# 5CS037 - Concepts and Technologies of AI.
Week - 03 - Workshop - 03

# Exploratory Data Analysis - Part-II.

## Some Advance Operation with Pandas and Introduction to Matplotlib.!!!

Siman Giri

Workshop - 03

November - 27 - 2024

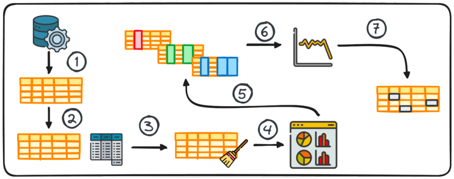
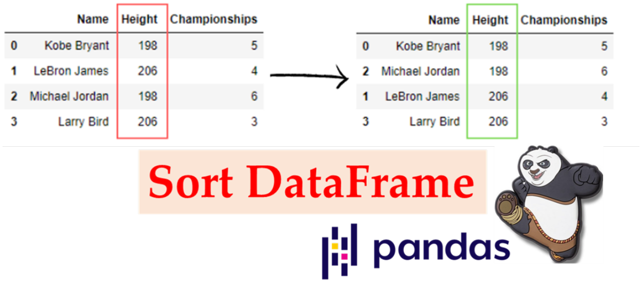


Image by : J. Ferrer – 7 steps to mastering Data Analysis {kDnuggets}

## Outline

## Advance Operation With Pandas.
## 1. Sorting and Susetting.



| | Name | Height | Championships |
|---|---|---|---|
| 0 | Kobe Bryant | 198 | 5 |
| 1 | LeBron James | 206 | 4 |
| 2 | Michael Jordan | 198 | 6 |
| 3 | Larry Bird | 206 | 3 |

| | Name | Height | Championships |
|---|---|---|---|
| 0 | Kobe Bryant | 198 | 5 |
| 2 | Michael Jordan | 198 | 6 |
| 1 | LeBron James | 206 | 4 |
| 3 | Larry Bird | 206 | 3 |

Sort DataFrame
pandas

## 1.1 Sorting:

Sorting arranges data in a specific order (ascending or descending) based on values or indices for better organization and analysis.

- `sort_values()`: Sorts a DataFrame or Series by its values.
- `sort_index()`: Sorts a DataFrame or Series by its index.

```python
# Syntax: Sorting by values
mydataframe.sort_values(by='column_name', ascending=True,
    inplace=False)

# Syntax: Sorting by index
mydataframe.sort_index(ascending=True, inplace=False)
```

## 1.1 Sorting - Code Example:

```python
import pandas as pd
# Creating a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [24, 19, 22, 25],
        'Score': [88, 92, 85, 95]}
df = pd.DataFrame(data)
print(df.head())
# Example 1: Sort by 'Age' using sort_values()
sorted_by_age = df.sort_values(by='Age')
print(sorted_by_age.head())
# Example 2: Sort by index using sort_index()
sorted_by_index = df.sort_index()
print(sorted_by_index.head())
```

## 1.2 Subsetting by Indices:

When working with data, you may not need all of the variables in your dataset.

- Subsetting allows you to create a smaller, focused DataFrame by selecting specific rows or columns using indices (positional or labels) or directly by column names, tailoring the data for analysis.

Subsetting by indices uses[] which allows you to extract specific rows, columns, or values from a DataFrame for further analysis.

- `iloc[]`: Access rows and columns using integer indices.

```
dataframe.iloc[row_start:row_end, col_start:col_end]
```

- `loc[]`: Access rows and columns by labels and conditions.

```
dataframe.loc[condition, ['col1', 'col2']]
```

- `[]` (brackets): Used for selecting columns.

```
dataframe[['col1', 'col2']]
```

## 1.2.1 Subsetting By Indices: Examples

```python
import pandas as pd
# Creating a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [24, 19, 22, 25],
        'Score': [88, 92, 85, 95]}
df = pd.DataFrame(data)
# 1.Using iloc[]:Accessing rows and columns by index
subset_iloc = df.iloc[1:3, 0:2]
print(subset_iloc)
# 2.Using loc[]:Accessing rows by condition and specific
    columns
subset_loc = df.loc[df['Age'] > 20, ['Name', 'Score']]
print(subset_loc)
# 3.Using []: Selecting specific columns
subset_brackets = df[['Name', 'Age']]
print(subset_brackets)
```

## 1.2.2 Subsetting by Values - Columns

Subsetting columns by values allows you to select specific columns directly using their names, either as a single column or multiple columns.

- Single Column: `dataframe['column_name']`
- Multiple Columns: `dataframe[['column_name1', 'column_name2']]`

```python
import pandas as pd
# Creating a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [24, 19, 22, 25],
        'Score': [88, 92, 85, 95]}
df = pd.DataFrame(data)
# Subsetting a single column
name_column = df['Name']
print(name_column)
# Subsetting multiple columns
name_and_age = df[['Name', 'Age']]
print(name_and_age)
```

## 1.3 Subsetting by Rows (Filtering):

Subsetting by rows, also known as filtering, allows you to extract rows from a DataFrame that meet specific conditions.

- Filter with a Single Condition: `dataframe[dataframe['column_name'] condition]`
- Filter with Multiple Conditions: You can filter for multiple conditions at once by using the `bitwise "and" operator,` → & `dataframe[(cond1) & (cond2)]`

```python
import pandas as pd
# df: Dataframe from slide 09
# Filter rows with a single condition (Age > 20)
filtered_single = df[df['Age'] > 20]
print(filtered_single)
# Filter rows with multiple conditions (Age > 20 and Score >
    85)
filtered_multiple = df[(df['Age'] > 20) & (df['Score'] > 85)
    ]
print(filtered_multiple)
```

## 1.3.1 Filtering on Categorical Values

Filtering on categorical values allows you to subset rows based on column content.

- Single Category Filter: Extracts rows where a specific category matches `dataframe[dataframe['categorical_col'] == 'category']`.
- Multiple Category Filter:Extracts rows where a column matches any value in a list, `dataframe[dataframe['categorical_col'].isin(['cat1', 'cat2'])]`.

## 1.3.2 Code Example: Filtering on Categorical Values

```python
import pandas as pd
# Creating a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Department': ['HR', 'IT', 'Finance', 'HR'],
        'Score': [88, 92, 85, 95]}
df = pd.DataFrame(data)
# Filter rows where 'Department' is 'HR'
filtered_single = df[df['Department'] == 'HR']
print(filtered_single)
# Filter rows where 'Department' is either 'HR' or 'IT'
filtered_multiple = df[df['Department'].isin(['HR', 'IT'])]
print(filtered_multiple)
```

## Advance Operation With Pandas.
## 2. The Group By Method.

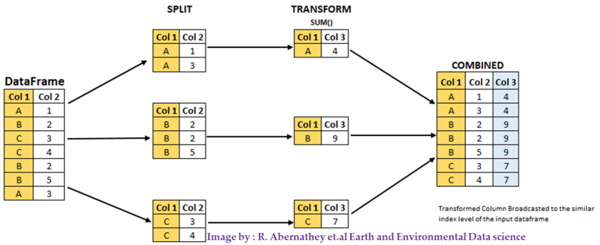
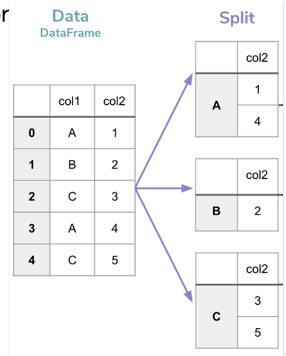


Image by : R. Abernathey et.al Earth and Environmental Data science

## 2.1 Group By Method - Introduction:

The Group By operation in pandas is a powerful tool for aggregating, transforming, or analyzing data. It follows the Split-Apply-Combine pattern:

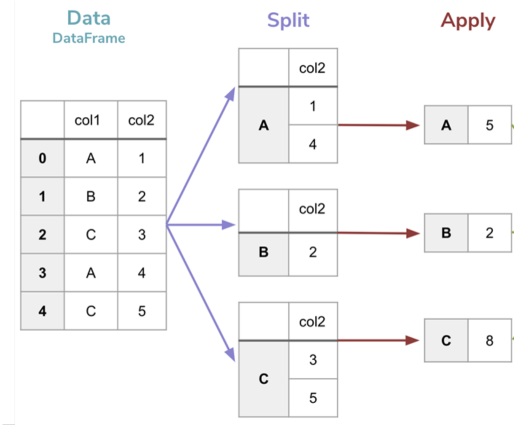
1. Split:
   - Divides data into groups based on values in one or more columns.
   - Creates subgroups of the dataset where each subgroup has a common key or categor



```
result = data.groupby('col1')['col2'].sum()
```

## 2.2 Group By Method - Apply:

2. Apply: Perform operations within each group.The Operation could be grouped to:
   - Aggregation; Transformation and Filtration.



```
result = data.groupby('col1')['col2'].sum()
```

## 2.2.1 Group By Method - Apply - Aggregation:

2. Apply: Perform operations within each group.The Operation could be grouped to:
   (a) Aggregation:Aggregation computes summary statistics for each group.
      - Purpose: Calculate metrics like mean, sum, max, etc., within each group.
      - Syntax: `dataframe.groupby('column').agg(func)`

```python
import pandas as pd
# Sample DataFrame
data = {'Category': ['A', 'B', 'A', 'B', 'A'],
        'Value': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)
# Aggregation: Calculate mean for each group
grouped = df.groupby('Category')['Value'].mean()
print(grouped)
```

## 2.2.2 Group By Method - Apply - Transformation:

2. Apply: Perform operations within each group.The Operation could be grouped to:
   (b) Transformation: Transformation applies a function to transform data within each group.
   - Purpose:Normalize or scale values within each group.
   - Syntax: `dataframe.groupby('column').transform(func)`

```python
import pandas as pd
# Sample DataFrame
data = {'Category': ['A', 'B', 'A', 'B', 'A'],
        'Value': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)
# Transformation: Normalize values within each group
df['Normalized'] = df.groupby('Category')['Value'].
    transform(lambda x: x / x.sum())
print(df)
```

## 2.2.3 Group By Method - Apply - Filtration:

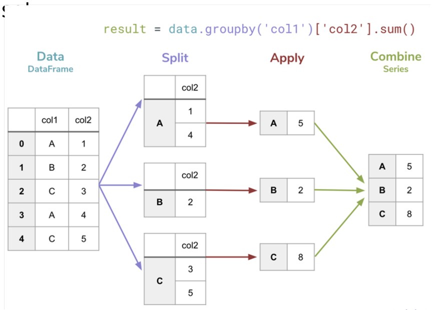
2. Apply: Perform operations within each group.The Operation could be grouped to:
   (c) Filtration: Transformation applies a function to transform data within each group.
   - Purpose:Exclude groups based on a condition.
   - Syntax: `dataframe.groupby('column').filter(func)`

```python
import pandas as pd
# Sample DataFrame
data = {'Category': ['A', 'B', 'A', 'B', 'A'],
        'Value': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)
# Filtration: Keep groups where sum of values > 60
filtered = df.groupby('Category').filter(lambda x: x
    ['Value'].sum() > 60)
print(filtered)
```

## 2.1 Group By Method - Introduction:

The Group By operation in pandas is a powerful tool for aggregating, transforming, or analyzing data. It follows the Split-Apply-Combine pattern:

3. Combine:
   - Combines the results from each subgroup back into a single DataFrame or Series.
   - Output can be a new DataFrame with group keys as the index or a modified datas



```
result = data.groupby('col1')['col2'].sum()
```

## Getting Started with Data Visualizations.
## 3. What is Data Visualizations?.

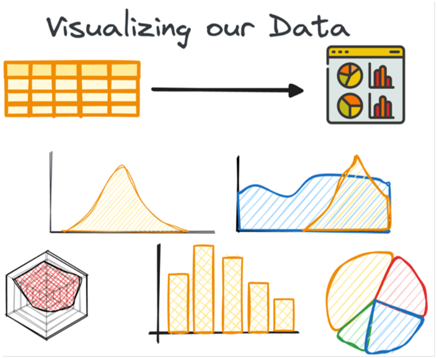


Image by : J. Ferrer – 7 steps to mastering Data Analysis {kDnuggets}

## 3.1 Data Visualization: Introduction

*The greatest value of a picture is when it forces us to*
*notice what we never expected to see.*

— *John Tukey*

*Disclaimer: Data Visualization is not just about the ability to create plots; it's about understanding which plots to use and what insights to draw from them.*

— *Siman Giri*

## 3.2 Data Visualization: Goals

- **Communicate(Explanatory)**
  - Present data and ideas.
  - Explain and inform.
  - Provide evidence and support.
  - Influence and persuade.
- **Analyze (Exploratory)**
  - Explore the data.
  - Assess a situation.
  - Determine how to proceed.
  - Decide what to do.

## 3.3 Effective Visualizations:

Followings are must for an effective visualizations:

1. Have a graphical integrity.
2. Keep it simple.
3. Use the right display.
4. Use color sensibly.

## 3.4 Tools for Visualizations:

Followings are must for an effective visualizations: Following three tools are the most popular tools used for **Data Visualizations and Exploration.** They are inter-mixable.

- Pandas Visualization module.
- Matplotlib
- Seaborn

```
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt

```

## Getting Started with Data Visualizations.
## 4. Pandas for Data Visualizations.

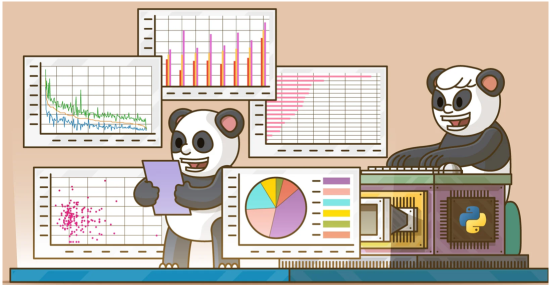


Image – R. Horvath – ReapPython.

## 4.1 Data Visualization with pandas

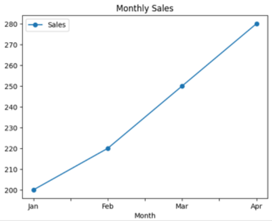
pandas offers built-in plotting capabilities powered by Matplotlib for quick and easy visualizations.

- Syntax:`dataframe.plot(kind='plot_type', ...)`
- Common plot types:
  - Line Plot
  - Bar Chart
  - Histogram
  - Scatter Plot
  - Box Plot

## 4.2 Line Plot Example

Line plots are useful for visualizing trends over time.

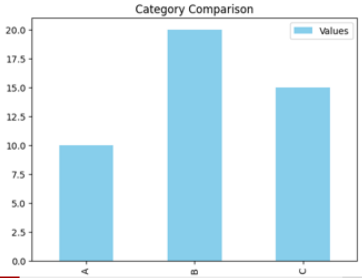
```python
import pandas as pd
import matplotlib.pyplot as plt
# Sample Data
data = {'Month': ['Jan', 'Feb', 'Mar', 'Apr'],
        'Sales': [200, 220, 250, 280]}
df = pd.DataFrame(data)
# Line Plot
df.plot(x='Month', y='Sales', kind='line', marker='o', title
    ='Monthly Sales')
plt.show()
```



## 4.3 Barchart Example

Bar charts are effective for comparing categories.

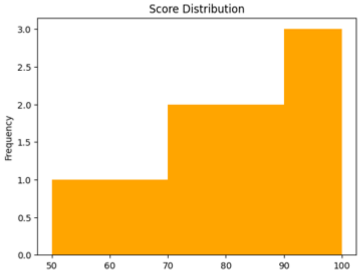
```python
# Sample Data
data = {'Category': ['A', 'B', 'C'],
        'Values': [10, 20, 15]}
df = pd.DataFrame(data)
# Bar Chart
df.plot(x='Category', y='Values', kind='bar', title='
    Category Comparison', color='skyblue')
plt.show()
```



## 4.4 Histogram Example

Histograms display the distribution of a single variable.

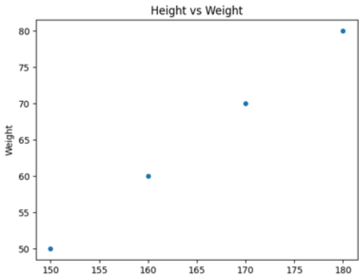
```python
#import---
# Sample Data
data = {'Scores': [50, 60, 70, 75, 80, 85, 90, 95, 100]}
df = pd.DataFrame(data)
# Histogram
df['Scores'].plot(kind='hist', bins=5, title='Score
    Distribution', color='orange')
plt.show()
```



## 4.5 Scatter Plot Example

Scatter plots show relationships between two variables.

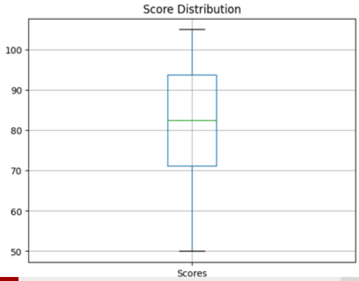
```
# Sample Data
data = {'Height': [150, 160, 170, 180],
        'Weight': [50, 60, 70, 80]}
df = pd.DataFrame(data)
# Scatter Plot
df.plot(x='Height', y='Weight', kind='scatter', title='
    Height vs Weight')
plt.show()
```



## 4.6 Box Plot Example

Box plots are used for visualizing the spread and outliers in data.

```
# Sample Data
data = {'Scores': [50, 60, 70, 75, 80, 85, 90, 95, 100,
    105]}
df = pd.DataFrame(data)
# Box Plot
df.boxplot(column='Scores')
plt.title('Score Distribution')
plt.show()
```



## 4.7 Comparison: Matplotlib vs. pandas

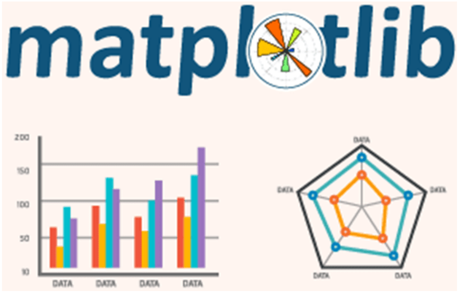
| Feature | Matplotlib | pandas |
|---|---|---|
| Control | High (detailed customization) | Low to moderate (basic customization) |
| Ease of Use | More verbose and complex syntax | Simple and direct |
| Flexibility | Excellent for complex and multi-faceted plots | Good for straightforward plots |
| Customizable | Extensive, with access to every plot element | Limited to plot options in DataFrame |

Table: Comparison of `Matplotlib` and `pandas` for Data Visualization

## 4.8 When to Use which?

- Use Matplotlib when you need precise control and complex customization for your plots.
- Use pandas for fast, simple plots when working directly with DataFrames, especially during initial data analysis.

## Getting Started with Data Visualizations.
## 5. Matplotlib for Data Visualizations - A Revision.



## 5.1 Plotting Your First Figure

- **Style:1**

```python
import matplotlib.pyplot as plt
days = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
steps_walked = [8934,14902,3409,25672,12300,2023,6890]
plt.plot(steps_walked)

```

- **Style:2**

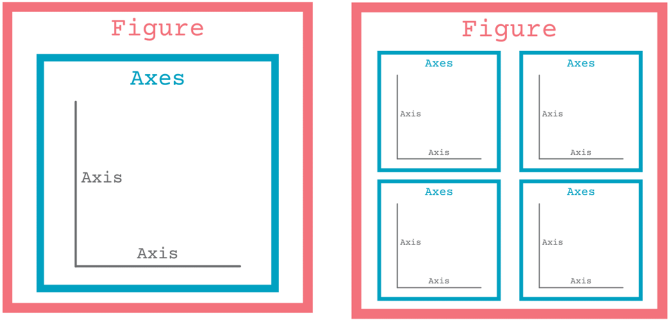
```python
import matplotlib.pyplot as plt
days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
steps_walked = [8934, 14902, 3409, 25672, 12300, 2023,
    6890]
plt.plot(days, steps_walked)
plt.show()

```

Observe the difference between above two outputs.

## 5.2 Anatomy of Matplotlib Figure.

- When working with data visualisation in Python, you'll want to have control over all aspects of your figure.
- In this section, you'll learn about the main components that make up a figure in Matplotlib.

Figure: **Components of Matplotlib Figure**



## 5.2.1 Anatomy of Matplotlib Figure.

Everything in Python is an object, and therefore, so is a Matplotlib figure. In fact, a Matplotlib figure is made up of several objects of different data types.
There are three main parts to a Matplotlib figure:

- **Figure:** This is the whole region of space that's created when you create any figure. The Figure object is the overall object that contains everything else.
- **Axes:** An Axes object is the object that contains the x-axis and y-axis for a 2D plot. Each Axes object corresponds to a plot or a graph. You can have more than one Axes object in a Figure, as you'll see later on in this Chapter.
- **Axis:** An Axis object contains one of the axes, the x-axis or the y-axis for a 2D plot.

## 5.3 Customising the plots

- **Add a custom marker.**

```python
import matplotlib.pyplot as plt
days = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
steps_walked = [8934,14902,3409,25672,12300,2023,6890]
plt.plot(days, steps_walked, "o")
plt.show()

```

The third argument in plot() now indicates what marker you'd like to use. The string "o" represents filled circles.

`Cautions:Please consult matplotlib documentation for updated version and type of marker available.`

## 5.3.1 Customising the plots:Titles and labels

- **Adding titles,labels and legends.**

```python
import matplotlib.pyplot as plt
days = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
steps_walked = [8934,14902,3409,25672,12300,2023,6890]
steps_last_week = [9788,8710,5308,17630,21309,4002,5223]
plt.plot(days, steps_walked, "o-g")
plt.plot(days, steps_last_week, "v--m")
plt.title("Step count | This week and last week")
plt.xlabel("Days of the week")
plt.ylabel("Steps walked")
plt.grid(True)
plt.legend(["This week", "Last week"])
plt.show()

```

Observe the output.

## 5.3.2 Creating Subplots

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Observe and find what are the arguments for `plt.subplot()`.

**Towards Worksheet - 3.**
**Thank You.**