Practical No.1

```
# Import all the required Python Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset into a pandas DataFrame
df = pd.read_csv("StudentsPerformance.csv")

# Display the first 5 rows of the dataset
print("First 5 rows of the dataset:")
print(df.head())

# Check the dimensions of the dataset
print("\nDataset Shape:", df.shape)

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Descriptive statistics of the dataset
print("\nDescriptive Statistics:")
print(df.describe(include='all'))

# Display data types of all columns
print("\nData Types of each column:")
print(df.dtypes)

# Convert data types if necessary (example shown)
```

```
df['math score'] = df['math score'].astype(float)
df['reading score'] = df['reading score'].astype(float)
df['writing score'] = df['writing score'].astype(float)

# Convert categorical variables into numerical using one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)

# Display the encoded dataframe
print("\nData after One-Hot Encoding:")
print(df_encoded.head())
```

Practical 2

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Step 1: Create a synthetic academic performance dataset
np.random.seed(42)
data = {
    'Student_ID': range(1, 101),
    'Gender': np.random.choice(['Male', 'Female'], 100),
    'Math_Score': np.append(np.random.normal(70, 10, 95), [150, -10, 200, 105, 2]),
    'Reading_Score': np.random.normal(65, 12, 100),
    'Writing_Score': np.random.normal(68, 15, 100),
    'Study_Hours': np.append(np.random.normal(10, 3, 95), [30, 35, 40, 1, 0]),
    'Parental_Education_Level': np.random.choice(['High School', 'Bachelor', 'Master'], 100)
}
df = pd.DataFrame(data)

# Introduce some missing values
df.loc[[5, 20, 55], 'Math_Score'] = np.nan
df.loc[[3, 25], 'Parental_Education_Level'] = np.nan

# --- Step 2: Handling Missing Values ---
print("\nMissing Values:\n", df.isnull().sum())

# For numerical: fill missing values with median
df['Math_Score'].fillna(df['Math_Score'].median(), inplace=True)
```

```python
# For categorical: fill missing values with mode
df['Parental_Education_Level'].fillna(df['Parental_Education_Level'].mode()[0], inplace=True)

# Verify
print("\nAfter Handling Missing Values:\n", df.isnull().sum())

# --- Step 3: Detecting Outliers in Numeric Variables ---
numeric_cols = ['Math_Score', 'Reading_Score', 'Writing_Score', 'Study_Hours']

# Boxplots for visualization
for col in numeric_cols:
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()

# Use Z-score to detect outliers
z_scores = np.abs(stats.zscore(df[numeric_cols]))
outliers = (z_scores > 3)
print("\nOutliers Detected:\n", outliers.sum())

# Remove outliers (Optional: we can also cap or impute)
df_no_outliers = df[(z_scores < 3).all(axis=1)]
print("\nShape after removing outliers:", df_no_outliers.shape)

# --- Step 4: Data Transformation ---
# Reason: Study_Hours is right-skewed. We'll apply log transformation to normalize it.
print("\nSkewness before transformation:", df_no_outliers['Study_Hours'].skew())

# Add 1 to avoid log(0)
df_no_outliers['Log_Study_Hours'] = np.log1p(df_no_outliers['Study_Hours'])
```

```
# Compare distributions
sns.histplot(df_no_outliers['Study_Hours'], kde=True)
plt.title("Original Study Hours Distribution")
plt.show()

sns.histplot(df_no_outliers['Log_Study_Hours'], kde=True)
plt.title("Log-Transformed Study Hours Distribution")
plt.show()

print("\nSkewness after transformation:", df_no_outliers['Log_Study_Hours'].skew())

# Final dataset preview
print("\nTransformed Dataset Head:\n", df_no_outliers.head())
```

Practical 3

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

# Step 1: Load the iris dataset
# You can replace 'iris.csv' with the path to your CSV file
df = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

# Step 2: Grouping numerical variables by a categorical variable (species)
grouped = df.groupby('species')

# Step 3: Summary statistics (mean, median, min, max, std) for each group
summary_stats = grouped.agg(['mean', 'median', 'min', 'max', 'std'])

print("=== Summary Statistics Grouped by Species ===\n")
print(summary_stats)

# Step 4: Create a list of numeric values for each response to the categorical variable
# For example: list of sepal_length values for each species
grouped_lists = grouped['sepal_length'].apply(list)

print("\n=== Sepal Length Lists by Species ===")
for species, values in grouped_lists.items():
    print(f"{species}: {values}")

# Step 5: Display basic statistical details like percentile, mean, std for each species
print("\n=== Detailed Statistical Description by Species ===")
for species in df['species'].unique():
    print(f"\n--- Statistics for {species} ---")
    species_data = df[df['species'] == species]
```

```
    desc = species_data.describe(percentiles=[.25, .5, .75])
    print(desc)

# Optional: You can save the summary stats to a CSV file
# summary_stats.to_csv("iris_summary_statistics.csv")
```

Practical 4

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.datasets import load_boston
import warnings
warnings.filterwarnings('ignore')
# Load the Boston Housing Dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target
# Display the first few rows
print("First 5 Rows:\n", df.head())
# Check for missing values
print("\nMissing values:\n", df.isnull().sum())

# Dataset shape and description
print("\nShape:", df.shape)
print("\nDescription:\n", df.describe())

# Correlation matrix heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Matrix")
plt.show()
```

```
# Feature and target variable selection
X = df.drop('PRICE', axis=1)
y = df['PRICE']

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

# Build the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Compare actual vs predicted prices
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("\nActual vs Predicted:\n", comparison.head())

# Evaluation metrics
print("\nMean Absolute Error:", metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R^2 Score:", metrics.r2_score(y_test, y_pred))

# Visualization - Actual vs Predicted
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Home Prices")
plt.show()
```

Practical 5

```
# Import necessary libraries
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.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score

# Step 1: Load the dataset
# Make sure you have the dataset in the correct path (Social_Network_Ads.csv)
df = pd.read_csv("Social_Network_Ads.csv")

# Step 2: Data Preprocessing
# Let's inspect the first few rows of the dataset
print(df.head())

# Drop unnecessary columns (if any)
# Here, 'User ID' and 'Gender' are not needed for our classification, so we'll drop them
df = df.drop(['User ID', 'Gender'], axis=1)

# Step 3: Splitting the dataset into training and testing sets
X = df.iloc[:, :-1].values  # Features (Age and EstimatedSalary)
y = df.iloc[:, -1].values   # Target (Purchased)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Step 4: Feature Scaling (Logistic Regression benefits from scaling)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Step 5: Logistic Regression Model
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

# Step 6: Predicting the results
y_pred = classifier.predict(X_test)

# Step 7: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Step 8: Calculate Performance Metrics
# True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN)
TP = cm[1, 1]
FP = cm[0, 1]
TN = cm[0, 0]
FN = cm[1, 0]

# Accuracy
accuracy = accuracy_score(y_test, y_pred)

# Error Rate
error_rate = 1 - accuracy

# Precision
precision = precision_score(y_test, y_pred)

# Recall
recall = recall_score(y_test, y_pred)
```

```
# F1-Score (Optional but a useful metric)
f1 = f1_score(y_test, y_pred)

# Step 9: Print all metrics
print("\nEvaluation Metrics:")
print(f"True Positives (TP): {TP}")
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")
print(f"Accuracy: {accuracy:.4f}")
print(f"Error Rate: {error_rate:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Practical 6

```
# Import necessary libraries

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


# Step 1: Load the Iris dataset

# You can replace the path with your local 'iris.csv' path

df = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")


# Step 2: Preprocessing the dataset

# Display first few rows of the dataset

print(df.head())


# Step 3: Split the data into features (X) and target variable (y)

X = df.iloc[:, :-1].values  # Features (sepal_length, sepal_width, petal_length, petal_width)

y = df.iloc[:, -1].values   # Target (species)


# Step 4: Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


# Step 5: Initialize the Naïve Bayes classifier

nb_classifier = GaussianNB()


# Step 6: Train the model on the training set

nb_classifier.fit(X_train, y_train)


# Step 7: Predict the test set results

y_pred = nb_classifier.predict(X_test)
```

```
# Step 8: Compute the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Step 9: Calculate Performance Metrics
# True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN)
TP = cm[1, 1]
FP = cm[0, 1]
TN = cm[0, 0]
FN = cm[1, 0]

# Accuracy
accuracy = accuracy_score(y_test, y_pred)

# Error Rate
error_rate = 1 - accuracy

# Precision
precision = precision_score(y_test, y_pred, average='weighted')

# Recall
recall = recall_score(y_test, y_pred, average='weighted')

# F1-Score (optional but useful)
f1 = f1_score(y_test, y_pred, average='weighted')

# Step 10: Print all metrics
print("\nEvaluation Metrics:")
print(f"True Positives (TP): {TP}")
```

```
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")
print(f"Accuracy: {accuracy:.4f}")
print(f"Error Rate: {error_rate:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Practical 7

```
# Importing necessary libraries
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk import pos_tag
from nltk.tokenize import sent_tokenize

# Download necessary resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

# Sample document for processing
document = """
Natural language processing (NLP) is a field of artificial intelligence that enables computers to understand,
interpret, and respond to human language. NLP tasks include text classification, sentiment analysis, machine translation,
and more. It involves the application of linguistic and statistical techniques to extract useful information from textual data.
"""

# 1. Tokenization
tokens = word_tokenize(document)
print("Tokens:", tokens)
```

```
# 2. Part-of-Speech (POS) Tagging
pos_tags = pos_tag(tokens)
print("\nPOS Tags:", pos_tags)

# 3. Stop Words Removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print("\nTokens after stop word removal:", filtered_tokens)

# 4. Stemming (Using PorterStemmer)
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
print("\nStemmed Tokens:", stemmed_tokens)

# 5. Lemmatization (Using WordNetLemmatizer)
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
print("\nLemmatized Tokens:", lemmatized_tokens)

# 6. TF-IDF Representation of Documents

# Example corpus of documents
corpus = [
    "Natural language processing enables computers to understand human language.",
    "Text classification is one of the important tasks in NLP.",
    "NLP helps in sentiment analysis, language translation, and more."
]

# Initialize the TF-IDF Vectorizer
vectorizer = TfidfVectorizer()
```

```
# Fit and transform the corpus to get the TF-IDF representation
tfidf_matrix = vectorizer.fit_transform(corpus)

# Convert the matrix to a DataFrame for better readability
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())
print("\nTF-IDF Representation of Corpus:")
print(tfidf_df)
```

Practical 8

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

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

# 1. Visualizing the patterns in the Titanic dataset
# Display the first few rows to understand the structure of the data
print(titanic.head())

# Plot a countplot for the 'survived' column to check the survival distribution
sns.countplot(x='survived', data=titanic)
plt.title('Survival Distribution on Titanic')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()

# Plot a barplot to see the relationship between 'class' and 'survived'
sns.barplot(x='class', y='survived', data=titanic)
plt.title('Survival Rate by Class')
plt.xlabel('Class')
plt.ylabel('Survival Rate')
plt.show()

# Plot a boxplot to see the distribution of fares by 'class'
sns.boxplot(x='class', y='fare', data=titanic)
plt.title('Fare Distribution by Class')
plt.xlabel('Class')
plt.ylabel('Fare')
```

```python
plt.show()

# 2. Plotting the histogram for the 'fare' column
plt.figure(figsize=(8, 6))
sns.histplot(titanic['fare'], kde=True, bins=30, color='skyblue')
plt.title('Distribution of Ticket Fare')
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.show()
```

Practical 9

```
-- Step 1: Create a Database
CREATE DATABASE IF NOT EXISTS sample_db;

-- Step 2: Use the database (Switch to the database created above)
USE sample_db;

-- Step 3: Create a Table
CREATE TABLE IF NOT EXISTS employees (
   employee_id INT,
   first_name STRING,
   last_name STRING,
   department STRING,
   salary FLOAT
)
STORED AS PARQUET;

-- Step 4: Insert Sample Data into the Table
INSERT INTO employees VALUES (1, 'John', 'Doe', 'Engineering', 75000);
INSERT INTO employees VALUES (2, 'Jane', 'Smith', 'Marketing', 65000);
INSERT INTO employees VALUES (3, 'Sam', 'Brown', 'Engineering', 72000);
INSERT INTO employees VALUES (4, 'Sally', 'Davis', 'Sales', 55000);
INSERT INTO employees VALUES (5, 'Tom', 'Wilson', 'Marketing', 68000);

-- Step 5: Run Simple Queries
-- 5.1: Query to Retrieve All Data from Employees Table
SELECT * FROM employees;

-- 5.2: Query to Get the Average Salary
SELECT AVG(salary) AS average_salary FROM employees;
```

```sql
-- 5.3: Query to Get Employees from the 'Engineering' Department
SELECT * FROM employees WHERE department = 'Engineering';

-- 5.4: Query to Count the Number of Employees in Each Department
SELECT department, COUNT(*) AS number_of_employees
FROM employees
GROUP BY department;

-- 5.5: Query to Get the Employee with the Highest Salary
SELECT * FROM employees ORDER BY salary DESC LIMIT 1;
```

Practical 10

```
-- Step 1: Create a Database
CREATE DATABASE IF NOT EXISTS company_db;

-- Step 2: Use the Database (Switch to the company_db)
USE company_db;

-- Step 3: Create a Table called 'employees'
CREATE TABLE IF NOT EXISTS employees (
    employee_id INT,
    first_name STRING,
    last_name STRING,
    department STRING,
    salary FLOAT
)
STORED AS PARQUET;  -- Using Parquet format for optimized storage

-- Step 4: Insert Sample Data into the 'employees' Table
INSERT INTO employees VALUES (1, 'John', 'Doe', 'Engineering', 75000);
INSERT INTO employees VALUES (2, 'Jane', 'Smith', 'Marketing', 65000);
INSERT INTO employees VALUES (3, 'Sam', 'Brown', 'Engineering', 72000);
INSERT INTO employees VALUES (4, 'Sally', 'Davis', 'Sales', 55000);
INSERT INTO employees VALUES (5, 'Tom', 'Wilson', 'Marketing', 68000);

-- Step 5: Run Simple Queries

-- 5.1: Query to Retrieve All Data from the Employees Table
SELECT * FROM employees;

-- 5.2: Query to Calculate the Average Salary
SELECT AVG(salary) AS average_salary FROM employees;
```

```sql
-- 5.3: Query to Get Employees from the 'Engineering' Department

SELECT * FROM employees WHERE department = 'Engineering';


-- 5.4: Query to Count the Number of Employees in Each Department

SELECT department, COUNT(*) AS number_of_employees

FROM employees

GROUP BY department;


-- 5.5: Query to Get the Employee with the Highest Salary

SELECT * FROM employees ORDER BY salary DESC LIMIT 1;
```