IMPORT LIBRARIES

In [42]:
```
from IPython import get_ipython
from IPython.display import display
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pickle
import joblib
from datetime import datetime,timedelta
from sklearn.neighbors import KNeighborsClassifier
```

LOAD DATA

In [43]:
```
df= pd.read_excel("Iris Data.xlsx")
```

In [44]:
```
df.head()
```

Out[44]:

	Column1
0	5,2.3,3.4,0.9,Iris-versicolor
1	5,2.4,4.6,1.6,Iris-virginica
2	5.1,1.9,3.6,0.9,Iris-versicolor
3	5.1,2.2,3.4,0.9,Iris-versicolor
4	5.2,2.4,3.1,1,Iris-versicolor

DATA CLEANING

In [45]:
```
# Split the string in each row into multiple columns
df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']] =
```

In [46]:
```
# Remove the original combined column
df= df.drop('Column1', axis=1)
```

In [47]:
```
df
```

Out[47]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5	2.3	3.4	0.9	Iris-versicolor
1	5	2.4	4.6	1.6	Iris-virginica
2	5.1	1.9	3.6	0.9	Iris-versicolor
3	5.1	2.2	3.4	0.9	Iris-versicolor
4	5.2	2.4	3.1	1	Iris-versicolor
...	...	...	...	...	...
101	7.8	2.5	7	2.2	Iris-virginica
102	7.8	2.7	6.8	1.9	Iris-virginica
103	7.8	2.9	6.2	2.2	Iris-virginica
104	7.8	3.7	6.8	2.1	Iris-virginica
105	8	3.7	6.5	1.9	Iris-virginica

106 rows × 5 columns

In [48]:

```
#Generate descriptive statistics
df.describe()
```

Out[48]:

	sepal_length	sepal_width	petal_length	petal_width	species
count	106	106	106	106	106
unique	29	16	35	16	2
top	6.4	2.9	4.6	1.4	Iris-versicolor
freq	10	20	9	13	57

In [49]:

```
# Check unique species
print("\nUnique Species:")
print(df['species'].unique())
```

```
Unique Species:
['Iris-versicolor' 'Iris-virginica']
```

In [50]:

```
# Display DataFrame info
df.info()
```

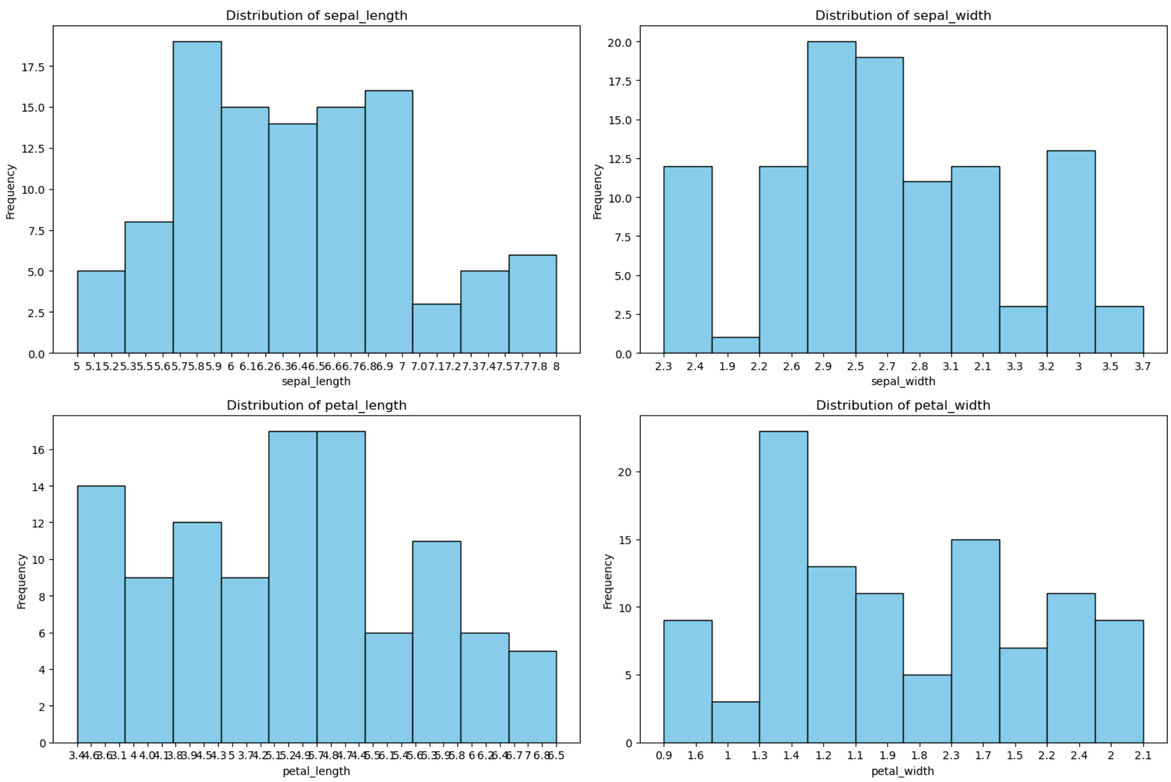
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106 entries, 0 to 105
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   sepal_length  106 non-null    object
 1   sepal_width   106 non-null    object
 2   petal_length  106 non-null    object
 3   petal_width   106 non-null    object
 4   species       106 non-null    object
dtypes: object(5)
memory usage: 4.3+ KB
```

In [51]:
```python
# Calculate and display the correlation matrix for numerical features only
print("\nCorrelation Matrix (Numerical Features Only):")
display(df.select_dtypes(include=['number']).corr())
```

```
Correlation Matrix (Numerical Features Only):
```

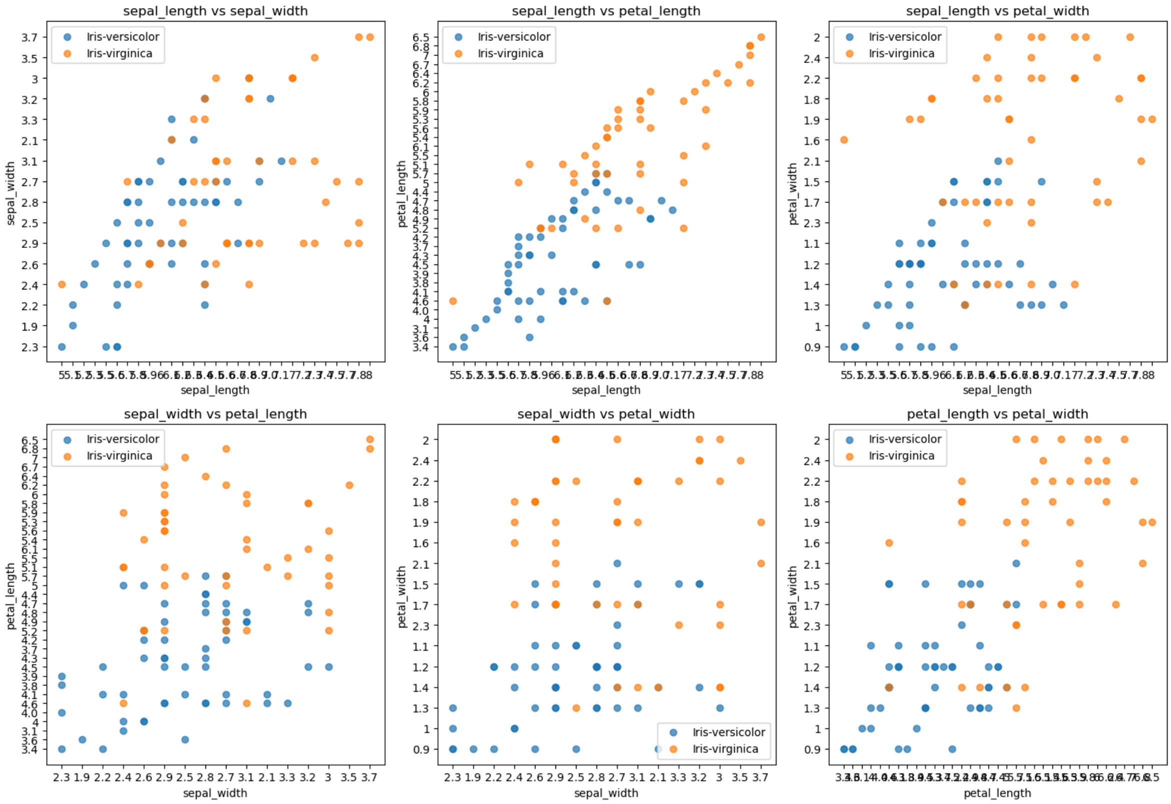
DATA VISUALIZATION

In [52]:
```python
# Histograms for each numerical feature
plt.figure(figsize=(15, 10))
for i, col in enumerate(['sepal_length', 'sepal_width', 'petal_length', 'petal_w
    plt.subplot(2, 2, i + 1)
    plt.hist(df[col], bins=10, color='skyblue', edgecolor='black')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



In [53]:
```python
# Scatter plots for all pairs of numerical features, colored by species
num_features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
num_plots = len(num_features)
plt.figure(figsize=(15, 15))

for i in range(num_plots):
    for j in range(i + 1, num_plots):
        plt.subplot(num_plots - 1, num_plots - 1, (num_plots - 1) * i + j - i *
        for species in df['species'].unique():
          subset = df[df['species'] == species]
          plt.scatter(subset[num_features[i]], subset[num_features[j]], label=sp
        plt.xlabel(num_features[i])
        plt.ylabel(num_features[j])
        plt.title(f'{num_features[i]} vs {num_features[j]}')
        plt.legend()
```

```
plt.tight_layout()
plt.show()
```



QUESTIONS

Question 2.1: Use a Kth-Nearest Neighbour model and the Euclidean distance algorithm to determine the variety of the following observation (6.6, 3.2, 5.1, 1.5). Use relevant theories of the model to justify your choice. (10 marks)

In [54]:
```
# Separate features and target variable (already done, but re-iterating for clar
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']
```

In [55]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
import numpy as np
```

In [56]:
```
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']
```

In [57]:
```
# Split the data into training and testing sets
# This is important to evaluate how the model performs on unseen data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [58]:
```
# Initialize and train a KNN model
# You can choose a value for k based on your previous analysis or cross-validati
# For this example, let's use k=5 as it performed well in the previous task.
knn_model = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_model.fit(X, y)
```

Out[58]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(metric='euclidean')
```

In [59]:

```
# Define the new observation
new_observation = np.array([[6.6, 3.2, 5.1, 1.5]])
```

In [60]:

```
# Define the new observation as a pandas DataFrame with the same column names as
new_observation_data = {'sepal_length': [6.6], 'sepal_width': [3.2], 'petal_leng
new_observation_df = pd.DataFrame(new_observation_data)
```

In [61]:

```
# Predict the species of the new observation using the DataFrame
predicted_species = knn_model.predict(new_observation_df)

print(f"The predicted species for the observation (6.6, 3.2, 5.1, 1.5) is: {pred
```

```
The predicted species for the observation (6.6, 3.2, 5.1, 1.5) is: Iris-virginica
```

Question 2.2 When we use the K-NN model to classify an unseen instance, a) what might be the problem if we set the value of K to an even number? Please provide the evidence combining the results from Question 2.1. b) What could be a possible solution?

The main problem with setting the value of K to an even number in a K-Nearest Neighbors (KNN) classifier is the potential for a tie in the voting process among the nearest neighbors.

In [98]:

```
# Import necessary libraries
from sklearn.metrics import confusion_matrix, accuracy_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
```

In [101...

```
from sklearn.preprocessing import StandardScaler
```

In [102...

```
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [103...

```
# The observation to predict as a dictionary or list
observation_data = [[6.6, 3.2, 5.1, 1.5]]

# Define the feature names (must match the names used to train the scaler)
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

# Create a DataFrame for the observation with feature names
observation_df = pd.DataFrame(observation_data, columns=features)

# Scale the observation using the same scaler fitted on the training data
observation_scaled = scaler.transform(observation_df)
```

In [106...

```
# Now you can use observation_scaled for prediction
predicted_variety = knn.predict(observation_scaled)
```

```
print(f"The predicted variety for the observation (6.6, 3.2, 5.1, 1.5) is: {pred
```

```
The predicted variety for the observation (6.6, 3.2, 5.1, 1.5) is: Iris-versicolo
r
```

In [105…

```
# Test K=3 (Odd Number)
knn_odd = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn_odd.fit(X_train, y_train)
predicted_odd = knn_odd.predict(observation_scaled)
print(f"Prediction with K=3 (odd): {predicted_odd[0]}")

# Test K=10 (Even Number)
knn_even = KNeighborsClassifier(n_neighbors=10, metric='euclidean')
knn_even.fit(X_train, y_train)
predicted_even = knn_even.predict(observation_scaled)
print(f"Prediction with K=10 (even): {predicted_even[0]}")
```

```
Prediction with K=3 (odd): Iris-versicolor
Prediction with K=10 (even): Iris-versicolor
```

Question 2.3: Apply 5-NN, 7-NN, and 9-NN models, a) show the confusion matrix, and b) calculate the accuracy of these models. c) Which model do you think is the best? Please explain your choice. (16 marks)

In [86]:

```
from sklearn.neighbors import KNeighborsClassifier
k_values = [5, 7, 9]
results = {}

for k in k_values:
    print(f"\nEvaluating K={k}...")
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train, y_train)
    y_pred = knn_model.predict(X_test)
```

```
Evaluating K=5...

Evaluating K=7...

Evaluating K=9...
```

In [87]:

```
# a) Show the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[13  0]
 [ 3  6]]
```

In [88]:

```
# b) Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

results[k] = {'confusion_matrix': cm, 'accuracy': accuracy}
```

```
Accuracy: 0.8636
```

In [89]:

```
# c) Which model do you think is the best? Explain your choice.
print("\nComparison of Models:")
best_k = None
```

```python
best_accuracy = -1

for k, result in results.items():
    print(f"K={k}: Accuracy = {result['accuracy']:.4f}")
    if result['accuracy'] > best_accuracy:
        best_accuracy = result['accuracy']
        best_k = k
```

```
Comparison of Models:
K=9: Accuracy = 0.8636
```

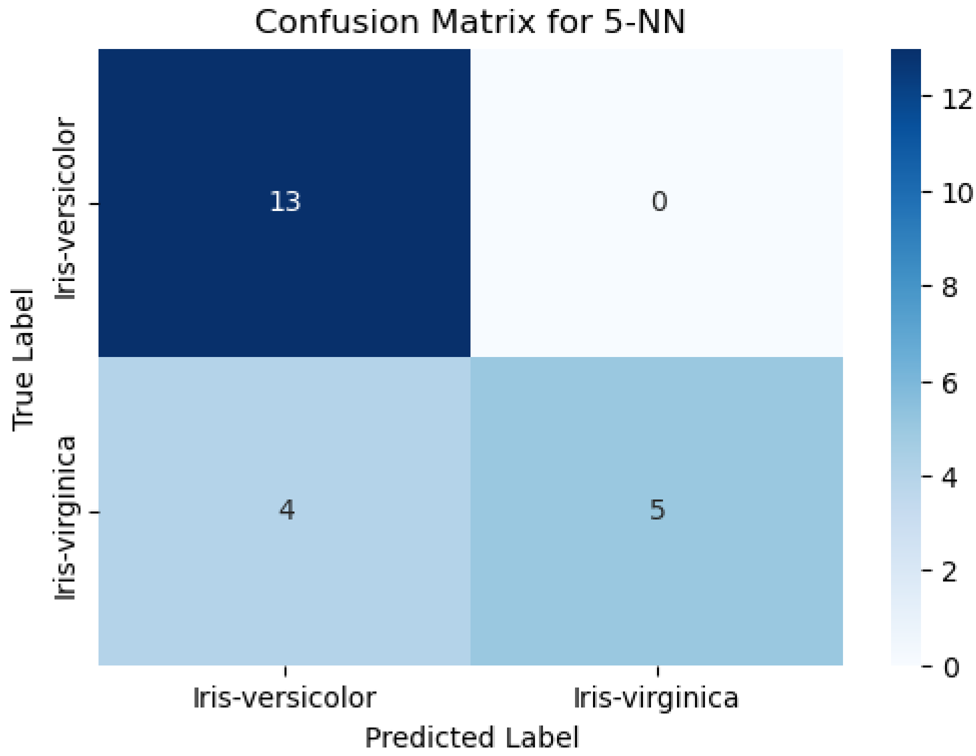
In [93]:

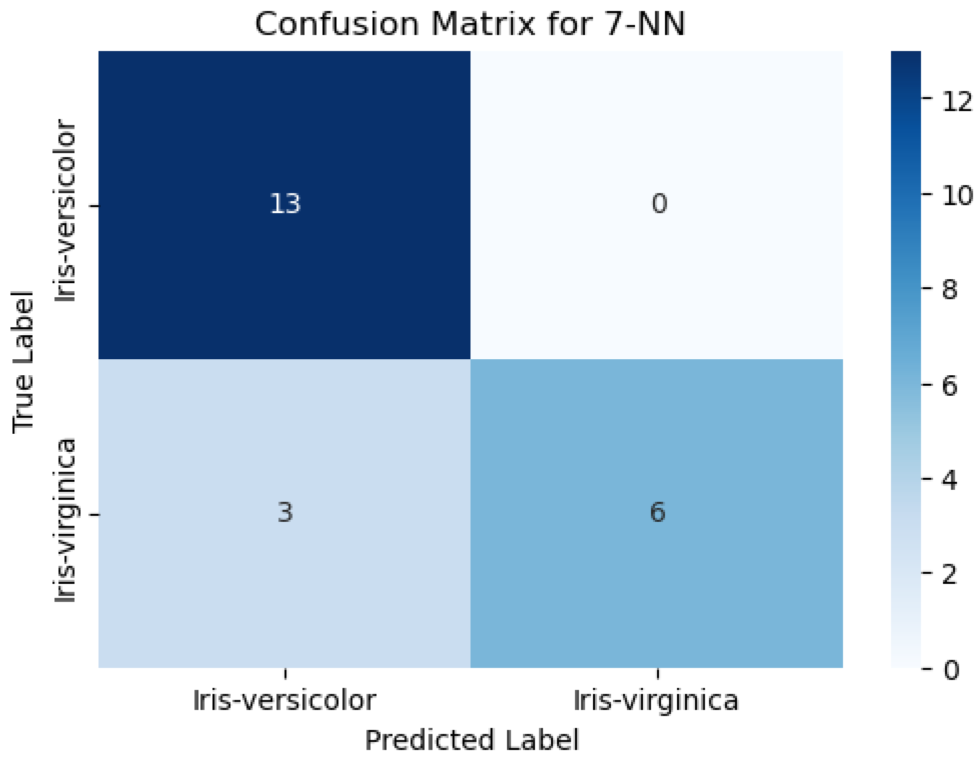
```python
# Define K values
k_values = [5, 7, 9]
confusion_matrices = {}

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    confusion_matrices[k] = cm
    print(f"Accuracy for {k}-NN: {accuracy_score(y_test, y_pred)}\n")
```

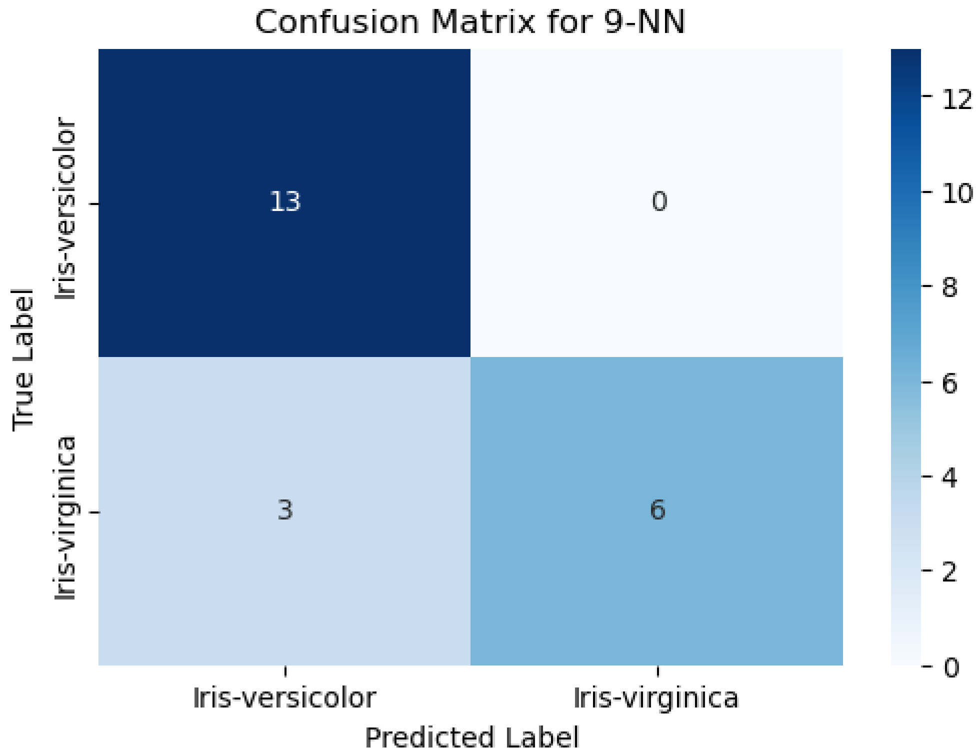
```
Accuracy for 5-NN: 0.8181818181818182

Accuracy for 7-NN: 0.8636363636363636

Accuracy for 9-NN: 0.8636363636363636
```

In [94]:

```python
# Define function to plot confusion matrix
def plot_confusion_matrix(cm, k):
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Iris-versic
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix for {k}-NN")
    plt.show()

# Plot confusion matrices for 5-NN, 7-NN, and 9-NN
k_values = [5, 7, 9]
for k in k_values:
    plot_confusion_matrix(confusion_matrices[k], k)
```


Confusion Matrix for 5-NN
True Label
Iris-versicolor
Iris-virginica
13
0
4
5
Predicted Label
Iris-versicolor
Iris-virginica
0
2
4
6
8
10
12


Confusion Matrix for 7-NN
True Label
Iris-versicolor
Iris-virginica
13
0
3
6
Predicted Label
Iris-versicolor
Iris-virginica
0
2
4
6
8
10
12



In [95]:
```
print(f"\nBased on the accuracies on the test set, the model with K={best_k} app
```

```
Based on the accuracies on the test set, the model with K=9 appears to be the bes
t.
```

In [96]:
```
# Explanation for choosing the best model:
print("\nExplanation:")
print("The best model is typically the one that achieves the highest accuracy on
print("A higher accuracy indicates that the model is better at correctly classif
print("While a more thorough evaluation might involve cross-validation, for this
print("We observe the accuracies for K=5, 7, and 9 and select the K value that y
print("It's important to note that the 'best' K can vary depending on the datase
```

```
Explanation:
The best model is typically the one that achieves the highest accuracy on the tes
t set.
A higher accuracy indicates that the model is better at correctly classifying uns
een instances.
While a more thorough evaluation might involve cross-validation, for this compari
son, we are using the accuracy on the single test split.
We observe the accuracies for K=5, 7, and 9 and select the K value that yielded t
he highest accuracy.
It's important to note that the 'best' K can vary depending on the dataset and th
e specific split of data into training and testing sets.
```