Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\aditya\Anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <
1.23.0 is required for this version of SciPy (detected version 1.23.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Loading Dataset

In [2]:
```
data = pd.read_csv('health care diabetes.csv')
```

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

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

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

Out[4]:
```
(768, 9)
```

Project Task: Week 1 -- Data Exploration and Missing Values Treatment

In [5]:
```
#checking null values
data.isnull().sum()
```

Out[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [6]:
```
data.isnull().any()
```

Out[6]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

Since the 0 value in Glucose,BloodPressure,SkinThickness,Insulin and BMI variables represent missing values.Lets find now many instances are there in each of the above variables

In [7]:
```
data[data['Glucose']==0]
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75	1	0	48	20	0	24.7	0.140	22	0
182	1	0	74	20	23	27.7	0.299	21	0
342	1	0	68	35	0	32.0	0.389	22	0
349	5	0	80	32	0	41.0	0.346	37	1
502	6	0	68	41	0	39.0	0.727	41	1

In [8]:
```
(5/765)*100
#only 0.6% of data is having missing value in glucose coloumn.
```

Out[8]:
```
0.6535947712418301
```

In [9]:
```
data[data['BloodPressure']==0]
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115	0	0	0	35.3	0.134	29	0
15	7	100	0	0	0	30.0	0.484	32	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
78	0	131	0	0	0	43.2	0.270	26	1
81	2	74	0	0	0	0.0	0.102	22	0
172	2	87	0	23	0	28.9	0.773	25	0
193	11	135	0	0	0	52.3	0.578	40	1
222	7	119	0	0	0	25.2	0.209	37	0
261	3	141	0	0	0	30.0	0.761	27	1
266	0	138	0	0	0	36.3	0.933	25	1
269	2	146	0	0	0	27.5	0.240	28	1
300	0	167	0	0	0	32.3	0.839	30	1
332	1	180	0	0	0	43.3	0.282	41	1
336	0	117	0	0	0	33.8	0.932	44	0
347	3	116	0	0	0	23.5	0.187	23	0
357	13	129	0	30	0	39.9	0.569	44	1
426	0	94	0	0	0	0.0	0.256	25	0
430	2	99	0	0	0	22.2	0.108	23	0
435	0	141	0	0	0	42.4	0.205	29	1
453	2	119	0	0	0	19.6	0.832	72	0
468	8	120	0	0	0	30.0	0.183	38	1
484	0	145	0	0	0	44.2	0.630	31	1
494	3	80	0	0	0	0.0	0.174	22	0
522	6	114	0	0	0	0.0	0.189	26	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
533	6	91	0	0	0	29.8	0.501	31	0
535	4	132	0	0	0	32.9	0.302	23	1
589	0	73	0	0	0	21.1	0.342	25	0
601	6	96	0	0	0	23.7	0.190	28	0
604	4	183	0	0	0	28.4	0.212	36	1
619	0	119	0	0	0	32.4	0.141	24	1
643	4	90	0	0	0	28.0	0.610	31	0
697	0	99	0	0	0	25.0	0.253	22	0
703	2	129	0	0	0	38.5	0.304	41	0
706	10	115	0	0	0	0.0	0.261	30	1

In [10]:
```
data[data['BloodPressure']==0].shape
```

Out[10]: (35, 9)

In [11]:
```
(35/765)*100
#4.5 % of data is having missing value in BP column
```

Out[11]: 4.57516339869281

In [12]:
```
data[data['SkinThickness']==0].shape
```

Out[12]: (227, 9)

In [13]:
```
(227/765)*100
#29.6% of data is having missing value values in skinthickness column
```

Out[13]: 29.673202614379086

In [14]:
```
data[data['Insulin']==0].shape
```

Out[14]: (374, 9)

In [15]:
```
(374/765)*100
#approx 49% of data is having missing
```

Out[15]:
```
48.888888888888886
```

In [16]:
```
data[data['BMI']==0].shape
```

Out[16]:
```
(11, 9)
```

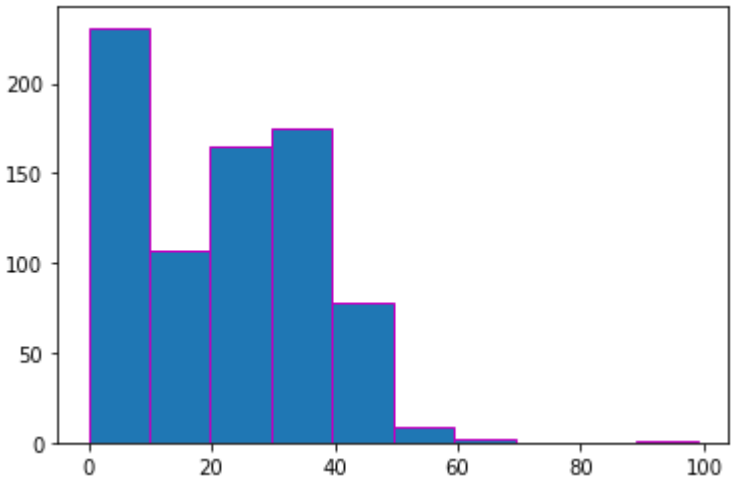
In [17]:
```
(11/765)*100
```

Out[17]:
```
1.4379084967320261
```

Since Insulin and SkinThickness are having higher percentages of missing values lets try to fill up the missing values

In [18]:
```
plt.hist(data['SkinThickness'],edgecolor='m')
```

Out[18]:
```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```



In [19]:
```
data[data['SkinThickness']!=0]['SkinThickness'].describe()
```

Out[19]:
```
count    541.000000
mean      29.153420
std       10.476982
min        7.000000
25%       22.000000
50%       29.000000
75%       36.000000
max       99.000000
Name: SkinThickness, dtype: float64
```

In [20]:
```
from numpy import nan
```

In [21]:
```
data[['SkinThickness']] = data[['SkinThickness']].replace(0,nan)
```

In [22]:
```
data.head(5)
```

Out[22]:

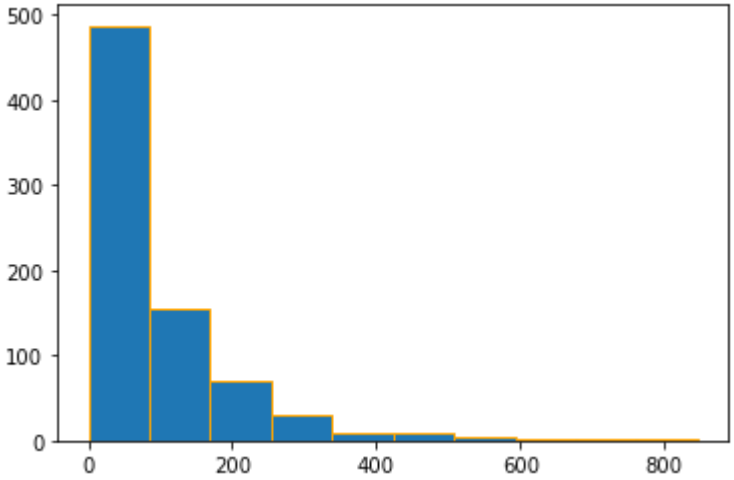
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35.0	0	33.6	0.627	50	1
1	1	85	66	29.0	0	26.6	0.351	31	0
2	8	183	64	NaN	0	23.3	0.672	32	1
3	1	89	66	23.0	94	28.1	0.167	21	0
4	0	137	40	35.0	168	43.1	2.288	33	1

In [23]:
```
data['SkinThickness'].isnull().sum()
```

Out[23]:
```
227
```

In [24]:
```
plt.hist(data['Insulin'],edgecolor='Orange')
```

Out[24]:
```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```



In [25]:
```
data[data['Insulin']!=0]['Insulin'].describe()
```

Out[25]:
```
count    394.000000
mean     155.548223
std      118.775855
min       14.000000
25%       76.250000
50%      125.000000
75%      190.000000
max      846.000000
Name: Insulin, dtype: float64
```

In [26]:
```
data[['Insulin']] = data[['Insulin']].replace(0,nan)
```

In [27]:
```
data['Insulin'].isnull().sum()
```

Out[27]: 374

Mean value of Skinthickness is ~29 and the mean value of Insulin is ~155 let impute the missing values with mean

In [28]:
```
data.fillna(data.mean(),inplace = True)
```

In [29]:
```
data['Insulin'].isnull().sum()
```

Out[29]: 0

In [30]:
```
data['SkinThickness'].isnull().sum()
```

Out[30]: 0

In [31]:
```
data[data['SkinThickness']==0].shape
```

Out[31]: (0, 9)

In [32]:
```
data[['Glucose']] = data[['Glucose']].replace(0,nan)
data['Glucose'].isnull().sum()
```

Out[32]: 5

In [33]:
```
data[['BloodPressure']] = data[['BloodPressure']].replace(0,nan)
data['BloodPressure'].isnull().sum()
```

Out[33]: 35

In [34]:
```
data[['BMI']] = data[['BMI']].replace(0,nan)
data['BMI'].isnull().sum()
```

Out[34]: 11

Maximum nan value after SkinThickness and Insulin is 35 in BloodPressure so we drop it .

In [35]:
```
data.dropna(axis = 0, inplace = True)
```

In [36]:
```
data.shape
```

Out[36]: (724, 9)

In [37]:
```
data.describe()
```

Out[37]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000
mean	3.866022	121.882597	72.400552	29.174664	155.823218	32.467127	0.474765	33.350829	0.343923
std	3.362803	30.750030	12.379870	9.018916	87.395661	6.888941	0.332315	11.765393	0.475344
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	118.250000	27.500000	0.245000	24.000000	0.000000
50%	3.000000	117.000000	72.000000	29.153420	155.548223	32.400000	0.379000	29.000000	0.000000
75%	6.000000	142.000000	80.000000	33.000000	155.548223	36.600000	0.627500	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [38]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 724 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               724 non-null    int64  
 1   Glucose                   724 non-null    float64
 2   BloodPressure             724 non-null    float64
 3   SkinThickness             724 non-null    float64
 4   Insulin                   724 non-null    float64
 5   BMI                       724 non-null    float64
 6   DiabetesPedigreeFunction  724 non-null    float64
 7   Age                       724 non-null    int64  
 8   Outcome                   724 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 56.6 KB
```

Project Task: Week 2 -- Corelation Analysis and Scatter Plots

In [39]:
```
Positive = data[data['Outcome']==1]
Positive.head()
```

Out[39]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.00000	155.548223	33.6	0.627	50	1
2	8	183.0	64.0	29.15342	155.548223	23.3	0.672	32	1
4	0	137.0	40.0	35.00000	168.000000	43.1	2.288	33	1
6	3	78.0	50.0	32.00000	88.000000	31.0	0.248	26	1
8	2	197.0	70.0	45.00000	543.000000	30.5	0.158	53	1

In [40]:
```
Negative = data[data['Outcome']==0]
Negative.head()
```

Out[40]:

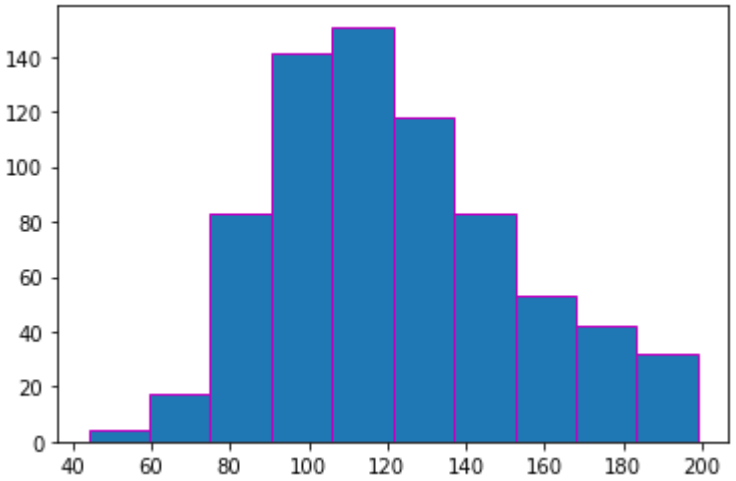
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	1	85.0	66.0	29.00000	155.548223	26.6	0.351	31	0
3	1	89.0	66.0	23.00000	94.000000	28.1	0.167	21	0
5	5	116.0	74.0	29.15342	155.548223	25.6	0.201	30	0
10	4	110.0	92.0	29.15342	155.548223	37.6	0.191	30	0
12	10	139.0	80.0	29.15342	155.548223	27.1	1.441	57	0

In [41]:
```
data['Glucose'].value_counts()
```

Out[41]:
```
100.0    16
99.0     15
111.0    14
106.0    14
112.0    13
         ..
61.0      1
178.0     1
160.0     1
182.0     1
190.0     1
Name: Glucose, Length: 135, dtype: int64
```

In [42]:
```
plt.hist(data['Glucose'],edgecolor = 'm')
```

Out[42]:
```
(array([  4.,  17.,  83., 141., 151., 118.,  83.,  53.,  42.,  32.]),
 array([ 44. ,  59.5,  75. ,  90.5, 106. , 121.5, 137. , 152.5, 168. ,
        183.5, 199. ]),
 <BarContainer object of 10 artists>)
```



In [43]:
```
data['BloodPressure'].value_counts()
```

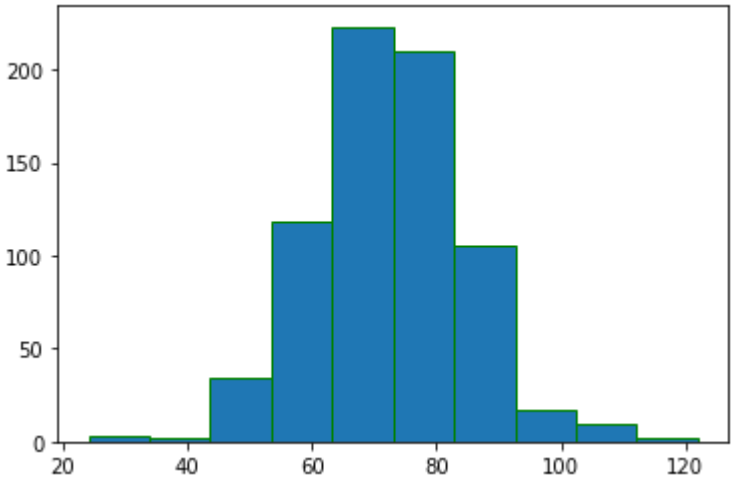
Out[43]:

70.0	57
74.0	51
78.0	45
72.0	44
68.0	43
64.0	42
76.0	39
80.0	39
60.0	37
62.0	34
66.0	30
82.0	29
88.0	25
84.0	23
90.0	22
58.0	21
86.0	21
50.0	13
56.0	12
54.0	11
52.0	11
92.0	8
75.0	7
65.0	7
85.0	6
94.0	6
44.0	4
48.0	4
106.0	3
110.0	3
98.0	3
100.0	3
96.0	3
46.0	2
104.0	2
108.0	2
55.0	2
30.0	2
122.0	1
95.0	1
102.0	1

```
61.0      1
24.0      1
38.0      1
40.0      1
114.0     1
Name: BloodPressure, dtype: int64
```

In [44]:
```
plt.hist(data['BloodPressure'],edgecolor = 'g')
```

Out[44]:
```
(array([  3.,   2.,  34., 118., 223., 210., 105.,  17.,  10.,   2.]),
 array([ 24. ,  33.8,  43.6,  53.4,  63.2,  73. ,  82.8,  92.6, 102.4,
        112.2, 122. ]),
 <BarContainer object of 10 artists>)
```



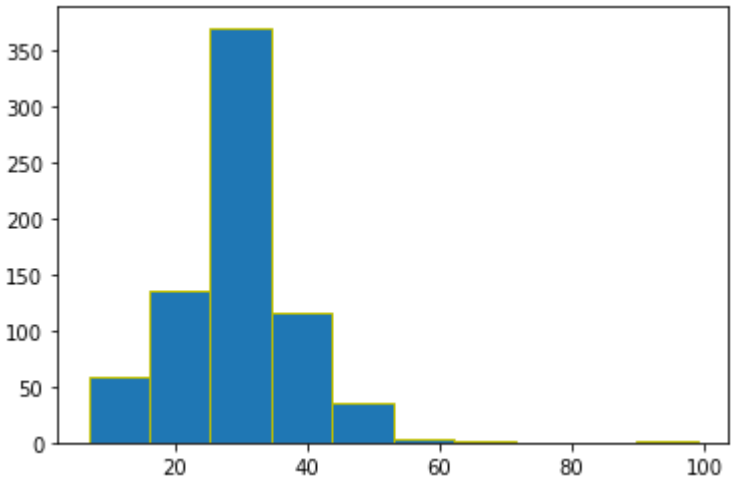
In [45]:
```
data['SkinThickness'].value_counts()
```

Out[45]:
```
29.15342    192
32.00000     30
30.00000     26
27.00000     23
28.00000     20
18.00000     20
33.00000     20
23.00000     19
31.00000     19
39.00000     18
19.00000     18
29.00000     17
37.00000     16
26.00000     16
25.00000     16
40.00000     16
22.00000     16
35.00000     14
36.00000     14
15.00000     14
41.00000     14
17.00000     14
24.00000     12
42.00000     11
13.00000     11
20.00000     11
21.00000     10
46.00000      8
34.00000      8
12.00000      7
38.00000      7
11.00000      6
43.00000      6
16.00000      6
45.00000      6
14.00000      6
44.00000      5
10.00000      5
48.00000      4
47.00000      4
49.00000      3
```

```
50.00000      3
8.00000       2
7.00000       2
52.00000      2
54.00000      2
63.00000      1
60.00000      1
56.00000      1
51.00000      1
99.00000      1
Name: SkinThickness, dtype: int64
```

In [46]:
```
plt.hist(data['SkinThickness'],edgecolor = 'y')
```

Out[46]:
```
(array([ 59., 136., 371., 116.,  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. ]),
 <BarContainer object of 10 artists>)
```

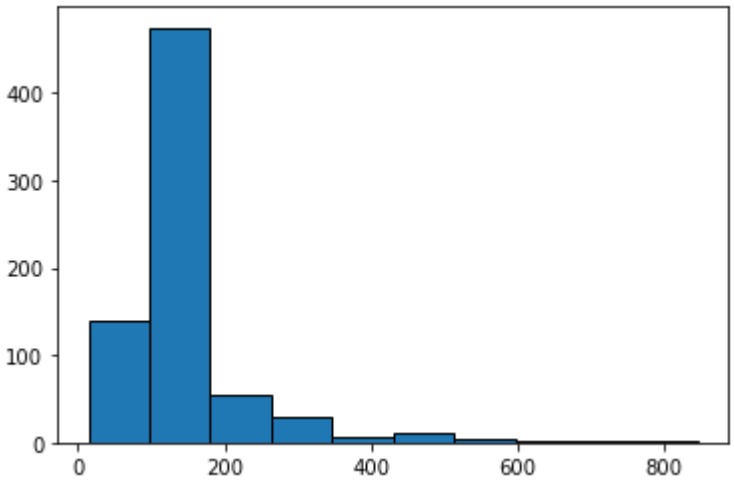


In [47]:
```
data['Insulin'].value_counts()
```

Out[47]:
```
155.548223    332
105.000000     11
130.000000      9
140.000000      9
120.000000      8
              ...
73.000000       1
171.000000      1
255.000000      1
52.000000       1
112.000000      1
Name: Insulin, Length: 185, dtype: int64
```

In [48]:
```
plt.hist(data['Insulin'],edgecolor = 'k')
```

Out[48]:
```
(array([140., 475.,  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. ]),
 <BarContainer object of 10 artists>)
```

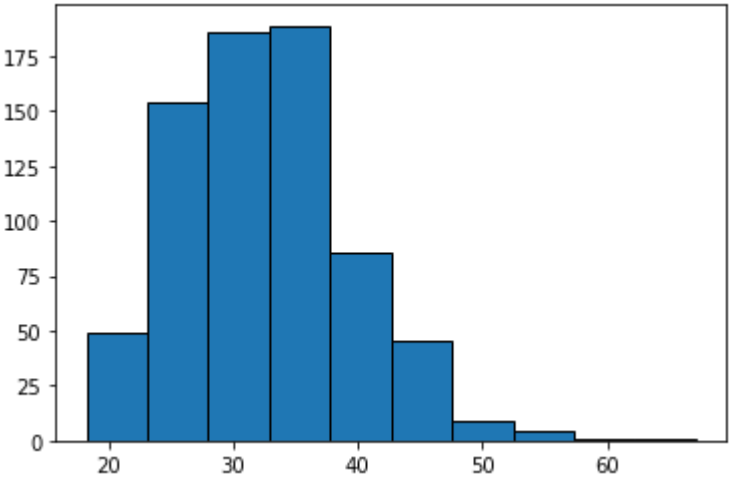


In [49]:
```
data['BMI'].value_counts()
```

Out[49]:
```
32.0    12
31.2    12
31.6    12
33.3    10
32.4     9
        ..
45.2     1
36.7     1
41.8     1
42.6     1
46.3     1
Name: BMI, Length: 245, dtype: int64
```

In [50]:
```
plt.hist(data['BMI'],edgecolor = 'k')
```

Out[50]:
```
(array([ 49., 154., 186., 189.,  86.,  45.,   9.,   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 ]),
 <BarContainer object of 10 artists>)
```



In [51]:
```
data.describe().transpose()
```

Out[51]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	724.0	3.866022	3.362803	0.000	1.000	3.000000	6.000000	17.00
Glucose	724.0	121.882597	30.750030	44.000	99.750	117.000000	142.000000	199.00
BloodPressure	724.0	72.400552	12.379870	24.000	64.000	72.000000	80.000000	122.00
SkinThickness	724.0	29.174664	9.018916	7.000	25.000	29.153420	33.000000	99.00
Insulin	724.0	155.823218	87.395661	14.000	118.250	155.548223	155.548223	846.00
BMI	724.0	32.467127	6.888941	18.200	27.500	32.400000	36.600000	67.10
DiabetesPedigreeFunction	724.0	0.474765	0.332315	0.078	0.245	0.379000	0.627500	2.42
Age	724.0	33.350829	11.765393	21.000	24.000	29.000000	41.000000	81.00
Outcome	724.0	0.343923	0.475344	0.000	0.000	0.000000	1.000000	1.00

In [52]:
```
Positive.shape
```

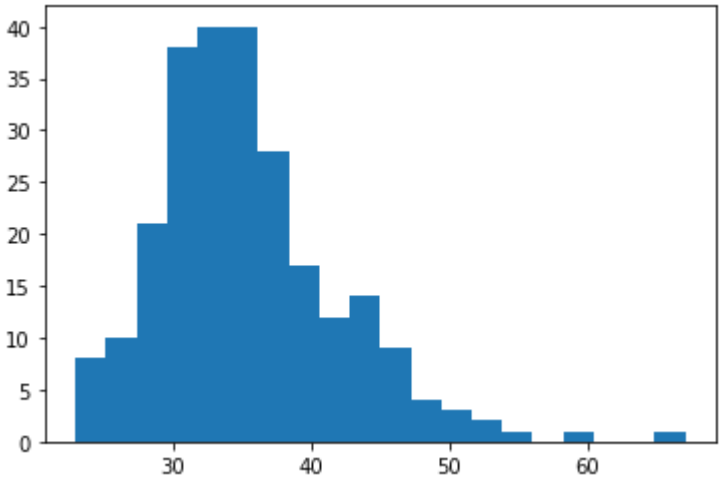
Out[52]: (249, 9)

In [53]:
```
Negative.shape
```

Out[53]: (475, 9)

In [54]:
```
plt.hist(Positive['BMI'],bins = 20)
```

Out[54]:
```
(array([ 8., 10., 21., 38., 40., 40., 28., 17., 12., 14.,  9.,  4.,  3.,
         2.,  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 ]),
 <BarContainer object of 20 artists>)
```

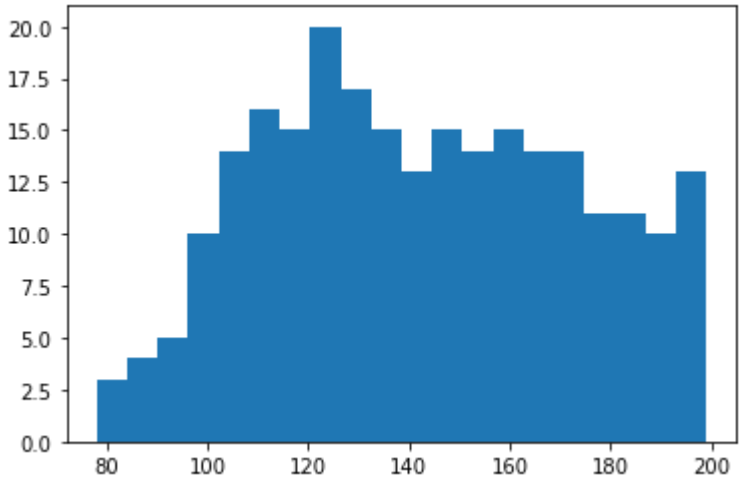


In [55]:
```
Positive['BMI'].value_counts()
```

Out[55]:
```
31.6    7
32.9    7
33.3    6
32.0    5
31.2    5
       ..
46.2    1
37.2    1
35.8    1
41.8    1
36.3    1
Name: BMI, Length: 143, dtype: int64
```

In [56]:
```
plt.hist(Positive['Glucose'],bins = 20)
```

Out[56]:
```
(array([ 3.,  4.,  5., 10., 14., 16., 15., 20., 17., 15., 13., 15., 14.,
        15., 14., 14., 11., 11., 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.  ]),
 <BarContainer object of 20 artists>)
```

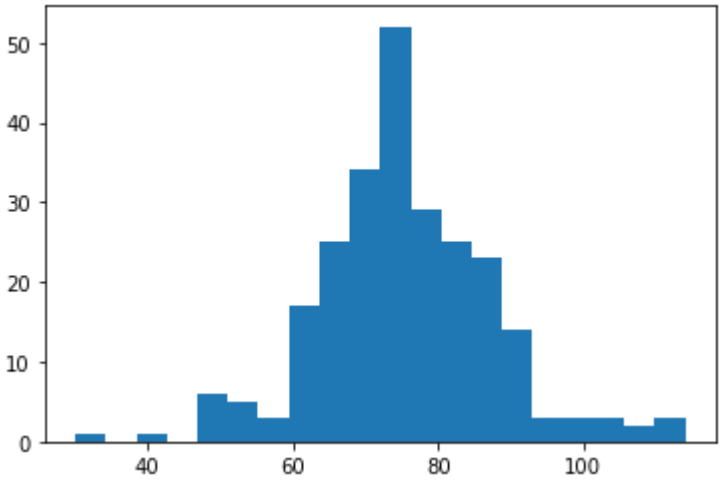


In [57]:
```
Positive['Glucose'].value_counts()
```

Out[57]:
```
158.0    6
128.0    6
125.0    6
109.0    5
124.0    5
        ..
116.0    1
193.0    1
172.0    1
175.0    1
190.0    1
Name: Glucose, Length: 102, dtype: int64
```

In [58]:
```
plt.hist(Positive['BloodPressure'],bins = 20)
```

Out[58]:
```
(array([ 1.,  0.,  1.,  0.,  6.,  5.,  3., 17., 25., 34., 52., 29., 25.,
        23., 14.,  3.,  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. ]),
 <BarContainer object of 20 artists>)
```

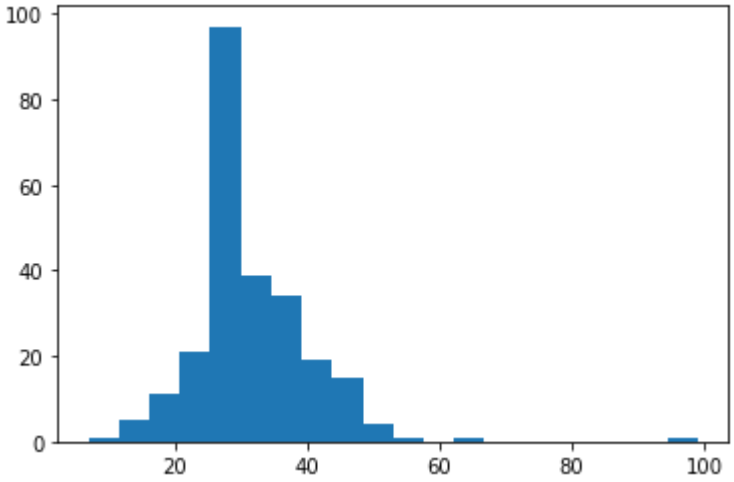


In [59]:
```
Positive['BloodPressure'].value_counts()
```

Out[59]:
```
70.0     23
76.0     18
78.0     17
74.0     17
72.0     16
82.0     13
64.0     13
84.0     12
80.0     12
66.0     11
90.0     11
88.0     11
68.0     11
62.0     10
86.0      9
60.0      7
50.0      5
85.0      3
92.0      3
94.0      3
52.0      3
110.0     2
58.0      2
54.0      2
98.0      2
104.0     2
106.0     1
100.0     1
75.0      1
102.0     1
40.0      1
48.0      1
65.0      1
56.0      1
30.0      1
108.0     1
114.0     1
Name: BloodPressure, dtype: int64
```

In [60]:
```
plt.hist(Positive['SkinThickness'],bins = 20)
```

Out[60]:
```
(array([ 1.,  5., 11., 21., 97., 39., 34., 19., 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. ]),
 <BarContainer object of 20 artists>)
```



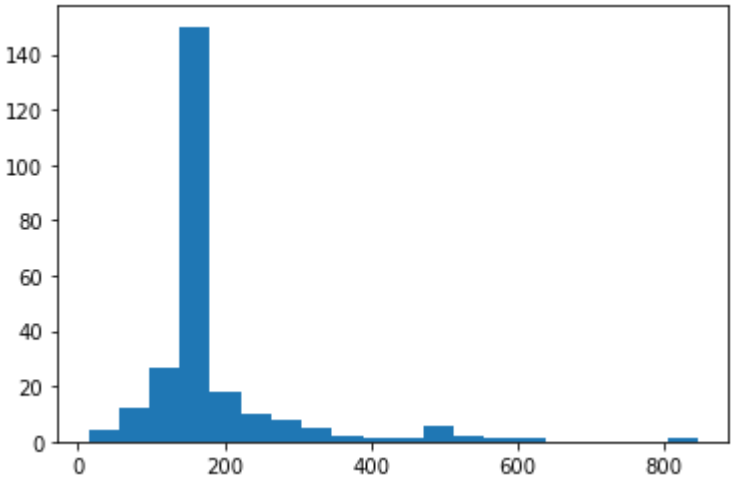
In [61]:
```
Positive['SkinThickness'].value_counts()
```

Out[61]:
```
29.15342    72
32.00000    13
33.00000     9
30.00000     8
39.00000     8
37.00000     8
36.00000     8
35.00000     8
27.00000     7
29.00000     7
42.00000     6
24.00000     6
31.00000     6
41.00000     6
26.00000     6
25.00000     5
40.00000     5
46.00000     5
28.00000     5
22.00000     4
23.00000     4
18.00000     4
19.00000     3
34.00000     3
49.00000     3
45.00000     3
44.00000     3
47.00000     2
48.00000     2
20.00000     2
14.00000     2
21.00000     2
43.00000     2
17.00000     2
38.00000     2
12.00000     1
63.00000     1
56.00000     1
7.00000      1
15.00000     1
13.00000     1
```

```
51.00000     1
99.00000     1
Name: SkinThickness, dtype: int64
```

In [62]:
```
plt.hist(Positive['Insulin'],bins = 20)
```

Out[62]:
```
(array([  4.,  12.,  27., 150.,  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. ]),
 <BarContainer object of 20 artists>)
```



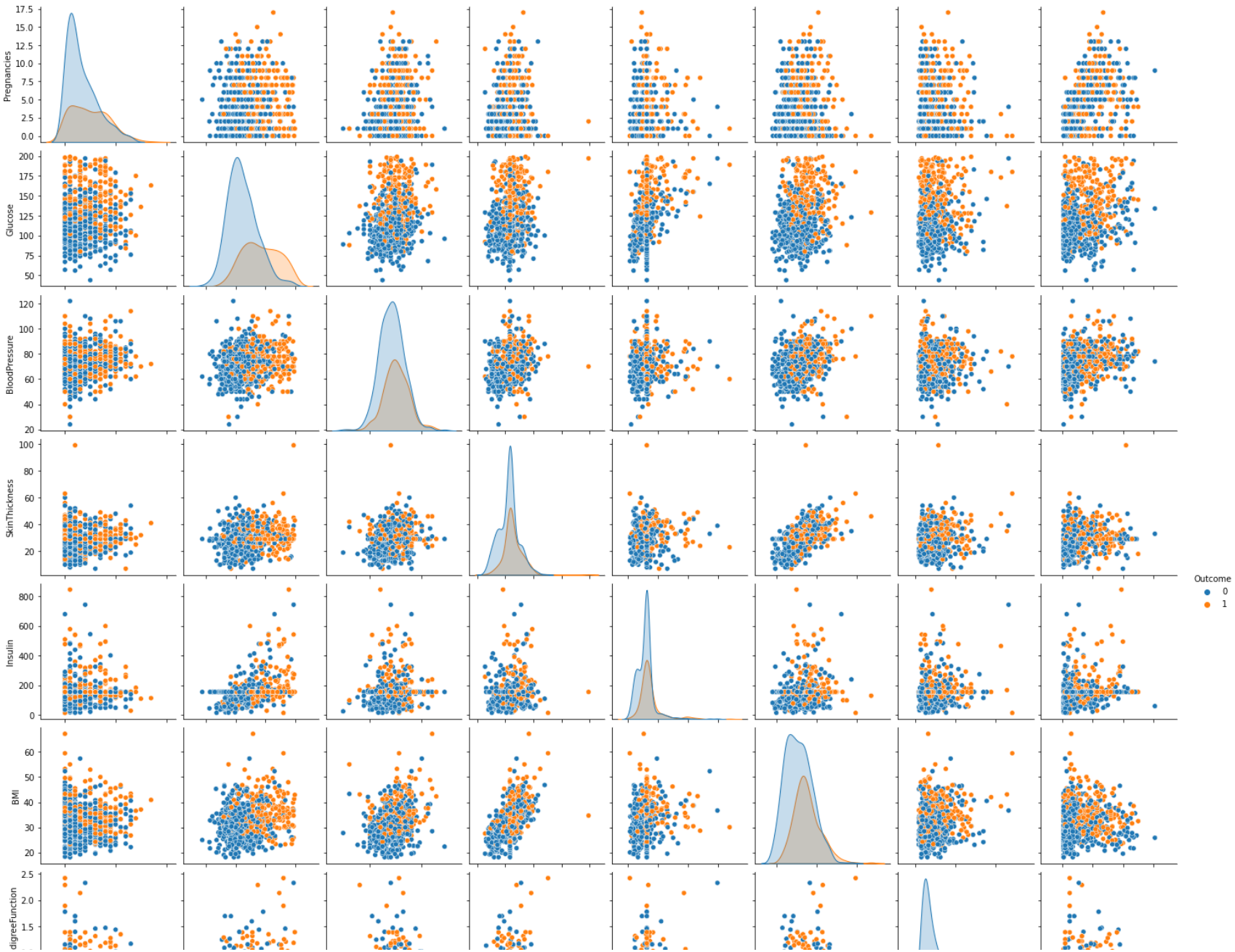
In [63]:
```
Positive['Insulin'].value_counts()
```

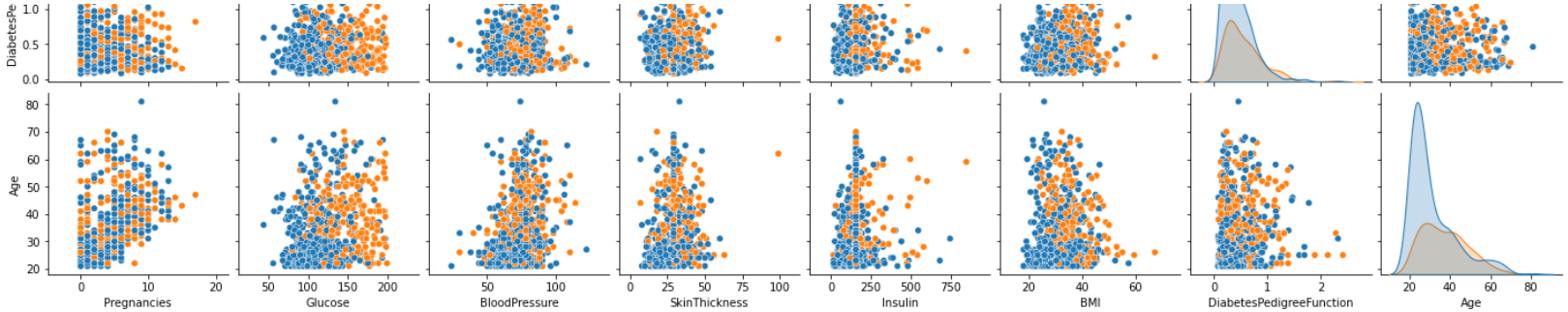
Out[63]:
```
155.548223    119
130.000000      6
180.000000      4
175.000000      3
156.000000      3
               ...
29.000000       1
171.000000      1
249.000000      1
140.000000      1
510.000000      1
Name: Insulin, Length: 93, dtype: int64
```

Scatter Plots

In [64]:
```
#Pair plot for whole data
sns.pairplot(data, hue = 'Outcome')
```

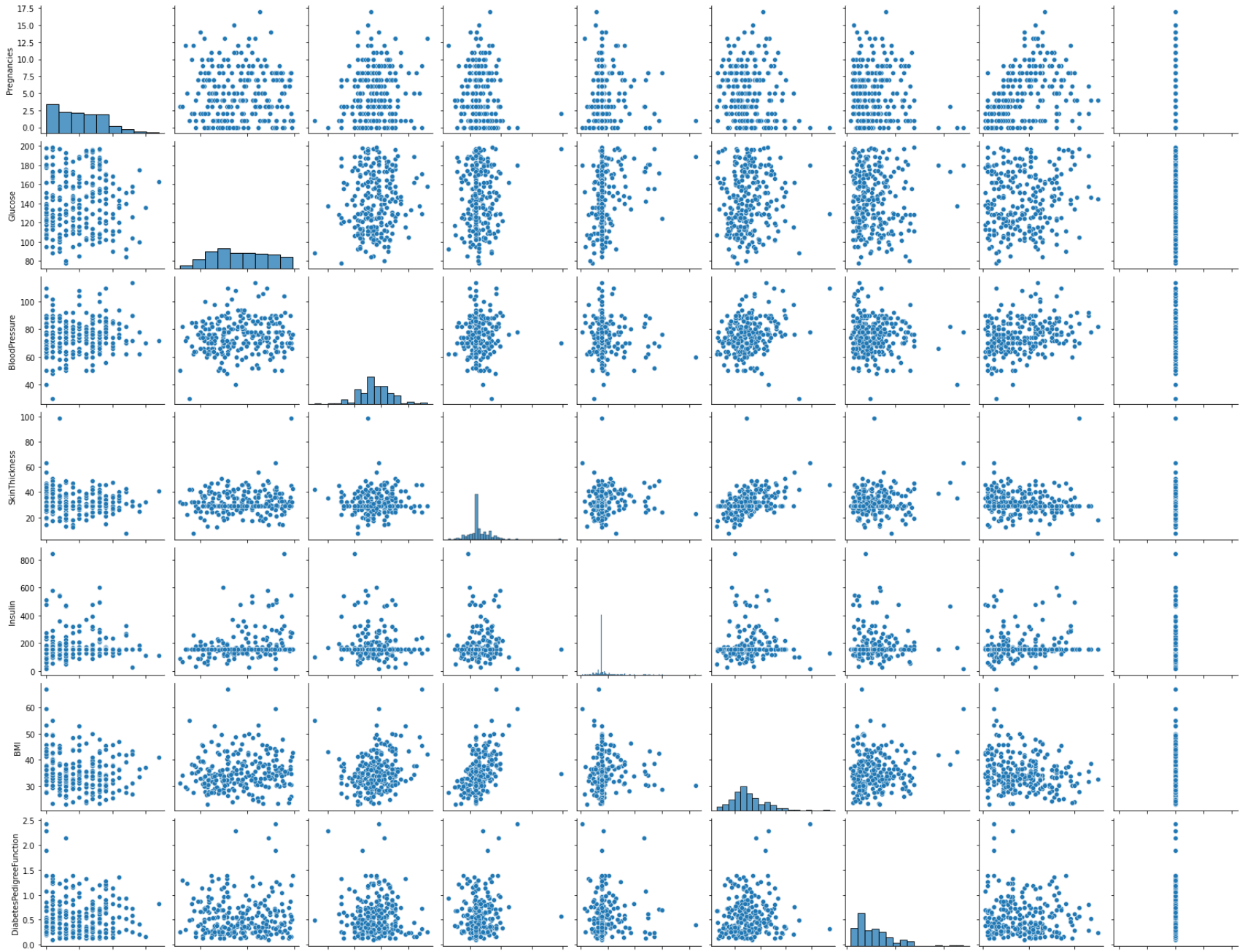
Out[64]:
```
<seaborn.axisgrid.PairGrid at 0x57ae694700>
```

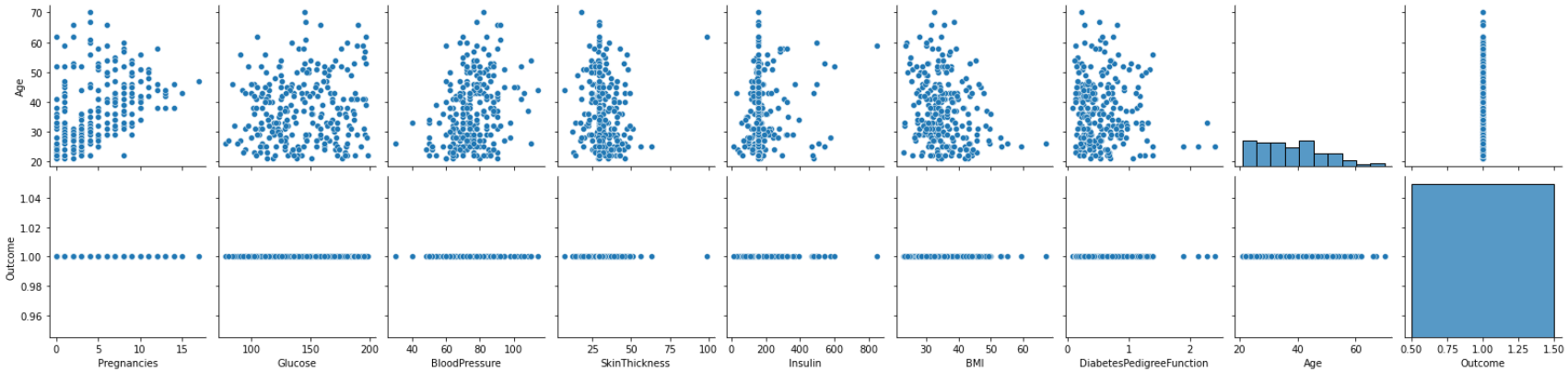



DiabetesPe
1.0
0.5
0.0
Age
80
70
60
50
40
30
20
0
10
20
Pregnancies
50
100
150
200
Glucose
50
100
BloodPressure
0
25
50
75
100
SkinThickness
0
250
500
750
Insulin
20
40
60
BMI
0
1
2
DiabetesPedigreeFunction
20
40
60
80
Age

In [65]:
```
# Pair plot for all positive cases
sns.pairplot(Positive)
```

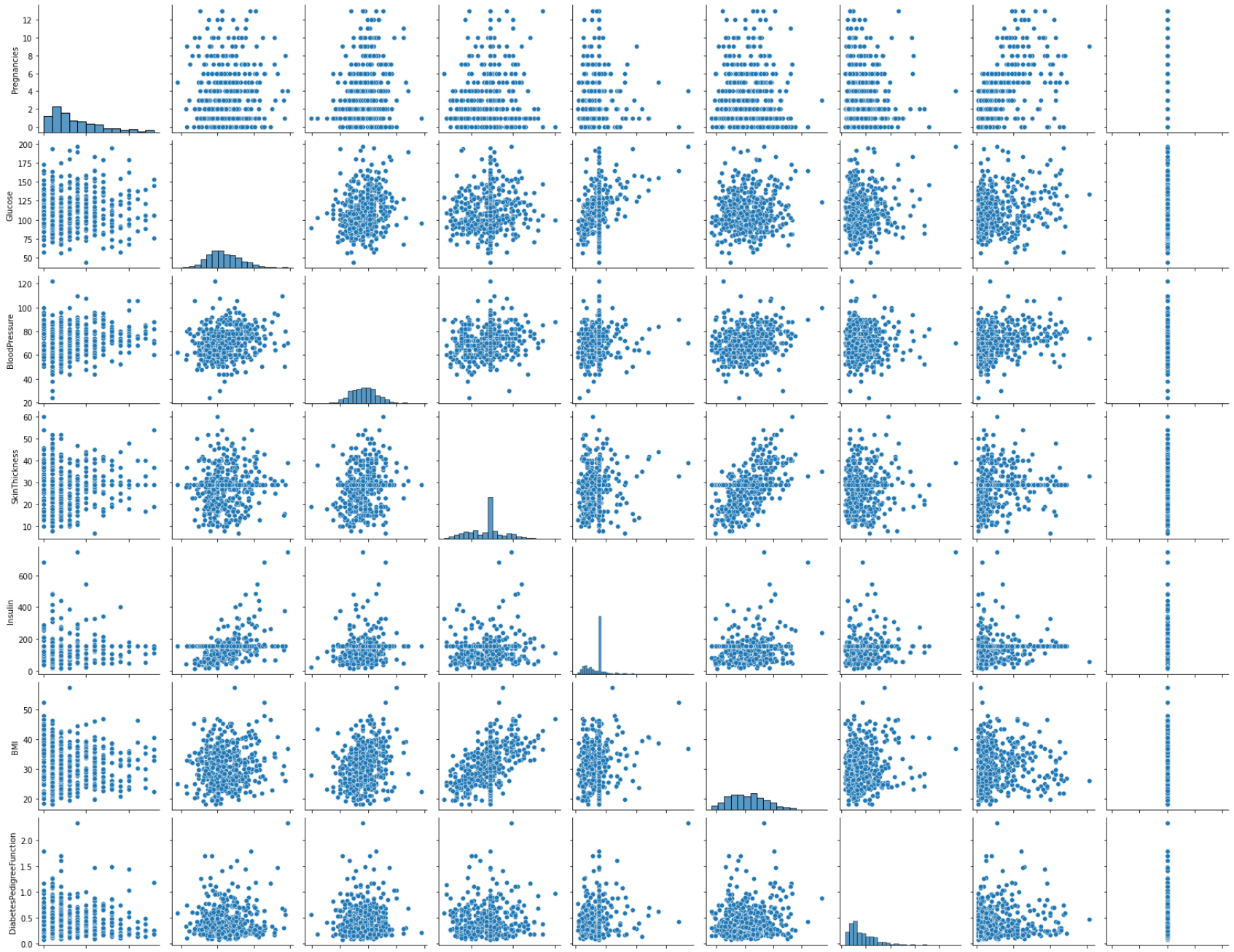
Out[65]:
```
<seaborn.axisgrid.PairGrid at 0x57b27d92b0>
```

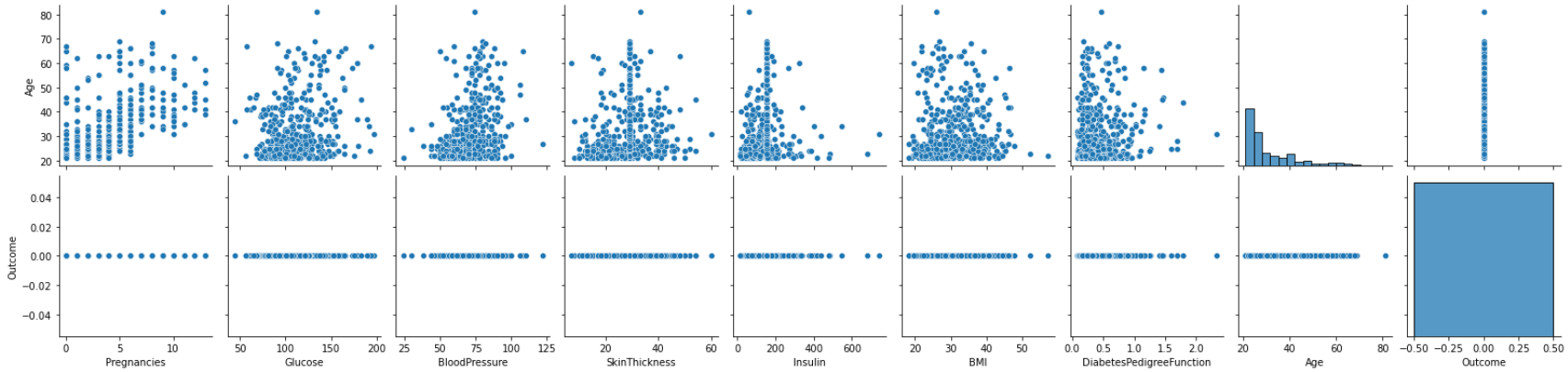



Age
Outcome
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

In [66]:
```
# Pair plot for all negative cases
sns.pairplot(Negative)
```

Out[66]:
```
<seaborn.axisgrid.PairGrid at 0x57ab37be50>
```





Co-Realation Analysis & Heat Map

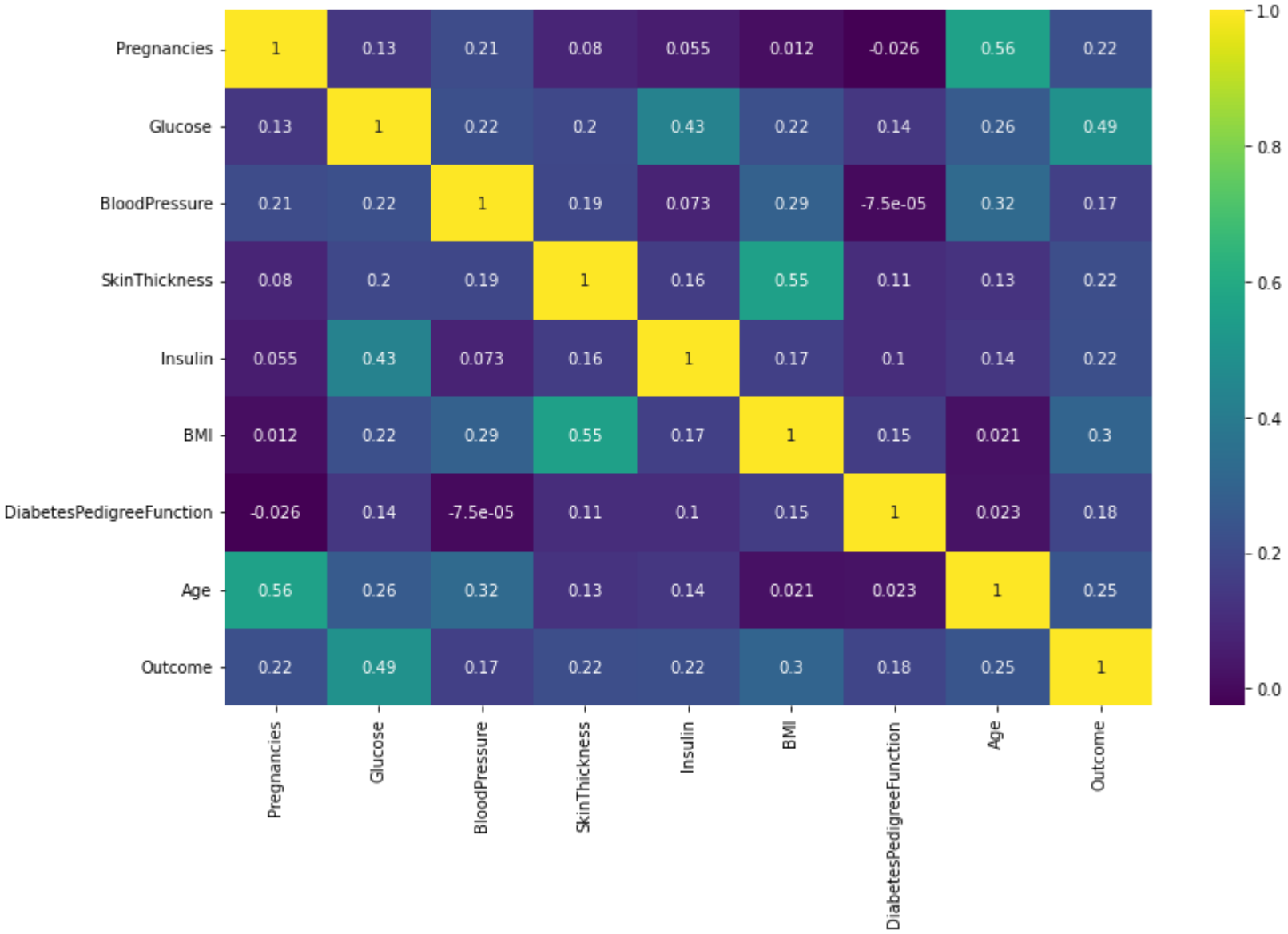
In [67]:
```
data.corr()
```

Out[67]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.000000	0.134915	0.209668	0.080012	0.054940	0.012342	-0.025996	0.557066
Glucose	0.134915	1.000000	0.223331	0.195695	0.429041	0.223276	0.136630	0.263560
BloodPressure	0.209668	0.223331	1.000000	0.192483	0.072682	0.287403	-0.000075	0.324897
SkinThickness	0.080012	0.195695	0.192483	1.000000	0.156224	0.554351	0.105612	0.126119
Insulin	0.054940	0.429041	0.072682	0.156224	1.000000	0.170125	0.104363	0.137744
BMI	0.012342	0.223276	0.287403	0.554351	0.170125	1.000000	0.154858	0.020835
DiabetesPedigreeFunction	-0.025996	0.136630	-0.000075	0.105612	0.104363	0.154858	1.000000	0.023098
Age	0.557066	0.263560	0.324897	0.126119	0.137744	0.020835	0.023098	1.000000
Outcome	0.224417	0.488384	0.166703	0.216661	0.219748	0.299375	0.184947	0.245741

In [68]:
```
plt.subplots(figsize = (13,8))
sns.heatmap(data.corr(),annot=True,cmap='viridis')
```

Out[68]: <AxesSubplot:>



Correlation Results :

There are not much multicolinearity,

Pregnancies and Age have some positive corelation,

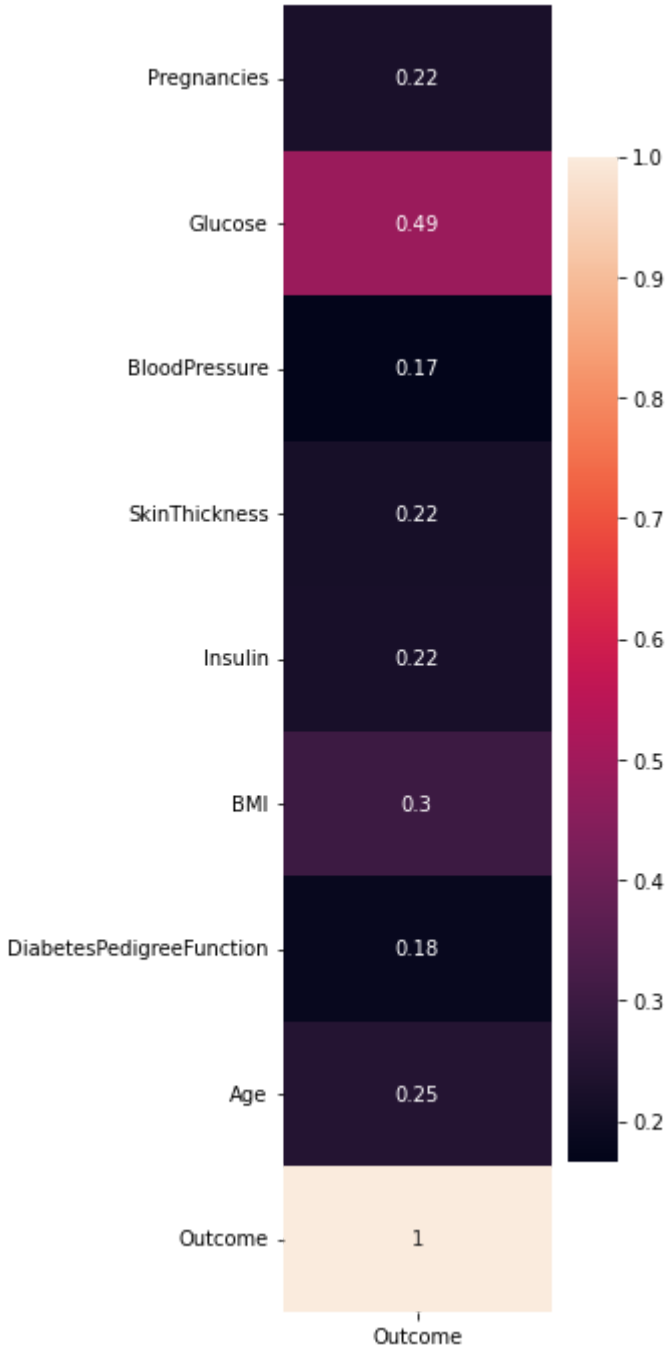
Glucose has some postive corelation with the outcome variable,

Skin thickness and BMI has some positive corelation,

Insulin and Glucose has some positive corelation

In [69]:
```
plt.figure(figsize=(3,12))
sns.heatmap(data.corr('pearson')[['Outcome']], annot = True)
```

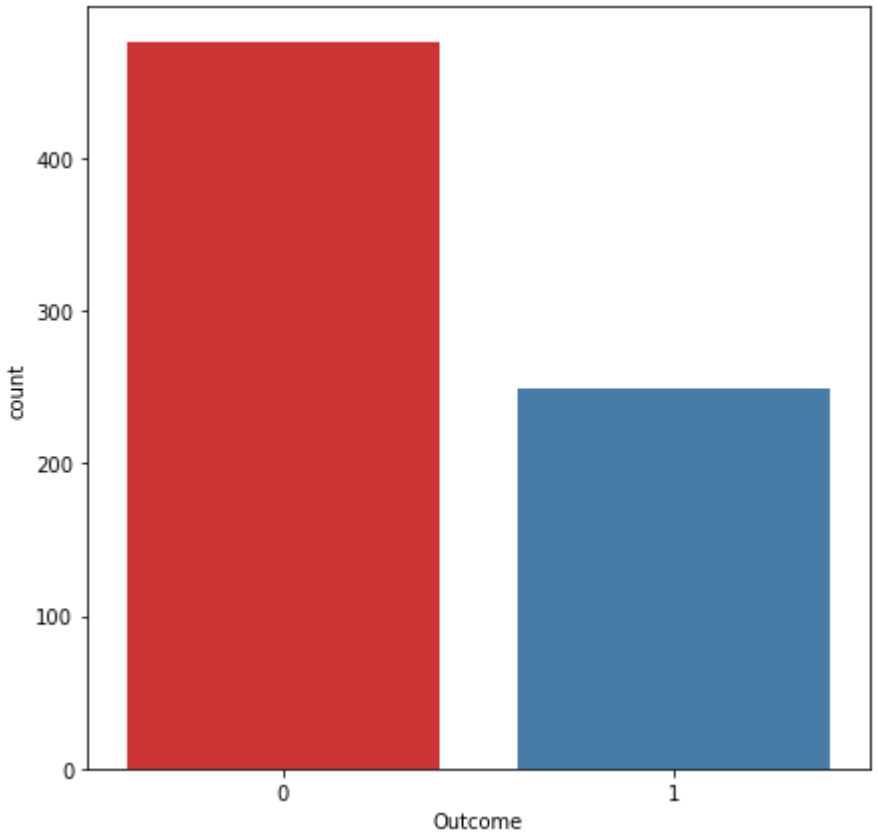
Out[69]:
```
<AxesSubplot:>
```



Checking Data Balance

In [70]:
```
plt.figure(figsize = (7,7))
sns.countplot(x = 'Outcome', data = data, palette = 'Set1')
```

Out[70]:
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



In [71]:
```
data['Outcome'].value_counts()/len(data)
```

Out[71]:
```
0    0.656077
1    0.343923
Name: Outcome, dtype: float64
```

After checking we found data is Imbalance to we try to balance it.

In [72]:
```
data['Outcome'].value_counts()
```

Out[72]:
```
0    475
1    249
Name: Outcome, dtype: int64
```

here we saw that the data is inbalance such that 66:34 ration between positive and negative case. so we can do balance our data

In [73]:
```
# Upsampling the True samples to eliminate Imbalanced Data Distribution
from sklearn.utils import resample
df_0 = data[data['Outcome'] == 0]
df_1 = data[data['Outcome'] == 1]
```

In [74]:
```
# Apply Resample
#n_sample = number of sample that we wanted to generate
df_1_upsample = resample(df_1, n_samples = 375, replace = True, random_state = 123)
```

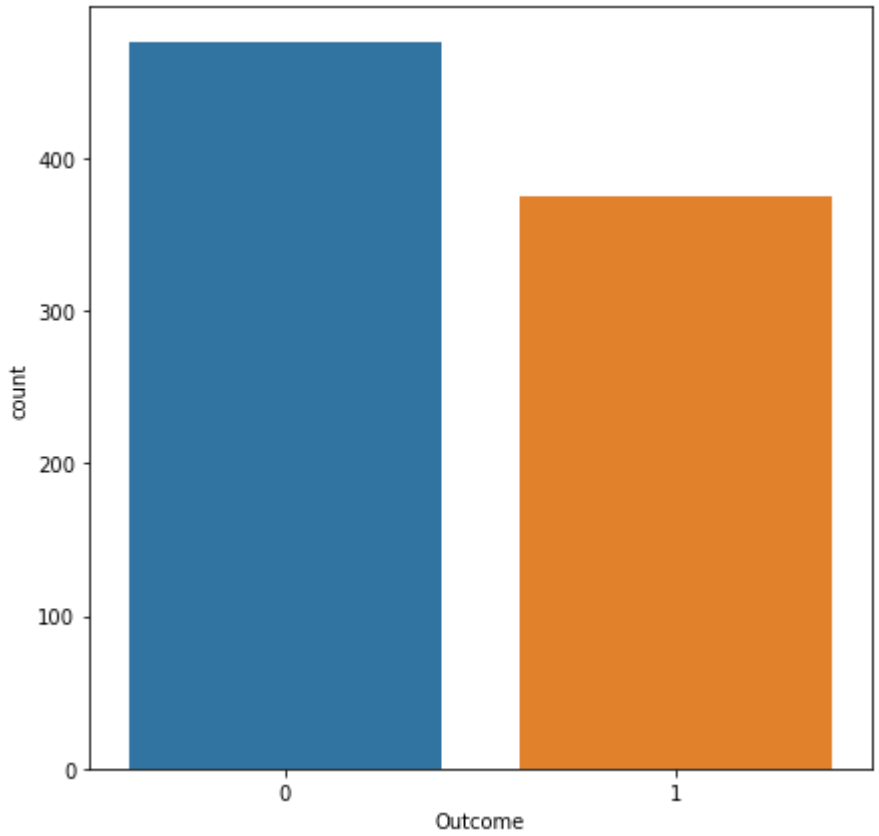
In [75]:
```
data1 = pd.concat([df_0, df_1_upsample])
data1
```

Out[75]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	1	85.0	66.0	29.00000	155.548223	26.6	0.351	31	0
3	1	89.0	66.0	23.00000	94.000000	28.1	0.167	21	0
5	5	116.0	74.0	29.15342	155.548223	25.6	0.201	30	0
10	4	110.0	92.0	29.15342	155.548223	37.6	0.191	30	0
12	10	139.0	80.0	29.15342	155.548223	27.1	1.441	57	0
...	...	...	...	...	...	...	...	...	...
417	4	144.0	82.0	32.00000	155.548223	38.5	0.554	37	1
93	4	134.0	72.0	29.15342	155.548223	23.8	0.277	60	1
498	7	195.0	70.0	33.00000	145.000000	25.1	0.163	55	1
366	6	124.0	72.0	29.15342	155.548223	27.6	0.368	29	1
280	0	146.0	70.0	29.15342	155.548223	37.9	0.334	28	1

850 rows × 9 columns

In [76]:
```
plt.figure(figsize=(7,7))
fig = sns.countplot(x='Outcome',data=data1)
```



In [77]:
```
data1['Outcome'].value_counts()
```

Out[77]:
```
0    475
1    375
Name: Outcome, dtype: int64
```

Project Task: Week 3 and Week 4 -- Data Modelling and Model Performance Evaluation

Model 1: Logistic Regression

In [78]:
```
data1.head(7)
```

Out[78]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	1	85.0	66.0	29.00000	155.548223	26.6	0.351	31	0
3	1	89.0	66.0	23.00000	94.000000	28.1	0.167	21	0
5	5	116.0	74.0	29.15342	155.548223	25.6	0.201	30	0
10	4	110.0	92.0	29.15342	155.548223	37.6	0.191	30	0
12	10	139.0	80.0	29.15342	155.548223	27.1	1.441	57	0
18	1	103.0	30.0	38.00000	83.000000	43.3	0.183	33	0
20	3	126.0	88.0	41.00000	235.000000	39.3	0.704	27	0

In [79]:
```
x = data1.drop(['Outcome'],axis = 1)
y = data1['Outcome']
```

In [80]:
```
x.shape
```

Out[80]:
```
(850, 8)
```

In [81]:
```
y.shape
```

Out[81]:
```
(850,)
```

In [82]:
```
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 = 1)
```

In [83]:
```
from sklearn.linear_model import LogisticRegression
# solver = 'Lbfgs', C = higher the better, max_iter = 1e7 , penalty = 'l2'
logit_model = LogisticRegression(solver = 'lbfgs', C = 1e5, max_iter=1e7, penalty='l2')
```

In [84]:
```
logit_model.fit(x_train, y_train)
```

Out[84]:
```
LogisticRegression(C=100000.0, max_iter=10000000.0)
```

In [85]:
```
print(logit_model.score(x_train,y_train))
print(logit_model.score(x_test,y_test))
```

```
0.7529411764705882
0.7588235294117647
```

In [86]:
```
y_pred = logit_model.predict(x_test)
```

In [87]:
```
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logit_model.score(x_test, y_test))
```

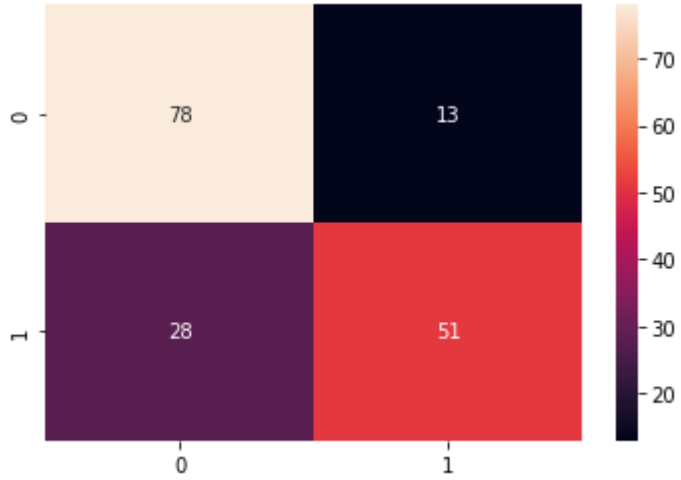
```
Accuracy of logistic regression classifier on test set: 0.76
```

In [88]:
```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix(y_test, y_pred)
```

Out[88]:
```
array([[78, 13],
       [28, 51]], dtype=int64)
```

In [89]:
```
# Diagonals are True Predictions & Non - Diagonals are false
sns.heatmap(confusion_matrix(y_test, y_pred), annot = True, fmt = '0.0f')
```

Out[89]: <AxesSubplot:>

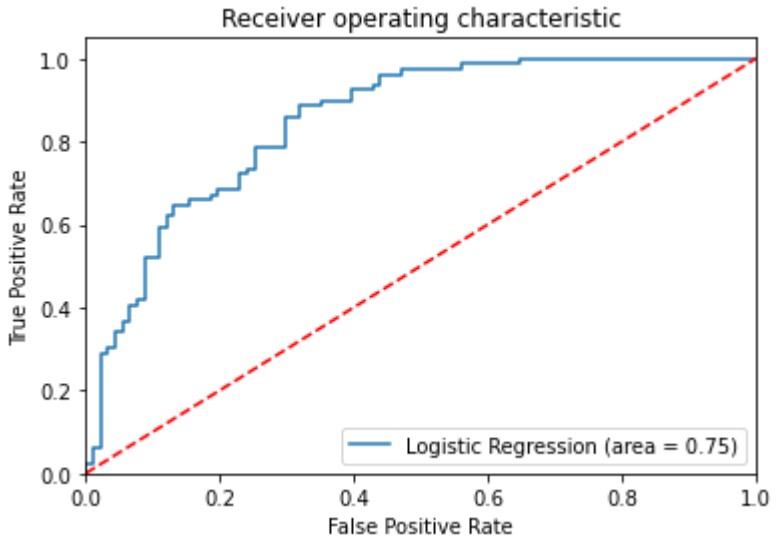


In [90]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.86      0.79        91
           1       0.80      0.65      0.71        79

    accuracy                           0.76       170
   macro avg       0.77      0.75      0.75       170
weighted avg       0.76      0.76      0.76       170
```

In [91]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logit_model.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, logit_model.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
print('AUC: %.3f' % logit_roc_auc)
plt.show()
```

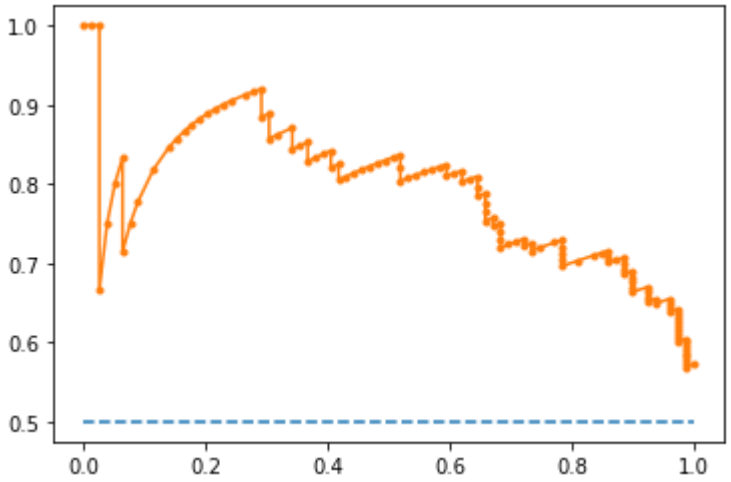
```
AUC: 0.751
```



In [92]:
```
#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 = logit_model.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = logit_model.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.713 auc=0.791 ap=0.795
```

Out[92]:
```
[<matplotlib.lines.Line2D at 0x57bd87db80>]
```



Model 2 : Decision Tree Classifier

In [93]:
```
#Hyper Parameter tuning of max_dept
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
for i in range(1,20):
    print("For max_depth = ",i)
    DTModel = DecisionTreeClassifier(max_depth=i)
    DTModel.fit(x_train,y_train)
    y_pred = DTModel.predict(x_test)
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
For max_depth =  1
Accuracy: 0.7
For max_depth =  2
Accuracy: 0.7
For max_depth =  3
Accuracy: 0.7
For max_depth =  4
Accuracy: 0.711764705882353
For max_depth =  5
Accuracy: 0.7411764705882353
For max_depth =  6
Accuracy: 0.7588235294117647
For max_depth =  7
Accuracy: 0.7647058823529411
For max_depth =  8
Accuracy: 0.7647058823529411
For max_depth =  9
Accuracy: 0.7823529411764706
For max_depth =  10
```

Highest accuracy of Decision Tree Model can be obtained on Max_Depth = 14

In [94]:
```
DTModel = DecisionTreeClassifier(criterion = 'entropy', random_state = 0, max_depth=14)
DTModel.fit(x_train,y_train)
```

Out[94]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=14, random_state=0)
```

In [95]:
```
y_pred = DTModel.predict(x_test)
```

In [96]:
```
print(DTModel.score(x_train,y_train))
print(DTModel.score(x_test,y_test))
```

```
0.9602941176470589
0.8176470588235294
```

In [97]:
```
print('Accuracy of Decision Tree regression classifier on test set: {:.2f}'.format(DTModel.score(x_test, y_test)
```

```
Accuracy of Decision Tree regression classifier on test set: 0.82
```

In [98]:
```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

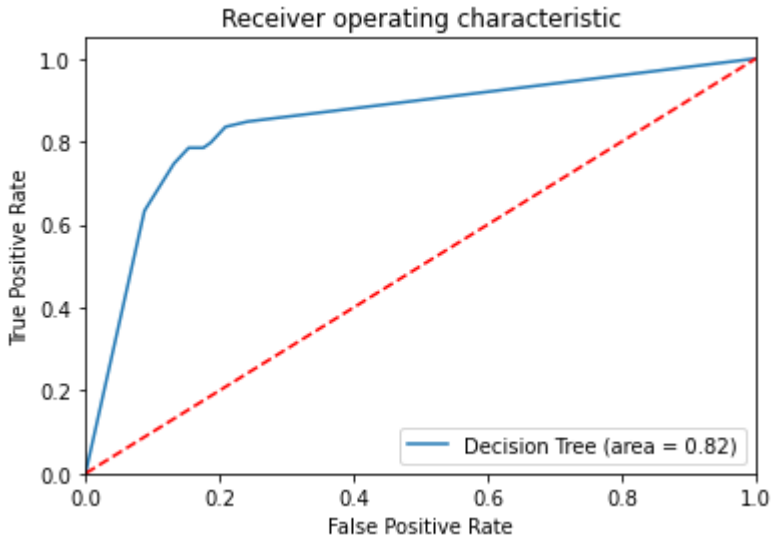
```
[[77 14]
 [17 62]]
```

In [99]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.85      0.83        91
           1       0.82      0.78      0.80        79

    accuracy                           0.82       170
   macro avg       0.82      0.82      0.82       170
weighted avg       0.82      0.82      0.82       170
```

In [100]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
dt_roc_auc = roc_auc_score(y_test, DTModel.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, DTModel.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Decision Tree (area = %0.2f)' % dt_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('DT_ROC')
print('AUC: %.3f' % dt_roc_auc)
plt.show()
```

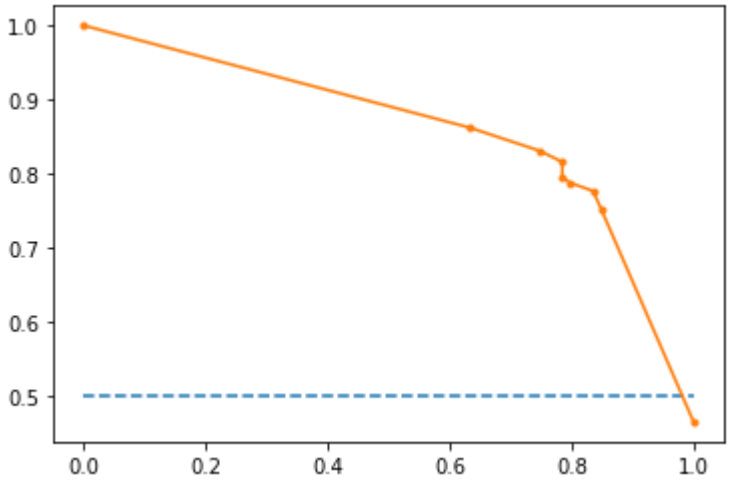
```
AUC: 0.815
```



In [101]:
```
#Precision 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 = DTModel.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = DTModel.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.800 auc=0.859 ap=0.791
```

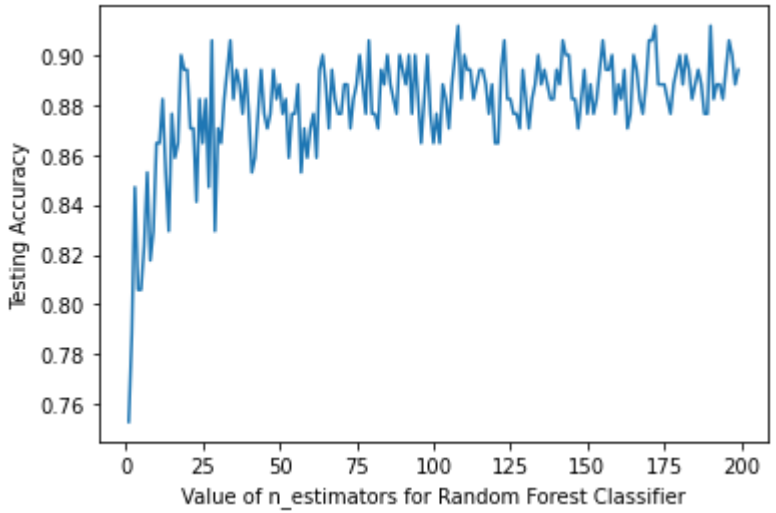
Out[101]:
```
[<matplotlib.lines.Line2D at 0x57bdadc160>]
```



Model 3 : Random Forest Classifier

In [102]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
scores =[]
for k in range(1, 200):
    rfc = RandomForestClassifier(n_estimators=k)
    rfc.fit(x_train, y_train)
    y_pred = rfc.predict(x_test)
    scores.append(accuracy_score(y_test, y_pred))
plt.plot(range(1, 200), scores)
plt.xlabel('Value of n_estimators for Random Forest Classifier')
plt.ylabel('Testing Accuracy')
```

Out[102]:
```
Text(0, 0.5, 'Testing Accuracy')
```



By above graph we find that at approx 160 of N_estimator give maximum accuracy of test data.

In [103]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 160)
rf.fit(x_train, y_train)
```

Out[103]:
```
RandomForestClassifier(n_estimators=160)
```

In [104]:
```
print(rf.score(x_train,y_train))
print(rf.score(x_test,y_test))
```

```
1.0
0.9
```

In [105]:
```
y_pred = rf.predict(x_test)
```

In [106]:
```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[83  8]
 [ 9 70]]
```

In [107]:
```
print(classification_report(y_test, y_pred))
```

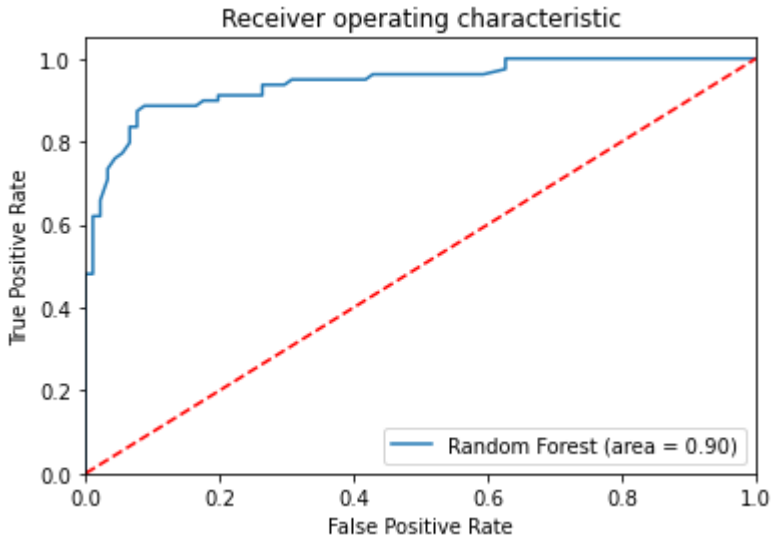
```
              precision    recall  f1-score   support

           0       0.90      0.91      0.91        91
           1       0.90      0.89      0.89        79

    accuracy                           0.90       170
   macro avg       0.90      0.90      0.90       170
weighted avg       0.90      0.90      0.90       170
```

In [108]:
```
print('Accuracy of Random Forest regression classifier on test set: {:.2f}'.format(rf.score(x_test, y_test)))
```

```
Accuracy of Random Forest regression classifier on test set: 0.90
```

In [109]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
rf_roc_auc = roc_auc_score(y_test, rf.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, rf.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Random Forest (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('RF_ROC')
print('AUC: %.3f' % rf_roc_auc)
plt.show()
```

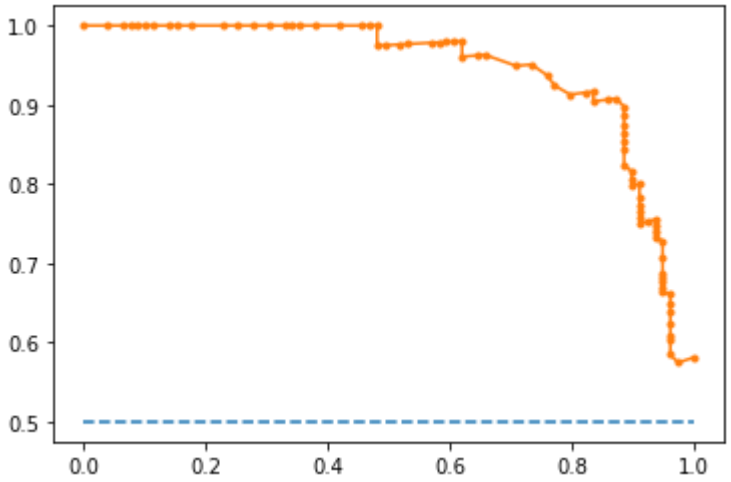
```
AUC: 0.899
```



In [110]:
```
#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 = rf.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = rf.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.892 auc=0.945 ap=0.944
```

Out[110]:
```
[<matplotlib.lines.Line2D at 0x57bdc2b0a0>]
```



Model 4 : Support Vector Machine

In [111]:
```
#Support Vector Classifier
from sklearn.svm import SVC
SVMmodel = SVC(kernel='rbf',gamma='auto', probability=True)
SVMmodel.fit(x_train,y_train)
```

Out[111]:
```
SVC(gamma='auto', probability=True)
```

In [112]:
```
print(SVMmodel.score(x_train,y_train))
print(SVMmodel.score(x_test,y_test))
```

```
1.0
0.8
```

In [113]:
```
y_pred = SVMmodel.predict(x_test)
```

In [114]:
```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[91  0]
 [34 45]]
```

In [115]:
```
print(classification_report(y_test, y_pred))
```

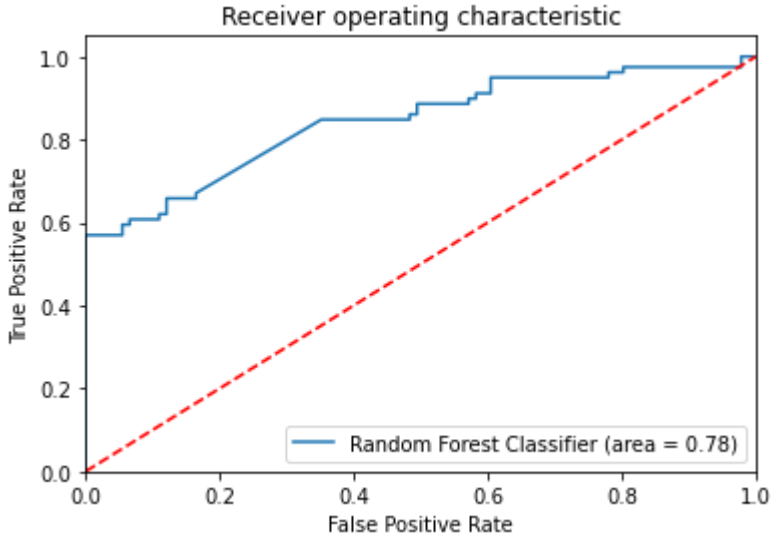
```
              precision    recall  f1-score   support

           0       0.73      1.00      0.84        91
           1       1.00      0.57      0.73        79

    accuracy                           0.80       170
   macro avg       0.86      0.78      0.78       170
weighted avg       0.85      0.80      0.79       170
```

In [116]:
```
print('Accuracy of Random Forest regression classifier on test set: {:.2f}'.format(SVMmodel.score(x_test, y_test
```

```
Accuracy of Random Forest regression classifier on test set: 0.80
```

In [117]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
rf_roc_auc = roc_auc_score(y_test, SVMmodel.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, SVMmodel.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Random Forest Classifier (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('SVM_ROC')
print('AUC: %.2f' % rf_roc_auc)
plt.show()
```

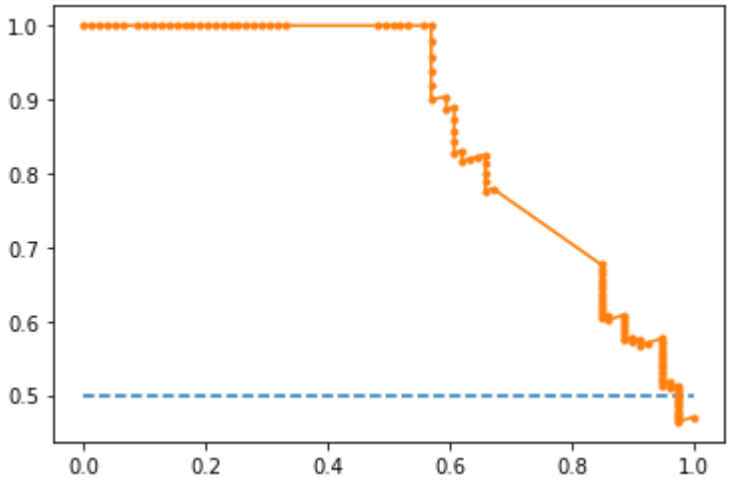
```
AUC: 0.78
```



In [118]:
```
#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 = SVMmodel.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = SVMmodel.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.726 auc=0.868 ap=0.860
```

Out[118]:
```
[<matplotlib.lines.Line2D at 0x57bdd011f0>]
```



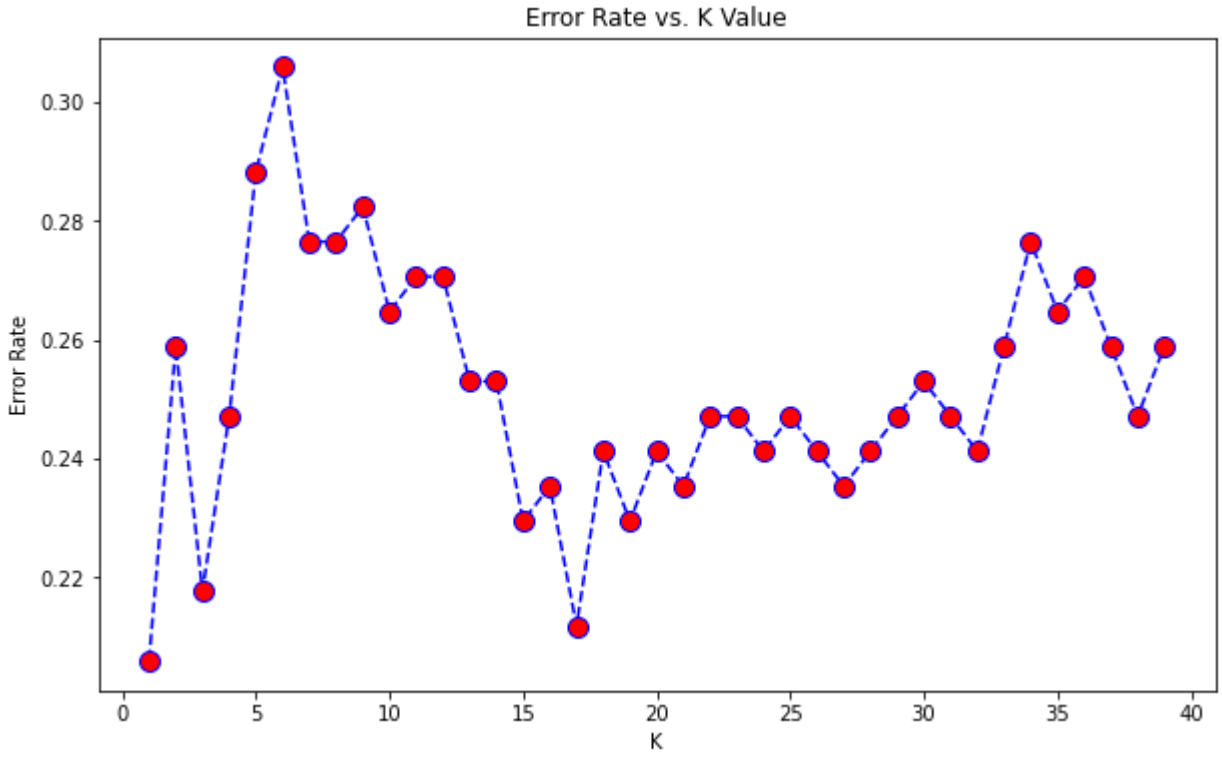
Model 5 : KNN Classifier

In [119]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
#Train Model and Predict
k = 4
neigh = KNeighborsClassifier(n_neighbors = k).fit(x_train,y_train)
Pred_y = neigh.predict(x_test)
print("Accuracy of model at K=4 is",metrics.accuracy_score(y_test, Pred_y))
print('we taken n_neighbors 4 arbitary')
```

```
Accuracy of model at K=4 is 0.7529411764705882
we taken n_neighbors 4 arbitary
```

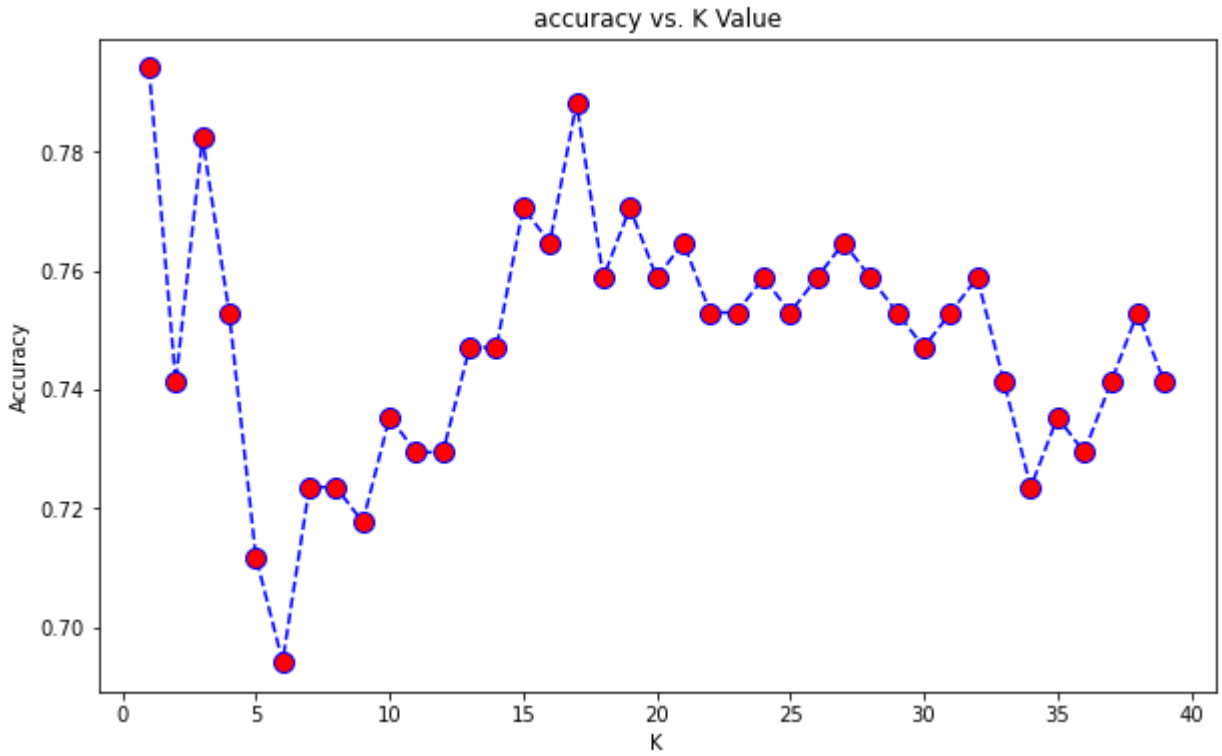
In [120]:
```
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(x_train,y_train)
 pred_i = knn.predict(x_test)
 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))
print('fromy plot graph between error and n_neighbors we saw at k = 0 the error is minimum')
```

```
Minimum error:- 0.20588235294117646 at K = 0
fromy plot graph between error and n_neighbors we saw at k = 0 the error is minimum
```



In [121]:
```
acc = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(x_train,y_train)
    yhat = neigh.predict(x_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))
print('to check maximum accuracy we plot graph between n_neighbors vs accuracy and we got at k= 0 maximum accura
```

```
Maximum accuracy:- 0.7941176470588235 at K = 0
to check maximum accuracy we plot graph between n_neighbors vs accuracy and we got at k= 0 maximum accuracy bu
t n_neighbores value must be greater than 0
```



In [122]:
```
#Applying K-NN
from sklearn.neighbors import KNeighborsClassifier
knnClassifier = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')
knnClassifier.fit(x_train,y_train)
```

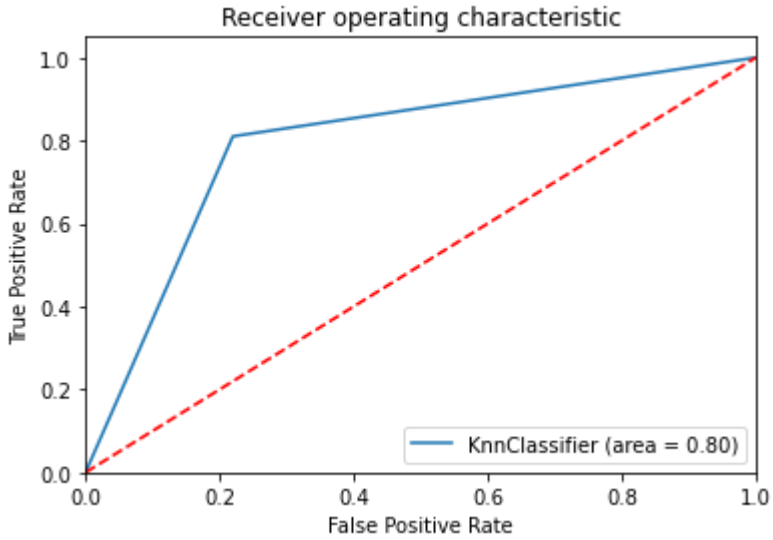
Out[122]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [123]:
```
print(knnClassifier.score(x_train,y_train))
print(knnClassifier.score(x_test,y_test))
```

```
1.0
0.7941176470588235
```

In [124]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
rf_roc_auc = roc_auc_score(y_test, knnClassifier.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, knnClassifier.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='KnnClassifier (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Knn_ROC')
print('AUC: %.2f' % rf_roc_auc)
plt.show()
```

```
AUC: 0.80
```



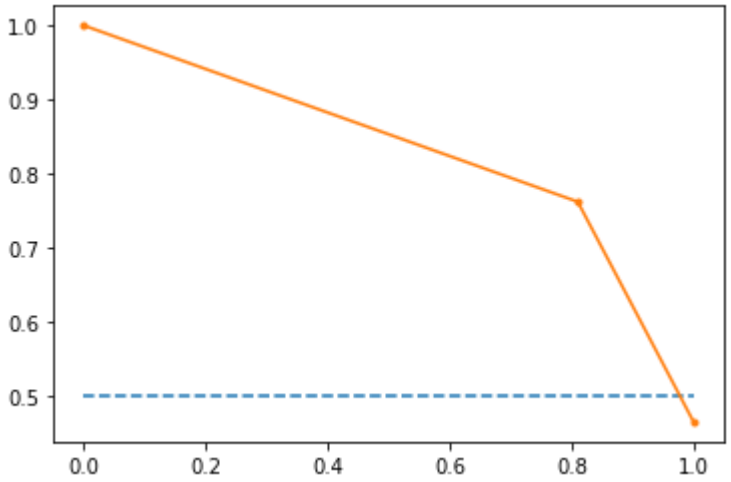
In [125]:
```
print('Accuracy of KNN classifier on test set: {:.2f}'.format(knnClassifier.score(x_test, y_test)))
```

```
Accuracy of KNN classifier on test set: 0.79
```

In [126]:
```
#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 = knnClassifier.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = knnClassifier.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.785 auc=0.830 ap=0.705
```

Out[126]:
```
[<matplotlib.lines.Line2D at 0x57bde6f6d0>]
```



Model 6 : Naive Bayes

In [127]:
```
from sklearn.naive_bayes import GaussianNB
```

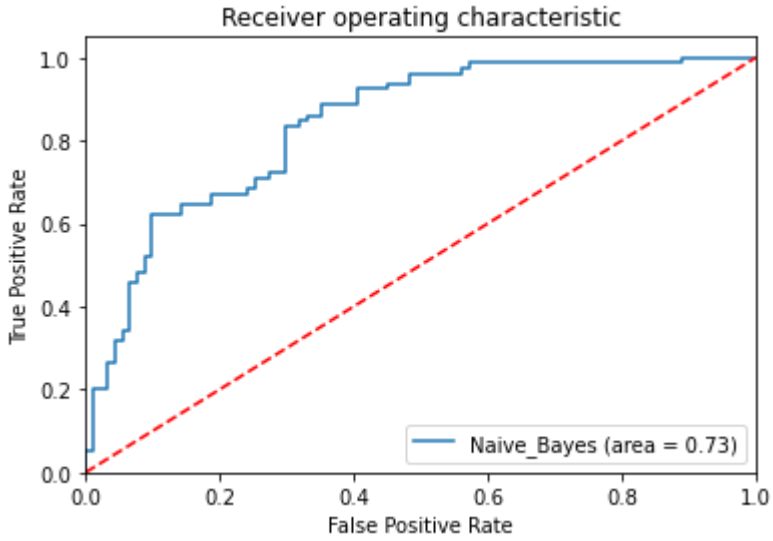
In [128]:
```
model_NB = GaussianNB()
model_NB.fit(x_train,y_train)
y_pred = model_NB.predict(x_test)
```

In [129]:
```
print(model_NB.score(x_test,y_test))
print(model_NB.score(x_train,y_train))
```

```
0.7411764705882353
0.7323529411764705
```

In [130]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
rf_roc_auc = roc_auc_score(y_test, model_NB.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, model_NB.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Naive_Bayes (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('NaiveBayes_ROC')
print('AUC: %.2f' % rf_roc_auc)
plt.show()
```

```
AUC: 0.73
```



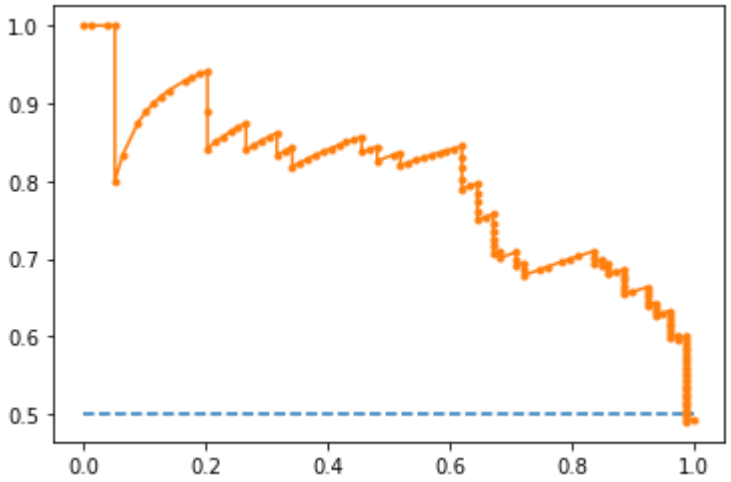
In [131]:
```
print('Accuracy of Naive Bayes on test set: {:.2f}'.format(model_NB.score(x_test, y_test)))
```

```
Accuracy of Naive Bayes on test set: 0.74
```

In [132]:
```
#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 = model_NB.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model_NB.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.699 auc=0.798 ap=0.801
```

Out[132]:
```
[<matplotlib.lines.Line2D at 0x57bf03ee20>]
```



MODEL 7 : XGBOOST

In [133]:
```
from xgboost.sklearn import XGBClassifier
```

In [134]:
```
model_XGB = XGBClassifier (learning_rate =.1, n_estimators=500,
max_depth=2, min_child_weight=3, gamma=0,
subsample=.8, colsample_bytree=.7, reg_alpha=1,
objective= 'reg:linear')
```

In [135]:
```
model_XGB.fit(x_train,y_train)
y_pred = model_XGB.predict(x_test)
```

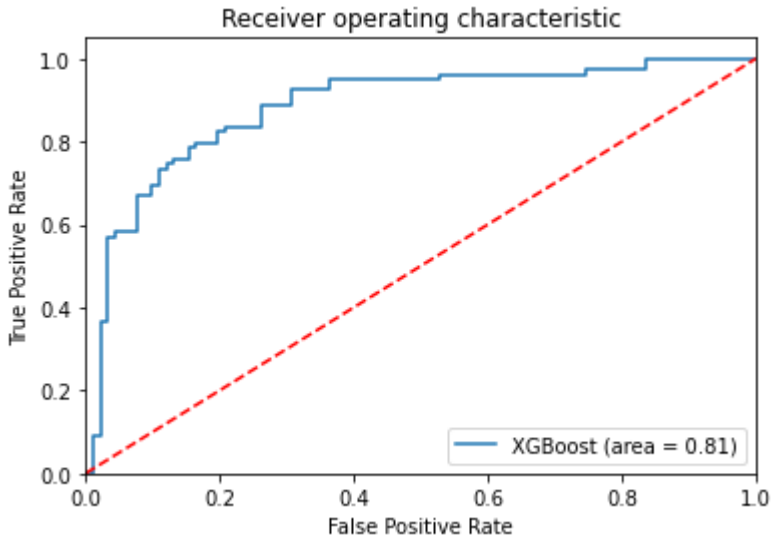
```
[13:29:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/objective/regression_obj.
cu:203: reg:linear is now deprecated in favor of reg:squarederror.
```

In [136]:
```
print(model_XGB.score(x_test,y_test))
print(model_XGB.score(x_train,y_train))
```

```
0.8117647058823529
0.9514705882352941
```

In [137]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
rf_roc_auc = roc_auc_score(y_test, model_XGB.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, model_XGB.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='XGBoost (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('XGBoost_ROC')
print('AUC: %.2f' % rf_roc_auc)
plt.show()
```

```
AUC: 0.81
```



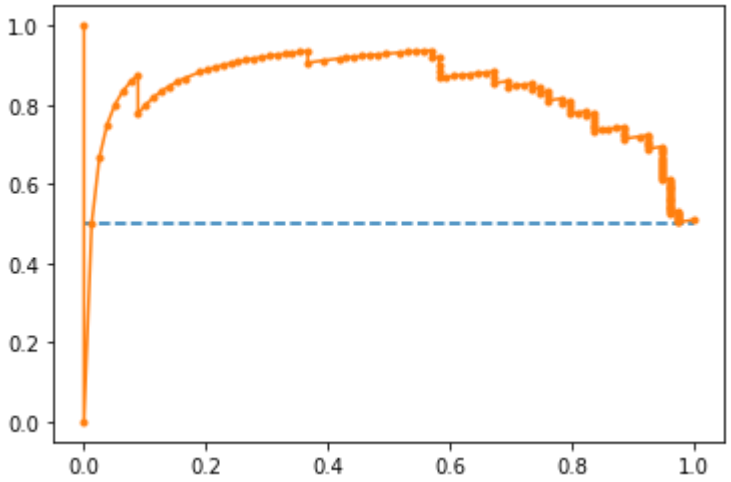
In [138]:
```
print('Accuracy of Naive Bayes on test set: {:.2f}'.format(model_XGB.score(x_test, y_test)))
```

```
Accuracy of Naive Bayes on test set: 0.81
```

In [139]:
```
#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 = model_XGB.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model_XGB.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
# calculate F1 score
f1 = f1_score(y_test, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, 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=0.802 auc=0.830 ap=0.837
```

Out[139]:
```
[<matplotlib.lines.Line2D at 0x57c0864f70>]
```



Comparison Between All model Test data Accuracy Score

In [140]:
```
print('Accuracy of KNN classifier on test set: {:.0f}%'.format(knnClassifier.score(x_test, y_test)*100))
print('Accuracy of Random Forest regression classifier on test set: {:.0f}%'.format(SVMmodel.score(x_test, y_tes
print('Accuracy of Random Forest regression classifier on test set: {:.0f}%'.format(rf.score(x_test, y_test)*100
print('Accuracy of Decision Tree regression classifier on test set: {:.0f}%'.format(DTModel.score(x_test, y_test
print('Accuracy of logistic regression classifier on test set: {:.0f}%'.format(logit_model.score(x_test, y_test)
print('Accuracy of Naive Bayes on test set: {:.2f}'.format(model_NB.score(x_test, y_test)*100))
print('Accuracy of Naive Bayes on test set: {:.2f}'.format(model_XGB.score(x_test, y_test)*100))
```

```
Accuracy of KNN classifier on test set: 79%
Accuracy of Random Forest regression classifier on test set: 80%
Accuracy of Random Forest regression classifier on test set: 90%
Accuracy of Decision Tree regression classifier on test set: 82%
Accuracy of logistic regression classifier on test set: 76%
Accuracy of Naive Bayes on test set: 74.12
Accuracy of Naive Bayes on test set: 81.18
```

We observed that Random Forest is Best Performing Model for this Data :

Accuracy of 89%

Precision = 0.86

Recall = 0.89

AUC = 0.88

Thank You..