Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

Problem Statement

The iris flower, scientifically known as Iris, is a distinctive genus of flowering plants. Within this genus, there are three primary species: Iris setosa, Iris versicolor, and Iris virginica. These species exhibit variations in their physical characteristics, particularly in the measurements of their sepal length, sepal width, petal length, and petal width.

Objective:

The objective of this project is to develop a machine learning model capable of learning from the measurements of iris flowers and accurately classifying them into their respective species. The model's primary goal is to automate the classification process based on the distinct characteristics of each iris species.

Project Details:

- **Iris Species**: The dataset consists of iris flowers, specifically from the species setosa, versicolor, and virginica.
- **Key Measurements:** The essential characteristics used for classification include sepal length, sepal width, petal length, and petal width.
- **Machine Learning Model:** The project involves the creation and training of a machine learning model to accurately classify iris flowers based on their measurements.

This project's significance lies in its potential to streamline and automate the classification of iris species, which can have broader applications in botany, horticulture, and environmental monitoring.

Let's Begin !

Import Libraries

```
# Import Libraries
# Importing Numpy & Pandas for data processing & data wrangling
import numpy as np
import pandas as pd
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

```
# Importing  tools for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Import evaluation metric libraries
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report

# Library used for data preprocessing
from sklearn.preprocessing import LabelEncoder

# Import model selection libraries
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, RepeatedStratifiedKFold

# Library used for ML Model implementation
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb

# Library used for ignore warnings
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

[ ]
```
# Load Dataset
df = pd.read_csv("C:/desktop/internship/Iris_Flower_classification.csv")
```

[ ]
```
# Dataset First Look
# View top 5 rows of the dataset
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

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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

```
[ ] # Dataset Rows & Columns count
    # Checking number of rows and columns of the dataset using shape
    print("Number of rows are: ",df.shape[0])
    print("Number of columns are: ",df.shape[1])
```

```
Number of rows are:  150
Number of columns are:  6
```

```
[ ] # Dataset Info
    # Checking information about the dataset using info
    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
```

Missing Values/Null Values

```
[ ] # Missing Values/Null Values Count
    df.isnull().sum()
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

```
[ ] # Missing Values/Null Values Count
    df.isnull().sum()
```

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

What did i know about the dataset?

- The Iris dataset consists of length and width mesurements of sepal and petal for different species in centimeter.
- There are 150 rows and 6 columns provided in the data.
- No duplicate values exist.
- No Null values exist.

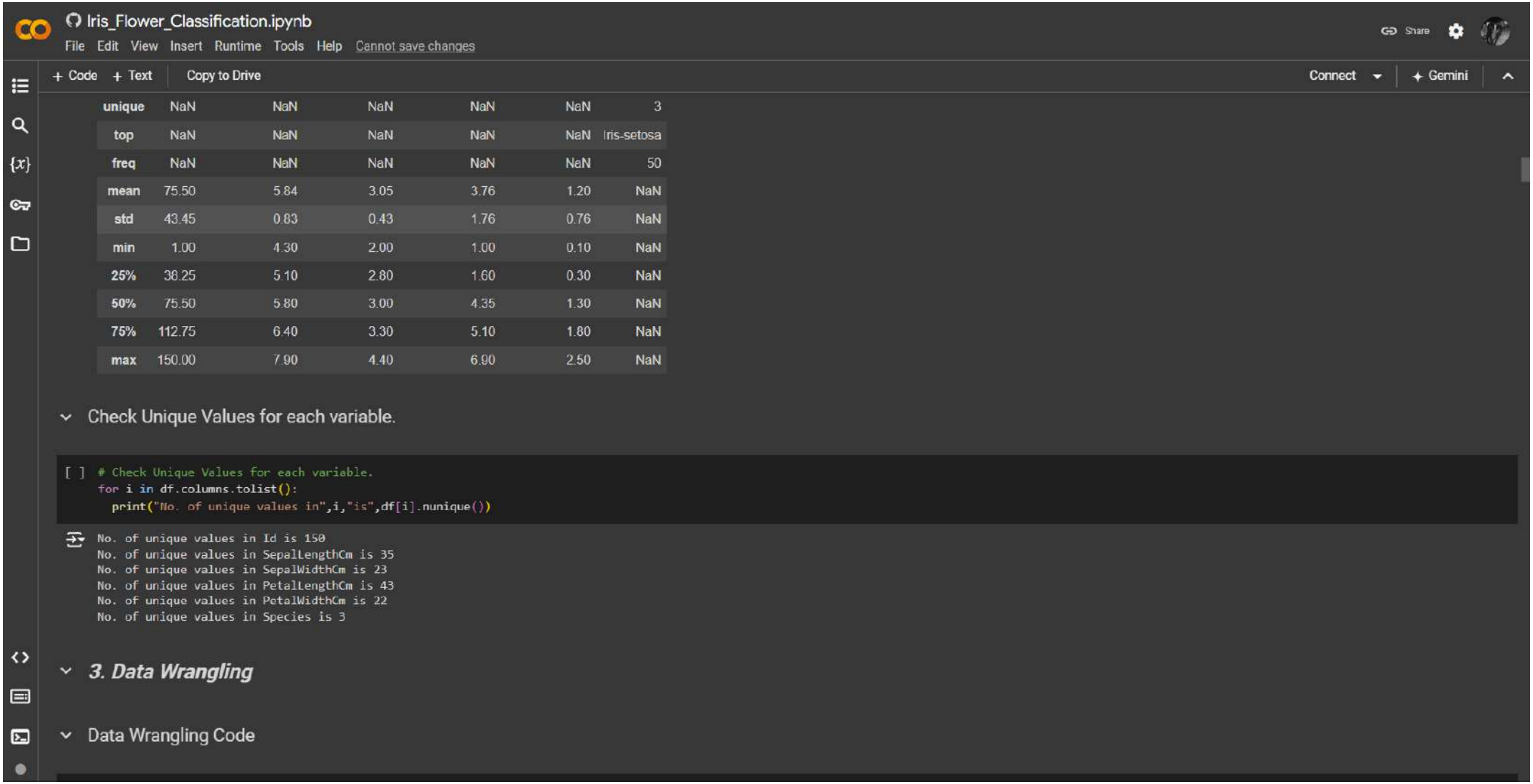
2. Understanding The Variables

```
[ ] # Dataset Columns
    df.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
[ ] # Dataset Describe (all columns included)
    df.describe(include= 'all').round(2)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
count	150.00	150.00	150.00	150.00	150.00	150



unique	NaN	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	NaN	Iris-setosa
freq	NaN	NaN	NaN	NaN	NaN	50
mean	75.50	5.84	3.05	3.76	1.20	NaN
std	43.45	0.83	0.43	1.76	0.76	NaN
min	1.00	4.30	2.00	1.00	0.10	NaN
25%	38.25	5.10	2.80	1.60	0.30	NaN
50%	75.50	5.80	3.00	4.35	1.30	NaN
75%	112.75	6.40	3.30	5.10	1.80	NaN
max	150.00	7.90	4.40	6.90	2.50	NaN

Check Unique Values for each variable.

```
[ ] # Check Unique Values for each variable.
    for i in df.columns.tolist():
      print("No. of unique values in",i,"is",df[i].nunique())
```

```
No. of unique values in Id is 150
No. of unique values in SepalLengthCm is 35
No. of unique values in SepalWidthCm is 23
No. of unique values in PetalLengthCm is 43
No. of unique values in PetalWidthCm is 22
No. of unique values in Species is 3
```

3. Data Wrangling

Data Wrangling Code

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

Data Wrangling Code

```
[ ] # We don't need the 1st column so let's drop that
    data=df.iloc[:,1:]
```

+ Code + Text

```
[ ] # New updated dataset
    data.head()
```

	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

What all manipulations have i done?

Only drop the first column of the dataset.

4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart - 1 : Distribution of Numerical Variables

```
[ ] # Chart - 1 Histogram visualization code for distribution of numerical variables
    # Create a figure with subplots
    plt.figure(figsize=(8, 6))
```

Iris_Flower_Classification.ipynb

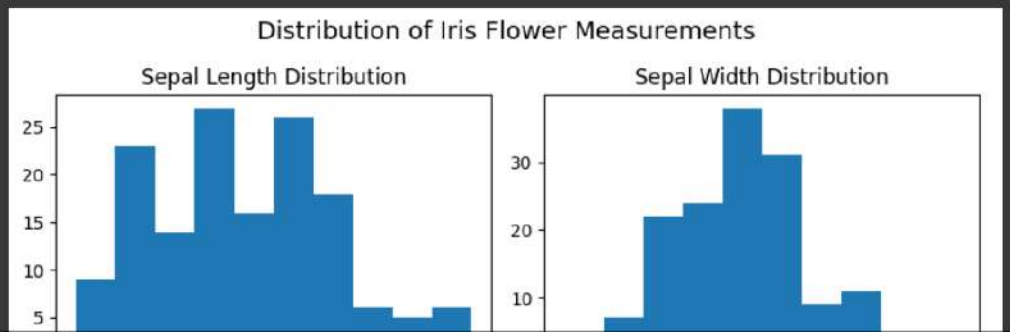
File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

```
[ ] # Chart - 1 Histogram visualization code for distribution of numerical variables
    # Create a figure with subplots
    plt.figure(figsize=(8, 6))
    plt.suptitle('Distribution of Iris Flower Measurements', fontsize=14)

    # Create a 2x2 grid of subplots
    plt.subplot(2, 2, 1)  # Subplot 1 (Top-Left)
    plt.hist(data['SepalLengthCm'])
    plt.title('Sepal Length Distribution')

    plt.subplot(2, 2, 2)  # Subplot 2 (Top-Right)
    plt.hist(data['SepalWidthCm'])
    plt.title('Sepal Width Distribution')

    plt.subplot(2, 2, 3)  # Subplot 3 (Bottom-Left)
    plt.hist(data['PetalLengthCm'])
    plt.title('Petal Length Distribution')

    plt.subplot(2, 2, 4)  # Subplot 4 (Bottom-Right)
    plt.hist(data['PetalWidthCm'])
    plt.title('Petal Width Distribution')

    # Display the subplots
    plt.tight_layout()  # Helps in adjusting the layout
    plt.show()
```



Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

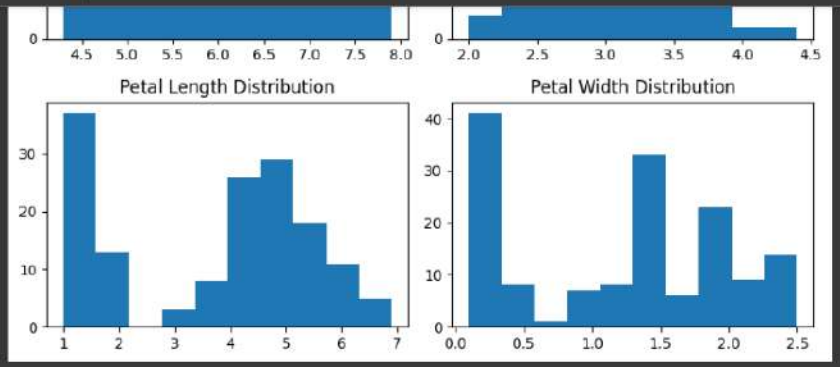


Chart - 2 : Sepal Length vs Sepal Width

```
[ ] # Define colors for each species and the corresponding species labels.
    colors = ['red', 'yellow', 'green']
    species = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```
[ ] # Chart - 2 Scatter plot visualization code for Sepal Length vs Sepal Width.
    # Create a scatter plot for Sepal Length vs Sepal Width for each species.
    for i in range(3):
        # Select data for the current species.
        x = data[data['Species'] == species[i]]

        # Create a scatter plot with the specified color and label for the current species.
        plt.scatter(x['SepalLengthCm'], x['SepalWidthCm'], c=colors[i], label=species[i])

    # Add labels to the x and y axes.
    plt.xlabel('Sepal Length')
    plt.ylabel('Sepal Width')

    # Add a legend to identify species based on colors.
    plt.legend()

    # Display the scatter plot.
    plt.show()
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

```
# Display the scatter plot.
plt.show()
```

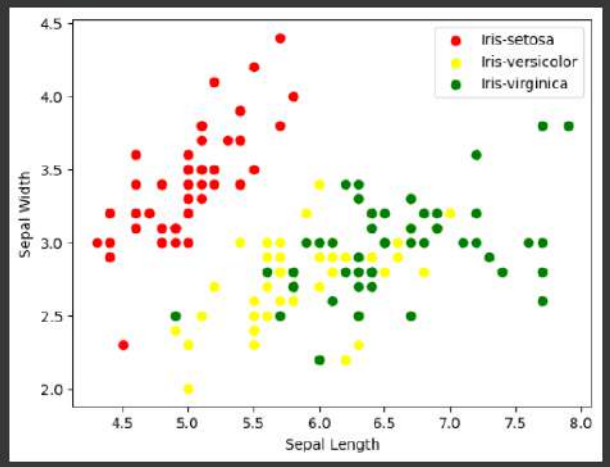


Chart - 3 : Petal Length vs Petal Width

```
[ ] # Chart - 3 Scatter plot visualization code for Petal Length vs Petal Width.
    # Create a scatter plot for Petal Length vs Petal Width for each species.
    for i in range(3):
        # Select data for the current species.
        x = data[data['Species'] == species[i]]

        # Create a scatter plot with the specified color and label for the current species.
        plt.scatter(x['PetalLengthCm'], x['PetalWidthCm'], c=colors[i], label=species[i])

    # Add labels to the x and y axes.
    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')

    # Add a legend to identify species based on colors.
    plt.legend()

    # Display the scatter plot.
    plt.show()
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect ✦ Gemini

```
# Display the scatter plot.
plt.show()
```

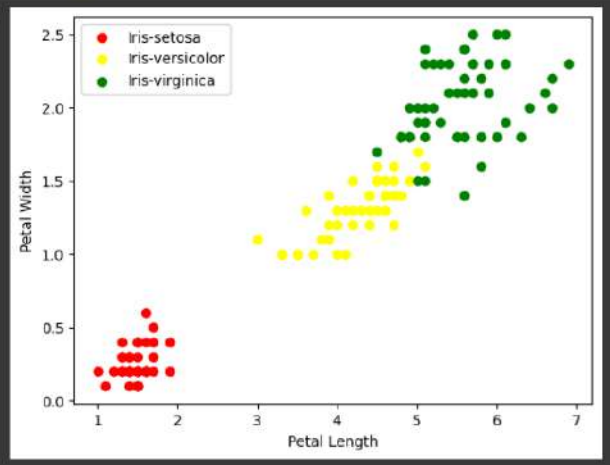


Chart - 4 : Sepal Length vs Petal Length

```
[ ] # Chart - 4 Scatter plot visualization code for Sepal Length vs Petal Length.
    # Create a scatter plot for Sepal Length vs Petal Length for each species.
    for i in range(3):
        # Select data for the current species.
        x = data[data['Species'] == species[i]]

        # Create a scatter plot with the specified color and label for the current species.
        plt.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c=colors[i], label=species[i])

    # Add labels to the x and y axes.
    plt.xlabel('Sepal Length')
    plt.ylabel('Petal Length')

    # Add a legend to identify species based on colors.
    plt.legend()

    # Display the scatter plot.
    plt.show()
```

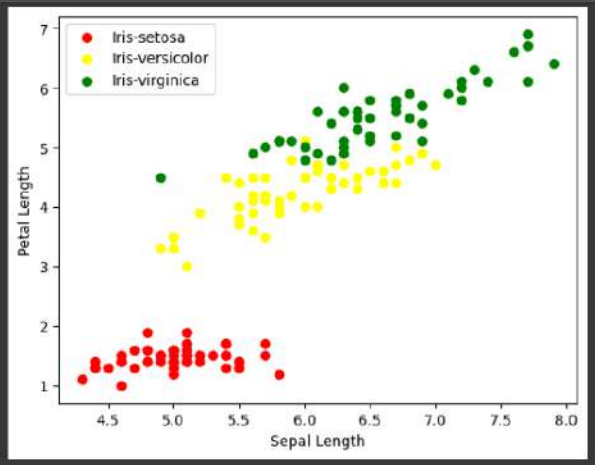
Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini



+ Code + Text

Chart - 5 : Sepal Width vs Petal Width

```
[ ] # Chart - 5 Scatter plot visualization code for Sepal Width vs Petal Width.
    # Create a scatter plot for Sepal Width vs Petal Width for each species.
    for i in range(3):
        # Select data for the current species.
        x = data[data['Species'] == species[i]]

        # Create a scatter plot with the specified color and label for the current species.
        plt.scatter(x['SepalWidthCm'], x['PetalWidthCm'], c=colors[i], label=species[i])

    # Add labels to the x and y axes.
    plt.xlabel('Sepal Width')
    plt.ylabel('Petal Width')

    # Add a legend to identify species based on colors.
    plt.legend()

    # Display the scatter plot.
    plt.show()
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

Connect Gemini

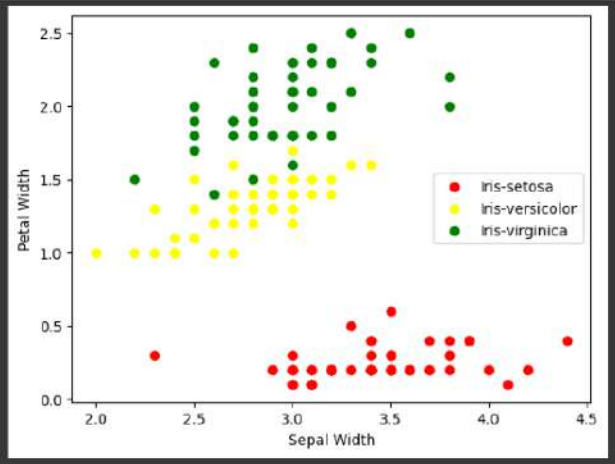
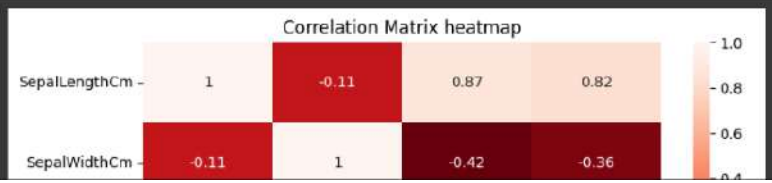


Chart - 6 : Correlation Heatmap

```
[ ] # Correlation Heatmap Visualization Code
    corr_matrix = data.corr()

    # Plot Heatmap
    plt.figure(figsize=(8, 4))
    sns.heatmap(corr_matrix, annot=True, cmap='Reds_r')

    # Setting Labels
    plt.title('Correlation Matrix heatmap')

    # Display Chart
    plt.show()
```



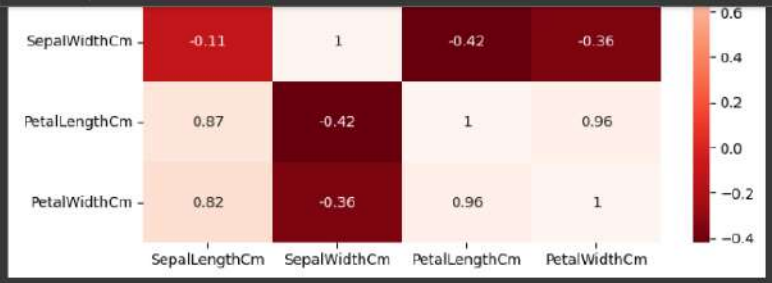
Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini



5. Feature Engineering & Data Pre-processing

1. Categorical Encoding

```
[ ] # Encode the categorical columns
    # Create a LabelEncoder object
    le = LabelEncoder()

    # Encode the 'Species' column to convert the species names to numerical labels
    data['Species'] = le.fit_transform(data['Species'])

    # Check the unique values in the 'Species' column after encoding
    unique_species = data['Species'].unique()

    # Display the unique encoded values
    print("Encoded Species Values:")
    print(unique_species) # 'Iris-setosa' == 0, 'Iris-versicolor' == 1, 'Iris-virginica' == 2
```

```
Encoded Species Values:
[0 1 2]
```

2. Data Scaling

```
[ ] # Defining the X and y
    x=data.drop(columns=['Species'], axis=1)
    y=data['Species']
```

3. Data Splitting

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect Gemini

```
[ ] # Defining the X and y
    x=data.drop(columns=['Species'], axis=1)
    y=data['Species']
```

3. Data Splitting

```
[ ] # Splitting the data to train and test
    x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.3)
```

```
[ ] # Checking the train distribution of dependent variable
    y_train.value_counts()
```

```
1    37
2    35
0    33
Name: Species, dtype: int64
```

6. ML Model Implementation

```
[ ] def evaluate_model(model, x_train, x_test, y_train, y_test):
        '''The function will take model, x train, x test, y train, y test
        and then it will fit the model, then make predictions on the trained model,
        it will then print roc-auc score of train and test, then plot the roc, auc curve,
        print confusion matrix for train and test, then print classification report for train and test,
        then plot the feature importances if the model has feature importances,
        and finally it will return the following scores as a list:
        recall_train, recall_test, acc_train, acc_test, F1_train, F1_test
        '''

        # Fit the model to the training data.
        model.fit(x_train, y_train)

        # make predictions on the test data
        y_pred_train = model.predict(x_train)
        y_pred_test = model.predict(x_test)

        # calculate confusion matrix
        cm_train = confusion_matrix(y_train, y_pred_train)
        cm_test = confusion_matrix(y_test, y_pred_test)

        fig, ax = plt.subplots(1, 2, figsize=(11,4))

        print("\nConfusion Matrix:")
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect + Gemini

```
sns.heatmap(cm_train, annot=True, xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'], cmap="Oranges", fmt='.4g', ax=ax[0])
ax[0].set_xlabel("Predicted Label")
ax[0].set_ylabel("True Label")
ax[0].set_title("Train Confusion Matrix")

sns.heatmap(cm_test, annot=True, xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'], cmap="Oranges", fmt='.4g', ax=ax[1])
ax[1].set_xlabel("Predicted Label")
ax[1].set_ylabel("True Label")
ax[1].set_title("Test Confusion Matrix")

plt.tight_layout()
plt.show()


# calculate classification report
cr_train = classification_report(y_train, y_pred_train, output_dict=True)
cr_test = classification_report(y_test, y_pred_test, output_dict=True)
print("\nTrain Classification Report:")
crt = pd.DataFrame(cr_train).T
print(crt.to_markdown())
# sns.heatmap(pd.DataFrame(cr_train).T.iloc[:, :-1], annot=True, cmap="Blues")
print("\nTest Classification Report:")
crt2 = pd.DataFrame(cr_test).T
print(crt2.to_markdown())
# sns.heatmap(pd.DataFrame(cr_test).T.iloc[:, :-1], annot=True, cmap="Blues")

precision_train = cr_train['weighted avg']['precision']
precision_test = cr_test['weighted avg']['precision']

recall_train = cr_train['weighted avg']['recall']
recall_test = cr_test['weighted avg']['recall']

acc_train = accuracy_score(y_true = y_train, y_pred = y_pred_train)
acc_test = accuracy_score(y_true = y_test, y_pred = y_pred_test)

F1_train = cr_train['weighted avg']['f1-score']
F1_test = cr_test['weighted avg']['f1-score']

model_score = [precision_train, precision_test, recall_train, recall_test, acc_train, acc_test, F1_train, F1_test ]
return model_score
```

```
[ ] # Create a score dataframe
    score = pd.DataFrame(index = ['Precision Train', 'Precision Test','Recall Train','Recall Test','Accuracy Train', 'Accuracy Test', 'F1 macro Train', 'F1 macro Test'])
```

ML Model - 1 : Logistic regression

```
[ ] # ML Model - 1 Implementation
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect + Gemini

```
[ ] # Create a score dataframe
    score = pd.DataFrame(index = ['Precision Train', 'Precision Test','Recall Train','Recall Test','Accuracy Train', 'Accuracy Test', 'F1 macro Train', 'F1 macro Test'])
```

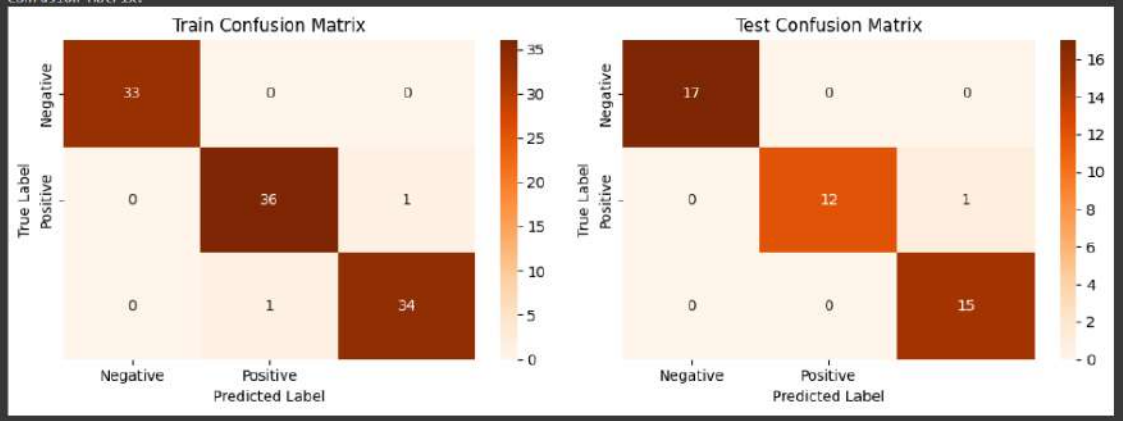
ML Model - 1 : Logistic regression

```
[ ] # ML Model - 1 Implementation
    lr_model = LogisticRegression(fit_intercept=True, max_iter=10000)

    # Model is trained (fit) and predicted in the evaluate model
```

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

```
[ ] # Visualizing evaluation Metric Score chart
    lr_score = evaluate_model(lr_model, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	33
1	0.972973	0.972973	0.972973	37
2	0.971429	0.971429	0.971429	35
accuracy	0.980952	0.980952	0.980952	0.980952

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect ✦ Gemini

```
Test Classification Report:
|              |   precision |    recall |   f1-score |   support |
|:-------------|------------:|----------:|-----------:|----------:|
| 0            |    1        | 1         |  1         | 17        |
| 1            |    1        | 0.923077  |  0.96      | 13        |
| 2            |    0.9375   | 1         |  0.967742  | 15        |
| accuracy     |    0.977778 | 0.977778  |  0.977778  |  0.977778 |
| macro avg    |    0.979167 | 0.974359  |  0.975914  | 45        |
| weighted avg |    0.979167 | 0.977778  |  0.977692  | 45        |
```

```
# Updated Evaluation metric Score Chart
score['Logistic regression'] = lr_score
score
```

	Logistic regression
Precision Train	0.980952
Precision Test	0.979167
Recall Train	0.980952
Recall Test	0.977778
Accuracy Train	0.980952
Accuracy Test	0.977778
F1 macro Train	0.980952
F1 macro Test	0.977692

2. Cross- Validation & Hyperparameter Tuning

```
[ ] # ML Model - 1 Implementation with hyperparameter optimization techniques (i.e., GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
    # Define the hyperparameter grid
    param_grid = {'C': [100,10,1,0.1,0.01,0.001,0.0001],
                  'penalty': ['l1', 'l2'],
                  'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}

    # Initializing the logistic regression model
    logreg = LogisticRegression(fit_intercept=True, max_iter=10000, random_state=0)

    # Repeated stratified kfold
    rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=4, random_state=0)

    # Using GridSearchCV to tune the hyperparameters using cross-validation
    grid = GridSearchCV(logreg, param_grid, cv=rskf)
    grid.fit(x_train, y_train)
```

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect + Gemini

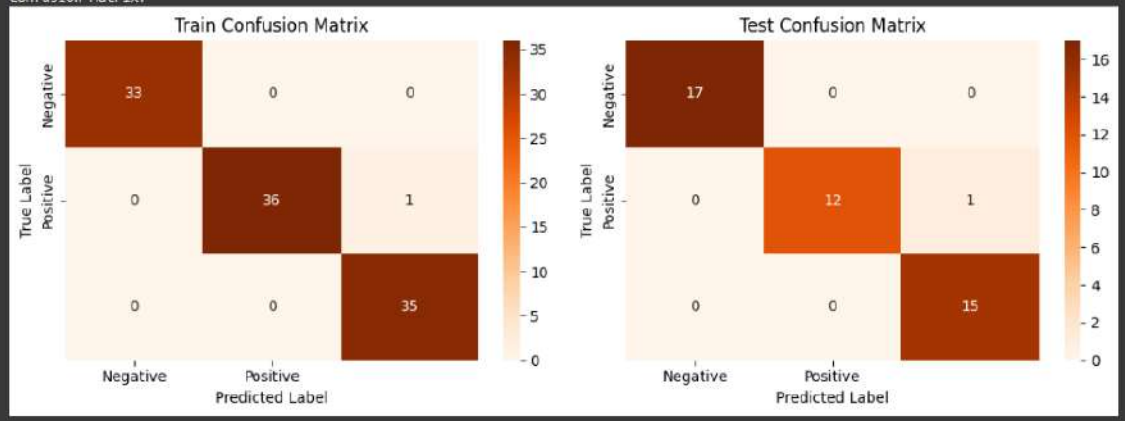
```
# Select the best hyperparameters found by GridSearchCV
best_params = grid.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'C': 10, 'penalty': 'l2', 'solver': 'saga'}
```

```
# Initiate model with best parameters
lr_model2 = LogisticRegression(C=best_params['C'],
                               penalty=best_params['penalty'],
                               solver=best_params['solver'],
                               max_iter=10000, random_state=0)
```

```
# Visualizing evaluation Metric Score chart
lr_score2 = evaluate_model(lr_model2, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	33
1	1	0.972973	0.986301	37
2	0.972222	1	0.985915	35
accuracy	0.990476	0.990476	0.990476	0.990476
macro avg	0.990741	0.990991	0.990739	105
weighted avg	0.990741	0.990476	0.990478	105

Iris_Flower_Classification.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect + Gemini

Test Classification Report:

	precision	recall	f1-score	support
0	1	1	1	17
1	1	0.923077	0.96	13
2	0.9375	1	0.967742	15
accuracy	0.977778	0.977778	0.977778	0.977778
macro avg	0.979167	0.974359	0.975914	45
weighted avg	0.979167	0.977778	0.977692	45

```
score['Logistic regression tuned'] = lr_score2
```

Which hyperparameter optimization technique have i used and why?

The hyperparameter optimization technique used is GridSearchCV. GridSearchCV is a method that performs an exhaustive search over a specified parameter grid to find the best hyperparameters for a model. It is a popular method for hyperparameter tuning because it is simple to implement and can be effective in finding good hyperparameters for a model.

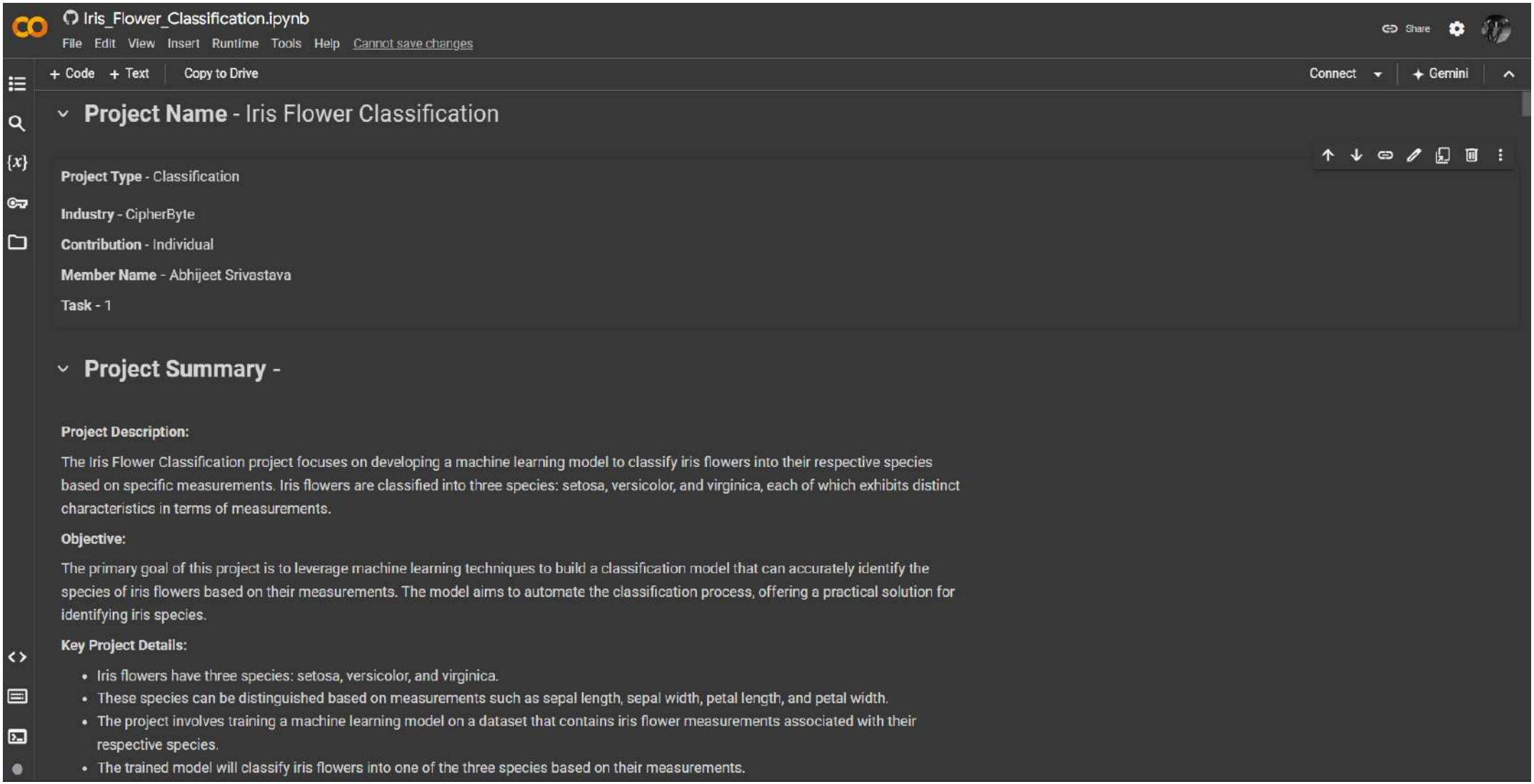
The choice of hyperparameter optimization technique depends on various factors such as the size of the parameter space, the computational resources available, and the time constraints. GridSearchCV can be a good choice when the parameter space is relatively small and computational resources are not a major concern.

Have i seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

```
[ ] # Updated Evaluation metric Score Chart
    score
```

	Logistic regression	Logistic regression tuned
Precision Train	0.980952	0.990741
Precision Test	0.979167	0.979167
Recall Train	0.980952	0.990476
Recall Test	0.977778	0.977778
Accuracy Train	0.980952	0.990476
Accuracy Test	0.977778	0.977778
F1 macro Train	0.980952	0.990478
F1 macro Test	0.977692	0.977692

It can seen that hyperparameter tuning did not improve the performance of the Logistic Regression model on the test set. The precision, recall


Iris_Flower_Classification.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Connect
Gemini
Project Name - Iris Flower Classification
Project Type - Classification
Industry - CipherByte
Contribution - Individual
Member Name - Abhijeet Srivastava
Task - 1
Project Summary -
Project Description:
The Iris Flower Classification project focuses on developing a machine learning model to classify iris flowers into their respective species based on specific measurements. Iris flowers are classified into three species: setosa, versicolor, and virginica, each of which exhibits distinct characteristics in terms of measurements.
Objective:
The primary goal of this project is to leverage machine learning techniques to build a classification model that can accurately identify the species of iris flowers based on their measurements. The model aims to automate the classification process, offering a practical solution for identifying iris species.
Key Project Details:
Iris flowers have three species: setosa, versicolor, and virginica.
These species can be distinguished based on measurements such as sepal length, sepal width, petal length, and petal width.
The project involves training a machine learning model on a dataset that contains iris flower measurements associated with their respective species.
The trained model will classify iris flowers into one of the three species based on their measurements.