By:- subhayan mukherjee

In [2]:

```
# import pandas as pd
import numpy as np
import pandas as pd
df = pd.read_csv('http://bit.ly/w-data')
df
```

Out[2]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [4]:

```
df.describe()
```

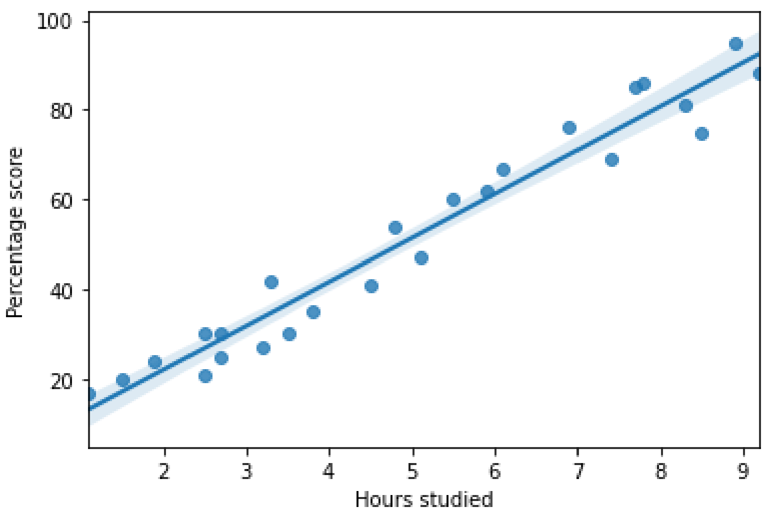
Out[4]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

In [37]:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.regplot(x='Hours',y='Scores',data=df)
print('This is the regression line with 95% confidence interval for that regression:')
plt.xlabel('Hours studied')
plt.ylabel('Percentage score')
plt.show()
```

```
This is the regression line with 95% confidence interval for that regressio
n:
```



In [4]:

```python
#for checking null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

In [9]:

```python
print('min score:', df['Hours'].min())
print('max score:', df['Hours'].max())
```

```
min score: 1.1
max score: 9.2
```

In [11]:

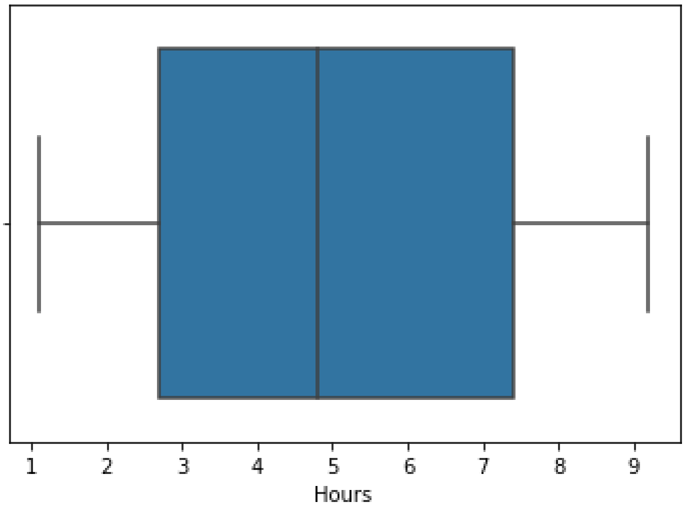
```python
print('min score:-', df['Scores'].min())
print('max score:-', df['Scores'].max())
```

```
min score:- 17
max score:- 95
```

In [14]:

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('http://bit.ly/w-data')
sns.boxplot(df["Hours"])
print('There is no outlier present')
```

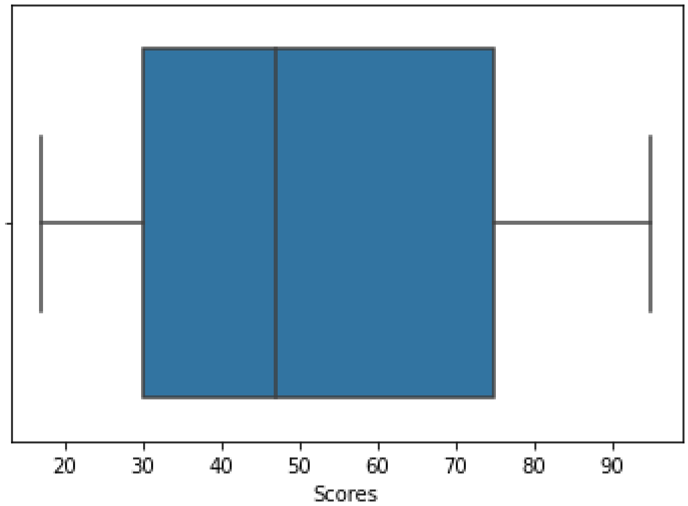
```
There is no outlier present

C:\Users\Subhayan\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.
12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```



In [16]:

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('http://bit.ly/w-data')
sns.boxplot(df["Scores"])
print('There is no outlier present')
```

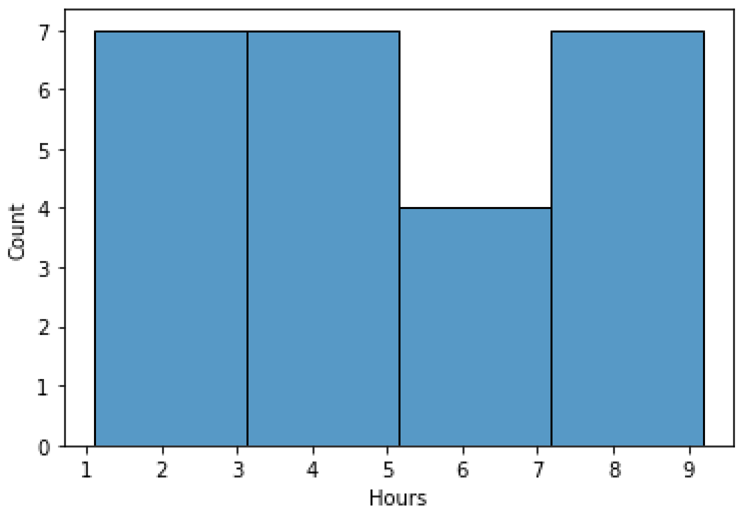
There is no outlier present



In [18]:

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('http://bit.ly/w-data')
sns.histplot(df["Hours"], bins=4)
print('There is no outlier present')
```

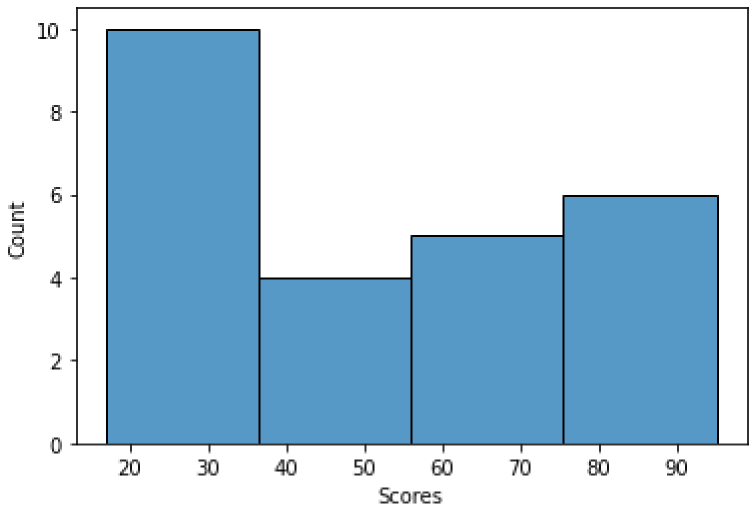
There is no outlier present



In [19]:

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('http://bit.ly/w-data')
sns.histplot(df["Scores"], bins=4)
print('There is no outlier present')
```

There is no outlier present



In [20]:

```
#The hours and Scores are distributed normally and we can perform linear regression eas
```

In [21]:

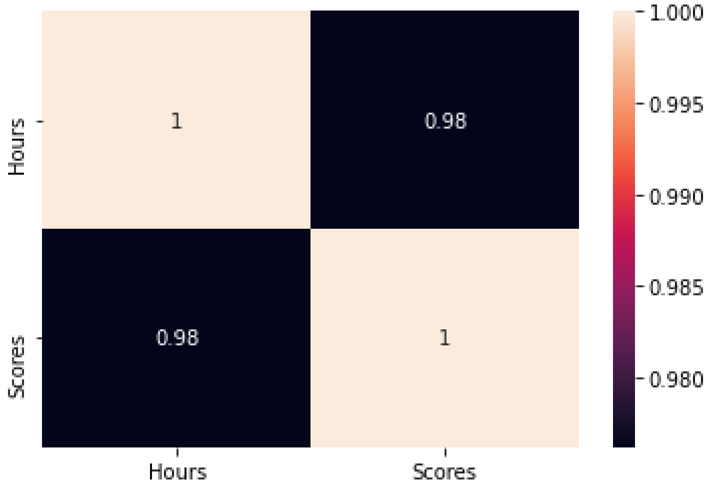
```
df = pd.read_csv('http://bit.ly/w-data')
column_1 = df["Hours"]
column_2 = df["Scores"]
correlation = column_1.corr(column_2)
correlation
```

Out[21]:

0.9761906560220887

In [34]:

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(df.corr(),annot=True)
plt.show()
print('The correlation value is greater zero')
```



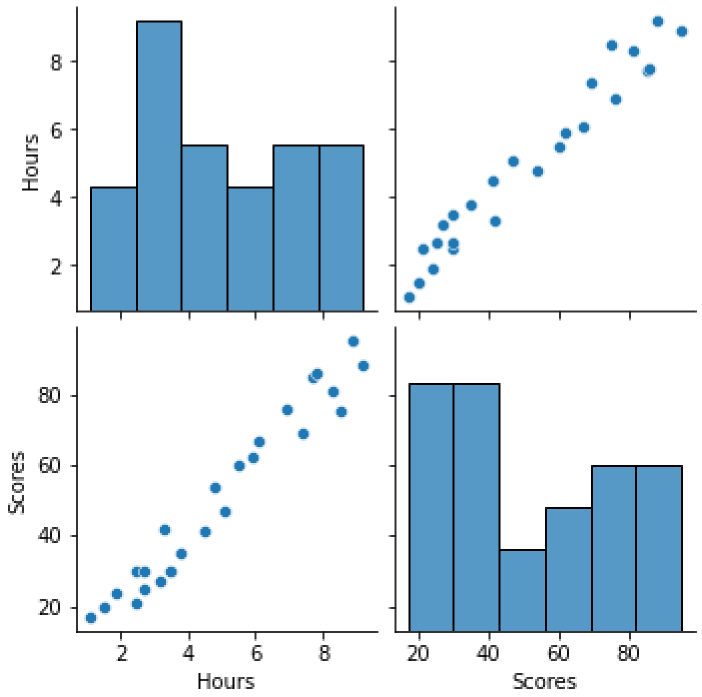

```
The correlation value is greater zero
```

In [38]:

```
sns.pairplot(df)
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x1f4be2d8520>
```



In [39]:

```
from sklearn.model_selection import train_test_split
```

In [41]:

```
x=df.iloc[:,:-1].values
y=df.iloc[:,1].values
x_train, x_test, y_train, y_test= train_test_split(x, y,train_size=0.60,test_size=0.40,
```

In [42]:

```
from sklearn.linear_model import LinearRegression
model= LinearRegression()
model.fit(x_train, y_train)
```

Out[42]:

```
LinearRegression()
```

In [43]:

```
y_pred = model.predict(x_test)
y_pred
```

Out[43]:

```
array([15.9477618 , 32.77394723, 74.344523  , 25.84551793, 59.49788879,
       38.71260091, 19.90686425, 78.30362545, 69.39564493, 11.98865934])
```

In [44]:

```
print('Test Score')
print(model.score(x_test, y_test))
print('Training Score')
print(model.score(x_train, y_train))
```

```
Test Score
0.956640847232559
Training Score
0.9440108159733135
```

In [48]:

```
print('Score of student who studied for 9.25 hours a day is:-', model.predict([[9.25]]
```

```
Score of student who studied for 9.25 hours a day is: [92.65537185]
```

```
summary:-
The dataset with 2 attributes Hours and Scores contains no null values. With the help of numpy, pandas, matplotlib, seaborn we have done the data analysis and visualization. e performed Linear Regression operation on the given dataset and the model had an accuracy of 95%. Thus, the model could predict the score for a student who studies for 9.25hrs in a day which is 92.65%.
```

In [5]:

```
# import pandas as pd
import numpy as np
import pandas as pd
df = pd.read_csv('http://bit.ly/w-data')
from sklearn.model_selection import train_test_split
x=df.iloc[:,:-1].values
y=df.iloc[:,1].values
x_train, x_test, y_train, y_test= train_test_split(x, y,train_size=0.60,test_size=0.40,
from sklearn.linear_model import LinearRegression
model= LinearRegression()
model.fit(x_train, y_train)
x_train
```

Out[5]:

```
array([[5.1],
       [7.7],
       [3.3],
       [8.3],
       [9.2],
       [6.1],
       [3.5],
       [2.7],
       [5.5],
       [2.7],
       [8.5],
       [2.5],
       [4.8],
       [8.9],
       [4.5]])
```

In [6]:

```
import pandas as pd
df = pd.read_csv("carprices.csv")
df.head()
```

Out[6]:

	Mileage	Age(yrs)	Sell Price($)
0	69000	6	18000
1	35000	3	34000
2	57000	5	26100
3	22500	2	40000
4	46000	4	31500

In [7]:

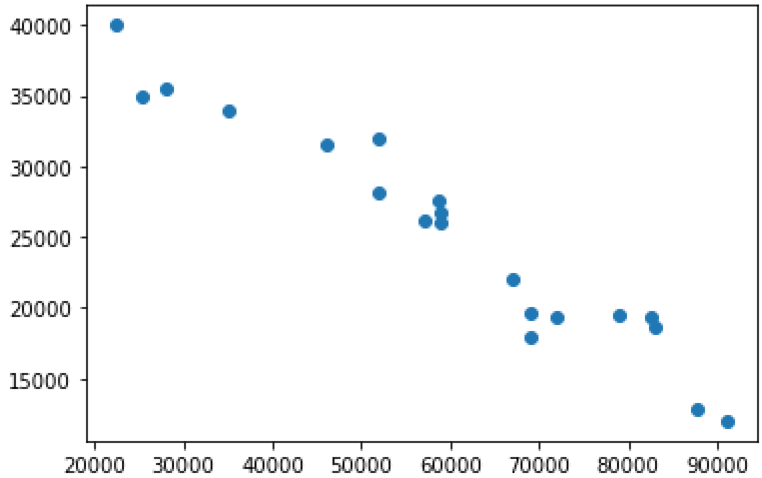
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [8]:

```
plt.scatter(df['Mileage'],df['Sell Price($)'])
```

Out[8]:

<matplotlib.collections.PathCollection at 0x230dcc83fa0>

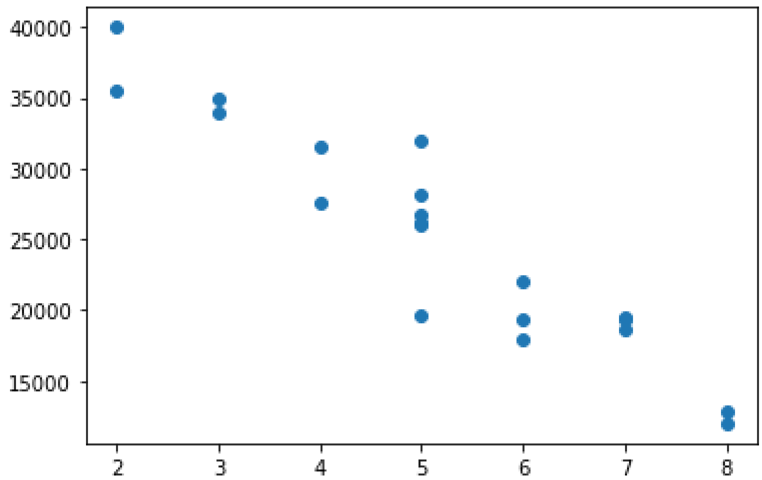


In [9]:

```
plt.scatter(df['Age(yrs)'],df['Sell Price($)'])
```

Out[9]:

<matplotlib.collections.PathCollection at 0x230dccf4af0>



Looking at above two scatter plots, using linear regression model makes sense as we can clearly see a linear relationship between our dependant (i.e. Sell Price) and independant variables (i.e. car age and car mileage)

The approach we are going to use here is to split available data in two sets

Training: We will train our model on this dataset Testing: We will use this subset to make actual predictions using trained model The reason we don't use same training set for testing is because our model has seen those samples before, using same samples for making predictions might give us wrong impression about accuracy of our model. It is like you ask same questions in exam paper as you tought the students in the class.

In [10]:

```
X = df[['Mileage','Age(yrs)']]
```

In [11]:

```
y = df['Sell Price($)']
```

In [12]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

In [13]:

```
X_train
```

Out[13]:

	Mileage	Age(yrs)
6	52000	5
3	22500	2
7	72000	6
10	83000	7
19	52000	5
8	91000	8
9	67000	6
1	35000	3
5	59000	5
14	82450	7
12	59000	5
15	25400	3
0	69000	6
4	46000	4

In [14]:

```
X_test
```

Out[14]:

	Mileage	Age(yrs)
13	58780	4
11	79000	7
18	87600	8
17	69000	5
2	57000	5
16	28000	2

In [15]:

```
y_train
```

Out[15]:

```
6      32000
3      40000
7      19300
10     18700
19     28200
8      12000
9      22000
1      34000
5      26750
14     19400
12     26000
15     35000
0      18000
4      31500
Name: Sell Price($), dtype: int64
```

In [16]:

```
y_test
```

Out[16]:

```
13     27500
11     19500
18     12800
17     19700
2      26100
16     35500
Name: Sell Price($), dtype: int64
```

In [17]:

```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)
```

Out[17]:

```
LinearRegression()
```

In [18]:

```
X_test
```

Out[18]:

	Mileage	Age(yrs)
13	58780	4
11	79000	7
18	87600	8
17	69000	5
2	57000	5
16	28000	2

In [19]:

```
clf.predict(X_test)
```

Out[19]:

```
array([27388.53934622, 18047.5867002 , 14462.8701768 , 23393.53984483,
       26432.6730151 , 37997.25809737])
```

In [20]:

```
y_test
```

Out[20]:

```
13    27500
11    19500
18    12800
17    19700
2     26100
16    35500
Name: Sell Price($), dtype: int64
```

In [21]:

```
clf.score(X_test, y_test)
```

Out[21]:

```
0.9201886360171605
```

random_state argument

In [22]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=10)
X_test
```

Out[22]:

	Mileage	Age(yrs)
7	72000	6
10	83000	7
5	59000	5
6	52000	5
3	22500	2
18	87600	8

KNN (K Nearest Neighbors) Classification: Machine Tutorial Using Python Sklearn

In [1]:

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

In [2]:

```
iris.feature_names
```

Out[2]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [3]:

```
iris.target_names
```

Out[3]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [6]:

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Out[6]:

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

In [7]:

```
df['target'] = iris.target
df.head()
```

Out[7]:

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

In [8]:

```
df[df.target==1].head()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

In [9]:

```
df[df.target==2].head()
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

In [10]:

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Out[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [11]:

```
df[45:55]
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

In [13]:

```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

In [14]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

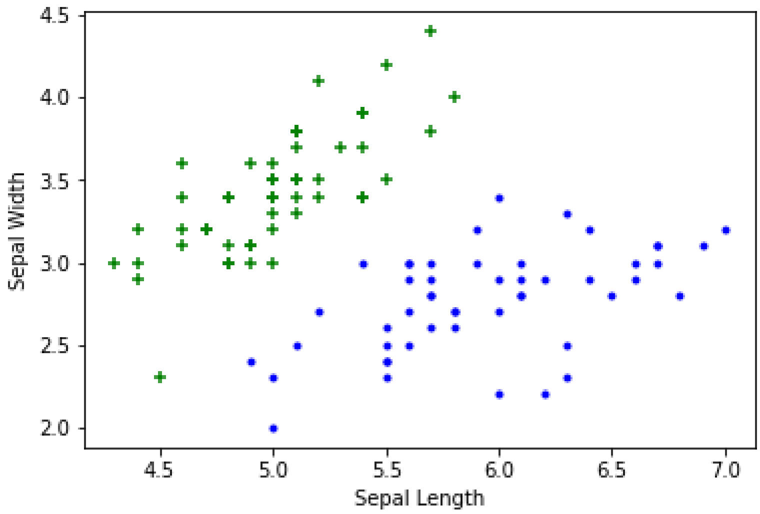
Sepal length vs Sepal Width (Setosa vs Versicolor)

In [15]:

```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+'
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

Out[15]:

```
<matplotlib.collections.PathCollection at 0x1f627711e80>
```



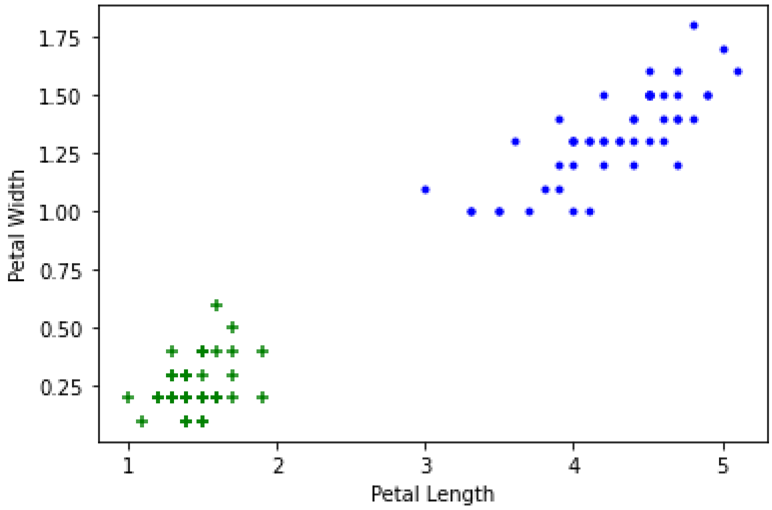
Petal length vs Pepal Width (Setosa vs Versicolor)

In [16]:

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

Out[16]:

```
<matplotlib.collections.PathCollection at 0x1f62780f6d0>
```



Train test split

In [17]:

```
from sklearn.model_selection import train_test_split
```

In [18]:

```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

In [19]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
```

In [20]:

```
len(X_train)
```

Out[20]:

```
120
```

In [21]:

```
len(X_test)
```

Out[21]:

```
30
```

Create KNN (K Neighrest Neighbour Classifier)

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
```

In [27]:

```
knn.fit(X_train, y_train)
```

Out[27]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [28]:

```
knn.score(X_test, y_test)
```

Out[28]:

```
1.0
```

In [30]:

```
knn.predict([[4.8,3.3,1.5,0.3]])
```

Out[30]:

```
array([0])
```

Plot Confusion Matrix

In [31]:

```
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[31]:

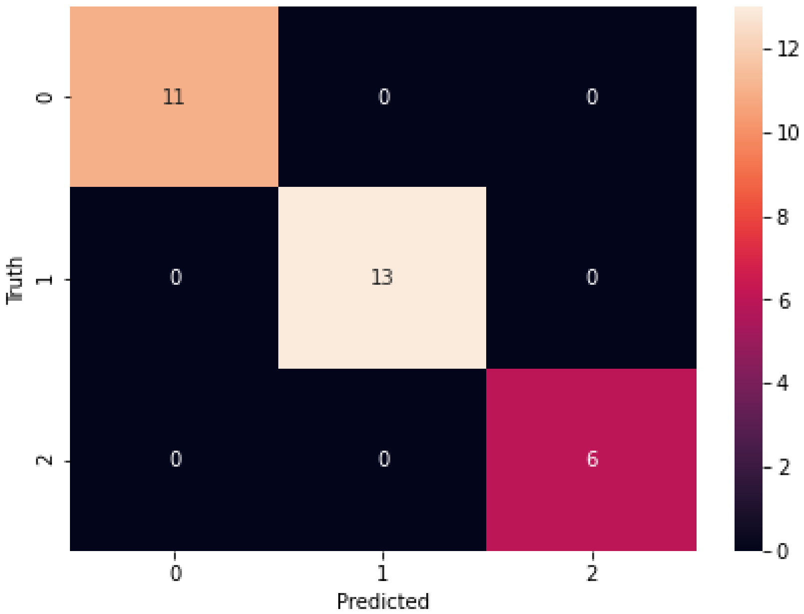
```
array([[11,  0,  0],
       [ 0, 13,  0],
       [ 0,  0,  6]], dtype=int64)
```

In [32]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[32]:

```
Text(42.0, 0.5, 'Truth')
```



Print classification report for precesion, recall and f1-score for each classes

In [33]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Decision Tree Classification

In [55]:

```
import pandas as pd
df = pd.read_csv("salaries.csv")
df.head()
```

Out[55]:

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0

In [56]:

```
inputs= df.drop('salary_more_then_100k',axis='columns')
```

In [57]:

```
target = df['salary_more_then_100k']
target
```

Out[57]:

```
0     0
1     0
2     1
3     1
4     0
5     1
6     0
7     0
8     0
9     1
10    1
11    1
12    1
13    1
14    1
15    1
Name: salary_more_then_100k, dtype: int64
```

In [58]:

```
from sklearn.preprocessing import LabelEncoder
company = LabelEncoder()
job = LabelEncoder()
degree = LabelEncoder()
```

In [59]:

```
inputs['company_n'] = company.fit_transform(inputs['company'])
inputs['job_n'] = job.fit_transform(inputs['job'])
inputs['degree_n'] = degree.fit_transform(inputs['degree'])
inputs
```

Out[59]:

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0

In [56]:

```
inputs_n = inputs.drop(['company','job','degree'],axis='columns')
inputs_n
```

Out[56]:

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

In [57]:

```
from sklearn import tree
model = tree.DecisionTreeClassifier()
```

In [58]:

```
model.fit(inputs_n, target)
```

Out[58]:

```
DecisionTreeClassifier()
```

In [59]:

```
model.score(inputs_n,target)
```

Out[59]:

```
1.0
```

Is salary of Google, Computer Engineer, Bachelors degree > 100 k ?

In [62]:

```
model.predict([[2,1,0]])
```

Out[62]:

```
array([0], dtype=int64)
```

Is salary of Google, Computer Engineer, Masters degree > 100 k ?

In [63]:

```
model.predict([[2,1,1]])
```

Out[63]:

```
array([1], dtype=int64)
```

Titanic exercise on decision trees

In [82]:

```
import pandas as pd
df = pd.read_csv("titanic.csv")
df.head()
```

Out[82]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [83]:

```
inputs= df.drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked','Surv
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
inputs.head()
```

Out[83]:

	Pclass	Sex	Age	Fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500

In [84]:

```
targets = df['Survived']
targets
```

Out[84]:

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

In [80]:

```
from sklearn.preprocessing import LabelEncoder
Sex = LabelEncoder()
```

In [114]:

```
inputs['Sex_n'] = Sex.fit_transform(inputs['Sex'])
inputs_n = inputs.drop('Sex',axis='columns')
inputs_n.head()
```

Out[114]:

	Pclass	Age	Fare	Sex_n
0	3	22.0	7.2500	1
1	1	38.0	71.2833	0
2	3	26.0	7.9250	0
3	1	35.0	53.1000	0
4	3	35.0	8.0500	1

In [120]:

```
from sklearn import tree
model = tree.DecisionTreeClassifier()
```

In [121]:

```
model.fit(inputs_n,targets)
```

Out[121]:

```
DecisionTreeClassifier()
```

In [122]:

```
model.score(inputs_n,targets)
```

Out[122]:

```
0.9797979797979798
```

In [126]:

```
model.predict([[1,33.0,70,0]])
```

Out[126]:

```
array([1], dtype=int64)
```

In [129]:

```
from sklearn.model_selection import train_test_split
```

In [131]:

```
X_train, X_test, y_train, y_test = train_test_split(inputs_n,targets,test_size=0.2)
```

In [132]:

```
model.fit(X_train,y_train)
```

Out[132]:

```
DecisionTreeClassifier()
```

In [133]:

```
model.score(X_test,y_test)
```

Out[133]:

```
0.7597765363128491
```

Predicting if a person would buy life insurnace based on his age using logistic regression#

Above is a binary logistic regression problem as there are only two possible outcomes (i.e. if person buys insurance or he/she doesn't).

In [2]:

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

In [3]:

```
df = pd.read_csv("insurance_data.csv")
df.head()
```

Out[3]:

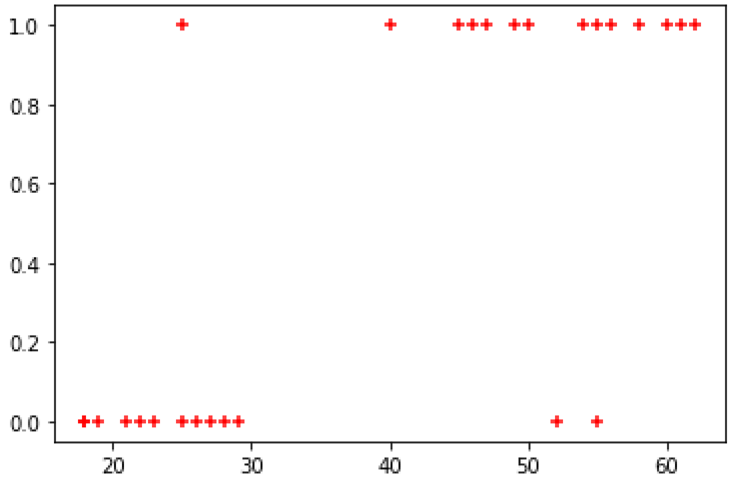
	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

In [4]:

```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

Out[4]:

<matplotlib.collections.PathCollection at 0x1b2b04671f0>



In [5]:

```
from sklearn.model_selection import train_test_split
```

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.bought_insurance,tra
```

In [12]:

```
X_test
```

Out[12]:

	age
23	45
19	18
10	18
26	23
11	28
22	40

In [13]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [14]:

```
model.fit(X_train, y_train)
```

Out[14]:

```
LogisticRegression()
```

In [15]:

```
X_test
```

Out[15]:

	age
23	45
19	18
10	18
26	23
11	28
22	40

In [16]:

```
model.predict(X_test)
```

Out[16]:

```
array([1, 0, 0, 0, 0, 1], dtype=int64)
```

In [17]:

```
model.predict_proba(X_test)
```

Out[17]:

```
array([[0.34548584, 0.65451416],
       [0.93039087, 0.06960913],
       [0.93039087, 0.06960913],
       [0.8801926 , 0.1198074 ],
       [0.80151679, 0.19848321],
       [0.48987945, 0.51012055]])
```

Exercise Download employee retention dataset from here: https://www.kaggle.com/giripujar/hr-analytics (https://www.kaggle.com/giripujar/hr-analytics).

Now do some exploratory data analysis to figure out which variables have direct and clear impact on employee retention (i.e. whether they leave the company or continue to work) Plot bar charts showing impact of employee salaries on retention Plot bar charts showing corelation between department and employee retention Now build logistic regression model using variables that were narrowed down in step 1 Measure the accuracy of the model

In [21]:

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

In [85]:

```
df = pd.read_csv("HR_comma_sep.csv")
df.head()
```

Out[85]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_compa
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

Data exploration and visualization

In [25]:

```
left = df[df.left==1]
left.shape
```

Out[25]:

```
(3571, 10)
```

In [40]:

```
retained = df[df.left==0]
retained
```

Out[40]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_co
2000	0.58	0.74	4	215	
2001	0.82	0.67	2	202	
2002	0.45	0.69	5	193	
2003	0.78	0.82	5	247	
2004	0.49	0.60	3	214	
...	...	...	...	...	
14206	0.90	0.55	3	259	
14207	0.74	0.95	5	266	
14208	0.85	0.54	3	185	
14209	0.33	0.65	3	172	
14210	0.50	0.73	4	180	

11428 rows × 10 columns

Average numbers for all columns

In [27]:

```
df.groupby('left').mean()
```

Out[27]:

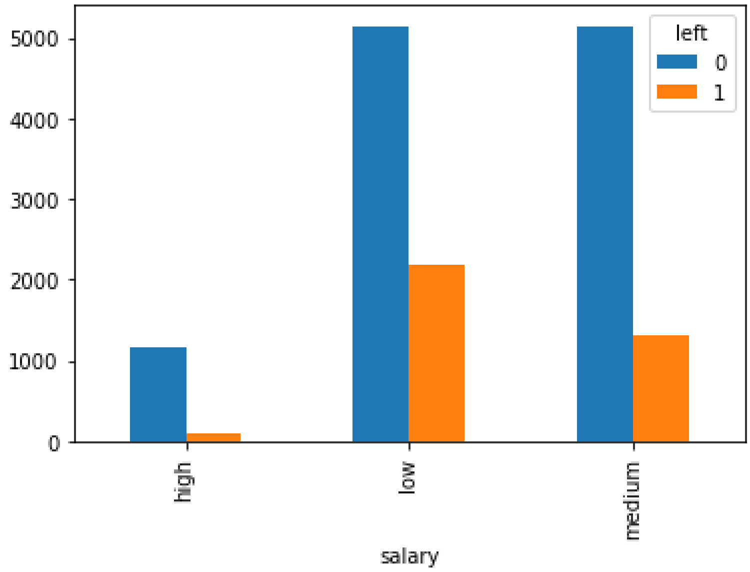
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_comp
left					
0	0.666810	0.715473	3.786664	199.060203	3.38
1	0.440098	0.718113	3.855503	207.419210	3.87

In [45]:

```
pd.crosstab(df.salary,df.left).plot(kind='bar')
```

Out[45]:

```
<AxesSubplot:xlabel='salary'>
```

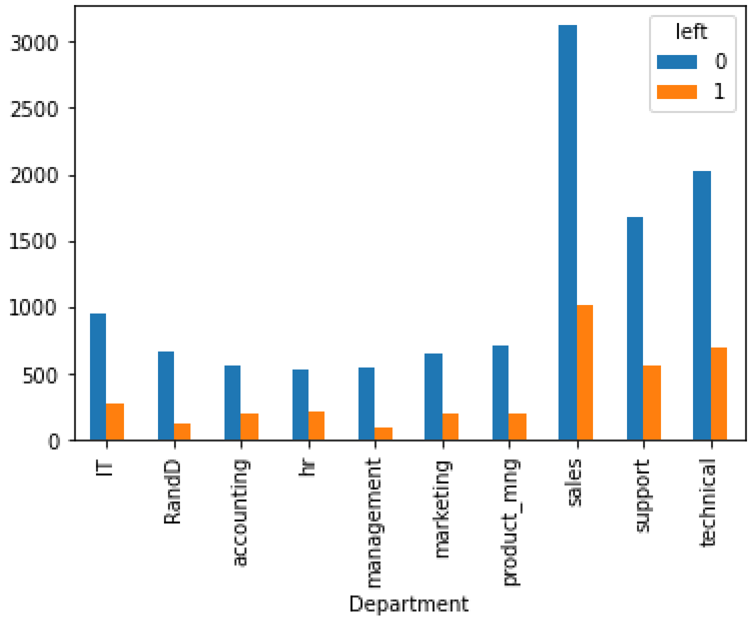


In [46]:

```
pd.crosstab(df.Department,df.left).plot(kind='bar')
```

Out[46]:

<AxesSubplot:xlabel='Department'>



In [47]:

```
subdf = df[['satisfaction_level','average_montly_hours','promotion_last_5years','salary
subdf.head()
```

Out[47]:

	satisfaction_level	average_montly_hours	promotion_last_5years	salary
0	0.38	157	0	low
1	0.80	262	0	medium
2	0.11	272	0	medium
3	0.72	223	0	low
4	0.37	159	0	low

In [48]:

```
salary_dummies = pd.get_dummies(subdf.salary, prefix="salary")
```

In [50]:

```
df_with_dummies = pd.concat([subdf,salary_dummies],axis='columns')
df_with_dummies.head()
```

Out[50]:

	satisfaction_level	average_montly_hours	promotion_last_5years	salary	salary_high	salary_
0	0.38	157	0	low	0	
1	0.80	262	0	medium	0	
2	0.11	272	0	medium	0	
3	0.72	223	0	low	0	
4	0.37	159	0	low	0	

In [68]:

```
from sklearn.preprocessing import LabelEncoder
salary = LabelEncoder()
```

In [115]:

```
subdf['salary_n'] = salary.fit_transform(subdf['salary'])
subdf.head(426)
```

```
<ipython-input-115-b314d2b54818>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  subdf['salary_n'] = salary.fit_transform(subdf['salary'])
```

Out[115]:

	satisfaction_level	average_montly_hours	promotion_last_5years	salary	salary_high	sala
0	0.38	157	0	low	1	
1	0.80	262	0	medium	2	
2	0.11	272	0	medium	2	
3	0.72	223	0	low	1	
4	0.37	159	0	low	1	
...	...	...	...	...	...	
421	0.10	287	0	medium	2	
422	0.86	257	0	medium	2	
423	0.40	143	0	high	0	
424	0.45	130	0	low	1	
425	0.42	136	0	medium	2	

426 rows × 6 columns

In [124]:

```
subdf1 = subdf.drop(['salary_high','salary'],axis='columns')
subdf1.head()
```

Out[124]:

	satisfaction_level	average_montly_hours	promotion_last_5years	salary_n
0	0.38	157	0	1
1	0.80	262	0	2
2	0.11	272	0	2
3	0.72	223	0	1
4	0.37	159	0	1

In [103]:

```
target = df['left']
target
```

Out[103]:

```
0        1
1        1
2        1
3        1
4        1
        ..
14994    1
14995    1
14996    1
14997    1
14998    1
Name: left, Length: 14999, dtype: int64
```

In [104]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(subdf1,target,train_size=0.3)
```

In [105]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [106]:

```
model.fit(subdf1,target)
```

Out[106]:

```
LogisticRegression()
```

In [123]:

```
model.predict([[0.01,100,0,0]])
```

Out[123]:

```
array([1], dtype=int64)
```

In [119]:

```
model.score(subdf1,target)
```

Out[119]:

```
0.7727848523234883
```

Support Vector Machine Tutorial Using Python Sklearn

In [125]:

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

In [126]:

```
iris.feature_names
```

Out[126]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [127]:

```
iris.target_names
```

Out[127]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [128]:

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Out[128]:

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

In [129]:

```
df['target'] = iris.target
df.head()
```

Out[129]:

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

In [141]:

```
df[df.target==0].head()
```

Out[141]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [142]:

```
df[df.target==1].head()
```

Out[142]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

In [143]:

```
df[df.target==2].head()
```

Out[143]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

In [132]:

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Out[132]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [133]:

```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

In [134]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

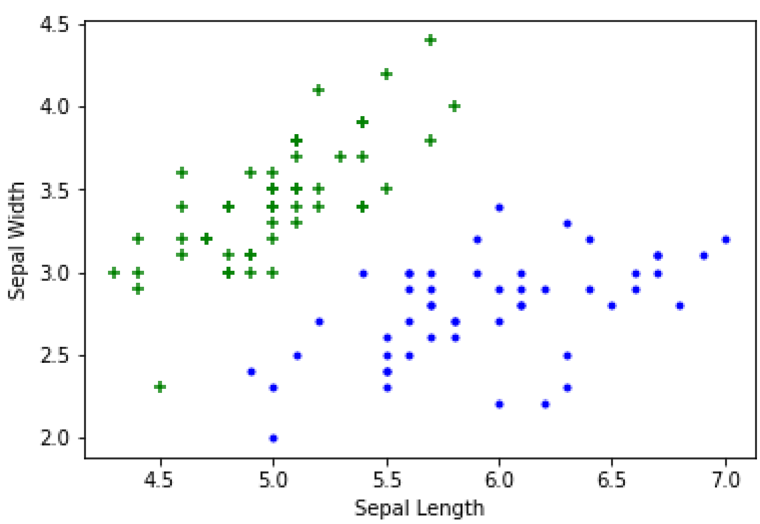
Sepal length vs Sepal Width (Setosa vs Versicolor)

In [135]:

```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+'
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

Out[135]:

```
<matplotlib.collections.PathCollection at 0x1b2b4550460>
```

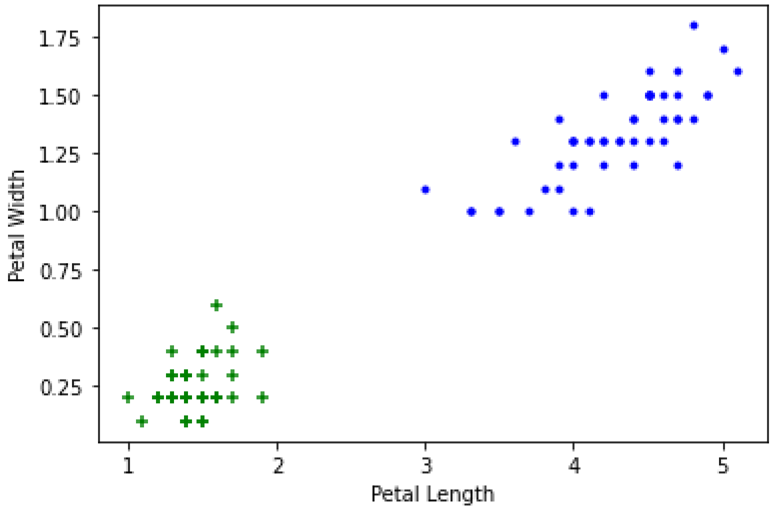


In [136]:

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

Out[136]:

```
<matplotlib.collections.PathCollection at 0x1b2b45a3ca0>
```



Train Using Support Vector Machine (SVM)

In [144]:

```
from sklearn.model_selection import train_test_split
```

In [145]:

```
target_1 = df['target']
target_1
```

Out[145]:

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: target, Length: 150, dtype: int32
```

In [147]:

```
df1 = df.drop(['target','flower_name'], axis='columns')
df1

```

Out[147]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [148]:

```
X_train, X_test, y_train, y_test = train_test_split(df1,target_1, test_size=0.2)
```

In [154]:

```
from sklearn.svm import SVC
model = SVC()
```

In [150]:

```
model.fit(df1,target_1)
```

Out[150]:

```
SVC()
```

In [151]:

```
model.score(df1,target_1)
```

Out[151]:

```
0.9733333333333334
```

In [153]:

```
model.predict([[3.1,4.5,5.8,3.5]])
```

Out[153]:

```
array([2])
```

Clustering With K Means - Python Tutorial

In [7]:

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

In [8]:

```
df = pd.read_csv("income.csv")
df.head()
```

Out[8]:

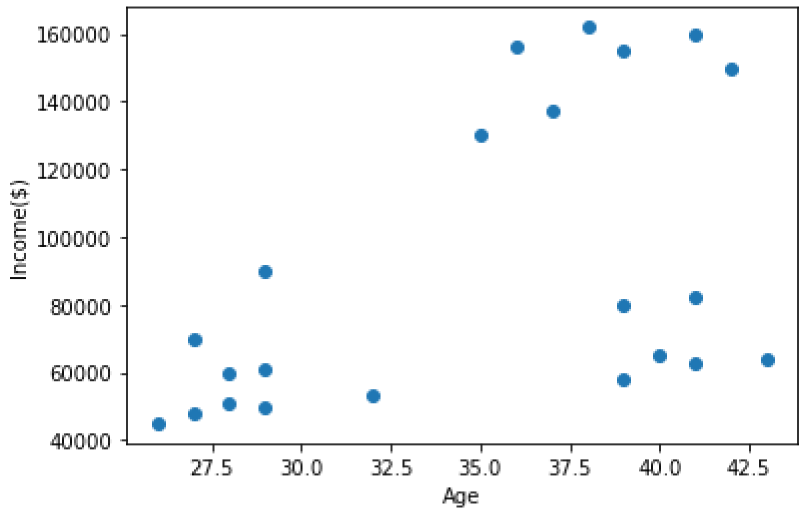
	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

In [9]:

```
plt.scatter(df['Age'],df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
```

Out[9]:

```
Text(0, 0.5, 'Income($)')
```



In [10]:

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3)
km
```

Out[10]:

```
KMeans(n_clusters=3)
```

In [11]:

```
km.fit(df[['Age','Income($)']])
```

Out[11]:

```
KMeans(n_clusters=3)
```

In [12]:

```
km.predict([[28,10000]])
```

Out[12]:

```
array([0])
```

In [13]:

```
target =  km.fit_predict(df[['Age','Income($)']])
target
```

Out[13]:

```
array([2, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0])
```

In [20]:

```
df['cluster_n'] = target
df.drop('cluster_n',axis='columns')
df
```

Out[20]:

	Name	Age	Income($)	cluster_n
0	Rob	0.058824	0.213675	2
1	Michael	0.176471	0.384615	2
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1
11	Tom	0.000000	0.000000	0
12	Arnold	0.058824	0.025641	0
13	Jared	0.117647	0.051282	0
14	Stark	0.176471	0.038462	0
15	Ranbir	0.352941	0.068376	0
16	Dipika	0.823529	0.170940	0
17	Priyanka	0.882353	0.153846	0
18	Nick	1.000000	0.162393	0
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	0

In [22]:

```
df.drop('cluster_n',axis='columns',inplace=True)
df
```

Out[22]:

	Name	Age	Income($)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436
5	Gautam	0.764706	0.940171
6	David	0.882353	0.982906
7	Andrea	0.705882	1.000000
8	Brad	0.588235	0.948718
9	Angelina	0.529412	0.726496
10	Donald	0.647059	0.786325
11	Tom	0.000000	0.000000
12	Arnold	0.058824	0.025641
13	Jared	0.117647	0.051282
14	Stark	0.176471	0.038462
15	Ranbir	0.352941	0.068376
16	Dipika	0.823529	0.170940
17	Priyanka	0.882353	0.153846
18	Nick	1.000000	0.162393
19	Alia	0.764706	0.299145
20	Sid	0.882353	0.316239
21	Abdul	0.764706	0.111111

Preprocessing using min max scaler

In [23]:

```
from sklearn.preprocessing import MinMaxScaler
Income = MinMaxScaler()
age = MinMaxScaler()
df['Income($)'] = Income.fit_transform(df[['Income($)']])
df['Age'] = age.fit_transform(df[['Age']])
df
```

Out[23]:

	Name	Age	Income($)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436
5	Gautam	0.764706	0.940171
6	David	0.882353	0.982906
7	Andrea	0.705882	1.000000
8	Brad	0.588235	0.948718
9	Angelina	0.529412	0.726496
10	Donald	0.647059	0.786325
11	Tom	0.000000	0.000000
12	Arnold	0.058824	0.025641
13	Jared	0.117647	0.051282
14	Stark	0.176471	0.038462
15	Ranbir	0.352941	0.068376
16	Dipika	0.823529	0.170940
17	Priyanka	0.882353	0.153846
18	Nick	1.000000	0.162393
19	Alia	0.764706	0.299145
20	Sid	0.882353	0.316239
21	Abdul	0.764706	0.111111

In [24]:

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3)
target =  km.fit_predict(df[['Age','Income($)']])
target
```

Out[24]:

```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2])
```

In [25]:

```
df['cluster'] = target
df
```

Out[25]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1
11	Tom	0.000000	0.000000	0
12	Arnold	0.058824	0.025641	0
13	Jared	0.117647	0.051282	0
14	Stark	0.176471	0.038462	0
15	Ranbir	0.352941	0.068376	0
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

In [28]:

```
km.cluster_centers_
```

Out[28]:

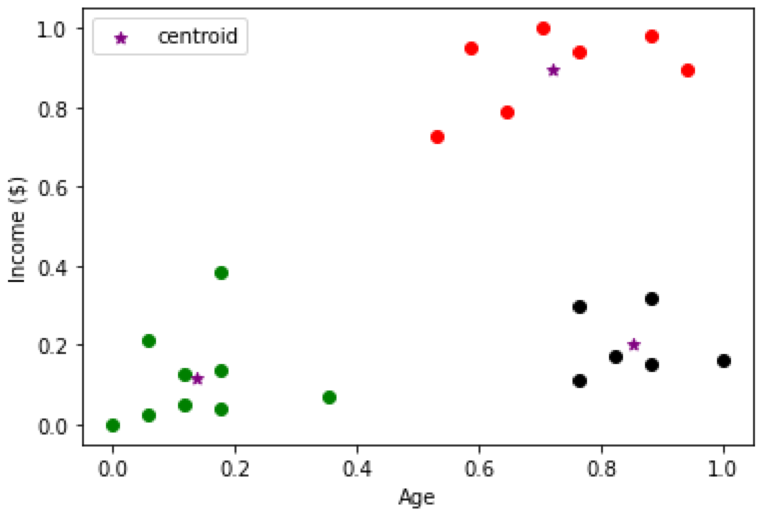
```
array([[0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ],
       [0.85294118, 0.2022792 ]])
```

In [27]:

```
df2 = df[df.cluster==0]
df3 = df[df.cluster==1]
df4 = df[df.cluster==2]
plt.scatter(df2['Age'],df2['Income($)'],color='green')
plt.scatter(df3['Age'],df3['Income($)'],color='red')
plt.scatter(df4['Age'],df4['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*'
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
```

Out[27]:

```
<matplotlib.legend.Legend at 0x1d7d53bbe50>
```



Elbow Plot

In [31]:

```
sse = []
k_range = range(1,10)
for k in k_range:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)

```

```
C:\Users\Subhayan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:88
1: UserWarning: KMeans is known to have a memory leak on Windows with MKL, w
hen there are less chunks than available threads. You can avoid it by settin
g the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [32]:

```
sse
```

Out[32]:

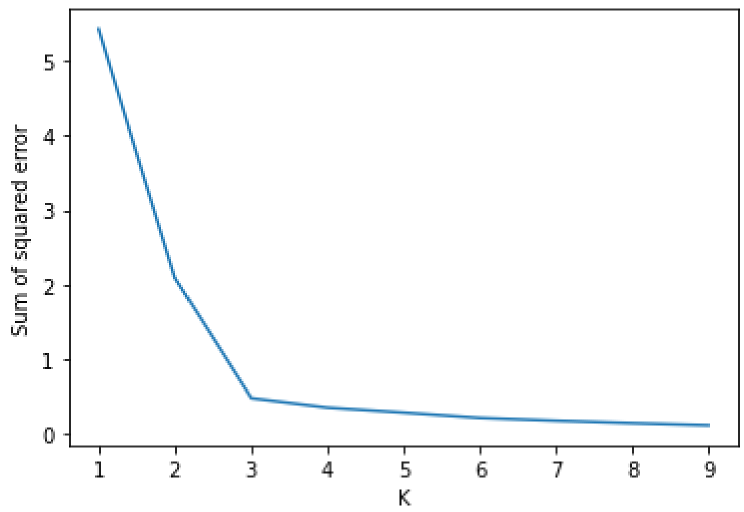
```
[5.434011511988176,
 2.0911363886990766,
 0.4750783498553095,
 0.3491047094419565,
 0.28184797443662374,
 0.2105547899547249,
 0.1729962193245546,
 0.13976844995388157,
 0.11123550695239098]
```

In [34]:

```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_range,sse)
```

Out[34]:

```
[<matplotlib.lines.Line2D at 0x1d7d54c56a0>]
```



Exercise K Means clustering

Use iris flower dataset from sklearn library and try to form clusters of flowers using petal width and length features. Drop other two features for simplicity. Figure out if any preprocessing such as scaling would help here Draw elbow plot and from that figure out optimal value of k

In [41]:

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
%matplotlib inline
```

In [45]:

```
iris = load_iris()
```

In [48]:

```
iris.target_names
```

Out[48]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [46]:

```
iris.feature_names
```

Out[46]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [47]:

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Out[47]:

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

In [61]:

```
df1 = df.drop(['sepal length (cm)', 'sepal width (cm)', 'flower'],axis='columns')
df1
```

Out[61]:

	petal length (cm)	petal width (cm)	cluster
0	1.4	0.2	0
1	1.4	0.2	0
2	1.3	0.2	0
3	1.5	0.2	0
4	1.4	0.2	0
...	...	...	...
145	5.2	2.3	2
146	5.0	1.9	2
147	5.2	2.0	2
148	5.4	2.3	2
149	5.1	1.8	2

150 rows × 3 columns

In [60]:

```
df1['cluster'] = iris.target
df1
```

Out[60]:

	petal length (cm)	petal width (cm)	cluster
0	1.4	0.2	0
1	1.4	0.2	0
2	1.3	0.2	0
3	1.5	0.2	0
4	1.4	0.2	0
...	...	...	...
145	5.2	2.3	2
146	5.0	1.9	2
147	5.2	2.0	2
148	5.4	2.3	2
149	5.1	1.8	2

150 rows × 3 columns

In [65]:

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3)
target = km.fit_predict(df1)
target
```

Out[65]:

```
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, 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, 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, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [67]:

```
#for reference purpose
df['flower_name'] =df.cluster.apply(lambda x: iris.target_names[x])
df.head()
```

Out[67]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower	cluster	flower_name
0	5.1	3.5	1.4	0.2	0	0	setosa
1	4.9	3.0	1.4	0.2	0	0	setosa
2	4.7	3.2	1.3	0.2	0	0	setosa
3	4.6	3.1	1.5	0.2	0	0	setosa
4	5.0	3.6	1.4	0.2	0	0	setosa

In [69]:

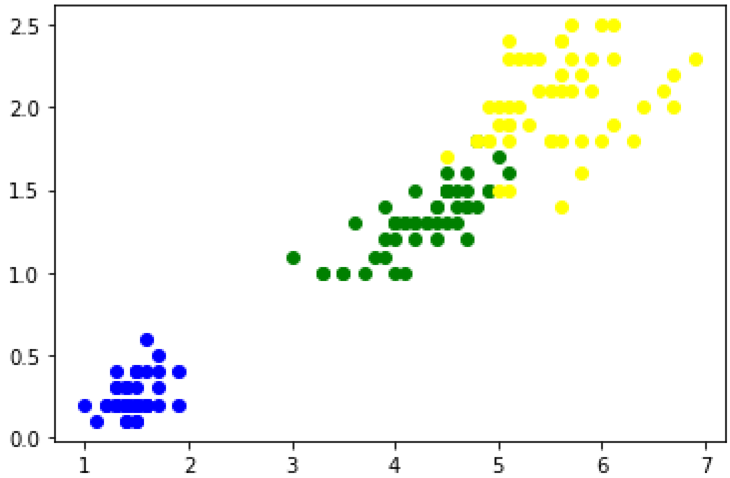
```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

In [70]:

```
plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color='blue')
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color='green')
plt.scatter(df2['petal length (cm)'],df2['petal width (cm)'],color='yellow')
```

Out[70]:

```
<matplotlib.collections.PathCollection at 0x1d7d568b2b0>
```



Elbow Plot

In [71]:

```
sse = []
k_range = range(1,10)
for k in k_range:
    km = KMeans(n_clusters=k)
    km.fit(df[['petal length (cm)','petal width (cm)']])
    sse.append(km.inertia_)

```

```
C:\Users\Subhayan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:88
1: UserWarning: KMeans is known to have a memory leak on Windows with MKL, w
hen there are less chunks than available threads. You can avoid it by settin
g the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [72]:

```
sse
```

Out[72]:

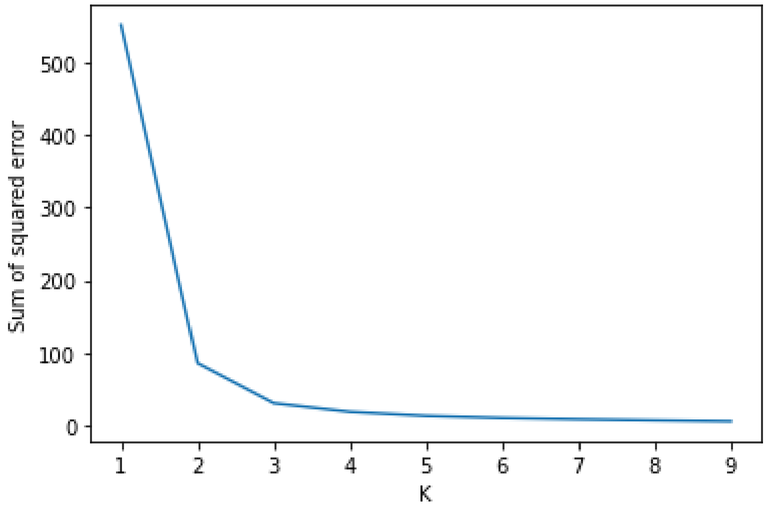
```
[550.8953333333333,
 86.39021984551391,
 31.371358974358966,
 19.477123363965468,
 13.91690875790876,
 11.040239971910458,
 9.236595959595961,
 7.783111506140917,
 6.456494541406302]
```

In [75]:

```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_range,sse)
```

Out[75]:

```
[<matplotlib.lines.Line2D at 0x1d7d5bd55b0>]
```



By this we can conclude that K=2 is the appropriate value

Recommender Systems with Python

Welcome to the code notebook for Recommender Systems with Python. In this lecture we will develop basic recommendation systems using Python and pandas.

In this notebook, we will focus on providing a basic recommendation system by suggesting items that are most similar to a particular item, in this case, movies. Keep in mind, this is not a true robust recommendation system, to describe it more accurately,it just tells you what movies/items are most similar to your movie choice.

There is no project for this topic, instead you have the option to work through the advanced lecture version of this notebook (totally optional!).

Let's get started!

Import Libraries

In [2]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('movies data.csv',names=column_names)
df.head()
```

Out[3]:

	user_id	item_id	rating	timestamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742

In [4]:

```
movie_titles = pd.read_excel("Movies titles.xlsx")
movie_titles.head()
```

Out[4]:

	item_id	title
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

In [5]:

```
df = pd.merge(df,movie_titles,on='item_id')
df.head()
```

Out[5]:

	user_id	item_id	rating	timestamp	title
0	0	50	5	881250949	Star Wars (1977)
1	290	50	5	880473582	Star Wars (1977)
2	79	50	4	891271545	Star Wars (1977)
3	2	50	5	888552084	Star Wars (1977)
4	8	50	5	879362124	Star Wars (1977)

EDA

Let's explore the data a bit and get a look at some of the best rated movies.

Visualization Imports

In [6]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Let's create a ratings dataframe with average rating and number of ratings:

In [7]:

```
df.groupby('rating')
df
```

Out[7]:

	user_id	item_id	rating	timestamp	title
0	0	50	5	881250949	Star Wars (1977)
1	290	50	5	880473582	Star Wars (1977)
2	79	50	4	891271545	Star Wars (1977)
3	2	50	5	888552084	Star Wars (1977)
4	8	50	5	879362124	Star Wars (1977)
...	...	...	...	...	...
99998	840	1674	4	891211682	Mamma Roma (1962)
99999	655	1640	3	888474646	Eighth Day, The (1996)
100000	655	1637	3	888984255	Girls Town (1996)
100001	655	1630	3	887428735	Silence of the Palace, The (Saimt el Qusur) (1...
100002	655	1641	3	887427810	Dadetown (1995)

100003 rows × 5 columns

In [8]:

```
df.groupby('title')['rating'].mean().sort_values(ascending=False).head()
```

Out[8]:

```
title
They Made Me a Criminal (1939)                5.0
Marlene Dietrich: Shadow and Light (1996)     5.0
Saint of Fort Washington, The (1993)          5.0
Someone Else's America (1995)                 5.0
Star Kid (1997)                               5.0
Name: rating, dtype: float64
```

In [21]:

```
df.groupby('title')['rating'].count().sort_values(ascending=False).head()
```

Out[21]:

```
title
Star Wars (1977)             584
Contact (1997)               509
Fargo (1996)                 508
Return of the Jedi (1983)    507
Liar Liar (1997)             485
Name: rating, dtype: int64
```

In [26]:

```
Avgratings = pd.DataFrame(df.groupby('title')['rating'].mean().sort_values(ascending=Fa
Avgratings
```

Out[26]:

	rating
title	
They Made Me a Criminal (1939)	5.0
Marlene Dietrich: Shadow and Light (1996)	5.0
Saint of Fort Washington, The (1993)	5.0
Someone Else's America (1995)	5.0
Star Kid (1997)	5.0
...	...
Eye of Vichy, The (Oeil de Vichy, L') (1993)	1.0
King of New York (1990)	1.0
Touki Bouki (Journey of the Hyena) (1973)	1.0
Bloody Child, The (1996)	1.0
Crude Oasis, The (1995)	1.0

1664 rows × 1 columns

Now set the number of ratings column:

In [28]:

```
Avgratings['num of ratings'] = pd.DataFrame(df.groupby('title')['rating'].count())
Avgratings
```

Out[28]:

title	rating	num of ratings
They Made Me a Criminal (1939)	5.0	1
Marlene Dietrich: Shadow and Light (1996)	5.0	1
Saint of Fort Washington, The (1993)	5.0	2
Someone Else's America (1995)	5.0	1
Star Kid (1997)	5.0	3
...	...	...
Eye of Vichy, The (Oeil de Vichy, L') (1993)	1.0	1
King of New York (1990)	1.0	1
Touki Bouki (Journey of the Hyena) (1973)	1.0	1
Bloody Child, The (1996)	1.0	1
Crude Oasis, The (1995)	1.0	1

1664 rows × 2 columns

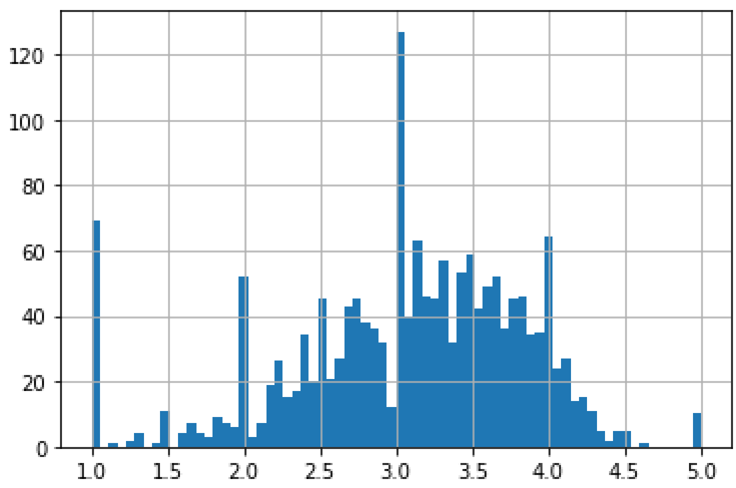
Now a few histograms:

In [33]:

```
import matplotlib.pyplot as plt
%matplotlib inline
Avgratings['rating'].hist(bins=70)
```

Out[33]:

<AxesSubplot:>

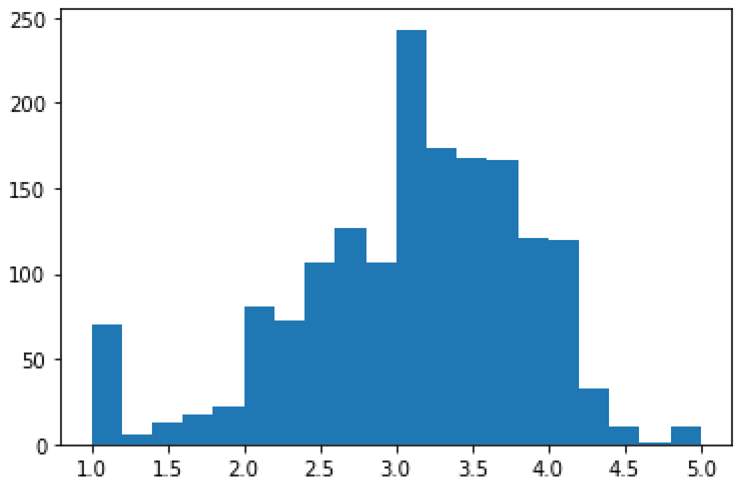


In [32]:

```
plt.hist(Avgratings['rating'],bins=20)
```

Out[32]:

```
(array([ 70.,   6.,  13.,  17.,  22.,  81.,  72., 106., 126., 107., 243.,
        174., 167., 166., 121., 119.,  33.,  10.,   1.,  10.]),
 array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. , 3.2, 3.4,
        3.6, 3.8, 4. , 4.2, 4.4, 4.6, 4.8, 5. ]),
 <BarContainer object of 20 artists>)
```

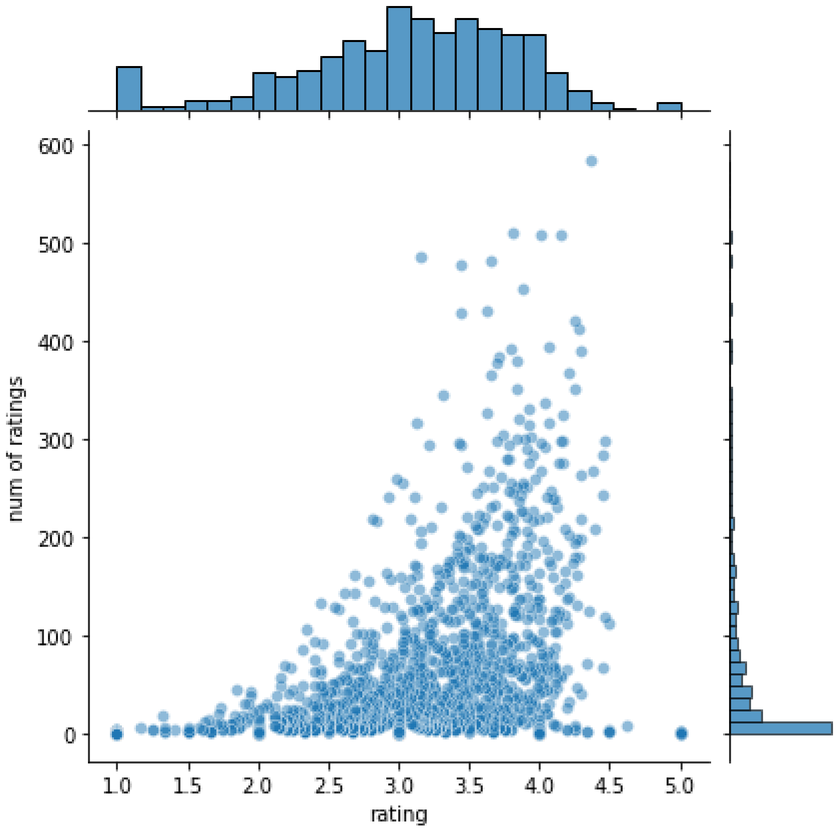


In [34]:

```
sns.jointplot(x='rating',y='num of ratings',data=Avgratings,alpha=0.5)
```

Out[34]:

```
<seaborn.axisgrid.JointGrid at 0x232160c4e20>
```



Okay! Now that we have a general idea of what the data looks like, let's move on to creating a simple recommendation system:

Recommending Similar Movies

Now let's create a matrix that has the user ids on one access and the movie title on another axis. Each cell will then consist of the rating the user gave to that movie. Note there will be a lot of NaN values, because most people have not seen most of the movies.

In [35]:

```
moviemat = df.pivot_table(index='user_id',columns='title',values='rating')
moviemat.head()
```

Out[35]:

12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Y Franken ('
NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
5.0	NaN	NaN	3.0	4.0	NaN	NaN	...	NaN	NaN	NaN	
NaN	NaN	NaN	NaN	NaN	1.0	NaN	...	NaN	NaN	NaN	
NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	

Most rated movie:

In [41]:

```
Avgratings.sort_values('num of ratings',ascending=False).head(10)
```

Out[41]:

title	rating	num of ratings
Star Wars (1977)	4.359589	584
Contact (1997)	3.803536	509
Fargo (1996)	4.155512	508
Return of the Jedi (1983)	4.007890	507
Liar Liar (1997)	3.156701	485
English Patient, The (1996)	3.656965	481
Scream (1996)	3.441423	478
Toy Story (1995)	3.878319	452
Air Force One (1997)	3.631090	431
Independence Day (ID4) (1996)	3.438228	429

Let's choose two movies: starwars, a sci-fi movie. And Liar Liar, a comedy.

Now let's grab the user ratings for those two movies:

In [44]:

```
starwars_user_ratings = moviemat['Star Wars (1977)']
starwars_user_ratings.head()
```

Out[44]:

```
user_id
0    5.0
1    5.0
2    5.0
3    NaN
4    5.0
Name: Star Wars (1977), dtype: float64
```

We can then use corrwith() method to get correlations between two pandas series:

In [46]:

```
similar_to_starwars = moviemat.corrwith(starwars_user_ratings)
similar_to_starwars.head()
```

Out[46]:

```
title
'Til There Was You (1997)    0.872872
1-900 (1994)                -0.645497
101 Dalmatians (1996)        0.211132
12 Angry Men (1957)          0.184289
187 (1997)                   0.027398
dtype: float64
```

In [49]:

```
corr_starwars = pd.DataFrame(similar_to_starwars,columns=['Correlation'])
corr_starwars.dropna(inplace=True)
corr_starwars.head()
```

Out[49]:

title	Correlation
'Til There Was You (1997)	0.872872
1-900 (1994)	-0.645497
101 Dalmatians (1996)	0.211132
12 Angry Men (1957)	0.184289
187 (1997)	0.027398

Now if we sort the dataframe by correlation, we should get the most similar movies, however note that we get some results that don't really make sense. This is because there are a lot of movies only watched once by users who also watched star wars (it was the most popular movie).

In [50]:

```
corr_starwars.sort_values('Correlation',ascending=False).head(10)
```

Out[50]:

title	Correlation
Hollow Reed (1996)	1.0
Commandments (1997)	1.0
Cosi (1996)	1.0
No Escape (1994)	1.0
Stripes (1981)	1.0
Star Wars (1977)	1.0
Man of the Year (1995)	1.0
Beans of Egypt, Maine, The (1994)	1.0
Old Lady Who Walked in the Sea, The (Vieille qui marchait dans la mer, La) (1991)	1.0
Outlaw, The (1943)	1.0

Let's fix this by filtering out movies that have less than 100 reviews (this value was chosen based off the histogram from earlier).

In [51]:

```
corr_starwars = corr_starwars.join(Avgratings['num of ratings'])
corr_starwars.head()
```

Out[51]:

title	Correlation	num of ratings
'Til There Was You (1997)	0.872872	9
1-900 (1994)	-0.645497	5
101 Dalmatians (1996)	0.211132	109
12 Angry Men (1957)	0.184289	125
187 (1997)	0.027398	41

Now sort the values and notice how the titles make a lot more sense:

In [58]:

```
corr_starwars[corr_starwars['num of ratings']>100].sort_values('Correlation',ascending=
```

Out[58]:

title	Correlation	num of ratings
Star Wars (1977)	1.000000	584
Empire Strikes Back, The (1980)	0.748353	368
Return of the Jedi (1983)	0.672556	507
Raiders of the Lost Ark (1981)	0.536117	420
Austin Powers: International Man of Mystery (1997)	0.377433	130
...	...	...
Edge, The (1997)	-0.127167	113
As Good As It Gets (1997)	-0.130466	112
Crash (1996)	-0.148507	128
G.I. Jane (1997)	-0.176734	175
First Wives Club, The (1996)	-0.194496	160

334 rows × 2 columns

Now the same for the comedy Liar Liar:

In [57]:

```
liarliar_user_ratings = moviemat['Liar Liar (1997)']
similar_to_liarliar = moviemat.corrwith(liarliar_user_ratings)
corr_liarliar = pd.DataFrame(similar_to_liarliar,columns=['Correlation'])
corr_liarliar.dropna(inplace=True)
corr_liarliar = corr_liarliar.join(Avgratings['num of ratings'])
corr_liarliar[corr_liarliar['num of ratings']>100].sort_values('Correlation',ascending=
```

```
C:\Users\Subhayan\anaconda3\lib\site-packages\numpy\lib\function_base.py:263
4: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\Subhayan\anaconda3\lib\site-packages\numpy\lib\function_base.py:249
3: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
```

Out[57]:

title	Correlation	num of ratings
Liar Liar (1997)	1.000000	485
Batman Forever (1995)	0.516968	114
Mask, The (1994)	0.484650	129
Down Periscope (1996)	0.472681	101
Con Air (1997)	0.469828	137

In [ ]:

```

```