PUCE Virtual
EDUCACIÓN FLEXIBLE

Adquisición, gestión y gobernanza de datos

Lenguajes de Preprocesamiento de datos

Clase 8

MAESTRÍA EN
SISTEMAS DE INFORMACIÓN
Mención Data Science

La excelencia no se improvisa

INTRODUCCIÓN

Los lenguajes de preprocesamiento de datos son herramientas esenciales en el ciclo de vida del análisis de datos, ya que permiten transformar, limpiar y preparar conjuntos de datos para su posterior análisis. Estos lenguajes facilitan la manipulación de datos a través de operaciones como la selección de variables, la eliminación de duplicados, el manejo de valores faltantes y la normalización de datos. Ejemplos destacados incluyen R, Python y SQL, que ofrecen bibliotecas y funciones específicas para realizar estas tareas de manera eficiente. La capacidad de automatizar el preprocesamiento es crucial para garantizar la calidad y consistencia de los datos, lo que, a su vez, impacta directamente en la fiabilidad de los modelos analíticos.

Además, el uso de lenguajes de preprocesamiento permite a los analistas y científicos de datos experimentar con diferentes enfoques antes de aplicar técnicas más complejas, como el *machine learning*. La flexibilidad y escalabilidad que ofrecen estas herramientas son fundamentales en entornos de trabajo con grandes volúmenes de datos, donde la agilidad en la preparación de los datos puede marcar la diferencia en la velocidad y efectividad de las decisiones basadas en datos. Así, dominar estos lenguajes no solo optimiza el flujo de trabajo, sino que también mejora la capacidad de extraer *insights* valiosos de los datos.

RDA2

Esquematizar metodologías apropiadas para la obtención de modelos inductivos (predictivos) y deductivos (descriptivos) que faciliten los procesos de toma de decisiones basadas en datos en las instituciones.

Clase 8: Lenguajes de Preprocesamiento de datos

8.1 Python como lenguaje de preprocesamiento de datos

Python se ha consolidado como uno de los lenguajes más utilizados en la ciencia de datos gracias a su facilidad de uso y a la riqueza de sus bibliotecas. Con su sintaxis clara, permite a los científicos de datos realizar tareas complejas de manera eficiente. Un ejemplo destacado es la biblioteca Pandas, que es fundamental para la manipulación y el análisis de datos. A continuación, se presenta un ejemplo de cómo cargar un conjunto de datos y realizar operaciones básicas de exploración:

Figura 1

Carga de datos con pandas

```python
import pandas as pd

# Cargar un conjunto de datos
df = pd.read_csv('data.csv')

# Mostrar las primeras filas del DataFrame
print(df.head())

# Describir las estadísticas del DataFrame
print(df.describe())
```

Fuente: Elaboración Propia

Este código carga un archivo CSV en un DataFrame de Pandas y muestra las primeras filas, así como un resumen estadístico de las columnas numéricas. Este tipo de exploración es crucial para entender la estructura y las características de los datos antes de aplicar técnicas de análisis más avanzadas (McKinney, 2010).

Además, Python es muy efectivo para la visualización de datos, lo que es esencial en el análisis de datos. Bibliotecas como Matplotlib y Seaborn permiten crear gráficos atractivos y significativos. A continuación, se presenta un ejemplo de cómo crear un gráfico de dispersión utilizando Matplotlib:

Figura 2

Visualización de datos con matplotlib

```
import matplotlib.pyplot as plt

# Crear un gráfico de dispersión
plt.scatter(df['variable_x'], df['variable_y'])
plt.title('Gráfico de Dispersión')
plt.xlabel('Variable X')
plt.ylabel('Variable Y')
plt.show()
```

Fuente: Elaboración Propia

Este fragmento de código genera un gráfico de dispersión entre dos variables, permitiendo visualizar la relación entre ellas. La visualización es una parte crucial del análisis de datos, ya que ayuda a identificar patrones, tendencias y anomalías en los datos (McKinney, 2010).

Tutorial: Notebooks en Anaconda Navigator

Anaconda Navigator es una interfaz gráfica que facilita el manejo de entornos y paquetes para Python y R, especialmente útil en ciencia de datos. Uno de sus componentes más destacados son los Jupyter Notebooks, que permiten a los usuarios crear documentos interactivos que combinan código ejecutable, visualizaciones y texto explicativo. A continuación, se presenta un tutorial básico para comenzar a usar notebooks en Anaconda Navigator.

Instalación de Anaconda

Primero, asegúrate de tener Anaconda instalado en tu sistema. Puedes descargarlo desde el sitio oficial de Anaconda. Sigue las instrucciones específicas para tu sistema operativo (Windows, macOS o Linux).

Iniciar Anaconda Navigator

Una vez instalado, abre Anaconda Navigator. Verás una interfaz con varias opciones. Desde aquí, puedes crear nuevos entornos y gestionar tus paquetes.

Abrir Jupyter Notebook

En la ventana principal de Anaconda Navigator, busca la opción "Jupyter Notebook" y haz clic en "Launch". Esto abrirá una nueva pestaña en tu navegador predeterminado, mostrando el dashboard de Jupyter.

Crear un Nuevo Notebook

En el dashboard de Jupyter, haz clic en "New" y selecciona "Python 3" (o la versión que tengas instalada) para crear un nuevo notebook. Se abrirá una nueva pestaña donde podrás escribir y ejecutar tu código.

Ejemplo de Código en un Notebook

A continuación, se presenta un sencillo ejemplo de cómo cargar y visualizar un conjunto de datos utilizando la biblioteca Pandas:

Figura 3

Carga y visualización de datos

```python
import pandas as pd
import matplotlib.pyplot as plt

# Cargar un conjunto de datos
df = pd.read_csv('data.csv')

# Mostrar las primeras filas del DataFrame
print(df.head())

# Crear un gráfico de dispersión
plt.scatter(df['variable_x'], df['variable_y'])
plt.title('Gráfico de Dispersión')
plt.xlabel('Variable X')
plt.ylabel('Variable Y')
plt.show()
```

Fuente: Elaboración Propia

Este código carga un archivo CSV y genera un gráfico de dispersión entre dos variables. Para ejecutar el código, simplemente selecciona la celda y presiona **Shift + Enter**.

Guardar y Compartir el Notebook

Para guardar tu trabajo, haz clic en "File" y luego en "Save and Checkpoint". También puedes exportar tu notebook a otros formatos, como PDF o HTML, seleccionando "File" > "Download as" y eligiendo el formato deseado.

8.1.1 Tipos de datos en Python y estructuras básicas

Python es un lenguaje de programación poderoso y flexible que proporciona una variedad de tipos de datos y estructuras básicas, lo que permite a los desarrolladores manejar información de manera eficiente. Entre los tipos de datos más utilizados se encuentran los enteros (int), los números de punto flotante (float), las cadenas de texto (str), las listas (list), las tuplas (tuple), los diccionarios (dict) y los conjuntos (set). Por ejemplo, los enteros se utilizan para representar números enteros, mientras que los números flotantes son ideales para cálculos que requieren decimales. A continuación, se presentan algunos ejemplos básicos:

Figura 4

Tipos de datos

```
# Tipos de datos
entero = 10                  # Tipo int
flotante = 10.5              # Tipo float
cadena = "Hola, Python!"     # Tipo str

print(entero, flotante, cadena)
```

Fuente: Elaboración Propia

Las estructuras de datos en Python son fundamentales para organizar y manipular información. Las listas son colecciones ordenadas y mutables, lo que significa que se pueden modificar después de su creación. Por ejemplo, se puede crear una lista de números y luego actualizarla:

Figura 5

Listas

```
# Lista
lista_numeros = [1, 2, 3, 4]
lista_numeros.append(5)      # Agregar un elemento
print(lista_numeros)          # Salida: [1, 2, 3, 4, 5]
```

Fuente: Elaboración Propia

Las tuplas son similares a las listas, pero son inmutables, lo que significa que no pueden modificarse después de su creación.

Figura 6

Tuplas

```
# Tupla
tupla_numeros = (1, 2, 3)
print(tupla_numeros)          # Salida: (1, 2, 3)
```

Fuente: Elaboración Propia

Los diccionarios son estructuras de datos clave-valor que permiten un acceso rápido a la información. Por ejemplo, se puede crear un diccionario que almacene la información de una persona:

Figura 7

Diccionarios

```
# Diccionario
persona = {
    "nombre": "Juan",
    "edad": 30,
    "ciudad": "Madrid"
}
print(persona["nombre"])      # Salida: Juan
```

Fuente: Elaboración Propia

Por último, los conjuntos son colecciones no ordenadas de elementos únicos, ideales para realizar operaciones matemáticas.

Figura 8

Sets de datos

```
# Conjunto
conjunto = {1, 2, 3, 4, 4}    # Los duplicados se eliminan
print(conjunto)                # Salida: {1, 2, 3, 4}
```

Fuente: Elaboración Propia

Conocer estos tipos de datos y estructuras básicas es esencial para cualquier desarrollador que trabaje con Python, ya que son la base para construir algoritmos y aplicaciones más complejas (Lutz, 2013).

Joins en Python

En Python, los "joins" son técnicas utilizadas para combinar datos de diferentes estructuras, como listas, diccionarios o DataFrames. Estas operaciones son esenciales para el análisis y la manipulación de datos, especialmente al trabajar con bibliotecas como Pandas, que simplifican tareas complejas y permiten un manejo eficiente de grandes volúmenes de datos (McKinney, 2018).

Joins en Listas

Para combinar listas en Python, se pueden utilizar métodos como extend() o el operador +. Sin embargo, estas técnicas son limitadas cuando se comparan con los joins en bases de datos. Por ejemplo, al trabajar con listas de diccionarios, se pueden aplicar comprensiones de lista para filtrar y combinar datos según ciertas condiciones. Este enfoque es muy útil para realizar joins simulados entre listas (Lutz, 2013).

Figura 9

Joins en listas

```
lista1 = [{'id': 1, 'nombre': 'Juan'}, {'id': 2, 'nombre': 'Ana'}]
lista2 = [{'id': 1, 'edad': 30}, {'id': 2, 'edad': 25}]

resultado = [{**x, **y} for x in lista1 for y in lista2 if x['id'] == y['id']]
print(resultado)
```

Fuente: Elaboración Propia

En este ejemplo, se combinan dos listas de diccionarios en función del campo 'id'. El resultado es una lista de diccionarios que contiene tanto el nombre como la edad de cada persona (Beazley, 2013).

Joins en Pandas

La biblioteca Pandas proporciona métodos avanzados para realizar joins de manera más eficiente. Funciones como merge(), join() y concat() permiten combinar DataFrames de diferentes maneras: joins internos, externos, por índices y más. Esto es particularmente útil para el análisis de datos complejos en entornos de ciencia de datos (VanderPlas, 2016).

Ejemplo de merge()

Figura 10

Mergue

```
import pandas as pd

df1 = pd.DataFrame({'id': [1, 2], 'nombre': ['Juan', 'Ana']})
df2 = pd.DataFrame({'id': [1, 2], 'edad': [30, 25]})

resultado = pd.merge(df1, df2, on='id')
print(resultado)
```

Fuente: Elaboración Propia

En este caso, pd.merge() realiza un join interno utilizando la columna 'id' como clave. El resultado es un nuevo DataFrame que contiene las columnas 'id', 'nombre' y 'edad' para cada registro que coincide (McKinney, 2018).

8.1.2 Tipos de Joins en Pandas

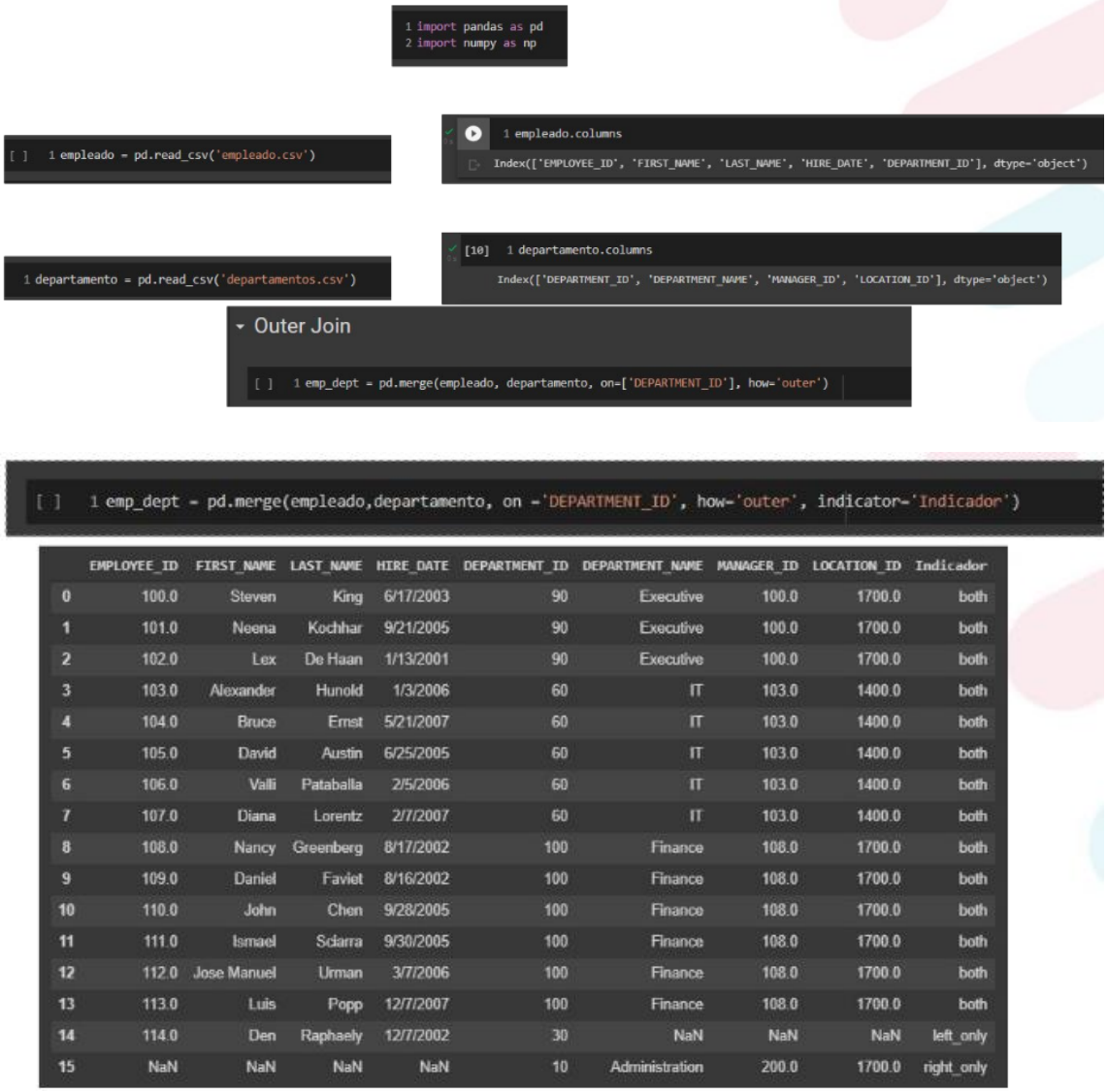
Pandas admite varios tipos de joins, que son fundamentales para la manipulación de datos:

- **Inner Join**: Devuelve solo las filas con claves coincidentes en ambos DataFrames.
- **Outer Join**: Devuelve todas las filas de ambos DataFrames, llenando con NaN donde no hay coincidencias.
- **Left Join**: Devuelve todas las filas del DataFrame izquierdo y las coincidencias del derecho.
- **Right Join**: Devuelve todas las filas del DataFrame derecho y las coincidencias del izquierdo (Ponnusamy, 2019).

A continuación, se muestra una imagen de ejemplo de todos los tipos de joins:

Figura 11

Joins



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID	Indicador
0	100.0	Steven	King	6/17/2003	90	Executive	100.0	1700.0	both
1	101.0	Neena	Kochhar	9/21/2005	90	Executive	100.0	1700.0	both
2	102.0	Lex	De Haan	1/13/2001	90	Executive	100.0	1700.0	both
3	103.0	Alexander	Hunold	1/3/2006	60	IT	103.0	1400.0	both
4	104.0	Bruce	Ernst	5/21/2007	60	IT	103.0	1400.0	both
5	105.0	David	Austin	6/25/2005	60	IT	103.0	1400.0	both
6	106.0	Valli	Pataballa	2/5/2006	60	IT	103.0	1400.0	both
7	107.0	Diana	Lorentz	2/7/2007	60	IT	103.0	1400.0	both
8	108.0	Nancy	Greenberg	8/17/2002	100	Finance	108.0	1700.0	both
9	109.0	Daniel	Faviet	8/16/2002	100	Finance	108.0	1700.0	both
10	110.0	John	Chen	9/28/2005	100	Finance	108.0	1700.0	both
11	111.0	Ismael	Sciarra	9/30/2005	100	Finance	108.0	1700.0	both
12	112.0	Jose Manuel	Urman	3/7/2006	100	Finance	108.0	1700.0	both
13	113.0	Luis	Popp	12/7/2007	100	Finance	108.0	1700.0	both
14	114.0	Den	Raphaely	12/7/2002	30	NaN	NaN	NaN	left_only
15	NaN	NaN	NaN	NaN	10	Administration	200.0	1700.0	right_only

Fuente: Elaboración Propia

Ejemplo de Outlier

Este ejemplo ilustra cómo se incluyen filas de ambos DataFrames, independientemente de si hay coincidencias. Esto es útil para analizar datos donde puede haber registros faltantes en uno de los conjuntos (VanderPlas, 2016).

Los joins son herramientas esenciales en Python para la manipulación y el análisis de datos. Al utilizar listas o la biblioteca Pandas, los joins permiten integrar diferentes fuentes de datos de manera efectiva, lo cual es crucial en el campo de la ciencia de datos y el análisis (McKinney, 2018).

8.1.3 Analíticos Descriptivos

Los análisis descriptivos son una parte fundamental del análisis de datos, ya que permiten resumir y comprender las características de un conjunto de datos. Estos análisis incluyen medidas de tendencia central, dispersión y visualización de datos, que son esenciales para extraer conocimientos iniciales y guiar decisiones posteriores. En Python, bibliotecas como Pandas y Matplotlib facilitan la realización de estos análisis de manera eficiente (McKinney, 2018).

Medidas de Tendencia Central

Las medidas de tendencia central, como la media, la mediana y la moda, son esenciales para describir el comportamiento general de un conjunto de datos. En Python, estas medidas se pueden calcular fácilmente utilizando Pandas.

Figura 12

Medidas de tendencia central

```python
import pandas as pd

data = {'valores': [10, 20, 20, 30, 40]}
df = pd.DataFrame(data)

media = df['valores'].mean()
mediana = df['valores'].median()
moda = df['valores'].mode()[0]

print(f'Media: {media}, Mediana: {mediana}, Moda: {moda}')
```

Fuente: Elaboración Propia

Este ejemplo muestra cómo calcular la media, la mediana y la moda de un conjunto de datos. La media proporciona un valor promedio, la mediana representa el valor central y la moda indica el valor más frecuente (VanderPlas, 2016).

Medidas de Dispersión

Las medidas de dispersión, como la varianza y la desviación estándar, ayudan a entender la variabilidad dentro de un conjunto de datos. Estas medidas son importantes para evaluar la confiabilidad y consistencia de los datos (Ponnusamy, 2019).

Figura 13

Medidas de dispersión

```
varianza = df['valores'].var()
desviacion_estandar = df['valores'].std()

print(f'Varianza: {varianza}, Desviación Estándar: {desviacion_estandar}')
```

Fuente: Elaboración Propia

En este caso, se calculan la varianza y la desviación estándar del conjunto de datos, lo que permite comprender la dispersión de los valores respecto a la media (McKinney, 2018).

8.1.4 Visualización de Datos

La visualización de datos es una herramienta clave en el análisis descriptivo, ya que permite representar gráficamente la información y facilita la identificación de patrones. Bibliotecas como Matplotlib y Seaborn son ampliamente utilizadas en Python para crear visualizaciones efectivas.

Figura 14

Histplot

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.histplot(df['valores'], bins=5, kde=True)
plt.title('Distribución de Valores')
plt.xlabel('Valores')
plt.ylabel('Frecuencia')
plt.show()
```

Fuente: Elaboración Propia

Los análisis descriptivos son fundamentales para el análisis inicial de datos, ya que permiten extraer información clave sobre un conjunto de datos. Utilizando Python y sus poderosas bibliotecas, los analistas pueden calcular medidas de tendencia central,

dispersión y realizar visualizaciones efectivas que facilitan la toma de decisiones basadas en datos (McKinney, 2018).

Análisis Exploratorio de Datos

El análisis exploratorio de datos consiste en realizar una **visión general** de las variables del conjunto de datos a analizar, con el objetivo de **obtener una comprensión inicial de los datos** con los que vamos a trabajar. Entre los análisis recomendados se encuentran:

- Analizar medidas de tendencia central y medidas de dispersión.
- Determinar la cantidad de valores nulos.
- Identificar el número de registros distintos.

Las visualizaciones comúnmente utilizadas en este tipo de análisis incluyen gráficos de cajas, histogramas, gráficos de dispersión, entre otros.

En el siguiente video, desde el minuto 1:00:00 hasta el 1:10:00, se muestra una librería para realizar análisis exploratorio de datos: Análisis exploratorio de datos.

Análisis de Componentes Principales

El análisis de componentes principales (PCA, por sus siglas en inglés) es un procedimiento matemático que transforma un conjunto de variables **correlacionadas** en un conjunto menor de variables **no correlacionadas**, denominadas componentes principales.

El PCA es una técnica **exploratoria** que no establece supuestos y puede aplicarse siempre. Su objetivo es encontrar **combinaciones lineales** de las variables originales que **maximicen la varianza** en el espacio generado por las variables originales, minimizando la pérdida de información lineal.

El análisis de componentes principales tiene varios objetivos y características clave:

- **Objetivo principal**: Reducir la cantidad de dimensiones al eliminar variables o información redundante.
- **Representación gráfica**: Permite representar de manera gráfica información **multidimensional**.
- **Normalización**: Las unidades de medición de las variables pueden influir en la variabilidad de las componentes **principales**. Por lo tanto, es conveniente normalizar las variables antes de aplicar el análisis.
- **Impacto de las unidades de medida**: Las unidades de medida de las variables pueden afectar el análisis. Por ejemplo, cambiar las unidades de medida de metros

a pulgadas (para la altura) o de kilogramos a libras (para el peso) podría generar resultados diferentes.

- **Estandarización o normalización**: Para evitar distorsiones por las unidades de medida, es recomendable **estandarizar** o **normalizar** los datos. Esto implica transformar los datos en rangos más pequeños, como [-1,1] o [0,1].

Formas de normalizar los datos

Min-Max Normalization: Supongamos que el valor mínimo y máximo del atributo "ingresos" son $12,000 y $98,000, respectivamente. Si queremos normalizar los datos en un rango de [0,1], el valor $73,600 se transformaría de la siguiente manera:

Figura 15

Min Max Normalization

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A.$$

$$\frac{73600-12000}{98000-12000} * (1-0) + 0 = 0.716$$

Fuente: Elaboración Propia

Z-Score Normalization

Supongamos que la media y la desviación estándar de los valores para los ingresos son $54,000 y $16,000, respectivamente. Usando la normalización Z-score, el valor de $73,600 para los ingresos se transformaría de la siguiente manera:

Figura 16

Z- Score Normalization

$$\frac{73600 - 54000}{16000} = 1.225$$

Fuente: Elaboración Propia

En el siguiente video, desde el minuto 1:20:00 hasta el 1:40:00, se muestra un ejemplo práctico de cómo implementar e interpretar las componentes principales: Componentes principales.

8.1.5 Ejemplos de aplicación de las componentes principales

8.1.5.1 Reconocimiento facial

El PCA se utiliza frecuentemente en el reconocimiento facial, donde ayuda a reducir la dimensionalidad de imágenes de alta resolución. Al transformar las imágenes en un espacio de menor dimensión, el PCA capta las características más relevantes de las caras, mejorando la eficiencia de los algoritmos de reconocimiento (Turk & Pentland, 1991).

8.1.5.2 Compresión de Datos

Otra aplicación del PCA es en la compresión de datos, como en archivos de audio y video. Al eliminar las componentes menos significativas, se puede reducir el tamaño del archivo mientras se conserva la mayor parte de la información. Esta técnica es especialmente útil en aplicaciones de streaming y almacenamiento (Wang et al., 2015).

8.1.5.3 Análisis de Clientes

En marketing, el PCA permite identificar patrones de comportamiento en grandes conjuntos de datos de clientes. Por ejemplo, en un análisis de segmentación, el PCA reduce las dimensiones de variables como hábitos de compra, edad y ubicación, facilitando la identificación de grupos de clientes similares (Hernández et al., 2015).

8.1.5.4 Bioinformática

En estudios genéticos, el PCA se aplica para analizar grandes conjuntos de datos de expresión génica. Al reducir la dimensionalidad, se facilita la identificación de patrones asociados con enfermedades, lo que puede ayudar en la investigación y diagnóstico (Ringner, 2008).

8.1.5.5 Detección de Anomalías

El PCA también se utiliza en la detección de fraudes, especialmente en el análisis de transacciones. Al reducir las dimensiones de las características de las transacciones, se pueden identificar fácilmente aquellas que se desvían de los patrones normales, lo que puede indicar posibles fraudes (Xia et al., 2015).

8.1.5.6 Visualización de Datos

En la visualización de datos multidimensionales, el PCA facilita la representación gráfica de conjuntos de datos con muchas variables. Al reducir las dimensiones a 2 o 3, se facilita la interpretación y el análisis visual de los datos (Gabriel, 1971).

8.1.5.7 Análisis de Sentimientos

En el análisis de texto, el PCA se utiliza para reducir la dimensionalidad de las características de texto, como los términos en un documento. Esto permite una representación más manejable de los datos textuales en tareas de clasificación o análisis de sentimientos (Blei et al., 2003).

Referencias

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research, 3*, 993-1022.

Gabriel, K. R. (1971). The biplot graphic display of matrices with application to principal component analysis. *Biometrics, 27*(3), 640-655.

Granger, B. E. (2017). Jupyter: A publishing system for reproducible scientific research. *Nature, 549*(7670), 1-2.

Hernández, L., Pineda, J., & Pérez, J. (2015). Segmentación de mercados mediante análisis de componentes principales y clustering. *Revista Latinoamericana de Administración, 11*(2), 25-41.

Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

McKinney, W. (2010). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media.

Pérez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science & Engineering, 9*(3), 21-29.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

VanderPlas, J. (2016). *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media.

Wes McKinney. (2010). *Data Analysis in Python*. O'Reilly Media.

PUCE Virtual
EDUCACIÓN FLEXIBLE

La excelencia no se improvisa

síguenos

MDASC - 03 - 251