Git

Configurar **Git** con sus credenciales.

```
git config --global user.name "Nombre de usuario"
git config --global user.email "correo@electrónico.com"
```

Módulos y paquetes

NumPy

Biblioteca fundamental para el procesamiento numérico en Python.

Proporciona un conjunto de **funciones y estructuras de datos** eficientes para trabajar con **arreglos multidimensionales.**

Instalar la biblioteca

Para tener los módulos y paquetes de **NumPy** en tu ordenador, debes **instalar** la librería:

```
pip install numpy
```

Importar la biblioteca

Para comenzar a utilizar **NumPy**, debes **importar** la librería en tu programa de Python:

```
import numpy as np
```

Creacion de arreglos (array)

NumPy ofrece varias funciones para crear **arreglos**.

Algunas de las formas comunes incluyen:

1. Crear un arreglo con **valores predefinidos.**

```
arr = np.array([1, 2, 3])
```

2. Crear un arreglo de **ceros**.

```
arr_Ceros = np.zeros((2,2))
```

3. Crear un arreglo de **unos**.

```
arr_unos = np.ones((3,3))
```

4. Crear un arreglo con **valores aleatorios**.

```
aleatorio = np.random.rand(2,2)
```

Operaciones con arreglos

NumPy proporciona un conjunto completo de operaciones matemáticas y lógicas para trabajar con arreglos.

Algunas de las operaciones comunes incluyen:

1. **Suma** de arreglos.

```
arr3 = np.add(arr1, arr2)
```

2. **Resta** de arreglos.

```
arr3 = np.subtract(arr1, arr2)
```

3. **Producto** de arreglos.

```
arr3 = np.multiply(arr1, arr2)
```

4. **División** de arreglos.

```
arr3 = np.divide(arr1, arr2)
```

Funciones de agregación

NumPy ofrece funciones de **agregación** para realizar **cálculos estadísticos** en arreglos.

Algunas de las funciones comunes incluyen:

1. Calcular la **media.**

```
media = np.mean(arr)
```

2. Calcular la **mediana.**

```
mediana = np.median(arr)
```

3. Calcular el **máximo valor.**

```
maximo = np.max(arr)
```

4. Calcular el **mínimo valor.**

```
minimo = np.min(arr)
```

Indexación y rebanado de arreglos

Puedes acceder a elementos individuales y subconjuntos de arreglos utilizando **indexación y rebanado.**

Algunas técnicas comunes incluyen:

1. Acceder a **elementos individuales.**

```
arr[0]
```

2. Acceder a **subconjunts.**

```
arr[1:4]
```

3. Acceder a **filas o columnas específicas.**

```
arr[:, 1]
```

Copias y vistas

Con **NumPy** puedes crear **copias o vistas** independientes de tus arreglos

1. Copia de un arreglo

```
x = arr.copy()
```

2. Vista de un arreglo

```
y = arr.view()
```

Concatenación

NumPy cuenta con funciones de concatenación para **tuplas de arreglos.**

```
arr3 = np.concatenate((arr1, arr2))
```

Búsqueda y ordenamiento

NumPy permite **buscar** elementos dentro del arreglo que cumplan con condiciones.

```
x = np.where(arr == 4)
```

Otra de las funciones más utiles es el ordenamiento de los elementos del arreglo.

```
x = np.sort(arr)
```

Operaciones de álgebra lineal

NumPy también es ampliamente utilizado en álgebra lineal.

Algunas de las operaciones disponibles incluyen:

1. Producto de matrices.

```
np.dot(matrix1, matrix2)
```

2. Calcular la descomposicion en valores singulares.

```
np.linalg.svd(matrix)
```

3. Resolver sistemas de ecuaciones lineales.

```
np.linalg.solve(matrix, vector)
```

Pandas

Biblioteca de análisis de datos en Python.

Permite **manipular y analizar** fácilmente estructuras de datos, como tablas, mediante el uso de **DataFrames**.

Instalar la biblioteca

Para tener los módulos y paquetes de **Pandas** en tu ordenador, debes **instalar** la librería:

```
pip install pandas
```

Importar la biblioteca

Para comenzar a utilizar **Pandas**, debes **importar** la librería en tu programa de Python:

```
import pandas as pd
```

Series

En **Pandas** las **Series** son como una columna de una tabla. Es decir, una **matriz unidimensional** que contiene **datos de cualquier tipo.**

```
a = np.array([1,2,3])

srs = pd.Series(a)
```

Los valores pueden ser etiquetados manualmente o ser inicializados con su número de índice.

```
srs = pd.Series(a, index = ["x", "y", "z"])
```

DataFrames

Pandas ofrece estructuras de datos bidimiensionales, como una tabla con filas y columnas llamadas **DataFrames.**

```
mydataset = {
  'operador': ["John Doe", "Sr. X", "Lorem ipsum"],
  'no. reloj': [101, 202, 303]
}

df = pd.DataFrame(mydataset)
```

Pandas ofrece una amplia gama de funciones para **manipular los datos.**

Algunas operaciones comunes incluyen:

1. **Seleccionar** columnas específicas.

```
data['columna']
```

2. **Seleccionar** filas específicas.

```
data.loc[0]
```

3. **Filtrar** filas basadas en condiciones.

```
data[data['columna'] > 10]
```

4. **Agregar** una nueva columna.

```
data['nueva_columna'] = ...
```

Explorando los datos

Pandas proporciona diversas funciones para **explorar y manipular** los datos.

Algunas de las operaciones comunes incluyen:

1. Inspeccionar las primeras filas del **DataFrame.**

```
data.head()
```

2. Obtener informacion del **DataFrame.**

```
data.info()
```

3. Calcular **estadísticas descriptivas.**

```
data.describe()
```

Cargar datos

Pandas puede **leer datos** de diferentes fuentes, como archivos **CSV, JSON o bases de datos.**

A continuación, se muestra un **ejemplo** de carga de datos desde un archivo CSV:

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

Limpieza de datos

Visualización de datos

Pandas también se **integra** con bibliotecas de visualización, como **Matplotlib** y **Seaborn**, para crear **gráficos y visualizaciones.**

A continuación, se muestra un ejemplo básico de **visualizacion de datos** utilizando **Matplotlib**

```
import matplotlib as plt

data.plot(x='columna_x', y='columna_y', kind='scatter')
plt.show()
```

Matplotlib

Biblioteca de visualización de datos en Python.

Proporciona una amplia variedad de **herramientas para crear gráficos y visualizaciones** de datos de manera flexible y personalizable.

Importar la biblioteca

Para comenzar a utilizar **Matplotlib**, debes **importar** la librería en tu programa de Python:

```
import matplotlib as plt
```

Creación de gráficos básicos

Matplotlib ofrece una variedad de **tipos de gráficos** para **visualizar datos.**

Algunos de los tipos comunes incluyen:

1. Gráfico de **líneas.**

```
plt.plot(x, y)
```

2. Gráfico de **dispersión.**

```
plt.scatter(x, y)
```

3. Gráfico de **barras.**

```
plt.bar(x, y)
```

4. Gráfico de **pastel.**

```
plt.pie(x, y)
```

Personalización de gráficos

Matplotlib permite **personalizar** varios aspectos de los gráficos, como el **estilo, colores, etiquetas, títulos, ejes, etc.**

Algunas opciones de personalización incluyen:

1. Cambiar los **colores.**

```
plt.plot(x, y, color='blue')
```

2. Agregar **etiquetas.**

```
plt.xlabel('Eje X')
```

3. Agregar **título.**

```
plt.title('Producción por Día')
```

4. Cambiar el **estilo de línea.**

```
plt.plot(x, y, linestyle='dashed')
```

Múltiples gráficos y subtramas

Puedes combinar varios gráficos en una misma figura o crear subtramas para visualizar diferentes aspectos de los datos.

Algunas opciones incluyen:

1. Crear una figura con varios gráficos.

```
plt.subplot(rows, cols, index)
```

2. Añadir un gráfico a la figura.

```
plt.plot(x, y)
```

3. Ajustar los márgenes entre subtramas.

```
plt.subplots_adjust(hspace=0.5)
```

Guardar y exportar gráficos

Matplotlib permite guardar los gráficos generados en diferentes formatos, como PNG, PDF, SVG, etc.

Algunas opciones de exportación incluyen:

1. Guardar como imagen PNG.

```
plt.savefig('ejemplo.png')
```

2. Guardar como PDF.

```
plt.savefig('ejemplo.pdf')
```

Análisis de Datos

Proceso que implica la inspección, limpieza, transformación y modelado de datos con el objetivo de descubrir información útil, sacar conclusiones y respaldar la toma de decisiones.

1. Definición del problema

Identifica el problema o la pregunta que deseas responder para establecer objetivos y requisitos.

2. Recopilación de datos

Reúne los datos alojados en fuentes como bases de datos, archivos CSV, JSON o APIs.

3. Limpieza de datos

Identifica y corrige errores. Trata los valores nulos, estandariza formatos, elimina duplicados.

4. Exploración de datos

Analiza los datos para obtener una comprensión más profunda de su contenido. Calcula estadísticas descriptivas, crea visualizaciones, explora relaciones entre variables.

5. Preparación de datos

Prepara los datos para el modelado. Selecciona características relevantes, normaliza escalas y codifica variables categóricas.

6. Modelado de datos

Aplica técnicas de modelado y análisis estadístico. Algoritmos de aprendizaje automático, análisis de regresión o análisis de series temporales.

7. Evaluación de resultados

Evalúa la validez y relevancia de los resultados del análisis.

8. Comunicación de resultados

Presenta los resultados de manera clara y comprensible utilizando visualizaciones, informes u otros medios apropiados. Explica las conclusiones y cómo se pueden utilizar para respaldar la toma de

Conclusiones

NumPy es una herramienta esencial para el procesamiento numérico y para trabajar eficientemente con arreglos multidimensionales en Python.

Pandas es una herramienta poderosa para el análisis, manipulación, exploración y visualizacion de datos en Python.

Matplotlib es una herramienta indispensable para la visualización y graficación de datos en Python.