Case Study: Diabetes Dataset

Problem Statement

[https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)](https://www.kaggle.com/uciml/pima-indians-diabetes-database)

Content

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Acknowledgements

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261--265). IEEE Computer Society Press.

Inspiration

Can you build a machine learning model to accurately predict whether or not the patients in the dataset have diabetes or not?

Import the Libraries

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

Q1. Load the data

In [2]:
```
d=pd.read_csv(r'C:\Users\lxm\Desktop\diabetes.csv')
```

In [3]:
```
d
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 9 columns

Q2. Print 10 samples from the dataset

In [6]:
```
d.sample(10)
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
401	6	137	61	0	0	24.2	0.151
141	5	106	82	30	0	39.5	0.286
661	1	199	76	43	0	42.9	1.394
248	9	124	70	33	402	35.4	0.282
289	5	108	72	43	75	36.1	0.263
105	1	126	56	29	152	28.7	0.801
537	0	57	60	0	0	21.7	0.735
164	0	131	88	0	0	31.6	0.743
719	5	97	76	27	0	35.6	0.378
512	9	91	68	0	0	24.2	0.200

In [7]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Q3 Explore data types and check data shape

In [161]:
```
d.dtypes
```

Out[161]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                     float64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [9]:
```
d.shape
```

Out[9]:
```
(768, 9)
```

In [11]:
```
d.isnull().sum()
```

Out[11]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Q4 Replace all the invalid 0s in the column (

based on your understanding of the data) with the median of the same column value accordingly.

In [46]:
```
d.median()
```

Out[46]:
```
Pregnancies                   3.0000
Glucose                     117.0000
BloodPressure                72.0000
SkinThickness                23.0000
Insulin                     127.2500
BMI                          32.0000
DiabetesPedigreeFunction      0.3725
Age                          29.0000
Outcome                       0.0000
dtype: float64
```

In [34]:
```
d['Pregnancies'].replace(0,3,inplace=True)
```

In [36]:
```
d['BloodPressure'].replace(0,72,inplace=True)
```

In [38]:
```
d['SkinThickness'].replace(0,23,inplace=True)
```

In [39]:
```
d['Insulin'].replace(0,127.25,inplace=True)
```

In [44]:
```
d['BMI'].replace(0,32,inplace=True)
```

In [49]:
```
d
```

Out[49]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	127.25	33.6	0.627
1	1	85	66	29	127.25	26.6	0.351
2	8	183	64	23	127.25	23.3	0.672
3	1	89	66	23	94.00	28.1	0.167
4	3	137	40	35	168.00	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180.00	32.9	0.171
764	2	122	70	27	127.25	36.8	0.340
765	5	121	72	23	112.00	26.2	0.245
766	1	126	60	23	127.25	30.1	0.349
767	1	93	70	31	127.25	30.4	0.315

768 rows × 9 columns

Q5 Do the descriptive statistics of the data

In [68]:
```
d.describe().T
```

Out[68]:

	count	mean	std	min	25%	50%	75%	
Pregnancies	768.0	4.278646	3.021516	1.000	2.00000	3.0000	6.00000	1
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	19
BloodPressure	768.0	72.386719	12.096642	24.000	64.00000	72.0000	80.00000	12
SkinThickness	768.0	27.334635	9.229014	7.000	23.00000	23.0000	32.00000	9
Insulin	768.0	141.767578	86.191132	14.000	121.50000	127.2500	127.43750	84
BMI	768.0	32.450911	6.875366	18.200	27.50000	32.0000	36.60000	6
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	8
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	

In [61]:
```
d[d['Outcome']==1].mean()
```

Out[61]:
```
Pregnancies                   5.291045
Glucose                     141.257463
BloodPressure                75.123134
SkinThickness                29.716418
Insulin                     165.860075
BMI                          35.381343
DiabetesPedigreeFunction      0.550500
Age                          37.067164
Outcome                       1.000000
dtype: float64
```

Q6 Perform EDA on the Dataset and check Class column

In [67]:
```
d['Outcome'].value_counts()
```

Out[67]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [65]:
```
d.corr()
```

Out[65]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BM
Pregnancies	1.000000	0.153834	0.247530	0.060706	0.033496	0.08054
Glucose	0.153834	1.000000	0.217870	0.158027	0.409603	0.21880
BloodPressure	0.247530	0.217870	1.000000	0.147809	0.047384	0.28113
SkinThickness	0.060706	0.158027	0.147809	1.000000	0.180969	0.54695
Insulin	0.033496	0.409603	0.047384	0.180969	1.000000	0.17956
BMI	0.080540	0.218806	0.281132	0.546951	0.179568	1.00000
DiabetesPedigreeFunction	-0.016151	0.137337	-0.002378	0.142977	0.124613	0.15350
Age	0.538169	0.263514	0.324915	0.054514	0.100084	0.02574
Outcome	0.245466	0.466581	0.165723	0.189065	0.204779	0.31224

In [141]:
```
pd.crosstab(d['Outcome'], d['Age'])
```

Out[141]:

Age	21	22	23	24	25	26	27	28	29	30	...	63	64	65	66	67	68	69	70	72	81
Outcome																					
0	58	61	31	38	34	25	24	25	16	15	...	4	1	3	2	2	1	2	0	1	1
1	5	11	7	8	14	8	8	10	13	6	...	0	0	0	2	1	0	0	1	0	0

2 rows × 52 columns

In [75]:
```
d[d['Age']==d['Age'].max()]
```

Out[75]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
459	9	134	74	33	60.0	25.9	0.46

In [80]:
```
d.groupby('Outcome')['Age'].sum().sort_values(ascending=False).head(10)
```

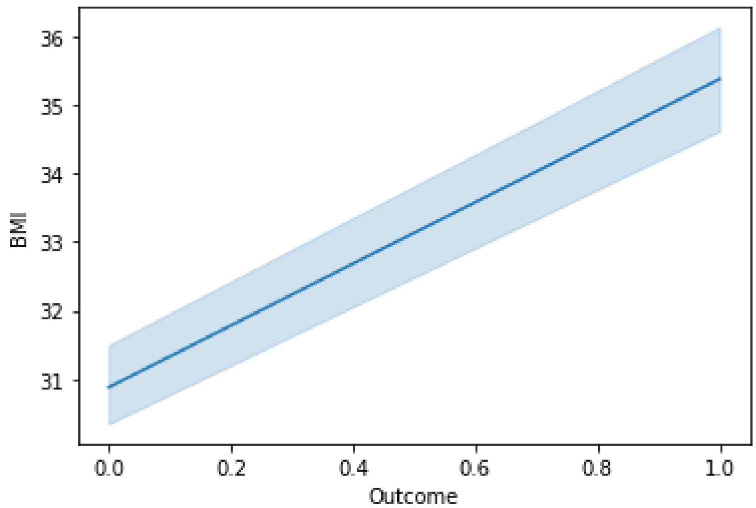
Out[80]:
```
Outcome
0    15595
1     9934
Name: Age, dtype: int64
```

In [78]:
```
d.groupby('Outcome')['Pregnancies'].sum().sort_values(ascending=False).head(10)
```

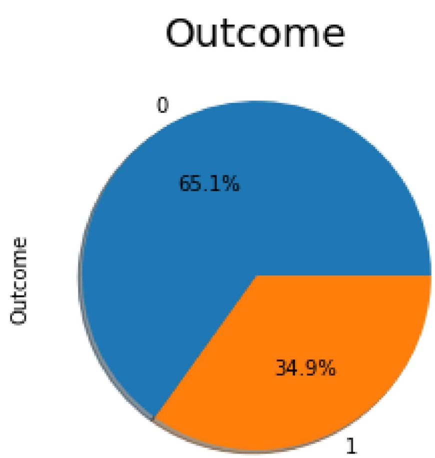
Out[78]:
```
Outcome
0    1868
1    1418
Name: Pregnancies, dtype: int64
```

In [81]:

In [84]:
```
sns.lineplot(x='Outcome',y='BMI',data=d)
plt.show()
```

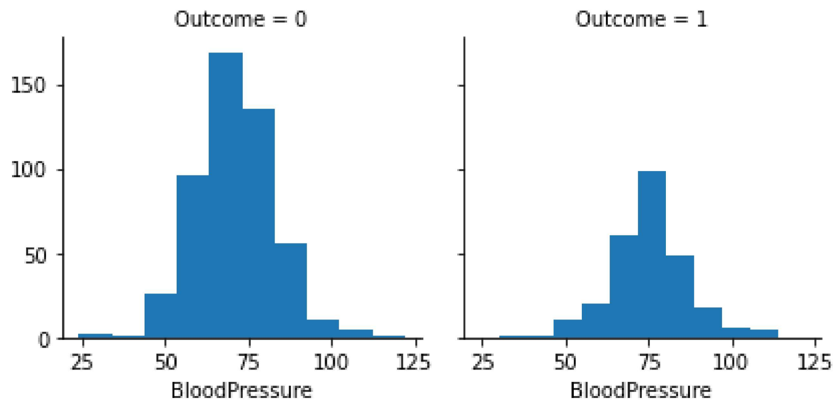


In [86]:
```
plt.title('Outcome',fontsize=20)
d['Outcome'].value_counts().plot.pie(autopct='%1.1f%%',shadow=True)
plt.show()
```



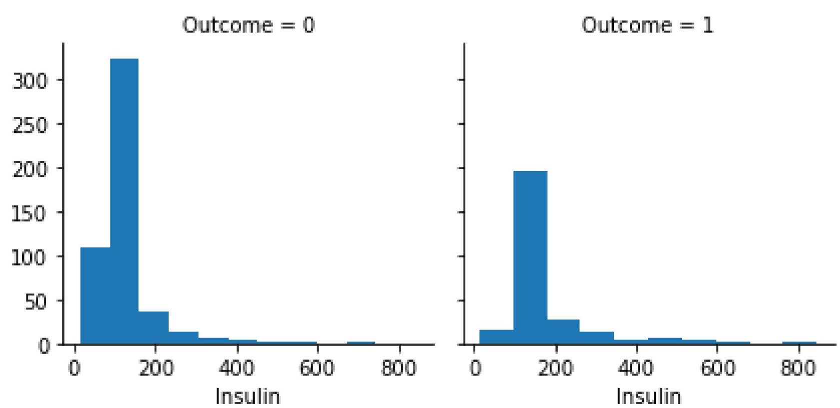
In [87]:
```
g=sns.FacetGrid(d,col='Outcome')
g.map(plt.hist,"BloodPressure")
```

Out[87]: <seaborn.axisgrid.FacetGrid at 0x17c3325f910>



In [88]:
```
g=sns.FacetGrid(d,col='Outcome')
g.map(plt.hist,"Insulin")
```

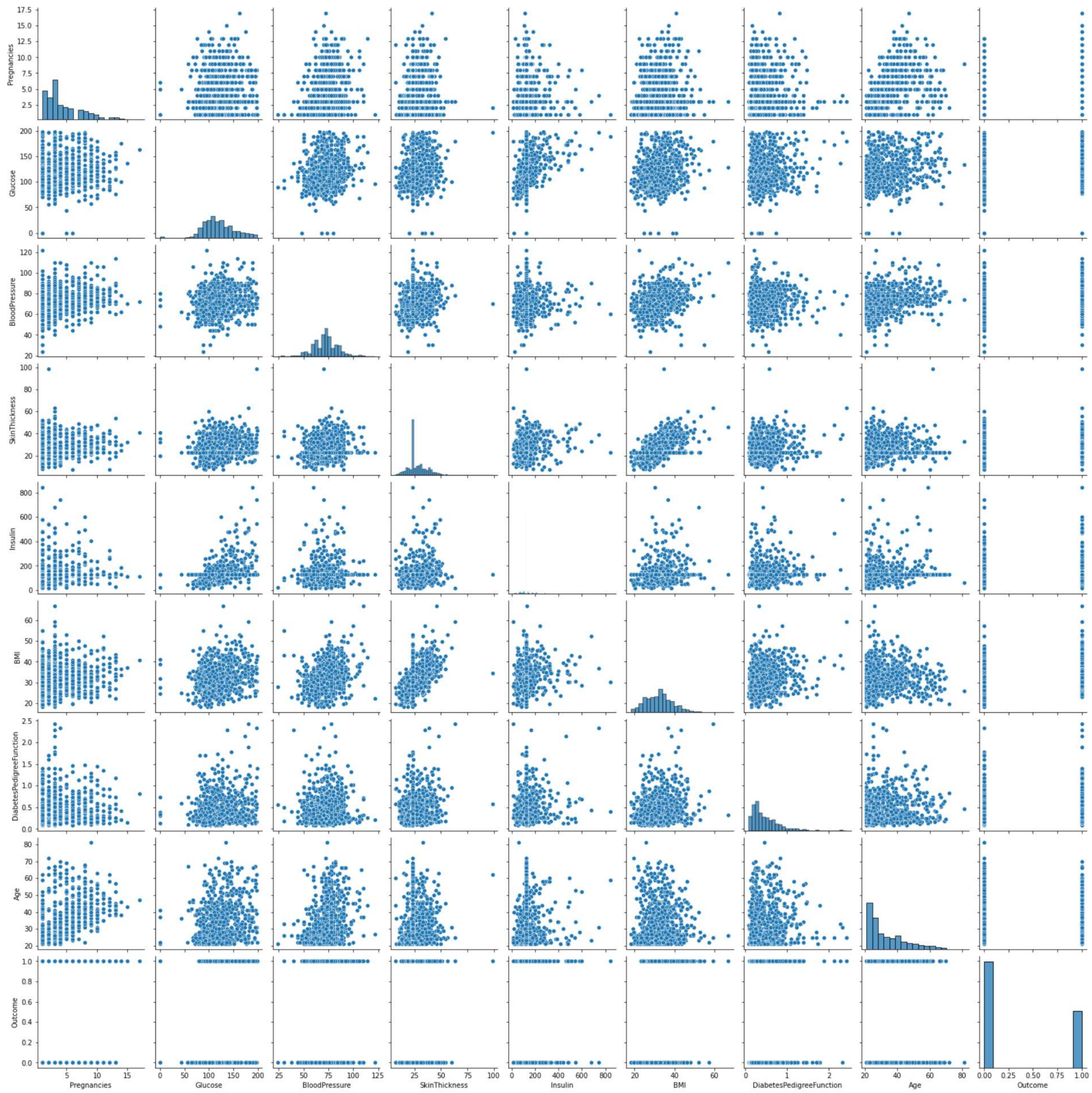
Out[88]: <seaborn.axisgrid.FacetGrid at 0x17c33264e20>



Q7. Use pairplots and correlation method to observe the relationship between different variables and state your insights.

In [89]:
```
sns.pairplot(d)
```

Out[89]: <seaborn.axisgrid.PairGrid at 0x17c33307e80>



In [142]:
```
plt.figure(figsize=(16,16))
corr=d.corr()
sns.heatmap(corr,annot=True)
```

Out[142]: <AxesSubplot:>

Conclusion:

There is 65% chance that people are suffer from the diabities and 35% chance that they do not have diabities.

On an average 5% ladies are suffer from diabities during their pregnancy. Also average 37 year of age, people suffered from the diabities.

As BMI rate getting increased the chance of diabities are increase.

There is a positive relation between Pregnancies and the skin thickness.

There is a low chance of diabities in insulin levels.

Let's make copy of your data

In [91]:
```
d1=d.copy()
```

In [92]:
```
d1
```

Out[92]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 9 columns

Q8 Split data into training and test set in the ratio of 70:30 (Training:Test)

In [93]:
```
x=d1.drop('Outcome',axis=1)
y=d1['Outcome']
```

In [94]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [95]:
```
x_train.shape,y_train.shape
```

Out[95]:
```
((537, 8), (537,))
```

In [96]:
```
y_test.shape, y_test.shape
```

Out[96]:
```
((231,), (231,))
```

Q9 Perfrom different ML Models and

compareresult from all the models

1) Linear Regression

In [97]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

In [98]:
```
lr.fit(x_train_transformed,y_train)
```

Out[98]:
```
LinearRegression()
```

In [119]:
```
print("Test Score",lr.score(x_test,y_test))
```

```
Test Score 0.8181818181818182
```

In [118]:
```
print("Training Score",lr.score(x_train,y_train))
```

```
Training Score 0.7653631284916201
```

2) Logistic Regression

In [101]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

In [117]:
```
lr.fit(x_train,y_train)
print("Training Acuuracy",lr.score(x_train,y_train))
print("Test Acuuracy",lr.score(x_test,y_test))
```

```
Training Acuuracy 0.7653631284916201
Test Acuuracy 0.8181818181818182
```

3) KNeighborsClassifier

In [103]:
```
import warnings
warnings.filterwarnings('ignore',module='sklearn')

from sklearn.preprocessing import MinMaxScaler

msc=MinMaxScaler()

data=pd.DataFrame(msc.fit_transform(d1),columns=d1.columns)
```

In [104]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [105]:
```
res={}
for k in range(3,20):
    Knn= KNeighborsClassifier(n_neighbors=k)
    Knn.fit(x_train,y_train)
    print("Training Accuracy for k={} is {}:".format(k,Knn.score(x_train,y_train)
    print("Testing Accuracy for K={} is {}:".format(k,Knn.score(x_test,y_test)))
    res[k]=Knn.score(x_test,y_test)
```

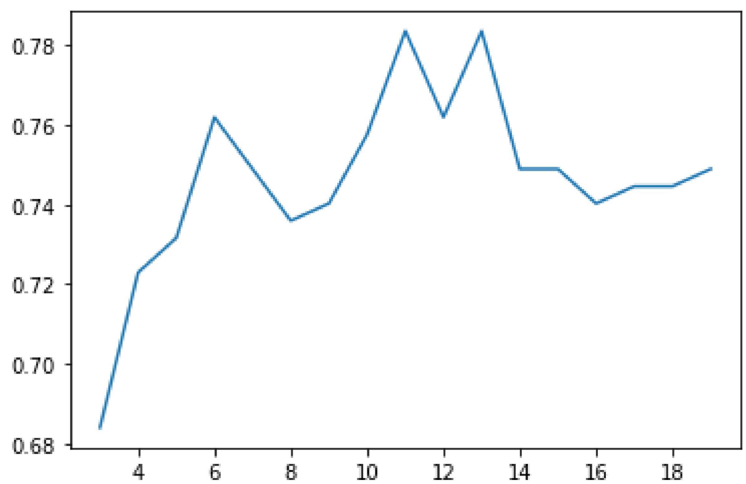
```
Training Accuracy for k=3 is 0.8491620111731844:
Testing Accuracy for K=3 is 0.683982683982684:
Training Accuracy for k=4 is 0.7914338919925512:
Testing Accuracy for K=4 is 0.7229437229437229:
Training Accuracy for k=5 is 0.8119180633147114:
Testing Accuracy for K=5 is 0.7316017316017316:
Training Accuracy for k=6 is 0.770949720670391:
Testing Accuracy for K=6 is 0.7619047619047619:
Training Accuracy for k=7 is 0.7914338919925512:
Testing Accuracy for K=7 is 0.7489177489177489:
Training Accuracy for k=8 is 0.7802607076350093:
Testing Accuracy for K=8 is 0.7359307359307359:
Training Accuracy for k=9 is 0.7802607076350093:
Testing Accuracy for K=9 is 0.7402597402597403:
Training Accuracy for k=10 is 0.7728119180633147:
Testing Accuracy for K=10 is 0.7575757575757576:
Training Accuracy for k=11 is 0.7597765363128491:
Testing Accuracy for K=11 is 0.7835497835497836:
Training Accuracy for k=12 is 0.7560521415270018:
Testing Accuracy for K=12 is 0.7619047619047619:
Training Accuracy for k=13 is 0.7728119180633147:
Testing Accuracy for K=13 is 0.7835497835497836:
Training Accuracy for k=14 is 0.7635009310986964:
Testing Accuracy for K=14 is 0.7489177489177489:
Training Accuracy for k=15 is 0.770949720670391:
Testing Accuracy for K=15 is 0.7489177489177489:
Training Accuracy for k=16 is 0.7746741154562383:
Testing Accuracy for K=16 is 0.7402597402597403:
Training Accuracy for k=17 is 0.7635009310986964:
Testing Accuracy for K=17 is 0.7445887445887446:
Training Accuracy for k=18 is 0.7672253258845437:
Testing Accuracy for K=18 is 0.7445887445887446:
Training Accuracy for k=19 is 0.7690875232774674:
Testing Accuracy for K=19 is 0.7489177489177489:
```

In [106]:
```
res
```

Out[106]:
```
{3: 0.683982683982684,
 4: 0.7229437229437229,
 5: 0.7316017316017316,
 6: 0.7619047619047619,
 7: 0.7489177489177489,
 8: 0.7359307359307359,
 9: 0.7402597402597403,
 10: 0.7575757575757576,
 11: 0.7835497835497836,
 12: 0.7619047619047619,
 13: 0.7835497835497836,
 14: 0.7489177489177489,
 15: 0.7489177489177489,
 16: 0.7402597402597403,
 17: 0.7445887445887446,
 18: 0.7445887445887446,
 19: 0.7489177489177489}
```

In [107]:
```
plt.plot(res.keys(),res.values())
```

Out[107]:
```
[<matplotlib.lines.Line2D at 0x2497121e0d0>]
```



In [108]:
```
Knn= KNeighborsClassifier(n_neighbors=19)
Knn.fit(x_train,y_train)
```

Out[108]:
```
KNeighborsClassifier(n_neighbors=19)
```

In [116]:
```
y_pred=Knn.predict(x_test)
y_pred
```

Out[116]:
```
array([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, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

In [110]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.76      0.83       183
           1       0.44      0.71      0.54        48

    accuracy                           0.75       231
   macro avg       0.67      0.73      0.68       231
weighted avg       0.81      0.75      0.77       231
```

DecisionTreeClassifier

In [115]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
print("Training Acuuracy",dt.score(x_train,y_train))
print("Test Acuuracy",dt.score(x_test,y_test))
predict=dt.predict(x_test)
```

```
Training Acuuracy 1.0
Test Acuuracy 0.6883116883116883
```

5) Gausian naive bayes

In [51]:
```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
```

In [52]:
```
nb.fit(x_train,y_train)
print("Training Acuuracy",nb.score(x_train,y_train))
print("Test Acuuracy",nb.score(x_test,y_test))
```

```
Training Acuuracy 0.7635009310986964
Test Acuuracy 0.7142857142857143
```

6) SVC

In [53]:
```
from sklearn.svm import SVC
```

In [54]:
```
svc=SVC(kernel='rbf',gamma=2,C=1)
svc.fit(x_train,y_train)
print(svc.score(x_train,y_train))
print(svc.score(x_test,y_test))
```

```
1.0
0.6493506493506493
```

In [55]:
```
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
adc=AdaBoostClassifier(LogisticRegression(), n_estimators=100, learning_rate=1)
gbc= GradientBoostingClassifier()
```

7) AdaBoostClassifier

In [60]:
```
adc.fit(x_train,y_train)
print("Training Acuuracy",adc.score(x_train,y_train))
print("Test Acuuracy",adc.score(x_test,y_test))
```

```
Training Acuuracy 0.7858472998137802
Test Acuuracy 0.7489177489177489
```

8) GradientBoostingClassifier

In [59]:
```
gbc.fit(x_train,y_train)
print("Training Acuuracy",gbc.score(x_train,y_train))
print("Test Acuuracy",gbc.score(x_test,y_test))
```

```
Training Acuuracy 0.9478584729981379
Test Acuuracy 0.7532467532467533
```

9) Bagging

In [57]:
```
from sklearn.ensemble import RandomForestClassifier
rfc= RandomForestClassifier(max_depth=10, max_features=4)
```

In [58]:
```
rfc.fit(x_train,y_train)
print("Training Acuuracy",rfc.score(x_train,y_train))
print("Test Acuuracy",rfc.score(x_test,y_test))
```

```
Training Acuuracy 0.9981378026070763
Test Acuuracy 0.7359307359307359
```

Q10 Evaluate model using different metrices

In [82]:
```
# RandomForestClassifer

predict=rfc.predict(x_test)
print(confusion_matrix(predict,y_test))
```

```
[[145  11]
 [ 14  61]]
```

In [83]:
```
# GradientBoostingClassifer

predict=gbc.predict(x_test)
print(confusion_matrix(predict,y_test))
```

```
[[144  13]
 [ 15  59]]
```

In [113]:
```
# DecisionTreeClassifier

predict=dt.predict(x_test)
print(confusion_matrix(predict,y_test))
print(classification_report(predict,y_test))
```

```
[[121  36]
 [ 32  42]]
              precision    recall  f1-score   support

           0       0.79      0.77      0.78       157
           1       0.54      0.57      0.55        74

    accuracy                           0.71       231
   macro avg       0.66      0.67      0.67       231
weighted avg       0.71      0.71      0.71       231
```

In [88]:
```
#AddaboostClassifier

predict=adc.predict(x_test)
print(confusion_matrix(predict,y_test))
```

```
[[141  31]
 [ 18  41]]
```

In [120]:
```
# LinearRegression

predict=lr.predict(x_test)
print(confusion_matrix(predict,y_test))
```

```
[[142  31]
 [ 11  47]]
```

Conclusion

We are fitting the Linear Regression, Logistic Regression, KNN, SKV,GausianNaiveBaise, RandomForestClassification, AddaBoostClassifer, GradientBoostingClassifier into the train and testing data.

After using the model we get the training and testing accuracy more than 65 percentage in each model.

Hence the Linear Regression model gives the accuracy more than others: Training Score 0.76 and Test Score 0.81. Also GradiantBoostingClassifier Training Acuuracy 0.94 and Test Acuuracy 0.75 which is good enough.

RandomForestClassifier and Linear Regression makes the confusion matrix with less error as compare to other models.