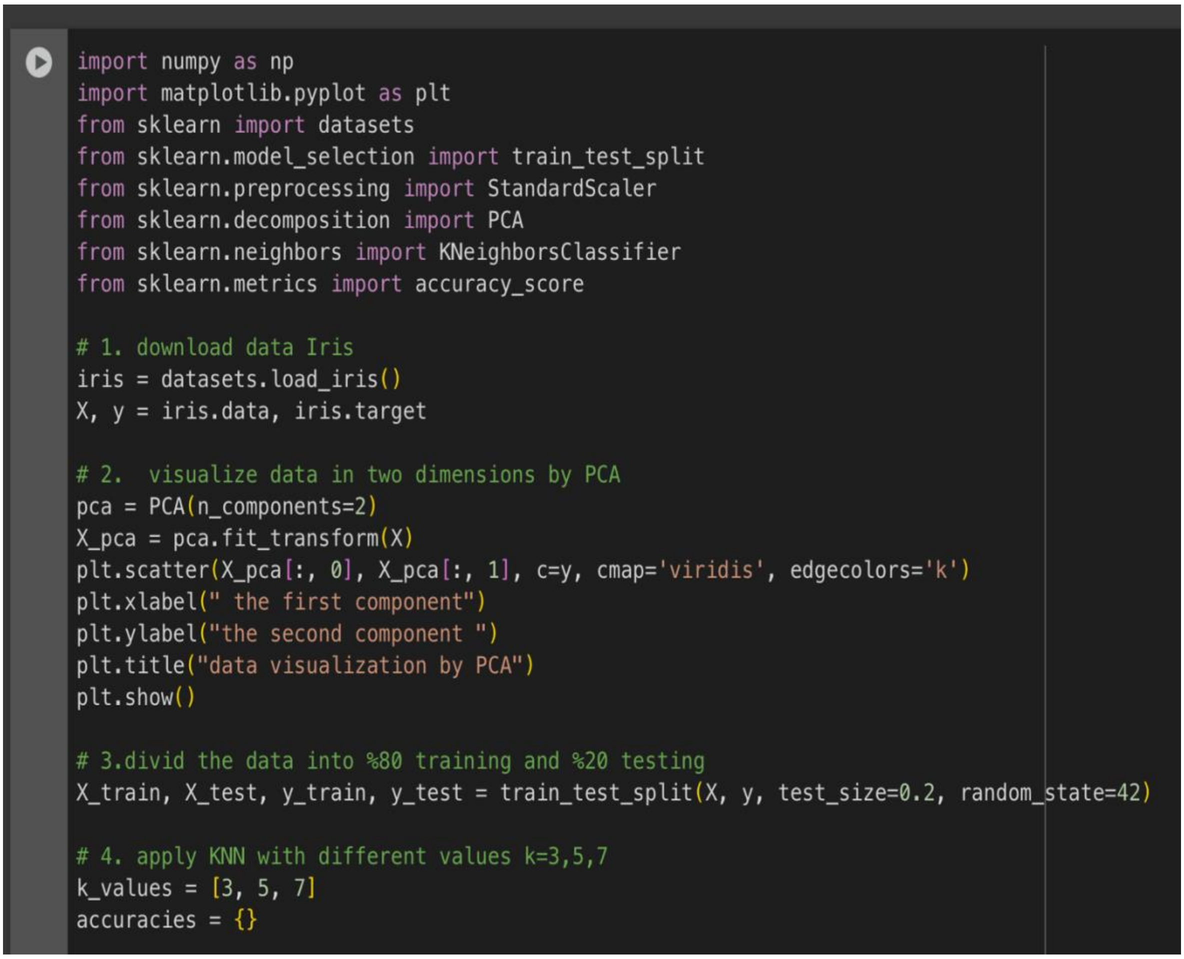
KNN Assignment

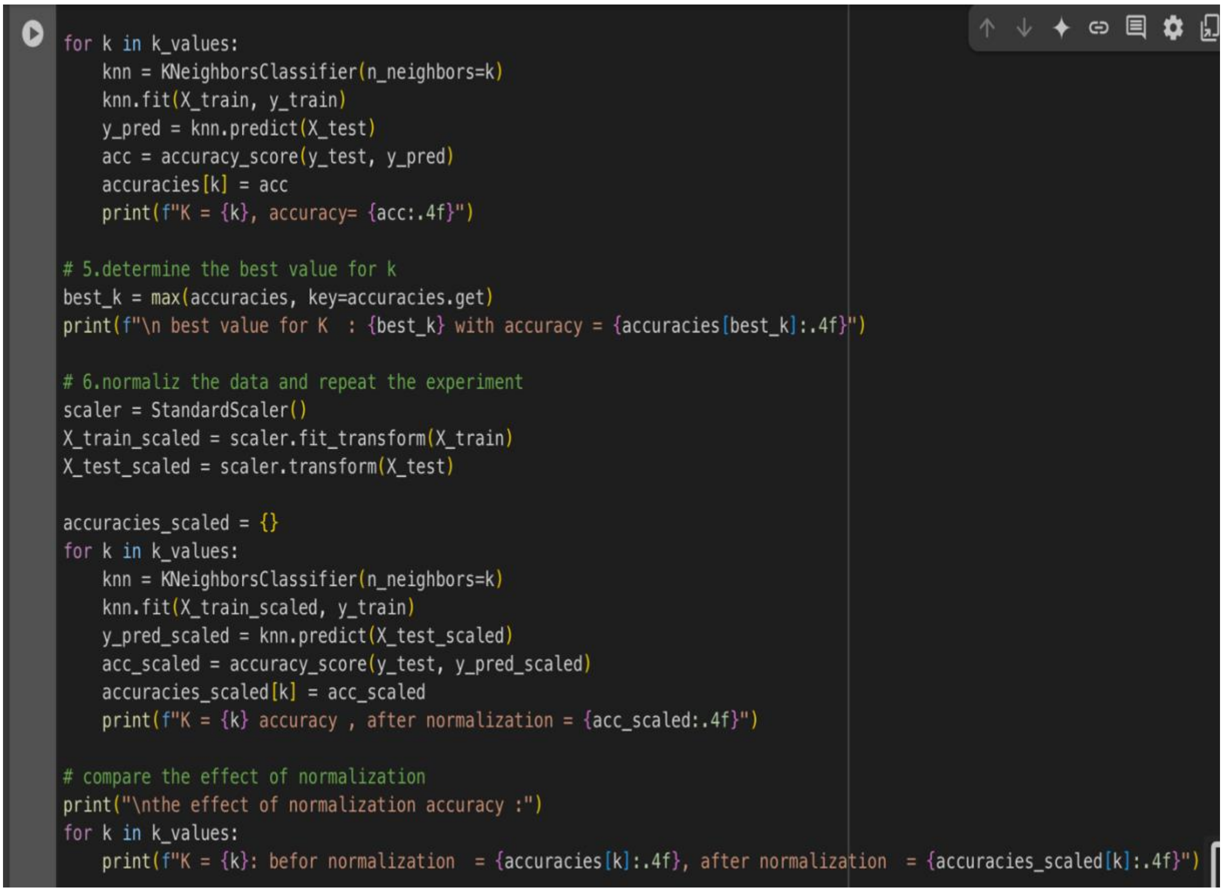
For solution I used 7 steps :

- Download data iris.
- Visualize data in two dimensions by PCA.
- Divid the data to %80 training and %20 testing .
- Apply KNN with different values k=3,5,7.
- Determine the best value for K.
- Normalize the data and repeat the experiment.
- Compare the effect of normalization**.**

Note: I used Google Colab to write the codes .

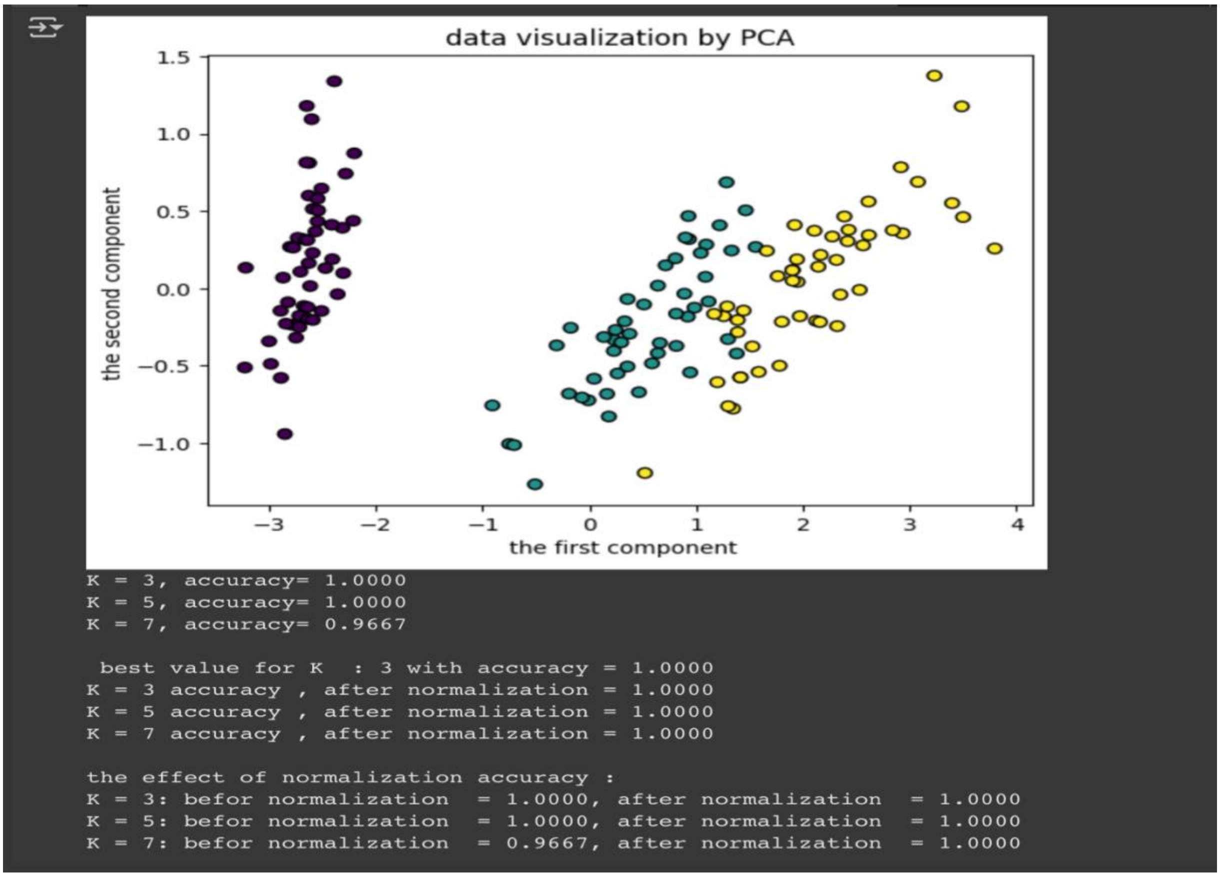


```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 1. download data Iris
iris = datasets.load_iris()
X, y = iris.data, iris.target

# 2.  visualize data in two dimensions by PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolors='k')
plt.xlabel(" the first component")
plt.ylabel("the second component ")
plt.title("data visualization by PCA")
plt.show()

# 3.divid the data into %80 training and %20 testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 4. apply KNN with different values k=3,5,7
k_values = [3, 5, 7]
accuracies = {}
```



```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies[k] = acc
    print(f"K = {k}, accuracy= {acc:.4f}")

# 5.determine the best value for k
best_k = max(accuracies, key=accuracies.get)
print(f"\n best value for K  : {best_k} with accuracy = {accuracies[best_k]:.4f}")

# 6.normaliz the data and repeat the experiment
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

accuracies_scaled = {}
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred_scaled = knn.predict(X_test_scaled)
    acc_scaled = accuracy_score(y_test, y_pred_scaled)
    accuracies_scaled[k] = acc_scaled
    print(f"K = {k} accuracy , after normalization = {acc_scaled:.4f}")

# compare the effect of normalization
print("\nthe effect of normalization accuracy :")
for k in k_values:
    print(f"K = {k}: befor normalization  = {accuracies[k]:.4f}, after normalization  = {accuracies_scaled[k]:.4f}")
```

Output :



```
K = 3, accuracy= 1.0000
K = 5, accuracy= 1.0000
K = 7, accuracy= 0.9667

 best value for K  : 3 with accuracy = 1.0000
K = 3 accuracy , after normalization = 1.0000
K = 5 accuracy , after normalization = 1.0000
K = 7 accuracy , after normalization = 1.0000

the effect of normalization accuracy :
K = 3: befor normalization  = 1.0000, after normalization  = 1.0000
K = 5: befor normalization  = 1.0000, after normalization  = 1.0000
K = 7: befor normalization  = 0.9667, after normalization  = 1.0000
```