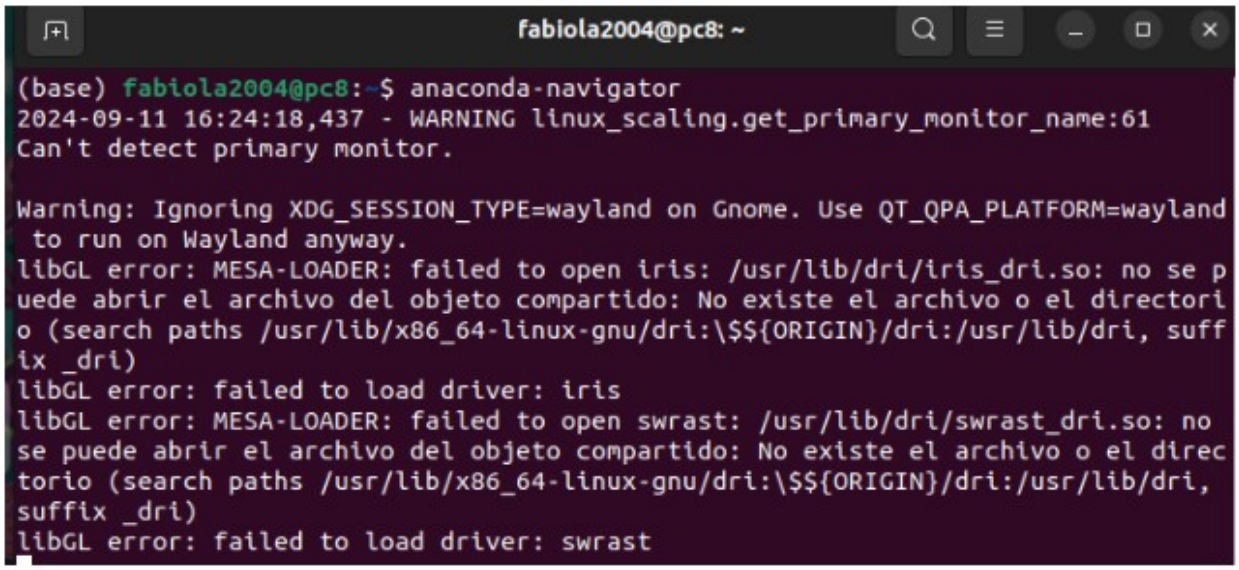
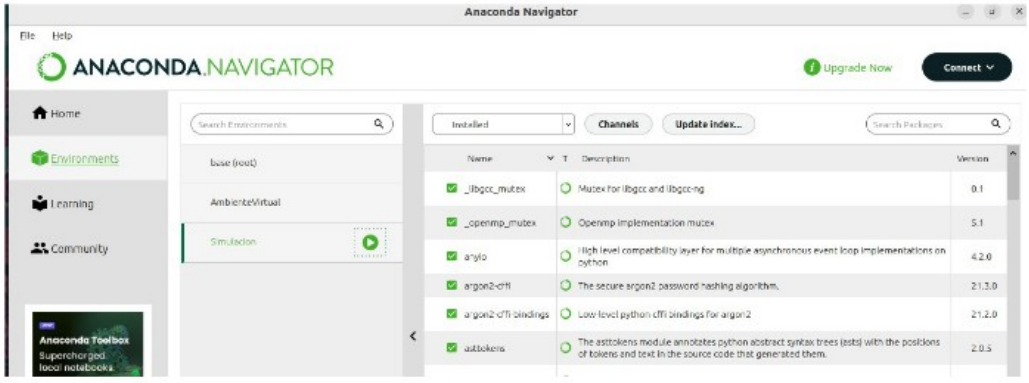
Nombre de la práctica	Pandas			No.	1
Asignatura:	Simulación	Carrera:	INGENIERÍA EN SISTEMAS COMPUTACIONALES	Duración de la práctica (Hrs)	

NOMBRE DEL ALUMNO: Fabiola Castañeda Mondragón
GRUPO: 3501

1.- Abrir Anaconda, desde la terminal con el comando a**naconda-navigator:**

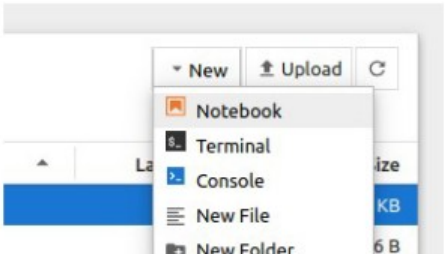


2.-Correr nuestro ambiente virtual:

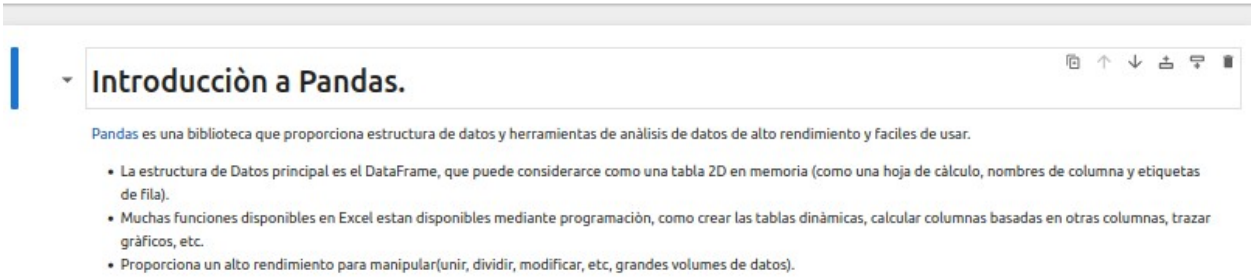


3.- Abrimos Jupyter:

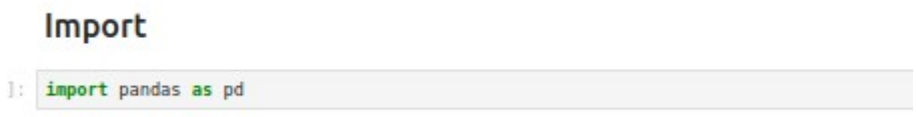
4.- Creamos un nuevo proyecto:



5.- Escribimos una breve introducción de Pandas:



6.-Muestra la importación de la biblioteca **Pandas** en Python con el alias `pd`. Esto permite utilizar las funciones de Pandas de manera más sencilla al escribir `pd` en lugar de `pandas` cada vez que se necesita. Pandas es clave para el manejo y análisis de datos.



7.- Explica las principales estructuras de datos en la biblioteca Pandas:

- **Series**: Un array unidimensional.
- **DataFrame**: Una tabla de dos dimensiones, como una hoja de cálculo.
- **Panel**: Un contenedor para múltiples DataFrames, similar a un diccionario de ellos (menos usado actualmente).

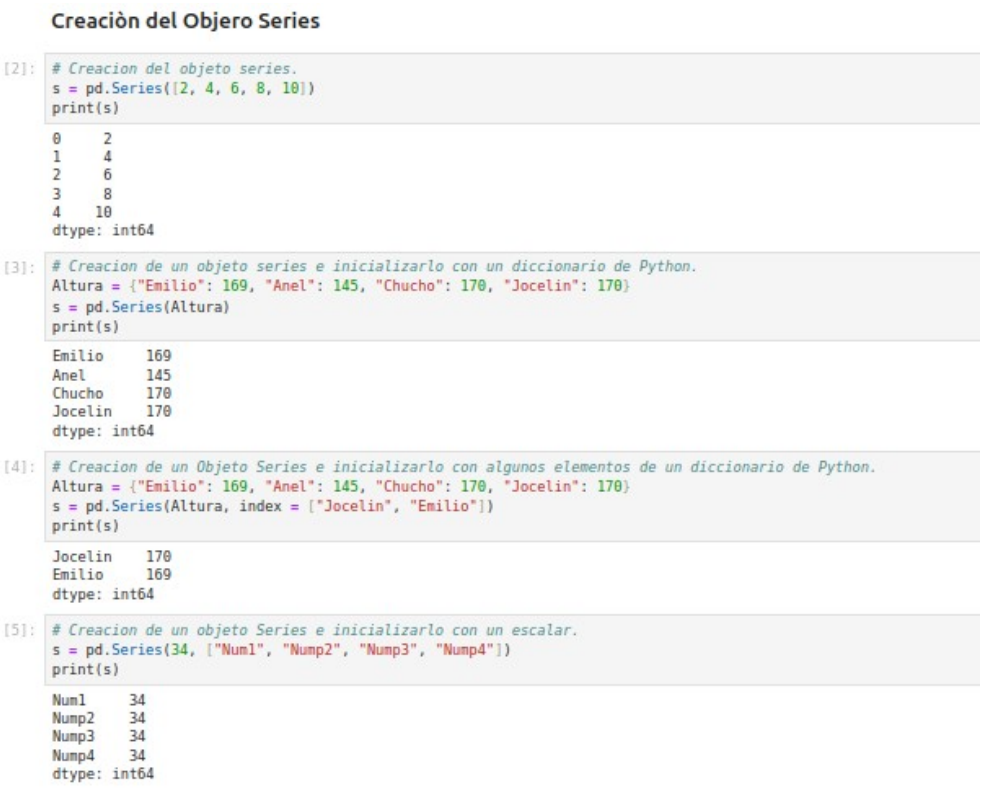
Estructuras de datos en Pandas

La Biblioteca Pandas, de manera gènerica contiene las siguientes estructuras de datos:

- **Series**: Array de una dimesion.
- **DataFrame**: Se corresponde con una tabla de dos dimensiones.
- **Panel**: Similar a un diccionario de DataFrames.

8.- Muestra ejemplos de cómo crear objetos **Series** en Pandas:

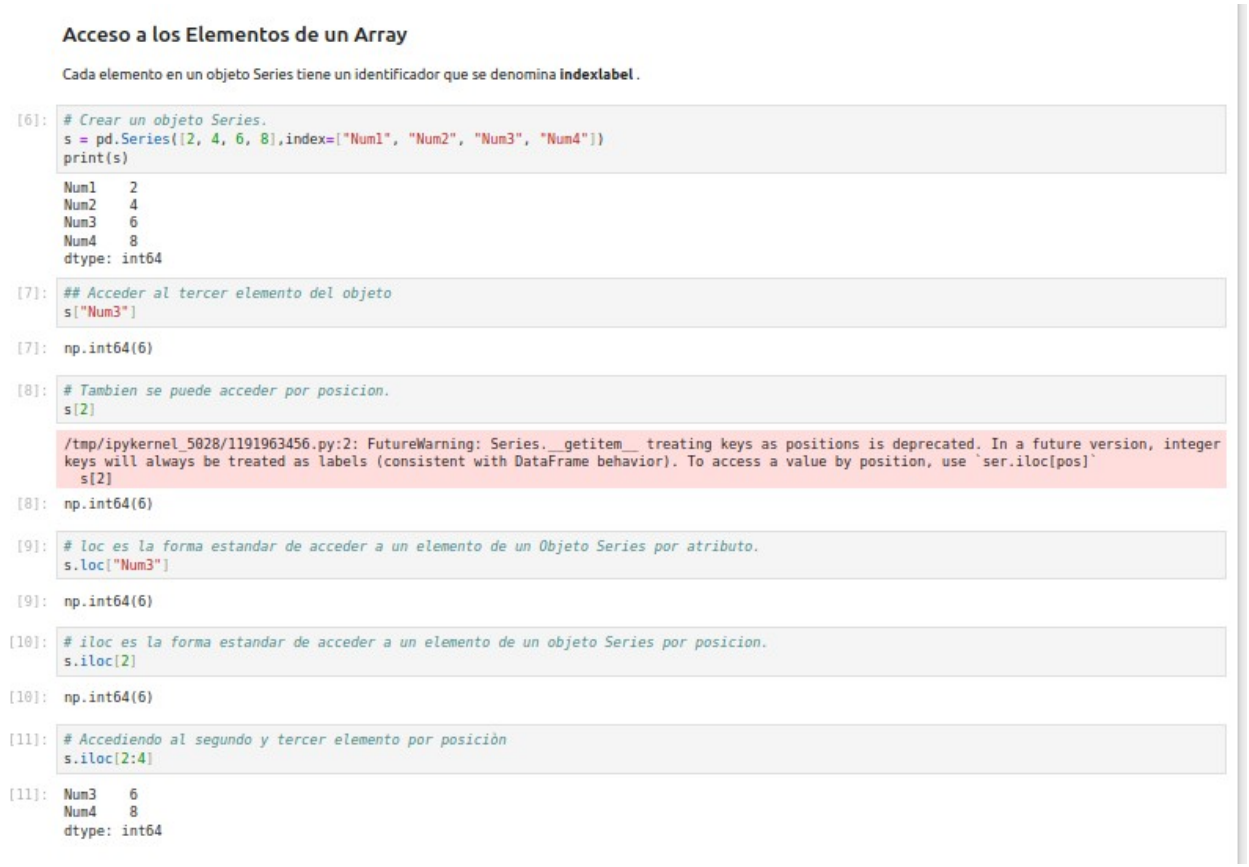
1. A partir de una lista de números.
2. A partir de un diccionario de Python.
3. Filtrando un diccionario usando índices específicos.
4. Inicializando una serie con un valor escalar repetido.



Creaciòn del Objero Series

```
[2]: # Creacion del objeto series.
     s = pd.Series([2, 4, 6, 8, 10])
     print(s)

     0     2
     1     4
     2     6
     3     8
     4    10
     dtype: int64

[3]: # Creacion de un objeto series e inicializarlo con un diccionario de Python.
     Altura = {"Emilio": 169, "Anel": 145, "Chucho": 170, "Jocelin": 170}
     s = pd.Series(Altura)
     print(s)

     Emilio     169
     Anel       145
     Chucho     170
     Jocelin    170
     dtype: int64

[4]: # Creacion de un Objeto Series e inicializarlo con algunos elementos de un diccionario de Python.
     Altura = {"Emilio": 169, "Anel": 145, "Chucho": 170, "Jocelin": 170}
     s = pd.Series(Altura, index = ["Jocelin", "Emilio"])
     print(s)

     Jocelin    170
     Emilio     169
     dtype: int64

[5]: # Creacion de un objeto Series e inicializarlo con un escalar.
     s = pd.Series(34, ["Num1", "Nump2", "Nump3", "Nump4"])
     print(s)

     Num1     34
     Nump2    34
     Nump3    34
     Nump4    34
     dtype: int64
```

9.- El código crea un objeto **Series** en Pandas con los valores `[2, 4, 6, 8]` y les asigna etiquetas personalizadas como índices: `"Num1"`, `"Num2"`, `"Num3"` y `"Num4"`. Luego imprime la serie resultante.



Acceso a los Elementos de un Array

Cada elemento en un objeto Series tiene un identificador que se denomina **indexlabel**.

```
[6]: # Crear un objeto Series.
s = pd.Series([2, 4, 6, 8],index=["Num1", "Num2", "Num3", "Num4"])
print(s)
```

```
Num1    2
Num2    4
Num3    6
Num4    8
dtype: int64
```

```
[7]: ## Acceder al tercer elemento del objeto
s["Num3"]
```

```
[7]: np.int64(6)
```

```
[8]: # Tambien se puede acceder por posicion.
s[2]
```

```
/tmp/ipykernel_5028/1191963456.py:2: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  s[2]
```

```
[8]: np.int64(6)
```

```
[9]: # loc es la forma estandar de acceder a un elemento de un Objeto Series por atributo.
s.loc["Num3"]
```

```
[9]: np.int64(6)
```

```
[10]: # iloc es la forma estandar de acceder a un elemento de un objeto Series por posicion.
s.iloc[2]
```

```
[10]: np.int64(6)
```

```
[11]: # Accediendo al segundo y tercer elemento por posición
s.iloc[2:4]
```

```
[11]: Num3    6
Num4    8
dtype: int64
```

10.- El código crea un objeto **Series** en Pandas con los valores `[2, 4, 6, 8, 10]` y los índices predeterminados (0, 1, 2, 3, 4). Luego imprime la serie resultante.

Operaciones Aritmeticas con Series

[12]:
```
# Crear un objeto Series
s = pd.Series([2, 4, 6, 8, 10])
print(s)
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

[13]:
```
# Los objetos Series son similares y compatibles con los Arrays de Numpy.
import numpy as np
# ufunc de Numpy para sumar los elementos.
np.sum(s)
```

[13]:
```
np.int64(30)
```

[14]:
```
s *2
```

[14]:
```
0     4
1     8
2    12
3    16
4    20
dtype: int64
```

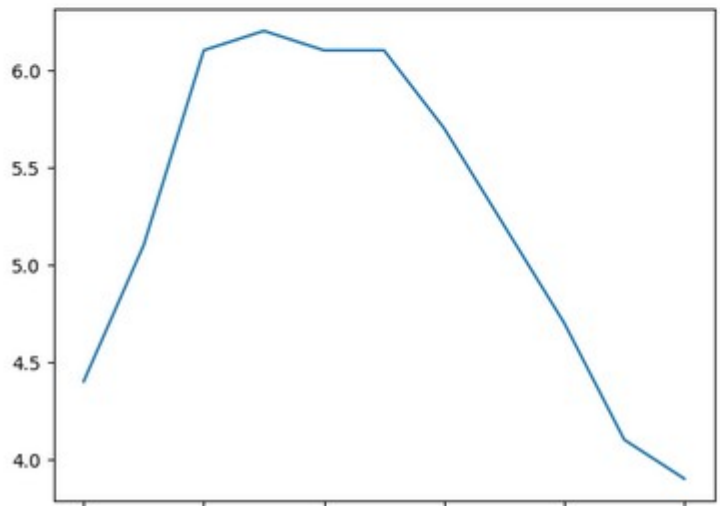
11.- En Pandas, el código `s = pd.Series(Temperaturas)` crea un objeto `Series` llamado `s` con los valores de la lista `Temperaturas`. Un `Series` es una estructura unidimensional que puede contener datos de diferentes tipos.

Representacion grafica de un objeto series

[15]:
```
# Crear un objeto Series denominado Temperaturas.
Temperaturas = [4.4, 5.1, 6.1, 6.2, 6.1, 6.1, 5.7, 5.2, 4.7, 4.1, 3.9]
s = pd.Series(Temperaturas)
s
```

[15]:
```
0     4.4
1     5.1
2     6.1
3     6.2
4     6.1
5     6.1
6     5.7
7     5.2
8     4.7
9     4.1
10    3.9
dtype: float64
```

[16]:
```
# Representacion grafica del objeto Series.
%matplotlib inline
import matplotlib.pyplot as plt

s.plot()
plt.show()
```



12.- Este código crea un `DataFrame` en Pandas a partir de un diccionario de `Series`:

1. **`Personas`**: Un diccionario donde:
    - **`"Peso"`**: `Series` con pesos y nombres como índices.
    - **`"Altura"`**: `Series` con alturas y nombres como índices.
    - **`"Mascotas"`**: `Series` con número de mascotas y nombres como índices.
2. **`df = pd.DataFrame(Personas)`**: Crea un `DataFrame` llamado `df` usando el diccionario `Personas`, que organiza los datos en una tabla con columnas para "Peso", "Altura" y "Mascotas"

Creacion de un objeto DataFrame .

[17]:
```
# Creacion de un DataFrame e inicializarlo con un diccionario de objetos Series.
Personas = {
    "Peso": pd.Series([72, 60, 74, 73], ["Emilio", "Anel", "Chucho", "Jocelin"]),
    "Altura": pd.Series({"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}),
    "Mascotas": pd.Series([2, 9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(Personas)
df
```

[17]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

Es posible forzar el DataFrame a que presente determinadas columnas y en orden determinado

[18]:
```
# Creacion de un DataFrame e inicializarlo con un diccionario de objetos Series.
Personas = {
    "Peso": pd.Series([72, 60, 74, 73], ["Emilio", "Anel", "Chucho", "Jocelin"]),
    "Altura": pd.Series({"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}),
    "Mascotas": pd.Series([2, 9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(
    Personas,
    columns = ["Altura", "Peso"],
    index = ["Chucho", "Emilio"])
df
```

[18]:

	Altura	Peso
Chucho	170	74
Emilio	169	72

[19]:
```
# Creacion de un DataFrame e inicializarlo con una lista de listas de Python.
# Nota: Deben especificarse las columnas e indices por separado.
Valores = [
    [169, 3, 72],
    [145, 2, 60],
    [170, 1, 74]
]

df = pd.DataFrame(
    Valores,
    columns = ["Altura", "Mascotas", "Peso"],
    index = ["Jocelin", "Emilio", "Anel"]
)
df
```

[19]:

	Altura	Mascotas	Peso
Jocelin	169	3	72
Emilio	145	2	60
Anel	170	1	74

[20]:
```
# Creación de un DataFrame e inicializarlo con un diccionario de Python.
Personas ={
    "Peso": {"Emilio":72, "Anel":60, "Chucho":74, "Jocelin":73},
    "Altura":{"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}}

df = pd.DataFrame(Personas)
df
```

[20]:

	Peso	Altura
Emilio	72	169
Anel	60	145
Chucho	74	170
Jocelin	73	170

13.- Este código crea un `DataFrame` con columnas "Peso", "Altura" y "Mascotas" usando un diccionario de `Series`. Cada `Series` usa nombres como índices y contiene datos relacionados con personas

Acceso a los elementos de un DataFrame

[21]:
```
# Creacion de un DataFrame e inicializarlo con un diccionario de Python
Personas = {
    "Peso": pd.Series([72, 60, 74, 73], ["Emilio", "Anel", "Chucho", "Jocelin"]),
    "Altura": pd.Series({"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}),
    "Mascotas": pd.Series([2, 9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(Personas)
df
```

[21]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

14.- a expresión `df["Peso"]` devuelve la columna "Peso" del `DataFrame`, que contiene los pesos de cada persona con sus respectivos índices.

Acesso a los elementos de las columnas de DataFrame

```
[22]: df["Peso"]
```

```
[22]: Anel      60
      Chucho    74
      Emilio    72
      Jocelin   73
      Name: Peso, dtype: int64
```

```
[24]: df [["Peso", "Altura"]]
```

[24]:

	Peso	Altura
Anel	60	145
Chucho	74	170
Emilio	72	169
Jocelin	73	170

```
[25]: # Pueden combinarce los elementos anteriores con expresiones boolenas.
      df["Peso"] > 73
```

```
[25]: Anel      False
      Chucho     True
      Emilio    False
      Jocelin   False
      Name: Peso, dtype: bool
```

```
[28]: # Pueden combunarce los metodos anteriores con expresiones booleanas y mostrar el DataFrmae
      df[df["Peso"]>72]
```

[28]:

	Peso	Altura	Mascotas
Chucho	74	170	NaN
Jocelin	73	170	9.0

15.- Accede nuevamente a los datos, pero esta vez de las filas.

Accediendo a los elementos de las filas del DataFrame

```
[29]: # Mostrar en DataFrame
      df
```

[29]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

```
[30]: df.loc["Emilio"]
```

```
[30]: Peso        72.0
      Altura     169.0
      Mascotas     NaN
      Name: Emilio, dtype: float64
```

```
[31]: df.iloc[1:3]
```

[31]:

	Peso	Altura	Mascotas
Chucho	74	170	NaN
Emilio	72	169	NaN

16.- El código `df.query("Altura >=170 and Peso > 73")` filtra el `DataFrame` para mostrar solo las filas donde la altura es mayor o igual a 170 y el peso es mayor a 73.

Consulta Avanzada de los elementos de un DataFrame ¶

[32]:
```
# Mostrar el DataFrame
df
```

[32]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

[34]:
```
df.query("Altura >=170 and Peso > 73")
```

[34]:

	Peso	Altura	Mascotas
Chucho	74	170	NaN

17.- Realiza una copia del DataFrame

Copiar un DataFrame

[35]:
```
# Crear un DataFrame e inicializarlo con un diccionario de Objetos Series.
# Creacion de un DataFrame e inicializarlo con un diccionario de Python
Personas = {
    "Peso": pd.Series([72, 60, 74, 73], ["Emilio", "Anel", "Chucho", "Jocelin"]),
    "Altura": pd.Series({"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}),
    "Mascotas": pd.Series([2, 9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(Personas)
df
```

[35]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

[36]:
```
# Copia del DataFrame df en df_copy
# Nota: Al modificar un elemento del df_copy no se modifica df.
df_copy = df.copy()
```

18.- Modifica el DataFRame

Modificacion de un DataFrame

[37]:
```
# Añadir una nueva columna al DataFrame
df["Anio_Nac"] = [2004, 2004, 2004, 2004]
df
```

[37]:

	Peso	Altura	Mascotas	Anio_Nac
Anel	60	145	2.0	2004
Chucho	74	170	NaN	2004
Emilio	72	169	NaN	2004
Jocelin	73	170	9.0	2004

[39]:
```
# Añadir una nueva columna calculada al DataFrame
df["Edad"] = 2024-df["Anio_Nac"]
df
```

[39]:

	Peso	Altura	Mascotas	Anio_Nac	Edad
Anel	60	145	2.0	2004	20
Chucho	74	170	NaN	2004	20
Emilio	72	169	NaN	2004	20
Jocelin	73	170	9.0	2004	20

[41]:
```
# Añadir una nueva columna creando un DataFrame nuevo.
df_mod = df.assign(Hijos = [2, 1, 2, 1])
df_mod
```

[41]:

	Peso	Altura	Mascotas	Anio_Nac	Edad	Hijos
Anel	60	145	2.0	2004	20	2
Chucho	74	170	NaN	2004	20	1
Emilio	72	169	NaN	2004	20	2
Jocelin	73	170	9.0	2004	20	1

[42]:
```
df
```

[42]:

	Peso	Altura	Mascotas	Anio_Nac	Edad
Anel	60	145	2.0	2004	20
Chucho	74	170	NaN	2004	20
Emilio	72	169	NaN	2004	20
Jocelin	73	170	9.0	2004	20

[43]:
```
# Eliminar una columna existente del DataFrame
del df["Peso"]
```

[44]:
```
df
```

[44]:

	Altura	Mascotas	Anio_Nac	Edad
Anel	145	2.0	2004	20
Chucho	170	NaN	2004	20
Emilio	169	NaN	2004	20
Jocelin	170	9.0	2004	20

[46]:
```
# Eliminar una columna existente, devolviendo una copia del DataFrame resultante
df_mod = df_mod.drop(["Hijos"], axis=1)
df_mod
```

[46]:

	Peso	Altura	Mascotas	Anio_Nac	Edad
Anel	60	145	2.0	2004	20
Chucho	74	170	NaN	2004	20
Emilio	72	169	NaN	2004	20
Jocelin	73	170	9.0	2004	20

19.- Evalua las expresiones sobre el DataFrame

Evaluacion de expresiones sobre un DataFrame

[49]:
```
# Crear un DataFrame e inicializarlo con un diccionario de Objetos series.
Personas = {
    "Peso": pd.Series([72, 60, 74, 73], ["Emilio", "Anel", "Chucho", "Jocelin"]),
    "Altura": pd.Series({"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}),
    "Mascotas": pd.Series([2, 9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(Personas)
df
```

[49]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

[51]:
```
# Elabora una funcion sobre una columna del DataFrame.
df.eval("Altura/2")
```

[51]:
```
Anel       72.5
Chucho     85.0
Emilio     84.5
Jocelin    85.0
Name: Altura, dtype: float64
```

[52]:
```
# Evaluar una funcion utilizando una variable local.
max_altura = 170
df.eval("Altura > @max_altura")
```

[52]:
```
Anel       False
Chucho     False
Emilio     False
Jocelin    False
Name: Altura, dtype: bool
```

[53]:
```
# Aplicar una funcion a una columna del DataFrame.
def func(x):
    return x + 2

df["Peso"].apply(func)
```

20.- Guardamos el DataFrame asi como tambien se realiza la carga

Guardar y cargar el DataFrame

[54]:
```
# Crear un DataFrame e inicializarlo con un diccionario de Objetos series.
Personas = {
    "Peso": pd.Series([72, 60, 74, 73], ["Emilio", "Anel", "Chucho", "Jocelin"]),
    "Altura": pd.Series({"Emilio": 169, "Anel":145, "Chucho": 170, "Jocelin":170}),
    "Mascotas": pd.Series([2, 9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(Personas)
df
```

[54]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

[56]:
```
# Guardar el DataFrame como CSV, HTML y JSON.
df.to_csv("df_Personas.csv")
df.to_html("df_Personas.html")
df.to_json("df_Personas.json")
```

[57]:
```
# Cargar el DataFrame en Jupyter.
df2 = pd.read_csv("df_Personas.csv")
```

[58]:
```
df2
```

[58]:

	Unnamed: 0	Peso	Altura	Mascotas
0	Anel	60	145	2.0
1	Chucho	74	170	NaN
2	Emilio	72	169	NaN
3	Jocelin	73	170	9.0

[59]:
```
#  Cargar el DataFrame con la primera columna correctamente asignada.
df2 = pd.read_csv("df_Personas.csv", index_col=0)
df2
```

21.- Se guarda en el Formato que nosotros queramos

[56]:
```
# Guardar el DataFrame como CSV, HTML y JSON.
df.to_csv("df_Personas.csv")
df.to_html("df_Personas.html")
df.to_json("df_Personas.json")
```

[57]:
```
# Cargar el DataFrame en Jupyter.
df2 = pd.read_csv("df_Personas.csv")
```

[58]:
```
df2
```

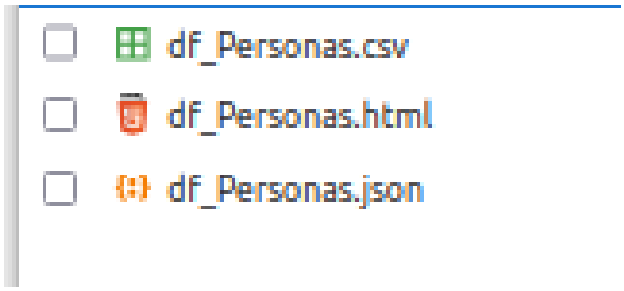
[58]:

	Unnamed: 0	Peso	Altura	Mascotas
0	Anel	60	145	2.0
1	Chucho	74	170	NaN
2	Emilio	72	169	NaN
3	Jocelin	73	170	9.0

[59]:
```
#  Cargar el DataFrame con la primera columna correctamente asignada.
df2 = pd.read_csv("df_Personas.csv", index_col=0)
df2
```

[59]:

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0



Conclusión.

Pandas es una herramienta clave en el análisis de datos en Python. Permite a los usuarios:

- Manipular Datos: Facilita la limpieza, transformación y preparación de datos para análisis.
- Estructuras Flexibles: Proporciona Series para datos unidimensionales y DataFrame para datos tabulares.
- Operaciones Avanzadas: Soporta operaciones complejas como agrupamiento, pivotamiento y consultas con sintaxis intuitiva.
- Integración: Se integra bien con otras bibliotecas de Python como NumPy, Matplotlib y scikit-learn, enriqueciendo el ecosistema de análisis de datos.

En resumen, Pandas simplifica el procesamiento y análisis de grandes conjuntos de datos, haciéndolo accesible y eficiente.