GUSTAVO CORONEL
DESARROLLA SOFTWARE

CIENCIA DE DATOS:
TALLER DE FUNDAMENTOS DE MACHINE LEARNING CON PYTHON

Módulo 03
ANÁLISIS DE DATOS ESTRUCTURADOS CON PANDAS: Series y DataFrames

Dr. Eric Gustavo Coronel Castillo
Docente UNI
gcoronel@uni.edu.pe

ÍNDICE

1 Presentación

2 Objetivo

2.1 Objetivo general

2.2 Objetivos específicos

3 Introducción a Pandas y su Ecosistema

Pandas es una librería de código abierto para el lenguaje Python, diseñada específicamente para la manipulación y el análisis de datos estructurados. Fue creada por Wes McKinney en 2008, mientras trabajaba en AQR Capital Management, con el propósito de disponer de una herramienta flexible y eficiente para trabajar con datos relacionales y con series temporales en Python (McKinney, 2022).

El nombre **pandas** proviene del término econométrico "panel data", que hace referencia a conjuntos de datos multidimensionales que involucran mediciones en el tiempo (McKinney, 2022). Actualmente, pandas es una de las librerías más utilizadas en el ecosistema de ciencia de datos con Python, junto con NumPy, Matplotlib y Scikit-learn.

> ***Nota:*** *pandas se construye sobre NumPy, por lo que los arreglos ndarray subyacen en las estructuras de datos de pandas. Esto le permite aprovechar operaciones vectorizadas de alto rendimiento (pandas Development Team, 2024).*

La librería ofrece dos estructuras de datos principales:

- Series: estructura unidimensional, etiquetada, similar a un arreglo o un diccionario.
- DataFrame: estructura bidimensional tabular con filas y columnas etiquetadas.

4 Data Series

4.1 ¿Qué es una pandas Series?

Una Series es un arreglo unidimensional que contiene una secuencia de valores y un índice asociado. Cada valor de la Series está vinculado a una etiqueta de índice, lo que permite acceder a los datos de forma flexible y expresiva (McKinney, 2022).

Formalmente, se puede pensar en una Series como una columna de una hoja de cálculo o como un diccionario ordenado de Python. Sin embargo, a diferencia de un diccionario, la Series mantiene el orden y admite operaciones numéricas vectorizadas.

Según VanderPlas (2023, p. 97), la Series puede entenderse como una generalización de los arreglos de NumPy, en la que el índice no necesita ser un número entero, sino que puede ser cualquier tipo de dato hashable (cadenas de texto, fechas, etc.).

4.1.1 Estructura de una Series

La estructura de una Series consta de componentes esenciales:

Componente	Descripción
Índice (index)	Etiquetas asociadas a cada elemento. Por defecto son enteros 0, 1, 2, ... pero pueden personalizarse.
Valores (values)	Los datos almacenados, que corresponden a un arreglo NumPy subyacente.
dtype	El tipo de dato de los valores (int64, float64, object, bool, datetime64, etc.).
name	Atributo opcional que identifica a la Serie (útil cuando forma parte de un DataFrame.)

4.2 Instalación e Importación de pandas

Si aún no tiene pandas instalado, puede hacerlo desde la terminal o desde una celda de Jupyter Notebook:

```
# Instalación desde terminal (con Anaconda)
conda install pandas

# Instalación con pip
pip install pandas
Para importar pandas en su script o notebook, la convención estándar de la comunidad es usar el alias pd (pandas Development Team, 2024):
import pandas as pd
import numpy as np  # Se recomienda también importar NumPy

# Verificar la versión instalada
print(pd.__version__)  # Ejemplo de salida: 2.2.1
```

4.3 Creación de una Series

La función constructora es `pd.Series(data, index, dtype, name)`, donde todos los parámetros, excepto data, son opcionales (pandas Development Team, 2024). A continuación, se detallan los métodos de creación más frecuentes.

4.3.1 Desde una lista de Python

La forma más directa de crear una Series es a partir de una lista. El índice se asigna automáticamente como 0, 1, 2, ...

```
import pandas as pd

# Crear una Series desde una lista
notas = pd.Series([15, 18, 12, 20, 14])
print(notas)
```

Resultado

```
0    15
1    18
2    12
3    20
4    14
dtype: int64
```

> ***Nota:*** *El dtype int64 indica que pandas infirió automáticamente que los valores son enteros de 64 bits. Este comportamiento es consistente con NumPy (McKinney, 2022).*

4.3.2 Desde una lista con índice personalizado

Es posible asignar etiquetas descriptivas al índice para hacer más legible el acceso a los datos:

```
alumnos = ['Ana', 'Luis', 'María', 'Carlos', 'Elena']
notas = pd.Series([15, 18, 12, 20, 14], index=alumnos, name='Notas_Finales')
print(notas)
```

Resultado

```
Ana       15
Luis      18
María     12
Carlos    20
Elena     14
Name: Notas_Finales, dtype: int64
```

4.3.3 Desde un diccionario de Python

Cuando se crea una Series desde un diccionario, las claves se convierten en el índice y los valores se convierten en los datos (VanderPlas, 2023):

```
poblacion = {
    'Lima':      10900000,
    'Arequipa':   1080000,
    'Trujillo':    990000,
    'Chiclayo':    600000
}

serie_poblacion = pd.Series(poblacion, name='Población_2023')
print(serie_poblacion)
```

Resultado

```
Lima        10900000
Arequipa     1080000
Trujillo      990000
Chiclayo      600000
Name: Población_2023, dtype: int64
```

4.3.4 Desde un escalar (valor constante)

Cuando se pasa un escalar como dato, pandas replica el valor para cada elemento del índice especificado:

```
# Crear una Series donde todos los valores son 0.0
serie_cero = pd.Series(0.0, index=['x', 'y', 'z', 'w'])
print(serie_cero)
```

Resultado

```
x    0.0
y    0.0
z    0.0
w    0.0
dtype: float64
```

4.3.5 Desde un arreglo NumPy

Dado que pandas se construye sobre NumPy, es posible crear una Series directamente desde un ndarray:

```
import numpy as np

arr = np.array([3.5, 7.1, 2.8, 9.3])
serie_np = pd.Series(arr, index=['a', 'b', 'c', 'd'], name='Mediciones')
print(serie_np)
```

Resultado

```
a    3.5
b    7.1
c    2.8
d    9.3
Name: Mediciones, dtype: float64
```

4.4 Atributos Esenciales de una Series

Los atributos permiten inspeccionar las propiedades de una Series sin modificarla. Los más relevantes son (pandas Development Team, 2024):

Atributo	Tipo de retorno	Descripción
.values	numpy.ndarray	Arreglo NumPy con los datos.
.index	pandas.Index	Etiquetas del índice.
.dtype	dtype	Tipo de dato de los valores.
.size	int	Número total de elementos.
.shape	tuple	Tupla de dimensiones, p.ej. (5,).
.name	str / None	Nombre asignado a la Serie.
.is_unique	bool	True si todos los valores son únicos.
.hasnans	bool	True si existen valores NaN.

Ejemplo

```
s = pd.Series([10, 20, 30, 40, 50],
              index=['a', 'b', 'c', 'd', 'e'],
              name='ejemplo')

print('Valores:  ', s.values)   # ndarray de NumPy
print('Índice:   ', s.index)    # RangeIndex o Index
print('Tipo:     ', s.dtype)    # tipo de dato
print('Tamaño:   ', s.size)     # número de elementos
print('Forma:    ', s.shape)    # tupla (n,)
print('Nombre:   ', s.name)     # nombre de la Serie
```

Resultado

```
Valores:   [10 20 30 40 50]
Índice:    Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
Tipo:      int64
Tamaño:    5
Forma:     (5,)
Nombre:    ejemplo
```

4.5 Acceso e Indexación

pandas ofrece múltiples mecanismos de acceso a los elementos de una Series. McKinney (2022) distingue tres formas principales.

4.5.1 Acceso por etiqueta con .loc[]

`.loc[]` permite seleccionar elementos por su etiqueta de índice. Es el método recomendado cuando se trabaja con índices con nombres:

```
precios = pd.Series({'manzana': 2.5, 'pera': 3.0,
                     'uva': 5.5, 'mango': 4.0})

# Acceso a un solo elemento
print(precios.loc['pera'])       # 3.0
print("-" * 20)

# Acceso a múltiples elementos (lista de etiquetas)
print(precios.loc[['manzana', 'mango']])
print("-" * 20)

# Slicing por etiqueta (incluye extremo final)
print(precios.loc['manzana':'uva'])
```

Resultado

```
3.0
--------------------
manzana    2.5
mango      4.0
dtype: float64
--------------------
manzana    2.5
pera       3.0
uva        5.5
dtype: float64
```

4.5.2 Acceso por posición con .iloc[]

`.iloc[]` permite acceder a los elementos por su posición numérica (0-based), independientemente del índice:

```
precios = pd.Series({'manzana': 2.5, 'pera': 3.0,
                     'uva': 5.5, 'mango': 4.0})

# Acceso a un solo elemento
print(precios.loc['pera'])       # 3.0
print("-" * 20)

# Acceso a múltiples elementos (lista de etiquetas)
print(precios.loc[['manzana', 'mango']])
print("-" * 20)

# Slicing por etiqueta (incluye extremo final)
print(precios.loc['manzana':'uva'])
```

Resultado

```
3.0
--------------------
manzana    2.5
mango      4.0
dtype: float64
--------------------
manzana    2.5
pera       3.0
uva        5.5
dtype: float64
```

> ***Nota:*** *Diferencia clave:* `.loc[]` *incluye el extremo final en el slicing, mientras que* `.iloc[]` *lo excluye, siguiendo la convención de Python (McKinney, 2022).*

4.5.3 Indexación booleana (filtrado condicional)

La indexación booleana permite filtrar elementos que cumplan una condición lógica:

```
# Seleccionar frutas con precio mayor a 3.0
print(precios[precios > 3.0])
print("-"*20)

# Condición compuesta: precio entre 3.0 y 5.0
print(precios[(precios >= 3.0) & (precios <= 5.0)])
```

Resultado

```
uva      5.5
mango    4.0
dtype: float64
--------------------
pera     3.0
mango    4.0
dtype: float64
```

4.6 Operaciones con Series

4.6.1 Operaciones aritméticas vectorizadas

Una de las ventajas más importantes de pandas es que las operaciones aritméticas se aplican de forma vectorizada a todos los elementos, sin necesidad de bucles (VanderPlas, 2023):

```
import numpy as np

ventas = pd.Series([1500, 2300, 1800, 3100], index=['Q1', 'Q2', 'Q3', 'Q4'])
print(ventas)
print("-"*20)

# Multiplicar todos los valores por 1.1 (incremento del 10%)
ventas_nuevas = ventas * 1.1
print(ventas_nuevas)
print("-"*20)

# Aplicar función matemática
print(np.sqrt(ventas))
```

4.6.2 Alineación automática por índice

Pandas alinea automáticamente los elementos por su índice al realizar operaciones entre dos Series. Si un índice no existe en alguna de las series, el resultado es NaN (Not a Number) (McKinney, 2022):

```
s1 = pd.Series({'a': 10, 'b': 20, 'c': 30})
s2 = pd.Series({'b': 5,  'c': 15, 'd': 25})

print(s1 + s2)
```

> ***Nota:*** *La alineación automática es análoga al comportamiento de un JOIN en bases de datos relacionales: se alinean los registros por la clave común y se producen nulos donde no hay correspondencia (McKinney, 2022).*

4.6.3 Métodos estadísticos descriptivos

La Series incluye métodos integrados para el cálculo de estadísticas descriptivas:

```
s = pd.Series([85, 92, 78, 95, 88, 76, 91, 83])

print(s.describe())    # Resumen estadístico completo
print("-" * 20)

print('Media:  ', s.mean())
print('Mediana:', s.median())
print('Std:    ', s.std())
print('Min:    ', s.min())
print('Max:    ', s.max())
```

4.7 Manejo de Valores Faltantes (NaN)

En el análisis de datos real, es frecuente encontrar valores faltantes. pandas representa estos valores con NaN (del inglés Not a Number), que es en realidad un valor de punto flotante especial de IEEE 754 (McKinney, 2022).

```
import numpy as np

s = pd.Series([10.0, np.nan, 30.0, np.nan, 50.0])

print(s.isnull())         # Detectar NaN (True donde hay NaN)
print(s.notnull())        # Detectar no-NaN
print(s.dropna())         # Eliminar NaN
print(s.fillna(0))        # Reemplazar NaN con 0
print(s.fillna(s.mean())) # Reemplazar NaN con la media
```

4.8 Métodos y Transformaciones Útiles

4.8.1 Método .apply() — Aplicar funciones personalizadas

El método `.apply()` permite aplicar cualquier función a cada elemento de la Series:

```
import pandas as pd

notas = pd.Series([15, 18, 12, 20, 9, 14])

# Clasificar notas con una función lambda
clasificacion = notas.apply(
    lambda x: 'Aprobado' if x >= 11 else 'Desaprobado')
print(clasificacion)

print("-" * 20)
print("Juntar como columnas")
df = pd.concat([notas, clasificacion], axis=1)
print(df)

print("-" * 20)
print("Creando un DataFrame")
df = pd.DataFrame({
    'Nota': notas,
    'Estado': clasificacion
})
print(df)
```

4.8.2 Método .map() — Transformaciones con diccionario o función

`.map()` es útil para transformar valores de una Series usando un diccionario de mapeo o una función:

```
genero = pd.Series(['M', 'F', 'M', 'F', 'M'])
mapa = {'M': 'Masculino', 'F': 'Femenino'}
print(genero.map(mapa))
```

4.8.3 Ordenamiento

```
ventas = pd.Series({'norte': 850, 'sur': 1200, 'este': 630, 'oeste': 980})

print(ventas.sort_values())              # Ascendente (default)
print("-" * 20)
print(ventas.sort_values(ascending=False))  # Descendente
print("-" * 20)
print(ventas.sort_index())               # Ordenar por índice
```

4.8.4 Conteo de valores únicos

```
departamentos = pd.Series(['Lima','Cusco','Lima','Arequipa','Cusco','Lima'])

print(departamentos.unique())        # Array de valores únicos
print("-" * 20)
print(departamentos.nunique())       # Cantidad de valores únicos
print("-" * 20)
print(departamentos.value_counts())  # Frecuencia de cada valor
```

4.9 Conversión de Tipos de Datos

Es frecuente necesitar convertir el tipo de dato de una Series mediante el método `.astype()` (pandas Development Team, 2024):

```
s = pd.Series(['1', '2', '3', '4'])  # dtype: object (texto)
s_int = s.astype(int)                # Convertir a entero
s_float = s.astype(float)            # Convertir a decimal

print(s.dtype)       # object
print(s_int.dtype)   # int64
print(s_float.dtype) # float64
```

4.10 Visualización Rápida de una Series

Pandas integra una interfaz de trazado basada en `Matplotlib` que permite generar gráficos rápidos directamente desde una Series con el método `.plot()` (pandas Development Team, 2024):

```
import matplotlib.pyplot as plt

ventas = pd.Series(
    [1500, 2300, 1800, 3100, 2700],
    index=['Ene', 'Feb', 'Mar', 'Abr', 'May'],
    name='Ventas Mensuales (S/.)'
)

# Gráfico de línea (default)
ventas.plot(kind='line', title='Ventas Mensuales', marker='o', color='steelblue')
plt.ylabel('Ventas (S/.')
plt.tight_layout()
plt.show()

# Gráfico de barras
ventas.plot(kind='bar', title='Ventas por Mes', color='coral')
plt.tight_layout()
plt.show()
```

> ***Nota:*** *Los tipos de gráfico disponibles con `.plot()` incluyen: 'line', 'bar', 'barh', 'hist', 'pie', 'box', 'area', entre otros. Se explorará a fondo en el Módulo 5.*

4.11 Resumen: Referencia Rápida de la Series

Operación	Código
Crear desde lista	pd.Series([1, 2, 3])
Crear desde dict	pd.Series({'a':1, 'b':2})
Con índice	pd.Series(data, index=[...])
Acceso por etiqueta	s.loc['etiqueta']
Acceso por posición	s.iloc[0]
Filtro booleano	s[s > 5]
Estadísticas	s.describe() / s.mean()
Detectar NaN	s.isnull() / s.notnull()
Eliminar NaN	s.dropna()
Rellenar NaN	s.fillna(valor)
Aplicar función	s.apply(lambda x: ...)
Mapear valores	s.map({'a': 1})
Ordenar valores	s.sort_values()
Contar únicos	s.value_counts()
Convertir tipo	s.astype(float)
Graficar	s.plot(kind='bar')

4.12 Detección y Manejo de Datos Atípicos (Outliers)

Un dato atípico (outlier) es una observación que se aleja considerablemente del patrón general de los datos. Su presencia puede distorsionar medidas estadísticas como la media y la desviación estándar, e impactar negativamente en modelos de análisis posteriores (Tukey, 1977, citado en McKinney, 2022). Por ello, su detección y tratamiento son pasos obligatorios en cualquier proceso de limpieza de datos.

Es importante diferenciar los outliers de los valores faltantes: mientras que un NaN indica ausencia de información, un outlier es un valor presente pero extremo. Ambos requieren estrategias de tratamiento distintas (pandas Development Team, 2024).

4.12.1 Método 1: Rango Intercuartílico (IQR)

El método IQR (Interquartile Range) es la técnica robusta más utilizada para detectar outliers, propuesta originalmente por Tukey (1977). Se basa en los cuartiles de la distribución y no se ve tan afectado por los valores extremos como la media. Se definen los siguientes límites (VanderPlas, 2023):

- IQR = Q3 − Q1 (diferencia entre el tercer y primer cuartil)
- Límite inferior: Q1 − 1.5 × IQR
- Límite superior: Q3 + 1.5 × IQR
- Todo valor fuera de ese rango se considera outlier.

Ejemplo

```
import pandas as pd

# Datos de salarios mensuales (S/.) con valores extremos
salarios = pd.Series([1800, 2100, 1950, 2300, 1700, 15000, 2050, 1900, 2400, 200])

# Calcular Q1, Q3 e IQR
Q1  = salarios.quantile(0.25)
Q3  = salarios.quantile(0.75)
IQR = Q3 - Q1

# Definir limites
lim_inf = Q1 - 1.5 * IQR
lim_sup = Q3 + 1.5 * IQR

print(f"Q1={Q1}, Q3={Q3}, IQR={IQR}")
print(f"Limite inf={lim_inf:.1f}, Limite sup={lim_sup:.1f}")

# Detectar outliers
outliers = salarios[(salarios < lim_inf) | (salarios > lim_sup)]
print("Outliers detectados:\n", outliers)
```

4.12.2 Método 2: Puntuación Z (Z-Score)

El Z-Score mide cuántas desviaciones estándar se aleja un valor de la media. La convención más extendida es considerar como outlier todo valor con `|Z| > 3`, lo que corresponde a una probabilidad de ocurrencia inferior al 0.3% bajo una distribución normal (VanderPlas, 2023).

Su fórmula es:

$$Z = \frac{x - \mu}{\sigma}$$

Donde:

- Z — Es el Z-score o puntaje estándar.
- X — Es el valor actual.
- μ — Valor promedio.
- σ — Desviación estandar

Ejemplo

```
import pandas as pd

# Datos de salarios mensuales (S/.) con valores extremos
salarios = pd.Series([1800, 2100, 1950, 2300, 1700, 15000, 2050, 1900, 2400, 200])

print(f"Promedio: {salarios.mean()}")
print(f"Desviacion estandar: {salarios.std()}")

# Z-Score calculado directamente con pandas (sin scipy)
z_scores = (salarios - salarios.mean()) / salarios.std()

# Identificar outliers con |Z| > 3
outliers_z = salarios[z_scores.abs() > 3]
print('Z-Scores:\n', z_scores.round(2))
print('Outliers por Z-Score:\n', outliers_z)
```

4.12.2.1 Limitación del Z-Score: el efecto de enmascaramiento

El Z-Score presenta una limitación estructural que es importante comprender antes de aplicarlo: utiliza la media (μ) y la desviación estándar (σ) calculadas sobre el conjunto completo de datos, incluyendo los propios valores atípicos que se pretende detectar. Esto genera un efecto denominado enmascaramiento (masking effect), descrito por Barnett y Lewis (1994) y citado en la literatura estadística aplicada a ciencia de datos (VanderPlas, 2023).

El mecanismo del enmascaramiento opera de la siguiente manera: cuando existen uno o más outliers de gran magnitud en el conjunto de datos, estos elevan la media e inflan la desviación estándar. Como consecuencia, la distancia del propio outlier a la media —medida en unidades de desviación estándar— se reduce artificialmente, haciendo que su puntuación Z no supere el umbral de detección. En otras palabras, el valor extremo "contamina" los parámetros estadísticos que sirven para detectarlo, volviendo el método ciego ante su propia presencia.

> ***Concepto clave — Efecto de enmascaramiento:*** *un outlier de gran magnitud desplaza la media y amplifica la desviación estándar, reduciendo su propio Z-Score. Cuanto más extremo es el valor, mayor es su influencia sobre los parámetros, y menor resulta aparentemente su puntuación Z.*

Este efecto se intensifica bajo las siguientes condiciones, donde el Z-Score pierde confiabilidad como detector de outliers (McKinney, 2022; VanderPlas, 2023):

- Tamaño de muestra pequeño: con pocos datos, un solo valor extremo tiene un peso proporcional mayor sobre la media y la desviación estándar.
- Outliers múltiples: la presencia de varios valores atípicos acumula su efecto sobre los parámetros, enmascarando a todos ellos simultáneamente.
- Distribuciones asiétricas: cuando los datos no siguen una distribución normal, el umbral $|Z| > 3$ pierde su justificación estadística y puede resultar demasiado permisivo o demasiado restrictivo según el caso.
- Outliers unilaterales muy extremos: un valor de magnitud muy superior al resto produce una desviación estándar tan grande que ningún otro valor del conjunto logra superar el umbral.

En contraste, el método IQR no se ve afectado por el enmascaramiento porque los cuartiles Q1 y Q3 son estadísticos robustos: su valor no cambia significativamente ante la presencia de valores extremos, ya que dependen de la posición relativa de los

datos y no de su magnitud absoluta. Por esta razón, el IQR es el método preferido cuando no se puede garantizar normalidad o cuando el tamaño de muestra es reducido (McKinney, 2022).

La recomendación práctica es aplicar ambos métodos de forma complementaria: si el IQR detecta outliers que el Z-Score no detecta, ello no significa que uno de los métodos esté equivocado, sino que probablemente el efecto de enmascaramiento está actuando y el IQR está siendo más sensible. La decisión final sobre qué valores tratar debe sustentarse también en el criterio de dominio del negocio o campo de estudio.

4.12.3 Estrategias de Tratamiento de Outliers

Una vez detectados los outliers, existen cuatro estrategias principales de tratamiento. La elección depende del contexto del negocio y de la causa raíz del valor extremo (McKinney, 2022):

4.12.3.1 Estrategia A: Eliminación

Se elimina el registro completo cuando se tiene certeza de que el valor es un error de captura o un registro irrelevante:

```
import pandas as pd

# Datos de salarios mensuales (S/.) con valores extremos
salarios = pd.Series([1800, 2100, 1950, 2300, 1700,
                      15000, 2050, 1900, 2400, 200])

# Calcular Q1, Q3 e IQR
Q1  = salarios.quantile(0.25)
Q3  = salarios.quantile(0.75)
IQR = Q3 - Q1

# Definir limites
lim_inf = Q1 - 1.5 * IQR
lim_sup = Q3 + 1.5 * IQR

# Conservar solo los valores dentro de los limites IQR
s_limpio = salarios[(salarios >= lim_inf) & (salarios <= lim_sup)]

# Reporte
print(f"Original:\n{salarios}")
print(f"Limpios:\n{s_limpio}")
```

4.12.3.2 Estrategia B: Imputación por la mediana

Se prefiere la mediana sobre la media para la imputación, porque la mediana es robusta a los propios outliers que se están tratando (McKinney, 2022):

```
mediana = salarios.median()
mascara_outlier = (salarios < lim_inf) | (salarios > lim_sup)
s_imputado = salarios.where(~mascara_outlier, other=mediana)

#Reporte
print(f"Mediana: {mediana}")
print(f"Mascara:\n{mascara_outlier}")
print(f"Imputado:\n{s_imputado}")
```

4.12.3.3 Estrategia C: Recorte (Winsorization / Capping)

El recorte reemplaza los valores extremos por los límites del rango aceptable, preservando el número de registros. pandas ofrece el método .clip() para este propósito (pandas Development Team, 2024):

```
# .clip() reemplaza valores fuera del rango por el limite correspondiente
s_recortado = salarios.clip(lower=lim_inf, upper=lim_sup)
print(s_recortado)
```

4.12.3.4 Estrategia D: Conversión a NaN para tratamiento posterior

Una práctica habitual es convertir los outliers en NaN para luego aplicarles las técnicas de la sección 4.7 (.fillna(), .dropna()), unificando el flujo de limpieza de datos:

```
import numpy as np
s_nan_outliers = salarios.where(~mascara_outlier, other=np.nan)
print(s_nan_outliers)
```

4.12.4 Comparación de Métodos y Cuándo Usar Cada Uno

La siguiente tabla sintetiza cuándo es más apropiado usar cada método de detección, de acuerdo con McKinney (2022) y VanderPlas (2023):

Criterio	Método IQR	Método Z-Score
Supuesto distribucional	Ninguno (no paramétrico)	Distribución normal
Robustez ante outliers	Alta (usa mediana/cuartiles)	Baja (media influenciada por outliers)
Umbral por defecto	Q1/Q3 ± 1.5 × IQR	|Z| > 3
Recomendado cuando	Datos asimétricos o con outliers múltiples	Datos aproximadamente normales
Implementación pandas	.quantile() + filtro booleano	Cálculo manual con .mean() y .std()

4.13 Ejercicios Propuestos

Los siguientes ejercicios tienen como propósito consolidar los conceptos desarrollados en esta unidad. Se recomienda resolverlos en Jupyter Notebook, documentando cada paso con celdas Markdown.

4.13.1 Ejercicio 1 — Creación y atributos

Cree una Series llamada temperaturas con los siguientes datos de temperatura promedio mensual (°C) para la ciudad de Lima, utilizando los nombres de los meses (Ene, Feb, ..., Jun) como índice:

Ene	Feb	Mar	Abr	May	Jun
23.5	24.1	22.8	20.3	18.1	16.5

Luego, imprima: (a) los valores como ndarray, (b) el índice, (c) el tipo de dato y (d) el tamaño.

4.13.2 Ejercicio 2 — Acceso e indexación

Con la Series temperaturas del ejercicio anterior:

1. Acceda a la temperatura de Marzo usando .loc[].
2. Obtenga las temperaturas de Febrero a Mayo usando slicing con .loc[].
3. Seleccione todos los meses con temperatura mayor a 20 °C usando indexación booleana.
4. Use .iloc[] para obtener los últimos dos elementos.

4.13.3 Ejercicio 3 — Operaciones y estadísticas

Se tienen los siguientes datos de ventas (en soles) de una tienda durante la primera semana del mes:

```
ventas = pd.Series([450, 380, 520, 610, 480, 700, 550],
                   index=['Lun','Mar','Mié','Jue','Vie','Sáb','Dom'])
```

1. Calcule el promedio, la mediana y la desviación estándar.
2. Incremente todas las ventas en un 8% (simular IGV incluido).
3. Identifique qué días la venta superó el promedio.
4. Ordene los valores de mayor a menor.

4.13.4 Ejercicio 4 — Valores faltantes

Cree la siguiente Series y realice las tareas indicadas:

```
import numpy as np
calificaciones = pd.Series([18, np.nan, 15, np.nan, 12, 20, np.nan, 14])
```

1. Determine cuántos valores faltantes hay.
2. Calcule la media ignorando los NaN.
3. Rellene los NaN con la media de los valores existentes.
4. Elimine los NaN y guarde el resultado en una nueva Series.

4.13.5 Ejercicio 5 — Caso integrador

Un investigador de mercado recopila datos sobre el precio (S/.) de cinco productos en tres tiendas distintas. Cree tres Series (una por tienda) y realice lo siguiente:

1. Calcule la diferencia de precios entre la Tienda A y la Tienda B para cada producto.
2. Determine en qué productos la Tienda C tiene el precio más bajo.
3. Genere un gráfico de barras comparando las tres tiendas para el mismo producto.

Datos sugeridos:

- manzana (A:2.5, B:2.8, C:2.3)
- pera (A:3.0, B:2.9, C:3.2)
- uva (A:5.5, B:5.0, C:5.8)
- mango (A:4.0, B:4.5, C:3.9)
- naranja (A:1.5, B:1.6, C:1.4).

5 7. Referencias

Jupyter Project. (2025). *Project Jupyter*. https://jupyter.org/

NumPy. (2025). *Array creation*. NumPy Documentation. https://numpy.org/doc/stable/user/basics.creation.html

NumPy. (2025). *Array methods*. NumPy Documentation. https://numpy.org/doc/stable/reference/arrays.ndarray.html

NumPy. (2025). *Installing NumPy*. NumPy Documentation. https://numpy.org/install/

NumPy. (2025). *News*. NumPy Official Website. https://numpy.org/news/

NumPy. (2025). *NumPy documentation*. NumPy Developers. . (2025). NumPy v2.4 Manual. . https://numpy.org/doc/stable/

NumPy. (2025). *NumPy quickstart*. NumPy Documentation. https://numpy.org/doc/stable/user/quickstart.html

NumPy. (2025). *numpy.array*. NumPy v2.4 Manual. https://numpy.org/doc/stable/reference/generated/numpy.array.html

NumPy. (2025). *numpy.ndarray*. NumPy v2.4 Manual. . (2025). . . https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

NumPy. (2025). *numpy.reshape*. NumPy v2.4 Manual. https://numpy.org/doc/stable/reference/generated/numpy.reshape.html

NumPy. (2025). *NumPy: the absolute basics for beginners*. NumPy Documentation. https://numpy.org/doc/stable/user/absolute_beginners.html

NumPy. (2025). *The N-dimensional array (ndarray)*. NumPy v2.4 Manual. https://numpy.org/doc/stable/reference/arrays.ndarray.html