5CS037 - Concepts and Technologies of AI.
Exploratory Data Analysis - Part -II.
Advance operations with Pandas and Review of Matplotlib.

Prepared By: Siman Giri {Module Leader - 5CS037}

November 27, 2024

1 Instructions

{Disclaimer:Exploratory Data Analysis is designed to be two part exercise and this is Part 2, which mostly focuses on advance operation with pandas for sorting and subsetting of data and applying group-by method on data for exploration and analysis of data.This worksheet also provides a reference on Matplotlib to revise your understanding from Level 4 Computational Mathematics Course.}

This worksheet contains programming exercises on Data cleaning and Data Transformation with pandas based on the material discussed from the slides.This is not a graded exercise and submission are optional but highly recommended as it will be the base of your first assignment.
Please answer the questions below using python in the Jupyter Notebook and follow the guidelines below:

- This worksheet must be completed individually.
- All the solutions must be written in Jupyter Notebook.
- Our Recommendation - Google Colaboratory.
- Dataset used for this session can be downloaded from shared drive.

Figure 1: Pandas.

2 Advance Operations with Pandas.

This Section contains all the sample code from the slides and are here for your reference, you are highly recommended to run all the code with some of the input changed in order to understand the meaning of the operations and also to be able to solve all the exercises from further sections.

- **Cautions!!!:**
  - This Guide may not contain sample output, as we expect you to re-write the code and observe the output.
  - If found: any error or bugs, please report to your instructor and Module leader.
    {Will hugely appreciate your effort.}

2.1 Sorting and Subsetting:

1. Sorting:

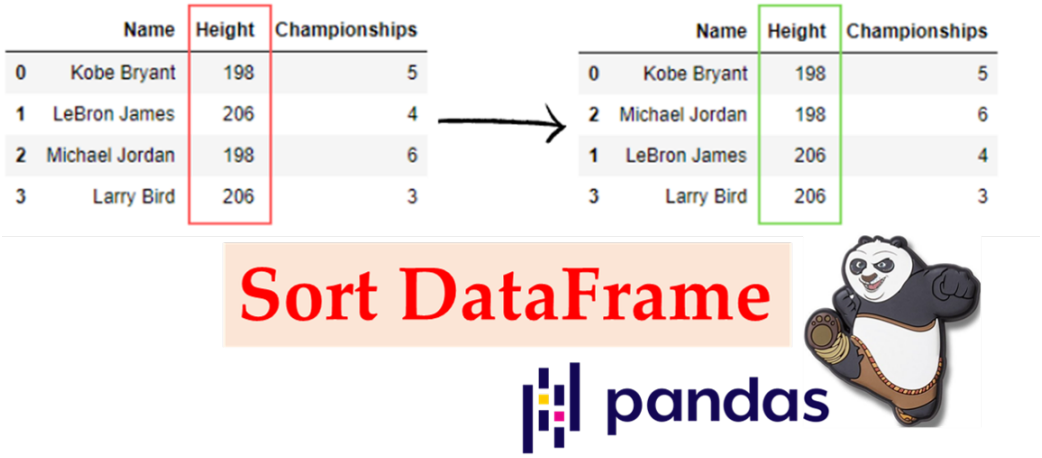

Figure 2: Sorting of Dataframe

Sample Code from Slide - 6 - Sorting Example.

```
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())
```

2. Subsetting - Indices:

Sample Code from Slide - 7 - 8 - Subsetting by indices.

```
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)
```

3. Subsetting by Values - Columns:

Sample Code from Slide - 9 - Subsetting by Columns.

```
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)
```

4. Subsetting By Rows - {aka Filtering}:

Sample Code from Slide - 10 - Subsetting by Row.

```
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)
```

5. Filtering on Categorical Values:

Sample Code from Slide - 11 - 12 - Filtering on Categorical Values.

```
#Transforming in-built data structures-DataFrame
#Style-1
import pandas as pd
pd.DataFrame({'Bob': ['I liked it.','It was awful'], 'Sue': ['Pretty good.', 'Bland.']})
#Style-2
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']},
             index=['Product A', 'Product B'])
```

2.2 The Group-By Method:

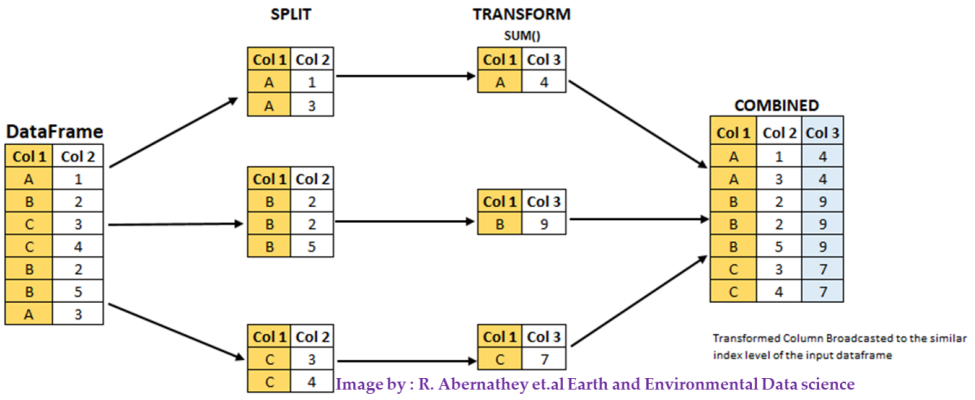

Figure 3: Group By: Split - Apply - Combined.

1. Split - Apply - Combined.

Sample Code from Slide - 16 - Split - Apply{Aggregation} - Combined.

```
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)
```

Sample Code from Slide - 17 - Split - Apply{Transformation} - Combined.

```
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)
```

Sample Code from Slide - 18 - Split - Apply{Filtration} - Combined.

```
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)
```

2.3 Summary - Some Advance Operations with Pandas:

Method/Function	Syntax
sort_values()	df.sort_values(by='column_name', ascending=True)
sort_index()	df.sort_index()
head()	df.head(n)
tail()	df.tail(n)
iloc[]	df.iloc[rows, columns]
loc[]	df.loc[condition]
[] (brackets)	df['column_name']
df[df['column_name'] > value]	
groupby()	df.groupby('column_name')
sum()	df.groupby('column_name')['value_column'].sum()
mean()	df.groupby('column_name')['value_column'].mean()
count()	df.groupby('column_name')['value_column'].count()
transform()	df.groupby('column_name')['value_column'].transform(lambda x: x - x.mean())
apply()	df.groupby('column_name').apply(custom_function)
filter()	df.groupby('column_name').filter(lambda x: x['value_column'].sum() > 100)
agg()	df.groupby('column_name').agg('value_column': ['sum', 'mean'])
size()	df.groupby('column_name').size()
isin()	df[df['column_name'].isin([value1, value2])]
pd.cut()	df['new_column'] = pd.cut(df['column_name'], bins, labels)

Table 1: Summary of Common Data Manipulation Methods and Functions in Pandas

Figure 4: Summary Table.

2.4 Data Visualization with Pandas:

1. Line Plot - Barchart - Histogram - Scatter - Boxplot:

Sample Code from Slide - 27 to 31 - Various Plots.

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

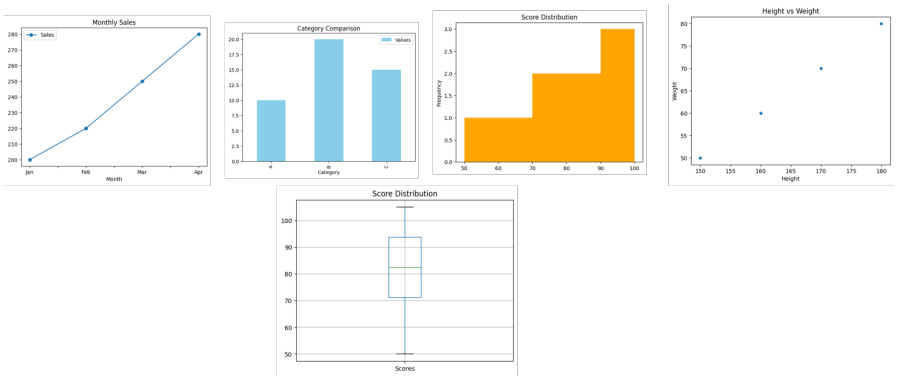

Figure 5: Sample Outputs - Line Chart - Barchart - Histogram - Scatter - Boxplot

2.5 Data Visualization with Matplotlib:

1. Plotting Your First Figure

- **Style:1**

Sample Code from Slide - 35 - Plotting Your First Figure.

```
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**

Sample Code from Slide - 35 - Plotting Your First Figure.

```
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.

2. Anatomy of Matplotlib Figure

When working with data visualization 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. Everything in

Figure 6: Components of Matplotlib Figure

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.

Customizing the plots

3. Add a custom marker

Sample Code from Slide - 38 - Add a Custom Marker.

```
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()
```

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

4. Adding titles,labels and legends.

Sample Code from Slide - 39 - Adding titles - -

```
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. Creating a Subplots

Sample Code from Slide - 40 - Creating Subplot.

```
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()`.

3 To - Do - Task

Please Complete all the problem listed below.

3.1 Warm Up Exercises:

1. Sorting and Subsetting:

Complete all following Task:

- Dataset for the Task: `"titanic.csv"`

Following task is common for all the problem:

1. Load the provided dataset and import in pandas DataFrame.
2. Check info of the DataFrame and identify following:

Problem 1 - Sorting:

1. Create a DataFrame called `fare` that contains only the Fare column of the Titanic dataset. Print the head of the result.
2. Create a DataFrame called `class_age` that contains only the `Pclass and Age columns` of the Titanic dataset, in that order. Print the head of the result.
3. Create a DataFrame called `survived_gender` that contains the `Survived and Sex columns of the Titanic dataset`, in that order.Print the head of the result.

Problem - 2 - Subsetting:

Complete all the following Task:

Subsetting Rows:

1. Filter the Titanic dataset for cases where `the passenger's fare is greater than 100`, assigning it to `fare_gt_100`.View the printed result.
2. Filter the Titanic dataset for cases where `the passenger's class (Pclass) is 1`, assigning it to `first_class`.View the printed result.
3. Filter the Titanic dataset for cases where `the passenger's age is less than 18 and the passenger is female (Sex is "female")`, assigning it to `female_under_18`. View the printed result.

Subsetting Rows by Categorical variables:

1. Filter the Titanic dataset for passengers whose `Embarked port is either "C" (Cherbourg) or "S" (Southampton),` assigning the result to `embarked_c_or_s`. View the printed result.
2. Filter the Titanic dataset for passengers whose `Pclass is in the list [1, 2] (indicating first or second class),` assigning the result to `first_second_class`.View the printed result.

3.2 Exploratory Data Analysis Practice Exercise - 1.

Warning: Handle missing values in the Age column by filling them with the median age of the dataset before performing the division.)

Answer the following questions from Dataset:

Which passenger had the highest fare paid relative to their age?

To answer the question perform following operations:

1. Add a column to the Titanic dataset, `fare_per_year`, containing `the fare divided by the age of the passenger`(**i.e.**, **Fare**/**Age**).
2. Subset rows where `fare_per_year` is higher than 5, assigning this to `high_fare_age`.
3. Sort `high_fare_age` by descending `fare_per_year`, assigning this to `high_fare_age_srt`.
4. Select only the Name and `fare_per_year` columns of `high_fare_age_srt` and save the result as result.
5. Look at the result.

Which adult male passenger (age $\geq$ 18 and Sex is 'male') paid the highest fare relative to their class?

To answer the question perform following operations:

1. Add a column to the Titanic dataset, `fare_per_class`, containing the `fare divided by the passenger class` i.e. `Fare / Pclass`.
2. `Subset rows` where the passenger is male (Sex is "male") and an adult (Age is greater than or equal to 18), assigning this to `adult_males`.
3. `Sort adult_males` by descending `fare_per_class`, assigning this to `adult_males_srt`.
4. `Select only the Name, Age, and fare_per_class` columns of `adult_males_sr` and save the result as result.
5. Look at the result.

3.3 Exploratory Data Analysis with Group-by Method Practice Exercise:

Based on the dataset Answer the following question:

What percent of the total fare revenue came from each passenger class?

To answer the question perform following operation:

1. Calculate the `total Fare` paid across all passengers in the Titanic dataset.
2. `Subset` for passengers in first class `(Pclass is 1)` and calculate their total fare.
3. Do the same for second class `(Pclass is 2)` and third class `(Pclass is 3)`.
4. `Combine` the fare totals from first, second, and third classes into a list.
5. Divide the totals for each class by the overall total fare to get the proportion of fare revenue by class.

Based on the dataset Answer the following question:

What percent of the total number of passengers on the Titanic belonged to each age group (e.g., child, adult, senior)?

To answer the question perform following operation:

1. Create a new column, `age_group`, that categorizes passengers into `"child" (age < 18)`, `"adult" (age 18{64)`, and `"senior" (age 65 and above)`.
2. Calculate the total number of passengers on the Titanic.
3. Count the number of passengers in each age group.
4. Divide the count of each age group by the total number of passengers to get the proportion of passengers in each age group.
5. Display the proportion as a percentage.

The - End -