IRIS Dataset

Machine Learning

Supervised Learning

Classification

- Logistic Regression / classification
- Decision Tree Classifier
- Random Forest Classifier
- K-N Neighbor Classifier
- Support Vector Machine - Linear, Ploly, RBF(Redial Basis Function)
- Naive Bayes - MultinomialNB, BurnoulliNB, GoussianNB

Regression

- Simple Linear Regression
- Multi- Linear Regression
- Decision Tree Regresor
- Random Forest Regressor
- K -N Neighbour Regressor
- Support Vector Machine Learning - Linear, polly, RBF - Redial Basis Function

Unsupervised Learning

- K-Mean Clustering
- Hierarchical Clustering

https://www.javatpoint.com/regression-vs-classification-in-machine-learning#:~:text=The%20main%20difference%20between%20Regression%20and%20Classifica
(https://www.javatpoint.com/regression-vs-classification-in-machine-learning#:~:text=The%20main%20difference%20between%20Regression%20and%20Classifica

Import libraries

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

Read DataSet

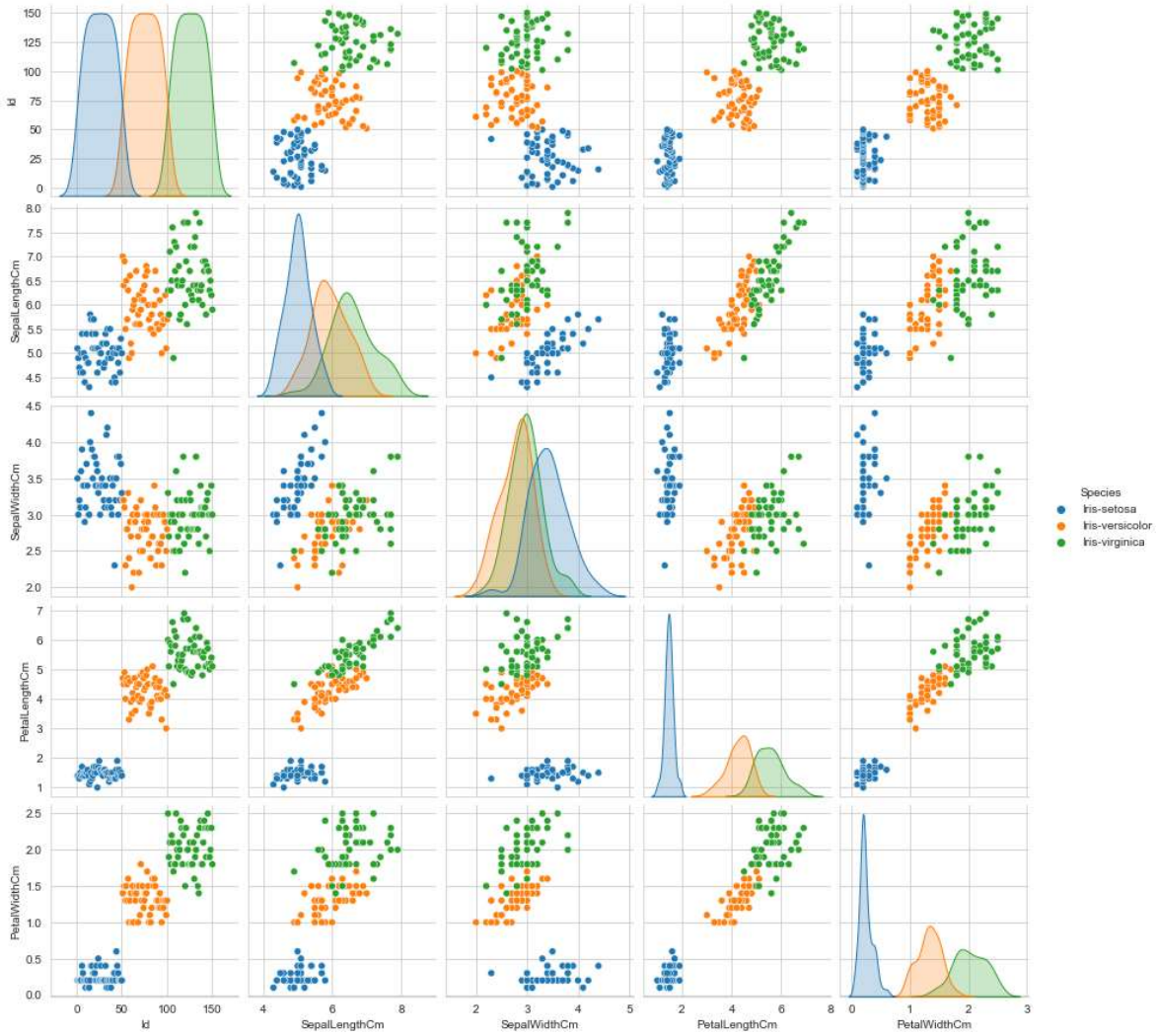
In [2]:
```
df=pd.read_csv("C:\\Users\\omkan\\Desktop\\Iris1.csv")
df1=df
```

In [3]:
```
df.head(10)
```

Out[3]:

	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 [4]:
```
sns.set_style("whitegrid")
sns.pairplot(df,hue="Species");
plt.show()
```



Analyses

In [5]:
```
df.corr()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

In [6]:
```
df.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 [7]:
```
df.describe()
```

Out[7]:

	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 [8]:
```
df.isnull().sum()
```

Out[8]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [9]:
```
df.head(2)
```

Out[9]:

	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

In [10]:
```
df.head(3)
```

Out[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

CLASSIFICATION

Logistic Regression

- In the case of Classification, we need to change Label/target/output into descrete form LabelEncoder

In [11]:
```
df['Species'].value_counts()
```

Out[11]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

In [12]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Species']=le.fit_transform(df['Species'])
df['Species'].value_counts()
```

Out[12]:
```
0    50
1    50
2    50
Name: Species, dtype: int64
```

In [13]:
```
df.head(2)
```

Out[13]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0

In [14]:
```
df01= df[df['Species']!=2]
```

In [15]:
```
df01['Species'].value_counts()
```

Out[15]:
```
0    50
1    50
Name: Species, dtype: int64
```

In [16]:
```
df01.head()
```

Out[16]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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

In [17]:
```
df01.tail()
```

Out[17]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
95	96	5.7	3.0	4.2	1.2	1
96	97	5.7	2.9	4.2	1.3	1
97	98	6.2	2.9	4.3	1.3	1
98	99	5.1	2.5	3.0	1.1	1
99	100	5.7	2.8	4.1	1.3	1

In [18]:
```
x=df01.iloc[:,1:5]      # independent variable
y=df01['Species']       # Dependent variable
x.shape,y.shape
```

Out[18]:
```
((100, 4), (100,))
```

In [19]:
```
x.head(3)
```

Out[19]:

	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

In [20]:
```
y.head(3)
```

Out[20]:
```
0    0
1    0
2    0
Name: Species, dtype: int32
```

In [21]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.25)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[21]:
```
((75, 4), (25, 4), (75,), (25,))
```

In [22]:
```
x_train.head(2)
```

Out[22]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
18	5.7	3.8	1.7	0.3
95	5.7	3.0	4.2	1.2

In [23]:
```
x_test.head(2)
```

Out[23]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
5	5.4	3.9	1.7	0.4
24	4.8	3.4	1.9	0.2

In [24]:
```
y_train.head(2)
```

Out[24]:
```
18    0
95    1
Name: Species, dtype: int32
```

In [25]:
```
y_test.head(2)
```

Out[25]:
```
5     0
24    0
Name: Species, dtype: int32
```

Import Model/algorithm

In [26]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

Train Model

In [27]:
```
lr.fit(x_train,y_train)
```

Out[27]:
```
LogisticRegression()
```

Predict Model

In [28]:
```
y_pred_lr=lr.predict(x_test)
y_pred_lr[:5],y_test.values[:5]
```

Out[28]:
```
(array([0, 0, 1, 1, 0]), array([0, 0, 1, 1, 0]))
```

Evaluation

In [29]:
```
print(lr.score(x_train,y_train))
print(lr.score(x_test,y_test))
```

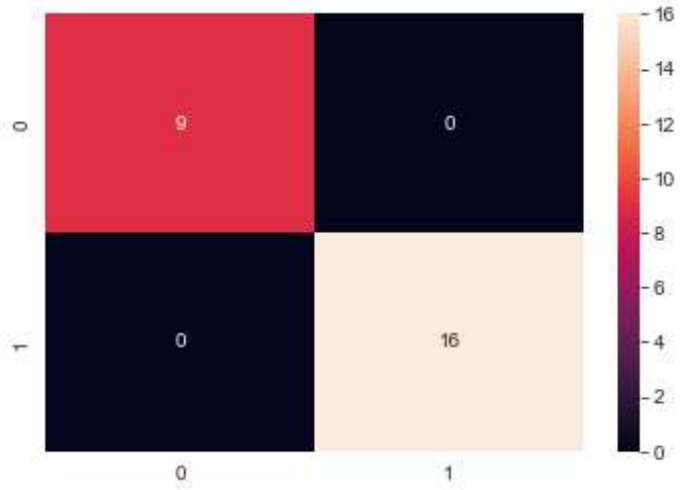
```
1.0
1.0
```

In [30]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_s
print(confusion_matrix(y_pred_lr,y_test))
print(classification_report(y_pred_lr,y_test))
print(f'model_score- {lr.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_lr,y_test)} ')
```

```
[[ 9  0]
 [ 0 16]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```

Vsualization

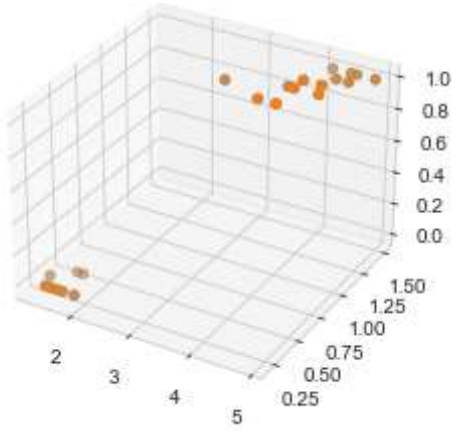
In [31]:
```
cm=confusion_matrix(y_pred_lr,y_test)
sns.heatmap(cm,annot=True)
plt.show()
```

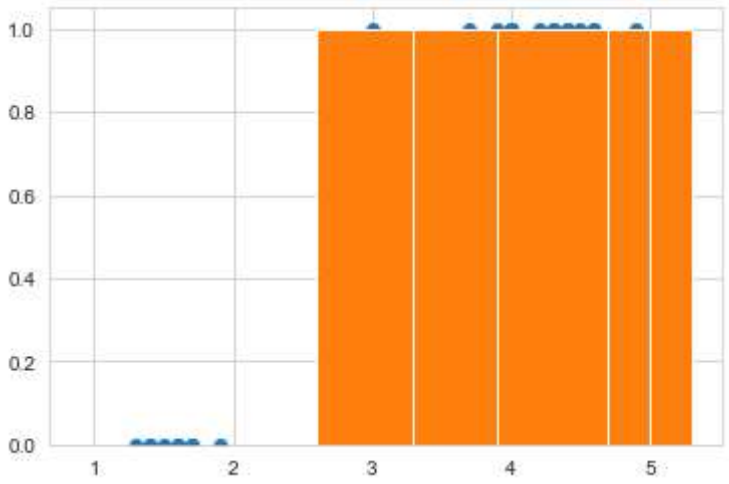


https://datascience.stackexchange.com/questions/65839/macro-average-and-weighted-average-meaning-in-classification-report (https://datascience.stackexchange.com/questions/65839/macro-average-and-weighted-average-meaning-in-classification-report)

https://www.kaggle.com/getting-started/175898 (https://www.kaggle.com/getting-started/175898)

In [32]:
```
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_lr,'black'
plt.show()
plt.scatter(x_test['PetalLengthCm'],y_test)
plt.bar(x_test['PetalLengthCm'],y_pred_lr)
plt.show()
```

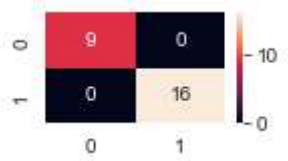




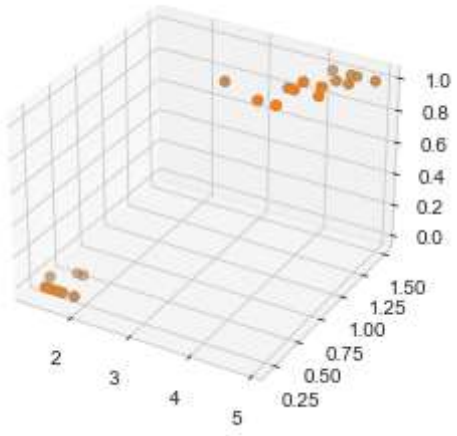
Decision Tree Classifier

In [33]:
```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train,y_train)
y_pred_dtc=dtc.predict(x_test)
print(f'Predicted_y{y_pred_dtc[:5]} Actual_y{y_test.values[:5]}')
print(confusion_matrix(y_pred_dtc,y_test))
from sklearn.metrics import confusion_matrix,classification_report,accuracy_s
cm=confusion_matrix(y_pred_dtc,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_dtc,y_test))
print(f'model_score- {dtc.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_dtc,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_dtc,'black
plt.show()
```

```
Predicted_y[0 0 1 1 0] Actual_y[0 0 1 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```

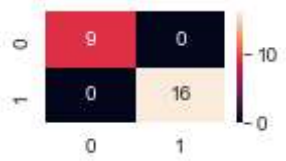


RandomForestClassifier

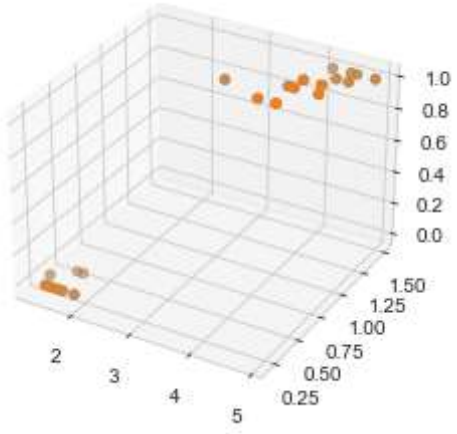
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
(https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

In [34]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
y_pred_rfc=rfc.predict(x_test)
print(f' predicted_y-{y_pred_rfc} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_rfc,y_test))
cm=confusion_matrix(y_pred_rfc,y_test)
plt.figure(figsize=(2,1))
sns.heatmap(cm,annot=True)
plt.show()
print(classification_report(y_pred_rfc,y_test))
print(f'model_score- {rfc.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_rfc,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_rfc,'black
plt.show()
```

```
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```

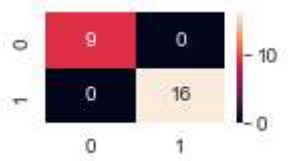


K-Nearest Neighbours

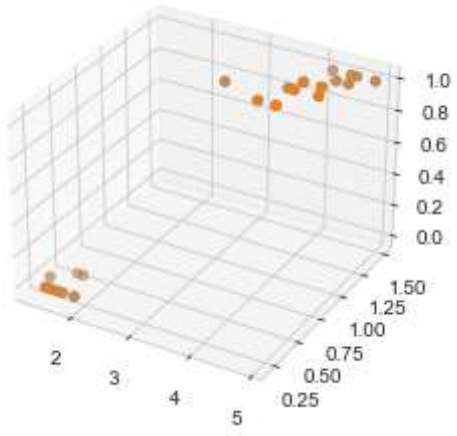
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
knc=KNeighborsClassifier()
knc.fit(x_train,y_train)
y_pred_knc=knc.predict(x_test)
print(f'Predicted_y{y_pred_knc[:5]} Actual_y{y_test.values[:5]}')
print(confusion_matrix(y_pred_knc,y_test))
from sklearn.metrics import confusion_matrix,classification_report,accuracy_s
cm=confusion_matrix(y_pred_knc,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_knc,y_test))
print(f'model_score- {dtc.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_knc,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_knc,'black
plt.show()
```

```
Predicted_y[0 0 1 1 0] Actual_y[0 0 1 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```

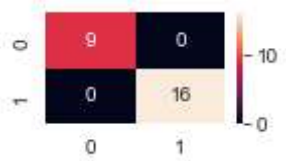


Naive Bayes

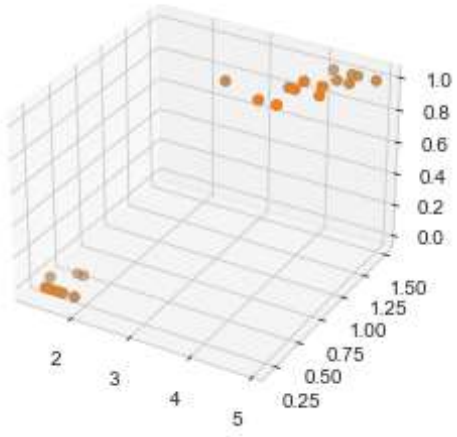
- gaussian
- bernoulli
- multinomial

In [36]:
```
print( 'Naive Bayes - gaussian')
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(x_train,y_train)
y_pred_gnb=gnb.predict(x_test)
print(f' predicted_y-{y_pred_gnb} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_gnb,y_test))
cm=confusion_matrix(y_pred_gnb,y_test)
plt.figure(figsize=(2,1))
sns.heatmap(cm,annot=True)
plt.show()
print(classification_report(y_pred_gnb,y_test))
print(f'model_score- {gnb.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_gnb,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_gnb,'black
plt.show()
```

```
Naive Bayes - gaussian
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
```

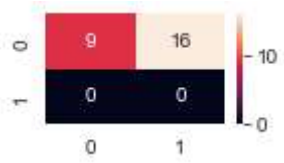



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```


1.0
0.8
0.6
0.4
0.2
0.0
1.50
1.25
1.00
0.75
0.50
0.25
2
3
4
5

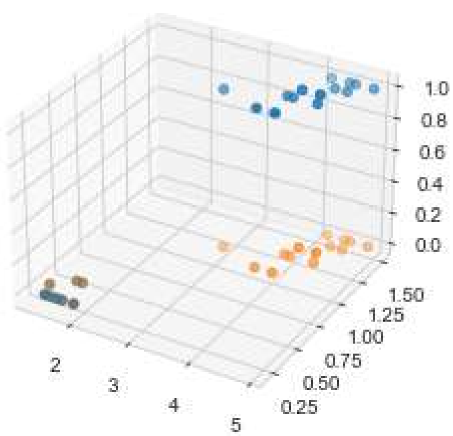
In [37]:
```
print( 'Naive Bayes - Bernoulli')
from sklearn.naive_bayes import BernoulliNB
bnb=BernoulliNB()
bnb.fit(x_train,y_train)
y_pred_bnb=bnb.predict(x_test)
print(f' predicted_y-{y_pred_bnb} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_bnb,y_test))
cm=confusion_matrix(y_pred_bnb,y_test)
plt.figure(figsize=(2,1))
sns.heatmap(cm,annot=True)
plt.show()
print(classification_report(y_pred_bnb,y_test))
print(f'model_score- {bnb.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_bnb,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_bnb,'black
plt.show()
plt.scatter(x_test['PetalLengthCm'],y_test)
plt.scatter(x_test['PetalLengthCm'],y_pred_bnb)
```

```
Naive Bayes - Bernoulli
 predicted_y-[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] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9 16]
 [ 0  0]]
```

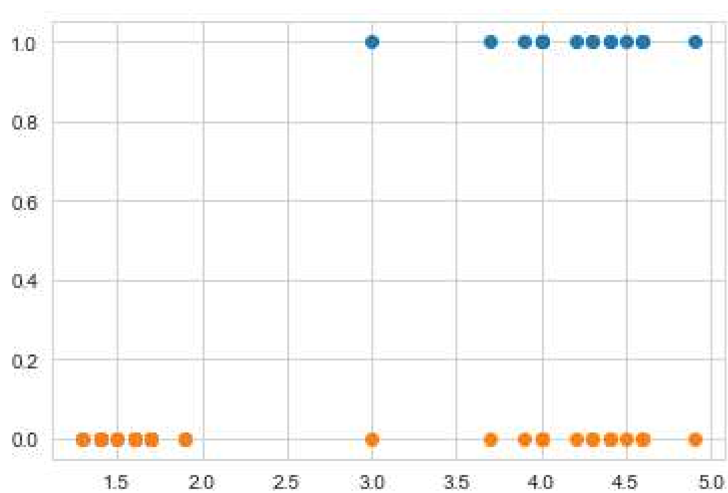



```
              precision    recall  f1-score   support

           0       1.00      0.36      0.53        25
           1       0.00      0.00      0.00         0

    accuracy                           0.36        25
   macro avg       0.50      0.18      0.26        25
weighted avg       1.00      0.36      0.53        25

model_score- 0.36
accuracy_score- 0.36
```

```
C:\Users\omkan\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1248: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\omkan\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1248: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\omkan\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1248: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

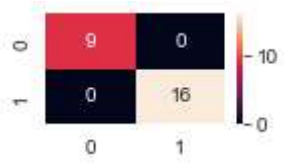


Out[37]: <matplotlib.collections.PathCollection at 0x1eb9cc6a5b0>

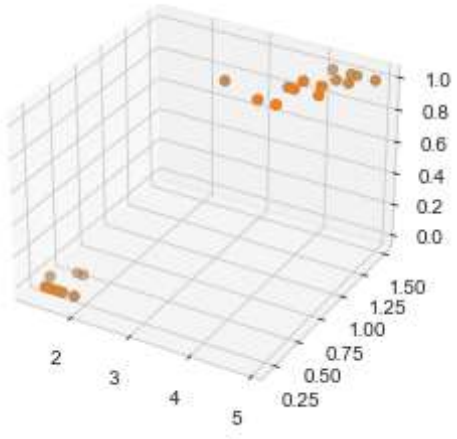


In [38]:
```
print( 'Naive Bayes - Multinomial')
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
y_pred_mnb=mnb.predict(x_test)
print(f' predicted_y-{y_pred_mnb} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_mnb,y_test))
cm=confusion_matrix(y_pred_mnb,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_mnb,y_test))
print(f'model_score- {mnb.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_mnb,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_mnb,'black
plt.show()
```

```
Naive Bayes - Multinomial
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```



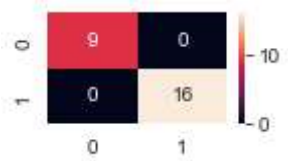
Support Vector Machine - Classification

- kernel - poly, linear, rbf (Radial Basis Function)

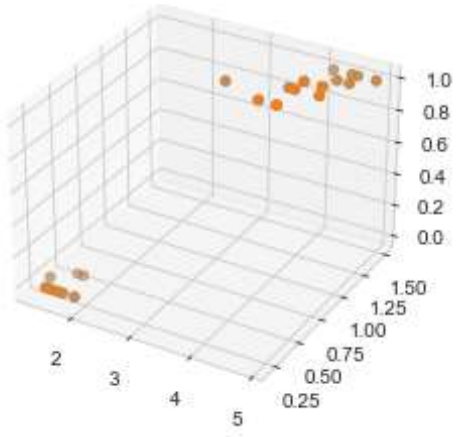
In [39]:
```
from sklearn.svm import SVC
```

In [40]:
```
print(' svc - poly')
svcp=SVC(kernel='poly', degree=2, C=.1)
svcp.fit(x_train,y_train)
y_pred_svcp=svcp.predict(x_test)
print(f' predicted_y-{y_pred_svcp} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_svcp,y_test))
cm=confusion_matrix(y_pred_svcp,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_svcp,y_test))
print(f'model_score- {svcp.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_svcp,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_svcp,'blac
plt.show()
```

```
 svc - poly
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```

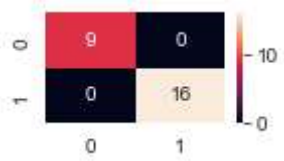



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```

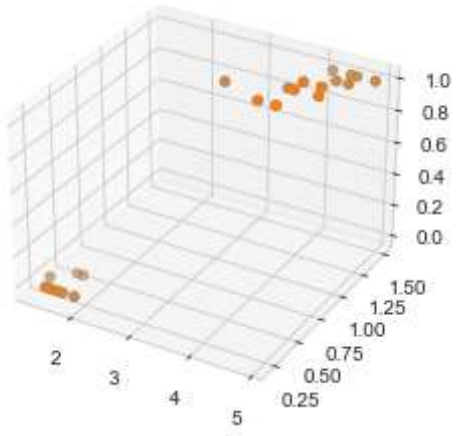

1.0
0.8
0.6
0.4
0.2
0.0
1.50
1.25
1.00
0.75
0.50
0.25
2
3
4
5

In [41]:
```
print(' svc - linear')
svcl=SVC(kernel='linear', degree=2, C=.1).fit(x_train,y_train)
y_pred_svcl=svcl.predict(x_test)
print(f' predicted_y-{y_pred_svcl} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_svcl,y_test))
cm=confusion_matrix(y_pred_svcl,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_svcl,y_test))
print(f'model_score- {svcl.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_svcl,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_svcl,'blac
plt.show()
```

```
 svc - linear
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```

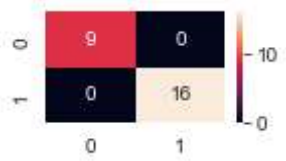



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```



In [42]:
```
print(' svc - rbf')
svcrbf=SVC(kernel='rbf', degree=2, C=.1)
svcrbf.fit(x_train,y_train)
y_pred_rbf=svcl.predict(x_test)
print(f' predicted_y-{y_pred_rbf} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_rbf,y_test))
cm=confusion_matrix(y_pred_rbf,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_rbf,y_test))
print(f'model_score- {svcrbf.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_rbf,y_test)} ')
```

```
 svc - rbf
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```



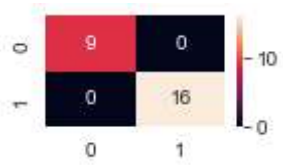

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```

In [43]:
```
print('linearSVC')
from sklearn.svm import SVC
lsvc=SVC(C=.1, degree=.2)
```

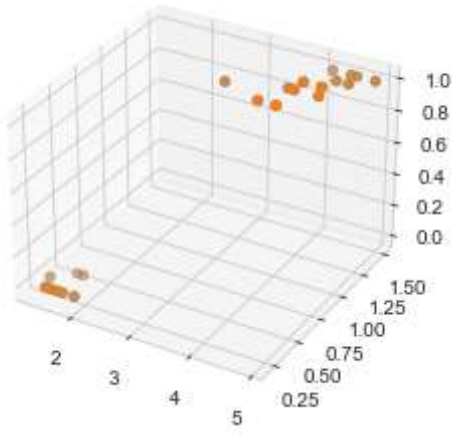
```
linearSVC
```

In [44]:
```
lsvc.fit(x_train,y_train)
y_pred_lsvc=lsvc.predict(x_test)
print(f' predicted_y-{y_pred_lsvc} actual_y-{y_test.values}')
print(confusion_matrix(y_pred_lsvc,y_test))
cm=confusion_matrix(y_pred_lsvc,y_test)
plt.figure(figsize=(2,1))
print(sns.heatmap(cm,annot=True))
plt.show()
print(classification_report(y_pred_lsvc,y_test))
print(f'model_score- {lsvc.score(x_test,y_test)} ')
print(f'accuracy_score- {accuracy_score(y_pred_lsvc,y_test)} ')
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_lsvc,'blac
plt.show()
```

```
 predicted_y-[0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0] actual_y-[0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0]
[[ 9  0]
 [ 0 16]]
AxesSubplot(0.125,0.125;0.62x0.755)
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        16

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

model_score- 1.0
accuracy_score- 1.0
```



https://datascience.stackexchange.com/questions/12355/support-vector-classification-kernels-linear-poly-rbf-has-all-same-score (https://datascience.stackexchange.com/questions/12355/support-vector-classification-kernels-linear-poly-rbf-has-all-same-score)

Neural Networks - classification

In [45]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
```

In [46]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[46]:
```
((75, 4), (25, 4), (75,), (25,))
```

In [47]:

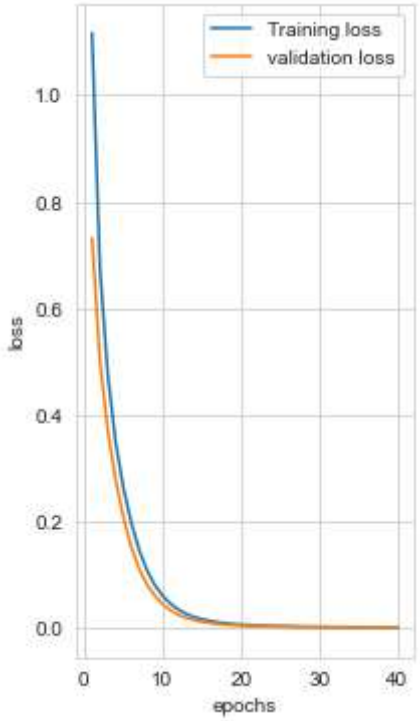
```
inputs = tf.keras.Input(shape=(4,))
x=tf.keras.layers.Dense(256,activation='relu')(inputs)
x=tf.keras.layers.Dense(256,activation='relu')(x)
# two hidden input layers and 256 node
outputs=tf.keras.layers.Dense(3,activation = 'softmax')(x)
# one hidden output layers and 3 node
model = tf.keras.Model(inputs,outputs)
model.compile(optimizer = 'adam',loss='sparse_categorical_crossentropy',metri
batch_size=32
epochs = 40 # batch size * number of iterations = epoch
history = model.fit(x_train,y_train,validation_split=0.2,batch_size=batch_siz
#model.fit(x_train,y_train,validation_split=0.2,batch_size=batch_size,epochs=
```

```
Epoch 1/40
2/2 [==============================] - 1s 153ms/step - loss: 1.1161 - accura
cy: 0.2333 - val_loss: 0.7317 - val_accuracy: 0.4667
Epoch 2/40
2/2 [==============================] - 0s 19ms/step - loss: 0.6763 - accurac
y: 0.6833 - val_loss: 0.5016 - val_accuracy: 1.0000
Epoch 3/40
2/2 [==============================] - 0s 16ms/step - loss: 0.4771 - accurac
y: 1.0000 - val_loss: 0.3722 - val_accuracy: 1.0000
Epoch 4/40
2/2 [==============================] - 0s 26ms/step - loss: 0.3496 - accurac
y: 1.0000 - val_loss: 0.2791 - val_accuracy: 1.0000
Epoch 5/40
2/2 [==============================] - 0s 24ms/step - loss: 0.2631 - accurac
y: 1.0000 - val_loss: 0.2077 - val_accuracy: 1.0000
Epoch 6/40
2/2 [==============================] - 0s 25ms/step - loss: 0.1985 - accurac
y: 1.0000 - val_loss: 0.1527 - val_accuracy: 1.0000
Epoch 7/40
2/2 [==============================] - 0s 24ms/step - loss: 0.1475 - accurac
y: 1.0000 - val_loss: 0.1118 - val_accuracy: 1.0000
Epoch 8/40
2/2 [==============================] - 0s 27ms/step - loss: 0.1089 - accurac
y: 1.0000 - val_loss: 0.0825 - val_accuracy: 1.0000
Epoch 9/40
2/2 [==============================] - 0s 32ms/step - loss: 0.0814 - accurac
y: 1.0000 - val_loss: 0.0612 - val_accuracy: 1.0000
Epoch 10/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0610 - accurac
y: 1.0000 - val_loss: 0.0456 - val_accuracy: 1.0000
Epoch 11/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0463 - accurac
y: 1.0000 - val_loss: 0.0338 - val_accuracy: 1.0000
Epoch 12/40
2/2 [==============================] - 0s 27ms/step - loss: 0.0351 - accurac
y: 1.0000 - val_loss: 0.0254 - val_accuracy: 1.0000
Epoch 13/40
2/2 [==============================] - 0s 28ms/step - loss: 0.0271 - accurac
y: 1.0000 - val_loss: 0.0195 - val_accuracy: 1.0000
Epoch 14/40
2/2 [==============================] - 0s 25ms/step - loss: 0.0213 - accurac
y: 1.0000 - val_loss: 0.0153 - val_accuracy: 1.0000
Epoch 15/40
2/2 [==============================] - 0s 25ms/step - loss: 0.0171 - accurac
y: 1.0000 - val_loss: 0.0123 - val_accuracy: 1.0000
Epoch 16/40
2/2 [==============================] - 0s 29ms/step - loss: 0.0140 - accurac
y: 1.0000 - val_loss: 0.0100 - val_accuracy: 1.0000
Epoch 17/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0115 - accurac
y: 1.0000 - val_loss: 0.0082 - val_accuracy: 1.0000
Epoch 18/40
2/2 [==============================] - 0s 19ms/step - loss: 0.0096 - accurac
y: 1.0000 - val_loss: 0.0069 - val_accuracy: 1.0000
Epoch 19/40
2/2 [==============================] - 0s 29ms/step - loss: 0.0083 - accurac
y: 1.0000 - val_loss: 0.0059 - val_accuracy: 1.0000
```

```
Epoch 20/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0071 - accurac
y: 1.0000 - val_loss: 0.0051 - val_accuracy: 1.0000
Epoch 21/40
2/2 [==============================] - 0s 20ms/step - loss: 0.0062 - accurac
y: 1.0000 - val_loss: 0.0044 - val_accuracy: 1.0000
Epoch 22/40
2/2 [==============================] - 0s 26ms/step - loss: 0.0055 - accurac
y: 1.0000 - val_loss: 0.0039 - val_accuracy: 1.0000
Epoch 23/40
2/2 [==============================] - 0s 16ms/step - loss: 0.0050 - accurac
y: 1.0000 - val_loss: 0.0035 - val_accuracy: 1.0000
Epoch 24/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0045 - accurac
y: 1.0000 - val_loss: 0.0032 - val_accuracy: 1.0000
Epoch 25/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0041 - accurac
y: 1.0000 - val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 26/40
2/2 [==============================] - 0s 32ms/step - loss: 0.0038 - accurac
y: 1.0000 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 27/40
2/2 [==============================] - 0s 33ms/step - loss: 0.0035 - accurac
y: 1.0000 - val_loss: 0.0025 - val_accuracy: 1.0000
Epoch 28/40
2/2 [==============================] - 0s 25ms/step - loss: 0.0032 - accurac
y: 1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 29/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0031 - accurac
y: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 30/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0029 - accurac
y: 1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 31/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0027 - accurac
y: 1.0000 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 32/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0026 - accurac
y: 1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 33/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0024 - accurac
y: 1.0000 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 34/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0023 - accurac
y: 1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 35/40
2/2 [==============================] - 0s 22ms/step - loss: 0.0022 - accurac
y: 1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 36/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0021 - accurac
y: 1.0000 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 37/40
2/2 [==============================] - 0s 32ms/step - loss: 0.0021 - accurac
y: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 38/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0020 - accurac
y: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
```

```
Epoch 39/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0019 - accurac
y: 1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 40/40
2/2 [==============================] - 0s 24ms/step - loss: 0.0018 - accurac
y: 1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
```

In [48]:
```
plt.figure(figsize = (3,6))
epoch_range = range(1,epochs +1)
train_loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(epoch_range,train_loss,label='Training loss')
plt.plot(epoch_range,val_loss, label = 'validation loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



In [49]:
```
loss,acc=model.evaluate(x_test,y_test)
print('loss',loss)
print('acc',acc)
```

```
1/1 [==============================] - 0s 16ms/step - loss: 0.0029 - accurac
y: 1.0000
loss 0.0029290372040122747
acc 1.0
```

REGRESSION

Independent variable and dependent variable In Regression

In [50]:
```
x=df.iloc[:,3:4]     # independent variable
y=df.iloc[:,4:5]     # dependent variable
```

In [51]:
```
x.head()
```

Out[51]:

	PetalLengthCm
0	1.4
1	1.4
2	1.3
3	1.5
4	1.4

In [52]:
```
y.head()
```

Out[52]:

	PetalWidthCm
0	0.2
1	0.2
2	0.2
3	0.2
4	0.2

In [53]:
```
x.shape, y.shape
```

Out[53]:
```
((150, 1), (150, 1))
```

split the data set

In [54]:
```
from sklearn.model_selection import train_test_split
```

In [55]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=.8)
```

In [56]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[56]:
```
((120, 1), (30, 1), (120, 1), (30, 1))
```

In [57]:
```
x_train.head(2)
```

Out[57]:

	PetalLengthCm
91	4.6
77	5.0

In [58]:
```
x_test.head(2)
```

Out[58]:

	PetalLengthCm
98	3.0
37	1.5

In [59]:
```
y_train.head(2)
```

Out[59]:

	PetalWidthCm
91	1.4
77	1.7

In [60]:
```
y_test.head(2)
```

Out[60]:

	PetalWidthCm
98	1.1
37	0.1

In [ ]:
```

```

SIMPLE LINEAR REGRESION

Import the model/algorithm

In [61]:
```
from sklearn.linear_model import LinearRegression # as slr
slr=LinearRegression()
```

Train the model

In [62]:
```
slr.fit(x_train,y_train)
```

Out[62]:
```
LinearRegression()
```

Predict

In [63]:
```
y_pred=slr.predict(x_test)
```

In [64]:
```
x_test.values[:5]
```

Out[64]:
```
array([[3. ],
       [1.5],
       [5.8],
       [5.4],
       [5.2]])
```

In [65]:
```
y_pred[:5]
```

Out[65]:
```
array([[0.87412307],
       [0.27081288],
       [2.00030209],
       [1.83941937],
       [1.75897802]])
```

In [66]:
```
y_test.values[:5]
```

Out[66]:
```
array([[1.1],
       [0.1],
       [2.2],
       [2.1],
       [2.3]])
```

evalution

In [67]:
```
print(slr.score(x_train,y_train))
print(slr.score(x_test,y_test))
```

```
0.9297411171423415
0.9131168764240425
```

In [68]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
print(mean_absolute_error(y_test,y_pred))
print(mean_squared_error(y_test,y_pred))
print(r2_score(y_test,y_pred))
```

```
0.19567010999121745
0.0703029274935456
0.9131168764240425
```

visualization

In [69]:
```
import matplotlib.pyplot as plt
```

In [70]:
```
slr.predict([[3]])
```

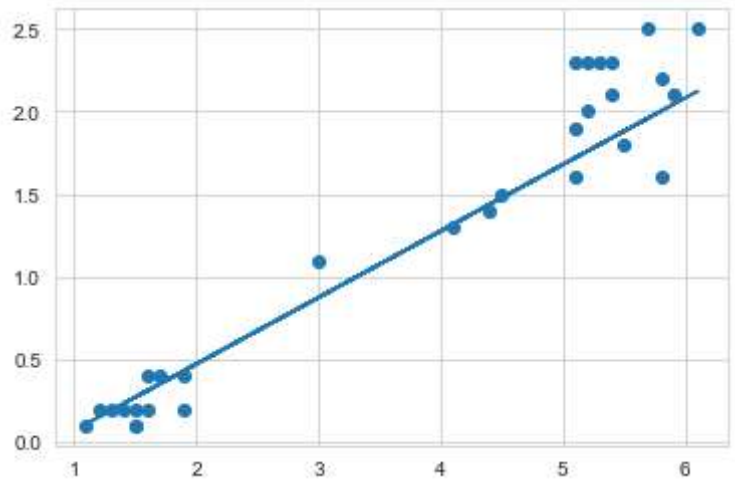
Out[70]:
```
array([[0.87412307]])
```

In [71]:
```
a=slr.coef_
b=slr.intercept_
```

In [72]:
```
y =a*3+b    # y=ax+b
y
```

Out[72]:
```
array([[0.87412307]])
```

In [73]:
```
plt.scatter(x_test,y_test)
plt.plot(x_test,y_pred)
plt.show()
```



MULTILINEAR REGRESSION

In [74]:
```
df.head()
```

Out[74]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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

Divide dataset into independent and dependent variable

In [75]:
```
x=df[['PetalLengthCm','PetalWidthCm']] # Independent variable
y=df['SepalLengthCm']                  # Dependent variable
x.shape,y.shape
```

Out[75]:
```
((150, 2), (150,))
```

Split independent & dependent variables into train dataset and test dataset

In [76]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=.8)

x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[76]:
```
((120, 2), (30, 2), (120,), (30,))
```

Import model from Library

In [77]:
```
from sklearn.linear_model import LinearRegression # as slr
mlr=LinearRegression()
```

Train Model with x & y train dataset

In [78]:
```
mlr.fit(x_train,y_train)
```

Out[78]:
```
LinearRegression()
```

Prediction with x_test and got predicted y

In [79]:
```
y_pred=mlr.predict(x_test)
y_pred[:5],y_test.values[:5]
```

Out[79]:
```
(array([6.26455548, 4.74433166, 4.93077365, 5.43991048, 4.84902194]),
 array([7. , 4.5, 5.4, 5.1, 4.8]))
```

Evaluation - Accuracy score

In [80]:
```
mlr.score(x_test,y_test)
```

Out[80]:
```
0.7140431598884107
```

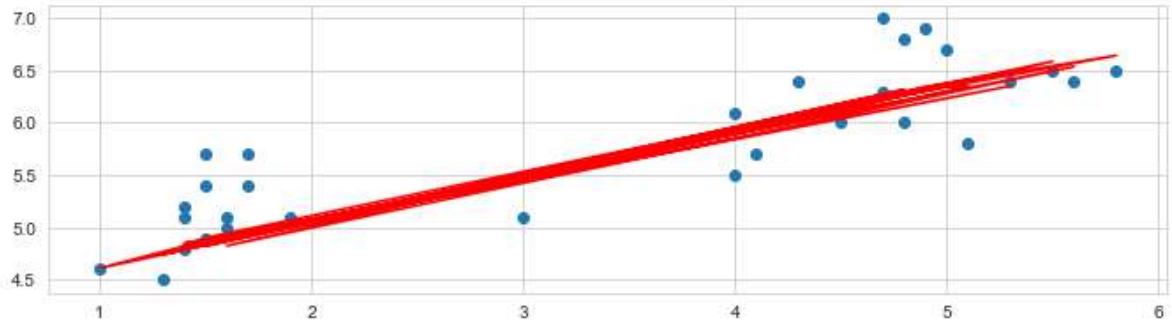
In [81]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
print(mean_absolute_error(y_test,y_pred))
print(mean_squared_error(y_test,y_pred))
print(r2_score(y_test,y_pred))
```

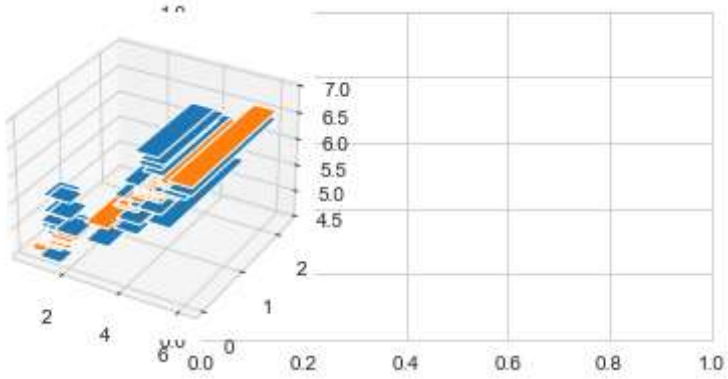
```
0.3028599497473194
0.1450246001117043
0.7140431598884107
```

Visualization

In [ ]:
```

```

In [82]:
```
plt.figure(figsize=(25,3))
plt.subplot(1,2,1)
plt.scatter(x_test['PetalLengthCm'],y_test)
plt.plot(x_test['PetalLengthCm'],y_pred,'red')
plt.show()
plt.figure(figsize=(10,3))
plt.subplot(1,2,2)
ax=plt.axes(projection='3d')
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred)
plt.show()
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred)
```





Out[82]: <BarContainer object of 30 artists>

In [ ]:

Polynomial Regression -

In [83]:
```
from sklearn.preprocessing import PolynomialFeatures
pf=PolynomialFeatures(degree=2)
x_train_pf=pf.fit_transform(x_train)
x_test_pf=pf.fit_transform(x_test)
```

In [84]:
```
x_test.head(2)
```

Out[84]:

	PetalLengthCm	PetalWidthCm
50	4.7	1.4
41	1.3	0.3

https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/#:~:text=Polynomial%20regression%20is%20a%20form%20of%20Linear%20regress (https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/#:~:text=Polynomial%20regression%20is%20a%20form%20of%20Linear%20regress

In [85]:
```
x_test_pf[:2]
```

Out[85]:
```
array([[ 1.  ,  4.7 ,  1.4 , 22.09,  6.58,  1.96],
       [ 1.  ,  1.3 ,  0.3 ,  1.69,  0.39,  0.09]])
```

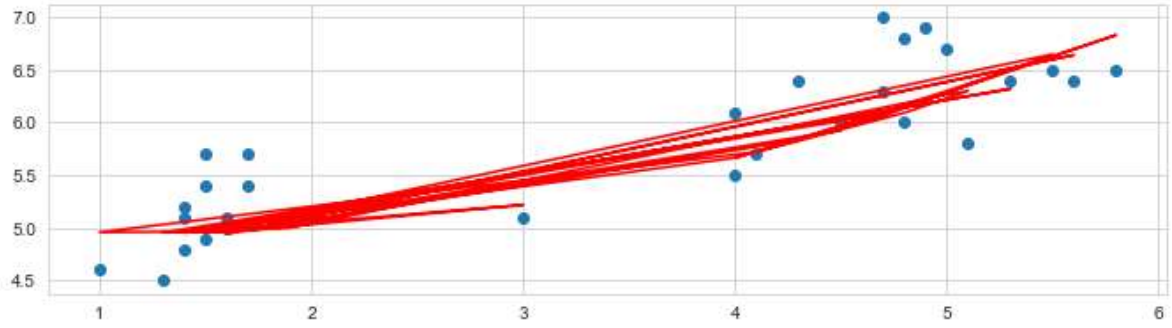
In [86]:
```
from sklearn.linear_model import LinearRegression
pr=LinearRegression()
pr.fit(x_train_pf,y_train)
y_pred_pr=pr.predict(x_test_pf)
y_pred_pr[:5],y_test.values[:5]
```

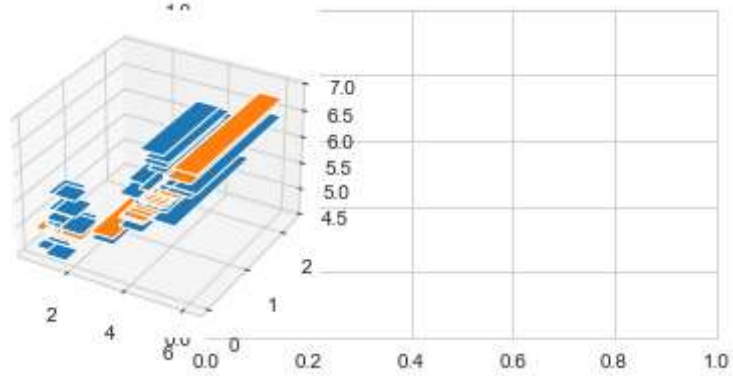
Out[86]:
```
(array([6.08103551, 4.95493418, 4.977728  , 5.21828609, 4.95334684]),
 array([7. , 4.5, 5.4, 5.1, 4.8]))
```

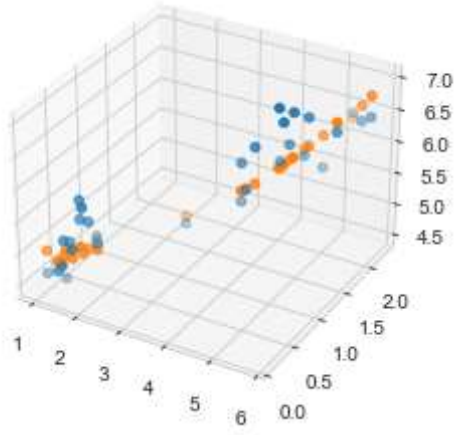
In [87]:
```
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_pr))
print('mean_squared_error',mean_squared_error(y_test,y_pred_pr))
mse=mean_squared_error(y_test,y_pred_pr)
print('r2_score',r2_score(y_test,y_pred_pr))
print('MODEL SCORE',pr.score(x_test_pf,y_test))
print(sqrt(mse))
```

```
------accuracy score--------
mean_absolute_error 0.3240276091521673
mean_squared_error 0.16369077927756345
r2_score 0.6772375310012113
MODEL SCORE 0.6772375310012113
0.4045871714199098
```

In [88]:
```
plt.figure(figsize=(25,3))
plt.subplot(1,2,1)
plt.scatter(x_test['PetalLengthCm'],y_test)
plt.plot(x_test['PetalLengthCm'],y_pred_pr,'red')
plt.show()
plt.figure(figsize=(10,3))
plt.subplot(1,2,2)
ax=plt.axes(projection='3d')
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_pr)
plt.show()
ax = plt.axes(projection ='3d')
fig = ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
fig = ax.scatter(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_pr,'re
plt.show()
```



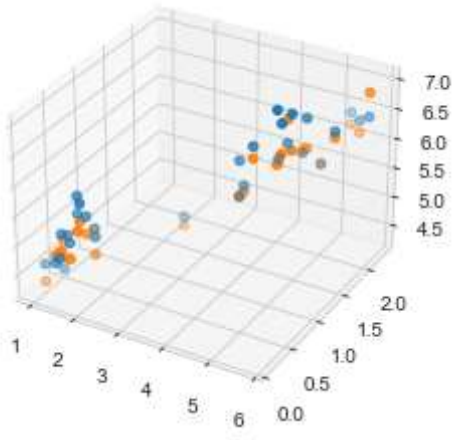




Decision tree Regressor

In [89]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor()
print(dtr.fit(x_train,y_train))
y_pred_dtr=dtr.predict(x_test)
print(y_pred_dtr[:5])
print(y_test.values[:5])
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_dtr))
print('mean_squared_error',mean_squared_error(y_test,y_pred_dtr))
mse=mean_squared_error(y_test,y_pred_dtr)
print('r2_score',r2_score(y_test,y_pred_dtr))
print('MODEL SCORE',dtr.score(x_test,y_test))
print(sqrt(mse))
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_dtr,'red')
plt.show()
```

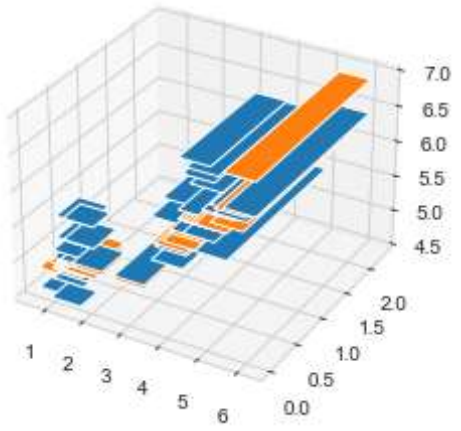
```
DecisionTreeRegressor()
[6.1        5.         5.1        4.95       4.92857143]
[7.  4.5 5.4 5.1 4.8]
------accuracy score--------
mean_absolute_error 0.29183333333333356
mean_squared_error 0.13057430272108855
r2_score 0.7425359906034097
MODEL SCORE 0.7425359906034097
0.3613506644813159
```



RandomForestRegressor

In [90]:
```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor(n_estimators=5,max_depth=3)
rfr.fit(x_train,y_train)
y_pred_rfr=rfr.predict(x_test)
print(y_pred_rfr[:5],y_test.values[:5])
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_rfr))
print('mean_squared_error',mean_squared_error(y_test,y_pred_rfr))
mse=mean_squared_error(y_test,y_pred_rfr)
print('r2_score',r2_score(y_test,y_pred_rfr))
print('MODEL SCORE',rfr.score(x_test,y_test))
print(sqrt(mse))
ax = plt.axes(projection ='3d')
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.bar(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_rfr)
plt.show()
```

```
[6.17281621 5.02102015 5.05012332 5.05012332 4.94959158] [7.  4.5 5.4 5.1 4.
8]
------accuracy score--------
mean_absolute_error 0.3080258431088217
mean_squared_error 0.15257221813355867
r2_score 0.699160905441673
MODEL SCORE 0.699160905441673
0.39060493869581153
```

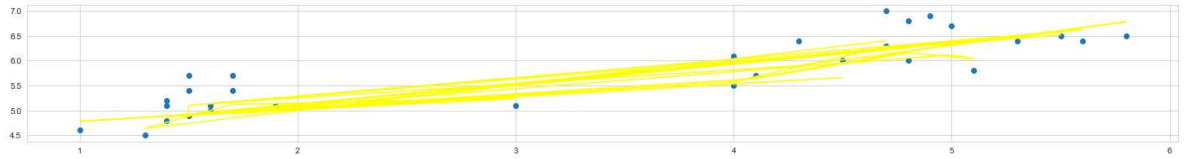


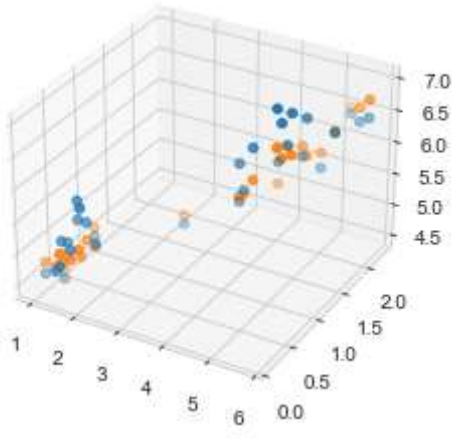
K-Nearest_neighbor

In [91]:
```
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train,y_train)
y_pred_knn=knn.predict(x_test)
print(y_pred_knn[:5],y_test.values[:5])
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_knn))
print('mean_squared_error',mean_squared_error(y_test,y_pred_knn))
print('r2_score',r2_score(y_test,y_pred_knn))
print('MODEL SCORE',knn.score(x_test,y_test))
```

```
[6.4  4.64 5.12 5.24 4.88] [7.  4.5 5.4 5.1 4.8]
------accuracy score--------
mean_absolute_error 0.29399999999999993
mean_squared_error 0.13582666666666662
r2_score 0.7321794759442644
MODEL SCORE 0.7321794759442644
```

In [92]:
```
plt.figure(figsize=(25,3))
plt.scatter(x_test['PetalLengthCm'],y_test)
plt.plot(x_test['PetalLengthCm'],y_pred_knn,'yellow')
plt.show()
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_knn)
plt.show()
```





Support Vector Machine - Regressor

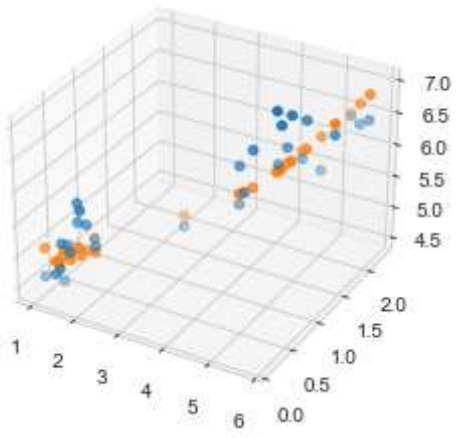
- https://www.saedsayad.com/support_vector_machine_reg.htm (https://www.saedsayad.com/support_vector_machine_reg.htm)
- https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/ (https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/)

In [93]:
```
from sklearn.svm import SVR
```

In [94]:
```
svr=SVR()
svr.fit(x_train,y_train)
y_pred_svr=svr.predict(x_test)
print(y_pred_svr[:5],y_test.values[:5])
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_svr))
print('mean_squared_error',mean_squared_error(y_test,y_pred_svr))
mse=mean_squared_error(y_test,y_pred_svr)
print('r2_score',r2_score(y_test,y_pred_svr))
print('MODEL SCORE',svr.score(x_test,y_test))
print(sqrt(mse))
```

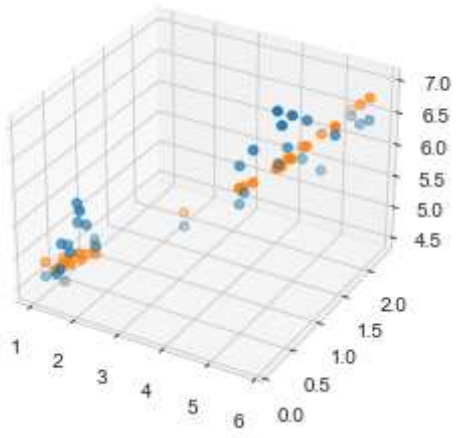
```
[6.06661625 5.01000585 4.99794693 5.27451893 5.00191259] [7.  4.5 5.4 5.1 4.
8]
------accuracy score--------
mean_absolute_error 0.3321669820345876
mean_squared_error 0.17025036123222761
r2_score 0.6643034679059574
MODEL SCORE 0.6643034679059574
0.41261405845199656
```

In [95]:
```
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_svr,'yello
plt.show()
```



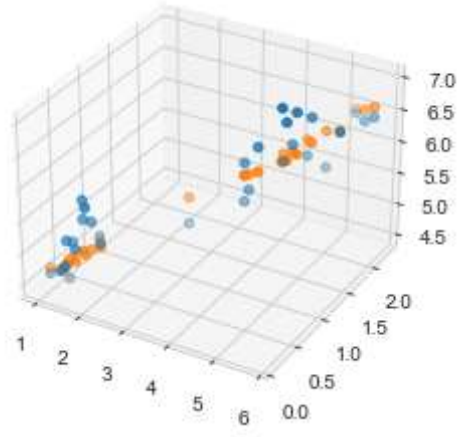
In [96]:
```
print(' svc - poly')
svcp=SVR(kernel='poly', degree=2, C=.1)
svcp.fit(x_train,y_train)
y_pred_svcp=svcp.predict(x_test)
print(f'predicted_y-{y_pred_svcp[:5]}')
print(f'actual_y-{y_test.values[:5]}')
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_svcp))
print('mean_squared_error',mean_squared_error(y_test,y_pred_svcp))
mse=mean_squared_error(y_test,y_pred_svcp)
print('r2_score',r2_score(y_test,y_pred_svcp))
print('MODEL SCORE',svcp.score(x_test,y_test))
print(sqrt(mse))
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_svcp,'yell
plt.show()
```

```
 svc - poly
predicted_y-[6.12104874 4.86709643 4.94047719 5.31962972 4.88163822]
actual_y-[7.  4.5 5.4 5.1 4.8]
------accuracy score--------
mean_absolute_error 0.3299633765603468
mean_squared_error 0.16463171630025422
r2_score 0.6753822086797197
MODEL SCORE 0.6753822086797197
0.4057483410936565
```



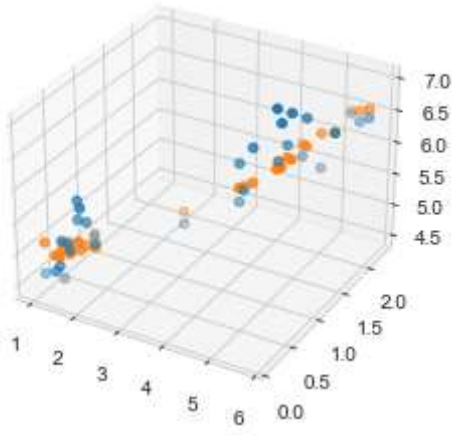
In [97]:
```
print(' svc - linear')
svcl=SVR(kernel='linear', degree=2, C=.1).fit(x_train,y_train)
y_pred_svcl=svcl.predict(x_test)
print(f'predicted_y-{y_pred_svcl[:5]}')
print(f'actual_y-{y_test.values[:5]}')
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_svcl))
print('mean_squared_error',mean_squared_error(y_test,y_pred_svcl))
mse=mean_squared_error(y_test,y_pred_svcl)
print('r2_score',r2_score(y_test,y_pred_svcl))
print('MODEL SCORE',svcl.score(x_test,y_test))
print(sqrt(mse))
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_svcl,'red'
plt.show()
```

```
 svc - linear
predicted_y-[6.18120077 4.80672237 4.96644849 5.51076795 4.83152816]
actual_y-[7.  4.5 5.4 5.1 4.8]
------accuracy score--------
mean_absolute_error 0.31253644964643434
mean_squared_error 0.15319478502714945
r2_score 0.6979333394872613
MODEL SCORE 0.6979333394872613
0.39140105394230795
```



In [98]:
```
print(' svc - rbf')
svcrbf=SVR(kernel='rbf', degree=2, C=.1)
svcrbf.fit(x_train,y_train)
y_pred_svcrbf=svcrbf.predict(x_test)
print(f'predicted_y-{y_pred_svcrbf[:5]}')
print(f'actual_y-{y_test.values[:5]}')
print('------accuracy score--------')
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from math import sqrt
print('mean_absolute_error',mean_absolute_error(y_test,y_pred_svcrbf))
print('mean_squared_error',mean_squared_error(y_test,y_pred_svcrbf))
mse=mean_squared_error(y_test,y_pred_svcrbf)
print('r2_score',r2_score(y_test,y_pred_svcrbf))
print('MODEL SCORE',svcl.score(x_test,y_test))
print(sqrt(mse))
ax = plt.axes(projection ='3d')
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_test)
ax.scatter3D(x_test['PetalLengthCm'],x_test['PetalWidthCm'],y_pred_svcrbf,'ye
plt.show()
```

```
 svc - rbf
predicted_y-[6.07551106 5.05798193 5.03970157 5.29232494 5.05507296]
actual_y-[7.  4.5 5.4 5.1 4.8]
------accuracy score--------
mean_absolute_error 0.3094455623876042
mean_squared_error 0.1580751891626363
r2_score 0.6883102483428871
MODEL SCORE 0.6979333394872613
0.39758670647122535
```



Unsupervised Learning

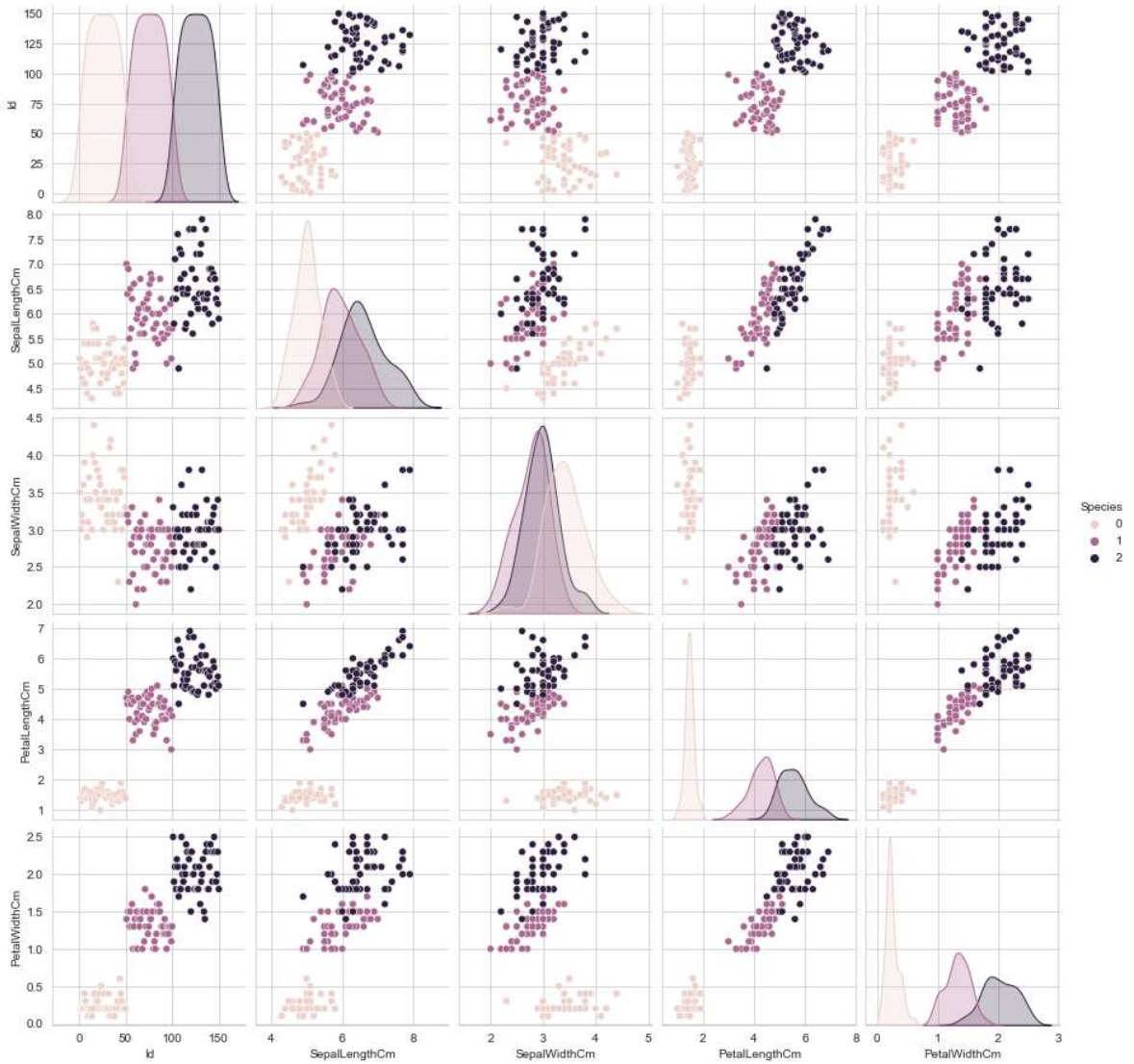
- K - Mean Cluster

Finding optimum no. of clusters

- K-Means clustring aims to partition data into k clusters in a way that data points in the same cluster are similar and data points in the different clusters are farther apart
- K-means clustering is unsupervised learning algorithm. Unsupervised learning algorthm is machine learning algorithm to analyze and cluster unlabeled datasets
- In this notebook we are going to cluster iris Dataset.

Data Set

In [99]:
```
sns.pairplot(data=df1,hue='Species')
plt.show()
```



In [100]:
```
df.head(2),df['Species'].value_counts()  # with label encoding on species
```

Out[100]:
```
(   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
 0   1            5.1           3.5            1.4           0.2        0
 1   2            4.9           3.0            1.4           0.2        0,
 0    50
 1    50
 2    50
 Name: Species, dtype: int64)
```

In [101]:
```
df1.head(2),df1['Species'].value_counts() # without label encoder
```

Out[101]:
```
(   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
 0   1            5.1           3.5            1.4           0.2        0
 1   2            4.9           3.0            1.4           0.2        0,
 0    50
 1    50
 2    50
 Name: Species, dtype: int64)
```

In [ ]:
```

```

Change data set into numpy arrays

In [102]:
```
dfkm=df.values          # dataframe convert into numpy array
print(dfkm[:3])
```

```
[[1.  5.1 3.5 1.4 0.2 0. ]
 [2.  4.9 3.  1.4 0.2 0. ]
 [3.  4.7 3.2 1.3 0.2 0. ]]
```

Define features/independent/ x variable and traget/dependent y variable

In [103]:
```
x=dfkm[:,1:5]    # feature
y=dfkm[:,-1]     # target
x[:3],y[:3]
```

Out[103]:
```
(array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2]]),
 array([0., 0., 0.]))
```

Import K_Means Model / Algorithm

In [104]:
```
from sklearn import cluster
cluster.KMeans
```

Out[104]:
```
sklearn.cluster._kmeans.KMeans
```

In [105]:
```
cluster.KMeans(n_clusters=3)
```

Out[105]:
```
KMeans(n_clusters=3)
```

```
km=cluster.KMeans(n_clusters=3,init='k-
means++',max_iter=300,n_init=10,random_state=0)
centroid=km.fit_predict(df.values[:,1:5])
```

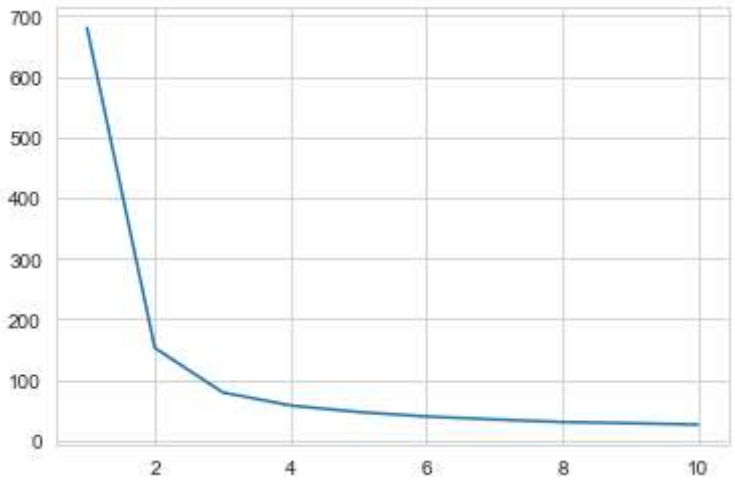
Import K_Means Model / Algorithm

In [106]:
```
from sklearn.cluster import KMeans
```

Defining cluster from elbow curve method

In [107]:
```
l=[]
for i in range(1,11):
    km=cluster.KMeans(i)
    km.fit(df.values[:,1:5])
    l.append(km.inertia_)
plt.plot(range(1,11),l[:10])
plt.show()
```

```
C:\Users\omkan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: U
serWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [108]:
```
km.inertia_
```

Out[108]:
```
25.909045084301678
```

https://stackoverflow.com/questions/69596239/how-to-avoid-memory-leak-when-dealing-with-kmeans-for-example-in-this-code-i-am (https://stackoverflow.com/questions/69596239/how-to-avoid-memory-leak-when-dealing-with-kmeans-for-example-in-this-code-i-am)

```
This is 'elbow curve method', it is widly use to identify optimum no. of
clusters.
the basic idea is for each k value, we calculate average distrances to the
centroid across all data points.

the curve looks an elbow. when average distances to the centroid doesn't
decrease significantly with every iteration, we can choose optimum number
of cluster. in the above plot. the elbow is at k-3
```

asign cluster

In [109]:
```
k_m=KMeans(n_clusters=3)    # define 3 cluster/group/class here
k_m
```

Out[109]:
```
KMeans(n_clusters=3)
```

Fitting the x/feature dataset here

In [110]:
```
km=k_m.fit(x)
```

In [111]:
```
km
```

Out[111]:
```
KMeans(n_clusters=3)
```

K-Mean algorithm clustring - output/target/y - Species

In [112]:
```
km_predict=km.labels_ # predicted categories of species
km_predict
```

Out[112]:
```
array([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, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

In [113]:
```
center_point=km.cluster_centers_     # Center point of each features with tar
```

In [114]:
```
df1.head(2),df1['Species'].value_counts()  # Associate with km.cluster_center
```

Out[114]:
```
(   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
 0   1            5.1           3.5            1.4           0.2        0
 1   2            4.9           3.0            1.4           0.2        0,
 0    50
 1    50
 2    50
 Name: Species, dtype: int64)
```

In [115]:
```
centroid=km_predict  # K_mean predict  y / target values
centroid             # now centroid has - setosa,versicolor,virginica as 0,1,
```

Out[115]:
```
array([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, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

In [116]:
```
x[:,1]    # all rows with sepallenthcm column
```

Out[116]:
```
array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3. ,
       3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
       3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. ,
       3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3, 3.2, 3.2,
       3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. , 2.2, 2.9, 2.9,
       3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. , 2.8, 3. ,
       2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5, 2.6,
       3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8, 3.3, 2.7, 3. , 2.9,
       3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3. , 2.5, 2.8, 3.2, 3. ,
       3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8, 3. , 2.8, 3. ,
       2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1, 3.1, 3.1, 2.7,
       3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ])
```

In [117]:
```
x[centroid==0,1]        #  x[ centroid == setosa,sepallength]
```

Out[117]:
```
array([3.1, 3. , 3.3, 3. , 2.9, 3. , 3. , 2.9, 2.5, 3.6, 3.2, 2.7, 3. ,
       3.2, 3. , 3.8, 2.6, 3.2, 2.8, 3.3, 3.2, 2.8, 3. , 2.8, 3.8, 2.8,
       2.6, 3. , 3.4, 3.1, 3.1, 3.1, 3.1, 3.2, 3.3, 3. , 3. , 3.4])
```

In [118]:
```
km.cluster_centers_
```

Out[118]:
```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

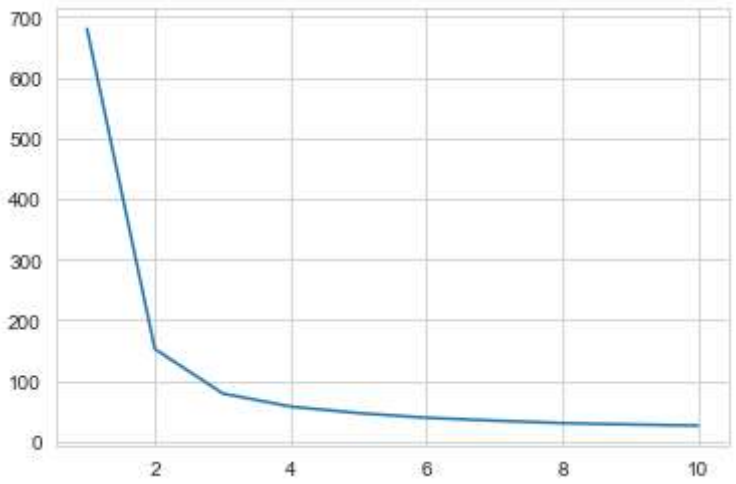
Plotting Prediction

In [119]:
```
km.cluster_centers_[:,0],km.cluster_centers_[:,1]
```

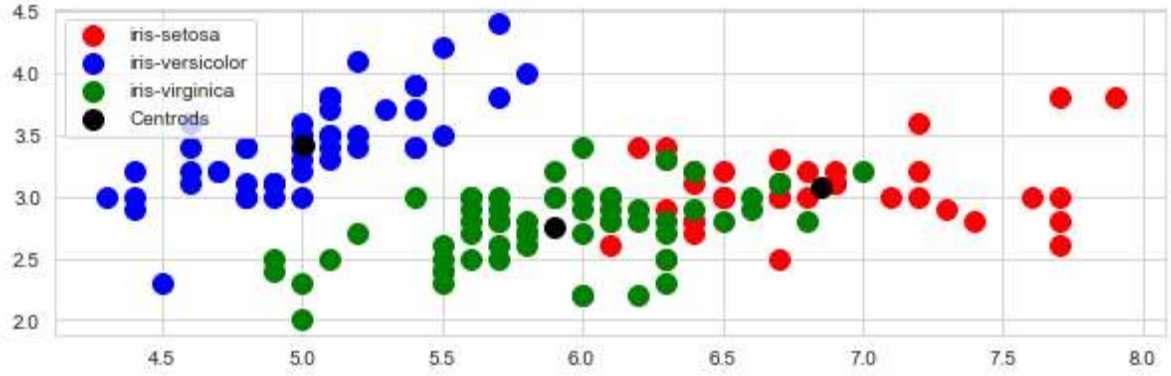
Out[119]:
```
(array([6.85     , 5.006    , 5.9016129]),
 array([3.07368421, 3.418     , 2.7483871 ]))
```

In [120]:
```
centroid_list=[]
no_of_clusters=range(1,11)
for i in no_of_clusters:
    kmeans=cluster.KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=1
    kmeans.fit(df.values[:,1:5])
    centroid_list.append(kmeans.inertia_)
plt.plot(no_of_clusters,centroid_list)
plt.show()
```

```
C:\Users\omkan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: U
serWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [121]:
```
plt.figure(figsize=(10,3))
plt.scatter(x[centroid==0,0],x[centroid==0,1],s=100,c='red',label='iris-setos
plt.scatter(x[centroid==1,0],x[centroid==1,1],s=100,c='blue',label='iris-vers
plt.scatter(x[centroid==2,0],x[centroid==2,1],s=100,c='green',label='iris-vir

plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],s=100,c='black'
plt.legend()
plt.show()
```



In [122]:
```
km.labels_
```

Out[122]:
```
array([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, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

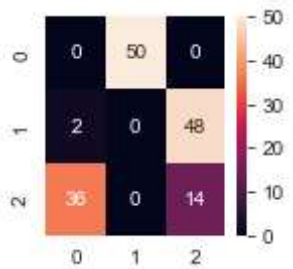
Checking our result

In [123]:
```
import pandas as pd
kmp=pd.crosstab(df1.values[:,-1],km.labels_)
print(kmp)
```

```
col_0   0   1   2
row_0
0.0     0  50   0
1.0     2   0  48
2.0    36   0  14
```

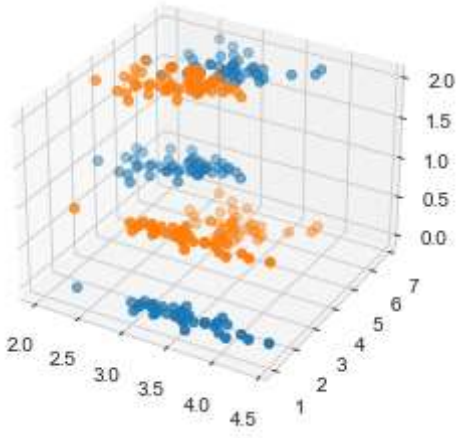
In [124]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_
cm=confusion_matrix(df.values[:,-1],km.labels_)
print(accuracy_score(df.values[:,-1],km.labels_))
print(classification_report(df.values[:,-1],km.labels_))
plt.figure(figsize=(2,2))
sns.heatmap(cm,annot=True)
plt.show()
```

```
0.09333333333333334
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        50
         1.0       0.00      0.00      0.00        50
         2.0       0.23      0.28      0.25        50

    accuracy                           0.09       150
   macro avg       0.08      0.09      0.08       150
weighted avg       0.08      0.09      0.08       150
```



In [ ]:

In [125]:
```
ax = plt.axes(projection ='3d')
ax.scatter3D(x[:,1],x[:,2],y)
ax.scatter3D(x[:,1],x[:,2],km.labels_,'yellow')
plt.show()
```

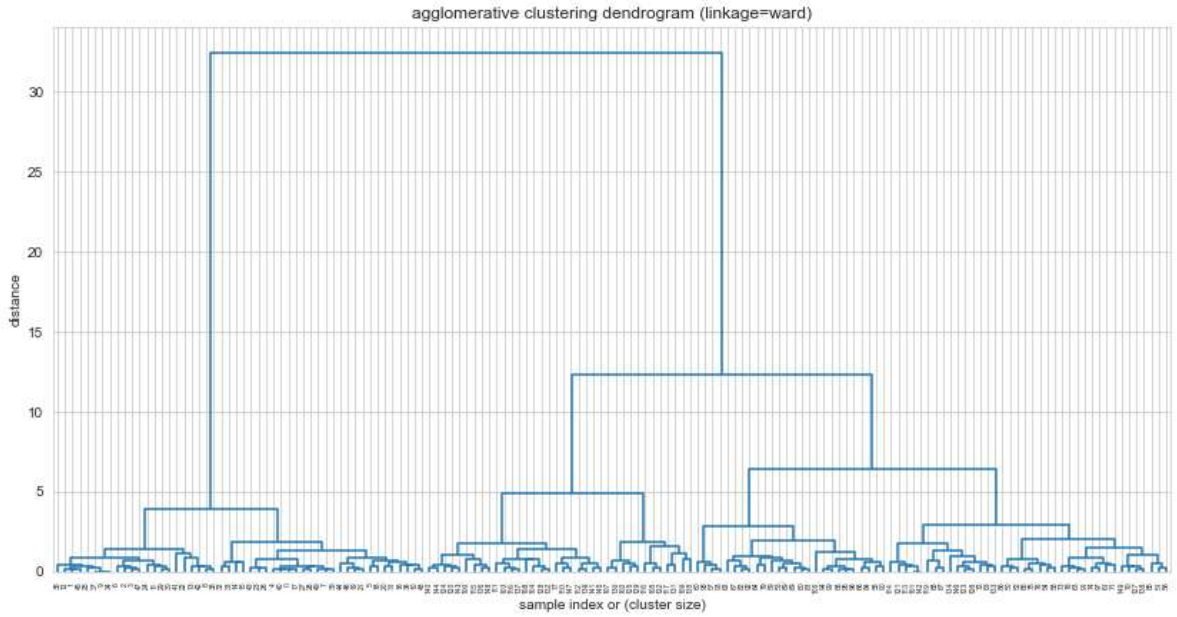


Hierachical Clustering

In [126]:
```
from sklearn import datasets
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
```

In [127]:
```
dfhc=df.values
dfhc[:5]
x=dfhc[:,1:5]
y=dfhc[:,-1]
```

In [128]:
```
linkage_matrix = linkage(x, 'ward')
plot = plt.figure(figsize=(14,7))
dendrogram(linkage_matrix,color_threshold=0)
plt.title('agglomerative clustering dendrogram (linkage=ward)')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
plt.show()
plt.show()
```



In [129]:
```
hc=AgglomerativeClustering(linkage='ward', n_clusters=3)
```

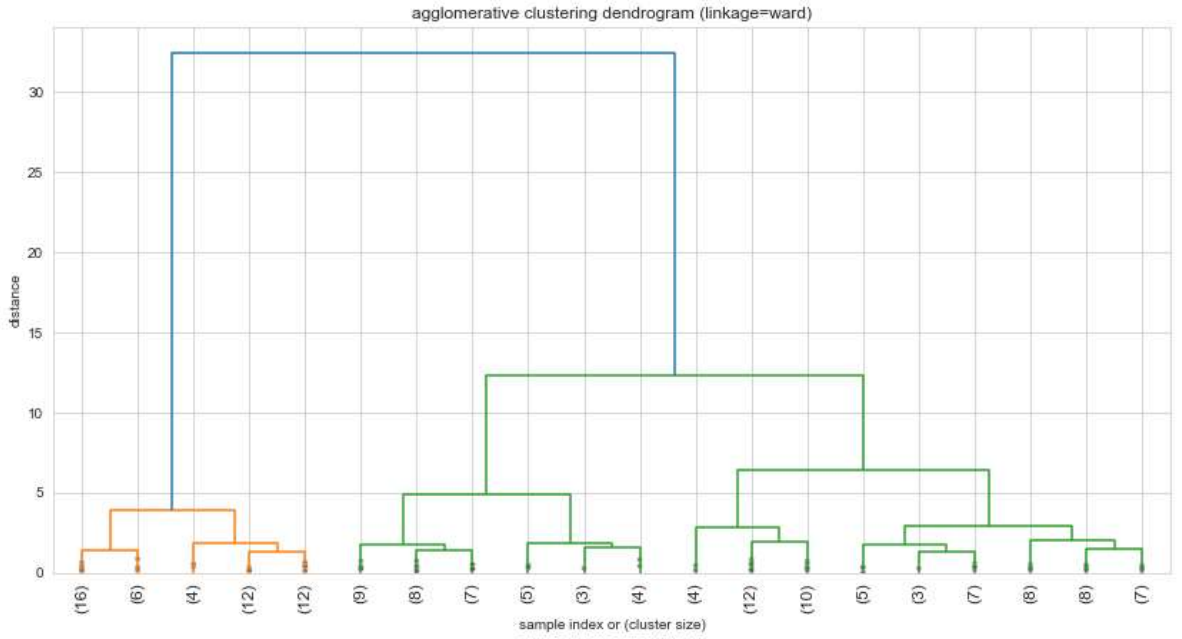
In [130]:
```
hc.fit(x)
hc_labels = hc.labels_
```

In [131]:
```
hc_labels
```

Out[131]:
```
array([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, 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, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int64)
```

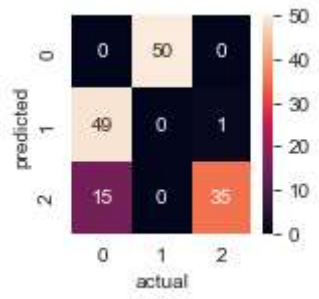
In [132]:

```
plot_2 = plt.figure(figsize=(14,7))
dendrogram(linkage_matrix,truncate_mode='lastp',   #dendrogram(linkage_matrix,
           p=20, # show only the last 20 merged clusters
           leaf_rotation=90., # rotates the x axis labels
           leaf_font_size=12.,# font size for x axis
           show_contracted=True) # to get a distribution impression
           # in truncated branches # color_threshold=0 )
plt.title('agglomerative clustering dendrogram (linkage=ward)')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
plt.show()
```



In [133]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_
cm=confusion_matrix(df.values[:,-1],hc.labels_)
print(accuracy_score(df.values[:,-1],hc.labels_))
print(classification_report(df.values[:,-1],hc.labels_))
plt.figure(figsize=(2,2))
sns.heatmap(cm,annot=True)
plt.xlabel('actual')
plt.ylabel('predicted')
plt.show()
import pandas as pd
kmp=pd.crosstab(df1.values[:,-1],hc.labels_)
print(kmp)
```

```
0.23333333333333334
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        50
         1.0       0.00      0.00      0.00        50
         2.0       0.97      0.70      0.81        50

    accuracy                           0.23       150
   macro avg       0.32      0.23      0.27       150
weighted avg       0.32      0.23      0.27       150
```




```
col_0   0   1   2
row_0
0.0     0  50   0
1.0    49   0   1
2.0    15   0  35
```

Om Kant Sharma