Irish Data Set

In [32]:
```
Steps:
    1.Importing libraries.
    2.Load Irish Data Set.
    3.visualize data.
    4.Applying Label encoding.
    5.Perform feature scaling.
    6.spliting features.
    7.split data for train and test.
    8.Perform evaluation metrics.
    9.use different classification techniques.
    10.Finally finding accuracy of our data set.
```

```
  File "C:\Users\K P COMPUTERS\AppData\Local\Temp\ipykernel_50296\2401622090.
py", line 1
    Steps:
          ^
SyntaxError: invalid syntax
```

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading IRISH DataSet

In [2]:
```
df=pd.read_csv('irish.csv')
df
```

Out[2]:

	sepal length	sepal width	petal length	petal width	flower type
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

Reading first five rows of the data set

In [3]:
```
df.head()
```

Out[3]:

	sepal length	sepal width	petal length	petal width	flower type
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

Reading last five rows of the dataset

In [4]:
```
df.tail()
```

Out[4]:

	sepal length	sepal width	petal length	petal width	flower type
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

Reading first ten rows of the dataset

In [5]:
```
df.head(10)
```

Out[5]:

	sepal length	sepal width	petal length	petal width	flower type
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
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

Reading rows randomly

In [6]:
```
df.sample(10)
```

Out[6]:

	sepal length	sepal width	petal length	petal width	flower type
135	7.7	3.0	6.1	2.3	Iris-virginica
44	5.1	3.8	1.9	0.4	Iris-setosa
51	6.4	3.2	4.5	1.5	Iris-versicolor
114	5.8	2.8	5.1	2.4	Iris-virginica
79	5.7	2.6	3.5	1.0	Iris-versicolor
78	6.0	2.9	4.5	1.5	Iris-versicolor
16	5.4	3.9	1.3	0.4	Iris-setosa
27	5.2	3.5	1.5	0.2	Iris-setosa
143	6.8	3.2	5.9	2.3	Iris-virginica
67	5.8	2.7	4.1	1.0	Iris-versicolor

Reading column names of the dataset

In [7]:
```
df.columns
```

Out[7]:
```
Index(['sepal length', 'sepal width', ' petal length ', 'petal width',
       ' flower type'],
      dtype='object')
```

Reading datatypes of each column of the dataset

In [8]:
```
df.dtypes
```

Out[8]:
```
sepal length      float64
sepal width       float64
 petal length     float64
petal width       float64
 flower type       object
dtype: object
```

Reading number of rows and columns in dataset

In [9]:
```
df.shape
```

Out[9]:
```
(150, 5)
```

Apply slicing

In [10]:
```
a=df[10:20]
print(a)
```

	sepal length	sepal width	petal length	petal width	flower type
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

Displaying particular columns with specific number of rows

In [11]:
```
col=['sepal length','sepal width',' flower type']
print(df[col].head(10))
```

	sepal length	sepal width	flower type
0	5.1	3.5	Iris-setosa
1	4.9	3.0	Iris-setosa
2	4.7	3.2	Iris-setosa
3	4.6	3.1	Iris-setosa
4	5.0	3.6	Iris-setosa
5	5.4	3.9	Iris-setosa
6	4.6	3.4	Iris-setosa
7	5.0	3.4	Iris-setosa
8	4.4	2.9	Iris-setosa
9	4.9	3.1	Iris-setosa

Getting rows using specific condition

In [12]:
```
df.loc[df[' flower type']=='Iris-setosa']
```

Out[12]:

	sepal length	sepal width	petal length	petal width	flower type
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
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa
20	5.4	3.4	1.7	0.2	Iris-setosa
21	5.1	3.7	1.5	0.4	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa
23	5.1	3.3	1.7	0.5	Iris-setosa
24	4.8	3.4	1.9	0.2	Iris-setosa
25	5.0	3.0	1.6	0.2	Iris-setosa
26	5.0	3.4	1.6	0.4	Iris-setosa
27	5.2	3.5	1.5	0.2	Iris-setosa
28	5.2	3.4	1.4	0.2	Iris-setosa
29	4.7	3.2	1.6	0.2	Iris-setosa
30	4.8	3.1	1.6	0.2	Iris-setosa
31	5.4	3.4	1.5	0.4	Iris-setosa
32	5.2	4.1	1.5	0.1	Iris-setosa
33	5.5	4.2	1.4	0.2	Iris-setosa
34	4.9	3.1	1.5	0.2	Iris-setosa
35	5.0	3.2	1.2	0.2	Iris-setosa

	sepal length	sepal width	petal length	petal width	flower type
36	5.5	3.5	1.3	0.2	Iris-setosa
37	4.9	3.6	1.4	0.1	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
39	5.1	3.4	1.5	0.2	Iris-setosa
40	5.0	3.5	1.3	0.3	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
43	5.0	3.5	1.6	0.6	Iris-setosa
44	5.1	3.8	1.9	0.4	Iris-setosa
45	4.8	3.0	1.4	0.3	Iris-setosa
46	5.1	3.8	1.6	0.2	Iris-setosa
47	4.6	3.2	1.4	0.2	Iris-setosa
48	5.3	3.7	1.5	0.2	Iris-setosa
49	5.0	3.3	1.4	0.2	Iris-setosa

In [13]:
```
df.loc[df[' flower type']=='Iris-setosa'].head(6)
```

Out[13]:

	sepal length	sepal width	petal length	petal width	flower type
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
5	5.4	3.9	1.7	0.4	Iris-setosa

In [14]:
```
a=df.loc[df[' flower type']=='Iris-virginica']
a.tail(2)
```

Out[14]:

	sepal length	sepal width	petal length	petal width	flower type
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Displaying number of rows and columns contains iris-virginica

In [15]:
```
a.shape
```

Out[15]:
```
(50, 5)
```

Display number of categories in specific column

In [16]:
```
df[' flower type'].value_counts()
```

Out[16]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name:  flower type, dtype: int64
```

Counting mean value of the particular column

In [17]:
```
a=df['petal width'].mean()
print('Mean:',a)
```

```
Mean: 1.199333333333334
```

Counting median value of the particular column

In [18]:
```
b=df['petal width'].median()
print('Median:',b)
```

```
Median: 1.3
```

Finding minimum value of the particular column

In [19]:
```
min_petal=df['petal width'].min()
print('Minimum petal width:',min_petal)
```

```
Minimum petal width: 0.1
```

Finding maximum values of the particular column

In [20]:
```
max_petal=df['petal width'].max()
print('Maximum petal width:',max_petal)
```

```
Maximum petal width: 2.5
```

Creating new column

In [21]:
```
col=df.columns
df1=df[col]
df['total_val']=df1[col].sum(axis=1)
df
```

```
C:\Users\K P COMPUTERS\AppData\Local\Temp\ipykernel_29116\1403859833.py:3: Fu
tureWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  df['total_val']=df1[col].sum(axis=1)
```

Out[21]:

	sepal length	sepal width	petal length	petal width	flower type	total_val
0	5.1	3.5	1.4	0.2	Iris-setosa	10.2
1	4.9	3.0	1.4	0.2	Iris-setosa	9.5
2	4.7	3.2	1.3	0.2	Iris-setosa	9.4
3	4.6	3.1	1.5	0.2	Iris-setosa	9.4
4	5.0	3.6	1.4	0.2	Iris-setosa	10.2
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica	17.2
146	6.3	2.5	5.0	1.9	Iris-virginica	15.7
147	6.5	3.0	5.2	2.0	Iris-virginica	16.7
148	6.2	3.4	5.4	2.3	Iris-virginica	17.3
149	5.9	3.0	5.1	1.8	Iris-virginica	15.8

150 rows × 6 columns

Renaming column names

In [22]:
```
df.rename(columns={'sepal length':'SEPAL LENGTH','sepal width':'SEPAL WIDTH','
```

Out[22]:

	SEPAL LENGTH	SEPAL WIDTH	petal length	PETAL WIDTH	FLOWER TYPE	TOTAL VALUE
0	5.1	3.5	1.4	0.2	Iris-setosa	10.2
1	4.9	3.0	1.4	0.2	Iris-setosa	9.5
2	4.7	3.2	1.3	0.2	Iris-setosa	9.4
3	4.6	3.1	1.5	0.2	Iris-setosa	9.4
4	5.0	3.6	1.4	0.2	Iris-setosa	10.2
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica	17.2
146	6.3	2.5	5.0	1.9	Iris-virginica	15.7
147	6.5	3.0	5.2	2.0	Iris-virginica	16.7
148	6.2	3.4	5.4	2.3	Iris-virginica	17.3
149	5.9	3.0	5.1	1.8	Iris-virginica	15.8

150 rows × 6 columns

Applying style to the dataset

In [23]:
```
df.style
```

Out[23]:

	sepal length	sepal width	petal length	petal width	flower type	total_val
0	5.100000	3.500000	1.400000	0.200000	Iris-setosa	10.200000
1	4.900000	3.000000	1.400000	0.200000	Iris-setosa	9.500000
2	4.700000	3.200000	1.300000	0.200000	Iris-setosa	9.400000
3	4.600000	3.100000	1.500000	0.200000	Iris-setosa	9.400000
4	5.000000	3.600000	1.400000	0.200000	Iris-setosa	10.200000
5	5.400000	3.900000	1.700000	0.400000	Iris-setosa	11.400000
6	4.600000	3.400000	1.400000	0.300000	Iris-setosa	9.700000
7	5.000000	3.400000	1.500000	0.200000	Iris-setosa	10.100000
8	4.400000	2.900000	1.400000	0.200000	Iris-setosa	8.900000
9	4.900000	3.100000	1.500000	0.100000	Iris-setosa	9.600000
10	5.400000	3.700000	1.500000	0.200000	Iris-setosa	10.800000
11	4.800000	3.400000	1.600000	0.200000	Iris-setosa	10.000000

Applying color to the maximum row of the dataset

In [24]:
```
df.head(10).style.highlight_max(color='yellow',axis=0)
```

Out[24]:

	sepal length	sepal width	petal length	petal width	flower type	total_val
0	5.100000	3.500000	1.400000	0.200000	Iris-setosa	10.200000
1	4.900000	3.000000	1.400000	0.200000	Iris-setosa	9.500000
2	4.700000	3.200000	1.300000	0.200000	Iris-setosa	9.400000
3	4.600000	3.100000	1.500000	0.200000	Iris-setosa	9.400000
4	5.000000	3.600000	1.400000	0.200000	Iris-setosa	10.200000
5	5.400000	3.900000	1.700000	0.400000	Iris-setosa	11.400000
6	4.600000	3.400000	1.400000	0.300000	Iris-setosa	9.700000
7	5.000000	3.400000	1.500000	0.200000	Iris-setosa	10.100000
8	4.400000	2.900000	1.400000	0.200000	Iris-setosa	8.900000
9	4.900000	3.100000	1.500000	0.100000	Iris-setosa	9.600000

Identifying how many number of null values in each column

In [25]:
```
df.isna().sum()
```

Out[25]:
```
sepal length      0
sepal width       0
 petal length     0
petal width       0
 flower type      0
total_val         0
dtype: int64
```

In [26]:
```
df.isna()
```

Out[26]:

	sepal length	sepal width	petal length	petal width	flower type	total_val
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

In [27]:
```
df.isna().sum().sum()
```

Out[27]:
```
0
```

EXPLORATORY DATA ANALYSIS

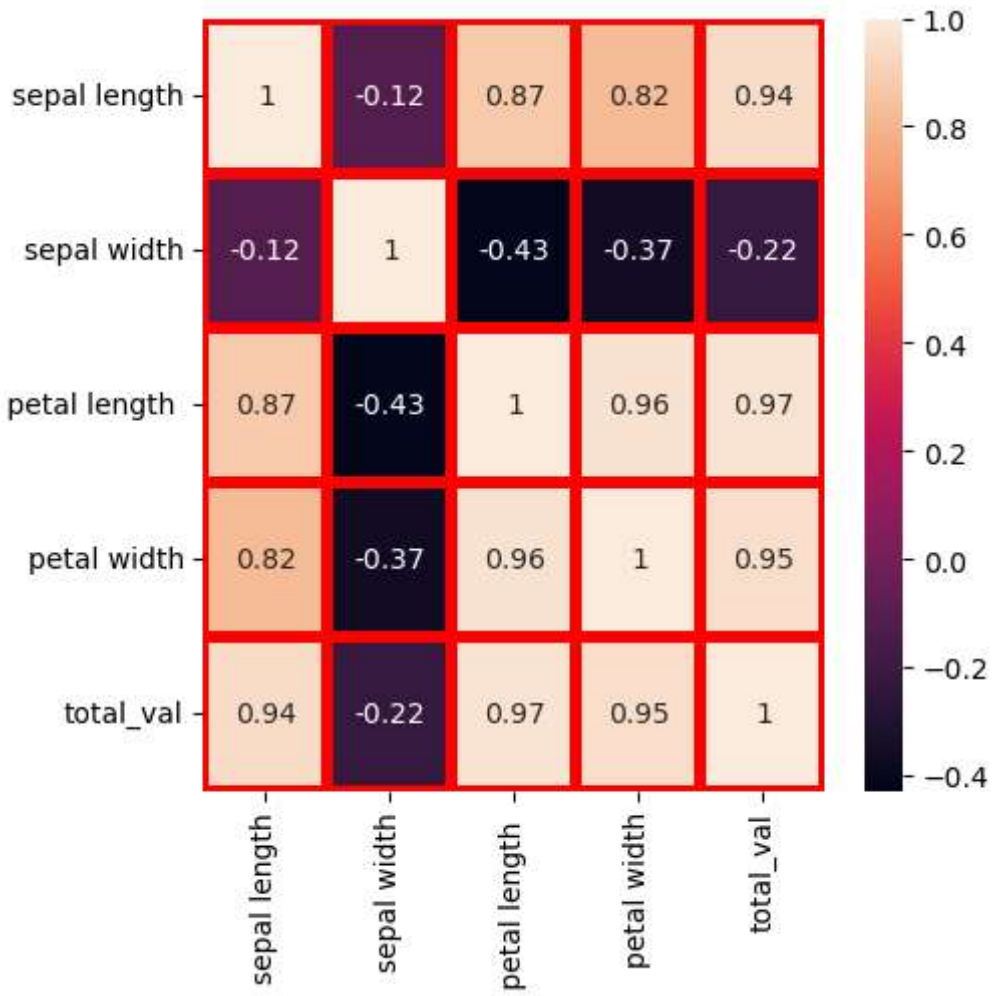
In [ ]:
```
Exploratory data analysis is one of the major step to fine-tune the given data
analysis to understand the insights of the key characteristics of the column,r
numpy and statistical methods.
```

Heat map

In [ ]:
```
A Heat map is a graphical representation of multivariate data that is structur
and columns.
   Heat map is very usefull in describing correlation among several numerical
```

In [28]:
```
plt.figure(figsize=(5,5))
sns.heatmap(df.corr(),linecolor='red',linewidths=3,annot=True)
```

Out[28]: <AxesSubplot:>



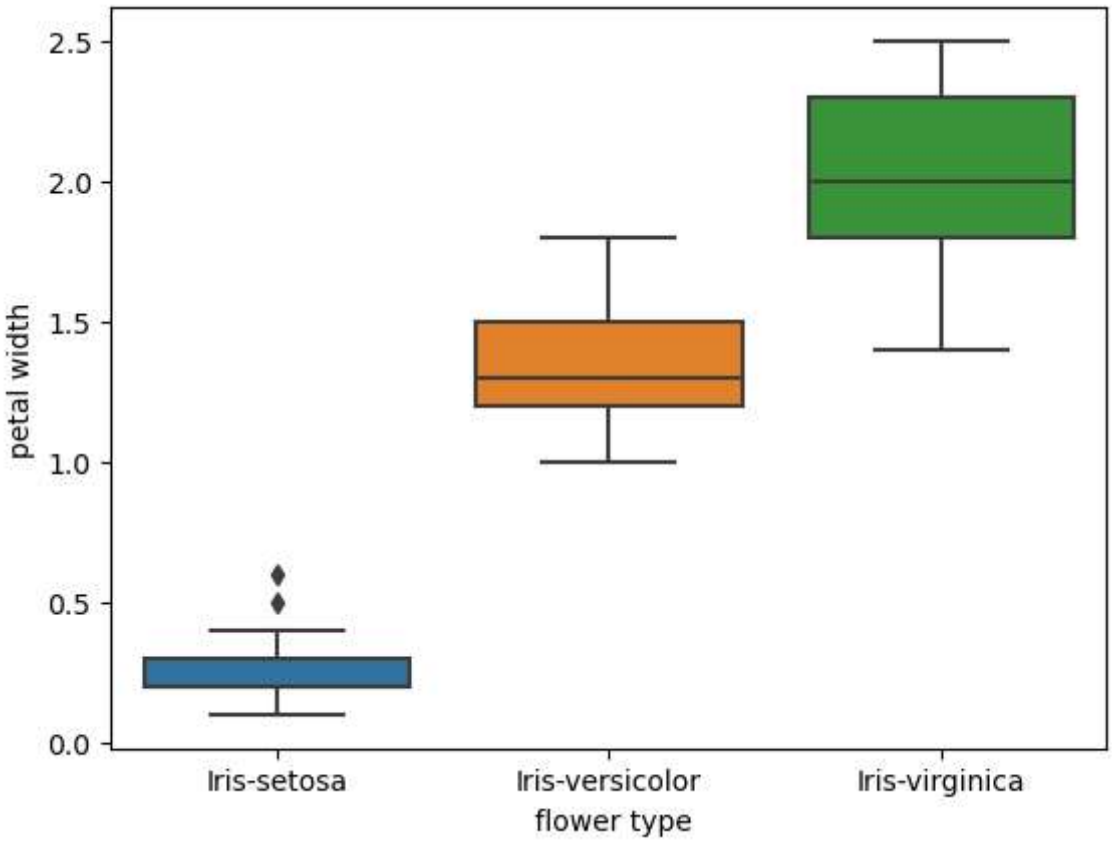
Showing different types of graphs

Box plot

In [ ]:
```
Box plot displays the five number summary of a set of data.The five-number sum
first quartile,median,third quartile,and maximum.
     In box plot we draw a box from first quartile to third quatile.A vertical
box at the median.
      Box plot is mainly used for identifying the outliers.Outliers are the da
above the data limit.
```

In [29]:
```
sns.boxplot(x=' flower type',y='petal width',data=df)
```

Out[29]: <AxesSubplot:xlabel=' flower type', ylabel='petal width'>

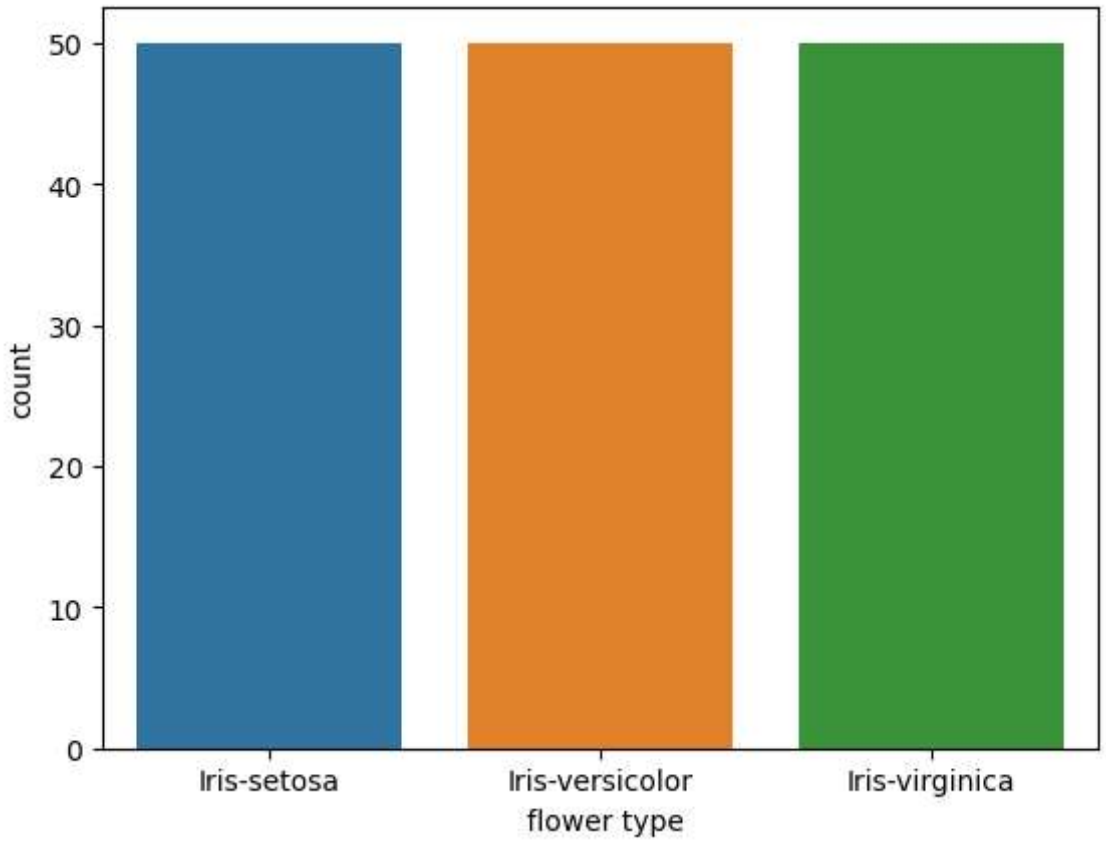


Count plot

In [ ]:
```
Count plot is used for data visualizing.It shows the observational count in di
bins with the help of bars.
```

In [30]:
```
sns.countplot(x=' flower type',data=df)
```

Out[30]:
```
<AxesSubplot:xlabel=' flower type', ylabel='count'>
```

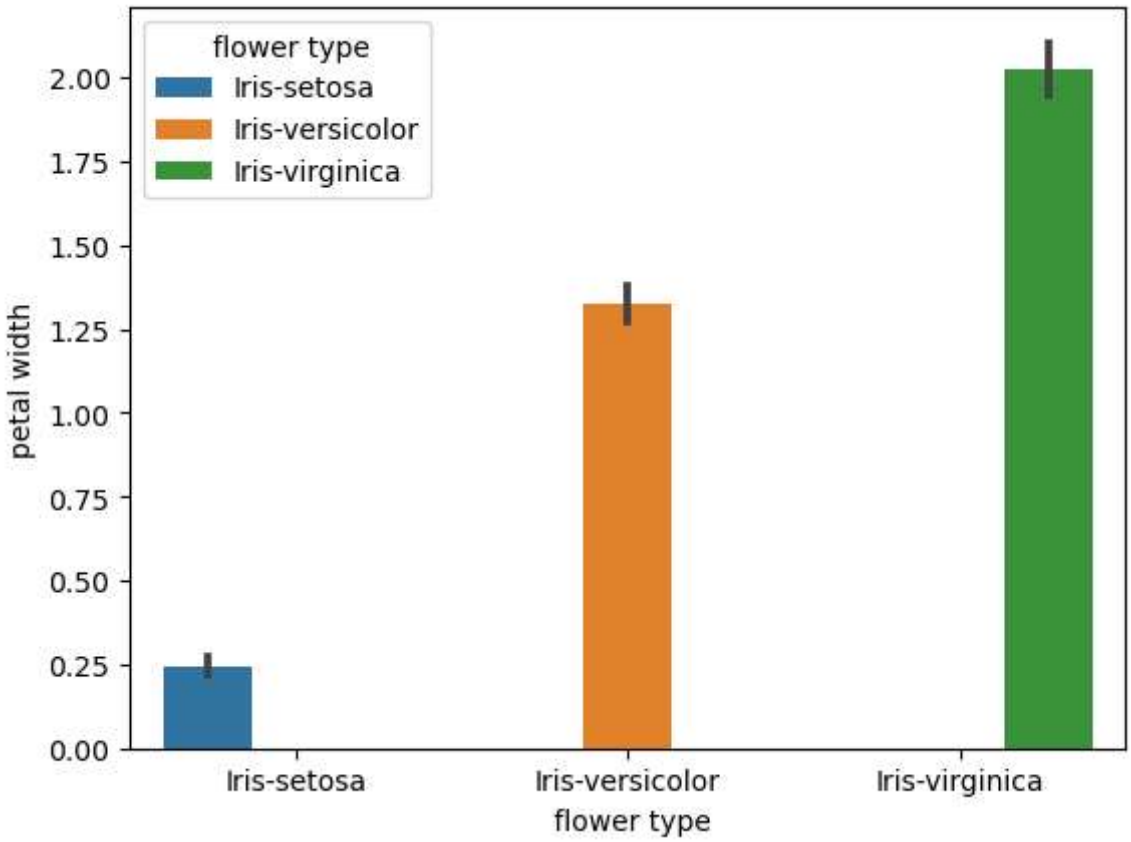


Bar plot

In [ ]:
```
Bar plot is also used for data visualization.Bar plot can be plot horizontally
    Bar plot describe the comparison between the discrete categories.Both the a
display different categories which are compared.
```

In [31]:
```
sns.barplot(x=' flower type',y='petal width',data=df,hue=' flower type')
```

Out[31]:
```
<AxesSubplot:xlabel=' flower type', ylabel='petal width'>
```

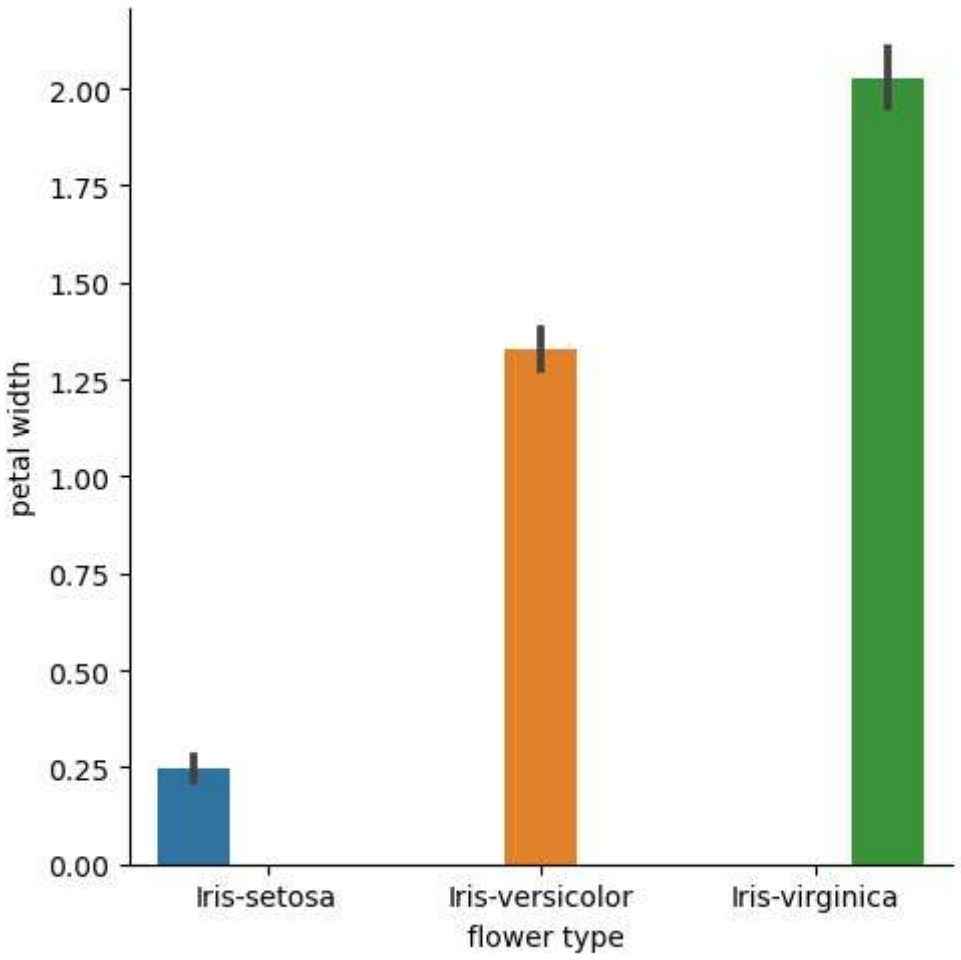


Cat plot

In [ ]:
```
Cat plot can handle 8 different plots currently available in seaborn.Cat plot
these types of plots and one needs to specify the type of plot one needs with
     Cat plot show the relationship between one or more categorical variables
using one of the several visual representations.
```

In [32]:
```
sns.catplot(x=' flower type',y='petal width',data=df,kind='bar',hue=' flower t
```

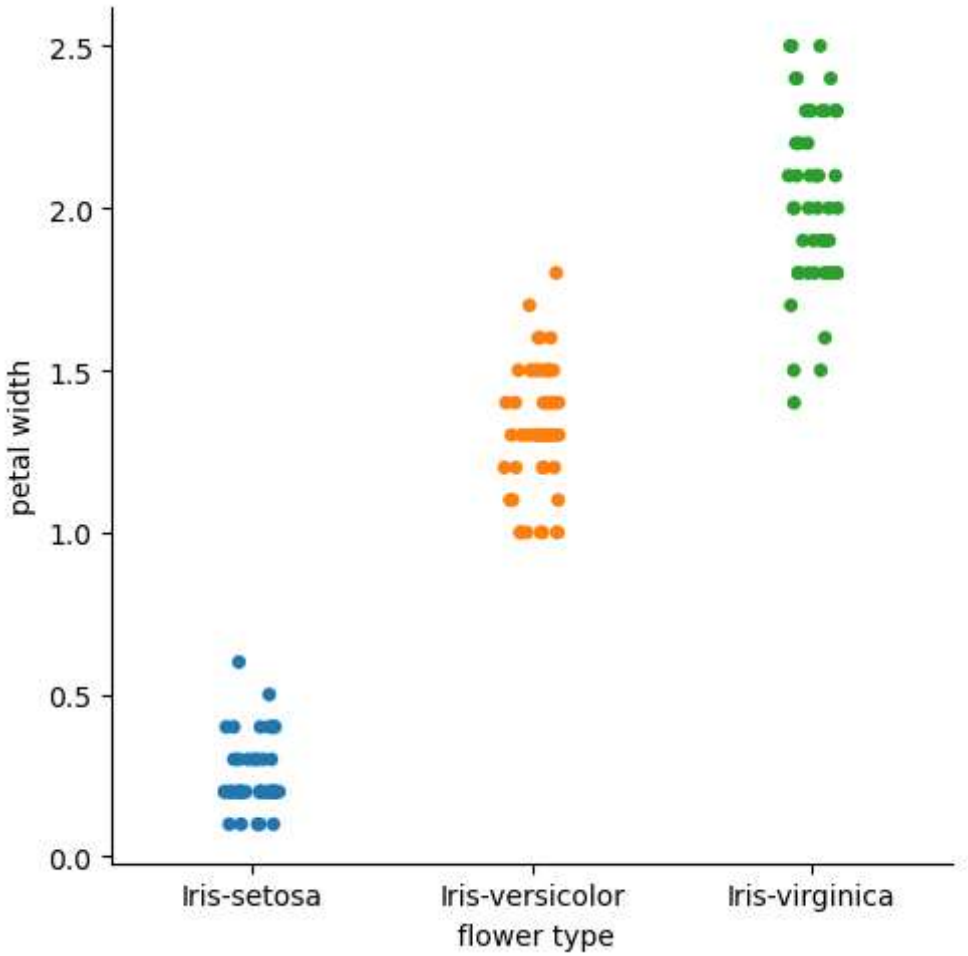
Out[32]:
```
<seaborn.axisgrid.FacetGrid at 0x27d340ffdc0>
```



Cat plot

In [33]:
```
sns.catplot(x=' flower type',y='petal width',data=df,hue=' flower type')
```

Out[33]: <seaborn.axisgrid.FacetGrid at 0x27d342ad910>

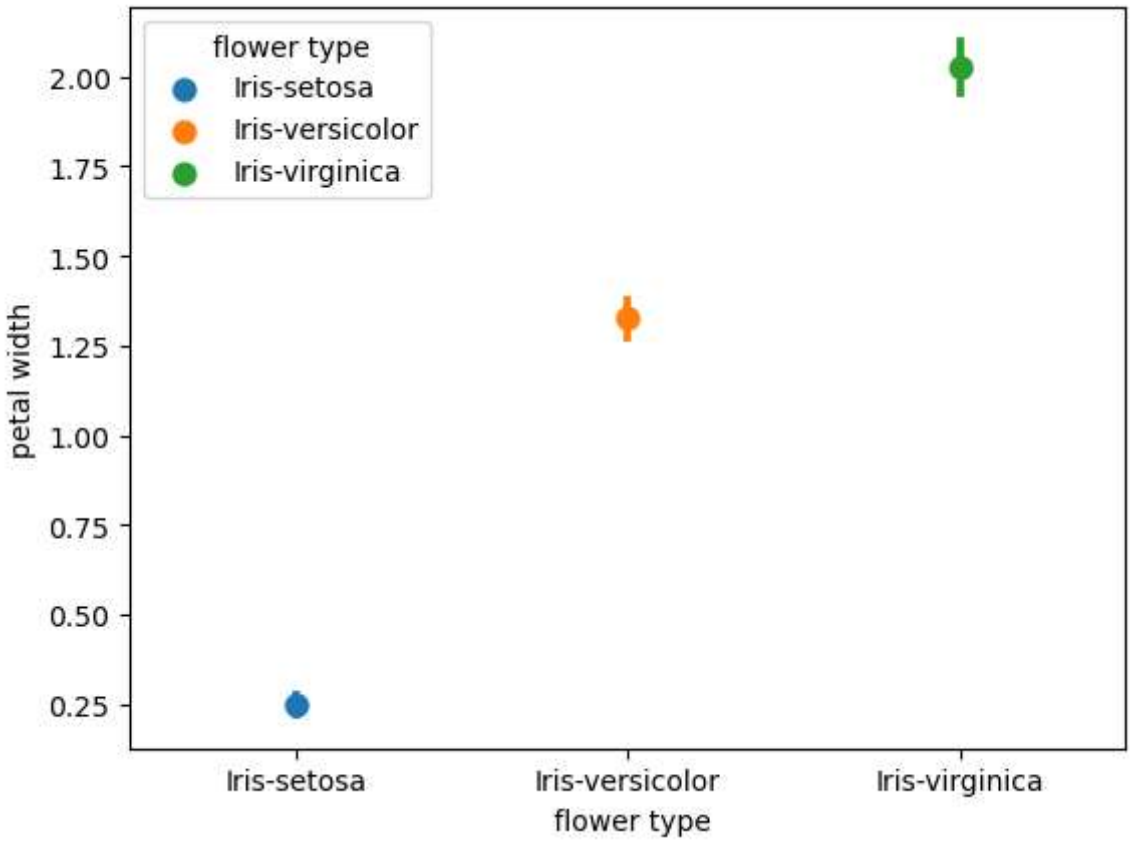


Point plot

In [ ]:
```
Point plot represents an estimate of central tendency for numerical variable b
scatter plot points.
```

In [34]:
```
sns.pointplot(x=' flower type',y='petal width',data=df,hue=' flower type')
```

Out[34]:
```
<AxesSubplot:xlabel=' flower type', ylabel='petal width'>
```

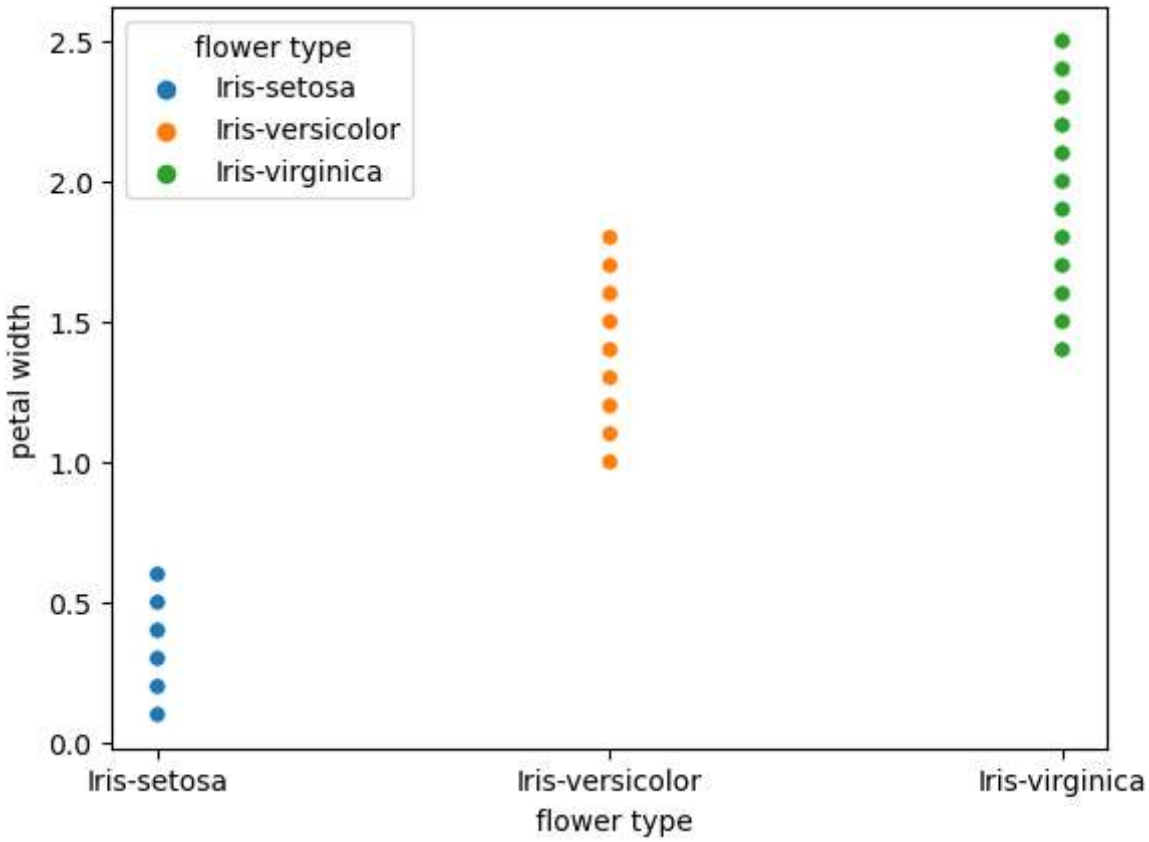


Scatter plot

In [ ]:
```
In scatter plot each value of the dataset is represented by a dot.Scatter plot
variable is affected by another.Scatter plots very much like line plots that tl
vertical data points.
```

In [35]:
```
sns.scatterplot(x=' flower type',y='petal width',data=df,hue=' flower type')
```

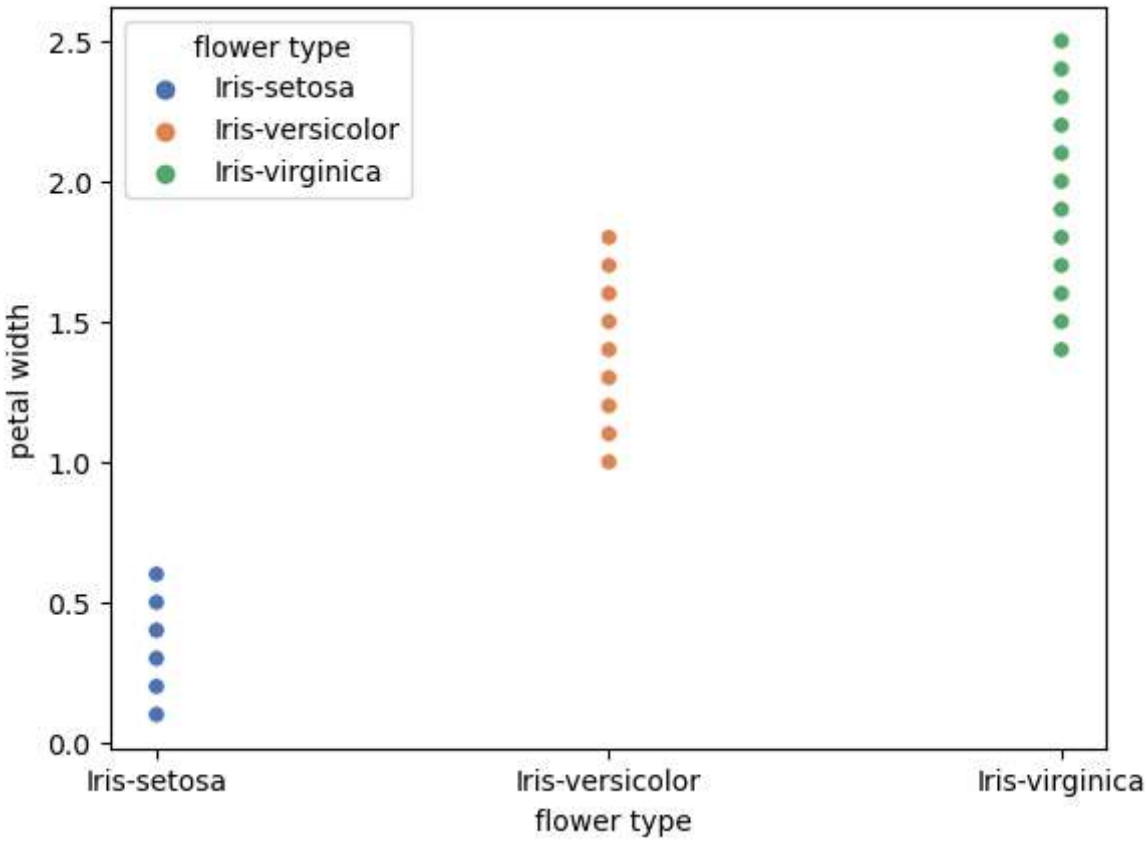
Out[35]:
```
<AxesSubplot:xlabel=' flower type', ylabel='petal width'>
```



Scatter plot

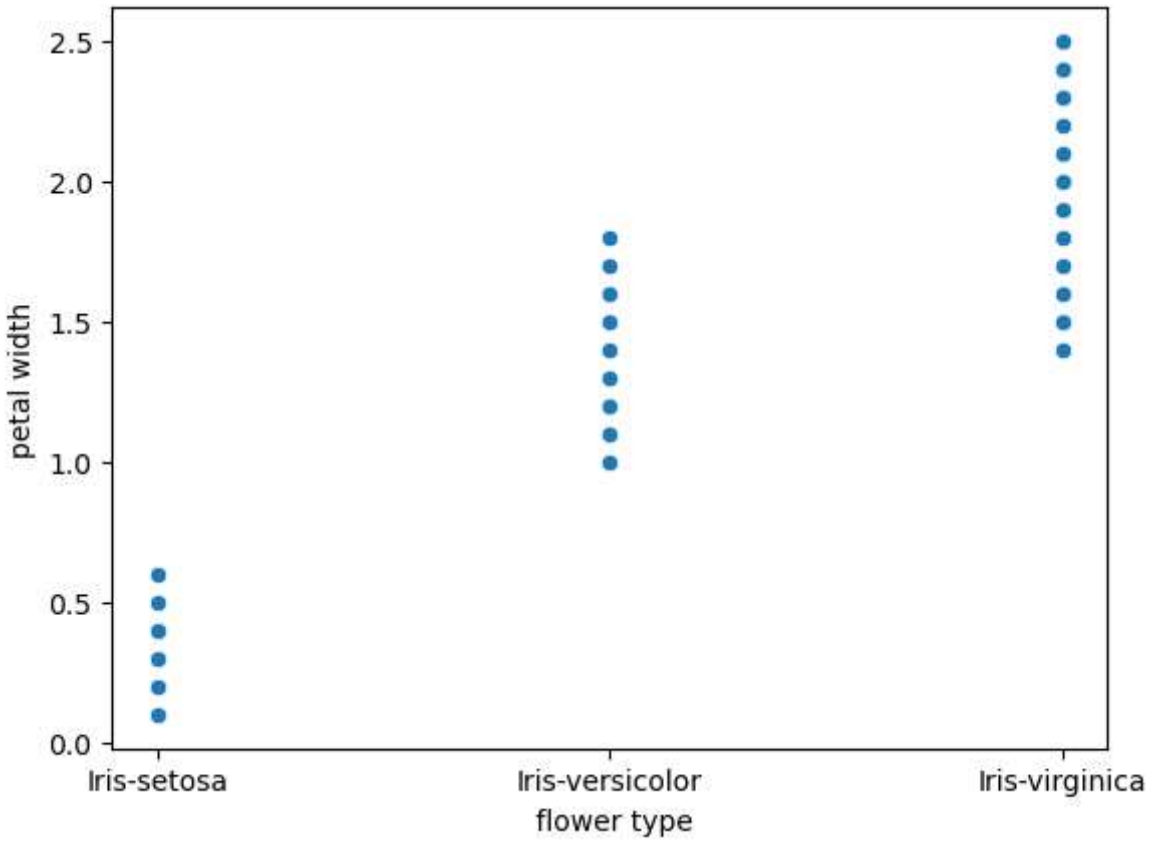
In [36]:
```
sns.scatterplot(x=' flower type',y='petal width',data=df,palette='deep',hue='
```

Out[36]:
```
<AxesSubplot:xlabel=' flower type', ylabel='petal width'>
```



In [37]:
```
sns.scatterplot(x=' flower type',y='petal width',data=df,palette='deep')
```

Out[37]:
```
<AxesSubplot:xlabel=' flower type', ylabel='petal width'>
```

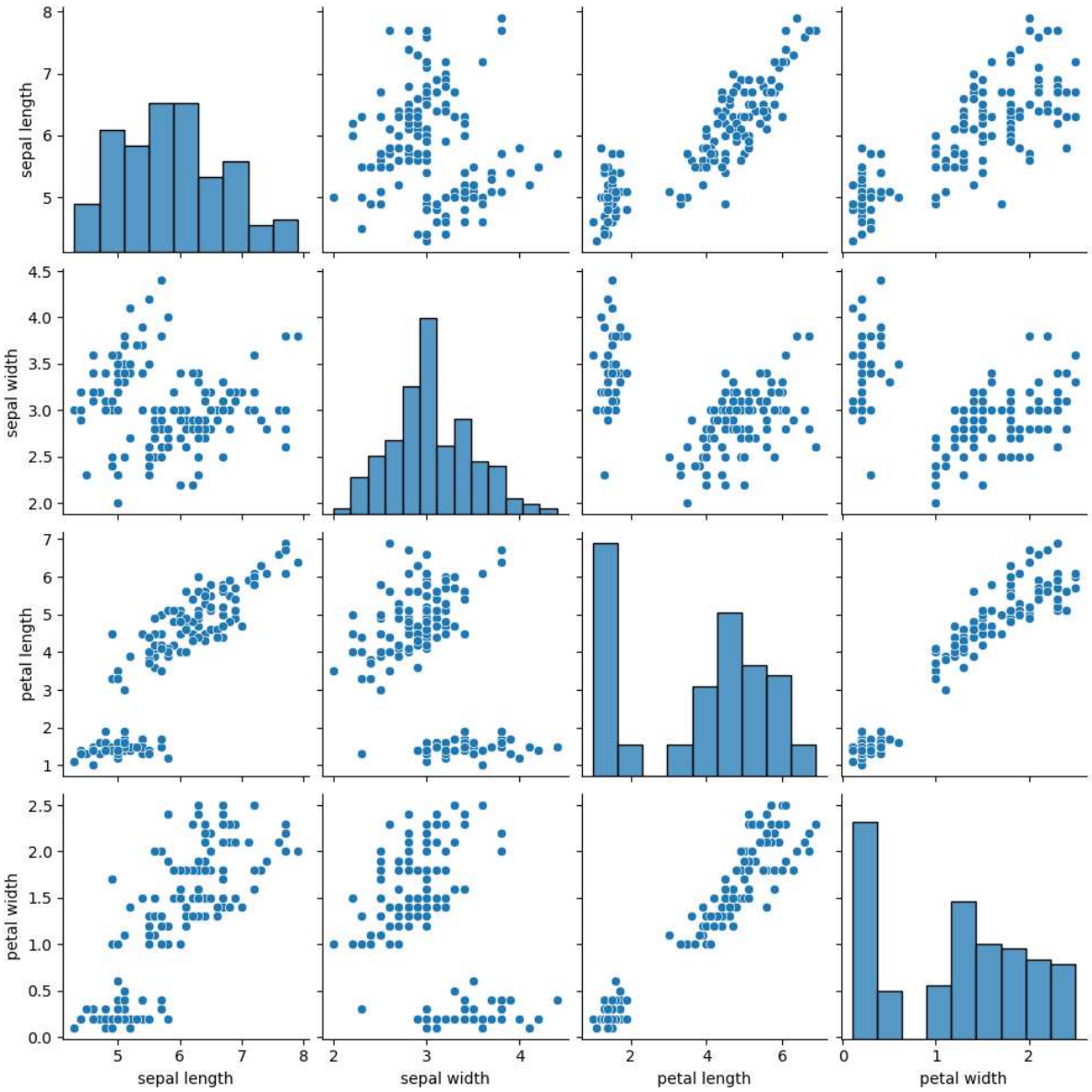


Pair plots

In [ ]:
```
The default pair plots in seaborn only plots through numerical columns.Pair pl
plots histogram and scatterplot.
```

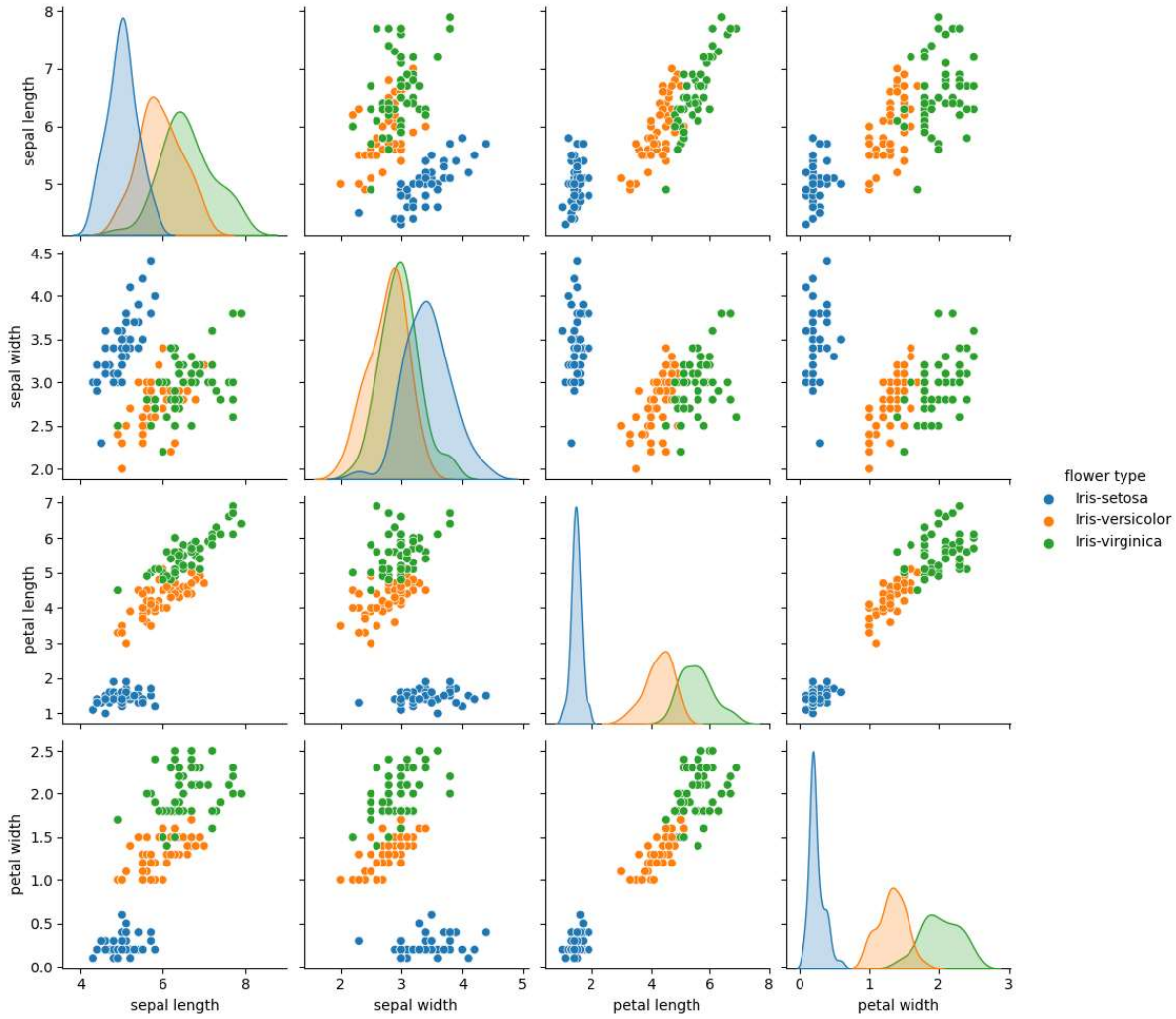
In [38]:
```
import pandas as pd
import seaborn as sns
df=pd.read_csv('irish.csv')
sns.pairplot(df)
```

Out[38]: <seaborn.axisgrid.PairGrid at 0x27d34315f10>



In [38]:
```
sns.pairplot(df,hue=' flower type')
```

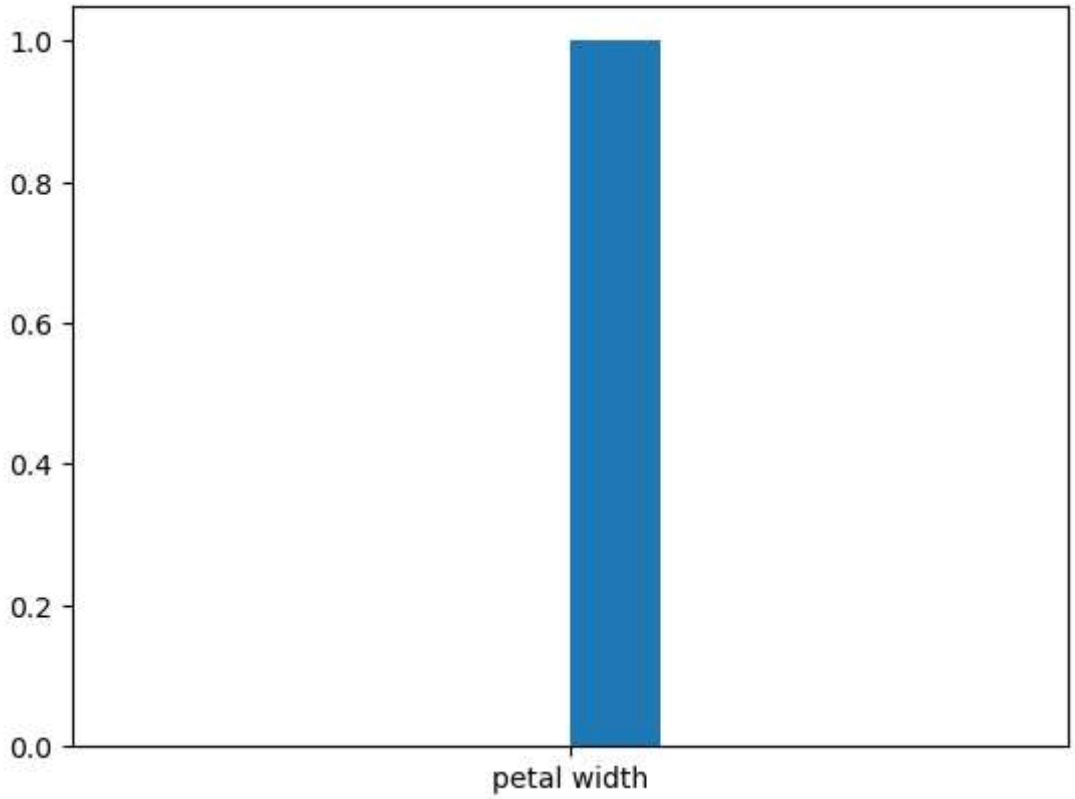
Out[38]:
```
<seaborn.axisgrid.PairGrid at 0x1d42bc30580>
```



Histogram

In [ ]:
```
A histogram is basically used to represent the data in the form of groups.It i
representing numeriical data.It is a type of bar plot where x-axis represents
y-axis represents information about the frequency.
```

In [39]:
```
x='petal width'
plt.hist(x)
plt.show()
```

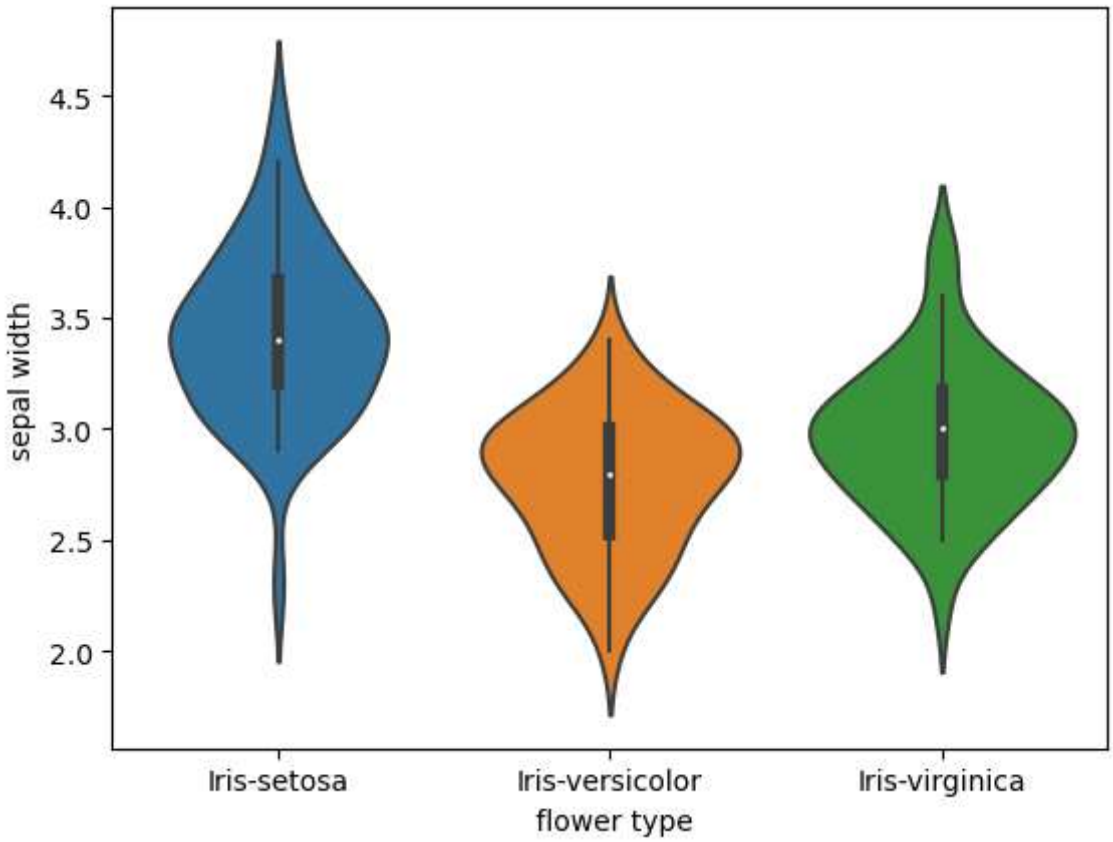


Violin plot

In [ ]:
```
A violin plot plays similar activity as box plot.it shows several quantitative
categorical variables.It can effective and attractive way to show multiple dat
numpy and pandas but pandas are more preferable.
```

In [40]:
```
sns.violinplot(x=' flower type',y='sepal width',data=df)
```

Out[40]:
```
<AxesSubplot:xlabel=' flower type', ylabel='sepal width'>
```



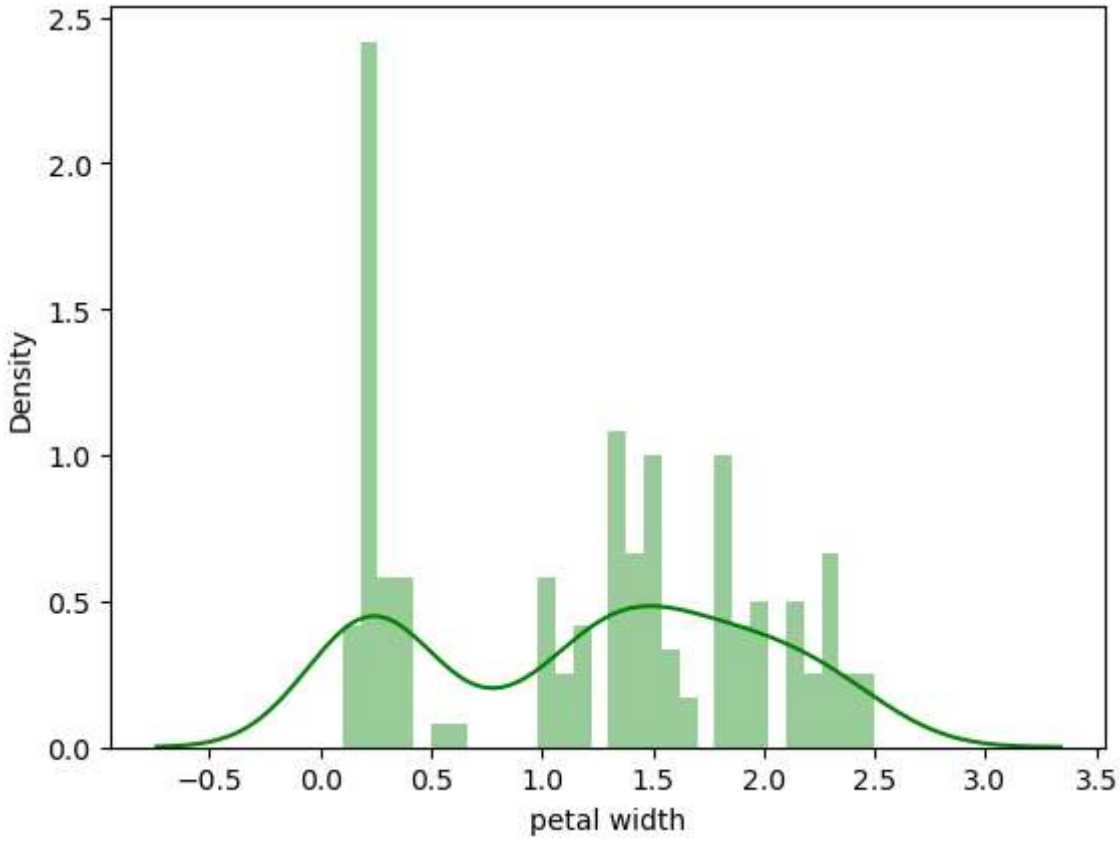
Displot

In [ ]:
```
Displot is basically used for unvariant set of observations and visualizes it
Only we have to choose one particular column of the data set.
```

In [41]:
```
sns.distplot(df['petal width'],color='green',bins=30)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[41]: <AxesSubplot:xlabel='petal width', ylabel='Density'>



In [3]:
```
df1=pd.read_csv('irish.csv')
df1.head()
```

Out[3]:

	sepal length	sepal width	petal length	petal width	flower type
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

In [ ]:
```
Label encoding is used to convert categorical column into numerical column usi
encoding will be done using map().
     In One Hot Encoding there chance of data redundancy but in label encoding
redundancy.
```

In [4]:
```
df1[' flower type']=df1[' flower type'].map({'Iris-setosa':1,'Iris-virginica':
df1.head()
```

Out[4]:

	sepal length	sepal width	petal length	petal width	flower type
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

In [5]:
```
df1.tail()
```

Out[5]:

	sepal length	sepal width	petal length	petal width	flower type
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

Feature Scaling

In [ ]:
```
 Feature Scaling can be performed before training our model.Data set contains
contains different values.If our model perform well before we must do Feature
values of the features into unique.This will help our model perform well.
     There are different feature scaling techniques,they are:
               1.MinMaxScaler
               2.StandardScaler
               3.RobustScaler
```

In [6]:
```
from sklearn.preprocessing import RobustScaler
scaler=RobustScaler()
df1_scaled=scaler.fit_transform(df1)
dfa=pd.DataFrame(df1_scaled,columns=df1.columns)
dfa
```

Out[6]:

	sepal length	sepal width	petal length	petal width	flower type
0	-0.538462	1.0	-0.842857	-0.733333	-0.5
1	-0.692308	0.0	-0.842857	-0.733333	-0.5
2	-0.846154	0.4	-0.871429	-0.733333	-0.5
3	-0.923077	0.2	-0.814286	-0.733333	-0.5
4	-0.615385	1.2	-0.842857	-0.733333	-0.5
...	...	...	...	...	...
145	0.692308	0.0	0.242857	0.666667	0.0
146	0.384615	-1.0	0.185714	0.400000	0.0
147	0.538462	0.0	0.242857	0.466667	0.0
148	0.307692	0.8	0.300000	0.666667	0.0
149	0.076923	0.0	0.214286	0.333333	0.0

150 rows × 5 columns

Split Train and Test Columns

In [7]:
```
x=df1.drop([' flower type'],axis=1)
y=df1[' flower type']
x.head()
```

Out[7]:

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [8]:
```
y.head()
```

Out[8]:
```
0    1
1    1
2    1
3    1
4    1
Name:  flower type, dtype: int64
```

Finding train and test data percentage

In [ ]:
```
```

In [9]:
```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=0.2,random_state=
print('total points in train_x:',train_x.shape)
print('total points in test_x:',test_x.shape)
print('total points in train_y:',train_y.shape)
print('total points in test_y:',test_y.shape)
```

```
total points in train_x: (120, 4)
total points in test_x: (30, 4)
total points in train_y: (120,)
total points in test_y: (30,)
```

In [10]:
```
print('total percentage in train_x:',(train_x.shape[0]/df1.shape[0])*100)
print('total percentage in test_x:',(test_x.shape[0]/df1.shape[0])*100)
print('total percentage in train_y:',(train_y.shape[0]/df1.shape[0])*100)
print('total percentage in test_y:',(test_y.shape[0]/df1.shape[0])*100)
```

```
total percentage in train_x: 80.0
total percentage in test_x: 20.0
total percentage in train_y: 80.0
total percentage in test_y: 20.0
```

Correlation

In [12]:
```
corr=x.corr()
print(corr)
top_features=corr.index
print(top_features)
plt.figure(figsize=(10,10))
```

```
                sepal length  sepal width   petal length   petal width
sepal length        1.000000    -0.117570       0.871754     0.817941
sepal width        -0.117570     1.000000      -0.428440    -0.366126
 petal length       0.871754    -0.428440       1.000000     0.962865
petal width         0.817941    -0.366126       0.962865     1.000000
Index(['sepal length', 'sepal width', ' petal length ', 'petal width'], dtype
='object')
```

Out[12]:
```
<Figure size 1000x1000 with 0 Axes>
```

```
<Figure size 1000x1000 with 0 Axes>
```

In [51]:
```
sns.heatmap(x[top_features].corr(),annot=True)
```

Out[51]:
```
<AxesSubplot:>
```

Distribution of data points

In [13]:
```
train_y.value_counts()/len(train_y)
```

Out[13]:
```
1    0.358333
3    0.325000
2    0.316667
Name:  flower type, dtype: float64
```

In [14]:
```
test_y.value_counts()/len(test_y)
```

Out[14]:
```
2    0.400000
3    0.366667
1    0.233333
Name:  flower type, dtype: float64
```

Hold out cross validation

In [ ]:
```
 In hold out validation divide data set into train,test and validation.This ho
used to check how well a machine learning model perform on the new data.
```

In [15]:
```
train_xx,test_x,train_yy,test_y=train_test_split(x,y,test_size=0.2,random_stat
train_x,val_x,train_y,val_y=train_test_split(train_xx,train_yy,test_size=0.2,r
print(train_x.shape)
print(test_x.shape)
print(val_x.shape)
print(val_y.shape)
print(train_y.shape)
print(test_y.shape)
```

```
(96, 4)
(30, 4)
(24, 4)
(24,)
(96,)
(30,)
```

Using KneighborsClassifier

In [ ]:
```
KNeighborsClassifier is also called as a KNN.It is a supervised machine learni
      KNN is used for both regression and classification problems but mostly u
problem.when new data enters using KNN algoorithm we can easily classifiy the
suitable.
```

In [55]:
```
from sklearn.neighbors import KNeighborsClassifier as kn
from sklearn.metrics import accuracy_score,f1_score
def knn_elbow(k):
    test_error=[]
    for i in k:
        cla=kn(n_neighbors=i)
        cla.fit(train_x,train_y)
        val_temp=cla.predict(test_x)
        accuracy=accuracy_score(val_temp,test_y)
        error=1-accuracy
        test_error.append(error)
    return test_error
k=range(4,30,3)
test_result=knn_elbow(k)
print(test_result)
```

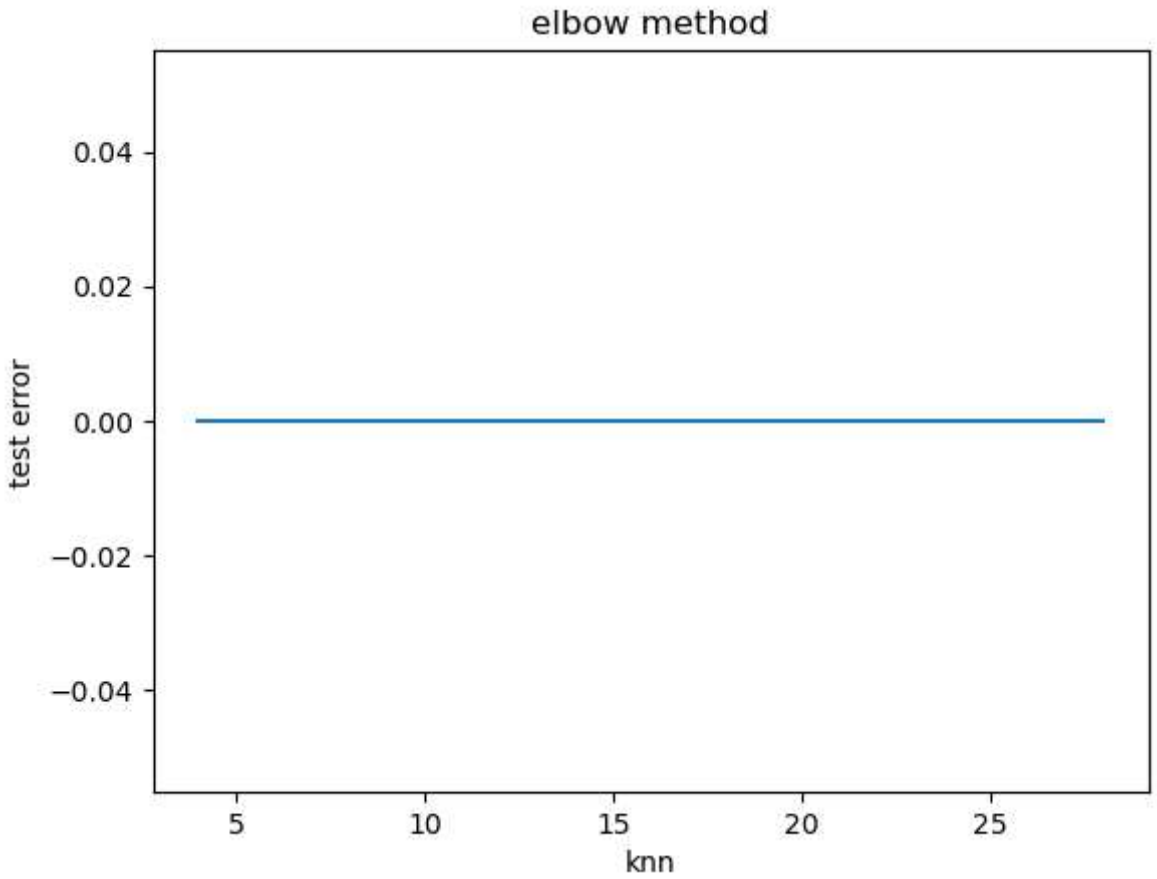
```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
```

```
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [56]:
```
plt.plot(k,test_result)
plt.xlabel('knn')
plt.ylabel('test error')
plt.title('elbow method')
```

Out[56]:
```
Text(0.5, 1.0, 'elbow method')
```



Logistic Regression

In [ ]:
```
   Logistic Regression is one of the most popular machine learning algorithm w
supervised machine learning technique.Logistic regression predict the output o
therefore the outcome must be categorical or discreate.
```

In [16]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(train_x,train_y)
y_pred=model.predict(test_x)
y_pred
```

Out[16]:
```
array([2, 2, 3, 2, 2, 1, 2, 1, 2, 2, 1, 2, 3, 1, 3, 1, 3, 2, 1, 3, 1, 3,
       2, 1, 1, 3, 3, 3, 1, 3], dtype=int64)
```

accuracy_score

In [ ]:
```
accuracy_score is one type evaluation metrics.It is used to find the accuracy 
the model.
     formula=(TP+TN)/(TP+TN+FP+FN)
```

In [20]:
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(test_y,y_pred)*100
print('accuracy of the model {:.2f}'.format(accuracy))
```

```
accuracy of the model 100.00
```

confusion_matrix

In [ ]:
```
Confusion matrix is also type of evaluation metrix.It is constructed based on
how many no of categories in targeted column.
```

In [22]:
```
from sklearn.metrics import confusion_matrix
accuracy1=confusion_matrix(test_y,y_pred)
print('confusion matrix is:',accuracy1)
```

```
confusion matrix is: [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

SVM

In [ ]:
```
SVM stands for support vector machine.It is a supervised machine learning
technique used for classification problem. using svm we can easily put the new
data into correct category.
```

In [65]:
```
from sklearn.svm import SVC
model1=SVC()
model1.fit(train_x,train_y)
pred_y=model1.predict(test_x)
accuracy1=accuracy_score(pred_y,test_y)
print('accuracy of the model using svm:',accuracy1*100)
```

```
accuracy of the model using svm: 100.0
```

DecisionTreeClassifier

In [ ]:
```
DecisionTreeClassifier is a type of supervised machine learning technique.It i
used to make predictions based on how previous set of questions were answered.
```

In [67]:
```
from sklearn.tree import DecisionTreeClassifier
model2=DecisionTreeClassifier()
model2.fit(train_x,train_y)
pred1_y=model2.predict(test_x)
accuracy2=accuracy_score(pred1_y,test_y)
print('accuracy of the model using decision tree classifier:',accuracy2*100)
```

```
accuracy of the model using decision tree classifier: 100.0
```

In [69]:
```
result=pd.DataFrame({'model name':['logistic regression','KNN','Support Vector
result
```

Out[69]:

	model name	Accuracy Score
0	logistic regression	100
1	KNN	100
2	Support Vector Machine	100
3	Decision Tree Classifier	100

Conclusion

In [ ]:
```
Finally,we got accuracy 100%,which shows that the model built is very accurate
```

In [ ]:
```
Thank you sir.
```