INTERN PROJECT PHASE – 1

PROJECT – 1

Analyze Iris Data

Aarsh Mehtani

Contents

1. Introduction
2. Objective
3. Introduction to the dataset
4. Importing the libraries
5. Data Overview
   - There are 150 rows and 6 columns in the dataset.
   - Column Description
   - Checking Missing Values
   - Checking Duplicates
6. Data Visualization
   - Bi Variate Analysis
   - Histograms with Distplot Plot, Histogram Plot, Box Plot
   - Box Plots
   - Handling Correlation
   - Heatmaps
7. Feature Engineering
   - Handling Outliers
   - Updating Outliers
   - Encoding
   - Feature Extraction
8. Building the ML model
   - Logistic Regression
   - Decision Tree
   - Random Forest
9. Conclusion
10. Automated ML model Pipeline!

-# INTRODUCTION

Iris Dataset is considered as the Hello World for data science. It contains five columns namely – **Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type**. Iris is a flowering plant. The researchers have measured various features of the different iris flowers and recorded them digitally.

OBJECTIVE

- Utilize the Iris dataset to perform a DATA SCIENCE TASK.

- Conduct a Simple EXPLORATORY DATA ANALYSIS (EDA) to gain insights into the dataset.

INTRODUCTION TO THE DATA SET

- *Id – Id number given to each tuple.*
- *SepalLengthCm – Define the length of the Sepal of each plant Species in cm.*
- *SepalWidthCm - Define the width of the Sepal of each plant Species in cm.*
- *PetalLengthCm - Define the length of the Petal of each plant Species in cm.*
- *PetalWidthCm - Define the width of the Petal of each plant Species in cm.*
- *Species – Define the Species according to the features.*
  - *Iris-setosa*
  - *Iris-versicolor*
  - *Iris-virginica*

Data Set Characteristics	N/A	Number of Instances	150
Attribute Characteristics	N/A	Number of Attribute	6
Associated Tasks	Classification	Missing Values	N/A

IMPORTING THE LIBRARIES

- Libraries for graphs and plots: matplotlib.pyplot, NumPy, Pandas, seaborn
- Libraries for ML classification: sklearn

1.. Lets read data !

```
In [1]: ## import necessary packages !

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

Data visuaslisation(EDA)

```
In [15]: import seaborn as sns
```

Encoding

```
In [34]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
```

Build Manchine Learning Model

In [39]:
```
from sklearn.model_selection import train_test_split
```

Models:

1. Logistic Regression

2. Decision Tree

3. Random Forest

In [46]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
model_L=LogisticRegression(multi_class='multinomial')
model_D=DecisionTreeClassifier(max_depth=8)
model_R=RandomForestRegressor()
```

In [52]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

DATA OVERVIEW

Import the dataset using pd.read_csv() function.
The first 5 rows of data set.

Importing dataset

```
Since data is in form of excel file we have to use pandas read_excel to load the data
```

In [2]:
```
data=pd.read_csv("Iris.csv")
df=data.copy()
```

In [3]:
```
df.head(5)
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

The describe() method prints the information about the DataSet.
The information conations the count, mean, std, min, max, 25%, 75%.

In [4]:
```
df.describe()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

We can see that only one column has categorical data, and all the other columns are of the numeric type with non-Null entries.

The info() method prints information about the DataFrame.
The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

```
In [5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Checking Missing Values

We will check if our data contains any missing values or not. Missing values can occur when no information is provided for one or more items or for a whole unit. We will use the isnull() method.

```
In [6]: ## After loading it is important to check null/missing values in a column or a row
        ## Missing value :  values which occur when no data is recorded for an observation..

        df.isnull().sum()

        ## train_data.isnull().sum(axis=0)
        ## by-default axis is 0 , ie it computes total missing values column-wise !

Out[6]: Id               0
        SepalLengthCm    0
        SepalWidthCm     0
        PetalLengthCm    0
        PetalWidthCm     0
        Species          0
        dtype: int64
```

We can see that no column has any missing value.

Checking Duplicates

Let's see if our dataset contains any duplicates or not. The panda drop_duplicatews() method helps in removing duplicates from the data frame.

2.b.. Lets deal with Duplicate values ..

```
In [7]: data_dup=df.drop_duplicates(subset='Species')
        data_dup
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
100	101	6.3	3.3	6.0	2.5	Iris-virginica

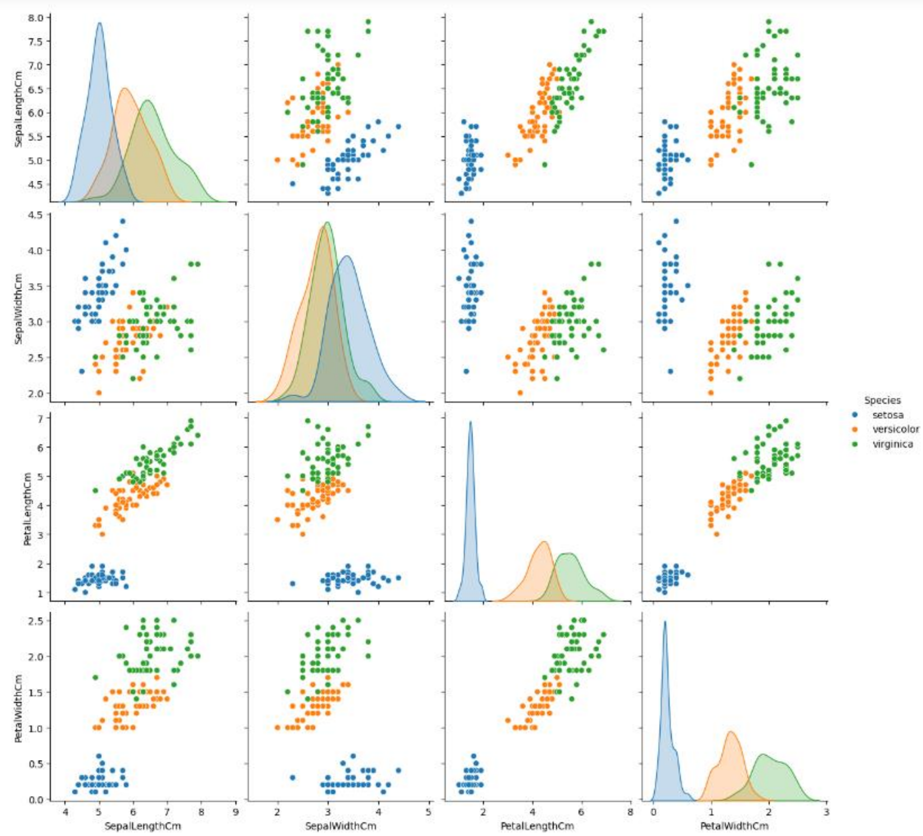
```
In [8]: df.value_counts('Species')

Out[8]: Species
        Iris-setosa        50
        Iris-versicolor    50
        Iris-virginica     50
        Name: count, dtype: int64
```

We can see that all the species contain an equal number of rows, so we should not delete any entries.

DATA VISUALIZATION

1. Relation between variables – Bi Variate Analysis

```
In [16]: # We can use pairplot for mutivariate analysis:
         sns.pairplot(df.drop(['Id'], axis = 1), hue='Species', height=3,)
```



Description:

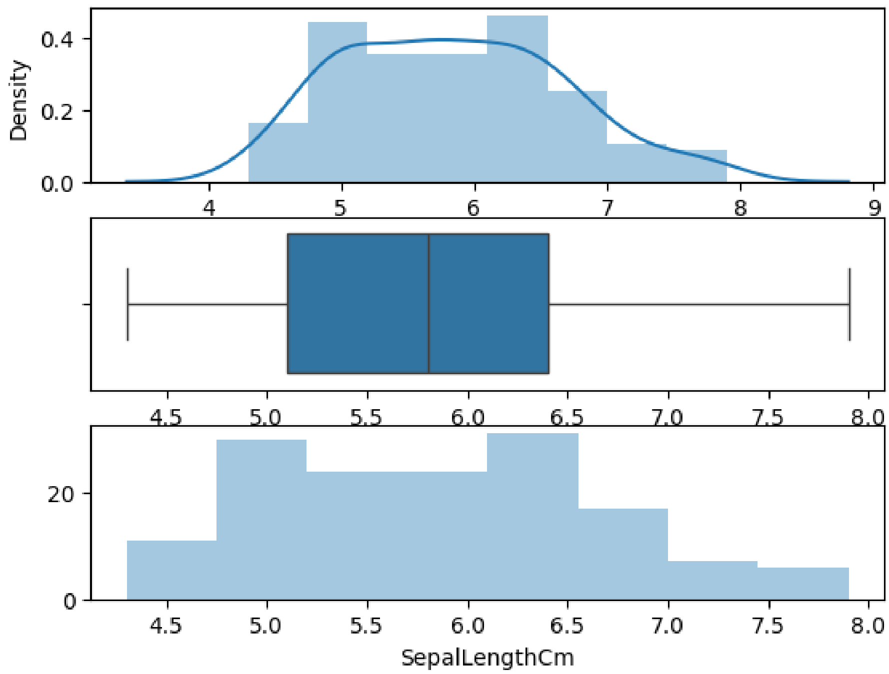
Overall Observations:

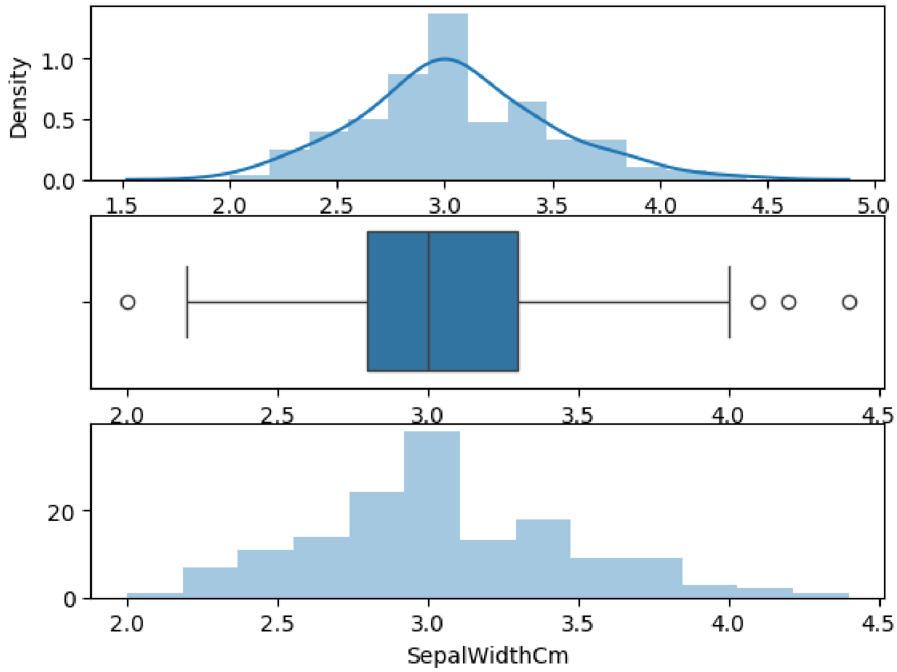
- ❖ Setosa has the smallest of petals widths and lengths.
- ❖ It also has the smallest sepal length but larger sepal widths.
- ❖ In the case of Sepal Length, there is a huge amount of overlapping.
- ❖ In the case of Sepal Width also, there is a huge amount of overlapping.
- ❖ In the case of Petal Length, there is a very little amount of overlapping.
- ❖ In the case of Petal Width also, there is a very little amount of overlapping.

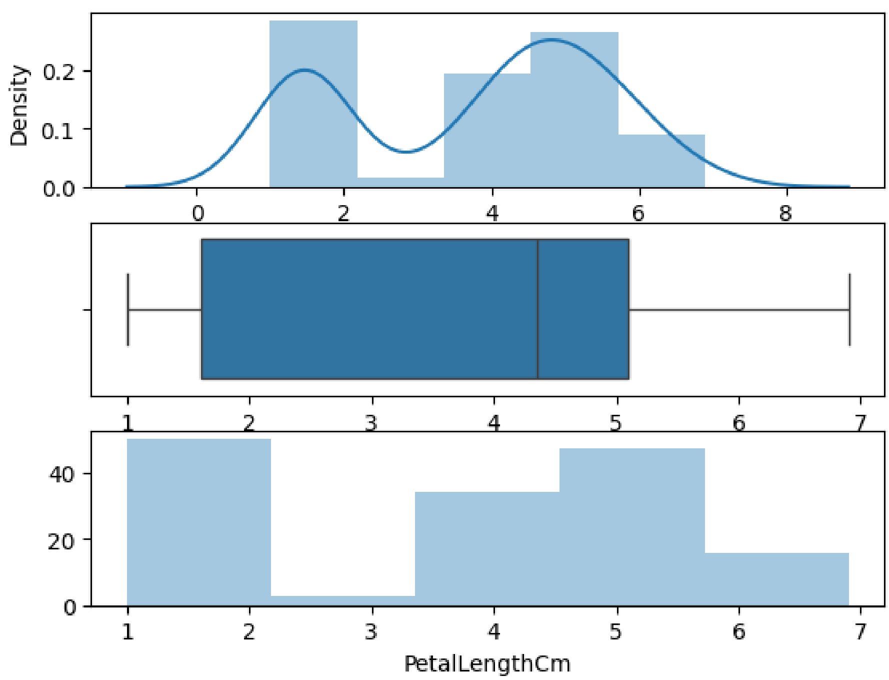
2. Histograms with Distplot Plot, Histogram Plot, Box Plot

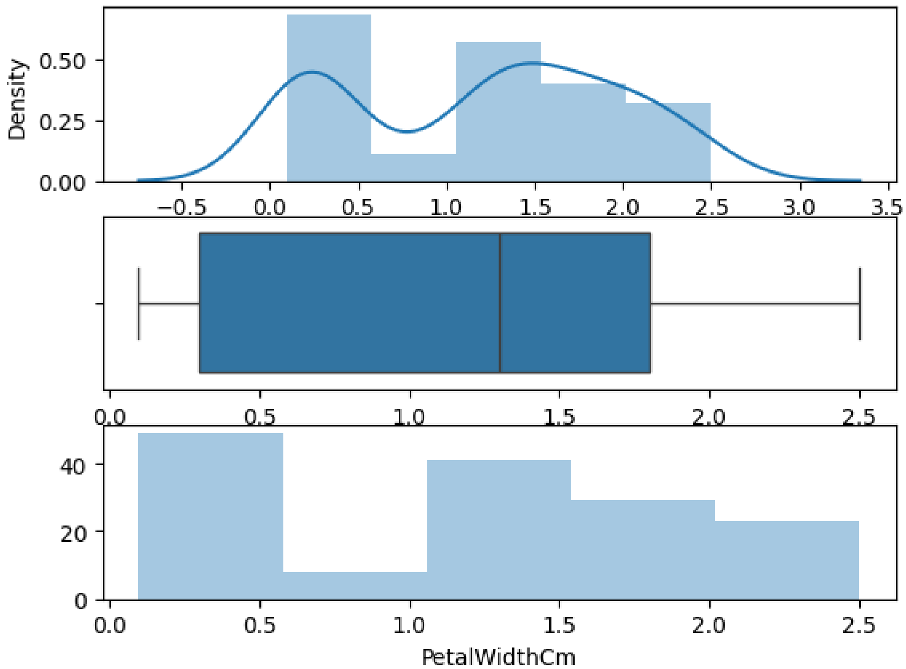
In [18]:
```
def plot_graph(df,col):
  fig, (ax1,ax2,ax3)= plt.subplots(3,1)
  sns.distplot(df[col],ax=ax1)
  sns.boxplot(df[col],ax=ax2,orient='h')
  sns.distplot(df[col],ax=ax3,kde=False)
```

In [19]:
```
print(plot_graph(df,'SepalLengthCm'))
print(plot_graph(df,'SepalWidthCm'))
print(plot_graph(df,'PetalLengthCm'))
print(plot_graph(df,'PetalWidthCm'))
```




Density
1.0
0.5
0.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
20
0
SepalWidthCm


Density
0.2
0.1
0.0
0
2
4
6
8
1
3
5
7
40
20
PetalLengthCm



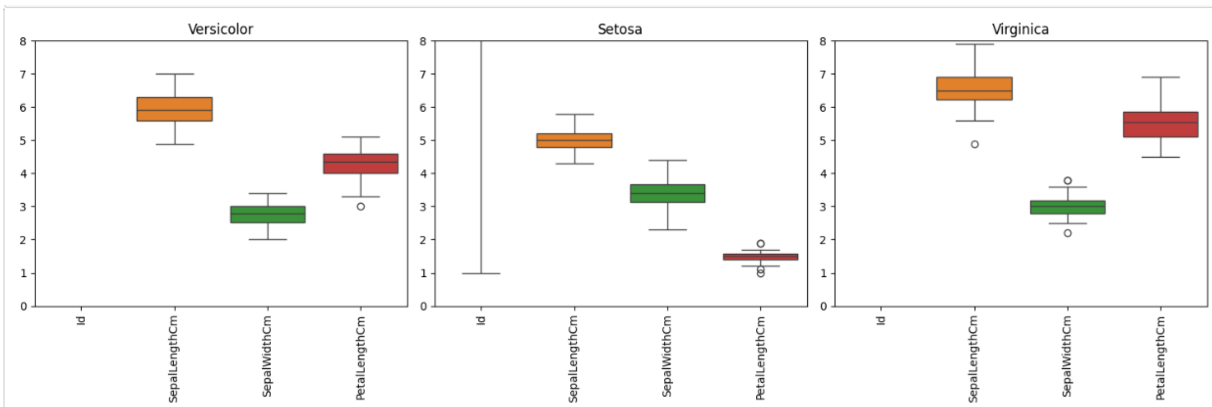
Description:

- The highest frequency of the sepal length is between 30 and 35 which is between 5.5 & 6.
- The highest frequency of the sepal Width is around 70 which is between 3.0 & 3.5.
- The highest frequency of the petal length is around 50 which is between 1 & 2.
- The highest frequency of the petal width is between 40 and 50 which is between 0.0 & 0.5.

3. Box Plots

We can use boxplots to see how the categorical value of distributed with other numerical values.

```
In [20]: irisVer = df[df['Species'] == "versicolor"]
         irisSet = df[df['Species'] == "setosa"]
         irisVir = df[df['Species'] == "virginica"]

         # Set up subplots
         fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

         # Boxplot for Versicolor
         sns.boxplot(data=irisVer.iloc[:, :4], ax=axes[0], )
         axes[0].set_title('Versicolor')
         axes[0].tick_params(axis='x', rotation=90)  # Rotate x-axis labels
         axes[0].set_ylim(0, 8)

         # Boxplot for Setosa
         sns.boxplot(data=irisSet.iloc[:, :4], ax=axes[1])
         axes[1].set_title('Setosa')
         axes[1].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

         axes[1].set_ylim(0, 8)

         # Boxplot for Virginica
         sns.boxplot(data=irisVir.iloc[:, :4], ax=axes[2])
         axes[2].set_title('Virginica')
         axes[2].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

         axes[2].set_ylim(0, 8)

         # Adjust layout
         plt.tight_layout()
         plt.show()
```



Description:

- ❖ Species Setosa has the smallest features and less distributed with some outliers.
- ❖ Species Versicolor has the average features.
- ❖ Species Virginica has the highest features.

4. Handling Correlation

Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

Handling Correlation

In [21]:
```
df.drop('Id',axis=1,inplace=True)
```

In [22]:
```
df.head()
```

Out[22]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

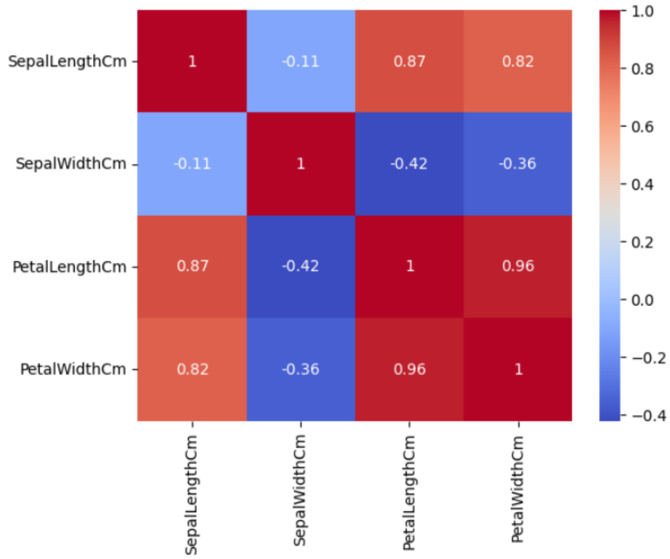
In [23]:
```
numeric_columns = df.select_dtypes(include=[float, int]).columns
df_numeric = df[numeric_columns]

# Calculate correlation
corr = df_numeric.corr(method='pearson')
print(corr)
```

```
               SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
SepalLengthCm       1.000000     -0.109369       0.871754      0.817954
SepalWidthCm       -0.109369      1.000000      -0.420516     -0.356544
PetalLengthCm       0.871754     -0.420516       1.000000      0.962757
PetalWidthCm        0.817954     -0.356544       0.962757      1.000000
```

5. Heatmaps

The heatmap is a data visualization technique that is used to analyze the dataset as colors in two dimensions. Basically, it shows a correlation between all numerical variables in the dataset. In simpler terms, we can plot the above-found correlation using the heatmaps.

In [24]:
```
sns.heatmap(df_numeric.corr(method='pearson'),
            annot = True,cmap='coolwarm');

plt.show()
```



Description:

- Petal width and petal length have high correlations.
- Petal length and sepal width have good correlations.
- Petal Width and Sepal length have good correlations.

FEATURE ENGINEERING

1. Handling Outliers

An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal)objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda's dataframe.

```
In [25]: def outliers_handling_function(df,col):
             #IQR Approach(Interquartile Range):
             q1=df[col].quantile(0.25)
             q3=df[col].quantile(0.75)
             iqr=q3-q1
             maxi=q3+(1.5*iqr)
             mini=q1-(1.5*iqr)
             print("Maxi: ",maxi)
             print("Mini: ",mini)
             print("Lenght of outliers: ",len([c for c in df[col] if c>=maxi or c<=mini]))
```

If Features Are Skewed We Use the below Technique which is IQR

Data which are greater than IQR +1.5 IQR and data which are below than IQR - 1.5 IQR are my outliers

where , IQR = 75th%ile data - 25th%ile data

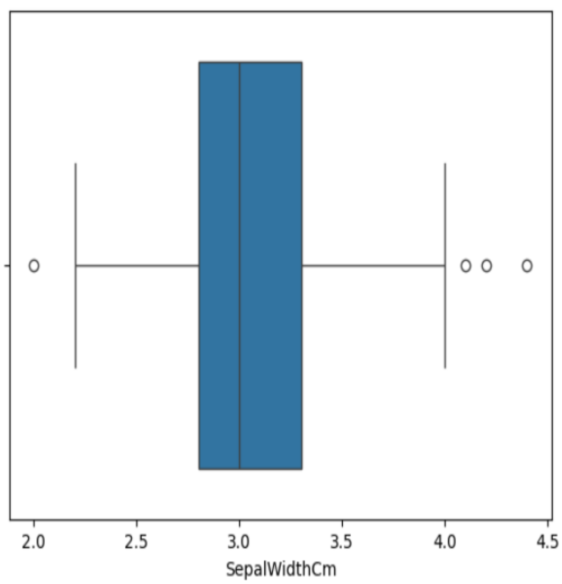
& IQR +- 1.5 IQR will be changed depending upon the domain ie it could be sometimes IQR +- 3IQR

In [27]:
```
print("Sepal_length")
outliers_handling_function(df,'SepalLengthCm')
print("\nSepal_Width")
outliers_handling_function(df,'SepalWidthCm')
print("\nPetal_length")
outliers_handling_function(df,'PetalLengthCm')
print("\nPetal_Width")
outliers_handling_function(df,'PetalWidthCm')
```

```
Sepal_length
Maxi:  8.350000000000001
Mini:  3.1499999999999986
Lenght of outliers:  0

Sepal_Width
Maxi:  4.05
Mini:  2.05
Lenght of outliers:  4

Petal_length
Maxi:  10.349999999999998
Mini:  -3.649999999999999
Lenght of outliers:  0

Petal_Width
Maxi:  4.05
Mini:  -1.95
Lenght of outliers:  0
```

In [28]:
```
sns.boxplot(x='SepalWidthCm', data=df)
```

Out[28]:
```
<Axes: xlabel='SepalWidthCm'>
```



Hence, There are 4 outliers in the SepalWidthCm attribute.

We will replace the outliers with its median of the SepalWidthCm attribute.

2. Updating Outliers

In [29]:
```
df["SepalWidthCm"].median()
```
Out[29]:
```
3.0
```

In [30]:
```
df['SepalWidthCm']=np.where(df['SepalWidthCm']>=4.05,df['SepalWidthCm'].median(),df['SepalWidthCm'])
df['SepalWidthCm']=np.where(df['SepalWidthCm']<=2.05,df['SepalWidthCm'].median(),df['SepalWidthCm'])
```

In [31]:
```
print("Sepal_Width")
outliers_handling_function(df,'SepalWidthCm')
```
```
Sepal_Width
Maxi:  4.05
Mini:  2.05
Lenght of outliers:  0
```

In [32]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

3. Encoding

Encoding

In [34]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [35]:
```
df.describe()
```
Out[35]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.036000	3.758667	1.198667
std	0.828066	0.389851	1.764420	0.763161
min	4.300000	2.200000	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	4.000000	6.900000	2.500000

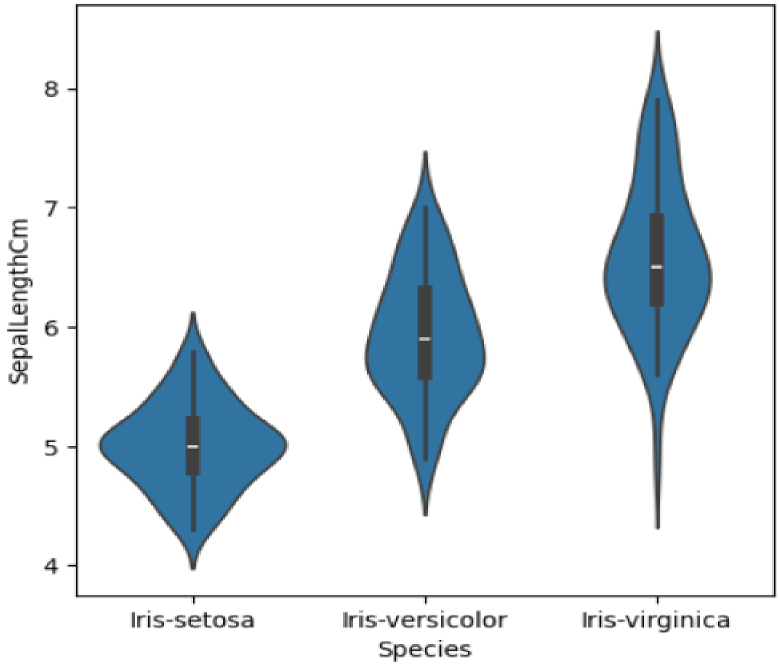
In [36]:
```
df['Species']=le.fit_transform(df['Species'])
df.head(3)
```
Out[36]:

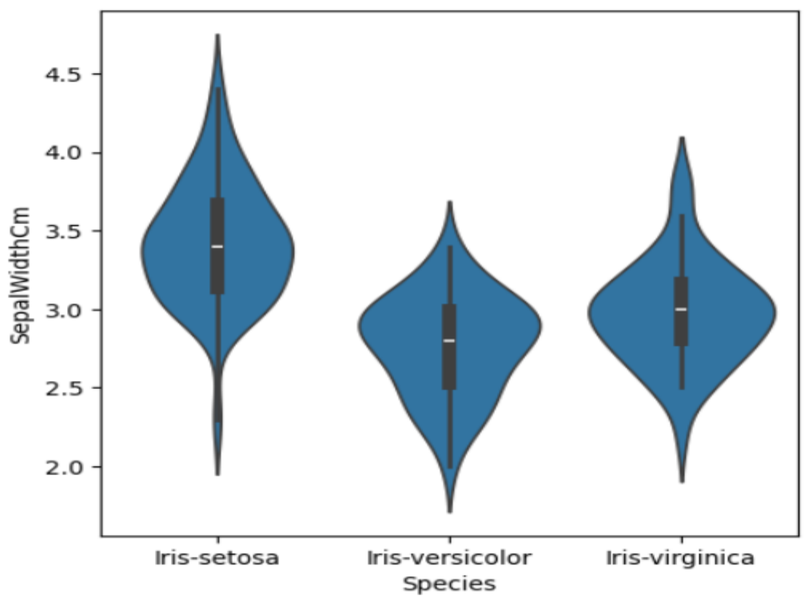
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

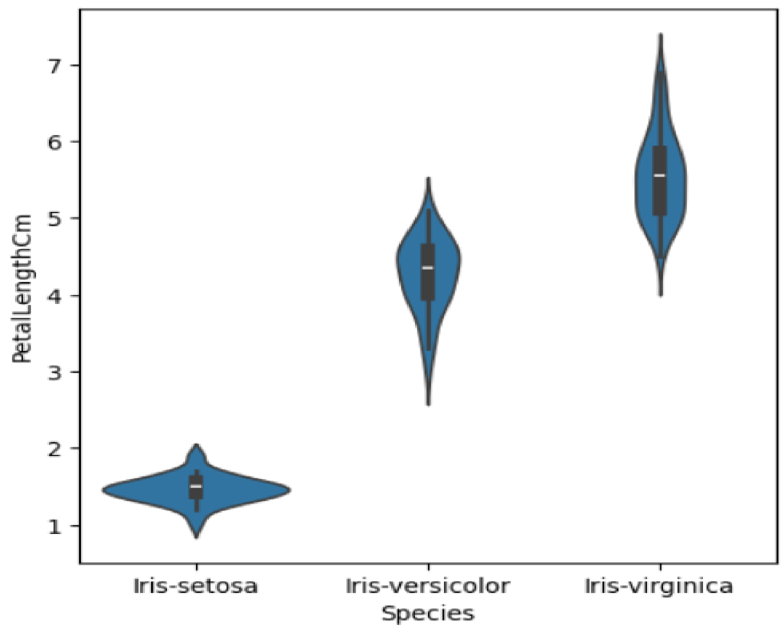
4. Feature Extraction

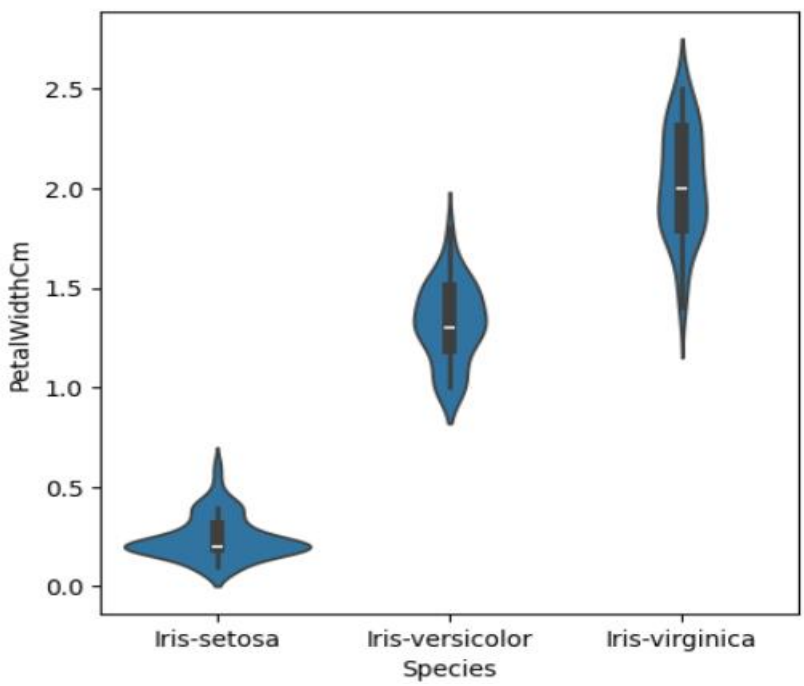
In [37]:
```
def get_plot(data,feature):
    plt.figure(figsize=(5,5))
    sns.violinplot(x='Species' , y=feature , data=data )

    plt.show()
```

In [38]:
```
features=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']
for feature in features:
    print(get_plot(data,feature))
```









Observation:

- Feature **'SepalLengthCm'** is not an interesting feature up to some extent in determining the Species.
- Feature **'SepalWidthCm'** is not an interesting feature up to some extent in determining the Species.
- Feature **'PetalLengthCm'** is an interesting feature up to some extent in determining the Species.
- Feature **'PetalWidthCm'** is an interesting feature up to some extent in determining the Species.

BUILDING THE ML MODEL

Division of data Set into two sets i.e. Training Data Set and Test Data Set with the ration of 75% and 25% respectively.

Build Manchine Learning Model

split dataset into train & test

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
X=df.drop(columns=['Species'])
Y=df['Species']
X.columns
```

Out[40]:
```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'], dtype='object')
```

In [41]:
```
X_new=df[['PetalLengthCm','PetalWidthCm']]
X_new
```

Out[41]:

	PetalLengthCm	PetalWidthCm
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...	...	...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

150 rows × 2 columns

In [42]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X_new,Y,test_size=0.25)
```

In [43]:
```
X_new.shape
```

Out[43]: (150, 2)

In [44]:
```
Y.shape
```

Out[44]: (150,)

In [45]:
```
Y.unique()
```

Out[45]: array([0, 1, 2])

Applying Logistic Regression:

- Logistic Regression is a statistical technique widely employed for binary classification problems, where the objective is to predict the probability of an instance belonging to one of two classes.
- It utilizes the sigmoid (logistic) function to map a linear combination of input features and weights to a value between 0 and 1. This function enables the algorithm to produce probabilities and establish a decision boundary, effectively separating the input space into regions corresponding to the two classes.
- Logistic Regression estimates its parameters through maximum likelihood estimation, maximizing the likelihood function to optimize model performance. Regularization can be incorporated to prevent overfitting.
- Despite its effectiveness, logistic regression may be less suitable for intricate relationships within the data, prompting consideration of more sophisticated algorithms in such scenarios.

1. Logistic Regression

```
In [47]: model_L.fit(X_train,Y_train)
Out[47]:
         LogisticRegression
LogisticRegression(multi_class='multinomial')

In [48]: y_pred_L=model_L.predict(X_test)

In [49]: y_pred_L
Out[49]: array([0, 2, 2, 2, 0, 0, 0, 1, 2, 2, 1, 0, 2, 0, 1, 1, 1, 0, 1, 0, 1, 1,
                0, 1, 1, 1, 0, 1, 1, 2, 0, 0, 1, 1, 2, 0, 1, 2])
```

```
In [53]: print(classification_report(Y_test,y_pred_L))
         print("Accuracy: ",model_L.score(X_test,Y_test)*100)

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.81      1.00      0.90        13
           2       1.00      0.75      0.86        12

    accuracy                           0.92        38
   macro avg       0.94      0.92      0.92        38
weighted avg       0.94      0.92      0.92        38

Accuracy:  92.10526315789474
```

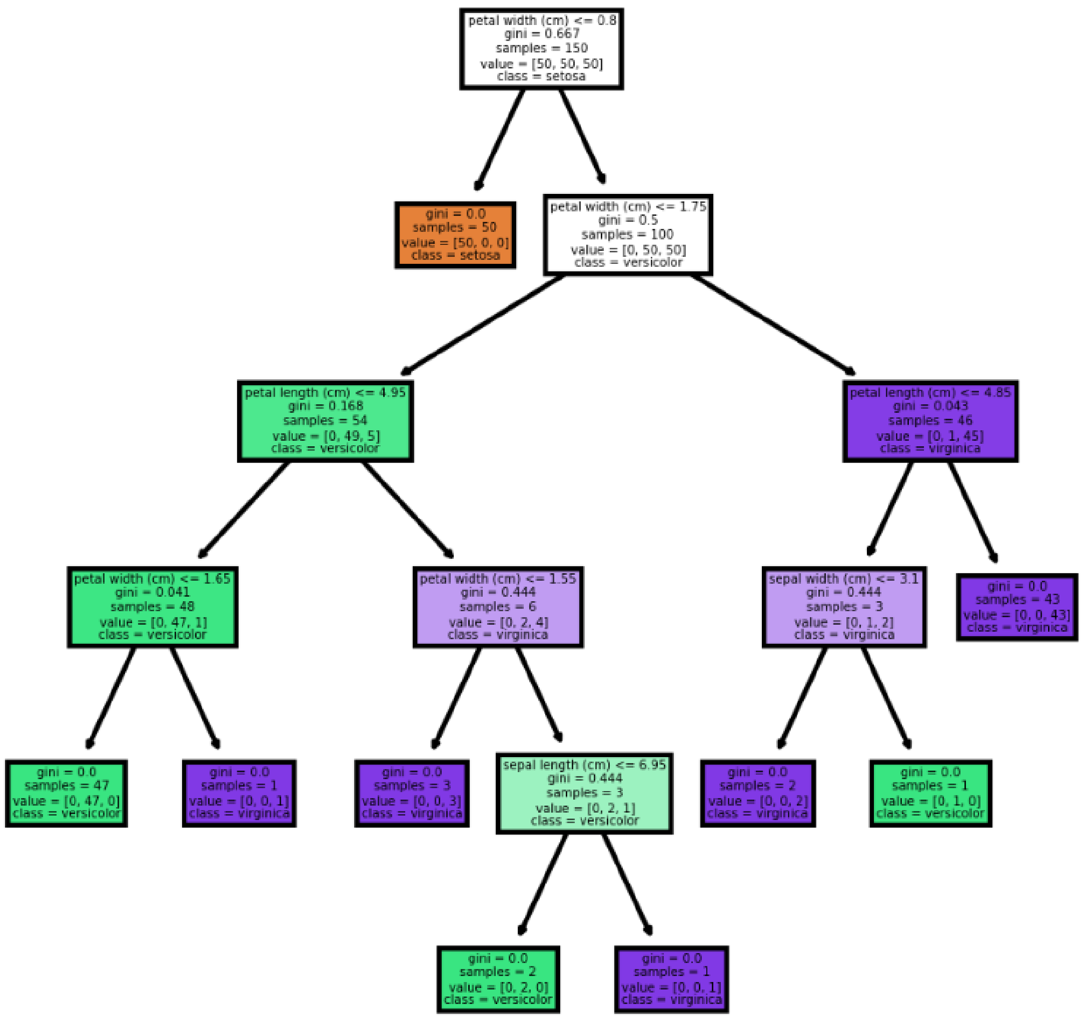
Applying Decision Tree:

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed based on features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, like a tree, it starts with the root node, which expands on further - branches and constructs a tree-like.

2. Decision Tree Classifier

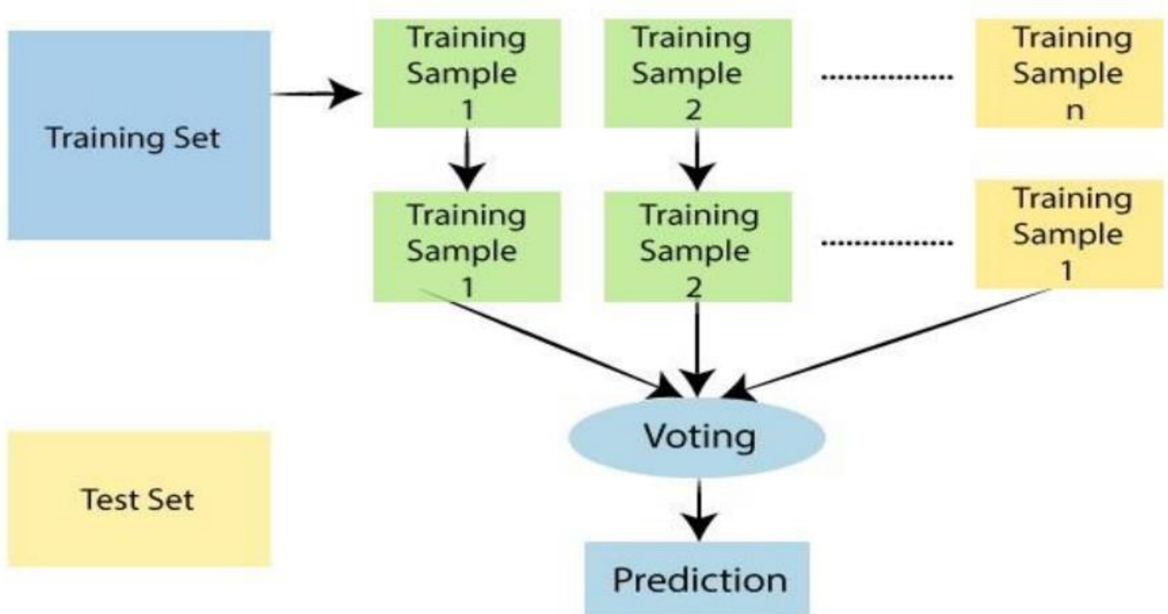
```
In [54]: model_D.fit(X_train,Y_train)

Out[54]:
         DecisionTreeClassifier
DecisionTreeClassifier(max_depth=8)

In [55]: y_pred_D=model_D.predict(X_test)

In [56]: print(classification_report(Y_test,y_pred_D))
         print("Accuracy: ",model_D.score(X_test,Y_test)*100)

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.93      1.00      0.96        13
           2       1.00      0.92      0.96        13

    accuracy                           0.97        38
   macro avg       0.98      0.97      0.97        38
weighted avg       0.98      0.97      0.97        38

Accuracy:  97.36842105263158
```

In [57]:
```python
from sklearn import tree

# Assuming model_D is your decision tree model and X, Y are your feature matrix and target variable
model_D.fit(X, Y)

fn=['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']
cn=['setosa', 'versicolor', 'virginica']

# Setting dpi = 300 to make image clearer than default
fig1, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)

tree.plot_tree(model_D,
           feature_names = fn,
           class_names=cn,
           filled = True);

# You can save your plot if you want
fig1.savefig('imagename.png')
```



Applying Random Forest Regression:

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML.
- It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.
- As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.
- The below diagram explains the working of the Random Forest algorithm:



3. RandomForestRegressor

```
In [58]: model_R.fit(X_train,Y_train)

Out[58]: ▾ RandomForestRegressor
         RandomForestRegressor()

In [59]: y_pred_R=model_R.predict(X_test)

In [60]: print("Accuracy: ",model_R.score(X_test,Y_test)*100)

         Accuracy:  91.84589786661962
```

CONCLUSION:

ML Models	Accuracy
Logistic Regression	92.10
Decision Tree	97.368
Random Forest Regression	91.84

AUTOMATE THE ML PIPELINE

Automating the machine learning (ML) model pipeline is a crucial aspect of streamlining and optimizing the entire process of developing, deploying, and maintaining ML models.

- **Data Preprocessing:**

  Automating data preprocessing involves tasks such as cleaning, transforming, and scaling data. Automation ensures that these steps are consistently applied, reducing the risk of errors and saving time.

- **Model Evaluation:**

  Automated model evaluation involves assessing performance metrics and comparing different models. This process helps select the best-performing model for deployment.

- **Model Deployment:**

  Automating model deployment ensures a seamless transition from development to production. Containerization tools like Docker, along with orchestration tools like Kubernetes, can be employed to deploy and manage ML models efficiently.

- **Monitoring and Maintenance:**

  Automation is crucial for continuous monitoring of deployed models. It helps track model performance over time, detect anomalies, and trigger retraining when necessary. This ensures models stay accurate and relevant.

- **Version Control and Reproducibility:**

  Automation tools support version control for both code and data, ensuring reproducibility. This is essential for tracking changes, reproducing experiments, and maintaining a reliable and auditable ML pipeline.

```python
In [62]: def predict(ml_model):
             model = ml_model.fit(X_train , Y_train)
             print('Training score : {}'.format(model.score(X_train , Y_train)))
             y_predection = model.predict(X_test)
             print('predictions are : {}'.format(y_predection))
             print('\n')
             r2_score = metrics.r2_score(Y_test , y_predection)
             print('r2 score : {}'.format(r2_score))
             print('MAE : {}'.format(metrics.mean_absolute_error(Y_test , y_predection)))
             print('MSE : {}'.format(metrics.mean_squared_error(Y_test , y_predection)))
             print('RMSE : {}'.format(np.sqrt(metrics.mean_squared_error(Y_test , y_predection))))
             sns.distplot(Y_test - y_predection)
```

- **r2_score**: R-Squared (R^2 or the coefficient of determination) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. In other words, r-squared shows how well the data fit the regression model (the goodness of fit).

- **MAE**: In statistics, mean absolute error (MAE) is a measure of errors between paired observations expressing the same phenomenon.

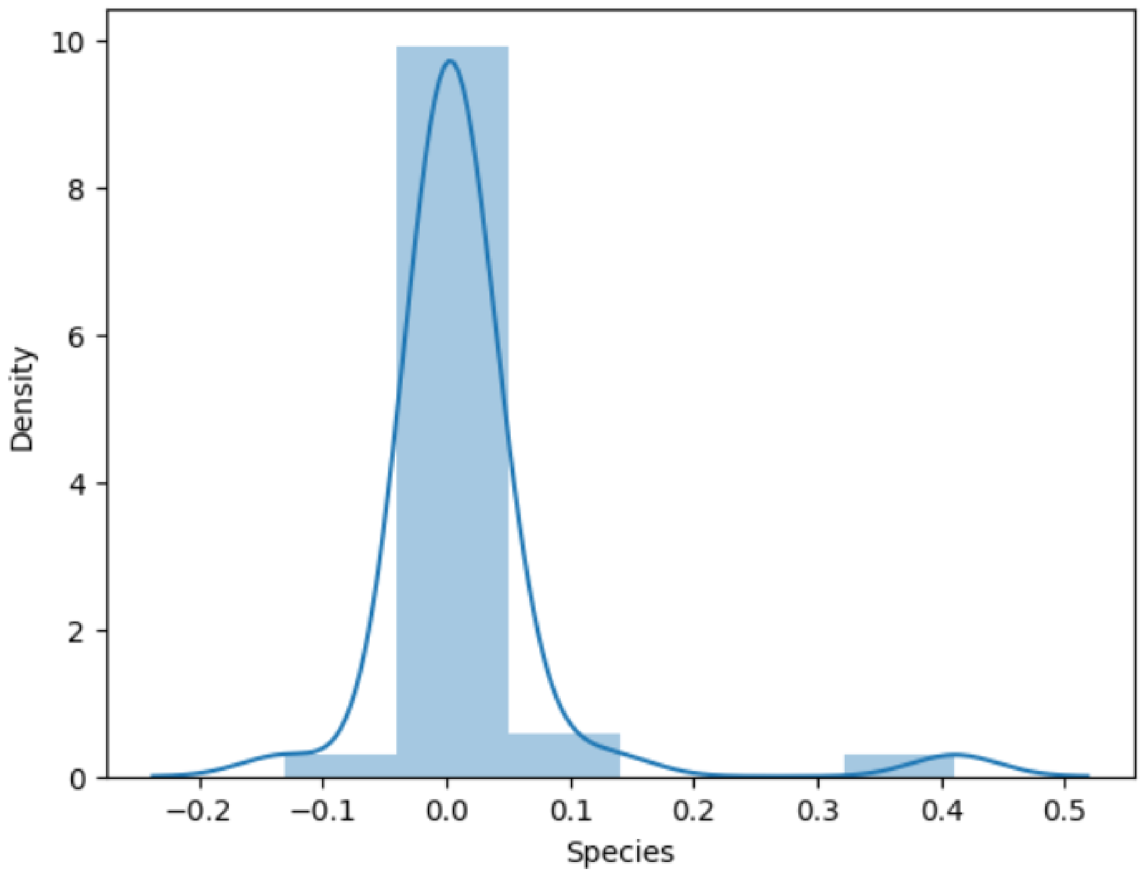
- **MSE**: The mean squared error (MSE) is one of many metrics you could use to measure your model's performance. You calculate the MSE by finding the errors, squaring them, and taking the mean.

- **RMSE**: The Root Mean Squared Error (RMSE) is one of the two main performance indicators for a regression model. It measures the average difference between values predicted by a model and the actual values. It provides an estimation of how well the model can predict the target value (accuracy).

Random-Forest-Regressor

In [63]:
```
predict(RandomForestRegressor())
```

```
Training score : 0.9860891687634772
predictions are : [1.13       0.         2.         0.         0.         0.
 1.         1.58866667 1.         1.94       1.86833333 1.
 2.         2.         1.         1.9675     1.         1.9675
 2.         1.         1.9675     0.         2.         0.
 0.         0.         0.         0.         1.         2.
 1.         1.         0.         1.9675     1.         1.
 1.         0.        ]


r2 score : 0.9915408467392577
MAE : 0.02271052631578948
MSE : 0.005559374269005854
RMSE : 0.07456121155806049
```



Decision-Tree-Regressor

In [65]:
```
predict(DecisionTreeRegressor())
```

```
Training score : 0.9933325395880462
predictions are : [1.  0.  2.  0.  0.  0.  1.  1.5 1.  2.  2.  1.  2.  2.  1.  2.  1.  2.
 2.  1.  2.  0.  2.  0.  0.  0.  0.  0.  1.  2.  1.  1.  0.  2.  1.  1.
 1.  0. ]


r2 score : 0.9899894625922023
MAE : 0.013157894736842105
MSE : 0.006578947368421052
RMSE : 0.08111071056538127
```