Practical No-1

Date of Conduction: **Date of Checking:**

Data Wrangling, I

Perform the following operations using Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.

2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site).

3. Load the Dataset into pandas data frame.

4. Data Preprocessing: check for missing values in the data using pandas insult(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.

6. Turn categorical variables into quantitative variables in Python. In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

Python Code:

# 1. Import all the required Python Libraries.

```
import pandas as pd

import numpy as np
```

# 2. Locate an open source data from the web.

```
# In this example, I'll use the Iris dataset available at UCI ML Repository.

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
```

# 3. Load the Dataset into pandas data frame.

```
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

iris_df = pd.read_csv(url, names=column_names)
```

```
# Display the first few rows of the dataset to verify the import.
print("First few rows of the Iris dataset:")
print(iris_df.head())

# 4. Data Preprocessing:
# Check for missing values using pandas info(), describe() functions.
print("\nInformation about the dataset:")
print(iris_df.info())

print("\nDescriptive statistics of the dataset:")
print(iris_df.describe())

# Variable Descriptions:
# - Sepal Length, Sepal Width, Petal Length, Petal Width: Numeric variables.
# - Class: Categorical variable representing the species of iris flowers.

# Check the dimensions of the data frame.
print("\nDimensions of the dataset (rows, columns):", iris_df.shape)

# 5. Data Formatting and Normalization:
# Summarize the types of variables by checking data types.
print("\nData Types of Variables:")
print(iris_df.dtypes)

# Ensure that numeric variables are in the correct data type.
# In this case, they are already in the correct data types (float64).

# 6. Turn categorical variables into quantitative variables.
# The 'class' variable is categorical; we can use one-hot encoding to convert it to quantitative.
iris_df = pd.get_dummies(iris_df, columns=['class'], drop_first=True)
```

```
# Display the updated dataframe.
print("\nUpdated DataFrame after one-hot encoding:")
print(iris_df.head())
```

Explanation:

- The code starts by importing necessary libraries, including Pandas for data manipulation and NumPy for numerical operations.
- The dataset URL is specified, and the read_csv function from Pandas is used to load the dataset into a Pandas DataFrame.
- The info() and describe() functions are used to obtain initial statistics and check for missing values.
- Variable descriptions are provided, and the dimensions of the DataFrame are printed.
- The data types of variables are displayed using dtypes.
- The 'class' variable is categorical, so one-hot encoding is applied using pd.get_dummies() to convert it into quantitative variables.
- The updated DataFrame is displayed.

Output:

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"

First few rows of the Iris dataset:

   sepal_length  sepal_width  petal_length  petal_width        class

0           5.1          3.5           1.4          0.2  Iris-setosa

1           4.9          3.0           1.4          0.2  Iris-setosa

2           4.7          3.2           1.3          0.2  Iris-setosa

3           4.6          3.1           1.5          0.2  Iris-setosa

4           5.0          3.6           1.4          0.2  Iris-setosa


Information about the dataset:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   class         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None

Descriptive statistics of the dataset:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Dimensions of the dataset (rows, columns): (150, 5)

Data Types of Variables:
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
class            object
```

```
dtype: object

Updated DataFrame after one-hot encoding:
   sepal_length  sepal_width  ...  class_Iris-versicolor  class_Iris-virginica
0           5.1          3.5  ...                  False                 False
1           4.9          3.0  ...                  False                 False
2           4.7          3.2  ...                  False                 False
3           4.6          3.1  ...                  False                 False
4           5.0          3.6  ...                  False                 False

[5 rows x 6 columns]

Process finished with exit code 0
```

Date :

Name & Signature of Instructor

Practical No-2

Date of Conduction: **Date of Checking:**

Data Wrangling II

Create an “Academic performance” dataset of students and perform the following operations using Python.

1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.

2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.

3. Apply data transformations on at least one of the variables.

The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution. Reason and document your approach properly.

Python Code:

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

# Set a random seed for reproducibility
np.random.seed(42)

# 1. Create the "Academic Performance" dataset
data = {
    'Student_ID': range(1, 101),
    'Math_Score': np.random.randint(50, 100, size=100),
    'English_Score': np.random.randint(40, 95, size=100),
    'Science_Score': np.random.randint(55, 98, size=100),
    'Attendance_Percentage': np.random.uniform(70, 100, size=100),
    'Study_Hours_Per_Day': np.random.uniform(1, 6, size=100),
}

academic_df = pd.DataFrame(data)

# Introduce missing values and inconsistencies for demonstration
academic_df.loc[10:20, 'Math_Score'] = np.nan
academic_df.loc[30:40, 'English_Score'] = np.nan
academic_df.loc[50:60, 'Science_Score'] = np.nan
academic_df.loc[70:80, 'Attendance_Percentage'] = np.nan

# Display first few rows of the dataset
print("First few rows of the Academic Performance dataset:")
print(academic_df.head())

# 1. Scan all variables for missing values and inconsistencies
# Use mean imputation for missing values and replace any negative values with NaN
academic_df.fillna(academic_df.mean(), inplace=True)
academic_df[academic_df < 0] = np.nan

# Display the updated dataset after handling missing values and inconsistencies
print("\nUpdated dataset after handling missing values and inconsistencies:")
print(academic_df.head())

# 2. Scan all numeric variables for outliers
# Use Z-score to identify and handle outliers
numeric_vars = ['Math_Score', 'English_Score', 'Science_Score', 'Attendance_Percentage',
'Study_Hours_Per_Day']

z_scores = (academic_df[numeric_vars] - academic_df[numeric_vars].mean()) /
academic_df[numeric_vars].std()
outliers = (z_scores > 3) | (z_scores < -3)

# Replace outliers with NaN
academic_df[outliers] = np.nan

# Display the dataset after handling outliers
print("\nDataset after handling outliers:")
print(academic_df.head())

# 3. Apply data transformations
# Log transformation on 'Study_Hours_Per_Day' to decrease skewness
academic_df['Log_Study_Hours'] = np.log1p(academic_df['Study_Hours_Per_Day'])

# Display the dataset after the log transformation
print("\nDataset after log transformation:")
print(academic_df.head())

# Visualize the distribution before and after the transformation
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(academic_df['Study_Hours_Per_Day'], kde=True)
plt.title('Study_Hours_Per_Day Distribution')

plt.subplot(1, 2, 2)
sns.histplot(academic_df['Log_Study_Hours'], kde=True)
plt.title('Log_Study_Hours Distribution')

plt.show()
```

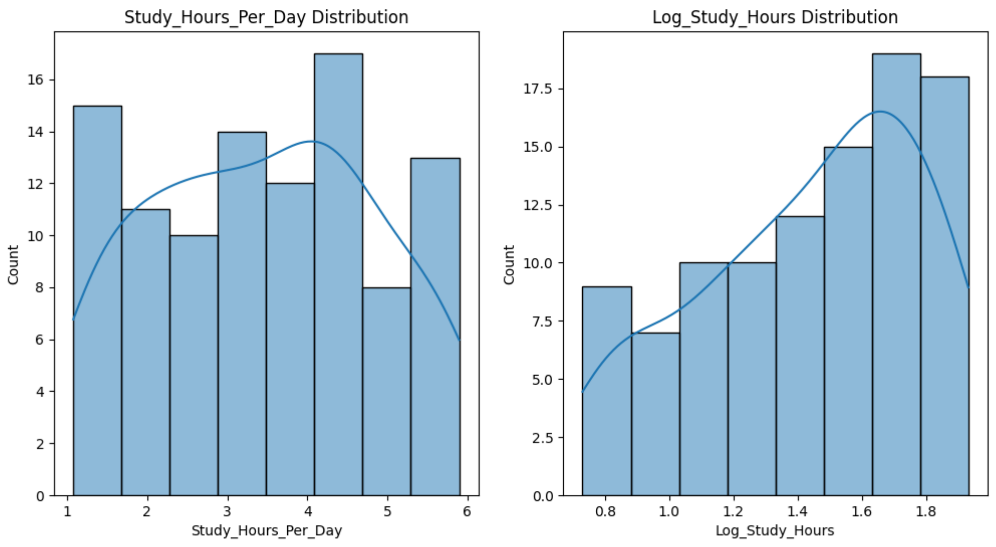
Explanation:

- The code starts by creating a sample "Academic Performance" dataset with variables such as Math_Score, English_Score, Science_Score, Attendance_Percentage, and Study_Hours_Per_Day.
- Some missing values and inconsistencies are introduced for demonstration purposes.
- Missing values and inconsistencies are handled using mean imputation for missing values and replacing negative values with NaN.
- Outliers are identified using Z-scores, and extreme values are replaced with NaN.
- A log transformation is applied to the 'Study_Hours_Per_Day' variable to decrease skewness and convert the distribution into a more normal shape.
- The code includes visualizations to compare the distribution before and after the log transformation.

Output:

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"

First few rows of the Academic Performance dataset:

   Student_ID  Math_Score  ...  Attendance_Percentage  Study_Hours_Per_Day
0           1        88.0  ...              81.168483             5.847684
1           2        78.0  ...              98.204003             4.572976
2           3        64.0  ...              99.209915             1.205338
3           4        92.0  ...              78.517629             2.994105
4           5        57.0  ...              79.160916             3.167604

[5 rows x 6 columns]

Updated dataset after handling missing values and inconsistencies:

   Student_ID  Math_Score  ...  Attendance_Percentage  Study_Hours_Per_Day
0           1        88.0  ...              81.168483             5.847684
1           2        78.0  ...              98.204003             4.572976
2           3        64.0  ...              99.209915             1.205338
3           4        92.0  ...              78.517629             2.994105
4           5        57.0  ...              79.160916             3.167604
```

```
[5 rows x 6 columns]
Dataset after handling outliers:
   Student_ID  Math_Score  ...  Attendance_Percentage  Study_Hours_Per_Day
0           1        88.0  ...              81.168483             5.847684
1           2        78.0  ...              98.204003             4.572976
2           3        64.0  ...              99.209915             1.205338
3           4        92.0  ...              78.517629             2.994105
4           5        57.0  ...              79.160916             3.167604
[5 rows x 6 columns]
Dataset after log transformation:
   Student_ID  Math_Score  ...  Study_Hours_Per_Day  Log_Study_Hours
0           1        88.0  ...             5.847684         1.923911
1           2        78.0  ...             4.572976         1.717929
2           3        64.0  ...             1.205338         0.790881
3           4        92.0  ...             2.994105         1.384819
4           5        57.0  ...             3.167604         1.427341

[5 rows x 7 columns]
```



Name & Signature of Instructor

Practical No-3

Date of Conduction: **Date of Checking:**

Descriptive Statistics - Measures of Central Tendency and variability

Perform the following operations on any open source dataset (e.g., data.csv)

1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.

Python Code:

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

# Load the Titanic dataset (you can replace this with your dataset)
titanic_df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv')

# Display first few rows of the dataset
print("First few rows of the Titanic dataset:")
print(titanic_df.head())

# 1. Provide summary statistics grouped by a categorical variable
# Let's use the 'Pclass' (passenger class) as the categorical variable and 'Age' as the quantitative variable
grouped_stats = titanic_df.groupby('Pclass')['Age'].describe()

# Display the summary statistics
print("\nSummary statistics of Age grouped by Pclass:")
print(grouped_stats)

# 2. Create a list that contains a numeric value for each response to the categorical variable
# In this case, create a list of mean ages for each passenger class
mean_age_by_class = titanic_df.groupby('Pclass')['Age'].mean().tolist()

# Display the list of mean ages for each passenger class
print("\nMean Age for each Passenger Class:")
print(mean_age_by_class)
```

Explanation:

- The code loads the Titanic dataset using pd.read_csv.
- It then displays the first few rows of the dataset to provide an overview.
- The dataset is grouped by the 'Pclass' (passenger class) variable, and summary statistics for the 'Age' variable within each group are calculated using the describe() function.
- The resulting grouped summary statistics are displayed.
- Additionally, a list containing the mean age for each passenger class is created.

OUTPUT:

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe"
"C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"
```

First few rows of the Titanic dataset:

```
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S

[5 rows x 12 columns]
```

Summary statistics of Age grouped by Pclass:

```
        count       mean        std   min   25%   50%   75%   max
Pclass
1       186.0  38.233441  14.802856  0.92  27.0  37.0  49.0  80.0
2       173.0  29.877630  14.001077  0.67  23.0  29.0  36.0  70.0
3       355.0  25.140620  12.495398  0.42  18.0  24.0  32.0  74.0
```

Mean Age for each Passenger Class:

```
[38.233440860215055, 29.87763005780347, 25.14061971830986]

Process finished with exit code 0
```

2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of ‘Iris-setosa’, ‘Iris-versicolor’ and ‘Iris-versicolor’ of iris.csv dataset. Provide the codes with outputs and explain everything that you do in this step.

Python Code:

```python
# Import necessary libraries
import seaborn as sns
import pandas as pd

# Load the Iris dataset
iris = sns.load_dataset('iris')

# Display the first few rows of the dataset
print("First few rows of the Iris dataset:")
print(iris.head())

# 1. Display basic statistical details for 'Iris-setosa'
setosa_stats = iris[iris['species'] == 'setosa'].describe()

# Display the statistical details for 'Iris-setosa'
print("\nStatistical details for 'Iris-setosa':")
print(setosa_stats)

# 2. Display basic statistical details for 'Iris-versicolor'
versicolor_stats = iris[iris['species'] == 'versicolor'].describe()

# Display the statistical details for 'Iris-versicolor'
print("\nStatistical details for 'Iris-versicolor':")
print(versicolor_stats)

# 3. Display basic statistical details for 'Iris-virginica'
virginica_stats = iris[iris['species'] == 'virginica'].describe()

# Display the statistical details for 'Iris-virginica'
print("\nStatistical details for 'Iris-virginica':")
print(virginica_stats)
```

Explanation:

- The code loads the Iris dataset using sns.load_dataset('iris') from the Seaborn library.
- The first few rows of the dataset are displayed to provide an overview.
- The dataset is then filtered for each species ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica') separately.
- The describe() function is used to calculate basic statistical details for each species, including percentiles, mean, standard deviation, etc.
- The statistical details for each species are displayed.

OUTPUT:

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe"
"C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"
```

First few rows of the Iris dataset:

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Statistical details for 'Iris-setosa':

```
       sepal_length  sepal_width  petal_length  petal_width
count      50.00000    50.000000     50.000000    50.000000
mean        5.00600     3.428000      1.462000     0.246000
std         0.35249     0.379064      0.173664     0.105386
min         4.30000     2.300000      1.000000     0.100000
25%         4.80000     3.200000      1.400000     0.200000
50%         5.00000     3.400000      1.500000     0.200000
75%         5.20000     3.675000      1.575000     0.300000
max         5.80000     4.400000      1.900000     0.600000
```

Statistical details for 'Iris-versicolor':

```
       sepal_length  sepal_width  petal_length  petal_width
count     50.000000    50.000000     50.000000    50.000000
mean       5.936000     2.770000      4.260000     1.326000
std        0.516171     0.313798      0.469911     0.197753
min        4.900000     2.000000      3.000000     1.000000
25%        5.600000     2.525000      4.000000     1.200000
50%        5.900000     2.800000      4.350000     1.300000
```

```
75%       6.300000    3.000000     4.600000    1.500000
max       7.000000    3.400000     5.100000    1.800000
```

Statistical details for 'Iris-virginica':

```
       sepal_length  sepal_width  petal_length  petal_width
count      50.00000    50.000000     50.000000     50.00000
mean        6.58800     2.974000      5.552000      2.02600
std         0.63588     0.322497      0.551895      0.27465
min         4.90000     2.200000      4.500000      1.40000
25%         6.22500     2.800000      5.100000      1.80000
50%         6.50000     3.000000      5.550000      2.00000
75%         6.90000     3.175000      5.875000      2.30000
max         7.90000     3.800000      6.900000      2.50000

Process finished with exit code 0
```

Name & Signature of Instructor

Practical No-4

Date of Conduction: **Date of Checking:**

Data Analytics I

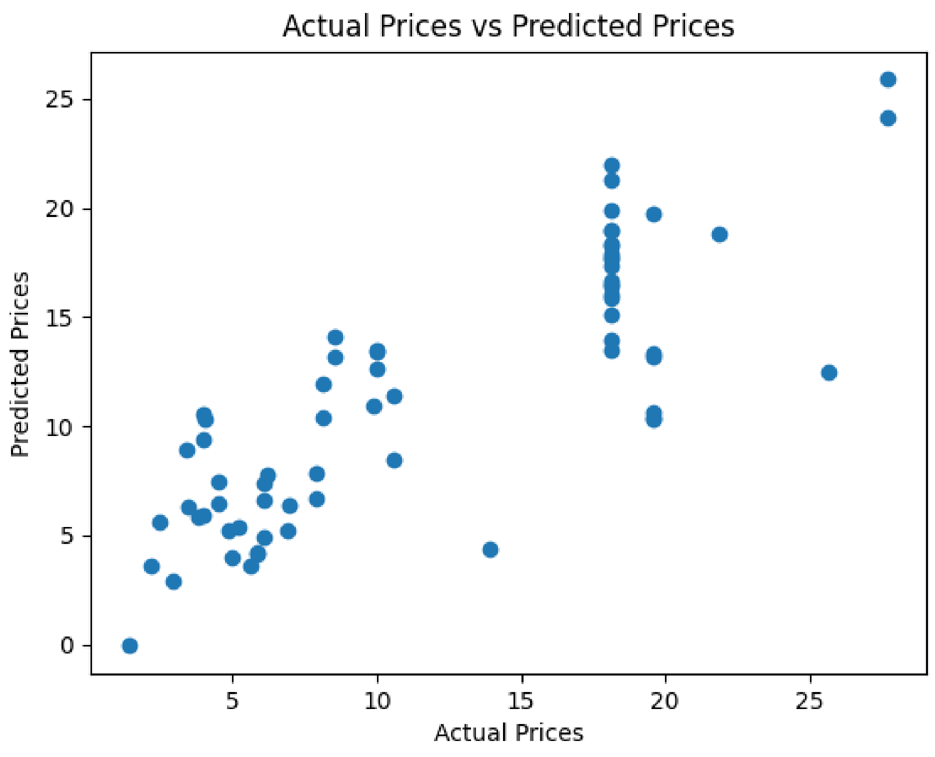
Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing).
The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.
The objective is to predict the value of prices of the house using the given features.

Python Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the Boston Housing Dataset
boston_data = pd.read_csv('train.csv')

# Explore the dataset
print(boston_data.head())

# Separate features (X) and target variable (y)
X = boston_data.drop(['ID', 'indus'], axis=1)
y = boston_data['indus']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Plotting predicted vs actual prices
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual Prices vs Predicted Prices')
plt.show()
```

OUTPUT

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe"
"C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"
   ID    crim   zn  indus  chas  ...  tax  ptratio   black  lstat  medv
0   1  0.00632  18.0   2.31     0  ...  296     15.3  396.90   4.98  24.0
1   2  0.02731   0.0   7.07     0  ...  242     17.8  396.90   9.14  21.6
2   4  0.03237   0.0   2.18     0  ...  222     18.7  394.63   2.94  33.4
3   5  0.06905   0.0   2.18     0  ...  222     18.7  396.90   5.33  36.2
4   7  0.08829  12.5   7.87     0  ...  311     15.2  395.60  12.43  22.9

[5 rows x 15 columns]
Mean Squared Error: 15.3919055697973
R-squared: 0.6933129963581055

Process finished with exit code 0
```



Name & Signature of Instructor

Practical No-5

Date of Conduction: **Date of Checking:**

Data Analytics II

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.

2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
# Load the Social_Network_Ads dataset
social_data = pd.read_csv('Social_Network_Ads.csv')

# Check for NaN values
print(social_data.head(25))

# Separate features (X) and target variable (y)
X = social_data[['Age', 'EstimatedSalary']]
y = social_data['Purchased']

# Handle NaN values by imputing the mean
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a Logistic Regression model
model = LogisticRegression(random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print('Confusion Matrix:')
print(conf_matrix)

# Compute Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Extract values from the confusion matrix
TN, FP, FN, TP = conf_matrix.ravel()

# Compute Performance Metrics
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the results
print("Confusion Matrix:")
print(conf_matrix)
print("\nTrue Positive (TP):", TP)
print("False Positive (FP):", FP)
print("True Negative (TN):", TN)
print("False Negative (FN):", FN)
print("\nAccuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='EstimatedSalary', hue='Purchased', data=social_data, palette='viridis')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')

# Plot the decision boundary
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='EstimatedSalary', hue='Purchased', data=social_data, palette='viridis')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')

# Plotting decision boundary
h = 0.5
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

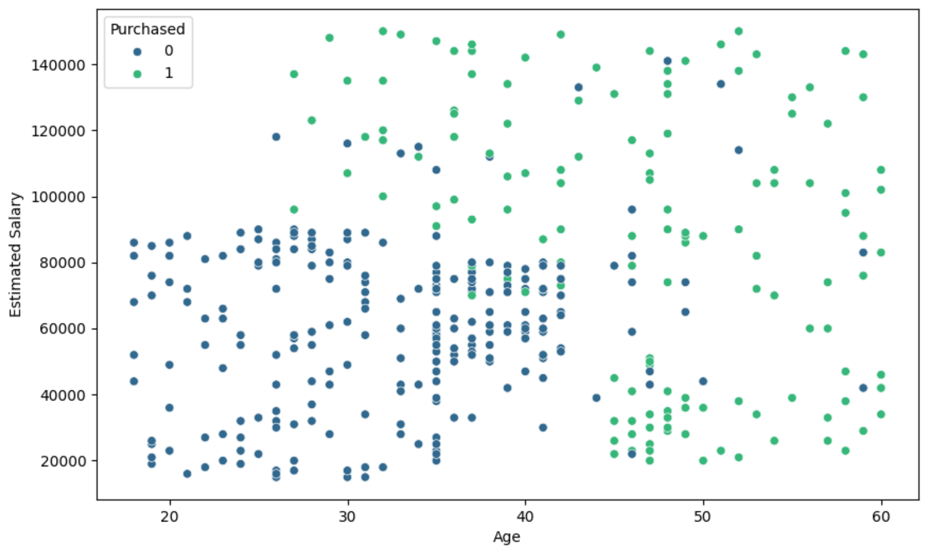
OUTPUT:

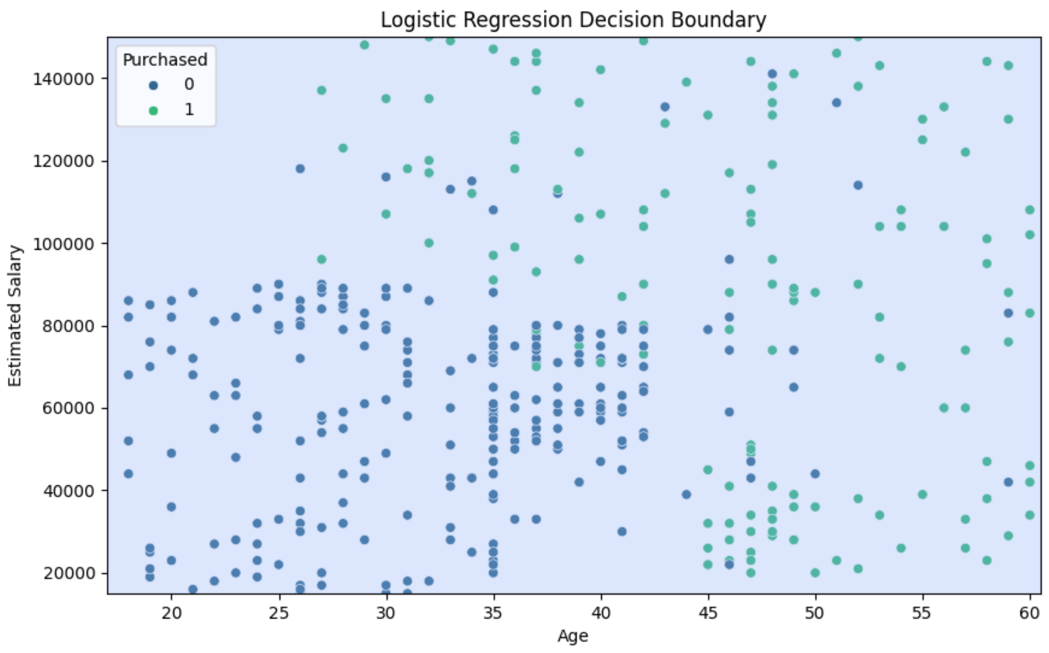
```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe"
"C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"
    Age  EstimatedSalary  Purchased
0    19           19000          0
1    35           20000          0
2    26           43000          0
3    27           57000          0
4    19           76000          0
5    27           58000          0
6    27           84000          0
7    32          150000          1
8    25           33000          0
9    35           65000          0
10   26           80000          0
11   26           52000          0
12   20           86000          0
13   32           18000          0
14   18           82000          0
15   29           80000          0
16   47           25000          1
17   45           26000          1
18   46           28000          1
19   48           29000          1
20   45           22000          1
21   47           49000          1
22   48           41000          1
23   45           22000          1
24   46           23000          1
Accuracy: 0.8625
```

```
Confusion Matrix:
[[50  2]
 [ 9 19]]
```

```
Confusion Matrix:
[[50  2]
 [ 9 19]]
```

```
True Positive (TP): 19
False Positive (FP): 2
True Negative (TN): 50
False Negative (FN): 9

Accuracy: 0.8625
Error Rate: 0.13749999999999996
Precision: 0.9047619047619048
Recall: 0.6785714285714286
F1 Score: 0.7755102040816326
```





Name & Signature of Instructor

Practical No-6

Date of Conduction: **Date of Checking:**

Data Analytics III

1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

Python Code

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Load the Iris dataset
iris_data = pd.read_csv('iris.csv')

# Separate features (X) and target variable (y)
X = iris_data.iloc[:, :-1]
y = iris_data.iloc[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Naïve Bayes model (Gaussian Naïve Bayes for continuous features)
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Compute Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Compute Performance Metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')  # weighted precision for multi-class
recall = recall_score(y_test, y_pred, average='weighted')  # weighted recall for multi-class
f1 = f1_score(y_test, y_pred, average='weighted')  # weighted F1 score for multi-class

# Print the results
print("Confusion Matrix:")
print(conf_matrix)
print("\nAccuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

OUTPUT

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe"
"C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Process finished with exit code 0
```

Name & Signature of Instructor

Practical No-7

Date of Conduction: **Date of Checking:**

Text Analytics
1. Extract Sample document and apply following document preprocessing methods:

Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.
2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

Theory:

Tokenization:

Definition: Tokenization is the process of breaking a text into individual units, known as tokens. Tokens can be words, phrases, symbols, or other meaningful elements.
Example: For the sentence "Natural language processing is interesting," tokenization would result in the tokens: ["Natural", "language", "processing", "is", "interesting"].

POS Tagging (Part-of-Speech Tagging):

Definition: POS tagging involves assigning a part-of-speech category (such as noun, verb, adjective, etc.) to each word in a sentence.
Example: For the sentence "She is reading a book," POS tagging would identify the parts of speech: [("She", "PRP"), ("is", "VBZ"), ("reading", "VBG"), ("a", "DT"), ("book", "NN")].

Stop Words Removal:

Definition: Stop words are common words (e.g., "and," "the," "is") that are often removed from text during preprocessing. These words don't contribute much to the meaning of the text and are often excluded to focus on more meaningful content.
Example: For the sentence "The quick brown fox jumps over the lazy dog," stop words removal might result in: ["quick", "brown", "fox", "jumps", "lazy", "dog"].

Stemming:

Definition: Stemming is the process of reducing words to their root or base form. It involves removing suffixes from words to obtain a common base form.
Example: For the words "running," "runner," and "ran," stemming might produce: ["run", "run", "run"].

Lemmatization:

Definition: Lemmatization is similar to stemming, but it involves reducing words to their base or dictionary form (lemma). It considers the meaning of the word and applies a morphological analysis to obtain the base form.

Example: For the words "running," "runner," and "ran," lemmatization might produce: ["run", "runner", "run"].

These preprocessing techniques are essential in natural language processing and text analytics to standardize and simplify textual data for further analysis. The choice of which methods to apply depends on the specific requirements of the task and the nature of the text data being processed.

Python Code:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.probability import FreqDist
from nltk.tag import pos_tag
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

# Sample document
sample_document = """Natural language processing (NLP) is a subfield of artificial intelligence (AI) that
focuses on the interaction between computers and humans using natural language. It enables computers
to understand, interpret, and generate human-like text. NLP involves various tasks, such as tokenization,
part-of-speech tagging, stop words removal, stemming, and lemmatization."""

# Tokenization
tokens = word_tokenize(sample_document)

# POS Tagging
pos_tags = pos_tag(tokens)

# Stop Words Removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

# Stemming
porter_stemmer = PorterStemmer()
stemmed_tokens = [porter_stemmer.stem(word) for word in filtered_tokens]

# Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

# TF-IDF Representation
documents = [sample_document]
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
feature_names = tfidf_vectorizer.get_feature_names_out()

# Display Results
print("Original Document:\n", sample_document, "\n")
print("Tokenization:\n", tokens, "\n")
print("POS Tagging:\n", pos_tags, "\n")
print("Stop Words Removal:\n", filtered_tokens, "\n")
print("Stemming:\n", stemmed_tokens, "\n")
print("Lemmatization:\n", lemmatized_tokens, "\n")
print("TF-IDF Representation:\n", tfidf_matrix.toarray(), "\n")
print("Feature Names:\n", feature_names)
```

OUTPUT:

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe"
"C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"
[nltk_data] Downloading package punkt to C:\Users\Ram Kumar
[nltk_data]     Solanki\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package stopwords to C:\Users\Ram Kumar
[nltk_data]     Solanki\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Ram Kumar
[nltk_data]     Solanki\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to C:\Users\Ram Kumar
[nltk_data]     Solanki\AppData\Roaming\nltk_data...
Original Document:
 Natural language processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans using natural language. It enables computers to understand, interpret, and generate human-like text. NLP involves various tasks, such as tokenization, part-of-speech tagging, stop words removal, stemming, and lemmatization.

Tokenization:
 ['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'a', 'subfield', 'of', 'artificial', 'intelligence', '(', 'AI', ')', 'that', 'focuses', 'on', 'the', 'interaction', 'between', 'computers', 'and', 'humans', 'using', 'natural', 'language', '.', 'It', 'enables', 'computers', 'to', 'understand', ',', 'interpret', ',', 'and', 'generate', 'human-like', 'text', '.', 'NLP', 'involves', 'various', 'tasks', ',', 'such', 'as', 'tokenization', ',', 'part-of-speech', 'tagging', ',', 'stop', 'words', 'removal', ',', 'stemming', ',', 'and', 'lemmatization', '.']

POS Tagging:
 [('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('(', '('), ('NLP', 'NNP'), (')', ')'), ('is', 'VBZ'), ('a', 'DT'), ('subfield', 'NN'), ('of', 'IN'), ('artificial', 'JJ'), ('intelligence', 'NN'), ('(', '('), ('AI', 'NNP'), (')', ')'), ('that', 'WDT'), ('focuses', 'VBZ'), ('on', 'IN'), ('the', 'DT'), ('interaction', 'NN'), ('between', 'IN'), ('computers', 'NNS'), ('and', 'CC'), ('humans', 'NNS'), ('using', 'VBG'), ('natural', 'JJ'), ('language', 'NN'), ('.', '.'), ('It', 'PRP'), ('enables', 'VBZ'), ('computers', 'NNS'), ('to', 'TO'), ('understand', 'VB'), (',', ','), ('interpret', 'VB'), (',', ','), ('and', 'CC'), ('generate', 'VB'), ('human-like', 'JJ'), ('text', 'NN'), ('.', '.'), ('NLP', 'NNP'), ('involves', 'VBZ'), ('various', 'JJ'), ('tasks', 'NNS'), (',', ','), ('such', 'JJ'), ('as', 'IN'), ('tokenization', 'NN'), (',', ','), ('part-of-speech', 'JJ'), ('tagging', 'NN'), (',', ','), ('stop', 'VB'), ('words', 'NNS'), ('removal', 'JJ'), (',', ','), ('stemming', 'VBG'), (',', ','), ('and', 'CC'), ('lemmatization', 'NN'), ('.', '.')]
```

```
Stop Words Removal:
 ['Natural', 'language', 'processing', '(', 'NLP', ')', 'subfield', 'artificial', 'intelligence', '(', 'AI', ')', 'focuses', 'interaction', 'computers', 'humans', 'using', 'natural', 'language', '.', 'enables', 'computers', 'understand', ',', 'interpret', ',', 'generate', 'human-like', 'text', '.', 'NLP', 'involves', 'various', 'tasks', ',', 'tokenization', ',', 'part-of-speech', 'tagging', ',', 'stop', 'words', 'removal', ',', 'stemming', ',', 'lemmatization', '.']

Stemming:
 ['natur', 'languag', 'process', '(', 'nlp', ')', 'subfield', 'artifici', 'intellig', '(', 'ai', ')', 'focus', 'interact', 'comput', 'human', 'use', 'natur', 'languag', '.', 'enabl', 'comput', 'understand', ',', 'interpret', ',', 'gener', 'human-lik', 'text', '.', 'nlp', 'involv', 'variou', 'task', ',', 'token', ',', 'part-of-speech', 'tag', ',', 'stop', 'word', 'remov', ',', 'stem', ',', 'lemmat', '.']

Lemmatization:
 ['Natural', 'language', 'processing', '(', 'NLP', ')', 'subfield', 'artificial', 'intelligence', '(', 'AI', ')', 'focus', 'interaction', 'computer', 'human', 'using', 'natural', 'language', '.', 'enables', 'computer', 'understand', ',', 'interpret', ',', 'generate', 'human-like', 'text', '.', 'NLP', 'involves', 'various', 'task', ',', 'tokenization', ',', 'part-of-speech', 'tagging', ',', 'stop', 'word', 'removal', ',', 'stemming', ',', 'lemmatization', '.']

TF-IDF Representation:
 [[0.12309149 0.36927447 0.12309149 0.12309149 0.12309149 0.24618298
  0.12309149 0.12309149 0.12309149 0.12309149 0.12309149 0.12309149
  0.12309149 0.12309149 0.12309149 0.12309149 0.12309149 0.24618298
  0.12309149 0.12309149 0.24618298 0.24618298 0.24618298 0.12309149
  0.12309149 0.12309149 0.12309149 0.12309149 0.12309149 0.12309149
  0.12309149 0.12309149 0.12309149 0.12309149 0.12309149 0.12309149
  0.12309149 0.12309149 0.12309149 0.12309149 0.12309149 0.12309149
  0.12309149]]

Feature Names:
 ['ai' 'and' 'artificial' 'as' 'between' 'computers' 'enables' 'focuses'
 'generate' 'human' 'humans' 'intelligence' 'interaction' 'interpret'
 'involves' 'is' 'it' 'language' 'lemmatization' 'like' 'natural' 'nlp'
 'of' 'on' 'part' 'processing' 'removal' 'speech' 'stemming' 'stop'
 'subfield' 'such' 'tagging' 'tasks' 'text' 'that' 'the' 'to'
 'tokenization' 'understand' 'using' 'various' 'words']

Process finished with exit code 0
```

Name & Signature of Instructor

Practical No-8

Date of Conduction: **Date of Checking:**

Data Visualization I

1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.
2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

Python Code

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Titanic dataset
titanic = sns.load_dataset('titanic')

# 1. Visualizing patterns in the data
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

# Use Seaborn's countplot to visualize the number of passengers in each class
sns.countplot(x='class', hue='survived', data=titanic, palette='Set1')
plt.title('Survival Count by Passenger Class')
plt.show()

# 2. Plotting a histogram for the distribution of ticket prices (fare)
plt.figure(figsize=(12, 6))

# Use Seaborn's histogram to visualize the distribution of ticket prices
sns.histplot(titanic['fare'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of Ticket Prices (Fare)')
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.show()
```

Explanation:

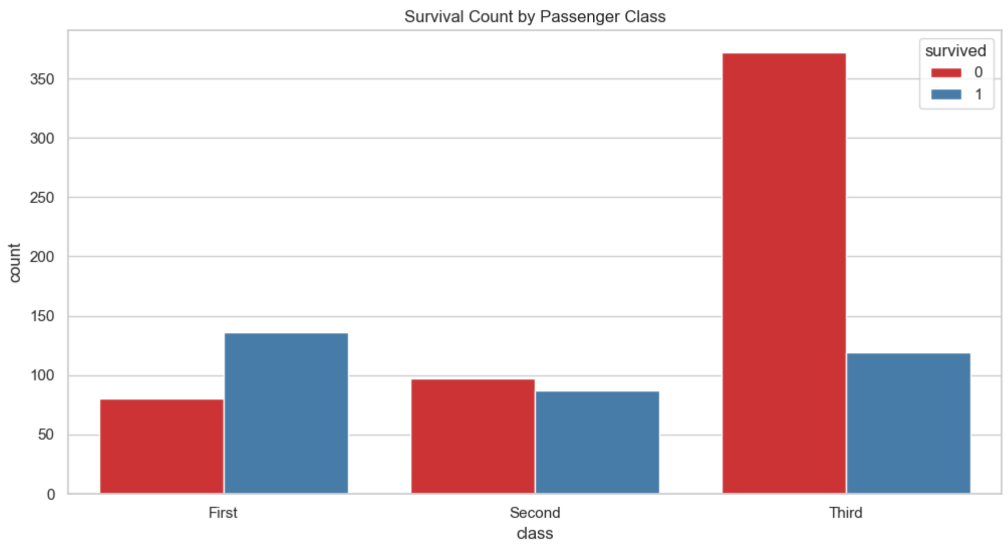
Visualizing patterns in the data:

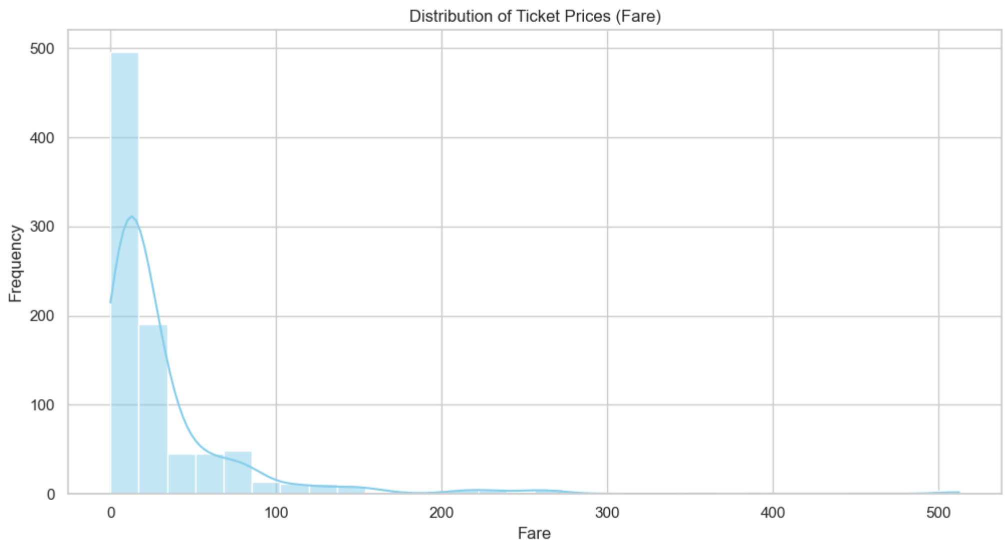
- This part uses Seaborn's countplot to visualize the number of passengers in each class (1st, 2nd, 3rd) and their survival status. The color hue represents survival (survived or not survived).
- Plotting a histogram for ticket prices (fare):

- This part uses Seaborn's histplot to plot a histogram for the distribution of ticket prices (fare column) with 30 bins and a kernel density estimate (kde) for a smooth representation of the distribution.

- Make sure to have Seaborn and Matplotlib installed (pip install seaborn matplotlib) before running the code. You can adjust the parameters as needed based on your preferences and analysis goals.

OUTPUT:





Name & Signature of Instructor

Practical No-9

Date of Conduction: **Date of Checking:**

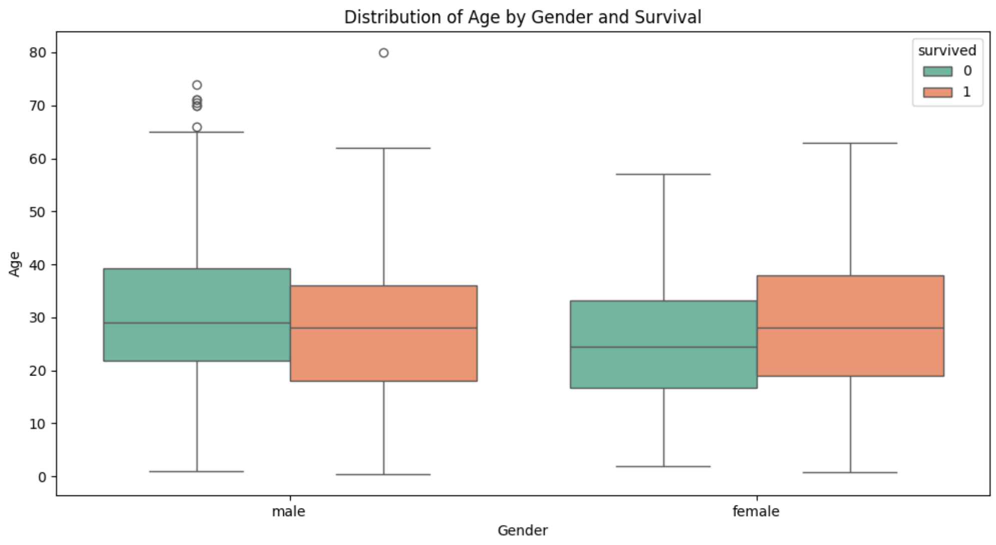
Data Visualization II

1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age')
2. Write observations on the inference from the above statistics.

Python Code:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Titanic dataset
titanic = sns.load_dataset('titanic')

# Plotting a box plot for distribution of age with respect to each gender and survival
plt.figure(figsize=(12, 6))
sns.boxplot(x='sex', y='age', hue='survived', data=titanic, palette='Set2')
plt.title('Distribution of Age by Gender and Survival')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.show()
```

OOTPUT:



Observations:

- The box plot provides insights into the distribution of age with respect to each gender and survival status.
- For both genders, the boxes represent the interquartile range (IQR), and the line inside the box represents the median age.
- Outliers are shown as individual points beyond the whiskers of the box.
- The color hue represents survival status (survived or not survived).
- You can observe the variation in age distribution between males and females and how it relates to their survival.

Name & Signature of Instructor

Practical No-10

Date of Conduction: **Date of Checking:**

Data Visualization III

Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as:

1. List down the features and their types (e.g., numeric, nominal) available in the dataset.
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
3. Create a box plot for each feature in the dataset.
4. Compare distributions and identify outliers.

Python Code:

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

# Download and load the Iris dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
column_names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
iris = pd.read_csv(url, header=None, names=column_names)

# 1. List down the features and their types
feature_types = iris.dtypes
print("Features and their types:")
print(feature_types)

# 2. Create a histogram for each feature
plt.figure(figsize=(12, 8))
iris.drop("class", axis=1).hist(edgecolor="black", linewidth=1.2, bins=20, figsize=(12, 8))
plt.suptitle("Histograms for Each Feature", y=0.92)
plt.show()

# 3. Create a box plot for each feature
plt.figure(figsize=(12, 8))
sns.boxplot(data=iris.drop("class", axis=1), palette="Set2")
plt.title("Box Plots for Each Feature")
plt.show()

# 4. Compare distributions and identify outliers
plt.figure(figsize=(12, 8))
sns.boxplot(x="class", y="sepal_length", data=iris, hue="class", palette="Set2", legend=False)
plt.title("Box Plot for Sepal Length by Class")
plt.show()
```

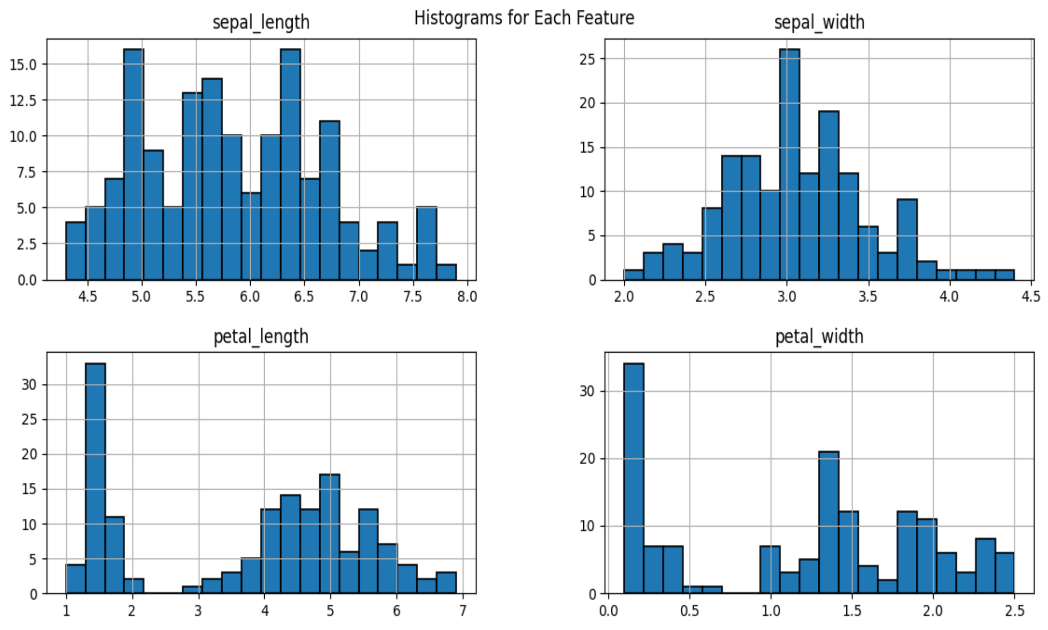
Explanation:

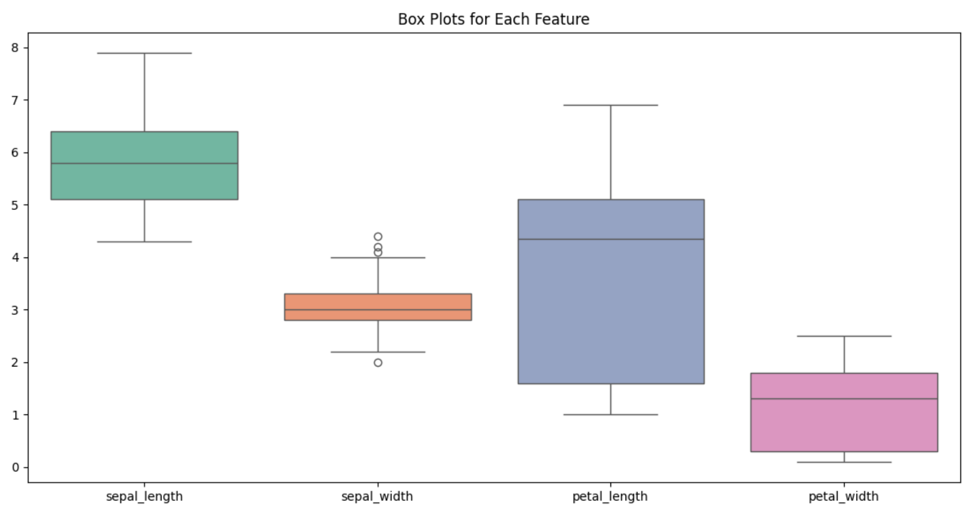
- The code uses the pandas library to load the Iris dataset from the given URL and names the columns.
- It then prints the types of features in the dataset (numeric).
- A histogram is created for each feature to illustrate their distributions.
- Box plots are generated for each feature to visually represent the distribution, median, and potential outliers.

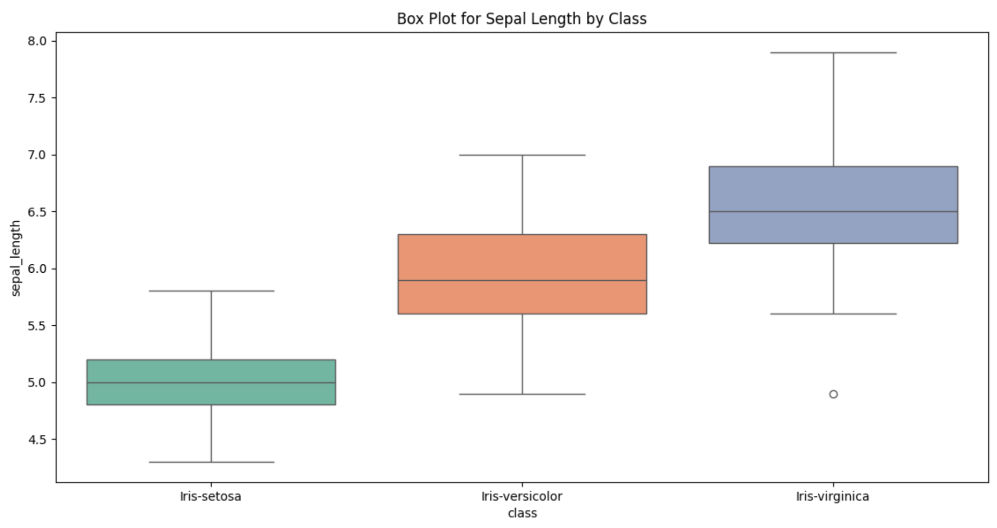
- A box plot for the Sepal Length is created for each class to compare distributions between classes.

OUTPUT:

```
"C:\Users\Ram Kumar Solanki\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\Ram Kumar Solanki\PycharmProjects\MBA_BFS\main.py"

Features and their types:

sepal_length    float64

sepal_width     float64

petal_length    float64

petal_width     float64

class           object

dtype: object


Process finished with exit code 0
```







Name & Signature of Instructor