Assignment 5

Set A

1) **Credit Risk Prediction Problem Statement: A bank wants to develop a model that can predict whether a loan applicant is likely to default on their loan. The model should help the bank in making informed lending decisions by accurately classifying applicants into "high risk" and "low risk" categories based on their financial history, credit score, income, and other demographic features. Objective: Build and tune a classification model that predicts loan default risk, balancing between model accuracy and interpretability. Use the German Credit Dataset or a similar dataset.**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder  # To encode categorical data
from sklearn.linear_model import LogisticRegression  # Simple classification model
from sklearn.metrics import accuracy_score

# Step 2: Load dataset
df = pd.read_csv("GermanCredit.csv")  # Read the dataset from a CSV file

le = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
   df[column] = label_encoder.fit_transform(df[column])

X = df.drop(['credit_history','amount','number_credits'], axis=1)
y = df['credit_risk']

# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)  # 70% training, 30% testing

# Step 6: Train a simple Logistic Regression model
model = LogisticRegression()  # Initialize the model
model.fit(X_train, y_train)  # Train the model with the training data

# Step 7: Make predictions on the test data
y_pred = model.predict(X_test)  # Predict the 'Risk' for test data

accuracy = accuracy_score(y_test, y_pred)  # Calculate accuracy
```

```
df = pd.DataFrame({
        'credit risk':y_test,
        'Predictions':y_pred})
Df
```

2) **Customer Churn Prediction Problem Statement: A telecommunications company is facing a high rate of customer churn. The company wants to build a predictive model that can identify customers who are likely to cancel their service subscription. The dataset includes customer demographics, service usage patterns, and customer support interactions. Objective: Select and optimize a model that accurately predicts customer churn, helping the company target retention efforts more effectively. Use the Telco Customer Churn dataset.**

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

data = pd.read_csv("telco.csv")
data.isnull().sum()
cols =
['customerID','gender','SeniorCitizen','Partner','Dependents','PhoneService','MultipleLines','InternetService','OnlineSecurity','OnlineBackup','DeviceProtection','TechSupport',
    'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
    'PaymentMethod', 'TotalCharges', 'Churn']
le = LabelEncoder()

for col in cols:
  data[col] = le.fit_transform(data[col])
data
corr_matrix = data.corr()
corr_matrix

x = data.drop('Churn', axis=1)
y = data['Churn']
x
y
X_train,X_test,y_train,y_test = train_test_split(x, y, test_size=0.2, random_state=42)
rf = RandomForestRegressor(n_estimators=100, random_state=42)
```

```
RF_model = rf.fit(X_train, y_train)
RF_model
y_pred_rf = rf.predict(X_test).round()
df1 = pd.DataFrame({
        'churn':y_test,
        'Predictions':y_pred_rf
        })
df1
```

Set B

1) **House Price Prediction Problem Statement: A real estate agency wants to build a model that predicts house prices based on various features such as location, size, number of rooms, and amenities. The goal is to provide accurate price estimates for houses in different neighborhoods. Objective: Develop and fine-tune a regression model to predict house prices, ensuring the model generalizes well across different regions. Use the Boston Housing Dataset.**

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
import random


def generate_row():
   location = random.choice(["Pune", "Mumbai", "Nashik", "Kolhapur","Nagpur"])
   size = random.randint(50, 200)
   rooms = random.randint(1, 5)
   amenities = ", ".join(random.sample(["Gym", "Pool", "Garden", "Parking", "AC"], random.randint(1, 3)))


   # Calculate price based on rooms and amenities
   base_price = 100000  # Base price for a 1-room house with no amenities
   price_per_room = 50000
   price_per_amenity = 20000


   price = base_price + (rooms - 1) * price_per_room + len(amenities.split(", ")) * price_per_amenity
```

```
    return [location, size, rooms, amenities, price]


data = [generate_row() for _ in range(300)]
df = pd.DataFrame(data, columns=["Location", "Size (sqft)", "Number of Rooms",
"Amenities", "Price"])


df.to_csv("house_data.csv")
df.isnull().sum()


cols = ['Location','Amenities']
le = LabelEncoder()


for col in cols:
    df[col] = le.fit_transform(df[col])
X = df.drop('Price', axis=1)
y = df['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
lr = LinearRegression()


# Fit model
LR_model = lr.fit(X_train, y_train)
LR_model
y_pred = LR_model.predict(X_test).round()
df = pd.DataFrame({
        'Price':y_test,
        'Predictions':y_pred})
Df
```

2) **Sentiment Analysis of Product Reviews Problem Statement: An e-commerce company wants to analyze customer reviews to automatically determine the sentiment (positive, negative, or neutral) of each review. The dataset consists of text reviews and associated metadata like review date and rating. Objective: Select and optimize a natural language processing (NLP) model to classify the sentiment of product reviews, improving the company’s ability to monitor customer satisfaction. Use the Amazon Product Reviews dataset**.

Set C

1) **Fraud Detection in Online Transactions Problem Statement: An online payment platform is concerned about fraudulent transactions and wants to build a model to detect potential fraud in real-time. The dataset includes transaction details such as amount, location, time, and device information. Objective: Build and tune a model that identifies fraudulent transactions with high accuracy while minimizing false positives to reduce unnecessary alerts. Use the Credit Card Fraud Detection dataset.**

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

data = pd.read_csv("creditcard.csv")
data
print(data.isnull().sum())
sns.countplot(x='Class', data=data)
plt.title('Class Distribution')
plt.show()
X = data.drop('Class', axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
data = pd.DataFrame({
        'Class':y_test,
        'Predictions':y_pred})
data
```

2) **Predicting Patient Readmission Rates Problem Statement: A hospital wants to reduce patient readmission rates by predicting which patients are likely to be readmitted within 30 days of discharge. The dataset includes patient demographics, medical history, and details of their hospital stay. Objective: Develop and optimize a predictive model to identify patients at risk of readmission, aiding in the development of targeted intervention programs to improve patient outcomes. Use the Hospital Readmission Rates dataset.**

```
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.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier

# Step 1: Load the data
data = pd.read_csv("patient.csv")

# Step 2: Handle missing data and drop columns that are not needed
data.isnull().sum()
df = data.drop(columns=['max_glu_serum','A1Cresult'])  # Dropping unnecessary columns
df.isnull().sum()  # Check if any missing values are left

# Step 3: Label Encoding for categorical columns
le = LabelEncoder()
categorical_columns = df.select_dtypes(include=['object']).columns
for col in categorical_columns:
   df[col] = le.fit_transform(df[col])

# Step 4: Split data into features (X) and target (y)
X = df.drop("readmitted", axis=1)  # Features
y = df["readmitted"]  # Target variable

# Step 5: 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=42)

# Step 6: Feature Scaling
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 7: Train a Random Forest Classifier model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Step 8: Make predictions
y_pred = model.predict(X_test)

# Step 9: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Step 10: Create a DataFrame to compare predictions with actual values
df_comparison = pd.DataFrame({
            'readmitted': y_test,
            'Predictions': y_pred})
print(df_comparison)
```