In [2]:
```
# 1.Read data from the cvs file
# 1st method
import pandas as pd
data=pd.read_csv('C:/Users/DELL/Downloads/Data.csv')
data
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780
568	92751	B	7.76	24.54	47.92	181.0	0.05263

569 rows × 32 columns

In [ ]:
```
```

In [5]:
```
# 2nd method
path="C:/Users/DELL/Downloads/Data.csv"
data=pd.read_csv(path, encoding='utf-8')
data
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780
568	92751	B	7.76	24.54	47.92	181.0	0.05263

569 rows × 32 columns

In [6]:
```
# 2.Dimensions of the data
data.shape
```

Out[6]:
```
(569, 32)
```

In [7]:
```
data.ndim
```

Out[7]:
```
2
```

In [8]:
```
# 3.Display data(top 5 rows and total data)
data.head()
```

Out[8]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

5 rows × 32 columns

In [9]:
```
data.tail()
```

Out[9]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780
568	92751	B	7.76	24.54	47.92	181.0	0.05263

5 rows × 32 columns

In [10]:
```
data.head(10)
```

Out[10]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030
5	843786	M	12.45	15.70	82.57	477.1	0.12780
6	844359	M	18.25	19.98	119.60	1040.0	0.09463
7	84458202	M	13.71	20.83	90.20	577.9	0.11890
8	844981	M	13.00	21.82	87.50	519.8	0.12730
9	84501001	M	12.46	24.04	83.97	475.9	0.11860

10 rows × 32 columns

In [11]:
```
data.tail(10)
```

Out[11]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
559	925291	B	11.51	23.93	74.52	403.5	0.09261
560	925292	B	14.05	27.15	91.38	600.4	0.09929
561	925311	B	11.20	29.37	70.67	386.0	0.07449
562	925622	M	15.22	30.62	103.40	716.9	0.10480
563	926125	M	20.92	25.09	143.00	1347.0	0.10990
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780
568	92751	B	7.76	24.54	47.92	181.0	0.05263

10 rows × 32 columns

In [12]:
```
# 4. List the column names of a data frame
data.columns
```

Out[12]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [13]:
```
list_1=list(data)
print(list_1)
print(type(list_1))
```

```
['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'sm
oothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetr
y_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'area_s
e', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry
_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'ar
ea_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave point
s_worst', 'symmetry_worst', 'fractal_dimension_worst']
<class 'list'>
```

In [14]:
```
# 5. Change columns of a data frame
#Rename the id columnn to 'serialno'
data = data.rename (columns={"id": "serialno"})

# Again, the inplace parameter will change the dataframe without assignment
data.rename (columns={"id": "serialno"}, inplace=True)
data
```

Out[14]:

	serialno	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780
568	92751	B	7.76	24.54	47.92	181.0	0.05263

569 rows × 32 columns

In [15]:
```
data.rename(
columns={
"serialno": "serial_no",
"diagnosis": "m/b"
},
inplace=True
)

# Rename all columns using a function, e.g. convert all column names to Lower case:
data.rename (columns=str.lower)
```

Out[15]:

	serial_no	m/b	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 32 columns

In [16]:
```
# 6. Display specific single column or multiple columns of a data frame
print(data['radius_mean'])
```

```
0      17.99
1      20.57
2      19.69
3      11.42
4      20.29
       ...
564    21.56
565    20.13
566    16.60
567    20.60
568     7.76
Name: radius_mean, Length: 569, dtype: float64
```

In [17]:
```
data.radius_mean
```

Out[17]:
```
0      17.99
1      20.57
2      19.69
3      11.42
4      20.29
       ...
564    21.56
565    20.13
566    16.60
567    20.60
568     7.76
Name: radius_mean, Length: 569, dtype: float64
```

In [18]:
```
data.iloc[:,2]
```

Out[18]:
```
0      17.99
1      20.57
2      19.69
3      11.42
4      20.29
       ...
564    21.56
565    20.13
566    16.60
567    20.60
568     7.76
Name: radius_mean, Length: 569, dtype: float64
```

In [24]:
```
# 7.Bind sets of rows of data frames
data_rbind_1 = pd.DataFrame({"x1": range(11, 16),
"x2":["keshav", "venky", "vamshi", "chetan", "vishnu"],
"x3": range(30, 25, -1),
"x4": range(30, 20, -2)})

print(data_rbind_1)
```

```
   x1      x2  x3  x4
0  11  keshav  30  30
1  12   venky  29  28
2  13  vamshi  28  26
3  14  chetan  27  24
4  15  vishnu  26  22
```

In [27]:
```
data_rbind_2 =pd.DataFrame({"x1": range (3, 10),
                            "x2":["x", "y", "y", "y", "x", "x", "y"],
                             "x3": range(20, 6, -2),
"x4": range(28, 21, - 1)})

print(data_rbind_2)
```

```
   x1 x2  x3  x4
0   3  x  20  28
1   4  y  18  27
2   5  y  16  26
3   6  y  14  25
4   7  x  12  24
5   8  x  10  23
6   9  y   8  22
```

In [30]:
```
# 8.Bind sets of columns of data frames
data_cbind_1 =pd.DataFrame({"x1": range(10, 16),
                            "x2": range(30, 24, -1),
                            "x3":["abc", "bcd", "cde", "def", "efg", "fgh"],
                            "x4": range (48, 42, -1)})
print(data_cbind_1)
```

```
   x1  x2   x3  x4
0  10  30  abc  48
1  11  29  bcd  47
2  12  28  cde  46
3  13  27  def  45
4  14  26  efg  44
5  15  25  fgh  43
```

In [31]:
```
data_cbind_2 = pd.DataFrame({"y1":["foo", "bar", "bar", "foo", "foo", "bar"], # Create
"y2":["x", "y", "z", "x", "y", "z"],
```

```
"y3": range(18, 0, - 3)})
print(data_cbind_2)
```

```
   y1 y2  y3
0  foo  x  18
1  bar  y  15
2  bar  z  12
3  foo  x   9
4  foo  y   6
5  bar  z   3
```

In [32]:
```
data_cbind_all = pd.concat([data_cbind_1.reset_index (drop = True),
                            data_cbind_2],
axis = 1)
print(data_cbind_all)
```

```
   x1  x2   x3  x4   y1 y2  y3
0  10  30  abc  48  foo  x  18
1  11  29  bcd  47  bar  y  15
2  12  28  cde  46  bar  z  12
3  13  27  def  45  foo  x   9
4  14  26  efg  44  foo  y   6
5  15  25  fgh  43  bar  z   3
```

In [34]:
```
# column-bind two pandas dataframes usimg ID column
data_merge_1 = pd.DataFrame({"ID": range (1, 5),
"x1": range(10, 14),
"x2": range(30, 26, -1),
"x3":["a", "b", "c", "d"], "x4": range(48, 44, -1)})
print(data_merge_1)
```

```
   ID  x1  x2 x3  x4
0   1  10  30  a  48
1   2  11  29  b  47
2   3  12  28  c  46
3   4  13  27  d  45
```

In [36]:
```
data_merge_2 = pd.DataFrame({ "ID": range(3, 9),
"y1":["abc", "iop", "qwe", "asd", "foo", "bar"],
"y2":["x", "y", "z", "x", "y", "z"],
"y3": range(18, 0, - 3)})
print(data_merge_2)
```

```
   ID   y1 y2  y3
0   3  abc  x  18
1   4  iop  y  15
2   5  qwe  z  12
3   6  asd  x   9
4   7  foo  y   6
5   8  bar  z   3
```

In [38]:
```
data_merge_all = pd.merge(data_merge_1,
data_merge_2,
on = "ID",
how ="outer")
print(data_merge_all)
```

```
   ID    x1    x2   x3    x4   y1   y2    y3
0   1  10.0  30.0    a  48.0  NaN  NaN   NaN
1   2  11.0  29.0    b  47.0  NaN  NaN   NaN
2   3  12.0  28.0    c  46.0  abc    x  18.0
3   4  13.0  27.0    d  45.0  iop    y  15.0
4   5   NaN   NaN  NaN   NaN  qwe    z  12.0
5   6   NaN   NaN  NaN   NaN  asd    x   9.0
6   7   NaN   NaN  NaN   NaN  foo    y   6.0
7   8   NaN   NaN  NaN   NaN  bar    z   3.0
```

In [39]:
```
# 9. Find missing values in the dataset
data.isnull()
```

Out[39]:

	serial_no	m/b	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
564	False	False	False	False	False	False	False	
565	False	False	False	False	False	False	False	
566	False	False	False	False	False	False	False	
567	False	False	False	False	False	False	False	
568	False	False	False	False	False	False	False	

569 rows × 32 columns

In [ ]:

Central Tendency

In [1]:
```
#to calculte mean
from collections import Counter
lt=[1,233,453,534,6,1,3,6,2,1,78,2,4,5,6,4,3,6]
n=len(lt)
s=sum(lt)
m=s/n
print("Mean or average of these numbers are (",*lt,') is ',m)
```

```
Mean or average of these numbers are ( 1 233 453 534 6 1 3 6 2 1 78 2 4 5 6 4 3 6 ) i
s  74.88888888888889
```

In [2]:
```
#to calculate median
n=len(lt)
lt.sort()
if(n%2==0):
    md=(lt[n//2]+lt[n//2-1])/2
else:
    md=lt[n//2]
print("Medain :- ",md)
```

```
Medain :-  4.5
```

In [3]:
```
#to calculate mode
n=len(lt)
val=Counter(lt)
d=dict(val)
mo=[k for k,v in d.items() if v == max(list(val.values()))]
if(len(mo)==n):
    fm="No Mode"
else:
#     fm="Mode :- "+str(mo[0])
    fm="Mode :-"+' '.join(map(str,mo))
print(fm)
```

```
Mode :-6
```

In [17]:
```
#quartile and interquartile
dt=sorted(list(map(int,input("Input numbers with space > ").split())))
n=len(dt)
i=n//2

if(n%2==0):
    md=(dt[i-1]+dt[i])/2
    q3i=0
else:
    md=dt[i]
    q3i=0

nquartile=n//2
i=nquartile//2

if(nquartile%2==0):
    q1=(dt[i-1]+dt[i])/2
    q3=(dt[q3i+nquartile+i-1]+dt[q3i+nquartile+i])/2
else:
```

```
    q1=dt[i]
    q3=dt[q3i+nquartile+i]

print(dt)
print("Q1 =",q1)
print("Q2 =",md, "(median)")
print("Q3 =", q3)
print("Interquartile =",q3-q1)
```

```
Input numbers with space >  6 5 7 8 4 1 2 3 4 5 6
[1, 2, 3, 4, 4, 5, 5, 6, 6, 7, 8]
Q1 = 3
Q2 = 5 (median)
Q3 = 6
Interquartile = 3
```

In [18]:
```
#standard deviation

from math import sqrt
n=[2,44,66,11,8,978,23,11,6]
mean=sum(n)/len(n)
s=0
for i in n:
    s+=(i-mean)**2
std=sqrt(s/len(n)-1)
print(std)
```

```
301.2848412309447
```

In [19]:
```
#variance
#define a function, to calculate variance
def variance(x):
    m=sum(x)/len(x)
    t=0
    for i in x:
        t=t+((i-m)**2)
    return t/len(x)

#call the function with data set
x=[1,2,3,4,5,6,7,8,9]
print("variance is:",variance(x))

y=[1,2,3,-4,5,6,7,8,9]
print("variance is:",variance(y))

z=[10,-20,30,-40,50,-60,70,-80]
print("variance is:",variance(z))
```

```
variance is: 6.666666666666667
variance is: 14.765432098765432
variance is: 2525.0
```

In [20]:
```
import pandas as pd

#create a Dictionary of series
d={'Name' : pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','Chanc
   'Age' : pd.Series([25,26,25,26,30,29,23,34,40,30,51,46]),
  'Rating' : pd.Series([4.23,4,23,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#crea a DataFrame
```

```
df = pd.DataFrame(d)
df
```

Out[20]:

	Name	Age	Rating
0	Tom	25.0	4.23
1	James	26.0	4.00
2	Ricky	25.0	23.00
3	Vin	26.0	3.98
4	Steve	30.0	2.56
5	Smith	29.0	3.20
6	Jack	23.0	4.60
7	Lee	34.0	3.80
8	Chanchal	40.0	3.78
9	Gasper	30.0	2.98
10	Naviya	51.0	4.80
11	Andres	46.0	4.10
12	NaN	NaN	3.65

In [21]:
```
df.mean()
```

```
C:\Users\sathw\AppData\Local\Temp\ipykernel_6208\3698961737.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  df.mean()
```

Out[21]:
```
Age       32.083333
Rating     5.283077
dtype: float64
```

In [22]:
```
df['Rating'].mean()
```

Out[22]:
```
5.283076923076923
```

In [23]:
```
df.median()
```

```
C:\Users\sathw\AppData\Local\Temp\ipykernel_6208\530051474.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  df.median()
```

Out[23]:
```
Age       29.50
Rating     3.98
dtype: float64
```

In [24]:
```
df['Rating'].median()
```

Out[24]:
```
3.98
```

In [25]:
```
df.mode()
```

Out[25]:

	Name	Age	Rating
0	Andres	25.0	2.56
1	Chanchal	26.0	2.98
2	Gasper	30.0	3.20
3	Jack	NaN	3.65
4	James	NaN	3.78
5	Lee	NaN	3.80
6	Naviya	NaN	3.98
7	Ricky	NaN	4.00
8	Smith	NaN	4.10
9	Steve	NaN	4.23
10	Tom	NaN	4.60
11	Vin	NaN	4.80
12	NaN	NaN	23.00

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

```
C:\Users\sathw\AppData\Local\Temp\ipykernel_6208\3390915376.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  df.std()
```

Out[26]:
```
Age       9.009675
Rating    5.358999
dtype: float64
```

In [27]:
```
df.var()
```

```
C:\Users\sathw\AppData\Local\Temp\ipykernel_6208\1568254755.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  df.var()
```

Out[27]:
```
Age       81.174242
Rating    28.718873
dtype: float64
```

In [28]:
```
#Interquartile range using numpy.median
#Import the numpy library as np
import numpy as np

dt=[32,36,46,56,69,75,79,88,89,91,92,93,96,97,101,105,112,116]

#First quartile(Q1)
Q1=np.median(dt[:10])
```

```
#Third quartile(Q3)
Q3=np.median(dt[10:])

#Interquartile range(IQR)
IQR=Q3-Q1

print(IQR)
```

```
27.0
```

In [31]:
```
#interquartile range using numpy.percentile

#Import numpy library

dt=[32,36,46,47,56,69,75,79,88,89,91,92,93,96,97,101,105,112,116]

#First quartile
Q1 = np.percentile(dt,25,interpolation='midpoint')

#Third quartile(q3)
q3 = np.percentile(dt,75,interpolation='midpoint')

#Interquartile range (IQR)
IQR = Q3 - Q1

print(IQR)
```

```
36.5
C:\Users\sathw\AppData\Local\Temp\ipykernel_6208\292779599.py:8: DeprecationWarning:
the `interpolation=` argument to percentile was renamed to `method=`, which has addit
ional options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to revi
ew the method they used. (Deprecated NumPy 1.22)
  Q1 = np.percentile(dt,25,interpolation='midpoint')
C:\Users\sathw\AppData\Local\Temp\ipykernel_6208\292779599.py:11: DeprecationWarning:
the `interpolation=` argument to percentile was renamed to `method=`, which has addit
ional options.
Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to revi
ew the method they used. (Deprecated NumPy 1.22)
  q3 = np.percentile(dt,75,interpolation='midpoint')
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [5]:
```
data=pd.read_csv("C:/Users/hp/Downloads/Iris.csv")
```

In [7]:
```
print(data.head(10))
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
5   6            5.4           3.9            1.7           0.4  Iris-setosa
6   7            4.6           3.4            1.4           0.3  Iris-setosa
7   8            5.0           3.4            1.5           0.2  Iris-setosa
8   9            4.4           2.9            1.4           0.2  Iris-setosa
9  10            4.9           3.1            1.5           0.1  Iris-setosa
```

In [9]:
```
data.describe()
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

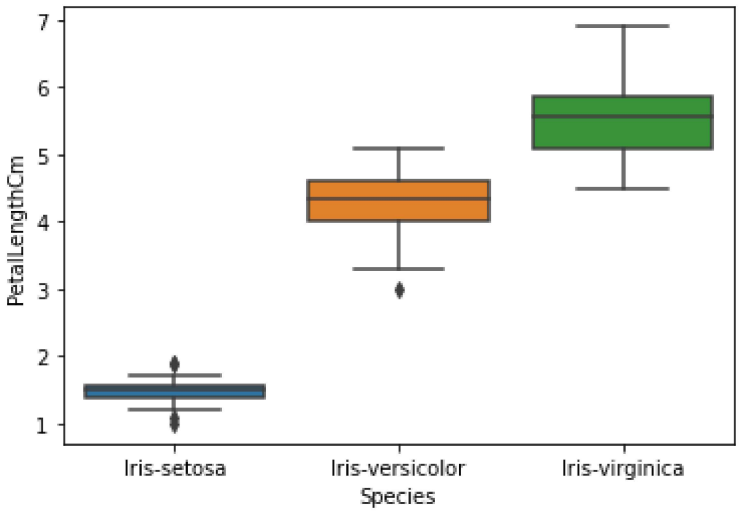
In [10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [15]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

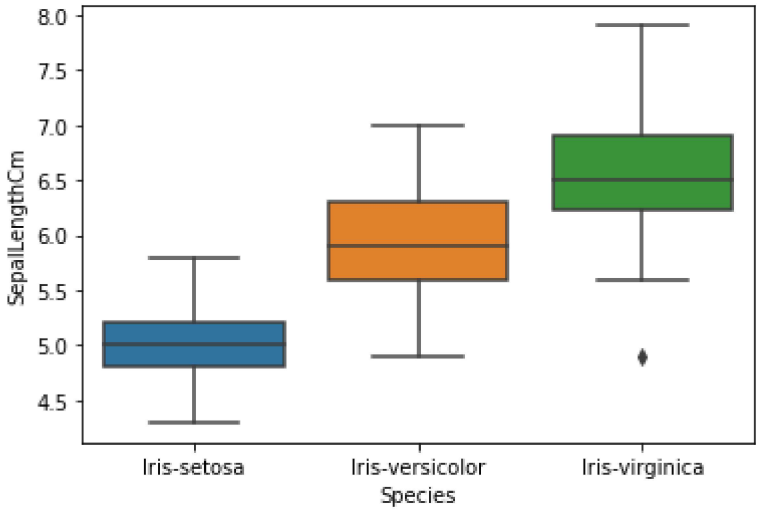
```
#box plot for petal length

sns.boxplot(x='Species',y='PetalLengthCm',data=data)
```

Out[15]: <AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>



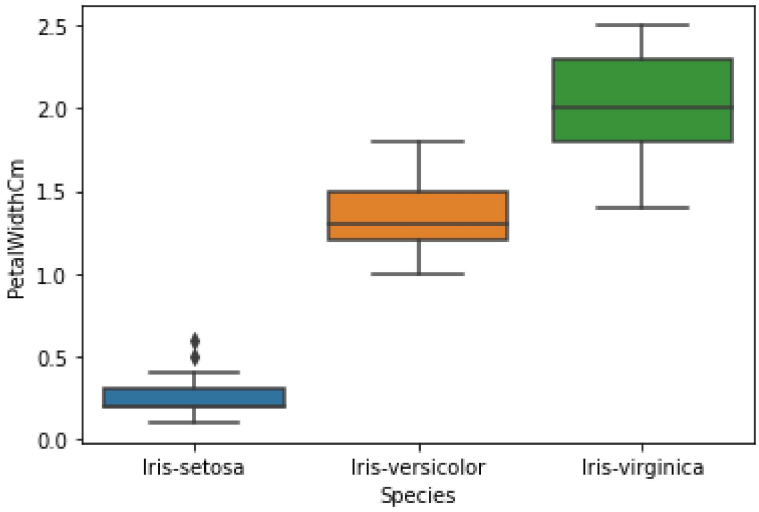
In [17]:
```
sns.boxplot(x='Species' ,y='SepalLengthCm', data=data)
```

Out[17]: <AxesSubplot:xlabel='Species', ylabel='SepalLengthCm'>

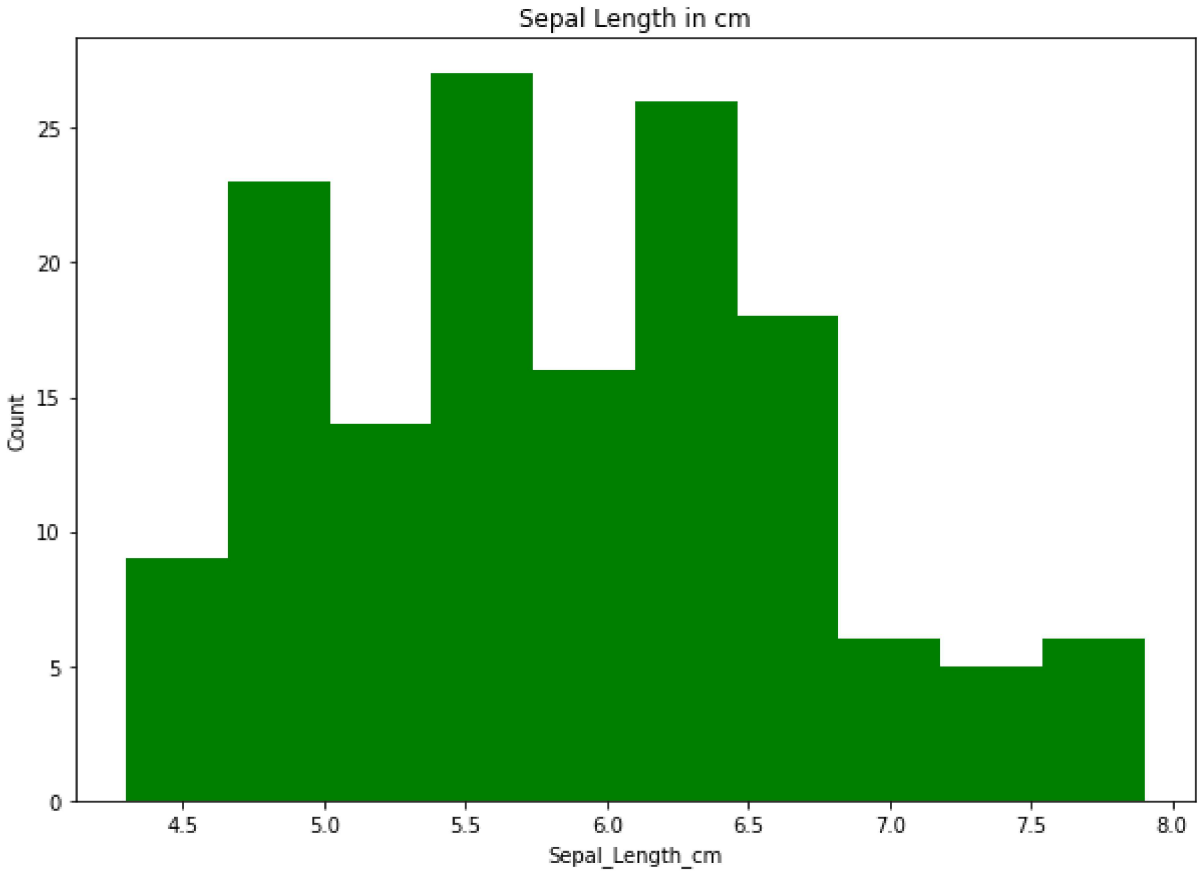


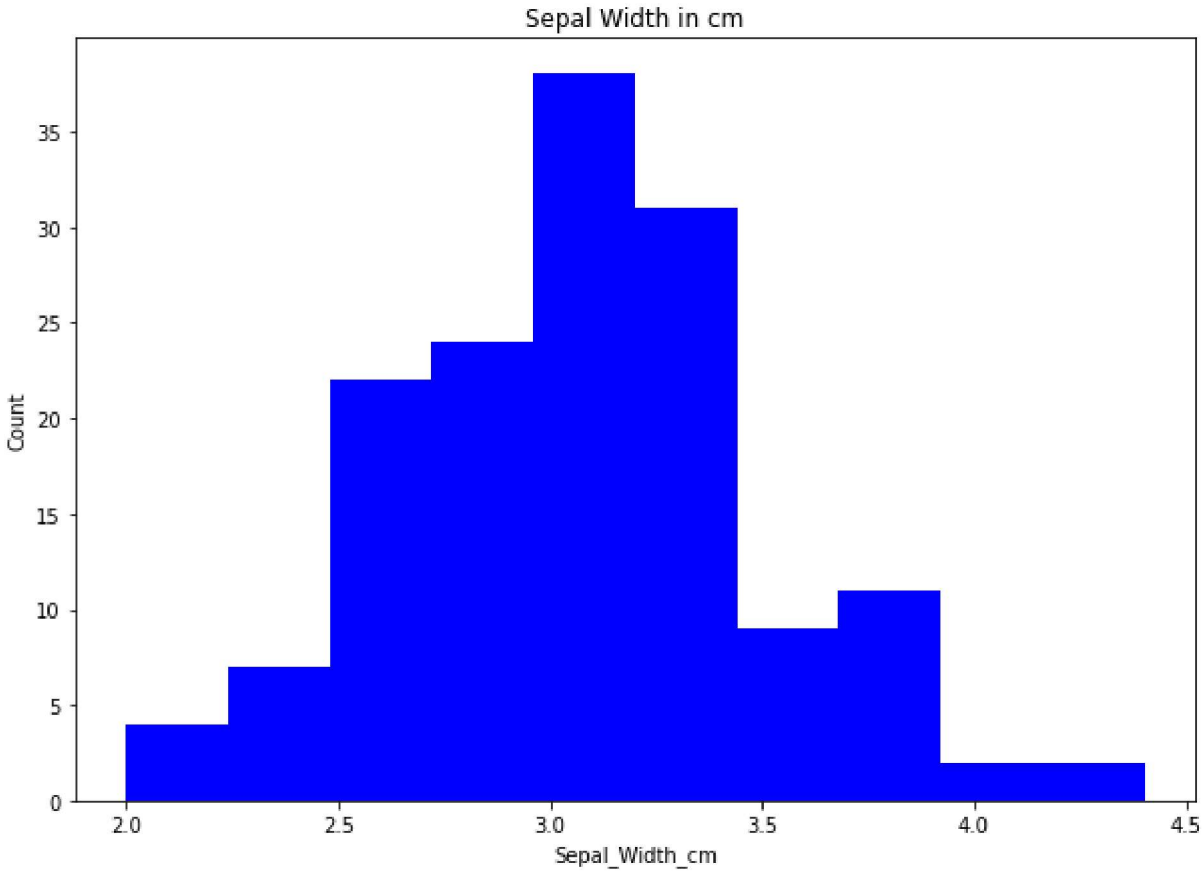
In [19]:
```
sns.boxplot(x='Species',y='PetalWidthCm',data=data)
```

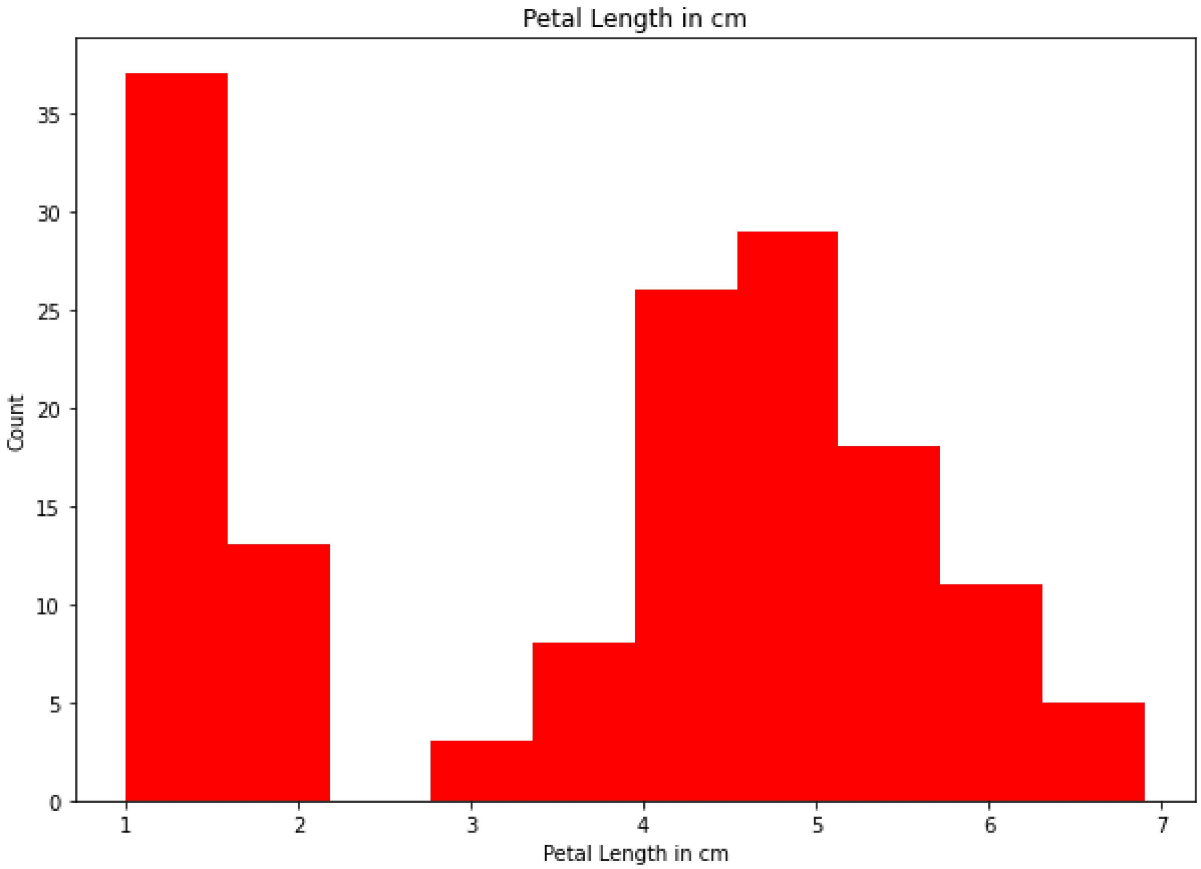
Out[19]: <AxesSubplot:xlabel='Species', ylabel='PetalWidthCm'>

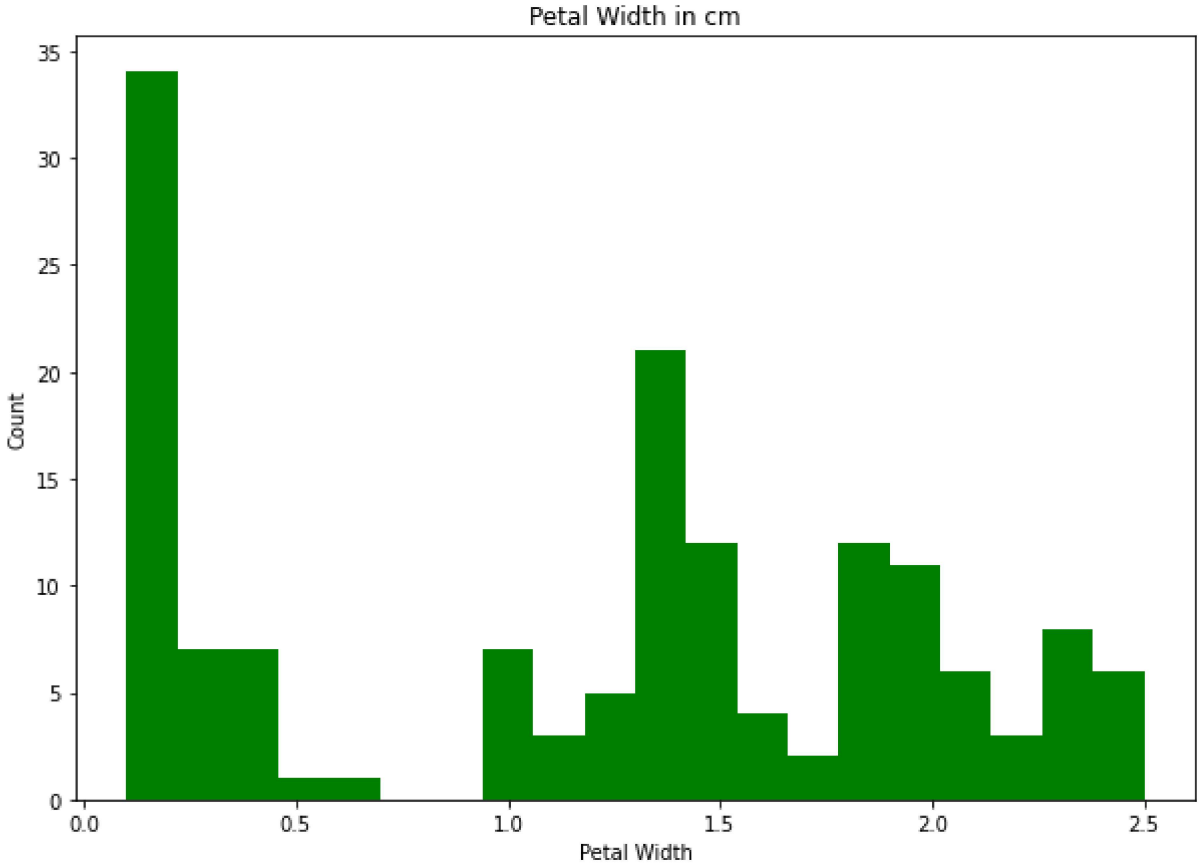


In [27]:
```
#Histogram for Sepal Length
plt.figure(figsize=(10,7))
x=data["SepalLengthCm"]

plt.hist(x,bins=10,color="green")
plt.title("Sepal Length in cm")
plt.xlabel("Sepal_Length_cm")
plt.ylabel("Count")

#Histogream for Sepal Length

plt.figure(figsize=(10,7))
x=data['SepalWidthCm']

plt.hist(x,bins=10,color="blue")
plt.title("Sepal Width in cm")
plt.xlabel("Sepal_Width_cm")
plt.ylabel("Count")

plt.show()

#Histogram for petal Length
plt.figure(figsize=(10,7))
x=data["PetalLengthCm"]
plt.hist(x,bins=10,color="red")
plt.title("Petal Length in cm")
plt.xlabel("Petal Length in cm")
plt.ylabel("Count")
plt.show()

#Histogram for petal Width
plt.figure(figsize=(10,7))
x=data['PetalWidthCm']
plt.hist(x,bins=20,color="green")
plt.title("Petal Width in cm")
plt.xlabel("Petal Width")
plt.ylabel("Count")
plt.show()
```


Sepal Length in cm
Count
Sepal_Length_cm
0
5
10
15
20
25
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Sepal Width in cm
Count
Sepal_Width_cm
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5

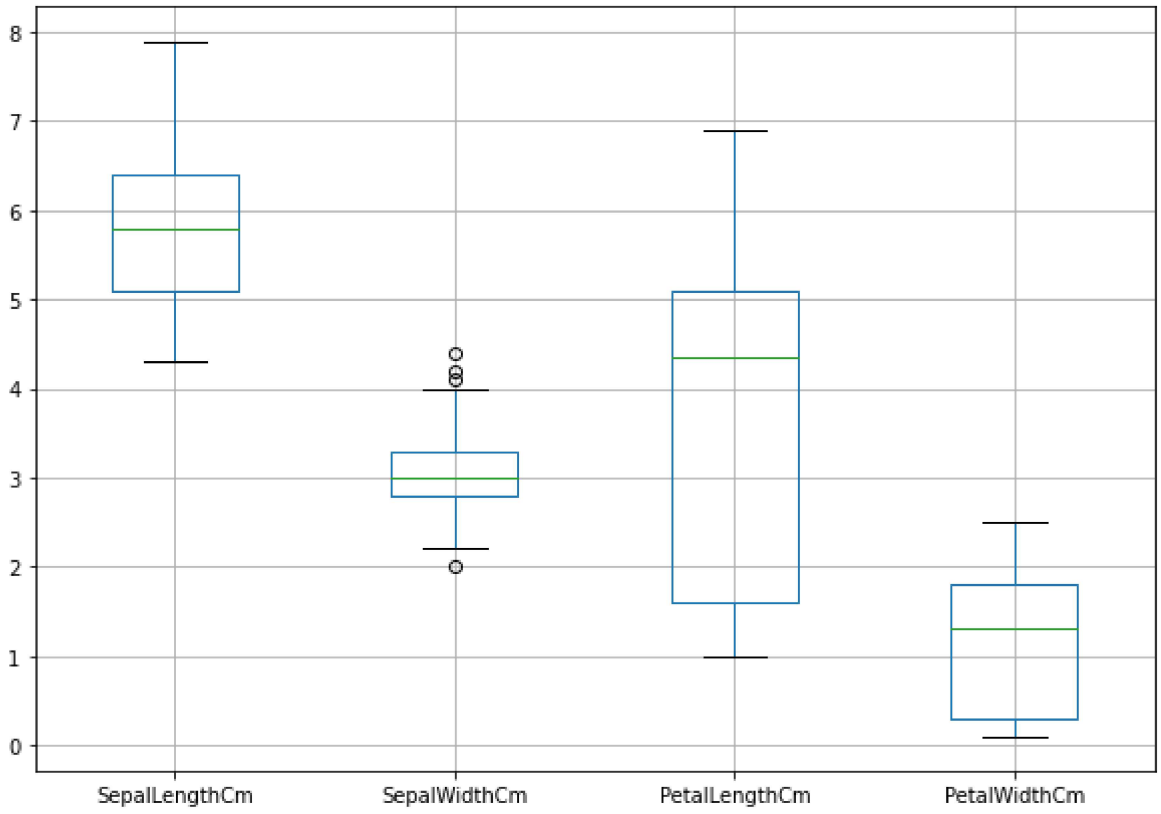




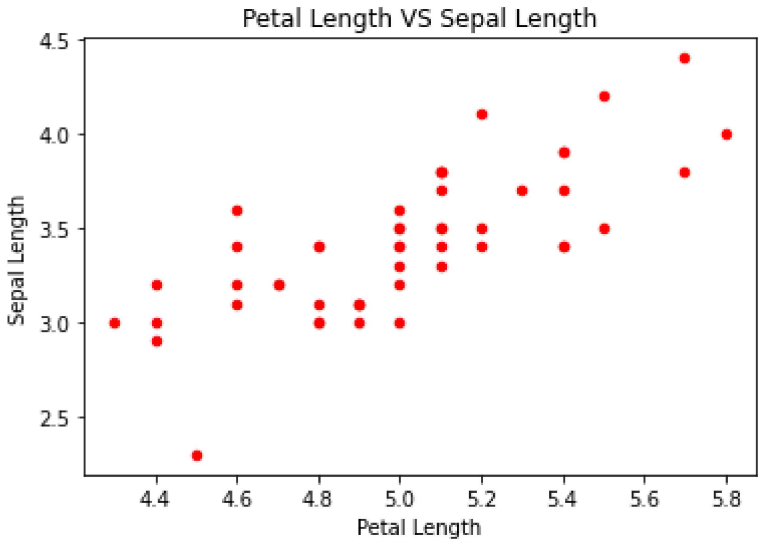
In [28]:
```
new_data = data[["SepalLengthCm","SepalWidthCm","PetalLengthCm","PetalWidthCm"]]
print(new_data.head())
plt.figure(figsize=(10,7))
new_data.boxplot()
```

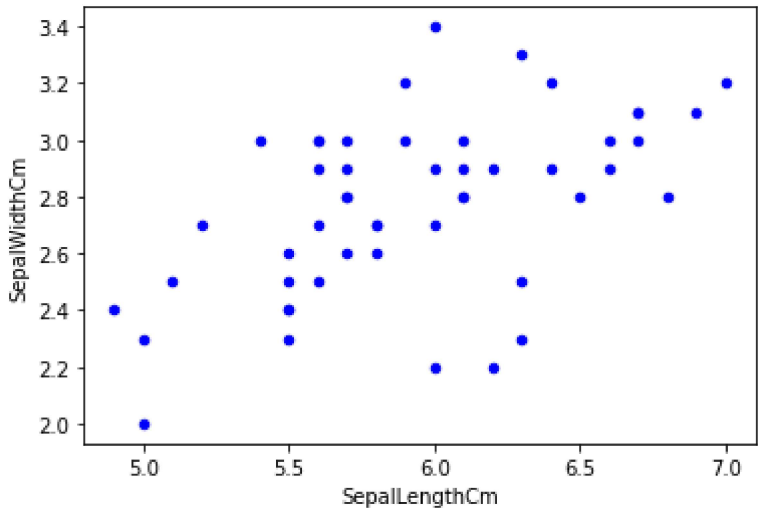
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

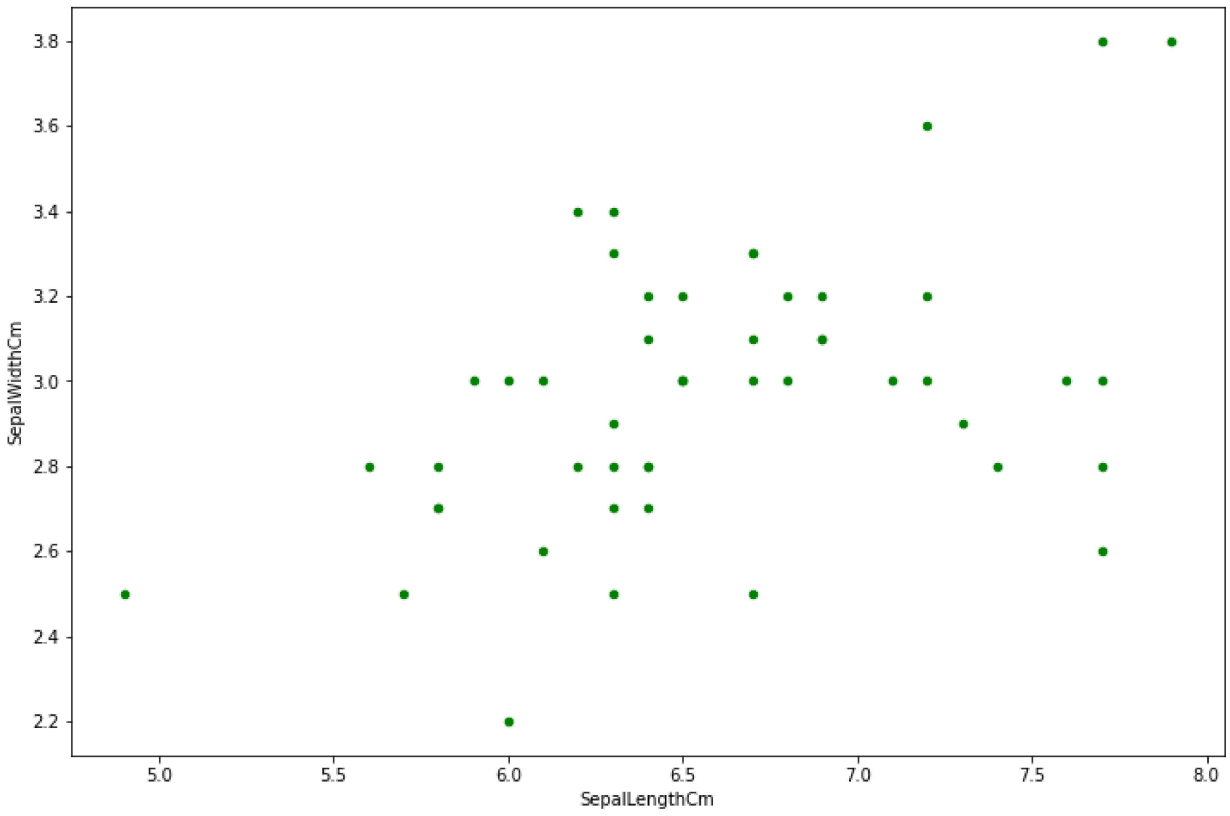
Out[28]: <AxesSubplot:>



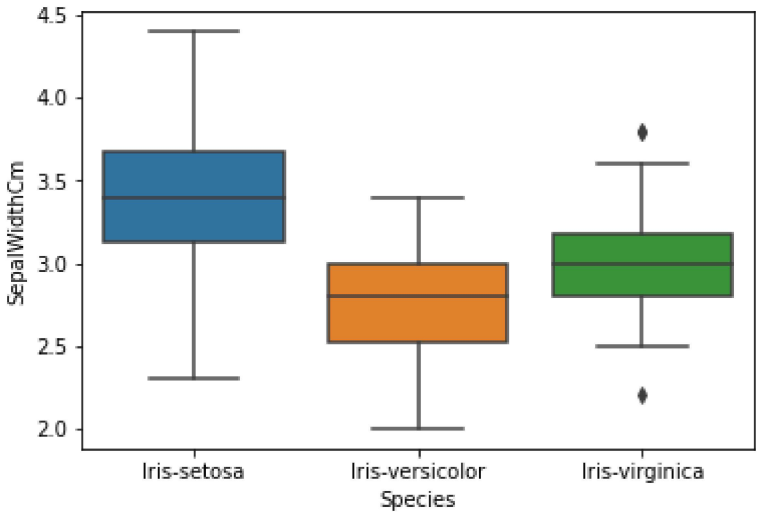
In [34]:
```
import pandas as pd
import matplotlib.pyplot as plt

fig=data[data.Species=='Iris-setosa'].plot(kind="scatter",x='SepalLengthCm',y="SepalWi
data[data.Species=='Iris-versicolor'].plot(kind="scatter",x='SepalLengthCm',y='SepalWi
data[data.Species=='Iris-virginica'].plot(kind='scatter',x='SepalLengthCm',y='SepalWic
fig.set_xlabel("Petal Length")
fig.set_ylabel("Sepal Length")
fig.set_title("Petal Length VS Sepal Length")
fig=plt.gcf()
fig.set_size_inches(12,8)
plt.show()
```


Petal Length VS Sepal Length
Sepal Length
Petal Length
4.5
4.0
3.5
3.0
2.5
4.4
4.6
4.8
5.0
5.2
5.4
5.6
5.8


SepalWidthCm
SepalLengthCm
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
5.0
5.5
6.0
6.5
7.0


SepalWidthCm
SepalLengthCm
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.2
5.0
5.5
6.0
6.5
7.0
7.5
8.0

In [31]:
```
sns.boxplot(x="Species",y="SepalWidthCm",data=data)
plt.show()
```



In [3]:
```
import pandas as pd
df = pd.read_csv('C:/Users/hp/Downloads/auto-mpg.csv')
```

In [4]:
```
df
```

Out[4]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

In [5]:
```
# replacing question marks with null values in horse power
for col in df.columns:
    df[col].replace('?',None,inplace=True)
df
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

In [6]:
```
# identifying missing values
df.isnull().sum()
```

Out[6]:
```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

In [7]:
```
#identify the outliers in displacement
max_threshold = df['cylinders'].quantile(0.95)
min_threshold = df['cylinders'].quantile(0.05)
df[(df['cylinders'] > max_threshold) | (df['cylinders'] < min_threshold)]
```

Out[7]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
71	19.0	3	70.0	97	2330	13.5	72	3	mazda rx2 coupe
111	18.0	3	70.0	90	2124	13.5	73	3	maxda rx3
243	21.5	3	80.0	110	2720	13.5	77	3	mazda rx-4
334	23.7	3	70.0	100	2420	12.5	80	3	mazda rx-7 gs

In [11]:
```
# dataframe after removing outliers
df[(df['cylinders'] > max_threshold) | (df['cylinders'] < min_threshold)]
df
```

Out[11]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

In [14]:
```
#identifying outliers in mpg
df.acceleration.mean()
df.acceleration.std()
max1 = df.acceleration.mean() + 3*df.acceleration.std()
min1 = df.acceleration.mean() - 3*df.acceleration.std()
print("The max value is",max1)
print("The min value is",min1)
```

```
The max value is 23.841157241699335
The min value is 7.295023662823281
```

In [15]:
```
# removing outliers
df[(df['acceleration'] > max1) | (df['acceleration'] < min1)]
```

Out[15]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
299	27.2	4	141.0	71	3190	24.8	79	2	peugeot 504
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup

In [16]:
```
df['zscore'] = df.acceleration - df.acceleration.mean() / df.acceleration.std()
df.head(5)
```

Out[16]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zscore
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	6.35466
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	5.85466
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	5.35466
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.35466
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	4.85466

In [17]:
```
df[(df.zscore <= 3) | (df.zscore > 3)]
```

Out[17]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zs
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	6.35
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	5.85
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	5.35
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.35
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	4.85
...	...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	9.95
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	18.95
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	5.95
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	12.95
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	13.75

398 rows × 10 columns

In [18]:
```
df[df['zscore'] > 3]
```

Out[18]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zs
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	6.35
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	5.85
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	5.35
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.35
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	4.85
...	...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	9.95
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	18.95
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	5.95
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	12.95
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	13.75

395 rows × 10 columns

In [20]:
```
# identifying outliers
Q1 = df.mpg.quantile(0.25)
Q3 = df.mpg.quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
df[(df.mpg < lower) | (df.mpg > lower)]
```

Out[20]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zs
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	6.35
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	5.85
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	5.35
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.35
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	4.85
...	...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	9.95
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	18.95
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	5.95
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	12.95
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	13.75

398 rows × 10 columns

In [21]:
```
newdf = df.dropna()
newdf
```

Out[21]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zs
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	6.35
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	5.85
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	5.35
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.35
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	4.85
...	...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	9.95
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	18.95
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	5.95
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	12.95
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	13.75

392 rows × 10 columns

In [22]:
```
std = df[(df['cylinders'] <= max_threshold) | (df['cylinders'] >= min_threshold)]
print('standard value',std)
```

```
standard value       mpg  cylinders  displacement horsepower  weight  acceleration  \
0    18.0          8         307.0         130    3504          12.0
1    15.0          8         350.0         165    3693          11.5
2    18.0          8         318.0         150    3436          11.0
3    16.0          8         304.0         150    3433          12.0
4    17.0          8         302.0         140    3449          10.5
..    ...        ...           ...         ...     ...           ...
393  27.0          4         140.0          86    2790          15.6
394  44.0          4          97.0          52    2130          24.6
395  32.0          4         135.0          84    2295          11.6
396  28.0          4         120.0          79    2625          18.6
397  31.0          4         119.0          82    2720          19.4

     model year  origin                  car name    zscore
0            70       1  chevrolet chevelle malibu   6.35466
1            70       1          buick skylark 320   5.85466
2            70       1         plymouth satellite   5.35466
3            70       1              amc rebel sst   6.35466
4            70       1                ford torino   4.85466
..          ...     ...                        ...       ...
393          82       1            ford mustang gl   9.95466
394          82       2                  vw pickup  18.95466
395          82       1              dodge rampage   5.95466
396          82       1                ford ranger  12.95466
397          82       1                 chevy s-10  13.75466

[398 rows x 10 columns]
```

In [24]:
```
df[(df['cylinders'] > max_threshold) | (df['cylinders'] < min_threshold)]
df[(df['cylinders'] > max_threshold) | (df['cylinders'] < min_threshold)]
```

Out[24]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zscore
71	19.0	3	70.0	97	2330	13.5	72	3	mazda rx2 coupe	7.85466
111	18.0	3	70.0	90	2124	13.5	73	3	maxda rx3	7.85466
243	21.5	3	80.0	110	2720	13.5	77	3	mazda rx-4	7.85466
334	23.7	3	70.0	100	2420	12.5	80	3	mazda rx-7 gs	6.85466

In [25]:
```
df['acceleration'].clip(min1,max1,inplace = True)
df[(df.acceleration > max1) | (df.acceleration < min1)]
```

Out[25]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	zscore

In [ ]:
```

```

Model Training

1. Holdout
2. K-Fold cross validation
3. Bootstrap Sampling

In [5]:
```
import pandas as pd
import numpy as np
ColumnNames  = ['Hours','Calories','Weight']
DataValues = [[1.0,2500,95],
              [2.0,2000,85],
              [2.5,1900,83],
              [3.0,1850,81],
              [3.5,1600,80],
              [4.0,1500,78],
              [5.0,1500,77],
              [5.5,1600,80],
              [6.0,1700,75],
              [6.5,1500,70]]

#Create the Data Frame
GymData=pd.DataFrame(data=DataValues,columns=ColumnNames)
GymData.head()

#Seperate Target Variable and Predictor Variables
TargetVariable  ='Weight'
Predictors = ['Hours','Calories']
X = GymData[Predictors].values
y = GymData[TargetVariable].values

#Bootstrapping

#Creating empty list to hold accuracy values
AccuracyValues=[]
n_times=5

##Performing Bootstraping
for i in range(n_times):
    #Split the data into training and testing
    from sklearn.model_selection import train_test_split
    #Chaning the seed value for each iteration
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_sta


    #Single Decision Tree Regression in Python
    from sklearn import tree
    #choose from different tunable hyper parameters
    RegModel = tree.DecisionTreeRegressor(max_depth=3,criterion='squared_error')

    #Creating the model on Training Data
    DTree  = RegModel.fit(X_train,y_train)
    prediction=DTree.predict(X_test)

    #Measuring accuracy on Testing Data
    Accuracy  = 100-(np.mean(np.abs((y_test - prediction)/y_test))*100)
```

```
    #Storing accuracy on Testing Data
    AccuracyValues.append(np.round(Accuracy))

#Rsult of all bootatraping trials
print(AccuracyValues)

#Final accuracy
print('Final average accuracy',np.mean(AccuracyValues))
```

```
[96.0, 96.0, 95.0, 96.0, 95.0]
Final average accuracy 95.6
```

In [6]:

```
#K-fold cross validation
#Defining a custom function to calculate accuracy
#Make sure there are no zeros in the target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE=np.mean(100*(np.abs(orig-pred)/orig))
    #print ('#'*70,'Accuracy:',100-MAPE)
    return (100-MAPE)

#Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring = make_scorer(Accuracy_Score,greater_is_better=True)

#Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

#Single Decision Tree Regression in python
from sklearn import tree
#choose from different tunable hyper parameters
RegModel = tree.DecisionTreeRegressor(max_depth=3,criterion='squared_error')

#Running 10-Fold cross validation on a given algorithm
#Passing full data x and y because the K-fold will split the data and automatically
Accuracy_Values=cross_val_score(RegModel,X,y,cv=10,scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation : \n',Accuracy_Values)
print('\nFinal Average Accuracy of the model : ',round(Accuracy_Values.mean(),2))
```

```
Accuracy values for 10-fold Cross Validation :
 [89.47368421 95.29411765 96.78714859 96.2962963  98.75       98.07692308
 96.42857143 95.83333333 94.4        92.85714286]

Final Average Accuracy of the model :  95.42
```

In [7]:

```
#Hold Out

#Split the data into training an testing set
from sklearn.model_selection import train_test_split
    #Chaning the seed value for each iteration
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5,random_state=0)

#Single Decision Tree Regression in  python
from sklearn import tree
#choose from different tunable hyper parameters
RegModel = tree.DecisionTreeRegressor(max_depth=3,criterion='squared_error')

#Creating the model on Training Data
DTree=RegModel.fit(X_train,y_train)
prediction=DTree.predict(X_test)
```

```
#Measuring accuracy on Testing on testing Data
Accuracy=100-(np.mean(np.abs((y_test-prediction)/y_test))*100)
print(Accuracy)
```

```
94.99356248523506
```

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#Regression model
from sklearn.linear_model import LogisticRegression
#Supervised Classsification
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
#fitting
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
data=pd.read_csv("spine_dataset.csv")
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 310 entries, 0 to 309
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Col1         310 non-null    float64
 1   Col2         310 non-null    float64
 2   Col3         310 non-null    float64
 3   Col4         310 non-null    float64
 4   Col5         310 non-null    float64
 5   Col6         310 non-null    float64
 6   Col7         310 non-null    float64
 7   Col8         310 non-null    float64
 8   Col9         310 non-null    float64
 9   Col10        310 non-null    float64
 10  Col11        310 non-null    float64
 11  Col12        310 non-null    float64
 12  Class_att    310 non-null    object 
 13  Unnamed: 13  14 non-null     object 
dtypes: float64(12), object(2)
memory usage: 34.0+ KB
```

In [4]:
```
#to check what class_att column contains
data.Class_att.unique()
```

Out[4]:
```
array(['Abnormal', 'Normal'], dtype=object)
```

In [5]:
```
data=data.drop('Unnamed: 13',axis=1)
```

In [6]:
```
data.head()
```

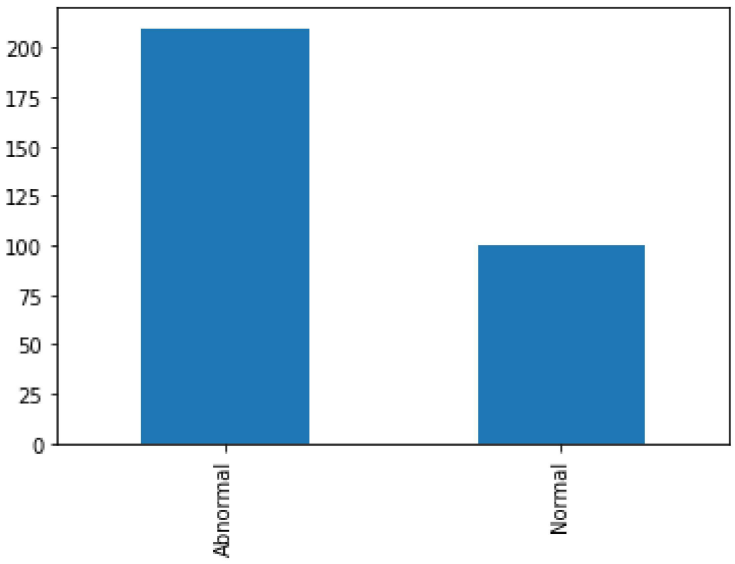
Out[6]:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	C
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	0.744503	12.5661	14.5386	15.3
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	0.415186	12.8874	17.5323	16.7
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	0.474889	26.8343	17.4861	16.6
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	0.369345	23.5603	12.7074	11.4
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	0.543360	35.4940	15.9546	8.8

In [7]:
```
data.rename(columns={
    "Col1":"pelvic_incidence",
    "Col2":"pelvic tilt",
    "Col3":"lumbar_lordosis_angle",
    "Col4":"sacral_slope",
    "Col5":"pelvic_radius",
    "Col6":"degree_spondylolisthesis",
    "Col7":"pelvic_slope",
    "Col8":"Direct_tilt",
    "Col9":"thoracic_slope",
    "Col10":"cervical_tilt",
    "Col11":"sacrum_angle",
    "Col12":"scoliosis_slope",
    "Class_att":"variable"
},inplace=True)
```

In [8]:
```
data["variable"].value_counts().sort_index().plot.bar()
```

Out[8]:
```
<AxesSubplot:>
```



In [9]:
```
#Changing categorical value to
# data.variable=data.variable.astype("category").cat.codes
from sklearn.preprocessing import LabelEncoder
label=LabelEncoder().fit_transform(data['variable'])
data.drop("variable",axis=1,inplace=True)
data["Condition"]=label
```

In [10]:
```
data.head()
```

Out[10]:

	pelvic_incidence	pelvic tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisth
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918

In [11]:
```
data=data.drop(['pelvic_slope','Direct_tilt','thoracic_slope','cervical_tilt','sacrum_
```

In [12]:
```
y=data.Condition
X=data.drop("Condition",axis=1)
```

In [13]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
```

In [14]:
```
#Training models
from sklearn.metrics import accuracy_score
acc_list=[]
model_list=[]
for model in [LogisticRegression,DecisionTreeClassifier,RandomForestClassifier,Gradier
    classifier=model().fit(X_train,y_train)
    %time predictions=classifier.predict(X_test)
    model_list.append((model).__name__)
    acc_list.append(accuracy_score(predictions,y_test))
```

```
CPU times: total: 0 ns
Wall time: 1.04 ms
CPU times: total: 0 ns
Wall time: 997 µs
CPU times: total: 15.6 ms
Wall time: 8.94 ms
CPU times: total: 0 ns
Wall time: 997 µs
CPU times: total: 0 ns
Wall time: 1.99 ms
```

In [15]:
```
results=pd.DataFrame({ 'ML Model': model_list,
    'Test Accuracy': acc_list})
results
```

Out[15]:

	ML Model	Test Accuracy
0	LogisticRegression	0.881720
1	DecisionTreeClassifier	0.827957
2	RandomForestClassifier	0.827957
3	GradientBoostingClassifier	0.817204
4	SVC	0.827957

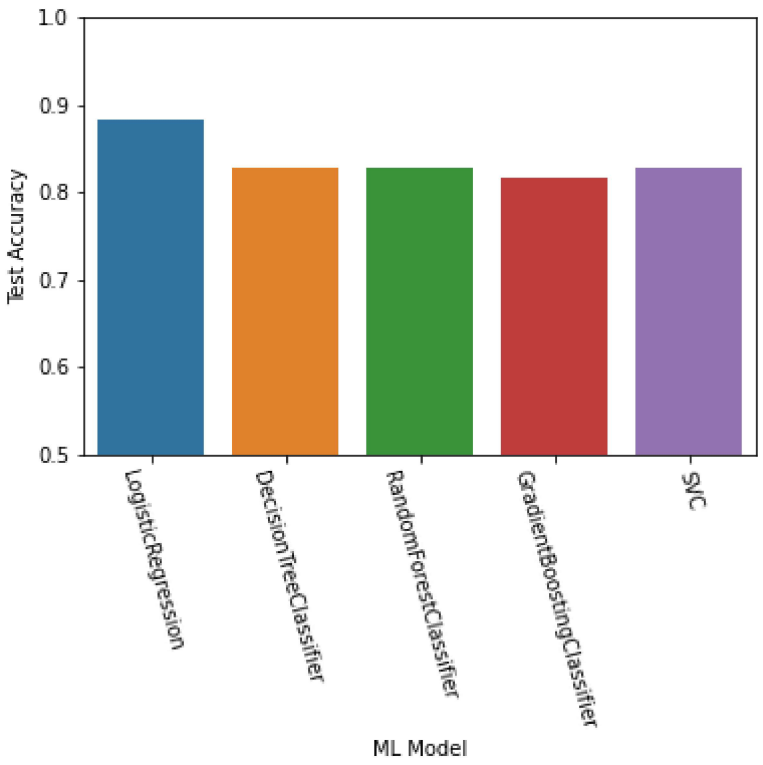
In [16]:
```
results.sort_values(by=['Test Accuracy'], ascending=False)
```

Out[16]:

	ML Model	Test Accuracy
0	LogisticRegression	0.881720
1	DecisionTreeClassifier	0.827957
2	RandomForestClassifier	0.827957
4	SVC	0.827957
3	GradientBoostingClassifier	0.817204

In [17]:
```
#displaying results in bargraph
sns.barplot(x=results["ML Model"],y=results["Test Accuracy"],data=results)
plt.xticks(rotation=-75)
plt.ylim(0.5,1)
```

Out[17]:
```
(0.5, 1.0)
```



In [18]:
```
#Unsupervised classification
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
km=KMeans(n_clusters=2,random_state=42)
km.fit_predict(X_train)
score=silhouette_score(X_train,km.labels_,metric='euclidean')
print('Silhouette_score:%.3f'%score)
```

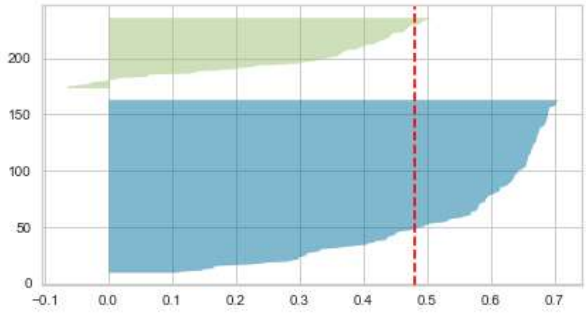
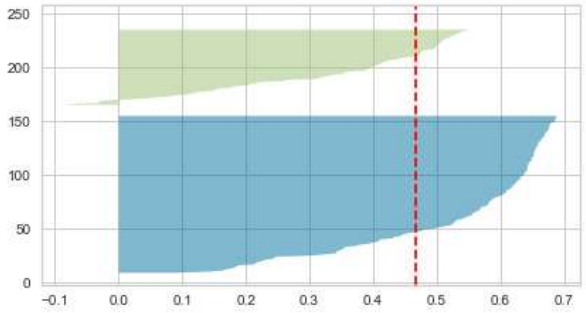
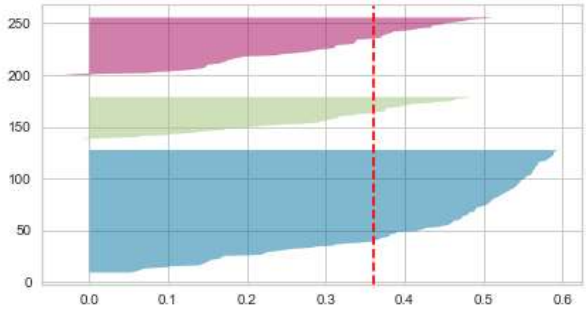
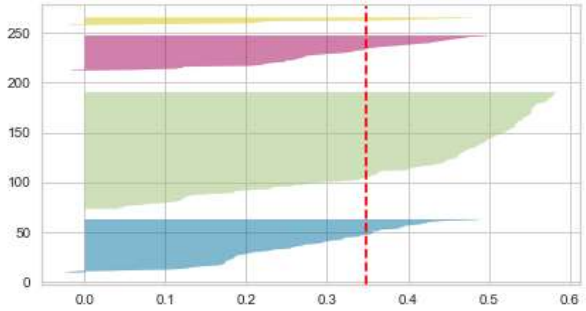
```
Silhouette_score:0.479
```

In [19]:
```
pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in c:\users\hp\anaconda3\lib\site-packages
(1.5)Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\users\hp\anaconda3\lib
\site-packages (from yellowbrick) (3.5.1)
Requirement already satisfied: scipy>=1.0.0 in c:\users\hp\anaconda3\lib\site-package
s (from yellowbrick) (1.7.3)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\hp\anaconda3\lib\site-
packages (from yellowbrick) (1.0.2)
Requirement already satisfied: numpy>=1.16.0 in c:\users\hp\anaconda3\lib\site-packag
es (from yellowbrick) (1.21.5)
Requirement already satisfied: cycler>=0.10.0 in c:\users\hp\anaconda3\lib\site-packa
ges (from yellowbrick) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\hp\anaconda3\lib\site-pac
kages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\anaconda3\lib\site
-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\hp\anaconda3\lib\site-pa
ckages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.3.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\hp\anaconda3\lib\site-packag
es (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.0.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\anaconda3\lib\site-pa
ckages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\anaconda3\lib\site-pack
ages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: six>=1.5 in c:\users\hp\anaconda3\lib\site-packages (f
rom python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\hp\anaconda3\lib\site-package
s (from scikit-learn>=1.0.0->yellowbrick) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\anaconda3\lib\site
-packages (from scikit-learn>=1.0.0->yellowbrick) (2.2.0)
```

In [20]:

```
from yellowbrick.cluster import SilhouetteVisualizer
fig,ax=plt.subplots(2,2,figsize=(15,8))
for i in [2,3,4,5]:
    #Kmeans instances for different clusters
    km=KMeans(n_clusters=i,random_state=42)
    q,mod=divmod(i,2)
    #Create SilhoutteVisualizer with KMeans instance
    visualizer=SilhouetteVisualizer(km,colors='yellowbrick',ax=ax[q-1][mod])
    visualizer.fit(X_train)
```

In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv("Iris (1).csv")
```

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

Out[3]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [5]:
```
dummies = pd.get_dummies(df.Species)
dummies
```

Out[5]:

	Iris-setosa	Iris-versicolor	Iris-virginica
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
...	...	...	...
145	0	0	1
146	0	0	1
147	0	0	1
148	0	0	1
149	0	0	1

150 rows × 3 columns

In [7]:
```
Iris = pd.concat([df, dummies], axis = 'columns')
Iris.head()
```

Out[7]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species	Iris-setosa	Iris-versicolor	Iris-virginica
0	1	5.1	3.5	1.4	0.2	Iris-setosa	1	0	0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	1	0	0
2	3	4.7	3.2	1.3	0.2	Iris-setosa	1	0	0
3	4	4.6	3.1	1.5	0.2	Iris-setosa	1	0	0
4	5	5.0	3.6	1.4	0.2	Iris-setosa	1	0	0

In [8]:
```
Iris.drop(['Species'], axis = 'columns')
Iris
```

Out[8]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species	Iris-setosa	Iris-versicolor	Iris-virginica
0	1	5.1	3.5	1.4	0.2	Iris-setosa	1	0	0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	1	0	0
2	3	4.7	3.2	1.3	0.2	Iris-setosa	1	0	0
3	4	4.6	3.1	1.5	0.2	Iris-setosa	1	0	0
4	5	5.0	3.6	1.4	0.2	Iris-setosa	1	0	0
...	...	...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica	0	0	1
146	147	6.3	2.5	5.0	1.9	Iris-virginica	0	0	1
147	148	6.5	3.0	5.2	2.0	Iris-virginica	0	0	1
148	149	6.2	3.4	5.4	2.3	Iris-virginica	0	0	1
149	150	5.9	3.0	5.1	1.8	Iris-virginica	0	0	1

150 rows × 9 columns

In [9]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
iris_1 = df
iris_1.Species = le.fit_transform(iris_1.Species)
iris_1
```

Out[9]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

150 rows × 6 columns

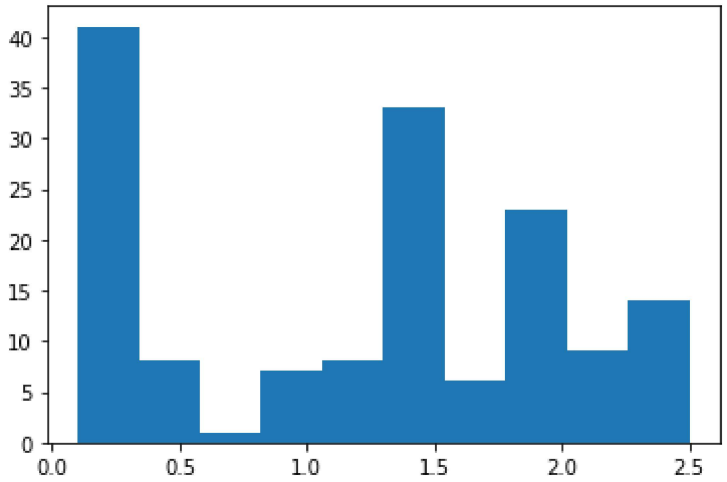
In [13]:
```
df.describe()
```

Out[13]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667	1.000000
std	43.445368	0.828066	0.433594	1.764420	0.763161	0.819232
min	1.000000	4.300000	2.000000	1.000000	0.100000	0.000000
25%	38.250000	5.100000	2.800000	1.600000	0.300000	0.000000
50%	75.500000	5.800000	3.000000	4.350000	1.300000	1.000000
75%	112.750000	6.400000	3.300000	5.100000	1.800000	2.000000
max	150.000000	7.900000	4.400000	6.900000	2.500000	2.000000

In [15]:
```
import matplotlib.pyplot as plt
x = df['Petal Width ']
plt.hist(x)
plt.show
```

Out[15]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [18]:
```
df['Petal category'] = pd.cut(df['Petal Width '], bins = [0, 1, 2, 2.5], labels = ['sm
print(df['Petal category'])
print(df['Petal category'].value_counts())
```

```
0       small
1       small
2       small
3       small
4       small
        ...
145      long
146    medium
147    medium
148      long
149    medium
Name: Petal category, Length: 150, dtype: category
Categories (3, object): ['small' < 'medium' < 'long']
medium    70
small     57
long      23
Name: Petal category, dtype: int64
```

In [20]:
```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
df['Petal category'] = encoder.fit_transform(df[['Petal category']])
df
```

Out[20]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species	Petal category
0	1	5.1	3.5	1.4	0.2	0	2.0
1	2	4.9	3.0	1.4	0.2	0	2.0
2	3	4.7	3.2	1.3	0.2	0	2.0
3	4	4.6	3.1	1.5	0.2	0	2.0
4	5	5.0	3.6	1.4	0.2	0	2.0
...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	2	0.0
146	147	6.3	2.5	5.0	1.9	2	1.0
147	148	6.5	3.0	5.2	2.0	2	1.0
148	149	6.2	3.4	5.4	2.3	2	0.0
149	150	5.9	3.0	5.1	1.8	2	1.0

150 rows × 7 columns

Feature Extraction

PCA SVD LDA Feature Subset Selection

In [22]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
df = datasets.load_iris()
df
```

Out[22]:
```
{'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. ],
                [6.6, 2.9, 4.6, 1.3],
                [5.2, 2.7, 3.9, 1.4],
```

```
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
```

```
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data S
et Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)
\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :Att
ribute Information:\n        - sepal length in cm\n        - sepal width in cm\n
- petal length in cm\n        - petal width in cm\n        - class:\n
- Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n
\n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ============
========\n                    Min  Max   Mean    SD   Class Correlation\n    ========
====== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9
5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    peta
l length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5
1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ==========
==========\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% for
each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHAL
L%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used
by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the s
ame as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong
data points.\n\nThis is perhaps the best known database to be found in the\npattern r
ecognition literature.  Fisher\'s paper is a classic in the field and\nis referenced
frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 cl
asses of 50 instances each, where each class refers to a\ntype of iris plant.  One cl
```

```
ass is linearly separable from the other 2; the\nlatter are NOT linearly separable fr
om each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple mea
surements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); a
lso in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   -
Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q32
7.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V.
(1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classificati
on Rule for Recognition in Partially Exposed\n     Environments".  IEEE Transactions
on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   -
Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on I
nformation Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  C
heeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in t
he data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

PCA

In [23]:
```
x = df.data
y = df.target
print(x.shape, y.shape)
```

```
(150, 4) (150,)
```

In [24]:
```
pca = PCA(n_components = 2)
pca.fit(x)
```

Out[24]:
```
PCA(n_components=2)
```

In [25]:
```
pca.components_
```

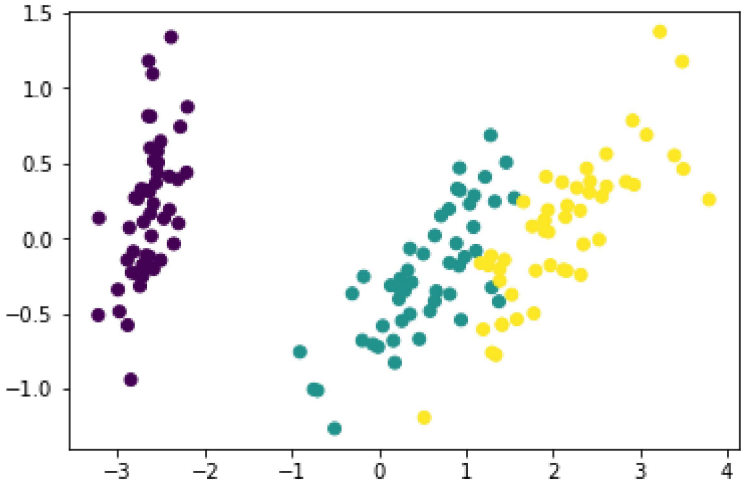
Out[25]:
```
array([[ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ],
       [ 0.65658877,  0.73016143, -0.17337266, -0.07548102]])
```

In [28]:
```
z = pca.transform(x)
z.shape
```

Out[28]:
```
(150, 2)
```

In [29]:
```
plt.scatter(z[:, 0], z[:, 1], c = y)
```

Out[29]:
```
<matplotlib.collections.PathCollection at 0x228d8c7d670>
```



In [30]:
```
pca.explained_variance_ratio_
```

Out[30]:
```
array([0.92461872, 0.05306648])
```

LDA

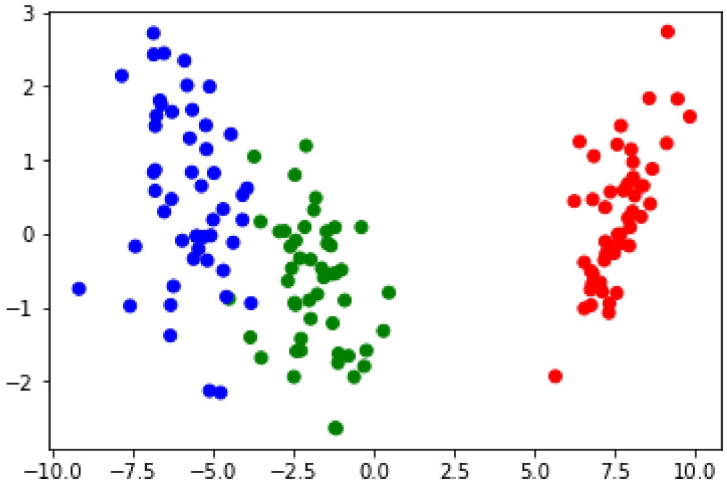
In [31]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components = 2)
x_r2 = lda.fit(x, y).transform(x)
```

In [32]:
```
lda.explained_variance_ratio_
```

Out[32]:
```
array([0.9912126, 0.0087874])
```

In [33]:
```
clrs = ['red', 'green', 'blue', 'yellow']
vectorizer = np.vectorize(lambda x: clrs[x % len(clrs)])
plt.scatter(x_r2[:, 0], x_r2[:, 1], c = vectorizer(y))
```

Out[33]:
```
<matplotlib.collections.PathCollection at 0x228d8d09520>
```



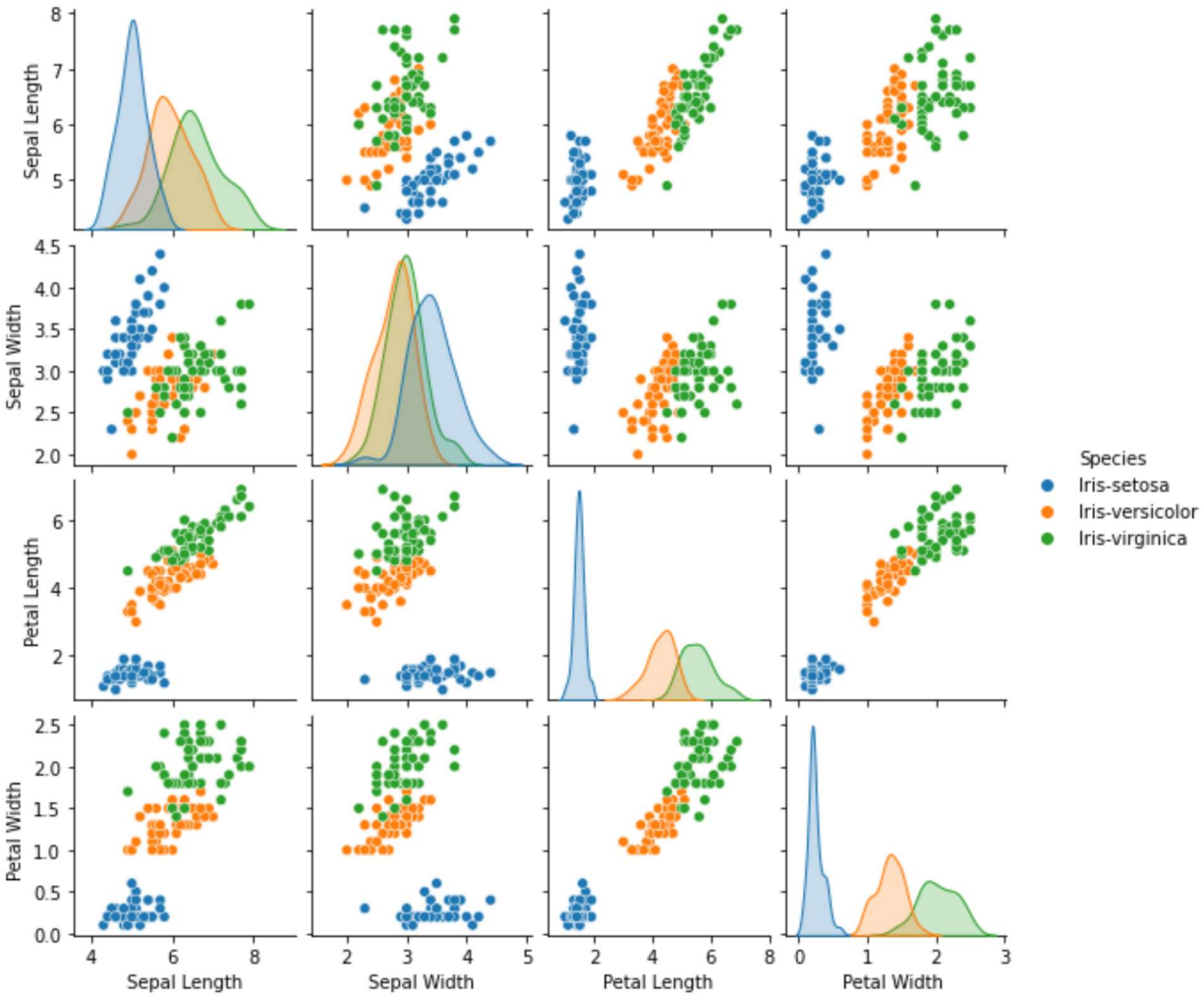
Feature subset selection

In [37]:
```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
iris = pd.read_csv('Iris (1).csv')
```

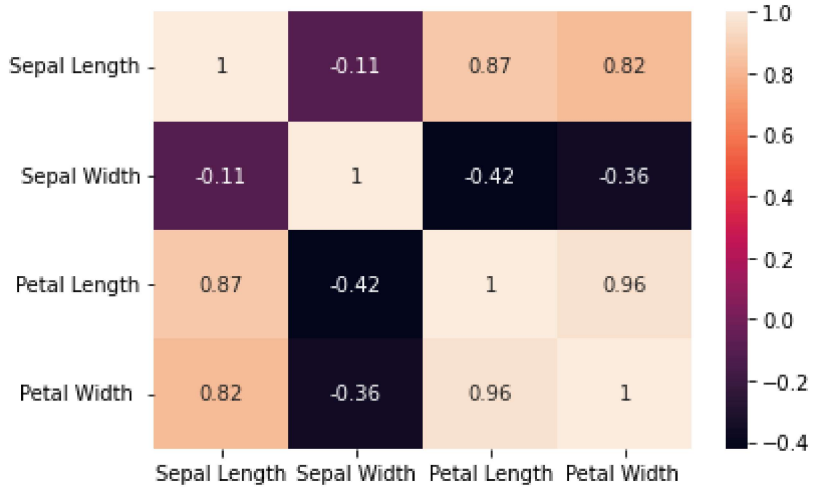
In [38]:
```
sns.pairplot(iris.drop(['Id'], axis = 1), hue = 'Species', height = 2)
```

Out[38]:
```
<seaborn.axisgrid.PairGrid at 0x228d8d78700>
```



In [40]:
```
sns.heatmap(iris.corr(method = 'pearson').drop(['Id'], axis = 1).drop(['Id'], axis = 0
```

Out[40]:
```
<AxesSubplot:>
```



In [ ]: