Ass 1. Installation of Python on Windows, Installing Packages, Loading data.

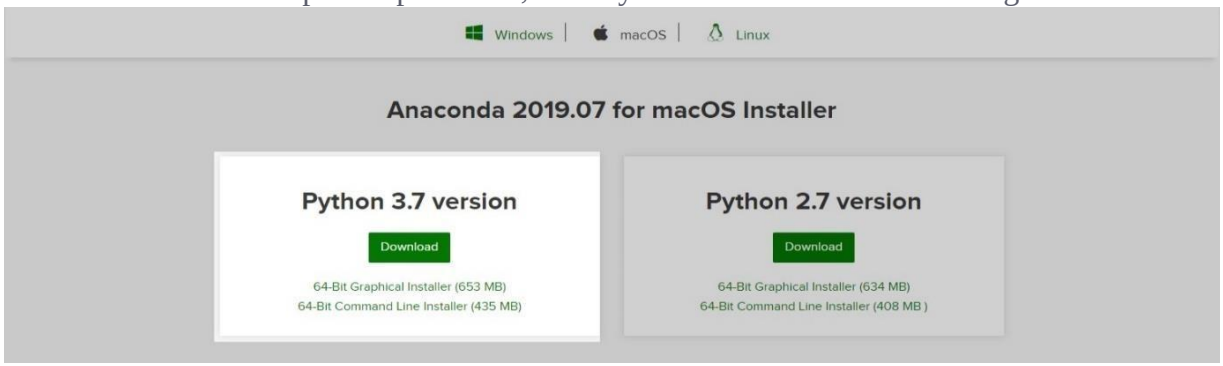
Download and Install Anaconda on Windows

Step #1: Go To Anaconda.com

Go to Anaconda.com, and download the Anaconda version for Windows.

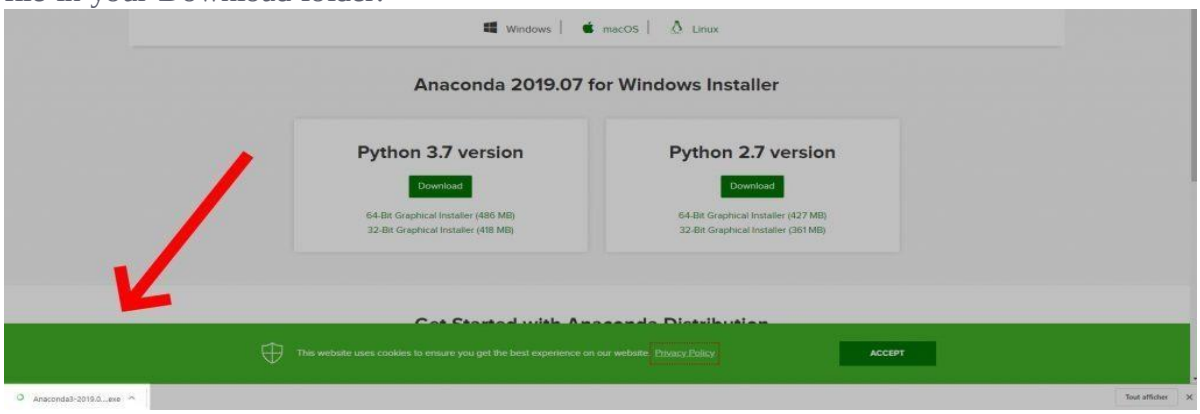
Step #2: Download the Python 3 version for Windows.

Version 2 will not be updated past 2020, so do yourself a favor and start using V3.



Step #3: Double-click on the executable file.

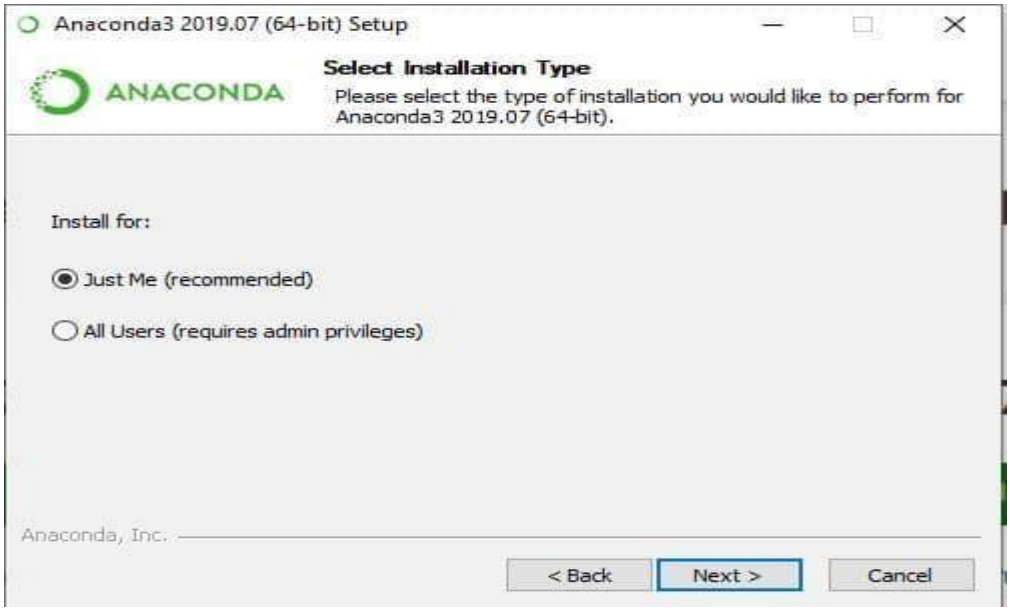
To get the installation of Anaconda started on your operating system open the executable file in your Download folder.



Step #4: Click Next

Step #5: Click I agree to the terms and conditions

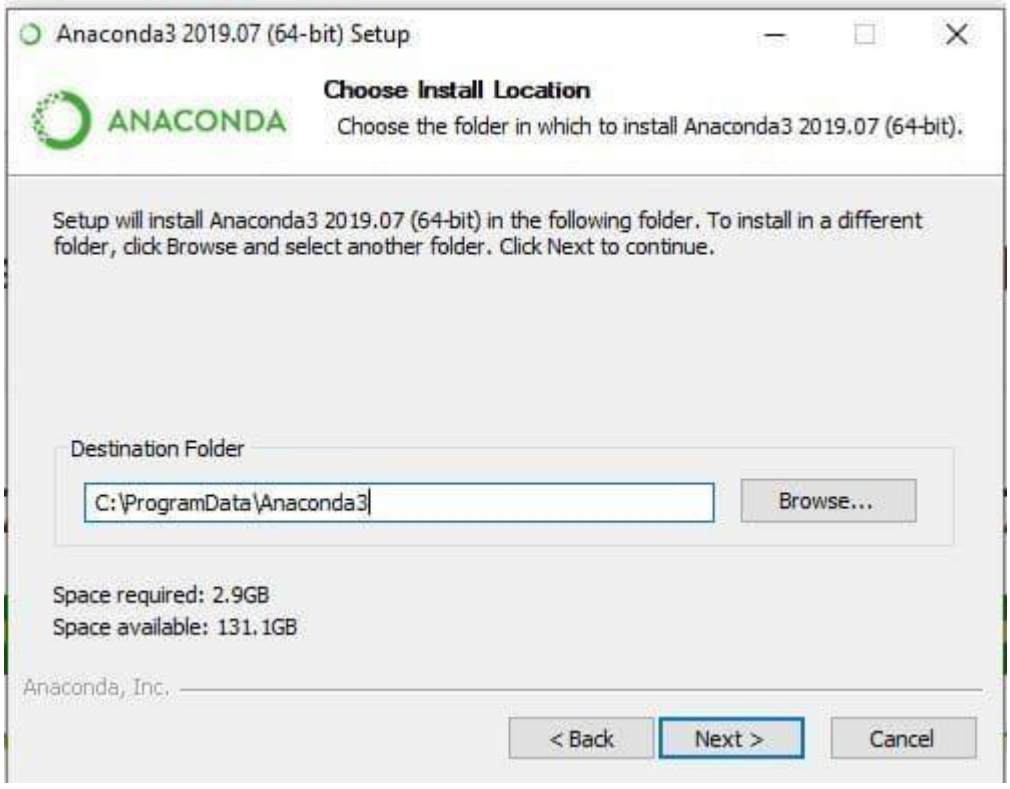
Step #6: Select Who You Want To Give Anaconda To



This step will ask you if you want to install Anaconda just for you or for all the users using this PC. Click “Just-Me”, or “All users”, depending on your preference. Both options will do but to select “all users” you will need admin privileges.

Step #7: Select the installation location

If you have selected “All users”, by default, Anaconda will get installed in the *C:\ProgramData\Anaconda3* folder. So make sure that you have at least the right amount of space available to install the subdirectory comparing it the the space required.

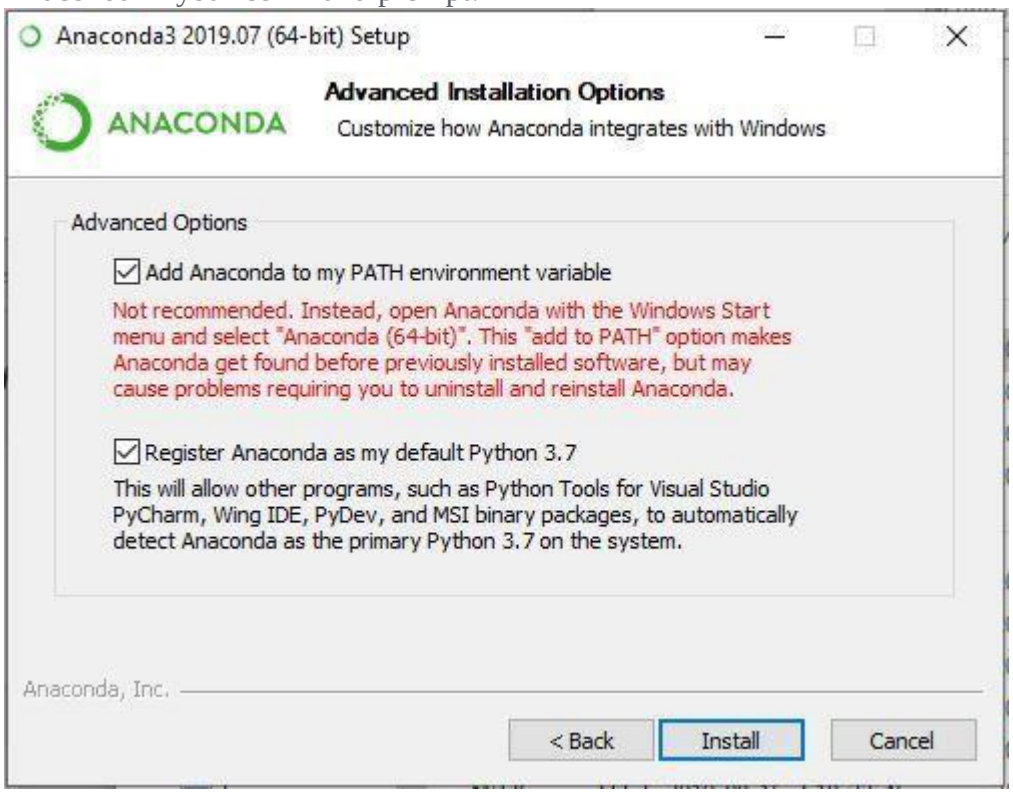


Step #8: Select the environment variables

Depending on if you have any version of Python already installed on your operating system, or not, to do different set-up.

If You Are Installing Python For The First Time

Check the *Add Anaconda to my PATH environment variable.* This will let you use Anaconda in your command prompt.

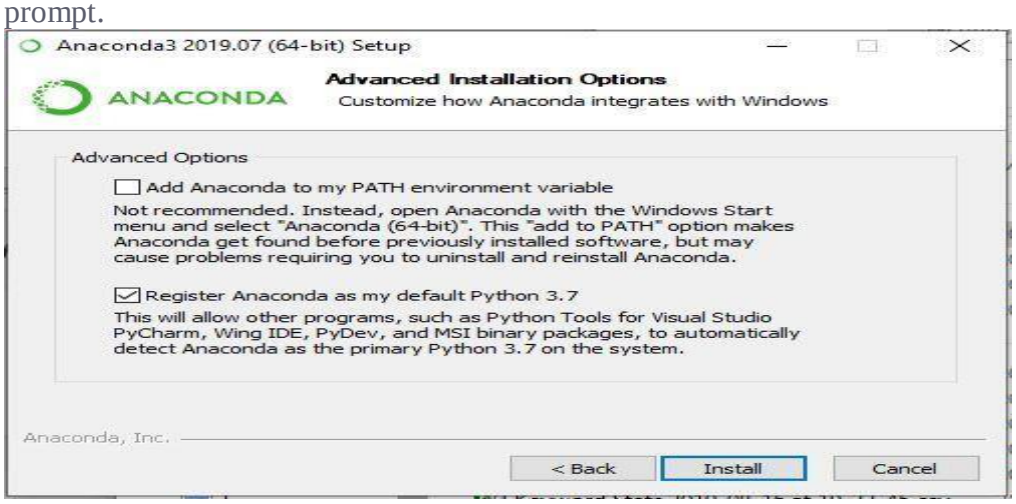


If You Already Have Python Installed

Leave *Add Anaconda to my PATH environment variable* unchecked.

Leaving it unchecked means that you will have to use Anaconda Command Prompt in order to use Anaconda.

So, unless you add the PATH later, you will not be able to use Python from your command prompt.

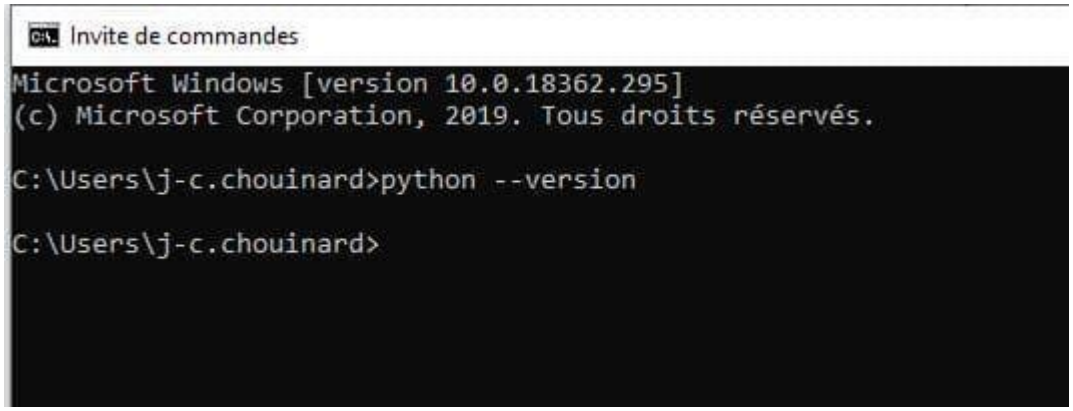


Python is not usually included by default on Windows, however we can check if any version exists on the system.

To know if you have Python Installed.

1. Go to Start Menu and type “Command Prompt” to open it.
2. Type the following command and hit the Enter key “python --version”
3. If nothing happens, you don’t have Python installed. Otherwise, you will get this result.

$ python --version
Python 3.7.0

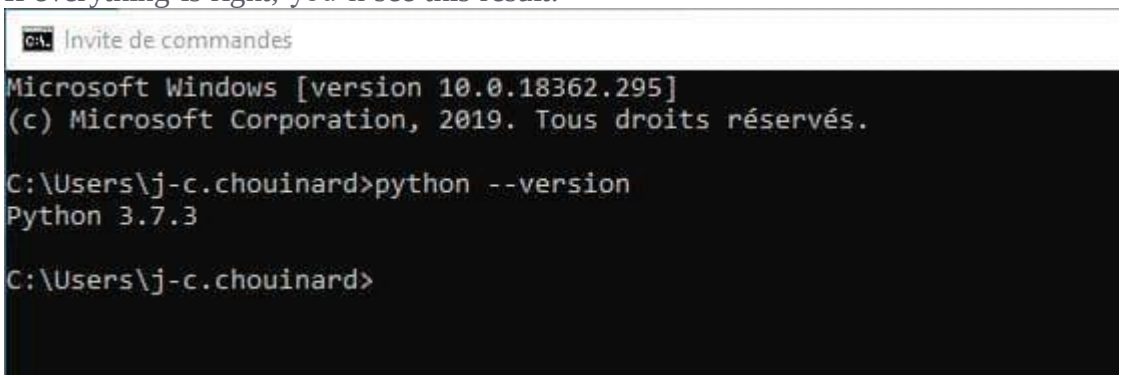


Step #9: Click Next and then “Finish”.

Step #10: See if Python Is Installed

If everything went right you can repeat the step 7 by opening your command prompt and enter “python --version”.

If everything is right, you’ll see this result.



Installing Packages:

Add packages to Anaconda environment in Python

Let's see some methods that can be used to install packages to Anaconda environment. There are many ways one can add pre-built packages to anaconda environment. So, let's see how to direct the path in anaconda and install them.

Using *pip* command :

1. Open Anaconda Command prompt as administrator
2. Use **cd** to come out of set directory or path.
3. Run **pip install** command.

E.g

pip install numpy

pip install scikit-learn

Loading data.

pandas is a powerful data analysis package. It makes data exploration and manipulation easy. It has several functions to read data from various sources.

```
import pandas as pd
mydata=pd.read_csv("C:\\Users\\Deepanshu\\Documents\\file1.csv")
```

Ass 2 Data Preparation using techniques like Data Cleansing

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

```
data = pd.read_csv('feedback.csv')
print(data)
```

OUTPUT:

	Rating	Review Title	Review	Customer Name	Date	Review ID
0	4	Works well	The product works fine, it is maybe a little e...	Phillip	October 10, 2021	#7653
1	3	good enough	NaN	elena	October 5, 2021	NaN
2	5	Everyone should buy this	You should buy this.	Olivia	NaN	NaN
3	5	Amazing product	Love everything about this product, it works g...	John	5th October	NaN
4	1	this is terrible	The product never worked for me.	Paula	44,491.00	#8563
5	2	Doesn't work	This doesn't work as advertised.	Ellie	NaN	NaN
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE	September 15, 2021	#4162
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo	NaN	NaN
8	5	Amazing product	Love everything about this product, it works g...	John	10/5/2021	#5675
9	5	Love this!!	I would 100% recommend this to everone.	CARA	NaN	NaN
10	100	Hate this	This doesn't do anything for me.	Helen	September 15, 2021	NaN
11	3	OK product	It does what it has to do, but the user experi...	emma	NaN	#7553

```
print(data.isnull()
```

OUTPUT:

	Rating	Review Title	Review	Customer Name	Date	Review ID
0	False	False	False	False	False	False
1	False	False	True	False	False	True
2	False	False	False	False	True	True
3	False	False	False	False	False	True
4	False	False	False	False	False	False
5	False	False	False	False	True	True
6	False	False	False	False	False	False
7	False	False	False	False	True	True
8	False	False	False	False	False	False
9	False	False	False	False	True	True
10	False	False	False	False	False	True
11	False	False	False	False	True	False

```
remove = ['Review ID','Date']
print(data.drop(remove, inplace =True, axis =1))
```

OUTPUT:

	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	NaN	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

```
print(data.isnull().sum())
```

OUTPUT:

```
Rating             0
Review Title       0
Review             1
Customer Name      0
Date               5
Review ID          7
dtype: int64
```

```
remove = ['Review ID','Date']
print(data.drop(remove, inplace =True, axis =1))
```

OUTPUT:

	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	NaN	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

```
print(data['Review'] = data['Review'].fillna('No review'))
```

OUTPUT:

	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

print(data.duplicated())

OUTPUT:

```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8       True
9      False
10     False
11     False
dtype: bool
```

print(data.drop_duplicates())

OUTPUT:

	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

Print(data['Rating'].describe())

OUTPUT:

```
count      12.000000
mean       11.833333
std        27.797427
min         1.000000
25%         3.000000
50%         4.500000
75%         5.000000
max       100.000000
Name: Rating, dtype: float64
```

```
Print(data.loc[10,'Rating'] = 1)
```

OUTPUT:

	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	1	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

```
print(data['Review Title'] = data['Review Title'].str.lower())
```

OUTPUT:

	Rating	Review Title	Review	Customer Name
0	4	works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	everyone should buy this	You should buy this.	Olivia
3	5	amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	best thing ever	Go and buy this right now, it's amazing.	Pablo
8	5	amazing product	Love everything about this product, it works g...	John
9	5	love this!!	I would 100% recommend this to everone.	CARA
10	1	hate this	This doesn't do anything for me.	Helen
11	3	ok product	It does what it has to do, but the user experi...	emma

Ass 3 Data Aggregation:

Data aggregation is any process whereby data is gathered and expressed in a summary form.

Data Frame

```
import pandas as pd
data={'corporation':['YAHOO','YAHOO','MSFT','MSFT','GOOGLE','GOOGLE'],
    'person':['Sanjay','Chetan','Smiti','Anjali','Shaliendra','Jagrati'],
    'sales_in_USD':[100,140,540,670,240,551]}
df=pd.DataFrame(data)
print(df)
```

output

	corporation	person	sales_in_USD
0	YAHOO	Sanjay	100
1	YAHOO	Chetan	140
2	MSFT	Smiti	540
3	MSFT	Anjali	670
4	GOOGLE	Shaliendra	240
5	GOOGLE	Jagrati	551

```
print(df.groupby('corporation'))
```

output

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001E9324FC9A0>
```

```
print(type(df.groupby('corporation')))
```

Output

```
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

```
group_data=df.groupby('corporation')
```

Aggregation function:

1) Sum() :

```
print(group_data.sum())
```

output

corporation	sales_in_USD
GOOGLE	791
MSFT	1210
YAHOO	240

2) mean():

```
print(group_data.mean())
```

output

corporation	
GOOGLE	395.5
MSFT	605.0
YAHOO	120.0

3) std():

```
print(group_data.std())
```

output

```
  sales_in_USD
corporation
GOOGLE        219.910209
MSFT          91.923882
YAHOO         28.284271
```

4) min():

```
print(group_data.min())
```

output

```
person  sales_in_USD
corporation
GOOGLE      Jagrati         240
MSFT        Anjali          540
YAHOO       Chetan          100
```

5) max():

```
print(group_data.max())
```

output

```
  person  sales_in_USD
corporation
GOOGLE      Shaliendra          551
MSFT          Smiti         670
YAHOO         Sanjay          140
```

6) count():

```
print(group_data.count())
```

output

```
person  sales_in_USD
corporation
GOOGLE          2           2
MSFT            2           2
YAHOO           2           2
```

7) describe() :

```
print(group_data.describe())
```

output

```
sales_in_USD                ...
               count   mean        std ...    50%    75%    max
corporation                        ...
GOOGLE           2.0  395.5  219.910209  ...  395.5  473.25  551.0
MSFT             2.0  605.0  91.923882  ...  605.0  637.50  670.0
YAHOO            2.0  120.0  28.284271  ...  120.0  130.00  140.0
```

```
print(group_data.describe().transpose())
```

output

```
corporation            GOOGLE        MSFT       YAHOO
sales_in_USD count    2.000000    2.000000    2.000000
          mean  395.500000  605.000000  120.000000
          std    219.910209  91.923882   28.284271
          min    240.000000  540.000000  100.000000
          25%    317.750000  572.500000  110.000000
          50%    395.500000  605.000000  120.000000
          75%    473.250000  637.500000  130.000000
          max    551.000000  670.000000  140.000000
```

```
print(group_data.describe().transpose()['GOOGLE'])
```

output

```
sales_in_USD  count      2.000000
          mean     395.500000
          std      219.910209
          min      240.000000
          25%      317.750000
          50%      395.500000
          75%      473.250000
          max      551.000000
```

Ass 4 Handling missing values, Feature Scaling, Inconsistent values in the given dataset.

Handling missing values :

In [34]:
```
import pandas as pd
```

In [35]:
```
import numpy as np
```

In [40]:
```
df=pd.DataFrame({
    "Date": pd.date_range(start="2021-10-01",periods=10,freq="D"),
    "Item": 1014,
    "Measure_1": np.random.randint(1,10,size=10),
    "Measure_2": np.random.random(10).round(2),
    "Measure_3": np.random.random(10).round(2),
    "Measure_4": np.random.random(10)
})
```

In [41]:
```
df
```

Out[41]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014	5	0.36	0.03	0.076459
1	2021-10-02	1014	7	0.77	0.02	0.364348
2	2021-10-03	1014	9	0.50	0.09	0.224930
3	2021-10-04	1014	7	0.83	0.84	0.632682
4	2021-10-05	1014	7	0.20	0.92	0.145471
5	2021-10-06	1014	5	0.86	0.25	0.048626
6	2021-10-07	1014	9	0.90	0.60	0.709231
7	2021-10-08	1014	5	0.54	0.59	0.373793
8	2021-10-09	1014	2	0.36	0.89	0.682035
9	2021-10-10	1014	4	0.68	0.65	0.762551

In [46]:
```
df.loc[[2,9],"Item"]=np.nan
df.loc[[2,7,9],"Measure_1"]=np.nan
df.loc[[2,3],"Measure_2"]=np.nan
df.loc[[2],"Measure_3"]=np.nan
df.loc[:6,"Measure_4"]=np.nan
```

In [47]:
```
df
```

Out[47]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014.0	5.0	0.36	0.03	NaN
1	2021-10-02	1014.0	7.0	0.77	0.02	NaN
2	2021-10-03	NaN	NaN	NaN	NaN	NaN
3	2021-10-04	1014.0	7.0	NaN	0.84	NaN
4	2021-10-05	1014.0	7.0	0.20	0.92	NaN
5	2021-10-06	1014.0	5.0	0.86	0.25	NaN
6	2021-10-07	1014.0	9.0	0.90	0.60	NaN
7	2021-10-08	1014.0	NaN	0.54	0.59	0.373793
8	2021-10-09	1014.0	2.0	0.36	0.89	0.682035
9	2021-10-10	NaN	NaN	0.68	0.65	0.762551

```
In [48]: df=df.astype({
             "Item":pd.Int64Dtype(),
             "Measure_1":pd.Int64Dtype()})

In [49]: df
```

Out[49]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014	5	0.36	0.03	NaN
1	2021-10-02	1014	7	0.77	0.02	NaN
2	2021-10-03	<NA>	<NA>	NaN	NaN	NaN
3	2021-10-04	1014	7	NaN	0.84	NaN
4	2021-10-05	1014	7	0.20	0.92	NaN
5	2021-10-06	1014	5	0.86	0.25	NaN
6	2021-10-07	1014	9	0.90	0.60	NaN
7	2021-10-08	1014	<NA>	0.54	0.59	0.373793
8	2021-10-09	1014	2	0.36	0.89	0.682035
9	2021-10-10	<NA>	<NA>	0.68	0.65	0.762551

1. Drop rows or columns that have a missing value

```
In [50]: df.dropna()
```

Out[50]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
8	2021-10-09	1014	2	0.36	0.89	0.682035

```
In [51]: df.dropna(axis=1)
```

Out[51]:

	Date
0	2021-10-01
1	2021-10-02
2	2021-10-03
3	2021-10-04
4	2021-10-05
5	2021-10-06
6	2021-10-07
7	2021-10-08
8	2021-10-09
9	2021-10-10

```
In [53]: df.dropna(how='all')
```

Out[53]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014	5	0.36	0.03	NaN
1	2021-10-02	1014	7	0.77	0.02	NaN
2	2021-10-03	<NA>	<NA>	NaN	NaN	NaN
3	2021-10-04	1014	7	NaN	0.84	NaN
4	2021-10-05	1014	7	0.20	0.92	NaN
5	2021-10-06	1014	5	0.86	0.25	NaN
6	2021-10-07	1014	9	0.90	0.60	NaN
7	2021-10-08	1014	<NA>	0.54	0.59	0.373793
8	2021-10-09	1014	2	0.36	0.89	0.682035
9	2021-10-10	<NA>	<NA>	0.68	0.65	0.762551

2. Drop rows or columns based on a threshold value

```
In [54]: df.dropna(thresh=4)
```

Out[54]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014	5	0.36	0.03	NaN
1	2021-10-02	1014	7	0.77	0.02	NaN
3	2021-10-04	1014	7	NaN	0.84	NaN
4	2021-10-05	1014	7	0.20	0.92	NaN
5	2021-10-06	1014	5	0.86	0.25	NaN
6	2021-10-07	1014	9	0.90	0.60	NaN
7	2021-10-08	1014	<NA>	0.54	0.59	0.373793
8	2021-10-09	1014	2	0.36	0.89	0.682035
9	2021-10-10	<NA>	<NA>	0.68	0.65	0.762551

3) Drop based on a particular subset of columns:

```
In [57]: df.dropna(subset=["Measure_2","Measure_3"])
```

Out[57]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014	5	0.36	0.03	NaN
1	2021-10-02	1014	7	0.77	0.02	NaN
4	2021-10-05	1014	7	0.20	0.92	NaN
5	2021-10-06	1014	5	0.86	0.25	NaN
6	2021-10-07	1014	9	0.90	0.60	NaN
7	2021-10-08	1014	<NA>	0.54	0.59	0.373793
8	2021-10-09	1014	2	0.36	0.89	0.682035
9	2021-10-10	<NA>	<NA>	0.68	0.65	0.762551

4) Fill with a constant value :

```
In [58]: 1 values={"Item":1014,"Measure_1":0}
         2 df.fillna(value=values)
```

Out[58]:

	Date	Item	Measure_1	Measure_2	Measure_3	Measure_4
0	2021-10-01	1014	5	0.36	0.03	NaN
1	2021-10-02	1014	7	0.77	0.02	NaN
2	2021-10-03	1014	0	NaN	NaN	NaN
3	2021-10-04	1014	7	NaN	0.84	NaN
4	2021-10-05	1014	7	0.20	0.92	NaN
5	2021-10-06	1014	5	0.86	0.25	NaN
6	2021-10-07	1014	9	0.90	0.60	NaN
7	2021-10-08	1014	0	0.54	0.59	0.373793
8	2021-10-09	1014	2	0.36	0.89	0.682035
9	2021-10-10	1014	0	0.68	0.65	0.762551

5. Fill with an aggregated value:

```
df["Measure_2"].fillna(df["Measure_2"].mean())
```

Handling Missing Values

In [5]:
```
import pandas as pd
data = pd.read_csv('abc.csv')
data
```

Out[5]:

	iteams	price
0	A	70.0
1	B	NaN
2	C	50.0
3	D	NaN
4	E	40.0
5	F	32.0
6	G	45.0
7	H	69.0
8	I	NaN
9	J	NaN

In [6]:
```
data['price'] = data['price'].fillna(data['price'].mean())
data
```

Out[6]:

	iteams	price
0	A	70.0
1	B	51.0
2	C	50.0
3	D	51.0
4	E	40.0
5	F	32.0
6	G	45.0
7	H	69.0
8	I	51.0
9	J	51.0

In [18]:
```
data['price'] = data['price'].fillna(data['price'].median())
data
```

Out[18]:

	iteams	price
0	A	70.0
1	B	47.5
2	C	50.0
3	D	47.5
4	E	40.0
5	F	32.0
6	G	45.0
7	H	69.0
8	I	47.5
9	J	47.5

In [16]:
```
data['price'] = data['price'].fillna(data['price'].std())
data
```

Out[16]:

	iteams	price
0	A	70.000000
1	B	15.517732
2	C	50.000000
3	D	15.517732
4	E	40.000000
5	F	32.000000
6	G	45.000000
7	H	69.000000
8	I	15.517732
9	J	15.517732

In [20]:
```
data['price'] = data['price'].fillna(data['price'].max())
data
```

Out[20]:

	iteams	price
0	A	70.0
1	B	70.0
2	C	50.0
3	D	70.0
4	E	40.0
5	F	32.0
6	G	45.0
7	H	69.0
8	I	70.0
9	J	70.0

In [13]:
```
data['price'] = data['price'].fillna(data['price'].min())
data
```

Out[13]:

	iteams	price
0	A	70.0
1	B	32.0
2	C	50.0
3	D	32.0
4	E	40.0
5	F	32.0
6	G	45.0
7	H	69.0
8	I	32.0
9	J	32.0

Feature Scaling

In [1]:
```
from pandas import read_csv
from numpy import set_printoptions
from sklearn import preprocessing
data = r'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'
names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
a = read_csv(data, names=names)
a
```

Out[1]:

	A	B	C	D	E	F	G	H	I
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [2]:
```
scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
rescaled = scaler.fit_transform(a)
set_printoptions(precision=2)
rescaled
```

Out[2]:
```
array([[0.35, 0.74, 0.59, ..., 0.23, 0.48, 1.  ],
       [0.06, 0.43, 0.54, ..., 0.12, 0.17, 0.  ],
       [0.47, 0.92, 0.52, ..., 0.25, 0.18, 1.  ],
       ...,
       [0.29, 0.61, 0.59, ..., 0.07, 0.15, 0.  ],
       [0.06, 0.63, 0.49, ..., 0.12, 0.43, 1.  ],
       [0.06, 0.47, 0.57, ..., 0.1 , 0.03, 0.  ]])
```

In [3]:
```
from sklearn.preprocessing import StandardScaler
data_scaler = StandardScaler().fit(a)
data_rescaled = data_scaler.transform(a)
data_rescaled
```

Out[3]:
```
array([[ 0.64,  0.85,  0.15, ...,  0.47,  1.43,  1.37],
       [-0.84, -1.12, -0.16, ..., -0.37, -0.19, -0.73],
       [ 1.23,  1.94, -0.26, ...,  0.6 , -0.11,  1.37],
       ...,
       [ 0.34,  0.  ,  0.15, ..., -0.69, -0.28, -0.73],
       [-0.84,  0.16, -0.47, ..., -0.37,  1.17,  1.37],
       [-0.84, -0.87,  0.05, ..., -0.47, -0.87, -0.73]])
```

In [1]:
Ass 5 Feature selection using techniques like univariate selection correlation heatmaps, Wrapper-based ,methods, Filter-based methods.

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
data = pd.read_csv("train.csv")
X = data.iloc[:,0:20]
y = data.iloc[:,-1]
```

In [2]:
```
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
```

In [3]:
```
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
```

In [4]:
```
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
```

In [5]:
```
featureScores
```

Out[5]:

	Specs	Score
0	battery_power	14129.866576
1	blue	0.723232
2	clock_speed	0.648366
3	dual_sim	0.631011
4	fc	10.135166
5	four_g	1.521572
6	int_memory	89.839124
7	m_dep	0.745820
8	mobile_wt	95.972863
9	n_cores	9.097556
10	pc	9.186054
11	px_height	17363.569536
12	px_width	9810.586750
13	ram	931267.519053
14	sc_h	9.614878
15	sc_w	16.480319
16	talk_time	13.236400
17	three_g	0.327643
18	touch_screen	1.928429
19	wifi	0.422091

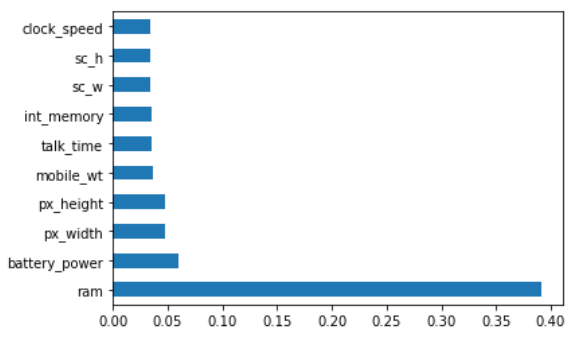
In [6]:
```
print(featureScores.nlargest(10,'Score'))
```

```
            Specs          Score
13            ram  931267.519053
11      px_height   17363.569536
0   battery_power   14129.866576
12       px_width    9810.586750
8       mobile_wt      95.972863
6      int_memory      89.839124
15           sc_w      16.480319
16      talk_time      13.236400
4              fc      10.135166
14           sc_h       9.614878
```

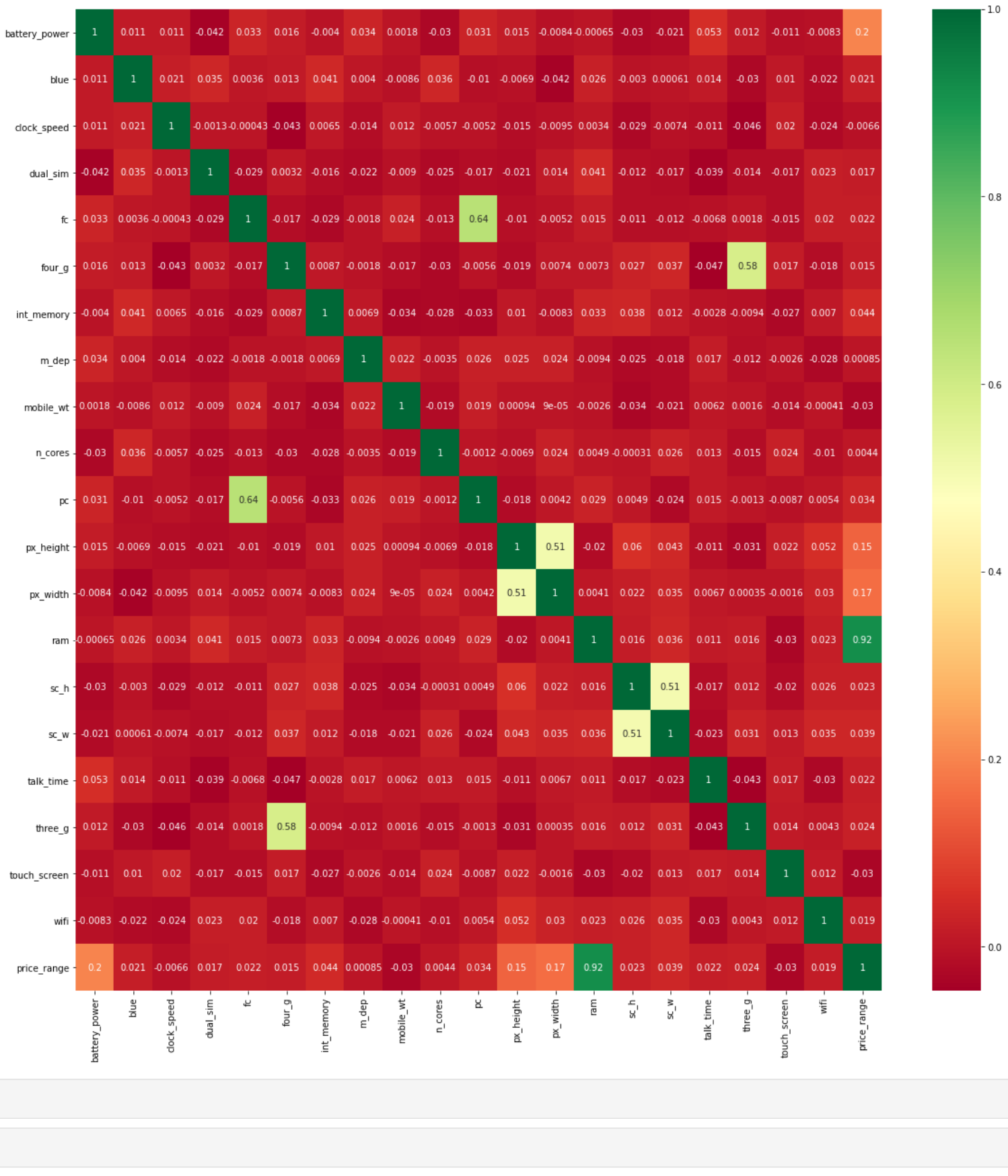
In [8]:
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
```

Out[8]:
```
ExtraTreesClassifier()
```

In [9]:
```
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



In [10]:
```
import seaborn as sns
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [ ]:

In [ ]:

Ass 6 Feature engineering using techniques like Outlier management, One-hot encoding, Log transform..

In [1]:
```
import pandas as pd
df = pd.read_csv("team.csv")
df
```

Out[1]:

	TEAM	YEAR
0	A	2000
1	B	2002
2	C	2003
3	D	2004
4	A	2005
5	C	2006
6	B	2007
7	A	2008
8	D	2009

In [2]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
dfle = df
dfle.TEAM = le.fit_transform(dfle.TEAM)
dfle
```

Out[2]:

	TEAM	YEAR
0	0	2000
1	1	2002
2	2	2003
3	3	2004
4	0	2005
5	2	2006
6	1	2007
7	0	2008
8	3	2009

In [3]:
```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import pandas as pd
# creating one hot encoder object
enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(dfle[['TEAM']]).toarray())
enc_df
```

Out[3]:

	0	1	2	3
0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0
2	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0
4	1.0	0.0	0.0	0.0
5	0.0	0.0	1.0	0.0
6	0.0	1.0	0.0	0.0
7	1.0	0.0	0.0	0.0
8	0.0	0.0	0.0	1.0

In [4]:
```
abc = dfle.join(enc_df)
abc
```

Out[4]:

	TEAM	YEAR	0	1	2	3
0	0	2000	1.0	0.0	0.0	0.0
1	1	2002	0.0	1.0	0.0	0.0

2	2	2003	0.0	0.0	1.0	0.0
3	3	2004	0.0	0.0	0.0	1.0
4	0	2005	1.0	0.0	0.0	0.0
5	2	2006	0.0	0.0	1.0	0.0
6	1	2007	0.0	1.0	0.0	0.0
7	0	2008	1.0	0.0	0.0	0.0
8	3	2009	0.0	0.0	0.0	1.0

In [5]:

```
final = abc.drop(['TEAM'], axis='columns')
final
```

Out[5]:

	YEAR	0	1	2	3
0	2000	1.0	0.0	0.0	0.0
1	2002	0.0	1.0	0.0	0.0
2	2003	0.0	0.0	1.0	0.0
3	2004	0.0	0.0	0.0	1.0
4	2005	1.0	0.0	0.0	0.0
5	2006	0.0	0.0	1.0	0.0
6	2007	0.0	1.0	0.0	0.0
7	2008	1.0	0.0	0.0	0.0
8	2009	0.0	0.0	0.0	1.0

In [ ]:

Ass 7 Implement Logistic regression classifier.

In [2]:
```
import pandas as pd
df = pd.read_csv("abcde.csv")
df.head(10)
```

Out[2]:

	age	results
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.results,train_size=0.8,random_state=10)
```

In [4]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[4]:
```
LogisticRegression()
```

In [5]:
```
y_predicted = model.predict(X_test)
y_predicted
```

Out[5]:
```
array([1, 1, 0, 0, 0, 0], dtype=int64)
```

In [6]:
```
model.score(X_test,y_test)
```

Out[6]:
```
1.0
```

In [1]:
Ass 8 Implement Naïve Bayes classifier.

```
# import libraries
import numpy as np
import pandas as pd
```

In [2]:
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

In [3]:
```
data.data
```

Out[3]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [4]:
```
data.target
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

In [5]:
```
data.target_names
```

Out[5]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [6]:
```
df = pd.DataFrame(np.c_[data.data, data.target], columns=[list(data.feature_names)+['target']])
df.head()
```

Out[6]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	smo
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	

5 rows × 31 columns

In [7]:
```
df.tail()
```

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	s
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	

5 rows × 31 columns

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

Out[8]: (569, 31)

In [9]:
```
"""### Split Data"""

X = df.iloc[:, 0:-1]
y = df.iloc[:, -1]
```

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2020)
```

In [11]:
```
print('Shape of X_train = ', X_train.shape)
print('Shape of y_train = ', y_train.shape)
print('Shape of X_test = ', X_test.shape)
print('Shape of y_test = ', y_test.shape)
```

```
Shape of X_train =  (455, 30)
Shape of y_train =  (455,)
Shape of X_test =  (114, 30)
Shape of y_test =  (114,)
```

In [13]:
```
"""## Train Naive Bayes Classifier Model : GaussianNB"""

from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
```

Out[13]: 0.9736842105263158

In [14]:
```
"""## Train Naive Bayes Classifier Model : MultinomialNB"""

from sklearn.naive_bayes import MultinomialNB
classifier_m = MultinomialNB()
classifier_m.fit(X_train, y_train)

classifier_m.score(X_test, y_test)
```

Out[14]: 0.8947368421052632

In [15]:
```
"""## Train Naive Bayes Classifier Model : BernoulliNB"""
from sklearn.naive_bayes import BernoulliNB
classifier_b = BernoulliNB()
classifier_b.fit(X_train, y_train)
```

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

Out[15]: 0.5789473684210527

In [16]:
```
"""## Predict Cancer"""

patient1 = [17.99,
 10.38,
 122.8,
 1001.0,
 0.1184,
 0.2776,
 0.3001,
 0.1471,
 0.2419,
 0.07871,
 1.095,
 0.9053,
 8.589,
 153.4,
 0.006399,
 0.04904,
 0.05373,
 0.01587,
 0.03003,
 0.006193,
 25.38,
 17.33,
 184.6,
 2019.0,
 0.1622,
 0.6656,
 0.7119,
 0.2654,
 0.4601,
 0.1189]
```

In [17]:
```
patient1 = np.array([patient1])    #convert 2d data
patient1
```

Out[17]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01]])
```

In [18]:
```
classifier.predict(patient1)   #patiendt dectect VALUE 0 means predict cancer
```

Out[18]:
```
array([0.])
```

In [19]:
```
data.target_names
```

Out[19]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [20]:
```
pred = classifier.predict(patient1)
```

In [21]:
```
if pred[0] == 0:
  print('Patient has Cancer (malignant tumor)')
else:
  print('Patient has no Cancer (malignant benign)')
```

```
Patient has Cancer (malignant tumor)
```

In [1]:
```
# Ass 9 Use confusion matrixes to describe performance of a classifier.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import RobustScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
```

In [4]:
```
df = pd.read_csv('churn modelling.csv', index_col=0)
df.head()
```

Out[4]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Es
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	
4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	

In [5]:
```
df.drop(['CustomerId', 'Surname'], axis=1, inplace=True)
```

In [6]:
```
df.shape
```

Out[6]:
```
(10000, 11)
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [8]:
```
X = df.drop('Exited', 1)
y = df.Exited
y.value_counts()
```

Out[8]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)
X.columns
```

Out[9]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary'],
      dtype='object')
```

In [10]:
```
num_cols = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
cat_cols = ['HasCrCard', 'IsActiveMember', 'Geography', 'Gender']
```

In [11]:
```
ct = ColumnTransformer([
    ('s1', RobustScaler(), num_cols),
    ('s2', OneHotEncoder(sparse=False, handle_unknown='ignore'), cat_cols)
])
```

In [12]:
```
p = Pipeline([
    ('ct', ct),
    ('mod', LogisticRegression(random_state=0))
])
```

In [13]:
```
p.fit(X_train, y_train)
```

Out[13]:
```
Pipeline(steps=[('ct',
                 ColumnTransformer(transformers=[('s1', RobustScaler(),
                                                  ['CreditScore', 'Age',
                                                   'Tenure', 'Balance',
                                                   'NumOfProducts',
                                                   'EstimatedSalary']),
                                                 ('s2',
                                                  OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['HasCrCard',
                                                   'IsActiveMember',
                                                   'Geography', 'Gender'])])),
                ('mod', LogisticRegression(random_state=0))])
```

In [14]:
```
preds = p.predict(X_test)
preds[:15]
```

Out[14]:
```
array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [15]:
```
np.array(y_test)[:15]
```

Out[15]:
```
array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [16]:
```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
confusion_matrix(y_true=y_test, y_pred=preds)
```

Out[16]:
```
array([[1530,   63],
       [ 319,   88]], dtype=int64)
```

In [17]:
```
p.classes_
```

Out[17]:
```
array([0, 1], dtype=int64)
```

In [18]:
```
confusion_matrix(y_test, preds, labels=(1,0))
```

Out[18]:
```
array([[  88,  319],
       [  63, 1530]], dtype=int64)
```

In [19]:
```
confusion_matrix(y_test, preds, labels=(1,0)).ravel()
```

Out[19]:
```
array([  88,  319,   63, 1530], dtype=int64)
```

In [38]:
```
accuracy_score(y_test, preds)
```

Out[38]:
```
0.809
```

In [39]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,\
fbeta_score, matthews_corrcoef
precision_score(y_test, preds)
```

Out[39]: 0.5827814569536424

In [40]:
```
tp, fn, fp, tn = confusion_matrix(y_test, preds, labels=(1,0)).ravel()
precision = tp/(tp+fp)
precision
```

Out[40]: 0.5827814569536424

In [41]:
```
recall_score(y_test, preds)
```

Out[41]: 0.21621621621621623

In [42]:
```
# harmonic mean of precision and recall
f1_score(y_test, preds)
```

Out[42]: 0.31541218637992835

In [2]:

Ass 10 Implement classifier using Support Vector Machines.

```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn import metrics

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [3]:

```
data_set
```

Out[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [4]:

```
x_test
```

Out[4]:

```
array([[-0.80480212,  0.50496393],
       [-0.01254409, -0.5677824 ],
       [-0.30964085,  0.1570462 ],
       [-0.80480212,  0.27301877],
       [-0.30964085, -0.5677824 ],
       [-1.10189888, -1.43757673],
       [-0.70576986, -1.58254245],
       [-0.21060859,  2.15757314],
       [-1.99318916, -0.04590581],
       [ 0.8787462 , -0.77073441],
       [-0.80480212, -0.59677555],
       [-1.00286662, -0.42281668],
       [-0.11157634, -0.42281668],
       [ 0.08648817,  0.21503249],
       [-1.79512465,  0.47597078],
       [-0.60673761,  1.37475825],
       [-0.11157634,  0.21503249],
       [-1.89415691,  0.44697764],
       [ 1.67100423,  1.75166912],
       [-0.30964085, -1.37959044],
       [-0.30964085, -0.65476184],
       [ 0.8787462 ,  2.15757314],
       [ 0.28455268, -0.53878926],
       [ 0.8787462 ,  1.02684052],
       [-1.49802789, -1.20563157],
       [ 1.07681071,  2.07059371],
       [-1.00286662,  0.50496393],
       [-0.90383437,  0.30201192],
       [-0.11157634, -0.21986468],
       [-0.60673761,  0.47597078],
```

```
       [-1.6960924 ,  0.53395707],
       [-0.11157634,  0.27301877],
       [ 1.86906873, -0.27785096],
       [-0.11157634, -0.48080297],
       [-1.39899564, -0.33583725],
       [-1.99318916, -0.50979612],
       [-1.59706014,  0.33100506],
       [-0.4086731 , -0.77073441],
       [-0.70576986, -1.03167271],
       [ 1.07681071, -0.97368642],
       [-1.10189888,  0.53395707],
       [ 0.28455268, -0.50979612],
       [-1.10189888,  0.41798449],
       [-0.30964085, -1.43757673],
       [ 0.48261718,  1.22979253],
       [-1.10189888, -0.33583725],
       [-0.11157634,  0.30201192],
       [ 1.37390747,  0.59194336],
       [-1.20093113, -1.14764529],
       [ 1.07681071,  0.47597078],
       [ 1.86906873,  1.51972397],
       [-0.4086731 , -1.29261101],
       [-0.30964085, -0.3648304 ],
       [-0.4086731 ,  1.31677196],
       [ 2.06713324,  0.53395707],
       [ 0.68068169, -1.089659  ],
       [-0.90383437,  0.38899135],
       [-1.20093113,  0.30201192],
       [ 1.07681071, -1.20563157],
       [-1.49802789, -1.43757673],
       [-0.60673761, -1.49556302],
       [ 2.1661655 , -0.79972756],
       [-1.89415691,  0.18603934],
       [-0.21060859,  0.85288166],
       [-1.89415691, -1.26361786],
       [ 2.1661655 ,  0.38899135],
       [-1.39899564,  0.56295021],
       [-1.10189888, -0.33583725],
       [ 0.18552042, -0.65476184],
       [ 0.38358493,  0.01208048],
       [-0.60673761,  2.331532  ],
       [-0.30964085,  0.21503249],
       [-1.59706014, -0.19087153],
       [ 0.68068169, -1.37959044],
       [-1.10189888,  0.56295021],
       [-1.99318916,  0.35999821],
       [ 0.38358493,  0.27301877],
       [ 0.18552042, -0.27785096],
       [ 1.47293972, -1.03167271],
       [ 0.8787462 ,  1.08482681],
       [ 1.96810099,  2.15757314],
       [ 2.06713324,  0.38899135],
       [-1.39899564, -0.42281668],
       [-1.20093113, -1.00267957],
       [ 1.96810099, -0.91570013],
       [ 0.38358493,  0.30201192],
       [ 0.18552042,  0.1570462 ],
       [ 2.06713324,  1.75166912],
       [ 0.77971394, -0.8287207 ],
       [ 0.28455268, -0.27785096],
       [ 0.38358493, -0.16187839],
       [-0.11157634,  2.21555943],
       [-1.49802789, -0.62576869],
       [-1.29996338, -1.06066585],
       [-1.39899564,  0.41798449],
       [-1.10189888,  0.76590222],
       [-1.49802789, -0.19087153],
       [ 0.97777845, -1.06066585],
       [ 0.97777845,  0.59194336],
       [ 0.38358493,  0.99784738]])
```

In [5]:
```
y_test
```

Out[5]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
      1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1], dtype=int64)
```

In [6]:
```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

Out[6]:
```
SVC(kernel='linear', random_state=0)
```

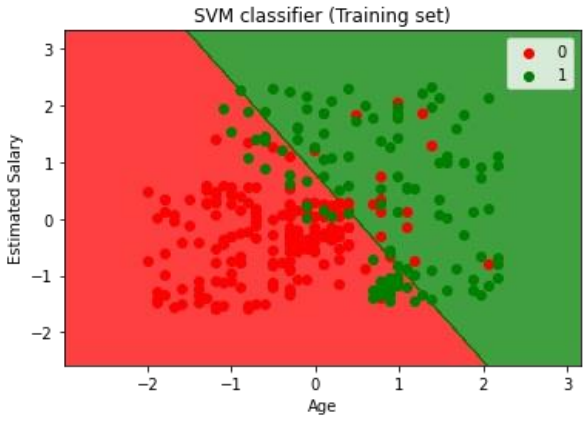
In [7]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
y_pred
```

Out[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

In [8]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

In [9]:
```
#Visualizing the training set result:

from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

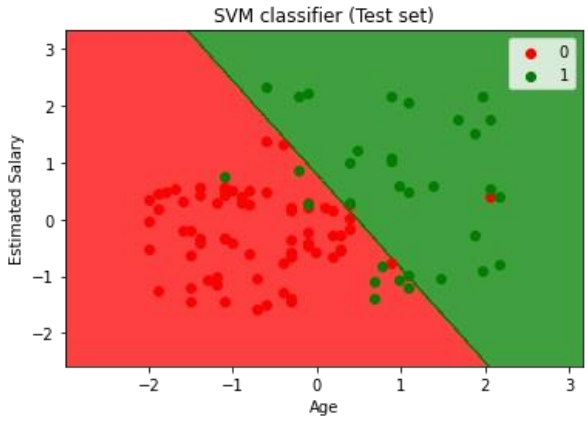
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [10]:
```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
```

```
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [15]:

```
accuracy = metrics.accuracy_score(y_test,y_pred)
report = metrics.classification_report(y_test,y_pred)
cm = metrics.confusion_matrix(y_test,y_pred)

print("Classification report:")
print("Accuracy: ", accuracy)
print(report)
print("Confusion matrix:")
print(cm)
```

```
Classification report:
Accuracy:  0.9
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        68
           1       0.92      0.75      0.83        32

    accuracy                           0.90       100
   macro avg       0.91      0.86      0.88       100
weighted avg       0.90      0.90      0.90       100

Confusion matrix:
[[66  2]
 [ 8 24]]
```

Ass 11 Build a decision tree classifier and evaluate performance of a classifier by printing classification report.

In [11]:
```
# Decision Tree CLassifier

# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics
```

In [12]:
```
# Importing the datasets

datasets = pd.read_csv('Social_Network_Ads.csv')
#feature_cols = ['Age', 'EstimatedSalary']
X = datasets.iloc[:, [2,3]].values
Y = datasets.iloc[:, 4].values
```

In [13]:
```
# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

In [14]:
```
# Feature Scaling

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)
```

In [15]:
```
# Fitting the classifier into the Training set

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', max_depth=3)

classifier.fit(X_Train, Y_Train)
```

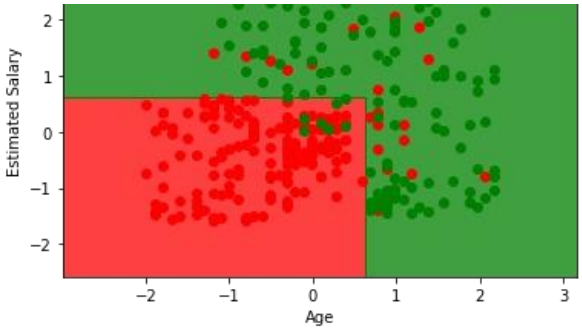
Out[15]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [16]:
```
# Predicting the test set results

Y_Pred = classifier.predict(X_Test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Y_Test, Y_Pred))
```

```
Accuracy: 0.94
```

In [17]:
```
from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Train, Y_Train
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

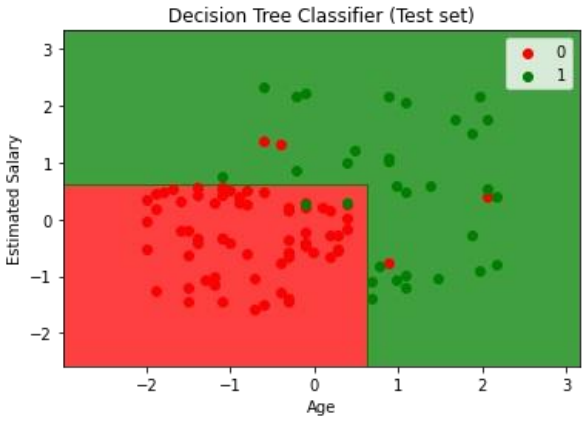
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D a
rray with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D a
rray with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [18]:

```
# Visualising the Test set results

from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Test, Y_Test
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [19]:

```
accuracy = metrics.accuracy_score(Y_Test,Y_Pred)
report = metrics.classification_report(Y_Pred, Y_Test)
cm = metrics.confusion_matrix(Y_Test, Y_Pred)

print("Classification report:")
print("Accuracy: ", accuracy)
print(report)
print("Confusion matrix:")
print(cm)
```

```
Classification report:
Accuracy:  0.94
              precision    recall  f1-score   support

           0       0.94      0.97      0.96        66
           1       0.94      0.88      0.91        34

    accuracy                           0.94       100
   macro avg       0.94      0.93      0.93       100
weighted avg       0.94      0.94      0.94       100
```

```
Confusion matrix:
[[64  4]
 [ 2 30]]
```

In [18]:

Ass 12 Build random forest and extremely random forest classifiers and analyze the output.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn import metrics

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [19]:

```
data_set
```

Out[19]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [20]:

```
#Fitting Decision Tree classifier to the training set  random forest
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
```

Out[20]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [21]:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

In [22]:

```
y_pred
```

Out[22]:

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

In [23]:

```
#Now we will create the confusion matrix to determine the correct and incorrect predictions.

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

In [24]:

```
cm
```

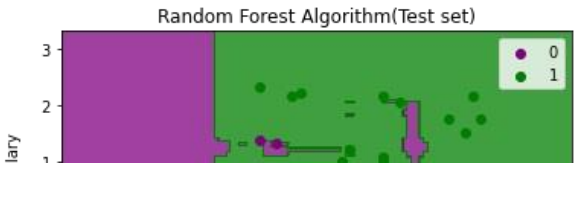
Out[24]:
```
array([[65,  3],
       [ 4, 28]], dtype=int64)
```

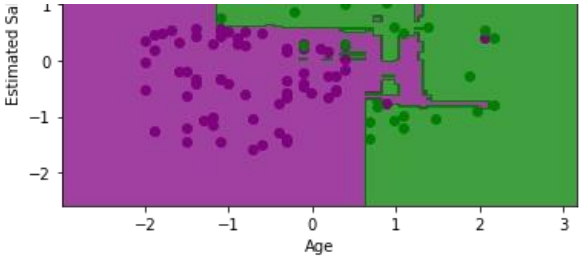
In [25]:
```
#Visualizing the training Set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

In [26]:
```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```





In [38]:

```
accuracy = metrics.accuracy_score(y_test,y_pred)
report = metrics.classification_report(y_test,y_pred)
cm = metrics.confusion_matrix(y_test,y_pred)

print("Classification report:")
print("Accuracy: ", accuracy)
print(report)
print("Confusion matrix:")
print(cm)
```

```
Classification report:
Accuracy:  0.93
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        68
           1       0.90      0.88      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100

Confusion matrix:
[[65  3]
 [ 4 28]]
```

In [1]:

Ass 13 Implement K-Means algorithm for clustering.

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
df = pd.read_csv("Book1.csv")
df.head()
```

Out[1]:

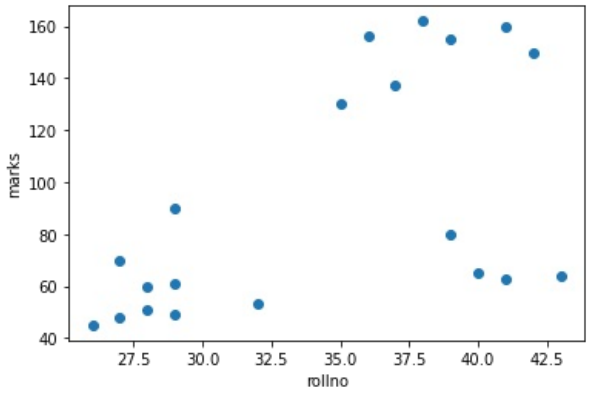
	name	rollno	marks
0	A	40	65
1	B	41	63
2	C	43	64
3	D	39	80
4	E	36	156

In [2]:

```
plt.scatter(df.rollno,df['marks'])
plt.xlabel('rollno')
plt.ylabel('marks')
```

Out[2]:

```
Text(0, 0.5, 'marks')
```



In [3]:

```
km = KMeans(n_clusters=3)
predicted = km.fit_predict(df[['rollno','marks']])
predicted
```

Out[3]:

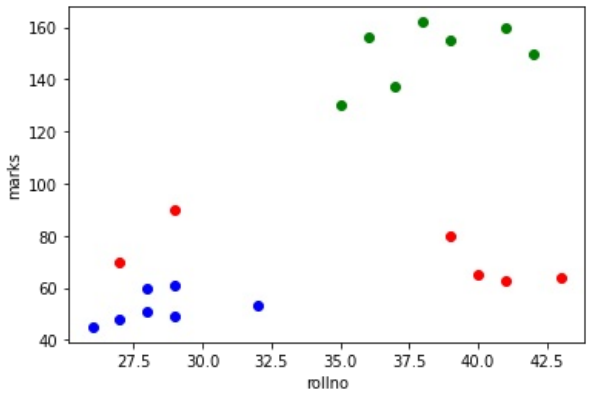
```
array([1, 1, 1, 1, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 2, 2, 0, 0, 0, 0])
```

In [4]:

```
df['cluster']=predicted
df.head()
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.rollno,df1['marks'],color='green')
plt.scatter(df2.rollno,df2['marks'],color='red')
plt.scatter(df3.rollno,df3['marks'],color='blue')
plt.xlabel('rollno')
plt.ylabel('marks')
```

Out[4]:

```
Text(0, 0.5, 'marks')
```



In [5]:
```
scale = MinMaxScaler()

scale.fit(df[['marks']])
df['marks'] = scale.transform(df[['marks']])

scale.fit(df[['rollno']])
df['rollno'] = scale.transform(df[['rollno']])
```

In [6]:
```
km = KMeans(n_clusters=3)
predicted = km.fit_predict(df[['rollno','marks']])
predicted
```

Out[6]:
```
array([2, 2, 2, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1])
```

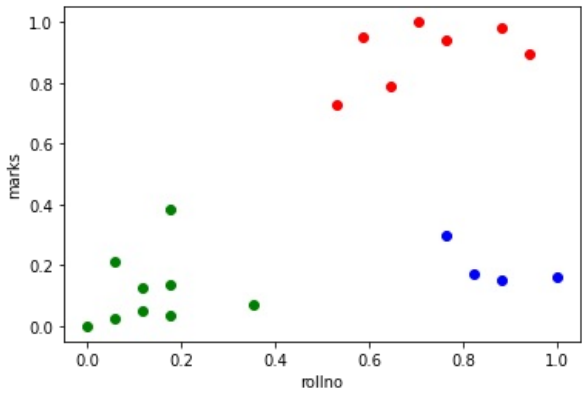
In [7]:
```
df = df.drop(['cluster'], axis='columns')

df['cluster']=predicted
df.head()
```

Out[7]:

	name	rollno	marks	cluster
0	A	0.823529	0.170940	2
1	B	0.882353	0.153846	2
2	C	1.000000	0.162393	2
3	D	0.764706	0.299145	2
4	E	0.588235	0.948718	1

In [8]:
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.rollno,df1['marks'],color='green')
plt.scatter(df2.rollno,df2['marks'],color='red')
plt.scatter(df3.rollno,df3['marks'],color='blue')
plt.xlabel('rollno')
plt.ylabel('marks')
```

Out[8]:
```
Text(0, 0.5, 'marks')
```



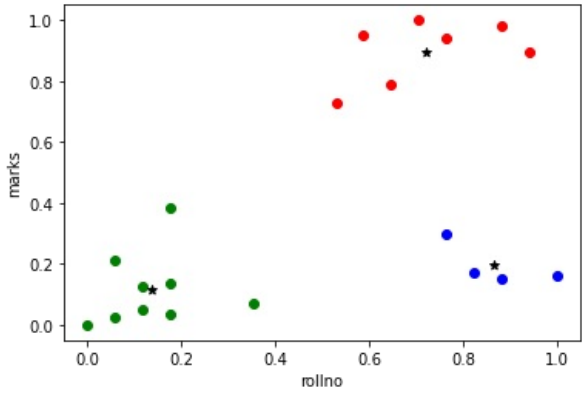
In [9]:
```
km.cluster_centers_
```

Out[9]:
```
array([[0.1372549 , 0.11585945],
       [0.72268908, 0.8974359 ],
       [0.86764706, 0.1965812 ]])
```

In [10]:
```
plt.scatter(df1.rollno,df1['marks'],color='green')
plt.scatter(df2.rollno,df2['marks'],color='red')
plt.scatter(df3.rollno,df3['marks'],color='blue')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='black',marker='*')
```

```
plt.xlabel('rollno')
plt.ylabel('marks')
```

Out[10]: Text(0, 0.5, 'marks')



In [ ]:

In [1]:

Ass 14 Build K-nearest classifier

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn import metrics

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [2]:

```
data_set
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [3]:

```
x_test
```

Out[3]:

```
array([[-0.80480212,  0.50496393],
       [-0.01254409, -0.5677824 ],
       [-0.30964085,  0.1570462 ],
       [-0.80480212,  0.27301877],
       [-0.30964085, -0.5677824 ],
       [-1.10189888, -1.43757673],
       [-0.70576986, -1.58254245],
       [-0.21060859,  2.15757314],
       [-1.99318916, -0.04590581],
       [ 0.8787462 , -0.77073441],
       [-0.80480212, -0.59677555],
       [-1.00286662, -0.42281668],
       [-0.11157634, -0.42281668],
       [ 0.08648817,  0.21503249],
       [-1.79512465,  0.47597078],
       [-0.60673761,  1.37475825],
       [-0.11157634,  0.21503249],
       [-1.89415691,  0.44697764],
       [ 1.67100423,  1.75166912],
       [-0.30964085, -1.37959044],
       [-0.30964085, -0.65476184],
       [ 0.8787462 ,  2.15757314],
       [ 0.28455268, -0.53878926],
       [ 0.8787462 ,  1.02684052],
       [-1.49802789, -1.20563157],
       [ 1.07681071,  2.07059371],
       [-1.00286662,  0.50496393],
       [-0.90383437,  0.30201192],
       [-0.11157634, -0.21986468],
       [-0.60673761,  0.47597078],
```

```
[-1.6960924 ,  0.53395707],
[-0.11157634,  0.27301877],
[ 1.86906873, -0.27785096],
[-0.11157634, -0.48080297],
[-1.39899564, -0.33583725],
[-1.99318916, -0.50979612],
[-1.59706014,  0.33100506],
[-0.4086731 , -0.77073441],
[-0.70576986, -1.03167271],
[ 1.07681071, -0.97368642],
[-1.10189888,  0.53395707],
[ 0.28455268, -0.50979612],
[-1.10189888,  0.41798449],
[-0.30964085, -1.43757673],
[ 0.48261718,  1.22979253],
[-1.10189888, -0.33583725],
[-0.11157634,  0.30201192],
[ 1.37390747,  0.59194336],
[-1.20093113, -1.14764529],
[ 1.07681071,  0.47597078],
[ 1.86906873,  1.51972397],
[-0.4086731 , -1.29261101],
[-0.30964085, -0.3648304 ],
[-0.4086731 ,  1.31677196],
[ 2.06713324,  0.53395707],
[ 0.68068169, -1.089659  ],
[-0.90383437,  0.38899135],
[-1.20093113,  0.30201192],
[ 1.07681071, -1.20563157],
[-1.49802789, -1.43757673],
[-0.60673761, -1.49556302],
[ 2.1661655 , -0.79972756],
[-1.89415691,  0.18603934],
[-0.21060859,  0.85288166],
[-1.89415691, -1.26361786],
[ 2.1661655 ,  0.38899135],
[-1.39899564,  0.56295021],
[-1.10189888, -0.33583725],
[ 0.18552042, -0.65476184],
[ 0.38358493,  0.01208048],
[-0.60673761,  2.331532  ],
[-0.30964085,  0.21503249],
[-1.59706014, -0.19087153],
[ 0.68068169, -1.37959044],
[-1.10189888,  0.56295021],
[-1.99318916,  0.35999821],
[ 0.38358493,  0.27301877],
[ 0.18552042, -0.27785096],
[ 1.47293972, -1.03167271],
[ 0.8787462 ,  1.08482681],
[ 1.96810099,  2.15757314],
[ 2.06713324,  0.38899135],
[-1.39899564, -0.42281668],
[-1.20093113, -1.00267957],
[ 1.96810099, -0.91570013],
[ 0.38358493,  0.30201192],
[ 0.18552042,  0.1570462 ],
[ 2.06713324,  1.75166912],
[ 0.77971394, -0.8287207 ],
[ 0.28455268, -0.27785096],
[ 0.38358493, -0.16187839],
[-0.11157634,  2.21555943],
[-1.49802789, -0.62576869],
[-1.29996338, -1.06066585],
[-1.39899564,  0.41798449],
[-1.10189888,  0.76590222],
[-1.49802789, -0.19087153],
[ 0.97777845, -1.06066585],
[ 0.97777845,  0.59194336],
[ 0.38358493,  0.99784738]])
```

In [4]:
```
y_test
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
      1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1], dtype=int64)
```

In [5]:
```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

Out[5]:
```
KNeighborsClassifier()
```

In [6]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

In [7]:
```
y_pred
```

Out[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
      1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], dtype=int64)
```

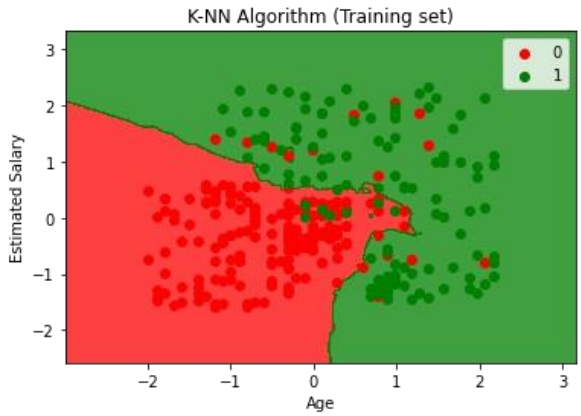
In [8]:
```
#Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

In [9]:
```
cm
```

Out[9]:
```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

In [10]:
```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

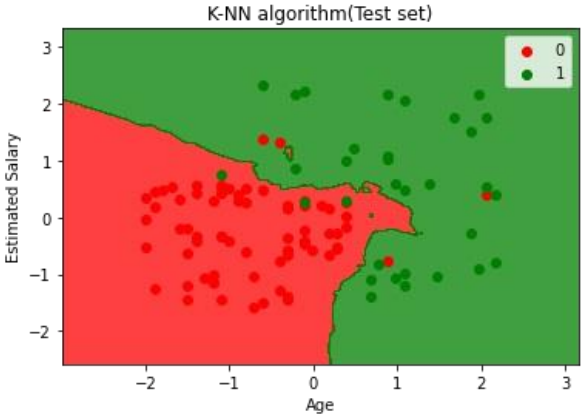


In [11]:
```
"""
As we can see the graph is showing the red point and green points.
The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-N
"""
```

Out[11]:
```
'\nAs we can see the graph is showing the red point and green points. \nThe green points are for Purchased(1) and
Red Points for not Purchased(0) variable.\nThe graph is showing an irregular boundary instead of showing any stra
ight line or any curve because it is a K-NN algorithm, \n'
```

In [14]:
```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have
precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D a
rray with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have
precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D a
rray with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [15]:
```
accuracy = metrics.accuracy_score(y_test,y_pred)
report = metrics.classification_report(y_test,y_pred)
cm = metrics.confusion_matrix(y_test,y_pred)

print("Classification report:")
print("Accuracy: ", accuracy)
print(report)
print("Confusion matrix:")
print(cm)
```

```
Classification report:
Accuracy:  0.93
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        68
           1       0.88      0.91      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100

Confusion matrix:
[[64  4]
 [ 3 29]]
```

In [1]:

Ass 15 Visualizing audio signals.

```
pip install pyaudio
```

```
Requirement already satisfied: pyaudio in c:\users\shree\anaconda3\lib\site-packages (0.2.12)
Note: you may need to restart the kernel to use updated packages.
```

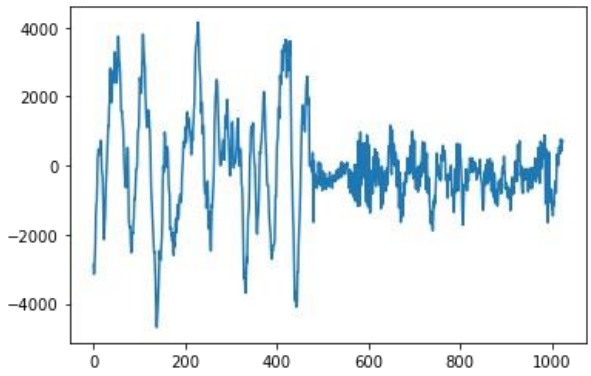
In [2]:

```
pip install wave
```

```
Requirement already satisfied: wave in c:\users\shree\anaconda3\lib\site-packages (0.0.2)
Note: you may need to restart the kernel to use updated packages.
```

In [24]:

```
import pyaudio
import wave

filename = 'file_example_WAV_1MG.wav'

# Set chunk size of 1024 samples per data frame
CHUNKSIZE = 1024

# Now open the sound file, name as wavefile
wavefile = wave.open ( filename, 'rb' )

# Create an interface to PortAudio
portaudio = pyaudio.PyAudio ()

# Open a .Stream object to write the WAV file to play the audio using pyaudio
# in this code, 'output = True' means that the audio will be played rather than recorded
stream = portaudio.open(format=pyaudio.paInt16, channels=1, rate=44100, input=True, frames_per_buffer=CHUNKSIZE)


# do this as long as you want fresh samples
data = stream.read(CHUNKSIZE)
numpydata = np.frombuffer(data, dtype=np.int16)

# plot data
plt.plot(numpydata)
plt.show()

# close stream
stream.stop_stream()
stream.close()
portaudio.terminate()
```



Ass 16 Transform audio signals to the frequency domain.

In [3]:
```
#Transforming audio signals to the frequency domain

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

In [4]:
```
# Read the audio file
sampling_freq, signal = wavfile.read('file_example_WAV_1MG.wav')
sampling_freq
```

Out[4]: 44100

In [5]:
```
signal
```

Out[5]: array([4395, 15134, 19572, ..., -5859,   701, 7220], dtype=int16)

In [7]:
```
# Normalize the values
signal = signal / np.power(2, 15)
signal
```

Out[7]:
```
array([ 4.09316272e-06,  1.40946358e-05,  1.82278454e-05, ...,
       -5.45661896e-06,  6.52857125e-07,  6.72414899e-06])
```

In [8]:
```
# Extract the length of the audio signal
len_signal = len(signal)
len_signal
```

Out[8]: 176400

In [9]:
```
# Extract the half length
len_half = np.ceil((len_signal + 1) / 2.0).astype(np.int)
len_half
```

```
<ipython-input-9-01ad0ebda8d0>:2: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To si
lence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.
int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your curr
ent use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  len_half = np.ceil((len_signal + 1) / 2.0).astype(np.int)
```

Out[9]: 88201

In [10]:
```
# Apply Fourier transform
freq_signal = np.fft.fft(signal)
freq_signal
```

Out[10]:
```
array([ 1.35409559e+00+0.j        , -3.86887177e-04-0.00086196j,
       -1.41686190e-04-0.00163807j, ...,  4.18115153e-04+0.00230829j,
       -1.41686190e-04+0.00163807j, -3.86887177e-04+0.00086196j])
```

In [11]:
```
# Normalization
freq_signal = abs(freq_signal[0:len_half]) / len_signal
freq_signal
```

Out[11]:
```
array([7.67627886e-06, 5.35606006e-09, 9.32077569e-09, ...,
       2.24123364e-09, 1.80002006e-09, 4.80554529e-09])
```

In [12]:
```

```

```
# Take the square
freq_signal **= 2
freq_signal
```

Out[12]:
```
array([5.89252571e-11, 2.86873793e-17, 8.68768595e-17, ...,
       5.02312823e-18, 3.24007223e-18, 2.30932655e-17])
```

In [13]:
```
# Extract the length of the frequency transformed signal
len_fts = len(freq_signal)
len_fts
```

Out[13]:
```
88201
```

In [14]:
```
# Adjust the signal for even and odd cases
if len_signal % 2:
    freq_signal[1:len_fts] *= 2
else:
    freq_signal[1:len_fts-1] *= 2
freq_signal
```

Out[14]:
```
array([5.89252571e-11, 5.73747586e-17, 1.73753719e-16, ...,
       1.00462565e-17, 6.48014446e-18, 2.30932655e-17])
```

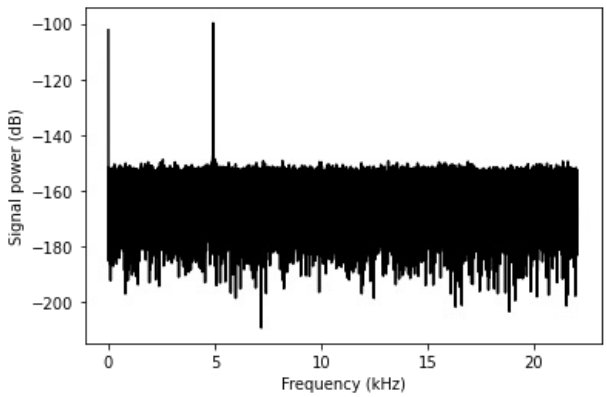
In [15]:
```
# Extract the power value in dB
signal_power = 10 * np.log10(freq_signal)
signal_power
```

Out[15]:
```
array([-102.29698514, -162.41279128, -157.60065891, ..., -169.9799574 ,
       -171.88415313, -166.36514651])
```

In [16]:
```
# Build the X axis
x_axis = np.arange(0, len_half, 1) * (sampling_freq / len_signal) / 1000.0
x_axis
```

Out[16]:
```
array([0.000000e+00, 2.500000e-04, 5.000000e-04, ..., 2.204950e+01,
       2.204975e+01, 2.205000e+01])
```

In [17]:
```
# Plot the figure
plt.figure()
plt.plot(x_axis, signal_power, color='black')
plt.xlabel('Frequency (kHz)')
plt.ylabel('Signal power (dB)')
plt.show()
```



Ass 17 Generate audio signals.

In [1]:
```
#Generating audio signals
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import write
```

In [2]:
```
# Output file where the audio will be saved
output_file = 'file_example_WAV_1MG.wav'
output_file
```

Out[2]:
```
'file_example_WAV_1MG.wav'
```

In [3]:
```
# Specify audio parameters
duration = 4 # in seconds
sampling_freq = 44100 # in Hz
tone_freq = 784
min_val = -4 * np.pi
max_val = 4 * np.pi
min_val
```

Out[3]:
```
-12.566370614359172
```

In [4]:
```
max_val
```

Out[4]:
```
12.566370614359172
```

In [5]:
```
# Generate the audio signal
t = np.linspace(min_val, max_val, duration * sampling_freq)
signal = np.sin(2 * np.pi * tone_freq * t)
signal
```

Out[5]:
```
array([-0.21545456,  0.46592985,  0.92707287, ..., -0.92707287,
       -0.46592985,  0.21545456])
```

In [6]:
```
# Add some noise to the signal
noise = 0.5 * np.random.rand(duration * sampling_freq)
signal += noise
signal
```

Out[6]:
```
array([ 0.01910156,  0.87679923,  0.93300202, ..., -0.90592815,
       -0.1433313 ,  0.61928448])
```

In [7]:
```
# Scale it to 16-bit integer values
scaling_factor = np.power(2, 15) - 1
signal_normalized = signal / np.max(np.abs(signal))
signal_scaled = np.int16(signal_normalized * scaling_factor)
```

In [8]:
```
# Save the audio signal in the output file
write(output_file, sampling_freq, signal_scaled)
```

In [9]:
```
# Extract the first 200 values from the audio signal
signal = signal[:200]
signal
```

Out[9]:
```
array([ 0.01910156,  0.87679923,  0.93300202,  1.20084095,  0.59751099,
        0.32800923, -0.65554471, -0.84904501, -0.73945895,  0.1659497 ,
        0.76538811,  1.12601182,  1.05160789,  0.99035534, -0.03240642,
       -0.66429789, -0.8941521 , -0.6072162 ,  0.14100141,  0.70354695,
        1.13369153,  0.93898058,  0.56456554,  0.22616012, -0.74885912,
       -0.81896298, -0.63122453, -0.08520803,  0.96808056,  0.96696053,
        1.28456513,  0.68600575,  0.14244555, -0.69029068, -0.9731353 ,
       -0.69631563,  0.1985799 ,  0.68164228,  1.37200428,  1.22068164,
        0.656345  , -0.0212136 , -0.63368662, -0.91217537, -0.31401328,
       -0.02706857,  1.04302608,  1.12495529,  1.27908605,  0.38221075,
```

```
-0.00730007, -0.64910701, -0.82597139, -0.33609708,  0.17725865,
 1.07104741,  1.11422346,  1.32034329,  0.77591782, -0.23074321,
-0.77770929, -0.73669159, -0.17638243,  0.35037067,  0.71883684,
 1.41915549,  1.23698307,  0.68168969,  0.04431045, -0.65386521,
-0.49548509, -0.27242616,  0.38794018,  0.69931448,  1.07446398,
 0.96296957,  0.34754574, -0.01000837, -0.62086286, -0.75843535,
-0.11476115,  0.11716503,  1.099311  ,  1.32815884,  0.99859262,
 0.33814941, -0.4223637 , -0.80954825, -0.83685166, -0.08861978,
 0.42982934,  0.79495695,  1.28088885,  0.91416244,  0.25438975,
-0.03703161, -0.70899847, -0.53170802, -0.42297118,  0.38635521,
 1.14344432,  1.43913599,  0.89300713,  0.3643961 , -0.39156018,
-0.69596112, -0.70829806, -0.3147556 ,  0.4804264 ,  1.14390386,
 1.20640961,  0.96288547,  0.35794489, -0.41440784, -0.54521299,
-0.76193466, -0.29332922,  0.34909692,  1.10474203,  1.06679532,
 1.05435966,  0.45134222, -0.05630375, -0.61076673, -0.82211217,
-0.31287767,  0.46267506,  0.98769215,  1.14066723,  0.70738832,
 0.35263067, -0.54483529, -0.51851751, -0.74955556, -0.02071301,
 0.550023  ,  1.13260921,  1.20591638,  1.10507611,  0.18948373,
-0.21059057, -0.88809368, -0.8071389 , -0.09520328,  0.39288437,
 1.25431077,  1.11930674,  0.86987735,  0.51571905, -0.51490029,
-0.50659655, -0.57149055, -0.23661753,  0.84042416,  1.32497186,
 1.37151453,  0.9548796 ,  0.17701768, -0.58117522, -0.56726062,
-0.5845032 ,  0.01859787,  0.53437672,  0.95420842,  1.15912141,
 1.05468748,  0.15098386, -0.20489737, -0.57467651, -0.62206065,
-0.12389054,  0.57857387,  1.16855834,  1.23475029,  0.61751239,
 0.27076494, -0.58345612, -0.5466227 , -0.51057451,  0.05983271,
 0.61350122,  1.39397562,  1.41011501,  0.68179405,  0.15145781,
-0.57506299, -0.98105661, -0.4395654 , -0.11883405,  0.87854301,
 1.32205519,  1.29773002,  0.95341002,  0.16688766, -0.35376941,
-0.72970514, -0.55570342,  0.15145027,  0.52255599,  1.35911237])
```

In [10]:

```
# Construct the time axis in milliseconds
time_axis = 1000 * np.arange(0, len(signal), 1) / float(sampling_freq)
time_axis
```

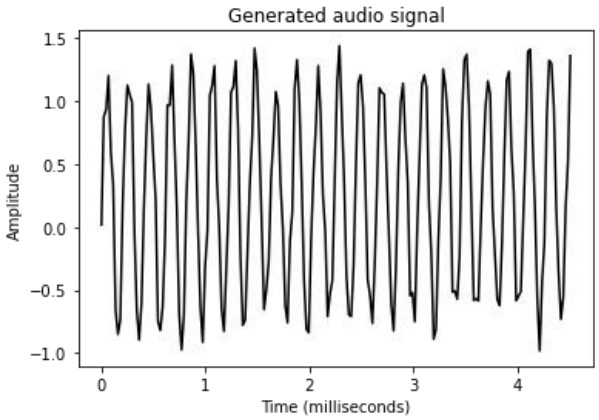
Out[10]:

```
array([0.        , 0.02267574, 0.04535147, 0.06802721, 0.09070295,
       0.11337868, 0.13605442, 0.15873016, 0.1814059 , 0.20408163,
       0.22675737, 0.24943311, 0.27210884, 0.29478458, 0.31746032,
       0.34013605, 0.36281179, 0.38548753, 0.40816327, 0.430839  ,
       0.45351474, 0.47619048, 0.49886621, 0.52154195, 0.54421769,
       0.56689342, 0.58956916, 0.6122449 , 0.63492063, 0.65759637,
       0.68027211, 0.70294785, 0.72562358, 0.74829932, 0.77097506,
       0.79365079, 0.81632653, 0.83900227, 0.861678  , 0.88435374,
       0.90702948, 0.92970522, 0.95238095, 0.97505669, 0.99773243,
       1.02040816, 1.0430839 , 1.06575964, 1.08843537, 1.11111111,
       1.13378685, 1.15646259, 1.17913832, 1.20181406, 1.2244898 ,
       1.24716553, 1.26984127, 1.29251701, 1.31519274, 1.33786848,
       1.36054422, 1.38321995, 1.40589569, 1.42857143, 1.45124717,
       1.4739229 , 1.49659864, 1.51927438, 1.54195011, 1.56462585,
       1.58730159, 1.60997732, 1.63265306, 1.6553288 , 1.67800454,
       1.70068027, 1.72335601, 1.74603175, 1.76870748, 1.79138322,
       1.81405896, 1.83673469, 1.85941043, 1.88208617, 1.9047619 ,
       1.92743764, 1.95011338, 1.97278912, 1.99546485, 2.01814059,
       2.04081633, 2.06349206, 2.0861678 , 2.10884354, 2.13151927,
       2.15419501, 2.17687075, 2.19954649, 2.22222222, 2.24489796,
       2.2675737 , 2.29024943, 2.31292517, 2.33560091, 2.35827664,
       2.38095238, 2.40362812, 2.42630385, 2.44897959, 2.47165533,
       2.49433107, 2.5170068 , 2.53968254, 2.56235828, 2.58503401,
       2.60770975, 2.63038549, 2.65306122, 2.67573696, 2.6984127 ,
       2.72108844, 2.74376417, 2.76643991, 2.78911565, 2.81179138,
       2.83446712, 2.85714286, 2.87981859, 2.90249433, 2.92517007,
       2.9478458 , 2.97052154, 2.99319728, 3.01587302, 3.03854875,
       3.06122449, 3.08390023, 3.10657596, 3.1292517 , 3.15192744,
       3.17460317, 3.19727891, 3.21995465, 3.24263039, 3.26530612,
       3.28798186, 3.3106576 , 3.33333333, 3.35600907, 3.37868481,
       3.40136054, 3.42403628, 3.44671202, 3.46938776, 3.49206349,
       3.51473923, 3.53741497, 3.5600907 , 3.58276644, 3.60544218,
       3.62811791, 3.65079365, 3.67346939, 3.69614512, 3.71882086,
       3.7414966 , 3.76417234, 3.78684807, 3.80952381, 3.83219955,
       3.85487528, 3.87755102, 3.90022676, 3.92290249, 3.94557823,
       3.96825397, 3.99092971, 4.01360544, 4.03628118, 4.05895692,
       4.08163265, 4.10430839, 4.12698413, 4.14965986, 4.1723356 ,
       4.19501134, 4.21768707, 4.24036281, 4.26303855, 4.28571429,
       4.30839002, 4.33106576, 4.3537415 , 4.37641723, 4.39909297,
       4.42176871, 4.44444444, 4.46712018, 4.48979592, 4.51247166])
```

In [11]:

```
# Plot the audio signal
plt.plot(time_axis, signal, color='black')
```

```
plt.xlabel('Time (milliseconds)')
plt.ylabel('Amplitude')
plt.title('Generated audio signal')
plt.show()
```



Ass 18 Installation of NLTK and tokenizing text data

In [1]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\shree\anaconda3\lib\site-packages (3.6.1)
Requirement already satisfied: regex in c:\users\shree\anaconda3\lib\site-packages (from nltk) (2021.4.4)
Requirement already satisfied: click in c:\users\shree\anaconda3\lib\site-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in c:\users\shree\anaconda3\lib\site-packages (from nltk) (4.59.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: joblib in c:\users\shree\anaconda3\lib\site-packages (from nltk) (1.0.1)
```

In [6]:
```
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Out[6]:
```
True
```

In [ ]:
```
pip install gensim
```

In [ ]:
```
pip install pattern
```

In [7]:
```
#Tokenizing text data
from nltk.tokenize import sent_tokenize, \
        word_tokenize, WordPunctTokenizer
```

In [12]:
```
# Define input text
input_text = "Do you know how tokenization works? It's actually  quite interesting! Let's analyze a couple of se
```

In [13]:
```
#Divide the input text into sentence tokens:
# Sentence tokenizer
print("\nSentence tokenizer:")
print(sent_tokenize(input_text))
```

```
Sentence tokenizer:
['Do you know how tokenization works?', "It's actually quite interesting!", "Let's analyze a couple of sentences
and  figure it out."]
```

In [14]:
```
#Divide the input text into word tokens:
# Word tokenizer
print("\nWord tokenizer:")
print(word_tokenize(input_text))
```

```
Word tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'s", 'actually', 'quite', 'interesting', '!', '
Let', "'s", 'analyze', 'a', 'couple', 'of', 'sentences', 'and', 'figure', 'it', 'out', '.']
```

In [17]:
```
#Divide the input text into word tokens using the WordPunct tokenizer:
# WordPunct tokenizer
print("\nWord punct tokenizer:")
print(WordPunctTokenizer().tokenize(input_text))
```

```
Word punct tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'", 's', 'actually', 'quite', 'interesting', '!
', 'Let', "'", 's', 'analyze', 'a', 'couple', 'of', 'sentences', 'and', 'figure', 'it', 'out', '.']
```

Ass 19 Converting words to their base forms using stemming, lemmatization.

In [1]:
```
#Converting words to their base forms using stemming
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer
```

In [2]:
```
#Define some input words:
input_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',
        'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']
```

In [3]:
```
# Create various stemmer objects
porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer('english')
```

In [4]:
```
# Create a list of stemmer names for display
stemmer_names = ['PORTER', 'LANCASTER', 'SNOWBALL']
formatted_text = '{:>16}' * (len(stemmer_names) + 1)
print('\n', formatted_text.format('INPUT WORD', *stemmer_names),
        '\n', '='*68)
```

```
      INPUT WORD           PORTER       LANCASTER        SNOWBALL
====================================================================
```

In [5]:
```
#Iterate through the words and stem them using the three stemmers:
# Stem each word and display the output
for word in input_words:
    output = [word, porter.stem(word),
            lancaster.stem(word), snowball.stem(word)]
    print(formatted_text.format(*output))
```

```
         writing           write            writ           write
          calves            calv            calv            calv
              be              be              be              be
        branded           brand           brand           brand
           horse           hors            hors            hors
       randomize          random          random          random
        possibly         possibl            poss         possibl
       provision          provis          provid          provis
        hospital          hospit          hospit          hospit
            kept            kept            kept            kept
        scratchy        scratchi        scratchy        scratchi
            code            code             cod            code
```

In [6]:
```
#Converting words to their base forms using lemmatization
#Create a new Python file and import the following packages:
from nltk.stem import WordNetLemmatizer
```

In [9]:
```
#Define some input words. We will be using the same set of words that we used in the previous section so that we
input_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',
        'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']
```

In [10]:
```
# Create lemmatizer object
lemmatizer = WordNetLemmatizer()
```

In [11]:
```
#Create a list of lemmatizer names for the table display and format the text accordingly:

lemmatizer_names = ['NOUN LEMMATIZER', 'VERB LEMMATIZER']
formatted_text = '{:>24}' * (len(lemmatizer_names) + 1)
print('\n', formatted_text.format('INPUT WORD', *lemmatizer_names),
        '\n', '='*75)
```

```
               INPUT WORD         NOUN LEMMATIZER         VERB LEMMATIZER
===========================================================================
```

In [12]:
```
# Lemmatize each word and display the output
for word in input_words:
    output = [word, lemmatizer.lemmatize(word, pos='n'),
           lemmatizer.lemmatize(word, pos='v')]
```

```
print(formatted_text.format(*output))
```

```
writing  writing  write
 calves     calf  calve
     be       be     be
branded  branded  brand
  horse    horse  horse
randomize randomize randomize
 possibly possibly possibly
provision provision provision
 hospital hospital hospital
     kept     kept   keep
scratchy scratchy scratchy
     code     code   code
```

In [ ]:

Ass 20 Extracting the frequency of terms using Bag of Words model.

```
#Extracting the frequency of terms using the Bag of Words model
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import brown
from text_chunker import chunker
# Read the data from the Brown corpus
input_data = ' '.join(brown.words()[:5400])


# Number of words in each chunk
chunk_size = 800

#Divide the input text into chunks:
text_chunks = chunker(input_data, chunk_size)
Convert the chunks into dictionary items:

# Convert to dict items
chunks = []
for count, chunk in enumerate(text_chunks):
    d = {'index': count, 'text': chunk}
    chunks.append(d)

# Extract the document term matrix
count_vectorizer = CountVectorizer(min_df=7, max_df=20)
document_term_matrix = count_vectorizer.fit_transform([chunk['text'] for chunk in chunks])

# Extract the vocabulary and display it
vocabulary = np.array(count_vectorizer.get_feature_names())
print("\nVocabulary:\n", vocabulary)


# Generate names for chunks
chunk_names = []
for i in range(len(text_chunks)):
    chunk_names.append('Chunk-' + str(i+1))

# Print the document term matrix
print("\nDocument term matrix:")
formatted_text = '{:>12}' * (len(chunk_names) + 1)
print('\n', formatted_text.format('Word', *chunk_names), '\n')
for word, item in zip(vocabulary, document_term_matrix.T):
    # 'item' is a 'csr_matrix' data structure
    output = [word] + [str(freq) for freq in item.data]
    print(formatted_text.format(*output))
```