Program 2a

Problem Statement

Develop a comprehensive Python program to analyze and visualize the Breast Cancer Wisconsin dataset using matplotlib. The primary objective of the program is to generate various plots to illustrate the distribution of data and the relationships between different attributes of the dataset.

```
import matplotlib.pyplot as plt # Importing the Matplotlib library for plotting

import pandas as pd # Importing the Pandas library for data manipulation

from sklearn.datasets import load_breast_cancer # Importing the Breast Cancer dataset from sklearn


# Load the breast cancer dataset

cancer = load_breast_cancer() # Loading the built-in Breast Cancer dataset


# Convert to pandas DataFrame

data = pd.DataFrame(cancer.data, columns=cancer.feature_names) # Converting the dataset to a pandas DataFrame

data['target'] = cancer.target # Adding the target column to the DataFrame


# Display a concise summary of the DataFrame

print(data.info()) # Displaying a concise summary of the DataFrame, including the number of entries, columns, non-null values, and data types


# Display the first few rows of the dataset

print(data.head(1)) # Displaying the first few rows of the DataFrame to get an initial look at the data


# Display basic statistics

print(data.describe()) # Displaying basic statistical details like mean, std deviation, min, and max values for each column


# Check for any missing values

print(data.isnull().sum()) # Checking for any missing values in the DataFrame
```

```
# Line Plot
plt.figure(figsize=(10, 6)) # Setting the figure size for the plot
plt.plot(data.index, data['mean radius'], label='Mean Radius') # Creating a line plot for the 'mean radius' column
plt.title('Line Plot of Mean Radius') # Adding a title to the plot
plt.xlabel('Index') # Adding a label to the X-axis
plt.ylabel('Mean Radius') # Adding a label to the Y-axis
plt.legend() # Adding a legend to the plot
plt.grid(True) # Enabling the grid for the plot
plt.show() # Displaying the plot

# Scatter Plot
plt.figure(figsize=(10, 6)) # Setting the figure size for the plot
plt.scatter(data['mean radius'], data['mean texture'], c=data['target'], cmap='coolwarm', alpha=0.5) # Creating a scatter plot with 'mean radius' and 'mean texture', color-coded by the target class
plt.title('Scatter Plot of Mean Radius vs Mean Texture') # Adding a title to the plot
plt.xlabel('Mean Radius') # Adding a label to the X-axis
plt.ylabel('Mean Texture') # Adding a label to the Y-axis
plt.grid(True) # Enabling the grid for the plot
plt.show() # Displaying the plot

# Bar Plot
plt.figure(figsize=(10, 6)) # Setting the figure size for the plot
plt.bar(data['target'].value_counts().index, data['target'].value_counts().values) # Creating a bar plot for the target class distribution
plt.title('Bar Plot of Target Class Distribution') # Adding a title to the plot
plt.xlabel('Target Class') # Adding a label to the X-axis
plt.ylabel('Count') # Adding a label to the Y-axis
plt.xticks(ticks=[0, 1], labels=['Malignant', 'Benign']) # Setting the ticks and labels for the X-axis
plt.grid(True) # Enabling the grid for the plot
plt.show() # Displaying the plot
```

```
# Histogram

plt.figure(figsize=(10, 6)) # Setting the figure size for the plot

plt.hist(data['mean area'], bins=30, alpha=0.7) # Creating a histogram for the 'mean area' column with 30 bins

plt.title('Histogram of Mean Area') # Adding a title to the plot

plt.xlabel('Mean Area') # Adding a label to the X-axis

plt.ylabel('Frequency') # Adding a label to the Y-axis

plt.grid(True) # Enabling the grid for the plot

plt.show() # Displaying the plot


# Box Plot

plt.figure(figsize=(10, 6)) # Setting the figure size for the plot

plt.boxplot([data[data['target'] == 0]['mean radius'], data[data['target'] == 1]['mean radius']], labels=['Malignant', 'Benign']) # Creating a box plot for the 'mean radius' column, grouped by the target class

plt.title('Box Plot of Mean Radius by Target Class') # Adding a title to the plot

plt.xlabel('Target Class') # Adding a label to the X-axis

plt.ylabel('Mean Radius') # Adding a label to the Y-axis

plt.grid(True) # Enabling the grid for the plot

plt.show() # Displaying the plot
```

Program 2b

Problem Statement

Develop a Python program to analyze and visualize the Breast Cancer Wisconsin dataset using the seaborn library. The primary objective of this program is to generate a variety of plots that help illustrate the distribution, relationships, and patterns within the dataset's attributes.

```
import seaborn as sns  # Import Seaborn for advanced data visualization

import pandas as pd  # Import Pandas for data manipulation

import matplotlib.pyplot as plt  # Import Matplotlib for plotting

from sklearn.datasets import load_breast_cancer  # Import the Breast Cancer dataset from sklearn
```

```
# Load the Breast Cancer Wisconsin dataset
cancer = load_breast_cancer()  # Fetch the dataset from sklearn's built-in datasets

# Convert the dataset to a pandas DataFrame
data = pd.DataFrame(cancer.data, columns=cancer.feature_names)  # Create a DataFrame with feature names as columns
data['target'] = cancer.target  # Add the target column to the DataFrame, which contains the class labels

# Display a concise summary of the DataFrame
print(data.info()) # Displaying a concise summary of the DataFrame, including the number of entries, columns, non-null values, and data types

# Display the first few rows of the dataset
print(data.head(1)) # Displaying the first few rows of the DataFrame to get an initial look at the data

# Display basic statistics
print(data.describe()) # Displaying basic statistical details like mean, std deviation, min, and max values for each column

# Check for any missing values
print(data.isnull().sum()) # Checking for any missing values in the DataFrame

# Count Plot
plt.figure(figsize=(6, 4))  # Set the size of the figure for the plot
sns.countplot(x='target', data=data, palette='coolwarm')  # Create a count plot to visualize the number of Malignant vs. Benign cases
plt.title('Count Plot of Target Classes')  # Add a title to the plot
plt.xlabel('Target Class')  # Add a label to the X-axis
plt.ylabel('Count')  # Add a label to the Y-axis
plt.xticks(ticks=[0, 1], labels=['Malignant', 'Benign'])  # Set the ticks and labels for the X-axis
plt.show()  # Display the plot
```

```
# KDE Plot
plt.figure(figsize=(10, 6))  # Set the size of the figure for the plot
sns.kdeplot(data=data[data['target'] == 0]['mean radius'], shade=True, label='Malignant', color='r')  # KDE plot for 'mean radius' for Malignant cases
sns.kdeplot(data=data[data['target'] == 1]['mean radius'], shade=True, label='Benign', color='b')  # KDE plot for 'mean radius' for Benign cases
plt.title('KDE Plot of Mean Radius')  # Add a title to the plot
plt.xlabel('Mean Radius')  # Add a label to the X-axis
plt.ylabel('Density')  # Add a label to the Y-axis
plt.legend()  # Add a legend to the plot
plt.show()  # Display the plot

# Violin Plot
plt.figure(figsize=(10, 6))  # Set the size of the figure for the plot
sns.violinplot(x='target', y='mean radius', data=data, palette='coolwarm')  # Create a violin plot for 'mean radius' by target class
plt.title('Violin Plot of Mean Radius by Target Class')  # Add a title to the plot
plt.xlabel('Target Class')  # Add a label to the X-axis
plt.ylabel('Mean Radius')  # Add a label to the Y-axis
plt.xticks(ticks=[0, 1], labels=['Malignant', 'Benign'])  # Set the ticks and labels for the X-axis
plt.show()  # Display the plot

# Pair Plot
sns.pairplot(data, vars=['mean radius', 'mean texture', 'mean perimeter', 'mean area'], hue='target', palette='coolwarm')  # Create a pair plot for selected features, color-coded by target class
plt.title('Pair Plot')  # Add a title to the plot
plt.show()  # Display the plot

# Heatmap
plt.figure(figsize=(20, 20))  # Set the size of the figure for the plot
```

```
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm')  # Create a heatmap for the correlation matrix of features

plt.title('Correlation Heatmap')  # Add a title to the plot

plt.show()  # Display the plot
```