Iris Dataset Analysis using Python | Classification | Machine Learning

we are going to analyze the tabular data with various visualizations and build a robust machine learning model to predict the class of the flower.

Dataset Information

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Attribute Information:-

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. species

- Iris Setosa
- Iris Versicolour
- Iris Virginica

Download the Iris Dataset here

Import modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

- pandas - used to perform data manipulation and analysis
- numpy - used to perform a wide variety of mathematical operations on arrays
- matplotlib - used for data visualization and graphical plotting
- seaborn - built on top of matplotlib with similar functionalities
- warnings - to manipulate warnings details

filterwarnings('ignore') is to ignore the warnings thrown by the modules (gives clean results)

Loading the Dataset

```
# load the csv data
df = pd.read_csv('Iris.csv')
df.head()
```

	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

```
# to display stats about data
df.describe()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

Preprocessing the Dataset

Let's check for NULL values in the dataset

```
# check for null values
df.isnull().sum()
```

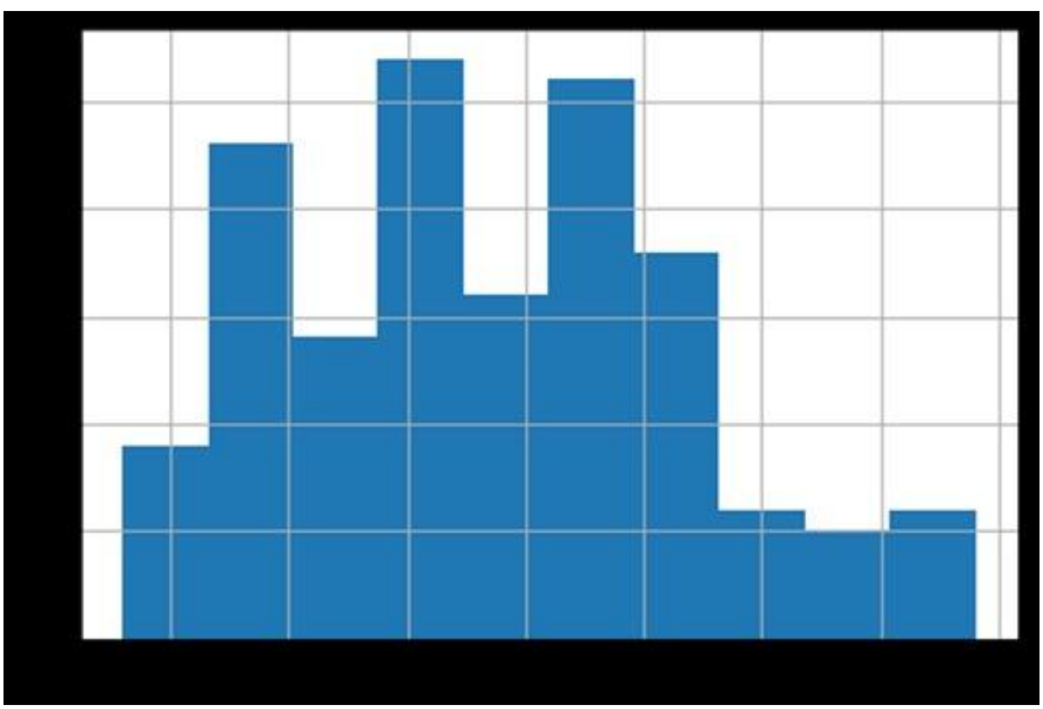
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

- There are no NULL values present in the dataset.
- If any NULL values are present, we have to fill all the NULL values before proceeding to model training.

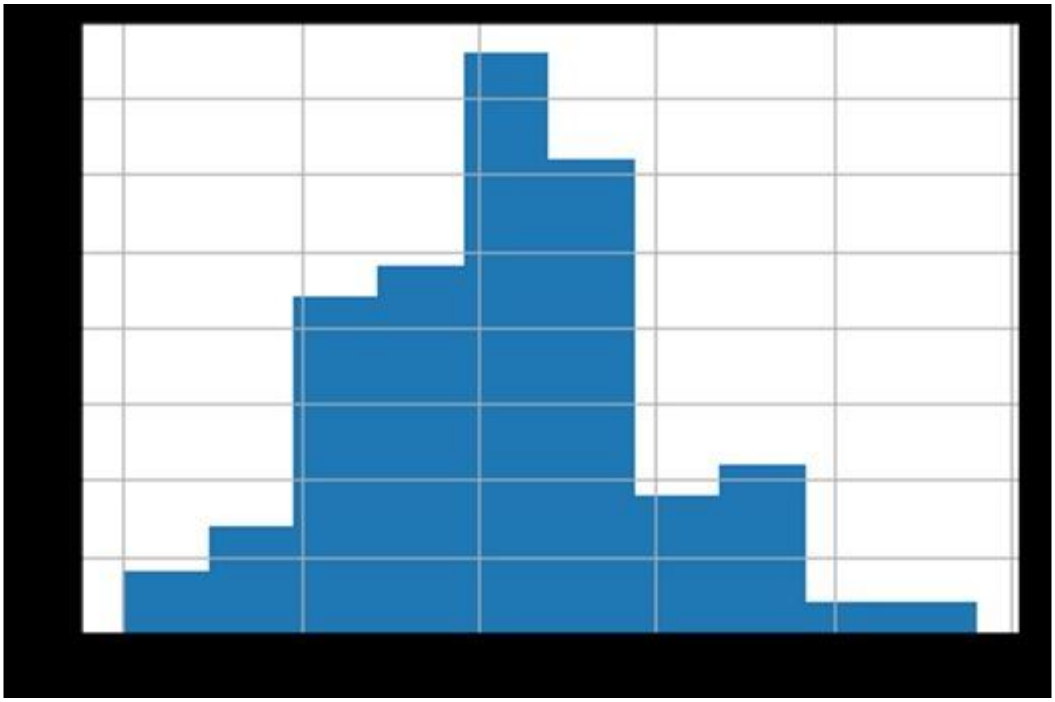
Exploratory Data Analysis

In Exploratory Data Analysis(EDA), we will visualize the data with different kinds of plots for inference. It is helpful to find some patterns (or) relations within the data

```
# histograms
df['SepalLengthCm'].hist()
```



```
df['SepalWidthCm'].hist()
```

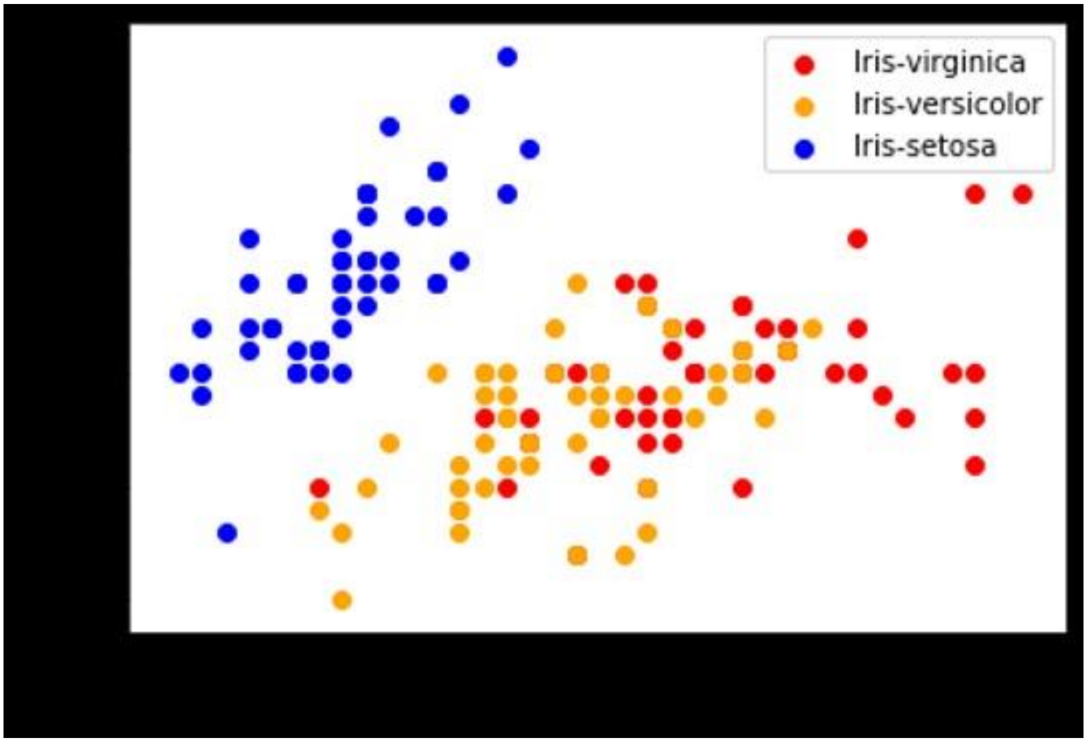


Let's create some scatter plots for inference

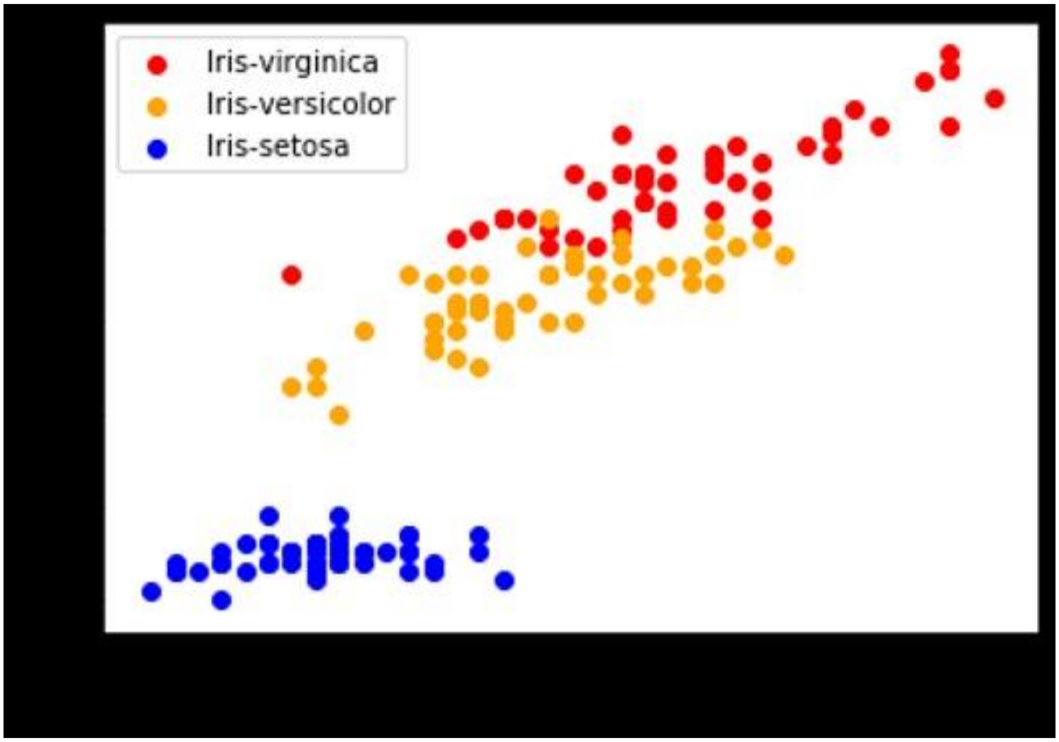
```
# create list of colors and class labels
colors = ['red', 'orange', 'blue']
species = ['Iris-virginica', 'Iris-versicolor', 'Iris-setosa']
```

- df[df['Species'] == species[i]] - filters samples for each class label
- plt.scatter() - generates a scatterplot for the data
- plt.xlabel() - label for x-axis
- plt.ylabel() - label for y-axis
- plt.legend() - display the legend for the plot

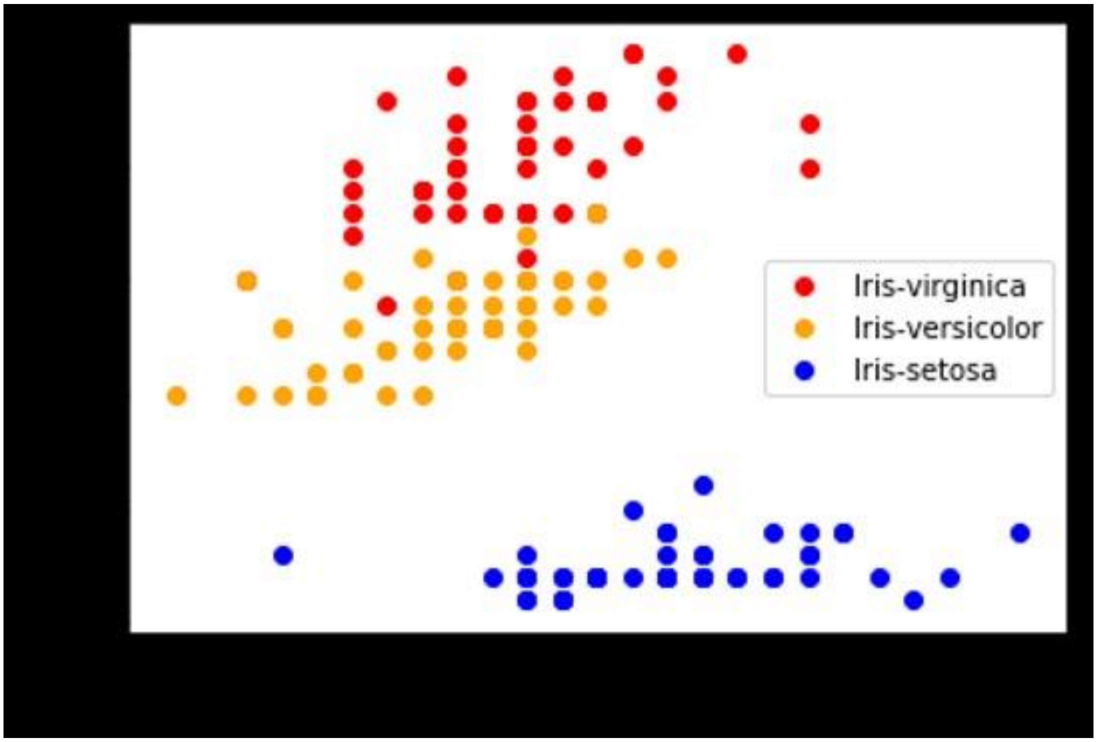
```
for i in range(3):
    # filter data on each class
    x = df[df['Species'] == species[i]]
    # plot the scatter plot
    plt.scatter(x['SepalLengthCm'], x['SepalWidthCm'], c = colors[i],
label=species[i])
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend()
```




```
for i in range(3):
    # filter data on each class
    x = df[df['Species'] == species[i]]
    # plot the scatter plot
    plt.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c = colors[i], 
label=species[i])
plt.xlabel("Sepal Length")
plt.ylabel("Petal Length")
plt.legend()
```




```
for i in range(3):
    # filter data on each class
    x = df[df['Species'] == species[i]]
    # plot the scatter plot
    plt.scatter(x['SepalWidthCm'], x['PetalWidthCm'], c = colors[i],
label=species[i])
plt.xlabel("Sepal Width")
plt.ylabel("Petal Width")
plt.legend()
```



- Here we can see, iris-setosa is easily separable from the other 2 classes
- In petal length and petal width plot, the classes plotted without overlapping
- In other plots, some samples are overlapping with other classes

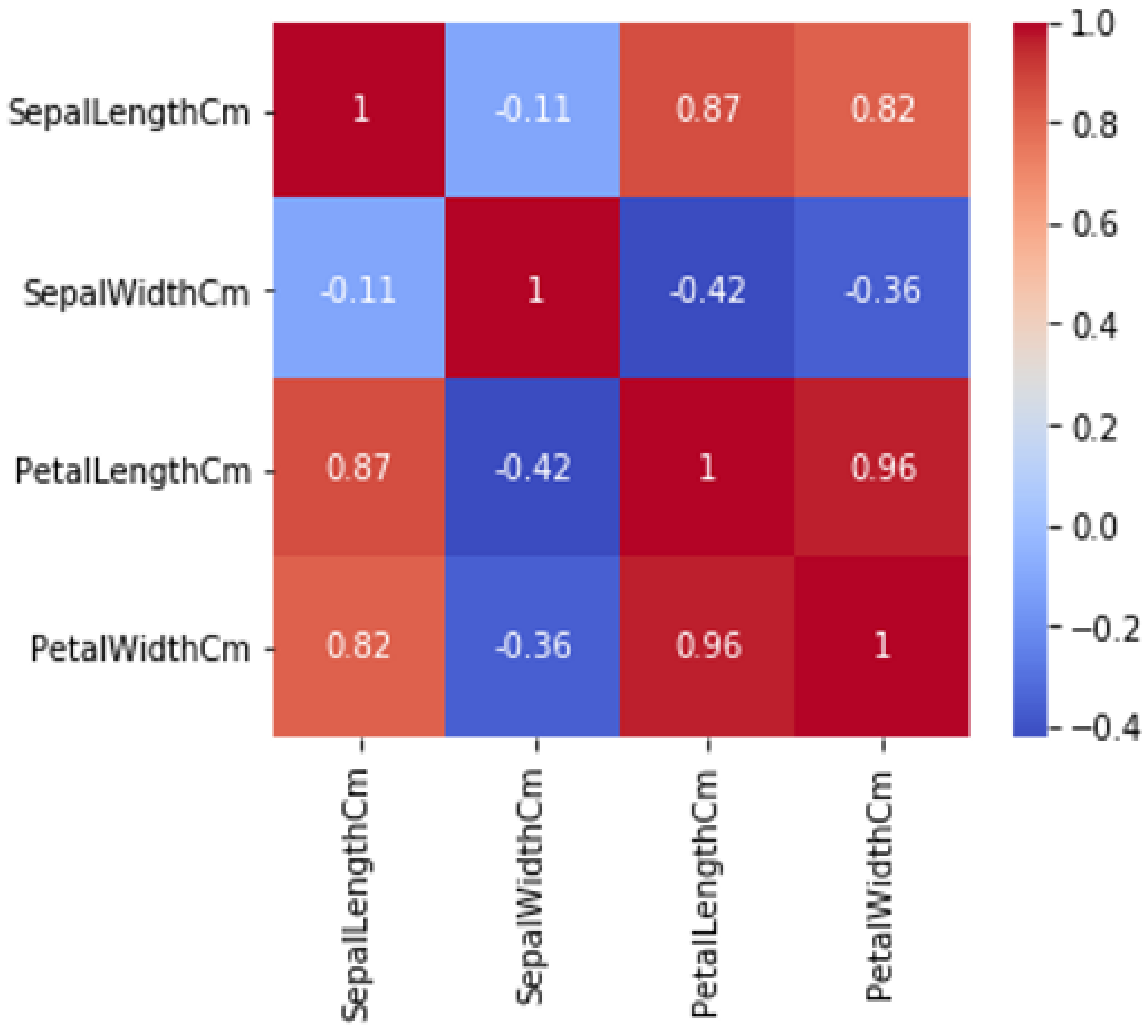
Correlation Matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The value is in the range of -1 to 1. If two variables have high correlation, we can neglect one variable from those two.

```
# display the correlation matrix
df.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

```
corr = df.corr()
# plot the heat map
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(corr, annot=True, ax=ax, cmap = 'coolwarm')
```



- Petal length and petal width have high positive correlation of 0.96
- If petal length value increases, petal width also increases
- Sepal length have high positive correlation with petal length and petal width

- Sepal width have negative correlation with petal length and petal width

Model Training and Testing

Now the preprocessing has been done, let's perform the model training and testing

```
from sklearn.model_selection import train_test_split
## train - 70%
## test - 30%

# input data
X = df.drop(columns=['Species'])
# output data
Y = df['Species']
# split the data for train and test
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)
```

- **X** - contains input attributes
- **Y** - contains the output attribute
- **train_test_split()** - splits the data for training and testing (here we are splitting 70% data for training and 30% for testing)

Let's import some models and train

```
# logistic regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
# model training
model.fit(x_train, y_train)
```

- **fit()** - used for training the model with the data

```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

Accuracy: 91.11111111111111

- **model.score()** - gives the accuracy for the test data

```
# knn - k-nearest neighbours
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(x_train, y_train)
```

```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

Accuracy: 100.0

```
# decision tree
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(x_train, y_train)
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

Accuracy: 91.11111111111111

Final Thoughts

- We have got around 100% accuracy for KNN with our test data split
- You can also try out various machine learning models similar to above
- More EDA can be done with boxplots, violinplot, barplot, etc.,