

DATA SCIENCE ASSIGNMENT

Pandas



UNDERSTANDING PANDAS LIBRARY

NAME: MOHAMMED USMAN
ROLL NO: 4830

SUBJECT: DATA SCIENCE
MSC IT PART-1
THAKUR COLLEGE
YEAR: 2023-2024



INTRODUCTION

- IN THIS ASSIGNMENT, YOU WILL EXPLORE THE FUNDAMENTALS OF PANDAS, A POWERFUL DATA MANIPULATION LIBRARY IN PYTHON. PANDAS IS ESSENTIAL FOR ANYONE WORKING WITH DATA ANALYSIS, CLEANING, AND MANIPULATION.
- PANDAS IS A HIGHLY VERSATILE AND WIDELY USED PYTHON LIBRARY FOR DATA MANIPULATION AND ANALYSIS. IT OFFERS A RICH SET OF DATA STRUCTURES AND FUNCTIONS THAT EMPOWER DATA SCIENTISTS, ANALYSTS, AND DEVELOPERS TO EFFICIENTLY WORK WITH STRUCTURED DATA.

What is Pandas?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.

Why Use Pandas?

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be easily loaded.
- Flexible reshaping and pivoting of data sets.
- Provides time-series functionality.

Where to Use Pandas?

Pandas is employed in numerous fields, including:

- Data analysis and exploration
- Data cleaning and preprocessing
- Statistical analysis and modeling

- Finance and economics
- Scientific research
- Business intelligence
- Machine learning and artificial intelligence

Key Benefits of the Pandas Package

Some key advantages of using Pandas include:

- Efficient data structures like DataFrames and Series
- Robust data cleaning and preprocessing capabilities
- Powerful data aggregation and analysis functions
- Simplified data visualization
- Flexibility to handle various data formats

What can you do using Pandas?

Pandas are generally used for data science.

- Data set cleaning, merging, and joining.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
- Columns can be inserted and deleted from DataFrame and higher dimensional objects.
- Powerful group by functionality for performing split-apply-omine operations on data sets.
- Data Visulaization

HERE ARE KEY POINTS TO INTRODUCE PANDAS

1. **Data Structures:** Pandas provides two fundamental data structures: Series and DataFrame.
 - i. **Series:** Represents one-dimensional data, similar to an array or a list.
 - ii. **DataFrame:** Organizes data in a tabular format with rows and columns, similar to a spreadsheet or database table.
2. **Data Handling:** Pandas is a *powerful tool* for loading, cleaning, and transforming data in a *variety of formats*, including CSV, Excel, and SQL.
3. **Data Exploration:** Pandas provides easy-to-use functions for filtering, sorting, and summarizing data, which is essential for data exploration.
4. **Data Visualization:** Pandas works well with Matplotlib and Seaborn to create data visualizations.
5. **Indexing and Selection:** Pandas indexing and selection lets you access specific data points or subsets of your dataset quickly.
6. **Time Series Data:** Pandas supports time series data, making it useful for financial analysis and forecasting.
7. **Integration with NumPy:** Pandas uses NumPy for efficient numerical and mathematical operations, making it suitable for scientific computing.
8. **Open Source:** Pandas is an open-source *Python library* with a large and active user base.
9. **Community and Documentation:** Pandas has good documentation and a supportive community, making it easy to learn and use.

Installation of Pandas

- If you have Python and PIP already installed on a system, then installation of Pandas is very easy.
- Install it using this command:

```
C:\>pip install pandas
```

 In Command Prompt

Import Pandas

- Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

Now Pandas is imported and ready to use.

```
import pandas
mydataset = {
    'cars': ["BMW", "Volvo", "Ford"],
    'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

OutPut:

	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.

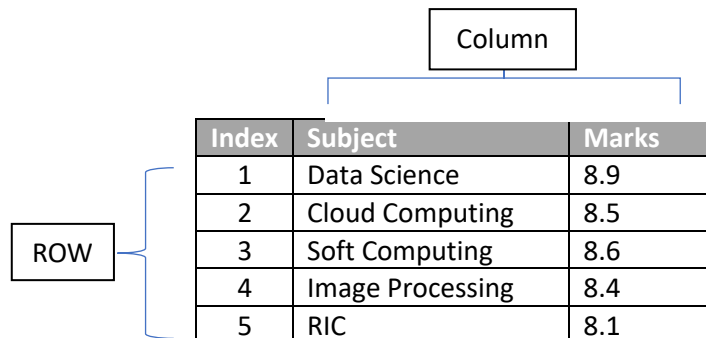
Example: Create a simple Pandas Series from a list:

```
import pandas as pd
a = [1, 7, 2]
```

```
myvar = pd.Series(a)
print(myvar)
```

OutPut:

```
0  1
1  7
2  2
```



The diagram illustrates a DataFrame structure. A box labeled 'Column' is positioned above a table, with a bracket indicating it applies to the columns. A box labeled 'ROW' is positioned to the left of the table, with a bracket indicating it applies to the rows. The table has three columns: 'Index', 'Subject', and 'Marks'. The rows are indexed from 1 to 5.

Index	Subject	Marks
1	Data Science	8.9
2	Cloud Computing	8.5
3	Soft Computing	8.6
4	Image Processing	8.4
5	RIC	8.1

Pandas DataFrame

In the real world,

- A Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file.
- Pandas DataFrame can be created from lists, dictionaries, and from a list of dictionaries, etc.

Example:

```
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)
```

OutPut:

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

Pandas 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'.

Example: Load the CSV into a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())
```

OutPut:

	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	Nan	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	Nan
10	60	103	147	329.3

Pandas 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.

Example: Load the JSON file into a DataFrame:

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

OutPut:

	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	Nan	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	Nan
10	60	103	147	329.3

Viewing the Data

i. head()

- There is a head() method for viewing the first rows of the DataFrame.
- The head() method returns the headers and a specified number of rows, starting from the top.

Example: Printing the first 5 rows of the DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head(4))
```

OutPut:

	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

i. tail()

- There is a tail() method for viewing the last rows of the DataFrame.
- tail() returns the last few rows of a DataFrame.

Example: Print the last 5 rows of the DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.tail())
```


OutPut:

	Duration	Pulse	Maxpulse	Calories
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	Nan
10	60	103	147	329.3

Pandas - Cleaning Data

- Data cleaning means fixing bad data in your data set.

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

i. Empty Cells

- Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

- One way to deal with empty cells is to remove rows that contain empty cells.

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2023/2/01'	110	130	409.1
1	60	'2023/2/02'	Nan	145	479.0
2	60	'2023/2/03'	103	135	340.0
3	45	'2023/2/04'	109	175	282.4
4	45	'2023/2/05'	117	148	Nan
5	60	'2023/2/06'	102	127	300.0
6	60	'2023/2/07'	110	136	374.0
7	450	'2023/2/08'	104	134	253.3
8	30	'2023/2/09'	109	133	195.1
9	60	'2023/2/10'	98	124	269.0
10	60	'2023/2/11'	103	147	Nan

Example: Return a new Data Frame with no empty cells:

```
import pandas as pd
df = pd.read_csv('data.csv')
new_df = df.dropna()
print(new_df.to_string())
```

OutPut:

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2023/2/01'	110	130	409.1
2	60	'2023/2/03'	103	135	340.0
3	45	'2023/2/04'	109	175	282.4
5	60	'2023/2/06'	102	127	300.0
6	60	'2023/2/07'	110	136	374.0
7	450	'2023/2/08'	104	134	253.3
8	30	'2023/2/09'	109	133	195.1
9	60	'2023/2/10'	98	124	269.0

- In the result that some rows have been removed (row 1, 4 and 10).
- These rows had cells with empty values.

ii. Data in Wrong Format

- Cells with data of wrong format can make it difficult, or even impossible, to analyze data.
- To fix it, you have two options: **remove the rows**, or **convert all cells in the columns into the same format**.
- Convert Into a Correct Format
- In our Data Frame, we have two cells with the wrong format. Check out row 1 and 7, the 'Date' column should be a string that represents a date:

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2023/2/01'	110	130	409.1
1	60	Nan	115	145	479.0
2	60	'2023/2/03'	103	135	340.0
3	45	'2023/2/04'	109	175	282.4
4	45	'2023/2/05'	117	148	Nan
5	60	'2023/2/06'	102	127	300.0
6	60	'2023/2/07'	110	136	374.0
7	450	2023208	104	134	253.3
8	30	'2023/2/09'	109	133	195.1
9	60	'2023/2/10'	98	124	269.0
10	60	'2023/2/11'	103	147	Nan

- Let's try to convert all cells in the 'Date' column into dates.
- Pandas has a `to_datetime()` method for this:

Example: *Convert to date:*

```
import pandas as pd
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

OutPut:

Duration		Date	Pulse	Maxpulse	Calories
0	60	'2023/2/01'	110	130	409.1
1	60	NaT	115	145	479.0
2	60	'2023/2/03'	103	135	340.0
3	45	'2023/2/04'	109	175	282.4
4	45	'2023/2/05'	117	148	Nan
5	60	'2023/2/06'	102	127	300.0
6	60	'2023/2/07'	110	136	374.0
7	450	'2023/2/08'	104	134	253.3
8	30	'2023/2/09'	109	133	195.1
9	60	'2023/2/10'	98	124	269.0
10	60	'2023/2/11'	103	147	Nan

As you can see from the result,

- The date in row '7' was fixed, but the empty date in row '1' got a NaT (Not a Time) value, in other words an empty value.
- One way to deal with empty values is simply removing the entire row.

iii. Wrong Data

- **Wrong data** does not have to be empty cells or wrong format, it can just be wrong, like if someone registered **199** instead of **1.99**.
- Spot wrong data by comparing it to expected values.
- You can see that in **row 7**, the duration is **450**, but for all the other rows the duration is between **30 and 60**.
- we conclude with the fact that this person did not work out in **450** minutes.

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3

Replacing Values

- One way to fix wrong values is to replace them with something else.
- In our example, it is most likely a typo, and the value should be "45" instead of **450**, and we could just insert **45** in **row 7**:

Example: Set "Duration" = 45 in row 7:

```
import pandas as pd
df = pd.read_csv('data.csv')
df.loc[7, 'Duration'] = 45
print(df.to_string())
```

OutPut:

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	45	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3

Removing Rows

- Another way of handling wrong data is to remove the rows that contains wrong data.
- This way, you Removing wrong data is easy and doesn't hurt analysis.

iv. Duplicates

Duplicate rows are rows that have been registered more than one time.

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/2/01'	110	130	409.1
1	60	'2020/2/02'	117	145	479.0
2	60	'2020/2/03'	103	135	340.0
3	45	'2020/2/04'	109	175	282.4
4	45	'2020/2/05'	117	148	406.0
5	60	'2020/2/06'	102	127	300.0
6	60	'2020/2/07'	110	136	374.0
7	60	'2020/2/07'	110	136	374.0
8	30	'2020/2/09'	109	133	195.1
9	60	'2020/2/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3

- By taking a look at our test data set, we can assume that row 6 and 7 are duplicates.
- To discover duplicates, we can use the uplicated() method.
- The uplicated() method returns a Boolean values for each row:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.duplicated())
```

Output:

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
7     True
8    False
9    False
10   False
```

Removing Duplicates:

- To remove duplicates, use the drop_duplicates() method.

```
import pandas as pd
df = pd.read_csv('data.csv')
df.drop_duplicates(inplace = True)
print(df.to_string())
```

OUTPUT :

	DURATION	DATE	PULSE	MAXPULSE	CALORIES
0	60	'2020/2/01'	110	130	409.1
1	60	'2020/2/02'	117	145	479.0
2	60	'2020/2/03'	103	135	340.0
3	45	'2020/2/04'	109	175	282.4
4	45	'2020/2/05'	117	148	406.0
5	60	'2020/2/06'	102	127	300.0
6	60	'2020/2/07'	110	136	374.0
8	30	'2020/2/09'	109	133	195.1
9	60	'2020/2/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3

#Notice that row 7 has been removed from the *OutPut*.

Plotting

- Plotting in Pandas is a way to visualize your data in different types of charts and graphs.
- Pandas has many plotting functions that can be *used* to *create* custom or pre-built plots.
- Plotting in Pandas is a great way to understand your data, find trends, and share your findings with others.

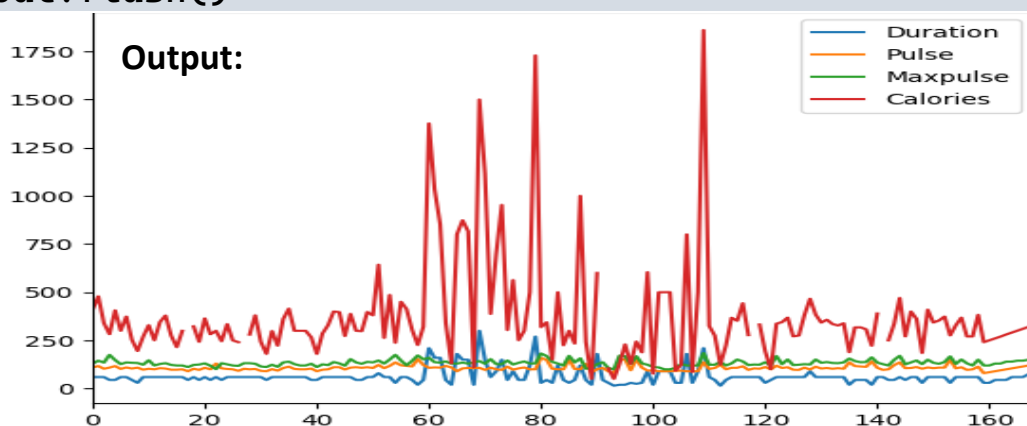
Example:

#Three lines to make our compiler able to draw:

```
import sys
import matplotlib
matplotlib.use('Agg')
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot()
plt.show()
```

#Two lines to make our compiler able to draw:

```
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



Scatter Plot

- A scatter plot is a graph that shows the relationship between two variables in a dataset.
- A scatter plot is a graph with two axes x and y, where each axis represents a different variable.
- The pattern of dots in a scatter plot shows the relationship between the two variables.
- Scatter plots are *used* to visualize data, find trends, and *assess* relationships between variables.

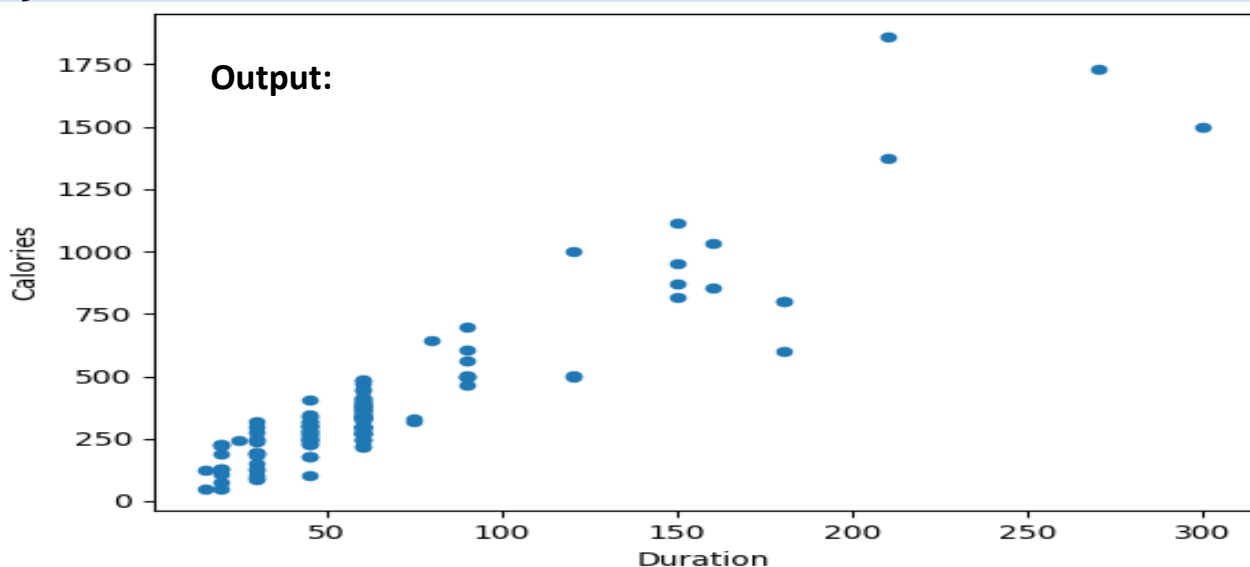
Example:

#Three lines to make our compiler able to draw:

```
import sys
import matplotlib
matplotlib.use('Agg')
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')
plt.show()
```

#Two lines to make our compiler able to draw:

```
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



Histogram

- A histogram in pandas is a graph that shows how data is distributed.
- A histogram is a bar graph that shows how often values appear in a dataset.
- Histograms are a useful way to visualize and understand the distribution of data.

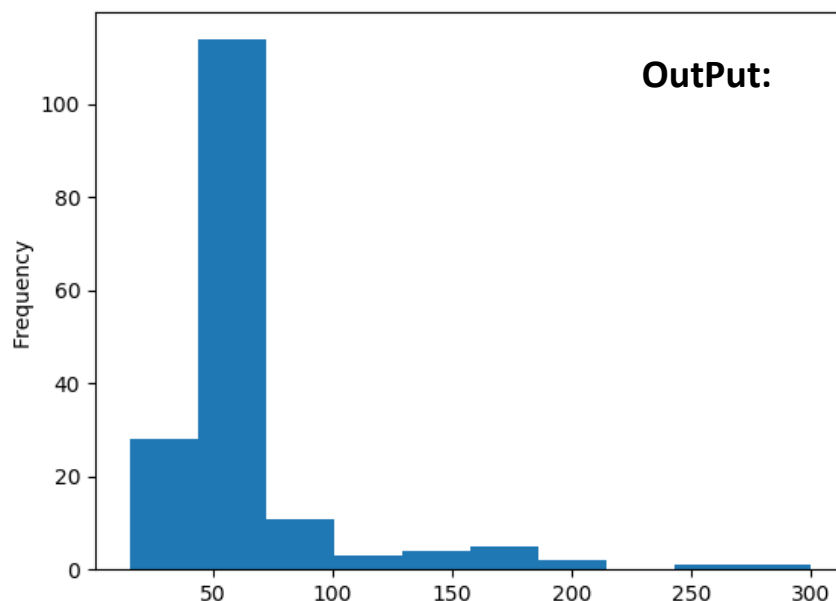
Example:

#Three lines to make our compiler able to draw:

```
import sys
import matplotlib
matplotlib.use('Agg')
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df["Duration"].plot(kind = 'hist')
plt.show()
```

#Two lines to make our compiler able to draw:

```
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



CONCLUSION

Pandas is a powerful Python library designed for data analysis.

Its primary data structures are DataFrames, which simplify analytical tasks in Python, making it a popular choice for data manipulation and exploration. Pandas provides a wide range of functions and capabilities for handling and analyzing data, making it a valuable tool in various domains, including data science, machine learning, and statistics.

Pandas is a crucial library for anyone dealing with data in Python. Its user-friendly and expressive syntax makes it a go-to choice for data manipulation, analysis, and exploration, whether you are a beginner or an experienced data professional.

Pandas allows users to work with large datasets, perform data cleaning, filtering, and transformation, and conduct statistical analyses. With its high-performance data structures and tools, Pandas is essential for anyone dealing with data in Python.

In summary, Pandas is a fundamental library that enhances Python's capabilities for data handling and analysis, making it a must-have for data professionals, researchers, and analysts. It simplifies the process of working with data, and its versatility is invaluable for tasks such as data preprocessing, exploration, and visualization.

THANK YOU