos faltantes se rellenan con la **media** o la **mediana**.
- Se demuestra cómo **limitar un valor máximo** con `clip()`.
- Se convierten tipos de datos con `to_numeric()`.
- Finalmente, se muestran estadísticas descriptivas (`describe()`) y el promedio de edad por ciudad (`groupby()`).

Conclusión

Pandas ofrece un conjunto completo de herramientas para **gestionar eficazmente la falta de datos**:

- Detectar (`isnull()`, `notnull()`),
- Eliminar (`dropna()`),
- Imputar (`fillna()` con media o mediana),
- Controlar valores atípicos (`clip()`),
- Asegurar tipos coherentes (`astype()`, `to_numeric()`),
- y Analizar los datos (`describe()`, `groupby()`).

Con estas técnicas, es posible **mantener la integridad del conjunto de datos** y garantizar que el análisis sea **preciso y confiable**, incluso cuando existan valores faltantes o inconsistencias.

Detección y eliminación de valores atípicos

Definición

Los **valores atípicos** son puntos de datos que **se desvían significativamente** del resto del conjunto de datos. Representan valores extremos que **no siguen el patrón general** y pueden aparecer por diversas causas.

Causas comunes de los valores atípicos

1. **Errores de medición:** fallos en instrumentos o dispositivos durante la recopilación de datos.
2. **Errores de entrada de datos:** equivocaciones o errores tipográficos al registrar información.
3. **Anomalías genuinas:** eventos reales y poco comunes que se alejan de la norma (por ejemplo, un pico de ventas inesperado).

Por qué importan los valores atípicos

- **Distorsionan** las medidas estadísticas (media, desviación estándar).
- **Afectan** los modelos de aprendizaje automático (predicciones menos precisas).
- **Dificultan** la interpretación visual de los datos.
  Por ello, **detectarlos y gestionarlos** correctamente es esencial para obtener conclusiones fiables.

Métodos para detectar valores atípicos

1. Puntuación Z (Z-score)

- Mide cuántas **desviaciones estándar** está un punto con respecto a la media.

- Los puntos con una puntuación mayor a **±3** suelen considerarse outliers.
- Ideal para datos **normalmente distribuidos**.

2. Rango Intercuartílico (IQR)

- Mide la dispersión del **50% central** de los datos.
- Se calcula como:

$$IQR = Q3 - Q1$$

  Donde:
  - **Q1** = primer cuartil (25%)
  - **Q3** = tercer cuartil (75%)
- Un punto es un valor atípico si:

$$x < Q1 - 1.5 \times IQR \quad \text{o} \quad x > Q3 + 1.5 \times IQR$$

- Más **resistente** a valores extremos que el método Z-score.

3. Visualización

- **Gráficos de caja (boxplots):** muestran claramente los cuartiles y los valores atípicos como puntos fuera de los “bigotes”.
- **Gráficos de dispersión:** permiten detectar visualmente valores inusuales.

Técnicas para manejar valores atípicos

1. **Recorte (trimming):**
   - Elimina los valores atípicos del conjunto de datos.
   - Eficaz, pero puede causar **pérdida de información valiosa**.
2. **Capado (winsorizing):**
   - Sustituye los valores extremos por un **límite máximo o mínimo**, como el percentil 95 o 5.

- Preserva más datos, reduciendo el impacto de los extremos.

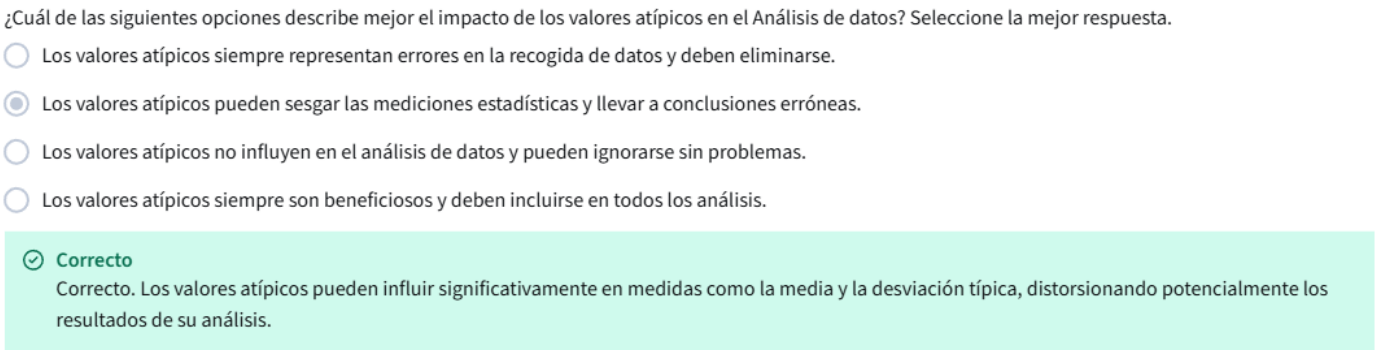
3. **Imputación:**
    - Sustituye los valores atípicos por valores estimados (media, mediana, etc.).
    - Debe usarse con precaución, ya que puede **introducir sesgos**.

Factores adicionales a considerar

- **Conocimiento del dominio:** puede ayudar a decidir si un valor extremo es un error o un dato legítimo.
- **Análisis de sensibilidad:** comparar resultados **con y sin outliers** para evaluar su influencia.
- **Consulta con expertos:** obtener la opinión de estadísticos o científicos de datos experimentados.

Conclusión

Los **valores atípicos** no deben eliminarse sin reflexión. La clave está en **comprender su origen y su impacto**, y aplicar el método más adecuado según el contexto del análisis.
Con herramientas como **Z-score, IQR y visualizaciones**, junto con estrategias de **recorte, capado o imputación**, se pueden manejar los valores atípicos de forma efectiva, asegurando un **análisis de datos sólido, fiable y preciso**.



Cuestionario: El héroe del análisis de datos, la limpieza de datos

1. Estás trabajando con un conjunto de datos que tiene diversos tipos de datos y formatos. ¿Cuáles de las siguientes son técnicas para garantizar la coherencia de los datos y la facilidad de uso para el análisis? Seleccione todas las que corresponda.

   - [x] Conversión de tipos de datos

   Correcto
   Correcto La conversión de tipos de datos garantiza la coherencia de los datos transformándolos en formatos adecuados.

   - [x] Imputación

   Correcto
   Correcto La imputación es un método habitual para sustituir los valores que faltan por valores estimados o calculados.

   - [ ] Ignorar los valores que faltan
   - [x] Detección de valores atípicos

   Correcto
   Correcto La detección de valores atípicos consiste en identificar los valores extremos que se desvían del patrón general de los datos.

2. ¿Cuál es un uso común del Análisis exploratorio de datos (EDA)? Seleccione todo lo que corresponda.

   - [x] Creación de estadísticas descriptivas

   Correcto
   Correcto Las estadísticas descriptivas como la media, la varianza y la desviación típica ayudan a sacar conclusiones basadas en los datos.

   - [ ] Recopilación de datos
   - [x] Generar agregaciones

   Correcto
   Correcto Los gráficos de dispersión suelen utilizarse para identificar relaciones entre variables continuas.

   - [x] Creación de visualizaciones

   Correcto
   Correcto Las visualizaciones como los gráficos se utilizan con frecuencia en EDA.

3. Está trabajando con un conjunto de datos que contiene algunos valores perdidos. Decide que, para su análisis específico, es mejor eliminar cualquier fila que contenga valores perdidos para evitar posibles sesgos o errores en sus cálculos. ¿Qué método de Pandas utilizaría para conseguirlo? Seleccione la mejor respuesta.

- (●) dropna()
- ( ) fillna()
- ( ) isnull()
- ( ) notnull()

Correcto
Correcto El método dropna() se utiliza para eliminar valores perdidos de un DataFrame.

4. Estás analizando los datos de un experimento científico y observas varios valores atípicos. ¿Qué es un valor atípico en un conjunto de datos? Selecciona la mejor respuesta.

- (●) Punto de datos que difiere significativamente de otras observaciones
- ( ) Un punto de datos que siempre es incorrecto
- ( ) Punto de datos que se encuentra dentro del intervalo intercuartílico
- ( ) Punto de datos que coincide exactamente con la media del conjunto de datos

Correcto
Correcto Los valores atípicos son puntos de datos que difieren significativamente de otras observaciones del conjunto de datos.

5. Está analizando los datos de ventas de una tienda y se da cuenta de que algunos registros de compra de clientes aparecen varias veces en el conjunto de datos. Se da cuenta de que estos registros duplicados podrían sesgar su análisis y llevar a conclusiones inexactas sobre las tendencias de ventas y el comportamiento de los clientes. ¿Cuál es la razón principal para eliminar estos registros duplicados durante el proceso de limpieza de datos? Seleccione la mejor respuesta.

- ( ) Para ahorrar espacio de almacenamiento.
- (●) Para evitar distorsionar los resultados de los análisis.
- ( ) Para mantener la integridad de los datos.
- ( ) Para aumentar la velocidad de procesamiento.

Correcto
Correcto Los registros duplicados pueden distorsionar los resultados de los análisis inflando artificialmente las cifras de ventas o falseando las pautas de compra de los clientes.

6. Está trabajando con un conjunto de datos que contiene información de clientes, incluidos nombres, direcciones e historial de compras. Descubre que algunos registros de clientes se han introducido varias veces sin querer, creando entradas redundantes. ¿Cuál de las siguientes técnicas sería más eficaz para identificar y tratar estas entradas repetidas a fin de garantizar la precisión de los análisis y los informes? Seleccione todas las que correspondan.

- [ ] Utilización del método .replace() para tratar los duplicados
- [x] Quitar duplicados con .drop_duplicates()

Correcto
Correcto El método .drop_duplicates() se utiliza habitualmente para eliminar valores duplicados de un conjunto de datos.

- [ ] Ordenar el conjunto de datos para encontrar duplicados
- [x] Marcar duplicados con una nueva columna

Correcto
Correcto Marcar los duplicados con una nueva columna puede ayudar a identificarlos y gestionarlos posteriormente.

7. Está trabajando con un conjunto de datos de información sobre clientes y se da cuenta de que faltan algunas direcciones. Quiere asegurarse de que sus análisis y campañas de marketing no se vean afectados negativamente por esta información incompleta. ¿Cuáles de las siguientes son razones potenciales para que falten estas direcciones? Seleccione todas las que correspondan.

- [x] Errores en la introducción de datos

Correcto
Correcto El error humano desempeña un papel importante en la falta de datos.

- [ ] Tipos de datos incorrectos
- [x] Usuarios que se saltan una pregunta

Correcto
Correcto Los usuarios pueden saltarse una pregunta, intencionadamente o no, durante el proceso de recogida de datos.

- [ ] Elevada varianza de los datos

Tipos de datos en Python: la elección correcta

En esta conversación, el chef enseña a Alex que en Python los **tipos de datos** son como los **ingredientes** de una receta: cada uno tiene su función y debe usarse correctamente para que el resultado sea exitoso.

Tipos de datos básicos en Python

1. **Enteros (int):** se usan para **contar** cosas, como el número de clientes.
2. **Flotantes (float):** sirven para **medidas precisas**, como la cantidad de especias.
3. **Cadenas (str):** representan **texto**, como nombres de platos o mensajes.
4. **Booleanos (bool):** son como un **interruptor** de luz: solo pueden ser **verdadero (True)** o **falso (False)**, y se usan para responder preguntas de **sí o no** (por ejemplo, si hay o no sal).

Importancia de usar el tipo correcto

Usar el tipo de dato adecuado es esencial, igual que usar el utensilio correcto en la cocina.
Si se usa un tipo incorrecto, Python puede producir **errores o resultados inesperados**, como mezclar azúcar en lugar de sal: una “receta para el desastre”.

Ejemplo práctico

Intentar multiplicar una **palabra (cadena)** por un **número** sin sentido es como mezclar aceite y agua: **no funciona**.
Cada tipo tiene su propósito y combinaciones válidas.

Conclusión

Con la **práctica**, elegir el tipo de dato correcto se vuelve natural, igual que un chef domina sus ingredientes.
Programar en Python es como cocinar: entender los **ingredientes (tipos de datos)** es clave para crear **recetas (programas)** exitosas.

Pandas para tareas esenciales de análisis

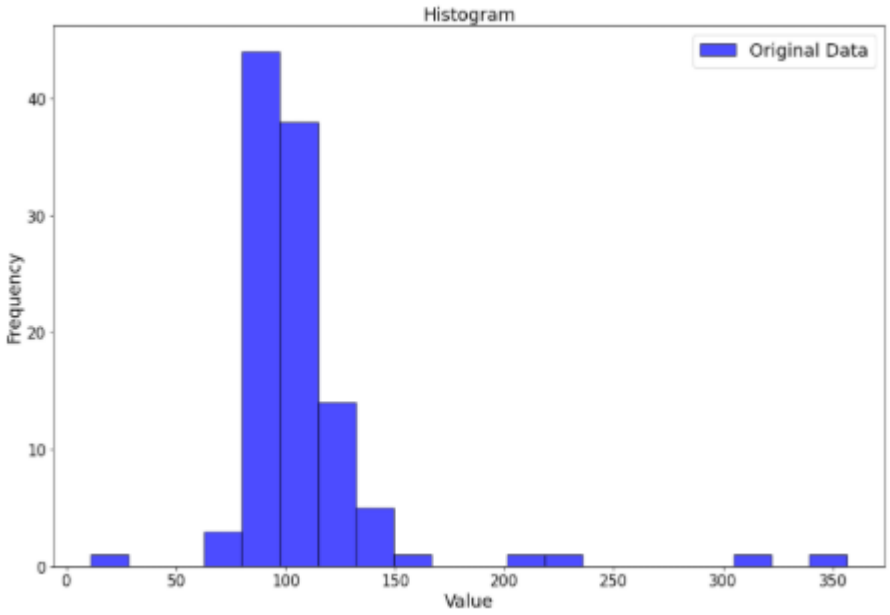
Pandas es una biblioteca fundamental en Python para **manipular y analizar datos de manera eficiente**, ayudando a transformar datos sin procesar en información útil.

1 Manejo de datos faltantes

- Los datos incompletos son comunes y pueden afectar la **precisión y confiabilidad** del análisis.
- **Identificación:**
    - `isnull()` y `notnull()` permiten localizar valores nulos en un DataFrame.
- **Tratamiento:**
    - `dropna()` elimina filas o columnas con valores faltantes, útil si la cantidad de datos faltantes es pequeña.
    - `fillna()` permite imputar valores, por ejemplo:
        - Numéricos: media o mediana.
        - Categóricos: categoría más frecuente o un valor marcador.
- Ejemplo: en un conjunto de clientes de e-commerce, se puede rellenar la edad o ingresos faltantes con el promedio para evitar sesgos.

2 Control de valores atípicos

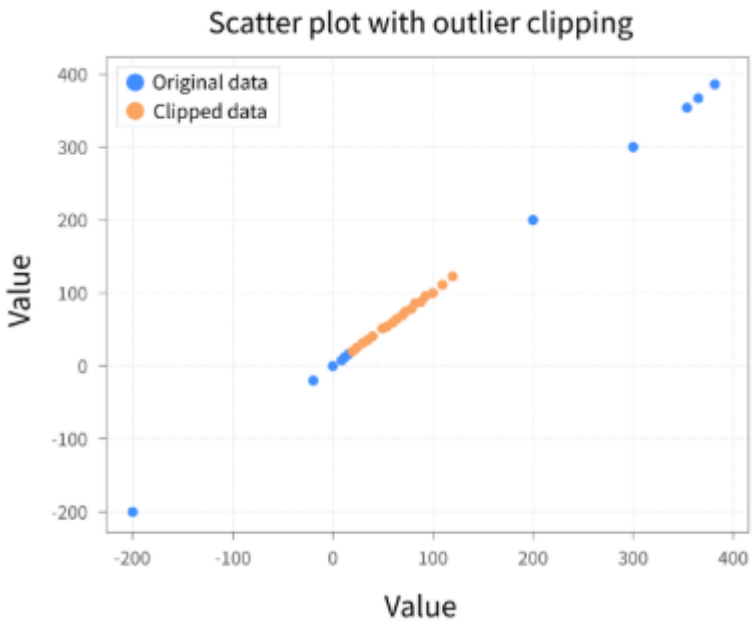
- Los valores atípicos son datos que se desvían significativamente de la norma y pueden distorsionar análisis estadísticos.
- **Identificación:**
    - `describe()` proporciona estadísticas resumidas, ayudando a detectar puntos fuera de la distribución esperada.



- `quantile()` permite definir umbrales basados en cuartiles y conocimiento del dominio.

- **Gestión:**

  - `clip()` limita los valores a un rango específico, reduciendo la influencia de valores extremos en el análisis.



Beneficios

- Mantener la integridad de los datos.
- Evitar sesgos en el análisis.
- Obtener conclusiones más **fiables y representativas** del comportamiento típico de los datos.

1 Conversiones de tipo de datos

- Los datos llegan en distintos formatos y es crucial **convertirlos al tipo adecuado** para análisis confiables.
- **Funciones clave:**
    - `astype()`: convierte columnas o DataFrames a un tipo específico.
    - `to_numeric()`, `to_datetime()`, `to_categorical()`: para conversiones especializadas, como fechas o categorías.
- Ejemplo: convertir cadenas que representan fechas a `datetime` permite análisis de series temporales y cálculos de intervalos.

2 Análisis exploratorio de datos (EDA)

- **Objetivo: resumir y visualizar** datos para entender patrones y detectar problemas.
- **Funciones útiles:**
    - `head()` y `tail()`: vistazo rápido al inicio y final del DataFrame.
    - `describe()`: estadísticas resumidas (media, desviación estándar, cuartiles).
    - `info()`: resumen de columnas, tipos de datos y valores nulos.
    - `groupby()`: agrupa datos por categorías para análisis agregados.
    - `plot()` + Matplotlib: visualizaciones (histogramas, diagramas de dispersión, gráficos de líneas) para identificar tendencias y relaciones.

3 Técnicas avanzadas

- **Indexación y selección:** filtrar datos con condiciones complejas o índices jerárquicos.
- **Series temporales:** remuestreo, ventanas móviles y desplazamientos temporales.
- **Integración con otras bibliotecas:**
    - NumPy para cálculos numéricos.
    - Scikit-learn para aprendizaje automático.

Beneficios de dominar Pandas

- Manejo eficiente de **datos faltantes y valores atípicos**.
- Conversión de tipos de datos para compatibilidad y precisión.
- Exploración, agregación y visualización para **obtener información valiosa**.
- Integración con otras herramientas de Python para análisis avanzado y pipelines de datos.

Demo: pandas para exploración y limpieza (contiene código)

1 Introducción

- **Pandas** es una biblioteca esencial en ciencia de datos para limpiar, transformar y analizar datos.
- Permite manejar conjuntos de datos desordenados con **valores faltantes, inconsistencias y formatos variados**, transformándolos en información lista para el análisis.

2 Preparación e inspección de datos

- Se carga un conjunto de datos (por ejemplo, un **CSV**) usando `pd.read_csv()`.
- Se examina con:
    - `head()` → muestra las primeras filas.
    - `info()` → muestra los nombres de columnas, tipos de datos y valores no nulos.
    - `isnull().sum()` → identifica y cuenta los valores faltantes.

3 Limpieza de datos

- **Eliminar columnas o filas irrelevantes o incompletas:**
    - `drop()` → **elimina columnas** (con `axis=1`) o filas.
    - `dropna()` → **elimina filas** con datos faltantes, conservando la estructura de las columnas.
- **Rellenar valores faltantes:**
    - `fillna()` → sustituye valores ausentes por un valor definido (por ejemplo, la media, 0, 1, etc.).
- **Convertir tipos de datos:**
    - `to_datetime()` → transforma columnas de texto en formato de fecha y hora.

4 Transformación de datos

- **Agrupar y resumir información:**
    - `groupby()` y `sum()` → agrupan datos y calculan totales o promedios, por ejemplo, ventas por producto.
- **Crear nuevas columnas:**
    - Ejemplo: calcular el **beneficio** restando el costo del precio de venta.

5 Exploración y análisis

- **Funciones estadísticas:**
    - `mean()`, `median()`, `describe()` → resumen de tendencias, dispersión y distribución.
- **Visualización:**
    - Integración con **Matplotlib** y **Seaborn** para crear gráficos (dispersión, barras, líneas, etc.) que revelan relaciones y patrones entre variables.

Conclusión

- Pandas es una herramienta **potente, flexible y esencial** para la manipulación y análisis de datos.
- Permite limpiar, transformar, explorar y visualizar datos de manera eficiente.
- Se recomienda seguir practicando y explorar su documentación para dominar todas sus capacidades.

Pregunta

Escenario: Está trabajando con un conjunto de datos que contiene información sobre las compras de los clientes. El conjunto de datos tiene algunos problemas: formatos de fecha inconsistentes, valores perdidos en la columna "Edad" y precios almacenados como texto en lugar de números.

Pregunta Stem: ¿Cómo puede ayudarle la librería Pandas a resolver estos problemas de calidad de datos? Seleccione la mejor respuesta.

- ○ pandas puede utilizarse para entrenar un modelo de aprendizaje automático que prediga los valores correctos para los datos que faltan.
- ○ pandas puede visualizar directamente los datos, resaltando las incoherencias y los errores.
- ◉ pandas proporciona funciones para convertir tipos de datos, manejar valores perdidos y estandarizar formatos.
- ○ pandas puede generar automáticamente datos sintéticos para reemplazar los valores de "Edad" que faltan.

✓ **Correcto**
Correcto! Pandas ofrece una amplia gama de herramientas para la limpieza y transformación de datos, incluyendo funciones para abordar los problemas descritos.

```python
#Show the info on the data:
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85 entries, 0 to 84
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   date                85 non-null     object
 1   region              82 non-null     object
 2   product             71 non-null     object
 3   sales               62 non-null     float64
 4   price               58 non-null     object
 5   cost                59 non-null     object
 6   irrelevant_column_1  82 non-null    object
 7   irrelevant_column_2  53 non-null    object
dtypes: float64(1), object(7)
memory usage: 5.4+ KB
None
```

```python
# Count the null values:
print(df.isnull().sum())
```

```
date                   0
region                 3
product               14
sales                 23
price                 27
cost                  26
irrelevant_column_1    3
irrelevant_column_2   32
dtype: int64
```

Remove the Irrelevant Column

```python
# Remove irrelevant column:

df = df.drop('irrelevant_column_1', axis=1)
```

Update the Date

```python
df['date'] = pd.to_datetime(df['date'])
```

Aggregate the DataFrame

```python
sales_by_product = df.groupby('product')['sales'].sum()
print(sales_by_product)
```

```
product
Widget A    16300.0
Widget B    15830.0
Widget C    16150.0
Name: sales, dtype: float64
```

Create Profit Column

```
# Debug and eradicate any strings or excessive floats:
df = df.dropna()
df['price'] = [ int(float(x)) for x in df['price']]
df['cost'] = [ int(float(x)) for x in df['cost']]
```

```
# Create the profit column:
df['profit'] = df['price'] - df['cost']
```

```
# Get intelligence on the data:
print(df['profit'].mean())
print(df['profit'].median())
print(df.describe())
```

```
12.5
13.0
                      date        sales      price       cost     profit
count                   12    12.000000  12.000000  12.000000  12.000000
mean   2024-01-23 08:00:00  1000.000000  22.166667   9.666667  12.500000
min    2024-01-01 00:00:00   700.000000  17.000000   7.000000   8.000000
25%    2024-01-07 18:00:00   837.500000  19.500000   8.750000  10.750000
50%    2024-01-20 12:00:00   975.000000  23.000000  10.000000  13.000000
75%    2024-02-04 12:00:00  1100.000000  25.000000  11.000000  14.250000
max    2024-02-25 00:00:00  1500.000000  26.000000  12.000000  16.000000
std                    NaN   242.149465   3.406767   1.723281   2.504541
```

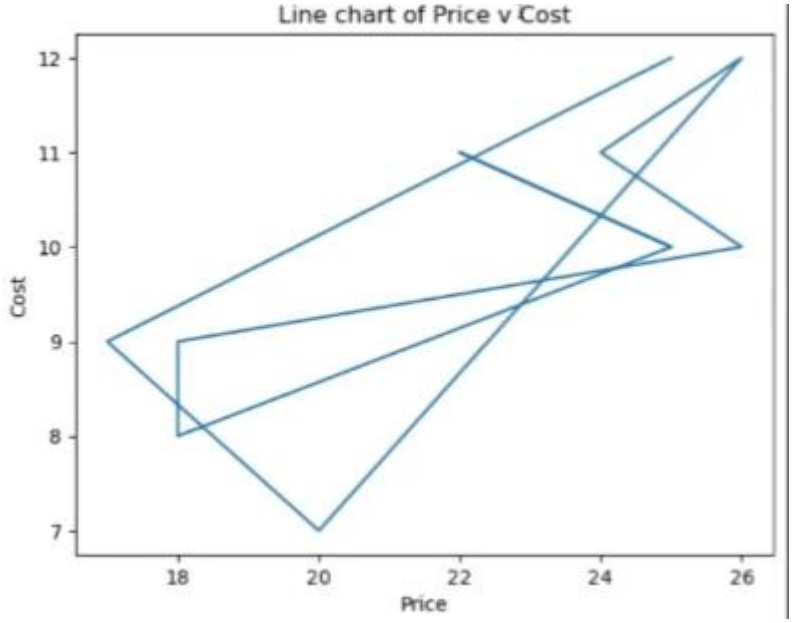
Figures

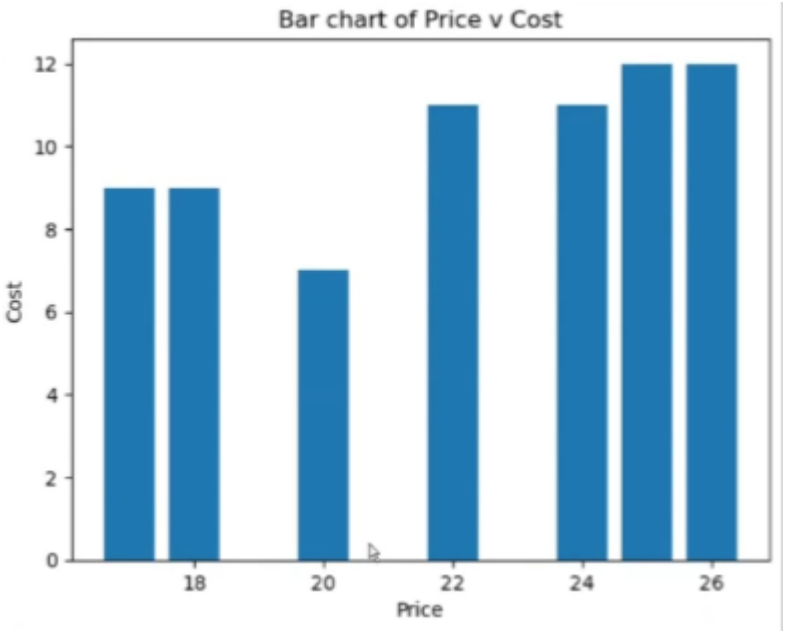
```
import matplotlib.pyplot as plt

X = df["price"]
y = df["cost"]

#Line Chart
plt.plot(X,y)
plt.title("Line chart of Price v Cost")
plt.xlabel("Price")
plt.ylabel("Cost")
plt.show()


#Bar Chart
plt.bar(X,y)
plt.title("Bar chart of Price v Cost")
plt.xlabel("Price")
plt.ylabel("Cost")
plt.show()
```





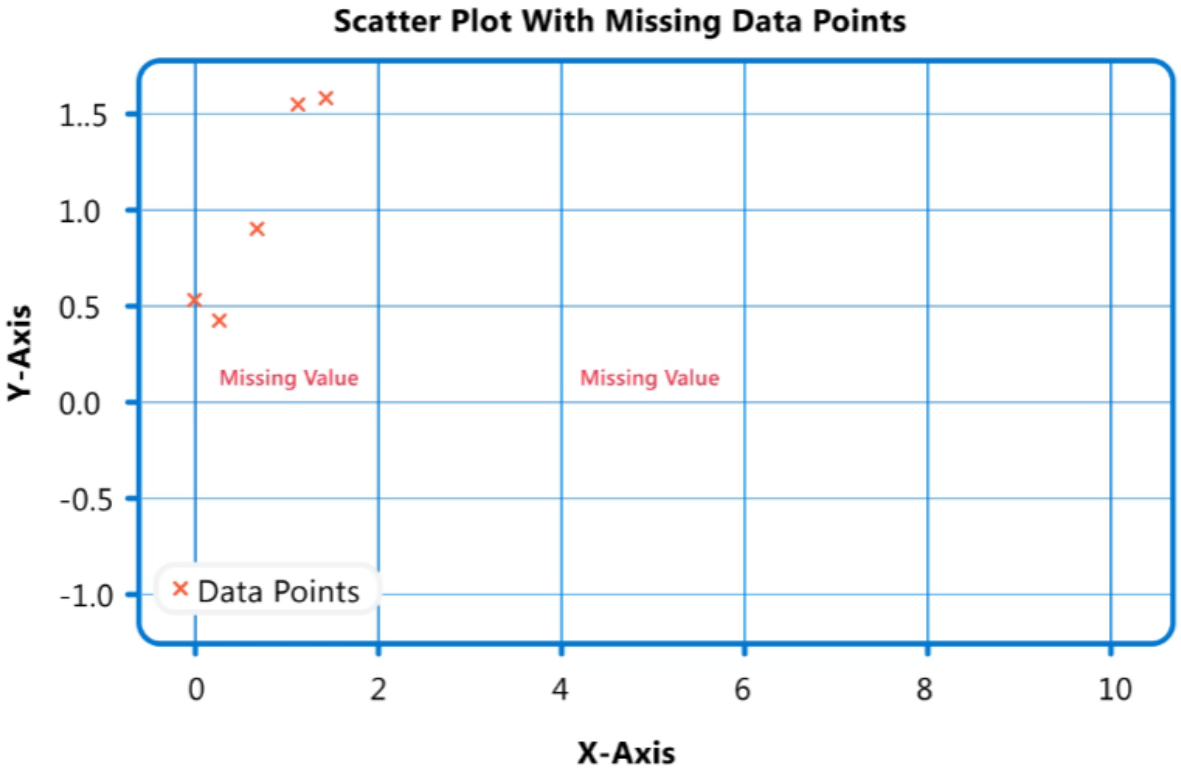
Datos desordenados con pandas

Introducción: los datos como escena del crimen

- En cualquier proyecto basado en datos (UI, marketing o investigación), **la calidad de los resultados depende de la calidad de los datos**.
- La **limpieza de datos** es el primer paso para transformar datos caóticos en información confiable.
- **Pandas**, biblioteca clave de Python, ofrece herramientas poderosas para limpiar, transformar y analizar datos de forma eficiente.

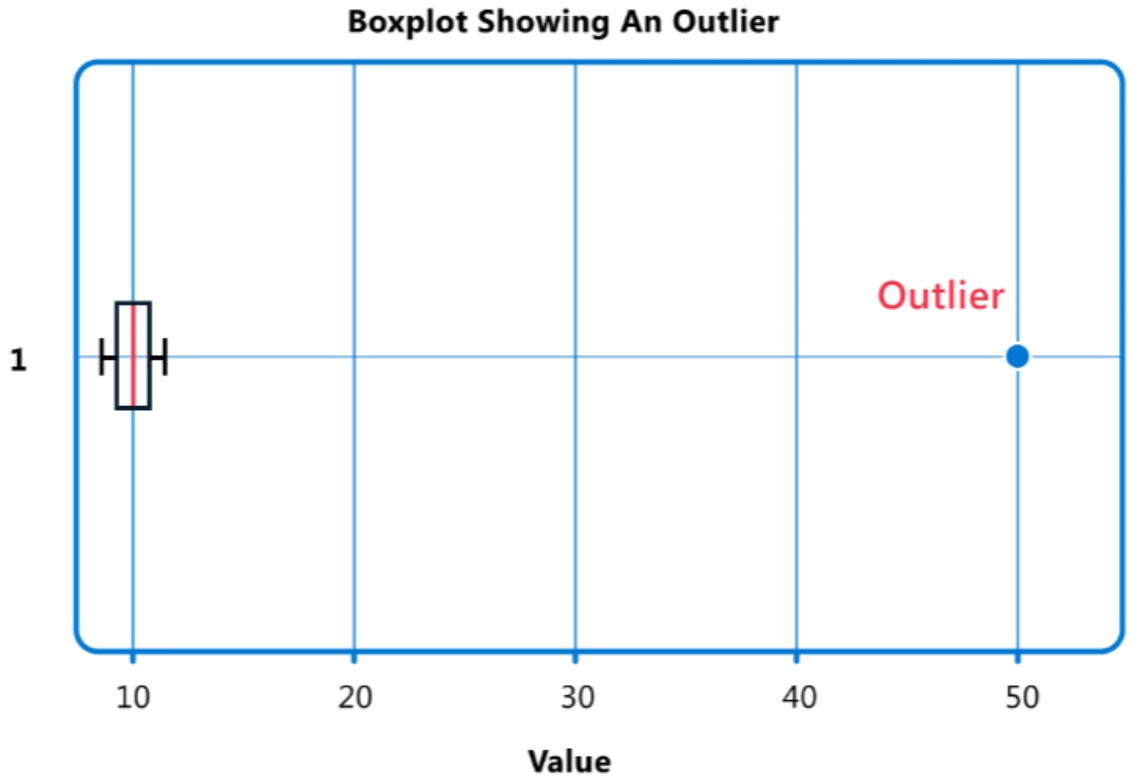
2 Problemas comunes de calidad de los datos

1. **Valores faltantes:**
    - Son los “espacios en blanco” del conjunto de datos.
    - Causas: sensores defectuosos, encuestas incompletas o errores humanos.
    - Impacto: resultados sesgados o pérdida de precisión.



2. **Valores atípicos (outliers):**

- Puntos que se alejan del patrón general (por ejemplo, un multimillonario en un barrio promedio).
- Pueden distorsionar promedios y conclusiones.



3. **Formato inconsistente:**
   - Fechas en distintos formatos, mayúsculas irregulares o espacios extra.
   - Dificultan comparaciones y análisis correctos.

Inconsistent Formatting Example

	Name	Date
1	Alice	01-01-2020
2	BOB	2020/01/01
3	Carol	Jan 1, 2020
4	bob	1st Jan 2020
5	Alice	2020-01-01

4. **Datos duplicados:**
    - Surgen de errores o combinación de fuentes.
    - Generan redundancia y pueden alterar los resultados.

3 Impacto de la mala calidad de los datos

- Distorsiona estadísticas, sesga decisiones y puede llevar a conclusiones erróneas.
- Dificulta la integración y comparación entre fuentes de datos.

4 Herramientas clave de Pandas para limpiar datos

Problema	Función de Pandas	Descripción

Valores faltantes	`fillna()`, `dropna()`	Rellena con la media, mediana o elimina filas/columnas incompletas.
Duplicados	`drop_duplicates()`	Elimina filas repetidas según columnas específicas.
Formato inconsistente	`astype()`, `str.strip()`, `to_datetime()`	Convierte tipos de datos, limpia texto y estandariza formatos.
Valores atípicos	Transformaciones y escalado	Identifica y ajusta valores extremos mediante visualización o métodos estadísticos.

5 Otras técnicas de limpieza

- **Normalización de datos:** Escala los valores numéricos (por ejemplo, min–max o z-score) para compararlos correctamente.
- **Manipulación de texto:** Usa operaciones de cadena y **expresiones regulares** (`str.replace`, `str.extract`) para limpiar y extraer información.
- **Validación:** Verifica que los valores estén dentro de rangos válidos o que las categorías sean coherentes.

Conclusión

- La limpieza de datos es **un proceso iterativo y esencial**.
- Pandas ofrece una base sólida para detectar, corregir y validar errores de forma eficiente.
- Con práctica, se convierte en una herramienta indispensable para transformar datos desordenados en **insights valiosos y confiables**.

Pregunta

¿Cuál de los siguientes es un problema común de calidad de datos que Pandas puede ayudarle a resolver? Seleccione la mejor respuesta.

- (●) Formato incoherente
- ( ) Despliegue de modelos
- ( ) Visualización de datos
- ( ) Selección del algoritmo

Correcto
Correcto. El formato incoherente es un reto común de calidad de datos que pandas puede ayudar a resolver mediante el uso de capacidades de manipulación de cadenas y expresiones regulares.

Uso de pandas para limpieza y depuración

1. ¿Cuál de los siguientes métodos se puede utilizar para el Análisis exploratorio de datos en Pandas? Seleccione todos los que correspondan.

   - [ ] fusionar()
   - [x] describir()

   Correcto
   Correcto El método describe() proporciona un resumen de las estadísticas de las columnas numéricas.

   - [x] info()

   Correcto
   Correcto El método info() proporciona un resumen conciso de un DataFrame, incluidos los tipos de datos y los recuentos no nulos.

   - [x] groupby()

   Correcto
   Correcto El método groupby() se utiliza para dividir los datos en grupos y realizar operaciones en cada grupo

2. ¿Cuál de las siguientes funciones de Pandas se utiliza normalmente para manejar los datos que faltan en un DataFrame mediante la eliminación de filas o columnas? Seleccione la mejor respuesta.

- ○ eliminar()
- ○ fillna()
- ◉ dropna()
- ○ reemplazar()

Correcto
Correcto La función dropna() se utiliza normalmente para eliminar los datos que faltan de un DataFrame. Buen trabajo identificando la función correcta.

3. ¿Cómo puede afectar la falta de datos a los resultados de un proyecto de Análisis de datos? Seleccione la mejor respuesta.

- ◉ Puede dar lugar a estimaciones sesgadas y conclusiones incorrectas.
- ○ No tiene ningún impacto si el conjunto de datos es lo suficientemente grande.
- ○ Sólo afecta a la parte de visualización del análisis.
- ○ Siempre da lugar a predicciones más precisas.

Correcto
Correcto Los datos que faltan pueden sesgar los resultados, dando lugar a estimaciones sesgadas y conclusiones potencialmente incorrectas.

4. ¿Cuáles de las siguientes son consecuencias potenciales de la mala calidad de los datos en los procesos de toma de decisiones? Seleccione todas las que procedan.

- [x] Asignación ineficiente de recursos

Correcto
Correcto La mala calidad de los datos puede dar lugar a una asignación ineficiente de los recursos debido a la falta de fiabilidad de la información.

- [x] Predicciones inexactas

Correcto
Correcto La mala calidad de los datos puede dar lugar a predicciones inexactas, lo que afecta a la fiabilidad de la toma de decisiones.

- [ ] Visualización de datos mejorada
- [x] Perspectivas engañosas

Correcto
Correcto La mala calidad de los datos puede dar lugar a percepciones erróneas, que pueden afectar significativamente a los procesos de toma de decisiones.

5. Está analizando un conjunto de datos de ventas y necesita determinar el tipo de datos adecuado para una columna que contiene precios de productos, que incluyen valores tanto en dólares como en céntimos. ¿Qué tipo de datos representaría mejor estos valores para realizar cálculos y análisis precisos? Seleccione la mejor respuesta.

- (•) float
- ( ) booleano
- ( ) cadena
- ( ) entero

Correcto
Correcto 'float' es el tipo de datos más apropiado para almacenar valores numéricos con decimales, como los precios.

ACTIVIDAD FINAL: Tratamiento y manipulación de datos

1. Está trabajando en un proyecto de Aprendizaje automático y su conjunto de datos contiene una cantidad significativa de valores perdidos. ¿Cuál es el paso inicial crucial para abordar este problema con eficacia? Seleccione la mejor respuesta.

- ○ Descartar todas las filas que contengan valores omitidos
- ◉ Investigar sistemáticamente la naturaleza y el patrón de los datos que faltan
- ○ Sustituir aleatoriamente los valores que faltan por valores de otras filas
- ○ Sustituya todos los valores que faltan por la media de sus respectivas columnas

Correcto
Correcto Comprender el patrón puede ayudar a determinar la estrategia de limpieza más adecuada.

2. Tiene dos DataFrames: uno contiene información sobre los clientes y el otro contiene su historial de pedidos. Necesita combinar estos DataFrames en un único DataFrame para analizar la relación entre los datos demográficos del cliente y su comportamiento de compra. ¿Cuál de las siguientes funciones se puede utilizar para lograr esto en Pandas? Seleccione todas las que correspondan.

- ☑ fusionar()

Correcto
Correcto La función merge() se utiliza para combinar objetos DataFrame realizando una unión de tipo base de datos.

- ☑ unir()

Correcto
Correcto La función join() se utiliza para unir columnas de otro DataFrame.

- ☐ combinar_primero()
- ☑ concat()

Correcto
Correcto La función concat() se utiliza para concatenar objetos pandas a lo largo de un eje determinado.

3. Usted está preparando un conjunto de datos para su análisis e identifica algunos valores perdidos que cree que pueden estimarse razonablemente basándose en los patrones de datos existentes. ¿Qué método de Pandas sería el más adecuado para rellenar estos valores perdidos utilizando interpolación lineal? Seleccione la mejor respuesta.

- (●) interpolar()
- ( ) fillna()
- ( ) dropna()
- ( ) isna()

Correcto
Correcto El método interpolate() puede utilizarse para realizar interpolación lineal u otros tipos de interpolación para estimar valores perdidos.

4. Mientras explora un conjunto de datos, observa que unos pocos puntos de datos parecen anormalmente distantes de la mayoría de los datos. Sospecha que se trata de valores atípicos. ¿Qué función de Pandas le ayudaría a confirmar su sospecha y a localizar estos valores atípicos? Seleccione la mejor respuesta.

- (●) cuantil()
- ( ) cuenta_valores()
- ( ) corr()
- ( ) describir()

quantile()

Correcto
Correcto El método quantile() puede utilizarse para detectar valores atípicos identificando valores en cuantiles específicos.

5. Está trabajando con un conjunto de datos que contiene una columna que representa el precio de los productos. Sin embargo, esta columna se almacena actualmente como texto e incluye algunas entradas con caracteres no numéricos. ¿Qué pasos seguiría para preparar estos datos para su conversión a un tipo de datos numérico en pandas? Seleccione todo lo que corresponda.

- [x] Utilización de la función astype()

Correcto
¡Correcto! La función astype() se utiliza habitualmente para la conversión de tipos de datos en pandas.

- [x] Comprobación de valores nulos antes de la conversión

Correcto
Correcto Es importante comprobar si hay valores nulos antes de la conversión del tipo de datos para evitar errores.

- [x] Garantizar que las columnas numéricas tengan el formato correcto

Correcto
Correcto Asegurarse de que las columnas numéricas tienen el formato correcto es esencial para una conversión precisa del tipo de datos.

- [ ] Eliminación de todas las filas con valores omitidos

6. Está explorando un conjunto de datos que contiene información sobre el rendimiento de los alumnos en una clase. Desea obtener rápidamente un resumen de la tendencia central, la dispersión y la forma de la distribución de las variables numéricas del conjunto de datos, como las calificaciones de los exámenes y las tareas. ¿Qué función de Pandas utilizarías para generar estos estadísticos descriptivos? Seleccione la mejor respuesta.

- ○ recuentos_de_valores
- ◉ describa
- ○ información
- ○ resumen

Correcto
Correcto La función describe calcula las estadísticas descriptivas de un DataFrame.

7. Ha recibido un conjunto de datos en forma de Archivo CSV. Para comenzar su análisis en Python, necesita importar este conjunto de datos a un DataFrame de pandas. ¿Qué función específica de Pandas emplearía para leer el Archivo CSV y crear el DataFrame? Seleccione la mejor respuesta.

- ○ pd.DataFrame()
- ○ pd.Series()
- ◉ pd.read_csv()
- ○ pd.read_excel()

Correcto
Correcto La función read_csv se utiliza para leer un Archivo CSV en un DataFrame.

8. Tienes un DataFrame de pandas que contiene datos de clientes con columnas como 'customer_id', 'city', y 'purchase_amount'. ¿Cuál de las siguientes operaciones le permitiría extraer un subconjunto específico de estos datos? Seleccione todas las que correspondan.

- [x] Utilizando `.iloc` para seleccionar las 10 primeras filas del DataFrame, independientemente de los detalles del cliente

  Correcto
  Correcto: `.iloc` permite la selección basada en posiciones de fila basadas en números enteros (segmentación).

- [ ] Calcular el importe_de_compra medio de cada "ciudad".

- [x] Utilizando `.loc` para seleccionar filas de clientes con valores específicos de 'customer_id'

  Correcto
  Correcto! `.loc` permite la selección basada en etiquetas de fila (que podría ser customer_id).

- [x] Utilización de la indexación booleana para seleccionar las filas en las que el "importe_compra" es superior a un determinado umbral

  Correcto
  Correcto Se trata de una forma de filtrado basada en una condición.

9. Imagina que tienes la tarea de implementar un algoritmo de Aprendizaje automático que requiere la manipulación y el procesamiento eficiente de grandes matrices numéricas. ¿Qué tipo de datos de Python sería el más adecuado para representar estas matrices, garantizando un rendimiento y una eficiencia computacional óptimos? Selecciona la mejor respuesta.

- ○ pandas DataFrame
- ○ Lista de listas
- ○ Diccionarios anidados
- ◉ Matriz NumPy

Correcto
¡Correcto! Las matrices NumPy están diseñadas específicamente para operaciones numéricas de alto rendimiento.

10. Estás aprendiendo sobre diccionarios en Python y quieres entender las diferentes formas de crearlos. ¿Cuáles de los siguientes métodos son válidos para crear un diccionario? Selecciona todos los que correspondan.

- ☑ Uso de llaves {}

Correcto
Correcto Se pueden utilizar llaves para definir un diccionario.

- ☐ Uso de paréntesis ()
- ☐ Utilizando corchetes []
- ☑ Uso de la función dict()

Correcto
Correcto La función dict() puede utilizarse para crear un diccionario.

Diálogo final

Durante la sesión de hoy, que cubrió la revisión de la lección "Uso de pandas para la limpieza y exploración", nos enfocamos en la identificación de errores, la corrección de errores, la mejora del rendimiento y el manejo de casos extremos.

Tus fortalezas:

- Identificaste con precisión los tipos de errores comunes en los datos (valores atípicos, faltantes, inconsistentes, duplicados).

- Demostraste un excelente conocimiento de las funciones de pandas para identificar y corregir estos errores, incluyendo `quantile()`, `describe()`, `isnull().sum()`, `fillna()`, `duplicated()` y `drop_duplicates()`.
- Articulaste claramente la importancia de la optimización del rendimiento y las estrategias para lograrla, como las operaciones vectorizadas y la optimización de tipos de datos.
- Proporcionaste un ejemplo muy pertinente de un caso extremo y sugeriste una solución adecuada con `clip()`.