The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

Import the necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
## Read the dataset and explore the required task
df = pd.read_csv("health care diabetes.csv")
```

```
df.shape
```

```
     (768, 9)
```

```
# Showing the columns and the outcome value
columns=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI",
X=df[columns]
y=df["Outcome"]
X.shape,y.shape
```

```
     ((768, 8), (768,))
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

Insights from Descriptive Analysis

There is 768 observations of 9 variable. Independent variables are Pregnencies , Glucose, BloodPressure, Insulin, BMI and DiabetesPedigree Function. Age is Outcome Variable. Average Age of Patients are 33.24 with minimum being 21 and maximum 81. Avg. value of independent variables are Preg = 3.845052,Glucose = 120.894531, BP = 69.105469, ST=20.536458, Insulin = 79.799479, BMI = 31.992578 DPF = 0.471876 . Variation in variables can be easily observed from table below :->

```
## 0 = Non -Diabetic
## 1 = Diabetic
```

```
df.isnull().sum()
```

```
    Pregnancies                 0
    Glucose                     0
    BloodPressure               0
    SkinThickness               0
    Insulin                     0
    BMI                         0
    DiabetesPedigreeFunction    0
    Age                         0
    Outcome                     0
    dtype: int64
```

```
df.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
```

*Treating Missing Values and Analysing Distribution of Data *

zero is not accepted in these coloumn so that it is replaced by mean

```
zero_not_accepted =['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
for column in zero_not_accepted:
  df[column]= df[column].replace(0,np.NaN)
  mean = int(df[column].mean(skipna=True))
  df[column]= df[column].replace(np.NaN, mean)
```

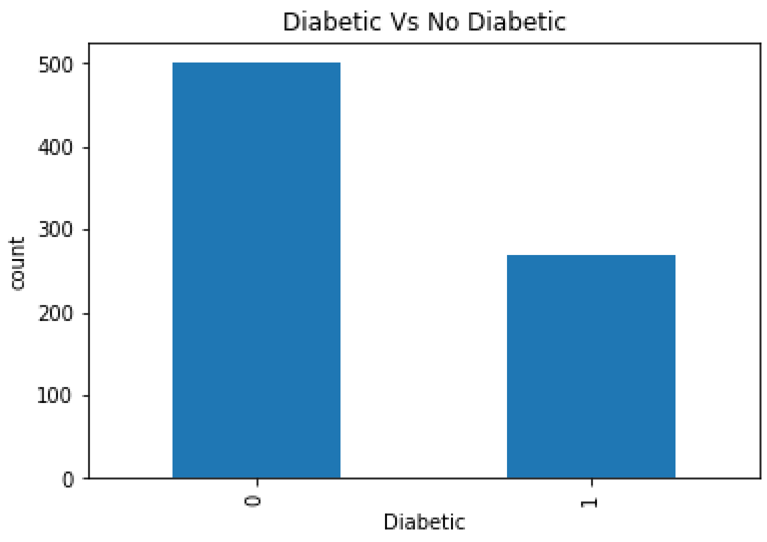
checking the dataset is balanced or not

```
df['Outcome'].value_counts()
```

```
     0    500
     1    268
     Name: Outcome, dtype: int64
```

```
df['Outcome'].value_counts().plot.bar(title = "Diabetic Vs No Diabetic", xlabel='Diabetic'
```

```
     <matplotlib.axes._subplots.AxesSubplot at 0x7fa98a6ab610>
```



By this plot we can easily see that the diabetic and non diabetic data is not evenly distributed so

```
df.shape
```

```
(768, 9)
```

```
positive = df[df['Outcome']==1]
positive.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
0	6	148.0	72.0	35.0	155.0	33.6	
2	8	183.0	64.0	29.0	155.0	23.3	
4	0	137.0	40.0	35.0	168.0	43.1	
6	3	78.0	50.0	32.0	88.0	31.0	
8	2	197.0	70.0	45.0	543.0	30.5	
9	8	125.0	96.0	29.0	155.0	32.0	
11	10	168.0	74.0	29.0	155.0	38.0	
13	1	189.0	60.0	23.0	846.0	30.1	
14	5	166.0	72.0	19.0	175.0	25.8	
15	7	100.0	72.0	29.0	155.0	30.0	

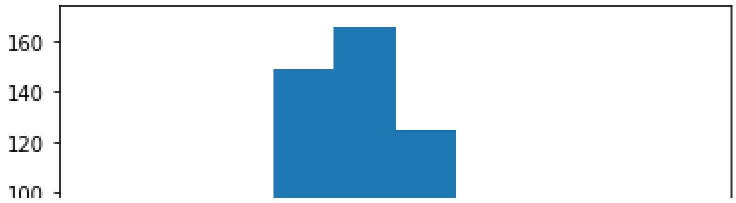
*Visually explore these variables using histograms. *

```
df['Glucose'].value_counts().head()
```

```
100.0    17
99.0     17
111.0    14
125.0    14
106.0    14
Name: Glucose, dtype: int64
```

```
plt.hist(df['Glucose'])
```

```
(array([  4.,  19.,  87., 149., 166., 125.,  88.,  54.,  44.,  32.]),
 array([ 44. ,  59.5,  75. ,  90.5, 106. , 121.5, 137. , 152.5, 168. ,
        183.5, 199. ]),
 <a list of 10 Patch objects>)
```

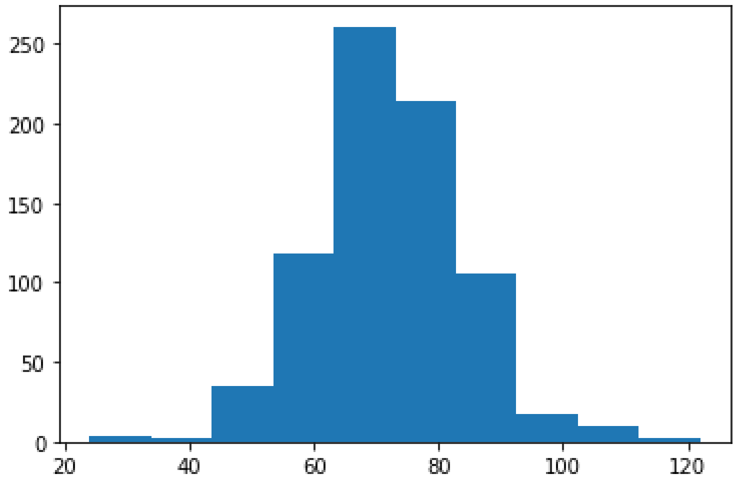


```
df['BloodPressure'].value_counts().head()
```

```
72.0    79
70.0    57
74.0    52
78.0    45
68.0    45
Name: BloodPressure, dtype: int64
```

```
plt.hist(df['BloodPressure'])
```

```
(array([  3.,   2.,  35., 118., 261., 214., 105.,  18.,  10.,   2.]),
 array([ 24. ,  33.8,  43.6,  53.4,  63.2,  73. ,  82.8,  92.6, 102.4,
        112.2, 122. ]),
 <a list of 10 Patch objects>)
```

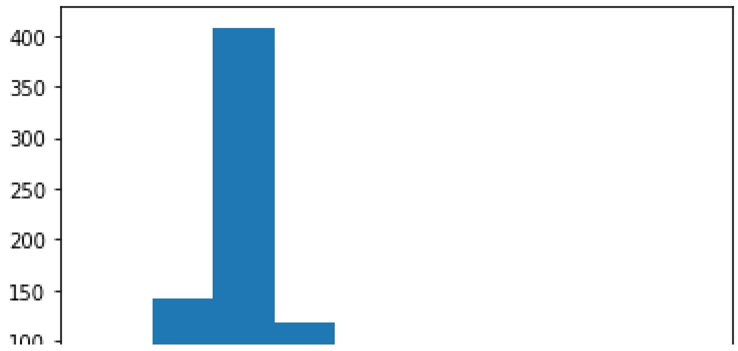


```
df['SkinThickness'].value_counts().head()
```

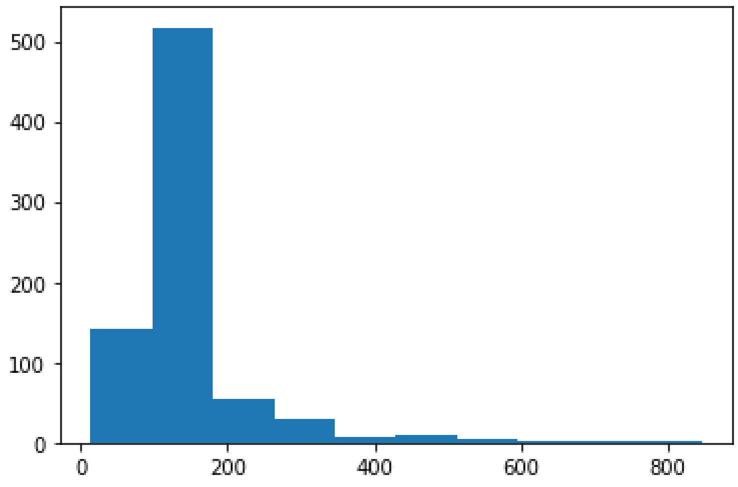
```
29.0    244
32.0     31
30.0     27
27.0     23
23.0     22
Name: SkinThickness, dtype: int64
```

```
plt.hist(df['SkinThickness'])
```

```
    (array([ 59., 141., 408., 118.,  36.,   4.,   1.,   0.,   0.,   1.]),
     array([ 7. , 16.2, 25.4, 34.6, 43.8, 53. , 62.2, 71.4, 80.6, 89.8, 99. ]),
     <a list of 10 Patch objects>)
```

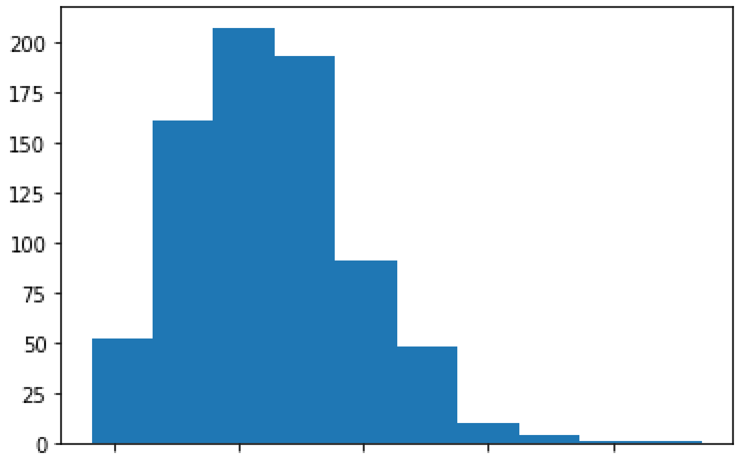


```
df['Insulin'].value_counts().head()

    155.0    378
    105.0     11
    130.0      9
    140.0      9
    120.0      8
    Name: Insulin, dtype: int64


plt.hist(df['Insulin'])

    (array([142., 517.,  55.,  29.,   7.,  10.,   4.,   1.,   2.,   1.]),
     array([ 14. ,  97.2, 180.4, 263.6, 346.8, 430. , 513.2, 596.4, 679.6,
            762.8, 846. ]),
     <a list of 10 Patch objects>)
```



```
df['BMI'].value_counts().head()

    32.0    24
    31.2    12
    31.6    12
    32.4    10
    33.3    10
    Name: BMI, dtype: int64


plt.hist(df['BMI'])
```

```
(array([ 52., 161., 207., 193.,  91.,  48.,  10.,   4.,   1.,   1.]),
 array([18.2 , 23.09, 27.98, 32.87, 37.76, 42.65, 47.54, 52.43, 57.32,
        62.21, 67.1 ]),
 <a list of 10 Patch objects>)
```



```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	
Glucose	768.0	121.682292	30.435999	44.000	99.75000	117.0000	14
BloodPressure	768.0	72.386719	12.096642	24.000	64.00000	72.0000	8
SkinThickness	768.0	29.108073	8.791221	7.000	25.00000	29.0000	3
Insulin	768.0	155.281250	85.021550	14.000	121.50000	155.0000	15
BMI	768.0	32.450911	6.875366	18.200	27.50000	32.0000	3
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	4
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	

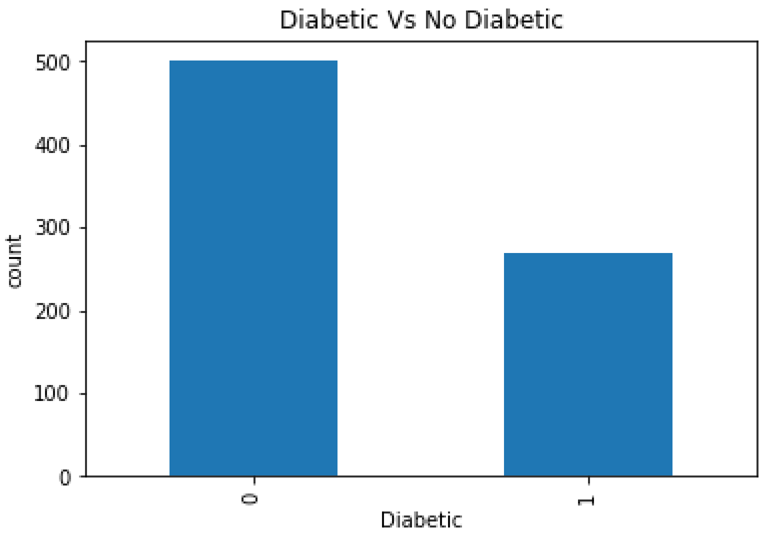
WEEK-2

Check the balance of the data by plotting the count of outcomes by their value

```
df['Outcome'].value_counts()
```

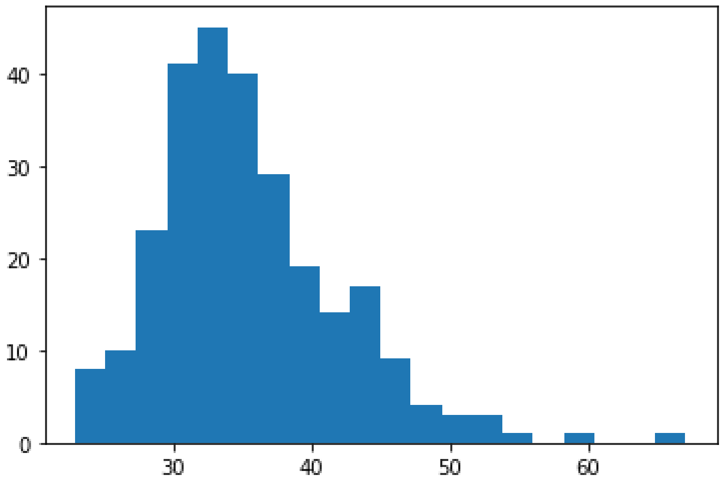
```
0    500
1    268
Name: Outcome, dtype: int64
```

```
df['Outcome'].value_counts().plot.bar(title = "Diabetic Vs No Diabetic", xlabel='Diabetic'
<matplotlib.axes._subplots.AxesSubplot at 0x7fa982d61c50>
```



```
plt.hist(positive['BMI'],histtype='stepfilled',bins=20)
```

```
(array([ 8., 10., 23., 41., 45., 40., 29., 19., 14., 17.,  9.,  4.,  3.,
         3.,  1.,  0.,  1.,  0.,  0.,  1.]),
 array([22.9 , 25.11, 27.32, 29.53, 31.74, 33.95, 36.16, 38.37, 40.58,
        42.79, 45.  , 47.21, 49.42, 51.63, 53.84, 56.05, 58.26, 60.47,
        62.68, 64.89, 67.1 ]),
 <a list of 1 Patch objects>)
```

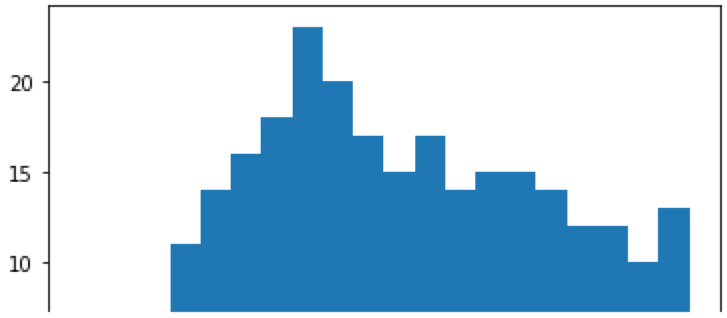


```
positive['BMI'].value_counts().head()
```

```
32.9    8
32.0    7
31.6    7
33.3    6
31.2    5
Name: BMI, dtype: int64
```

```
plt.hist(positive['Glucose'],histtype='stepfilled',bins=20)
```

```
(array([ 3.,  4.,  5., 11., 14., 16., 18., 23., 20., 17., 15., 17., 14.,
        15., 15., 14., 12., 12., 10., 13.]),
 array([ 78.  ,  84.05,  90.1 ,  96.15, 102.2 , 108.25, 114.3 , 120.35,
        126.4 , 132.45, 138.5 , 144.55, 150.6 , 156.65, 162.7 , 168.75,
        174.8 , 180.85, 186.9 , 192.95, 199.  ]),
 <a list of 1 Patch objects>)
```

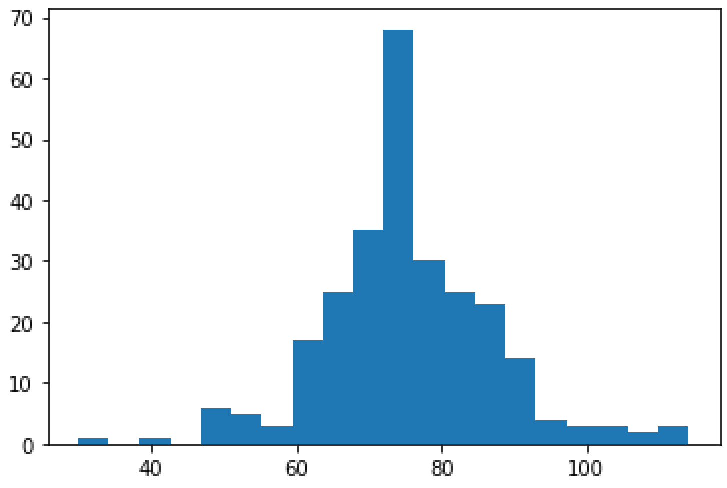


```
positive['Glucose'].value_counts().head()
```

```
125.0    7
158.0    6
129.0    6
115.0    6
128.0    6
Name: Glucose, dtype: int64
```

```
plt.hist(positive['BloodPressure'],histtype='stepfilled',bins=20)
```

```
(array([ 1.,  0.,  1.,  0.,  6.,  5.,  3., 17., 25., 35., 68., 30., 25.,
        23., 14.,  4.,  3.,  3.,  2.,  3.]),
 array([ 30. ,  34.2,  38.4,  42.6,  46.8,  51. ,  55.2,  59.4,  63.6,
         67.8,  72. ,  76.2,  80.4,  84.6,  88.8,  93. ,  97.2, 101.4,
        105.6, 109.8, 114. ]),
 <a list of 1 Patch objects>)
```

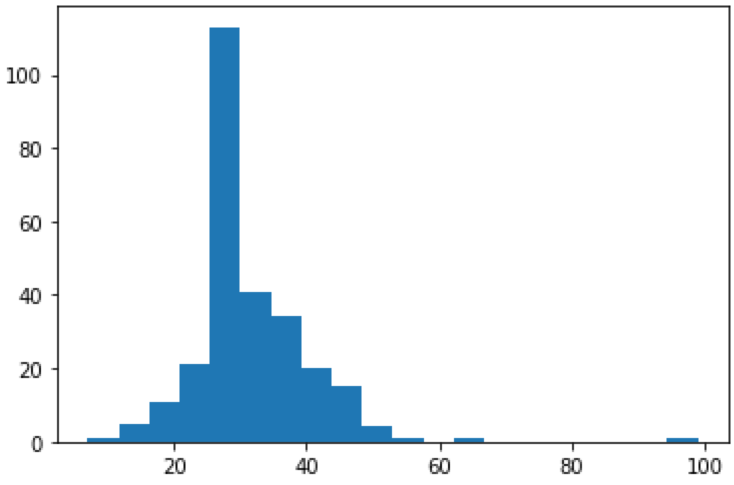


```
positive['BloodPressure'].value_counts().head()
```

```
72.0    32
70.0    23
76.0    18
78.0    17
74.0    17
Name: BloodPressure, dtype: int64
```

```
plt.hist(positive['SkinThickness'],histtype='stepfilled',bins=20)
```

```
(array([  1.,   5.,  11.,  21., 113.,  41.,  34.,  20.,  15.,   4.,   1.,
          0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   1.]),
 array([ 7. , 11.6, 16.2, 20.8, 25.4, 30. , 34.6, 39.2, 43.8, 48.4, 53. ,
        57.6, 62.2, 66.8, 71.4, 76. , 80.6, 85.2, 89.8, 94.4, 99. ]),
 <a list of 1 Patch objects>)
```

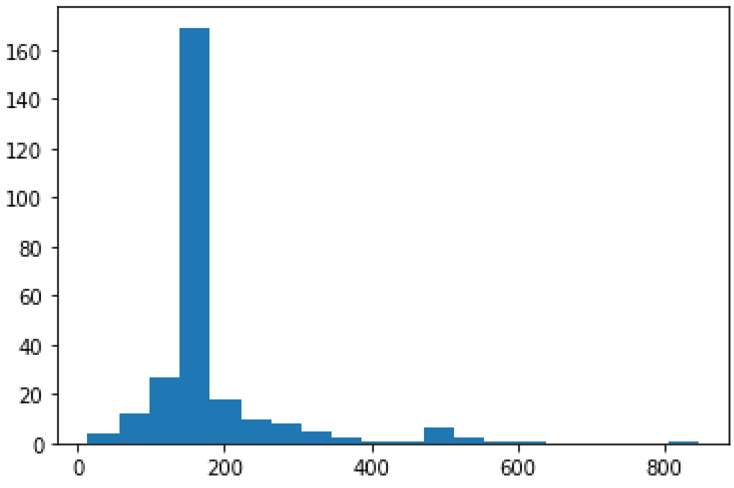


```
positive['SkinThickness'].value_counts().head()
```

```
29.0    95
32.0    14
30.0     9
33.0     9
36.0     8
Name: SkinThickness, dtype: int64
```

```
plt.hist(positive['Insulin'],histtype='stepfilled',bins=20)
```

```
(array([  4.,  12.,  27., 169.,  18.,  10.,   8.,   5.,   2.,   1.,   1.,
          6.,   2.,   1.,   1.,   0.,   0.,   0.,   0.,   1.]),
 array([ 14. ,  55.6,  97.2, 138.8, 180.4, 222. , 263.6, 305.2, 346.8,
        388.4, 430. , 471.6, 513.2, 554.8, 596.4, 638. , 679.6, 721.2,
        762.8, 804.4, 846. ]),
 <a list of 1 Patch objects>)
```



```
positive['Insulin'].value_counts().head()
```

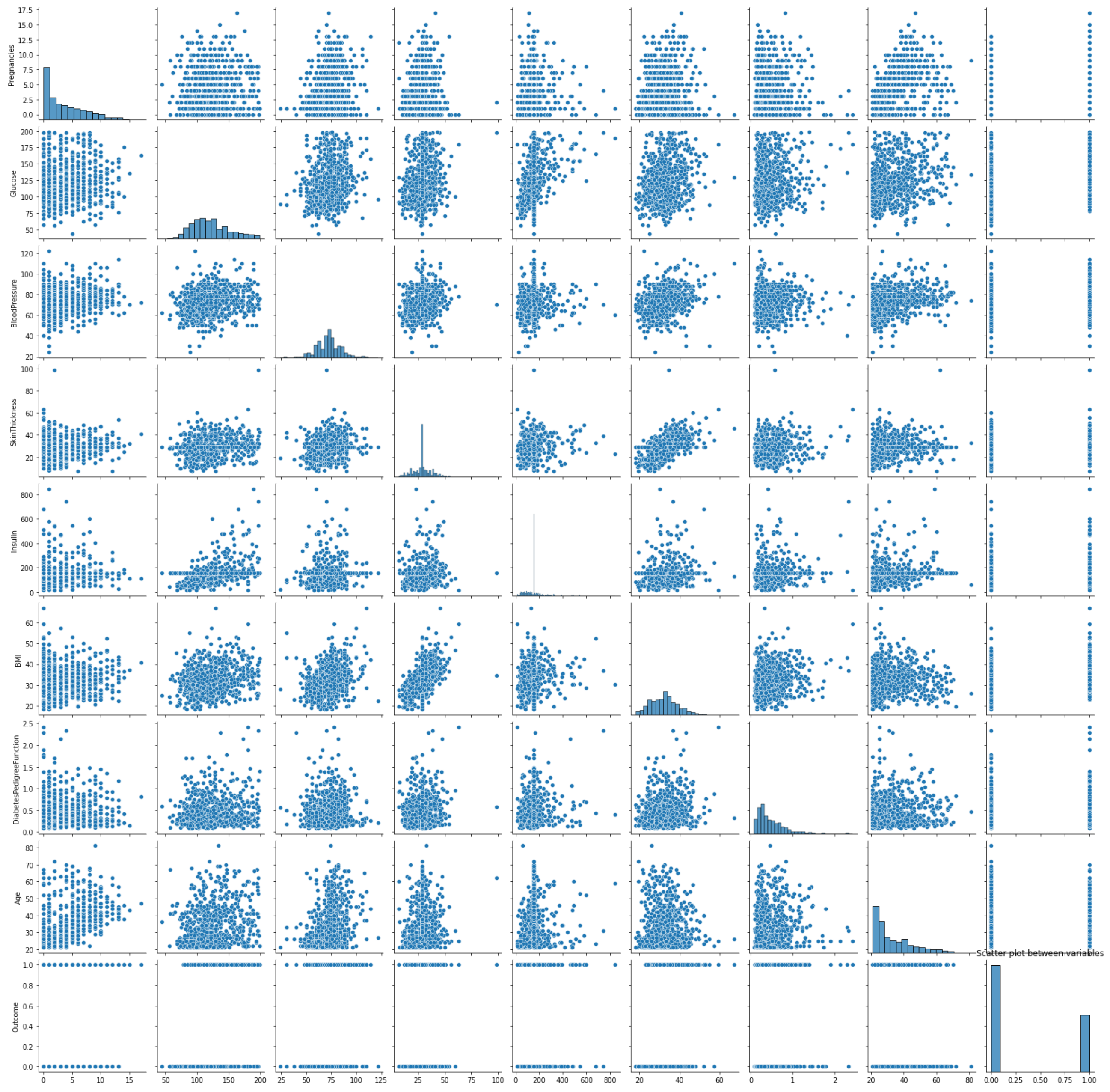
```
155.0    140
130.0      6
180.0      4
175.0      3
156.0      3
Name: Insulin, dtype: int64
```

** Satter Plot**

** Create scatter charts between the pair of variables to understand the relationships. Describe your findings.**

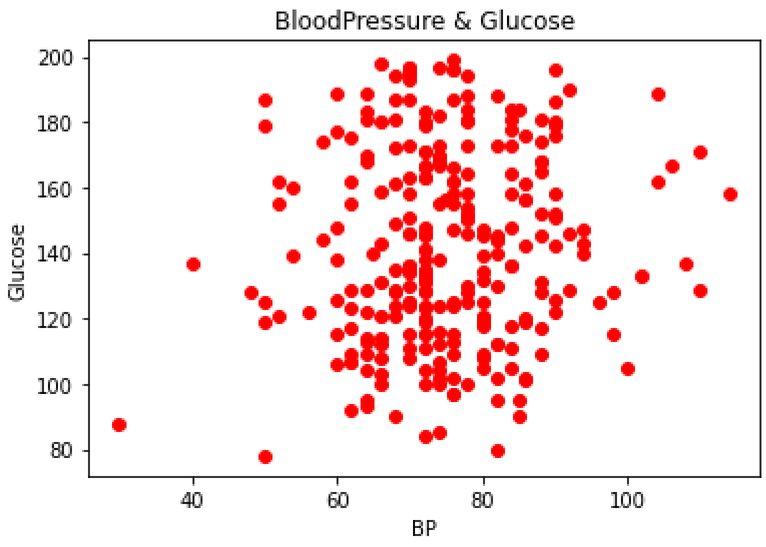
```
sns.pairplot(df)
plt.title('Scatter plot between variables')
```

```
Text(0.5, 1.0, 'Scatter plot between variables')
```

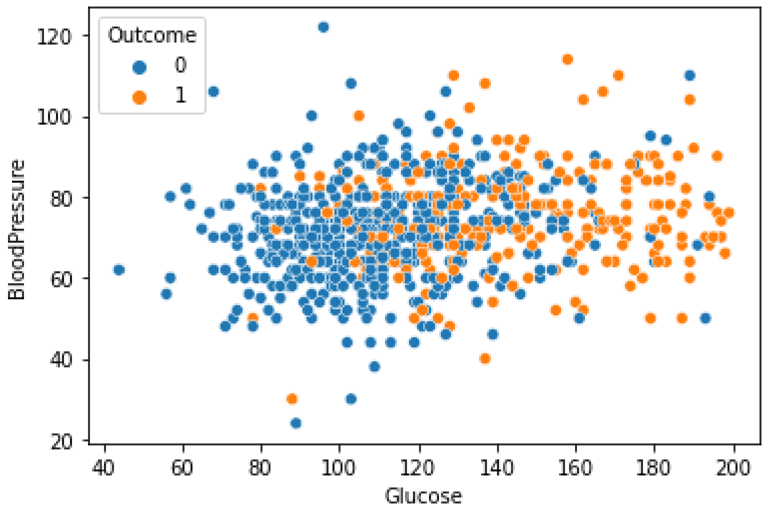


```
BloodPressure = positive['BloodPressure']
```

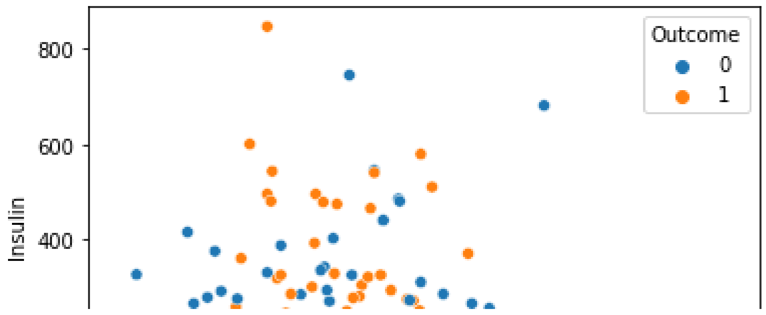
```
Glucose = positive['Glucose']
SkinThickness = positive['SkinThickness']
Insulin = positive['Insulin']
BMI = positive['BMI']

plt.scatter(BloodPressure,Glucose, color=['r'])
plt.xlabel('BP')
plt.ylabel('Glucose')
plt.title('BloodPressure & Glucose')
plt.show()
```



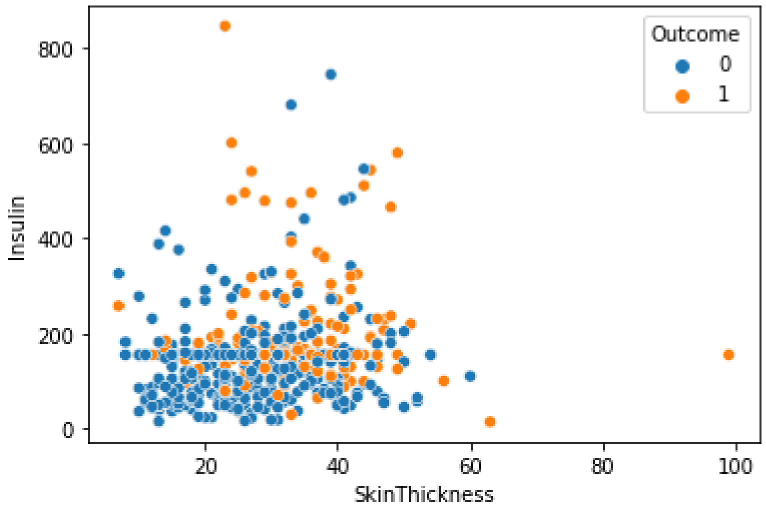
```
g =sns.scatterplot(x="Glucose", y="BloodPressure",
                   hue="Outcome",data=df)
```



```
B =sns.scatterplot(x="BMI", y="Insulin",
                   hue="Outcome",data=df)
```



```
S =sns.scatterplot(x="SkinThickness", y="Insulin",
                   hue="Outcome",data=df)
```



Double-click (or enter) to edit

Correlation Matrix

** Perform correlation analysis. Visually explore it using a heat map.**

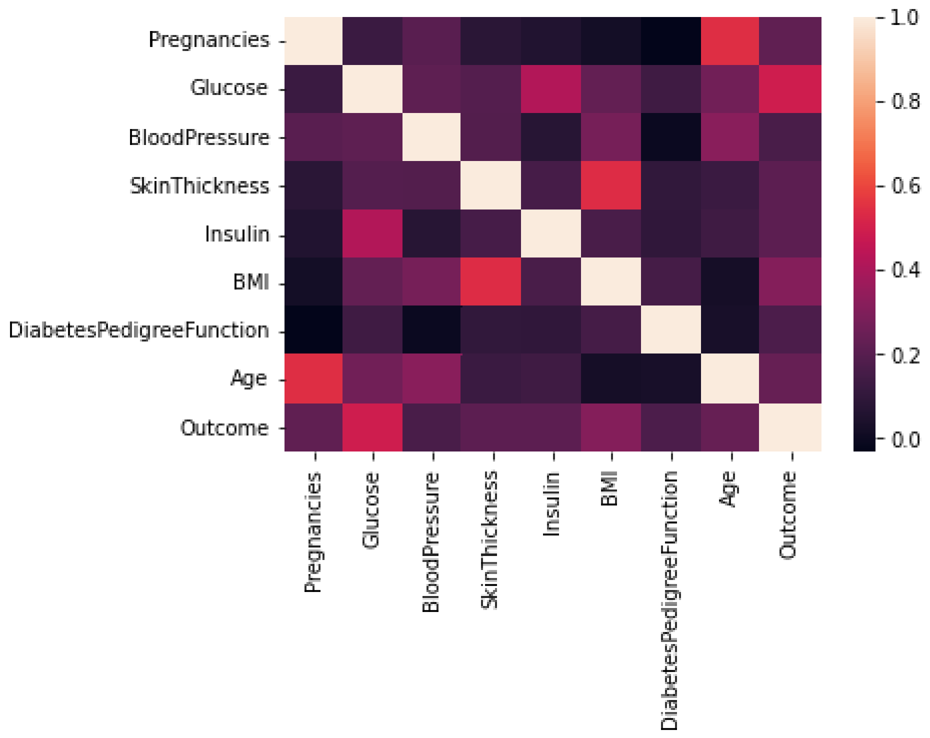
```
df.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insul
Pregnancies	1.000000	0.127957	0.208615	0.081770	0.0554
Glucose	0.127957	1.000000	0.218615	0.192677	0.4203
Insulin	0.055478	0.420391	0.072041	0.158132	1.0000

We can clearly see that Glucose and BMI has good impact on outcome. There is a strong positive correlation between BMI and Skinthickness or Pregnancies and age

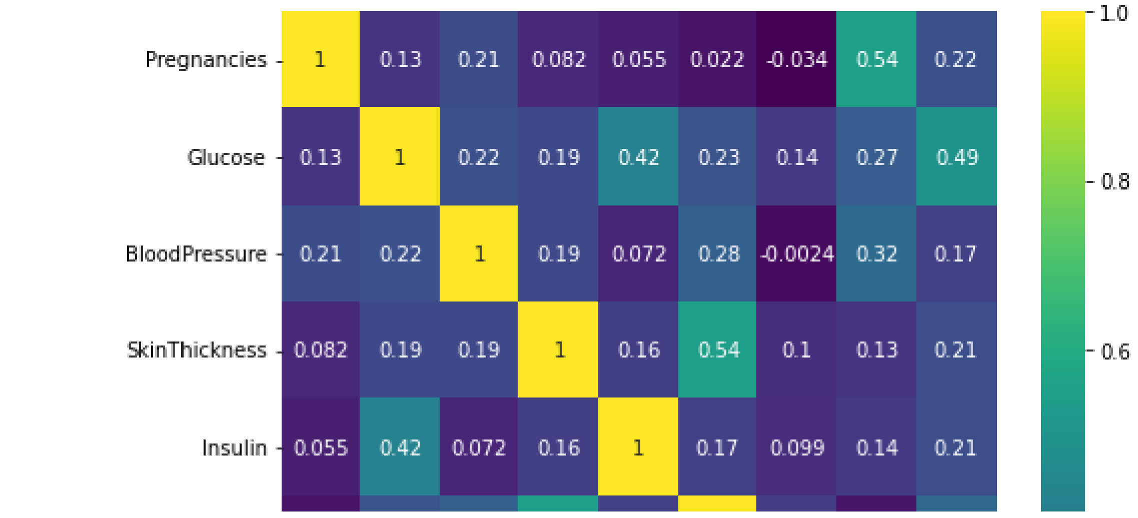
```
sns.heatmap(df.corr())
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7fa97ee5de10>
```



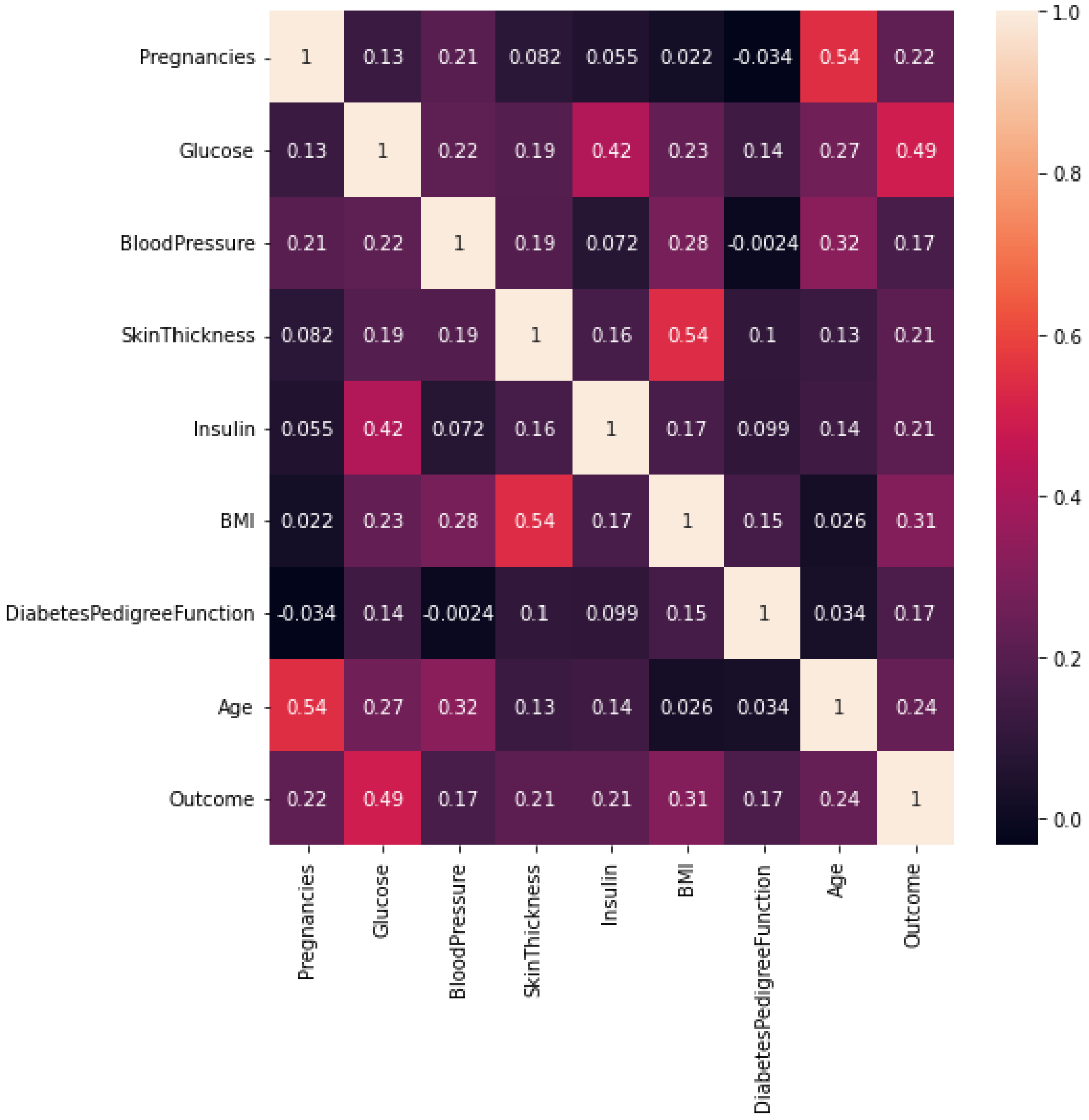
```
plt.subplots(figsize=(8,8))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa97ede76d0>
```



```
plt.subplots(figsize=(8,8))
sns.heatmap(df.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa97f0222d0>
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.21	0.082	0.055	0.022	-0.034	0.54	0.22
Glucose	0.13	1	0.22	0.19	0.42	0.23	0.14	0.27	0.49
BloodPressure	0.21	0.22	1	0.19	0.072	0.28	-0.0024	0.32	0.17
SkinThickness	0.082	0.19	0.19	1	0.16	0.54	0.1	0.13	0.21
Insulin	0.055	0.42	0.072	0.16	1	0.17	0.099	0.14	0.21
BMI	0.022	0.23	0.28	0.54	0.17	1	0.15	0.026	0.31
DiabetesPedigreeFunction	-0.034	0.14	-0.0024	0.1	0.099	0.15	1	0.034	0.17
Age	0.54	0.27	0.32	0.13	0.14	0.026	0.034	1	0.24
Outcome	0.22	0.49	0.17	0.21	0.21	0.31	0.17	0.24	1

As we can see that our dataset is highly imbalace so that we have to first balance the dataset by using oversampling method SMOTE function

```
from imblearn.over_sampling import SMOTE
```

```
smot=SMOTE()
X,y=smot.fit_resample(X,y)
```

```
print(X.shape,y.shape)
```

```
(1000, 8) (1000,)
```

The data is sampled and now the dataset is ready to move in the model buliding process

Model Building

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
0	6	148.0	72.0	35.0	155.0	33.6	
1	1	85.0	66.0	29.0	155.0	26.6	
2	8	183.0	64.0	29.0	155.0	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	

```
## features and label
features =df.iloc[:,[0,1,2,3,4,5,6,7]].values
label = df.iloc[:,8].values
```

Spliting the dataset by using train test split function

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

```
X_train,X_test,y_train,y_test = train_test_split(features,label,
                                                 test_size=0.2,random_state=10)
```

```
## Create Model
```

##Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
```

```
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.7768729641693811
0.7402597402597403
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(label,model.predict(features))
cm
```

```
array([[443,  57],
       [120, 148]])
```

```
from sklearn.metrics import classification_report
print(classification_report(label,model.predict(features)))
```

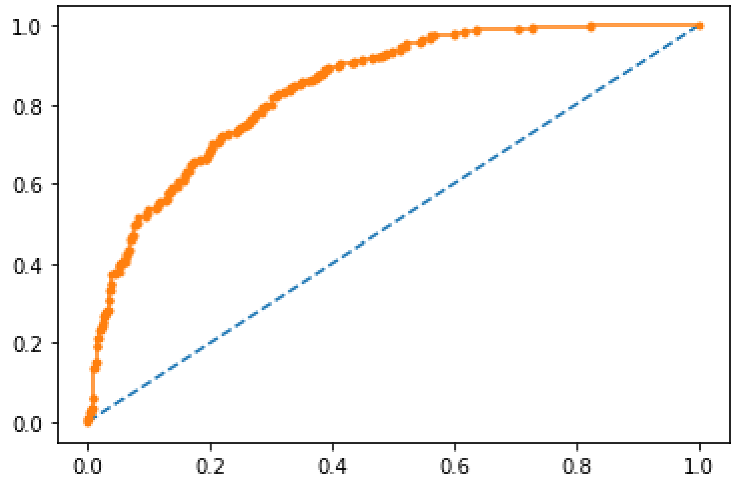
```
              precision    recall  f1-score   support

           0       0.79      0.89      0.83       500
           1       0.72      0.55      0.63       268

    accuracy                           0.77       768
   macro avg       0.75      0.72      0.73       768
weighted avg       0.76      0.77      0.76       768
```

```
### ROC(Receiver Operating characteristics Curve) curve
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

#predict probabilities
probs = model.predict_proba(features)

## probabilities for the positive outcome only
probs = probs[:, 1]
```

```
# calculate auc
auc = roc_auc_score(label, probs)
print('AUC: %.5f' % auc)

#calculate roc curve
fpr,tpr,thresholds = roc_curve(label, probs)

# plot no skill
plt.plot([0,1],[0,1], linestyle='--')
plt.plot(fpr,tpr,marker ='.')
```

```
    AUC: 0.84107
    [<matplotlib.lines.Line2D at 0x7fa980827250>]
```



```
## Applying Decision Tree Classifier
```

## Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier(max_depth=5)
model3.fit(X_train,y_train)
```

```
    DecisionTreeClassifier(max_depth=5)
```

```
model3.score(X_train,y_train)
```

```
    0.8192182410423453
```

```
model3.score(X_test,y_test)
```

```
    0.7532467532467533
```

```
## applying Random Forest
```

## Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model4= RandomForestClassifier(n_estimators=11)
model4.fit(X_train,y_train)
```

```
RandomForestClassifier(n_estimators=11)
```

```
model4.score(X_train,y_train)
```

```
0.990228013029316
```

```
model4.score(X_test,y_test)
```

```
0.6818181818181818
```

```
## Support Vector Classifier
```

```
from sklearn.svm import SVC
model5 = SVC(kernel='rbf',
             gamma='auto')
model5.fit(X_train,y_train)
```

```
SVC(gamma='auto')
```

```
model5.score(X_train,y_train)
```

```
1.0
```

```
model5.score(X_test,y_test)
```

```
0.6168831168831169
```

```
### Applying K-NN
```

## K-NN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7,metric='minkowski',p = 2)
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

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

```
0.7931596091205212
```

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

```
0.7207792207792207
```

Now we move to perform K-FOld Cross Validation with scikit Learn and going to train the model using 5 Fold cross validation and calculating the accuracy

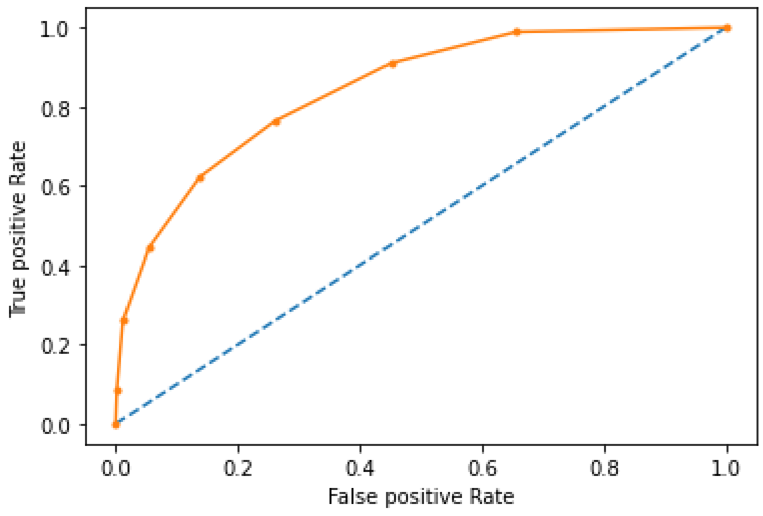
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
KFold = KFold(n_splits=5,shuffle=True,random_state=10)
scores=cross_val_score(model,X,y,cv=KFold,scoring='accuracy').mean()

print(scores)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.7340000000000001
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Plotting the AUC Curve with respective models:

```
### ROC curve for K-NN


from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict probabilities
probs= knn.predict_proba(features)

## probabilities for positive outcomes only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(label, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr,threshold = roc_curve(label,probs)
print("True positive Rate - {}, False positive Rate - {} Threshold - {}".format(tpr,fpr,th
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# roc curve for the model
plt.plot(fpr,tpr,marker='.')
plt.xlabel("False positive Rate")
plt.ylabel("True positive Rate")
```

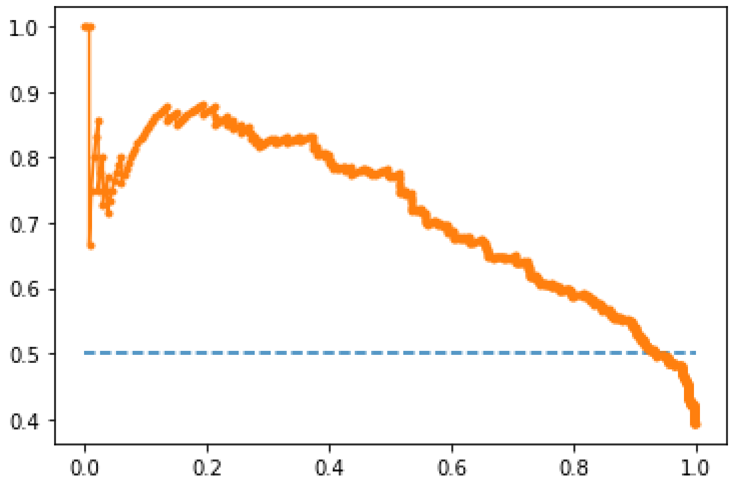
```
AUC: 0.842
True positive Rate - [0.         0.0858209  0.26119403 0.44402985 0.62313433 0.76492
 0.91044776 0.98880597 1.        ], False positive Rate - [0.    0.002 0.012 0.054 0
 0.54545455 0.63636364 0.72727273 0.81818182 0.90909091 1.        ]
Text(0, 0.5, 'True positive Rate')
```



```
#Precision Recall Curve for Logistic Regression

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
```

```
probs = model.predict_proba(features)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, probs)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

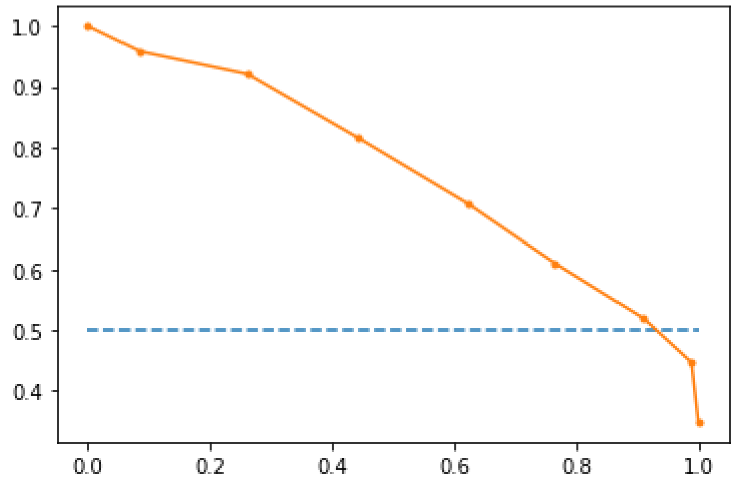
```
f1=0.626 auc=0.717 ap=0.719
[<matplotlib.lines.Line2D at 0x7fa97dcd66d0>]
```



```
#Precision Recall Curve for KNN

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
probs = knn.predict_proba(features)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = knn.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, probs)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, probs)
```

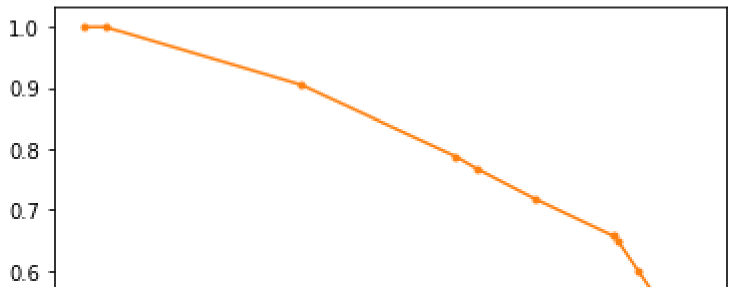
```
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.663 auc=0.762 ap=0.721
[<matplotlib.lines.Line2D at 0x7fa97dcc48d0>]
```



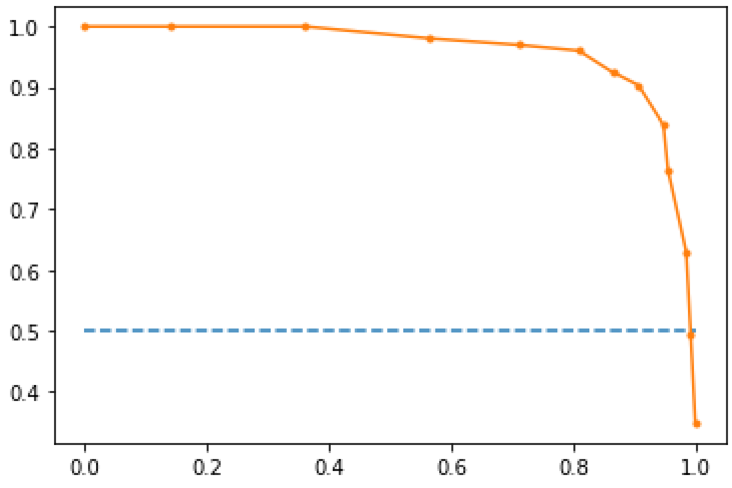
```
#Precission Recall Curve for Decission Tree Classifier

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
probs = model3.predict_proba(features)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model3.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, probs)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.686 auc=0.812 ap=0.771
[<matplotlib.lines.Line2D at 0x7fa97dc38350>]
```



```
#Precision Recall Curve for Random Forest

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
probs = model4.predict_proba(features)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model4.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, probs)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.894 auc=0.961 ap=0.951
[<matplotlib.lines.Line2D at 0x7fa97db9fd50>]
```



Colab paid products - Cancel contracts here

✓ 0s completed at 12:18 PM ● ✕