

```
In [40]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
In [2]: df = pd.read_csv('Bank Customer Churn Prediction.csv')
```

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

Out[3]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card
0	15634602	619	France	Female	42	2	0.00		1
1	15647311	608	Spain	Female	41	1	83807.86		1
2	15619304	502	France	Female	42	8	159660.80		3
3	15701354	699	France	Female	39	1	0.00		2
4	15737888	850	Spain	Female	43	2	125510.82		1



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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customer_id           10000 non-null  int64
1   credit_score           10000 non-null  int64
2   country                10000 non-null  object
3   gender                 10000 non-null  object
4   age                    10000 non-null  int64
5   tenure                 10000 non-null  int64
6   balance                10000 non-null  float64
7   products_number        10000 non-null  int64
8   credit_card            10000 non-null  int64
9   active_member          10000 non-null  int64
10  estimated_salary        10000 non-null  float64
11  churn                  10000 non-null  int64
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

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

Out[5]:

	customer_id	credit_score	age	tenure	balance	products_number	credit_card
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.530200
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.491654
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.000000
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.000000
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	0.500000

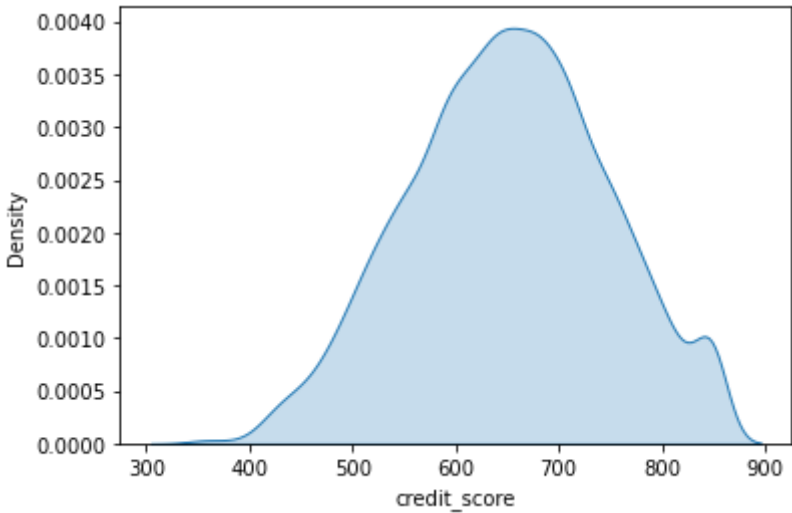
8/31/22, 12:09 PM

Bank\_Customer\_Churn\_Prediction\_1

	customer_id	credit_score	age	tenure	balance	products_number	churn
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	

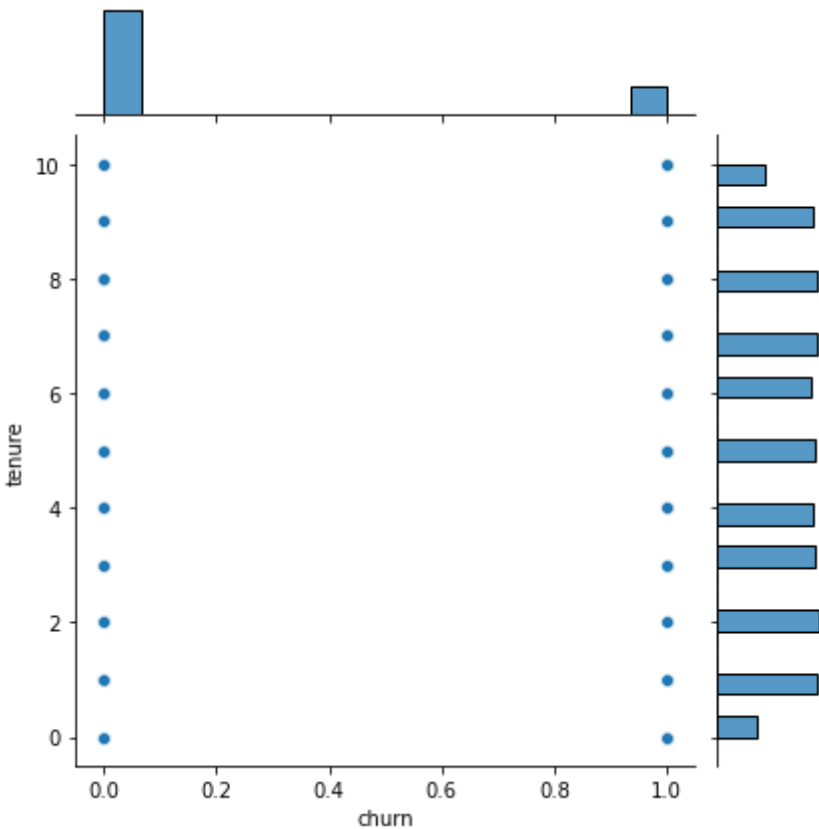
```
In [8]: sns.kdeplot(data = df.credit_score,shade=True)
```

Out[8]: <AxesSubplot:xlabel='credit\_score', ylabel='Density'>



```
In [18]: sns.jointplot(data=df,x=df.churn,y=df.tenure)
```

Out[18]: <seaborn.axisgrid.JointGrid at 0x165310fe640>

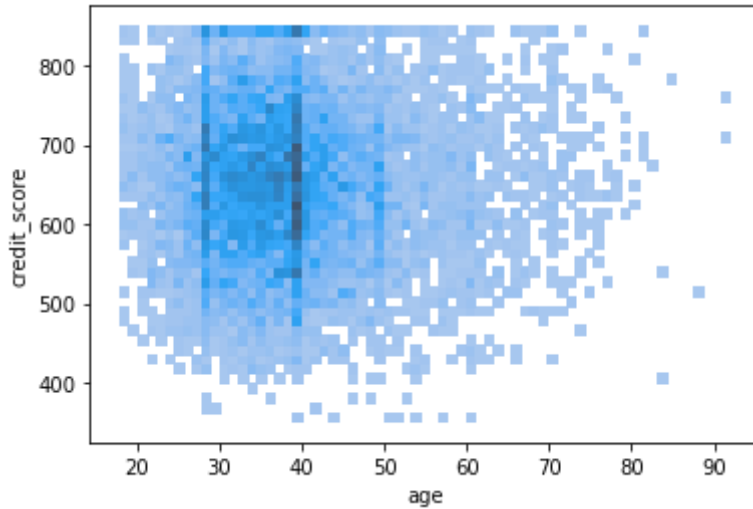


```
In [11]: df.columns
```

```
Out[11]: Index(['customer_id', 'credit_score', 'country', 'gender', 'age', 'tenure',  
          'balance', 'products_number', 'credit_card', 'active_member',  
          'estimated_salary', 'churn'],  
          dtype='object')
```

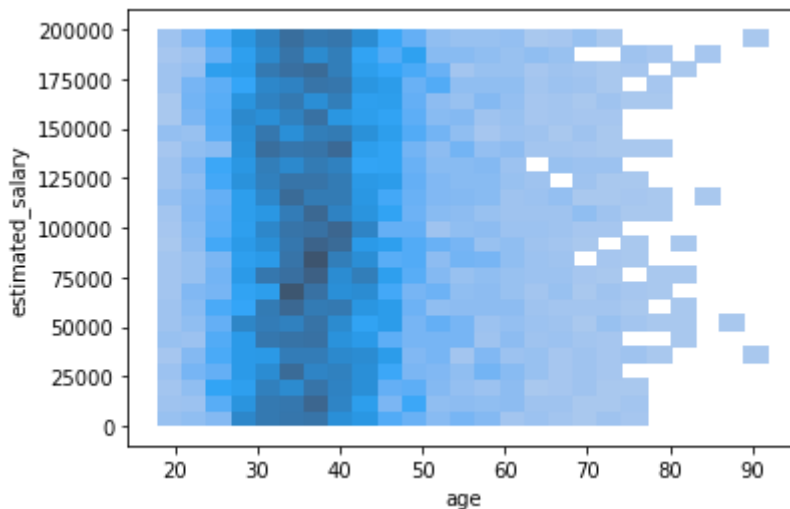
```
In [31]: sns.histplot(x=df.age,y=df.credit_score)
```

```
Out[31]: <AxesSubplot:xlabel='age', ylabel='credit_score'>
```



```
In [34]: sns.histplot(x=df.age,y=df.estimated_salary,bins=25)
```

```
Out[34]: <AxesSubplot:xlabel='age', ylabel='estimated_salary'>
```



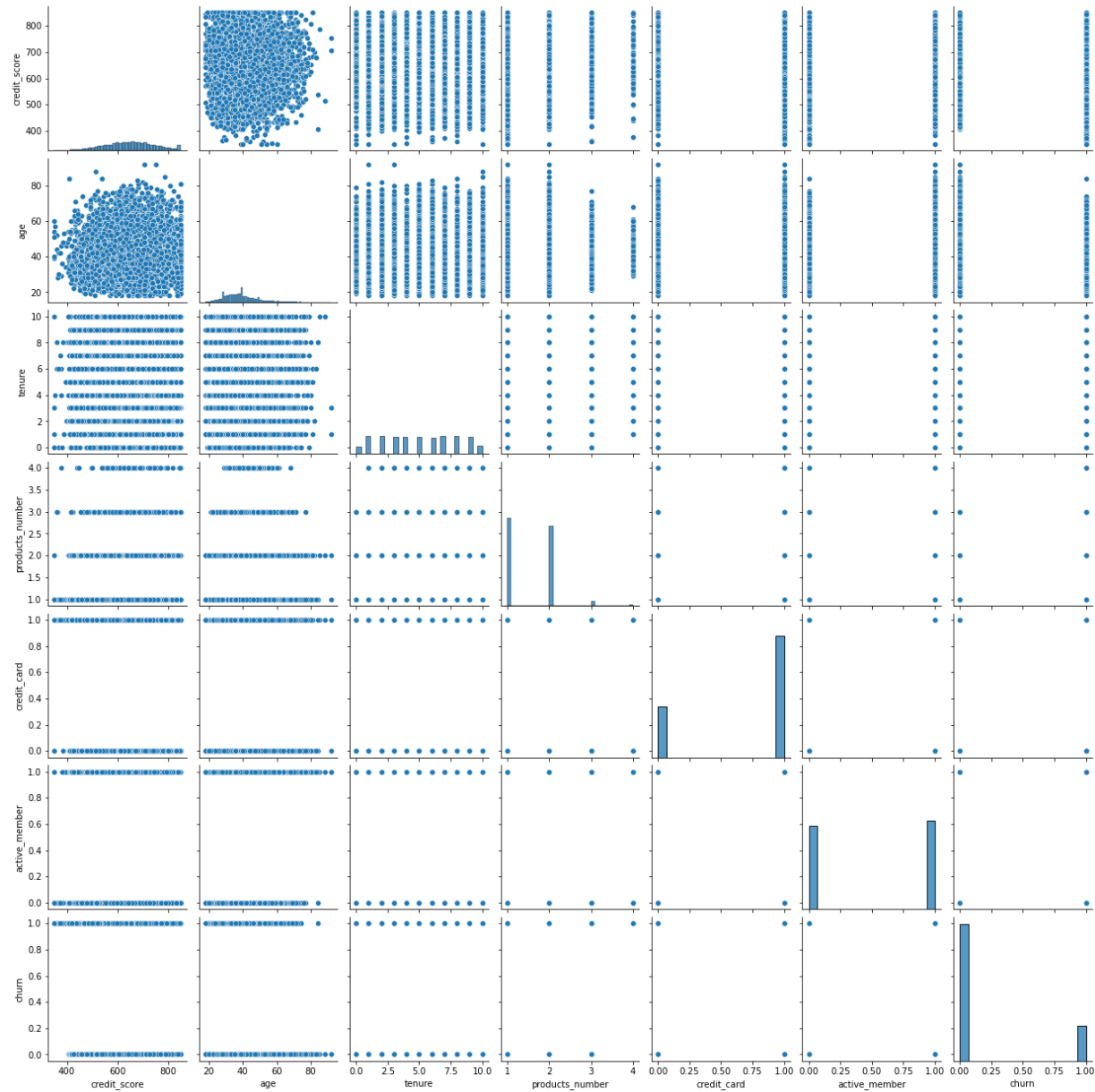
```
In [64]: kd = df.select_dtypes(include=['int64'])  
kd  
kd.drop(columns=['customer_id'],inplace=True)
```

```
In [47]: np.corrcoef(df.age,df.churn)
```

```
Out[47]: array([[1.          , 0.28532304],  
               [0.28532304, 1.          ]])
```

```
In [65]: sns.pairplot(kd)  
plt.figure(figsize = (20,10))
```

Out[65]: <Figure size 1440x720 with 0 Axes>

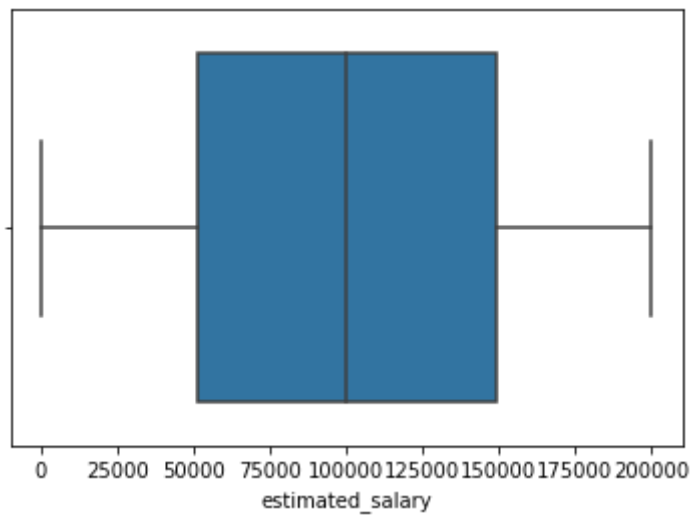


<Figure size 1440x720 with 0 Axes>

## Check outliers in important features

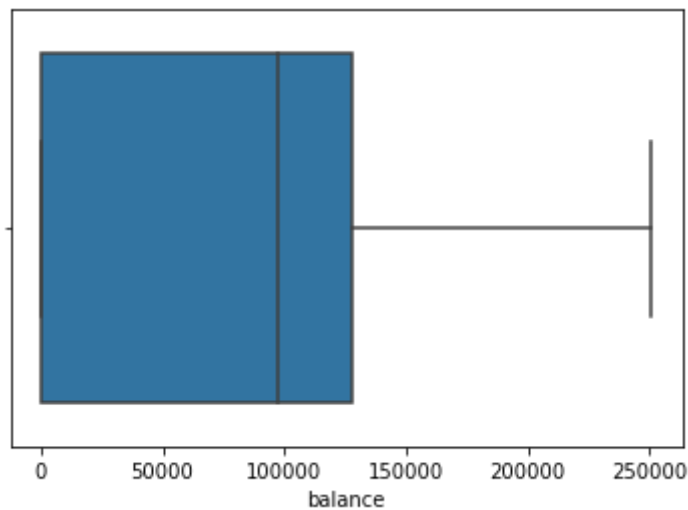
```
In [404... sns.boxplot(x=df['estimated_salary'])
```

Out[404... <AxesSubplot:xlabel='estimated\_salary'>



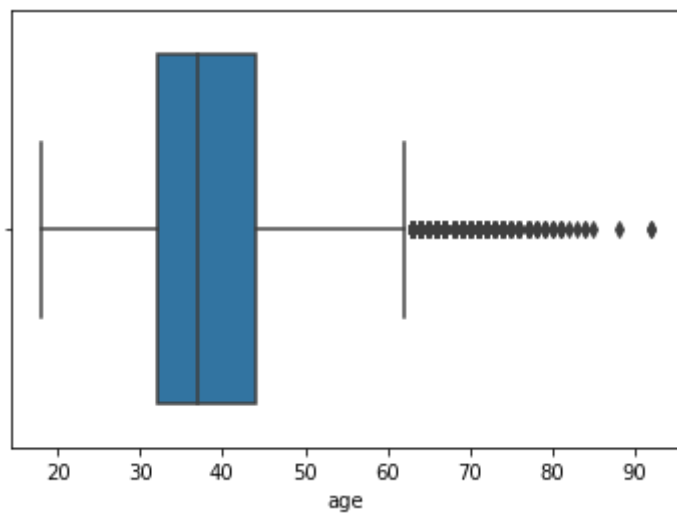
```
In [405... sns.boxplot(x=df['balance'])
```

```
Out[405... <AxesSubplot:xlabel='balance'>
```



```
In [406... sns.boxplot(x=df['age'])
```

```
Out[406... <AxesSubplot:xlabel='age'>
```



```
In [408... df[df.age>70].age.value_counts()
```

```
Out[408... 71    27
          72    21
          74    18
          73    13
          76    11
          77    10
          75     9
          78     5
          81     4
          79     4
          80     3
          84     2
          92     2
          82     1
          83     1
          85     1
          88     1
          Name: age, dtype: int64
```

```
In [428... df.shape
```

```
Out[428... (9930, 12)
```

```
In [429... df.columns
```

```
Out[429... Index(['customer_id', 'credit_score', 'country', 'gender', 'age', 'tenure',
          'balance', 'products_number', 'credit_card', 'active_member',
          'estimated_salary', 'churn'],
          dtype='object')
```

```
In [430... df.country.value_counts()
```

```
Out[430... 1    4984
          3    2485
          2    2461
          Name: country, dtype: int64
```

```
In [431... df.products_number[df.churn == 1].value_counts()
```

```
Out[431... 1    1399
          2     348
          3     218
          4      59
          Name: products_number, dtype: int64
```

```
In [432... df.products_number.value_counts()
```

```
Out[432... 1    5048
          2    4559
          3     264
          4      59
          Name: products_number, dtype: int64
```

```
In [433... df.gender[df.churn == 1].value_counts()
```

```
Out[433... 0    1129
          1     895
          Name: gender, dtype: int64
```

**More number of females are founded to be churn customers**

In [434... `df[df.churn==1].groupby('country')['products_number','credit_score','age','tenure',]`

<ipython-input-434-2d01a026bb29>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.  
`df[df.churn==1].groupby('country')['products_number','credit_score','age','tenure',].mean()`

Out[434... 

	products_number	credit_score	age	tenure
country				
1	1.477019	641.930435	45.136646	5.011180
2	1.512195	648.290244	44.121951	4.663415
3	1.453646	647.415328	44.915946	5.001236

In [435... `df.country[df.churn==1].value_counts()`

Out[435... 

3	809
1	805
2	410

  
Name: country, dtype: int64

**Comparitively less number of churn customers from spain almost half from other 2 countries**

In [436... `df.credit_score[df.churn==1].value_counts()`

Out[436... 

850	43
651	17
705	16
727	13
625	13
..	
437	1
436	1
367	1
431	1
522	1

  
Name: credit\_score, Length: 420, dtype: int64

**Possibility that people with low credit score are less likely to be churn customers**

In [437... `df[df.churn==1].groupby('gender')['products_number','credit_score','age','tenure',].`

<ipython-input-437-77e00ecc3223>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.  
`df[df.churn==1].groupby('gender')['products_number','credit_score','age','tenure',].mean()`

Out[437... 

	products_number	credit_score	age	tenure
gender				
0	1.511957	647.054916	44.798937	4.937112
1	1.427933	643.337430	44.898324	4.936313

In [438... `df[df.churn==0].groupby('gender')['products_number','credit_score','age','tenure',].`

```
<ipython-input-438-d54be157b625>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
df[df.churn==0].groupby('gender')['products_number','credit_score','age','tenure',].mean()
```

Out[438...

	products_number	credit_score	age	tenure
gender				
0	1.554338	652.211726	37.384661	4.975126
1	1.536763	651.727534	37.422610	5.074189

In [439...

```
df.active_member[df.churn==1].value_counts()
```

Out[439...

```
0    1291
1     733
Name: active_member, dtype: int64
```

In [440...

```
df['gender'].replace(['Male','Female'],[1,0],inplace=True)

df
```

Out[440...

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit
0	15634602	619	1	0	42	2	0.00		1
1	15647311	608	2	0	41	1	83807.86		1
2	15619304	502	1	0	42	8	159660.80		3
3	15701354	699	1	0	39	1	0.00		2
4	15737888	850	2	0	43	2	125510.82		1
...	...	...	...	...	...	...	...		...
9995	15606229	771	1	1	39	5	0.00		2
9996	15569892	516	1	1	35	10	57369.61		1
9997	15584532	709	1	0	36	7	0.00		1
9998	15682355	772	3	1	42	3	75075.31		2
9999	15628319	792	1	0	28	4	130142.79		1

9930 rows × 12 columns



In [441...

```
df['country'].replace(['France','Spain','Germany'],[1,2,3],inplace=True)
df
df.
```

```
File "<ipython-input-441-c5a1d0f45c24>", line 3
df.
  ^
SyntaxError: invalid syntax
```

### In active members can be considered as churn customers

Now we will extract our features for our model

In [442...



```
X = df[['credit_score','tenure','active_member','age','country','gender','credit_car
X
y = df['churn']
y
```

```
Out[442...] 0      1
            1      0
            2      1
            3      0
            4      0
            ..
           9995     0
           9996     0
           9997     1
           9998     1
           9999     0
            Name: churn, Length: 9930, dtype: int64
```

## Logistic regression

```
In [443...] from sklearn.linear_model import LogisticRegression
            model = LogisticRegression(random_state=0)
```

```
In [444...] from sklearn.model_selection import train_test_split
            train_x,test_x,train_y,test_y = train_test_split(X,y)
```

```
In [445...] model.fit(train_x,train_y)
```

```
Out[445...] LogisticRegression(random_state=0)
```

```
In [446...] y_predict = model.predict(test_x)
            y_predict[0:10]
```

```
Out[446...] array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

```
In [447...] from sklearn.metrics import mean_absolute_error
            mae = mean_absolute_error(test_y,y_predict)
            mae
```

```
Out[447...] 0.22150624244865083
```

```
In [448...] from sklearn.metrics import accuracy_score
            acs = accuracy_score(test_y,y_predict)
            acs
```

```
Out[448...] 0.7784937575513492
```

```
In [ ]:
```

## Decision Tree Classification

```
In [449...] from sklearn.tree import DecisionTreeClassifier
            model_dtc = DecisionTreeClassifier(random_state=0)
```

```
model_dtc
```

```
Out[449... DecisionTreeClassifier(random_state=0)
```

```
In [450... model_dtc.fit(train_x,train_y)
```

```
Out[450... DecisionTreeClassifier(random_state=0)
```

```
In [451... y_pred_dtc = model_dtc.predict(test_x)
y_pred_dtc
```

```
Out[451... array([0, 0, 0, ..., 0, 1, 1], dtype=int64)
```

```
In [452... mae_dtc = mean_absolute_error(test_y,y_pred_dtc)
mae_dtc
```

```
Out[452... 0.25251711639146196
```

```
In [453... acs_dtc = accuracy_score(test_y,y_pred_dtc)
acs_dtc
```

```
Out[453... 0.7474828836085381
```

```
In [ ]:
```

## Randomforest classifier

```
In [454... from sklearn.ensemble import RandomForestClassifier
model_rfc = RandomForestClassifier(random_state=1)
```

```
In [455... model_rfc.fit(train_x,train_y)
```

```
Out[455... RandomForestClassifier(random_state=1)
```

```
In [456... y_pred_rfc = model.predict(test_x)
```

```
In [457... mae_rfc = mean_absolute_error(test_y,y_pred_rfc)
mae_rfc
```

```
Out[457... 0.22150624244865083
```

```
In [458... acs_rfc = accuracy_score(test_y,y_pred_rfc)
acs_rfc
```

```
Out[458... 0.7784937575513492
```

## Support Vector Machine

```
In [459... from sklearn.svm import SVC  
model_svm = SVC()
```

```
In [460... model_svm.fit(train_x,train_y)
```

Out[460... SVC()

```
In [461... y_pred_svm = model_svm.predict(test_x)
```

```
In [462... mae_svm = mean_absolute_error(test_y,y_pred_svm)  
mae_svm
```

Out[462... 0.21506242448650825

```
In [463... acs_svm = accuracy_score(test_y,y_pred_svm)  
acs_svm
```

Out[463... 0.7849375755134917

## KNN

```
In [464... from sklearn.neighbors import KNeighborsClassifier  
model_knc = KNeighborsClassifier()
```

```
In [465... model_knc.fit(train_x,train_y)
```

Out[465... KNeighborsClassifier()

```
In [466... y_pred_knc = model_knc.predict(test_x)
```

```
In [467... mae_knc = mean_absolute_error(test_y,y_pred_knc)  
mae_knc
```

Out[467... 0.2400322190898107

## Naive Bayes

```
In [468... from sklearn.naive_bayes import BernoulliNB  
model_nb = BernoulliNB()
```

```
In [469... model_nb.fit(train_x,train_y)
```

Out[469... BernoulliNB()

```
In [470... y_p_nb = model_nb.predict(test_x)
```

```
In [471...
```

```
mae_nb = mean_absolute_error(test_y,y_p_nb)
mae_nb
```

Out[471... 0.21506242448650825

In [ ]:

In [483...

```
ddd = pd.DataFrame({
    'Naive Bayes': (1-mae_nb)*100,
    'KNN':(1-mae_knc)*100,
    'SVM':acs_svm*100,
    'Random_forest':acs_rfc*100,
    'DecisionTree':acs_dtc*100,
    "LogisticRegression":acs*100

},index=['Accuracy'])
ddd.T
```

Out[483...

	Accuracy
Naive Bayes	78.493758
KNN	75.996778
SVM	78.493758
Random_forest	77.849376
DecisionTree	74.748288
LogisticRegression	77.849376

We are getting maximum accuracy from Naive Bayes And SVM

In [ ]:

In [ ]: