Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Department of Computer Science and Engineering

Assignment – II

Regulations – R2021

Degree B.E. / B. Tech.	B.E.	**Branch**	CSE
Semester	II	**Academic Year**	**2022-2023**
Subject Code & Name	UCS2202 – Foundations of Data Science		
Batch: 2022-2026			**Maximum: 30 Marks**

By: **AKSHATHA ANABALAGAN** _ 31 2222 5001 **007**

ABHISHEK REDDY LINGAREDDY _ 31 2222 5001 **003**

1.Identify your own dataset to perform data modeling.

The dataset chosen is Iris dataset, The dataset consists of measurements of different attributes of iris flowers, such as sepal length, sepal width, petal length, and petal width, along with their corresponding species (Iris setosa, Iris versicolor, and Iris virginica).

The dataset presents some difficulties in creating an effective model due to factors such as overlapping measurements and subtle variations among the iris species. The challenge lies in identifying and leveraging the subtle differences in the measurements that distinguish one species from another.

The Iris dataset is often used for classification tasks, where the goal is to predict the species of an iris flower based on its measurements. It is considered a good dataset for model creation .

Although the Iris dataset may appear straightforward at first glance, its unique characteristics and intricacies make it a challenging task to build a model that can accurately classify the iris flowers. Therefore, it is important to approach the model-building process for this dataset with careful consideration and meticulous analysis to achieve satisfactory results.

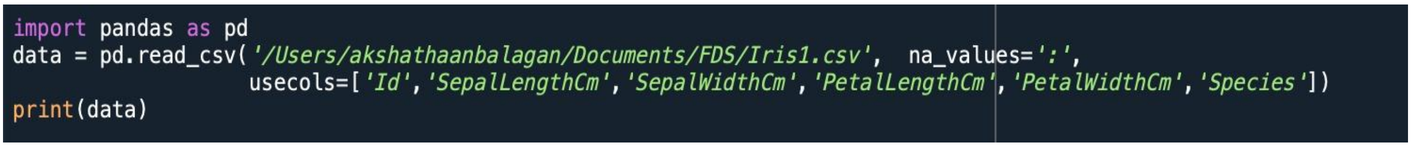
Dataset Source:

https://www.kaggle.com/datasets/saurabh00007/iriscsv?resource=download

2.Develop a python code to perform the following:

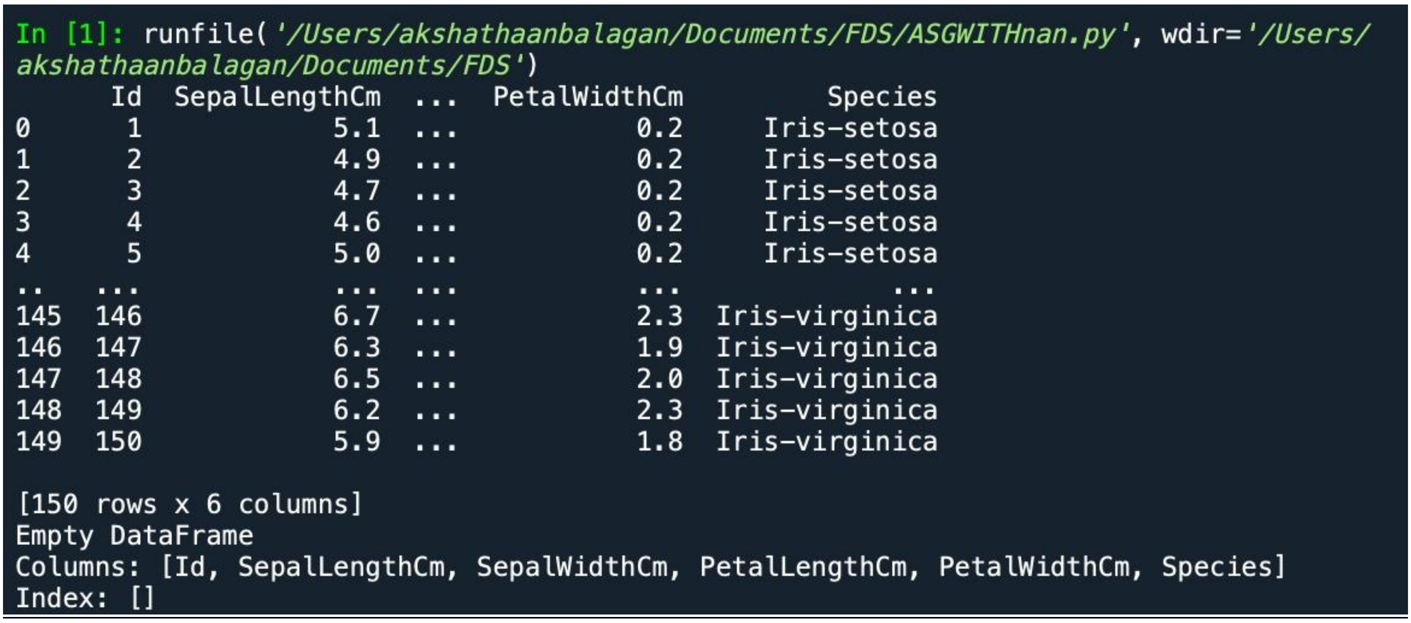
- To read the dataset

CODE:



```
import pandas as pd
data = pd.read_csv('/Users/akshathaanbalagan/Documents/FDS/Iris1.csv',  na_values=':',
                   usecols=['Id','SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Species'])
print(data)
```

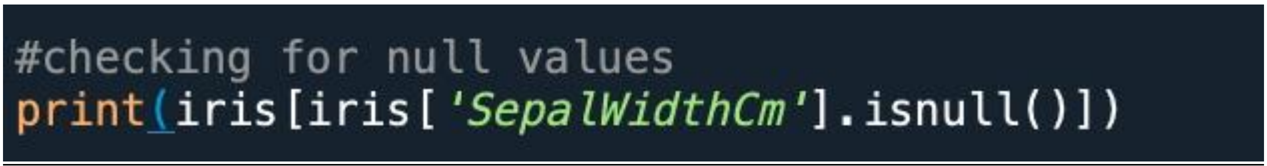
OUTPUT:



```
In [1]: runfile('/Users/akshathaanbalagan/Documents/FDS/ASGWITHnan.py', wdir='/Users/
akshathaanbalagan/Documents/FDS')
      Id  SepalLengthCm  ...  PetalWidthCm         Species
0      1            5.1  ...           0.2     Iris-setosa
1      2            4.9  ...           0.2     Iris-setosa
2      3            4.7  ...           0.2     Iris-setosa
3      4            4.6  ...           0.2     Iris-setosa
4      5            5.0  ...           0.2     Iris-setosa
..   ...            ...  ...           ...             ...
145  146            6.7  ...           2.3  Iris-virginica
146  147            6.3  ...           1.9  Iris-virginica
147  148            6.5  ...           2.0  Iris-virginica
148  149            6.2  ...           2.3  Iris-virginica
149  150            5.9  ...           1.8  Iris-virginica

[150 rows x 6 columns]
Empty DataFrame
Columns: [Id, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm, Species]
Index: []
```

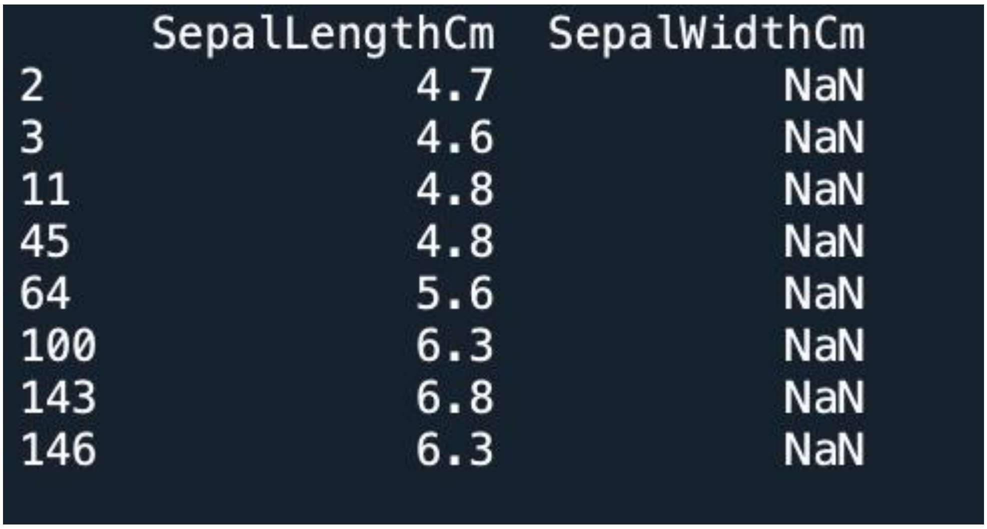
- To check for null values

CODE:



```
#checking for null values
print(iris[iris['SepalWidthCm'].isnull()])
```

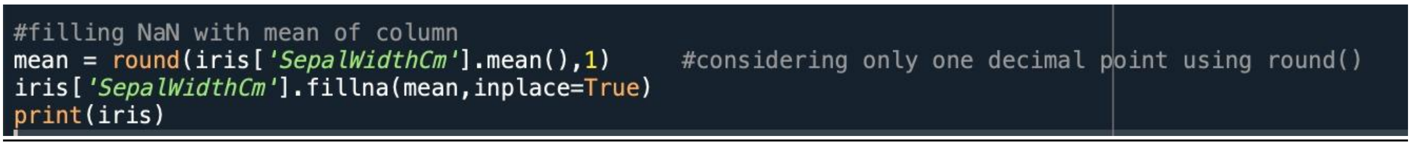
OUTPUT:



```
     SepalLengthCm  SepalWidthCm
2              4.7           NaN
3              4.6           NaN
11             4.8           NaN
45             4.8           NaN
64             5.6           NaN
100            6.3           NaN
143            6.8           NaN
146            6.3           NaN
```

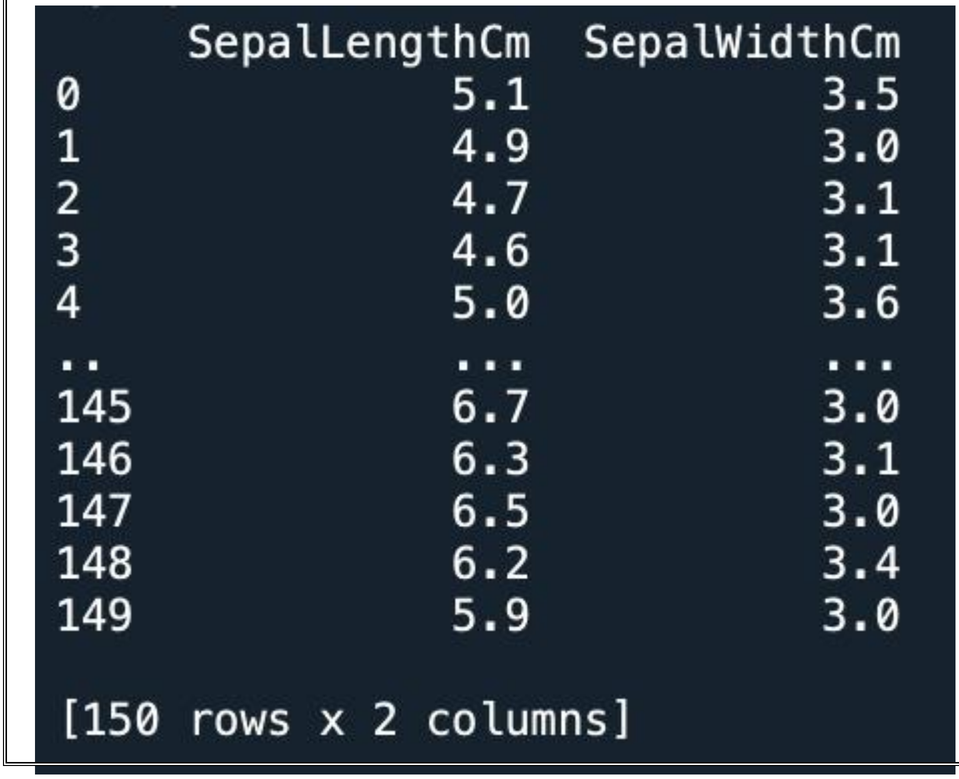
- To handle missing values

CODE:



```
#filling NaN with mean of column
mean = round(iris['SepalWidthCm'].mean(),1)     #considering only one decimal point using round()
iris['SepalWidthCm'].fillna(mean,inplace=True)
print(iris)
```

OUTPUT:



```
     SepalLengthCm  SepalWidthCm
0              5.1           3.5
1              4.9           3.0
2              4.7           3.1
3              4.6           3.1
4              5.0           3.6
..             ...           ...
145            6.7           3.0
146            6.3           3.1
147            6.5           3.0
148            6.2           3.4
149            5.9           3.0

[150 rows x 2 columns]
```

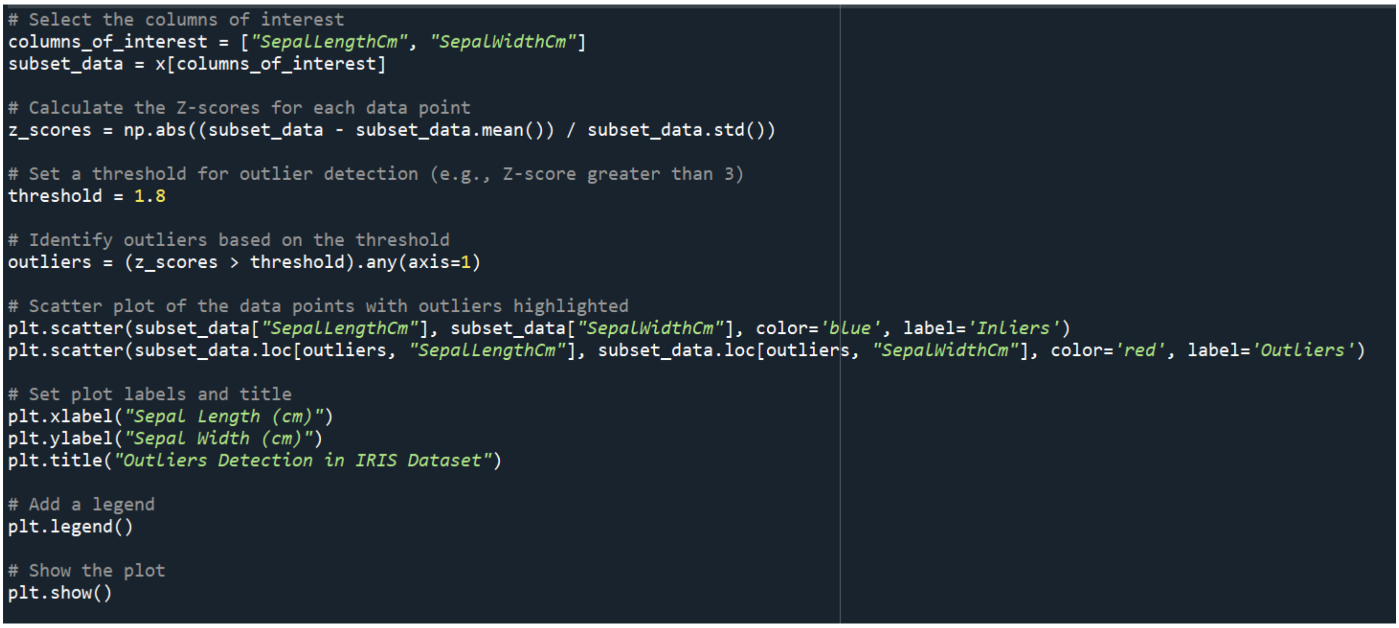
3.Examine all the approaches suitable for the identified dataset based on the exploratory data analysis.

OUTLIERS:

By employing statistical methods such as the Z-score approach, outliers in these columns can be identified. The Z-score measures the number of standard deviations a data point is away from the mean. Using a defined threshold, such as a Z-score greater than 1.8, allows for the detection of outliers.

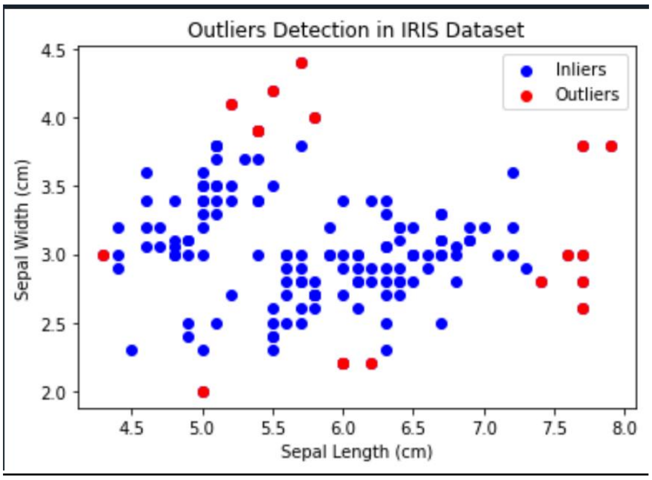
These outliers may indicate measurement errors, data entry mistakes, or truly unique and distinct observations within the dataset. It is crucial to carefully examine outliers to determine their validity and potential impact on analysis or modeling tasks. Understanding and addressing outliers can contribute to improved data quality and more accurate interpretations of the iris flower attributes present in the IRIS dataset.

CODE:



```
# Select the columns of interest
columns_of_interest = ["SepalLengthCm", "SepalWidthCm"]
subset_data = x[columns_of_interest]

# Calculate the Z-scores for each data point
z_scores = np.abs((subset_data - subset_data.mean()) / subset_data.std())

# Set a threshold for outlier detection (e.g., Z-score greater than 3)
threshold = 1.8

# Identify outliers based on the threshold
outliers = (z_scores > threshold).any(axis=1)

# Scatter plot of the data points with outliers highlighted
plt.scatter(subset_data["SepalLengthCm"], subset_data["SepalWidthCm"], color='blue', label='Inliers')
plt.scatter(subset_data.loc[outliers, "SepalLengthCm"], subset_data.loc[outliers, "SepalWidthCm"], color='red', label='Outliers')

# Set plot labels and title
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("Outliers Detection in IRIS Dataset")

# Add a legend
plt.legend()

# Show the plot
plt.show()
```

OUTPUT:

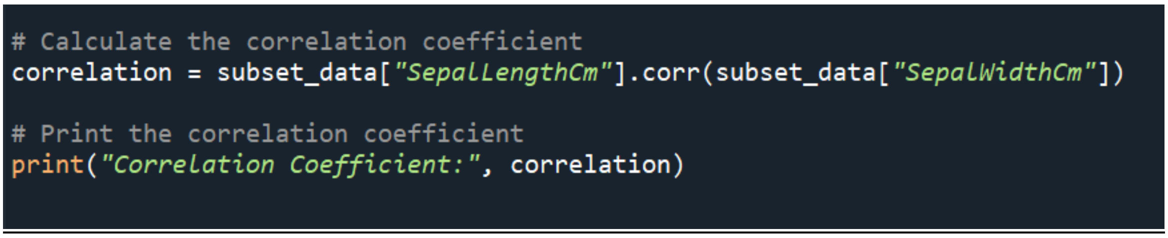


CORRELATION COEFFICIENT:

By calculating the correlation coefficient between the "SepalLengthCm" and "SepalWidthCm" columns, we can assess how closely related these attributes are.

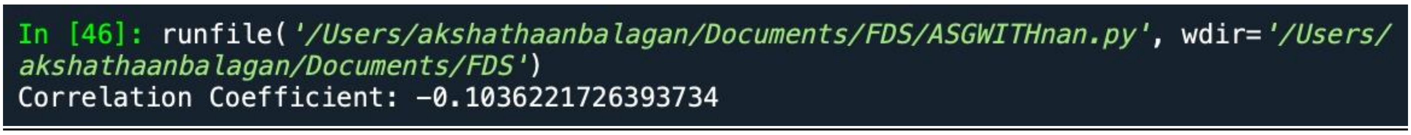
After calculating the correlation coefficient for the "SepalLengthCm" and "SepalWidthCm" columns in the IRIS dataset, we can interpret the value to understand the degree and direction of the correlation. This insight aids in understanding the relationship between sepal length and sepal width, which can be crucial for further analysis and decision-making in areas such as species classification, feature selection, or understanding the underlying biology of the iris flowers.

CODE:



```
# Calculate the correlation coefficient
correlation = subset_data["SepalLengthCm"].corr(subset_data["SepalWidthCm"])

# Print the correlation coefficient
print("Correlation Coefficient:", correlation)
```

OUTPUT:



```
In [46]: runfile('/Users/akshathaanbalagan/Documents/FDS/ASGWITHnan.py', wdir='/Users/
akshathaanbalagan/Documents/FDS')
Correlation Coefficient: -0.1036221726393734
```

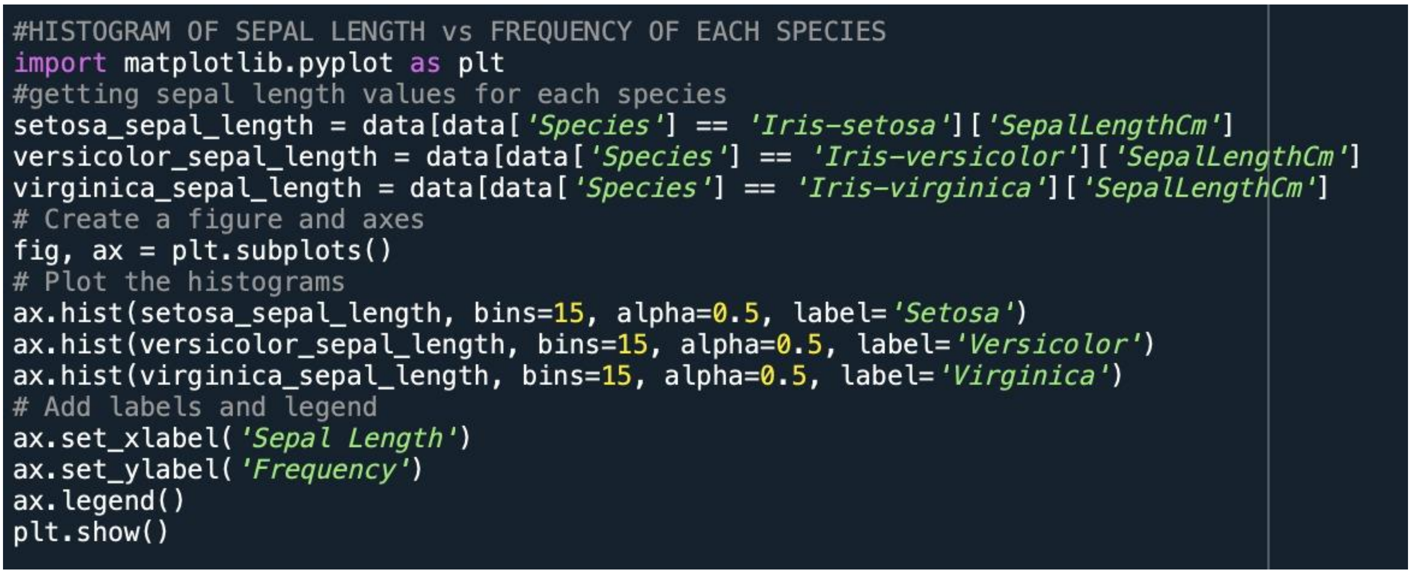
HISTOGRAM:

By creating a histogram for the sepal length of the three species, we can gain insights into the distribution of sepal lengths and identify any patterns or differences between the species. This visualization allows us to understand the range and frequency of sepal lengths for each species.

Through the histogram, we can observe the shape of the distributions, whether they are skewed or symmetric, and identify any potential outliers or gaps in the data. Additionally, comparing the histograms of different species helps us distinguish their unique characteristics and potentially differentiate them based on sepal length.

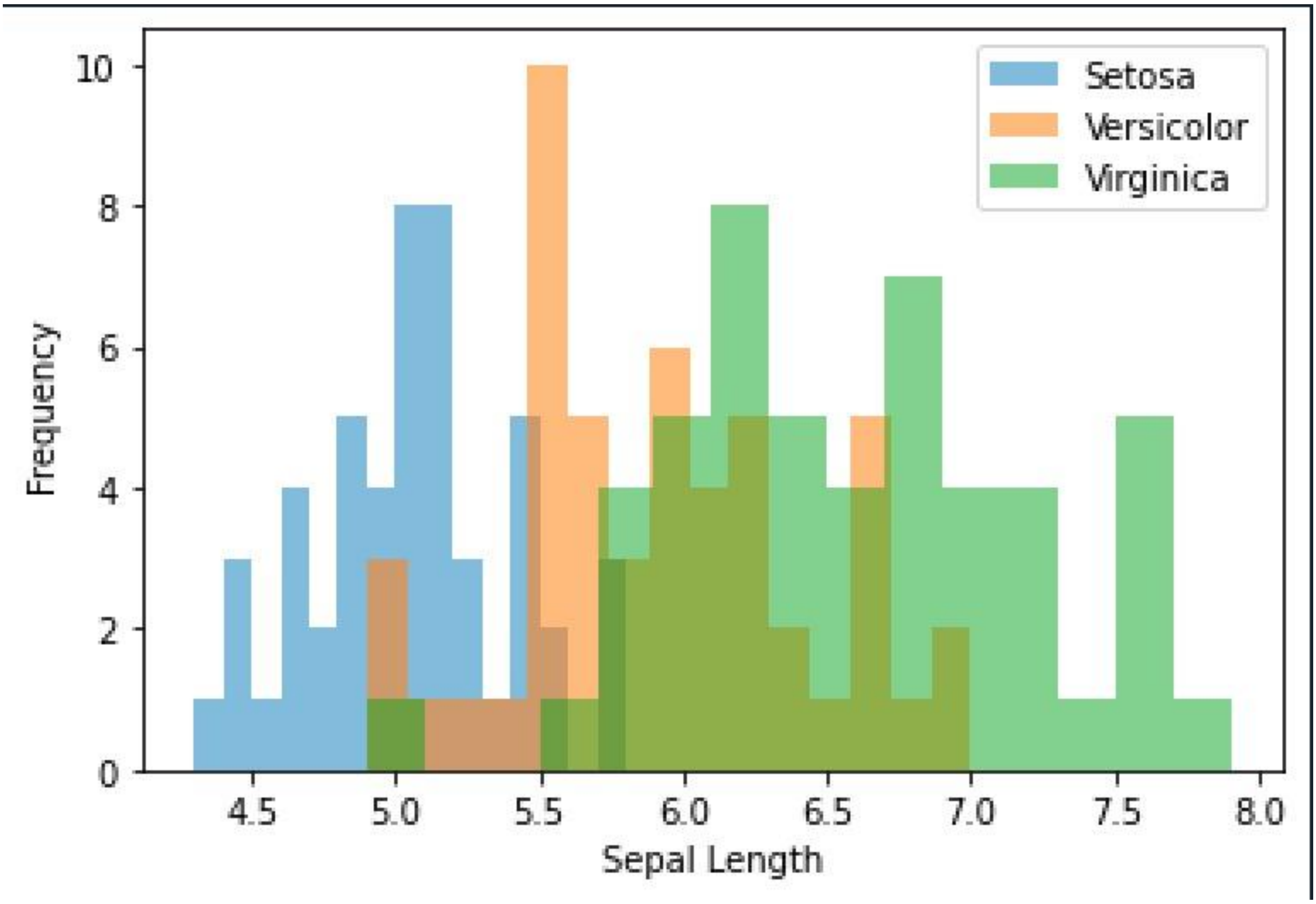
Analyzing the sepal length distribution using histograms enhances our understanding of the iris dataset, enabling us to draw conclusions about the sepal length variation across different species and aiding in the identification and classification of iris flowers based on this attribute.

CODE:



```
#HISTOGRAM OF SEPAL LENGTH vs FREQUENCY OF EACH SPECIES
import matplotlib.pyplot as plt
#getting sepal length values for each species
setosa_sepal_length = data[data['Species'] == 'Iris-setosa']['SepalLengthCm']
versicolor_sepal_length = data[data['Species'] == 'Iris-versicolor']['SepalLengthCm']
virginica_sepal_length = data[data['Species'] == 'Iris-virginica']['SepalLengthCm']
# Create a figure and axes
fig, ax = plt.subplots()
# Plot the histograms
ax.hist(setosa_sepal_length, bins=15, alpha=0.5, label='Setosa')
ax.hist(versicolor_sepal_length, bins=15, alpha=0.5, label='Versicolor')
ax.hist(virginica_sepal_length, bins=15, alpha=0.5, label='Virginica')
# Add labels and legend
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Frequency')
ax.legend()
plt.show()
```

OUTPUT:

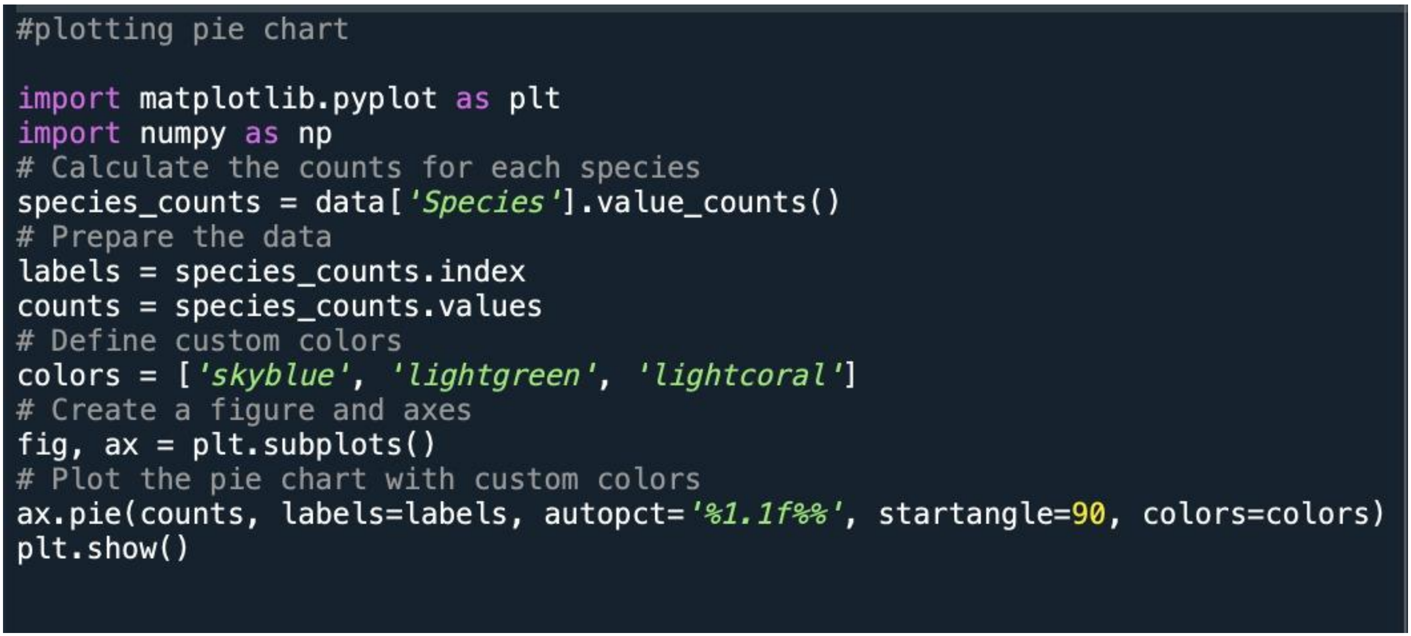


PIE CHART:

By creating a pie chart for the species distribution, we can visually compare the percentage or frequency of each species within the dataset. The size of each "slice" of the pie represents the proportion of that species compared to the total number of species in the dataset.

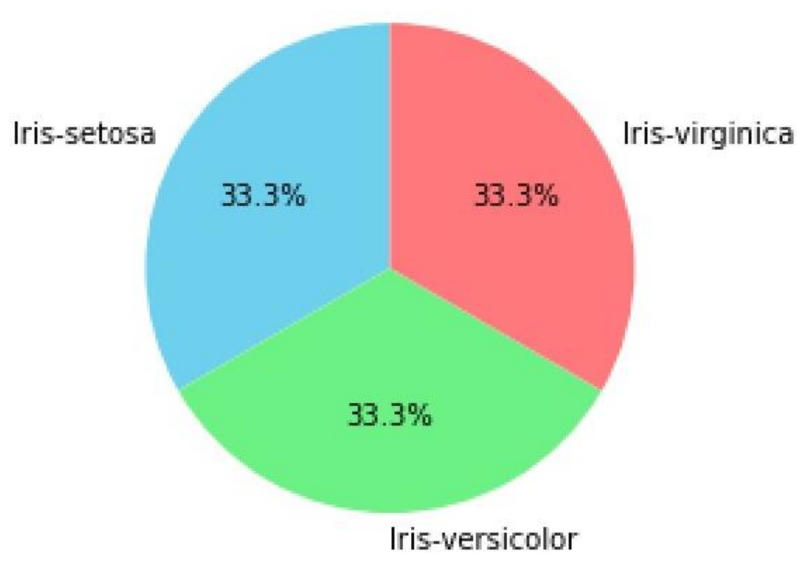
This pie chart provides a quick and intuitive overview of the distribution of iris species in the dataset. It allows us to identify any dominance or imbalance in species representation and visualize the relative frequencies or proportions of each species.

CODE:



```
#plotting pie chart

import matplotlib.pyplot as plt
import numpy as np
# Calculate the counts for each species
species_counts = data['Species'].value_counts()
# Prepare the data
labels = species_counts.index
counts = species_counts.values
# Define custom colors
colors = ['skyblue', 'lightgreen', 'lightcoral']
# Create a figure and axes
fig, ax = plt.subplots()
# Plot the pie chart with custom colors
ax.pie(counts, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors)
plt.show()
```

OUTPUT:



4.Choose an appropriate approach that can be used for model building.

K-Nearest Neighbors (KNN) is indeed a suitable approach for model building with the IRIS dataset. KNN is a non-parametric classification algorithm that works well with small to medium-sized datasets. The IRIS dataset consists of 150 samples and 4 input features, making it an appropriate size for KNN.

KNN classifies new instances based on the "k" closest training instances in the feature space. It measures the distance between the new instance and the existing instances and assigns the class label based on the majority class of the nearest neighbors. KNN is a simple yet effective algorithm that doesn't make strong assumptions about the underlying data distribution.

To build a KNN model for the IRIS dataset, you can use the following steps:

1.Load the dataset using a library like pandas.

2.Separate the input features (e.g., sepal length, sepal width, petal length, petal width) into a feature matrix.

3.Separate the target variable (e.g., species) into a target vector.

4.Split the dataset into training and testing sets using functions like train_test_split from sklearn.model_selection.

5.Scale the feature matrix using techniques like StandardScaler from sklearn.preprocessing, as KNN is sensitive to the scale of the features.

6. Create an instance of the KNeighborsClassifier from sklearn.neighbors. Fit the model to the training data using the fit method.

7. Fit the model to the training data using the fit method.

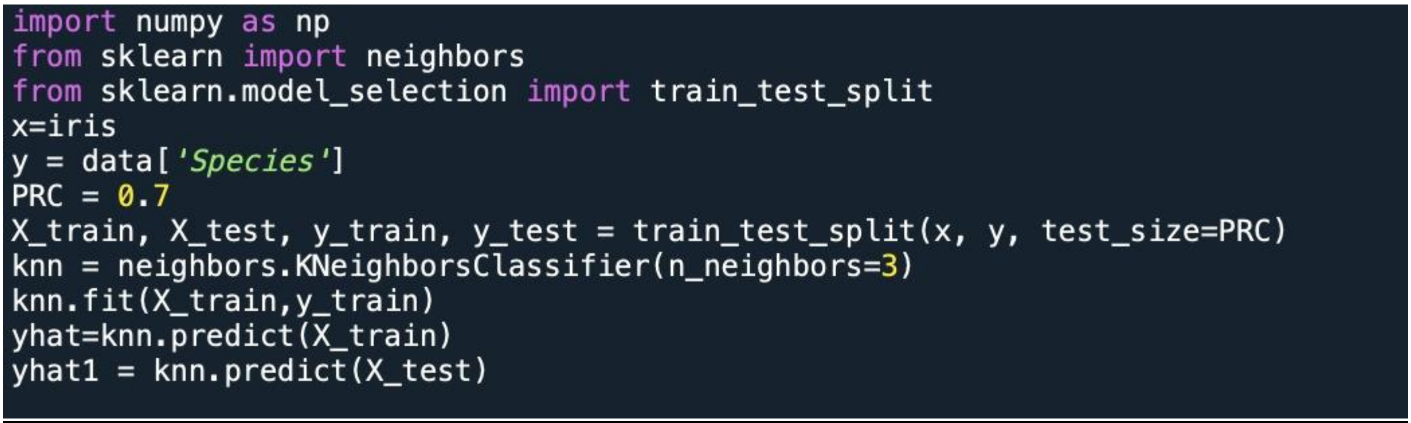
8. Use the trained model to make predictions on the testing data using the predict method.

9.Evaluate the performance of the model using metrics like accuracy_score, confusion_matrix, or classification_report from sklearn.metrics.

Overall, KNN is a suitable approach for building a model with the IRIS dataset due to its simplicity, ability to handle multi-class classification, and relatively small size of the dataset.

5.Develop a python code to build the model.

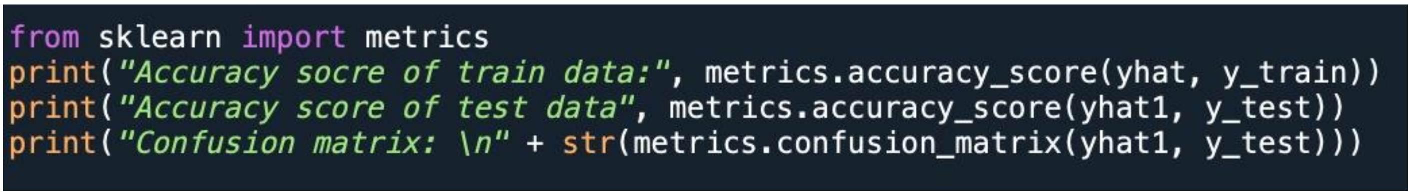
CODE:



```
import numpy as np
from sklearn import neighbors
from sklearn.model_selection import train_test_split
x=iris
y = data['Species']
PRC = 0.7
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=PRC)
knn = neighbors.KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
yhat=knn.predict(X_train)
yhat1 = knn.predict(X_test)
```

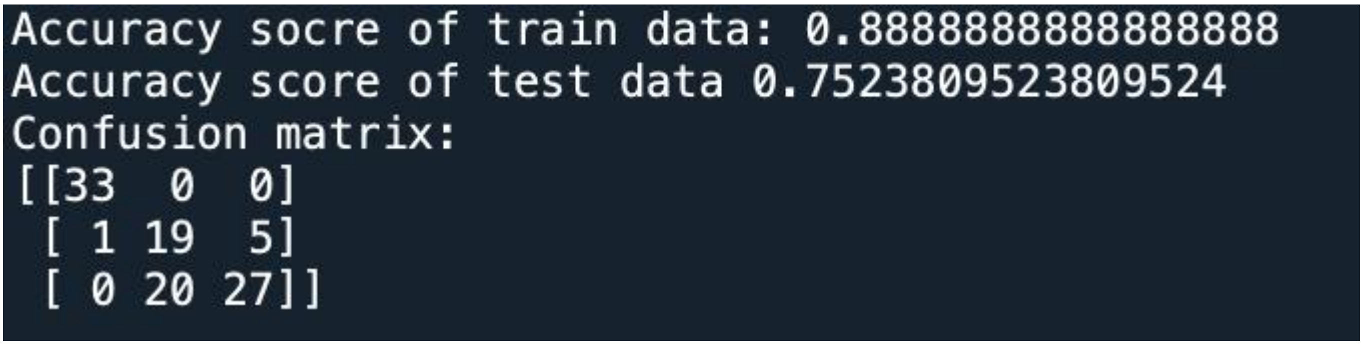
6.Evaluate the built model using appropriate evaluation metrics.

CODE:



```
from sklearn import metrics
print("Accuracy socre of train data:", metrics.accuracy_score(yhat, y_train))
print("Accuracy score of test data", metrics.accuracy_score(yhat1, y_test))
print("Confusion matrix: \n" + str(metrics.confusion_matrix(yhat1, y_test)))
```

OUTPUT:



```
Accuracy socre of train data: 0.8888888888888888
Accuracy score of test data 0.7523809523809524
Confusion matrix:
[[33  0  0]
 [ 1 19  5]
 [ 0 20 27]]
```