MAPÚA UNIVERSITY
SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 7:
Web Scraping & Data Analysis

CPE106L (Software Design Laboratory)

Brian Matthew E. Clemente

Maria Angelica Acantilado

Chalzea Fransen C. Dytianquin

Vance David G. Samia

Group No.: **4**
Section: **FOPI01**

PreLab

Readings, Insights, and Reflection

METIS Book: Lambert, K. A. (2023). *Fundamentals Of Python: First Programs* (3rd ed.). Cengage Learning US. https://bookshelf.vitalsource.com/books/9780357881132
The group has developed a deeper understanding of the fundamental principles of data analysis and its practical applications after engaging with Chapter 11. This chapter effectively illustrates the significance of data analysis across various industries, including finance, healthcare, and entertainment, emphasizing its role in informed decision-making. Key concepts such as big data, machine learning, and data mining were introduced, providing a foundational perspective on the broader field of data science. Furthermore, the chapter's focus on working with real-world data, particularly CSV files, underscored the necessity of data cleaning and preparation, as raw datasets often contain inconsistencies and missing values. The introduction to Python's pandas module was particularly beneficial, as it offered practical tools for structuring and manipulating data efficiently. Concepts such as data frames and series were explored, further enhancing our ability to organize and analyze tabular data effectively. The insights gained from this chapter align with the principles outlined in *Fundamentals of Python: First Programs* by Lambert (2023), reinforcing the importance of structured data processing in computational problem-solving. Overall, this chapter has strengthened the group's comprehension of data analysis techniques and their significance.

Getting started — Matplotlib 3.8.0 documentation and Beautiful Soup: Build a Web Scraper With Python – Real Python
The group has developed a structured understanding of web scraping and data visualization, recognizing their essential roles in data analysis. Web scraping is a key technique for extracting large datasets from online sources, allowing for systematic data collection that supports further analysis. However, the process of data extraction is incomplete without proper cleaning and organization, as raw data often contains inconsistencies and missing values. The discussion emphasized the importance of using Python's pandas module to structure scraped data into formats suitable for analysis. Furthermore, the exploration of Matplotlib highlighted its significance in transforming numerical data into meaningful visual representations. By utilizing charts, histograms, line graphs, and scatter plots, analysts can identify trends, correlations, and patterns that may not be immediately apparent in raw data. The introduction to data frames and series provided further insight into efficient data management, reinforcing the necessity of structured storage and organization for large datasets. Overall, this discussion has provided the group with a framework for effectively collecting, processing, and visualizing data to support data-driven decision-making.

METIS #1

Clemente (Chapter 11)

Chapter 11 of *Fundamentals of Python: First Programs* explores essential concepts in data analysis, emphasizing techniques for discovering patterns and relationships within datasets. It highlights the importance of statistical functions such as mean, median, mode, and standard deviation in summarizing data trends. Additionally, the chapter underscores the significance of data visualization, demonstrating how graphical representations enhance interpretability.
Practical applications are also addressed, with CSV files recognized as a widely used format for data storage and exchange across analytical tools. The chapter introduces data frames and series, foundational structures for managing and manipulating datasets efficiently. These elements collectively support a structured approach to data analysis, enabling informed decision-making through both computational and visual insights.

Acantilado (Chapter 11)

Chapter 11 highlights the widespread role of data analysis in today's world, illustrating how it supports decision-making in fields ranging from financial fraud detection to personalized recommendations on streaming platforms. By framing data analysis as a valuable, real-world skill rather than just an academic subject, the chapter underscores its broad applicability across multiple industries. It introduces key concepts such as data sets, big data, data mining, and machine learning, providing a foundational understanding for those interested in exploring these topics further. A particularly important section discusses handling real-world data, focusing on CSV files and the challenges of data cleaning, which is a crucial step in ensuring accurate and meaningful analysis. The chapter emphasizes the necessity of working with imperfect data and introduces Python's pandas module as a tool for cleaning, processing, and organizing information efficiently. Concepts like data frames and series are also explained, offering insight into structuring and manipulating tabular data effectively for analysis. While the examples remain straightforward, they provide a clear and practical introduction to essential techniques in preparing data for accurate interpretation.

Dytianquin (Chapter 11)

Chapter 11 emphasizes the importance of data analysis in making informed decisions across various fields. It introduces fundamental concepts such as data sets, data mining, and machine learning, highlighting their role in identifying trends and patterns. A key takeaway from the chapter is the necessity of working with real-world data, particularly CSV files, which are commonly used for storing structured information. The discussion on data cleaning is particularly relevant, as raw data often contains missing values or inconsistencies that can affect the accuracy of analysis. The introduction to Python's pandas module provides practical tools for handling data efficiently. Concepts like data frames and series help in structuring tabular data, making it easier to filter, sort, and analyze. For example, in the financial sector, data analysts use pandas to process stock market data, identifying fluctuations and trends that influence investment decisions. Similarly, in healthcare, patient records stored in CSV files can be cleaned and analyzed to track disease outbreaks or evaluate treatment effectiveness. By exploring these applications, the chapter reinforces the real-world significance of data analysis techniques.

Samia (Chapter 11)
The chapter opens by highlighting how common data analysis is in contemporary life. Data analysis is essential to decision-making in a variety of sectors, from banking fraud detection to entertainment platform recommendation algorithms. This contextualization aids readers in understanding why mastering data analysis may be considered a useful tool with a variety of applications rather than merely being an academic exercise. In addition to giving a general introduction to the field, the chapter covers important words including data sets, big data, data mining, and machine learning. Although these ideas are simply mentioned in passing, they provide readers who might want to delve more into these subjects in the future with a helpful introduction.

The chapter's explanation of working with real-world data sets, especially those saved in CSV files, is one of its most notable features. The chapter introduces readers to the process of data cleaning using Python's pandas module, acknowledging that real-world data is frequently dirty, with missing or inaccurate values. Since raw data rarely comes in a format that is ready for analysis, this is an essential ability for anyone working in data science. Data frames and series, two crucial structures for arranging and working with tabular data, are also introduced in this chapter. Even though the examples are simple, they do a good job of showing the procedures needed to get data ready for analysis.

METIS #2

Clemente (Web Scraping & Matplotlib)
Web scraping plays a crucial role in data collection, enabling the extraction of valuable information from websites for analysis and decision-making. This process provides raw data that can be structured and analyzed to uncover patterns and relationships. Once collected, data analysis techniques, such as statistical measures including mean, median, mode, and standard deviation, help summarize key insights. Visualization methods, such as charts and graphical plots, further enhance the interpretability of trends within the dataset.

The use of Matplotlib is particularly beneficial in this context, as it allows for the creation of various visual representations, such as histograms, line graphs, and scatter plots. By leveraging Matplotlib, analysts can transform raw data into meaningful visuals that highlight trends and correlations, making complex datasets more comprehensible. Additionally, data organization is essential for effective processing, with structures like data frames and series offering a systematic approach to managing large datasets. CSV files serve as a practical format for storing and sharing scraped data across various analytical applications.

Acantilado (Web Scraping & Matplotlib)
Web scraping has become an indispensable tool for gathering data from websites, providing analysts with raw information that can be refined and examined to extract meaningful insights. This method enables the large-scale collection of data, which, once processed, can reveal trends, correlations, and patterns critical for informed decision-making. After scraping, statistical techniques such as mean, median, mode, and standard deviation are employed to summarize and interpret the data, offering a clearer snapshot of its characteristics. Beyond numerical analysis, visualization plays a key role in making complex datasets more digestible, allowing trends to be spotted with ease through graphical representations.

On the other hand, Matplotlib, a powerful library for data visualization, is especially useful for transforming raw figures into comprehensible charts, including histograms, line graphs, and scatter plots. These visual tools enhance data interpretation, making it easier to identify relationships that may not be immediately apparent in raw numbers. Additionally, efficient data management is crucial, with structures like data frames and series offering a systematic approach to handling large volumes of information. CSV files, widely used for data storage, provide a convenient format for organizing and sharing scraped content across platforms. When combined, these techniques create a streamlined process for collecting, processing, and analyzing web-based data, allowing analysts to derive insights with precision.

Dytianquin (Web Scraping & Matplotlib)
Web scraping is a powerful tool for automating data collection from websites, allowing analysts to gather large datasets efficiently. This technique is especially useful for tracking price changes in e-commerce, monitoring social media trends, or compiling financial reports from multiple sources. However, scraped data is often unstructured, requiring cleaning and organization before meaningful analysis can take place. The pandas module in Python is essential for converting this unstructured data into data frames and other structured representations that make processing easier.

Data visualization is equally important in making sense of large datasets. Matplotlib enables users to create charts, histograms, and scatter plots that reveal patterns and relationships. For instance, in sports analytics, player performance metrics collected from different games can be visualized using line graphs to track improvements over time. In climate science, historical temperature data can be plotted to identify long-term trends in global warming. These examples highlight how web scraping and data visualization work together to extract, process, and present data in ways that drive informed decision-making.

Samia (Web Scraping & Matplotlib)
Web scraping has emerged as a crucial method for automated data extraction, enabling users to effectively gather data for analysis from web pages. When structured data is retrieved, this procedure is very helpful since it allows for the extraction of important insights through statistical analysis using metrics like mean, median, and standard deviation. Web scraping is an

effective tool for researchers, companies, and developers since it can automate data collecting, saving time and effort compared to manual collection.

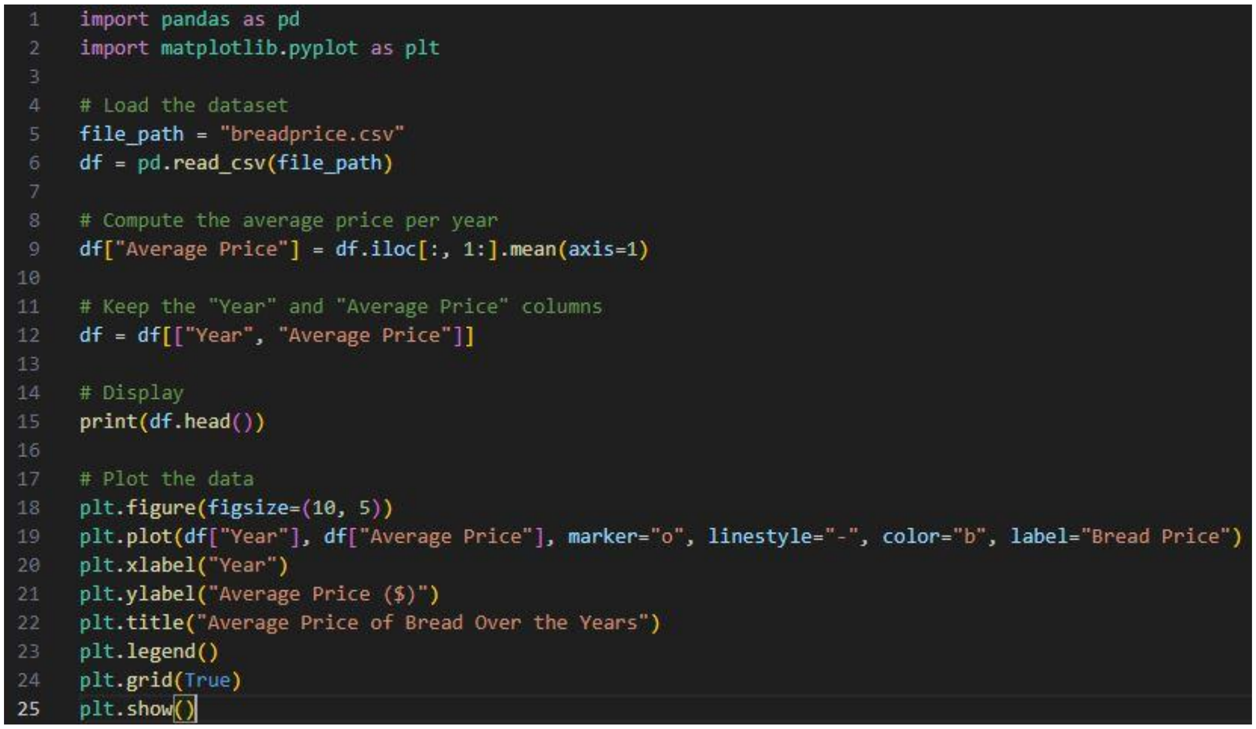
Visualization is essential for deciphering trends and patterns after data has been gathered. A popular Python package called Matplotlib makes it possible to create a variety of graphical representations, such as scatter plots, line graphs, and histograms. Large datasets can be easier to grasp thanks to these visualizations, which can help analysts find relationships and make better data-driven decisions. A uniform format for storing and sharing scraped data across many platforms is offered by CSV files, while proper data organization utilizing structures like data frames improves processing performance. Users may turn raw online data into insights by combining web scraping with data visualization tools like Matplotlib, showcasing the value of automation in data-driven decision-making.

PostLab

Programming Problems

This post-lab activity focuses on the implementation of web scraping techniques and fundamental data analysis concepts, reinforcing key principles in data collection, organization, and visualization. The tasks involve extracting structured and unstructured data using automated scripts, followed by processing and analyzing the retrieved information. Additionally, essential data analysis functions, such as mean, median, mode, and standard deviation, are applied to identify patterns and relationships within the dataset. To enhance data interpretation, Matplotlib is utilized to generate visual representations, such as line graphs, bar charts, and scatter plots, allowing for clearer insights into trends and distributions. Each programming task was assigned to different group members, with contributions documented within the code files to ensure clarity and organization. The development process emphasized structured data handling techniques, ensuring the accuracy of insights derived from the collected data. Version control tools were used to track modifications, facilitating seamless collaboration and data integrity.

Programming Exercise #5



```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = "breadprice.csv"
df = pd.read_csv(file_path)

# Compute the average price per year
df["Average Price"] = df.iloc[:, 1:].mean(axis=1)

# Keep the "Year" and "Average Price" columns
df = df[["Year", "Average Price"]]

# Display
print(df.head())

# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df["Year"], df["Average Price"], marker="o", linestyle="-", color="b", label="Bread Price")
plt.xlabel("Year")
plt.ylabel("Average Price ($)")
plt.title("Average Price of Bread Over the Years")
plt.legend()
plt.grid(True)
plt.show()
```

Figure 1.1 Screenshot of breadprice.py Source Code

Figure 1.1 showcases the Python script that processes and visualizes bread price data. A CSV file with monthly bread prices for each year is read by the script, which then computes the average price annually and creates a line plot to display price patterns over time.

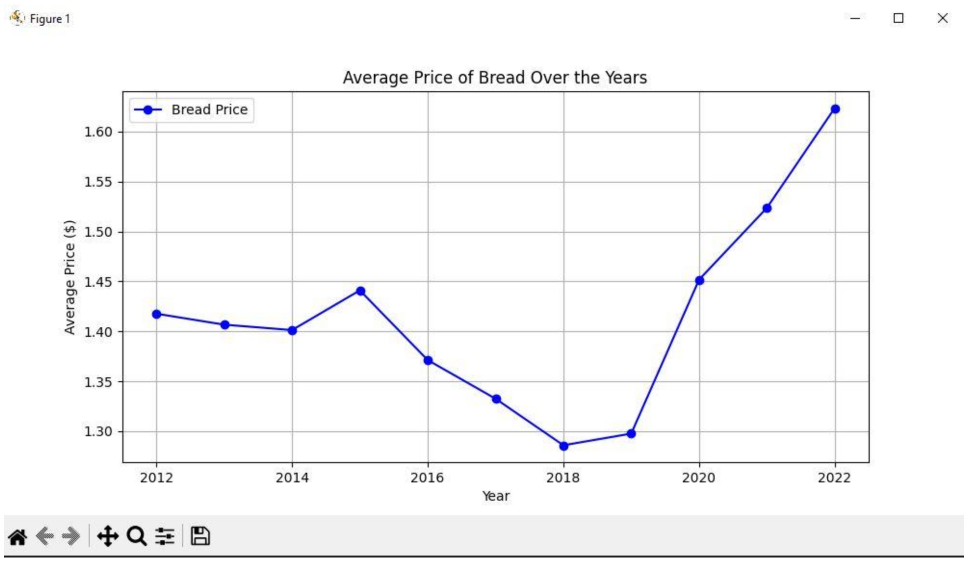
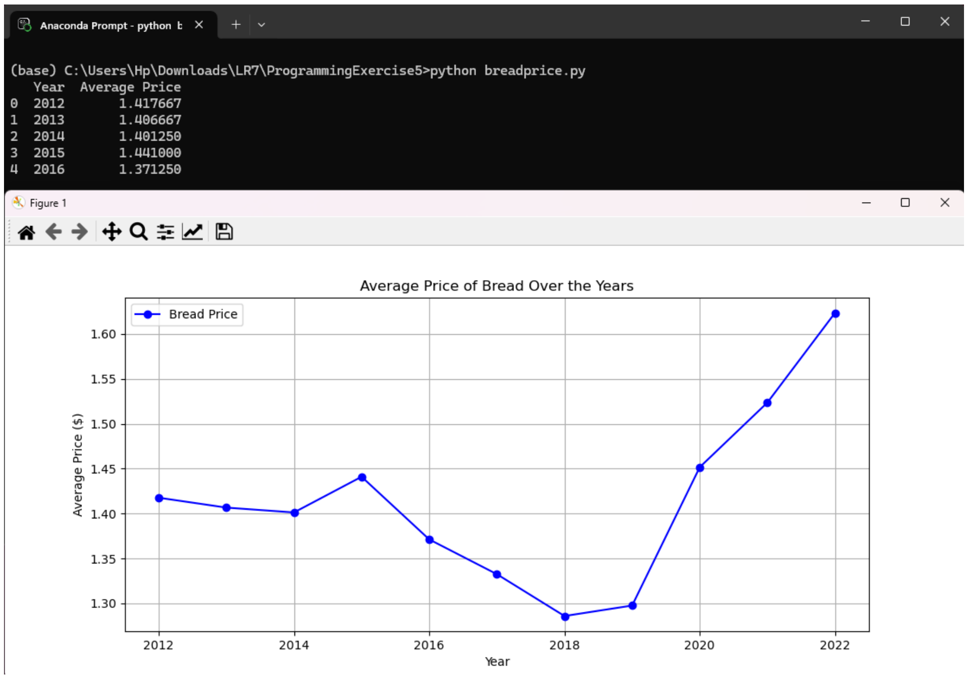


Figure 1.2 Matplotlib Graph Illustrated after the breadprice.py File is Executed

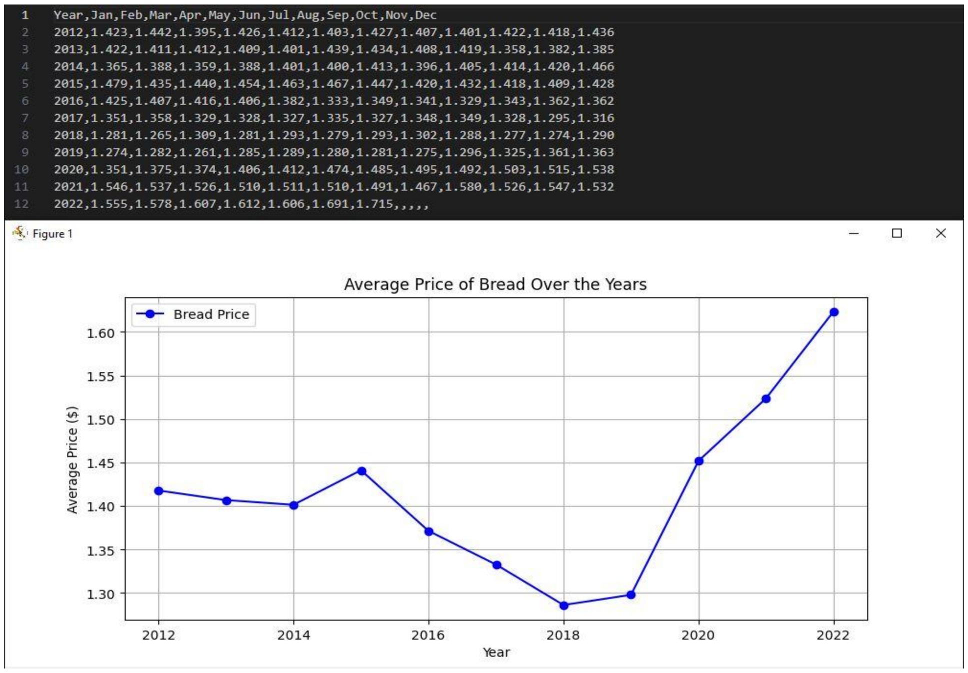
Figure 1.2 visualizes the data on bread prices through the use of Matplotlib. The script creates a line plot to display pricing trends over time after reading a CSV file with monthly bread prices for each year and calculating the average price annually.



```
(base) C:\Users\Hp\Downloads\LR7\ProgrammingExercise5>python breadprice.py
   Year  Average Price
0  2012       1.417667
1  2013       1.406667
2  2014       1.401250
3  2015       1.441000
4  2016       1.371250
```

Figure 1.3 breadprice.py Executed in Anaconda Prompt

Figure 1.3 displays the execution of `breadprice.py` in the Anaconda environment. After reading the bread price dataset from a CSV file, the application computes the average annual price and outputs the results to the console. Additionally, a graphical representation of the data is produced by the execution, aiding in the visualization of bread price patterns over time. The script loads, processes, and analyzes the dataset appropriately, as confirmed by the successful execution.

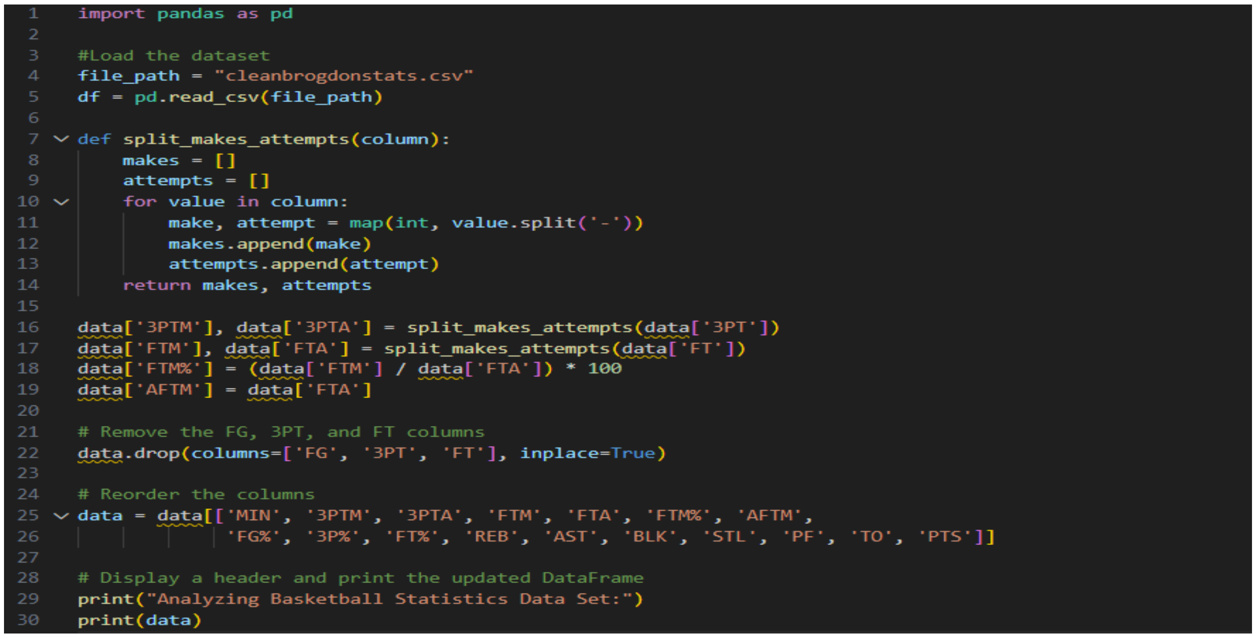


```
Year,Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
2012,1.423,1.442,1.395,1.426,1.412,1.403,1.427,1.407,1.401,1.422,1.418,1.436
2013,1.422,1.411,1.412,1.409,1.401,1.439,1.434,1.408,1.419,1.358,1.382,1.385
2014,1.365,1.388,1.359,1.388,1.401,1.400,1.413,1.396,1.405,1.414,1.420,1.466
2015,1.479,1.435,1.440,1.454,1.463,1.467,1.447,1.420,1.432,1.418,1.409,1.428
2016,1.425,1.407,1.416,1.406,1.382,1.333,1.349,1.341,1.329,1.343,1.362,1.362
2017,1.351,1.358,1.329,1.328,1.327,1.335,1.327,1.348,1.349,1.328,1.295,1.316
2018,1.281,1.265,1.309,1.281,1.293,1.279,1.293,1.302,1.288,1.277,1.274,1.290
2019,1.274,1.282,1.261,1.285,1.289,1.280,1.281,1.275,1.296,1.325,1.361,1.363
2020,1.351,1.375,1.374,1.406,1.412,1.474,1.485,1.495,1.492,1.503,1.515,1.538
2021,1.546,1.537,1.526,1.510,1.511,1.510,1.491,1.467,1.580,1.526,1.547,1.532
2022,1.555,1.578,1.607,1.612,1.606,1.691,1.715,,,,,
```

Figure 1.4 breadprice.py Matlplotlib Graph in Comparison with the Tabular Data from the breadprice.csv File

Figure 1.4 displays the generated line graph and the raw data seen in its corresponding CSV file, `breadprice.csv`. Compared to the tabular data in the CSV file, the graph makes it easier to understand trends by graphically representing the price of bread over time.

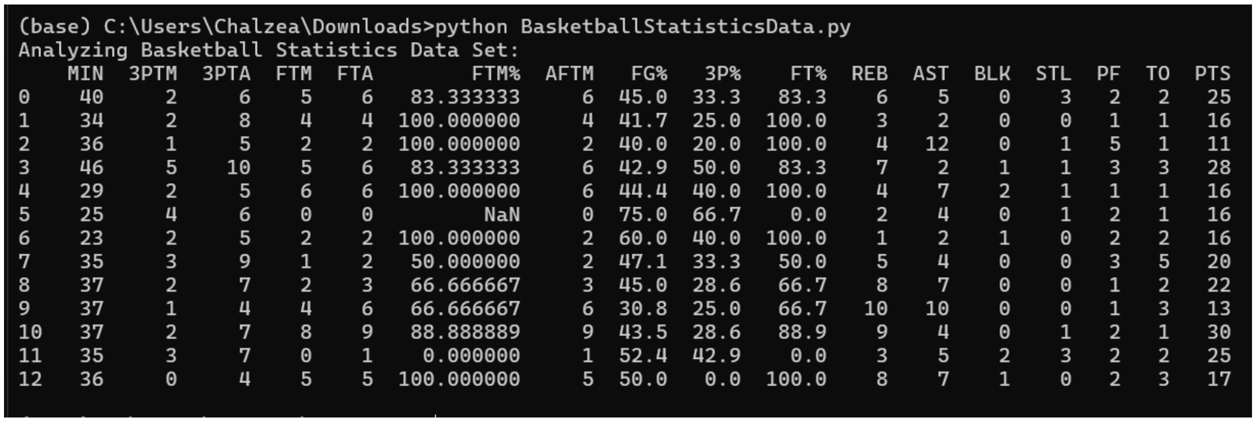
Programming Exercise #6



```python
import pandas as pd

#Load the dataset
file_path = "cleanbrogdonstats.csv"
df = pd.read_csv(file_path)

def split_makes_attempts(column):
    makes = []
    attempts = []
    for value in column:
        make, attempt = map(int, value.split('-'))
        makes.append(make)
        attempts.append(attempt)
    return makes, attempts

data['3PTM'], data['3PTA'] = split_makes_attempts(data['3PT'])
data['FTM'], data['FTA'] = split_makes_attempts(data['FT'])
data['FTM%'] = (data['FTM'] / data['FTA']) * 100
data['AFTM'] = data['FTA']

# Remove the FG, 3PT, and FT columns
data.drop(columns=['FG', '3PT', 'FT'], inplace=True)

# Reorder the columns
data = data[['MIN', '3PTM', '3PTA', 'FTM', 'FTA', 'FTM%', 'AFTM',
             'FG%', '3P%', 'FT%', 'REB', 'AST', 'BLK', 'STL', 'PF', 'TO', 'PTS']]

# Display a header and print the updated DataFrame
print("Analyzing Basketball Statistics Data Set:")
print(data)
```

Figure 2.1 Screenshot of BasketballStatisticsData.py Source Code

Figure 2.1 presents a Python script that reads basketball data from the file `cleanbrogdonstats.csv` using the Pandas library, then cleans and reorganizes it for further analysis. The code removes the original columns, reorders the remaining columns, and divides the combined columns. Likewise, it prints out the cleaned `DataFrame` to provide a quick preview of the updated basketball statistics.

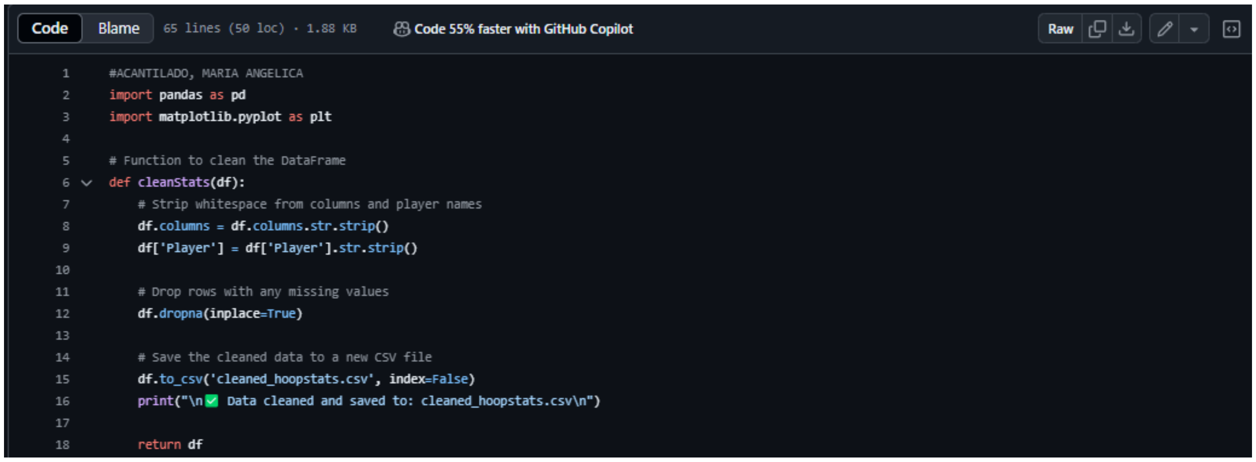


```
(base) C:\Users\Chalzea\Downloads>python BasketballStatisticsData.py
Analyzing Basketball Statistics Data Set:
    MIN  3PTM  3PTA  FTM  FTA        FTM%  AFTM   FG%   3P%    FT%  REB  AST  BLK  STL  PF  TO  PTS
0    40     2     6    5    6   83.333333     6  45.0  33.3   83.3    6    5    0    3   2   2   25
1    34     2     8    4    4  100.000000     4  41.7  25.0  100.0    3    2    0    0   1   1   16
2    36     1     5    2    2  100.000000     2  40.0  20.0  100.0    4   12    0    1   5   1   11
3    46     5    10    5    6   83.333333     6  42.9  50.0   83.3    7    2    1    1   3   3   28
4    29     2     5    6    6  100.000000     6  44.4  40.0  100.0    4    7    2    1   1   1   16
5    25     4     6    0    0         NaN     0  75.0  66.7    0.0    2    4    0    1   2   1   16
6    23     2     5    2    2  100.000000     2  60.0  40.0  100.0    1    2    1    0   2   2   16
7    35     3     9    1    2   50.000000     2  47.1  33.3   50.0    5    4    0    0   3   5   20
8    37     2     7    2    3   66.666667     3  45.0  28.6   66.7    8    7    0    0   1   2   22
9    37     1     4    4    6   66.666667     6  30.8  25.0   66.7   10   10    0    0   1   3   13
10   37     2     7    8    9   88.888889     9  43.5  28.6   88.9    9    4    0    1   2   1   30
11   35     3     7    0    1    0.000000     1  52.4  42.9    0.0    3    5    2    3   2   2   25
12   36     0     4    5    5  100.000000     5  50.0   0.0  100.0    8    7    1    0   2   3   17
```

Figure 2.2 Screenshot of BasketballStatisticsData.py updated tabular data

On the other hand, *Figure 2.2* shows the updated table of basketball statistics after the data has been cleaned and reorganized. It displays each player's performance metrics, including minutes played, successful and attempted three-point shots made, successful and attempted free throws made, successful and attempted field goals made, shooting percentages, rebounds, assists, blocks, steals, personal fouls, turnovers, and total points.

Programming Exercise #7

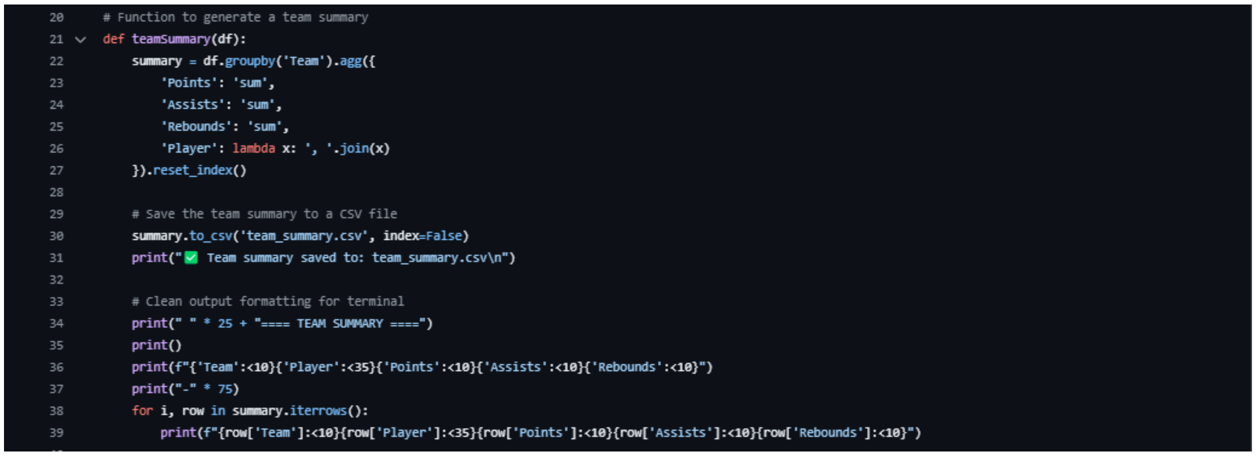


Code Blame 65 lines (50 loc) · 1.88 KB Code 55% faster with GitHub Copilot Raw

```python
#ACANTILADO, MARIA ANGELICA
import pandas as pd
import matplotlib.pyplot as plt

# Function to clean the DataFrame
def cleanStats(df):
    # Strip whitespace from columns and player names
    df.columns = df.columns.str.strip()
    df['Player'] = df['Player'].str.strip()

    # Drop rows with any missing values
    df.dropna(inplace=True)

    # Save the cleaned data to a new CSV file
    df.to_csv('cleaned_hoopstats.csv', index=False)
    print("\n✅ Data cleaned and saved to: cleaned_hoopstats.csv\n")

    return df
```

Figure 3.1 Screenshot of Data Cleaning Function

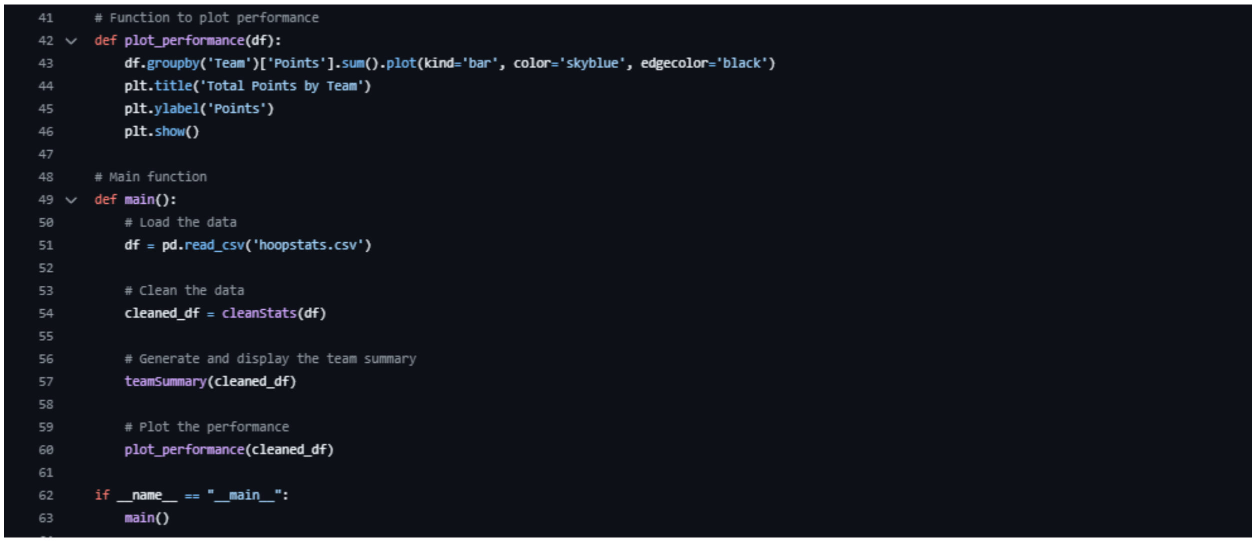
Figure 3.1 shows the Python function `cleanStats()` designed to preprocess and clean the basketball statistics data. The function utilizes the Pandas library to manipulate the data frame by stripping whitespace from column headers and player names to ensure uniformity. Additionally, it handles missing values by removing any rows with incomplete data through the `dropna()` function. The cleaned data is then saved as a new CSV file named `cleaned_hoopstats.csv`, ensuring that the modified data is accessible for further analysis.



```python
# Function to generate a team summary
def teamSummary(df):
    summary = df.groupby('Team').agg({
        'Points': 'sum',
        'Assists': 'sum',
        'Rebounds': 'sum',
        'Player': lambda x: ', '.join(x)
    }).reset_index()

    # Save the team summary to a CSV file
    summary.to_csv('team_summary.csv', index=False)
    print("✅ Team summary saved to: team_summary.csv\n")

    # Clean output formatting for terminal
    print(" " * 25 + "==== TEAM SUMMARY ====")
    print()
    print(f"{'Team':<10}{'Player':<35}{'Points':<10}{'Assists':<10}{'Rebounds':<10}")
    print("-" * 75)
    for i, row in summary.iterrows():
        print(f"{row['Team']:<10}{row['Player']:<35}{row['Points']:<10}{row['Assists']:<10}{row['Rebounds']:<10}")
```

Figure 3.2 Screenshot of Team Summary and Terminal Output Functions

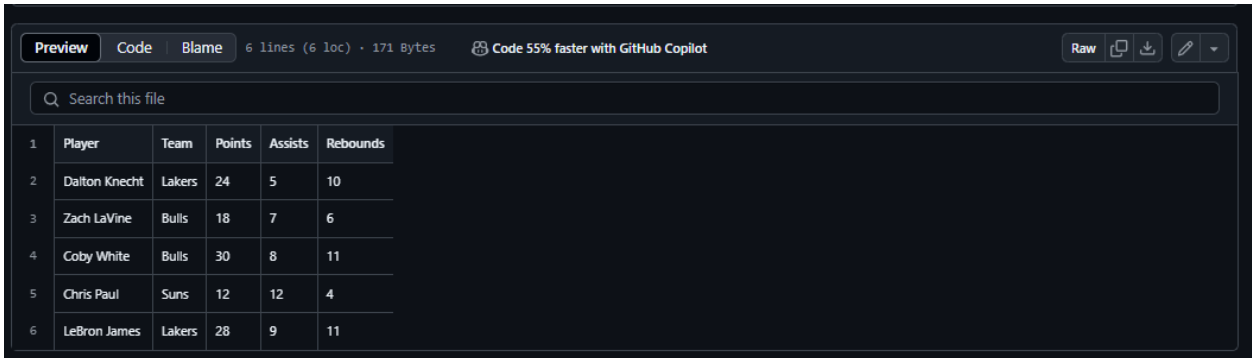
Figure 3.2 presents the Python function `teamSummary()` used to generate a statistical summary of basketball teams based on their total points, assists, and rebounds. Using the `groupby()` method from the Pandas library, the data is aggregated by team, summing up the values for each statistical category. The players from each team are concatenated into a single string separated by commas, ensuring that all contributing players are represented. The resulting summary is saved as a new CSV file named `team_summary.csv` for future reference.



```python
# Function to plot performance
def plot_performance(df):
    df.groupby('Team')['Points'].sum().plot(kind='bar', color='skyblue', edgecolor='black')
    plt.title('Total Points by Team')
    plt.ylabel('Points')
    plt.show()

# Main function
def main():
    # Load the data
    df = pd.read_csv('hoopstats.csv')

    # Clean the data
    cleaned_df = cleanStats(df)

    # Generate and display the team summary
    teamSummary(cleaned_df)

    # Plot the performance
    plot_performance(cleaned_df)

if __name__ == "__main__":
    main()
```

Figure 3.3 Screenshot of Plot Performance and Main Function

Figure 3.3 shows the `plot_performance()` function, which uses Matplotlib's plot() method to create a bar graph that visualizes the total points scored by each team. It also shows the `main()` function responsible for running the basketball statistics application. The `main()` function serves as the entry point of the program, orchestrating the sequence of tasks: loading the dataset from hoopstats.csv, cleaning the data using the `cleanStats()` function, generating a team summary with `teamSummary()`, and visualizing the performance using `plot_performance()`. The condition `if __name__ == "__main__":` ensures that the script runs only when executed directly, maintaining code modularity and reusability.

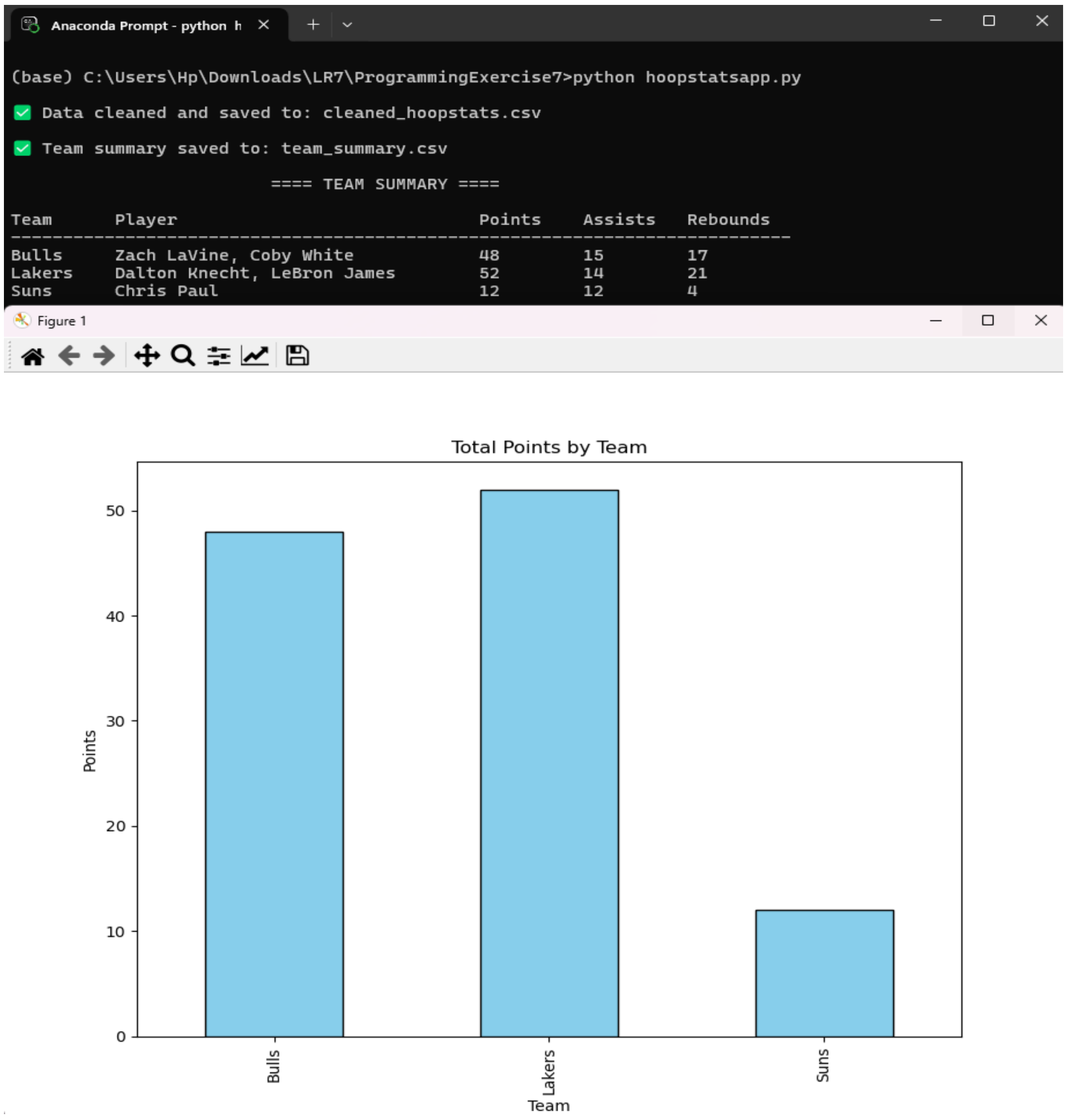


Preview
Code
Blame
6 lines (6 loc) · 171 Bytes
Code 55% faster with GitHub Copilot
Raw
Search this file

Player	Team	Points	Assists	Rebounds
Dalton Knecht	Lakers	24	5	10
Zach LaVine	Bulls	18	7	6
Coby White	Bulls	30	8	11
Chris Paul	Suns	12	12	4
LeBron James	Lakers	28	9	11

Figure 3.4 Screenshot of Raw Data from hoopstats.csv File

Figure 3.4 presents a preview of the raw basketball statistics dataset stored in hoopstats.csv. The dataset consists of five columns: Player, Team, Points, Assists, and Rebounds. It will serve as the input data for the Python program to generate a team summary, clean the data, and visualize team performance.



```
(base) C:\Users\Hp\Downloads\LR7\ProgrammingExercise7>python hoopstatsapp.py
✅ Data cleaned and saved to: cleaned_hoopstats.csv
✅ Team summary saved to: team_summary.csv
                          ==== TEAM SUMMARY ====
Team      Player                             Points    Assists   Rebounds
---------------------------------------------------------------------------
Bulls     Zach LaVine, Coby White            48        15        17
Lakers    Dalton Knecht, LeBron James        52        14        21
Suns      Chris Paul                         12        12        4
```

Figure 3.5 Screenshot of Output in Anaconda Prompt

Lastly, *Figure 3.5* presents the summarized basketball statistics by team, derived from the `team_summary.csv` file. It displays the terminal output and the generated bar graph from executing `hoopstatsapp.py` in Anaconda Prompt. The terminal output confirms that the raw data was cleaned and saved to `cleaned_hoopstats.csv` and the team summary was saved to `team_summary.csv`. The output also visualizes the total points scored by each team using a bar graph that was implemented through Matplotlib.