VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

AssignmentReport
on

Data Visualization

Submitted By

Mohammed Uzair - 1RN21CD027

Under the Guidance of

Ms Vinutha S
Asst. Professor
Department of CSE (Data Science)

RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

2024-2025

Table of Contents

1. Introduction

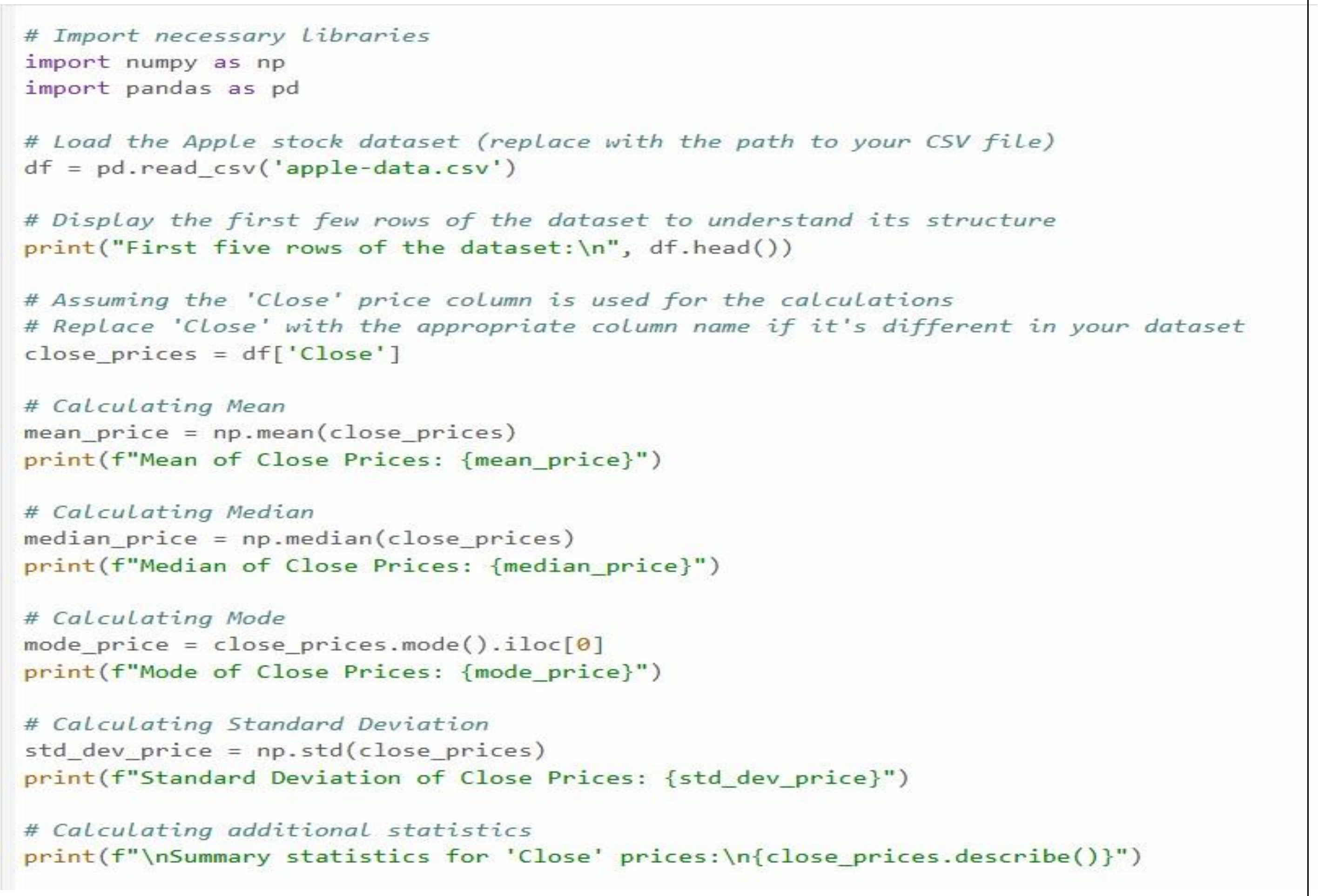
This report presents a comprehensive analysis of various datasets, including Apple stock prices, TikTok video performance metrics, and agricultural crop yield statistics. Leveraging Python libraries such as NumPy, Pandas, Matplotlib, and Seaborn, this report addresses a range of data analysis and visualization tasks, exploring specific aspects of each dataset.

2. Question 1: Statistical Analysis of Apple Stock Data

Objective

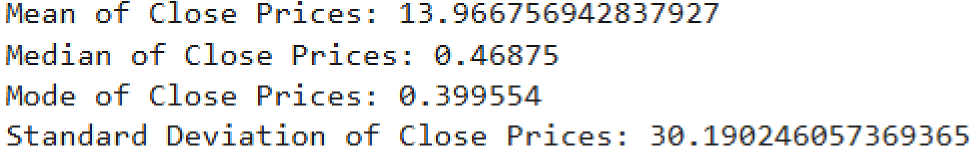
To demonstrate the calculation of mean, median, mode, and standard deviation using Numpy and Pandas with the Apple stock dataset.

Code Snippet:



```python
# Import necessary libraries
import numpy as np
import pandas as pd

# Load the Apple stock dataset (replace with the path to your CSV file)
df = pd.read_csv('apple-data.csv')

# Display the first few rows of the dataset to understand its structure
print("First five rows of the dataset:\n", df.head())

# Assuming the 'Close' price column is used for the calculations
# Replace 'Close' with the appropriate column name if it's different in your dataset
close_prices = df['Close']

# Calculating Mean
mean_price = np.mean(close_prices)
print(f"Mean of Close Prices: {mean_price}")

# Calculating Median
median_price = np.median(close_prices)
print(f"Median of Close Prices: {median_price}")

# Calculating Mode
mode_price = close_prices.mode().iloc[0]
print(f"Mode of Close Prices: {mode_price}")

# Calculating Standard Deviation
std_dev_price = np.std(close_prices)
print(f"Standard Deviation of Close Prices: {std_dev_price}")

# Calculating additional statistics
print(f"\nSummary statistics for 'Close' prices:\n{close_prices.describe()}")
```

Output:



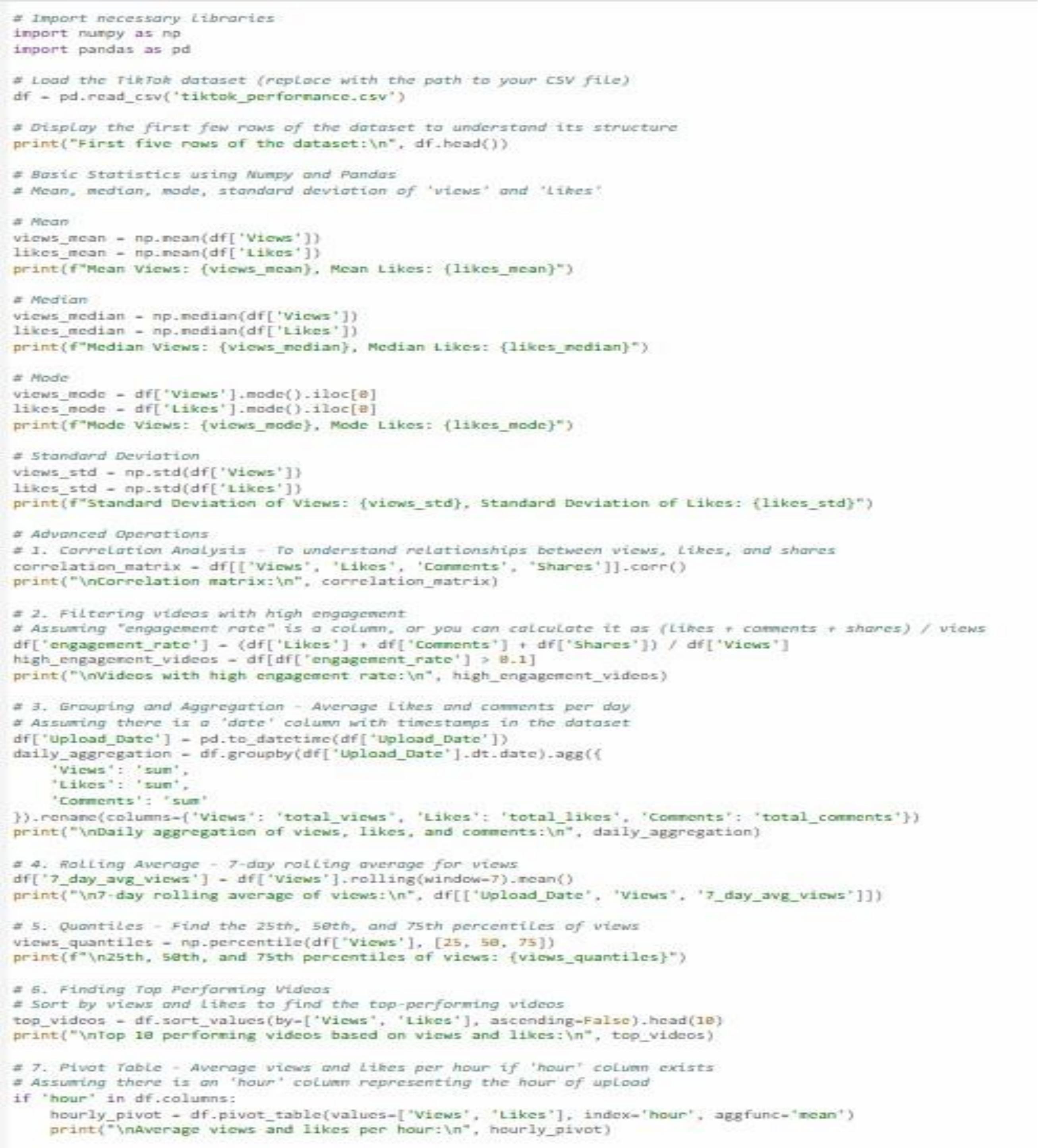
```
Mean of Close Prices: 13.966756942837927
Median of Close Prices: 0.46875
Mode of Close Prices: 0.399554
Standard Deviation of Close Prices: 30.190246057369365
```

3. Question 2: TikTok Video Performance Analysis

Objective:

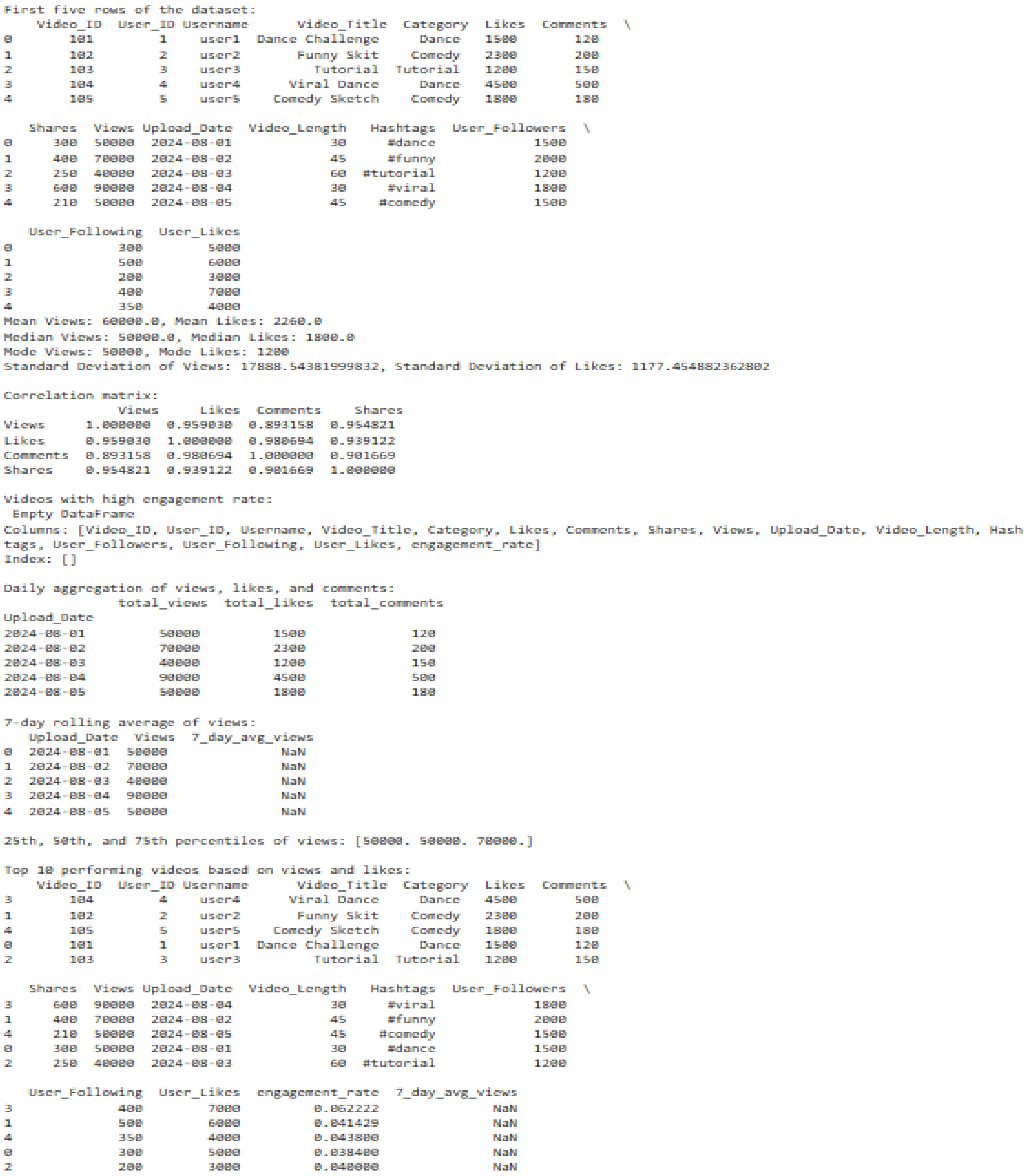
To perform basic to advanced operations using Numpy and Pandas on a TikTok video performance dataset.

Code Snippet:



```
# Import necessary libraries
import numpy as np
import pandas as pd

# Load the TikTok dataset (replace with the path to your CSV file)
df = pd.read_csv('tiktok_performance.csv')

# Display the first few rows of the dataset to understand its structure
print("First five rows of the dataset:\n", df.head())

# Basic Statistics using Numpy and Pandas
# Mean, median, mode, standard deviation of 'views' and 'likes'

# Mean
views_mean = np.mean(df['Views'])
likes_mean = np.mean(df['Likes'])
print(f"Mean Views: {views_mean}, Mean Likes: {likes_mean}")

# Median
views_median = np.median(df['Views'])
likes_median = np.median(df['Likes'])
print(f"Median Views: {views_median}, Median Likes: {likes_median}")

# Mode
views_mode = df['Views'].mode().iloc[0]
likes_mode = df['Likes'].mode().iloc[0]
print(f"Mode Views: {views_mode}, Mode Likes: {likes_mode}")

# Standard Deviation
views_std = np.std(df['Views'])
likes_std = np.std(df['Likes'])
print(f"Standard Deviation of Views: {views_std}, Standard Deviation of Likes: {likes_std}")

# Advanced Operations
# 1. Correlation Analysis - To understand relationships between views, likes, and shares
correlation_matrix = df[['Views', 'Likes', 'Comments', 'Shares']].corr()
print("\nCorrelation matrix:\n", correlation_matrix)

# 2. Filtering videos with high engagement
# Assuming "engagement rate" is a column, or you can calculate it as (likes + comments + shares) / views
df['engagement_rate'] = (df['Likes'] + df['Comments'] + df['Shares']) / df['Views']
high_engagement_videos = df[df['engagement_rate'] > 0.1]
print("\nVideos with high engagement rate:\n", high_engagement_videos)

# 3. Grouping and Aggregation - Average likes and comments per day
# Assuming there is a 'date' column with timestamps in the dataset
df['Upload_Date'] = pd.to_datetime(df['Upload_Date'])
daily_aggregation = df.groupby(df['Upload_Date'].dt.date).agg({
    'Views': 'sum',
    'Likes': 'sum',
    'Comments': 'sum'
}).rename(columns={'Views': 'total_views', 'Likes': 'total_likes', 'Comments': 'total_comments'})
print("\nDaily aggregation of views, likes, and comments:\n", daily_aggregation)

# 4. Rolling Average - 7-day rolling average for views
df['7_day_avg_views'] = df['Views'].rolling(window=7).mean()
print("\n7-day rolling average of views:\n", df[['Upload_Date', 'Views', '7_day_avg_views']])

# 5. Quantiles - Find the 25th, 50th, and 75th percentiles of views
views_quantiles = np.percentile(df['Views'], [25, 50, 75])
print(f"\n25th, 50th, and 75th percentiles of views: {views_quantiles}")

# 6. Finding Top Performing Videos
# Sort by views and likes to find the top-performing videos
top_videos = df.sort_values(by=['Views', 'Likes'], ascending=False).head(10)
print("\nTop 10 performing videos based on views and likes:\n", top_videos)

# 7. Pivot Table - Average views and likes per hour if 'hour' column exists
# Assuming there is an 'hour' column representing the hour of upload
if 'hour' in df.columns:
    hourly_pivot = df.pivot_table(values=['Views', 'Likes'], index='hour', aggfunc='mean')
    print("\nAverage views and likes per hour:\n", hourly_pivot)
```

Output:



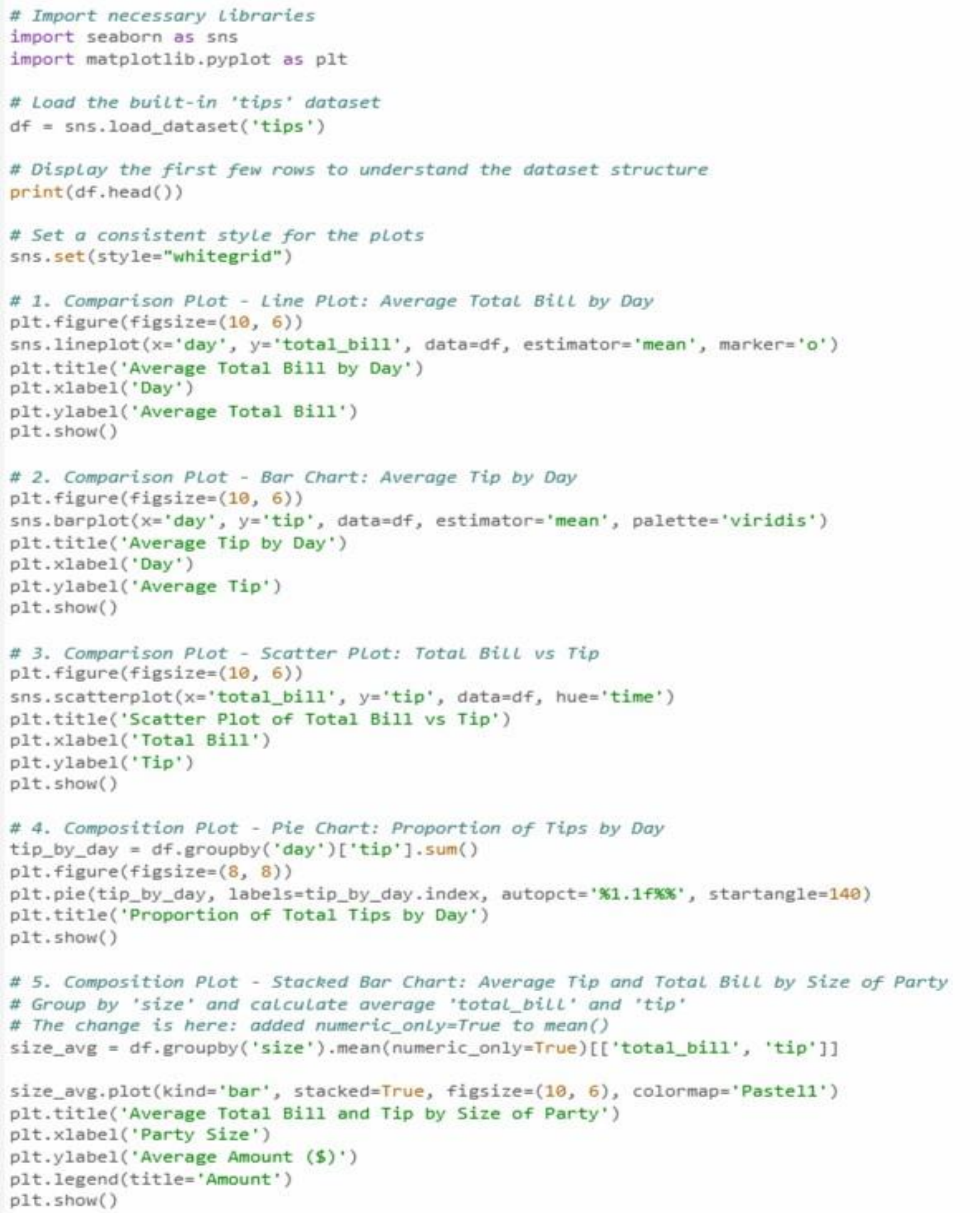
```
First five rows of the dataset:
   Video_ID  User_ID Username      Video_Title  Category  Likes  Comments  \
0       101        1    user1  Dance Challenge     Dance   1500       120
1       102        2    user2       Funny Skit    Comedy   2300       200
2       103        3    user3         Tutorial  Tutorial   1200       150
3       104        4    user4      Viral Dance     Dance   4500       500
4       105        5    user5    Comedy Sketch    Comedy   1800       180

   Shares  Views Upload_Date  Video_Length   Hashtags  User_Followers  \
0     300  50000  2024-08-01            30     #dance            1500
1     400  70000  2024-08-02            45     #funny            2000
2     250  40000  2024-08-03            60  #tutorial            1200
3     600  90000  2024-08-04            30     #viral            1800
4     210  50000  2024-08-05            45    #comedy            1500

   User_Following  User_Likes
0             300        5000
1             500        6000
2             200        3000
3             400        7000
4             350        4000
Mean Views: 60000.0, Mean Likes: 2260.0
Median Views: 50000.0, Median Likes: 1800.0
Mode Views: 50000, Mode Likes: 1200
Standard Deviation of Views: 17888.54381999832, Standard Deviation of Likes: 1177.454882362802

Correlation matrix:
             Views     Likes  Comments    Shares
Views     1.000000  0.959030  0.893158  0.954821
Likes     0.959030  1.000000  0.980694  0.939122
Comments  0.893158  0.980694  1.000000  0.901669
Shares    0.954821  0.939122  0.901669  1.000000

Videos with high engagement rate:
 Empty DataFrame
Columns: [Video_ID, User_ID, Username, Video_Title, Category, Likes, Comments, Shares, Views, Upload_Date, Video_Length, Hash
tags, User_Followers, User_Following, User_Likes, engagement_rate]
Index: []

Daily aggregation of views, likes, and comments:
             total_views  total_likes  total_comments
Upload_Date
2024-08-01         50000         1500             120
2024-08-02         70000         2300             200
2024-08-03         40000         1200             150
2024-08-04         90000         4500             500
2024-08-05         50000         1800             180

7-day rolling average of views:
  Upload_Date  Views  7_day_avg_views
0  2024-08-01  50000              NaN
1  2024-08-02  70000              NaN
2  2024-08-03  40000              NaN
3  2024-08-04  90000              NaN
4  2024-08-05  50000              NaN

25th, 50th, and 75th percentiles of views: [50000. 50000. 70000.]

Top 10 performing videos based on views and likes:
   Video_ID  User_ID Username      Video_Title  Category  Likes  Comments  \
3       104        4    user4      Viral Dance     Dance   4500       500
1       102        2    user2       Funny Skit    Comedy   2300       200
4       105        5    user5    Comedy Sketch    Comedy   1800       180
0       101        1    user1  Dance Challenge     Dance   1500       120
2       103        3    user3         Tutorial  Tutorial   1200       150

   Shares  Views Upload_Date  Video_Length   Hashtags  User_Followers  \
3     600  90000  2024-08-04            30     #viral            1800
1     400  70000  2024-08-02            45     #funny            2000
4     210  50000  2024-08-05            45    #comedy            1500
0     300  50000  2024-08-01            30     #dance            1500
2     250  40000  2024-08-03            60  #tutorial            1200

   User_Following  User_Likes  engagement_rate  7_day_avg_views
3             400        7000         0.062222              NaN
1             500        6000         0.041429              NaN
4             350        4000         0.043800              NaN
0             300        5000         0.038400              NaN
2             200        3000         0.040000              NaN
```

4. Question 3: Comparison and Composition Plots

Objective:

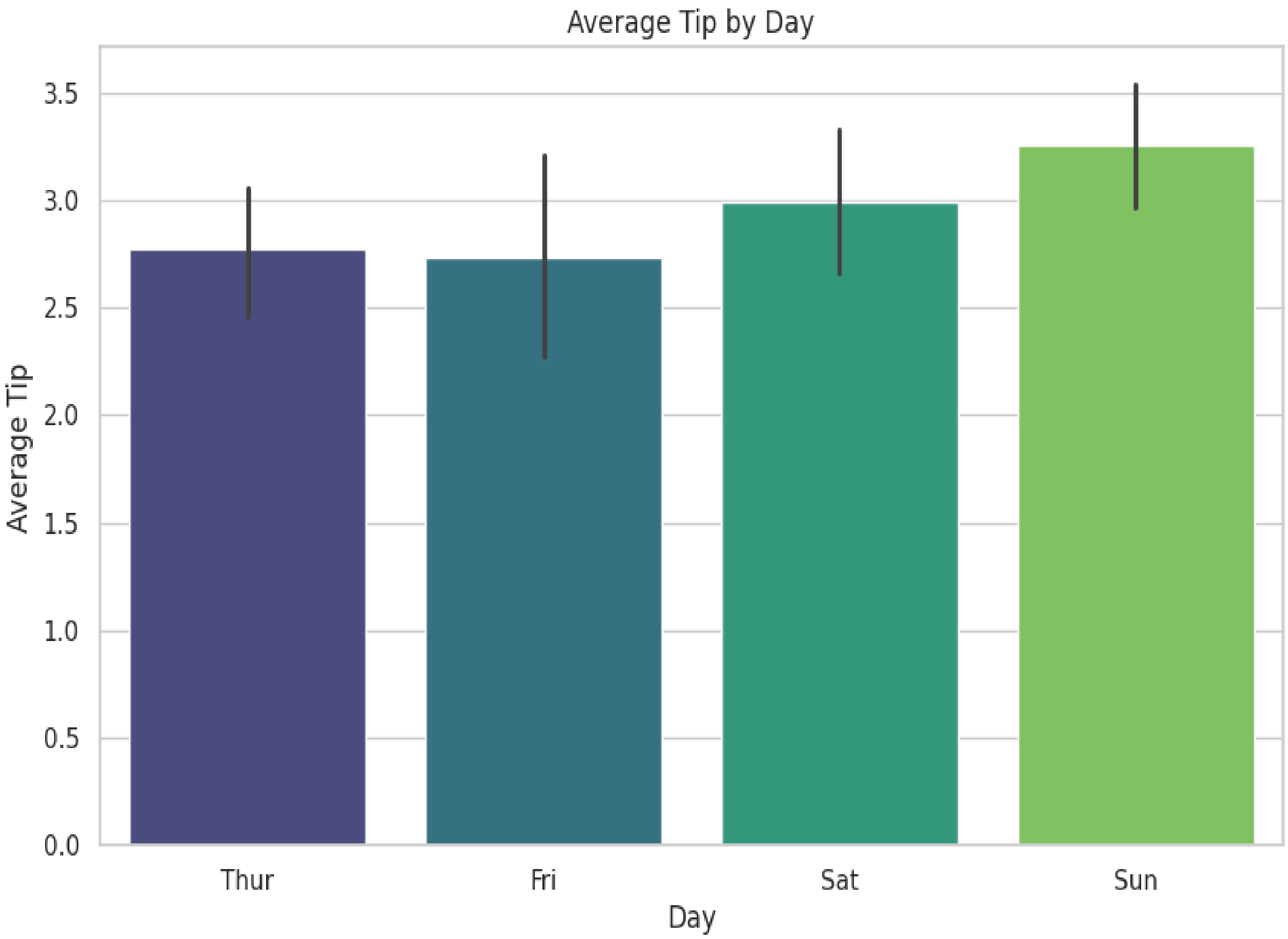
To plot different comparison plots and composition plots using a suitable dataset.

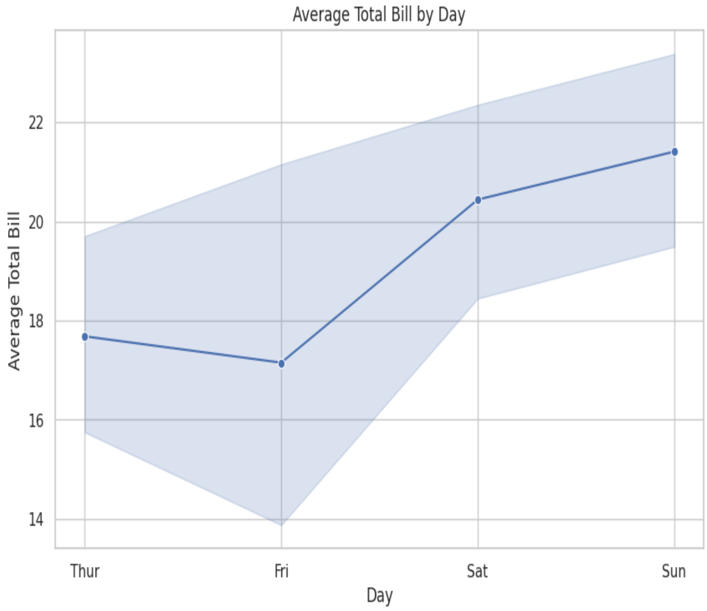
Code Snippet:

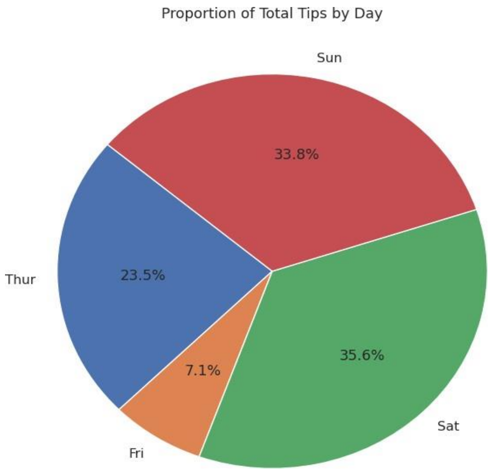


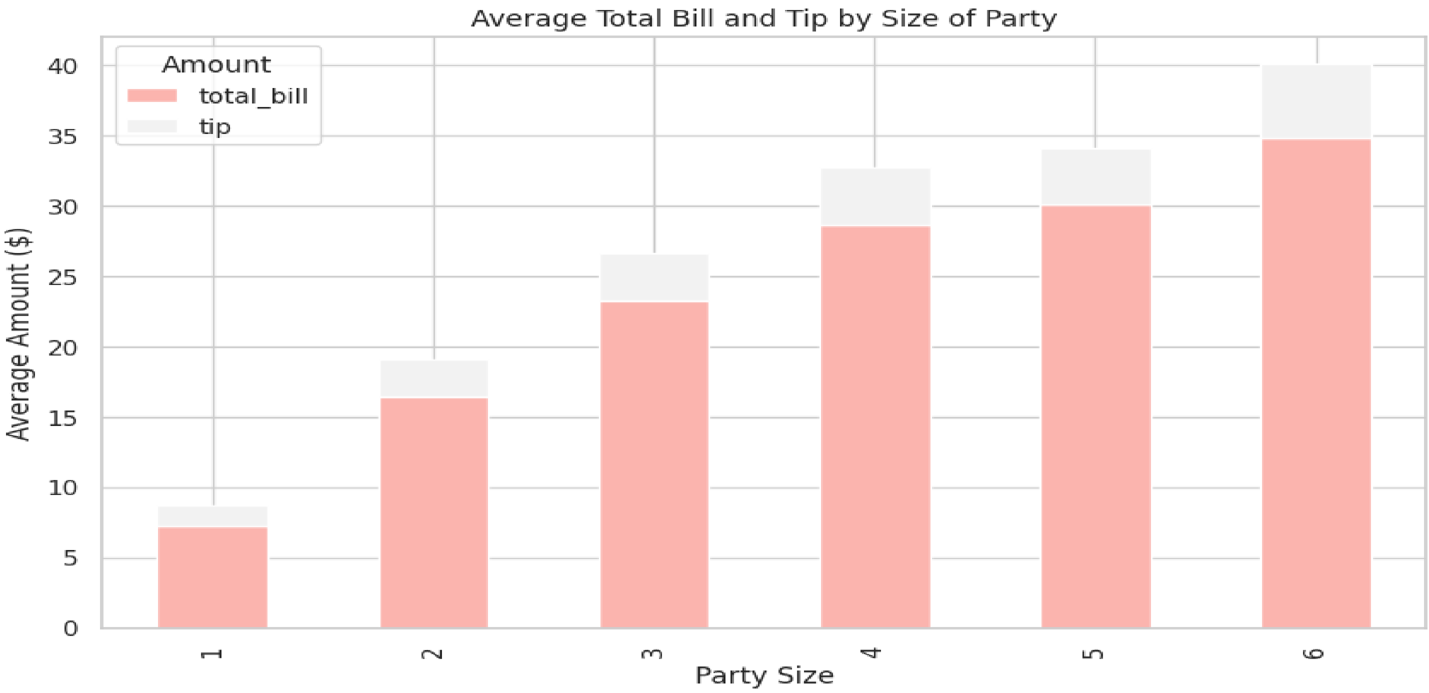
```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset('tips')

# Display the first few rows to understand the dataset structure
print(df.head())

# Set a consistent style for the plots
sns.set(style="whitegrid")

# 1. Comparison Plot - Line Plot: Average Total Bill by Day
plt.figure(figsize=(10, 6))
sns.lineplot(x='day', y='total_bill', data=df, estimator='mean', marker='o')
plt.title('Average Total Bill by Day')
plt.xlabel('Day')
plt.ylabel('Average Total Bill')
plt.show()

# 2. Comparison Plot - Bar Chart: Average Tip by Day
plt.figure(figsize=(10, 6))
sns.barplot(x='day', y='tip', data=df, estimator='mean', palette='viridis')
plt.title('Average Tip by Day')
plt.xlabel('Day')
plt.ylabel('Average Tip')
plt.show()

# 3. Comparison Plot - Scatter Plot: Total Bill vs Tip
plt.figure(figsize=(10, 6))
sns.scatterplot(x='total_bill', y='tip', data=df, hue='time')
plt.title('Scatter Plot of Total Bill vs Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()

# 4. Composition Plot - Pie Chart: Proportion of Tips by Day
tip_by_day = df.groupby('day')['tip'].sum()
plt.figure(figsize=(8, 8))
plt.pie(tip_by_day, labels=tip_by_day.index, autopct='%1.1f%%', startangle=140)
plt.title('Proportion of Total Tips by Day')
plt.show()

# 5. Composition Plot - Stacked Bar Chart: Average Tip and Total Bill by Size of Party
# Group by 'size' and calculate average 'total_bill' and 'tip'
# The change is here: added numeric_only=True to mean()
size_avg = df.groupby('size').mean(numeric_only=True)[['total_bill', 'tip']]

size_avg.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='Pastel1')
plt.title('Average Total Bill and Tip by Size of Party')
plt.xlabel('Party Size')
plt.ylabel('Average Amount ($)')
plt.legend(title='Amount')
plt.show()
```

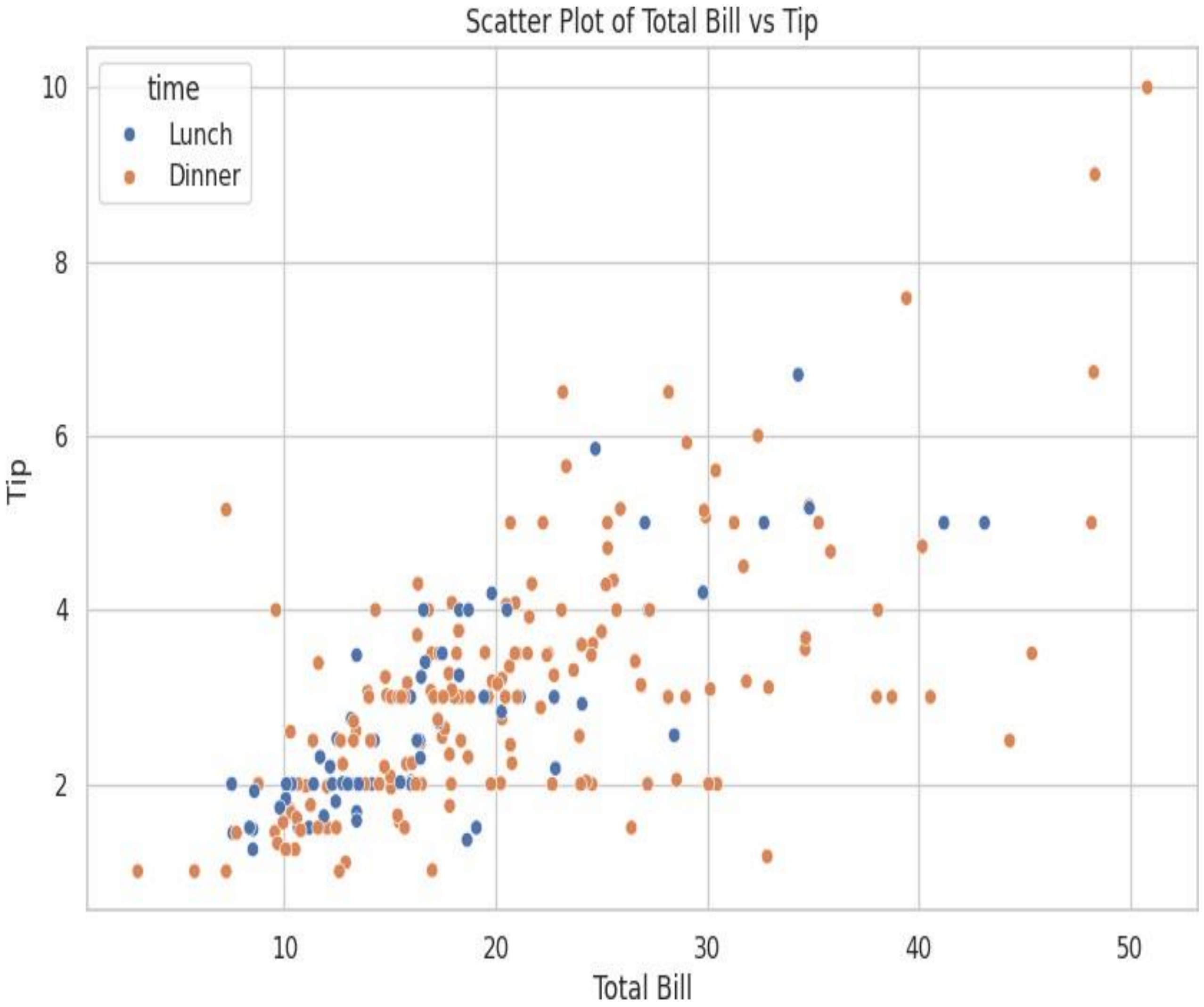
Output:


Average Tip by Day
Average Tip
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Thur
Fri
Sat
Sun
Day


Average Total Bill by Day
Average Total Bill
22
20
18
16
14
Thur
Fri
Sat
Sun
Day


Proportion of Total Tips by Day
Sun
33.8%
Thur
23.5%
7.1%
35.6%
Sat
Fri


Average Total Bill and Tip by Size of Party
Amount
total_bill
tip
Average Amount ($)
0
5
10
15
20
25
30
35
40
1
2
3
4
5
6
Party Size

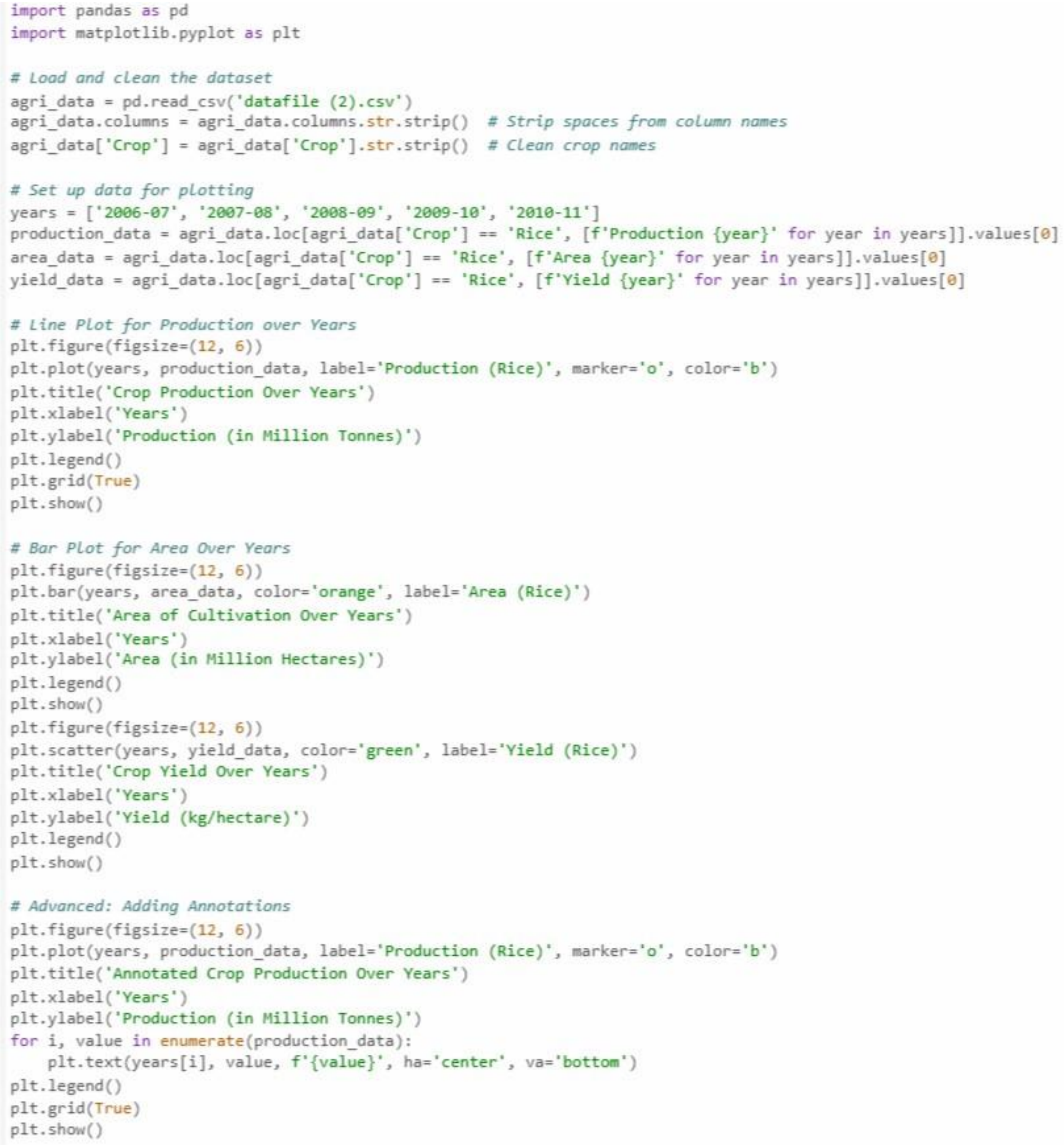

Scatter Plot of Total Bill vs Tip
time
Lunch
Dinner
Tip
2
4
6
8
10
10
20
30
40
50
Total Bill

5. Question 4 Develop a code using Matplotlib performing all Pyplot basics operation basic text and legend using Agriculture crop yield data set

Objective

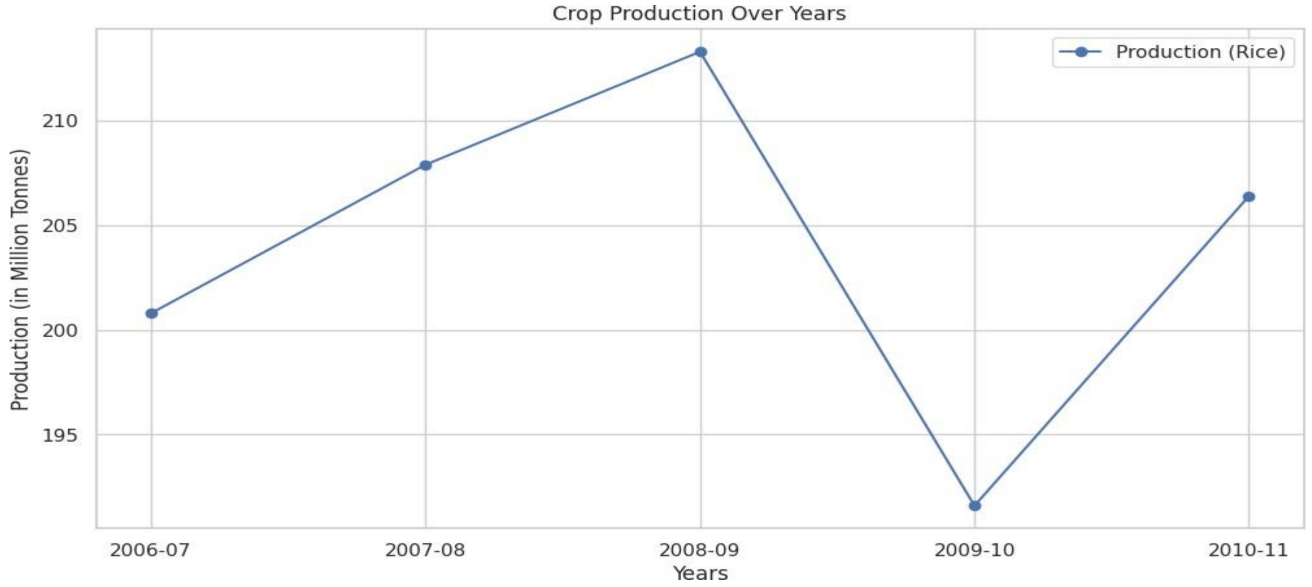
To perform basic operations using Matplotlib with an agriculture crop yield dataset

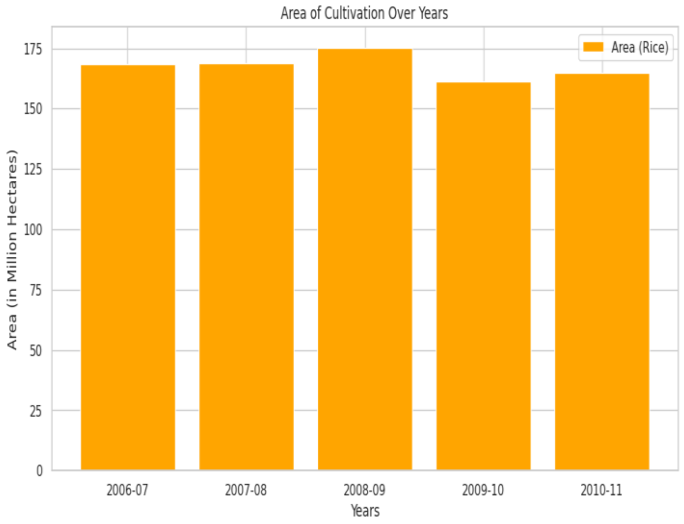
Code Snippet:

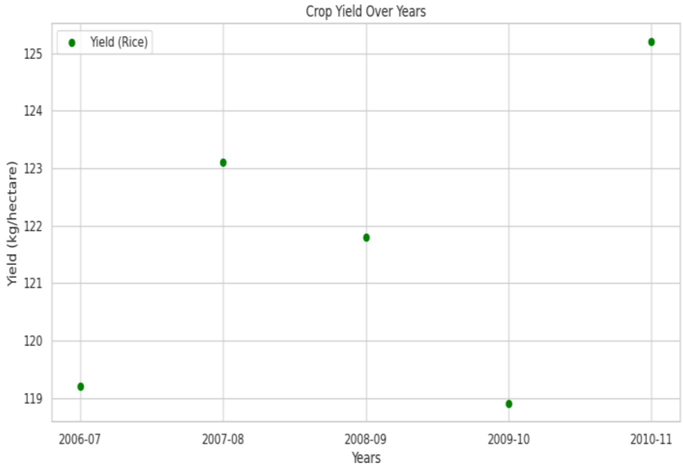


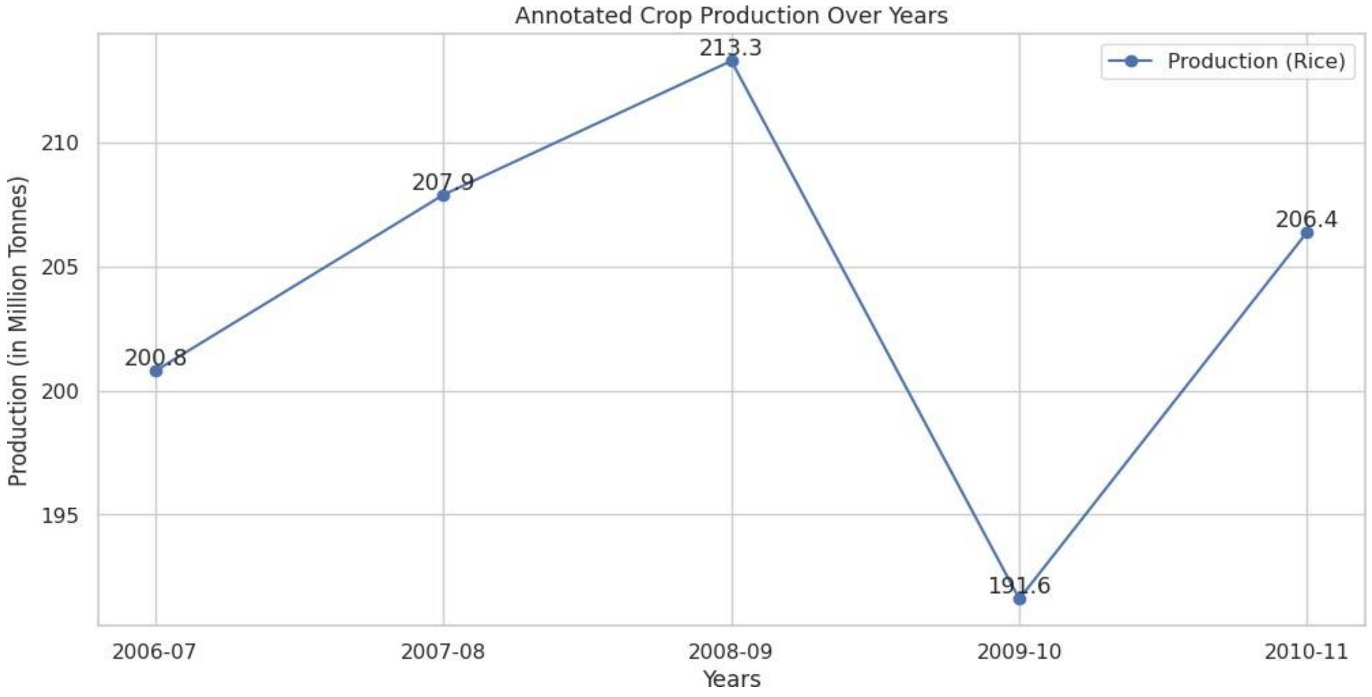
```
import pandas as pd
import matplotlib.pyplot as plt

# Load and clean the dataset
agri_data = pd.read_csv('datafile (2).csv')
agri_data.columns = agri_data.columns.str.strip()  # Strip spaces from column names
agri_data['Crop'] = agri_data['Crop'].str.strip()  # Clean crop names

# Set up data for plotting
years = ['2006-07', '2007-08', '2008-09', '2009-10', '2010-11']
production_data = agri_data.loc[agri_data['Crop'] == 'Rice', [f'Production {year}' for year in years]].values[0]
area_data = agri_data.loc[agri_data['Crop'] == 'Rice', [f'Area {year}' for year in years]].values[0]
yield_data = agri_data.loc[agri_data['Crop'] == 'Rice', [f'Yield {year}' for year in years]].values[0]

# Line Plot for Production over Years
plt.figure(figsize=(12, 6))
plt.plot(years, production_data, label='Production (Rice)', marker='o', color='b')
plt.title('Crop Production Over Years')
plt.xlabel('Years')
plt.ylabel('Production (in Million Tonnes)')
plt.legend()
plt.grid(True)
plt.show()

# Bar Plot for Area Over Years
plt.figure(figsize=(12, 6))
plt.bar(years, area_data, color='orange', label='Area (Rice)')
plt.title('Area of Cultivation Over Years')
plt.xlabel('Years')
plt.ylabel('Area (in Million Hectares)')
plt.legend()
plt.show()
plt.figure(figsize=(12, 6))
plt.scatter(years, yield_data, color='green', label='Yield (Rice)')
plt.title('Crop Yield Over Years')
plt.xlabel('Years')
plt.ylabel('Yield (kg/hectare)')
plt.legend()
plt.show()

# Advanced: Adding Annotations
plt.figure(figsize=(12, 6))
plt.plot(years, production_data, label='Production (Rice)', marker='o', color='b')
plt.title('Annotated Crop Production Over Years')
plt.xlabel('Years')
plt.ylabel('Production (in Million Tonnes)')
for i, value in enumerate(production_data):
    plt.text(years[i], value, f'{value}', ha='center', va='bottom')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

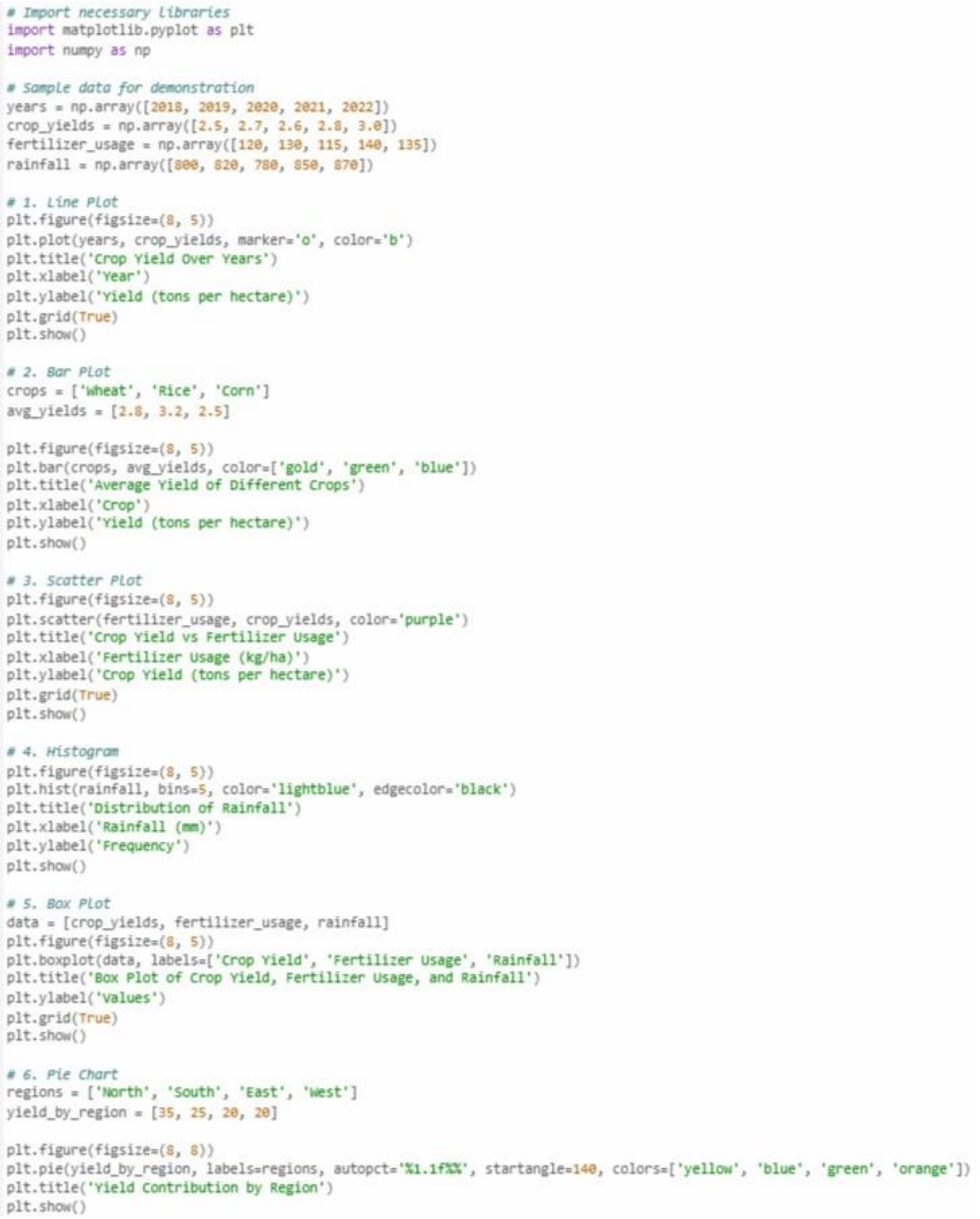

Crop Production Over Years
Production (Rice)
Production (in Million Tonnes)
210
205
200
195
2006-07
2007-08
2008-09
2009-10
2010-11
Years


Area of Cultivation Over Years
Area (Rice)
Area (in Million Hectares)
175
150
125
100
75
50
25
0
2006-07
2007-08
2008-09
2009-10
2010-11
Years


Crop Yield Over Years
Yield (Rice)
Yield (kg/hectare)
125
124
123
122
121
120
119
2006-07
2007-08
2008-09
2009-10
2010-11
Years

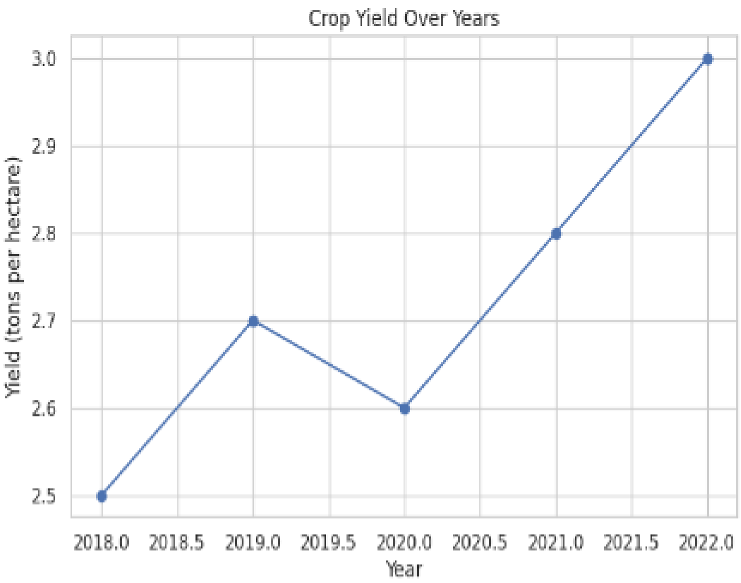

Annotated Crop Production Over Years
Production (Rice)
213.3
207.9
206.4
200.8
191.6
Production (in Million Tonnes)
210
205
200
195
2006-07
2007-08
2008-09
2009-10
2010-11
Years

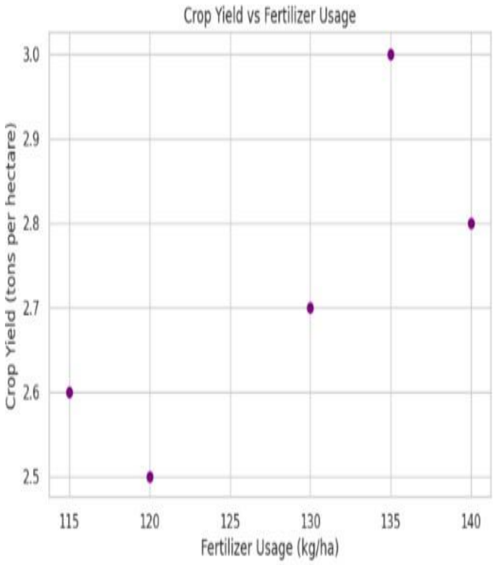
6. Question 5: Displaying Basic Plots with Matplotlib

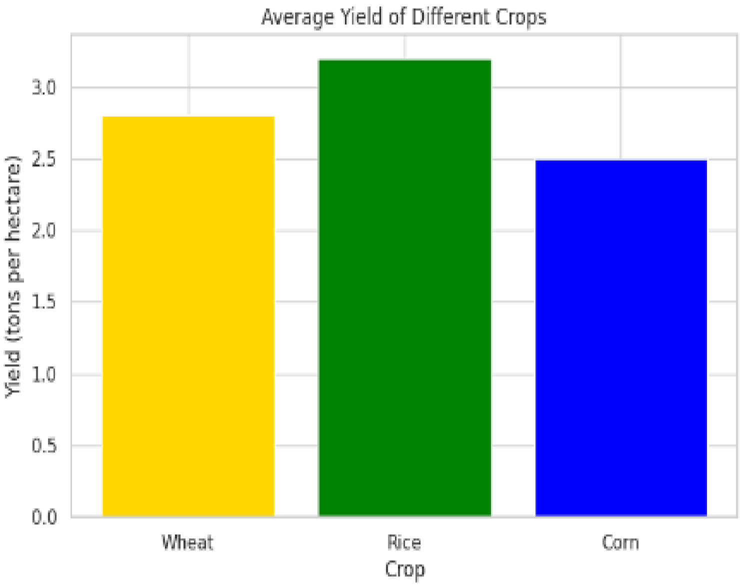


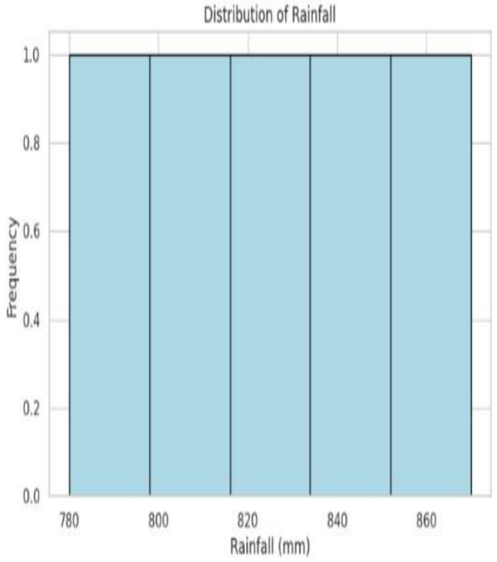
```
# Import necessary libraries
import matplotlib.pyplot as plt
import numpy as np

# Sample data for demonstration
years = np.array([2018, 2019, 2020, 2021, 2022])
crop_yields = np.array([2.5, 2.7, 2.6, 2.8, 3.0])
fertilizer_usage = np.array([120, 130, 115, 140, 135])
rainfall = np.array([800, 820, 780, 850, 870])

# 1. Line Plot
plt.figure(figsize=(8, 5))
plt.plot(years, crop_yields, marker='o', color='b')
plt.title('Crop Yield Over Years')
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')
plt.grid(True)
plt.show()

# 2. Bar Plot
crops = ['Wheat', 'Rice', 'Corn']
avg_yields = [2.8, 3.2, 2.5]

plt.figure(figsize=(8, 5))
plt.bar(crops, avg_yields, color=['gold', 'green', 'blue'])
plt.title('Average Yield of Different Crops')
plt.xlabel('Crop')
plt.ylabel('Yield (tons per hectare)')
plt.show()

# 3. Scatter Plot
plt.figure(figsize=(8, 5))
plt.scatter(fertilizer_usage, crop_yields, color='purple')
plt.title('Crop Yield vs Fertilizer Usage')
plt.xlabel('Fertilizer Usage (kg/ha)')
plt.ylabel('Crop Yield (tons per hectare)')
plt.grid(True)
plt.show()

# 4. Histogram
plt.figure(figsize=(8, 5))
plt.hist(rainfall, bins=5, color='lightblue', edgecolor='black')
plt.title('Distribution of Rainfall')
plt.xlabel('Rainfall (mm)')
plt.ylabel('Frequency')
plt.show()

# 5. Box Plot
data = [crop_yields, fertilizer_usage, rainfall]
plt.figure(figsize=(8, 5))
plt.boxplot(data, labels=['Crop Yield', 'Fertilizer Usage', 'Rainfall'])
plt.title('Box Plot of Crop Yield, Fertilizer Usage, and Rainfall')
plt.ylabel('Values')
plt.grid(True)
plt.show()

# 6. Pie Chart
regions = ['North', 'South', 'East', 'West']
yield_by_region = [35, 25, 20, 20]

plt.figure(figsize=(8, 8))
plt.pie(yield_by_region, labels=regions, autopct='%1.1f%%', startangle=140, colors=['yellow', 'blue', 'green', 'orange'])
plt.title('Yield Contribution by Region')
plt.show()
```

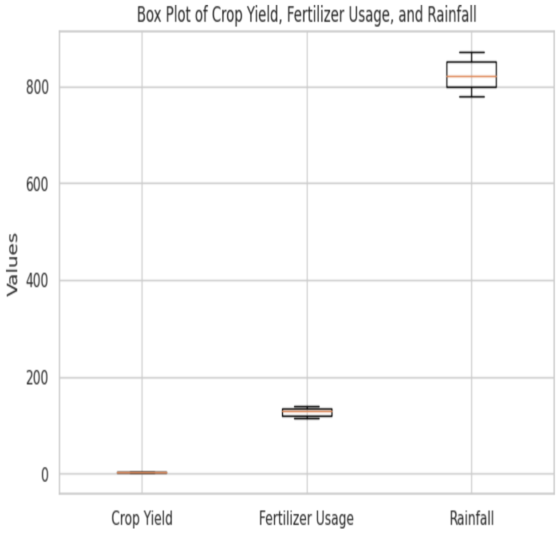
Output:

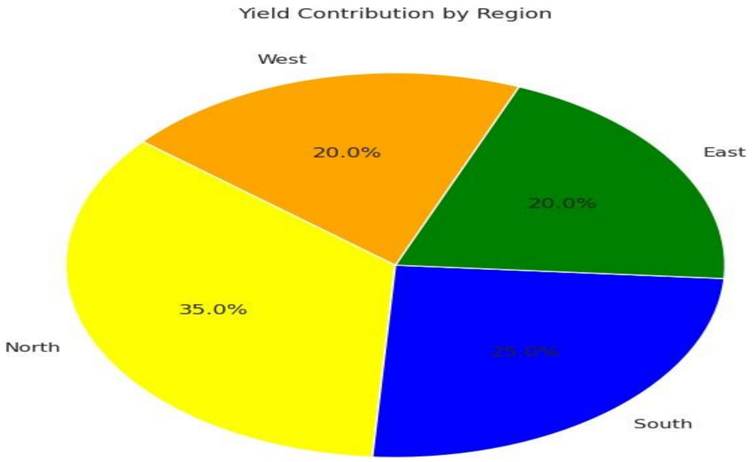

Crop Yield Over Years
Yield (tons per hectare)
3.0
2.9
2.8
2.7
2.6
2.5
2018.0
2018.5
2019.0
2019.5
2020.0
2020.5
2021.0
2021.5
2022.0
Year


Crop Yield vs Fertilizer Usage
Crop Yield (tons per hectare)
3.0
2.9
2.8
2.7
2.6
2.5
115
120
125
130
135
140
Fertilizer Usage (kg/ha)


Average Yield of Different Crops
Yield (tons per hectare)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Wheat
Rice
Corn
Crop


Distribution of Rainfall
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
780
800
820
840
860
Rainfall (mm)


Box Plot of Crop Yield, Fertilizer Usage, and Rainfall
Values
800
600
400
200
0
Crop Yield
Fertilizer Usage
Rainfall


Yield Contribution by Region
West
20.0%
East
20.0%
North
35.0%
25.0%
South

7. Question 6: Advantages of Seaborn and Aesthetic Control

Objective

To illustrate the advantages of Seaborn and demonstrate aesthetic control using Seaborn. Seaborn is a powerful visualization library in Python that builds on Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics. Below are some advantages of using Seaborn compared to Matplotlib, along with a code snippet illustrating how to control figure aesthetics.

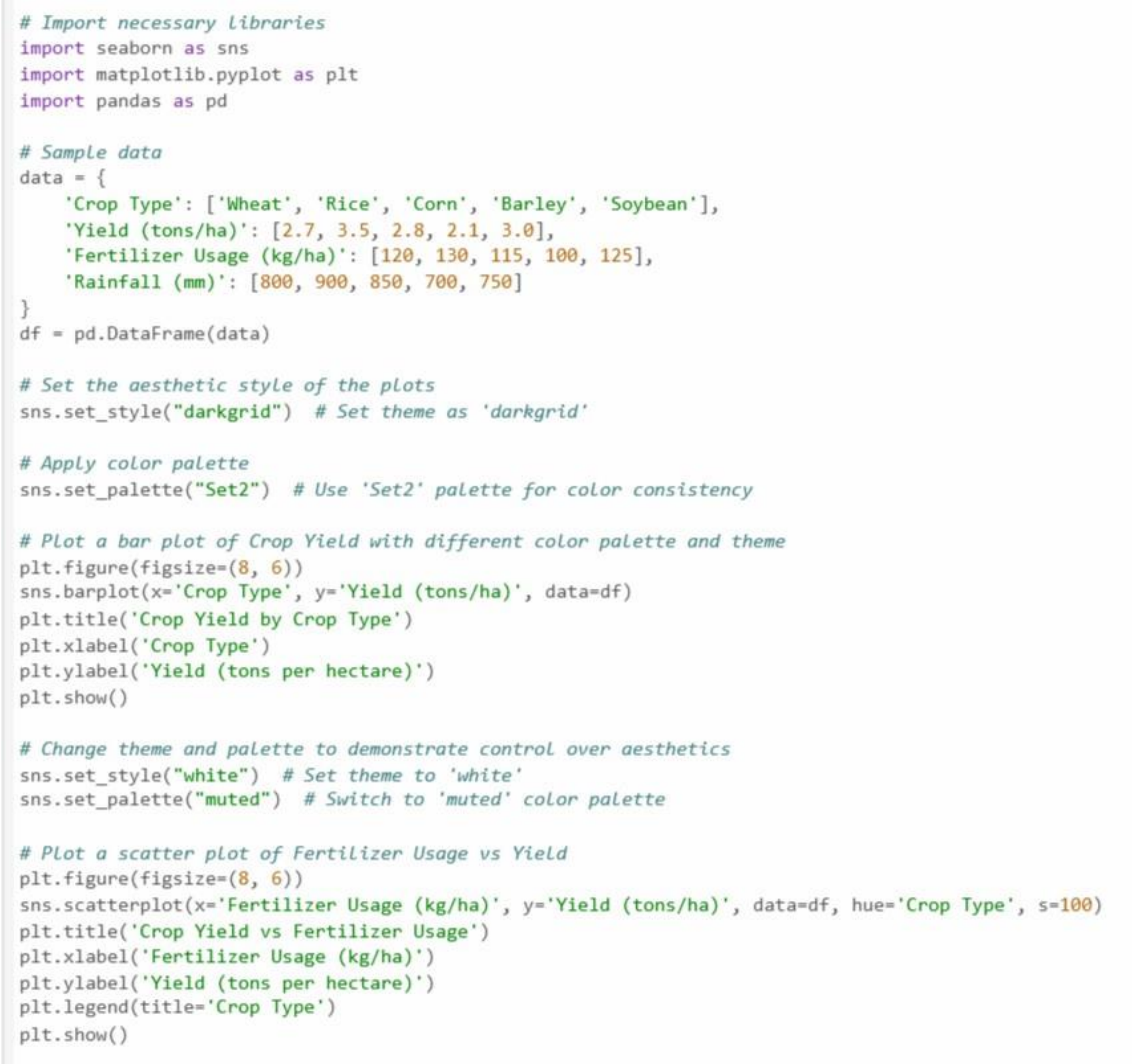
Advantages of Seaborn over Matplotlib Simplified Syntax:

Seaborn offers a more intuitive approach to data visualization, reducing the amount of code required to create sophisticated plots. It handles many common tasks automatically, such as configuring axes and legends. Additionally, Seaborn provides built-in functions for visualizing statistical relationships and distributions, making it easier to explore and understand data.

Seaborn's default styles are visually appealing and customizable, offering various themes to enhance the overall look of plots. It integrates seamlessly with Pandas DataFrames, allowing for direct plotting of data from these data structures. Seaborn also supports a range of advanced plot types, such as violin plots, pair plots, and heatmaps, which are not readily available in Matplotlib.

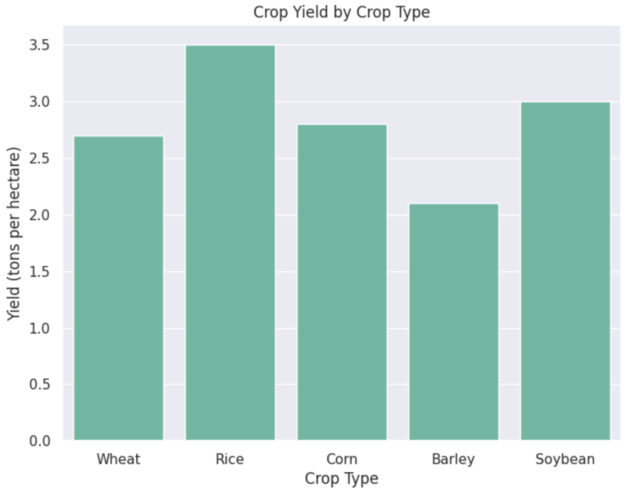
Here’s how to implement and control figure aesthetics in the enhanced box plot example:

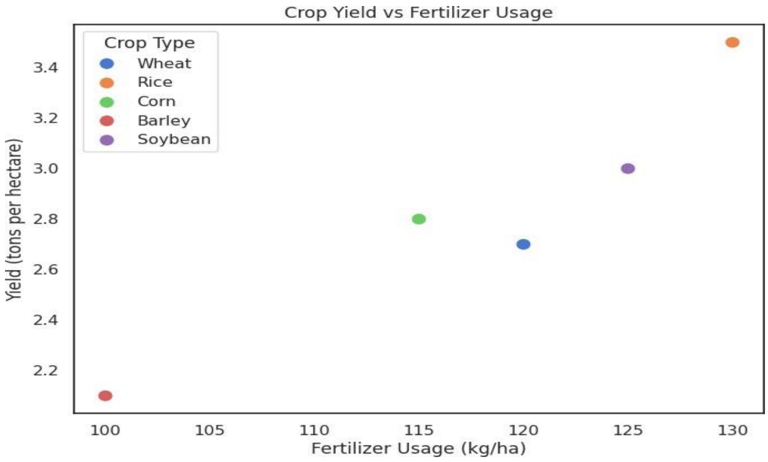
Code Snippet:



```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample data
data = {
    'Crop Type': ['Wheat', 'Rice', 'Corn', 'Barley', 'Soybean'],
    'Yield (tons/ha)': [2.7, 3.5, 2.8, 2.1, 3.0],
    'Fertilizer Usage (kg/ha)': [120, 130, 115, 100, 125],
    'Rainfall (mm)': [800, 900, 850, 700, 750]
}
df = pd.DataFrame(data)

# Set the aesthetic style of the plots
sns.set_style("darkgrid")  # Set theme as 'darkgrid'

# Apply color palette
sns.set_palette("Set2")  # Use 'Set2' palette for color consistency

# Plot a bar plot of Crop Yield with different color palette and theme
plt.figure(figsize=(8, 6))
sns.barplot(x='Crop Type', y='Yield (tons/ha)', data=df)
plt.title('Crop Yield by Crop Type')
plt.xlabel('Crop Type')
plt.ylabel('Yield (tons per hectare)')
plt.show()

# Change theme and palette to demonstrate control over aesthetics
sns.set_style("white")  # Set theme to 'white'
sns.set_palette("muted")  # Switch to 'muted' color palette

# Plot a scatter plot of Fertilizer Usage vs Yield
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Fertilizer Usage (kg/ha)', y='Yield (tons/ha)', data=df, hue='Crop Type', s=100)
plt.title('Crop Yield vs Fertilizer Usage')
plt.xlabel('Fertilizer Usage (kg/ha)')
plt.ylabel('Yield (tons per hectare)')
plt.legend(title='Crop Type')
plt.show()
```

Output:





8. Conclusion

This report presents a comprehensive analysis of various datasets, employing Python libraries such as NumPy, Pandas, Matplotlib, and Seaborn. Each question in this report addresses a specific data analysis and visualization task, demonstrating the versatility and effectiveness of these libraries.

9. References

- Pandas Documentation
- Numpy Documentation
- Matplotlib Documentation
- Seaborn Documentation

GitHub Repository: https://github.com/Uzi0312/Data-Visualization-A-1