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.

```
# iris dataset
```

1 import Necessary Library

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

2 import Dataset

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

3 Data Analysis

```
df.head()
```

	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

```
df.tail()
```

	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
```

```
df.shape
```

```
(150, 5)
```

```
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
```

```
df.dtypes
```

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

```
df.describe()
```

```
       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
```

```
df.corr
```

```
<bound method DataFrame.corr of      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
..            ...          ...           ...          ...        ...
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]>
```

```
df.ndim
```

```
2
```

```
df.columns
```

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

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

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

4 Data cleaning and Preprocessing:

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

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

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

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

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

df.head()
```

	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

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

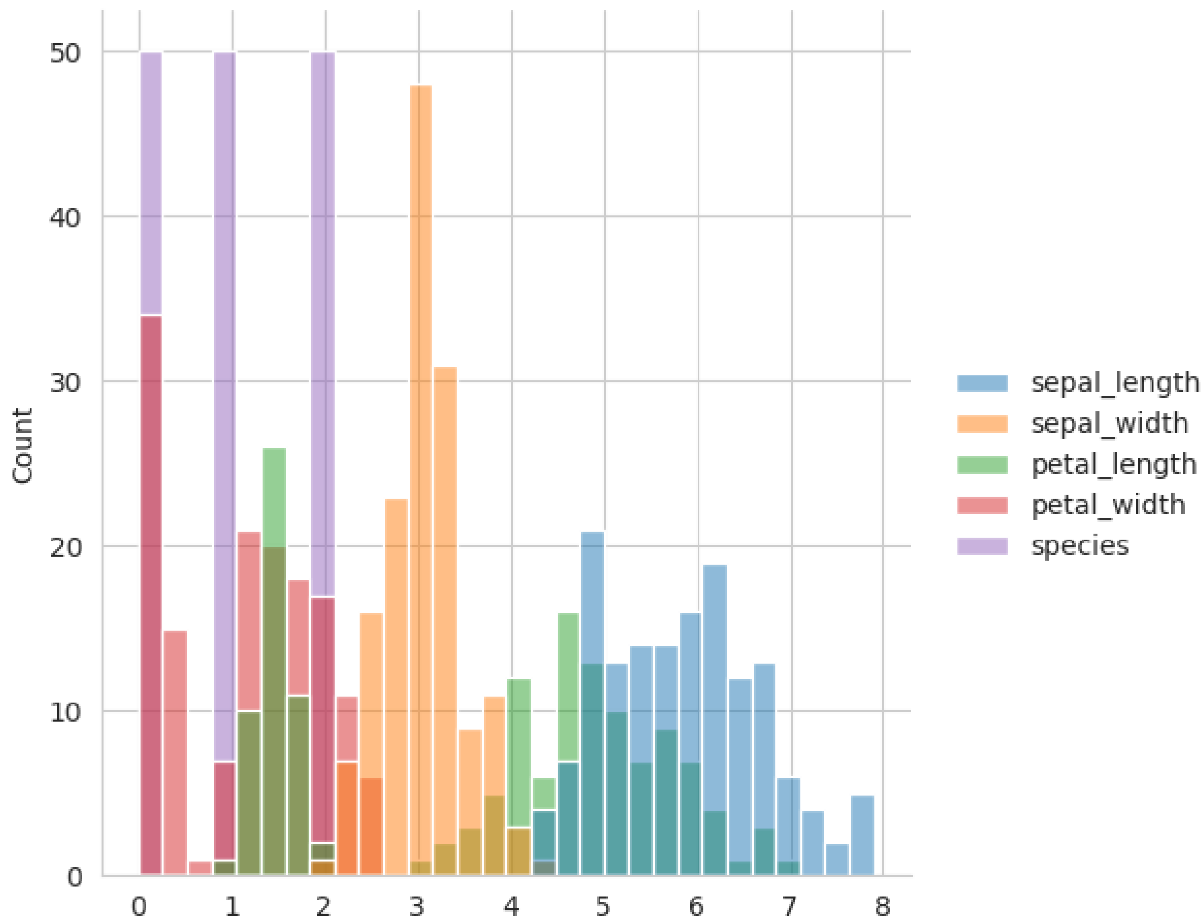
```
species
0    50
1    50
2    50
Name: count, dtype: int64
```

5| Data visualisation 📊 📉

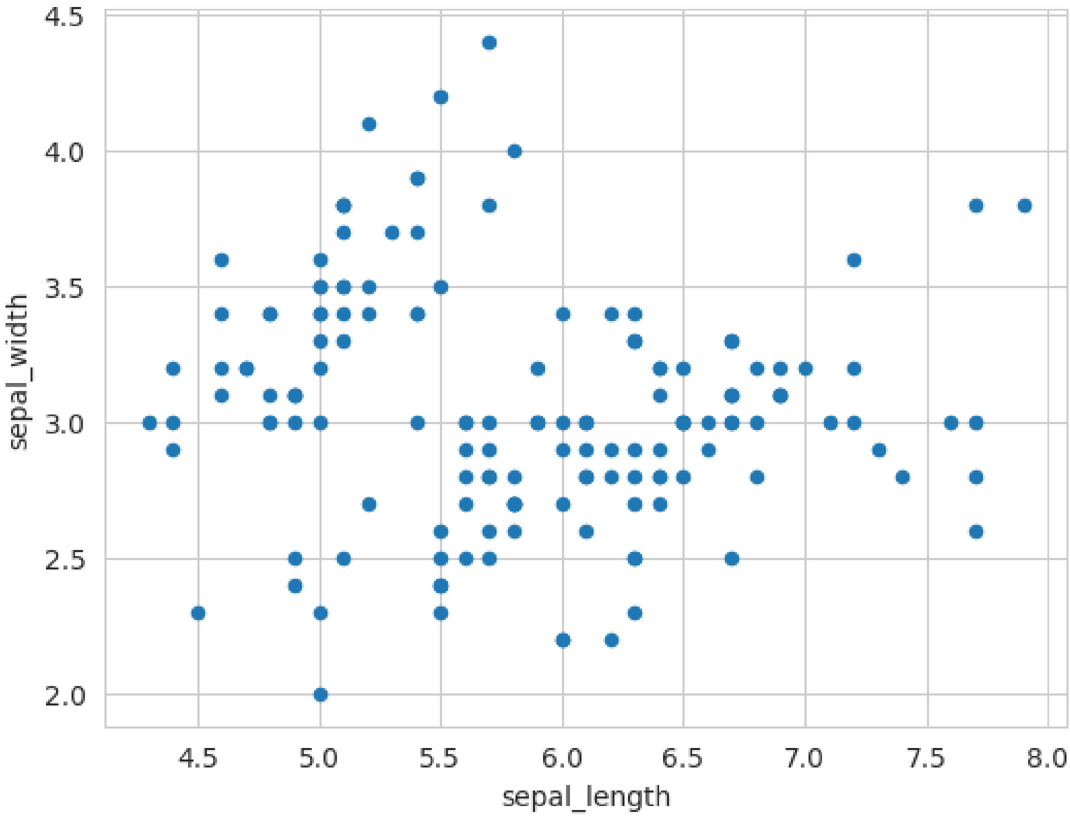
EDA (Exploratory Data Analysis)

5.1 displot

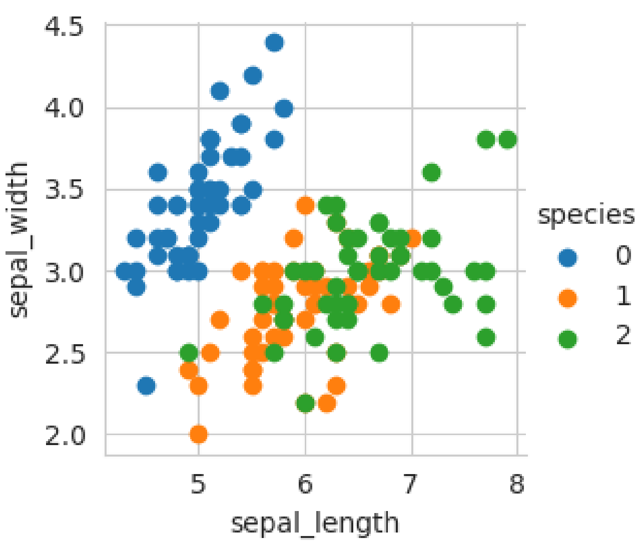
```
sns.displot(df, kde=False, bins=30)
plt.show()
```




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



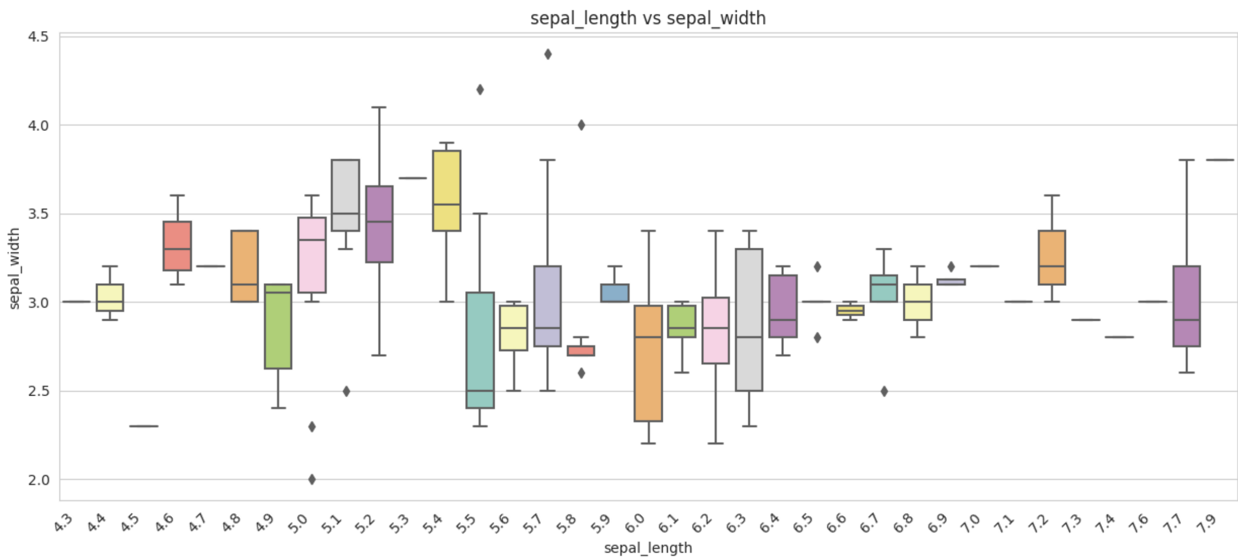

```
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="species").map(plt.scatter, "sepal_length",
"sepal_width").add_legend();
plt.show();
```



5.2 BoxPlot

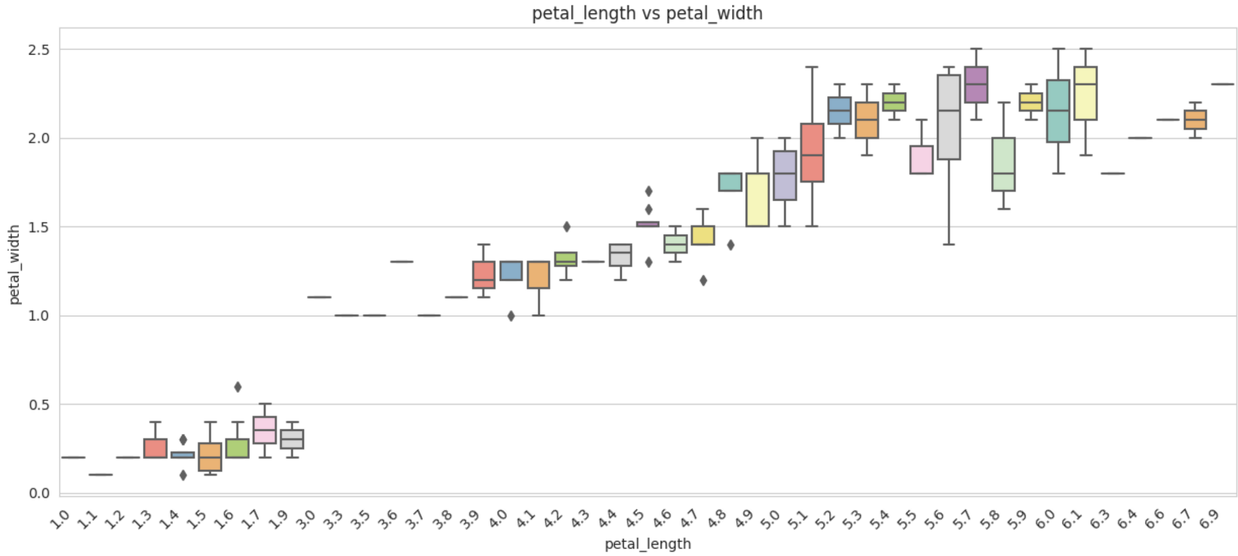
```
# sepal_length vs sepal_width boxplot

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



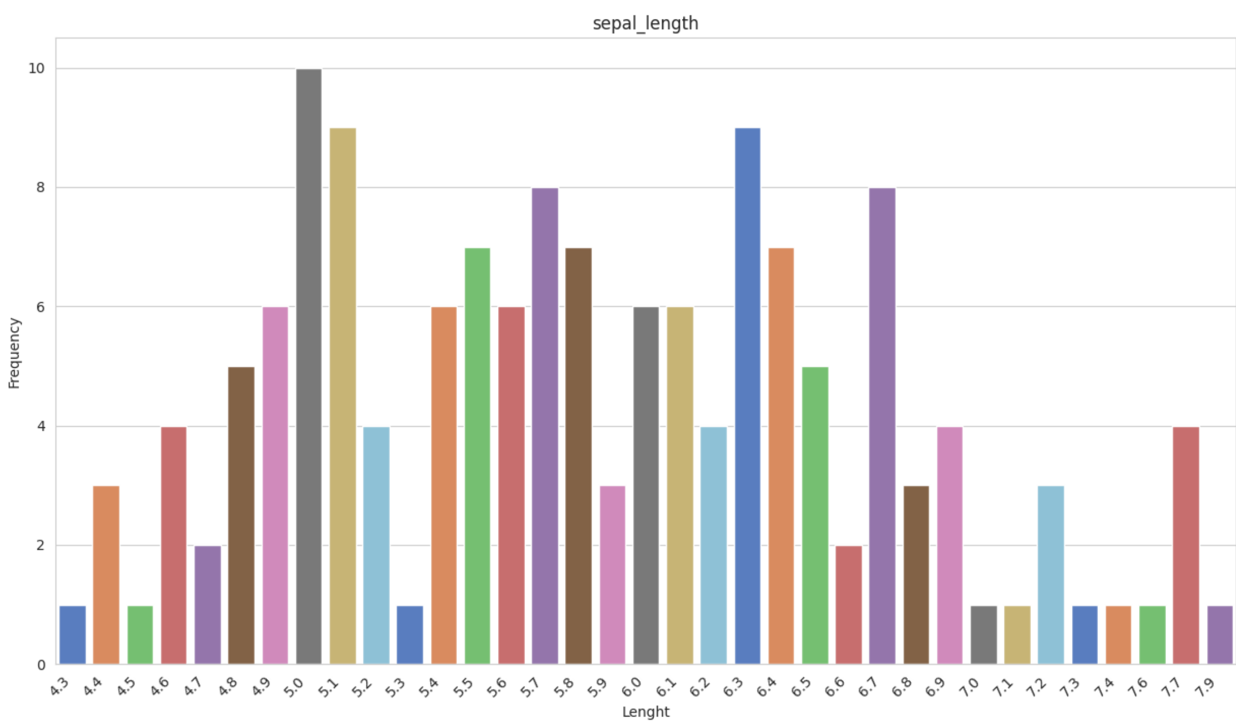

```
# petal_length vs petal_width boxplot

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()
```

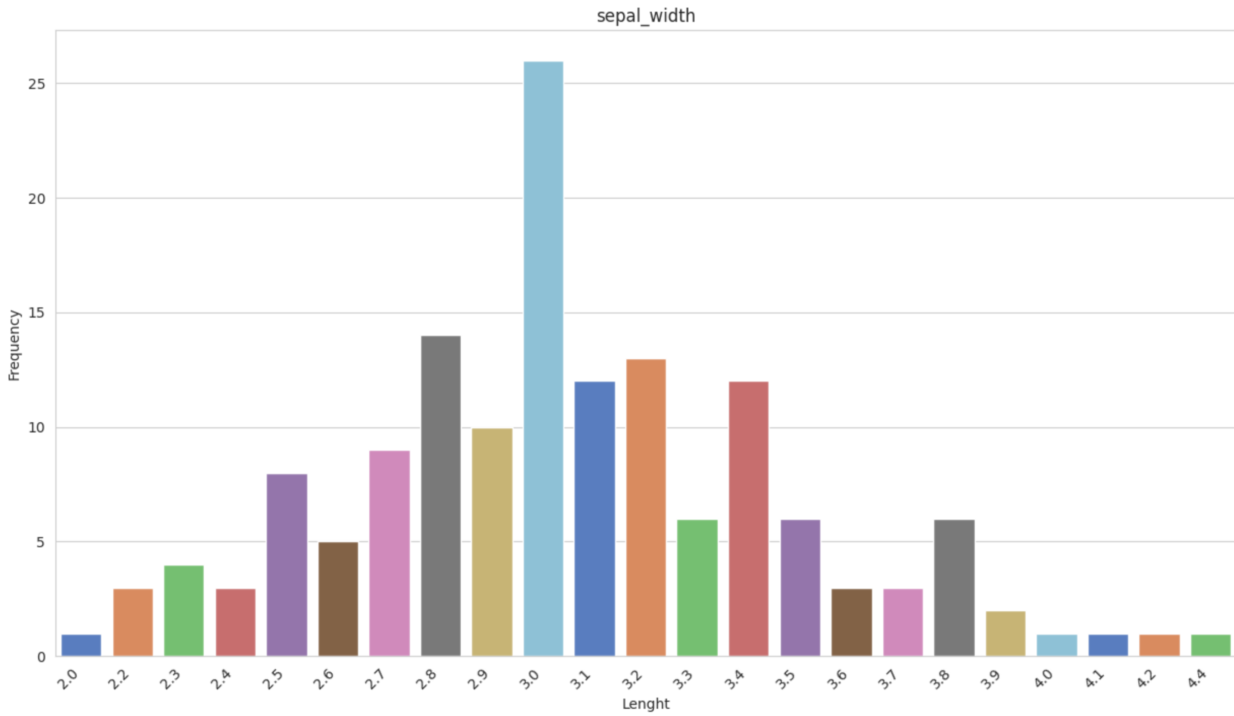


5.3 countplot

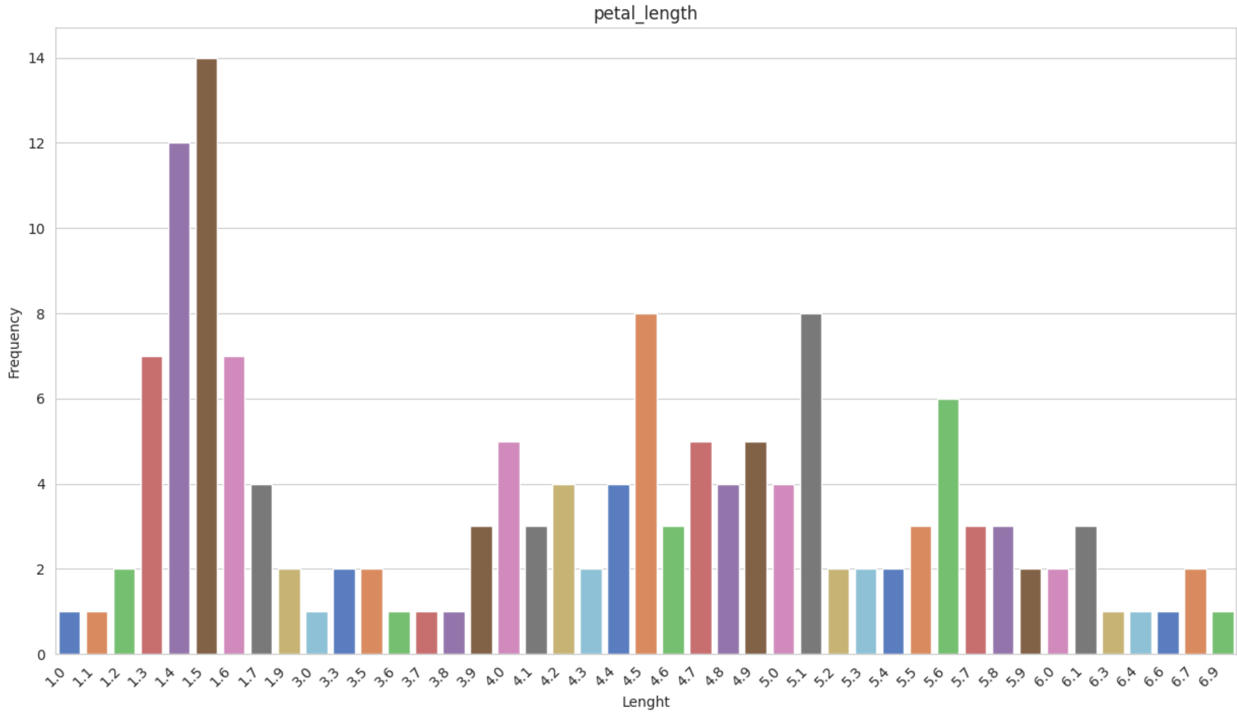
```
# sepal_length
plt.figure(figsize=(15, 8))
sns.countplot(x='sepal_length', data=df, palette='muted')
plt.title('sepal_length')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```



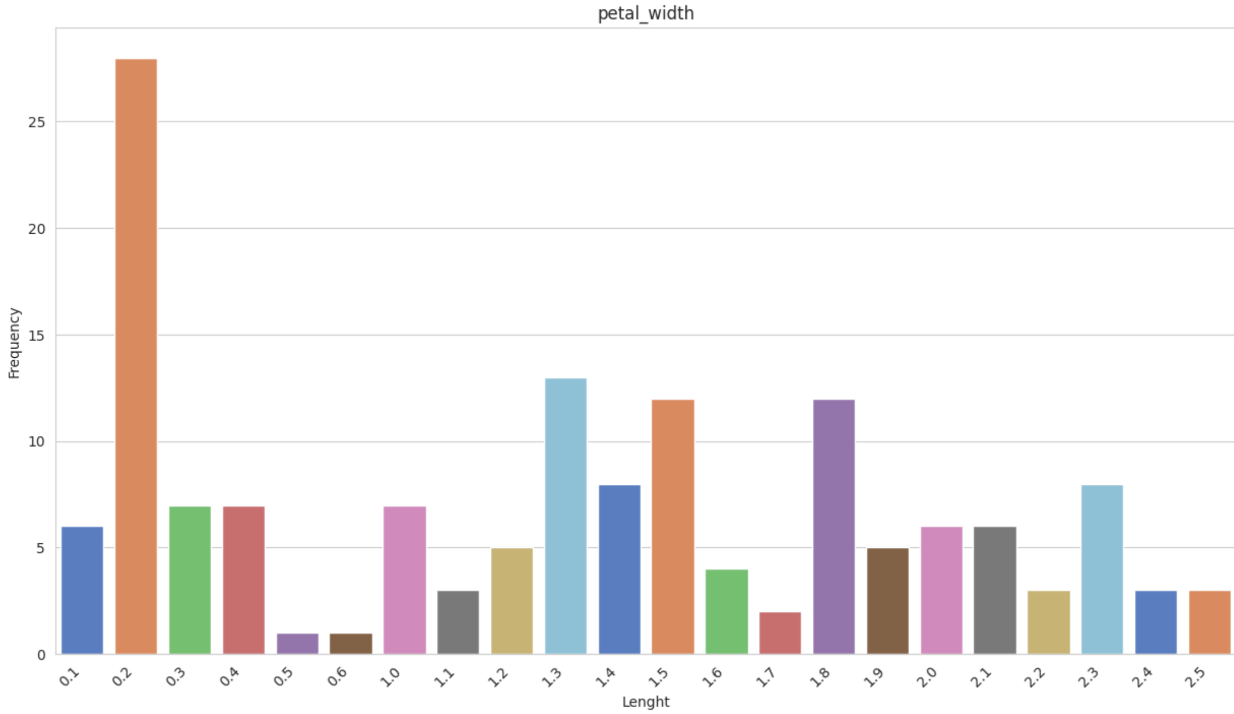
```python
# sepal_width
plt.figure(figsize=(15, 8))
sns.countplot(x='sepal_width', data=df, palette='muted')
plt.title('sepal_width')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```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()
```




```
# 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()
```



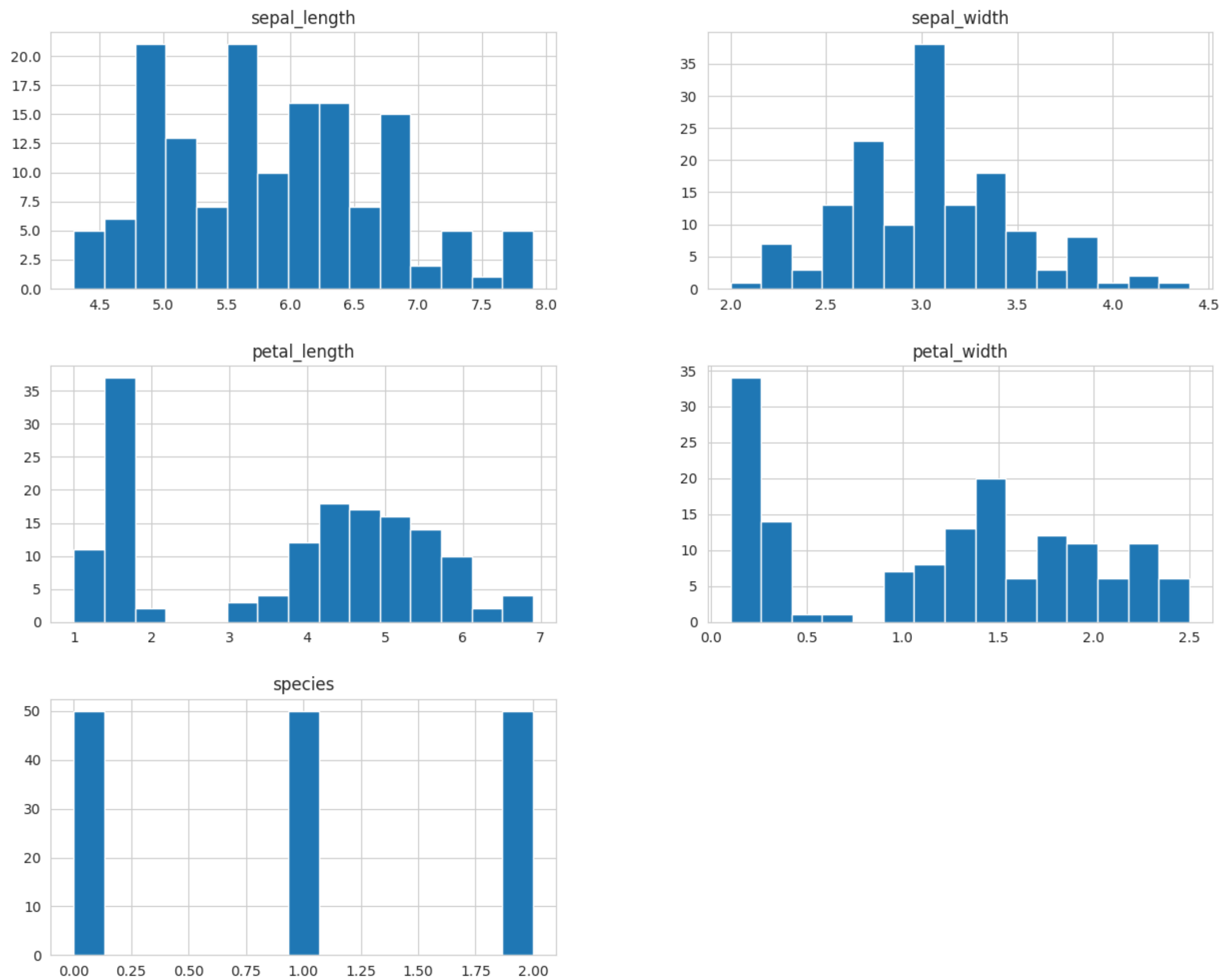
5.4 pairplot

```
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: The `size` parameter has been renamed to `height`; please
update your code.
  warnings.warn(msg, UserWarning)
```

5.5 hist Plot

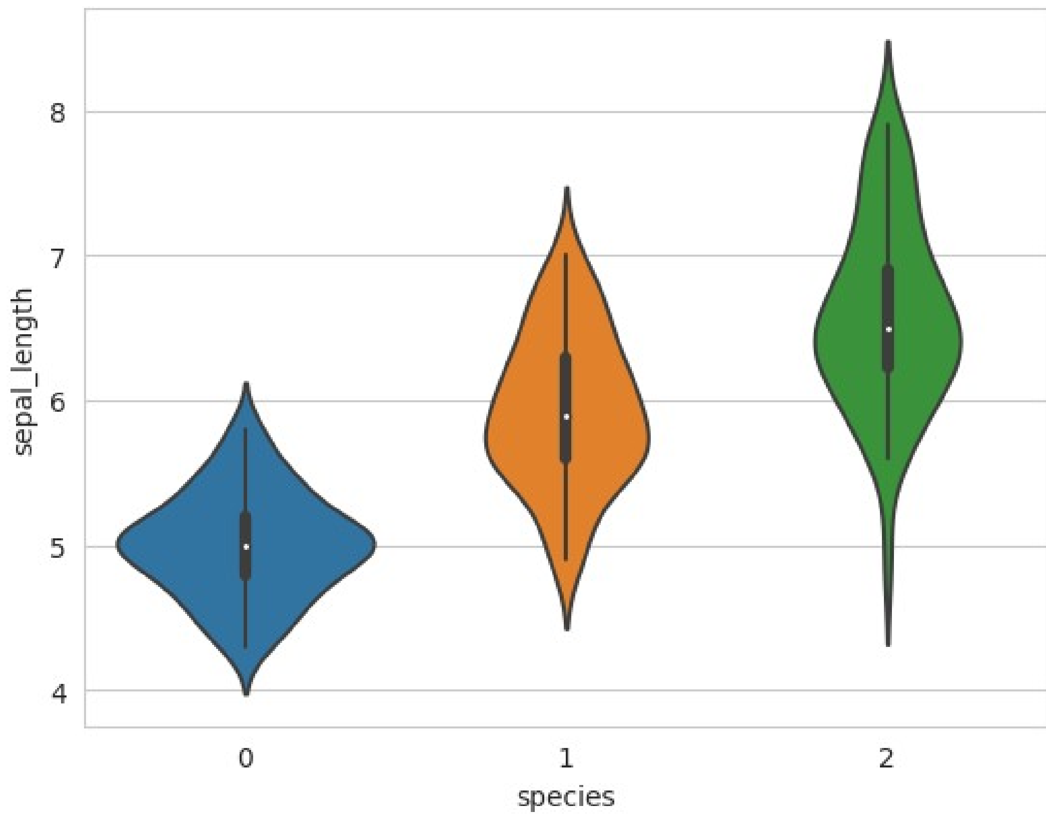
```
df.hist(figsize=(15,12),bins = 15)
plt.title("Features Distribution")
plt.show()
```



5.6 violinplot

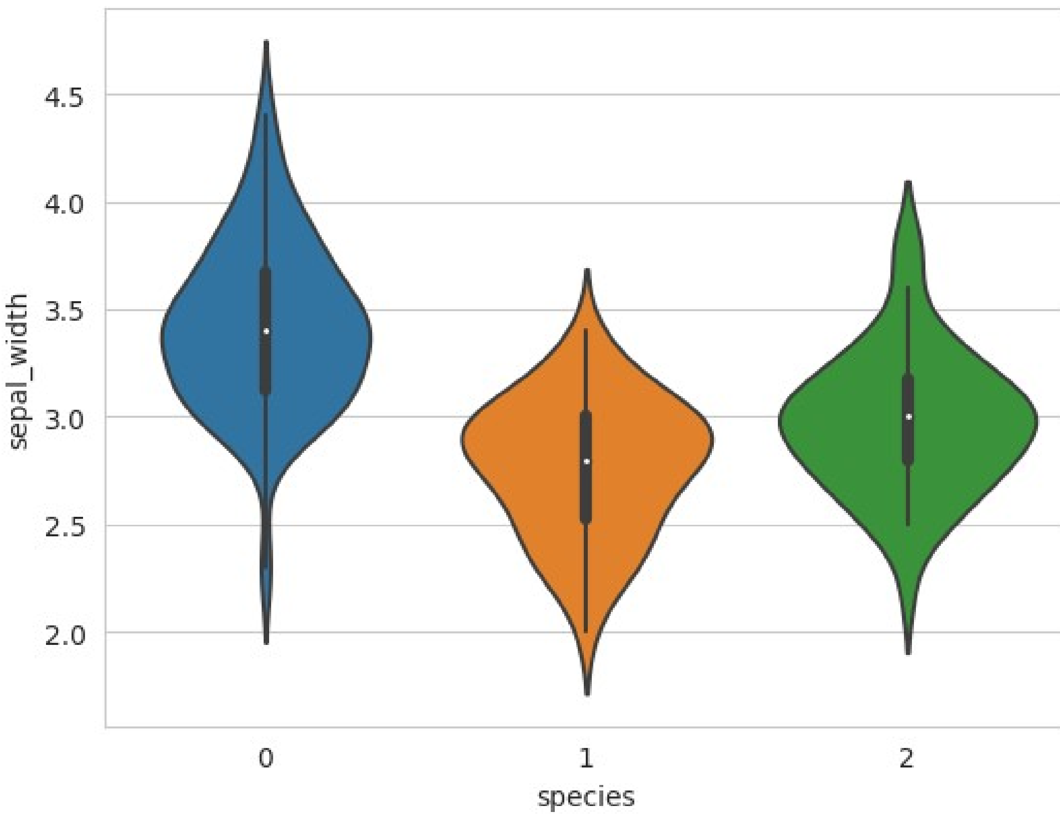
```
sns.violinplot(x="species",y="sepal_length", data=df, size = 8)
```

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



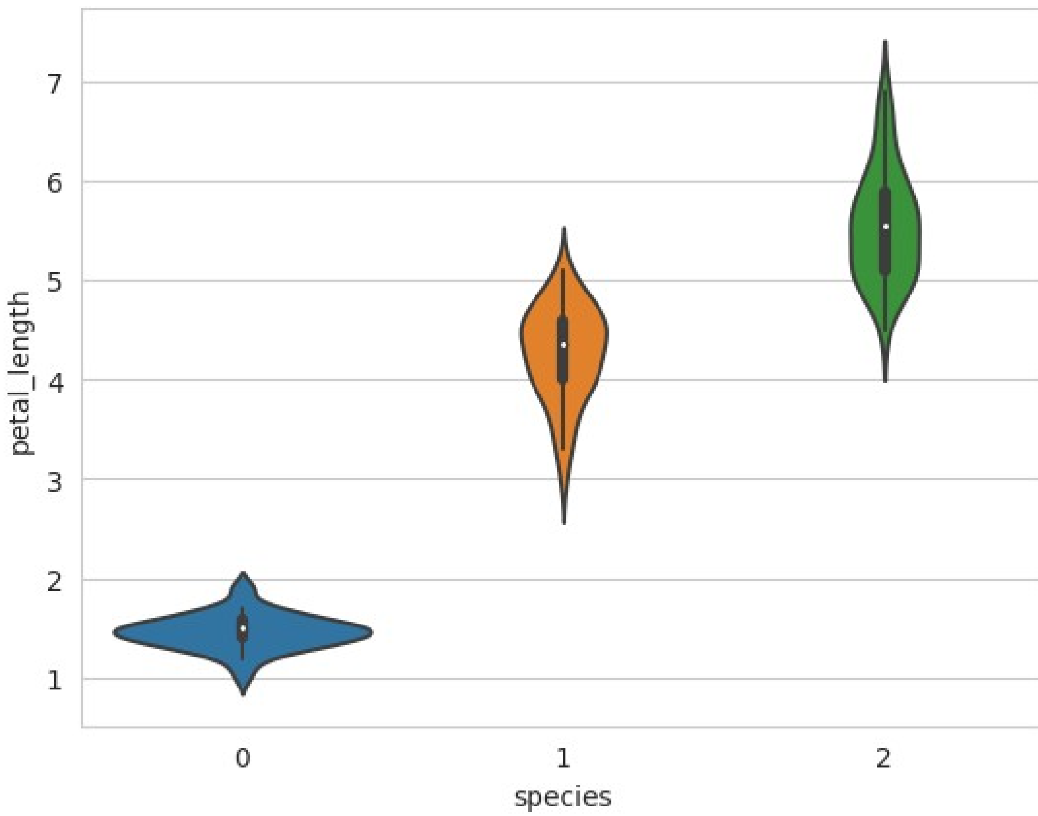

```
sns.violinplot(x="species",y="sepal_width", data=df, size = 8)

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



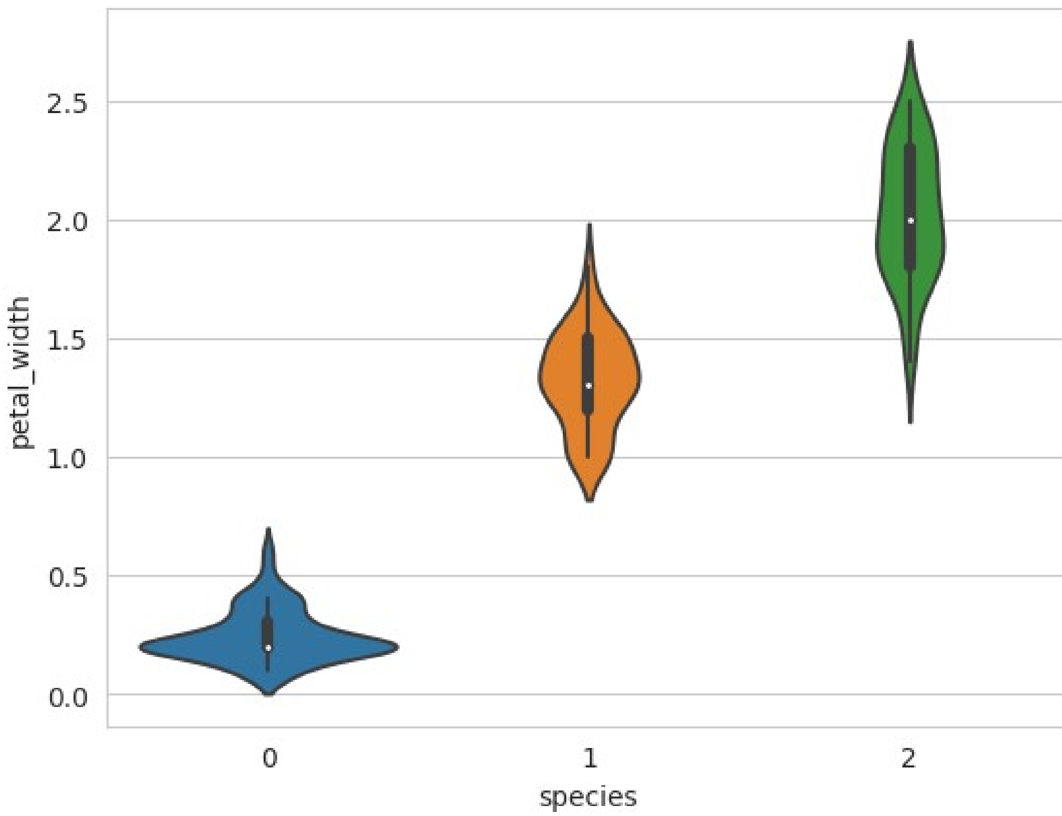

```
sns.violinplot(x="species",y="petal_length", data=df, size = 8)
```

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



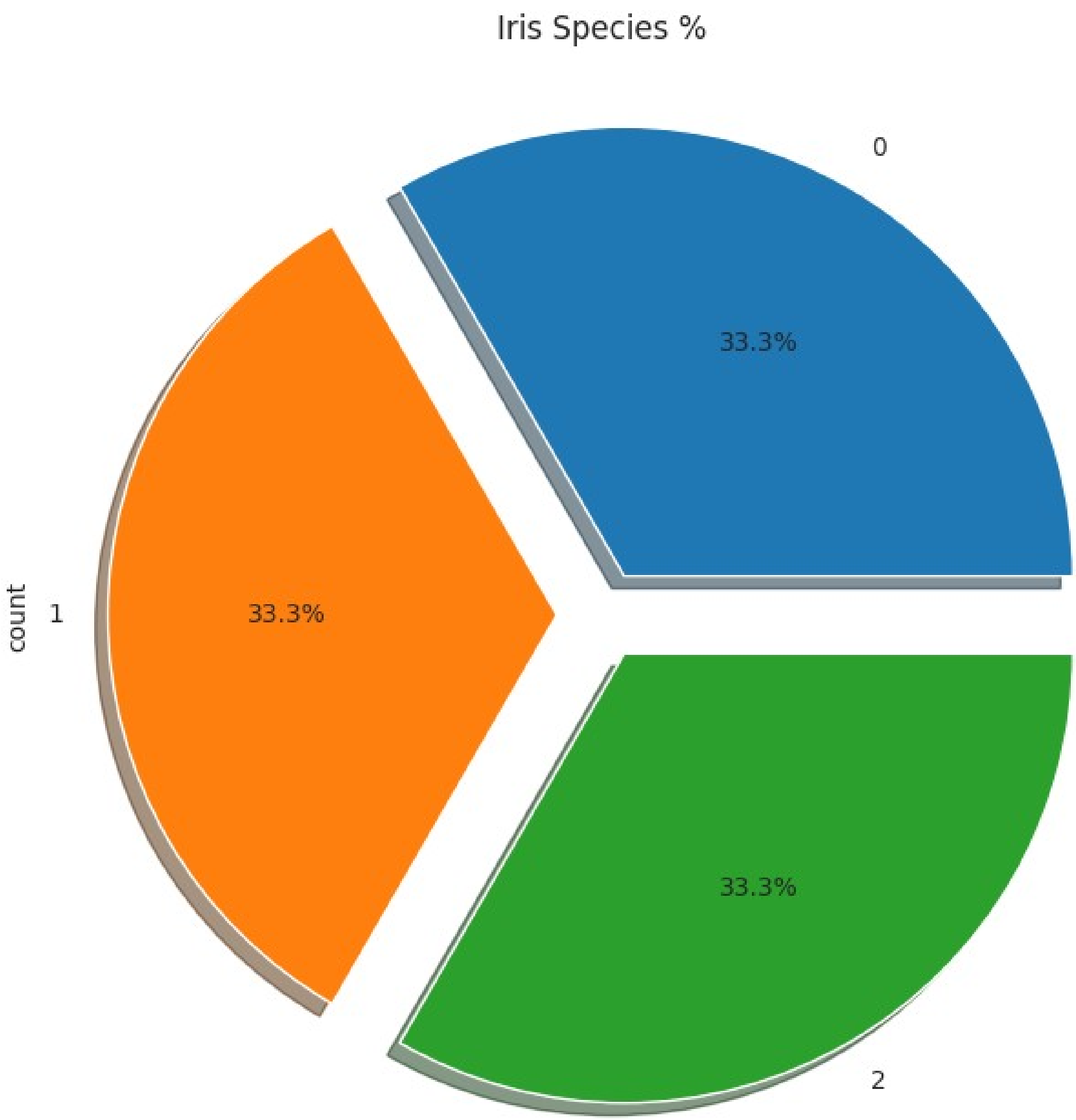

```
sns.violinplot(x="species",y="petal_width", data=df, size = 8)
```

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



5.7 Pie Plot

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



6 | Split the Dataset

```
from sklearn.model_selection import train_test_split

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

y = df['species']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=.2, random_state=42)

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

```
((120, 4), (30, 4), (120,), (30,))
```

7 | PCA (Principal Component Analysis)

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

pca

PCA(n_components=2)

X_pca = pca.fit_transform(X)

X_pca[0]

array([-2.68420713,  0.32660731])

print("Explained Variance Ratio:")
print(pca.explained_variance_ratio_)

Explained Variance Ratio:
[0.92461621 0.05301557]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_ft = scaler.fit_transform(X)

X_ft[0]

array([0.22222222, 0.625     , 0.06779661, 0.04166667])

X_ft[1]

array([0.16666667, 0.41666667, 0.06779661, 0.04166667])
```

Algorithm 🔄

(1) KNN 🔄

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

knn_classifier = KNeighborsClassifier(n_neighbors=3)

knn_classifier.fit(X_train, y_train)

KNeighborsClassifier(n_neighbors=3)
```

```
train_predictions = knn_classifier.predict(X_train)

train_accuracy1 = accuracy_score(y_train, train_predictions)

test_predictions = knn_classifier.predict(X_test)

test_accuracy1 = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy1}")
print(f"Testing Accuracy: {test_accuracy1}")
```

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

(2) Naive Bayes classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics

# GaussianNB

G_classifier = GaussianNB()

G_classifier.fit(X_train, y_train)
```

```
GaussianNB()
```

```
train_predictions = G_classifier.predict(X_train)

train_accuracy21 = accuracy_score(y_train, train_predictions)

test_predictions = G_classifier.predict(X_test)

test_accuracy21 = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy21}")
print(f"Testing Accuracy: {test_accuracy21}")
```

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

```
# BernoulliNB

B_classifier = BernoulliNB()

B_classifier.fit(X_train, y_train)
```

```
BernoulliNB()
```

```python
train_predictions = B_classifier.predict(X_train)

train_accuracy22 = accuracy_score(y_train, train_predictions)

test_predictions = G_classifier.predict(X_test)

test_accuracy22 = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy22}")
print(f"Testing Accuracy: {test_accuracy22}")
```

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

```python
# MultinomialNB

M_classifier = MultinomialNB()

M_classifier.fit(X_train, y_train)
```

```
MultinomialNB()
```

```python
train_predictions = M_classifier.predict(X_train)

train_accuracy23 = accuracy_score(y_train, train_predictions)

test_predictions = M_classifier.predict(X_test)

test_accuracy23 = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy23}")
print(f"Testing Accuracy: {test_accuracy23}")
```

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

☞ 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 🔄

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

DecisionTreeClassifier()

train_predictions = clf.predict(X_train)

train_accuracy3 = accuracy_score(y_train, train_predictions)

test_predictions = clf.predict(X_test)

test_accuracy3 = accuracy_score(y_test, test_predictions)

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

Training Accuracy: 1.0
Testing Accuracy: 1.0
```

(4) Random Forest

```
from sklearn.ensemble import RandomForestClassifier

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

rf_classifier.fit(X_train, y_train)

RandomForestClassifier(random_state=42)

train_predictions = rf_classifier.predict(X_train)

train_accuracy4 = accuracy_score(y_train, train_predictions)

test_predictions = rf_classifier.predict(X_test)

test_accuracy4 = accuracy_score(y_test, test_predictions)

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

Training Accuracy: 1.0
Testing Accuracy: 1.0
```

(5) Boosting Algorithm

```
from sklearn.ensemble import AdaBoostClassifier

base_classifier = DecisionTreeClassifier(max_depth=1)

adaboost_classifier = AdaBoostClassifier(base_classifier,
n_estimators=50, random_state=42)

adaboost_classifier.fit(X_train, y_train)

AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
                   random_state=42)

train_predictions = adaboost_classifier.predict(X_train)

train_accuracy5 = accuracy_score(y_train, train_predictions)

test_predictions = adaboost_classifier.predict(X_test)

test_accuracy5 = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy5}")
print(f"Testing Accuracy: {test_accuracy5}")
```

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

(6).SVM

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_classifier = SVC(kernel='linear', C=1.0)

svm_classifier.fit(X_train, y_train)

SVC(kernel='linear')

train_predictions = svm_classifier.predict(X_train)

train_accuracy6 = accuracy_score(y_train, train_predictions)

test_predictions = svm_classifier.predict(X_test)

test_accuracy6 = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy6}")
print(f"Testing Accuracy: {test_accuracy6}")

Training Accuracy: 0.9833333333333333
Testing Accuracy: 0.9666666666666667
```

(7). Logistic Regression

```
from sklearn import linear_model

lrg = linear_model.LogisticRegression()

lrg.fit(X_train, y_train)

LogisticRegression()

train_predictions = lrg.predict(X_train)

train_accuracy7 = accuracy_score(y_train, train_predictions)

test_predictions = lrg.predict(X_test)

test_accuracy7 = accuracy_score(y_test, test_predictions)
```

```
print(f"Training Accuracy: {train_accuracy7}")
print(f"Testing Accuracy: {test_accuracy7}")

Training Accuracy: 0.9666666666666667
Testing Accuracy: 1.0
```

(8).Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()

model.fit(X_train, y_train)

LinearRegression()

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(

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(

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)

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)

model.fit(X_train, y_train)
```

```
GradientBoostingClassifier(random_state=42)

train_predictions = model.predict(X_train)

train_accuracy9 = accuracy_score(y_train, train_predictions)

test_predictions = model.predict(X_test)

test_accuracy9 = accuracy_score(y_test, test_predictions)

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

Training Accuracy: 1.0
Testing Accuracy: 1.0

table

+---------------+----------------+
| 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

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

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

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

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