Predictions on Iris dataset

Data Scientist : Simon Pierre Charbel

In this code we will apply 5 machine learning models to the iris dataset.

In [44]:

```python
# Import necessary libraries and packages
import pandas as pd  # For data manipulation and analysis
import numpy as np  # For numerical operations

from sklearn.datasets import load_iris  # To load the Iris dataset

from sklearn.model_selection import train_test_split  # For splitting t

from sklearn.svm import SVC # Import the Support Vector Machine (SVM) c
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score  # For calculating the accur
from sklearn.metrics import confusion_matrix # For creating a confusion
from sklearn.metrics import classification_report # For generating a cl

import pickle # Save the module

# For data visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Suppress all warnings
import warnings
warnings.filterwarnings("ignore")

file_path = r'C:\Users\simon\AppData\Roaming\JetBrains\PyCharmCE2022.2\
```

1. Loading and pre-processing the data

In [45]:
```
# Load the Iris dataset
iris = load_iris()

# Create a pandas DataFrame
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Add the target column to the DataFrame
iris_df['target'] = iris.target
# Add the target names as a new column
iris_df['species'] = iris.target_names[iris.target]

# Drop the Duplicates
iris_df.drop_duplicates(inplace=True)

# Print shape and the first few rows of the dataframe
print(f"DataFrame Shape: {iris_df.shape[0]} rows and {iris_df.shape[1]}
iris_df.head()
```

```
DataFrame Shape: 149 rows and 6 columns (shape).
```

Out[45]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

2. Spliting the data into train (75%) and test (25%)

In [46]:
```
# Separating the independent variables (features) from the dependent va
X = iris_df.iloc[:, 0:4]  # X contains the feature columns (sepal lengt
y = iris_df.iloc[:, -2]   # y contains the target column (species)

# Splitting the dataset into training and testing sets
# Shuffle the dataset randomly
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
                                                    random_state=42)
```

3. Prediction with different models

In [47]:
```
# Mapping dictionary to convert class labels
class_labels = {0: 'Iris Setosa', 1: 'Iris Versicolor', 2: 'Iris Virgin
```

3.1 SVM

In [48]:

```
# Create a Support Vector Machine (SVM) classifier
svm = SVC()

# Train the SVM classifier on the training data
svm.fit(X_train, y_train)
```

Out[48]:

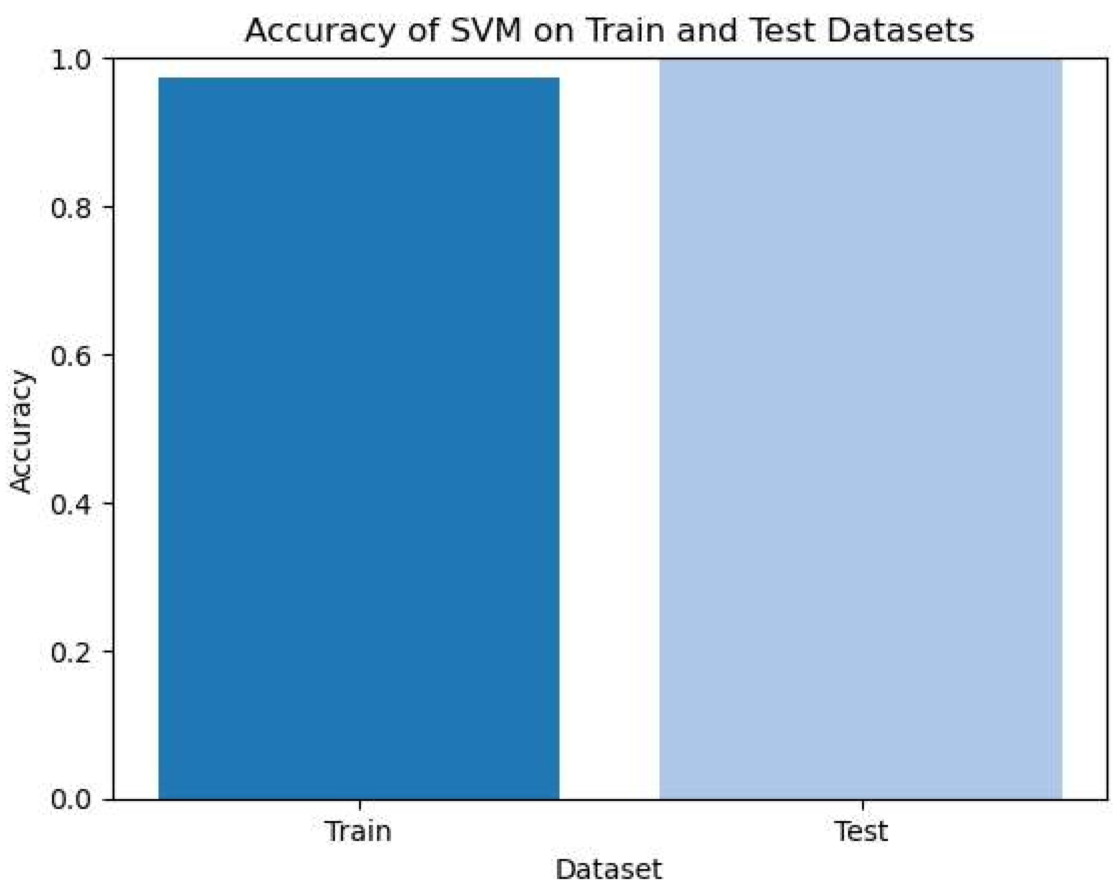
```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

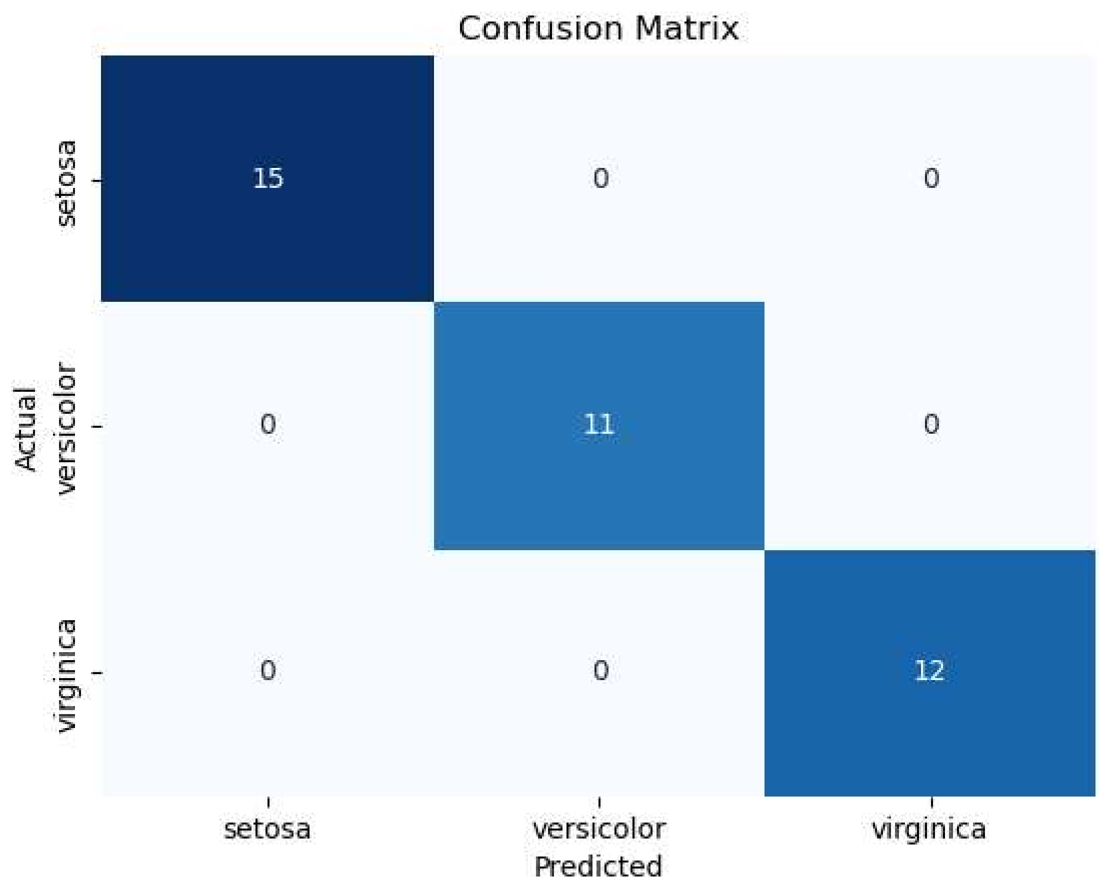
In [49]:

```python
# Make predictions on both the training and test data
train_pred_svm = svm.predict(X_train)
test_pred_svm = svm.predict(X_test)

# Evaluate the SVM classifier
# Calculate accuracy for both datasets
train_accuracy_svm = accuracy_score(y_train, train_pred_svm)
print(f'Train Accuracy : {round(train_accuracy_svm, 2)*100}%')
test_accuracy_svm = accuracy_score(y_test, test_pred_svm)
print(f'Test Accuracy  : {round(test_accuracy_svm, 2)*100}%')


# Create a bar chart to visualize the accuracy of the train and test da
labels = ['Train', 'Test']
accuracy_values_svm = [train_accuracy_svm, test_accuracy_svm]

blue_palette = ['#1f77b4', '#aec7e8']
plt.bar(labels, accuracy_values_svm, color=blue_palette)
plt.ylim(0, 1)  # Set the y-axis limits to range from 0 to 1 (100%)
plt.xlabel('Dataset')
plt.ylabel('Accuracy')
plt.title('Accuracy of SVM on Train and Test Datasets')
plt.show()
```

```
Train Accuracy : 97.0%
Test Accuracy  : 100.0%
```



In [50]:
```
# Visualize the confusion matrix as a heatmap
conf_matrix = confusion_matrix(y_test, test_pred_svm)
sns.heatmap(conf_matrix, annot=True, fmt='d',
            cmap='Blues', cbar=False, xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.figure(figsize=(4, 4))
plt.show()

# Generate a classification report
class_report = classification_report(y_test, test_pred_svm)
print("\nClassification Report:")
print(class_report)
```




```
<Figure size 400x400 with 0 Axes>
```

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

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

In [51]:

```
# Prediction
# Create a new data point with custom input values
sepal_length = 6
sepal_width = 4
petal_length = 4
petal_width = 4

new_data_point = np.array([[sepal_length, sepal_width, petal_length, pe

# Use the model to make predictions
prediction = svm.predict(new_data_point)
predicted_class_label = svm.predict(new_data_point)[0]  # Get the predi
predicted_species = class_labels[predicted_class_label]  # Map to speci

print(f'Predicted Species: {predicted_species} ({predicted_class_label}
```

```
Predicted Species: Iris Virginica (2)
```

In [52]:

```
# Save the mode
with open(file_path+'\svm_model.pkl', 'wb') as model_file:
    pickle.dump(svm, model_file)
print('Model Saved!')


# Load the SVM model from a file
#with open(file_path+'\svm_model.pkl', 'rb') as model_file:
#    loaded_svm = pickle.load(model_file)

```

```
Model Saved!
```

3.2 Logistic Regression

In [53]:

```
# Create a Logistic Regression classifier
logistic_reg = LogisticRegression(max_iter=1000)

# Train the LR classifier on the training data
logistic_reg.fit(X_train, y_train)
```

Out[53]:

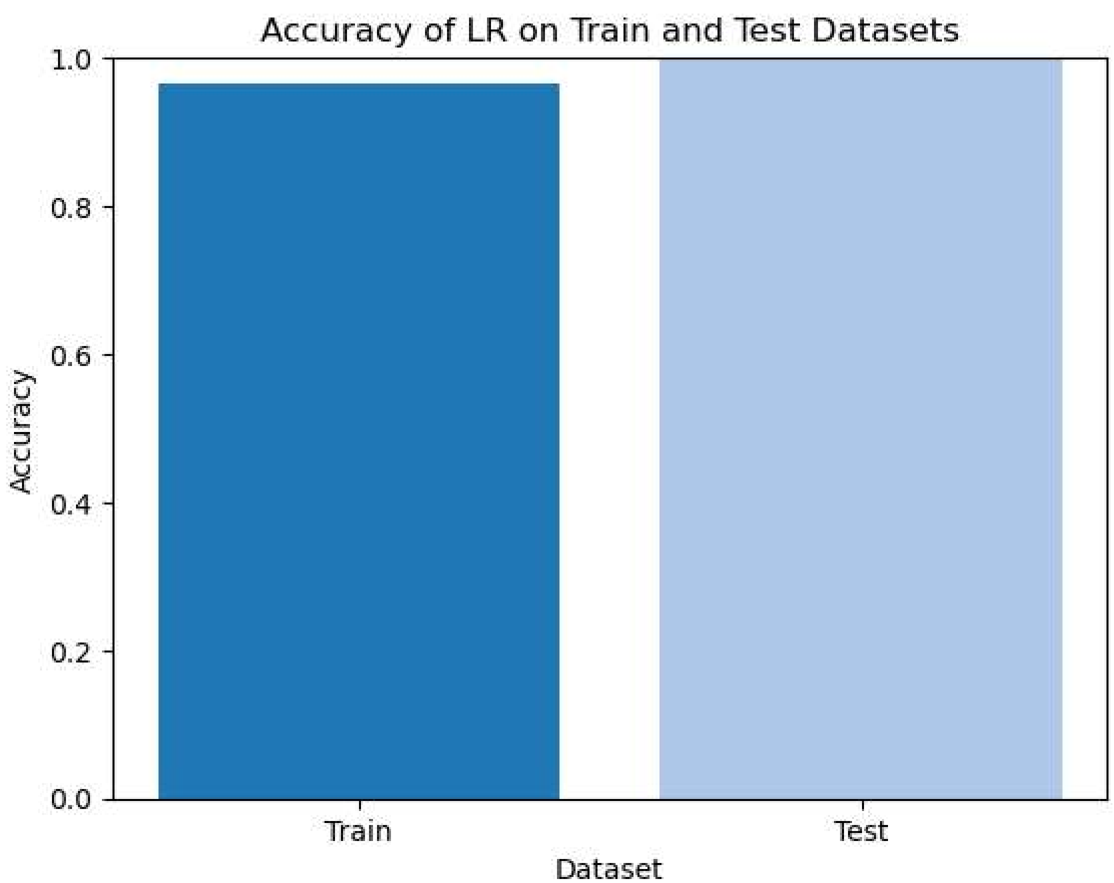
```
LogisticRegression(max_iter=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [54]:

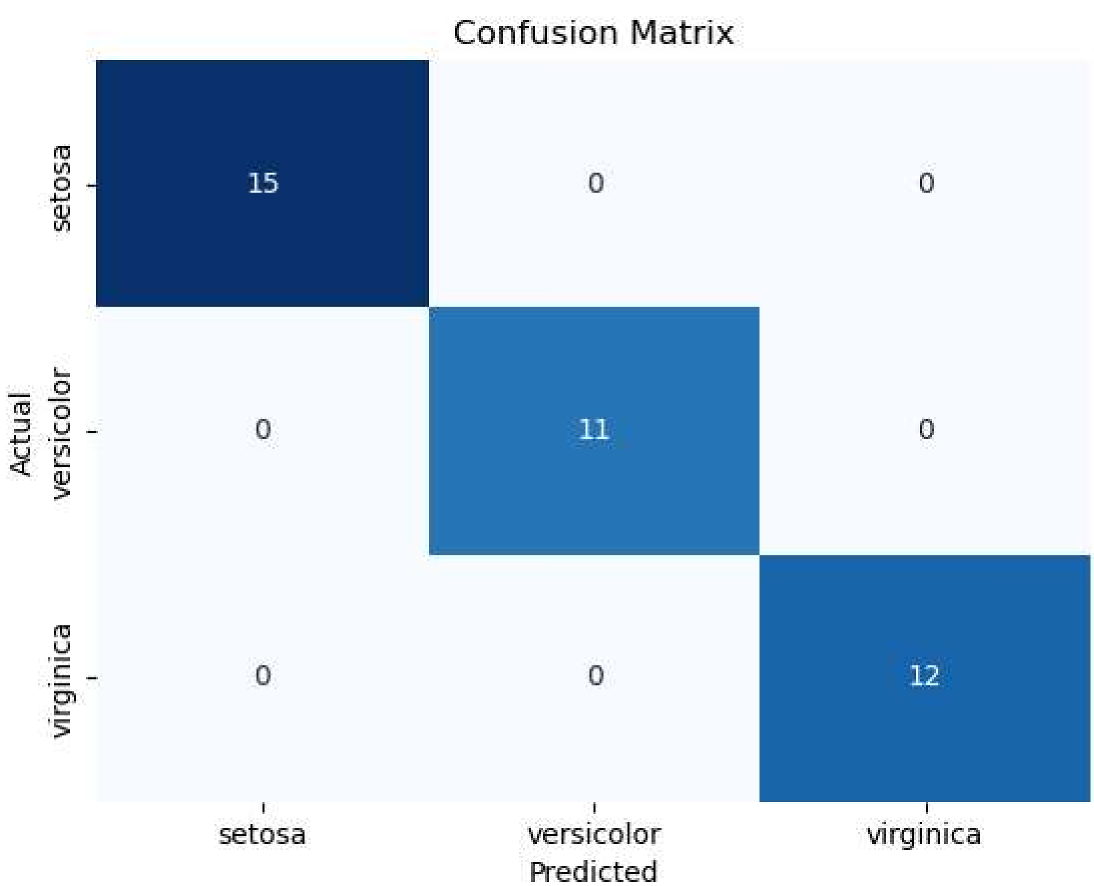
```
# Make predictions on both the training and test data
train_pred_logistic_reg = logistic_reg.predict(X_train)
test_pred_logistic_reg = logistic_reg.predict(X_test)

# Evaluate the LR classifier
# Calculate accuracy for both datasets
train_accuracy_logistic_reg = accuracy_score(y_train, train_pred_logist
print(f'Train Accuracy : {round(train_accuracy_logistic_reg, 2)*100}%')
test_accuracy_logistic_reg = accuracy_score(y_test, test_pred_logistic_
print(f'Test Accuracy  : {round(test_accuracy_logistic_reg, 2)*100}%')


# Create a bar chart to visualize the accuracy of the train and test da
labels = ['Train', 'Test']
accuracy_values_logistic_reg = [train_accuracy_logistic_reg, test_accur

blue_palette = ['#1f77b4', '#aec7e8']
plt.bar(labels, accuracy_values_logistic_reg, color=blue_palette)
plt.ylim(0, 1)  # Set the y-axis limits to range from 0 to 1 (100%)
plt.xlabel('Dataset')
plt.ylabel('Accuracy')
plt.title('Accuracy of LR on Train and Test Datasets')
plt.show()
```

```
Train Accuracy : 96.0%
Test Accuracy  : 100.0%
```



In [55]:

```
# Visualize the confusion matrix as a heatmap
conf_matrix = confusion_matrix(y_test, test_pred_logistic_reg)
sns.heatmap(conf_matrix, annot=True, fmt='d',
            cmap='Blues', cbar=False, xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.figure(figsize=(4, 4))
plt.show()

# Generate a classification report
class_report = classification_report(y_test, test_pred_logistic_reg)
print("\nClassification Report:")
print(class_report)
```




```
<Figure size 400x400 with 0 Axes>
```

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

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

In [56]:

```
# Prediction
# Create a new data point with custom input values
sepal_length = 1
sepal_width = 1
petal_length = 1
petal_width = 1

new_data_point = np.array([[sepal_length, sepal_width, petal_length, pe

# Use the model to make predictions
prediction = logistic_reg.predict(new_data_point)
predicted_class_label = logistic_reg.predict(new_data_point)[0]  # Get
predicted_species = class_labels[predicted_class_label]  # Map to speci

print(f'Predicted Species: {predicted_species} ({predicted_class_label}
```

```
Predicted Species: Iris Setosa (0)
```

In [57]:

```
# Save the mode
with open(file_path+'\logistic_reg_model.pkl', 'wb') as model_file:
    pickle.dump(logistic_reg, model_file)
print('Model Saved!')


# Load the LR model from a file
#with open(file_path+'\logistic_reg_model.pkl', 'rb') as model_file:
#    loaded_svm = pickle.load(model_file)
```

```
Model Saved!
```

3.3 Decision Tree

In [58]:
```python
# Create Decision Tree classifer object
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Make predictions on both the training and test data
train_pred_dt = dt.predict(X_train)
test_pred_dt = dt.predict(X_test)

# Evaluate the DT classifier
# Calculate accuracy for both datasets
train_accuracy_dt = accuracy_score(y_train, train_pred_dt)
print(f'Train Accuracy : {round(train_accuracy_dt, 2)*100}%')
test_accuracy_dt = accuracy_score(y_test, test_pred_dt)
print(f'Test Accuracy  : {round(test_accuracy_dt, 2)*100}%')

# Generate a classification report
class_report = classification_report(y_test, test_pred_dt)
print("\nClassification Report:")
print(class_report)

# Save the mode
with open(file_path+'\dt_model.pkl', 'wb') as model_file:
    pickle.dump(dt, model_file)
print('Model Saved!')
```

```
Train Accuracy : 100.0%
Test Accuracy  : 100.0%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38

Model Saved!
```

3.4 K-NN

In [59]:

```python
# Create Decision Tree classifer object
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions on both the training and test data
train_pred_knn = knn.predict(X_train)
test_pred_knn = knn.predict(X_test)

# Evaluate the DT classifier
# Calculate accuracy for both datasets
train_accuracy_knn = accuracy_score(y_train, train_pred_knn)
print(f'Train Accuracy : {round(train_accuracy_knn, 2)*100}%')
test_accuracy_knn = accuracy_score(y_test, test_pred_knn)
print(f'Test Accuracy  : {round(test_accuracy_knn, 2)*100}%')

# Generate a classification report
class_report = classification_report(y_test, test_pred_knn)
print("\nClassification Report:")
print(class_report)

# Save the mode
with open(file_path+'\knn_model.pkl', 'wb') as model_file:
    pickle.dump(knn, model_file)
print('Model Saved!')
```

```
Train Accuracy : 95.0%
Test Accuracy  : 100.0%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38

Model Saved!
```

3.5 Gaussian Naive Bayes

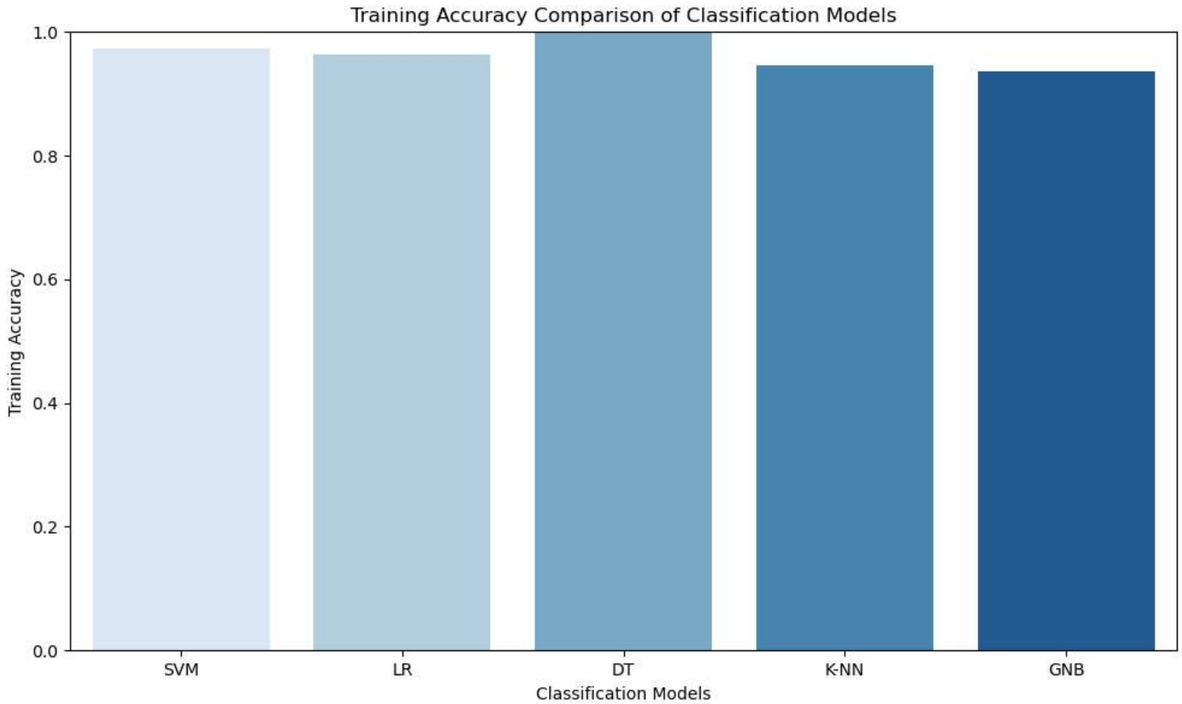
In [60]:
```python
# Create Decision Tree classifer object
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Make predictions on both the training and test data
train_pred_gnb = gnb.predict(X_train)
test_pred_gnb= gnb.predict(X_test)

# Evaluate the DT classifier
# Calculate accuracy for both datasets
train_accuracy_gnb = accuracy_score(y_train, train_pred_gnb)
print(f'Train Accuracy : {round(train_accuracy_gnb, 2)*100}%')
test_accuracy_gnb = accuracy_score(y_test, test_pred_gnb)
print(f'Test Accuracy  : {round(test_accuracy_gnb, 2)*100}%')

# Generate a classification report
class_report = classification_report(y_test, test_pred_gnb)
print("\nClassification Report:")
print(class_report)

# Save the mode
with open(file_path+'\gnb_model.pkl', 'wb') as model_file:
    pickle.dump(gnb, model_file)
print('Model Saved!')
```

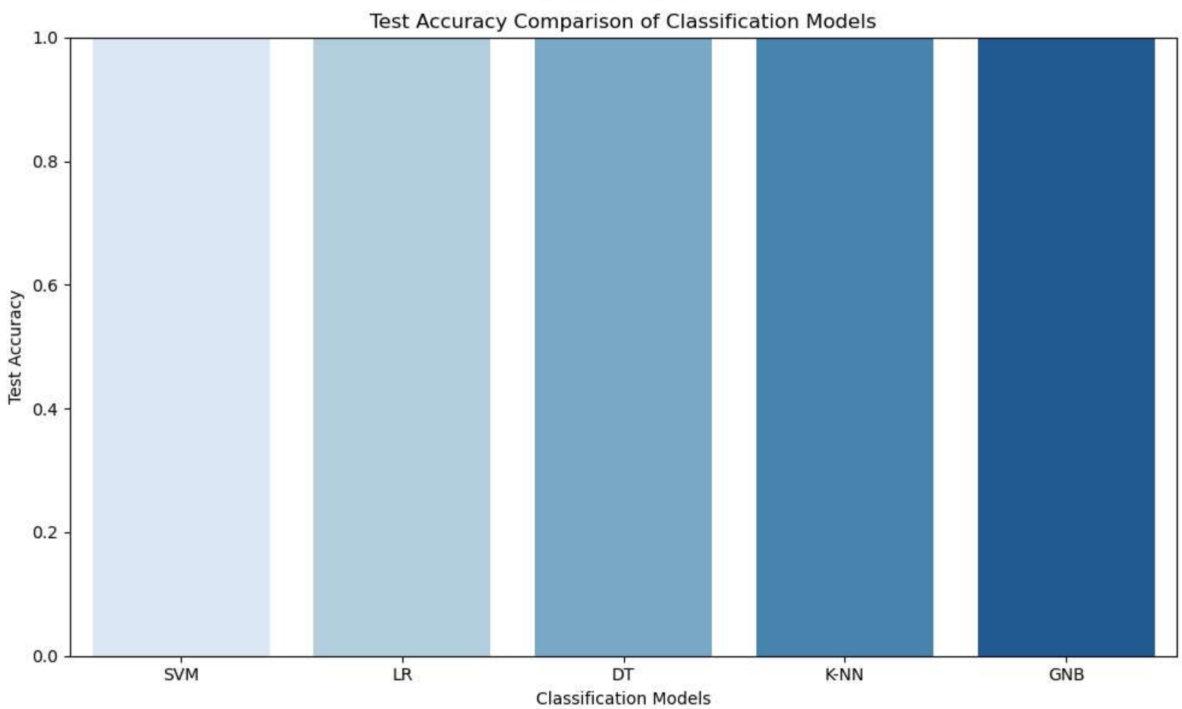
```
Train Accuracy : 94.0%
Test Accuracy  : 100.0%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38

Model Saved!
```

4. Compare the models

In [61]:

```
# Model names and their corresponding accuracies
models = ['SVM', 'LR', 'DT', 'K-NN', 'GNB']
train_accuracies = [train_accuracy_svm, train_accuracy_logistic_reg, tr
test_accuracies = [test_accuracy_svm, test_accuracy_logistic_reg, test_

# Create subplots with two rows and one column
fig, axes = plt.subplots(2, 1, figsize=(10, 12))

# Plot the training accuracies
sns.barplot(x=models, y=train_accuracies, ax=axes[0], palette="Blues")
axes[0].set_xlabel('Classification Models')
axes[0].set_ylabel('Training Accuracy')
axes[0].set_title('Training Accuracy Comparison of Classification Model
axes[0].set_ylim(0, 1)  # Set the y-axis limits to range from 0 to 1 (1

# Plot the test accuracies
sns.barplot(x=models, y=test_accuracies, ax=axes[1], palette="Blues")
axes[1].set_xlabel('Classification Models')
axes[1].set_ylabel('Test Accuracy')
axes[1].set_title('Test Accuracy Comparison of Classification Models')
axes[1].set_ylim(0, 1)  # Set the y-axis limits to range from 0 to 1 (1

# Adjust spacing between subplots
plt.tight_layout()

# Show the combined plot
plt.show()

```





- Conclusion:

  In summary, all models achieved 100% accuracy on the testing data, indicating their strong performance in classifying the Iris dataset. However, when evaluating their accuracy on the training data, the Decision Tree (DT) model outperformed the other models, demonstrating its effectiveness in capturing patterns within the dataset.