1. What is the K-Nearest Neighbors (KNN) algorithm? What are the basic steps involved in the working of KNN? (No need to explain in detail—just list the steps.) How do we choose the optimal value of K in KNN?

KNN is a supervised learning algorithm used for classification and regression. • It predicts the label of a new data point by looking at the 'K' closest labeled points (neighbors) in the training set.
KNN works by following steps:

1. Choose the value of K
   (e.g., K = 3)
2. Measure distances
   Calculate the distance between the new data point and all training data points (e.g., using Euclidean distance).
3. Find K nearest neighbors
   Select the K data points with the smallest distances.
4. Vote for the class (for classification)
   Count the classes among the K neighbors and choose the most frequent one.
5. Predict the output
   Assign the majority class (for classification) or average value (for regression) to the new data point.

The optimal value of K in KNN is determined using:

- Start with small odd values (e.g., K = 1, 3, 5) to avoid ties in classification.
- Use Cross-Validation:
  Split data into training and validation sets.
  Try different K values.
  Choose the K that gives the best validation accuracy.
- Avoid too small K:
  K = 1 is very sensitive to noise (overfitting).
- Avoid too large K:
  Large K may include distant or irrelevant points (underfitting).

2. What is Euclidean Distance? Why is it used in the KNN algorithm? Why is it important to standardize or normalize feature values in KNN?

Euclidean distance is defined as the metric that determines the distance between two vectors by calculating the square root of the sum of the squared differences of their corresponding components.

Euclidean distance is used in KNN algorithm as KNN depends on the distance between points (Euclidean distance, for instance). If one feature (e.g., Income) has much larger values than another feature (e.g., Age), it will dominate the distance calculation and make the algorithm biased. The distance will be dominated by the Income difference, causing the model to give more importance to Income and ignore Age. Let's say we have 5 data points with Age and Income: Age: [25, 30, 35, 40, 45] and Income: [50,000, 55,000, 60,000, 65,000, 70,000]. We apply the Z-score formula to standardize Age and Income.

By standardizing the features, we ensure that both Age and Income are on the same scale, and neither dominates the other in distance calculations.

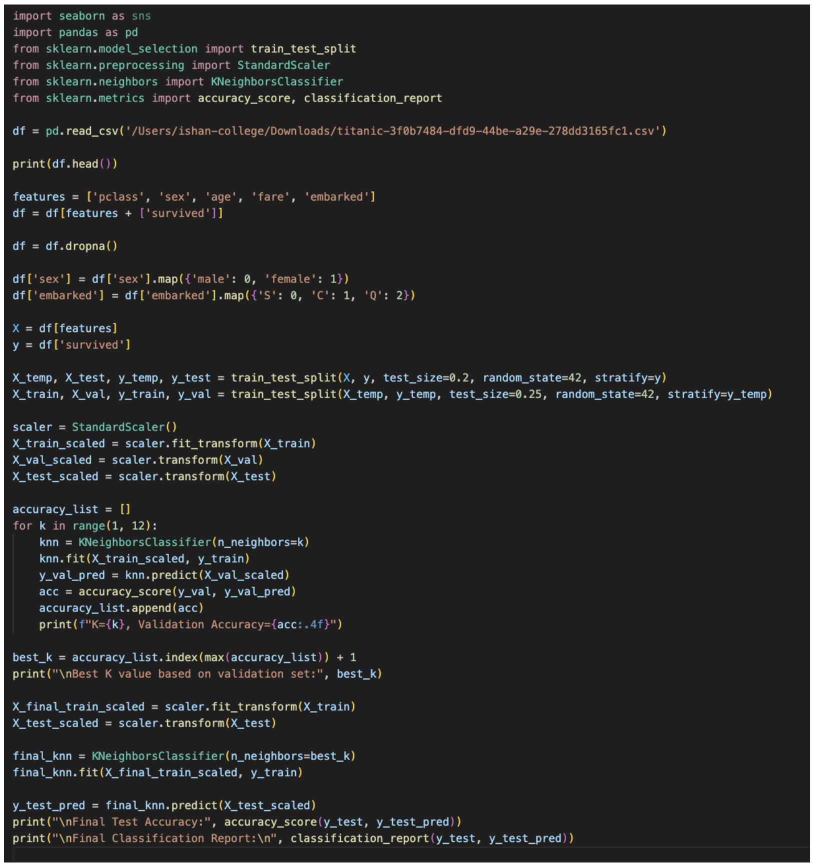
3. Implement the KNN algorithm using a real-world dataset.

- Use Python (with libraries such as pandas, scikit-learn, etc.) to:
  Load and preprocess a dataset (e.g., Iris, Titanic, or any other classification dataset).
  Train and test a KNN model using appropriate train_test_split.
  Display the classification results and evaluate the model using relevant performance metrics (e.g., accuracy, precision, recall).

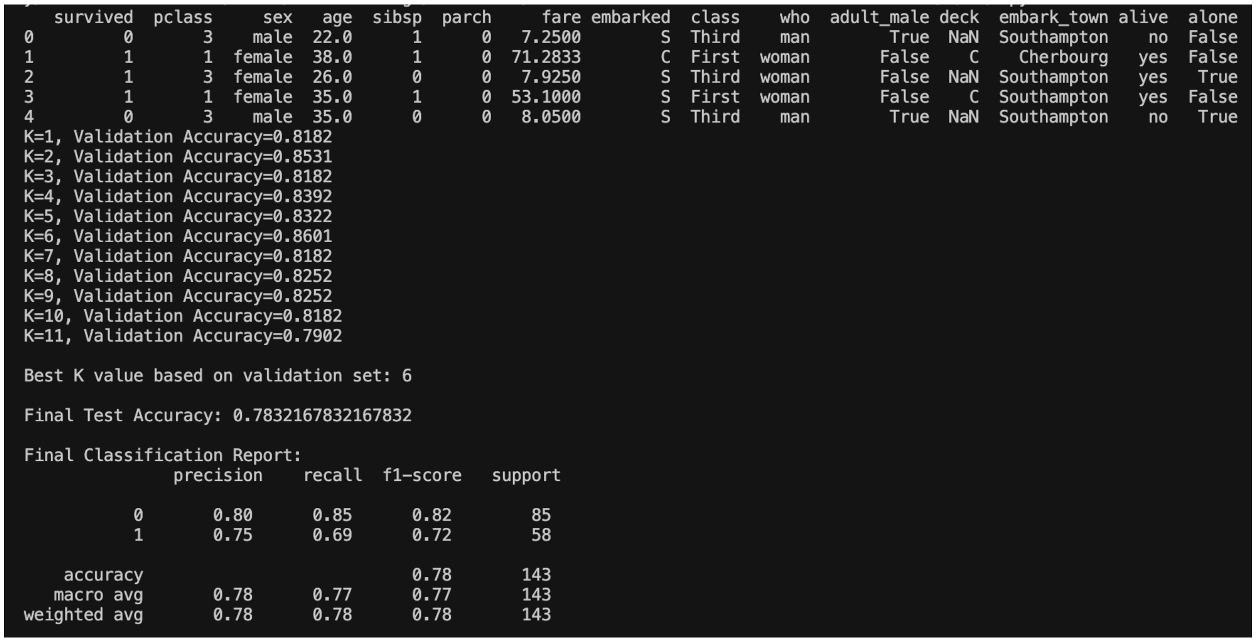
Task:

◦ Write clean and well-commented Python code.

◦ Include a screenshot of your program's output, showing predictions, accuracy score, or confusion matrix.



```python
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv('/Users/ishan-college/Downloads/titanic-3f0b7484-dfd9-44be-a29e-278dd3165fc1.csv')

print(df.head())

features = ['pclass', 'sex', 'age', 'fare', 'embarked']
df = df[features + ['survived']]

df = df.dropna()

df['sex'] = df['sex'].map({'male': 0, 'female': 1})
df['embarked'] = df['embarked'].map({'S': 0, 'C': 1, 'Q': 2})

X = df[features]
y = df['survived']

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42, stratify=y_temp)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

accuracy_list = []
for k in range(1, 12):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_val_pred = knn.predict(X_val_scaled)
    acc = accuracy_score(y_val, y_val_pred)
    accuracy_list.append(acc)
    print(f"K={k}, Validation Accuracy={acc:.4f}")

best_k = accuracy_list.index(max(accuracy_list)) + 1
print("\nBest K value based on validation set:", best_k)

X_final_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

final_knn = KNeighborsClassifier(n_neighbors=best_k)
final_knn.fit(X_final_train_scaled, y_train)

y_test_pred = final_knn.predict(X_test_scaled)
print("\nFinal Test Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nFinal Classification Report:\n", classification_report(y_test, y_test_pred))
```



```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class    who  adult_male deck  embark_town alive  alone
0         0       3    male  22.0      1      0   7.2500        S  Third    man        True  NaN  Southampton    no  False
1         1       1  female  38.0      1      0  71.2833        C  First  woman       False    C    Cherbourg   yes  False
2         1       3  female  26.0      0      0   7.9250        S  Third  woman       False  NaN  Southampton   yes   True
3         1       1  female  35.0      1      0  53.1000        S  First  woman       False    C  Southampton   yes  False
4         0       3    male  35.0      0      0   8.0500        S  Third    man        True  NaN  Southampton    no   True
K=1, Validation Accuracy=0.8182
K=2, Validation Accuracy=0.8531
K=3, Validation Accuracy=0.8182
K=4, Validation Accuracy=0.8392
K=5, Validation Accuracy=0.8322
K=6, Validation Accuracy=0.8601
K=7, Validation Accuracy=0.8182
K=8, Validation Accuracy=0.8252
K=9, Validation Accuracy=0.8252
K=10, Validation Accuracy=0.8182
K=11, Validation Accuracy=0.7902

Best K value based on validation set: 6

Final Test Accuracy: 0.7832167832167832

Final Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.85      0.82        85
           1       0.75      0.69      0.72        58

    accuracy                           0.78       143
   macro avg       0.78      0.77      0.77       143
weighted avg       0.78      0.78      0.78       143
```

4. What is a confusion matrix?A fraud detection model was tested on 1,000 transactions. The actual and predicted outcomes are summarized as follows:

• Actual Outcomes:

◦ 950 transactions were legitimate.

◦ 50 transactions were fraudulent.

• Model Predictions:

◦ It labeled 930 transactions as legitimate, including 20 that were actually fraudulent.

◦ It labeled 70 transactions as fraudulent, of which 30 were actually fraudulent.

Task:

Construct the confusion matrix for this binary classification problem, where:

• Positive class = Fraudulent

• Negative class = Legitimate

Identify the values of:

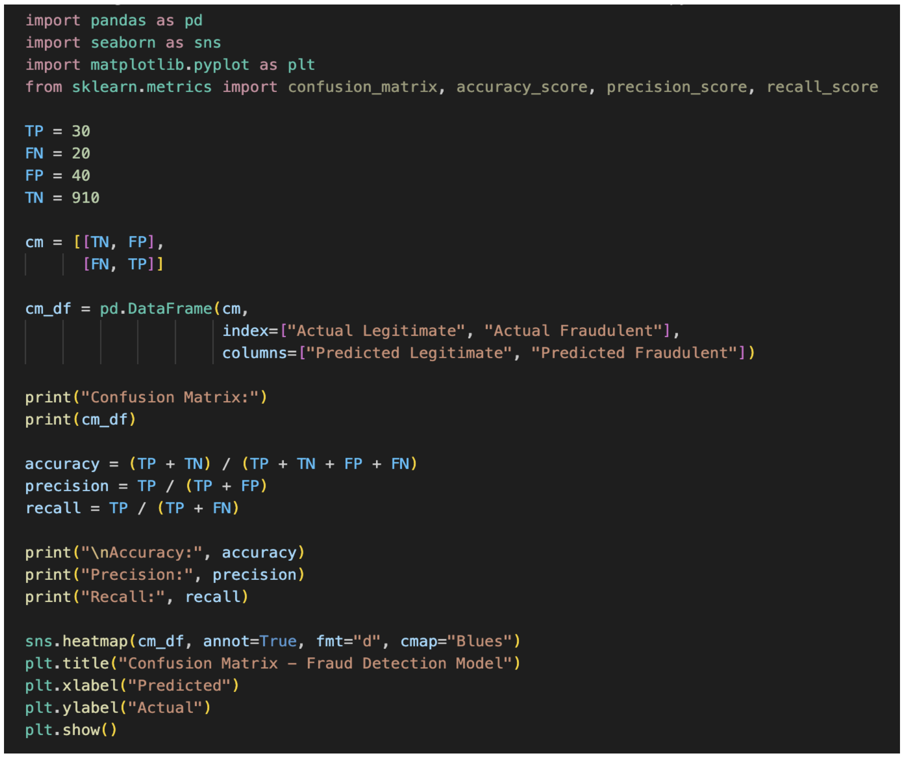
• True Positives (TP)

• False Positives (FP)

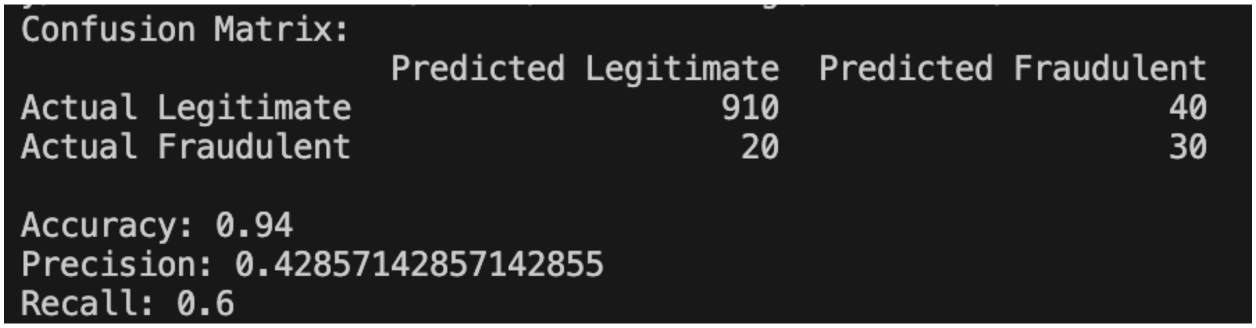
• False Negatives (FN)

• True Negatives (TN)

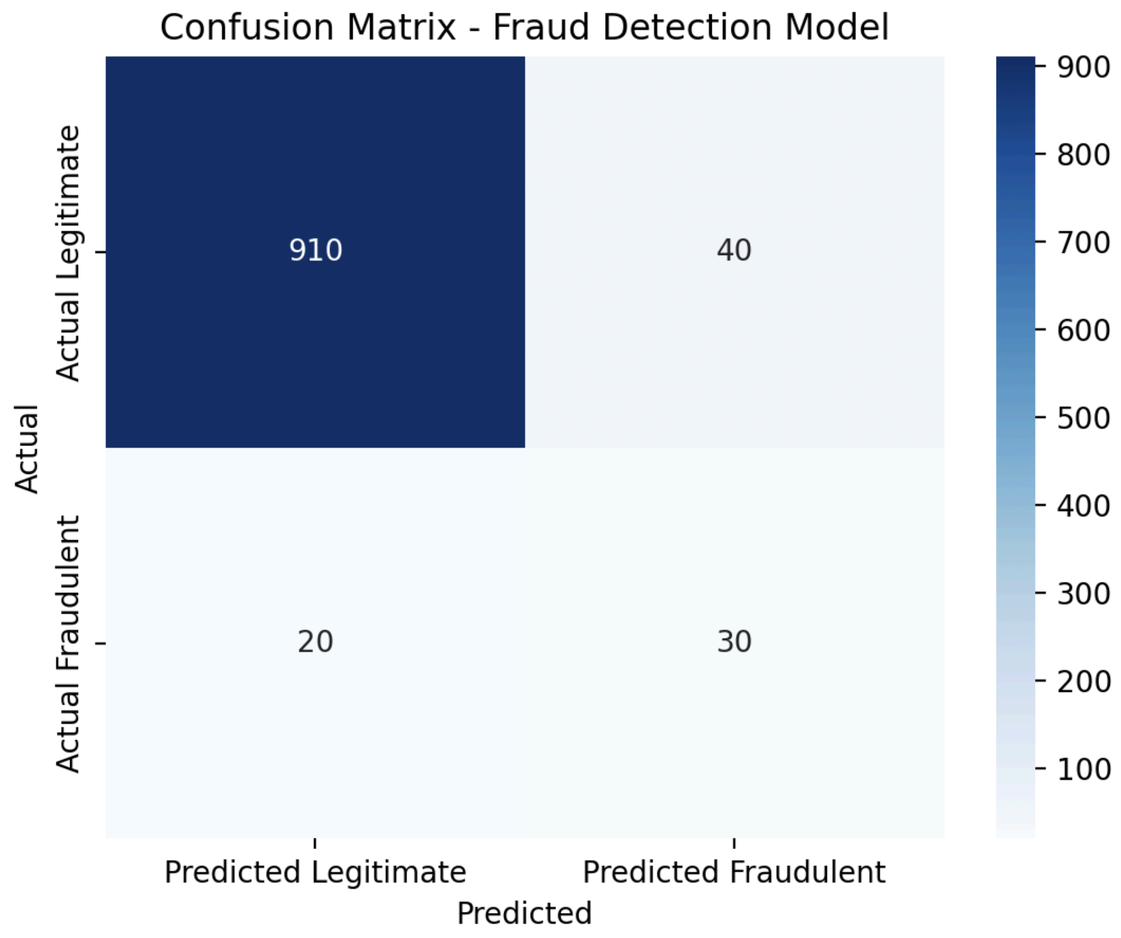
• Also calculate the values of Accuracy, Precision, Recall and Accuracy.



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

TP = 30
FN = 20
FP = 40
TN = 910

cm = [[TN, FP],
      [FN, TP]]

cm_df = pd.DataFrame(cm,
                     index=["Actual Legitimate", "Actual Fraudulent"],
                     columns=["Predicted Legitimate", "Predicted Fraudulent"])

print("Confusion Matrix:")
print(cm_df)

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)

print("\nAccuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

sns.heatmap(cm_df, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Fraud Detection Model")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



```
Confusion Matrix:
                   Predicted Legitimate  Predicted Fraudulent
Actual Legitimate                   910                    40
Actual Fraudulent                    20                    30

Accuracy: 0.94
Precision: 0.42857142857142855
Recall: 0.6
```


Confusion Matrix - Fraud Detection Model
Actual
Actual Legitimate
Actual Fraudulent
Predicted Legitimate
Predicted Fraudulent
Predicted