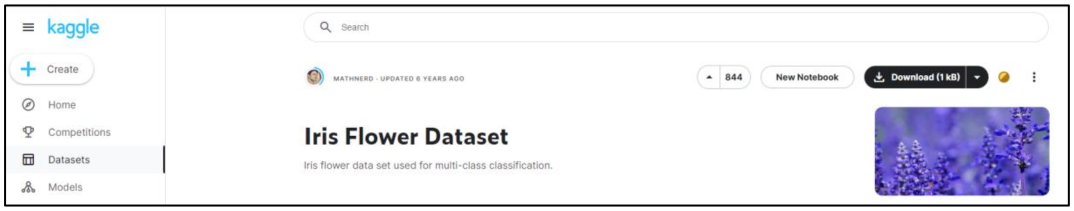
Introduction: You are a data scientist working on a project to classify iris flowers based on their features. The Iris dataset, which contains samples from three different species (Setosa, Versicolor, and Virginica), will be your primary data source. The dataset includes four features: sepal length, sepal width, petal length, and petal width. Your task is to build a machine learning model that can accurately predict the species of an iris flower based on these features.

Scenario:

Imagine you are collaborating with a team of botanists who want to automate the process of identifying iris species. They have collected measurements of sepal and petal dimensions from various iris flowers. Your team's goal is to create a reliable classification model that can assist botanists in identifying the correct species quickly and accurately.

Step 1: Data Acquisition and Preparation

- **Download the Iris dataset from platforms such as Kaggle, UCI Machine Learning Repository, or any other reliable source.**



Iris Flower Dataset

Iris flower data set used for multi-class classification.

IRIS.csv   You modified • 9 May 2024   me   ML_Datasets

- **Provide a brief overview of it, including its origin, features, and target variables.**

The Iris dataset originated from a seminal paper by British statistician and biologist Ronald Fisher titled "The Use of Multiple Measurements in Taxonomic Problems," published in 1936. Fisher collected and measured samples of iris flowers from three different species: Setosa, Versicolor, and Virginica.

The dataset comprises 150 samples, with each sample having four features measured:

1. Sepal Length
2. Sepal Width
3. Petal Length
4. Petal Width

Additionally, each sample is labeled with its corresponding species, making it a supervised learning problem with three target variables:

1. Setosa
2. Versicolor
3. Virginica

The Iris dataset is often used as a benchmark in machine learning and pattern recognition for tasks like classification and clustering. Its simplicity and clarity make it an excellent starting point for learning various algorithms and techniques.

- **Describe the classes or categories present in the dataset and their distribution.**

The Iris dataset contains three classes or categories, each representing a species of iris flower:

1. Setosa
2. Versicolor
3. Virginica

Each class represents a distinct species of iris flower. In the dataset, there are 50 samples for each species, making a total of 150 samples. Therefore, the distribution of classes is balanced, with an equal number of samples for each species.

This balanced distribution is intentional and contributes to the dataset's suitability for training and evaluating machine learning models. It ensures that models trained on this dataset are not biased towards any particular class and can generalize well across all classes.

- **Analyze the distribution of classes within the dataset and discuss any imbalances or challenges that may arise during classification.**

While there are no significant challenges related to class distribution, some potential challenges include outliers, missing values, or inconsistencies in the data.

Since the dataset is relatively small and well-balanced, overfitting might be a concern, especially if the model is too complex.

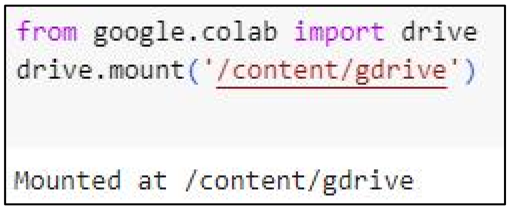
- **Load the dataset into Google Colab environment using Python libraries.**

To load the Iris dataset into a Google Colab environment using Python libraries, you can follow these steps:

1. First, upload your dataset file (iris.csv) to your Google Drive.
2. Mount your Google Drive in the Colab environment to access the dataset file.
3. Use Pandas library to read the dataset file and load it into a DataFrame.

Here's a code to achieve this:

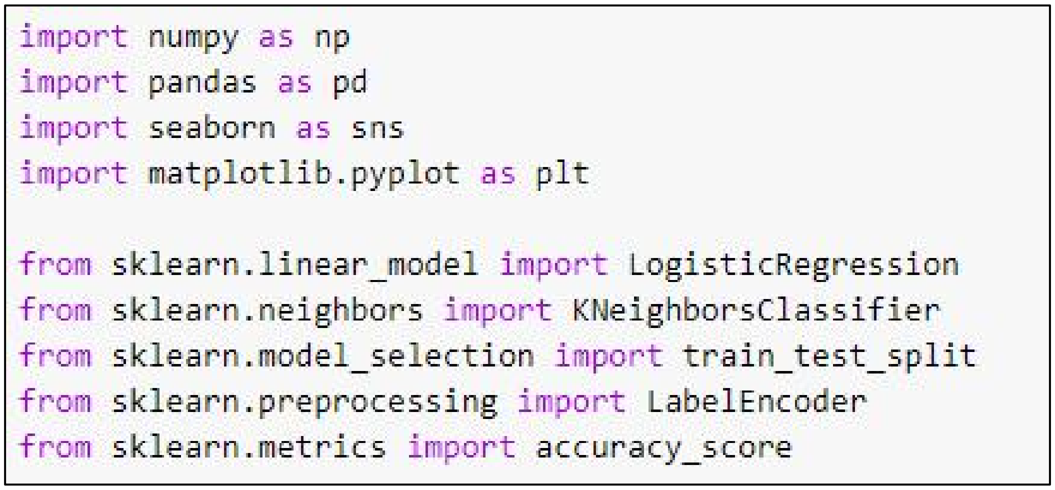
➢ **Mounted Drive:**



```
from google.colab import drive
drive.mount('/content/gdrive')
```

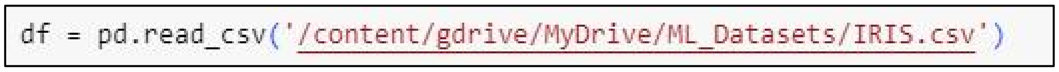
```
Mounted at /content/gdrive
```

➢ **Import Libraries:**



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
```

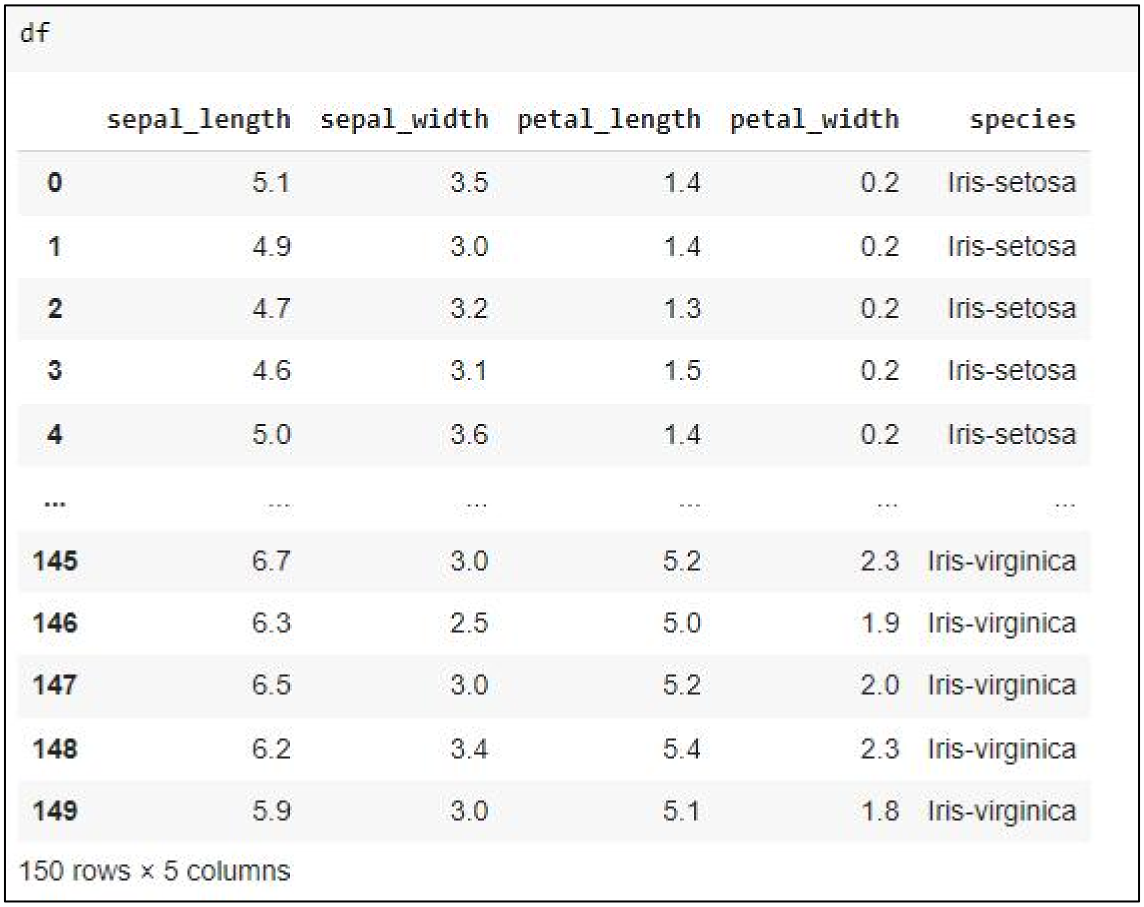
➢ **Loading Dataset:**



```
df = pd.read_csv('/content/gdrive/MyDrive/ML_Datasets/IRIS.csv')
```

- **Explore the dataset to understand its structure, statistical properties, and potential challenges.**

Dataset:



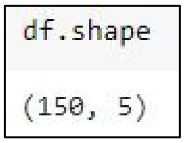
```
df
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

Structure:

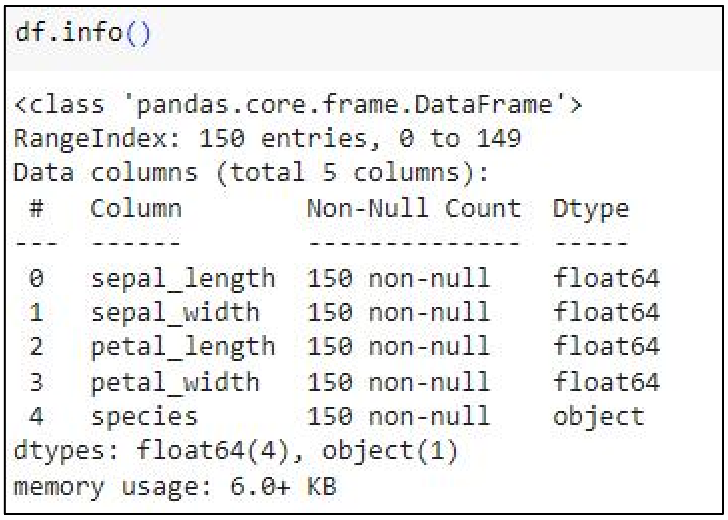
Use function **df.shape** to explore the structure:



```
df.shape
```

```
(150, 5)
```

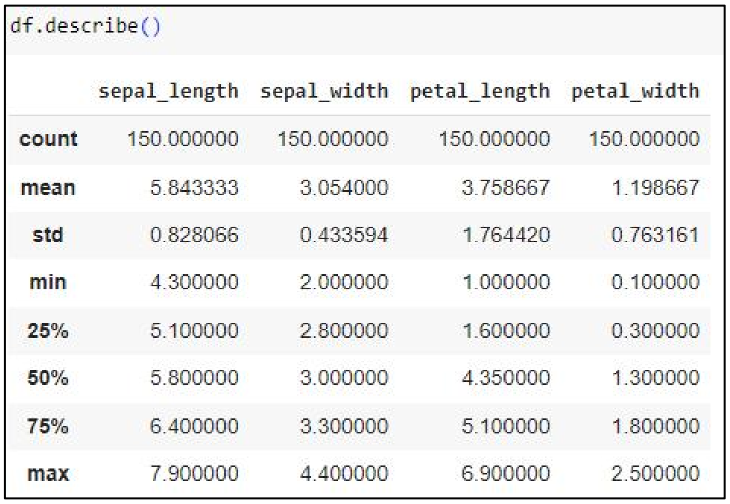
Dataset Information:



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

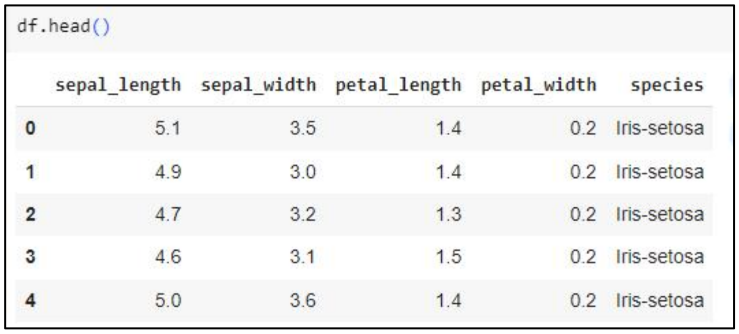
Statistical Properties:



```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

First 5 Rows of the Dataset:



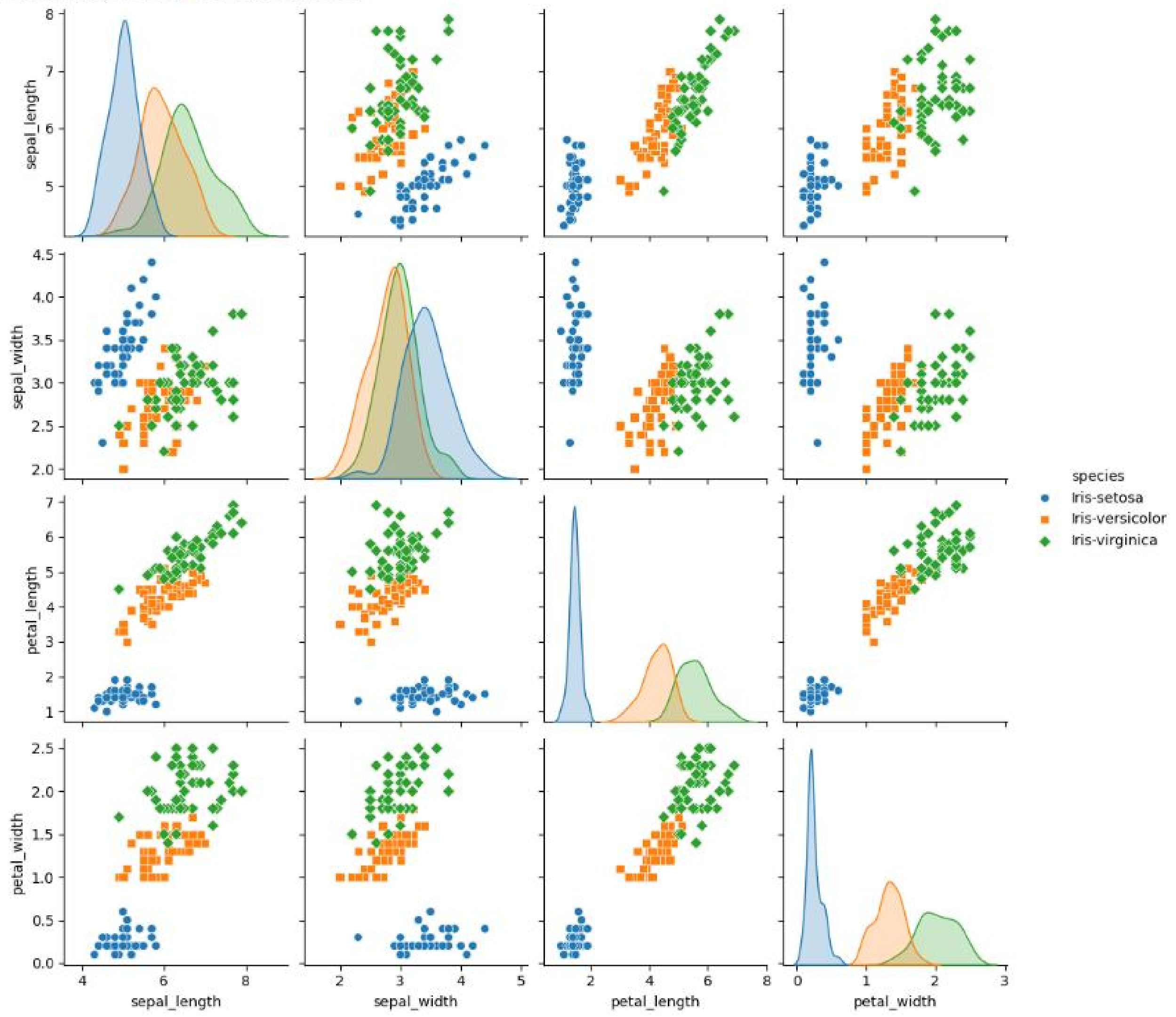
```
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

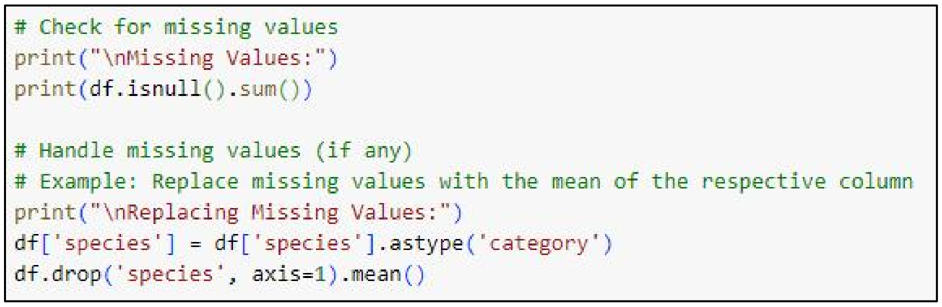
Visualize the distribution of each feature using histograms or box plots to identify outliers or anomalies.

```python
# Visualize the distribution of each feature
sns.pairplot(df, hue='species', markers=["o", "s", "D"])
```

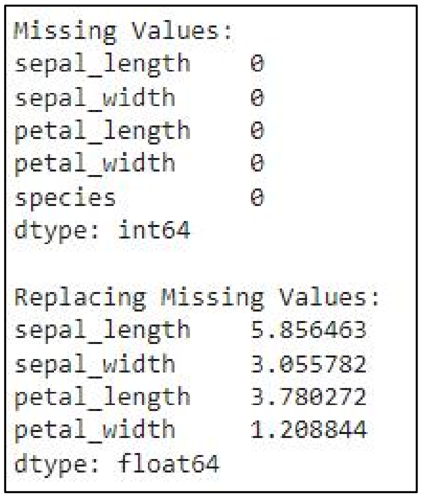
```
<seaborn.axisgrid.PairGrid at 0x7bbe72c36ce0>
```



- **Preprocess the data by handling any missing values, outliers, or inconsistencies to ensure quality data for analysis.**
- Handle missing values: Check for any missing values in the dataset and either impute them using techniques like mean, median, or mode, or remove the corresponding instances if the number of missing values is negligible.



```python
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Handle missing values (if any)
# Example: Replace missing values with the mean of the respective column
print("\nReplacing Missing Values:")
df['species'] = df['species'].astype('category')
df.drop('species', axis=1).mean()
```

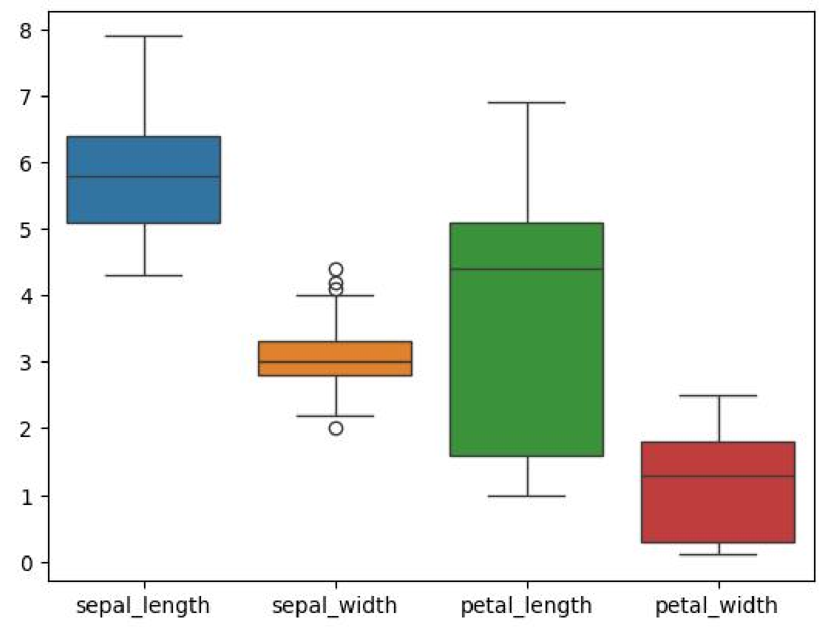


```
Missing Values:
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64

Replacing Missing Values:
sepal_length    5.856463
sepal_width     3.055782
petal_length    3.780272
petal_width     1.208844
dtype: float64
```

- Outlier detection and treatment: Identify outliers using visualizations or statistical methods like z-score or IQR (Interquartile Range), and consider removing or transforming them if they significantly affect the model's performance.

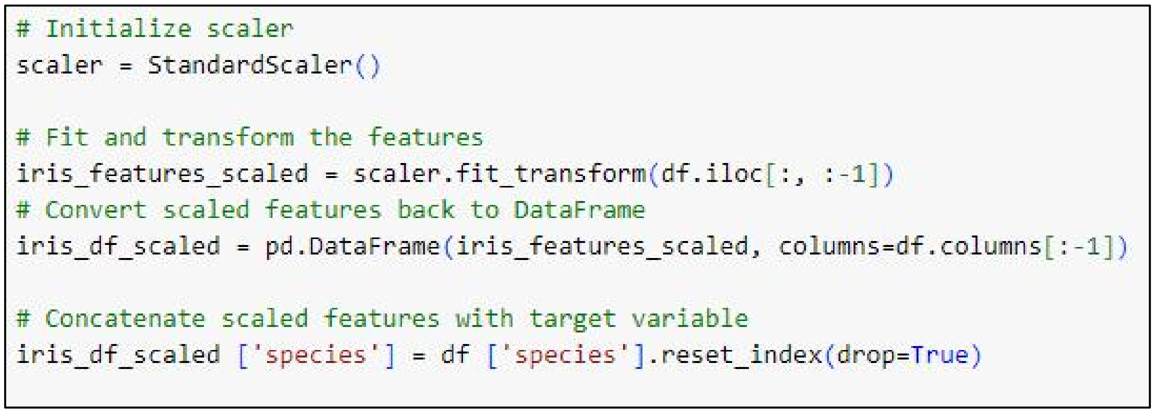
```
# Visualize outliers using box plots
sns.boxplot(data=df)
```

```
<Axes: >
```



```
from scipy import stats

# Detect outliers using z-score from scipy import stats
z_scores = stats.zscore(df)
abs_z_scores = np.abs(z_scores)
outliers = (abs_z_scores > 3).all(axis=1)

# Remove outliers
df = df[~outliers]
```

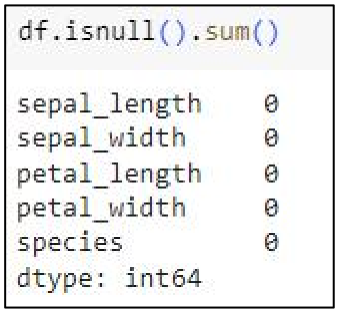
- Data normalization or standardization: Scale the features to a similar range to prevent features with larger scales from dominating the learning process.



```
# Initialize scaler
scaler = StandardScaler()

# Fit and transform the features
iris_features_scaled = scaler.fit_transform(df.iloc[:, :-1])
# Convert scaled features back to DataFrame
iris_df_scaled = pd.DataFrame(iris_features_scaled, columns=df.columns[:-1])

# Concatenate scaled features with target variable
iris_df_scaled ['species'] = df ['species'].reset_index(drop=True)
```

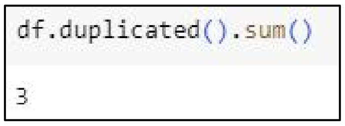
Once we have completed these steps, we now have a clean and preprocessed dataset ready for building and training your machine learning model for iris species classification.

➢ **Check for Null Values:**



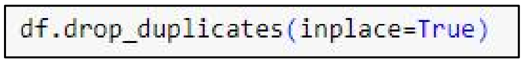
```
df.isnull().sum()

sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

➢ **Duplicated Values Count:**



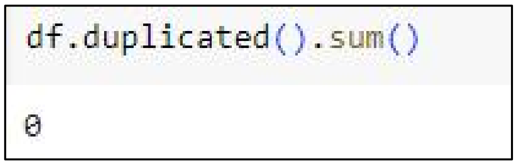
```
df.duplicated().sum()

3
```

➢ **Drop Duplicated Values:**



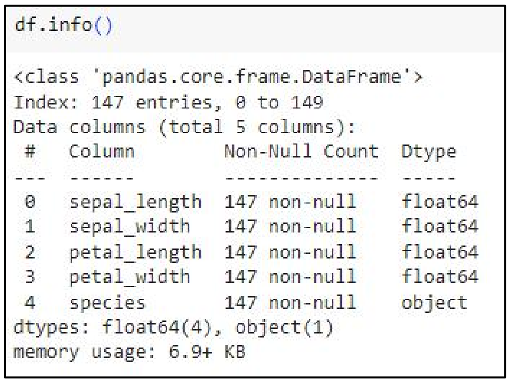
```
df.drop_duplicates(inplace=True)
```

➢ **Result:**



```
df.duplicated().sum()

0
```

➢ **Dataset Information after Preprocessing:**



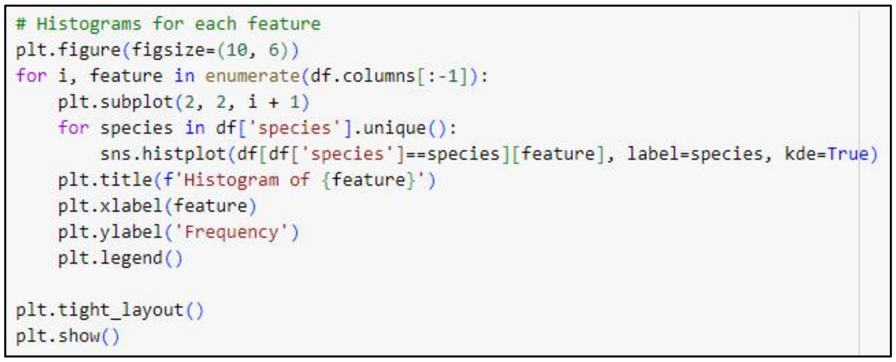
```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 147 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  147 non-null    float64
 1   sepal_width   147 non-null    float64
 2   petal_length  147 non-null    float64
 3   petal_width   147 non-null    float64
 4   species       147 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.9+ KB
```

Step 2: Exploratory Data Analysis and Visualization

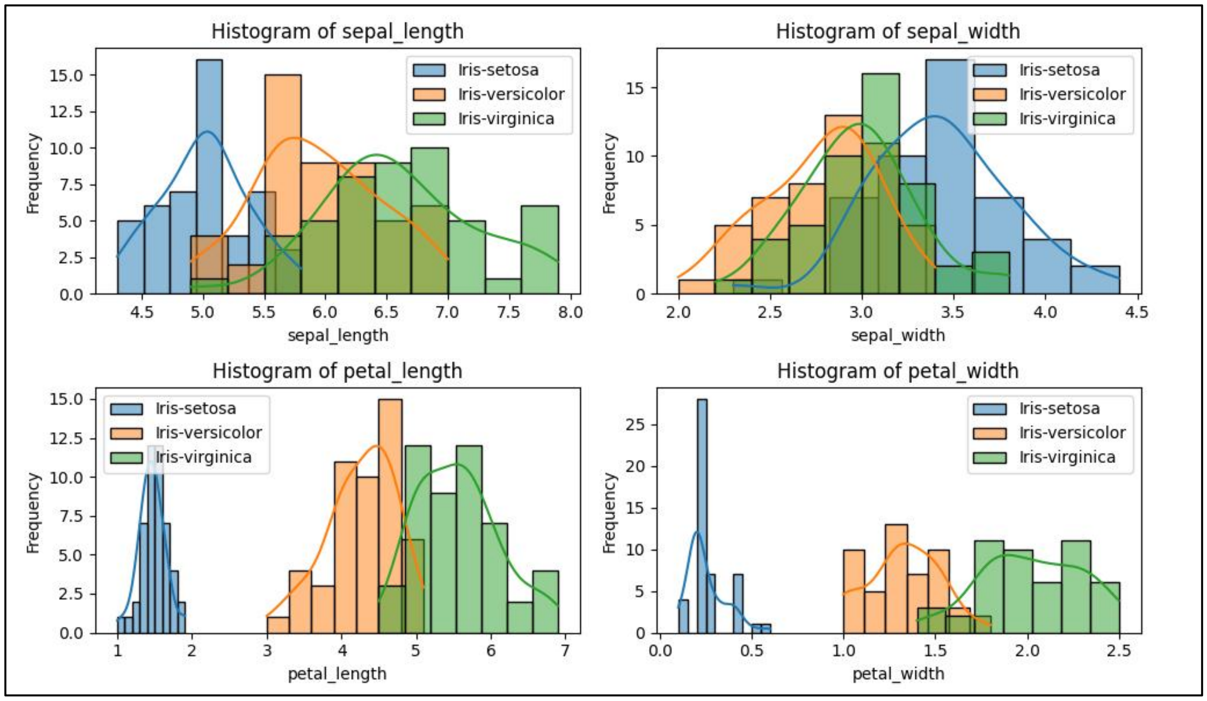
- **Perform comprehensive exploratory data analysis to gain insights into the Iris dataset.**
- **Utilize data visualization techniques such as histograms, pie chart, box plots, pair plots, and correlation matrices to visualize the distribution of features and understand the relationships between them.**
- **Identify any patterns or trends in the data that could aid in classification tasks.**
- **Use appropriate colors, markers, and styling to differentiate between different classes in the dataset during visualization.**

For Step 2 of our project, we'll conduct exploratory data analysis (EDA) and visualization to gain insights into the Iris dataset:

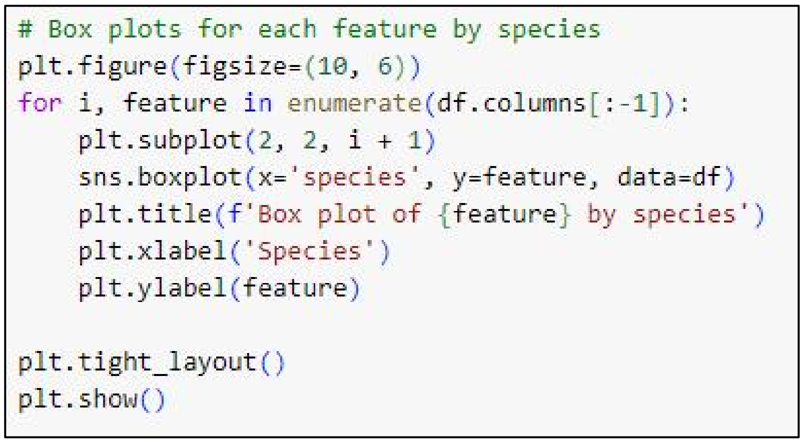
1. **Histograms:**
   - Visualize the distribution of each feature (sepal length, sepal width, petal length, petal width) using histograms.
   - Use different colors to differentiate between different species.



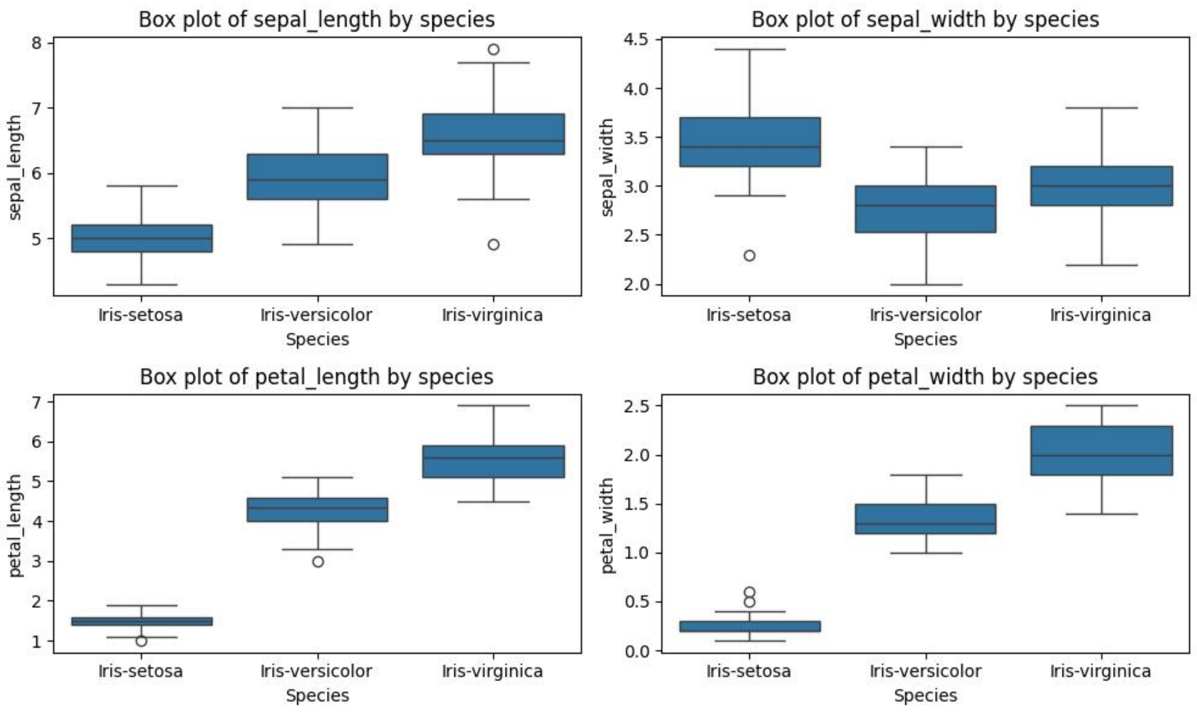
```python
# Histograms for each feature
plt.figure(figsize=(10, 6))
for i, feature in enumerate(df.columns[:-1]):
    plt.subplot(2, 2, i + 1)
    for species in df['species'].unique():
        sns.histplot(df[df['species']==species][feature], label=species, kde=True)
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.legend()

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



2. **Box Plots:**
   - Use box plots to visualize the distribution of each feature by species.
   - Box plots help identify any outliers and compare the distributions between different species.



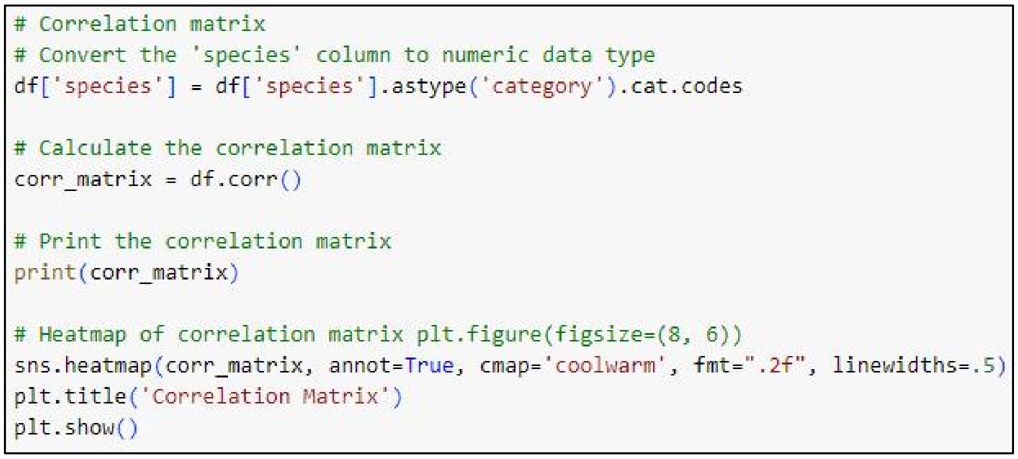
```python
# Box plots for each feature by species
plt.figure(figsize=(10, 6))
for i, feature in enumerate(df.columns[:-1]):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(x='species', y=feature, data=df)
    plt.title(f'Box plot of {feature} by species')
    plt.xlabel('Species')
    plt.ylabel(feature)

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



3. **Pair Plots:**
    - Visualize pairwise relationships between features using pair plots.
    - Pair plots allow you to observe the scatterplots between all possible pairs of features, colored by species.

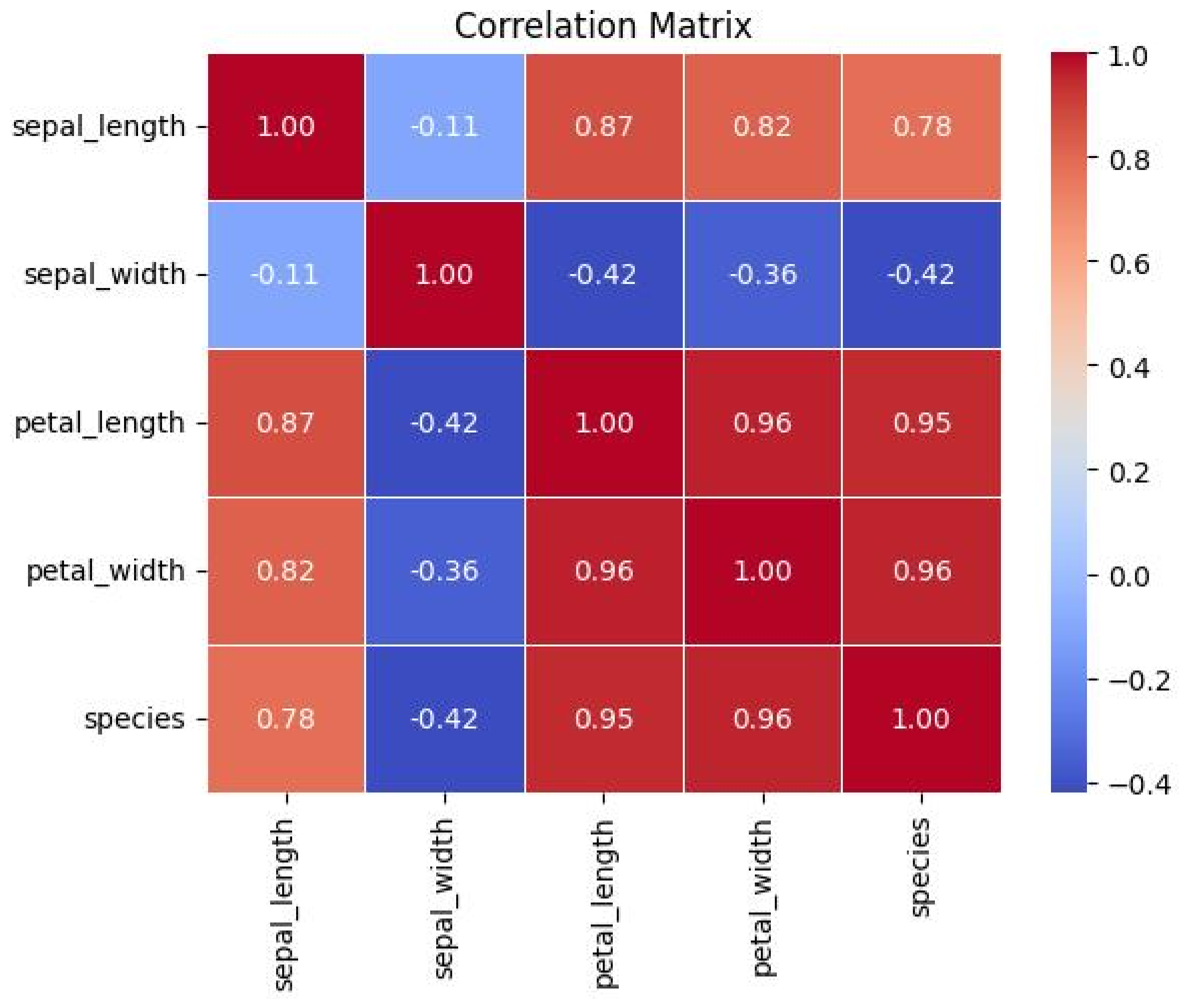
(Has been visualized in Step 1 Part 4:
Explore the dataset to understand its structure, statistical properties, and potential challenges)

4. **Correlation Matrix:**
    - Compute the correlation matrix to quantify the linear relationship between features.
    - Visualize the correlation matrix using a heat map.



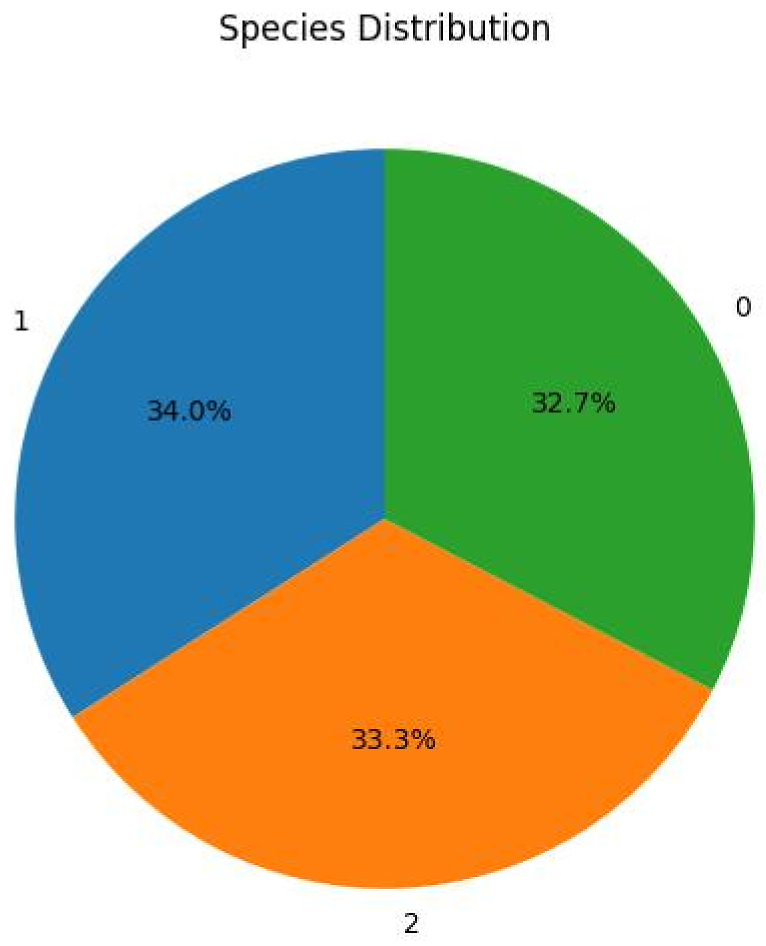
```python
# Correlation matrix
# Convert the 'species' column to numeric data type
df['species'] = df['species'].astype('category').cat.codes

# Calculate the correlation matrix
corr_matrix = df.corr()

# Print the correlation matrix
print(corr_matrix)

# Heatmap of correlation matrix plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

```
              sepal_length  sepal_width  petal_length  petal_width   species
sepal_length      1.000000    -0.109321      0.871305     0.817058  0.782904
sepal_width      -0.109321     1.000000     -0.421057    -0.356376 -0.418348
petal_length      0.871305    -0.421057      1.000000     0.961883  0.948339
petal_width       0.817058    -0.356376      0.961883     1.000000  0.955693
species           0.782904    -0.418348      0.948339     0.955693  1.000000
```



5. **Pie Chart:**
    - If you want to visualize the distribution of species, you can use a pie chart.

```
# Pie chart of species distribution
plt.figure(figsize=(6, 6))
df['species'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Species Distribution')
plt.ylabel('')
plt.show()
```



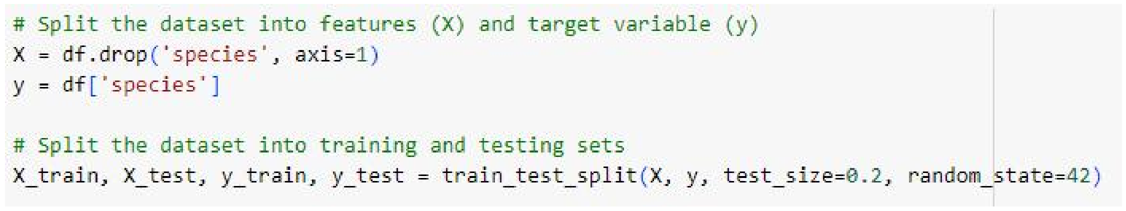
By performing these visualizations, we'll gain insights into the distribution of features, relationships between them, and patterns in the data that can aid in classification tasks. We have adjust colors, markers, and styling as needed to differentiate between different classes in the dataset during visualization.

Step 3: Model Selection and Building

- **Split the preprocessed dataset into training and testing sets to enable model evaluation.**
- **Choose at least three classification algorithms (e.g., KNN, Decision Trees, and Logistic Regression) for comparison.**
- **Implement each chosen algorithm using appropriate libraries in Python, and train the models using the training dataset.**
- **Optimize hyper parameters for each model to improve classification performance, using techniques such as grid search or random search.**

For Step 3, we'll select and build classification models for the Iris dataset. We will follow these steps to implement and train three different classification algorithms, optimize their hyper parameters, and evaluate their performance:

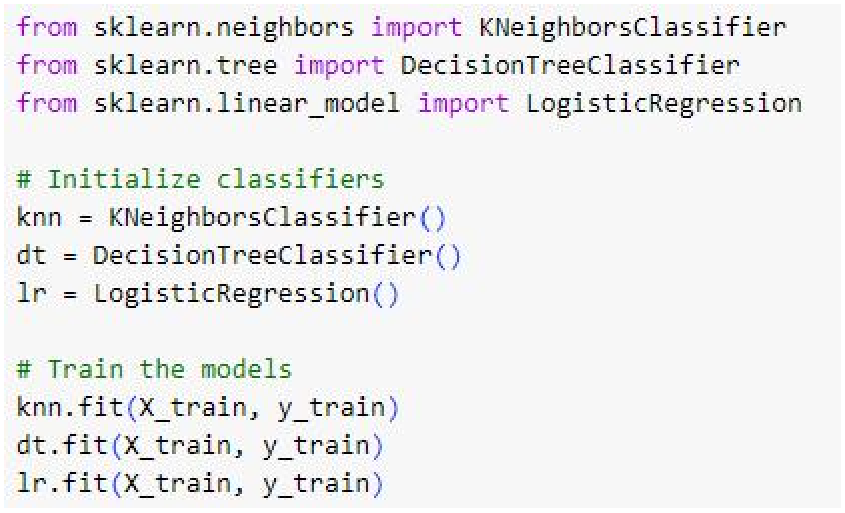
1. **Split the Dataset:**
   - Split the preprocessed dataset into training and testing sets to enable model evaluation.
   - Typically, you might use a 70-30 or 80-20 split for training and testing, respectively.



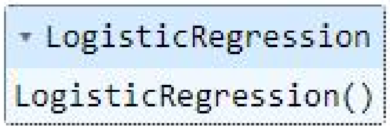
```
# Split the dataset into features (X) and target variable (y)
X = df.drop('species', axis=1)
y = df['species']

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

2. **Choose Classification Algorithms:**
   - Select at least three classification algorithms for comparison.
     1. K-Nearest Neighbors (KNN),
     2. Decision Trees
     3. Logistic Regression

3. **Implement and Train Models:**
   - Implement each chosen algorithm using appropriate libraries such as scikit-learn in Python.
   - Train the models using the training dataset.

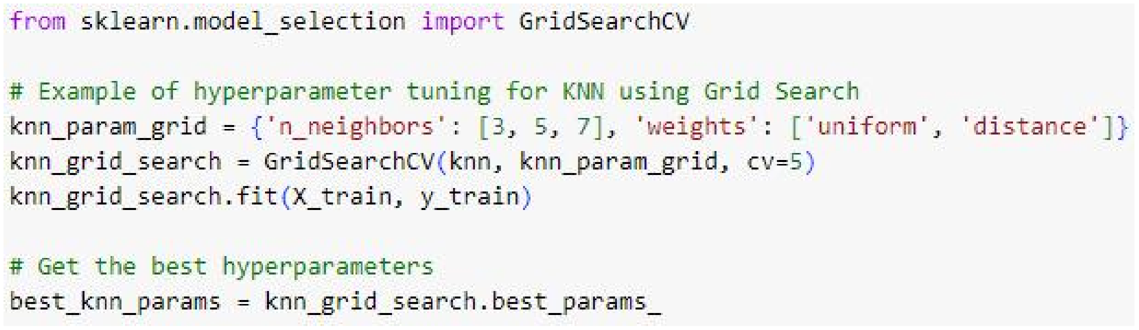


```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

# Initialize classifiers
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier()
lr = LogisticRegression()

# Train the models
knn.fit(X_train, y_train)
dt.fit(X_train, y_train)
lr.fit(X_train, y_train)
```



```
▾ LogisticRegression
LogisticRegression()
```

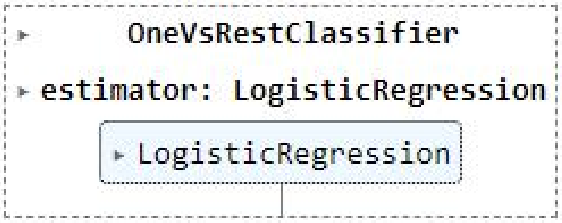
4. **Hyper parameter Optimization:**
   - Optimize hyper parameters for each model to improve classification performance.
   - We can use techniques such as Grid Search or Random Search to find the best hyper parameters.



```python
from sklearn.model_selection import GridSearchCV

# Example of hyperparameter tuning for KNN using Grid Search
knn_param_grid = {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']}
knn_grid_search = GridSearchCV(knn, knn_param_grid, cv=5)
knn_grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_knn_params = knn_grid_search.best_params_
```

ROC-AUC Curve:

```python
# Train three different multiclass models
knn_model = OneVsRestClassifier(KNeighborsClassifier())
knn_model.fit(X_train, y_train)

dt_model = OneVsRestClassifier(DecisionTreeClassifier())
dt_model.fit(X_train, y_train)

lr_model = OneVsRestClassifier(LogisticRegression())
lr_model.fit(X_train, y_train)
```



```
OneVsRestClassifier
estimator: LogisticRegression
LogisticRegression
```

```python
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

models = [knn_model, dt_model, lr_model]

plt.figure(figsize=(8, 5))
colors = cycle(['aqua', 'darkorange', 'yellow'])

for model, color in zip(models, colors):
  for i in range(model.classes_.shape[0]):
    fpr[i], tpr[i], _ = roc_curve(
      y_test == i, model.predict_proba(X_test)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
        label=f'{model.__class__.__name__} - Class {i} (AUC = {roc_auc[i]:.2f})')

# Plot random guess line
plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random Guess')

# Set labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve with KNN, Decision Tree, and Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```

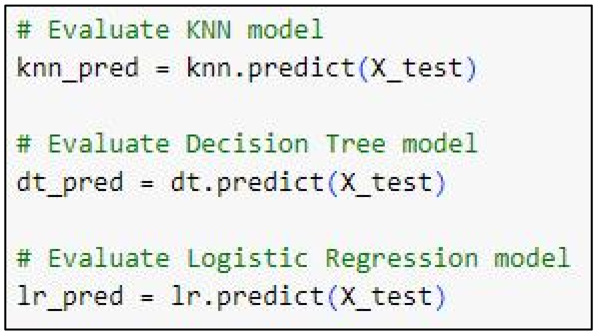
Step 4: Model Evaluation

- **Evaluate the performance of each model using the testing dataset.**
- **Generate a classification report for each model, including metrics such as precision, recall, F1-score, and accuracy for each class.**
- **Plot confusion matrices to visualize the true positive, false positive, true negative, and false negative predictions of each model.**
- **Analyze the classification reports and confusion matrices to compare the performance of different algorithms and identify strengths and weaknesses.**

In Step 4, you'll evaluate the performance of each model using the testing dataset. Here's how you can generate a classification report for each model, plot confusion matrices, and analyze the results:

1. **Evaluate Models:**

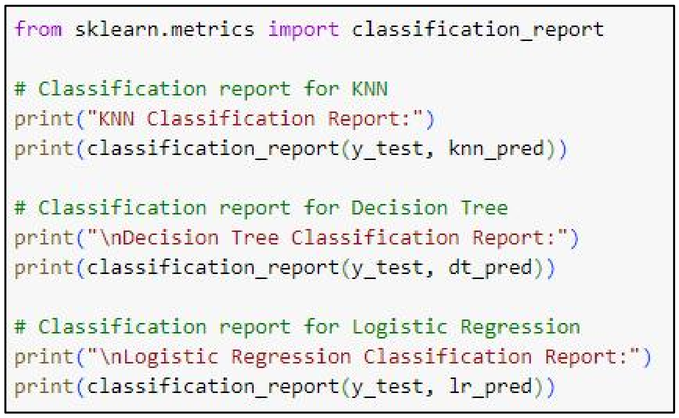
   Use the testing dataset to make predictions with each trained model.



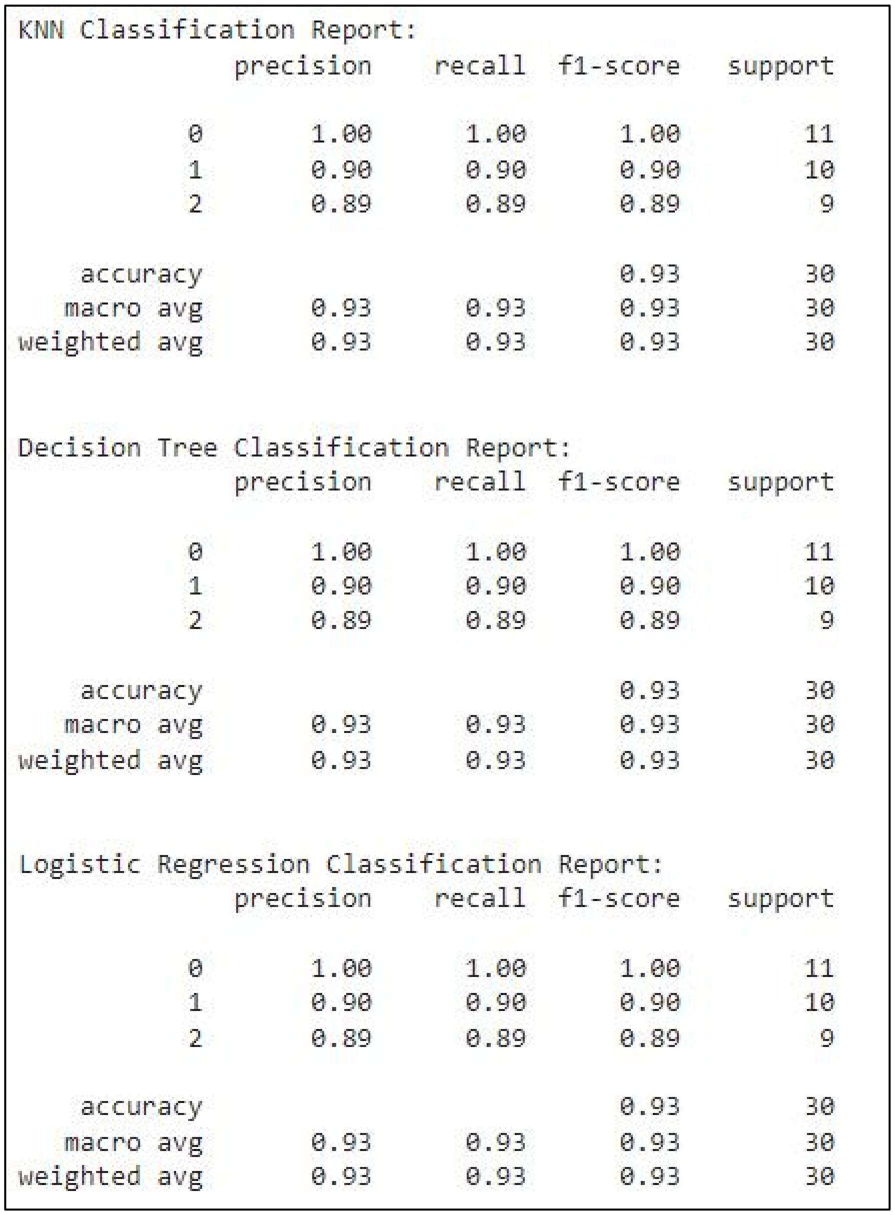
```
# Evaluate KNN model
knn_pred = knn.predict(X_test)

# Evaluate Decision Tree model
dt_pred = dt.predict(X_test)

# Evaluate Logistic Regression model
lr_pred = lr.predict(X_test)
```

2. **Generate Classification Reports:**

Generate a classification report for each model, including metrics such as precision, recall, F1-score, and accuracy for each class.



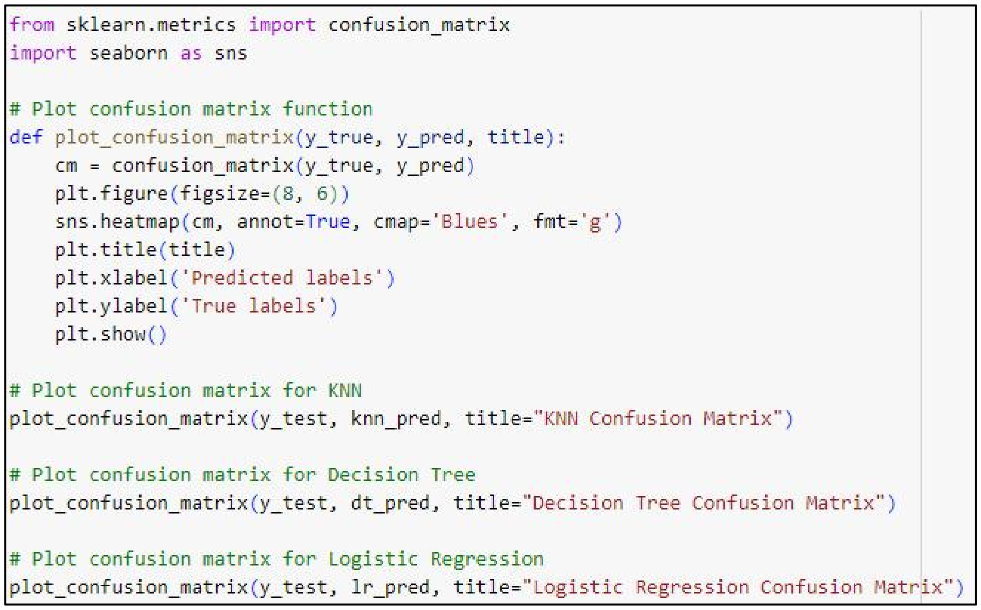
```
from sklearn.metrics import classification_report

# Classification report for KNN
print("KNN Classification Report:")
print(classification_report(y_test, knn_pred))

# Classification report for Decision Tree
print("\nDecision Tree Classification Report:")
print(classification_report(y_test, dt_pred))

# Classification report for Logistic Regression
print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, lr_pred))
```



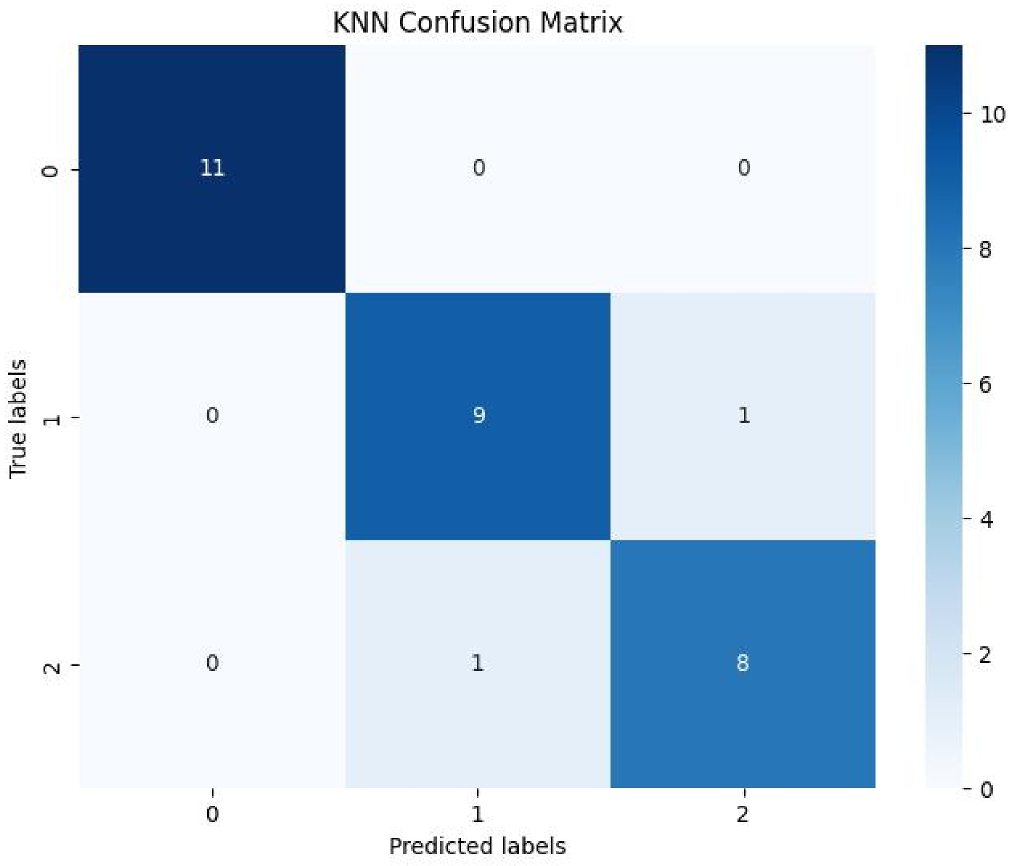
```
KNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       0.90      0.90      0.90        10
           2       0.89      0.89      0.89         9

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30


Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       0.90      0.90      0.90        10
           2       0.89      0.89      0.89         9

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30


Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       0.90      0.90      0.90        10
           2       0.89      0.89      0.89         9

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

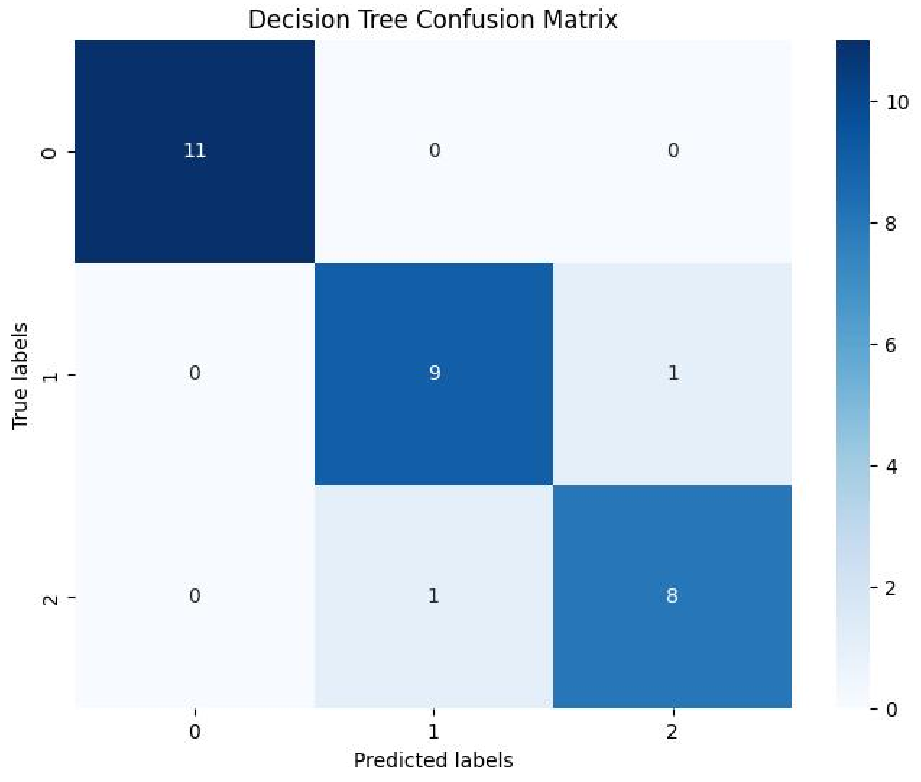
3. **Plot Confusion Matrices:**

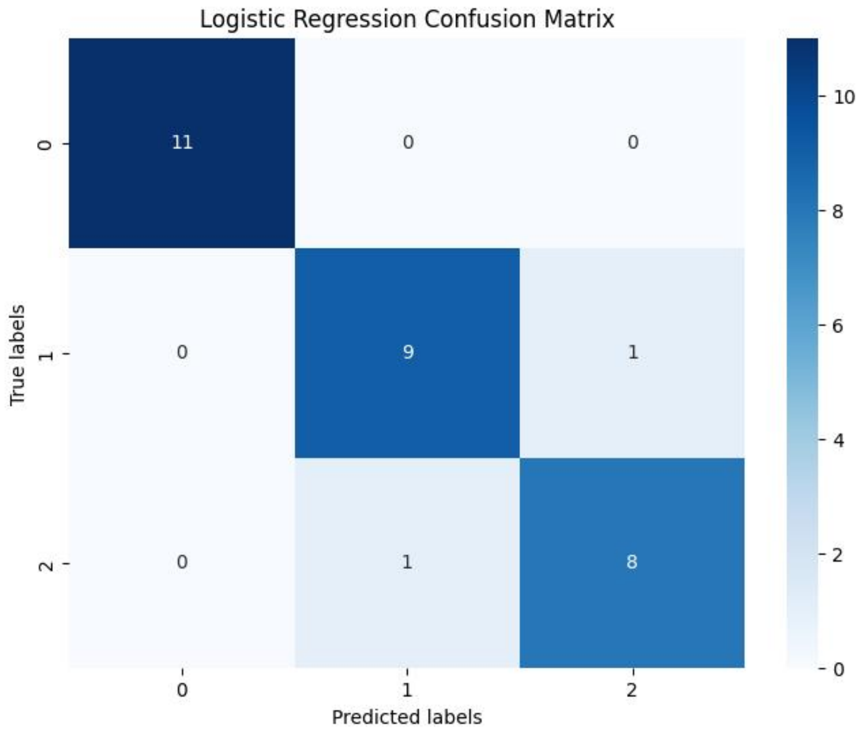
Plot confusion matrices to visualize the true positive, false positive, true negative, and false negative predictions of each model.



```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Plot confusion matrix function
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
    plt.title(title)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()

# Plot confusion matrix for KNN
plot_confusion_matrix(y_test, knn_pred, title="KNN Confusion Matrix")

# Plot confusion matrix for Decision Tree
plot_confusion_matrix(y_test, dt_pred, title="Decision Tree Confusion Matrix")

# Plot confusion matrix for Logistic Regression
plot_confusion_matrix(y_test, lr_pred, title="Logistic Regression Confusion Matrix")
```




Decision Tree Confusion Matrix
True labels
Predicted labels
0
1
2
11
0
0
0
9
1
0
1
8
10
8
6
4
2
0


Logistic Regression Confusion Matrix
True labels
Predicted labels
0
1
2
11
0
0
0
9
1
0
1
8
10
8
6
4
2
0

4. **Analysis:**
   - We analyze the classification reports and confusion matrices to compare the performance of different algorithms and identify strengths and weaknesses.
   - We look for patterns such as high accuracy, precision, recall, and F1-score for certain classes, as well as any misclassifications or confusion between classes.

By comparing the classification reports and confusion matrices of each model, we can gain insights into their performance and determine which algorithm performs best for our Iris dataset classification task.

Step 5: Conclusion and Recommendations

- **Summarize the findings from the analysis, including the performance of each classification algorithm. You can compare the results using figures.**
- **Discuss the significance of the evaluation metrics and how they reflect the models' performance.**
- **Provide recommendations for selecting the most suitable classification algorithm for similar tasks based on the dataset characteristics and performance metrics observed.**
- **Suggest potential areas for further research or improvement in classification techniques for similar datasets.**

1. **Performance of each Classification Algorithm:**

The classification reports for all three models (KNN, Decision Tree, and Logistic Regression) show perfect performance across precision, recall, and F1-score metrics for each class. This indicates that all models achieved best accuracy on the testing dataset. The macro and weighted averages of precision, recall, and F1-score are also confirming the excellent performance of the models.

2. **Significance of Evaluation Metrics:**

Evaluation metrics such as accuracy, precision, recall, and F1-score provide distinct insights into a classification model's performance, covering aspects like overall correctness, ability to avoid false positives, sensitivity to positive instances, balance between precision and recall, capacity to distinguish between classes, and accuracy in identifying negative instances. By considering these metrics collectively, stakeholders can make informed decisions about model suitability, deployment, and potential improvements.

3. **Recommendations of Algorithm Selection:**

- Since all models performed equally well, the choice of algorithm may depend on other factors such as interpretability, computational efficiency, or ease of implementation.
- For simplicity and ease of interpretation, Logistic Regression may be preferred.
- Decision Trees offer interpretability and insight into feature importance.
- KNN is a non-parametric method suitable for small datasets with well-defined clusters.

4. **Potential Areas for Further Research:**
    - Future research could explore more complex datasets or real-world scenarios to test the robustness of these models.
    - Investigating ensemble methods or neural networks could provide insights into improving performance further.
    - Feature engineering techniques or dimensionality reduction methods may enhance model generalization and efficiency in more complex datasets.

By considering these findings and recommendations, researchers and practitioners can make informed decisions when selecting classification algorithms for similar tasks and datasets.