```
import numpy as np import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report from sklearn.naive_bayes import
GaussianNB

from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import
RandomForestClassifier


# Paths to the data files

train_file_path = '/content/optdigits.tra' test_file_path = '/content/optdigits.tes'



# Load the training and testing data

train_data = pd.read_csv(train_file_path, header=None) test_data = pd.read_csv(test_file_path,
header=None)



# Separate features and labels

X_train = train_data.iloc[:, :-1].values # All columns except the last one y_train = train_data.iloc[:, -
1].values  # The last column as labels


X_test = test_data.iloc[:, :-1].values  # All columns except the last one y_test = test_data.iloc[:, -
1].values # The last column as labels


# Dictionary to store model names and their accuracies model_accuracies = {}


# Initialize and train the K-Nearest Neighbors (KNN) classifier knn =
KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)


# Make predictions on the test set y_pred = knn.predict(X_test)

model_accuracies['KNN'] = accuracy_score(y_test, y_pred) print(f'Accuracy: {accuracy_score(y_test,
y_pred) * 100:.2f}%')


# Loop over different values of k for i, k in enumerate(neighbors):
```

```
# Setup a k-NN Classifier with k neighbors: knn knn = KNeighborsClassifier(n_neighbors=k)


# Fit the classifier to the training data knn.fit(X_train, y_train)


#Compute accuracy on the training set train_accuracy[i] = knn.score(X_train, y_train)


#Compute accuracy on the testing set test_accuracy[i] = knn.score(X_test, y_test)


# Generate plot

import matplotlib.pyplot as plt

plt.title('k-NN: Varying Number of Neighbors') plt.plot(neighbors, test_accuracy, label = 'Testing
Accuracy') plt.plot(neighbors, train_accuracy, label = 'Training Accuracy') plt.legend()

plt.xlabel('Number of Neighbors') plt.ylabel('Accuracy') plt.show()


nb_model = GaussianNB() nb_model.fit(X_train, y_train) y_pred_nb = nb_model.predict(X_test)

model_accuracies['Naive Bayes'] = accuracy_score(y_test, y_pred_nb) print(f'Accuracy:
{accuracy_score(y_test, y_pred_nb) * 100:.2f}%')



# 2. Decision Tree Classifier

dt_model = DecisionTreeClassifier(random_state=0) dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)

model_accuracies['Decision Tree'] = accuracy_score(y_test, y_pred_dt) print(f'Accuracy:
{accuracy_score(y_test, y_pred_dt) * 100:.2f}%')



# 3. Random Forest Classifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=0) rf_model.fit(X_train,
y_train)

y_pred_rf = rf_model.predict(X_test)

model_accuracies['Random Forest'] = accuracy_score(y_test, y_pred_rf) print(f'Accuracy:
{accuracy_score(y_test, y_pred_rf) * 100:.2f}%')
```

```
# Plotting the test accuracies plt.figure(figsize=(10, 6))

plt.bar(model_accuracies.keys(), model_accuracies.values(), color=['blue', 'yellow', 'green', 'red'])
plt.xlabel('Model')

plt.ylabel('Test Accuracy')

plt.title('Test Accuracy of Different Models')

plt.ylim(0, 1) # Scale the y-axis from 0 to 1 for percentage accuracy plt.show()



# List of classifiers to evaluate classifiers = {

"K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5), "Naive Bayes": GaussianNB(),

"Decision Tree": DecisionTreeClassifier(random_state=0),

"Random Forest": RandomForestClassifier(n_estimators=100, random_state=0)

}


# Dictionary to store performance metrics for each classifier performance_data = []


# Evaluate each classifier

for name, model in classifiers.items(): # Train the model model.fit(X_train, y_train)


# Make predictions on the test set y_pred = model.predict(X_test)


# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)


# Generate classification report

report = classification_report(y_test, y_pred, output_dict=True) precision = report['weighted
avg']['precision']

recall = report['weighted avg']['recall'] f1_score = report['weighted avg']['f1-score']


# Append the results to the performance data performance_data.append({

"Classifier": name, "Accuracy": accuracy, "Precision": precision, "Recall": recall,
"F1 Score": f1_score

})


# Create a DataFrame from the performance data performance_df =
pd.DataFrame(performance_data)


# Display the performance table print(performance_df)


# Plotting the F1 scores plt.figure(figsize=(10, 6))

plt.bar(f1_scores.keys(), f1_scores.values(), color=['pink', 'yellow', 'orange', 'green'])
plt.xlabel('Model')

plt.ylabel('F1 Score')

plt.title('Comparison of F1 Scores for Different Classifiers') plt.ylim(0, 1) # Scale y-axis from 0 to 1

plt.show()
```