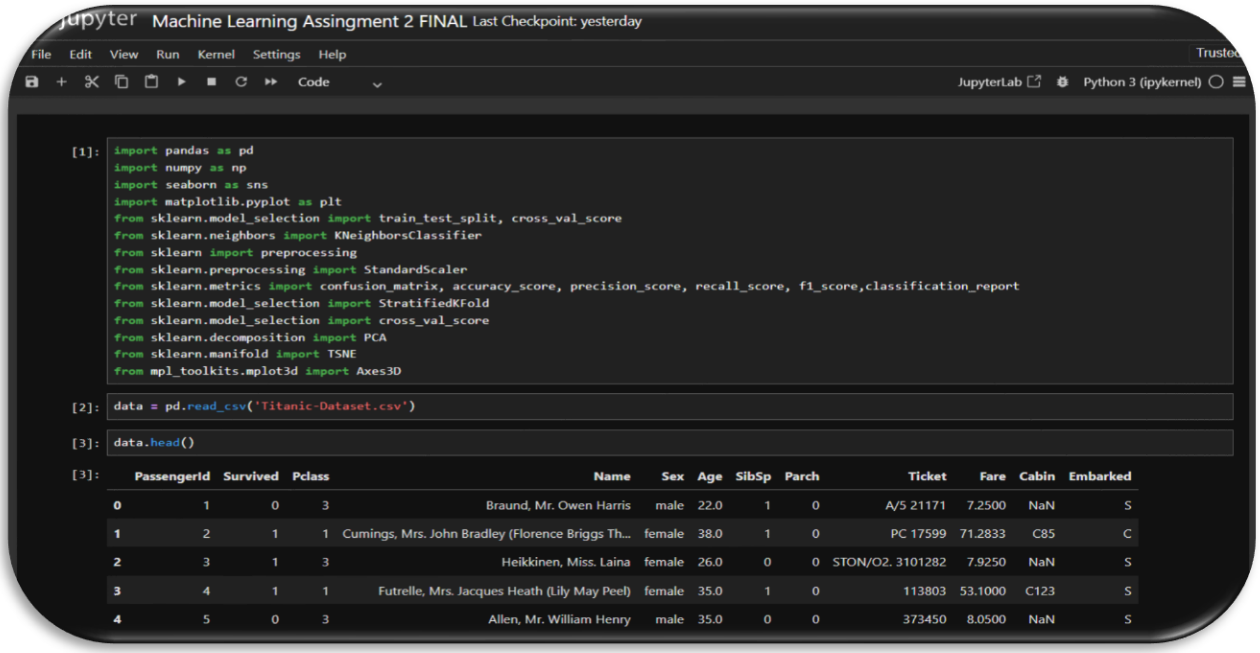
# MACHINE LEARNING ASSIGNMENT 2


## Introduction:

The Titanic dataset is a historical dataset that contains details about passengers aboard the Titanic, such as age, class, fare, and sex. The primary goal is to develop a machine learning model using K-Nearest Neighbors algorithm (KNN) to predict whether a passenger survived or not. This involves data preprocessing, model training, evaluation and analysis to understand the factors that influenced survival and assess the model's predictive performance.

## Data preprocessing:

First of all, we started importing the important libraries we needed in our notebook, then started reading the dataset…



jupyter Machine Learning Assingment 2 FINAL Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
[1]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score,classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from mpl_toolkits.mplot3d import Axes3D

[2]: data = pd.read_csv('Titanic-Dataset.csv')

[3]: data.head()
```

[3]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

## Data Cleaning:

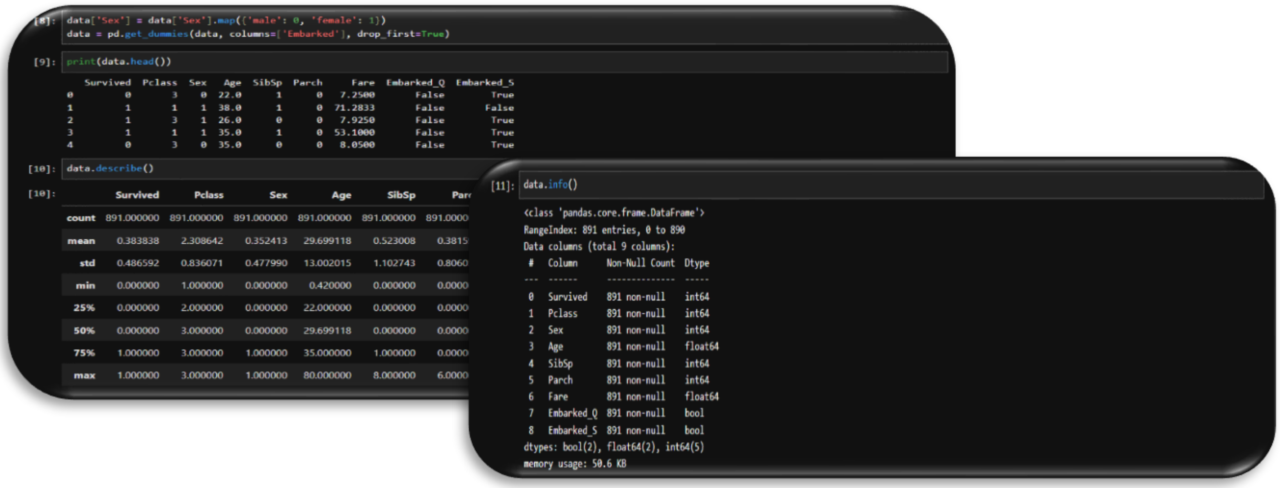
- Checking the null values ,then dropping the columns that we are not going to use ,then start filling the null values to make the data ready for working.
- Print the data types of the columns in the dataset.

```
[4]: data.isnull().sum()

[4]: PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

[5]: data = data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
data['Age'].fillna(data['Age'].mean(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

[6]: data.isnull().sum()

[6]: Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64

[7]: print(data.dtypes)

Survived      int64
Pclass        int64
Sex          object
Age         float64
SibSp         int64
Parch         int64
Fare        float64
Embarked     object
dtype: object
```

## Encoding:

The process of converting categorical **(non-numeric)** data into a **numerical format** so that machine learning models can understand and work with it.

- **One-Encoding** for categorical features like **'Sex'** and **'Embarked'**.
- Data description and some info about it.



```
[8]: data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
     data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

[9]: print(data.head())

   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked_Q  Embarked_S
0         0       3    0  22.0      1      0   7.2500       False        True
1         1       1    1  38.0      1      0  71.2833       False       False
2         1       3    1  26.0      0      0   7.9250       False        True
3         1       1    1  35.0      1      0  53.1000       False        True
4         0       3    0  35.0      0      0   8.0500       False        True

[10]: data.describe()
```

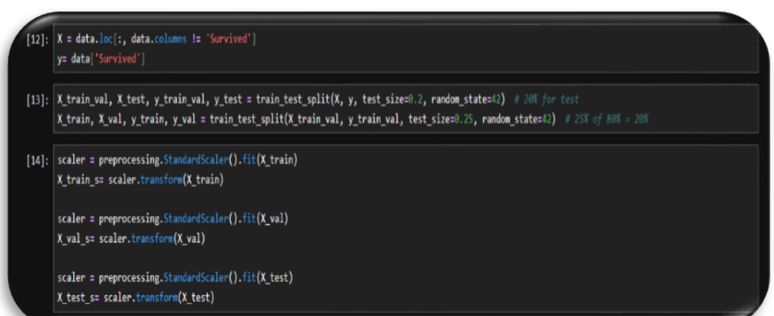
| [10]: | Survived | Pclass | Sex | Age | SibSp | Par |
|---|---|---|---|---|---|---|
| count | 891.000000 | 891.000000 | 891.000000 | 891.000000 | 891.000000 | 891.0000 |
| mean | 0.383838 | 2.308642 | 0.352413 | 29.699118 | 0.523008 | 0.3815 |
| std | 0.486592 | 0.836071 | 0.477990 | 13.002015 | 1.102743 | 0.8060 |
| min | 0.000000 | 1.000000 | 0.000000 | 0.420000 | 0.000000 | 0.0000 |
| 25% | 0.000000 | 2.000000 | 0.000000 | 22.000000 | 0.000000 | 0.0000 |
| 50% | 0.000000 | 3.000000 | 0.000000 | 29.699118 | 0.000000 | 0.0000 |
| 75% | 1.000000 | 3.000000 | 1.000000 | 35.000000 | 1.000000 | 0.0000 |
| max | 1.000000 | 3.000000 | 1.000000 | 80.000000 | 8.000000 | 6.0000 |

```
[11]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    int64
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked_Q  891 non-null    bool
 8   Embarked_S  891 non-null    bool
dtypes: bool(2), float64(2), int64(5)
memory usage: 50.6 KB
```

- 'Sex' column instead of male & female became "0" for male and "1" for female .
- 'Embarked' column became "True" & "False" (Boolean Feature).

## Data Scaling & Splitting:

- Features were scaled using "**StandardScaler"** normalize the range and ensure fair distance measurement in KNN.
- Dataset was split into "60% training set", "20% validation set", "20% testing set".



```
[12]: X = data.loc[:, data.columns != 'Survived']
      y= data['Survived']

[13]: X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  # 20% for test
      X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=42)  # 25% of 80% = 20%

[14]: scaler = preprocessing.StandardScaler().fit(X_train)
      X_train_s= scaler.transform(X_train)

      scaler = preprocessing.StandardScaler().fit(X_val)
      X_val_s= scaler.transform(X_val)

      scaler = preprocessing.StandardScaler().fit(X_test)
      X_test_s= scaler.transform(X_test)
```

## KNN Model:

**K-Nearest Neighbors**, one of the simplest and most intuitive algorithms. It is a supervised learning technique used for both classification and regression, most commonly used in classification.

```
k_values = range(1, 31)
validation_accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_s, y_train)
    y_val_pred = knn.predict(X_val_s)
    acc = accuracy_score(y_val, y_val_pred)
    validation_accuracies.append(acc)

print("Validation Accuracies: ", validation_accuracies)

# Best K
best_k = k_values[validation_accuracies.index(max(validation_accuracies))]
print("Best K is: ", best_k)
```

```
Validation Accuracies:  [0.6741573033707865, 0.7415730337078652, 0.7471910112359551, 0.797752808988764, 0.7696629213483146, 0.7808988764044944, 0.797752808988764, 0.8202247191011236, 0.8202247191011236, 0.8314606741573034, 0.8314606741573034, 0.8258426966292135, 0.8314606741573034, 0.8202247191011236, 0.8258426966292135, 0.8089887640449438, 0.8146067415730337, 0.797752808988764, 0.8258426966292135, 0.7921348314606742, 0.8033707865168539, 0.797752808988764, 0.8033707865168539, 0.8089887640449438, 0.8146067415730337, 0.8146067415730337, 0.8033707865168539, 0.797752808988764, 0.8033707865168539, 0.797752808988764]
Best K is:  10
```

```
[16]: # Model with best_k
final_knn = KNeighborsClassifier(n_neighbors=best_k)
final_knn.fit(X_train_s, y_train)
```

```
[16]: KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

```
[17]: y_test_pred = final_knn.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(test_accuracy)
```

```
0.664804469273743
```

- Finding the best k we got “k=10”.
- Performing the best k  selected as the final model.
- Final model achieved a test set accuracy of approximately 66.48%.

## Cross Validation:

A technique used to evaluate the performance of a machine learning model by splitting the dataset into multiple parts (folds), training the model on some parts, and testing it on the remaining parts. This helps ensure the model generalizes well and isn’t just performing well by chance on a particular train-test split.

```
[18]: # 5. Cross-Validation with Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(final_knn, X_train_s, y_train, cv=skf)
print("Stratified Cross-Validation Scores:", cv_scores)
print("Average Stratified Cross-Validation Score:", np.mean(cv_scores))

Stratified Cross-Validation Scores: [0.85046729 0.80373832 0.74766355 0.77570093 0.88679245]
Average Stratified Cross-Validation Score: 0.8128725092576264
```

### K-Fold Cross-Validation:

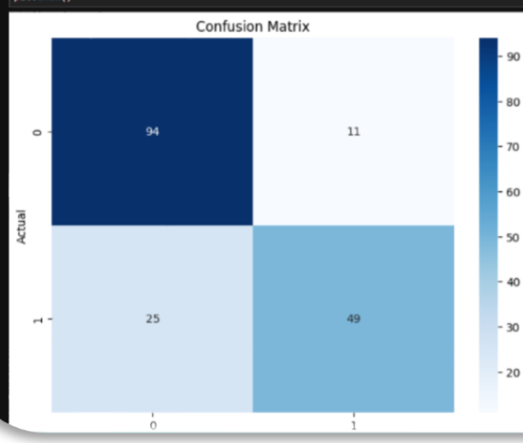
One of the most widely used methods to evaluate model performance and ensure it generalizes well to unseen data.

- Applied 5-Fold Cross-Validation on the training set.
- Average cross-validation was approximately 81.287%.
- Cross-Validation helped validate model generalizability and avoid overfitting to a specific split.

# Confusion Matrix:

```
[20]: # Step 3: Confusion Matrix
      cm = confusion_matrix(y_test, y_pred)
      print("\nConfusion Matrix:")
      print(cm)

      Confusion Matrix:
      [[94 11]
       [25 49]]

[21]: # Step 4: Classification Report (includes Precision, Recall, F1-Score)
      report = classification_report(y_test, y_pred)
      print("\nClassification Report:")
      print(report)

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.79      0.90      0.84       105
                 1       0.82      0.66      0.73        74

          accuracy                           0.80       179
         macro avg       0.80      0.78      0.79       179
      weighted avg       0.80      0.80      0.79       179
```

```
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




- A confusion matrix is a table that is often used to describe the performance of a classification model. It shows the actual versus predicted classifications, helping us assess the model's accuracy, precision, recall, and other metrics.
- The classification report is used to quickly summarize key metrics like "Precision" , "recall" and "F1-score" across all classes in a classification problem.

**Both are typically used together to evaluate and improve a model's performance in a classification task.**

## Overfitting Discussion:

Overfitting occurs when a model performs well on the training data but poorly on unseen data due to learning noise or specific patterns that don't generalize.

In our KNN model,we compared the accuracy across the training, validation and test sets to assess the overfitting.

- Training Accuracy: 83.33%
- Validation Accuracy: 83.15%
- Testing Accuracy: 79.89%
- Cross-Validation Accuracy: 81.29%

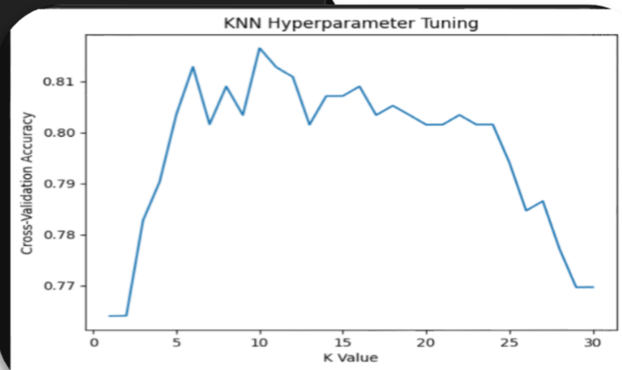
```
[23]: train_accuracy = final_knn.score(X_train_s, y_train)
      validation_accuracy = final_knn.score(X_val_s, y_val)
      test_accuracy = final_knn.score(X_test_s, y_test)

      print("\nPerformance Summary:")
      print(f"Training Accuracy: {train_accuracy:.4f}")
      print(f"Validation Accuracy: {validation_accuracy:.4f}")
      print(f"Test Accuracy: {test_accuracy:.4f}")
      print(f"Cross-Validation Accuracy: {cv_scores.mean():.4f}")

      Performance Summary:
      Training Accuracy: 0.8333
      Validation Accuracy: 0.8315
      Test Accuracy: 0.7989
      Cross-Validation Accuracy: 0.8129
```

- A noticeable gap between training and validation/test accuracy suggests that the model may be slightly overfitting to the training data.
- Overfitting in KNN often occurs when **K is too small**, making the model overly sensitive to noise and outliers.

## KNN Hyperparameter Tunning:

- Plotting a graph between "K Value" and "Cross Validation Accuracy" to improve the model's accuracy and generalization by finding the most effective values for key parameters that influence how the algorithm makes predictions.

```
[24]: k_values = range(1, 31)
      cv_scores = []

      for k in k_values:
          knn = KNeighborsClassifier(n_neighbors=k)
          scores = cross_val_score(knn, X_train_s, y_train, cv=5, scoring='accuracy')
          cv_scores.append(scores.mean())

      # Plot
      plt.plot(k_values, cv_scores)
      plt.xlabel('K Value')
      plt.ylabel('Cross-Validation Accuracy')
      plt.title('KNN Hyperparameter Tuning')
      plt.show()

      # Best K
      best_k = k_values[cv_scores.index(max(cv_scores))]
      print(f"Best K: {best_k}")
```




```
Best K: 10
```

## PCA (Principle Component Analysis):

PCA helps in reducing overfitting by:

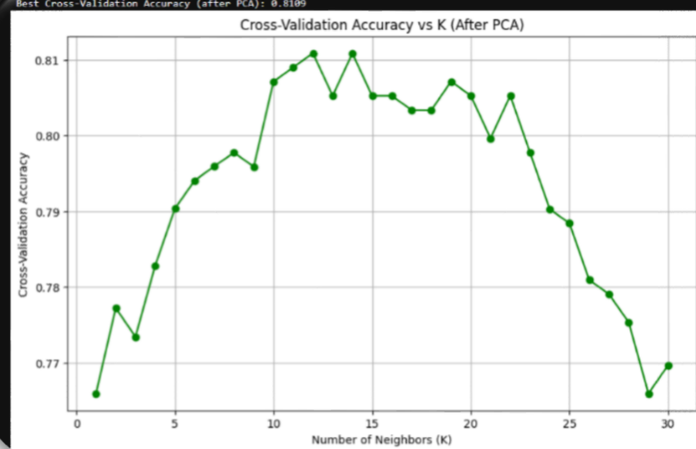
- Removing redundant features by combining correlated features into fewer uncorrelated components reducing the model's fit with noise.
- Reducing Dimensionality by projecting data into a lower-dimensional space, PCA simplifies the learning task making the model less fit with the noise.
- Filtering the noise by keeping only the components that explain the most variance, representing the true structure of data.

```
[25]: pca = PCA(n_components=0.95)  # Retain 95% variance
      X_train_pca = pca.fit_transform(X_train_s)
      X_test_pca = pca.transform(X_test_s)
      print(f"Original number of features: {X_train_s.shape[1]}")
      print(f"Reduced number of features after PCA: {X_train_pca.shape[1]}")

      Original number of features: 8
      Reduced number of features after PCA: 7
```

## KNN Hyperparameter Tunning after PCA:

```python
# 4. Cross-Validation to Tune K (After PCA)
cv_scores_pca = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_pca, y_train, cv=5, scoring='accuracy')
    cv_scores_pca.append(scores.mean())

best_k_pca = k_values[np.argmax(cv_scores_pca)]
best_score_pca = max(cv_scores_pca)

print(f"Best K (after PCA): {best_k_pca}")
print(f"Best Cross-Validation Accuracy (after PCA): {best_score_pca:.4f}")

# Plot K vs Cross-Validation Accuracy (After PCA)
plt.figure(figsize=(10, 6))
plt.plot(k_values, cv_scores_pca, marker='o', color='green')
plt.title('Cross-Validation Accuracy vs K (After PCA)')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Cross-Validation Accuracy')
plt.grid(True)
plt.show()
```

```
Best K (after PCA): 12
Best Cross-Validation Accuracy (after PCA): 0.8109
```




- ✓ We notice that "K Value" inctreased to 12 instead of 10.
- ✓ Visualizing the graph between "Number Of Neighbors" and "Cross Validation Accuracy".

## Retraining the KNN-Model:

The best configuration is chosen k=12, we started to retrain the model using both training and validation sets. Allowing the model to learn from the maximum amount of data, then start making the final predictions on the unseen test set.

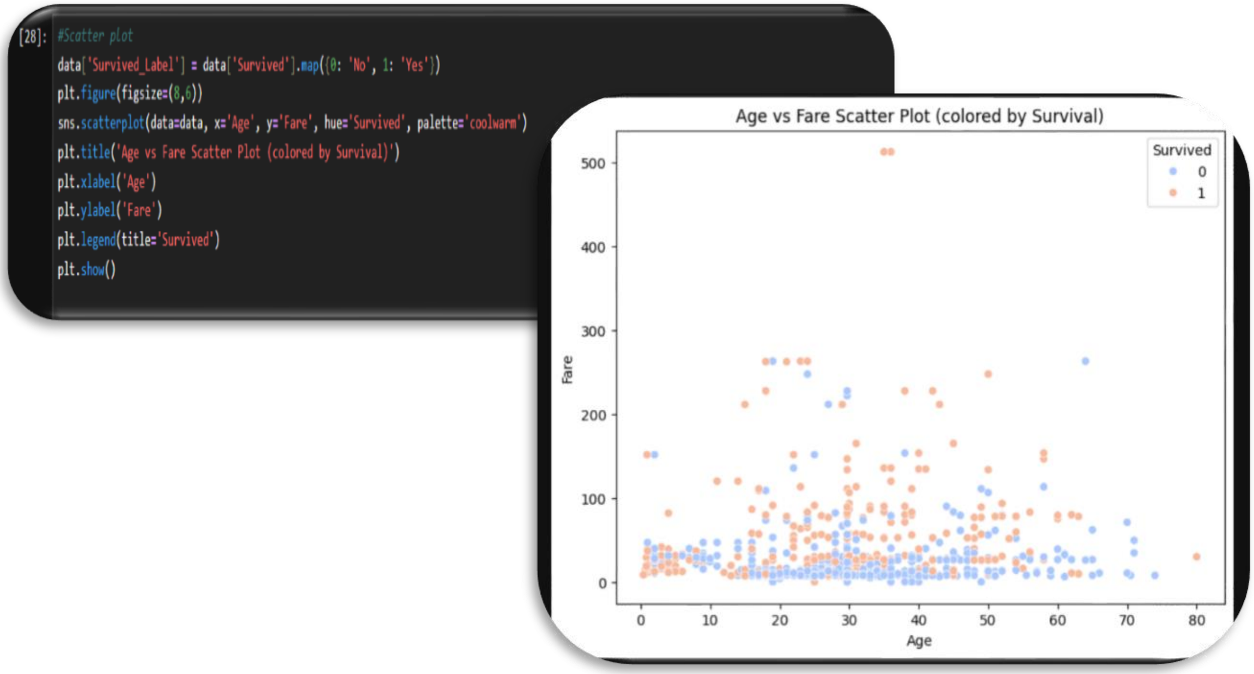
```python
# 5. Retrain Final KNN Model (After PCA)

final_knn_pca = KNeighborsClassifier(n_neighbors=best_k_pca)
final_knn_pca.fit(X_train_pca, y_train)

# Predictions
y_pred_pca = final_knn_pca.predict(X_test_pca)

report = classification_report(y_test, y_pred_pca)
print("\nClassification Report:")
print(report)
```

Classification Report:

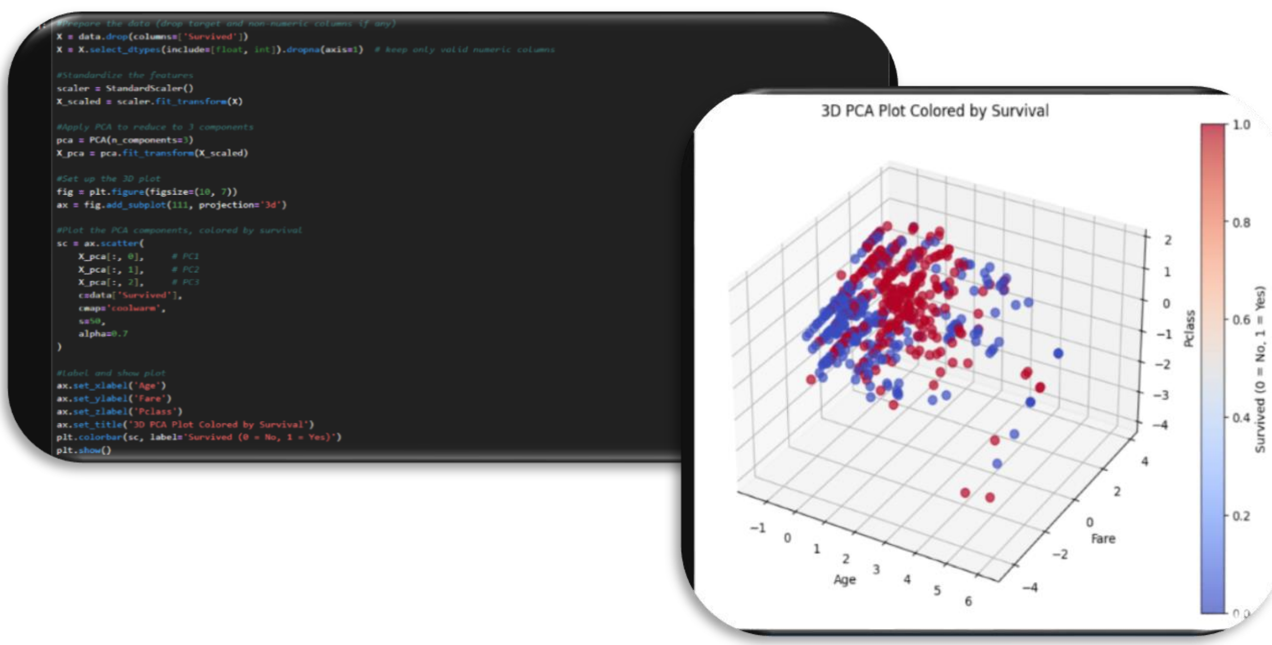
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.78 | 0.89 | 0.83 | 105 |
| 1 | 0.80 | 0.65 | 0.72 | 74 |
| accuracy | | | 0.79 | 179 |
| macro avg | 0.79 | 0.77 | 0.77 | 179 |
| weighted avg | 0.79 | 0.79 | 0.78 | 179 |

## Visualizations:

- **2D Plot (Age vs. Fare)**: Revealed some survival patterns, with some clustering of survivors by fare and age.




- **3D Plot (Age, Fare, Pclass)**: Provided deeper insight into class separability in multi-feature space.



## Conclusion:

This project applied a K-Nearest Neighbors (KNN) classifier to predict Titanic passenger survival. After thorough preprocessing and hyperparameter tuning, the model achieved reasonable accuracy and generalization, as shown by validation and test metrics. Confusion matrix analysis and cross-validation helped assess performance and detect overfitting. While KNN proved effective for this task, future work could explore more advanced models and feature selection to further improve results.

### Team members and their roles:

- Farah Walid (23010036)
- Basmala Hossam El-Din (23011052)
- Rodina Mohamed (23010014)

| Role | Team Member |
|---|---|
| Data splitting & KNN (Notebook Markdowns) | Basmala |
| Cross Validation & Confusion Matrix | Rodina |
| Overfitting & Visualizations (Report Documentation) | Farah |