Pandas

What is Pandas?

- Pandas is a Python library used for working with data sets
- Pandas is a **powerful library** for **data analysis and manipulation** in Python. It provides **DataFrames** and **Series**, which make handling structured data easy.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

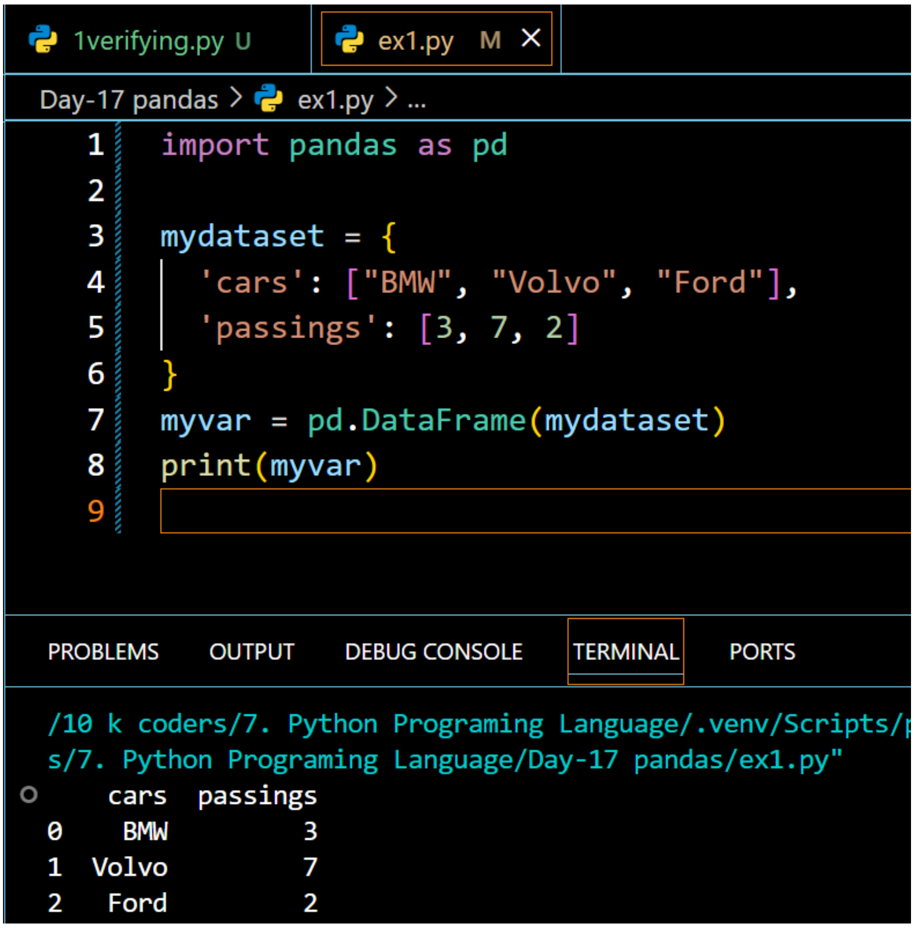
Installation of pandas:

To install pandas, follow these steps based on your setup:
1. Using pip (Recommended)
If you have Python installed, you can install pandas using pip:
pip install pandas
If you need to upgrade pandas, use:
pip install --upgrade pandas

Verifying Installation

After installation, check if pandas is installed by running:

import pandas as pd
print(pd.__version__)



```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

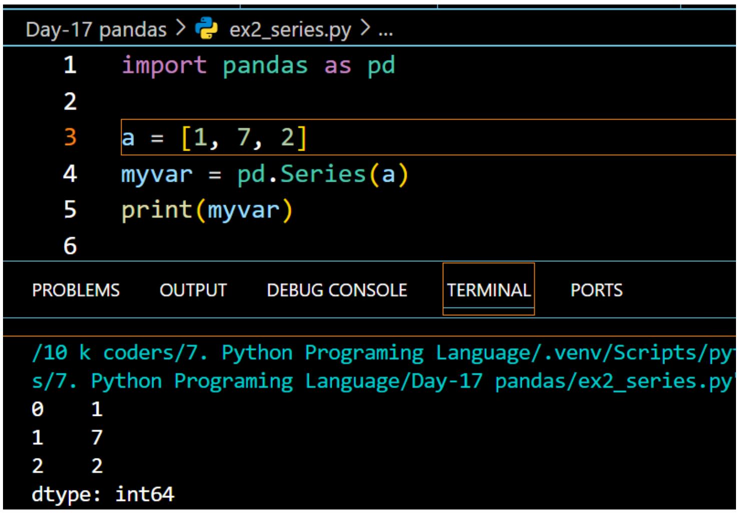
```
/10 k coders/7. Python Programing Language/.venv/Scripts/p
s/7. Python Programing Language/Day-17 pandas/ex1.py"
     cars  passings
0     BMW         3
1   Volvo         7
2    Ford         2
```

Pandas Series

What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.



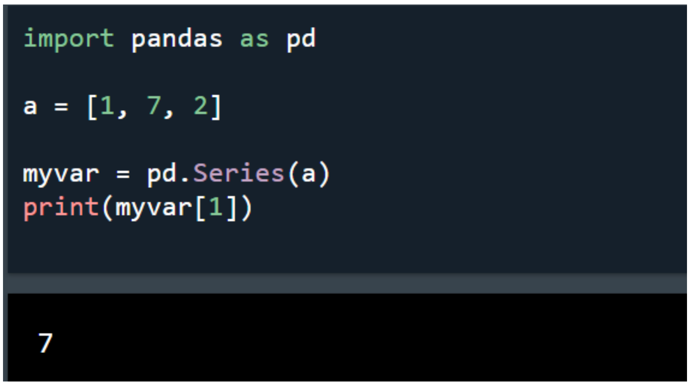
```
import pandas as pd

a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)

```

```
/10 k coders/7. Python Programing Language/.venv/Scripts/py
s/7. Python Programing Language/Day-17 pandas/ex2_series.py
0    1
1    7
2    2
dtype: int64
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

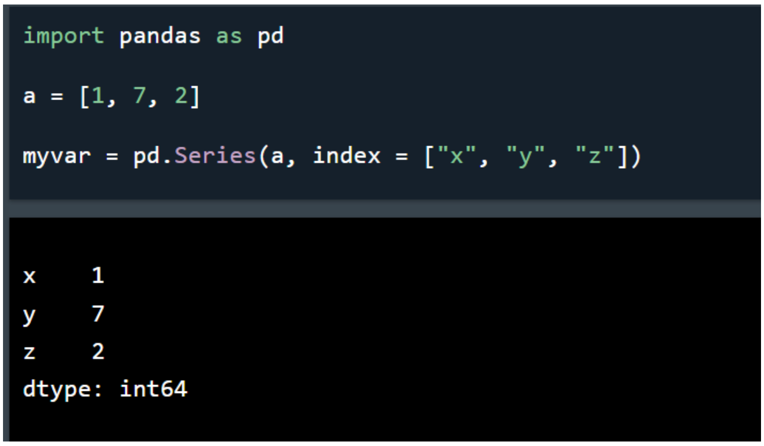


```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)
print(myvar[1])
```

```
7
```

Create Labels

With the index argument, you can name your own labels.

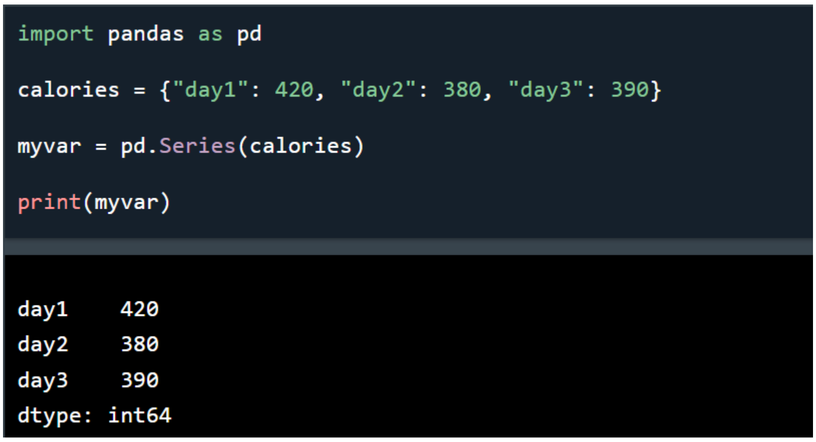


```
import pandas as pd

a = [1, 7, 2]

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

```
x    1
y    7
z    2
dtype: int64
```

Key/Value Objects as Series

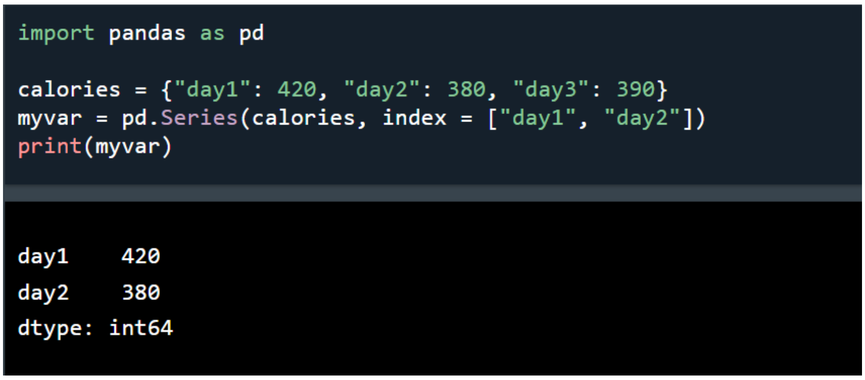
You can also use a key/value object, like a dictionary, when creating a Series.



```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

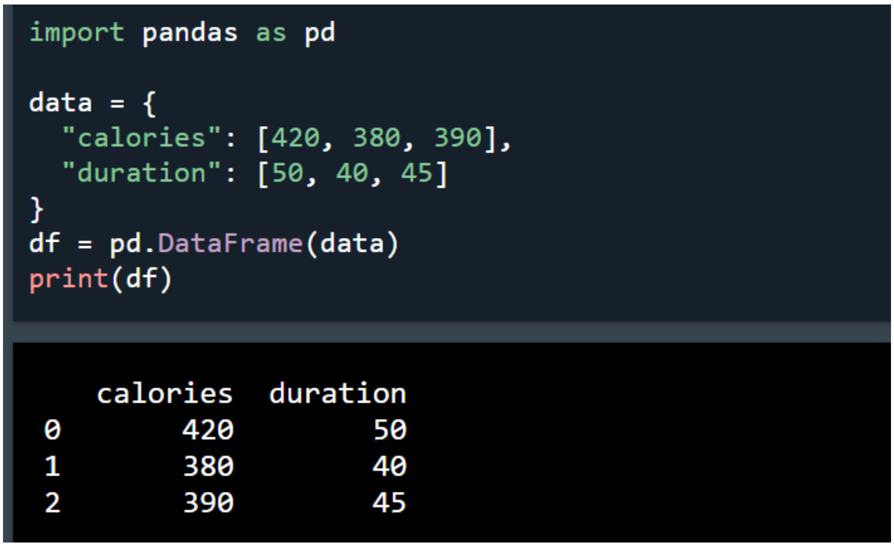


```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

DataFrames

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.



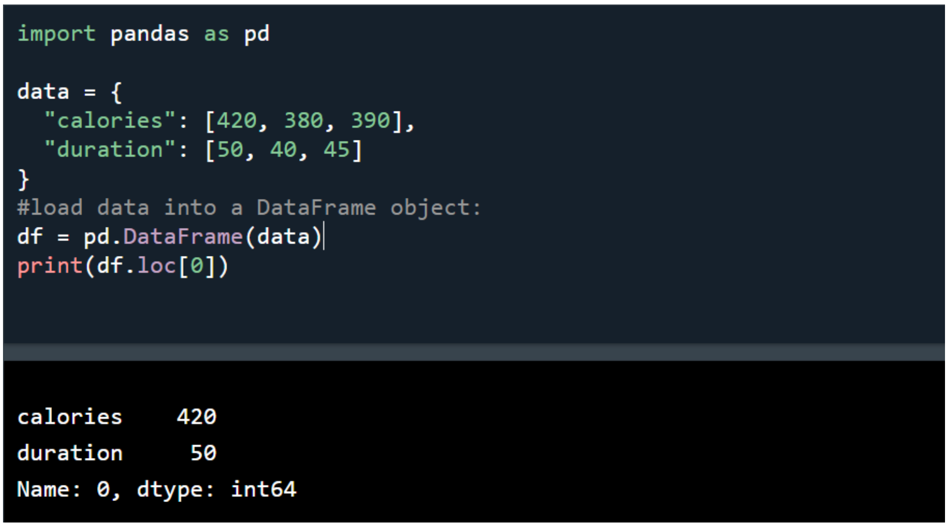
```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data)
print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

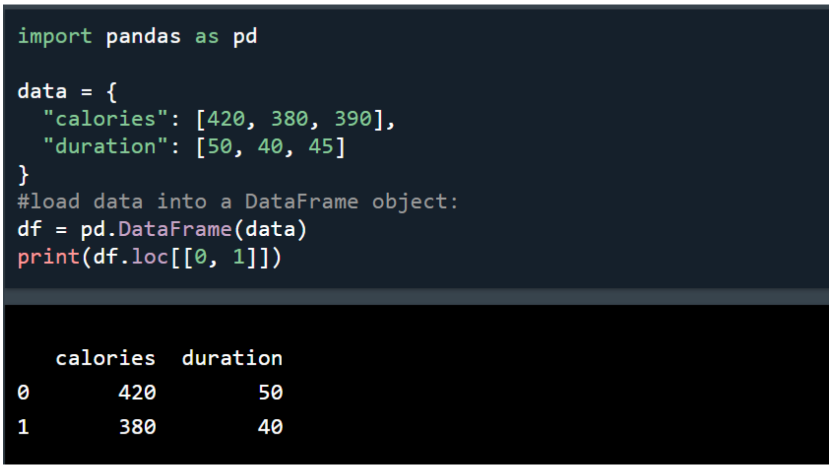


```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df.loc[0])
```

```
calories    420
duration     50
Name: 0, dtype: int64
```

Example

Return row 0 and 1:



```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df.loc[[0, 1]])
```

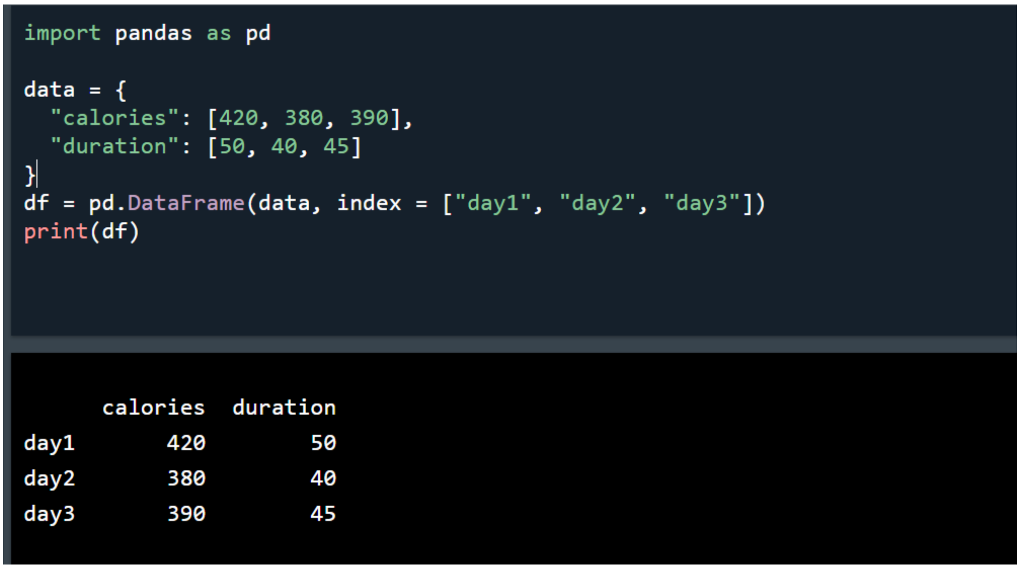
```
   calories  duration
0       420        50
1       380        40
```

Named Indexes

With the index argument, you can name your own indexes.

Example

Add a list of names to give each row a name:

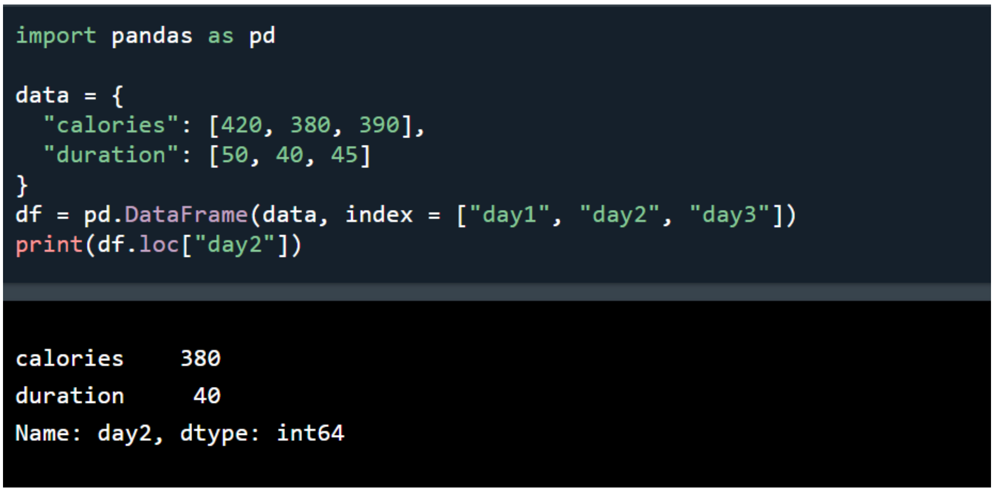


```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Locate Named Indexes

Use the named index in the loc attribute to return the specified row(s).

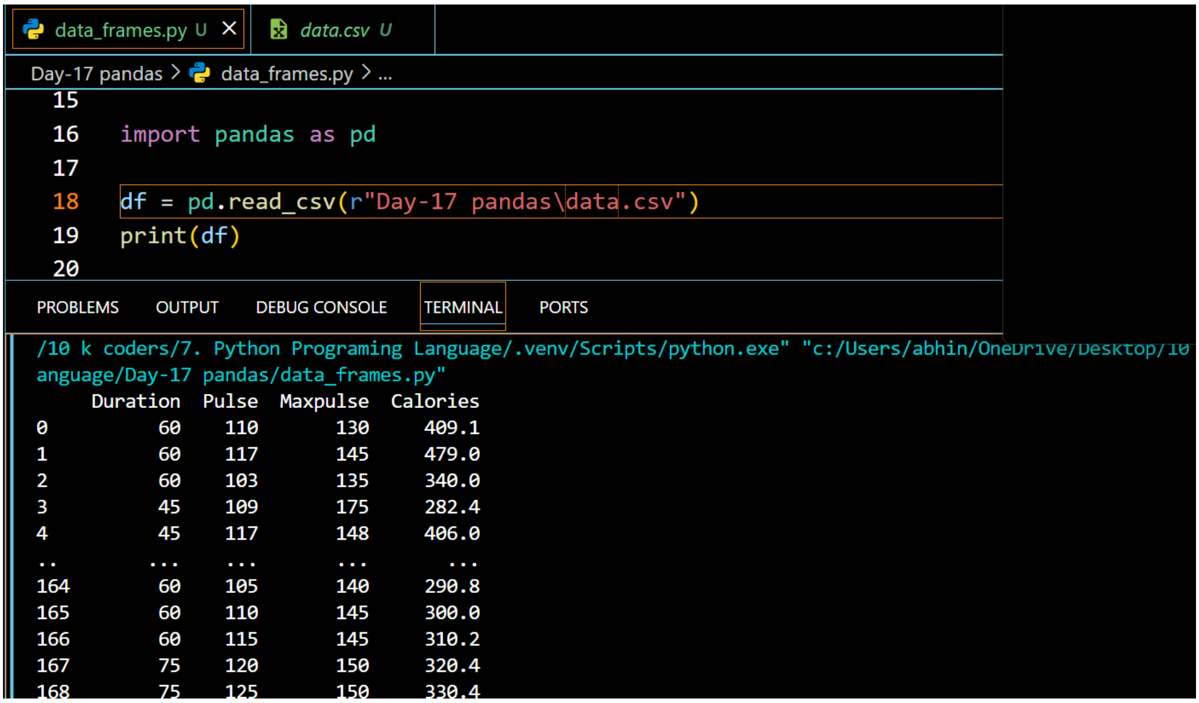


```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df.loc["day2"])
```

```
calories    380
duration     40
Name: day2, dtype: int64
```

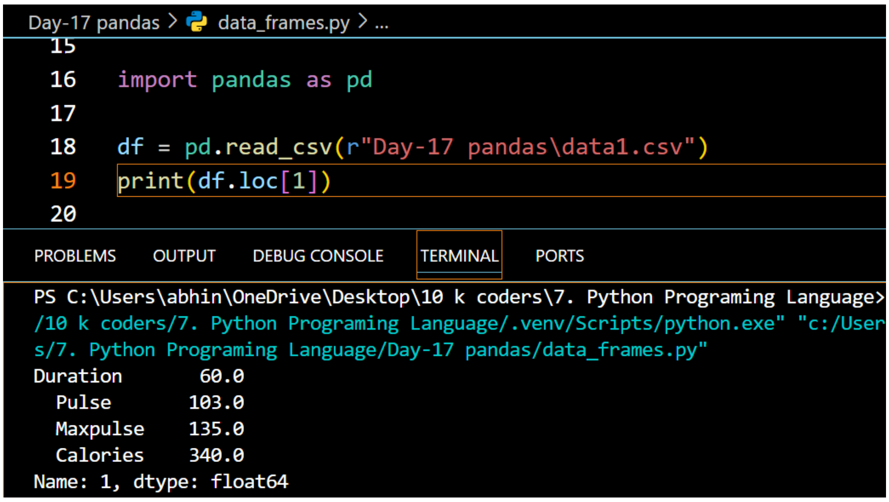
Read CSV Files

- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.
- In our examples we will be using a CSV file called 'data.csv'.
- Download data.csv. or Open data.csv.



```

import pandas as pd

df = pd.read_csv(r"Day-17 pandas\data.csv")
print(df)

```

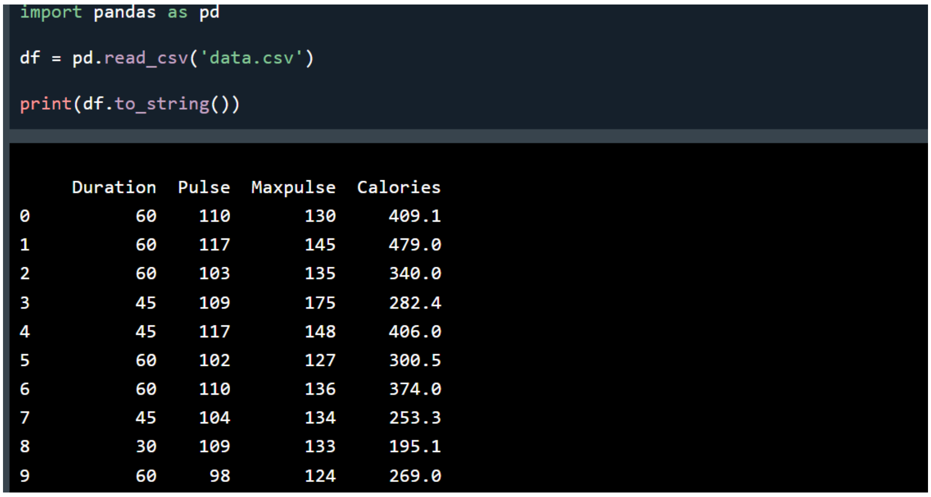
```
/10 k coders/7. Python Programing Language/.venv/Scripts/python.exe" "c:/Users/abhin/OneDrive/Desktop/10
anguage/Day-17 pandas/data_frames.py"
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```



```python
import pandas as pd

df = pd.read_csv(r"Day-17 pandas\data1.csv")
print(df.loc[1])
```

```
PS C:\Users\abhin\OneDrive\Desktop\10 k coders\7. Python Programing Language>
/10 k coders/7. Python Programing Language/.venv/Scripts/python.exe" "c:/User
s/7. Python Programing Language/Day-17 pandas/data_frames.py"
Duration       60.0
  Pulse       103.0
  Maxpulse    135.0
  Calories    340.0
Name: 1, dtype: float64
```

Tip: use to_string() to print the entire DataFrame.

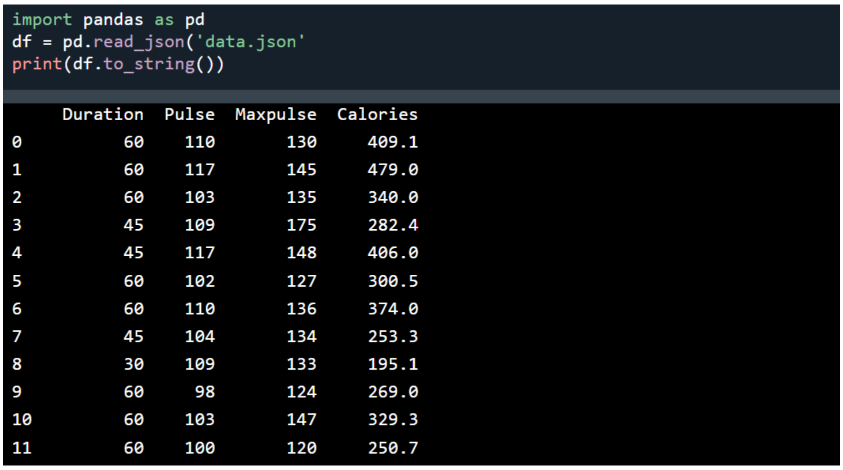


```python
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
```

Read JSON

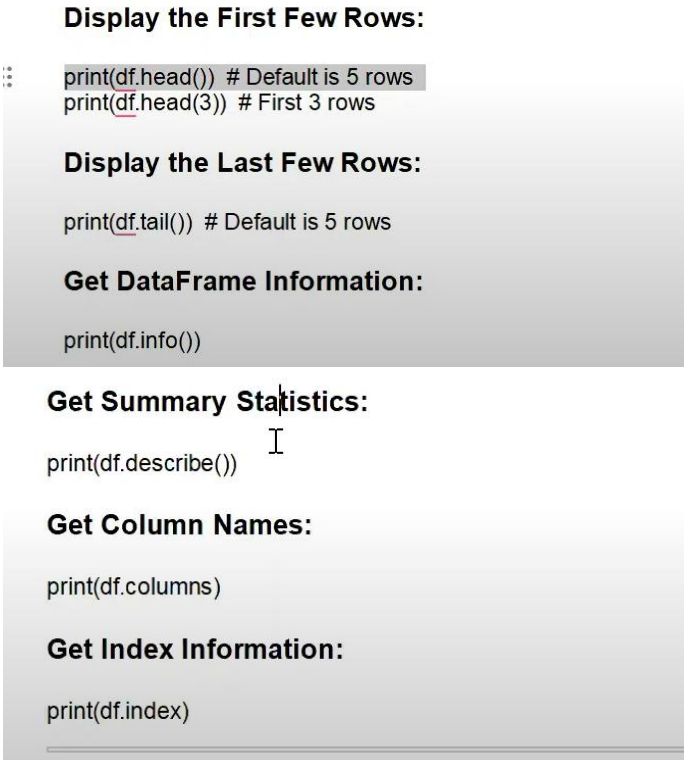
Big data sets are often stored, or extracted as JSON.
JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.
In our examples we will be using a JSON file called 'data.json'



```python
import pandas as pd
df = pd.read_json('data.json'
print(df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
5          60    102       127     300.5
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
```

Basic Methods:



Display the First Few Rows:

```
print(df.head())  # Default is 5 rows
print(df.head(3))  # First 3 rows
```

Display the Last Few Rows:

```
print(df.tail())  # Default is 5 rows
```

Get DataFrame Information:

```
print(df.info())
```

Get Summary Statistics:

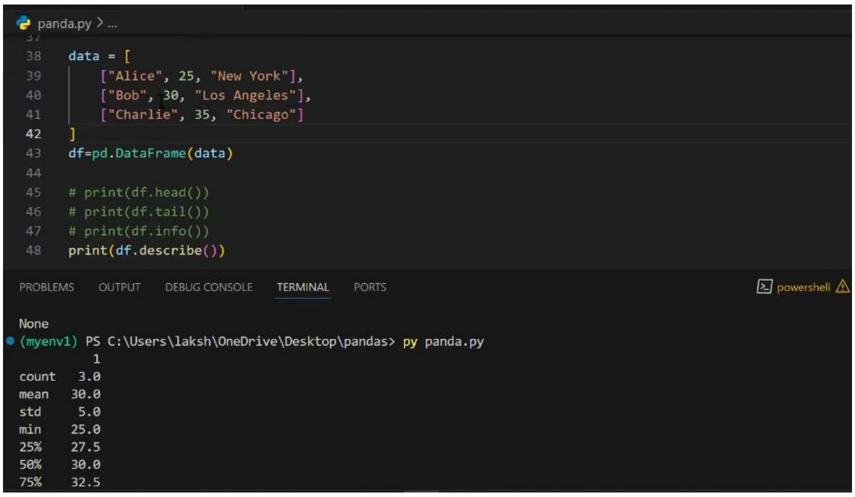
```
print(df.describe())
```

Get Column Names:

```
print(df.columns)
```

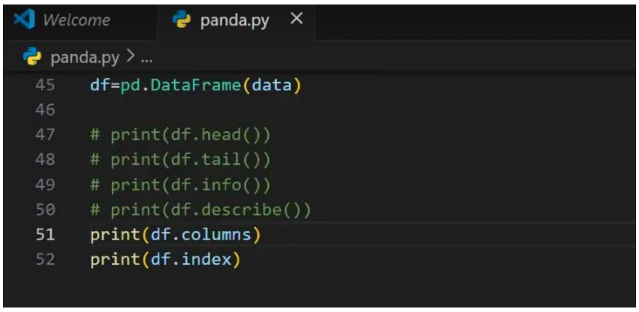
Get Index Information:

```
print(df.index)
```

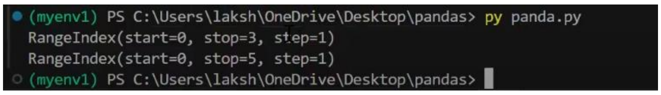


```
data = [
    ["Alice", 25, "New York"],
    ["Bob", 30, "Los Angeles"],
    ["Charlie", 35, "Chicago"]
]
df=pd.DataFrame(data)

# print(df.head())
# print(df.tail())
# print(df.info())
print(df.describe())
```

```
None
(myenv1) PS C:\Users\laksh\OneDrive\Desktop\pandas> py panda.py
          1
count   3.0
mean   30.0
std     5.0
min    25.0
25%    27.5
50%    30.0
75%    32.5
```

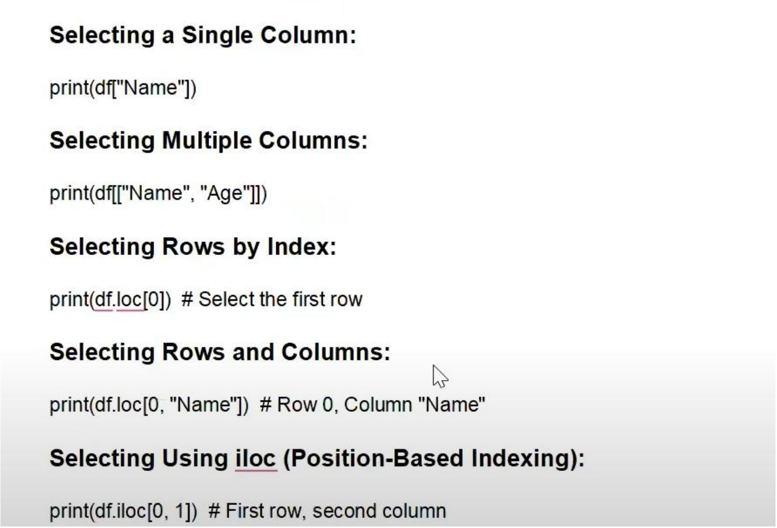


```
df=pd.DataFrame(data)

# print(df.head())
# print(df.tail())
# print(df.info())
# print(df.describe())
print(df.columns)
print(df.index)
```



```
(myenv1) PS C:\Users\laksh\OneDrive\Desktop\pandas> py panda.py
RangeIndex(start=0, stop=3, step=1)
RangeIndex(start=0, stop=5, step=1)
(myenv1) PS C:\Users\laksh\OneDrive\Desktop\pandas>
```

Selecting methods



Selecting a Single Column:

print(df["Name"])

Selecting Multiple Columns:

print(df[["Name", "Age"]])

Selecting Rows by Index:

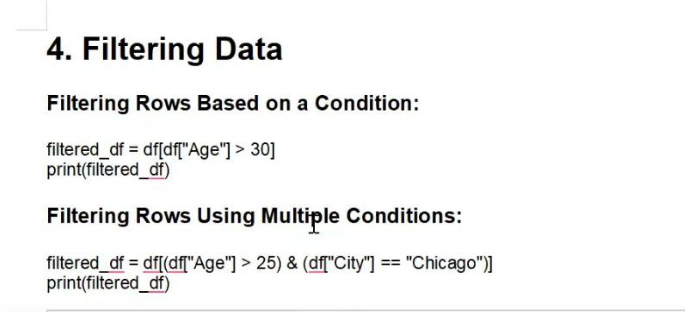
print(df.loc[0]) # Select the first row

Selecting Rows and Columns:

print(df.loc[0, "Name"]) # Row 0, Column "Name"

Selecting Using iloc (Position-Based Indexing):

print(df.iloc[0, 1]) # First row, second column



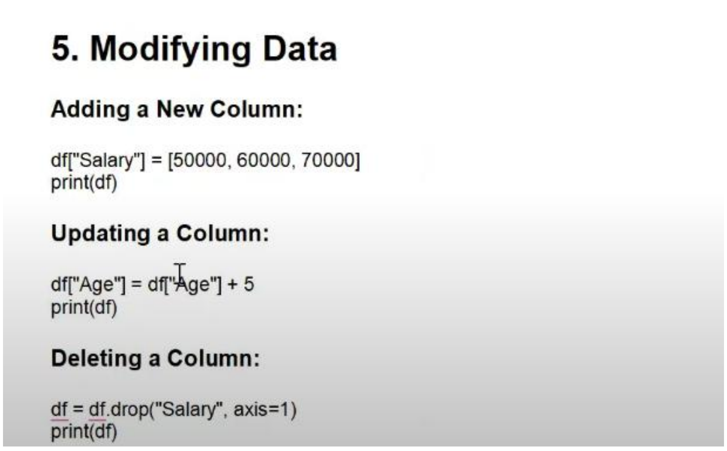
4. Filtering Data

Filtering Rows Based on a Condition:

```
filtered_df = df[df["Age"] > 30]
print(filtered_df)
```

Filtering Rows Using Multiple Conditions:

```
filtered_df = df[(df["Age"] > 25) & (df["City"] == "Chicago")]
print(filtered_df)
```



5. Modifying Data

Adding a New Column:

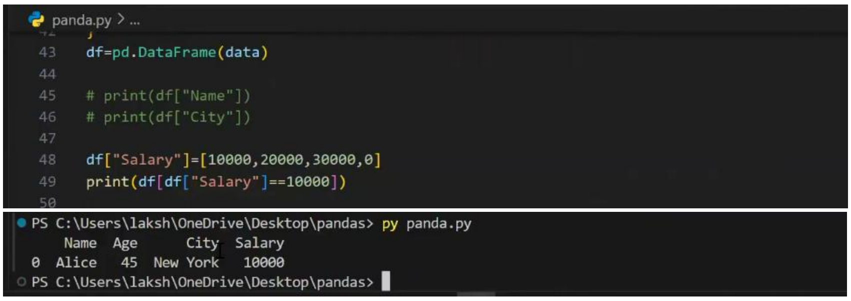
```
df["Salary"] = [50000, 60000, 70000]
print(df)
```

Updating a Column:

```
df["Age"] = df["Age"] + 5
print(df)
```

Deleting a Column:

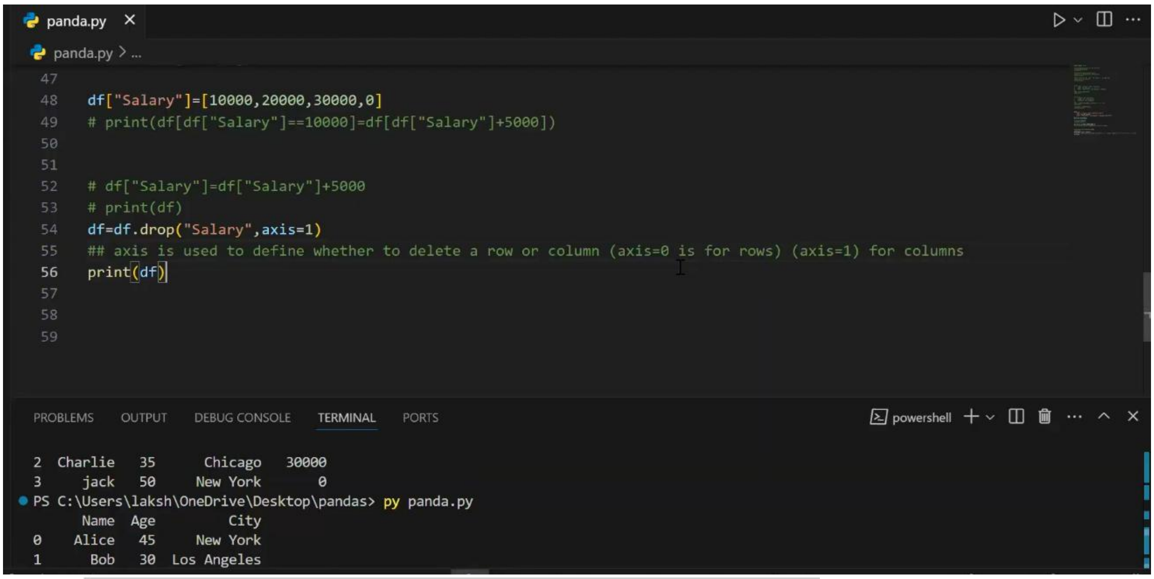
```
df = df.drop("Salary", axis=1)
print(df)
```



panda.py > ...

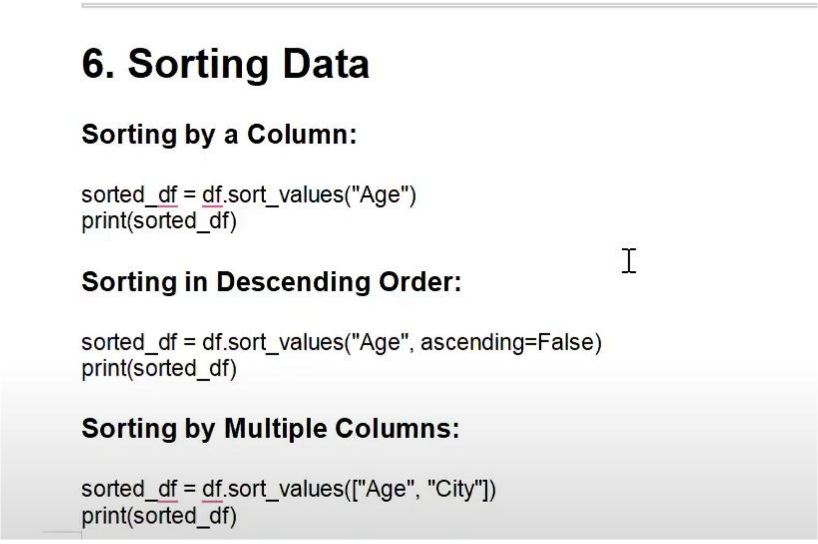
```
df=pd.DataFrame(data)

# print(df["Name"])
# print(df["City"])

df["Salary"]=[10000,20000,30000,0]
print(df[df["Salary"]==10000])
```

```
PS C:\Users\laksh\OneDrive\Desktop\pandas> py panda.py
    Name  Age      City  Salary
0  Alice   45  New York   10000
PS C:\Users\laksh\OneDrive\Desktop\pandas> 
```



```python

df["Salary"]=[10000,20000,30000,0]
# print(df[df["Salary"]==10000]=df[df["Salary"]+5000])


# df["Salary"]=df["Salary"]+5000
# print(df)
df=df.drop("Salary",axis=1)
## axis is used to define whether to delete a row or column (axis=0 is for rows) (axis=1) for columns
print(df)
```

```
2  Charlie   35      Chicago   30000
3     jack   50     New York       0
PS C:\Users\laksh\OneDrive\Desktop\pandas> py panda.py
      Name  Age         City
0    Alice   45     New York
1      Bob   30  Los Angeles
```



6. Sorting Data

Sorting by a Column:

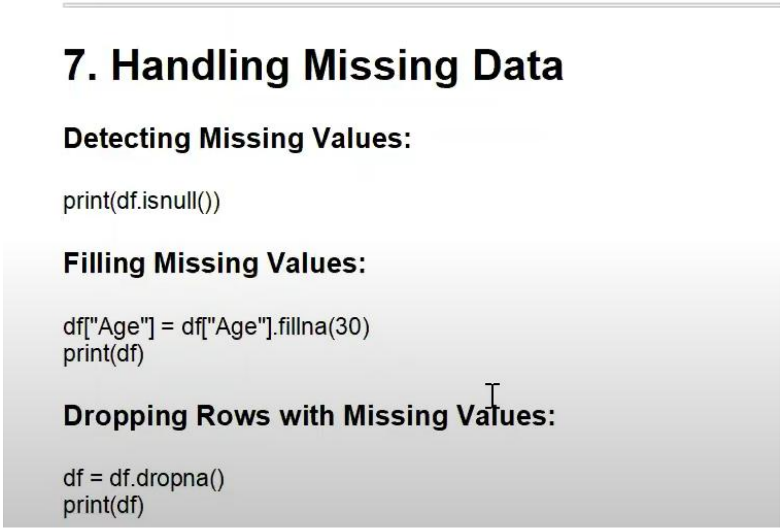
```
sorted_df = df.sort_values("Age")
print(sorted_df)
```

Sorting in Descending Order:

```
sorted_df = df.sort_values("Age", ascending=False)
print(sorted_df)
```

Sorting by Multiple Columns:

```
sorted_df = df.sort_values(["Age", "City"])
print(sorted_df)
```



7. Handling Missing Data

Detecting Missing Values:

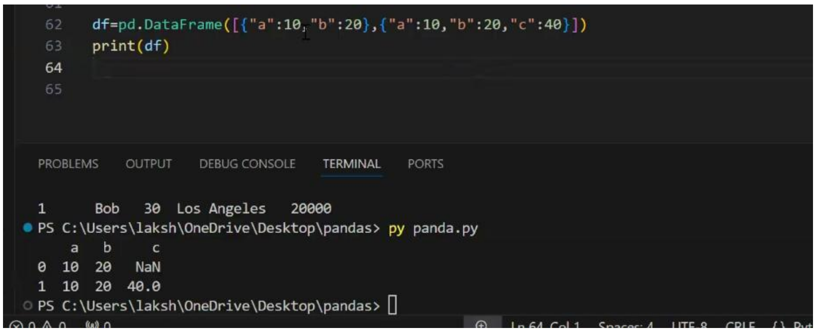
```
print(df.isnull())
```

Filling Missing Values:

```
df["Age"] = df["Age"].fillna(30)
print(df)
```

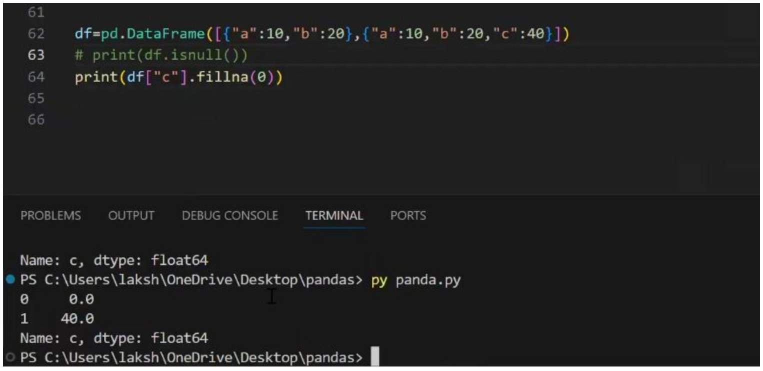
Dropping Rows with Missing Values:

```
df = df.dropna()
print(df)
```



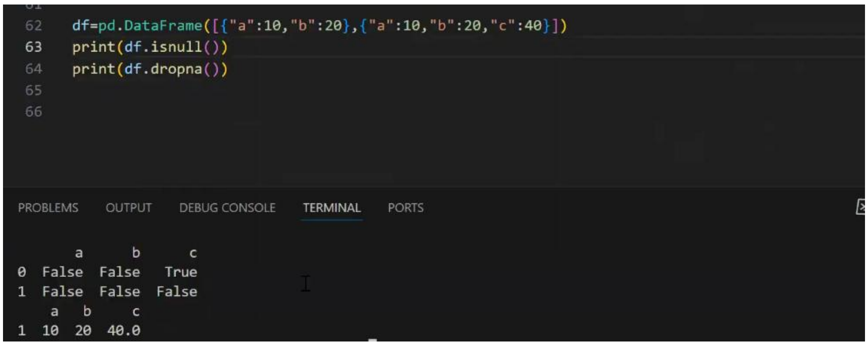
```python
df=pd.DataFrame([{"a":10,"b":20},{"a":10,"b":20,"c":40}])
print(df)
```

```
1      Bob   30  Los Angeles   20000
PS C:\Users\laksh\OneDrive\Desktop\pandas> py panda.py
    a   b     c
0  10  20   NaN
1  10  20  40.0
PS C:\Users\laksh\OneDrive\Desktop\pandas>
```



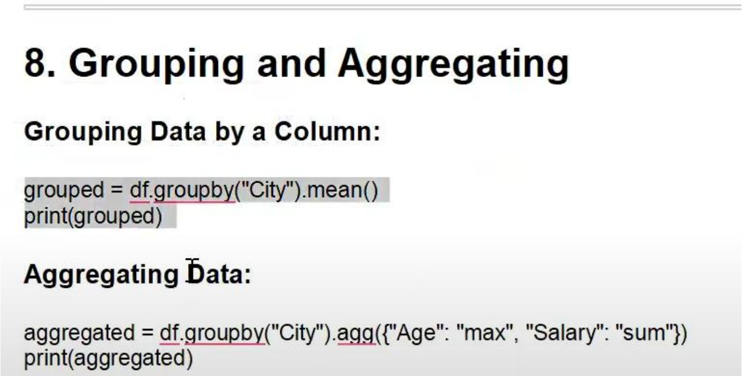
```python
df=pd.DataFrame([{"a":10,"b":20},{"a":10,"b":20,"c":40}])
# print(df.isnull())
print(df["c"].fillna(0))
```

```
Name: c, dtype: float64
PS C:\Users\laksh\OneDrive\Desktop\pandas> py panda.py
0     0.0
1    40.0
Name: c, dtype: float64
PS C:\Users\laksh\OneDrive\Desktop\pandas>
```



```python
df=pd.DataFrame([{"a":10,"b":20},{"a":10,"b":20,"c":40}])
print(df.isnull())
print(df.dropna())
```

```
       a      b      c
0  False  False   True
1  False  False  False
    a   b     c
1  10  20  40.0
```



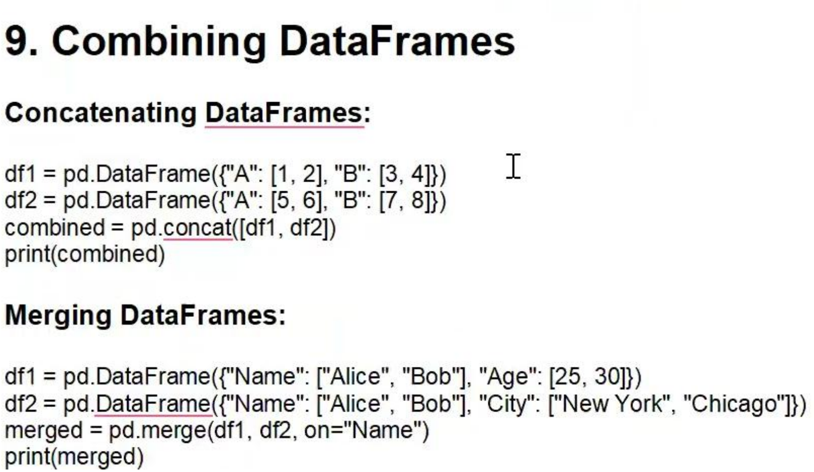
8. Grouping and Aggregating

Grouping Data by a Column:

```python
grouped = df.groupby("City").mean()
print(grouped)
```

Aggregating Data:

```python
aggregated = df.groupby("City").agg({"Age": "max", "Salary": "sum"})
print(aggregated)
```



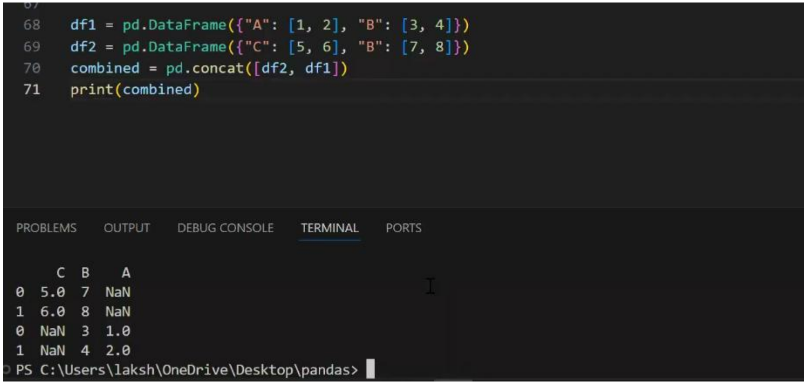
9. Combining DataFrames

Concatenating DataFrames:

```
df1 = pd.DataFrame({"A": [1, 2], "B": [3, 4]})
df2 = pd.DataFrame({"A": [5, 6], "B": [7, 8]})
combined = pd.concat([df1, df2])
print(combined)
```

Merging DataFrames:

```
df1 = pd.DataFrame({"Name": ["Alice", "Bob"], "Age": [25, 30]})
df2 = pd.DataFrame({"Name": ["Alice", "Bob"], "City": ["New York", "Chicago"]})
merged = pd.merge(df1, df2, on="Name")
print(merged)
```

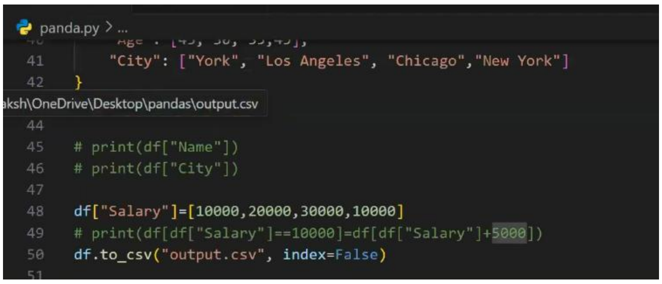


```
df1 = pd.DataFrame({"A": [1, 2], "B": [3, 4]})
df2 = pd.DataFrame({"C": [5, 6], "B": [7, 8]})
combined = pd.concat([df2, df1])
print(combined)
```

```
     C  B    A
0  5.0  7  NaN
1  6.0  8  NaN
0  NaN  3  1.0
1  NaN  4  2.0
PS C:\Users\laksh\OneDrive\Desktop\pandas>
```

Writing to a CSV file:

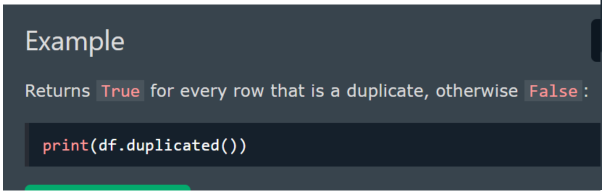
df.to_csv(‘output.csv’, index=False)



```
        "City": ["York", "Los Angeles", "Chicago","New York"]
    }
aksh\OneDrive\Desktop\pandas\output.csv

    # print(df["Name"])
    # print(df["City"])

    df["Salary"]=[10000,20000,30000,10000]
    # print(df[df["Salary"]==10000]=df[df["Salary"]+5000])
    df.to_csv("output.csv", index=False)
```

Removing Duplicates

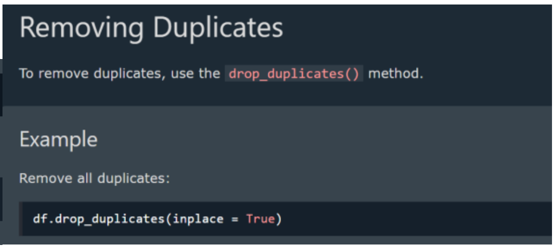
To discover duplicates, we can use the duplicated() method.
The duplicated() method returns a Boolean values for each row:



Example

Returns True for every row that is a duplicate, otherwise False:

```
print(df.duplicated())
```



Removing Duplicates

To remove duplicates, use the drop_duplicates() method.

Example

Remove all duplicates:

```
df.drop_duplicates(inplace = True)
```

Pandas - Data Correlations

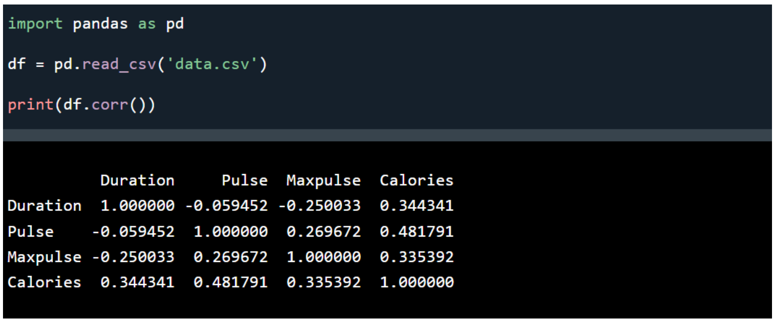
Finding Relationships

A great aspect of the Pandas module is the corr() method.

The corr() method calculates the relationship between each column in your data set.

The examples in this page uses a CSV file called: 'data.csv'.

Download data.csv. or Open data.csv



```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.corr())
```

```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.059452 -0.250033  0.344341
Pulse    -0.059452  1.000000  0.269672  0.481791
Maxpulse -0.250033  0.269672  1.000000  0.335392
Calories  0.344341  0.481791  0.335392  1.000000
```

Result Explained

- The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.
- The number varies from -1 to 1.
- 1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.
- 0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.
- -0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.
- 0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

Plotting

- Pandas uses the plot() method to create diagrams.
- We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.
- Read more about ploting