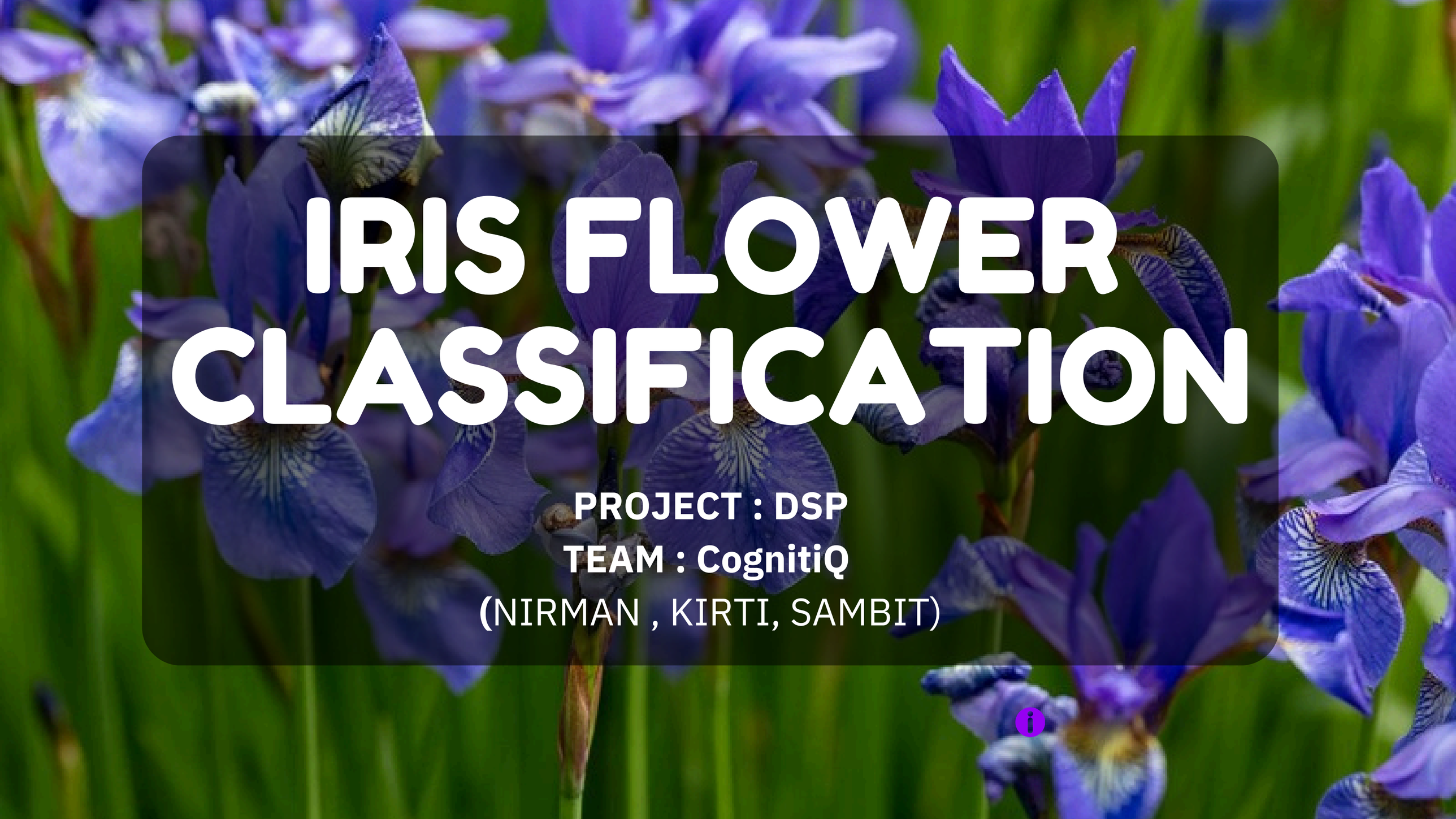


IRIS FLOWER CLASSIFICATION

PROJECT : DSP

TEAM : CognitiQ

(NIRMAN , KIRTI, SAMBIT)

CONTENTS

Our model organism

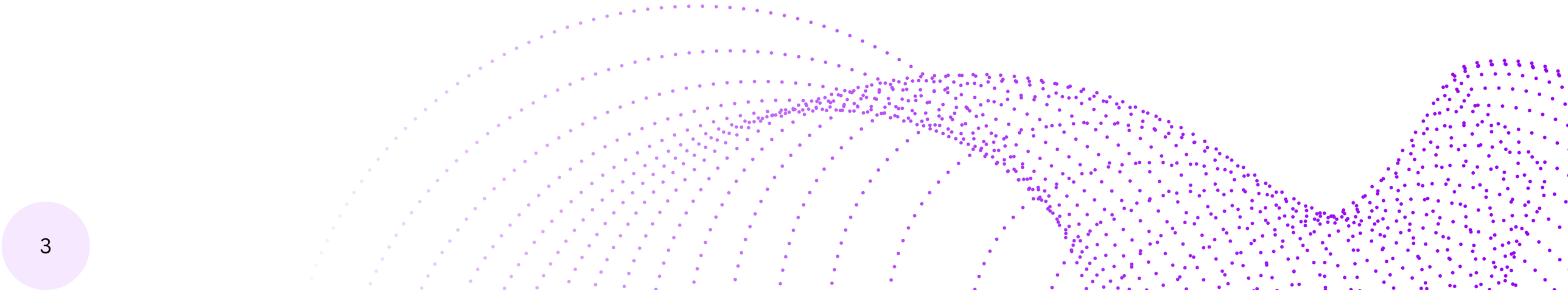
Iris versicolor

Iris setosa

Iris verginica

PROBLEM STATEMENT

Given a set of features corresponding to Iris flowers. Classify the features using standard classification techniques to categorize the flowers into 3 categories, setosa, versicolor, and Virginia, so that you can achieve at least 80% accuracy.



EXPLANATION

We are given a data set of 3 species of Iris and 4 features of each species flowers (**Sepal length**, **Sepal Width**, **Petal length, Petal Width**). We have to apply classification techniques on our data set which can classify a given flower to its correct species based on its 4 features with 80% or more accuracy.

DATA SET

Iris Dataset: Contains 150 samples of three species (setosa, versicolour, virginica).

	sepal_length	sepal_width	petal_length	petal_width	class
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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

Basic Data Analysis

LABEL ENCODER

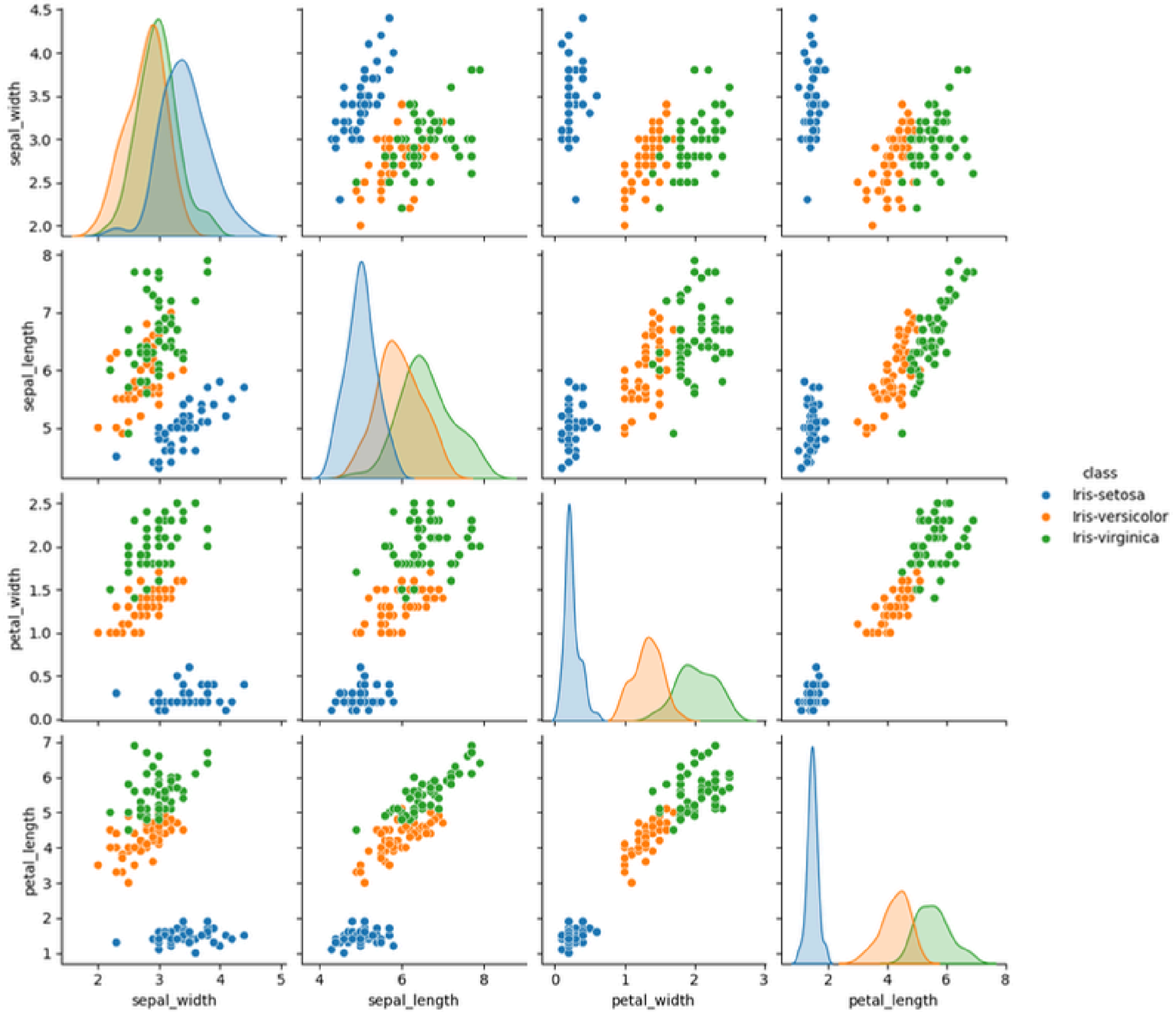
Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine-learning project.

```
lab_en = ppr.LabelEncoder()
iris['encoded_target'] = lab_en.fit_transform(iris['class'])
```

PAIR PLOT

A pair plot allows us to see both distribution of single variables and relationships between two variables.

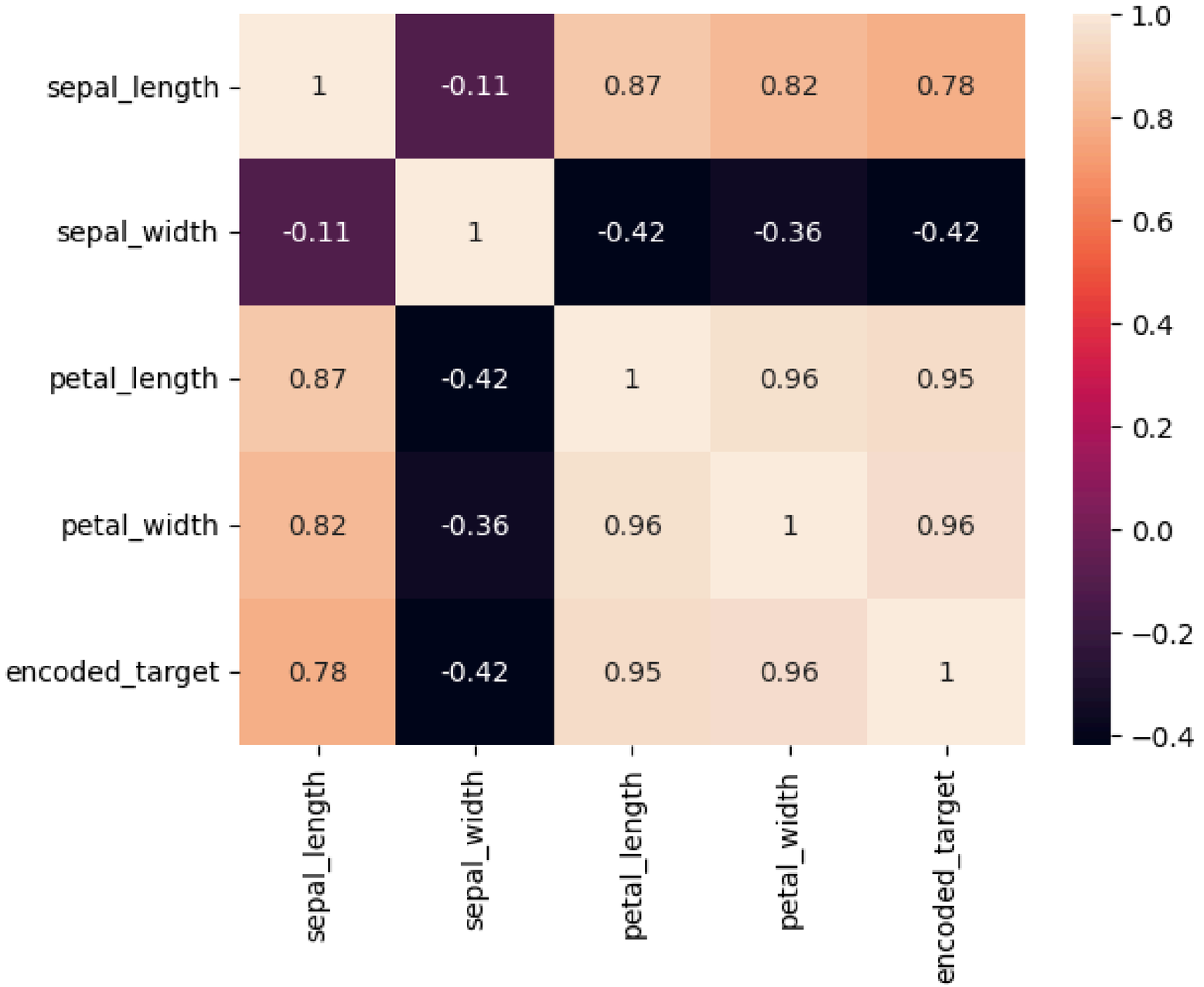
```
g1 = sns.pairplot( iris,
        vars = ["sepal_width",
                "sepal_length",
                "petal_width",
                "petal_length"],
        hue = 'class')
plt.show()
```



CORRELATION MATRIX

The correlation matrix is a matrix that shows the correlation between variables. It gives the correlation between all the possible pairs of values in a matrix format.

```
iris_numeric = iris.select_dtypes(
        include=[float,int])
cor_mat = iris_numeric.corr()
g2 = sns.heatmap(cor_mat,
        annot = True)
plt.show()
```

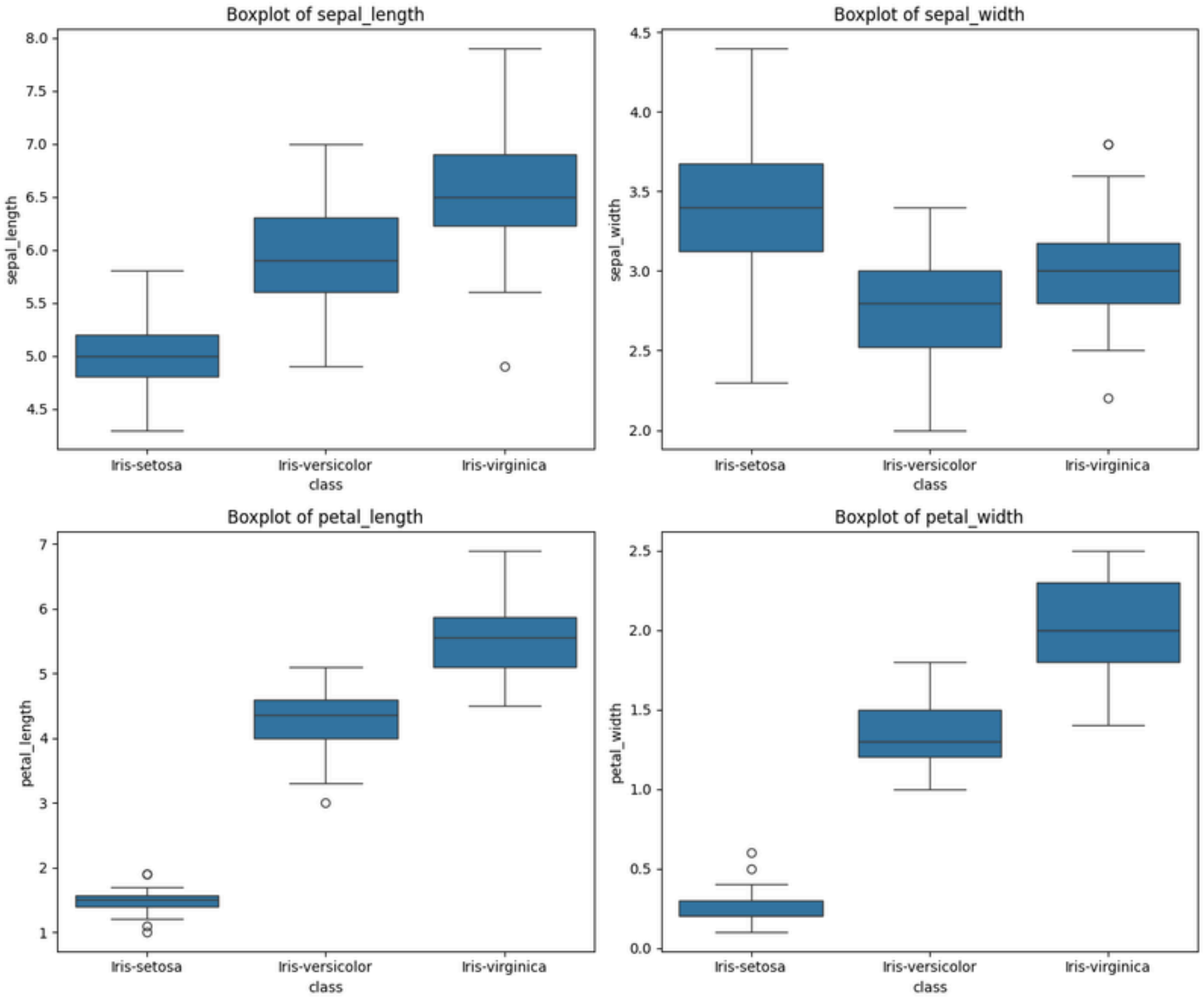


BOX PLOT

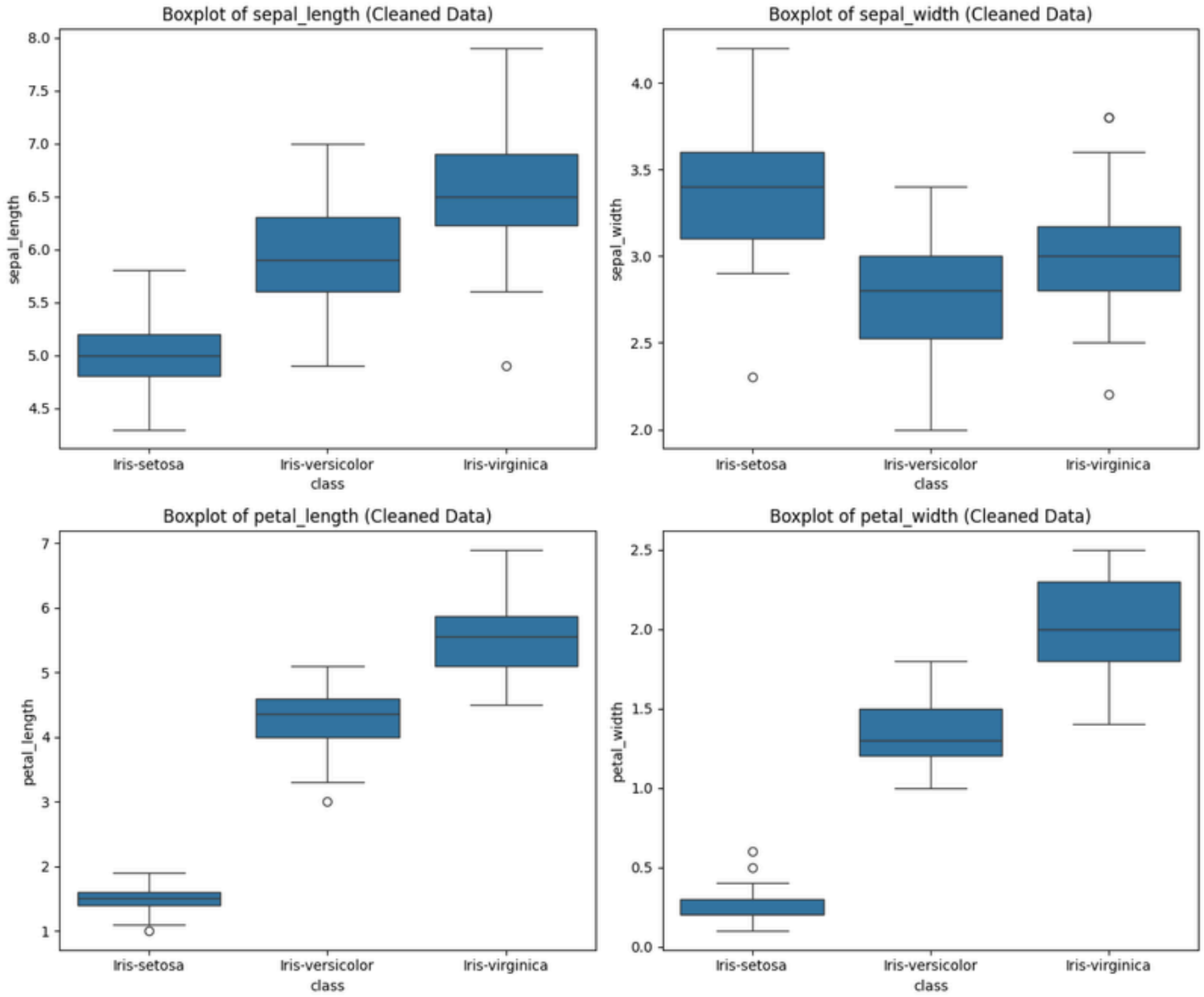
Box plots visually show the distribution of numerical data and skewness by displaying the data quartiles (or percentiles) and averages.
Box plots show the five-number summary of a set of data: including the minimum score, first (lower) quartile, median, third (upper) quartile, and maximum score.

```
# Create subplots for multiple boxplots
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
# Create a boxplot for each feature
for i, feature in enumerate(features):
        sns.boxplot(x='class', y=feature, data=iris, ax=axes[i//2, i%2])
        axes[i//2, i%2].set_title(f'Boxplot of {feature}')
plt.tight_layout()
plt.show()
```

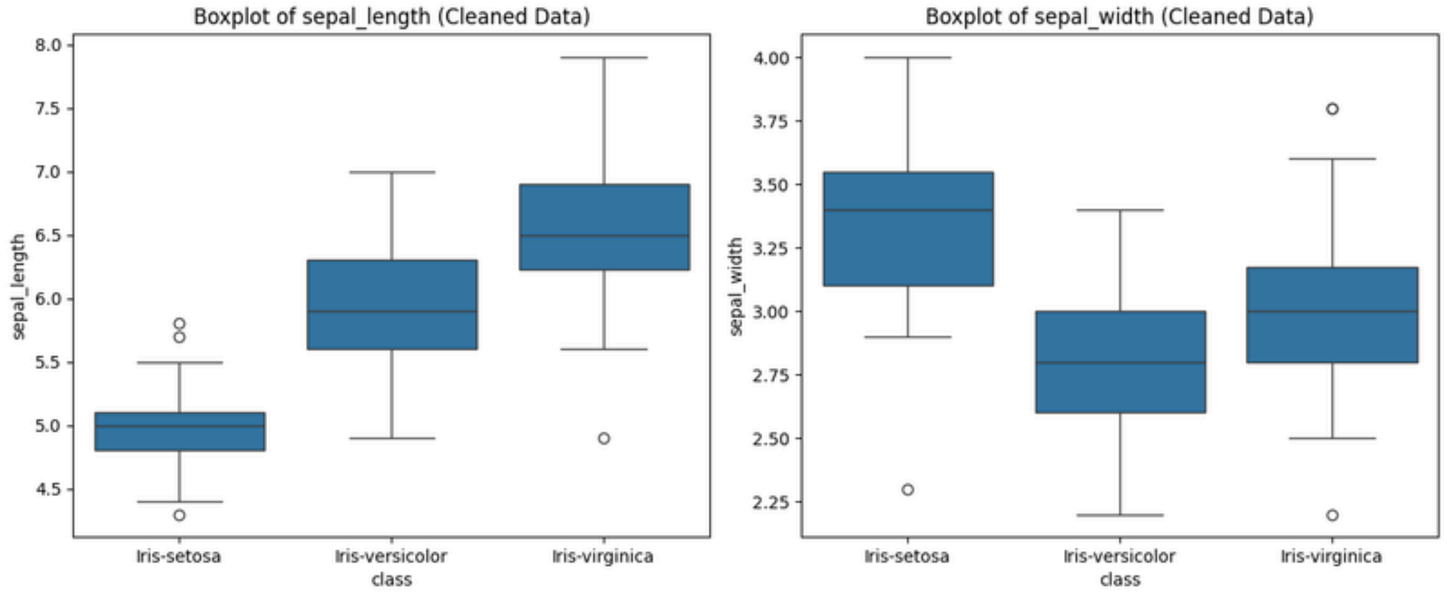
BOX PLOT BEFORE DATA CLEANING

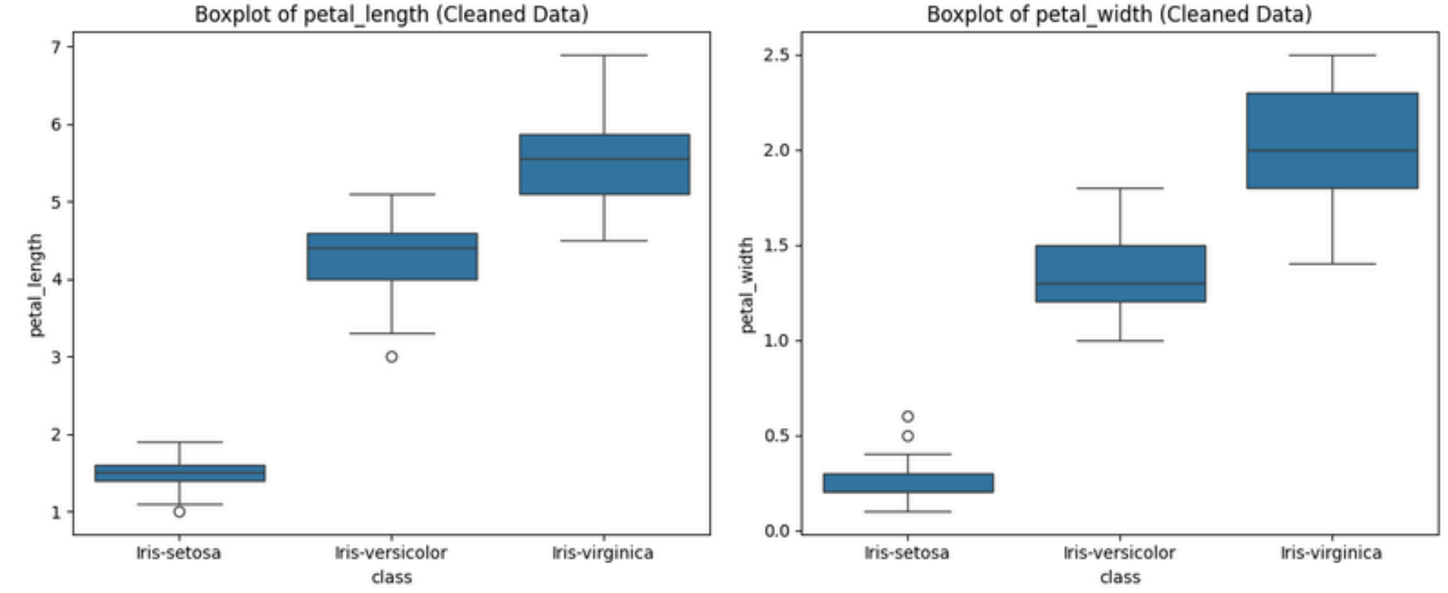


BOX PLOT BEFORE DATA CLEANING



BOX PLOT AFTER DATA CLEANING



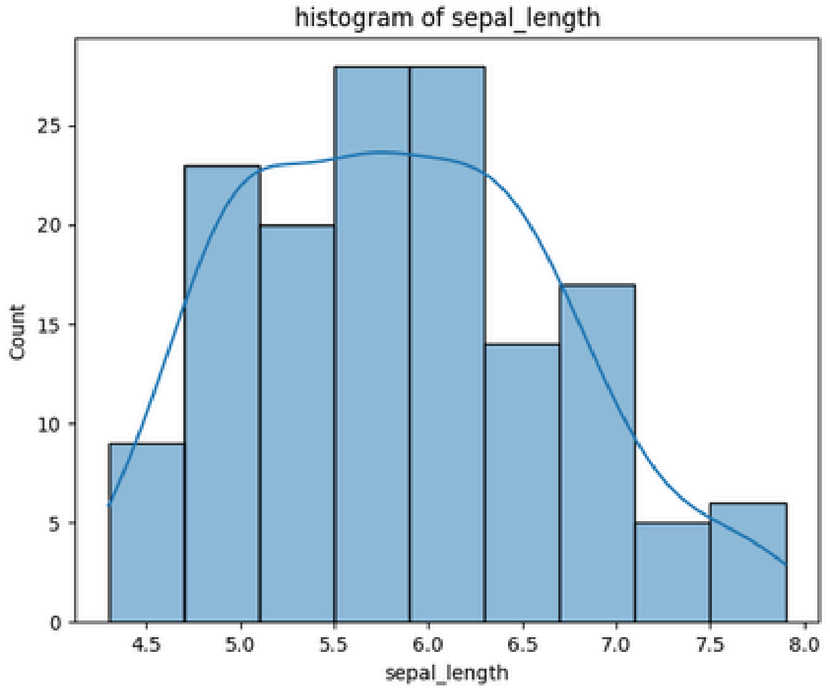


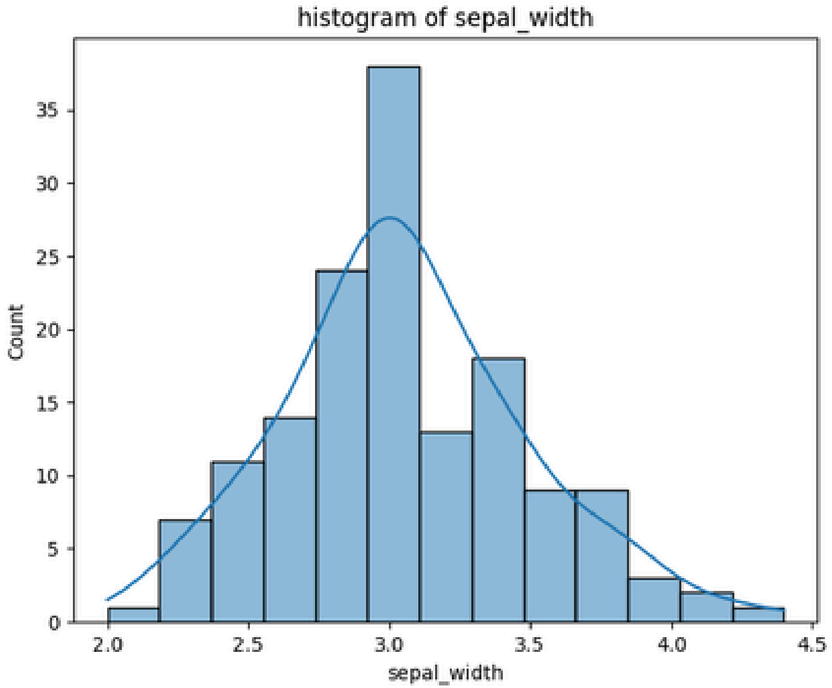
HISTOGRAM

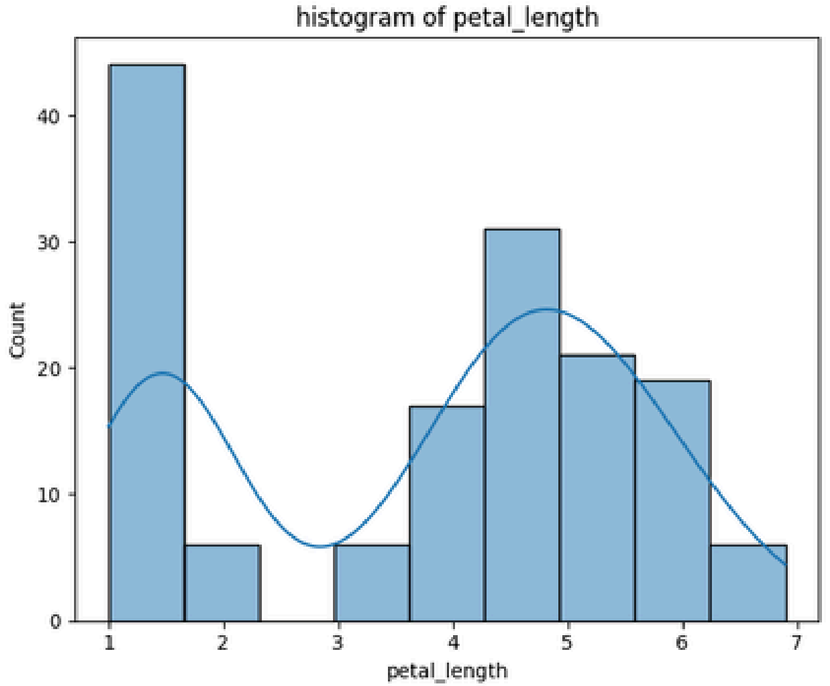
A histogram consists of bars where the height of each bar represents the frequency or probability of the occurrence of data within specific intervals, known as bins. The x-axis represents the value ranges (bins), and the y-axis represents the frequency or density. Kernel Density Estimation (KDE) is a non-parametric method for estimating the probability density function of a random variable. It provides a smooth estimate of the distribution, overcoming some of the limitations of histograms, such as sensitivity to bin size.

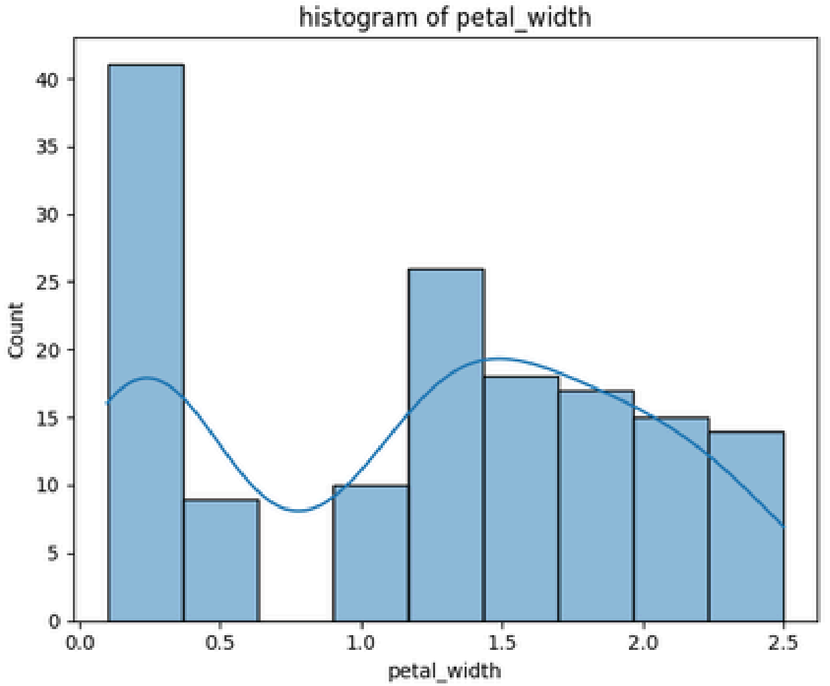
```
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
# Create a histogram for each feature
for i, feature in enumerate(features):
        sns.histplot(iris[feature],kde = True, ax=axes[i//2, i%2])
        axes[i//2, i%2].set_title(f'histogram of {feature}')
plt.tight_layout()
plt.show()
```

HISTOGRAM


histogram of sepal_length
Count
sepal_length
0
5
10
15
20
25
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


histogram of sepal_width
Count
sepal_width
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5

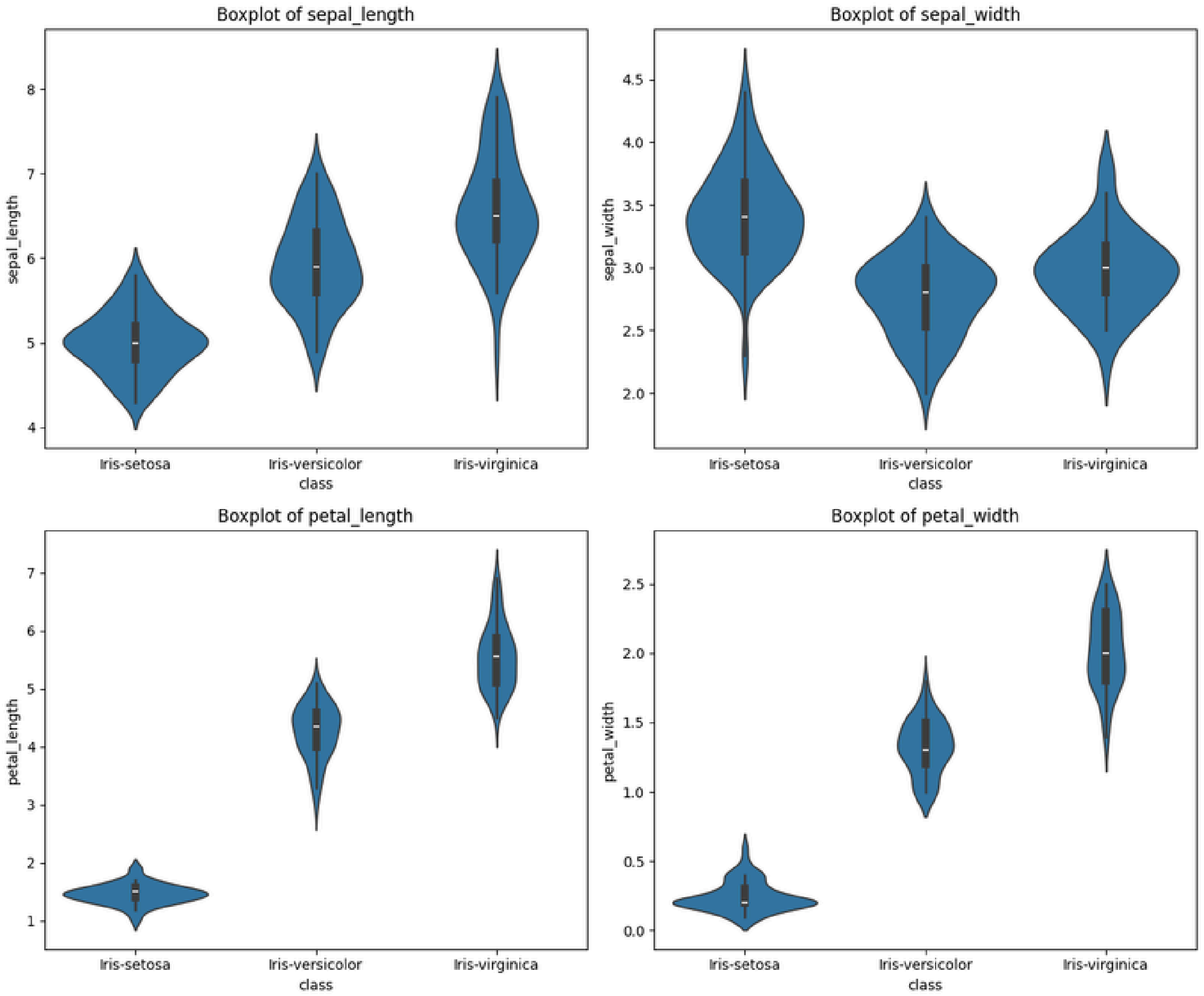

histogram of petal_length
Count
petal_length
0
10
20
30
40
1
2
3
4
5
6
7


histogram of petal_width
Count
petal_width
0
5
10
15
20
25
30
35
40
0.0
0.5
1.0
1.5
2.0
2.5

VILOLIN PLOT

Violin Plot is a method to visualize the distribution of numerical data of different variables. It is quite similar to Box Plot but with a rotated plot on each side, giving more information about the density estimate on the y-axis.

```
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
# Create a violin for each feature
for i, feature in enumerate(features):
  sns.violinplot(x='class', y=feature, data=iris, ax=axes[i//2, i%2])
  axes[i//2, i%2].set_title(f'Boxplot of {feature}')
plt.tight_layout()
plt.show()
```

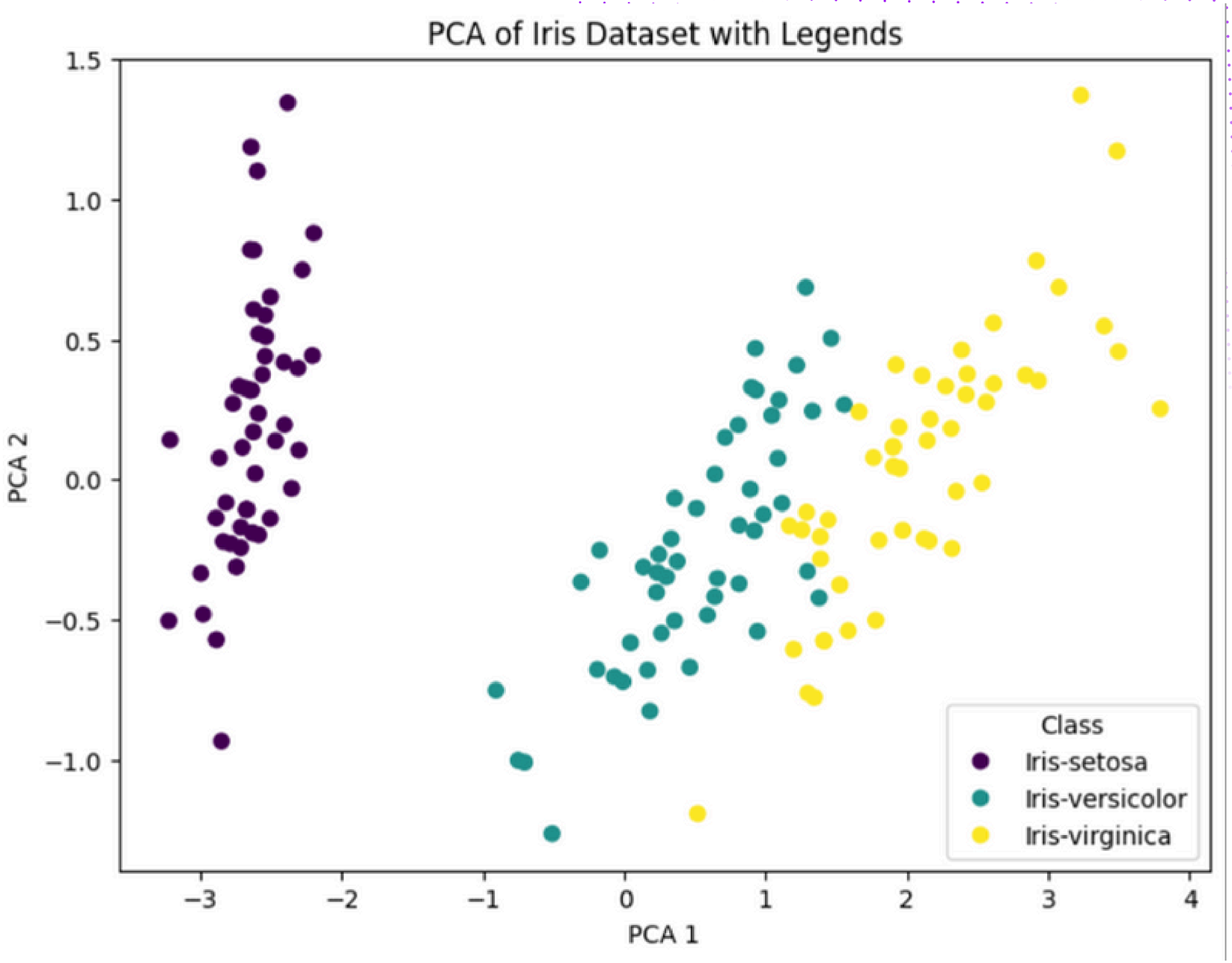

Boxplot of sepal_length
sepal_length
8
7
6
5
4
Iris-setosa
Iris-versicolor
Iris-virginica
class
Boxplot of sepal_width
sepal_width
4.5
4.0
3.5
3.0
2.5
2.0
Iris-setosa
Iris-versicolor
Iris-virginica
class
Boxplot of petal_length
petal_length
7
6
5
4
3
2
1
Iris-setosa
Iris-versicolor
Iris-virginica
class
Boxplot of petal_width
petal_width
2.5
2.0
1.5
1.0
0.5
0.0
Iris-setosa
Iris-versicolor
Iris-virginica
class

PCA : PRINCIPLE COMPONENT ANALYSIS

We run PCA on our data set to reduce the dimensionality of the dataset and visualize how the classes separate in a lower-dimensional space.

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(iris[['sepal_length', 'sepal_width',
          'petal_length', 'petal_width']])
plt.figure(figsize=(8,6))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1],
         c= iris['encoded_target'], cmap='viridis')
legend_labels = iris['class'].unique()
handles = scatter.legend_elements()[0]
plt.legend(handles, legend_labels, title="Class")
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('PCA of Iris Dataset with Legends')
plt.show()
```

PCA : PRINCIPLE COMPONENT ANALYSIS

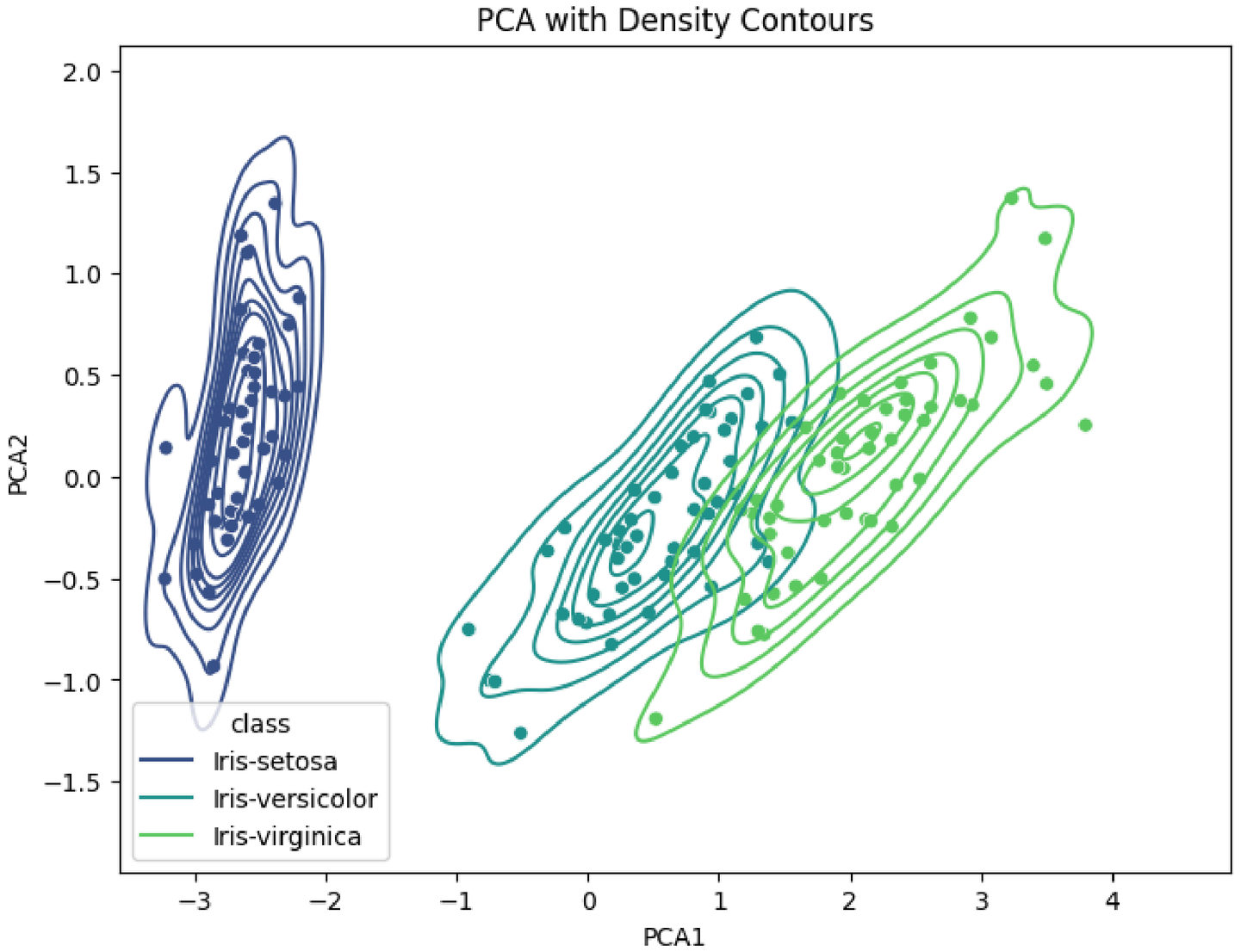


PCA + DENSITY CONTOURS

Density contours on scatter plot allow visualise the kernel density estimation (KDE) contours. Contours represent the estimated density of points in different regions of the plot. Useful for understanding where the data is concentrated.

```
# Add PCA results to the DataFrame
iris['PCA1'] = pca_result[:, 0]
iris['PCA2'] = pca_result[:, 1]
# Set up the figure
plt.figure(figsize=(8,6))
# Create the scatter plot with density contours
sns.scatterplot(x='PCA1', y='PCA2', hue='class', data=iris,
        palette='viridis', legend=False)
sns.kdeplot(x='PCA1', y='PCA2', hue='class', data=iris, fill=False,
        levels=10, palette='viridis')
# Add labels and title
plt.title('PCA with Density Contours')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.show()
```

PCA : PRINCIPLE COMPONENT ANALYSIS

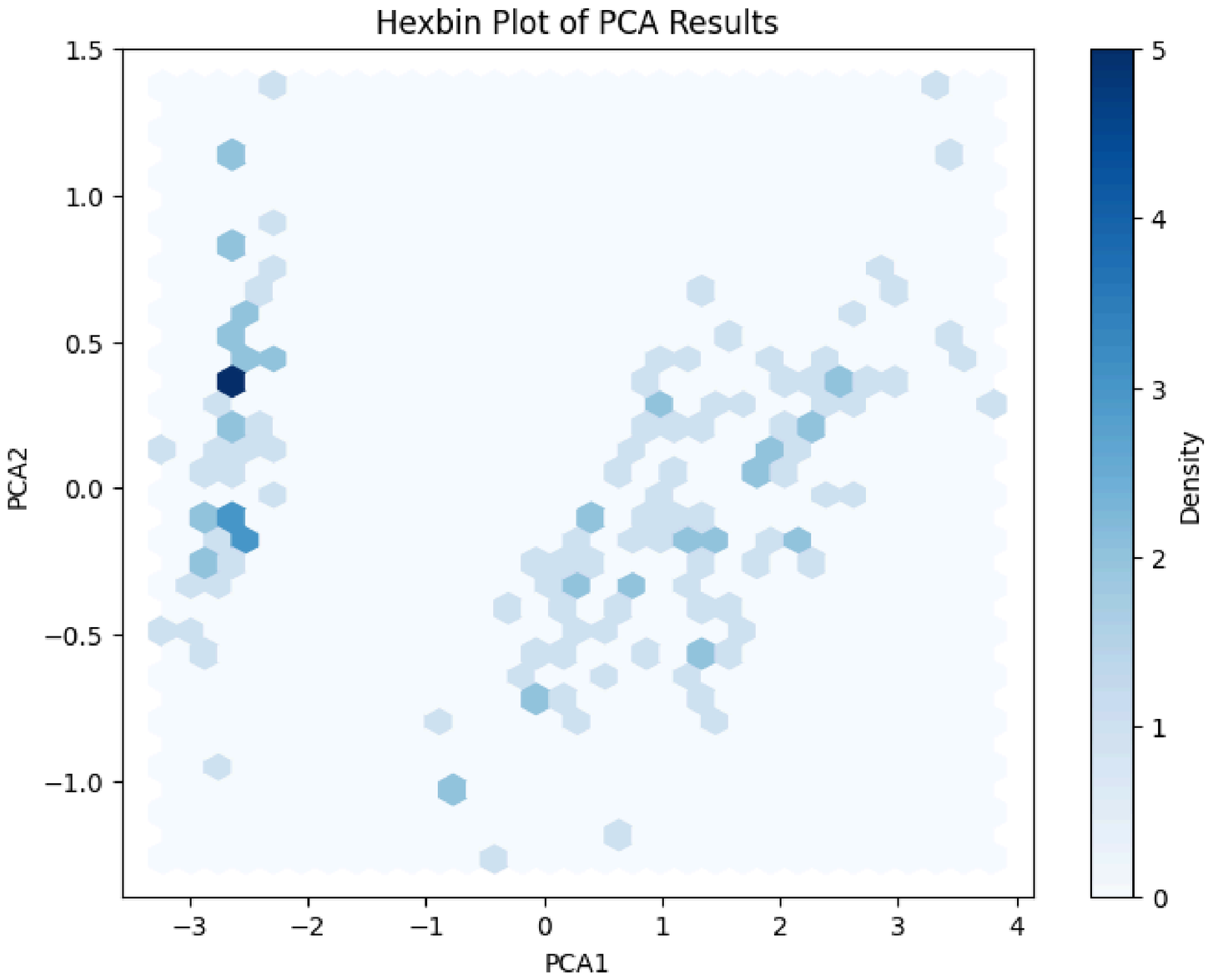


HEXBIN PLOT

Hexbin plots are used to visualise the density of points in 2D. Individual plots may lead to overplotting. Hexbin plots aggregate points within hexagonal bins and color them based on the number of points in each bin.

```
# Create a hexbin plot
plt.figure(figsize=(8, 6))
plt.hexbin(pca_result[:, 0], pca_result[:, 1], gridsize=30,
           cmap='Blues')
# Add a color bar to show the density scale
plt.colorbar(label='Density')
# Add labels and title
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.title('Hexbin Plot of PCA Results')
plt.show()
```

HEXBIN PLOT

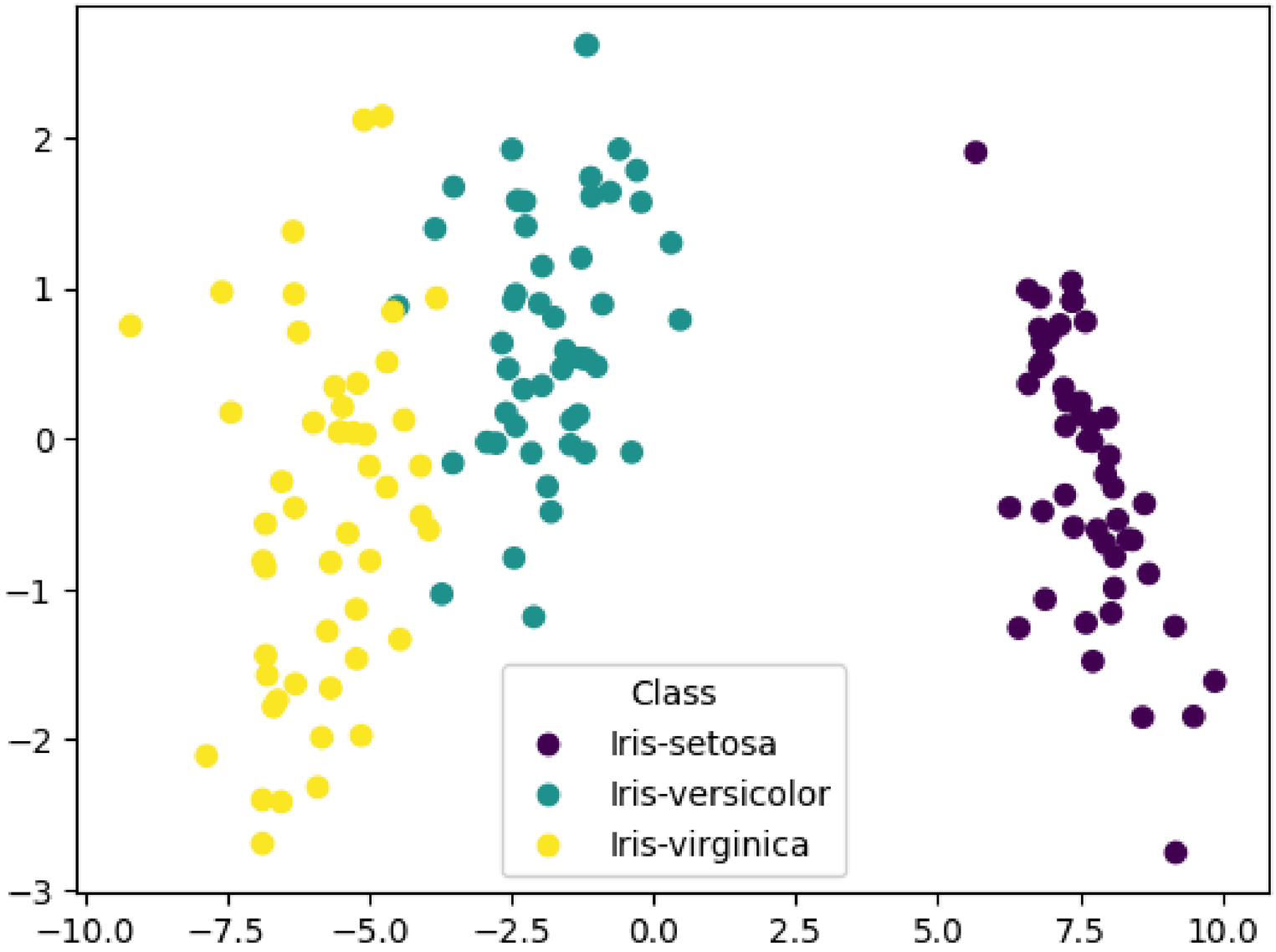

Hexbin Plot of PCA Results
PCA2
PCA1
Density
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4
5
4
3
2
1
0

LDA : LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis (LDA), also known as Normal Discriminant Analysis or Discriminant Function Analysis, is a dimensionality reduction technique primarily utilized in supervised classification problems. It facilitates the modeling of distinctions between groups, effectively separating two or more classes.

```
# Running LDA (linear discriminant analysis) on the data
lda = LDA(n_components=2)
lda_result = lda.fit_transform(iris[['sepal_length', 'sepal_width',
                'petal_length', 'petal_width']], iris['encoded_target'])
plt.scatter(lda_result[:, 0], lda_result[:, 1], c=iris['encoded_target'] )
# Create a legend by assigning labels manually
legend_labels = iris['class'].unique()
handles = scatter.legend_elements()[0]
plt.legend(handles, legend_labels, title="Class")
plt.show()
```

LDA

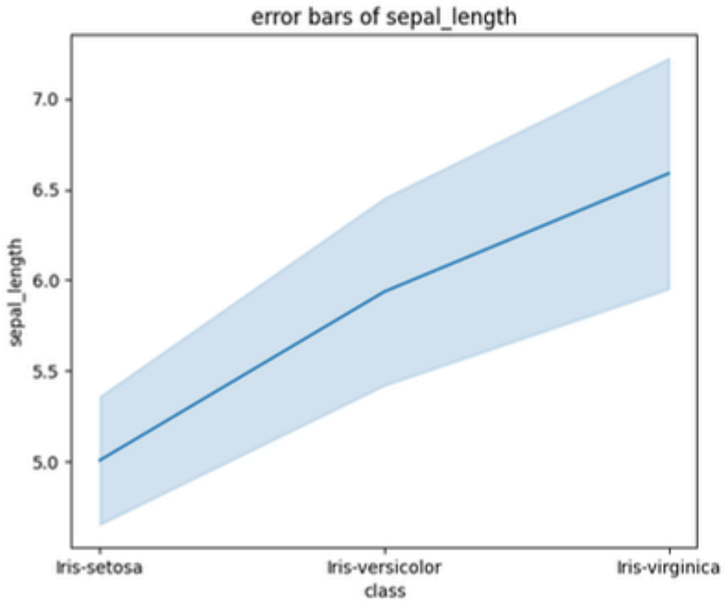

2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Class
Iris-setosa
Iris-versicolor
Iris-virginica

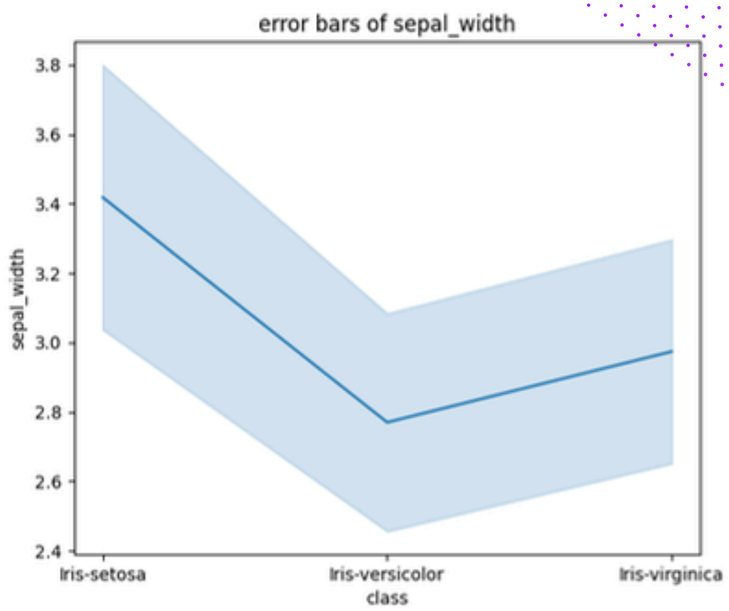
ERROR BARS

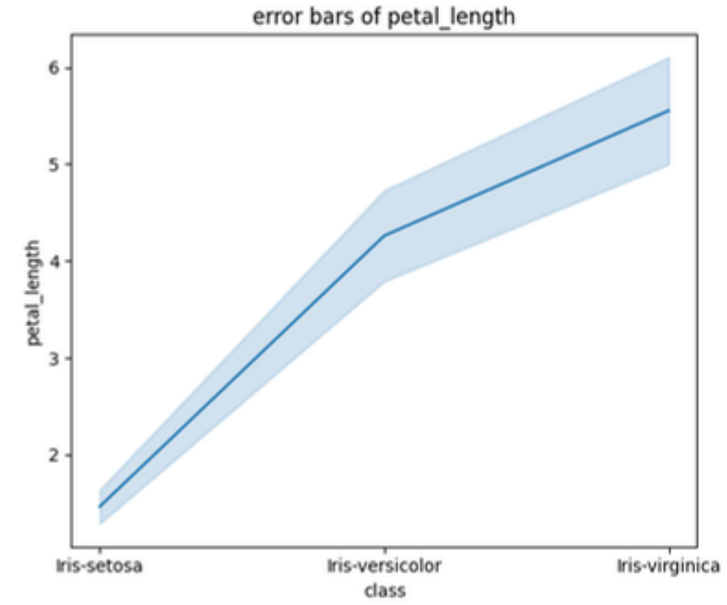
Error bars are used to visualise the variability of data in plots.

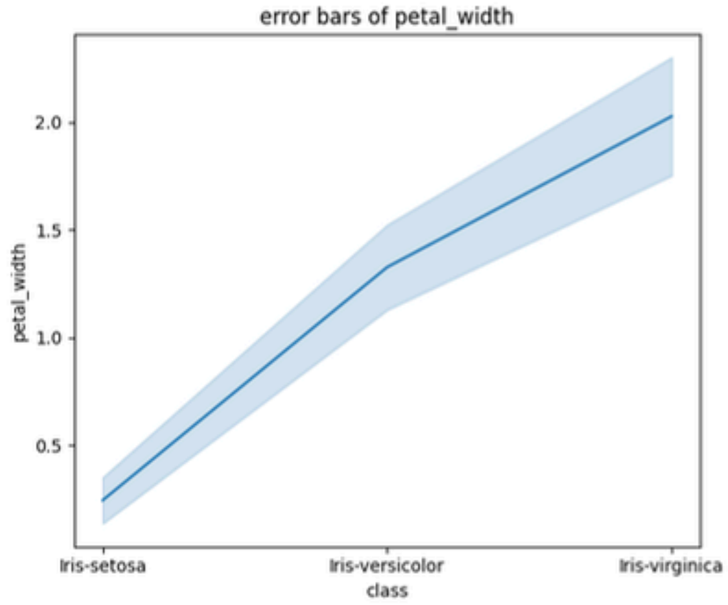
```
#errorbars of the features
features = ['sepal_length', 'sepal_width', 'petal_length',
            'petal_width']
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Create a error bars for each feature
for i, feature in enumerate(features):
            sns.lineplot(x='class', y=feature, data=iris, ax=axes[i//2, i%2],
                  errorbar= 'sd')
            axes[i//2, i%2].set_title(f'error bars of {feature}')

plt.tight_layout()
plt.show()
```

ERROR BARS


error bars of sepal_length
sepal_length
7.0
6.5
6.0
5.5
5.0
Iris-setosa
Iris-versicolor
Iris-virginica
class


error bars of sepal_width
sepal_width
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
Iris-setosa
Iris-versicolor
Iris-virginica
class


error bars of petal_length
petal_length
6
5
4
3
2
Iris-setosa
Iris-versicolor
Iris-virginica
class


error bars of petal_width
petal_width
2.0
1.5
1.0
0.5
Iris-setosa
Iris-versicolor
Iris-virginica
class

OBSERVATIONS

- We observe that Iris setosa’s features range distinctly from the other two species. While Iris versicolor and Iris verginica overlap in some features but on LDA we can observe that even versicolor and verginica also have quite distinct range.
- Petal length and petal width are most correlating with their class so they can be helpful in further classification.

FUTURE PLANS

Moving forward, we plan to apply Supervised Machine Learning techniques like SVM, K nearest neighbour, Decision Trees, Gaussian Naive Bayes, Neural Network, etc. so as to classify the three species based on the features given to us. We will make the model on training data which is the 70% of the complete dataset, and then evaluate the model on testing data. We will compare each model’s accuracy, recall, precision and F1 score so that the accuracy is more than 80%.

OTHER WORKS DONE ON IRIS DATASET

2023 3rd International Conference on Intelligent Technologies (CONIT)
Karnataka, India. June 23-25, 2023

Iris Flower Species Detection using Machine Learning Technique

Hemalatha D[1]
Department of CSE
Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
Avadi, Chennai, Tamil Nadu, India
hema24294@gmail.com

Yaramala Tarunkumar Reddy[2]
Department of CSE
Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
Avadi, Chennai, Tamil Nadu, India
yaramalatharunkumarreddy2000@gmail.com

Lakshmikanth K[3]
Department of CSE
Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
Avadi, Chennai, Tamil Nadu, India
lakshmikanthkari@gmail.com

2.2023.10205566

Iris Flower Species Detection using Machine Learning Technique

This paper presents machine learning techniques to classify iris flower species using Decision Trees, Gaussian Naive Bayes, Support Vector Machines, and k-nearest neighbor models. Their results show that the Gaussian Naive Bayes model performs the best, achieving an accuracy of 1.0 on the Iris dataset. We have found that the Gaussian Naive Bayes classifier gives the best accuracy compared to KNN, SVM, and Decision Tree models.

OTHER WORKS DONE ON IRIS DATASET

International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 1, February 2020, pp. 1079~1084
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i1.pp1079-1084

A new model for iris data set classification based on linear support vector machine parameter's optimization

Zahraa Faiz Hussain[1], Hind Raad Ibraheem[2], Mohammad Alsajri[3], Ahmed Hussein Ali[4], Mohd Arfian Ismail[5], Shahreen Kasim[6], Tole Sutikno[7]
[1,2,3,4]Computer Science Department, AL Salam University College, Iraq
[3,4]Department of Computer Science, College of Education, Al-Iraqia University, Iraq
[5]Faculty of Computer Systems & Software Engineering, Universiti Malaysia Pahang, Malaysia
[6]Faculty of Computer Science & Information Technology, Universiti Tun Hussein Onn Malaysia, Malaysia
[7]Department of Electrical and Computer Engineering, Universitas Ahmad Dahlan, Indonesia

A new model for iris data set classification based on linear support vector machine parameter's optimization

This paper presents the SVM technique to classify iris flower species a newly mode for classifying iris data set using SVM classifier and genetic algorithm to optimize c and gamma parameters of linear SVM, in addition principle components analysis (PCA) algorithm was use for features reduction.
The obtained results showed that the accuracy of the SVM increased to 98%.

OTHER WORKS DONE ON IRIS DATASET

Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

Flower Classification using Supervised Learning

Asmita Shukla
Computer Science and Engineering
Babu Banarasi Das National Institute of Technology and Management Lucknow, India

Ankita Agarwal
Computer Science and Engineering
Babu Banarasi Das National Institute of Technology and Management Lucknow, India

Hemlata Pant
Computer Science and Engineering
Babu Banarasi Das National Institute of Technology and Management Lucknow, India

Priyanka Mishra
Computer Science and Engineering
Babu Banarasi Das National Institute of Technology and Management Lucknow, India

The paper focuses on how Machine Learning algorithms can automatically recognize the class of flower with the help of high degree of accuracy rather than approximately.

The approach has been implementented in three phases: segmentation, feature extraction and classification. Using Neural Network, Logistic Regression, Support Vector Machine and k-Nearest Neighbors.

They found that SVM has most accuracy.

Table3. The accuracy of Classification model

	Algorithm used for Classification	Accuracy of Model
1.	Neural Network (NN)	96.6667%
2.	Logistic Regression (LR)	96.6667%
3.	Support Vector Machine (SVM)	98.00%
4.	k-nearest neighbors (k-NN)	96.67%

American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-6, Issue-1, pp-200-205
www.ajer.org
Research Paper Open Access

Simulation of Back Propagation Neural Network for Iris Flower Classification

R.A.Abdulkadir[1],Khalipha A. Imam [2] M.B. Jibril[3]
[1,3]*Electrical Engineering Department, Kano University of Science and TechnologyWudil*
[2]*Electrical Engineering Department, Kano State Polytechnic*

International Journal of All Research Education and Scientific Methods (IJARESM), ISSN: 2455-6211
Volume 9, Issue 6, June -2021, Impact Factor: 7.429, Available online at: www.ijaresm.com

Iris Flower Classification Using Machine Learning

T. Srinivas Rao[1], M. Hema[2], K. Sai Priya[3], K. Vamsi Krishna[4], M. Sakhavath Ali[5]

[1]M.Tech, PhD Professo
[2,3,4,5]Student, CSE

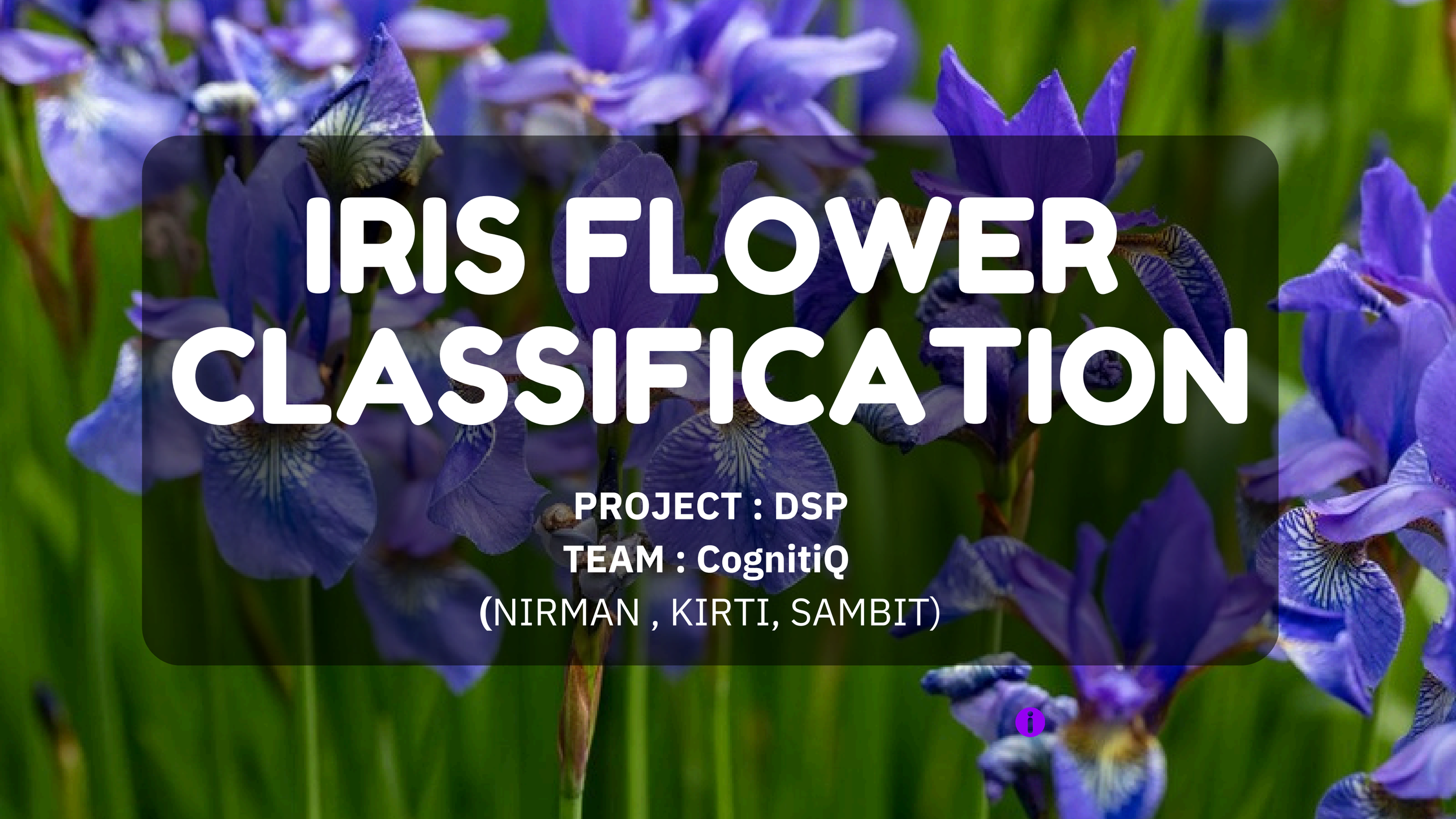
Advances in Artificial Intelligence Research (AAIR)
ISSN 2757-7422
www.dergipark.com/aair/

Vol. 2 (No. 1), pp. 7-14, 2022
doi: 10.54569/aair.1018444
Published online: Feb 16, 2022

Classification of Iris Flower by Random Forest Algorithm

Hilmi Cenk Bayrakçı [1,*], Abdullah Burak Keşkekçi [2], Recep Arslan [3]

[1] Isparta University of Applied Sciences, Dept. of Mechatronics Engineering, Isparta, Turkey;
[2] Isparta University of Applied Sciences, Dept. of Mechatronics Engineering, Isparta
[3] Isparta University of Applied Sciences, Dept. of Mechatronics Engineering, Isparta, Turkey;



IRIS FLOWER CLASSIFICATION

PROJECT : DSP

TEAM : CognitiQ

(NIRMAN , KIRTI, SAMBIT)

MENU

kNN classifier

Random forest

SVM

Decision tree

ANN

Logistic Regression

Gaussian Naive Bayes

Ensemble of Ensembles

Adaboost

Gradient Boost

XGBoost

CatBoost

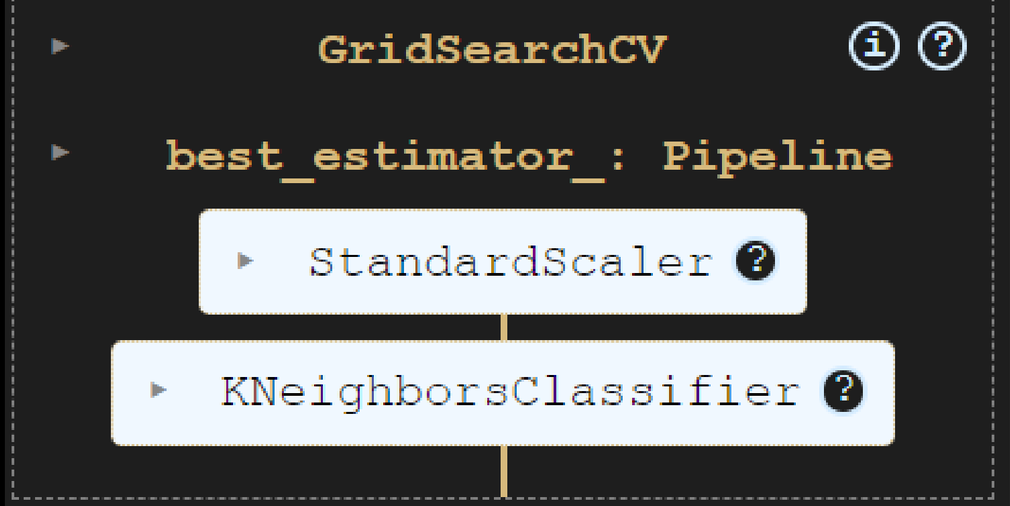
KNN CLASSIFIER

The K-NN algorithm works by finding the K nearest neighbors to a given data point based on a distance metric, such as Euclidean distance. The class or value of the data point is then determined by the majority vote or average of the K neighbors. This approach allows the algorithm to adapt to different patterns and make predictions based on the local structure of the data.

KNN and Standard Scaling: KNN is distance-based, and standard scaling typically benefits KNN by ensuring all features contribute equally to the distance calculation. However, if PCA is applied after scaling, the transformed features might not preserve the original relationships in the data, causing KNN to perform worse.

```
pipe = Pipeline([('std', StandardScaler()), ('knn', KNeighborsClassifier())],
                verbose = True)
pipe.fit(X,Y)
param_grid = {
    'knn__n_neighbors': [3, 5, 7, 9],      # Tuning the number of neighbors
    'knn__weights': ['uniform', 'distance'], # Weighting options
    'knn__metric': ['euclidean', 'manhattan'] # Distance metrics
}
gs_knn = GridSearchCV(pipe, param_grid, cv=5, scoring='accuracy')
gs_knn.fit(X, Y)
```

```
from sklearn.metrics import classification_report
y_pred = gs_knn.predict(x_test)
from sklearn.metrics import confusion_matrix
print("Confusion Matrix\n\n",confusion_matrix(y_test,y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



Confusion Matrix

```
 [[17  0  0]
 [ 0 11  0]
 [ 0  0 16]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17
1	1.00	1.00	1.00	11
2	1.00	1.00	1.00	16
accuracy			1.00	44
macro avg	1.00	1.00	1.00	44
weighted avg	1.00	1.00	1.00	44

Random Forest Classifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Trees in the forest use the best split strategy, i.e. equivalent to passing splitter="best" to the underlying DecisionTreeRegressor. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X,Y)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'max_features': ['sqrt', 'log2'],
    'min_samples_split': [2, 5, 10]
}
gs_rf = GridSearchCV(estimator=rf, param_grid=param_grid,
                     cv=5, scoring='accuracy')
gs_rf.fit(X, Y)
y_pred = gs_rf.predict(x_test)
print("Confusion Matrix\n\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix

 [[17  0  0]
 [ 0 10  1]
 [ 0  0 16]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      0.91      0.95        11
           2       0.94      1.00      0.97        16

    accuracy                           0.98        44
   macro avg       0.98      0.97      0.97        44
weighted avg       0.98      0.98      0.98        44
```

SVM

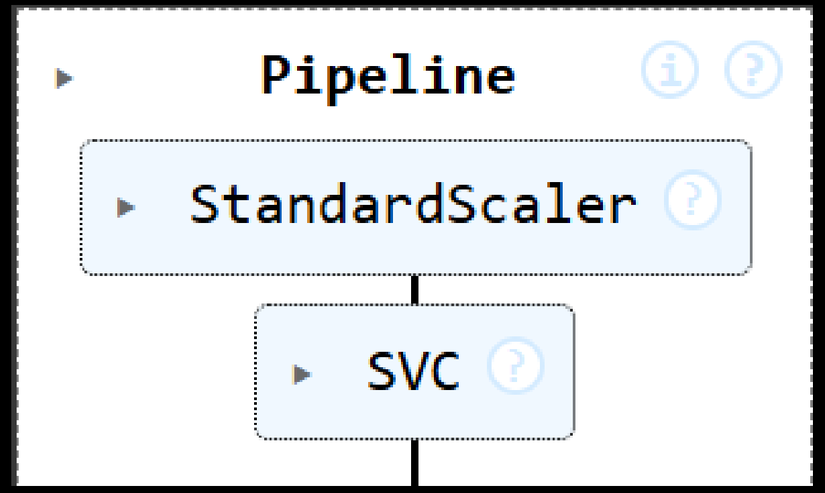
Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression. It finds the optimal hyperplane that maximizes the margin between different classes, effectively separating data points into distinct categories by maximizing the distance (or margin) from the nearest points (support vectors) in each class.

CODE

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([ ('std', StandardScaler()), ('svm', SVC())], verbose = True)

#fitting training_data into pipe
pipe.fit(x_train,y_train)

y_pred = pipe.predict(x_test)
y_pred
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report\n",classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.89      1.00      0.94         8
           2       1.00      0.92      0.96        13

    accuracy                           0.97        37
   macro avg       0.96      0.97      0.97        37
weighted avg       0.98      0.97      0.97        37
```



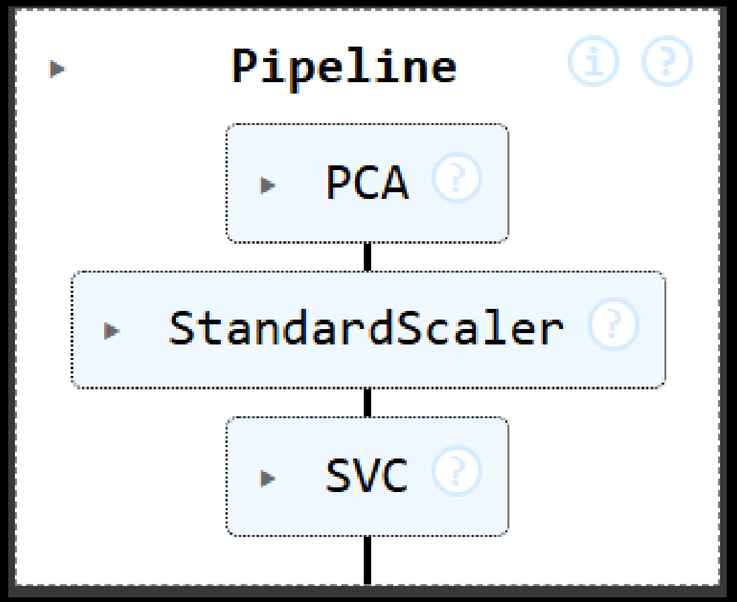
CODE (PCA)

```
pipe1 = Pipeline([('pca', PCA(n_components = 2)),('std', StandardScaler()), ('svm', SVC())], verbose = True)

#fitting training_data into pipe
pipe1.fit(x_train,y_train)
```

```
y_pred = pipe1.predict(x_test)
y_pred
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report\n",classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.80      1.00      0.89         8
           2       1.00      0.85      0.92        13

    accuracy                           0.95        37
   macro avg       0.93      0.95      0.94        37
weighted avg       0.96      0.95      0.95        37
```

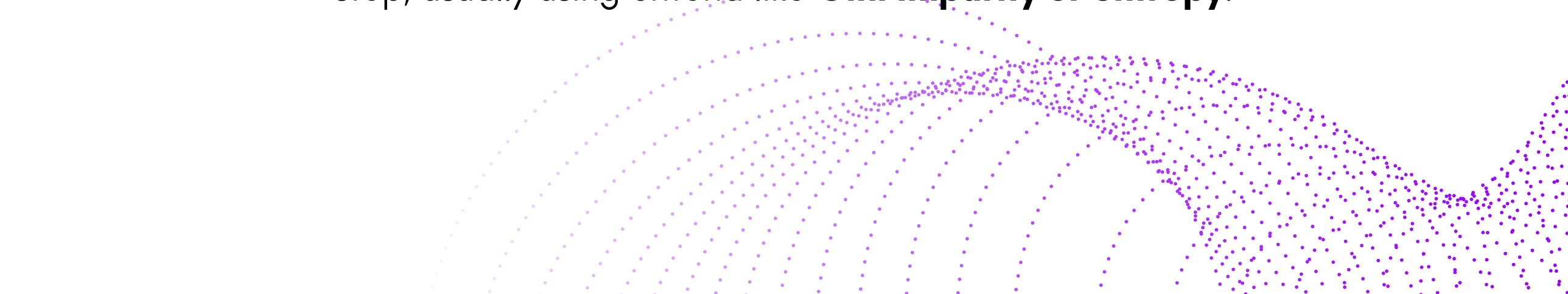


DECISION TREE

A Decision Tree is a supervised machine learning algorithm used for classification and regression. It splits the data into subsets based on feature values, creating a tree-like structure of decisions and outcomes.

Each internal node represents a feature, each branch represents a decision rule, and each leaf node represents a class label (for classification) or a value (for regression).

The tree is built by recursively choosing the feature that best splits the data at each step, usually using criteria like **Gini impurity or entropy**.



CODE

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([('decision_tree', DecisionTreeClassifier())], verbose = True)

#fitting training_data into pipe
pipe.fit(x_train,y_train)

decision_tree = pipe.named_steps['decision_tree']
```

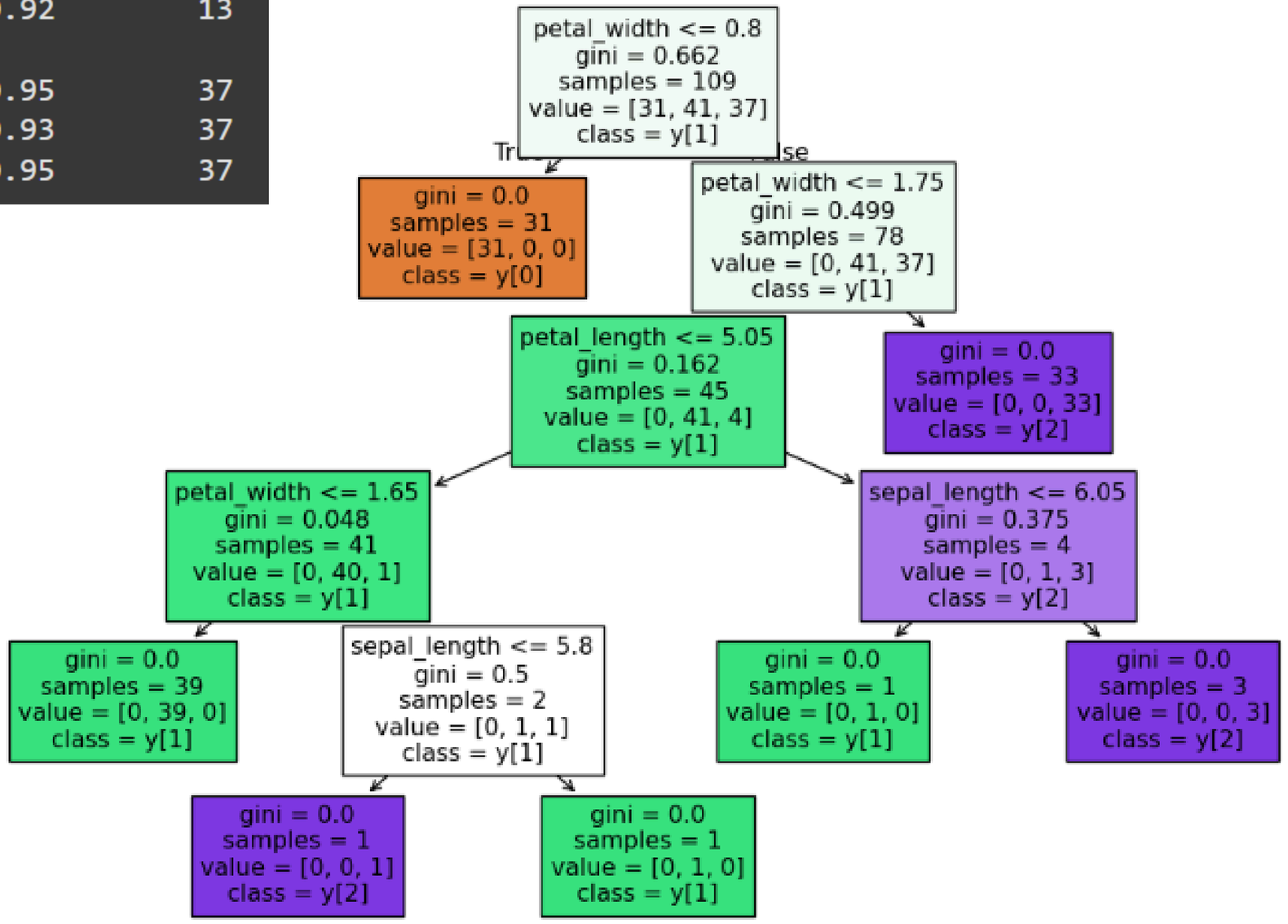
```
y_pred = pipe.predict(x_test)
y_pred
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report\n",classification_report(y_test, y_pred))
```

```
plt.figure(figsize=(12, 8))
plot_tree(decision_tree, filled=True, feature_names=x_train.columns, class_names=True)
plt.show()
```

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	16
1	0.88	0.88	0.88	8
2	0.92	0.92	0.92	13
accuracy			0.95	37
macro avg	0.93	0.93	0.93	37
weighted avg	0.95	0.95	0.95	37



GAUSSIAN NAIVE BAYES

Gaussian Naive Bayes is a probabilistic classifier based on Bayes' theorem, which assumes that the features are normally distributed (Gaussian distribution). It calculates the probability of each class given the input features and assigns the class with the highest probability. It is called "naive" because it assumes that features are independent, which is often not true in real-world data.

CODE

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([('naive_bayes', GaussianNB())], verbose = True)


#fitting training_data into pipe
pipe.fit(x_train,y_train)
```

```
y_pred = pipe.predict(x_test)
y_pred
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report\n",classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.88      0.88      0.88         8
           2       0.92      0.92      0.92        13

    accuracy                           0.95        37
   macro avg       0.93      0.93      0.93        37
weighted avg       0.95      0.95      0.95        37
```

ANN

Artificial Neural Networks contain artificial neurons which are called units . These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

```python
from sklearn.preprocessing import OneHotEncoder, StandardScaler

encoder = OneHotEncoder(sparse_output=False)
y_train = encoder.fit_transform(y_train)
y_test = encoder.fit_transform(y_test)
```

✓ 0.0s

```python
from keras.models import Sequential
from keras.layers import Dense

def create_ann_model():
    model = Sequential()
    model.add(Dense(3, input_dim = 4, activation='relu'))
    model.add(Dense(5,activation='relu'))
    model.add(Dense(3,activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 0.0s

```python
from scikeras.wrappers import KerasClassifier
# Wrapping the Keras model for compatibility with scikit-learn
ann_model = KerasClassifier(build_fn=create_ann_model, epochs=300, batch_size=8, verbose=0)
```

✓ 0.0s

```python
from sklearn.pipeline import Pipeline
# Define the pipeline
pipe = Pipeline([
    ('ann', ann_model)  # ANN model as the final classifier
], verbose=True)
```

✓ 0.0s

```python
pipe.fit(X_train,y_train)
```

```
Confusion Matrix

 [[12  0  0]
 [ 0  7  1]
 [ 0  0 10]]

Classification Report for Ensemble of Ensembles

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.88      0.93         8
           2       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30
```

LOGISTIC REGRESSION

Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. It’s referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

model = LogisticRegression(
    C=0.5,                # Regularization strength
    penalty='l2',         # Ridge regularization
    solver='lbfgs',       # Suitable for multiclass
    multi_class='multinomial',  # Multiclass classification
    max_iter=200,         # Increase iteration limit
    tol=1e-4              # Convergence tolerance
)
```

✓ 0.0s

```python
from sklearn.pipeline import Pipeline
# Define the pipeline
pipe = Pipeline([
    ('std', StandardScaler()),  # Standardization
    ('logistic regression', model)  # ANN model as the final classifier
], verbose=True)
```

✓ 0.0s

```python
pipe.fit(X_train,y_train)
```

✓ 0.0s

```
Confusion Matrix

 [[12  0  0]
 [ 0  7  1]
 [ 0  1  9]]

Classification Report for Logistic Regression

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.88      0.88      0.88         8
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.92      0.92      0.92        30
weighted avg       0.93      0.93      0.93        30
```

ENSEMBLES OF ENSEMBLES

Ensembles of ensembles, or "stacked" ensembles, combine multiple ensemble models (e.g., Random Forests, Gradient Boosting) to improve prediction performance further. A meta-model aggregates predictions from the base ensemble models to produce a final prediction. This approach helps capture diverse patterns in the data, often yielding highly accurate and robust models, especially in complex prediction tasks.

Code

```python
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline

# Create base learners (first layer of ensemble)
base_learners = [
    ('svm', Pipeline([('svm', SVC(probability=True))])),  # SVM with probability estimates for compatibility
    ('random_forest', Pipeline([('rf', RandomForestClassifier())])),  # Random Forest
    ('decision_tree', Pipeline([('dt', DecisionTreeClassifier())]))  # Decision Tree
]
```

✓ 0.0s

```python
from sklearn.ensemble import GradientBoostingClassifier
stacking_ensemble = StackingClassifier(
    estimators=base_learners,
    final_estimator=GradientBoostingClassifier()
)
```

✓ 0.0s

```python
# Train the model
stacking_ensemble.fit(X_train, y_train)

# Predict on the test set
y_pred_ensemble = stacking_ensemble.predict(X_test)
```

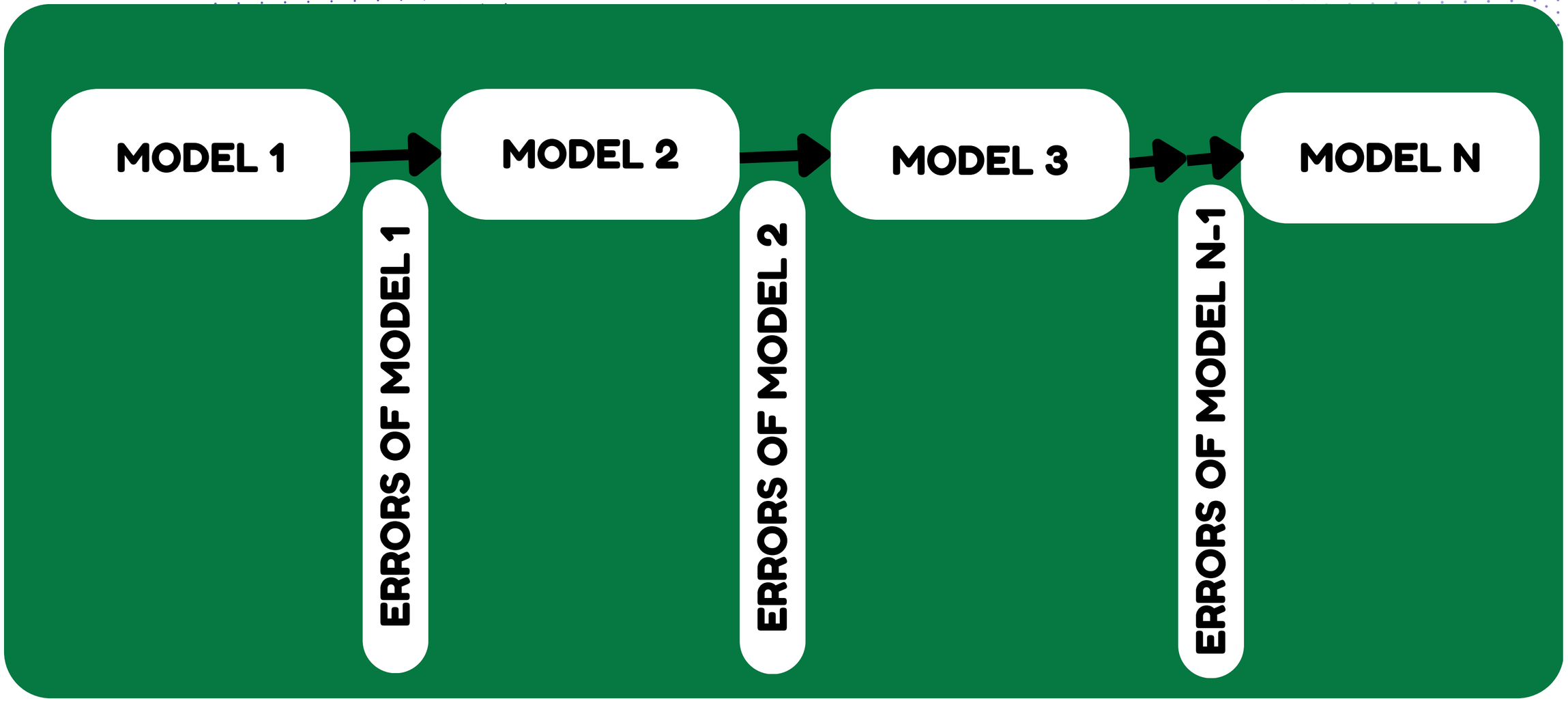
```
Confusion Matrix

 [[12  0  0]
 [ 0  7  1]
 [ 0  0 10]]

Classification Report for Ensemble of Ensembles

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.88      0.93         8
           2       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Boosting

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

Advantages of Boosting

- Improved Accuracy
- Robustness to Overfitting
- Better handling of imbalanced data
- Better Interpretability


MODEL 1
MODEL 2
MODEL 3
MODEL N
ERRORS OF MODEL 1
ERRORS OF MODEL 2
ERRORS OF MODEL N-1

AdaBoost

AdaBoost is one of the first boosting algorithms to have been introduced. It is mainly used for classification, and the base learner (the machine learning algorithm that is boosted) is usually a decision tree with only one level, also called as stumps.
It makes use of weighted errors to build a strong classifier from a series of weak classifiers.

It works in the following steps:

1. Initially, Adaboost **selects a training subset** randomly
2. It iteratively trains the AdaBoost machine learning model by selecting the training set **based on the accurate prediction of the last training**
3. It assigns the **higher weight to wrong classified observations** so that in the next iteration these observations will get the high probability for classification
4. Also, It assigns the **weight to the trained classifier** in each iteration **according to the accuracy of the classifier**. The more accurate classifier will get high weight
5. This process iterates until the complete training data fits without any error or until reached to the specified maximum number of estimators

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
```

```python
base_estimator = DecisionTreeClassifier(max_depth=1)  # Weak learner
ada = AdaBoostClassifier(base_estimator, n_estimators=50, learning_rate=1.0,
                         random_state=100)
ada.fit(X, Y)
```

```python
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1.0]
}
gs_ada = GridSearchCV(estimator=ada, param_grid=param_grid, cv=5,
                      scoring='accuracy')
gs_ada.fit(X,Y)
y_pred = gs_ada.predict(x_test)
print("Confusion Matrix\n\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

AdaBoostClassifier

```
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
                   random_state=100)
```

estimator: DecisionTreeClassifier

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=1)
```

```
[[17  0  0]
 [ 0 10  1]
 [ 0  0 16]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17
1	1.00	0.91	0.95	11
2	0.94	1.00	0.97	16
accuracy			0.98	44
macro avg	0.98	0.97	0.97	44
weighted avg	0.98	0.98	0.98	44

GRADIENT BOOST

Gradient Boosting is an ensemble learning method that builds a model in a sequential manner by combining weak learners (usually decision trees). Each new tree corrects the errors made by the previous ones by focusing on the residuals (the differences between predicted and actual values). The model is trained iteratively, with each new tree minimizing the loss function (e.g., mean squared error for regression) using gradient descent. Gradient Boosting is known for its high accuracy and ability to handle both classification and regression tasks. Variants of gradient boosting include popular implementations like XGBoost, LightGBM, and CatBoost.

CODE

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([('gbc', GradientBoostingClassifier())])

#fitting training_data into pipe
pipe.fit(x_train,y_train)
```

```
y_pred = pipe.predict(x_test)
y_pred
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report\n",classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.88      0.88      0.88         8
           2       0.92      0.92      0.92        13

    accuracy                           0.95        37
   macro avg       0.93      0.93      0.93        37
weighted avg       0.95      0.95      0.95        37
```

XGBOOST

XGBoost (Extreme Gradient Boosting) is a powerful and efficient implementation of gradient boosting for supervised learning tasks. It builds an ensemble of decision trees sequentially, where each tree corrects the errors of the previous one. XGBoost is known for its high performance, scalability, and regularization capabilities, making it particularly effective for large datasets and complex machine learning problems.

CODE

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([('xgboost', XGBClassifier())])

#fitting training_data into pipe
pipe.fit(x_train,y_train)
```

```
y_pred = pipe.predict(x_test)
y_pred
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report\n",classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.88      0.88      0.88         8
           2       0.92      0.92      0.92        13

    accuracy                           0.95        37
   macro avg       0.93      0.93      0.93        37
weighted avg       0.95      0.95      0.95        37
```

CATBOOST

Catboost is a variant of gradient boosting that can handle both categorical and numerical features. It does not require any feature encodings techniques like One-Hot Encoder or Label Encoder to convert categorical features into numerical features. It also uses an algorithm called symmetric weighted quantile sketch(SWQS) which automatically handles the missing values in the dataset to reduce overfitting and improve the overall performance of the dataset.

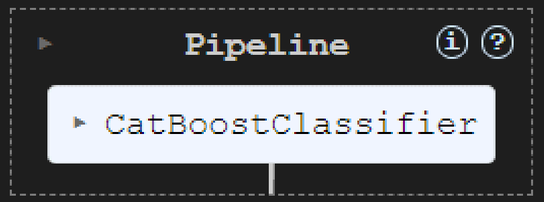
```python
# Define the pipeline
pipe_catboost = Pipeline([
    ('categorical boosting', CatBoostClassifier(verbose=0))
], verbose=True)
```

✓ 0.0s

```python
pipe_catboost.fit(X_train,y_train)
```

✓ 1.2s

```
[Pipeline]  (step 1 of 1) Processing categorical boosting, total=   1.2s
```




```python
y_pred_catboost = pipe_catboost.predict(X_test)
```

✓ 0.0s

```
Confusion Matrix

 [[12  0  0]
 [ 0  7  1]
 [ 0  1  9]]

Classification Report for Logistic Regression

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.88      0.88      0.88         8
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.92      0.92      0.92        30
weighted avg       0.93      0.93      0.93        30
```

MODELS	ACCURACY
kNN classifier	1.00
Adaboost	0.98
Random forest	0.98
SVM	0.97
Decision tree	0.95
Gaussian Naive Bayes	0.95
XGBoost	0.95
Gradient Boost	0.95
Logistic Regression	0.93
CatBoost	0.93
Ensemble of Ensembles	0.97
ANN	0.97

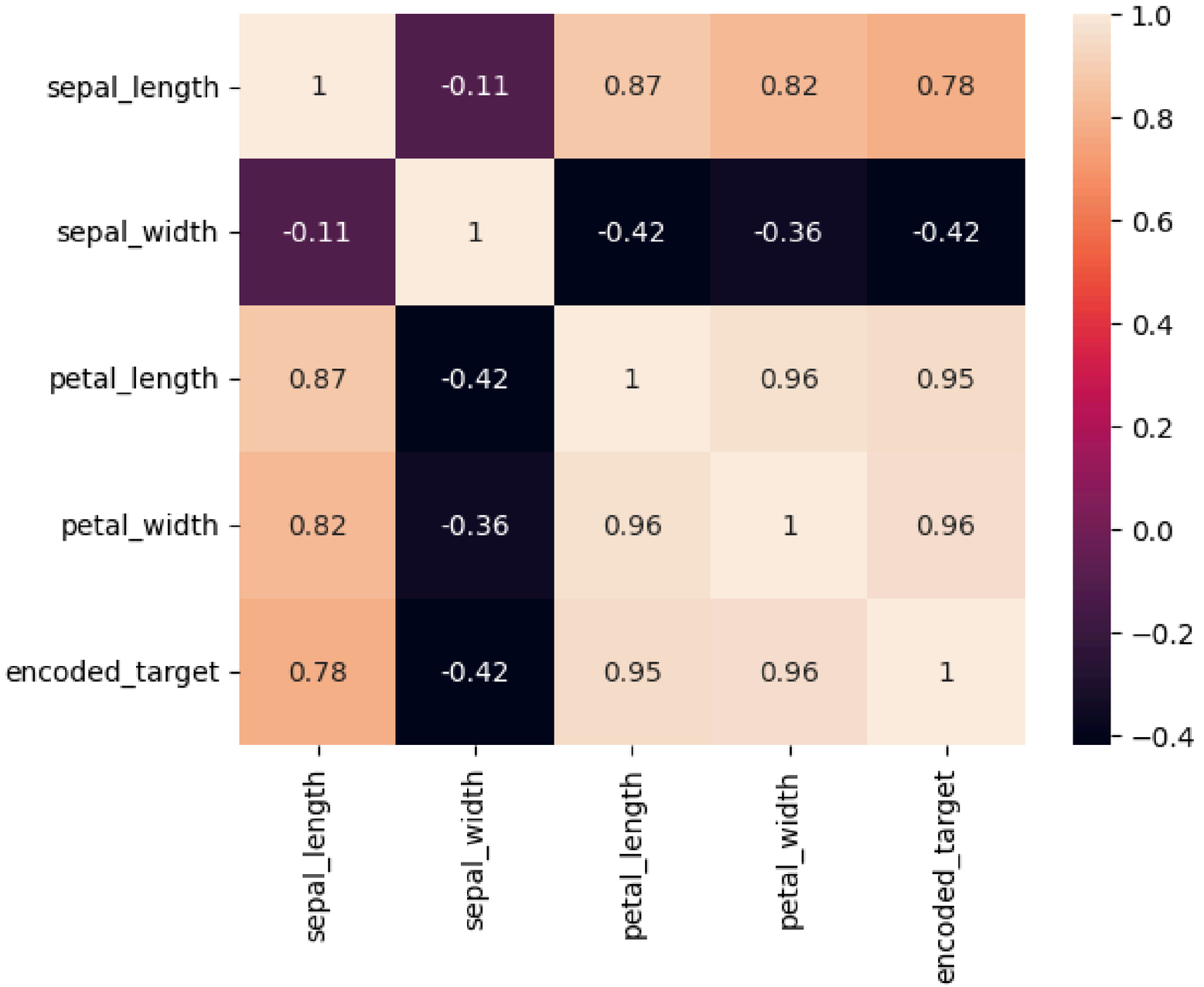
RESULTS

We obseved that kNN classifier is the best classifier for iris dataset as it showed accuracy of 1.

CORRELATION MATRIX

The correlation matrix is a matrix that shows the correlation between variables. It gives the correlation between all the possible pairs of values in a matrix format.

```
iris_numeric = iris.select_dtypes(
        include=[float,int])
cor_mat = iris_numeric.corr()
g2 = sns.heatmap(cor_mat,
        annot = True)
plt.show()
```



```
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
for i in range(0,3):
    for j in range(1,4):
        x_updated = X[[features[i],features[j]]]
        xtest_updated = x_test[[features[i],features[j]]]
        pipe = Pipeline([('std', StandardScaler()), ('knn', KNeighborsClassifier())],
                        verbose = True)
        pipe.fit(x_updated,Y)
        param_grid = {
            'knn__n_neighbors': [3, 5, 7, 9],      # Tuning the number of neighbors
            'knn__weights': ['uniform', 'distance'], # Weighting options
            'knn__metric': ['euclidean', 'manhattan'] # Distance metrics
        }
        gs_knn = GridSearchCV(pipe, param_grid, cv=5, scoring='accuracy')
        gs_knn.fit(x_updated, Y)
        y_pred = pipe.predict(xtest_updated)
        print(features[i],features[j])
        print("Confusion Matrix\n\n",confusion_matrix(y_test,y_pred))
        print("Classification Report:")
        print(classification_report(y_test, y_pred))
```

RESULTS

We observed that in all the combinations of two features sepal_length and sepal_width is least accurate which was estimated by the correlation matrix.

FEATURES	ACCURACY
sepal_length, sepal_width	0.80
sepal_length, petal_length	0.98
sepal_length, petal_width	0.98
sepal_width, petal_length	0.98
sepal_width, petal_width	0.95
petal_length, petal_width	0.98

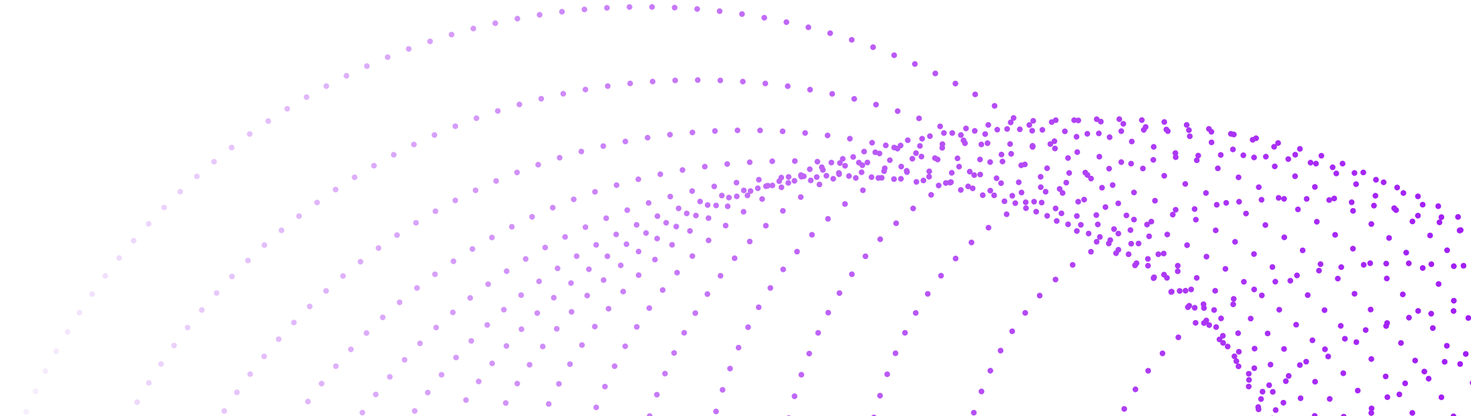
RESULTS

I checked if we can classify dataset by even a single feature. this shows that sepal length and sepal width have very less accuracy compared to the petal width and petal length.

FEATURES	ACCURACY
sepal_length	0.77
petal_length	0.98
petal_width	1.00
sepal_width	0.55

REFERENCES

- Flower classification using supervised learning A Shukla, A Agarwal, H Pant, P Mishra - Int. J. Eng. Res, 2020
- Iris Flower Classification Using Machine Learning T. Srinivas Rao1, M. Hema2, K. Sai Priya3, K. Vamsi Krishna4, M. Sakhavath Ali5 D, Hemalatha et al. “Iris Flower Species Detection using Machine Learning Technique.” *2023 3rd International Conference on Intelligent Technologies (CONIT)* (2023): 1-4.
- A new model for iris data set classification based on linear support vector machine parameter's optimization Zahraa Faiz Hussain1, Hind Raad Ibraheem2, Mohammad Alsajri3, Ahmed Hussein Ali4, Mohd Arfian Ismail5, Shahreen Kasim6, Tole Sutikno
- MachineLearningApproachtoImproveFlower ClassificationUsingMultipleFeatureSet, Kishotha, S, B.Mayurathan
- https://www.geeksforgeeks.org/
- https://scikit-learn.org/stable/modules/generated
- https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd
- https://scholar.google.com/
- https://www.google.co.in/
- https://openai.com/chatgpt/



Iris-CV: Classifying Iris Flowers Is Not as Easy as You Thought

Itamar de Paiva Ro celos Teixeira[1],
João Walla ousa[1],
Ana Clara Cha Manuel Ferreira ís Ramos[2],
Cecília Silva[3], T encio do Rego, Yuri Malheiros[1],
nd Teixeira Filho[1]

[1] Un deral da P João Pesso l
[2] Univ lo Rio Grande do Nort zil
[3] deral de Pernambuco, Re