VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

Assignment Report
on

Data Visualization

Submitted By

OMKAR MURTHY P - 1RN21CD034

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 solutions to various data analysis and visualization tasks using Python libraries such as Numpy, Pandas, Matplotlib, and Seaborn. The datasets used include Apple stock data, TikTok video performance data, and agriculture crop yield data. Each question addresses a specific aspect of data analysis and visualization.

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 data
data = pd.read_csv(r'C:\Users\Abhishek P\Downloads\archive (1)\HistoricalQuotes.csv')

# Clean up column names by removing leading/trailing spaces
data.columns = data.columns.str.strip()

# Remove dollar signs and convert 'Close/Last' to a numeric type
data['Close/Last'] = data['Close/Last'].replace('[\$,]', '', regex=True).astype(float)

# Calculate the mean, median, mode, and standard deviation of the 'Close/Last' prices
mean_close = np.mean(data['Close/Last'])
median_close = np.median(data['Close/Last'])
mode_close = data['Close/Last'].mode()[0]  # Taking the first mode in case of multiple modes
std_dev_close = np.std(data['Close/Last'])

# Print the results
print(f"Mean of 'Close/Last' prices: {mean_close}")
print(f"Median of 'Close/Last' prices: {median_close}")
print(f"Mode of 'Close/Last' prices: {mode_close}")
print(f"Standard Deviation of 'Close/Last' prices: {std_dev_close}")
```

Output:

```
Mean of 'Close/Last' prices: 114.76952227958698
Median of 'Close/Last' prices: 101.09
Mode of 'Close/Last' prices: 97.34
Standard Deviation of 'Close/Last' prices: 60.65035824572462
```

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 data
tiktok_data = pd.read_csv(r'C:\Users\Abhishek P\Downloads\archive (2)\tiktok_performance.csv')

# Display basic information
print("Basic Information:")
print(tiktok_data.info())
print("\nDescriptive Statistics:")
print(tiktok_data.describe())

# Basic operations
# 1. Calculate the total number of likes and comments across all videos
total_likes = tiktok_data['Likes'].sum()
total_comments = tiktok_data['Comments'].sum()
print(f"\nTotal Likes: {total_likes}")
print(f"Total Comments: {total_comments}")

# 2. Calculate the mean number of views per category
mean_views_category = tiktok_data.groupby('Category')['Views'].mean()
print("\nMean Views per Category:")
print(mean_views_category)

# 3. Find the most liked video
most_liked_video = tiktok_data[tiktok_data['Likes'] == tiktok_data['Likes'].max()]
print("\nMost Liked Video:")
print(most_liked_video[['Video_Title', 'Likes']])

# Advanced operations
# 4. Add a new column for the engagement rate (likes + comments + shares) / views
tiktok_data['Engagement_Rate'] = (tiktok_data['Likes'] + tiktok_data['Comments'] +
tiktok_data['Shares']) / tiktok_data['Views']
print("\nEngagement Rate (Top 5 rows):")
print(tiktok_data[['Video_Title', 'Engagement_Rate']].head())

# 5. Normalize 'User_Followers' using Min-Max scaling
tiktok_data['Normalized_Followers'] = (tiktok_data['User_Followers'] -
tiktok_data['User_Followers'].min()) / (tiktok_data['User_Followers'].max() -
tiktok_data['User_Followers'].min())
print("\nNormalized Followers (Top 5 rows):")
print(tiktok_data[['Username', 'User_Followers', 'Normalized_Followers']].head())

# 6. Calculate the correlation matrix for numeric features
correlation_matrix = tiktok_data[['Likes', 'Comments', 'Shares', 'Views', 'User_Followers',
'User_Following', 'User_Likes']].corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)

# 7. Advanced Aggregation: Find the average engagement rate per category
avg_engagement_rate_category = tiktok_data.groupby('Category')['Engagement_Rate'].mean()
print("\nAverage Engagement Rate per Category:")
print(avg_engagement_rate_category)

# 8. Top 5 videos with the highest engagement rate
top_videos_engagement = tiktok_data.nlargest(5, 'Engagement_Rate')[['Video_Title', 'Engagement_Rate']]
print("\nTop 5 Videos with Highest Engagement Rate:")
print(top_videos_engagement)

# Save the updated data with engagement rate and normalized followers as a new CSV file
tiktok_data.to_csv(r'C:\Users\Abhishek P\Downloads\updated_tiktok_performance.csv', index=False)
print("\nUpdated dataset saved as 'updated_tiktok_performance.csv'")
```

Output:

```
Basic Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Video_ID        5 non-null      int64
 1   User_ID         5 non-null      int64
 2   Username        5 non-null      object
 3   Video_Title     5 non-null      object
 4   Category        5 non-null      object
 5   Likes           5 non-null      int64
 6   Comments        5 non-null      int64
 7   Shares          5 non-null      int64
 8   Views           5 non-null      int64
 9   Upload_Date     5 non-null      object
 10  Video_Length    5 non-null      int64
 11  Hashtags        5 non-null      object
 12  User_Followers  5 non-null      int64
 13  User_Following  5 non-null      int64
 14  User_Likes      5 non-null      int64
dtypes: int64(10), object(5)
memory usage: 732.0+ bytes
None

Descriptive Statistics:
         Video_ID   User_ID        Likes    Comments      Shares    Views  \
count    5.000000  5.000000     5.000000    5.000000    5.000000      5.0
mean   103.000000  3.000000  2260.000000  230.000000  352.000000  60000.0
std      1.581139  1.581139  1316.434579  153.948043  155.788318  20000.0
min    101.000000  1.000000  1200.000000  120.000000  210.000000  40000.0
25%    102.000000  2.000000  1500.000000  150.000000  250.000000  50000.0
50%    103.000000  3.000000  1800.000000  180.000000  300.000000  50000.0
75%    104.000000  4.000000  2300.000000  200.000000  400.000000  70000.0
max    105.000000  5.000000  4500.000000  500.000000  600.000000  90000.0

       Video_Length  User_Followers  User_Following  User_Likes
count        5.0000          5.0000        5.000000     5.00000
mean        42.0000       1600.0000      350.000000  5000.00000
std         12.5499        308.2207      111.803399  1581.13883
min         30.0000       1200.0000      200.000000  3000.00000
25%         30.0000       1500.0000      300.000000  4000.00000
50%         45.0000       1500.0000      350.000000  5000.00000
75%         45.0000       1800.0000      400.000000  6000.00000
max         60.0000       2000.0000      500.000000  7000.00000

Total Likes: 11300
Total Comments: 1150
```

```
Engagement Rate (Top 5 rows):
       Video_Title  Engagement_Rate
0  Dance Challenge         0.038400
1       Funny Skit         0.041429
2         Tutorial         0.040000
3       Viral Dance        0.062222
4    Comedy Sketch         0.043800

Normalized Followers (Top 5 rows):
  Username  User_Followers  Normalized_Followers
0    user1            1500                 0.375
1    user2            2000                 1.000
2    user3            1200                 0.000
3    user4            1800                 0.750
4    user5            1500                 0.375

Correlation Matrix:
                   Likes  Comments    Shares     Views  User_Followers  \
Likes           1.000000  0.980694  0.939122  0.959030        0.622301
Comments        0.980694  1.000000  0.901669  0.893158        0.474184
Shares          0.939122  0.901669  1.000000  0.954821        0.671635
Views           0.959030  0.893158  0.954821  1.000000        0.811107
User_Followers  0.622301  0.474184  0.671635  0.811107        1.000000
User_Following  0.535052  0.384908  0.538247  0.726722        0.979393
User_Likes      0.852764  0.739483  0.903286  0.948683        0.872082

                User_Following  User_Likes
Likes                 0.535052    0.852764
Comments              0.384908    0.739483
Shares                0.538247    0.903286
Views                 0.726722    0.948683
User_Followers        0.979393    0.872082
User_Following        1.000000    0.777817
User_Likes            0.777817    1.000000

Average Engagement Rate per Category:
Category
Comedy      0.042614
Dance       0.050311
Tutorial    0.040000
Name: Engagement_Rate, dtype: float64

Top 5 Videos with Highest Engagement Rate:
       Video_Title  Engagement_Rate
3      Viral Dance         0.062222
4    Comedy Sketch         0.043800
1       Funny Skit         0.041429
2         Tutorial         0.040000
0  Dance Challenge         0.038400

Updated dataset saved as 'updated_tiktok_performance.csv'
```

4. Question 3: Comparison and Composition Plots

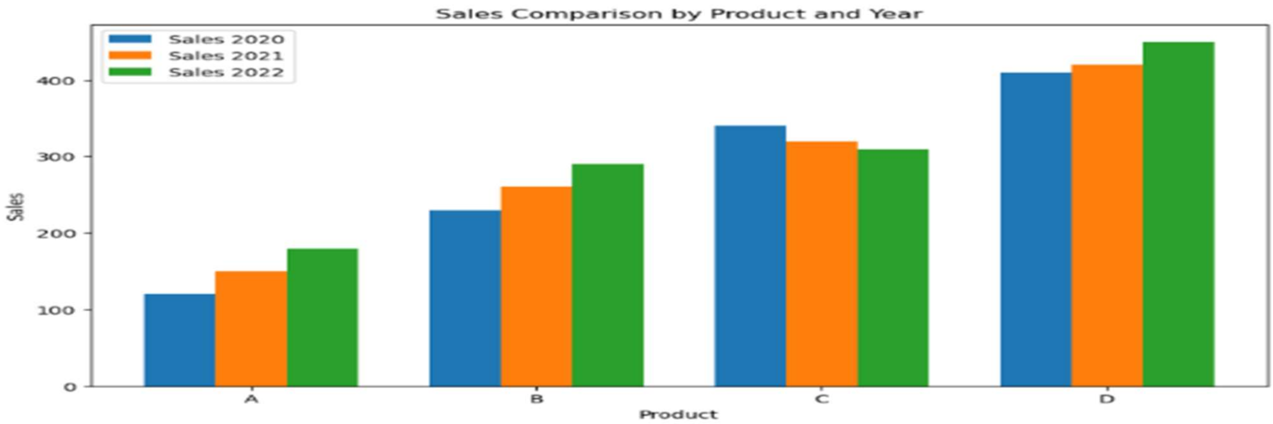
Objective:

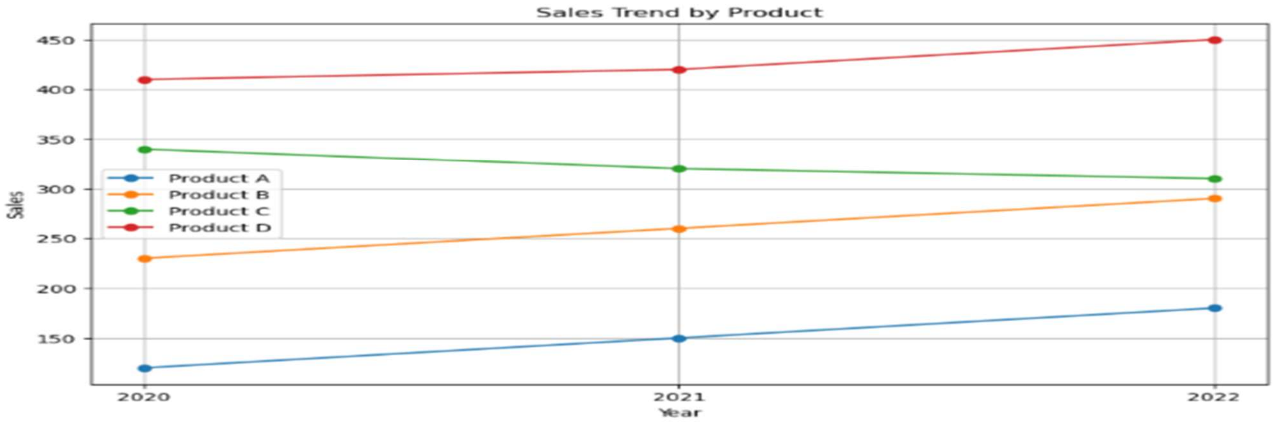
To plot different comparison plots and composition plots using a suitable dataset.

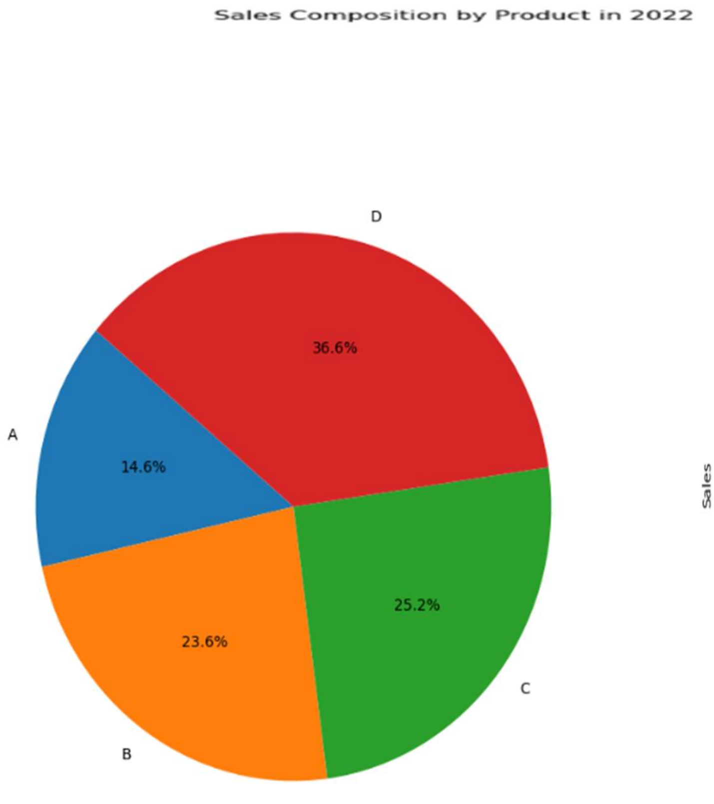
Code Snippet:

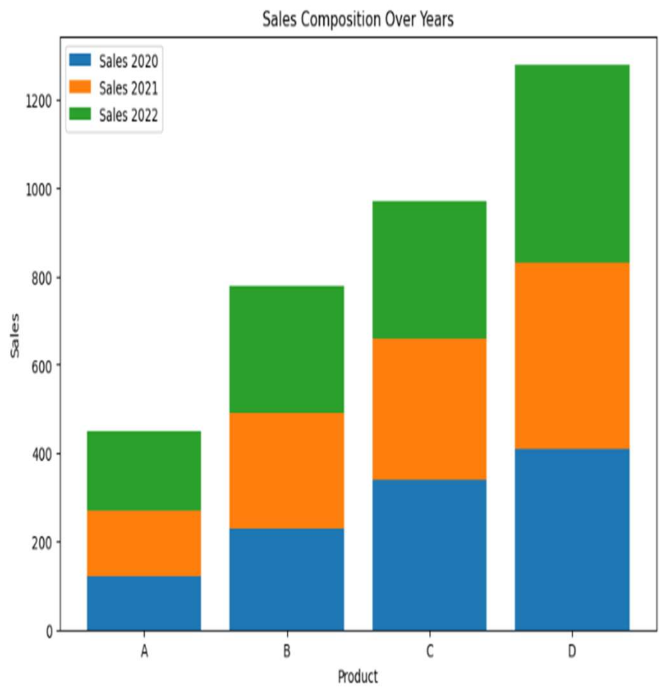
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Sample data for demonstration
data = {
    'Product': ['A', 'B', 'C', 'D'],
    'Sales 2020': [120, 230, 340, 410],
    'Sales 2021': [150, 260, 320, 420],
    'Sales 2022': [180, 290, 310, 450],
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Comparison Plot 1: Bar Plot for Sales Comparison Across Years
plt.figure(figsize=(10, 6))
x = np.arange(len(df['Product']))  # Label locations
width = 0.25  # Width of bars

# Bar plot for each year
plt.bar(x - width, df['Sales 2020'], width, label='Sales 2020')
plt.bar(x, df['Sales 2021'], width, label='Sales 2021')
plt.bar(x + width, df['Sales 2022'], width, label='Sales 2022')

# Add labels, title, and legend
plt.xlabel('Product')
plt.ylabel('Sales')
plt.title('Sales Comparison by Product and Year')
plt.xticks(x, df['Product'])
plt.legend()
plt.show()

# Comparison Plot 2: Line Plot for Sales Trend Over Years
years = ['2020', '2021', '2022']
plt.figure(figsize=(10, 6))

for i, product in enumerate(df['Product']):
    plt.plot(years, df.iloc[i, 1:], label=f'Product {product}', marker='o')

plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Sales Trend by Product')
plt.legend()
plt.grid(True)
plt.show()

# Composition Plot 1: Pie Chart for Product Sales Composition in 2022
plt.figure(figsize=(8, 8))
plt.pie(df['Sales 2022'], labels=df['Product'], autopct='%1.1f%%', startangle=140)
plt.title('Sales Composition by Product in 2022')
plt.show()

# Composition Plot 2: Stacked Bar Chart for Sales Composition Over Years
plt.figure(figsize=(10, 6))
bottom_values = np.zeros(len(df['Product']))

# Create stacked bar segments for each year
for i, year in enumerate(years):
    plt.bar(df['Product'], df[f'Sales {year}'], label=f'Sales {year}',
bottom=bottom_values)
bottom_values += df[f'Sales {year}']

plt.xlabel('Product')
plt.ylabel('Sales')
plt.title('Sales Composition Over Years')
plt.legend()
plt.show()
```

Output:


Sales Comparison by Product and Year
Sales 2020
Sales 2021
Sales 2022
Sales
0
100
200
300
400
A
B
C
D
Product


Sales Trend by Product
Product A
Product B
Product C
Product D
Sales
150
200
250
300
350
400
450
2020
2021
2022
Year


Sales Composition by Product in 2022
D
A
C
B
36.6%
14.6%
23.6%
25.2%


Sales Composition Over Years
Sales 2020
Sales 2021
Sales 2022
Sales
0
200
400
600
800
1000
1200
A
B
C
D
Product

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

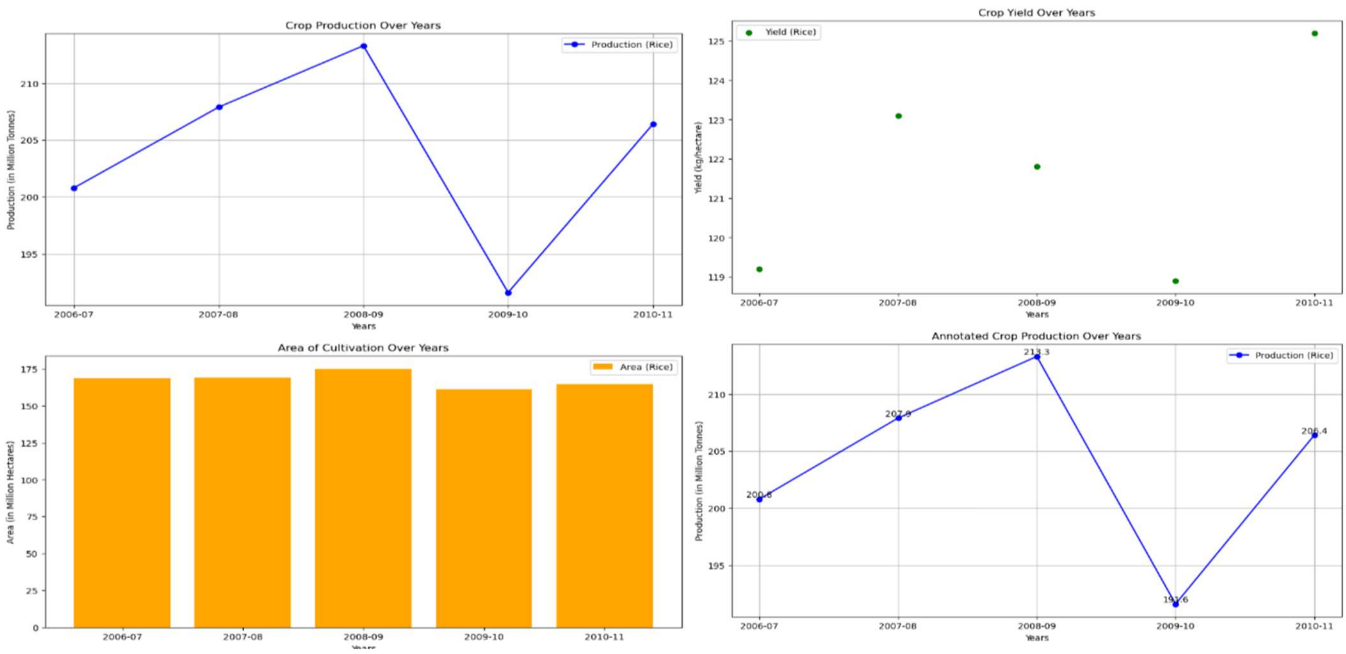
Objective

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

Code Snippet:

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

# Load and clean the dataset
agri_data = pd.read_csv(r'C:\Users\Abhishek P\Downloads\archive (3)\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()

# Scatter Plot for Yield Over Years
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:

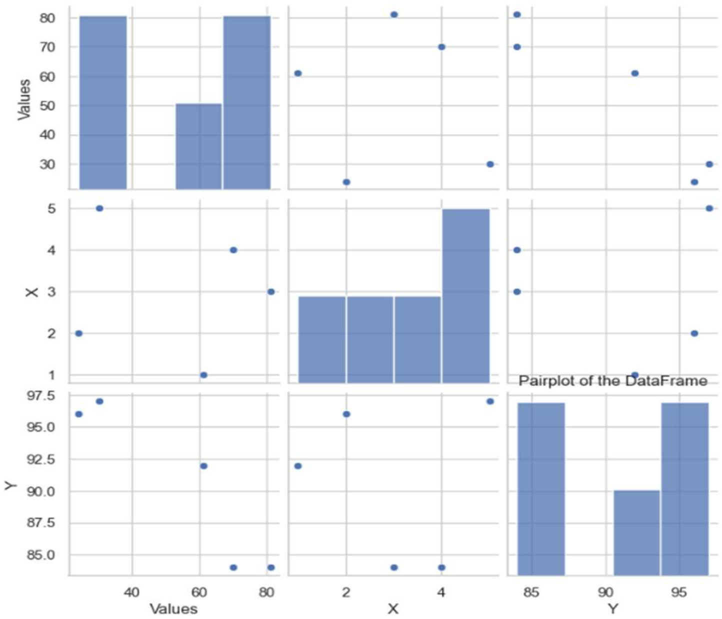


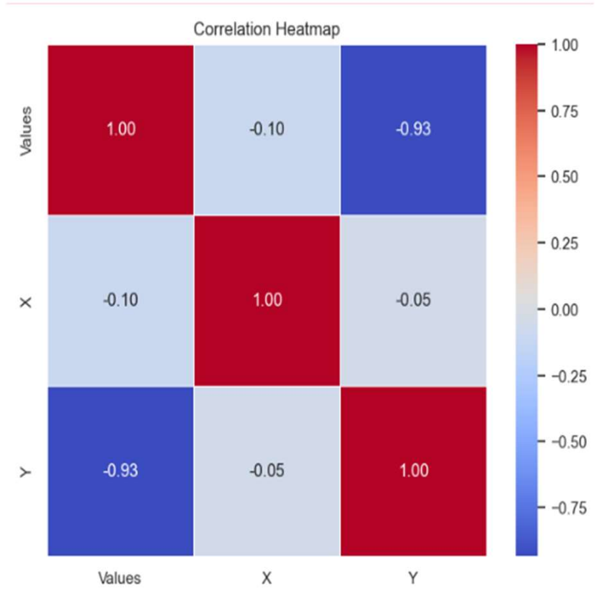
6. Question 5: Displaying Basic Plots with Matplotlib

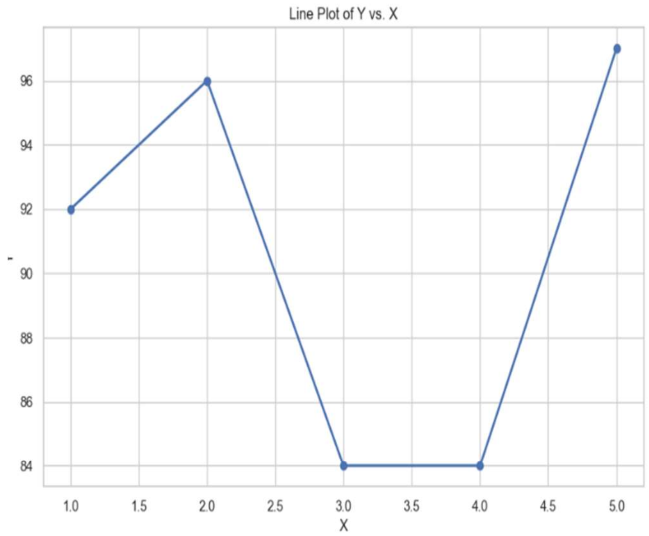
```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load and clean the dataset
agri_data = pd.read_csv(r'C:\Users\Abhishek P\Downloads\archive (3)\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()

# Scatter Plot for Yield Over Years
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()
```

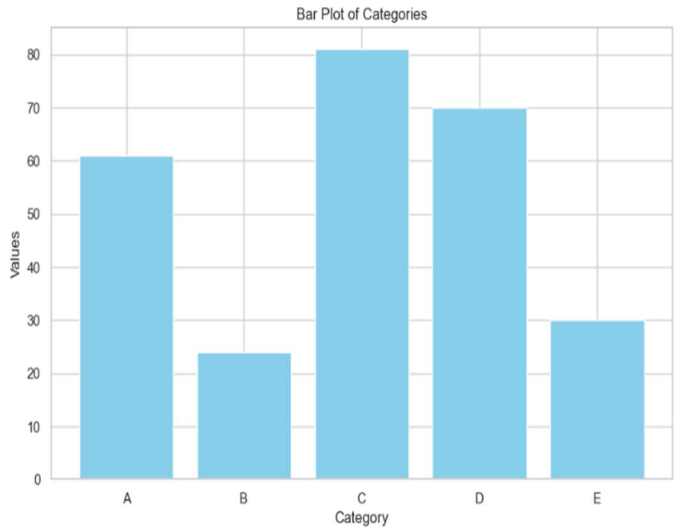
Output:

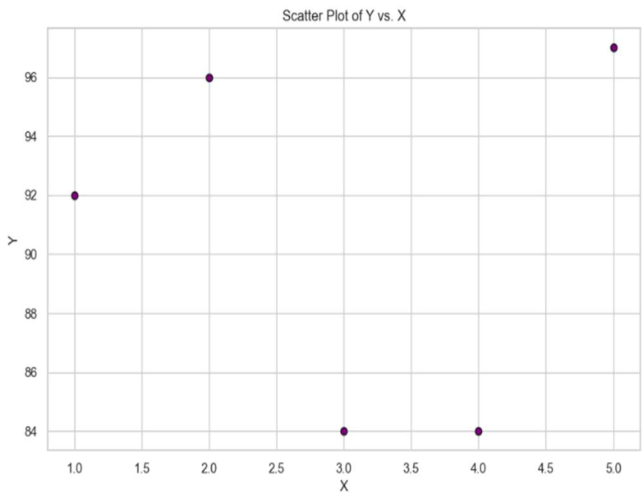
```
Statistical Measures for Values:
Mean: 53.20
Median: 61.00
Standard Deviation: 25.03
```

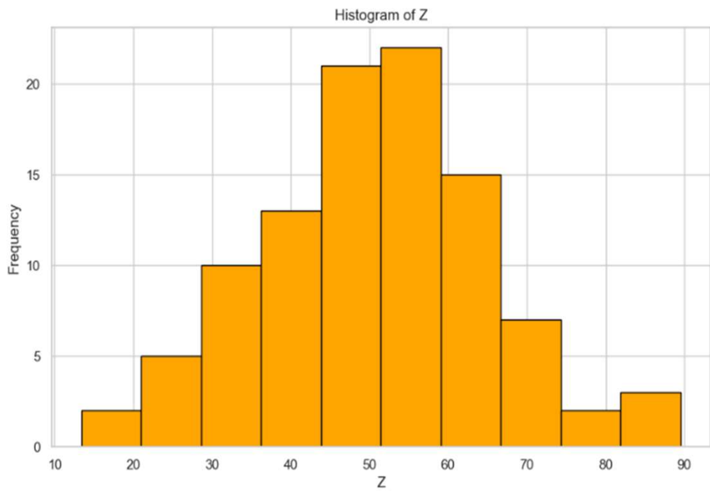


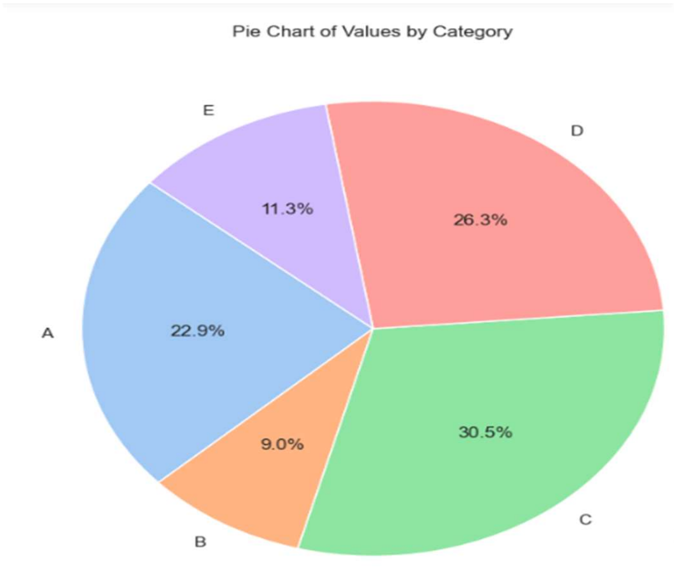


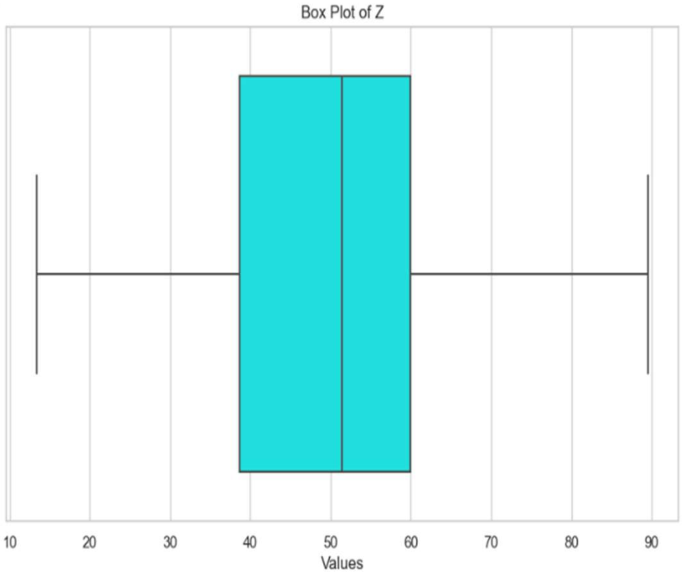












7. Question 6: Advantages of Seaborn and Aesthetic Control

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 provides a more user-friendly API for creating complex visualizations with fewer lines of code. It handles many tasks automatically, such as setting up axes and handling legend placements. Statistical Functions:

Seaborn comes with built-in support for visualizing statistical relationships and distributions, making it easier to create plots that convey data distributions, trends, and comparisons. Enhanced Default Aesthetics:

Seaborn's default styles are more visually appealing than Matplotlib's. It offers several themes (e.g., darkgrid, whitegrid) that can enhance the overall appearance of plots without extensive customization. Integration with Pandas:

Seaborn works seamlessly with Pandas DataFrames, allowing for easy plotting of data contained in DataFrames with straightforward syntax. Advanced Plot Types:

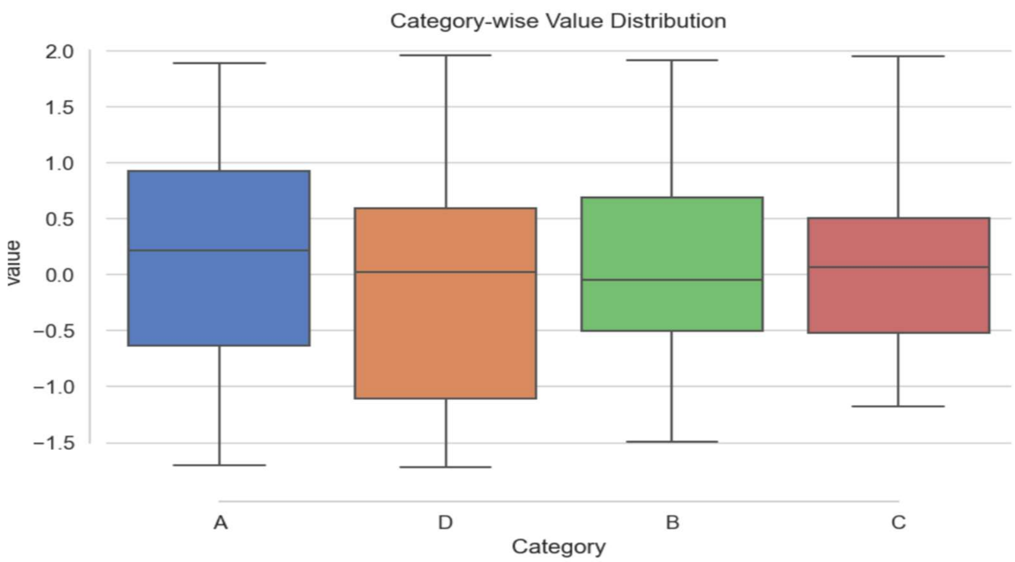
Seaborn supports a variety of specialized plot types (e.g., violin plots, pair plots, heatmaps) that are not available in Matplotlib without additional coding. Controlling Figure Aesthetics with Seaborn When creating visualizations, controlling aesthetics is crucial for enhancing clarity and appeal. Seaborn provides various ways to adjust figure aesthetics, including color palettes, font sizes, and styles.

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

Code Snippet:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample dataset
np.random.seed(0)
df = pd.DataFrame({
    'Category': np.random.choice(['A',
'B', 'C', 'D'], 100),
    'Value': np.random.normal(0, 1,
100)
})

# Set the aesthetic style
sns.set(style='whitegrid',
palette='muted', font_scale=1.2)

# Plot using Seaborn
plt.figure(figsize=(10, 6))
sns.boxplot(x='Category', y='Value',
data=df)
sns.despine(offset=10, trim=True)
plt.title('Category-wise Value
Distribution')
plt.show()
```

Output:



This snippet demonstrates Seaborn's ability to enhance plot aesthetics through sns.set, which adjusts the style, color palette, and font sizes for a cohesive look. The sns.despine function removes the top and right borders, adding to the minimalist and modern aesthetic, while the muted color palette keeps visual elements subtle yet distinctive.

Seaborn thus provides powerful tools to control and enhance figure aesthetics, making it ideal for producing visually engaging, insightful, and professional visualizations with minimal code.

8. Conclusion

This report demonstrates various data analysis and visualization techniques using Python libraries such as Numpy, Pandas, Matplotlib, and Seaborn. Each question addresses a specific aspect of data analysis and visualization, showcasing the capabilities of these libraries.

9. References

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

GitHub Repo Link: https://github.com/omkar-252003/DATA-VIZUALIZATION-WITH-MATPLOTLIB-AND-SEABORN