

# Iris Classification - rubangino.in

## Overview

This notebook has Iris Classification project, which includes the development of four different machine learning models of Logistic Regression, K-nearest classifier, Support Vector Classification, and Naive Bayes Classification to classify the Iris flowers into different species. This project involves the selection and training of a model, testing its performance, and deploying it with a Streamlit front-end. Among the four models, Logistic Regression was chosen to predict the Iris Dataset. The trained model is saved as a `model.pkl` file for an easy integration into the application.

Github repository link: [https://github.com/Ruban2205/Iris\\_Classification](https://github.com/Ruban2205/Iris_Classification)  
([https://github.com/Ruban2205/Iris\\_Classification](https://github.com/Ruban2205/Iris_Classification))

Star 🌟 this repository for Future use 😊

## Importing the Libraries

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr

import warnings
warnings.filterwarnings('ignore')
```

## Loading the Dataset

```
In [3]: data_frame = pd.read_csv("Dataset/Iris.csv")
data_frame
```

```
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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

## Data preprocessing

```
In [4]: # To check the Statistics of Data
data_frame.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

```
In [5]: # To check the DataTypes

data_frame.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
```

```
In [6]: # To Check if the dataset has null values.

# If more than 15% of null value is present in the particular column. DROP the Column.
# If it is less than 15% use Imputations techniques Like Mean, Median and Mode.

data_frame.isna().sum()
```

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

```
In [7]: # Check if the Duplicate rows exists.
# If any duplicated row is found. Then remove the entire row.

data_frame.duplicated().sum()
```

```
Out[7]: 0
```

```
In [8]: # Drop the ID Column since, We don't need the column.
# Use Normalisation methods - StandardScalar, MinmaxScalar

data_frame.drop(columns=["Id"], inplace=True)
data_frame.head()
```

```
Out[8]:
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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

```
In [9]: # Scaling the Features.

# In this dataset the Inputs are Sepal Length, Sepal Width, Petal Length, and Petal Width
# The Outputs = Species

# X => Inputs -> SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm
# Y => Output -> Species

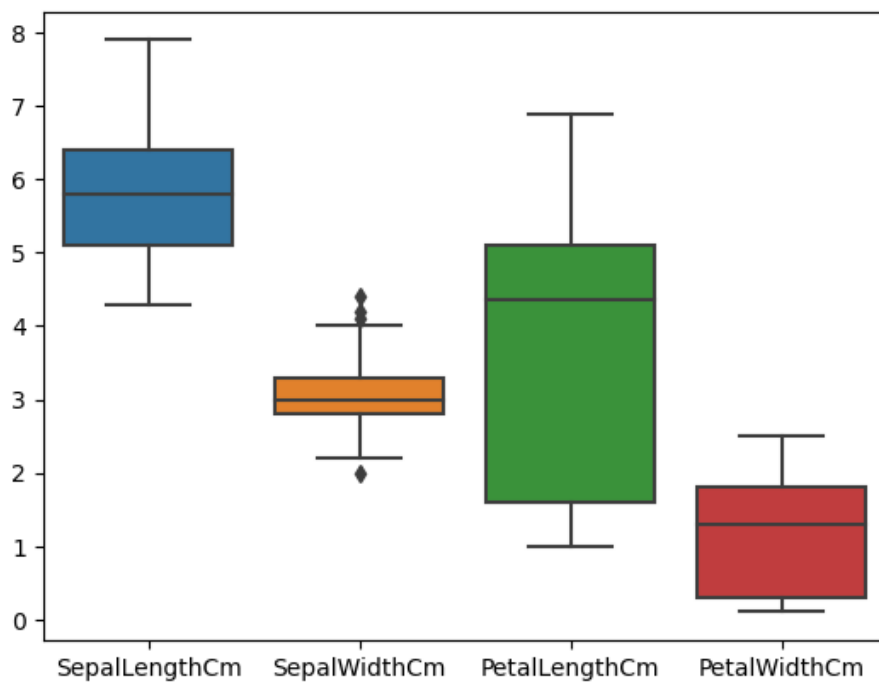
features = ["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]
x = data_frame[features].values #Input
y = data_frame["Species"].values #Output
```

## Box plot 📊

```
In [10]: # Plotting the data to find outliers

sns.boxplot(data=data_frame)
```

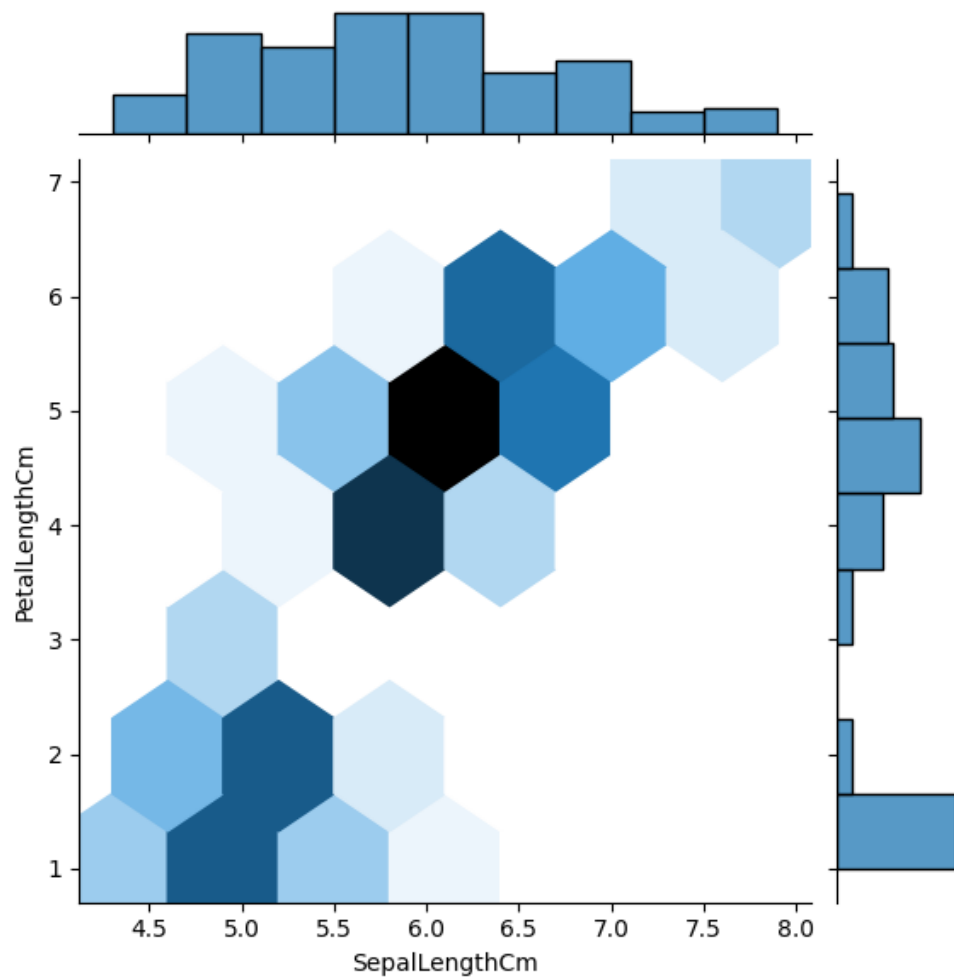
```
Out[10]: <AxesSubplot:>
```



## Joint Plot

```
In [11]: sns.jointplot(x=data_frame['SepalLengthCm'], y=data_frame['PetalLengthCm'], data=data_frame,
```

```
Out[11]: <seaborn.axisgrid.JointGrid at 0x1aaacb1dcc8>
```

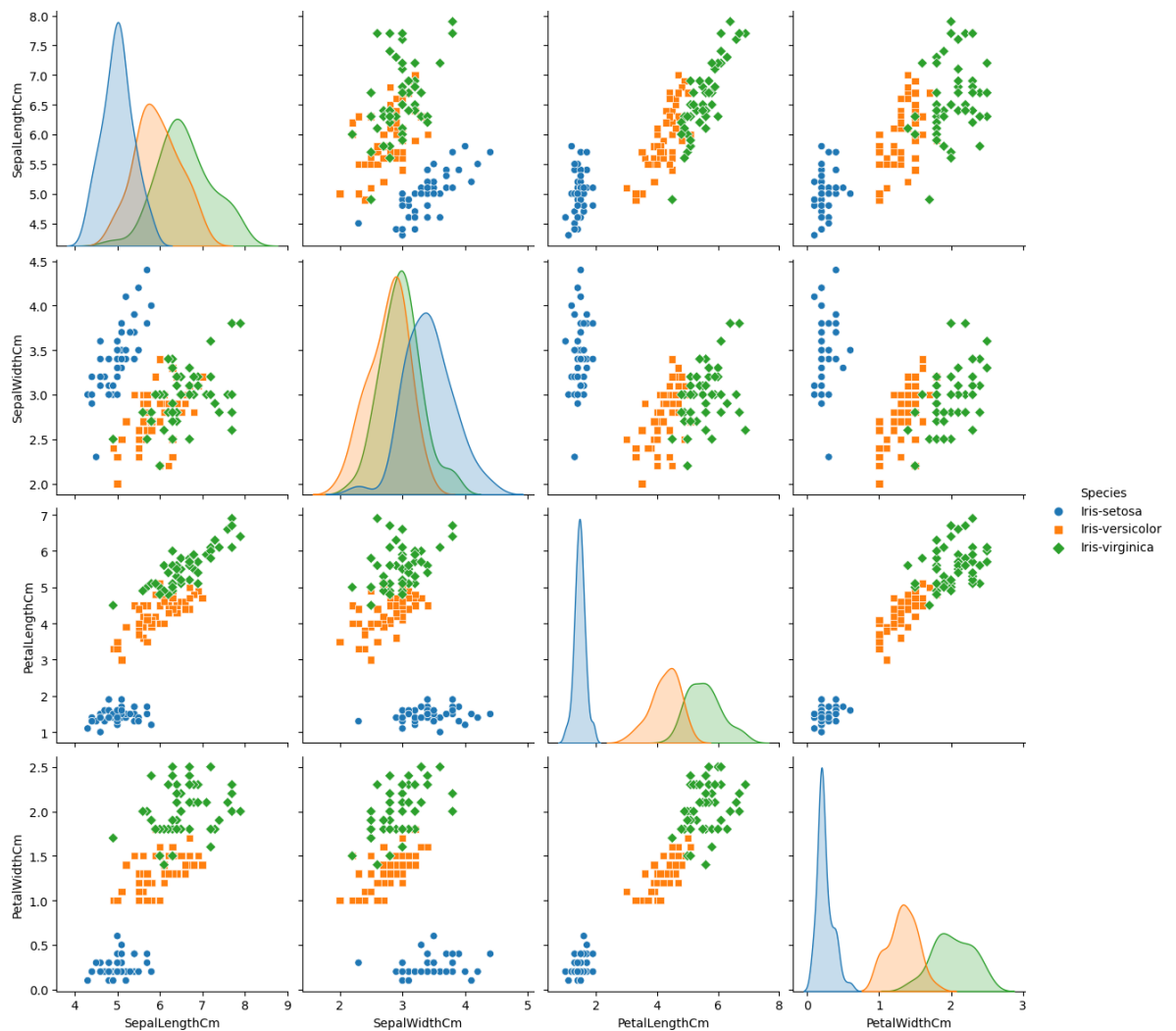


## Pair plot

In [12]: *# Pair plot - to find the outliers*

```
sns.pairplot(data_frame, hue="Species", size=3, markers=["o", "s", "D"])
```

Out[12]: <seaborn.axisgrid.PairGrid at 0x1aaaca869c8>



## Feature Scaling

In [13]: *# Separate the data set into Train and Test*

```
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, random_state=0)
```

## Label Encoder

In [14]: *# Label Encoder*  
*# A technique is used in Machine Learning to convert categorical variables into numerical fo*

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(y)
```

Out[14]: LabelEncoder()

```
In [15]: # dir(le)
```

```
le.classes_
```

```
Out[15]: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
In [16]: # Use mapping method to add the index to decode the output.
```

```
mapping = dict(zip(le.classes_, range(len(le.classes_))))  
mapping
```

```
Out[16]: {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
```

```
In [17]: y = le.transform(y)
```

## Standard Scaler

```
In [18]: # Standard Scaler - To make all attribute values in a similar range (Normalization)  
# Removes the mean and scales each feature/variable to unit variance.
```

```
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
scaler.fit(X_train)
```

```
Out[18]: StandardScaler()
```

```
In [19]: X_train_scaled = scaler.transform(X_train)  
X_test_scaled = scaler.transform(X_test)
```

## Model Training

```
In [20]: # Model Training Libraries
```

```
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import train_test_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import svm  
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

## Logistic Regression

```
In [21]: # Fitting the values in Logistic Regression model
```

```
model = LogisticRegression()  
model.fit(X_train_scaled, y_train)
```

```
Out[21]: LogisticRegression()
```

```
In [22]: model_prediction = model.predict(X_test_scaled)
```

```
In [23]: accuracy_score(model_prediction, y_test)
```

```
Out[23]: 1.0
```

## K-Nearest Neighbors

```
In [24]: # Fitting the models in K-Nearest Neighbors (KNN)
```

```
classi = KNeighborsClassifier(n_neighbors=3)  
classi.fit(X_train_scaled, y_train)
```

```
Out[24]: KNeighborsClassifier(n_neighbors=3)
```

```
In [25]: classi_pred = classi.predict(X_test_scaled)
```

```
In [26]: accuracy_score(y_test, classi_pred)
```

```
Out[26]: 0.9666666666666667
```

## Support Vector Classifier (SVC)

```
In [27]: # Fitting the models in support Vector Classifier (SVC)
```

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train_scaled, y_train)
```

```
Out[27]: SVC()
```

```
In [28]: svc_pred = svc.predict(X_test_scaled)
```

```
In [29]: accuracy_score(y_test, svc_pred)
```

```
Out[29]: 1.0
```

## Naive Bayes

```
In [30]: # Fitting the models in Naive Bayes
```

```
from sklearn.naive_bayes import GaussianNB

naive_b = GaussianNB()
naive_b.fit(X_train_scaled, y_train)
```

```
Out[30]: GaussianNB()
```

```
In [31]: naive_b_pred = naive_b.predict(X_test_scaled)
```

```
In [32]: accuracy_score(y_test, naive_b_pred)
```

```
Out[32]: 0.9666666666666667
```

## Model Evaluation

```
In [33]: # Confusion Matrix
```

```
confusion_matrix(y_test, model_prediction)
```

```
Out[33]: array([[11,  0,  0],
                [ 0, 13,  0],
                [ 0,  0,  6]], dtype=int64)
```

```
In [34]: confusion_matrix(y_test, classi_pred)
```

```
Out[34]: array([[11,  0,  0],
                [ 0, 13,  0],
                [ 0,  1,  5]], dtype=int64)
```

```
In [35]: confusion_matrix(y_test, svc_pred)
```

```
Out[35]: array([[11,  0,  0],
                [ 0, 13,  0],
                [ 0,  0,  6]], dtype=int64)
```

```
In [36]: confusion_matrix(y_test, naive_b_pred)
```

```
Out[36]: array([[11,  0,  0],
                [ 0, 13,  0],
                [ 0,  1,  5]], dtype=int64)
```

## Combining all the model scores

```
In [37]: # ALL Model Scores
```

```
results = pd.DataFrame({
    'Model' : ['Logistic Regression', 'SVM', 'KNN', 'Naive Bayes'],
    'Score' : [1.0, 1.0, 0.96, 0.96]
})

results_df = results.sort_values(by="Score", ascending=False)
results_df = results_df.set_index('Score')
results_df.head(9)
```

```
Out[37]:
```

	Model
Score	
1.00	Logistic Regression
1.00	SVM
0.96	KNN
0.96	Naive Bayes

## Entire Model Prediction

```
In [38]: # Model Prediction
```

```
model_prediction
```

```
Out[38]: array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
                'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
                'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
                'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
                'Iris-versicolor', 'Iris-setosa'], dtype=object)
```

```
In [39]: classi_pred
```

```
Out[39]: array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
                'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
                'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
                'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
                'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
                'Iris-versicolor', 'Iris-setosa'], dtype=object)
```



```
In [40]: svc_pred
```

```
Out[40]: array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',  
               'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',  
               'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',  
               'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',  
               'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',  
               'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',  
               'Iris-versicolor', 'Iris-setosa'], dtype=object)
```

```
In [41]: naive_b_pred
```

```
Out[41]: array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',  
               'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',  
               'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',  
               'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',  
               'Iris-versicolor', 'Iris-setosa'], dtype='<U15')
```

## Model Building and Deployment

### Overall model deployment flow

1. Data Collection
2. Data Preparation
3. Model building and Evaluation
4. Save model as pkl file
5. Load model using fastapi and deploy in localhost
6. Build and simple ui using streamlit and using post method access the model.
7. Build the front-end in react
8. Host frontend and backend in internet

```
In [42]: ##### To save the model in a pkl file
```

```
import pickle as pkl  
  
pkl.dump(model, open('model.pkl', 'wb'))  
pkl.dump(scaler, open('scaler.pkl', 'wb'))
```

```
In [43]: # 2, 3, 4, 5
```

```
# Load the scalar.pkl  
with open('scaler.pkl', 'rb') as scaler_file:  
    data = pkl.load(scaler_file)
```

```
In [44]: type(data)
```

```
Out[44]: sklearn.preprocessing._data.StandardScaler
```

```
In [45]: # Normalize using standard scalar
sample_input = np.array([2.0, 5.8, 2.8, 4.2]).reshape(-1,4)
processdata = data.transform(sample_input)
```

```
In [46]: processdata
```

```
Out[46]: array([[ -4.58247406,  6.38606206, -0.57151125,  3.83969158]])
```

```
In [50]: # Load the model.pkl

with open('model.pkl', 'rb') as model_file:
    model_data = pickle.load(model_file)

# Find ypred
model_predict = model_data.predict(processdata)

# Print the predicted class with decoded category label
model_predict
```

```
Out[50]: array(['Iris-setosa'], dtype=object)
```

```
In [ ]: # Below code is to get the library informations.
# Run all the code block inorder to get the Correct informations.

# !pip install session_info
# import session_info
# session_info.show()
```

## Acknowledgments

I would like to acknowledge the creators of the Iris Dataset and the developers of the python libraries and tools that help to make this project possible.

The Dataset that I have used in this notebook has taken from Kaggle.

Iris Dataset link: <https://www.kaggle.com/datasets/uciml/iris> (<https://www.kaggle.com/datasets/uciml/iris>)

## Contact Information

For any inquiries, feedback, or collaboration opportunities regarding the Iris Classification IPYNB file. Please feel free to reach out to me through the following channels:

 Email: [info@rubangino.in](mailto:info@rubangino.in) (<mailto:info@rubangino.in>)

 LinkedIn: [ruban-gino-singh](https://www.linkedin.com/in/ruban-gino-singh/) (<https://www.linkedin.com/in/ruban-gino-singh/>)

 X: [rubangino](https://twitter.com/Rubangino) (<https://twitter.com/Rubangino>)

 GitHub: [Ruban2205](https://github.com/Ruban2205) (<https://github.com/Ruban2205>)

I'm open to discussions, questions, and suggestions related to Iris\_Classification, Python programming, Machine Learning, Full stack Development and Data analysis. Don't hesitate to connect with me and start a conversation. Let's explore the fascinating world of Technology together!

Looking forward to connecting with you and sharing insights on the World of Technology!