Kruti Bagwe Roll no.05 TE INFT-A Batch-1

Experiment-3

Titanic Dataset

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

```
sns.get_dataset_names()
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

```
df = sns.load_dataset ("titanic")
```

```
df
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

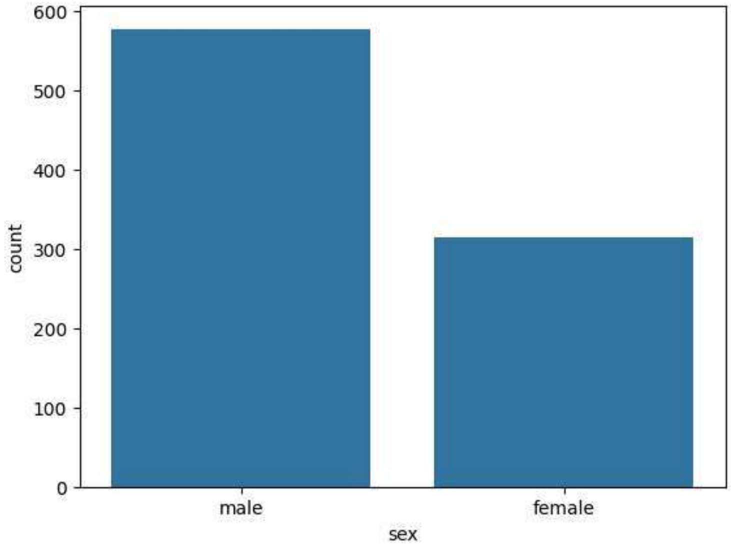
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
```

```
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

```
#count plot categorical data
sns.countplot(x="sex", data = df)
```

```
<Axes: xlabel='sex', ylabel='count'>
```



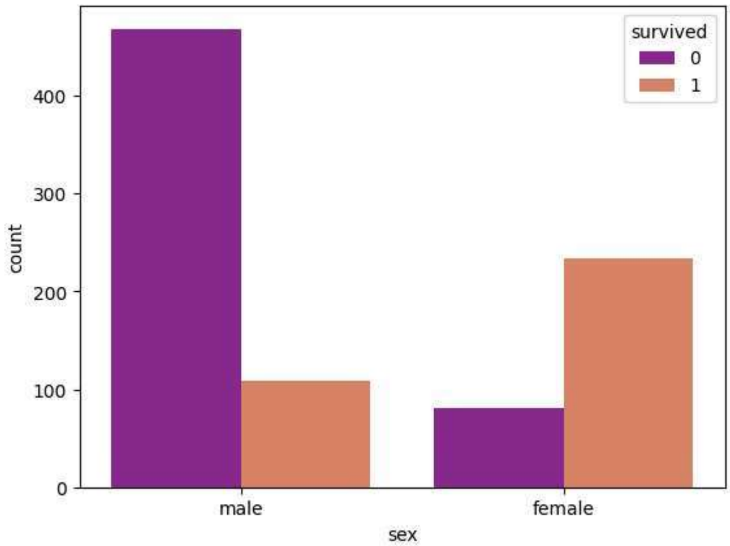

```
df.groupby("sex")["survived"].value_counts()
```

sex	survived	count
female	**1**	233
	0	81
male	**0**	468
	1	109

dtype: int64

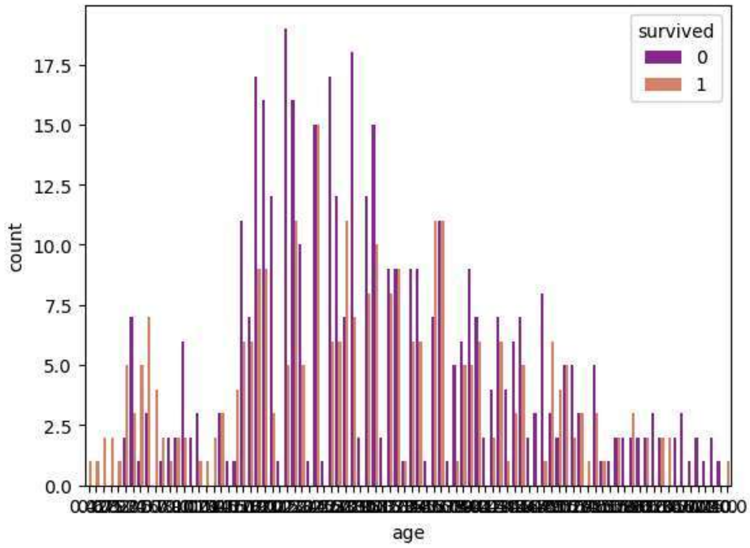
```
sns.countplot(x ="sex", hue = "survived", data = df, palette = "plasma")
```

```
<Axes: xlabel='sex', ylabel='count'>
```



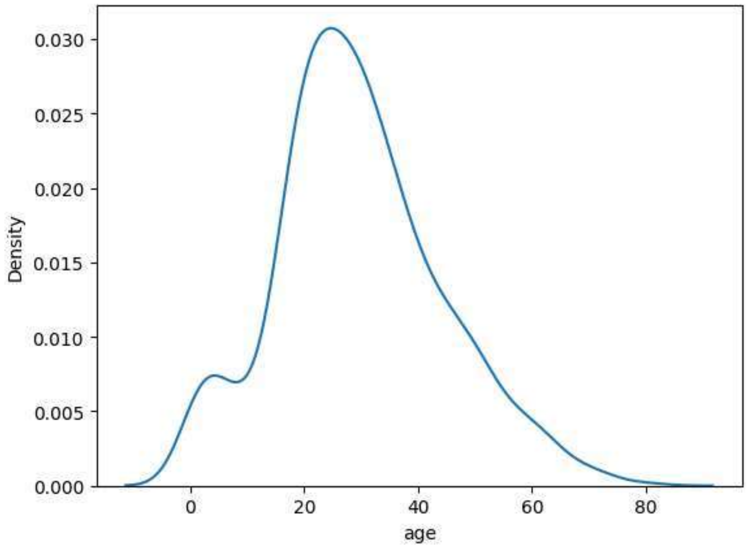

```
sns.countplot(x ="age", hue = "survived", data = df, palette = "plasma")
```

```
<Axes: xlabel='age', ylabel='count'>
```



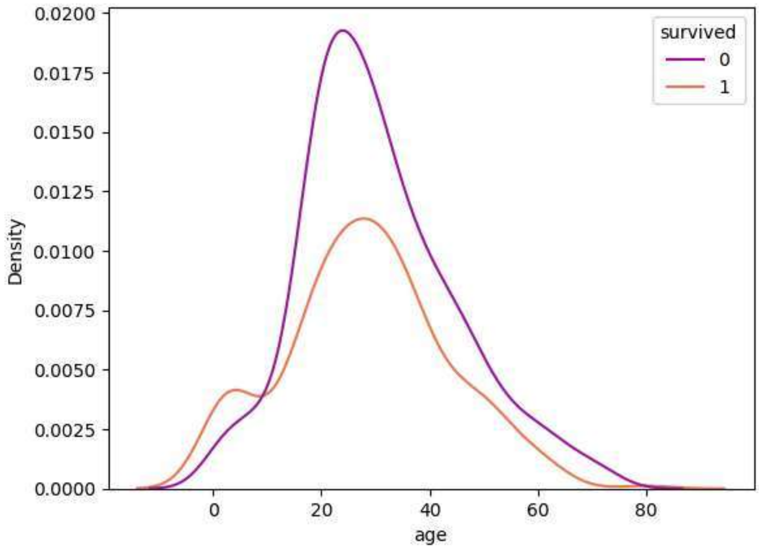

```
sns.kdeplot(x="age", data=df,palette = "plasma")
```

```
<ipython-input-58-72a6893a33ce>:1: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.kdeplot(x="age", data=df,palette = "plasma")
<Axes: xlabel='age', ylabel='Density'>
```



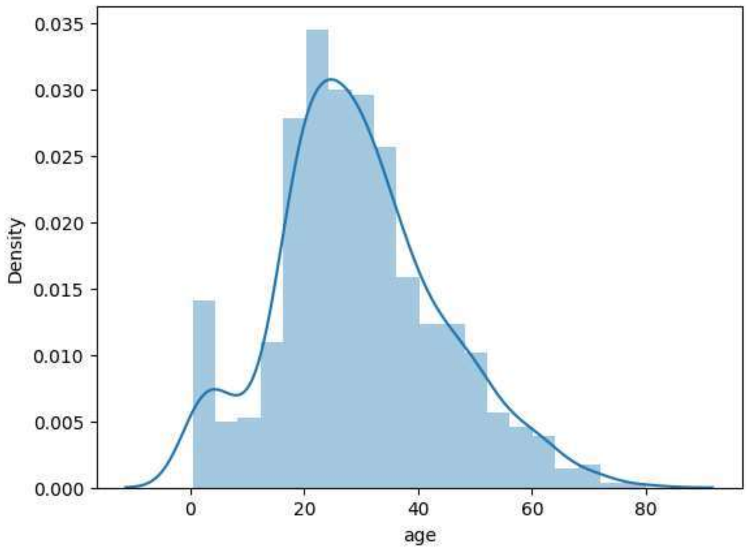

```
#KDE plot : Kernel density estimate plot : showing distribution of continuos data
sns.kdeplot(x ="age", hue = "survived", data = df, palette = "plasma")
```

```
<Axes: xlabel='age', ylabel='Density'>
```



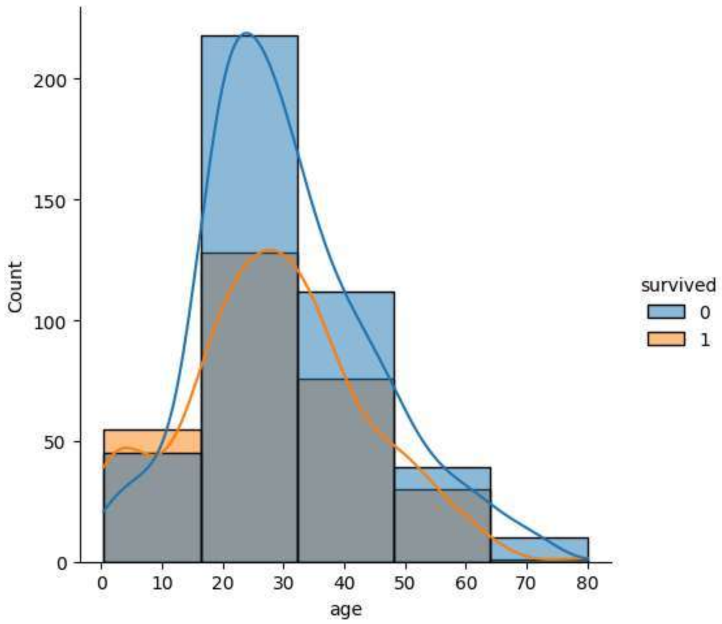
```
sns.distplot(df["age"])
```

```
<ipython-input-61-eef84e7ff8f0>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["age"])
<Axes: xlabel='age', ylabel='Density'>
```



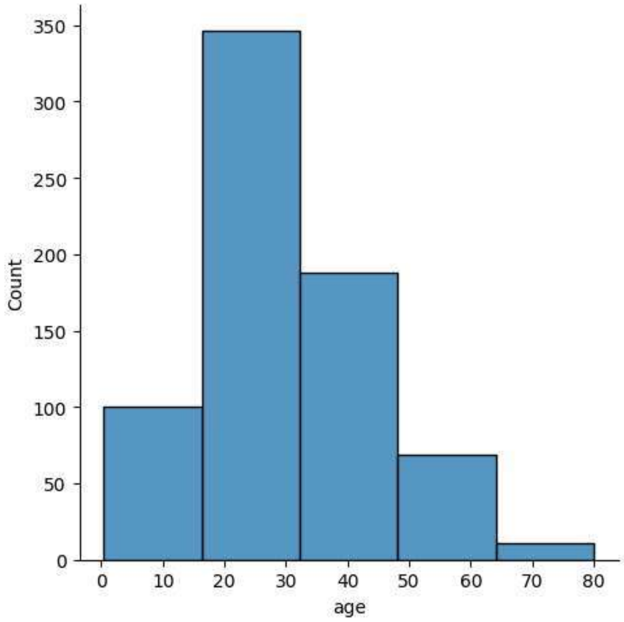

```
#distribution plot --> similar kde-->probability across other values
sns.displot(x = "age", hue="survived", bins= 5, data = df, kde = True)
```

```
<seaborn.axisgrid.FacetGrid at 0x787c34caa3d0>
```



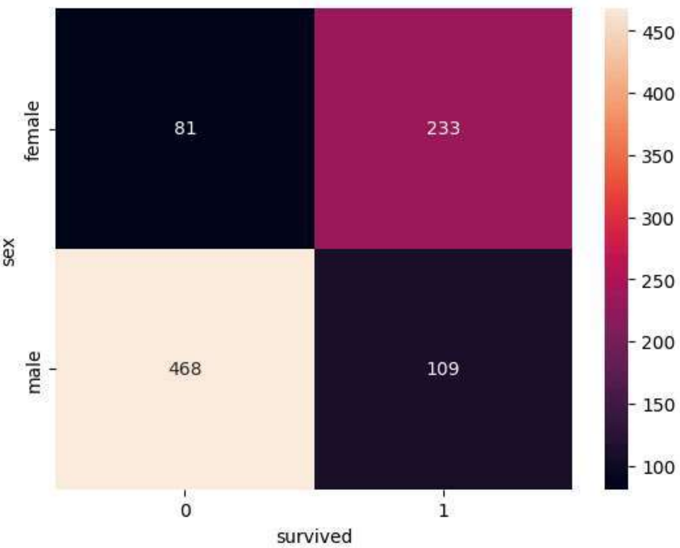

```
sns.displot(x="age", bins=5, data = df)
```

<seaborn.axisgrid.FacetGrid at 0x787c2be6ab90>



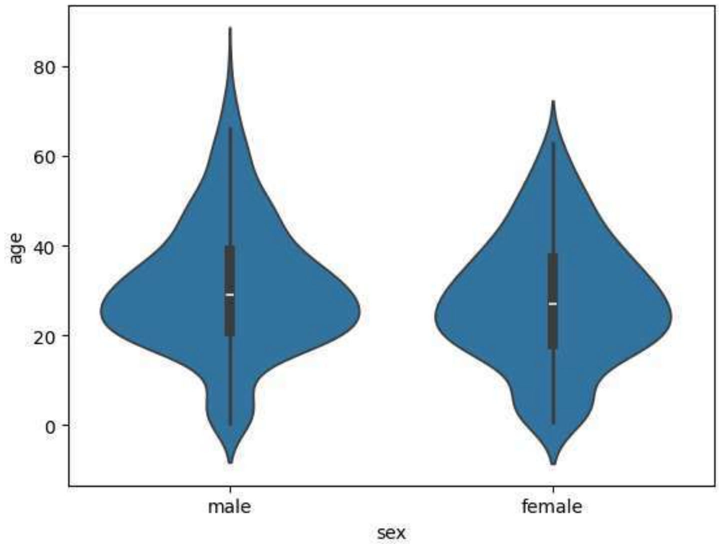

```
#heat Map:
group = df.groupby(['sex', 'survived'])
class_survived = group.size().unstack()
sns.heatmap(class_survived, annot = True, fmt="d")
```

<Axes: xlabel='survived', ylabel='sex'>



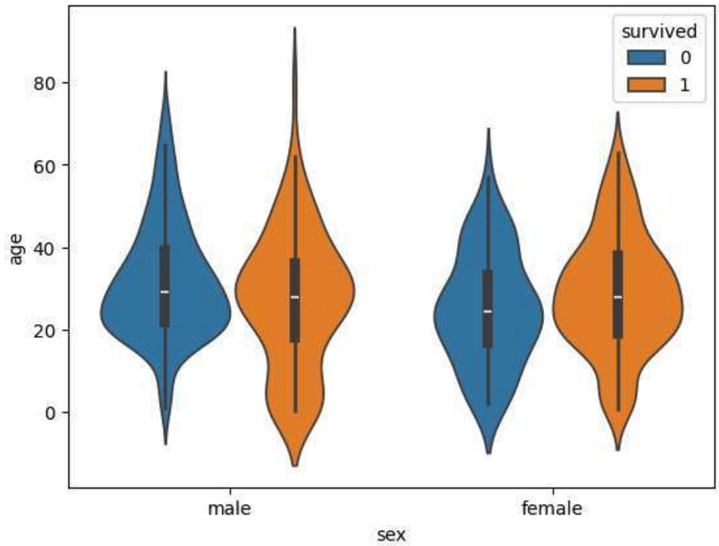

```
sns.violinplot(x="sex",y="age",data=df)
```

```
<Axes: xlabel='sex', ylabel='age'>
```



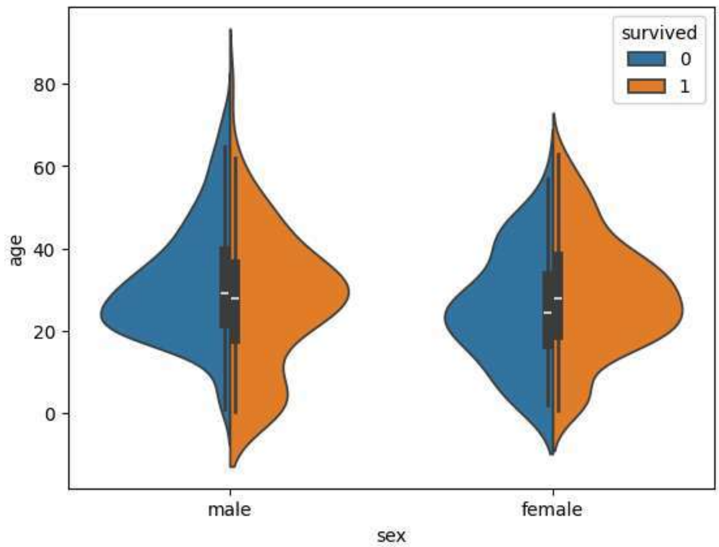

```
sns.violinplot(x="sex", y="age", hue="survived", data = df)
```

```
<Axes: xlabel='sex', ylabel='age'>
```




```
#violin plot : distribution of data
#show distrbution of surival rate according age value
sns.violinplot(x ="sex", y ="age", hue = "survived", data = df, split = True)
```

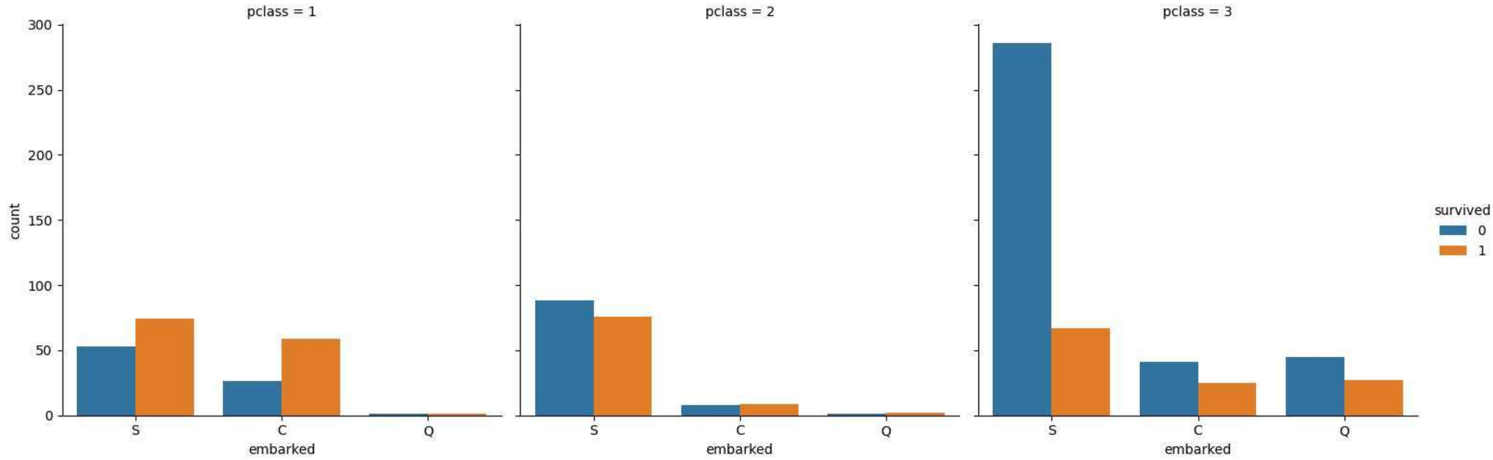
```
<Axes: xlabel='sex', ylabel='age'>
```




```
#countplot #catplot
```

```
sns.catplot(x = "embarked", kind ="count", col = "pclass", hue = "survived", data = df )
```

```
<seaborn.axisgrid.FacetGrid at 0x787c306af790>
```



Iris Dataset

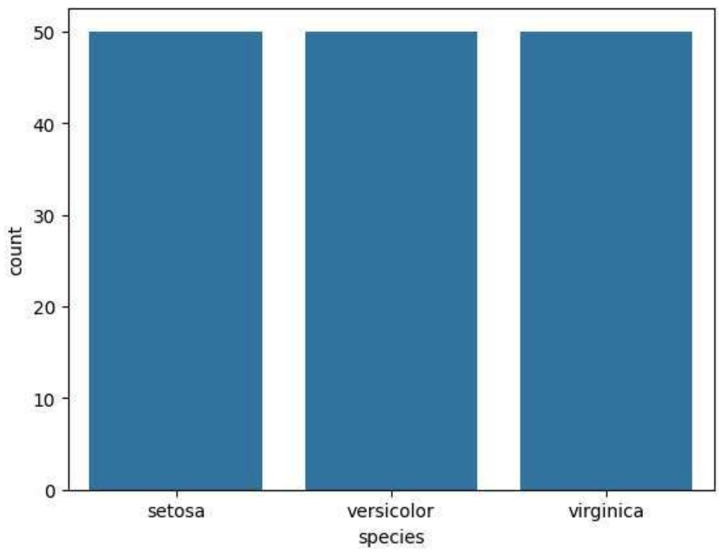
```
df = sns.load_dataset ("iris")
df
```

	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 × 5 columns

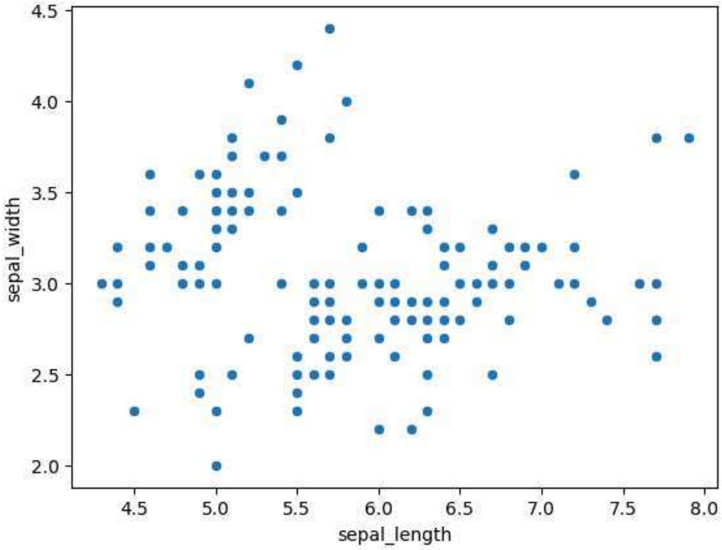
```
sns.countplot(x = "species", data = df)
```

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



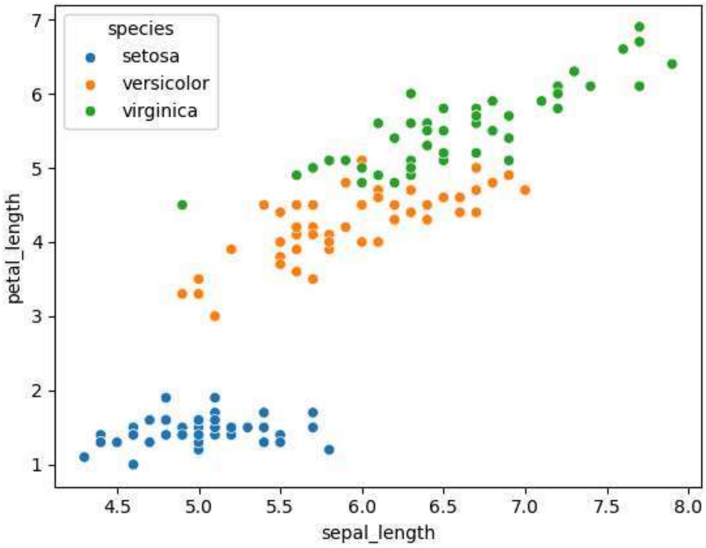
```
sns.scatterplot(x="sepal_length", y="sepal_width", data=df)
```

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



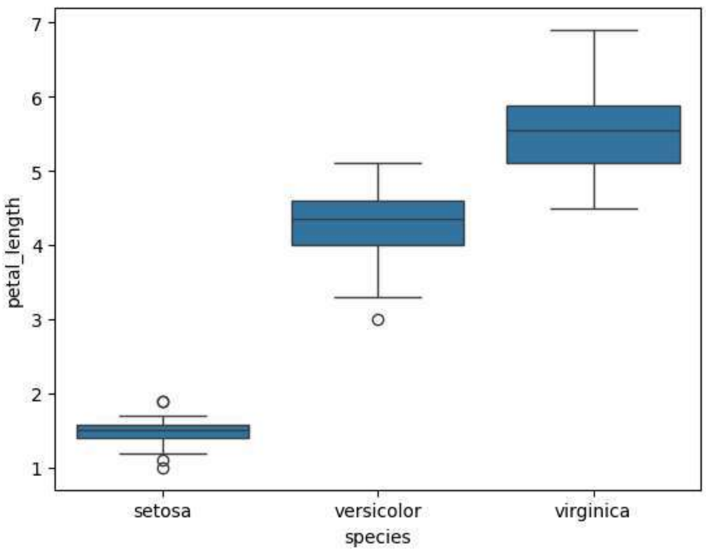

```
sns.scatterplot(x ="sepal_length", y = "petal_length", hue = "species", data = df)
```

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



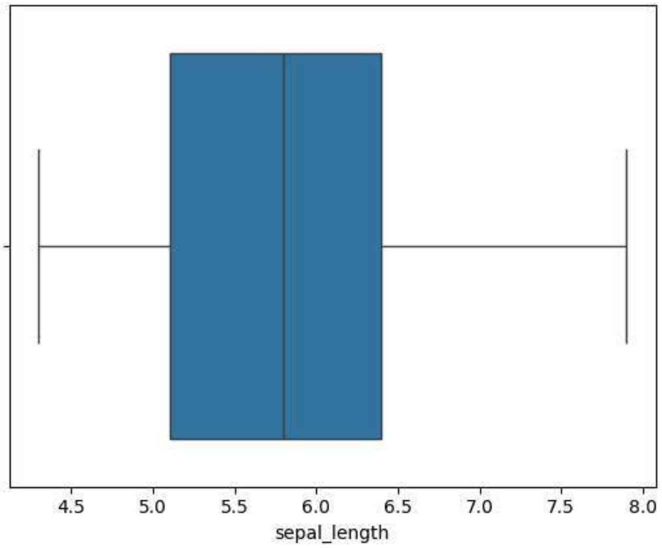

```
sns.boxplot (x = "species", y= "petal_length", data =df)
#interquartile range: Q3 - Q1
#outliers = 1.5 * IQR
```

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



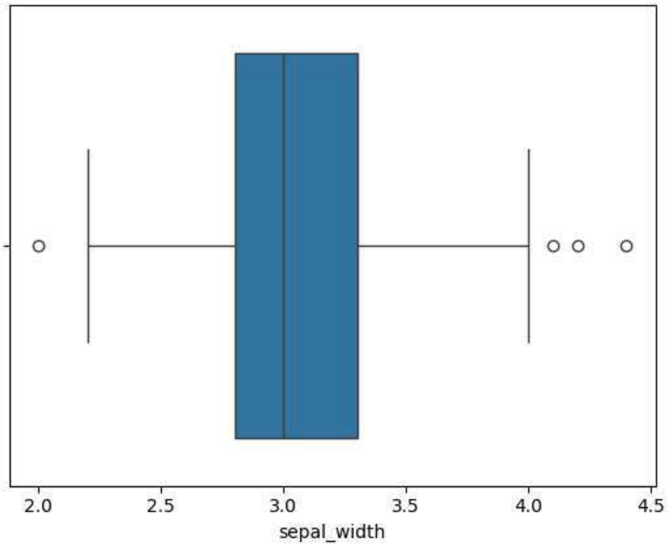
```
sns.boxplot(x = "sepal_length", data = df)
```

<Axes: xlabel='sepal_length'>



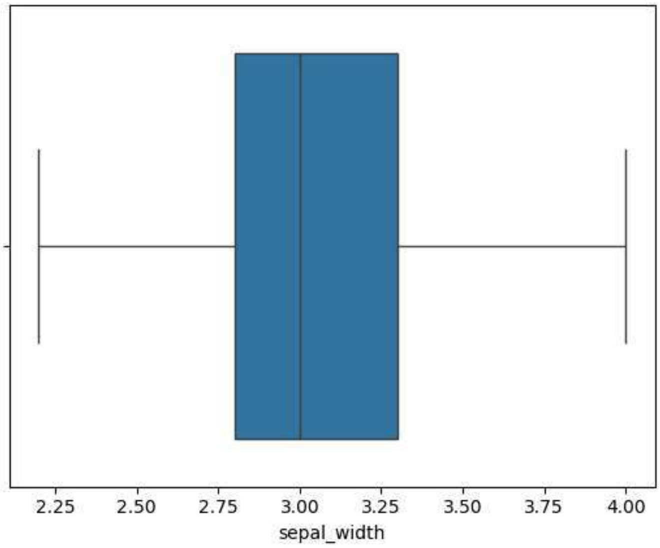

```
sns.boxplot(x = "sepal_width", data = df)
```

<Axes: xlabel='sepal_width'>



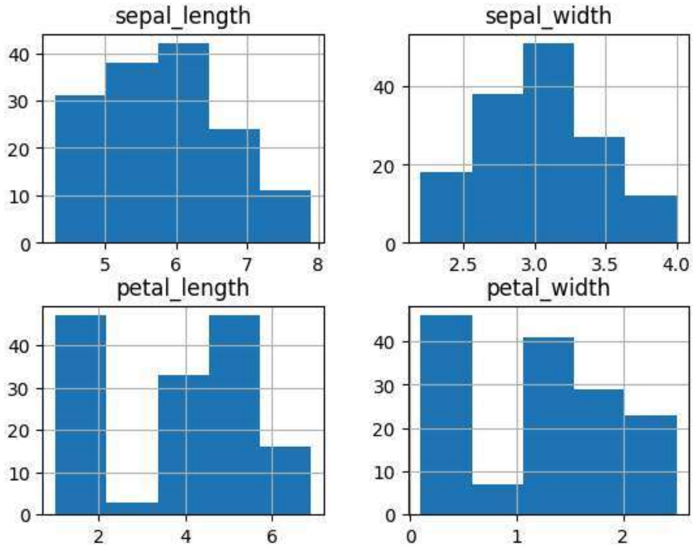

```
import sklearn
#from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
import numpy as np
# Load the dataset
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
# IQR
Q1= np.percentile (df['sepal_width'], 25, interpolation = 'midpoint')
Q3 = np.percentile (df['sepal_width'], 75, interpolation = 'midpoint')
IQR = Q3 - Q1
print("Old Shape: ", df.shape)
# Upper bound
upper = np. where (df['sepal_width'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(df['sepal_width'] <= (Q1-1.5*IQR))
#Removing the Outliers
df.drop(upper[0], inplace=True)
df.drop(lower[0], inplace=True)
print("New Shape: ", df.shape)
sns.boxplot(x='sepal_width', data=df)
```

```
Old Shape:  (150, 5)
New Shape:  (146, 5)
<Axes: xlabel='sepal_width'>
```



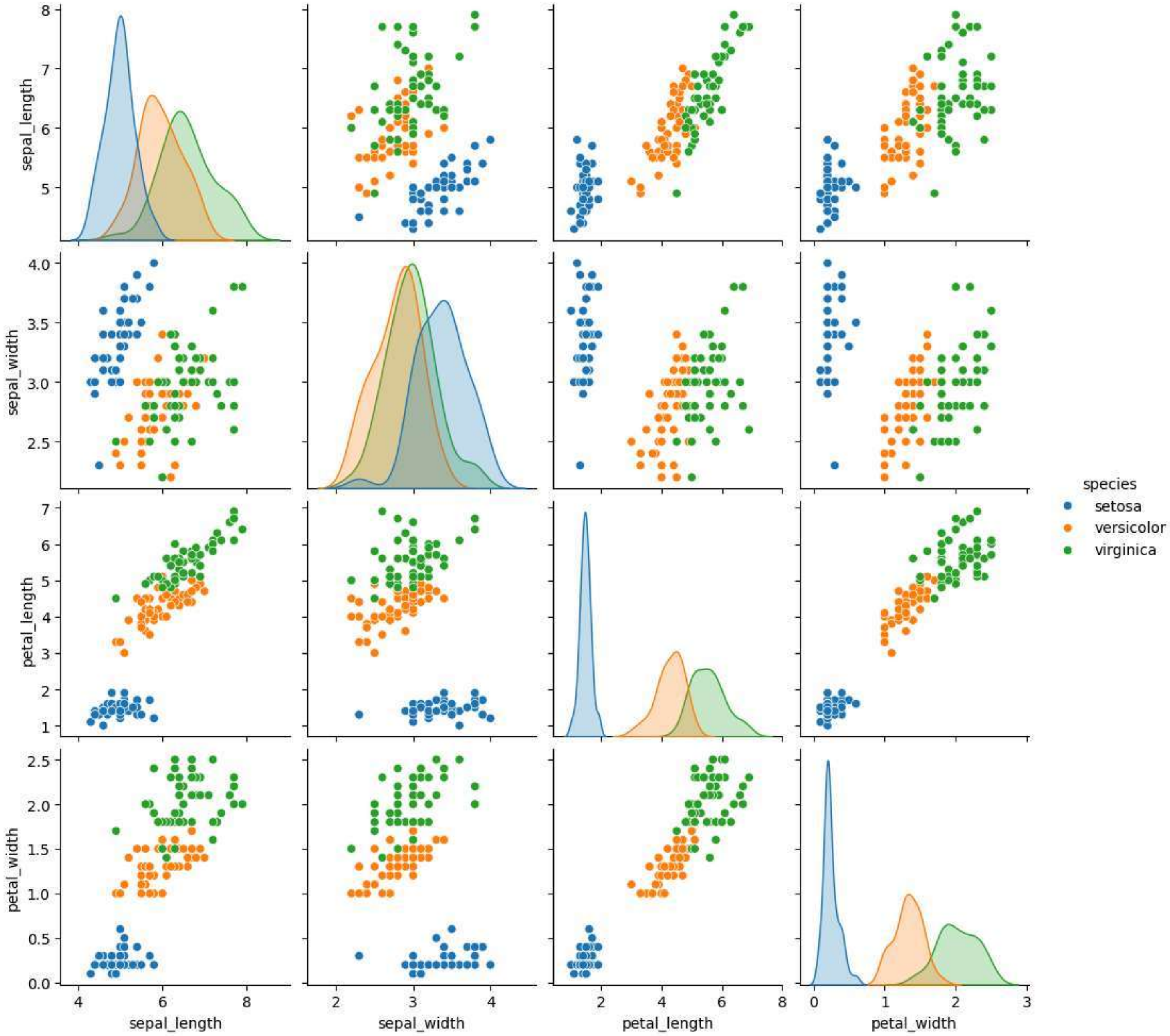

```
df.hist(bins=5)
```

```
array([[<Axes: title={'center': 'sepal_length'}>,
        <Axes: title={'center': 'sepal_width'}>],
       [<Axes: title={'center': 'petal_length'}>,
        <Axes: title={'center': 'petal_width'}>]], dtype=object)
```



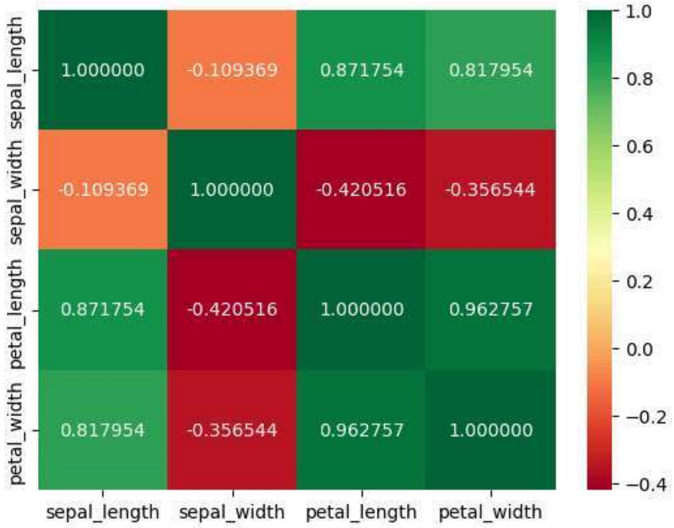

```
#pairplot scatter plot (multiple)
sns.pairplot(data = df, hue="species")
```

<seaborn.axisgrid.PairGrid at 0x787c2f13ed10>




```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
numeric_df = df.select_dtypes(include=[np.number])
corr_matrix = numeric_df.corr()
sns.heatmap(corr_matrix, cmap="RdYlGn", annot=True, fmt="f")
```

<Axes: >




```
#normalization and standarization
#data distribution
#standardize the dataframe
import pandas as pd
#create data frame
df = pd.DataFrame({'y': [8, 12, 15, 14, 19, 23, 25, 29],
                   'x1': [5, 7, 7, 9, 12, 9, 9, 4], 'x2': [11, 8, 10, 6, 6, 5, 9, 12],
                   'x3' :[2, 2, 3, 2, 5, 5, 7, 9]})
#view data frame
df
```

	y	x1	x2	x3
0	8	5	11	2
1	12	7	8	2
2	15	7	10	3
3	14	9	6	2
4	19	12	6	5
5	23	9	5	5
6	25	9	9	7
7	29	4	12	9

```
#standardize the values in each column, rescaling of the data (0-1) #mean, std
#v' = v - mean / std
df_new = (df-df.mean())/df.std()
#view new data frame
print(df_new)
```

```
          y        x1        x2        x3
0 -1.418032 -1.078639  1.025393 -0.908151
1 -0.857822 -0.294174 -0.146485 -0.908151
2 -0.437664 -0.294174  0.634767 -0.525772
3 -0.577717  0.490290 -0.927736 -0.908151
4  0.122546  1.666987 -0.927736  0.238987
5  0.682756  0.490290 -1.318362  0.238987
6  0.962861  0.490290  0.244141  1.003746
7  1.523071 -1.470871  1.416019  1.768505
```

```
print(df_new.mean())
print(df_new.std())
```

```
y     0.000000e+00
x1    2.775558e-17
x2   -1.387779e-17
x3    5.551115e-17
dtype: float64
y     1.0
x1    1.0
x2    1.0
x3    1.0
dtype: float64
```

```
#define predictor variable columns
df_x=df[['x1', 'x2', 'x3']]
#standardize the values for each predictor variable
df[['x1', 'x2', 'x3']] = (df_x-df_x.mean())/df_x.std()
#view new data frame
df
```

	y	x1	x2	x3
0	8	-1.078639	1.025393	-0.908151
1	12	-0.294174	-0.146485	-0.908151
2	15	-0.294174	0.634767	-0.525772
3	14	0.490290	-0.927736	-0.908151
4	19	1.666987	-0.927736	0.238987
5	23	0.490290	-1.318362	0.238987
6	25	0.490290	0.244141	1.003746
7	29	-1.470871	1.416019	1.768505

```
import pandas as pd
#create DataFrame
df = pd.DataFrame({'points': [25, 12, 15, 14, 19], 'assists': [5, 7, 7, 9, 12], 'rebounds':[11, 8, 10, 6, 6]})
#view DataFrame
print(df)
```

```
   points  assists  rebounds
0      25        5        11
1      12        7         8
2      15        7        10
3      14        9         6
4      19       12         6
```

```
(df-df.min())/(df.max()-df.min())
```

	points	assists	rebounds
0	1.000000	0.000000	1.0
1	0.000000	0.285714	0.4
2	0.230769	0.285714	0.8
3	0.153846	0.571429	0.0
4	0.538462	1.000000	0.0

```
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
pd.crosstab(index=df['species'], columns=df['sepal_length'], margins = True)
```

sepal_length	4.3	4.4	4.5	4.6	4.7	4.8	4.9	5.0	5.1	5.2	...	6.9	7.0	7.1	7.2	7.3	7.4	7.6	7.7	7.9	All
species																					
setosa	1	3	1	4	2	5	4	8	8	3	...	0	0	0	0	0	0	0	0	0	50
versicolor	0	0	0	0	0	0	1	2	1	1	...	1	1	0	0	0	0	0	0	0	50
virginica	0	0	0	0	0	0	1	0	0	0	...	3	0	1	3	1	1	1	4	1	50
All	1	3	1	4	2	5	6	10	9	4	...	4	1	1	3	1	1	1	4	1	150

4 rows × 36 columns

Data Distribution using sklearn

```
from sklearn import datasets
from sklearn. preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
# loading the Iris data
iris = datasets.load_iris()
print(iris)
X = iris.data # array for the features
y = iris.target # array for the target
feature_names = iris.feature_names # feature names
target_names = iris.target_names
# target names
```

```
# spliting the data into training and testing data sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=2020)
X_train
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
        [4.8, 3. , 1.4, 0.3],
        [5.1, 3.8, 1.6, 0.2],
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
        [6.4, 3.2, 4.5, 1.5],
        [6.9, 3.1, 4.9, 1.5],
        [5.5, 2.3, 4. , 1.3],
        [6.5, 2.8, 4.6, 1.5],
        [5.7, 2.8, 4.5, 1.3],
        [6.3, 3.3, 4.7, 1.6],
        [4.9, 2.4, 3.3, 1. ],
```

```
# normalization on partitioned data using sklearn package
normZ = StandardScaler()
X_train_Z = normZ.fit_transform(X_train)
X_test_Z = normZ.transform(X_test)
```

```
X_train_Z.mean (axis=0)
```

```
array([-1.97672566e-15,  3.88578059e-16,  3.64787565e-17, -9.19899078e-17])
```

```
X_train_Z.std (axis=0)
```

```
array([1., 1., 1., 1.])
```

```
X_test_Z.mean (axis=0)
X_test_Z.std (axis=0)
```

```
array([0.83359705, 0.85365417, 1.00126802, 0.9868196 ])
```

```
#normalization using min_max method
normMinMax = MinMaxScaler()
X_train_MinMax = normMinMax.fit_transform(X_train)
X_test_MinMax = normMinMax.transform (X_test)
```