TIC

Actividad 2

Pandas

Universidad Tecnológica de Pereira

faceIT

Por qué Pandas es esencial para el análisis de datos y manipulación de datos en IA

Estructuras de Datos Poderosas:

Pandas ofrece dos estructuras de datos principales: Series y DataFrames. Los DataFrames son especialmente poderosos y permiten organizar datos de manera tabular, similar a una hoja de cálculo, con las y columnas. Estas estructuras son exibles y e cientes para representar y manipular datos complejos.

Manipulación de Datos E ciente: Pandas proporciona una amplia gama de funciones para la manipulación de datos, como ltrado, selección, agregación y transformación. Estas operaciones son esenciales en el preprocesamiento de datos para el entrenamiento de modelos de aprendizaje automático.

Por qué Pandas es esencial para el análisis de datos y manipulación de datos en IA

Integración con NumPy:

Pandas se integra bien con NumPy, otra biblioteca esencial en el ámbito de la inteligencia arti cial. Los DataFrames de Pandas pueden contener columnas de tipo NumPy array, lo que facilita la integración de las funcionalidades de ambas bibliotecas.

Manejo de Datos Faltantes: El mundo real a menudo presenta conjuntos de datos con valores faltantes o nulos. Pandas ofrece herramientas para manejar estos casos, ya sea eliminando datos faltantes o llenándolos con valores especí cos.

Por qué Pandas es esencial para el análisis de datos y manipulación de datos en IA

Indexación Intuitiva:

La indexación en Pandas permite acceder y manipular datos de manera intuitiva. Puedes indexar DataFrames por etiquetas de la y columna, facilitando la referencia a datos especí cos y la realización de operaciones complejas.

Operaciones de Agrupación y Agregación: **Pandas facilita la realización de operaciones de agrupación y agregación en datos. Puedes agrupar datos en función de ciertos criterios y aplicar funciones de agregación, como sumas o promedios, a grupos especí cos.**

Por qué Pandas es esencial para el análisis de datos y manipulación de datos en IA

Entrada y Salida de Datos Simpli cada: **Pandas simpli ca la entrada y salida de datos en diversos formatos, como CSV, Excel, SQL, y más. Esto facilita la carga y la escritura de datos desde y hacia diferentes fuentes, lo cual es crucial en el ciclo de vida de un proyecto de inteligencia arti cial.**

Visualización de Datos Integrada: **Aunque no es la función principal de Pandas, la biblioteca proporciona herramientas para visualizar datos de manera rápida y sencilla, lo cual es útil en la exploración inicial de datos.**

Instalación de pandas

Para instalar Pandas, puedes usar pip, el administrador de paquetes de Python.

Abre tu terminal o símbolo del sistema y ejecuta el siguiente comando:

```
pip install pandas
```

Este comando descargará e instalará la última versión de Pandas en tu sistema. Una vez completada la instalación, podrás importar Pandas en tus scripts de Python para comenzar a trabajar con él.

La convención de importación para Pandas es utilizar el alias pd. Esto signi ca que al importar Pandas, se utiliza la palabra clave import seguida del nombre del paquete (pandas) y luego se asigna un alias (as pd). De esta manera, cada vez que necesites hacer referencia a una función o clase de Pandas en tu código, puedes utilizar el alias pd.

Presentar Series y DataFrames como estructuras principales de Datos en Pandas

En Pandas, las estructuras de datos fundamentales son las Series y los DataFrames.

Series: Una Serie es un arreglo unidimensional etiquetado capaz de contener cualquier tipo de dato. Puedes pensar en ella como una columna en una hoja de cálculo o como un conjunto unidimensional de datos. Etiquetas: Cada elemento en una Serie está asociado con una etiqueta o índice, que puede ser numérico o basado en etiquetas. Homogeneidad: Todos los elementos en una Serie deben ser del mismo tipo de datos. Creación de una Serie:

Presentar Series y DataFrames como estructuras principales de Datos en Pandas

Creación de una Serie:

```
import pandas as pd

data = [1, 2, 3, 4, 5]
etiquetas = ['a', 'b', 'c', 'd', 'e']

serie = pd.Series(data, index=etiquetas)
```

Otra forma de pensar sobre una serie es como un dict ordenado de longitud ja, ya que es un mapeo de valores de índice a valores de datos. Se puede usar en muchos contextos donde podría usar un dict:

Presentar Series y DataFrames como estructuras principales de Datos en Pandas

DataFrames: **Un DataFrame es una estructura bidimensional etiquetada con columnas que pueden contener diferentes tipos de datos. Puedes pensar en él como una hoja de cálculo o una tabla SQL.**

Columnas: **Cada columna en un DataFrame es esencialmente una Serie**

Índices: Los DataFrames tienen índices tanto para las como para columnas.

Creación de un DataFrame:

```
import pandas as pd

datos = {'Nombre': ['Alice', 'Bob', 'Charlie'],
         'Edad': [25, 30, 35],
         'Ciudad': ['A', 'B', 'C']}

df = pd.DataFrame(datos)
```

Presentar Series y DataFrames como estructuras principales de Datos en Pandas

Hay muchas formas de construir un DataFrame, aunque una de las más comunes es de un dict de listas de igual longitud o matrices numpy:

```
data = {'state':['Ohio','Ohio','Ohio','Nevada','Nevada','Nevada'],
'year': [2000, 2001, 2002, 2001, 2002, 2003],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
frame
```

Para grandes marcos de datos, el método head selecciona solo las primeras cinco las:

```
frame.head()
```

Posibles entradas de datos al constructor de DataFrame.

Ambas estructuras son esenciales en el análisis de datos y la manipulación en Pandas. Las Series son útiles para representar datos unidimensionales, mientras que los DataFrames permiten trabajar con datos bidimensionales, lo que facilita la manipulación y análisis de conjuntos de datos más complejos en el ámbito de la inteligencia arti cial y la ciencia de datos.

Función	Descripción
2D ndarray	Una matriz de datos, pasando etiquetas de fila y columna opcionales
dict of arrays, lists, or tuples	Cada secuencia se convierte en una columna en el DataFrame; todas las secuencias deben tener la misma longitud
NumPy structured/record array	Tratado como el caso "dict of arrays"
dict of Series	Cada valor se convierte en una columna; índices de cada serie se unen para formar el índice de la fila del resultado si no se pasa un índice explícito
dict of dicts	Cada dictado interno se convierte en una columna; las claves se unen para formar el índice de fila como en el "dict of Caso de la serie
List of dicts or Series	Cada elemento se convierte en una fila en el DataFrame; unión de claves dict o índices de serie se convierten en el Etiquetas de columna de DataFrame
List of lists or tuples	Tratado como el caso "2D ndarray"
Another DataFrame	Los índices de DataFrame se utilizan a menos que se pasen otros diferentes
NumPy MaskedArray	Como el caso "2D ndarray", excepto que los valores enmascarados se vuelven NA/faltan en el resultado de DataFrame

Presentar Series y DataFrames como estructuras principales de Datos en Pandas

Manejo de DataFrames en pandas Columnas de DataFrame **Si especi ca una secuencia de columnas, las columnas de DataFrame se organizarán en esa orden:**

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

Si pasa una columna que no está contenida en el dict, aparecerá con los valores faltantes en el resultado:

```
frame2 = pd.DataFrame(data, columns=['year','state','pop','debt'],
                      index=['one','two','three','four','five','six'])
frame2
frame2.columns
```

Presentar Series y DataFrames como estructuras principales de Datos en Pandas

llamar una columna: **Se puede llamar una columna en un DataFrame como una Serie, ya sea por notación similar a dict o por atributo:**

```
frame2['state']
frame2.year
```

rame2[columna] funciona para cualquier nombre de columnframe2.columna Solo funci ona cuando el nombre de la columna es una variable Python válida nombre.

Indexación Avanzada en Pandas

Mostrar cómo utilizar etiquetas y posiciones para indexar en Pandas. La indexación avanzada en Pandas es una característica poderosa que permite acceder y manipular datos de manera más especí ca utilizando etiquetas y posiciones.

Indexación por Etiquetas: **Para acceder a datos utilizando etiquetas, puedes usar los métodos loc[ ] o at[ ]. El siguiente ejemplo ilustra cómo seleccionar una la y una columna especí cas por etiquetas:**

```
import pandas as pd

datos = {'Nombre': ['Alice', 'Bob', 'Charlie'],
         'Edad': [25, 30, 35],
         'Ciudad': ['A', 'B', 'C']}

df = pd.DataFrame(datos, index=['a', 'b', 'c'])

# Seleccionar datos con loc[]
seleccion_loc = df.loc['b', 'Edad']
print(seleccion_loc)  # Resultado: 30

# Seleccionar datos con at[]
seleccion_at = df.at['b', 'Edad']
print(seleccion_at)  # Resultado: 30
```

Indexación Avanzada en Pandas

Indexación por Posiciones: **Para acceder a datos utilizando posiciones, puedes usar los métodos iloc[ ] o iat[ ]. Aquí hay un ejemplo de cómo seleccionar datos basados en posiciones:**

Estos métodos proporcionan exibilidad para acceder a datos especí cos en un DataFrame de Pandas, ya sea mediante etiquetas o posiciones numéricas. La elección entre ellos dependerá de tus necesidades especí cas de indexación en el análisis de datos.

```python
import pandas as pd

datos = {'Nombre': ['Alice', 'Bob', 'Charlie'],
         'Edad': [25, 30, 35],
         'Ciudad': ['A', 'B', 'C']}

df = pd.DataFrame(datos)

# Seleccionar datos con iloc[]
seleccion_iloc = df.iloc[1, 1]
print(seleccion_iloc)  # Resultado: 30

# Seleccionar datos con iat[]
seleccion_iat = df.iat[1, 1]
print(seleccion_iat)  # Resultado: 30
```

Operaciones Aritméticas en DataFrames de pandas

En Pandas, los DataFrames admiten una amplia variedad de operaciones aritméticas que facilitan el procesamiento y análisis de datos. Las operaciones aritméticas comunes en DataFrames:

Operaciones Elemento a Elemento: **Puedes realizar operaciones aritméticas elemento a elemento en un DataFrame. Por ejemplo, sumar un valor constante a todas las entradas del DataFrame:**

```
import pandas as pd

datos = {'A': [1, 2, 3],
         'B': [4, 5, 6]}
df = pd.DataFrame(datos)

# Sumar 10 a todas las entradas
df_suma = df + 10
print(df_suma)
```

Operaciones Aritméticas en DataFrames de pandas

```python
import pandas as pd

datos1 = {'A': [1, 2, 3],
          'B': [4, 5, 6]}
df1 = pd.DataFrame(datos1)

datos2 = {'A': [10, 20, 30],
          'B': [40, 50, 60]}
df2 = pd.DataFrame(datos2)

# Sumar DataFrames
df_suma = df1 + df2
print(df_suma)
```

Operaciones entre DataFrames: También puedes realizar operaciones entre DataFrames. Las operaciones se realizan en función de las etiquetas y las columnas coincidentes:

Operaciones Aritméticas en DataFrames de pandas

```python
import pandas as pd

datos = {'A': [1, 2, 3],
         'B': [4, 5, 6]}
df = pd.DataFrame(datos)

serie = pd.Series([10, 20], index=['A', 'B'])

# Sumar DataFrame y Serie
resultado = df + serie
print(resultado)
```

Operaciones con Series: **Las operaciones aritméticas también se pueden realizar entre DataFrames y Series. Pandas alinea automáticamente los índices antes de realizar la operación:**

Estas operaciones aritméticas son esenciales para realizar cálculos e cientes en conjuntos de datos de Pandas, facilitando el análisis y procesamiento de datos en el ámbito de la inteligencia arti cial y la ciencia de datos.

Estadísticas descriptivas y resumidas:

Método	Descripción
count	Número de valores no NA
describe	Calcule un conjunto de estadísticas de resumen para Series o cada columna de DataFrame
min, max	Calcular valores mínimos y máximos
argmin, argmax	Calcule las ubicaciones de índice (enteros) en las que se obtuvo el valor mínimo o máximo, respectivamente
idxmin, idxmax	Calcule las etiquetas de índice en las que se obtuvo el valor mínimo o máximo, respectivamente

Estadísticas descriptivas y resumidas:

Método	Descripción
count	Número de valores no NA
describe	Calcule un conjunto de estadísticas de resumen para Series o cada columna de DataFrame
min, max	Calcular valores mínimos y máximos
argmin, argmax	Calcule las ubicaciones de índice (enteros) en las que se obtuvo el valor mínimo o máximo, respectivamente
quantile	Calcule el cuantil de muestra que va de 0 a 1
sum	Suma de valores
mean	Media de valores
median	Mediana aritmética (50% cuantil) de valores

Estadísticas descriptivas y resumidas:

Método	Descripción
mad	Desviación absoluta media del valor medio
prod	Producto de todos los valores
var	Ejemplo de varianza de valores
std	Ejemplo de desviación estándar de valores
skew	Sesgo muestral (tercer momento) de los valores
kurt	Ejemplo de curtosis (cuarto momento) de valores
cumsum	Suma acumulada de valores
cummin, cummax	Mínimo o máximo acumulativo de valores, respectivamente
cumprod	Producto acumulativo de valores
diff	Calcular la primera diferencia aritmética (útil para series de tiempo)
pct_change	Calcular cambios porcentuales

Cargar una base de datos a Pandas

Puedes utilizar la función read_csv() si tu base de datos está en formato CSV (valores separados por comas), o read_excel() si tu base de datos está en formato Excel.

Cargar desde un archivo CSV:

```
import pandas as pd

# Cargar la base de datos desde un archivo CSV
data = pd.read_csv('nombre_del_archivo.csv')

# Mostrar las primeras filas de la base de datos
print(data.head())
```

Cargar desde un archivo Excel:

```
import pandas as pd

# Cargar la base de datos desde un archivo Excel
data = pd.read_excel('nombre_del_archivo.xlsx')

# Mostrar las primeras filas de la base de datos
print(data.head())
```

Cargar una base de datos a Pandas

En ambos casos, nombre_del_archivo.csv y nombre_del_archivo.xlsx deben reemplazarse con la ruta y el nombre de tu archivo CSV o Excel. La función head() se utiliza para mostrar las primeras las de la base de datos cargada, lo que te permite veri car que se haya cargado correctamente.

Además de estas dos funciones, Pandas también ofrece métodos para leer datos desde otros formatos de archivo, como archivos JSON, archivos HTML, bases de datos SQL, entre otros. Puedes explorar la documentación de Pandas para obtener más información sobre cómo cargar datos desde diferentes fuentes.

Visualización de datos en pandas

La visualización de datos es una parte crucial del análisis exploratorio de datos, ya que nos permite comprender mejor los patrones, tendencias y relaciones dentro de nuestros conjuntos de datos. Pandas ofrece una variedad de opciones para crear visualizaciones efectivas directamente desde los DataFrames.

Gráfico de Barras para la Distribución de una Variable Categórica:

Los grá cos de barras son útiles para visualizar la distribución de variables categóricas mostrando la frecuencia o conteo de cada categoría.

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

# Crear un DataFrame con datos de ejemplo
data = {'Categoría': ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'A'],
        'Valor': [10, 20, 15, 25, 30, 20, 35, 40, 45]}
df = pd.DataFrame(data)

# Crear un gráfico de barras
df['Categoría'].value_counts().plot(kind='bar')
plt.title('Distribución de la Variable Categórica')
plt.xlabel('Categoría')
plt.ylabel('Frecuencia')
plt.show()
```

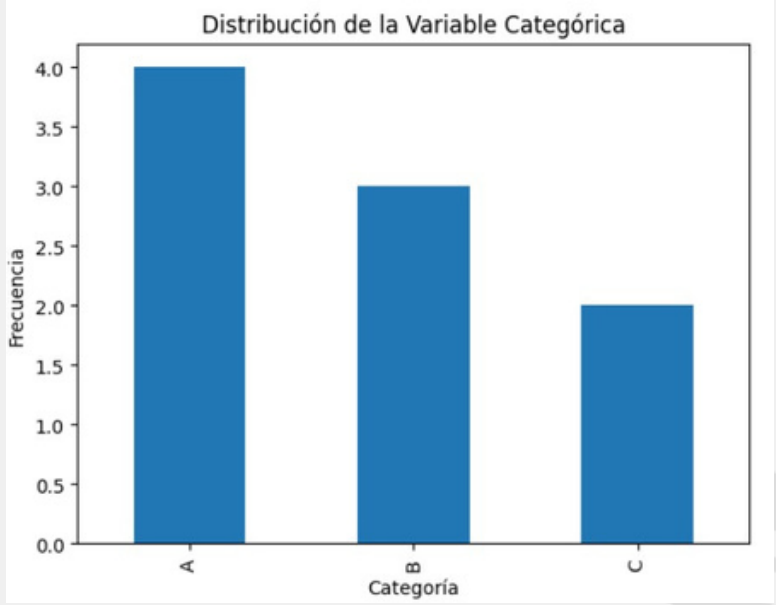

Gráfico de Dispersión para la Relación entre dos Variables Numéricas:

Los grá cos de dispersión son ideales para visualizar la relación entre dos variables numéricas, mostrando cómo se distribuyen los puntos en un plano cartesiano.

```
# Crear un gráfico de dispersión
df.plot(kind='scatter', x='Categoría', y='Valor', color='blue')
plt.title('Relación entre dos Variables Numéricas')
plt.xlabel('Categoría')
plt.ylabel('Valor')
plt.show()
```

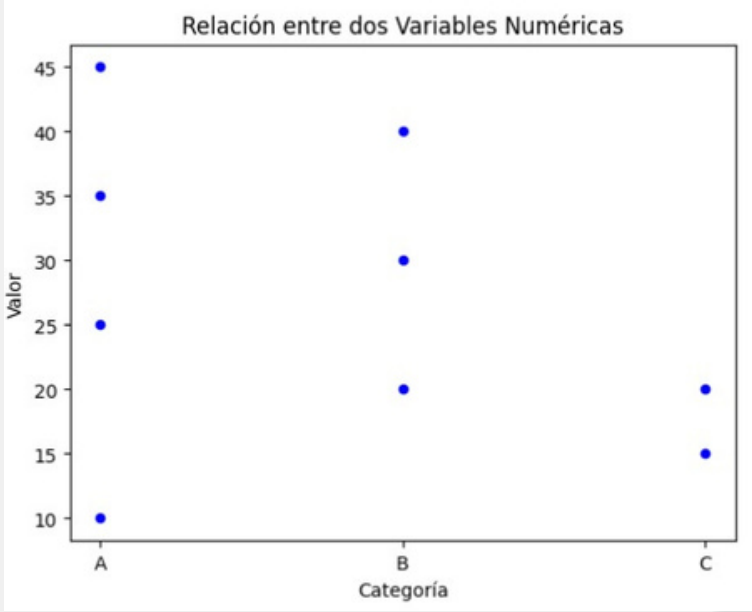

Gráfico de Líneas para Visualizar Tendencias a lo Largo del Tiempo:

Los grá cos de líneas son efectivos para visualizar tendencias y cambios en una variable a lo largo del tiempo o de alguna otra dimensión temporal.

```
# Crear un DataFrame con datos de ejemplo
time_series = pd.date_range('2023-01-01', periods=10)
data = {'Fecha': time_series,
        'Valor': [10, 25, 15, 25, 30, 35, 40, 50, 45, 55]}
df = pd.DataFrame(data)

# Crear un gráfico de líneas
df.plot(x='Fecha', y='Valor', color='green')
plt.title('Tendencia a lo Largo del Tiempo')
plt.xlabel('Fecha')
plt.ylabel('Valor')
plt.show()
```

Gráficos de Líneas

Estos ejemplos ilustran cómo crear visualizaciones básicas utilizando Pandas y Matplotlib. Sin embargo, Pandas también proporciona métodos más avanzados para la visualización de datos, como diagramas de caja, histogramas, grá cos de densidad y más, que pueden ser explorados para mejorar la comprensión de tus datos.

Ejercicios para practicar con Pandas

Cargar y Explorar Datos

a. Carga un conjunto de datos de tu elección en un DataFrame.
b. Muestra las primeras filas del DataFrame.
c. Muestra las últimas filas del DataFrame.
d. Obtén información del DataFrame, incluyendo tipos de datos y valores no nulos.

Selección y Filtrado de Datos

a. Selecciona una columna específica del DataFrame.
b. Filtra el DataFrame para mostrar solo las filas que cumplen una condición.
c. Selecciona múltiples columnas específicas del DataFrame utilizando la indexación.

Ejercicios para practicar con Pandas

Manipulación de Datos:

a. **Agrega una nueva columna al DataFrame.**
b. **Elimina una columna del DataFrame.**
c **Renombra una columna del DataFrame.**
d. **Ordena el DataFrame por una columna específica.**

Agrupación y Resumen de Datos:

a.**Agrupa el DataFrame por una columna específica y calcula estadísticas resumidas, como la media, la mediana y la desviación estándar.**
b. **en el DataFrame agrupado Calcula la suma, el promedio y el recuento de una columna específica para cada grupo**
c.**Calcula la correlación entre dos columnas del DataFrame.**

Ejercicios para practicar con Pandas

Visualización de Datos:

a. **Crea un gráfico de barras para mostrar la distribución de una variable categórica.**
b. **Crea un gráfico de dispersión para mostrar la relación entre dos variables numéricas.**
c. **Crea un gráfico de líneas para visualizar tendencias a lo largo del tiempo.**

Estos ejercicios te ayudarán a familiarizarte con las funcionalidades básicas y avanzadas de Pandas, así como a practicar con la manipulación y análisis de datos en Python. ¡Diviértete explorando y manipulando tus conjuntos de datos!