This code demonstrates how to use Scikit-learn to perform K-Nearest Neighbors (KNN) classification on the Iris dataset. It loads the dataset, splits it into training and testing sets, trains a KNN model with 3 neighbors on the training data, then predicts and evaluates the model's accuracy on the test data, printing both the accuracy score and a detailed classification report.

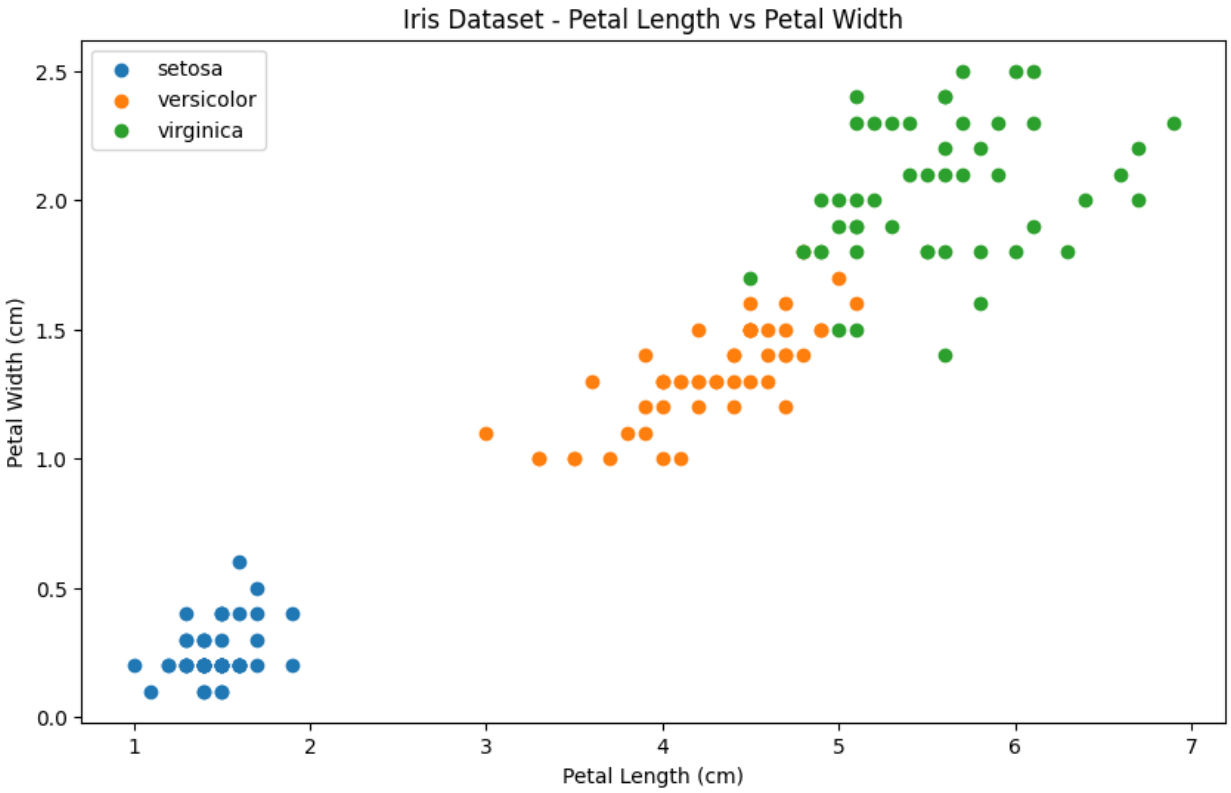
```
# Import libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1. Load the Iris dataset
iris = load_iris()
X = iris.data      # Features (sepal length, sepal width, etc.)
y = iris.target    # Labels (species)

# 2. Split the data into training and testing sets (70% train, 30%
test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# 3. Create the KNN classifier (choose number of neighbors, e.g., k=3)
knn = KNeighborsClassifier(n_neighbors=3)

# 4. Train the model on the training data
knn.fit(X_train, y_train)

# 5. Predict the labels for the test set
y_pred = knn.predict(X_test)

# 6. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=iris.target_names))
```

```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
```

```
weighted avg       1.00      1.00      1.00        45
```

Graph

```
import matplotlib.pyplot as plt
import pandas as pd

# Convert to pandas DataFrame for easier plotting
iris_df = pd.DataFrame(X, columns=iris.feature_names)
iris_df['species'] = iris.target
iris_df['species'] = iris_df['species'].map({0: 'setosa', 1:
'versicolor', 2: 'virginica'})

# Create a scatter plot
plt.figure(figsize=(10, 6))
for species in iris_df['species'].unique():
    subset = iris_df[iris_df['species'] == species]
    plt.scatter(subset['petal length (cm)'], subset['petal width
(cm)'], label=species)

plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Iris Dataset - Petal Length vs Petal Width')
plt.legend()
plt.show()
```



Understanding k in KNN:

Low k (e.g., k=1) → Overfitting: Model memorizes training data, very sensitive to noise.

High k (e.g., k=15+) → Underfitting: Model too generalized, misses patterns in data.

Optimal k (e.g., k=3 to 7) → Good fitting: Balanced generalization and accuracy.

 Modified Code: Show Overfitting, Underfitting, and Good Fitting

Here's a version of your code testing multiple k values to demonstrate:

```
# Import libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

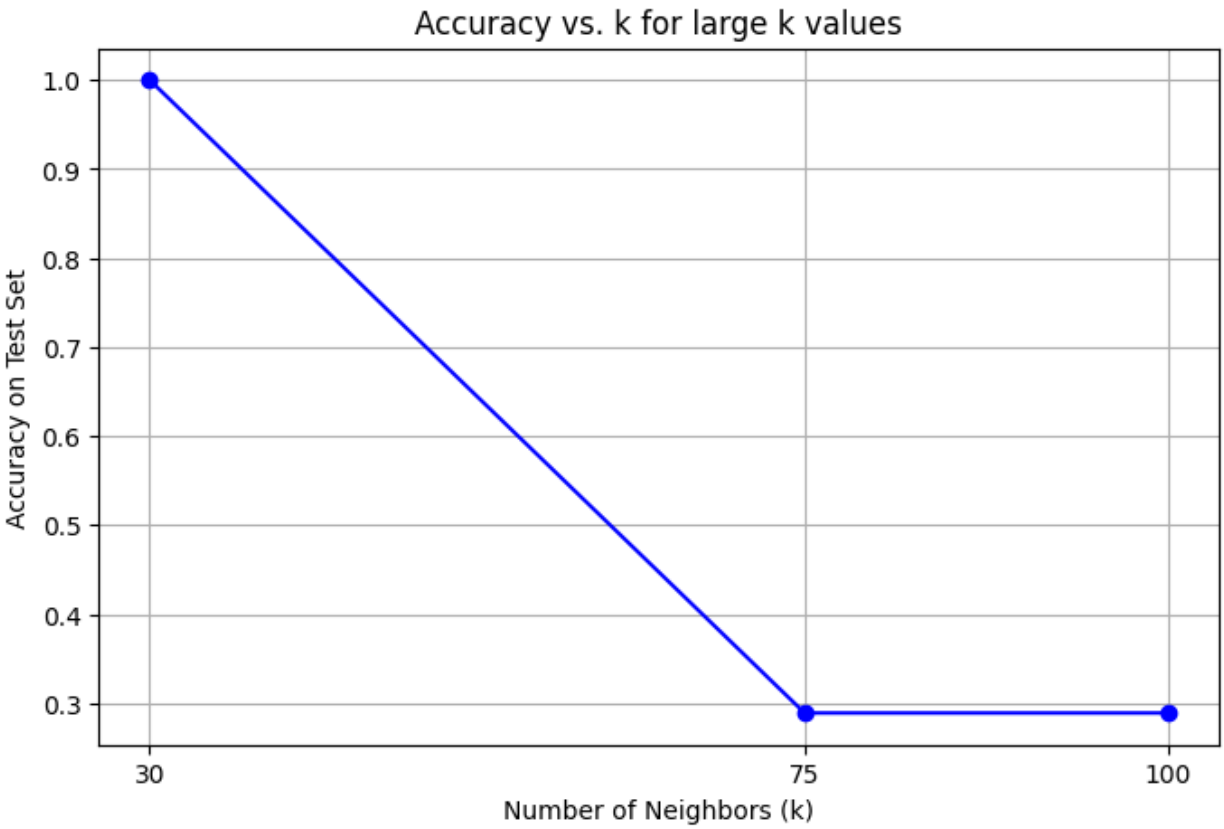
```python
# Try different values of k to observe underfitting, good fitting, and
overfitting
k_values = [30, 75, 100]

for k in k_values:
    print(f"\n--- Results for k = {k} ---")

    # Create and train KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict on test set
    y_pred = knn.predict(X_test)

    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred,
target_names=iris.target_names))
```

```
--- Results for k = 30 ---
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


--- Results for k = 75 ---
Accuracy: 0.29
Classification Report:
              precision    recall  f1-score   support

      setosa       0.00      0.00      0.00        19
  versicolor       0.29      1.00      0.45        13
   virginica       0.00      0.00      0.00        13

    accuracy                           0.29        45
   macro avg       0.10      0.33      0.15        45
weighted avg       0.08      0.29      0.13        45


--- Results for k = 100 ---
```

```
Accuracy: 0.29
Classification Report:
              precision    recall  f1-score   support

      setosa       0.00      0.00      0.00        19
  versicolor       0.29      1.00      0.45        13
   virginica       0.00      0.00      0.00        13

    accuracy                           0.29        45
   macro avg       0.10      0.33      0.15        45
weighted avg       0.08      0.29      0.13        45


/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

Visualize the Effect

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# List of k values you want to test
k_values = [30, 75, 100]
accuracies = []

for k in k_values:
    # Create and train KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict on test set
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)
    print(f"k = {k}, Accuracy = {acc:.2f}")

# Plotting
plt.figure(figsize=(8,5))
plt.plot(k_values, accuracies, marker='o', linestyle='-', color='b')
plt.title("Accuracy vs. k for large k values")
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("Accuracy on Test Set")
plt.xticks(k_values)  # Show ticks only at tested k values
plt.grid(True)
plt.show()


k = 30, Accuracy = 1.00
k = 75, Accuracy = 0.29
k = 100, Accuracy = 0.29
```


Accuracy vs. k for large k values
Accuracy on Test Set
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
30
75
100
Number of Neighbors (k)