Introduction

The Iris dataset, introduced by Ronald A. Fisher in 1936, is a classic dataset in machine learning. It consists of 150 samples of iris flowers, each belonging to one of three species: setosa, versicolor, and virginica. The dataset's simplicity lies in its four features—sepal length, sepal width, petal length, and petal width—measured in centimeters. Widely used for pattern recognition and classification tasks, the iris dataset serves as a foundational tool for exploring and evaluating machine learning algorithms, making it a standard reference in both educational and research contexts.

In [545...

```
# iris dataset
```

1 import Necessary Library

In [546...

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

2 import Dataset

In [547...

```
df = pd.read_csv("/kaggle/input/iris-data/iris.csv")
```

3 Data Analysis

In [548...

```
df.head()
```

Out[548...

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [549…

```
df.tail()
```

Out[549…

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

In [550…

```
df.shape
```

Out[550…

```
(150, 5)
```

In [551…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [552…

```
df.dtypes
```

Out[552…

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

In [553…

```
df.describe()
```

Out[553…

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [554…

```
df.corr
```

Out[554…

```
<bound method DataFrame.corr of      sepal_length  sepal_width  petal_length  pe
tal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]>
```

In [555…

```
df.ndim
```

Out[555…

```
2
```

In [556…

```
df.columns
```

Out[556…

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [557…

```
df["species"].value_counts()
```

Out[557…

```
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

4 Data cleaning and Preprocessing:

In [558…

```
df.isnull().sum()
```

Out[558…

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [559…

```
df['species'].value_counts()
```

Out[559…

```
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

In [560…

```
df.species.replace(['setosa', 'versicolor', 'virginica'], [0, 1, 2], inplace=Tr
```

In [561…

```
df.head()
```

Out[561…

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [562…

```
df['species'].value_counts()
```

Out[562…

```
species
0    50
1    50
2    50
Name: count, dtype: int64
```

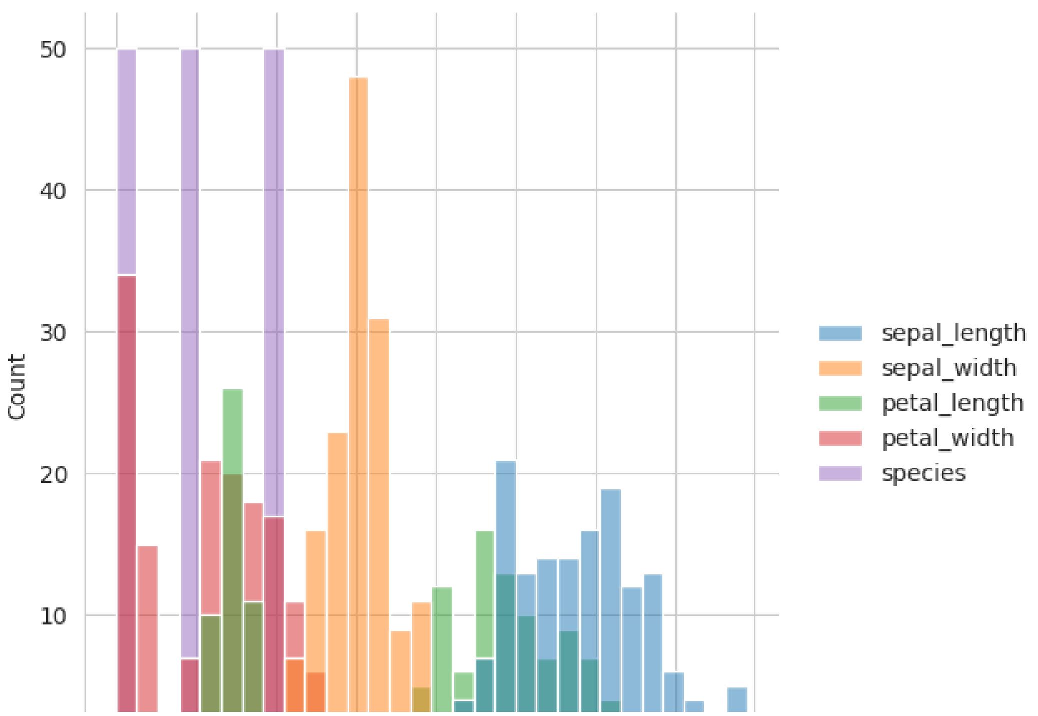
5| Data visualisation 📊 📉

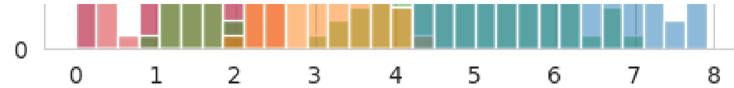
EDA (Exploratory Data Analysis)

5.1 displot

In [563…

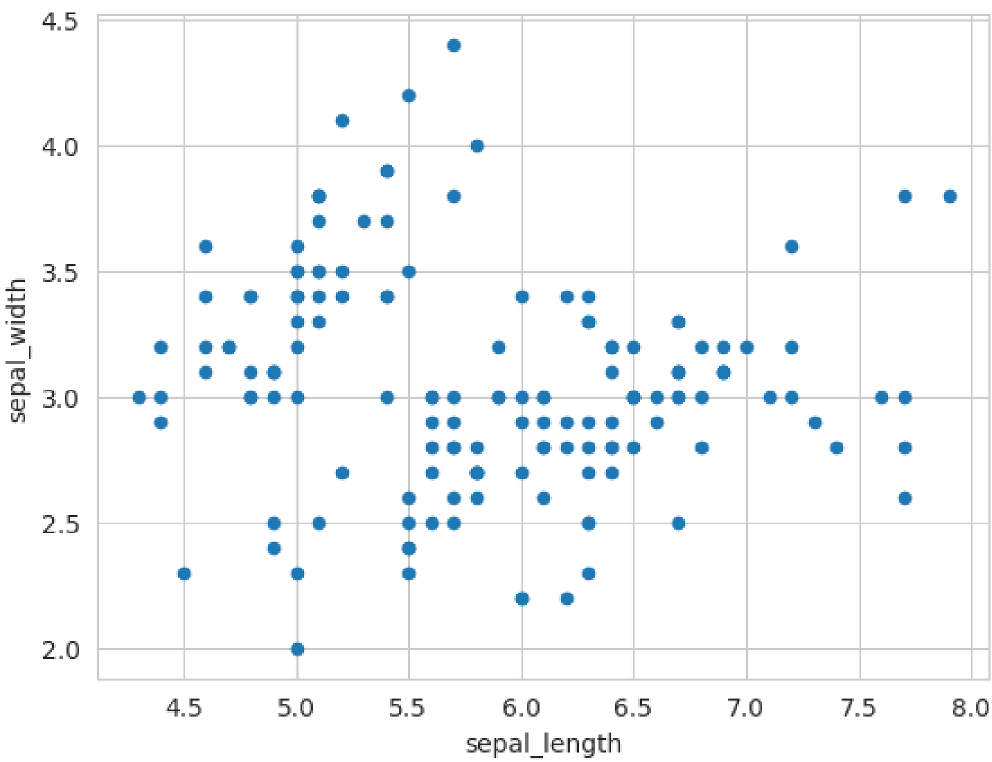
```
sns.displot(df, kde=False, bins=30)
plt.show()
```





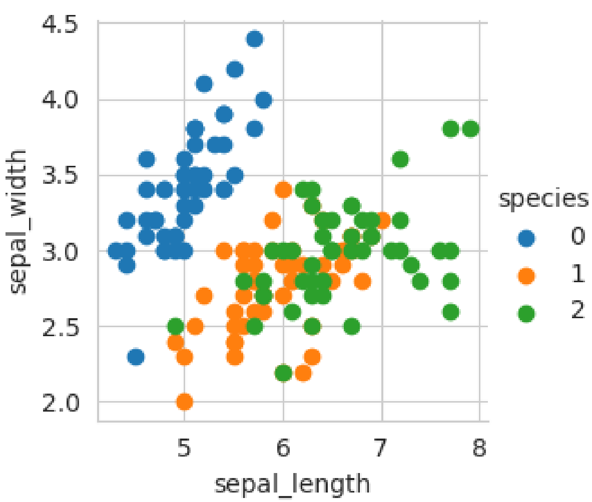
In [564…

```
df.plot(kind='scatter', x='sepal_length', y='sepal_width')
plt.show()
```



In [565…

```
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="species").map(plt.scatter, "sepal_length", "sepal_width"
plt.show();
```



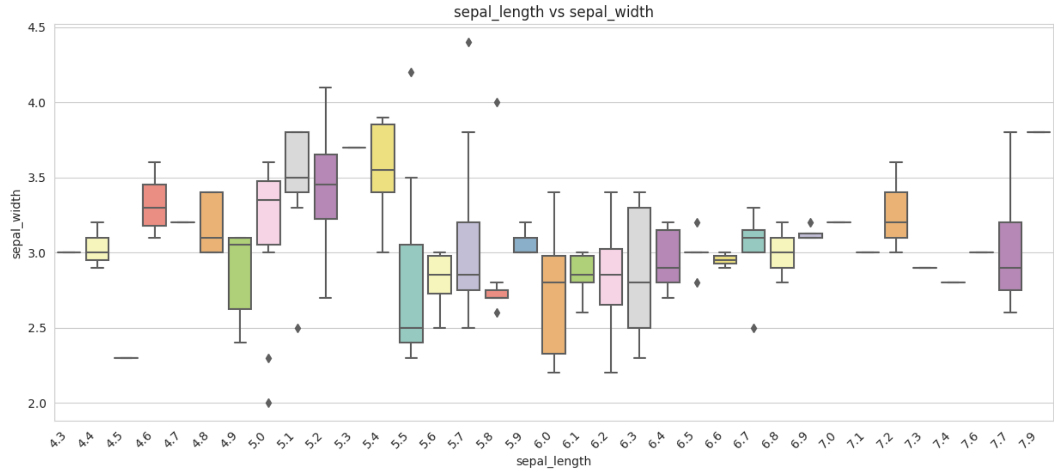
5.2 BoxPlot

In [566…

```
# sepal_length vs sepal_width boxplot

plt.figure(figsize=(15, 6))
```

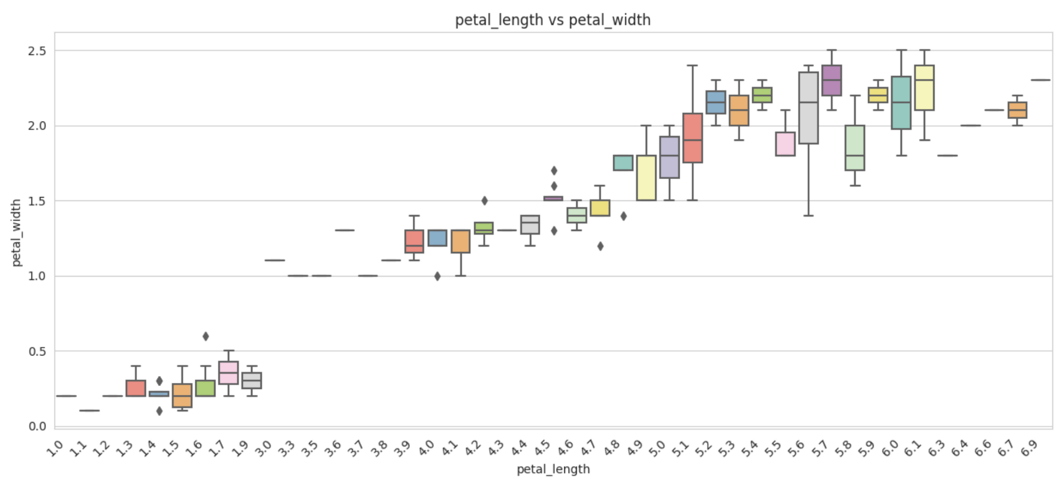
```python
sns.boxplot(x='sepal_length', y='sepal_width', data=df, palette='Set3')
plt.title('sepal_length vs sepal_width')
plt.xlabel('sepal_length')
plt.ylabel('sepal_width')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [567…

```python
# petal_length vs petal_width boxplot

plt.figure(figsize=(15, 6))
sns.boxplot(x='petal_length', y='petal_width', data=df, palette='Set3')
plt.title('petal_length vs petal_width')
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plt.xticks(rotation=45, ha='right')
plt.show()
```

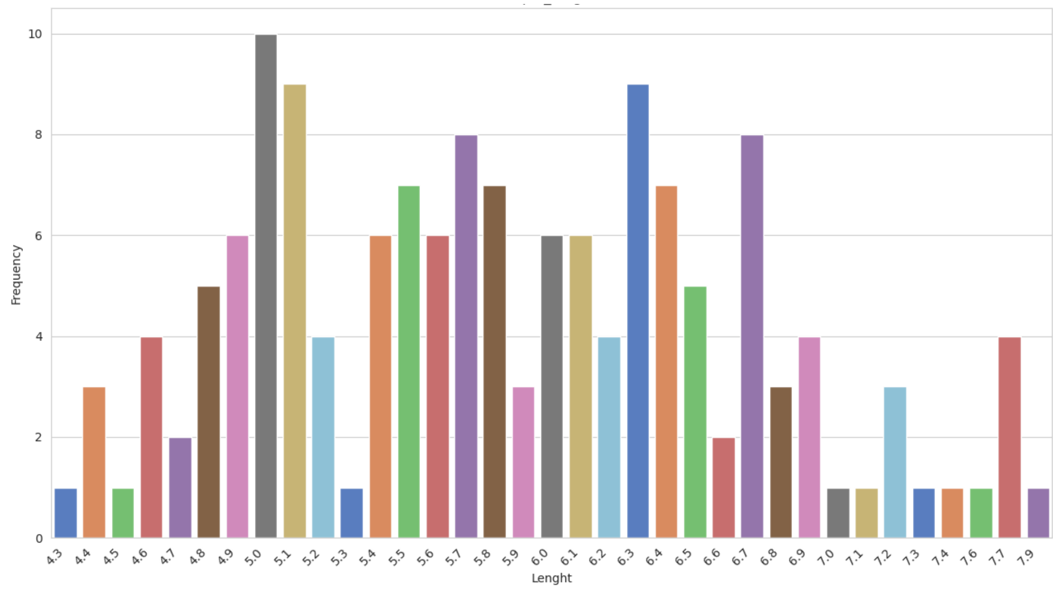


5.3 countplot

In [568…

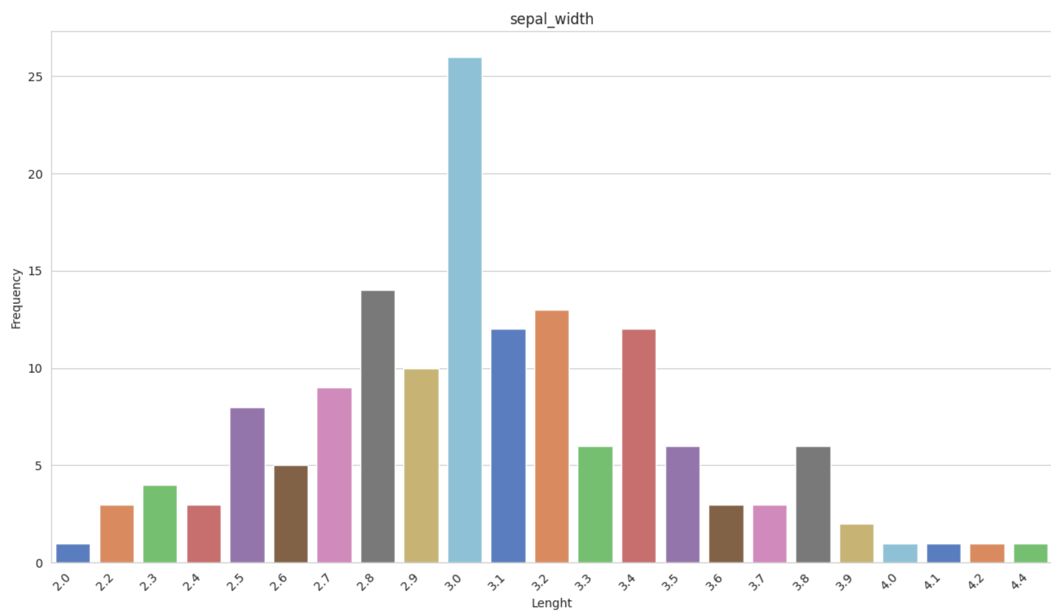
```python
# sepal_length
plt.figure(figsize=(15, 8))
sns.countplot(x='sepal_length', data=df, palette='muted')
plt.title('sepal_length')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```

sepal_length



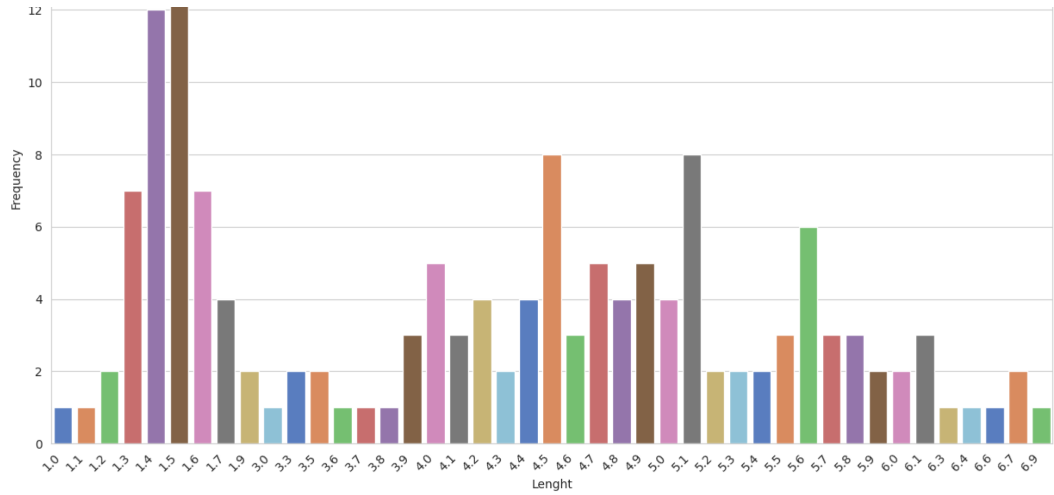
In [569…

```python
# sepal_width
plt.figure(figsize=(15, 8))
sns.countplot(x='sepal_width', data=df, palette='muted')
plt.title('sepal_width')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```



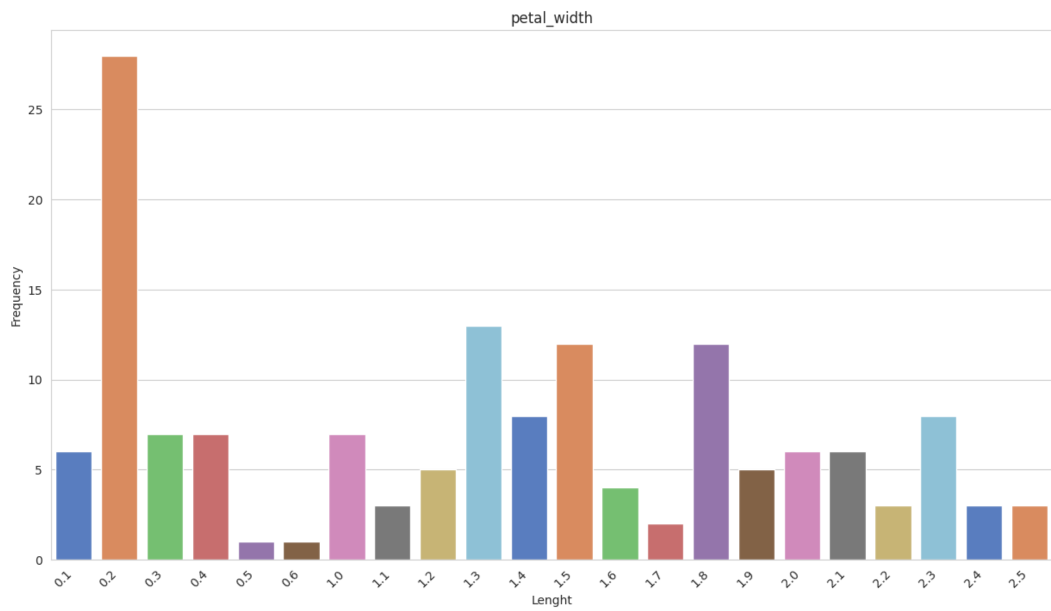
In [570…

```python
# petal_length
plt.figure(figsize=(15, 8))
sns.countplot(x='petal_length', data=df, palette='muted')
plt.title('petal_length')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [571…

```
# sepal_width
plt.figure(figsize=(15, 8))
sns.countplot(x='petal_width', data=df, palette='muted')
plt.title('petal_width')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```

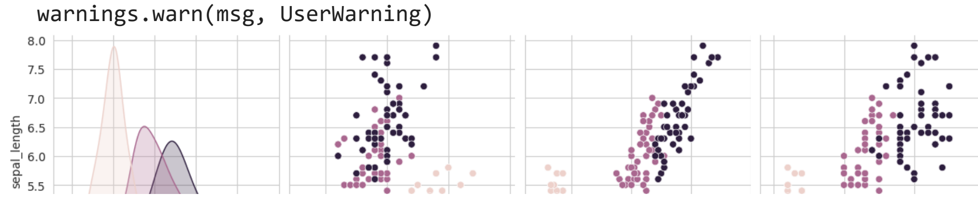


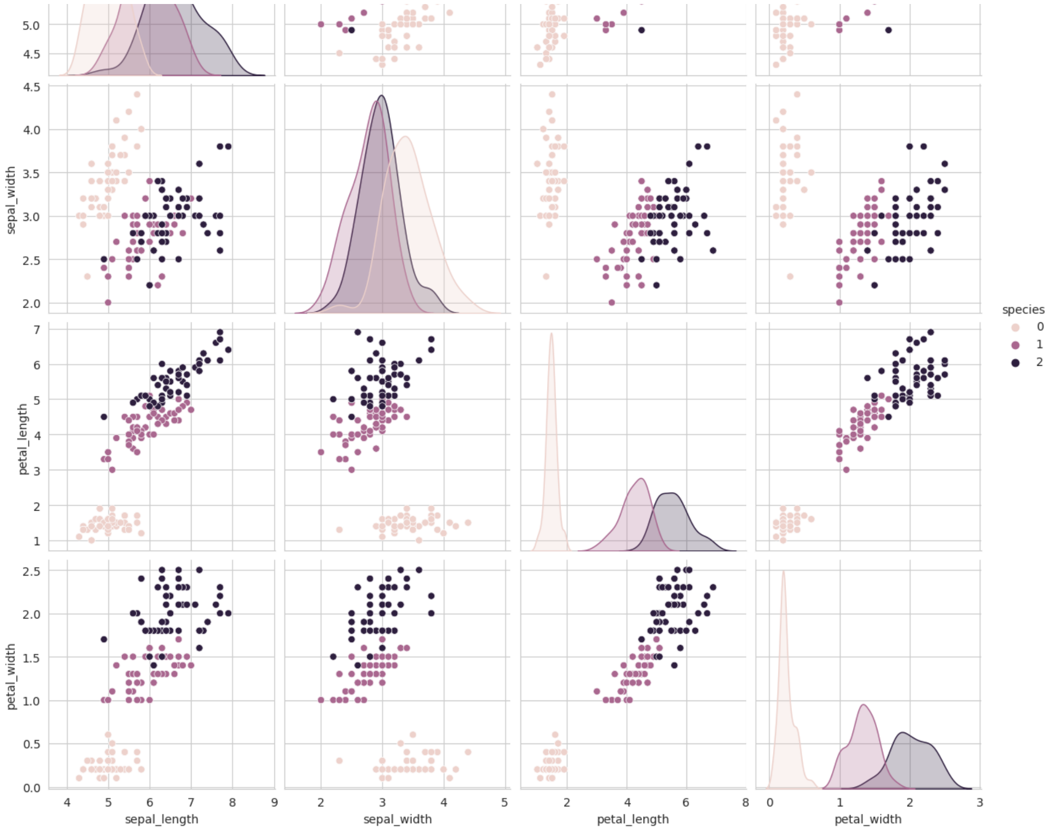
5.4 pairplot

In [667…

```
sns.set_style("whitegrid")
sns.pairplot(df, hue="species", size=3)
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:2095: UserWarning: Th
e `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

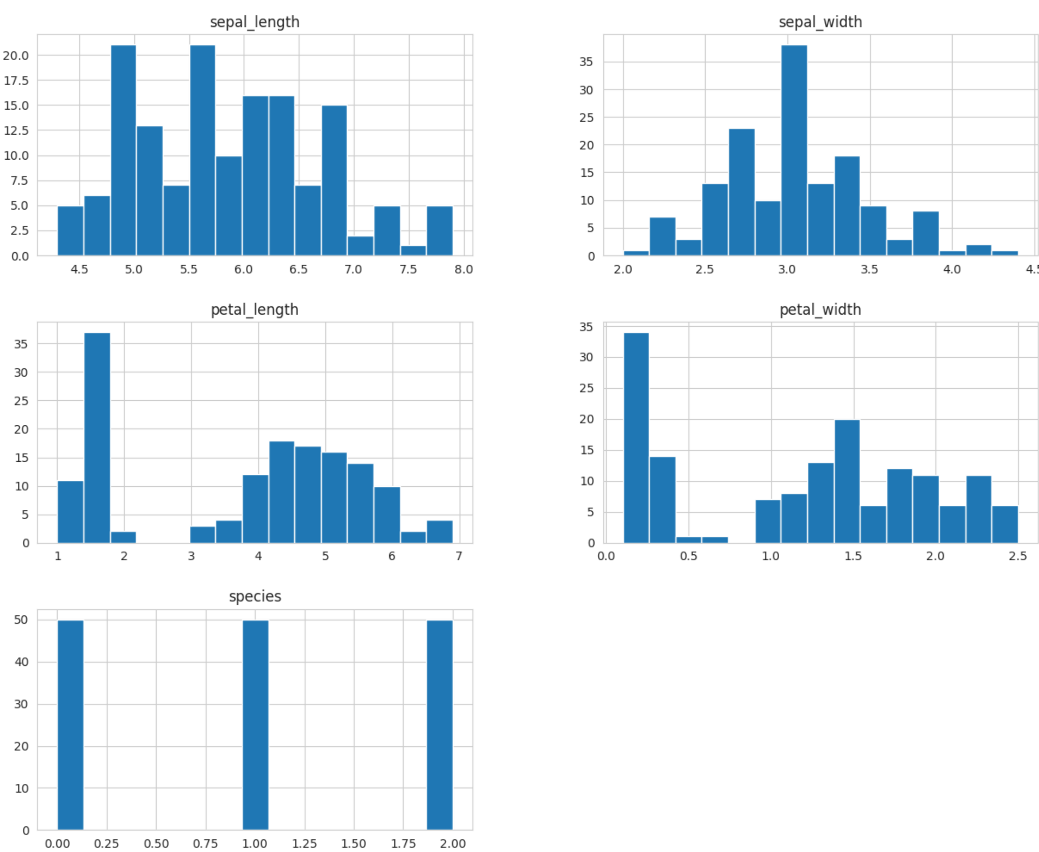




5.5 hist Plot

In [573…

```
df.hist(figsize=(15,12),bins = 15)
plt.title("Features Distribution")
plt.show()
```



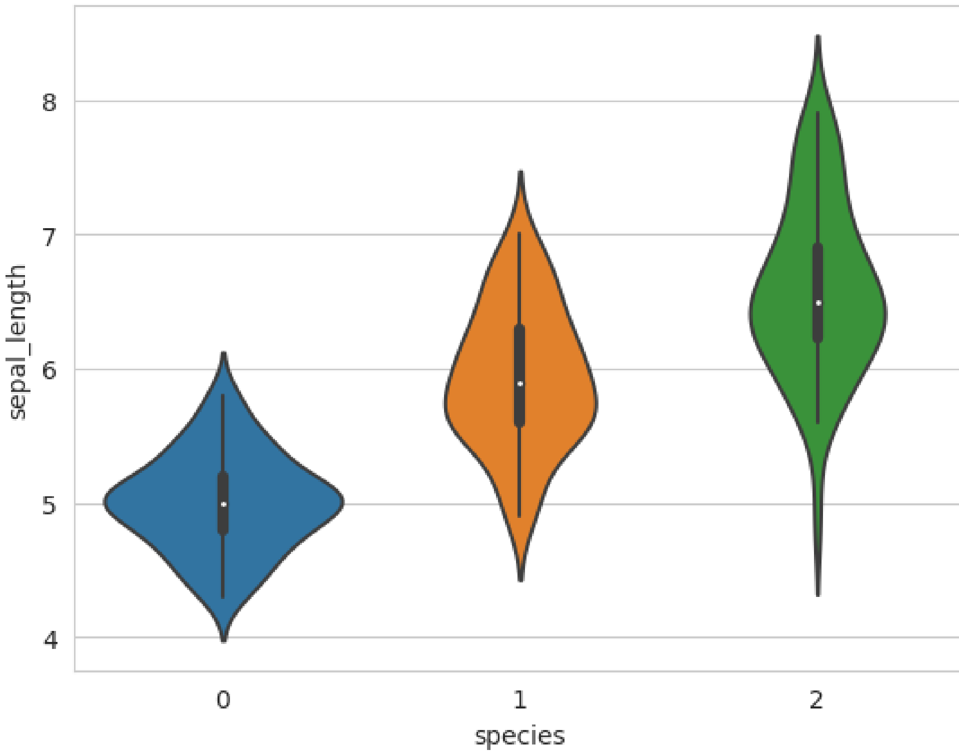
5.6 violinplot

In [574…

```
sns.violinplot(x="species",y="sepal_length", data=df, size = 8)
```

Out[574…

```
<Axes: xlabel='species', ylabel='sepal_length'>
```

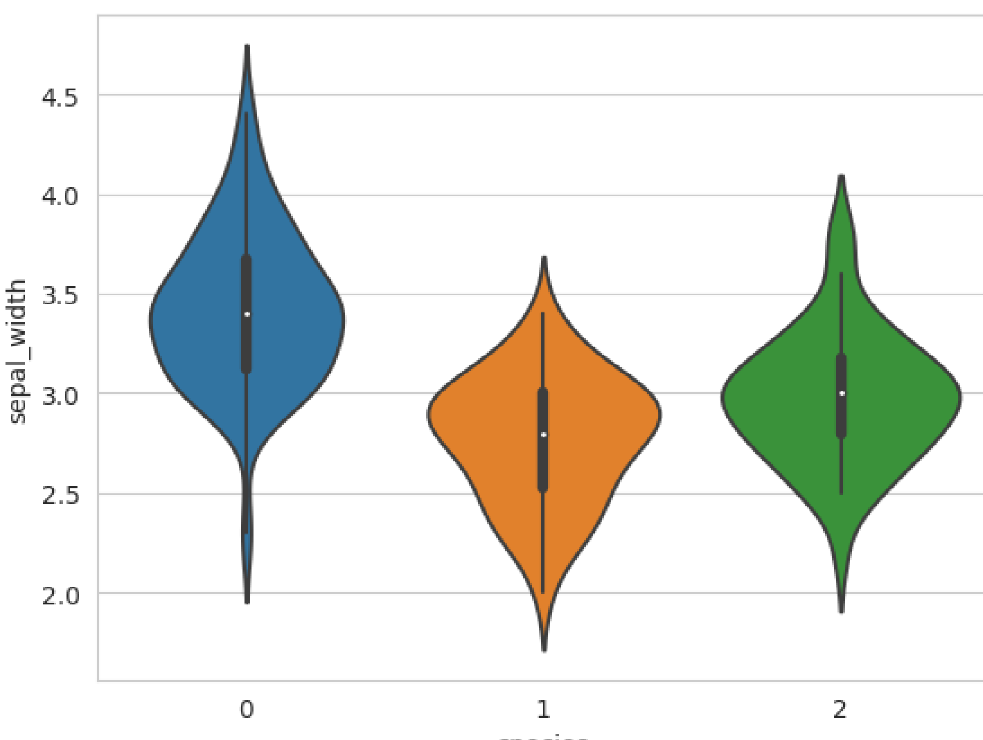


In [575…

```
sns.violinplot(x="species",y="sepal_width", data=df, size = 8)
```

Out[575…

```
<Axes: xlabel='species', ylabel='sepal_width'>
```

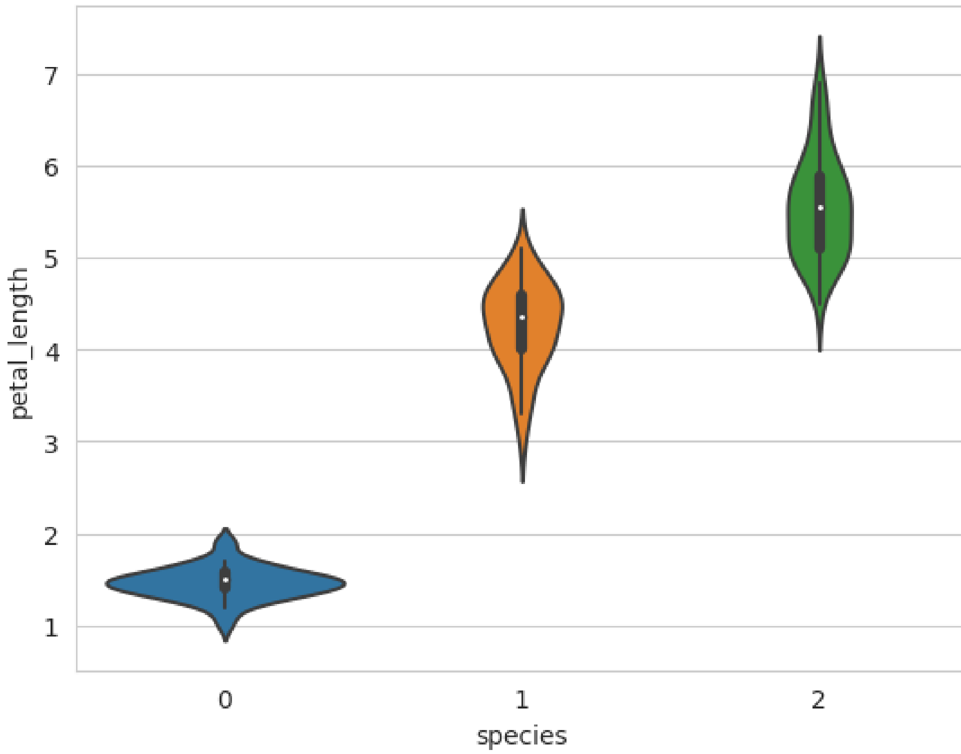


In [576…

```
sns.violinplot(x="species",y="petal_length", data=df, size = 8)
```

Out[576…

```
<Axes: xlabel='species', ylabel='petal_length'>
```

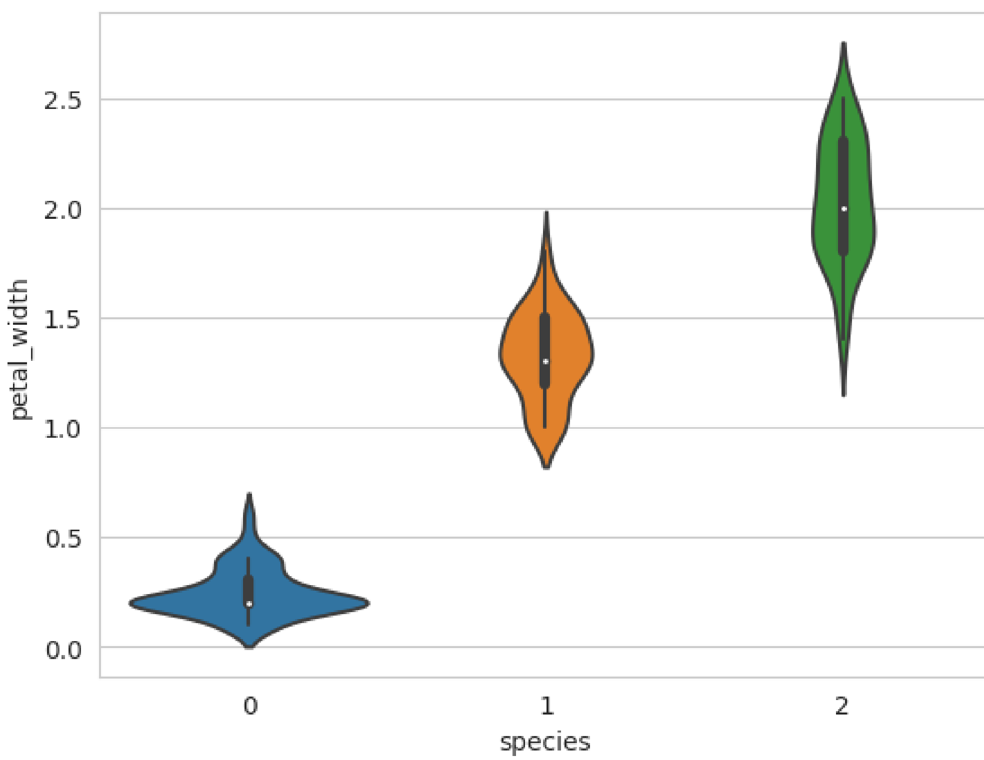


In [577…

```
sns.violinplot(x="species",y="petal_width", data=df, size = 8)
```

Out[577…

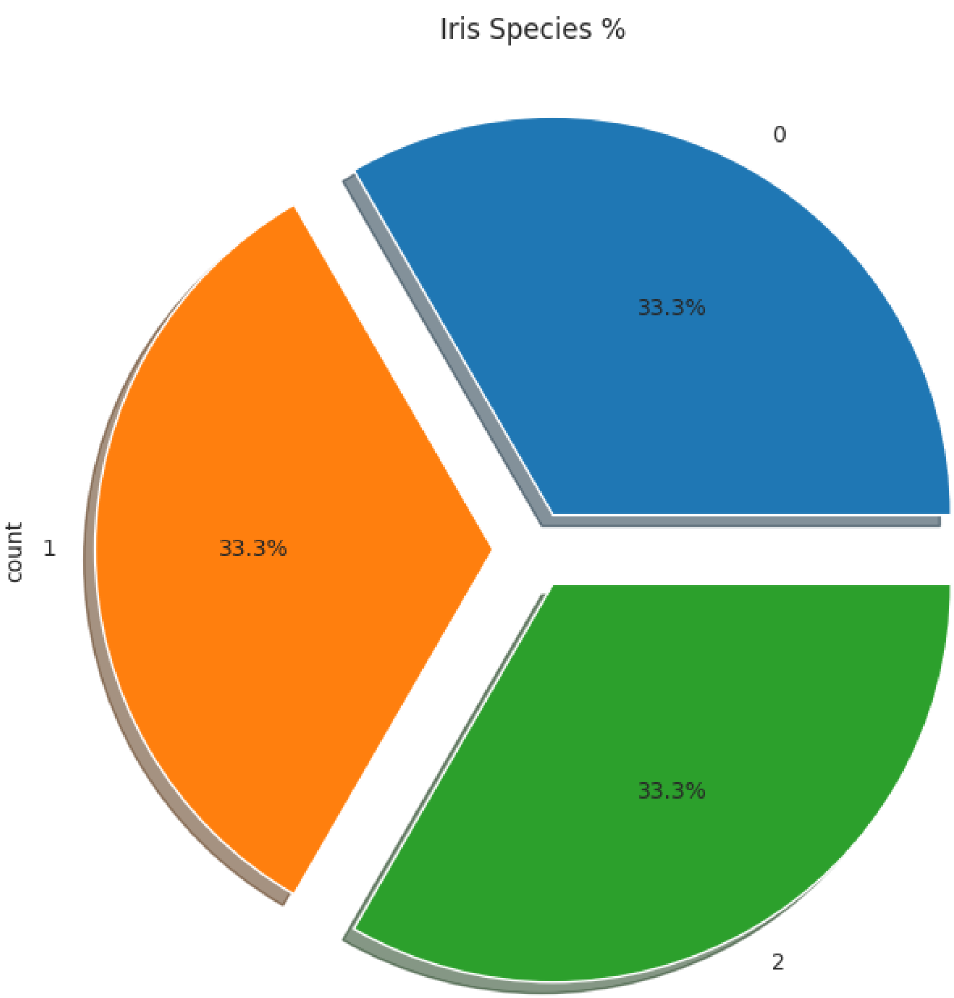
```
<Axes: xlabel='species', ylabel='petal_width'>
```



5.7 Pie Plot

In [578…

```
ax=plt.subplots(1,1,figsize=(10,8))
df['species'].value_counts().plot.pie(explode=[0.1,0.1,0.1],autopct='%1.1f%%',s
plt.title("Iris Species %")
plt.show()
```



6 | Split the Dataset

In [579…

```
from sklearn.model_selection import train_test_split
```

In [580…

```
X = df[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
```

In [581…

```
y = df['species']
```

In [582…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, randon
```

In [583…

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[583…
```
((120, 4), (30, 4), (120,), (30,))
```

7 | PCA (Principal Component Analysis)

In [584…
```python
from sklearn.decomposition import PCA
```

In [585…
```python
pca = PCA(n_components=2)
```

In [586…
```python
pca
```

Out[586…
```
PCA(n_components=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [587…
```python
X_pca = pca.fit_transform(X)
```

In [588…
```python
X_pca[0]
```

Out[588…
```
array([-2.68420713,  0.32660731])
```

In [589…
```python
print("Explained Variance Ratio:")
print(pca.explained_variance_ratio_)
```

```
Explained Variance Ratio:
[0.92461621 0.05301557]
```

In [590…
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_ft = scaler.fit_transform(X)
```

In [591…
```python
X_ft[0]
```

Out[591…
```
array([0.22222222, 0.625     , 0.06779661, 0.04166667])
```

In [592…
```python
X_ft[1]
```

Out[592…
```
array([0.16666667, 0.41666667, 0.06779661, 0.04166667])
```

Machine Learning Algorithm

Algorithm

(1) KNN

(1) KNN

In [593…
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [594…
```
knn_classifier = KNeighborsClassifier(n_neighbors=3)
```

In [595…
```
knn_classifier.fit(X_train, y_train)
```

Out[595…
```
KNeighborsClassifier(n_neighbors=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [596…
```
train_predictions = knn_classifier.predict(X_train)

train_accuracy1 = accuracy_score(y_train, train_predictions)
```

In [597…
```
test_predictions = knn_classifier.predict(X_test)

test_accuracy1 = accuracy_score(y_test, test_predictions)
```

In [598…
```
print(f"Training Accuracy: {train_accuracy1}")
print(f"Testing Accuracy: {test_accuracy1}")
```

```
Training Accuracy: 0.95
Testing Accuracy: 1.0
```

(2) Naive Bayes classifier

In [599…
```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics
```

In [600…
```
# GaussianNB
```

In [601…
```
G_classifier = GaussianNB()
```

In [602…
```
G_classifier.fit(X_train, y_train)
```

Out[602…
```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [603…
```
train_predictions = G_classifier.predict(X_train)

train_accuracy21 = accuracy_score(y_train, train_predictions)
```

In [604…
```
test_predictions = G_classifier.predict(X_test)

test_accuracy21 = accuracy_score(y_test, test_predictions)
```

In [605…
```
print(f"Training Accuracy: {train_accuracy21}")
print(f"Testing Accuracy: {test_accuracy21}")
```

```
Training Accuracy: 0.95
Testing Accuracy: 1.0
```

In [606…
```
# BernoulliNB
```

In [607…
```
B_classifier = BernoulliNB()
```

In [608…
```
B_classifier.fit(X_train, y_train)
```

Out[608…
```
BernoulliNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [609…
```
train_predictions = B_classifier.predict(X_train)

train_accuracy22 = accuracy_score(y_train, train_predictions)
```

In [610…
```
test_predictions = G_classifier.predict(X_test)

test_accuracy22 = accuracy_score(y_test, test_predictions)
```

In [611…
```
print(f"Training Accuracy: {train_accuracy22}")
print(f"Testing Accuracy: {test_accuracy22}")
```

```
Training Accuracy: 0.3416666666666667
Testing Accuracy: 1.0
```

In [612…
```
# MultinomialNB
```

In [613…
```
M_classifier = MultinomialNB()
```

In [614…
```
M_classifier.fit(X_train, y_train)
```

Out[614…
```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this

page with nbviewer.org.

In [615…
```
train_predictions = M_classifier.predict(X_train)

train_accuracy23 = accuracy_score(y_train, train_predictions)
```

In [616…
```
test_predictions = M_classifier.predict(X_test)

test_accuracy23 = accuracy_score(y_test, test_predictions)
```

In [617…
```
print(f"Training Accuracy: {train_accuracy23}")
print(f"Testing Accuracy: {test_accuracy23}")
```

```
Training Accuracy: 0.95
Testing Accuracy: 0.9
```

In [ ]:

👉 GaussianNB

Training Accuracy: 0.95

Testing Accuracy: 1.0

👉 BernoulliNB

Training Accuracy: 0.3416666666666667

Testing Accuracy: 1.0

👉 MultinomialNB

Training Accuracy: 0.95

Testing Accuracy: 0.9

Being the best of them | 🔥 GaussianNB |

(3) Decision Tree 🔄

In [618…
```
from sklearn.tree import DecisionTreeClassifier
```

In [619…
```
clf = DecisionTreeClassifier()
```

In [620…
```
clf.fit(X_train, y_train)
```

Out[620…
```
DecisionTreeClassifier()
```

Out[620…

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [621…

```
train_predictions = clf.predict(X_train)

train_accuracy3 = accuracy_score(y_train, train_predictions)
```

In [622…

```
test_predictions = clf.predict(X_test)

test_accuracy3 = accuracy_score(y_test, test_predictions)
```

In [623…

```
print(f"Training Accuracy: {train_accuracy3}")
print(f"Testing Accuracy: {test_accuracy3}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

(4) Random Forest

In [624…

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

In [625…

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [626…

```
rf_classifier.fit(X_train, y_train)
```

Out[626…

```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [627…

```
train_predictions = rf_classifier.predict(X_train)

train_accuracy4 = accuracy_score(y_train, train_predictions)
```

In [628…

```
test_predictions = rf_classifier.predict(X_test)

test_accuracy4 = accuracy_score(y_test, test_predictions)
```

In [629…

```
print(f"Training Accuracy: {train_accuracy4}")
print(f"Testing Accuracy: {test_accuracy4}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

(5) Boosting Algorithm

In [630…
```
from sklearn.ensemble import AdaBoostClassifier
```

In [631…
```
base_classifier = DecisionTreeClassifier(max_depth=1)
```

In [632…
```
adaboost_classifier = AdaBoostClassifier(base_classifier, n_estimators=50, rar
```

In [633…
```
adaboost_classifier.fit(X_train, y_train)
```

Out[633…
```
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
                   random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [634…
```
train_predictions = adaboost_classifier.predict(X_train)

train_accuracy5 = accuracy_score(y_train, train_predictions)
```

In [635…
```
test_predictions = adaboost_classifier.predict(X_test)

test_accuracy5 = accuracy_score(y_test, test_predictions)
```

In [636…
```
print(f"Training Accuracy: {train_accuracy5}")
print(f"Testing Accuracy: {test_accuracy5}")
```

```
Training Accuracy: 0.9666666666666667
Testing Accuracy: 1.0
```

(6).SVM

In [637…
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

In [638…
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [639…
```
svm_classifier = SVC(kernel='linear', C=1.0)
```

In [640…
```
svm_classifier.fit(X_train, y_train)
```

Out[640…
```
SVC(kernel='linear')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [641…
```
train_predictions = svm_classifier.predict(X_train)

train_accuracy6 = accuracy_score(y_train, train_predictions)
```

In [642…
```
test_predictions = svm_classifier.predict(X_test)

test_accuracy6 = accuracy_score(y_test, test_predictions)
```

In [643…
```
print(f"Training Accuracy: {train_accuracy6}")
print(f"Testing Accuracy: {test_accuracy6}")
```

```
Training Accuracy: 0.9833333333333333
Testing Accuracy: 0.9666666666666667
```

(7). Logistic Regression

In [644…
```
from sklearn import linear_model
```

In [645…
```
lrg = linear_model.LogisticRegression()
```

In [646…
```
lrg.fit(X_train, y_train)
```

Out[646…
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [647…
```
train_predictions = lrg.predict(X_train)

train_accuracy7 = accuracy_score(y_train, train_predictions)
```

In [648…
```
test_predictions = lrg.predict(X_test)

test_accuracy7 = accuracy_score(y_test, test_predictions)
```

In [649…
```
print(f"Training Accuracy: {train_accuracy7}")
print(f"Testing Accuracy: {test_accuracy7}")
```

```
Training Accuracy: 0.9666666666666667
Testing Accuracy: 1.0
```

(8).Linear Regression

In [650…
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [651…
```
model = LinearRegression()
```

In [652…

```
model.fit(X_train, y_train)
```

Out[652…

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [653…

```
train_predictions = clf.predict(X_train)

train_accuracy8 = accuracy_score(y_train, train_predictions)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(
```

In [654…

```
test_predictions = clf.predict(X_test)

test_accuracy8 = accuracy_score(y_test, test_predictions)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(
```

In [655…

```
print(f"Training Accuracy: {train_accuracy8}")
print(f"Testing Accuracy: {test_accuracy8}")
```

```
Training Accuracy: 0.3333333333333333
Testing Accuracy: 0.3333333333333333
```

(9).Gradient Boosting Machines (GBM)

In [656…

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [657…

```
model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_de
```

In [658…

```
model.fit(X_train, y_train)
```

Out[658…

```
GradientBoostingClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [659…

```
train_predictions = model.predict(X_train)

train_accuracy9 = accuracy_score(y_train, train_predictions)
```

In [660…

```
test_predictions = model.predict(X_test)
```

```
test_accuracy9 = accuracy_score(y_test, test_predictions)
```

In [661...

```
print(f"Training Accuracy: {train_accuracy9}")
print(f"Testing Accuracy: {test_accuracy9}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

In [662...

```
table
```

Out[662...

Test Accuracy	Train Accuracy
1.0	0.95

Random Forest, Decision Tree, Gradient Boosting Machines (GBM), Algorithm is the best accuracy

(GradientBoostingClassifier)

accuracy = 1.0

8 | Hierarchical Clustering

In [663…

```
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [664…

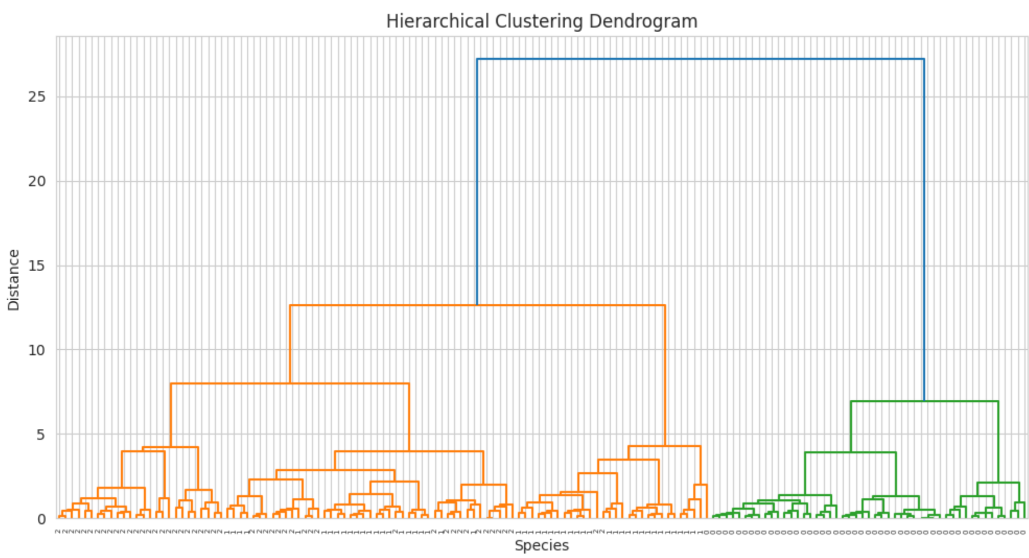
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [665…

```
linkage_matrix = linkage(X_scaled, method='ward')
```

In [666…

```
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix, labels=df['species'].values, orientation='top', dis
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Species')
plt.ylabel('Distance')
plt.show()
```



In [ ]: