



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FACPYA

FACULTAD DE CONTADURÍA PÚBLICA Y ADMINISTRACIÓN

EVIDENCIA 2

ESTRUCTURA DE DATOS Y SU PROCESAMIENTO

Licenciado en Tecnologías de la Información

Semestre: 3° **Grupo:** 31

Maestro: Imelda Lizette Ramírez Rodríguez

Integrantes de equipo:

| | |
|------------------------------------|----------|
| Hernández González Angela Michelle | #2020879 |
| Molina Ramírez Fernando Nicolas | #2015797 |
| Rentería Flores Yaressi Abigail | #2052959 |
| Rodríguez Sánchez Giovanni Missael | #1955224 |
| Valerio Aguirre Giselle Nahomi | #2012254 |
| Rodríguez Delgado Emerson Aldair | #2017900 |
| Ramos Espinosa Luis Daniel | #2059274 |
| Zambrano Alcorta Roxana Dior | #2025417 |

Ciudad Universitaria, 06 de septiembre 2024.

INDICE

| | |
|-----------------------------------------------------|----|
| INTRODUCCION | 3 |
| Ventajas de Pandas..... | 4 |
| Ventajas NumPy | 4 |
| Ejemplos de Ventajas Pandas | 7 |
| Código ejemplos en Pandas | 8 |
| Series y Agregar datos a la serie..... | 10 |
| Convertir Series en Data Frame y agregar datos..... | 11 |
| Agregar datos al diccionario | 13 |
| Convertir Diccionario a DataFrame | 14 |
| Agregar datos al diccionario | 14 |
| Listas de listas DataFrame..... | 15 |
| Agregar listas de listas a dataframe..... | 16 |
| Código ejemplos NumPy | 17 |
| Crear diccionario para almacenar arrays..... | 17 |
| Arreglos de Numpy | 20 |
| Arreglos unidimensionales..... | 20 |
| Arreglos bidimensionales | 21 |
| Arreglos Ceros y Unos Ejemplos..... | 22 |
| VALIDACION DE REPOSITORIO EN GIT..... | 23 |
| CONCLUSION GRUPAL..... | 24 |
| CONCLUSION INDIVIDUAL | 25 |

INTRODUCCION

En el ámbito de la ciencia de datos y la programación, Python se ha consolidado como uno de los lenguajes más poderosos y versátiles, en gran parte gracias a sus bibliotecas especializadas como pandas y NumPy. Estas herramientas son fundamentales para el manejo y análisis de datos, permitiendo realizar operaciones complejas de manera eficiente y con una sintaxis clara y concisa.

Pandas es una biblioteca que proporciona estructuras de datos flexibles y expresivas, como los DataFrames, que facilitan la manipulación y análisis de grandes volúmenes de datos. Se usa mucho en tareas de limpieza, transformación y exploración de datos, gracias a su capacidad para manejar datos heterogéneos y su integración con otras bibliotecas de Python.

Por otro lado, NumPy se centra en la gestión de “Arrays” multidimensionales y operaciones matemáticas de alto rendimiento. La base sobre la que se construyen otras bibliotecas de Python y es valiosa para cálculos numéricos, álgebra lineal y procesamiento de datos a gran escala.

En este trabajo, exploraremos las características clave de Pandas y NumPy, su sintaxis básica y cómo estas herramientas se integran para resolver problemas complejos en el análisis de datos. A través de ejemplos prácticos, demostraremos cómo estas bibliotecas son esenciales para el trabajo en ciencia de datos y cómo se pueden aprovechar para maximizar la eficiencia y precisión en proyectos de análisis.

Ventajas de Pandas

- ❖ Reduce líneas de código a diferencia de Java o C
- ❖ Facilidad de uso, Pandas es muy fácil de usar gracias a su sintaxis clara y concisa. Con esta librería, puedes cargar, manipular y analizar datos tabulares en cuestión de minutos, lo que la hace ideal para el análisis de datos a gran escala.
- ❖ Gran funcionalidad, Pandas tiene una gran cantidad de funciones y métodos disponibles para manipular datos, desde la limpieza y la transformación hasta la agregación y la visualización. También es capaz de manejar datos de diferentes tipos, como texto, números, fechas y objetos complejos.
- ❖ Eficiencia, Pandas está diseñado para ser eficiente en el manejo de grandes cantidades de datos, lo que la hace ideal para el análisis de datos a gran escala. Además, Pandas utiliza la biblioteca NumPy para manejar grandes conjuntos de datos, lo que la hace aún más eficiente.
- ❖ Integración con otras librerías, Pandas se integra fácilmente con otras librerías de Python, como Matplotlib y SciPy, lo que la hace aún más útil para el análisis de datos.

Ventajas NumPy

- ❖ Eficiencia en operaciones matemáticas. Una de las ventajas más destacadas de NumPy es su eficiencia en operaciones matemáticas y numéricas.
- ❖ Las operaciones en arreglos NumPy se ejecutan a velocidades considerablemente más altas que las operaciones equivalentes en listas de Python. Esto se debe a que NumPy está implementado en C y utiliza arreglos contiguos en memoria, lo que minimiza la sobrecarga y mejora el rendimiento.
- ❖ Manipulación de arreglos multidimensionales. NumPy permite la creación y manipulación de arreglos multidimensionales, lo que resulta fundamental en aplicaciones científicas y de análisis de datos. Estos arreglos pueden tener cualquier número de dimensiones y contienen elementos del mismo tipo, lo que facilita la representación de datos complejos.
- ❖ Amplia colección de funciones matemáticas. La librería NumPy ofrece una amplia gama de funciones matemáticas que facilitan el procesamiento de datos. Desde operaciones básicas como suma y resta hasta funciones trigonométricas y estadísticas avanzadas, NumPy proporciona herramientas esenciales para el análisis y manipulación de datos numéricos.

Ejemplos de Ventajas de NumPy

NumPy es una herramienta poderosa que permite trabajar con datos numéricos de manera eficiente, realizar cálculos científicos complejos, e integrarse con otros lenguajes y bibliotecas. Su uso es fundamental en campos como la ciencia de datos, el aprendizaje automático y la computación científica en Python.

Código:

```
import numpy as np
import pandas as pd

#Crear array de datos
datos = np.array([112,11,20,37,21,66,468,88])
print('Los datos son:', datos)
#Operaciones aritmeticas al array



suma = datos + 90
resta = datos -12
multiplicacion = datos *10
division = datos/4

print('Suma (+90) es:', suma)
print('Resta (-12) es:', resta)
print('Multiplicacion (*10) es:', multiplicacion)
print('Division (/4) es:', division)

# Calcular estadísticas
media = np.mean(datos)
desviacion_estandar = np.std(datos)
maximo = np.max(datos)
```

```
minimo = np.min(datos)
print('Media:', media)
print('Desviación estándar:', desviacion_estandar)
print('Valor máximo:', maximo)
print('Valor mínimo:', minimo)
```

Demostración:

Running Python  

Output

```
Los datos son: [112  11  20  37  21  66 468  88]
Suma (+90) es: [202 101 110 127 111 156 558 178]
Resta (-12) es: [100  -1   8  25   9  54 456  76]
Multiplicacion (*10) es: [1120  110  200  370  210  660 4680  880]
Division (/4) es: [ 28.    2.75   5.    9.25   5.25  16.5  117.
 22.  ]

Media: 102.875
Desviación estándar: 142.00918764291274
Valor máximo: 468
Valor mínimo: 11
```

Ejemplos de Ventajas Pandas

Como se ha mencionado, Pandas es una poderosa biblioteca en Python que facilita la manipulación y análisis de datos. Un ejemplo en código es que demuestra algunas de las ventajas de Pandas, como la capacidad para filtrar, agrupar y calcular estadísticas sobre un conjunto de datos.

Manejo Eficiente de Datos: Pandas permite manejar grandes volúmenes de datos de manera eficiente. Puedes leer, escribir, manipular, y analizar datos con facilidad, incluso cuando se trata de datasets de millones de filas.

Operaciones Rápidas y Flexibles: Las operaciones como la limpieza de datos, filtrado, agrupamiento, y agregación son rápidas y fáciles de realizar con Pandas. Puedes aplicar funciones personalizadas a tus datos y combinar conjuntos de datos de diferentes formas.

Soporte para Diversos Tipos de Datos: Pandas soporta una amplia gama de formatos de datos, como CSV, Excel, JSON, SQL, y más, lo que permite la fácil integración con otras herramientas y sistemas.

Manipulación de Series Temporales: Pandas es particularmente útil para trabajar con datos de series temporales. Puedes realizar operaciones como reindexado, resampling, y cálculos de ventanas móviles de manera simple y eficiente.

Visualización de Datos: Aunque Pandas no es una herramienta de visualización de datos en sí, se integra bien con bibliotecas como Matplotlib y Seaborn, facilitando la creación de gráficos a partir de DataFrames.

Explicación:

Filtrar datos: Se filtran las filas donde las ventas son mayores a 200.

Agrupar datos: Se agrupan las ventas por producto y se suman.

Operación numérica: Se calcula un nuevo valor para el precio final restando el descuento de las ventas.

Estadísticas descriptivas: Se calcula el promedio de ventas en todo el DataFrame.

Este código muestra algunas de las ventajas de Pandas, como la facilidad para realizar operaciones numéricas, filtrar, agrupar, y generar estadísticas de forma eficiente y concisa.

Código ejemplos en Pandas

```
import pandas as pd

# Crear un DataFrame de ejemplo
data = {
    'Producto': ['A', 'B', 'A', 'B', 'C', 'A', 'C', 'B'],
    'Ventas': [100, 150, 200, 250, 300, 350, 400, 450],
    'Descuento': [10, 20, 15, 25, 10, 20, 30, 25]
}

df = pd.DataFrame(data)

#Filtrar datos: Ventas mayores a 200
ventas_mayores_200 = df[df['Ventas'] > 200]

#Sumar las ventas por producto
ventas_por_producto = df.groupby('Producto')['Ventas'].sum()

#Calcular el precio final después de aplicar descuentos
df['Precio_Final'] = df['Ventas'] - df['Descuento']

#Calcular el promedio de ventas
promedio_ventas = df['Ventas'].mean()

print("Ventas mayores a 200:\n", ventas_mayores_200)
print("\nVentas por producto:\n", ventas_por_producto)
print("\nDataFrame con el precio final calculado:\n", df)
print("\nPromedio de ventas:", promedio_ventas)
```


Demostración:

```
Ev2.ipynb X Release Notes: 1.93.0
C: > Users > rxand > OneDrive - Universidad Autonoma de Nuevo León > py > Ev2.ipynb > Ventajas de Pandas > import pandas as pd
+ Code + Markdown | ▶ Run All ↺ Restart ☰ Clear All Outputs | 📄 Variables ☰ Outline ...
```

```
... Ventas mayores a 200:
      Producto Ventas Descuento
3           B    250         25
4           C    300         10
5           A    350         20
6           C    400         30
7           B    450         25

Ventas por producto:
      Producto
A         650
B         850
C         700
Name: Ventas, dtype: int64

DataFrame con el precio final calculado:
      Producto Ventas Descuento Precio_Final
0           A    100         10          90
1           B    150         20         130
2           A    200         15         185
3           B    250         25         225
4           C    300         10         290
5           A    350         20         330
6           C    400         30         370
7           B    450         25         425

Promedio de ventas: 275.0
```

```
PROBLEMS 19 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
```

Series y Agregar datos a la serie

CÓDIGO:

```
import pandas as pd

# Crear una Serie inicial con algunos datos
serie = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
print("Serie original:")
print(serie)
```

Demostración:

Agregar datos a una Serie

```
import pandas as pd

nueva_serie = pd.Series([4, 5, 6], index=['d', 'e', 'f'])
serie = pd.concat([serie, nueva_serie])

print("\nSerie con mas datos:")
print(serie)
```

✓ 00s

Serie con mas datos:

```
a    1
b    2
c    3
d    4
e    5
f    6
dtype: int64
```

Convertir Series en Data Frame y agregar datos

Código:

```
import pandas as pd

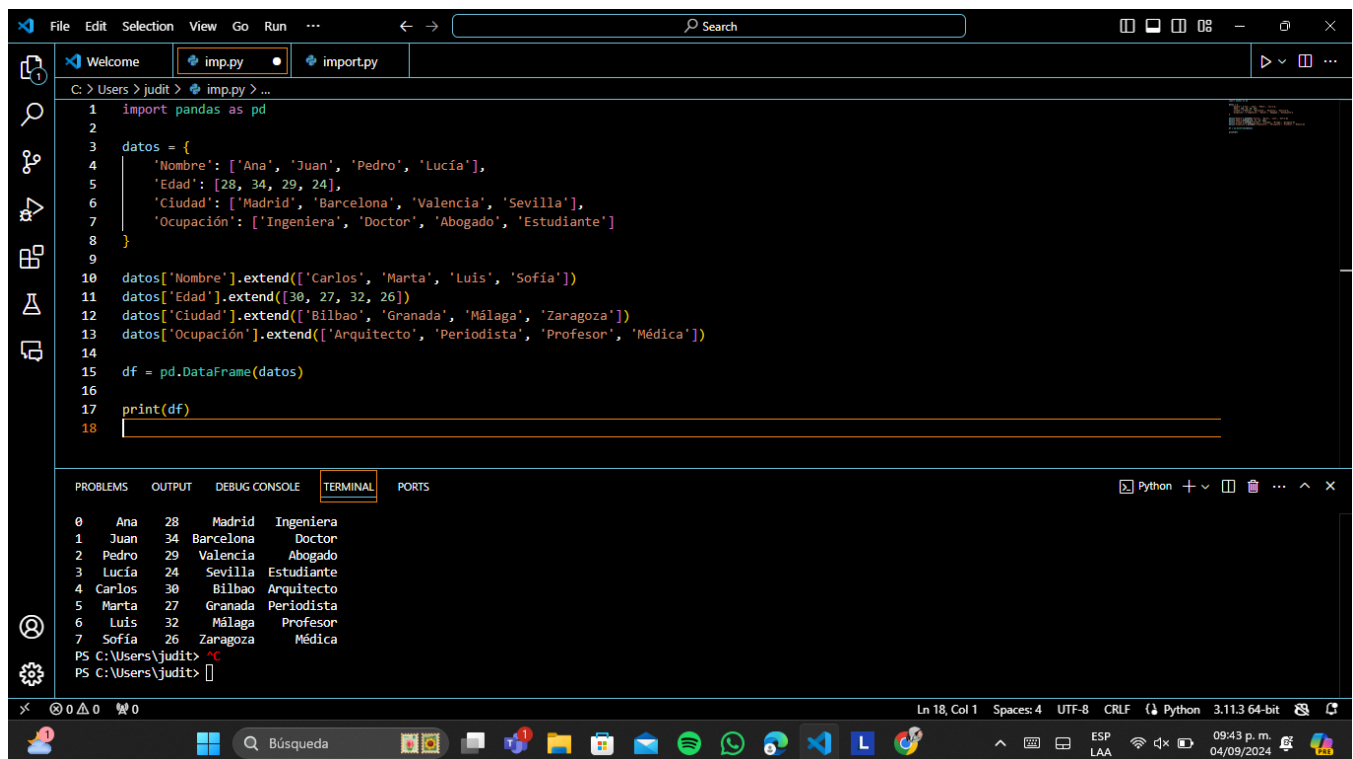
datos = {
    'Nombre': ['Ana', 'Juan', 'Pedro', 'Lucía'],
    'Edad': [28, 34, 29, 24],
    'Ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla'],
    'Ocupación': ['Ingeniera', 'Doctor', 'Abogado', 'Estudiante']
}

datos['Nombre'].extend(['Carlos', 'Marta', 'Luis', 'Sofía'])
datos['Edad'].extend([30, 27, 32, 26])
datos['Ciudad'].extend(['Bilbao', 'Granada', 'Málaga', 'Zaragoza'])
datos['Ocupación'].extend(['Arquitecto', 'Periodista', 'Profesor', 'Médica'])

df = pd.DataFrame(datos)

print(df)
```

Demostración:



The screenshot shows a Python IDE with the following code in the editor:

```
1 import pandas as pd
2
3 datos = {
4     'Nombre': ['Ana', 'Juan', 'Pedro', 'Lucía'],
5     'Edad': [28, 34, 29, 24],
6     'Ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla'],
7     'Ocupación': ['Ingeniera', 'Doctor', 'Abogado', 'Estudiante']
8 }
9
10 datos['Nombre'].extend(['Carlos', 'Marta', 'Luis', 'Sofía'])
11 datos['Edad'].extend([30, 27, 32, 26])
12 datos['Ciudad'].extend(['Bilbao', 'Granada', 'Málaga', 'Zaragoza'])
13 datos['Ocupación'].extend(['Arquitecto', 'Periodista', 'Profesor', 'Médica'])
14
15 df = pd.DataFrame(datos)
16
17 print(df)
18
```

The terminal output shows the resulting DataFrame:

| | Nombre | Edad | Ciudad | Ocupación |
|---|--------|------|-----------|------------|
| 0 | Ana | 28 | Madrid | Ingeniera |
| 1 | Juan | 34 | Barcelona | Doctor |
| 2 | Pedro | 29 | Valencia | Abogado |
| 3 | Lucía | 24 | Sevilla | Estudiante |
| 4 | Carlos | 30 | Bilbao | Arquitecto |
| 5 | Marta | 27 | Granada | Periodista |
| 6 | Luis | 32 | Málaga | Profesor |
| 7 | Sofía | 26 | Zaragoza | Médica |

Diccionarios convertir a Serie

Código:

```
import pandas as pd

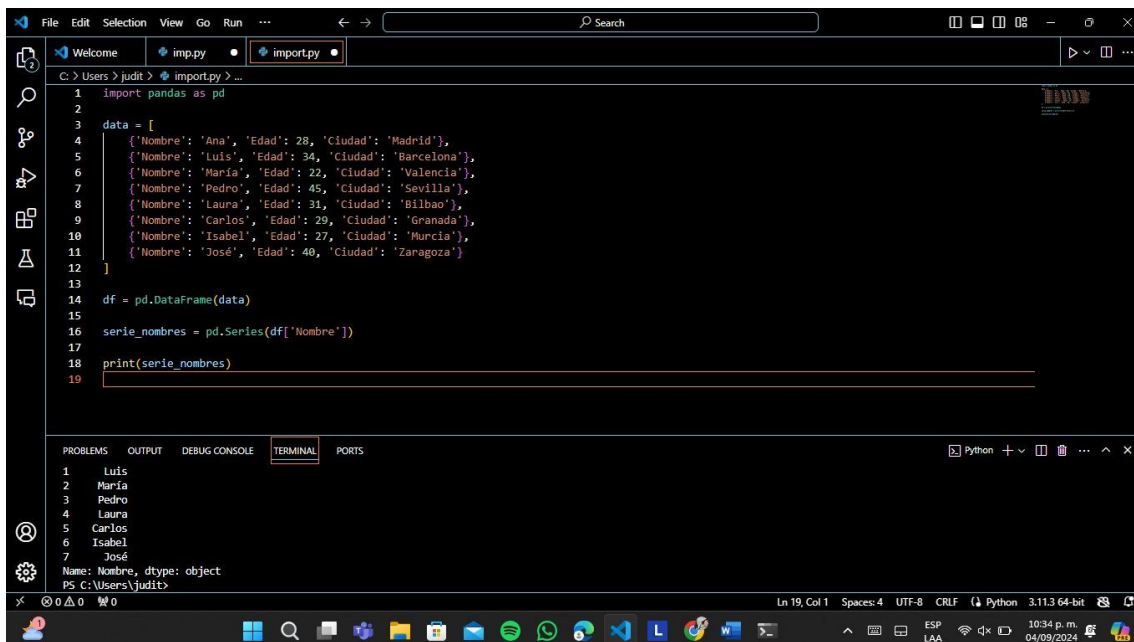
data = [
    {'Nombre': 'Ana', 'Edad': 28, 'Ciudad': 'Madrid'},
    {'Nombre': 'Luis', 'Edad': 34, 'Ciudad': 'Barcelona'},
    {'Nombre': 'María', 'Edad': 22, 'Ciudad': 'Valencia'},
    {'Nombre': 'Pedro', 'Edad': 45, 'Ciudad': 'Sevilla'},
    {'Nombre': 'Laura', 'Edad': 31, 'Ciudad': 'Bilbao'},
    {'Nombre': 'Carlos', 'Edad': 29, 'Ciudad': 'Granada'},
    {'Nombre': 'Isabel', 'Edad': 27, 'Ciudad': 'Murcia'},
    {'Nombre': 'José', 'Edad': 40, 'Ciudad': 'Zaragoza'}
]

df = pd.DataFrame(data)

serie_nombres = pd.Series(df['Nombre'])

print(serie_nombres)
```

Demostración:



```
File Edit Selection View Go Run ... Search
Welcome imp.py importpy
C:\Users\judit> imp.py
1 import pandas as pd
2
3 data = [
4     {'Nombre': 'Ana', 'Edad': 28, 'Ciudad': 'Madrid'},
5     {'Nombre': 'Luis', 'Edad': 34, 'Ciudad': 'Barcelona'},
6     {'Nombre': 'María', 'Edad': 22, 'Ciudad': 'Valencia'},
7     {'Nombre': 'Pedro', 'Edad': 45, 'Ciudad': 'Sevilla'},
8     {'Nombre': 'Laura', 'Edad': 31, 'Ciudad': 'Bilbao'},
9     {'Nombre': 'Carlos', 'Edad': 29, 'Ciudad': 'Granada'},
10    {'Nombre': 'Isabel', 'Edad': 27, 'Ciudad': 'Murcia'},
11    {'Nombre': 'José', 'Edad': 40, 'Ciudad': 'Zaragoza'}
12 ]
13
14 df = pd.DataFrame(data)
15
16 serie_nombres = pd.Series(df['Nombre'])
17
18 print(serie_nombres)
19

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
1 Luis
2 María
3 Pedro
4 Laura
5 Carlos
6 Isabel
7 José
Name: Nombre, dtype: object
PS C:\Users\judit>
```

Agregar datos al diccionario

Código:

```
import pandas as pd

data = [
    {'Nombre': 'Ana', 'Edad': 28, 'Ciudad': 'Madrid', 'Profesión': 'Ingeniera', 'Salario': 3000},
    {'Nombre': 'Luis', 'Edad': 34, 'Ciudad': 'Barcelona', 'Profesión': 'Abogado', 'Salario': 4000},
    {'Nombre': 'María', 'Edad': 22, 'Ciudad': 'Valencia', 'Profesión': 'Diseñadora', 'Salario': 2500},
    {'Nombre': 'Pedro', 'Edad': 45, 'Ciudad': 'Sevilla', 'Profesión': 'Profesor', 'Salario': 3500},
    {'Nombre': 'Laura', 'Edad': 31, 'Ciudad': 'Bilbao', 'Profesión': 'Doctora', 'Salario': 4200},
    {'Nombre': 'Carlos', 'Edad': 29, 'Ciudad': 'Granada', 'Profesión': 'Arquitecto', 'Salario': 3200},
    {'Nombre': 'Isabel', 'Edad': 27, 'Ciudad': 'Murcia', 'Profesión': 'Periodista', 'Salario': 2800},
    {'Nombre': 'José', 'Edad': 40, 'Ciudad': 'Zaragoza', 'Profesión': 'Contador', 'Salario': 3700},
    {'Nombre': 'Miguel', 'Edad': 38, 'Ciudad': 'Málaga', 'Profesión': 'Chef', 'Salario': 2900},
    {'Nombre': 'Lucía', 'Edad': 24, 'Ciudad': 'Córdoba', 'Profesión': 'Enfermera', 'Salario': 2700}
]

df = pd.DataFrame(data)

serie_nombres = pd.Series(df['Nombre'])

print(serie_nombres)
```

Ejemplo:

Agregar datos al diccionario

```
#DataFrame inicial del diccionario

df= pd.DataFrame({
    'Nombre':['Deyanira','Eugenio','Elvira','Santiago'],
    'Edad':[19,20,33,21],
    'Trabajo':['Veterinaria', 'Consulado', 'Ferretería','Farmacia']

})

# Agregar nueva columna

df['Licencia de manejo']= ['Estándar','Tipo A', 'Tipo C', 'Sin licencia']

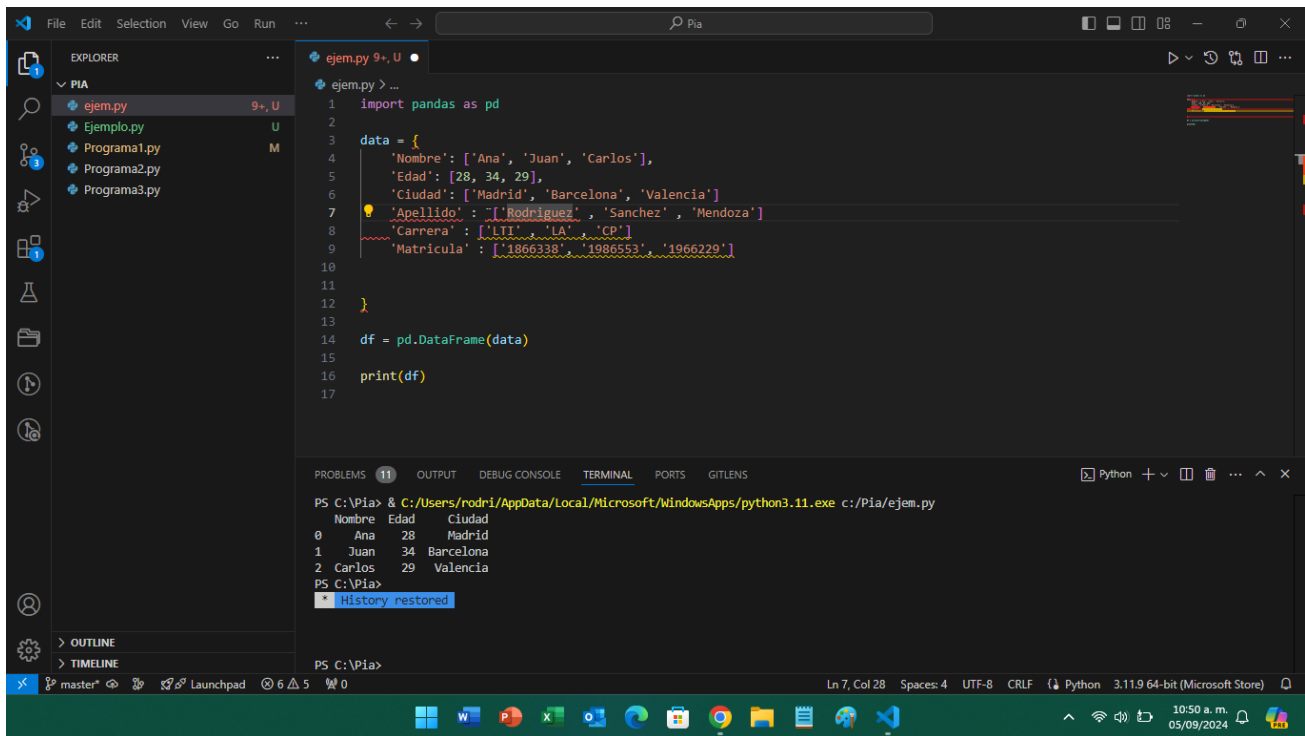
print(df)
```

[55] ✓ 00s

| | Nombre | Edad | Trabajo | Licencia de manejo |
|---|----------|------|-------------|--------------------|
| 0 | Deyanira | 19 | Veterinaria | Estándar |
| 1 | Eugenio | 20 | Consulado | Tipo A |
| 2 | Elvira | 33 | Ferretería | Tipo C |
| 3 | Santiago | 21 | Farmacia | Sin licencia |

[+ Code](#) [+ Markdown](#)

Convertir Diccionario a DataFrame



```
ejem.py 9+, U
1 import pandas as pd
2
3 data = {
4     'Nombre': ['Ana', 'Juan', 'Carlos'],
5     'Edad': [28, 34, 29],
6     'Ciudad': ['Madrid', 'Barcelona', 'Valencia'],
7     'Apellido': ['Rodriguez', 'Sanchez', 'Mendoza'],
8     'Carrera': ['LTI', 'LA', 'CP'],
9     'Matricula': ['1866338', '1986553', '1966229']
10 }
11
12 df = pd.DataFrame(data)
13
14 print(df)
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Pia> & C:/Users/rodri/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Pia/ejem.py
Nombre Edad Ciudad
0 Ana 28 Madrid
1 Juan 34 Barcelona
2 Carlos 29 Valencia
PS C:\Pia>
```

History restored

PS C:\Pia>

Ln 7, Col 28 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store)

Agregar datos al diccionario

```
ejem.py > ...
1 import pandas as pd
2
3 data = {
4     'Nombre': ['Ana', 'Juan', 'Carlos', 'Maria', 'Pedro', 'Lucia', 'Raul', 'Marta', 'Alberto', 'Sofia', 'David'],
5     'Apellido': ['Rodriguez', 'Sanchez', 'Mendoza', 'Lopez', 'Gomez', 'Martinez', 'Hernandez', 'Perez', 'Diaz',
6     'Edad': [28, 34, 29, 25, 33, 22, 30, 27, 35, 24, 31],
7     'Ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla', 'Bilbao', 'Granada', 'Malaga', 'Zaragoza', 'Vigo',
8     'Carrera': ['LTI', 'LA', 'CP', 'IC', 'ADE', 'DERE', 'MED', 'ING', 'ARQ', 'FIL', 'QUIM'],
9     'Matricula': ['1866338', '1986553', '1966229', '1755332', '1855221', '1923444', '1902233', '1988990', '19445
10 }
11
12 df = pd.DataFrame(data)
13
14 print(df)
```

Listas de listas DataFrame

```
1 import pandas as pd
2
3 # Crear una lista de listas
4 usuarios = [['Montgomery', 'Humes', 27, 'Male'],
5             ['Dagmar', 'Elstow', 41, 'Female'],
6             ['Reeba', 'Wattisham', 29, 'Female'],
7             ['Shalom', 'Alen', 29, 'Male'],
8             ['Broddy', 'Keningham', 21, 'Male'],
9             ['Aurelia', 'Breachin', 33, 'Female']]
10 # Crear el DataFrame pasando la lista de listas
11 df = pd.DataFrame(usuarios , columns=['First Name', 'Last Name', 'Age', 'Gender'])
12
13 print(df)
```

Running Python



Output

| | First Name | Last Name | Age | Gender |
|---|------------|-----------|-----|--------|
| 0 | Montgomery | Humes | 27 | Male |
| 1 | Dagmar | Elstow | 41 | Female |
| 2 | Reeba | Wattisham | 29 | Female |
| 3 | Shalom | Alen | 29 | Male |
| 4 | Broddy | Keningham | 21 | Male |
| 5 | Aurelia | Breachin | 33 | Female |

Agregar listas de listas a dataframe

```
13
14 # Nueva fila a agregar
15 new_row = ['Reeba', 'Wattisham', 29, 'Female']
16
17 # Método 1: Usando append()
18 df = df.append(pd.Series(new_row, index=df.columns), ignore_index=True)
19
20 # Método 2: Usando pd.concat()
21 new_row_df = pd.DataFrame([new_row], columns=df.columns)
22 df = pd.concat([df, new_row_df], ignore_index=True)
23
24 print("\nDataFrame después de agregar filas:")
25 print(df)
```

Running Python



Output

DataFrame después de agregar filas:

| | First Name | Last Name | Age | Gender |
|---|------------|-----------|-----|--------|
| 0 | Montgomery | Humes | 27 | Male |
| 1 | Dagmar | Elstow | 41 | Female |
| 2 | Reeba | Wattisham | 29 | Female |
| 3 | Shalom | Alen | 29 | Male |
| 4 | Broddy | Keningham | 21 | Male |
| 5 | Aurelia | Brechin | 33 | Female |
| 6 | Reeba | Wattisham | 29 | Female |
| 7 | Reeba | Wattisham | 29 | Female |

Código ejemplos NumPy

Crear diccionario para almacenar arrays

```
#CODIGO

import numpy as np

diccionario = {}

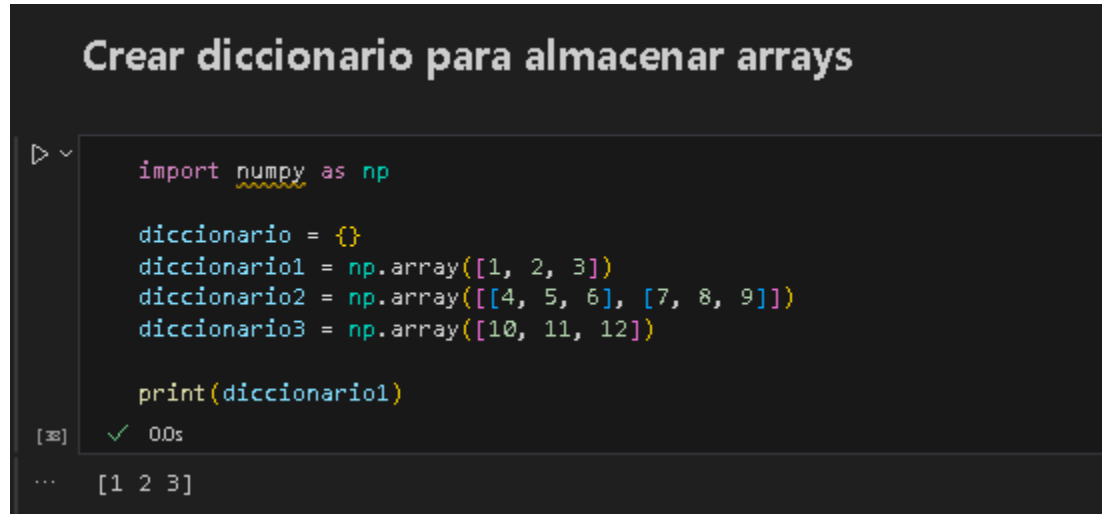
diccionario1 = np.array([1, 2, 3])

diccionario2 = np.array([[4, 5, 6], [7, 8, 9]])

diccionario3 = np.array([10, 11, 12])

print(diccionario1)
```

Crear diccionario para almacenar arrays



```
import numpy as np

diccionario = {}
diccionario1 = np.array([1, 2, 3])
diccionario2 = np.array([[4, 5, 6], [7, 8, 9]])
diccionario3 = np.array([10, 11, 12])

print(diccionario1)
```

[] ✓ 00s

... [1 2 3]

Crear DataFrame en NumPy

```
import numpy as np

import pandas as pd

array = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

df = pd.DataFrame(array, columns=['A', 'B', 'C'])

print(df)
```

Crear dataframe en NumPy

```
import numpy as np
import pandas as pd

array = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

df = pd.DataFrame(array, columns=['A', 'B', 'C'])

print(df)
```

✓ 00s

| | A | B | C |
|---|----|----|----|
| 0 | 10 | 20 | 30 |
| 1 | 40 | 50 | 60 |
| 2 | 70 | 80 | 90 |

Crear serie a partir de un array en NumPy

```
import pandas as pd
import numpy as np

#Se convierte el array en numpy
array_nu = np.array([
    ['Tylenol', '40 mg', 'Tomar cada 8 horas'],
    ['Buscapina', '10mg', 'Tomar cada 12 horas'],
    ['Dorixina', '50 mg', 'Tomar en caso de dolor'],
    ['Ibuprofeno', '150 mg', 'Tomar cada 6 horas']
])

#Convertir el array a dataframe
df=pd.DataFrame(array_nu, columns=['Medicina', 'Dosis', 'Prescripción médica'])
print(df)

#Convertir la columna Medicina en serie
serieMed= df['Medicina']

print(serieMed)
```

C:\> Users > rmand > OneDrive - Universidad Autonoma de Nuevo León > py > Ev2.ipynb > Ventajas de Pandas > Crear serie a partir de un array en numpy > import pandas as pd

+ Code + Markdown | ▶ Run All ⏮ Restart ⏮ Clear All Outputs | 📄 Variables 📄 Outline ...

```
▶ 40 mg', 'Tomar cada 8 horas'],
  ['Buscapina', '10mg', 'Tomar cada 12 horas'],
  ['Dorixina', '50 mg', 'Tomar en caso de dolor'],
  ['Ibuprofeno', '150 mg', 'Tomar cada 6 horas']
])

#Convertir el array a dataframe
df=pd.DataFrame(array_nu, columns=['Medicina', 'Dosis', 'Prescripción médica'])
print(df)

#Convertir la columna Medicina en serie
serieMed= df['Medicina']
print(serieMed)
```

[45] ✓ 0.0s

| | Medicina | Dosis | Prescripción médica |
|---|------------|--------|------------------------|
| 0 | Tylenol | 40 mg | Tomar cada 8 horas |
| 1 | Buscapina | 10mg | Tomar cada 12 horas |
| 2 | Dorixina | 50 mg | Tomar en caso de dolor |
| 3 | Ibuprofeno | 150 mg | Tomar cada 6 horas |

0 Tylenol
1 Buscapina
2 Dorixina
3 Ibuprofeno
Name: Medicina, dtype: object

PROBLEMS 15 OUTPUT DEBUGCONSOLE TERMINAL PORTS JUPYTER

Filter (eq, text, exclude, yes/no)

Arreglos de Numpy

Arreglos unidimensionales

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print("Arreglo unidimensional:")
print(arr)
```

Arreglos de Numpy

Unidimensionales

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print("Arreglo unidimensional:")
print(arr)
```

[99]

✓ 00s

```
... Arreglo unidimensional:
[1 2 3 4 5]
```

Arreglos bidimensionales

```
import numpy as np

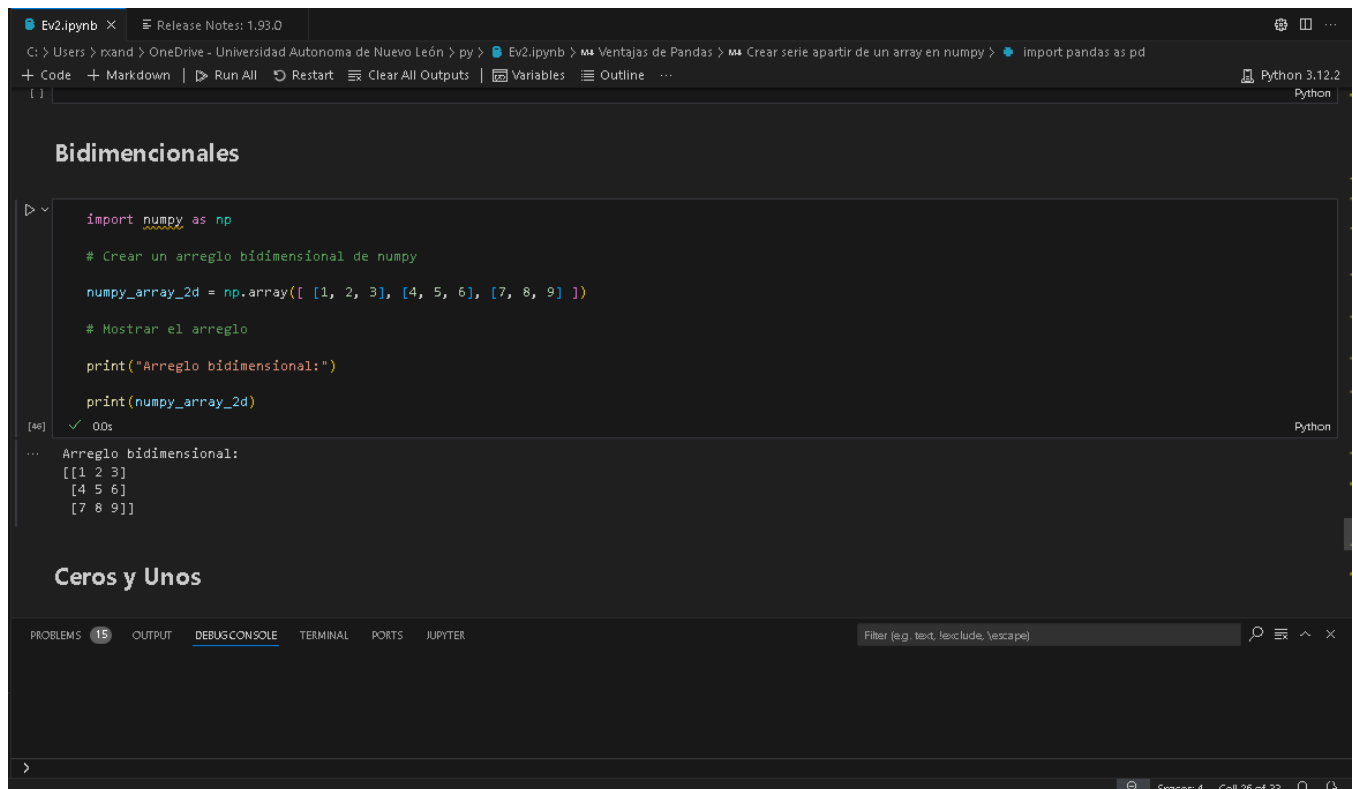
# Crear un arreglo bidimensional de numpy

numpy_array_2d = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])

# Mostrar el arreglo

print("Arreglo bidimensional:")

print(numpy_array_2d)
```



The screenshot shows a Jupyter Notebook window with the title "Ev2.ipynb". The notebook is open to a cell titled "Bidimensionales". The code in the cell is as follows:

```
import numpy as np

# Crear un arreglo bidimensional de numpy

numpy_array_2d = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])

# Mostrar el arreglo

print("Arreglo bidimensional:")

print(numpy_array_2d)
```

The cell has been executed, and the output is displayed below the code:

```
Arreglo bidimensional:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

The Jupyter Notebook interface includes a top bar with the file name "Ev2.ipynb" and a "Release Notes: 1.93.0" link. Below the top bar is a toolbar with icons for running, restarting, and clearing the cell. The bottom of the notebook shows a status bar with "Spaces: 4" and "Cell 26 of 33".

Arreglos Ceros y Unos Ejemplos

```
import pandas as pd

import numpy as np

df_zeros = pd.DataFrame(np.zeros((3, 3)), columns=['A', 'B', 'C'])

print("DataFrame de ceros:")

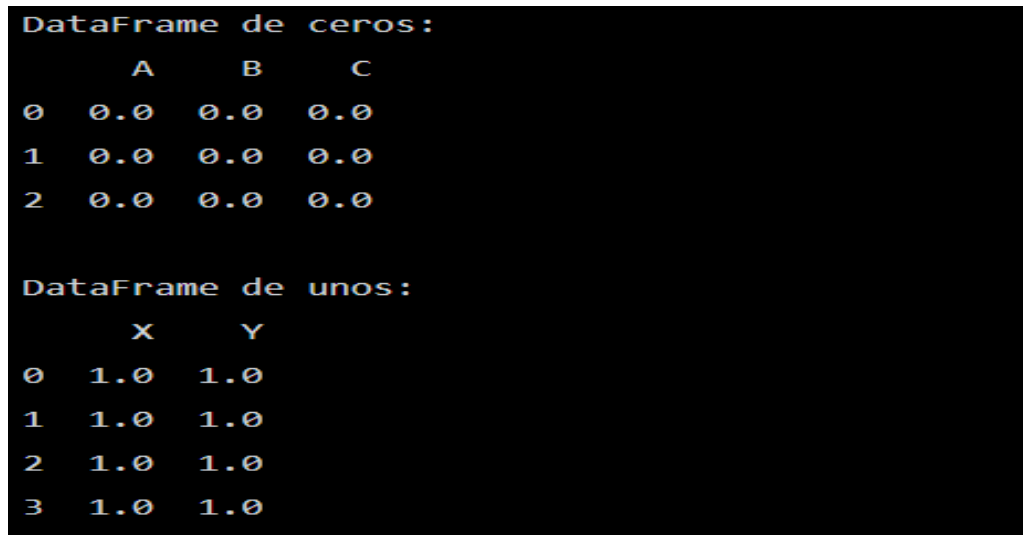
print(df_zeros)


df_ones = pd.DataFrame(np.ones((4, 2)), columns=['X', 'Y'])

print("\nDataFrame de unos:")

print(df_ones)
```

SCREENSHOT DE LA EJECUCIÓN



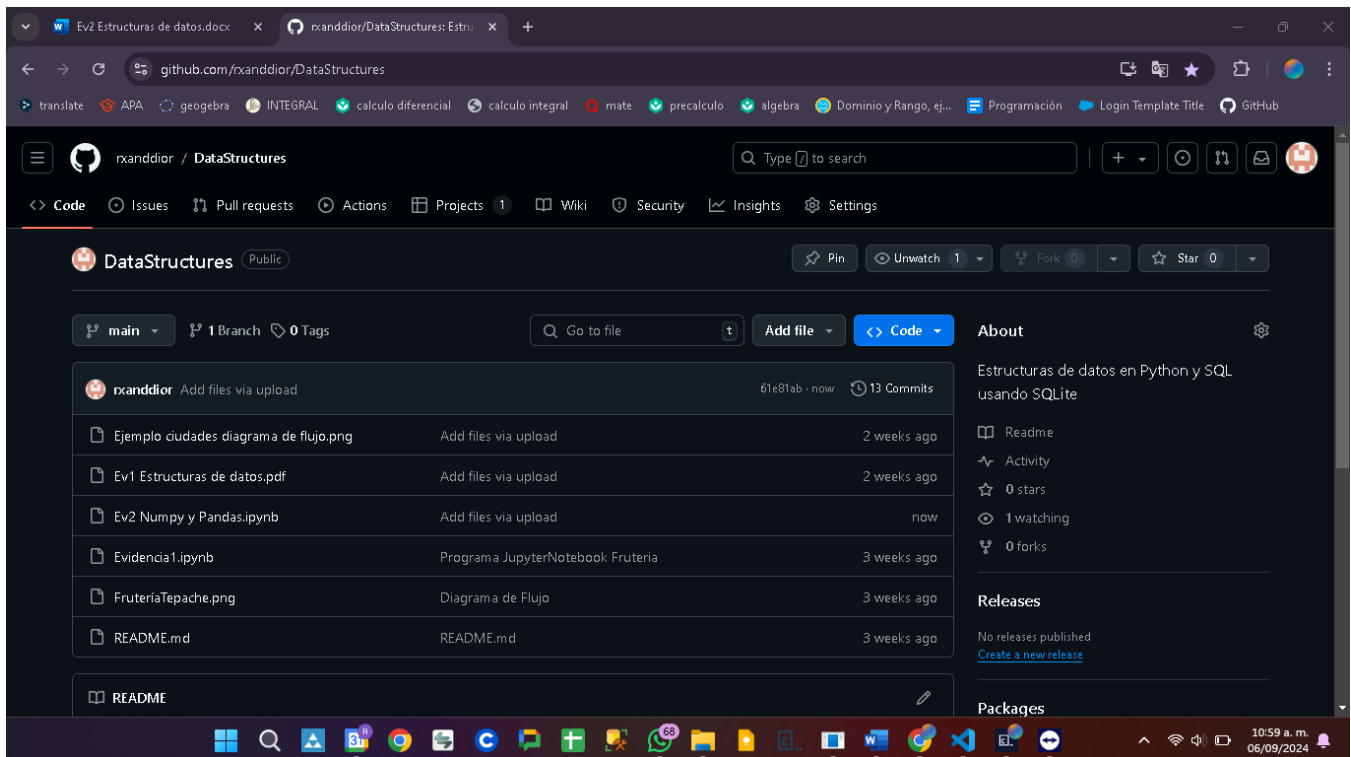
```
DataFrame de ceros:
   A    B    C
0  0.0  0.0  0.0
1  0.0  0.0  0.0
2  0.0  0.0  0.0

DataFrame de unos:
   X    Y
0  1.0  1.0
1  1.0  1.0
2  1.0  1.0
3  1.0  1.0
```

VALIDACION DE REPOSITORIO EN GIT

Enlace del Repositorio GitHub

<https://github.com/rxanddior/DataStructures>



CONCLUSION GRUPAL

Como equipo, aprendimos a llegar a apreciar la importancia de Pandas y NumPy en el análisis de datos. Al comenzar con estas bibliotecas, al principio sentimos que había mucho que aprender, pero rápidamente reconocimos cómo facilitan y aceleran las tareas que antes parecían complejas.

NumPy nos permitió entender la eficiencia en el manejo de datos numéricos y la facilidad con la que podemos realizar operaciones matemáticas avanzadas. Aunque algunos de nosotros no estábamos acostumbrados a trabajar con arrays, vimos cómo NumPy mejora la precisión y la velocidad de nuestras operaciones.

Por otro lado, Pandas nos sorprendió con su capacidad para manipular y analizar grandes conjuntos de datos de manera intuitiva. A medida que explorábamos Pandas, nos dimos cuenta de que muchas tareas que antes considerábamos tediosas, como la limpieza y reestructuración de datos, se volvían mucho más manejables. Esto no solo nos ahorró tiempo, sino que también nos ayudó a profundizar en el análisis de datos, obteniendo resultados más significativos y precisos.

CONCLUSION INDIVIDUAL

Rodríguez Sánchez Giovanni Missael: He aprendido a valorar la importancia de Pandas y NumPy en el análisis de datos. Aunque al principio parecía mucho por aprender, pronto descubrí cómo estas bibliotecas hacen que tareas complejas sean más rápidas y eficientes. NumPy mejoró mi manejo de datos numéricos, permitiendo operaciones avanzadas con precisión y velocidad. Por otro lado, Pandas me facilitó la manipulación de grandes conjuntos de datos, simplificando tareas tediosas y profundizando en el análisis.

Zambrano Alcorta Roxana Dior: En conclusión, he aprendido sobre el uso de pandas en lo que son las operaciones en python así como la visualización de los datos y sus estructuras, como operar en listas, tuplas, diccionarios, así como el uso de numpy para crear arreglos y usarlo en conjunto con pandas, así mismo para la creación de dataframes que nos harán más sencillo el uso y manejo de operaciones complejas en python para el análisis de datos.

Rodríguez Delgado Emerson Aldair: En conclusión, a esta evidencia que utilizamos pandas me pareció una buena herramienta que no conocía muy bien, pero al momento de tener la clase me llene más de conocimientos y a la hora de hacer los ejemplos o ejercicios me resulto sencillo y ver que se pueden realizar diferentes labores con esta misma.

Rentería Flores Yaressi Abigail: En conclusion Pandas es una herramienta fundamental en el mundo de la ciencia de datos y el análisis en Python. Su capacidad para manejar y manipular datos de manera eficiente transforma el proceso de trabajar con datos tabulares en algo mucho más accesible y menos laborioso. El DataFrame, la estructura central de pandas es comparable a una hoja de cálculo de Excel en la que puedes fácilmente organizar, analizar y visualizar datos.

La conversión de datos a un DataFrame es una parte esencial de este proceso. Imagina que tienes datos en varias formas: listas de Python, diccionarios, arrays de NumPy, o incluso archivos CSV. Pandas hace que convertir estos datos en un DataFrame sea un proceso sencillo y directo. Esto es crucial porque el DataFrame ofrece un formato estructurado y estandarizado que facilita el análisis, la manipulación y la limpieza de datos.

Los pandas no solo proporcionan una forma de trabajar con datos, sino que también simplifica y potencia el proceso. Su capacidad para convertir diversos tipos de datos en un formato organizado y manejable lo hace indispensable para quien trabaje con datos en Python. Con pandas, el análisis de datos se convierte en una tarea más eficiente y accesible, permitiéndote concentrarte en descubrir insights y tomar decisiones basadas en datos.

Valerio Aguirre Giselle Nahomi: Trabajar con pandas y numpy es fundamental para la manipulación y el análisis de datos en Python. Pandas y numpy son herramientas poderosas que, cuando se usan juntas, permiten un análisis de datos eficiente y flexible. En resumen, dominar pandas y numpy es crucial para cualquier profesional que desee realizar un análisis de datos exhaustivo y eficiente.

Ramos Espinosa Luis Daniel: En conclusión las pandas es una manera más flexible que ha revolucionado el análisis y la manipulación de datos en Python además es muy utilizada por desarrolladores, analistas y científicos de datos debido a su capacidad para gestionar grandes volúmenes de información de manera eficiente y con gran simplicidad por otra parte esta Data Frame y las Series que así mismo Pandas permite organizar y operar sobre datos tabulares de manera intuitiva, minimizando el tiempo y el esfuerzo necesario para llevar a cabo tareas complejas.

Molina Ramírez Fernando Nicolas: Los pandas nos pueden ayudar para llevar una mejor estructura en lo que queremos lograr en el proyecto, se esta manera lograr un poco más de organizacion en lo que estaremos llevando, como el ejemplo de nuestra fruteria que estamos logrando en equipo, me ayudó bastante a comprender más o menos como poder lograrlo de manera logica, y así poder concretarlo todo de manera más rapida, sencilla y eficiente.

Hernández González Angela Michelle: El uso de pandas y numpy es esencial en Python para el manejo y análisis de datos. Estas dos bibliotecas ofrecen capacidades avanzadas que, al combinarse, facilitan un análisis de datos ágil y adaptable. Al adquirir competencias en pandas y numpy es indispensable para cualquier profesional que busque llevar a cabo un análisis de datos detallado y efectivo.