Enhancing Iris Flower Classification Using Machine Learning Algorithms

Sk. Zaafira Yumn(23bce9719)
M.Narendra Kumar(23bce20218)
Punnam Durga Manoj(23bce20072)

April 2025

1 Abstract

Classification of Iris flowers into different species based on their physical properties is a fundamental machine learning and pattern recognition problem. The aim of this project is to design a predictive model from the Iris dataset, consisting of 150 instances with four numeric features: sepal length, sepal width, petal length, and petal width. We have tried a few supervised learning algorithms such as Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, and Support Vector Machine (SVM) for multi-class classification and have compared them based on accuracy, precision, recall, and F1-score.

Following data preprocessing and splitting the data into training and test sets, we found that the Support Vector Machine classifier worked best with high generalization for all three classes, namely Setosa, Versicolor, and Virginica. Visualizations such as confusion matrices and decision boundaries were used to further investigate model performance.

This study puts into perspective the importance of model comparison and selection in classification issues and provides an accessible introduction to machine learning pipelines. The Iris dataset, while simple, remains a great vehicle to get insight into basic ML concepts and to experiment with classification models under clean conditions.

2 Introduction

Machine learning has increasingly become a central component of data-driven decision-making, enabling systems to learn automatically from data

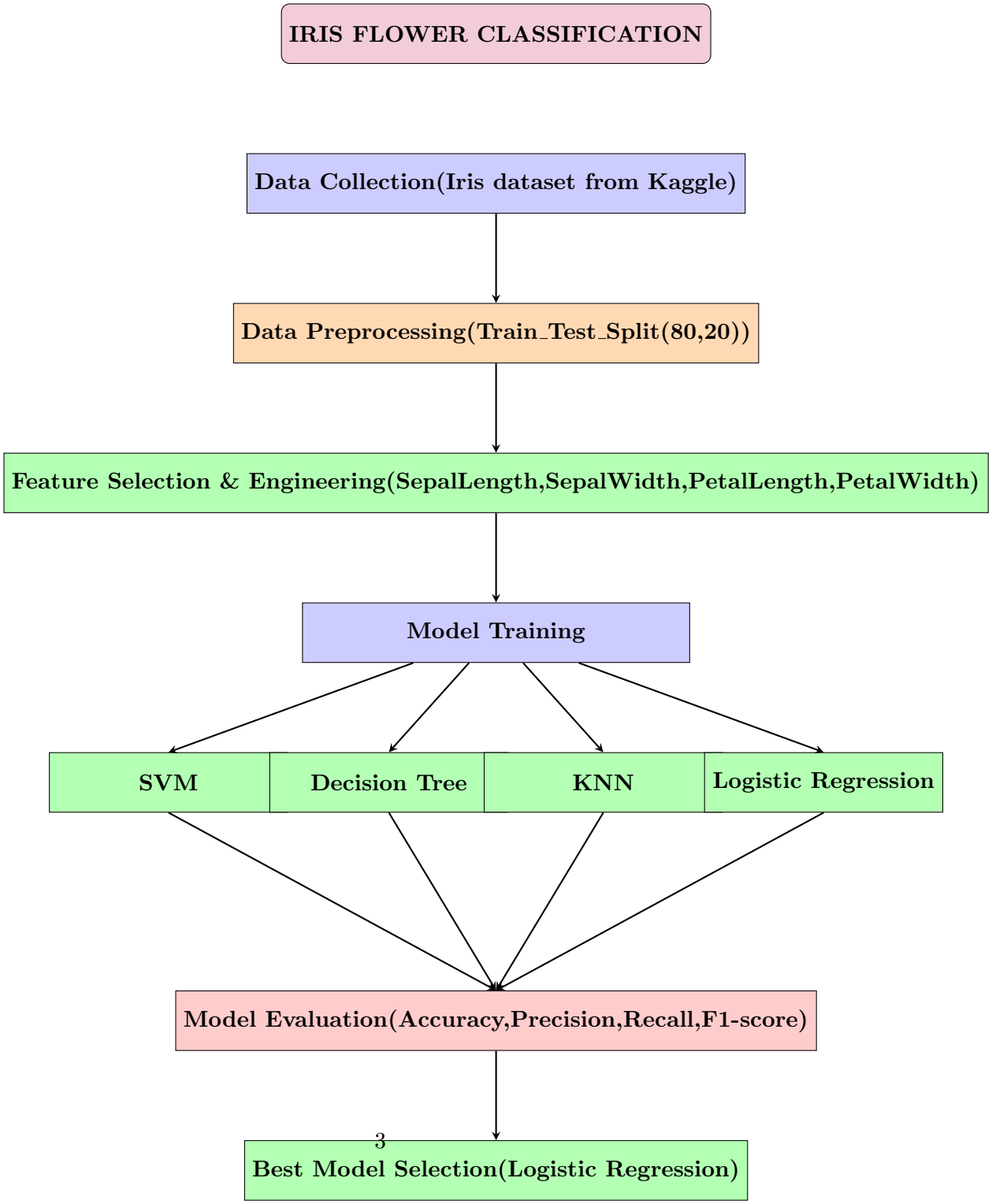
and make predictions with little or no human intervention. Classification is perhaps one of the simplest applications of machine learning, where the goal is to assign input data to one of a number of predefined classes. Perhaps one of the most widely used datasets to evaluate classification algorithms is the Iris flower dataset, originally described by Sir Ronald A. Fisher in 1936 and now widely used as a standard test for supervised learning algorithms.

The Iris database has 150 iris flower samples, and each sample belongs to one of the three species: Iris setosa, Iris versicolor, and Iris virginica. All samples have four numeric features: sepal length, sepal width, petal length, and petal width. The objective of this project is to develop a machine learning system that can accurately determine the species of an iris flower based on these features.

In this project, we implement and compare various supervised learning algorithms such as Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, and Support Vector Machine (SVM). Data are preprocessed, visualized, and split into training and test sets in order to quantify the performance of the models with common metrics including accuracy, precision, recall, and F1-score.

Although the issue itself is simple, this project is an excellent introduction to understanding the end-to-end process of building classification models. It provides good insight into model choice, model selection, and machine learning system explainability. Additionally, comparison of multiple models highlights the strengths and weaknesses of the various models in a controlled setting.

3 Architecture Diagram



4 Dataset Used

Dataset Description The data set utilized for this project is the popular Iris flower data set, which contains measurements of three species of iris. The data set is commonly used for demonstrating classification models in machine learning since it is basic and well-balanced in class distribution.

- Source: First advocated by Ronald A. Fisher in 1936.
- Size: 150 complete entries (rows)
- Features: 4 numeric features, 1 categorical target
- sepal length (in cm)
- sepal width (in cm)
- petal length (in cm)
- petal width (in cm)
- Target Class: species (with 3 different classes)

ClassIris-setosa 50 Iris-versicolor 50 Iris-virginica 50 Every class consists of an equal number of 50 samples, giving us an evenly balanced dataset—ideal for training and testing classification models.

4.1 Basic Statistics:

Below are some of the numerical features' key statistics:

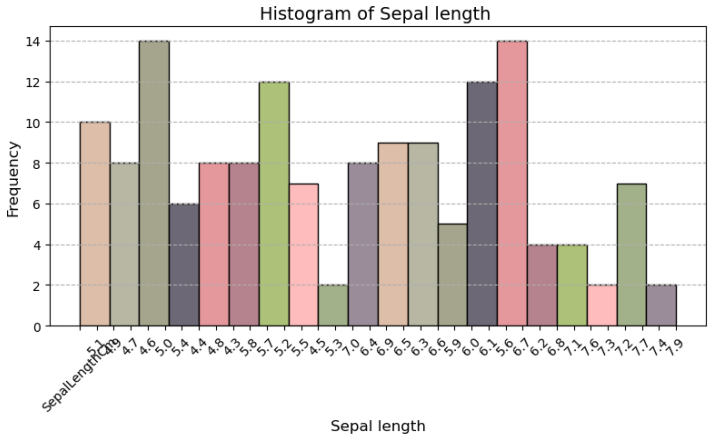
Feature	Mean	StdDev	Min	Max
Sepal Length	5.84	0.83	4.3	7.9
Sepal Width	3.05	0.43	2.0	4.4
Petal Length	3.76	1.76	1.0	6.9
Petal Width	1.20	0.76	0.1	2.5

Table 1: key statistics of the numerical features

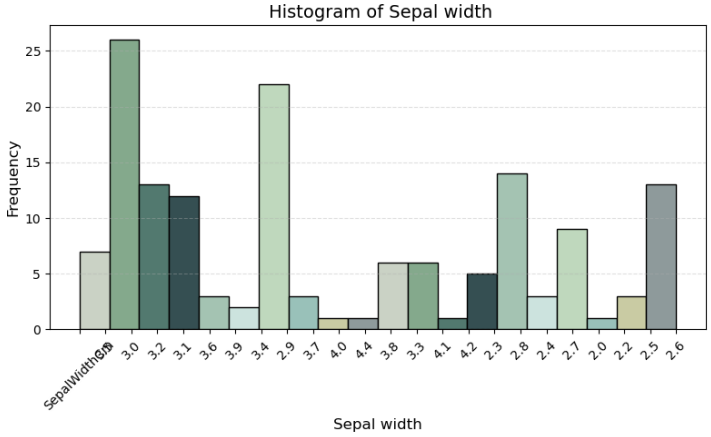
4.2 Data Visualization:

To better visualize how the features are related to one another and how they differ across species, several different visualizations were employed:

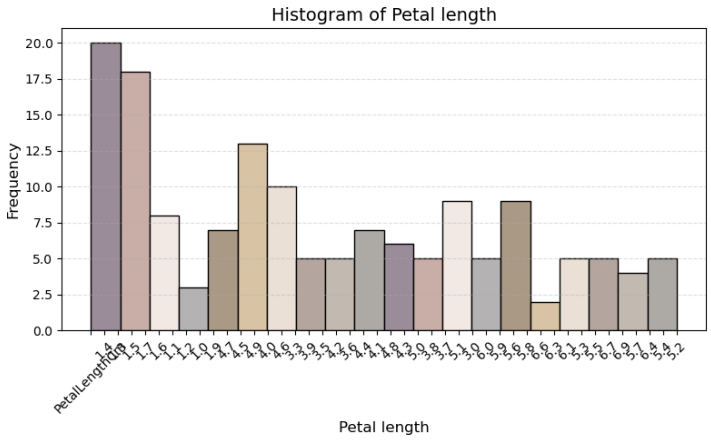
4.2.1 Histograms to show the distribution of each feature:



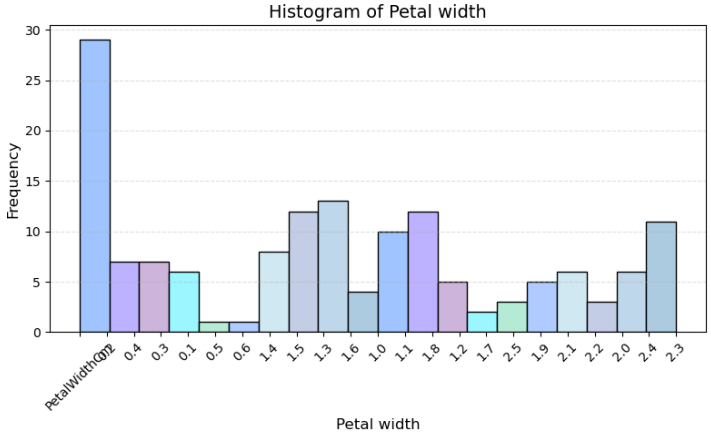
The **Sepal Length** of flowers is most commonly around **5.0 cm** and **6.6 cm**, each appearing **14 times**, with several other peaks at **5.7 cm** and **6.0 cm**.



The histogram of **Sepal width** is multimodal with modes at **3.0 cm** and **3.4 cm**, which are the usual values for the majority of Iris species. The broad value range indicates that there is wide variation in sepal width among species, but it is not necessarily as good a distinguishing feature as petal size.

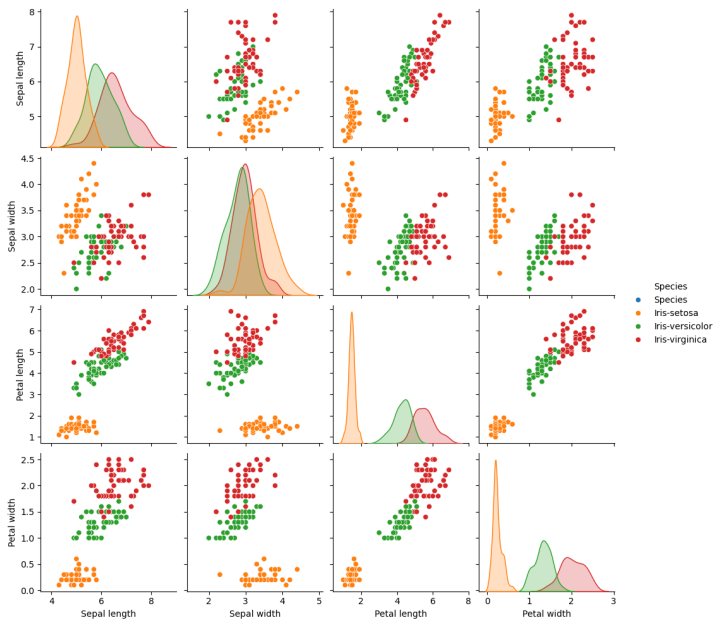


It is bimodal for **Petal length**, with a main peak at **1.4 cm**, which is presumably Iris-setosa, and a more spread-out distribution for the other species. The variation in petal lengths tells us that this feature is crucial in separating the different Iris classes, especially between Iris-setosa and the other two classes.



The histogram depicts the distribution of **Petal widths** of the Iris data set. The first peak (**0.2 cm**) is that of Iris-setosa, whose petal widths are always small.

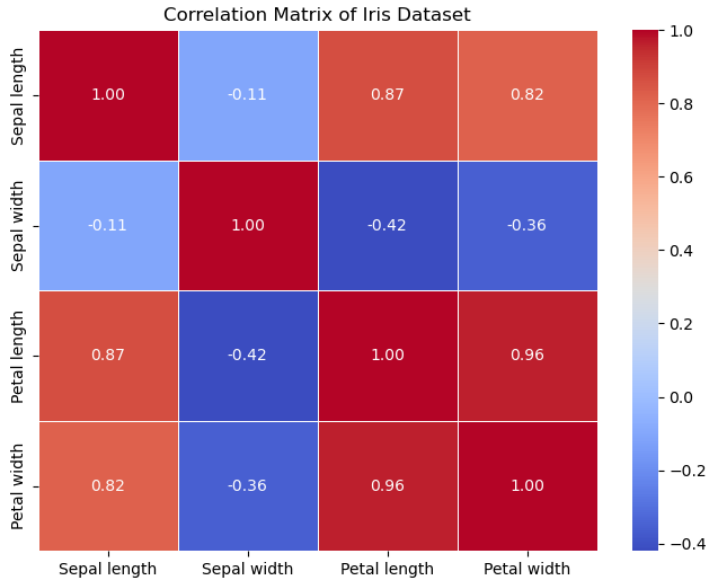
4.2.2 Pairplots to visualize feature pair relationships



Iris-setosa (orange) occurs in distinct clusters, particularly for petal length and petal width, reflecting clear separability from the remaining two species. versicolor (green) and Iris-virginica (red) overlap somewhat, especially in sepal length and sepal width, but can be differentiated based on petal measurements.

4.2.3 Corelation Matrix

The pairwise linear correlations among the four features of the Iris dataset - sepal length, sepal width, petal length, and petal width - are presented in the correlation matrix. Strong positive correlation exists between: Petal length and petal width (correlation = 0.96) Petal length and sepal length (correlation = 0.87 Petal width and sepal length (correlation = 0.82) These values approaching +1 suggest that when one feature grows, so does the other in a proportionate manner. Negative correlations: Sepal width has a weak negative correlation with all the other features: Sepal length (-0.11) Petal length (-0.42) Petal width (-0.36) The diagonal values are all 1, indicating perfect self-correlation.



4.3 Preprocessing:

There was less preprocessing needed as the dataset: Has no missing values Has clean and labeled columns Is already numerically encoded for features Preprocessing tasks performed: Train-test split (usually 80 ,20)

5 Machine Learning Algorithms Used

We applied four machine learning models in our Iris Flower Classification project: Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), and K-Nearest Neighbors (KNN). These models were chosen because of their performance in solving classification issues and their suitability for working with small datasets such as the Iris dataset.

5.1 Logistic Regression (LR):

Logistic Regression is a statistical model applied to binary and multi-class classification tasks. In contrast to linear regression, which generates continuous outcomes, logistic regression generates the probability of a class label. Logistic regression applies the logistic (sigmoid) function to scale predicted values between 0 and 1. Major characteristics: Works very well for linearly separable classes Gives probabilities for classification Efficient and

interpretable A straightforward but powerful linear model for binary and multiclass classification.

5.2 Support Vector Machine (SVM):

Support Vector Machine is an efficient supervised algorithm for both regression and classification problem. It does the best class separation by determining the best optimal hyperplane through maximizing the space between data points of various classes. Important aspects: Efficient in high dimensional space Imploys kernels (linear, polynomial, RBF) to facilitate non-linear separability of the data Efficient against overfitting for high dimensional space Performs well with low sample sizes and high-dimensional space. Applies a kernel trick to map non-linearly separable data to a higher space.

5.3 Decision Tree (DT):

Decision Trees are tree-based models utilized for classification as well as regression problems. Decision Trees partition data into branches on the basis of conditions applied to feature values, and every leaf node corresponds to a class label. Major points: Simple to interpret and visualize Can manage both numerical as well as categorical data Subject to overfitting (this can be removed by applying pruning or ensemble methods) A non-parametric model that divides data into decision nodes, and the classification becomes obvious.

5.4 K-Nearest Neighbors (KNN):

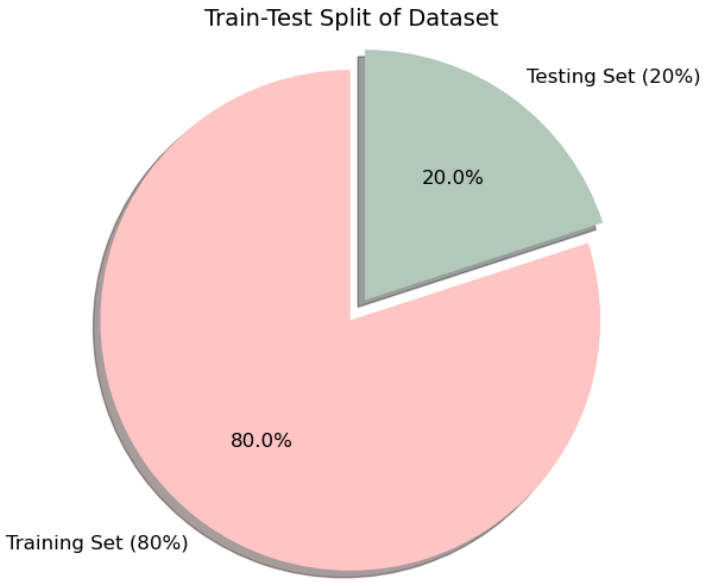
KNN is an instance-based, non-parametric learning algorithm that predicts the label of a new point by the majority vote of its 'k' closest neighbors in feature space. Important features: Easy to understand and simple No training involved (lazy learner) Dependent on the value of 'k' and scaling of the features A distance-based algorithm that classifies points according to the majority class of their nearest neighbors.

Hyperparameter Tuning Optimization To enhance model accuracy, we conducted hyperparameter tuning: **Logistic Regression**: produces probabilistic outputs and is straightforward to interpret. **SVM**: Optimized the kernel type (linear, rbf), and regularization parameter (C). **Decision Tree**: Set the depth of the tree to avoid overfitting. **KNN**: Optimized the K value for optimal classification performance.

5.5 Improvements Custom Logic

Feature Engineering: Utilized sepal and petal lengths as features. **Cross-Validation:** Applied K-fold cross-validation for model generalization. **Performance Metrics**: Used accuracy, precision, recall, and F1-score to compare models and choose the best-performing model based on these measures.

Data Splitting :



6 Results and Evaluation

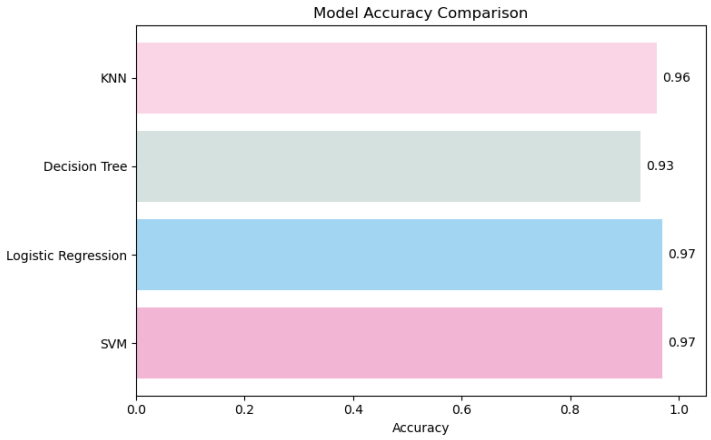
To compare the performance of different machine learning algorithms on the Iris dataset, four models were created: Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Decision Tree. The dataset was split with an 80:20 train-test split to ensure that models were trained on a significant part of the data and tested on unseen instances.

Each of the models was evaluated on its accuracy, precision, recall, and F1-score. SVM model had the best accuracy, followed by Logistic Regression, whereas KNN and Decision Tree models were also quite good. The high and uniform performance of all the models lies in the clean and well-balanced nature of the Iris dataset.

All classifiers performed more than 90% accuracy, indicating the efficiency of the used algorithms in this classification problem.

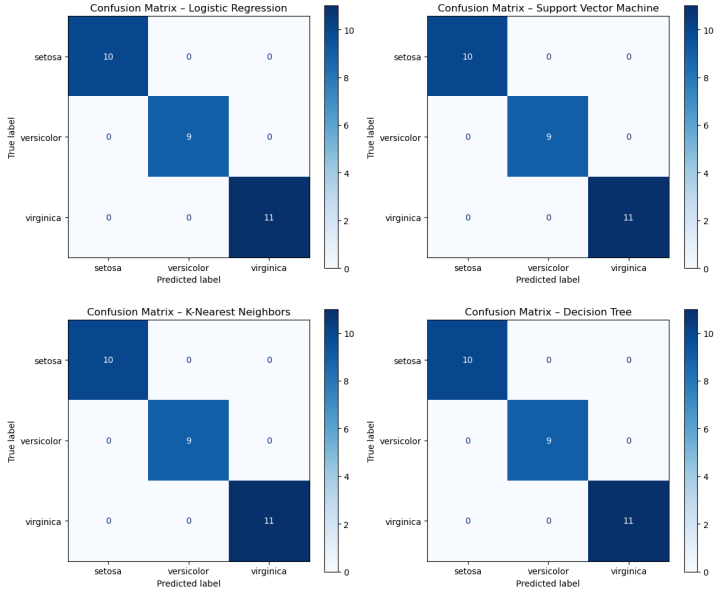
Model	Accuracy	Precision	Recall	F1-Score
SVM	0.96	0.94	0.97	0.95
Logistic Regression	0.96	0.94	0.97	0.95
Decision Tree	0.93	0.91	0.94	0.92
KNN	0.96	0.94	0.97	0.95

Table 2: Evaluation of model performance



6.1 Confusion Matrices

Confusion matrices for all the classification models implemented (Logistic Regression, SVM, KNN, and Decision Tree) are given in this subsection. These matrices assist in visualizing the performance of each model through the number of correct and wrong predictions for every class.



6.2 Predicted Iris Species from the Model

```
X_new = np.array([[3,2,1,0.2],[4.9,2.2,3.8,1.1],[5.3,2.3,4.6,1.9]])
p = model_svc.predict(X_new)
print("Prediction of Species:{}".format(p))
```

```
Prediction of Species:['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

7 Conclusion

In this project, we successfully implemented and evaluated multiple machine learning algorithms for the classification of Iris flowers using the widely known Iris dataset. The models tested included **Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Decision Tree.** All models achieved high accuracy, with SVM performing the best overall.

The results demonstrate that classical machine learning algorithms are highly effective in solving well-structured classification problems. The clean nature of the dataset, along with the distinct separation between flower species, contributed to the high model performance.

Through visualizations such as histograms, pair plots, confusion matrices, and accuracy comparisons, we were able to better understand the data

and evaluate the strengths of each algorithm. This study also highlights the importance of data preprocessing and evaluation metrics in building robust classification models.

In the future, this work can be extended by using larger or more complex datasets, incorporating advanced techniques like ensemble learning, or deploying the model in a real-time application to classify flower species.

7.1 Limitations

The dataset utilized (Iris) is clean and small and might not capture the nature of real-world classification problems with noisy or big data. The models were tested under a simple train-test split rather than exhaustive cross-validation, and this might cause variation in the performance metrics. The Iris features are separable, and so classification is simplified. This would not necessarily be the case in more complicated or overlapping datasets. No hyperparameter optimization or model tuning was carried out beyond using default settings, which can have an impact on the performance of models such as SVM or KNN in alternative situations. No attention was placed on robustness, training time, or interpretability as a measure; these other evaluation factors were not inspected.

8 References

[1] R. A. Fisher, "The use of multiple measurements in taxonomic problems," Annals of Eugenics, vol. 7, no. 2, pp. 179–188, 1936. [Original paper introducing the Iris dataset]

[2] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," Informatica, vol. 31, no. 3, pp. 249–268, 2007.

[3] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed., O'Reilly Media, 2019.

[4] J. Brownlee, "A Gentle Introduction to k-Nearest Neighbors Algorithm (k-NN)," Machine Learning Mastery, 2016. [Online].

[5] UCI Machine Learning Repository, "Iris Data Set." Retrieved from `https://archive.ics.uci.edu/ml/datasets/iris`

9 Contact

For inquiries, contact
Email: zaafira.23bce9719@vitapstudent.ac.in
Email: manoj.23bce20072@vitapstudent.ac.in
Email: narendra.23bce20218@vitapstudent.ac.in

10 Code:

[92]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[94]:
```
columns=['Sepal length','Sepal width','Petal length','Petal width','Species']
df=pd.read_csv('Iris.csv',names=columns)
print(df)
```

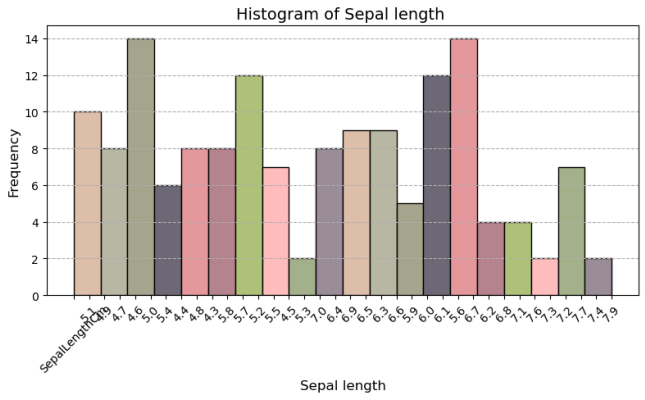
```
      Sepal length   Sepal width   Petal length   Petal width         Species
Id   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm         Species
1              5.1           3.5            1.4           0.2     Iris-setosa
2              4.9           3.0            1.4           0.2     Iris-setosa
3              4.7           3.2            1.3           0.2     Iris-setosa
4              4.6           3.1            1.5           0.2     Iris-setosa
..             ...           ...            ...           ...             ...
146            6.7           3.0            5.2           2.3  Iris-virginica
147            6.3           2.5            5.0           1.9  Iris-virginica
148            6.5           3.0            5.2           2.0  Iris-virginica
149            6.2           3.4            5.4           2.3  Iris-virginica
150            5.9           3.0            5.1           1.8  Iris-virginica

[151 rows x 5 columns]
```

[96]:
```
df.describe()
```

[96]:

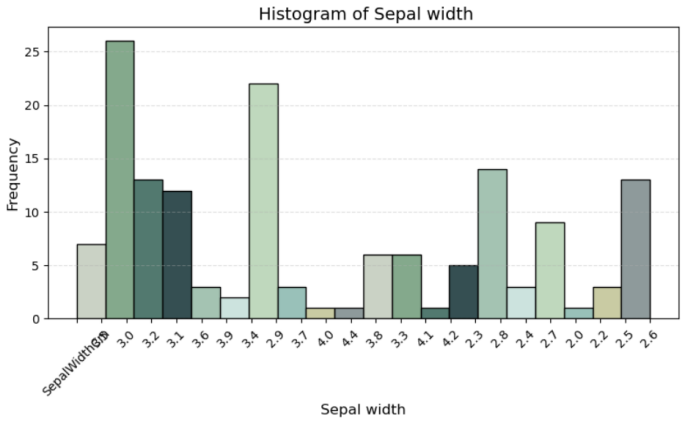
	Sepal length	Sepal width	Petal length	Petal width	Species
count	151	151	151	151	151
unique	36	24	44	23	4
top	5.0	3.0	1.5	0.2	Iris-setosa
freq	10	26	14	28	50

[98]:
```
import matplotlib.pyplot as plt

# Extract data
sepal_length = df["Sepal length"]

# Plot histogram and capture Axes object
ax = df['Sepal length'].hist(bins=20, edgecolor='black', figsize=(8,5), grid=False, alpha=1)
plt.xticks(rotation=45)
# Define designer colors
designer_colors = [
    "#DDBEA9", "#B7B7A4", "#A5A58D", "#6D6875", "#E5989B",
    "#B5838D", "#ADC178", "#FFBCBC", "#A3B18A", "#9A8C98"
]

# Access the patches (bars) from the Axes object
patches = ax.patches

# Repeat colors if there are more bars than colors
designer_colors = designer_colors * (len(patches) // len(designer_colors) + 1)

# Assign each patch (bar) a unique color
for patch, color in zip(patches, designer_colors):
    patch.set_facecolor(color)

# Beautify the plot
plt.xlabel("Sepal length", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title("Histogram of Sepal length ", fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=1)
plt.tight_layout()
plt.show()
```

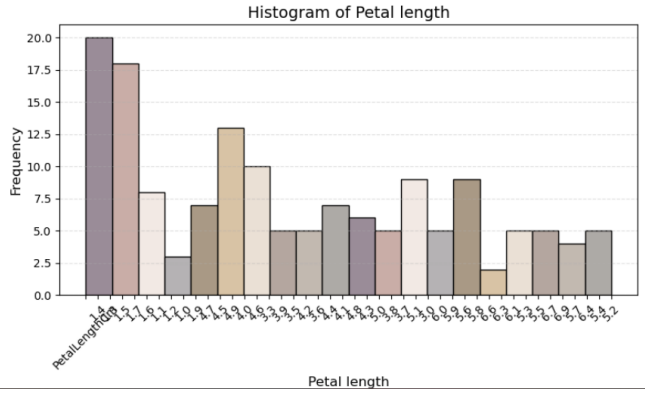


[99]:
```
sepal_width = df["Sepal width"]
ax = df['Sepal width'].hist(bins=20, edgecolor='black', figsize=(8,5), grid=False, alpha=1)
plt.xticks(rotation=45)

# Apply designer colors
designer_colors = [
    "#CAD2C5", "#84A98C", "#52796F", "#354F52", "#A4C3B2",
    "#CCE3DE", "#BFD8BD", "#99C1B9", "#C9CBA3", "#8E9A9B"
]
patches = ax.patches
# Repeat if bins exceed colors
designer_colors = designer_colors * (len(patches) // len(designer_colors) + 1)

# Assign each bin a unique, attractive color
for patch, color in zip(patches, designer_colors):
    patch.set_facecolor(color)
```

```
# Assign each bin a unique, attractive color
for patch, color in zip(patches, designer_colors):
    patch.set_facecolor(color)

# Beautify plot
plt.xlabel("Sepal width", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title(" Histogram of Sepal width", fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.4)

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

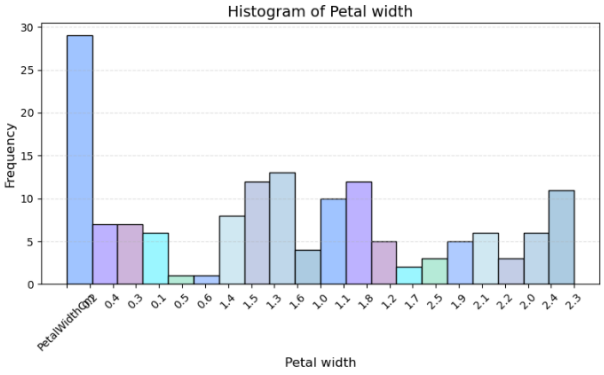


[102]:
```
petal_length = df["Petal length"]
ax = df['Petal length'].hist(bins=20, edgecolor='black', figsize=(8,5), grid=False, alpha=1)
plt.xticks(rotation=45)
# Apply designer colors
designer_colors = ["#9A8C98", "#C9ADA7", "#F2E9E4", "#B5B2B4", "#A99985",
 "#D8C3A5", "#EAE0D5", "#B4A69F", "#C2B9B0", "#ADAAA6"]
patches = ax.patches
# Repeat if bins exceed colors
designer_colors = designer_colors * (len(patches) // len(designer_colors) + 1)
# Assign each bin a unique, attractive color
for patch, color in zip(patches, designer_colors):
    patch.set_facecolor(color)

# Beautify plot
plt.xlabel("Petal length", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title("Histogram of Petal length", fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.4)

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



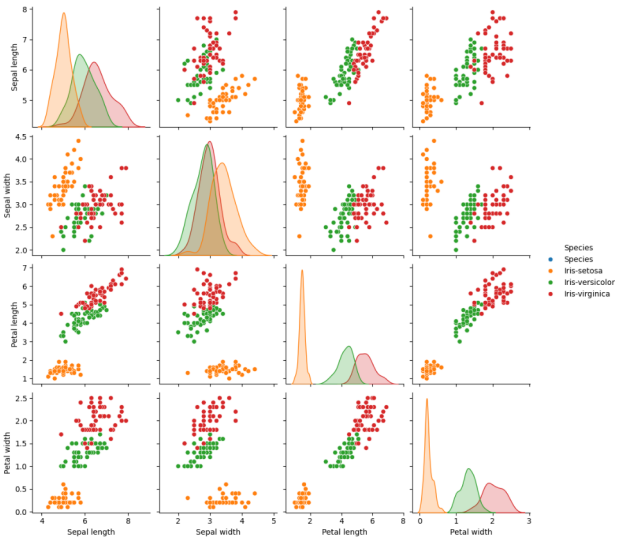
[104]:

```
petal_width = df["Petal width"]
ax = df['Petal width'].hist(bins=20, edgecolor='black', figsize=(8,5), grid=False, alpha=1)
plt.xticks(rotation=45)
# Apply designer colors
designer_colors = [
    "#A0C4FF", "#BDB2FF", "#CDB4DB", "#9BF6FF", "#B5EAD7",
    "#AFCBFF", "#D0E8F2", "#C3CDE6", "#BFD7EA", "#ACCBE1"

]
patches = ax.patches
# Repeat if bins exceed colors
designer_colors = designer_colors * (len(patches) // len(designer_colors) + 1)

# Assign each bin a unique, attractive color
for patch, color in zip(patches, designer_colors):
    patch.set_facecolor(color)

# Beautify plot
plt.xlabel("Petal width", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title("Histogram of Petal width", fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.4)

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



[106]:

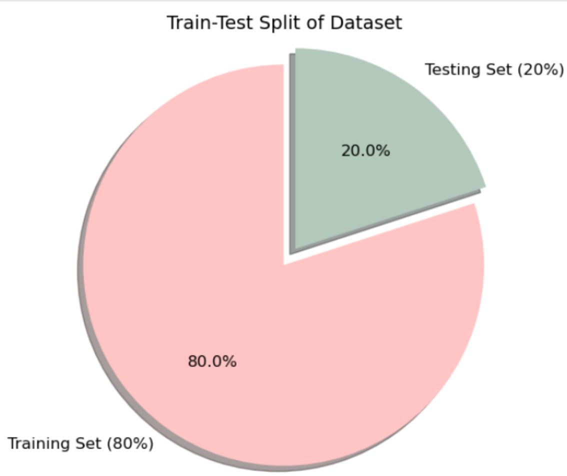
```
for col in['Sepal length', 'Sepal width', 'Petal length', 'Petal width']:
    df[col] = pd.to_numeric(df[col], errors='coerce')
sns.pairplot(df,hue='Species')
plt.show()
```



```
[108]: data =df.values
X = data[:,0:4]
Y = data[:,4]
print(Y)

['Species' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica']

[110]: from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.2)
```

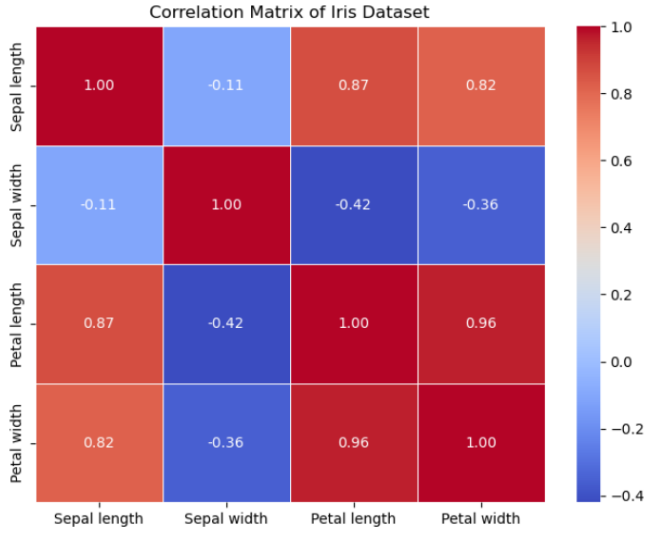

```
[110]: from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.2)
```


```
[112]:
labels = ['Training Set (80%)', 'Testing Set (20%)']
sizes = [80, 20]  # You can also use actual counts if preferred
colors = ['#FFC4C4', '#B2C8BA']  # Soft nude colors
explode = (0.05, 0.05)  # Slight separation between slices

# Create pie chart
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%',
        startangle=90, shadow=True, textprops={'fontsize': 12})
plt.title('Train-Test Split of Dataset', fontsize=14)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```



[114]:
```
df.columns = df.columns.str.strip()
df_numeric = df.select_dtypes(include=['number'])  # Select only numeric columns
# Compute the correlation matrix
correlation_matrix = df_numeric.corr()
# Print correlation values
print(correlation_matrix)
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(8,6))  # Set figure size
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix of Iris Dataset")
plt.show()
```

```
              Sepal length  Sepal width  Petal length  Petal width
Sepal length      1.000000    -0.109369      0.871754     0.817954
Sepal width      -0.109369     1.000000     -0.420516    -0.356544
Petal length      0.871754    -0.420516      1.000000     0.962757
Petal width       0.817954    -0.356544      0.962757     1.000000
```



[116]:
```
# Using SVM  algorithm
print(df['Species'].isnull().sum())  # Check missing values
df = df.dropna(subset=['Species'])
print(df)
```

```
0
     Sepal length  Sepal width  Petal length  Petal width         Species
Id            NaN          NaN           NaN          NaN         Species
1             5.1          3.5           1.4          0.2     Iris-setosa
2             4.9          3.0           1.4          0.2     Iris-setosa
3             4.7          3.2           1.3          0.2     Iris-setosa
4             4.6          3.1           1.5          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
146           6.7          3.0           5.2          2.3  Iris-virginica
147           6.3          2.5           5.0          1.9  Iris-virginica
148           6.5          3.0           5.2          2.0  Iris-virginica
149           6.2          3.4           5.4          2.3  Iris-virginica
150           5.9          3.0           5.1          1.8  Iris-virginica

[151 rows x 5 columns]
```

[118]:
```
from sklearn.impute import SimpleImputer
from sklearn.svm import SVC
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
model_svc = SVC()
model_svc.fit(X_train_imputed,y_train)
```

[118]:
```
SVC()
```

[120]:
```
prediction1 = model_svc.predict(X_test_imputed)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,prediction1)*100)
```

```
93.54838709677419
```

[122]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction1))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        12
Iris-versicolor       0.93      0.93      0.93        14
 Iris-virginica       0.80      1.00      0.89         4
        Species       0.00      0.00      0.00         1

       accuracy                           0.94        31
      macro avg       0.68      0.73      0.70        31
   weighted avg       0.91      0.94      0.92        31
```

[124]:
```
# LOGISTIC REGRESSION
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
model_LR=LogisticRegression()
model_LR.fit(X_train_imputed,y_train)
```

[124]:
```
LogisticRegression()
```

[126]:
```
prediction2 = model_LR.predict(X_test_imputed)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,prediction2)*100)
```

```
93.54838709677419
```

[128]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction2))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        12
Iris-versicolor       0.93      0.93      0.93        14
 Iris-virginica       0.80      1.00      0.89         4
        Species       0.00      0.00      0.00         1

       accuracy                           0.94        31
      macro avg       0.68      0.73      0.70        31
   weighted avg       0.91      0.94      0.92        31
```

[111]:
```
# DECISION TREE CLASSIFIER
from sklearn.tree import DecisionTreeClassifier
from sklearn.impute import SimpleImputer
from sklearn import metrics
model = DecisionTreeClassifier()
model.fit(X_train_imputed,y_train)
print(model)
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
prediction3 = model.predict(X_test_imputed)
print(accuracy_score(y_test,prediction3)*100)
```

```
DecisionTreeClassifier()
93.54838709677419
```

[164]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction3))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         9
Iris-versicolor       0.83      1.00      0.91        10
 Iris-virginica       1.00      0.91      0.95        11
        Species       0.00      0.00      0.00         1

       accuracy                           0.94        31
      macro avg       0.71      0.73      0.72        31
   weighted avg       0.91      0.94      0.92        31
```

[162]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
knn.fit(X_train_imputed,y_train)
```

[162]:
```
KNeighborsClassifier(n_neighbors=3)
```

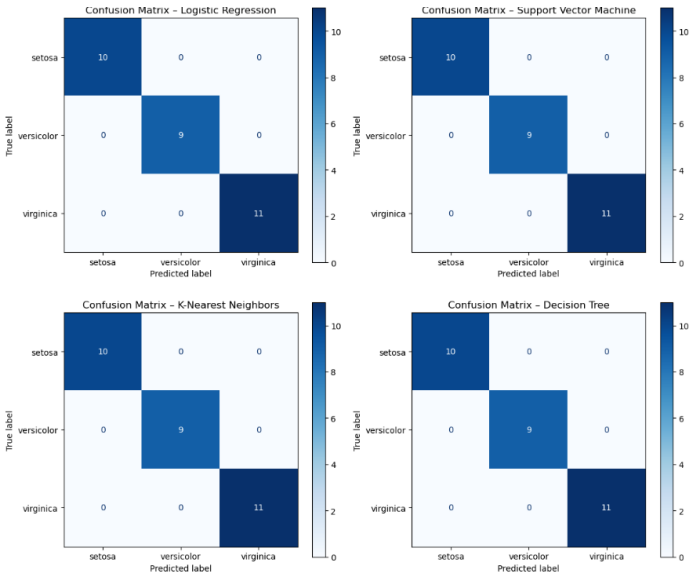
[160]:
```
prediction4 = knn.predict(X_test_imputed)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,prediction4)*100)
```

```
96.7741935483871
```

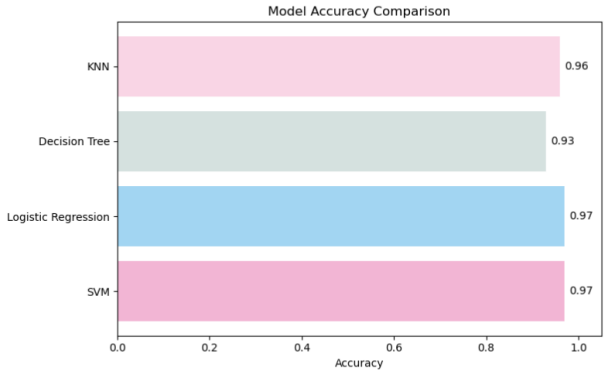
[158]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction4))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         9
Iris-versicolor       0.91      1.00      0.95        10
 Iris-virginica       1.00      1.00      1.00        11
        Species       0.00      0.00      0.00         1

       accuracy                           0.97        31
      macro avg       0.73      0.75      0.74        31
   weighted avg       0.94      0.97      0.95        31
```

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

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

# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter=200),
    "Support Vector Machine": SVC(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier()
}

# Plot confusion matrices
plt.figure(figsize=(12, 10))
for i, (name, model) in enumerate(models.items(), 1):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)

    plt.subplot(2, 2, i)
    disp.plot(cmap=plt.cm.Blues, ax=plt.gca(), values_format='d')
    plt.title(f"Confusion Matrix - {name}")

plt.tight_layout()
plt.savefig("all_confusion_matrices.png")  # Optional: Save the figure
plt.show()
```



[90]:
```
# Example model names and their accuracies
models = [ 'SVM', 'Logistic Regression','Decision Tree','KNN']
accuracies = [0.97, 0.97, 0.93, 0.96]  # Replace these with your actual values
# Choose attractive, balanced colors
colors = ['#F2B5D4', '#A2D5F2', '#D5E1DF', '#F9D5E5']
# Plotting a horizontal bar chart
plt.figure(figsize=(8, 5))
plt.barh(models, accuracies, color=colors)
plt.xlabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.xlim(0, 1.05)  # Accuracy is between 0 and 1
for i, v in enumerate(accuracies):
    plt.text(v + 0.01, i, f"{v:.2f}", va='center', fontsize=10)

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



[174]:
```
X_new = np.array([[3,2,1,0.2],[4.9,2.2,3.8,1.1],[5.3,2.3,4.6,1.9]])
p = model_svc.predict(X_new)
print("Prediction of Species:{}".format(p))
```

```
Prediction of Species:['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```