Project 2 - Health Care

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

Project Task: Week 1

Data Exploration:

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:
    - `Glucose`
    - `BloodPressure`
    - `SkinThickness`
    - `Insulin`
    - `BMI`
2. Visually explore these variables using histograms. Treat the missing values accordingly.
3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

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.isnull().sum()
```

Out[4]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

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

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

Out[6]:
```
(768, 9)
```

In [7]:
```
df = data[data['Outcome']==1]
```

In [8]:
```
df.head()
```

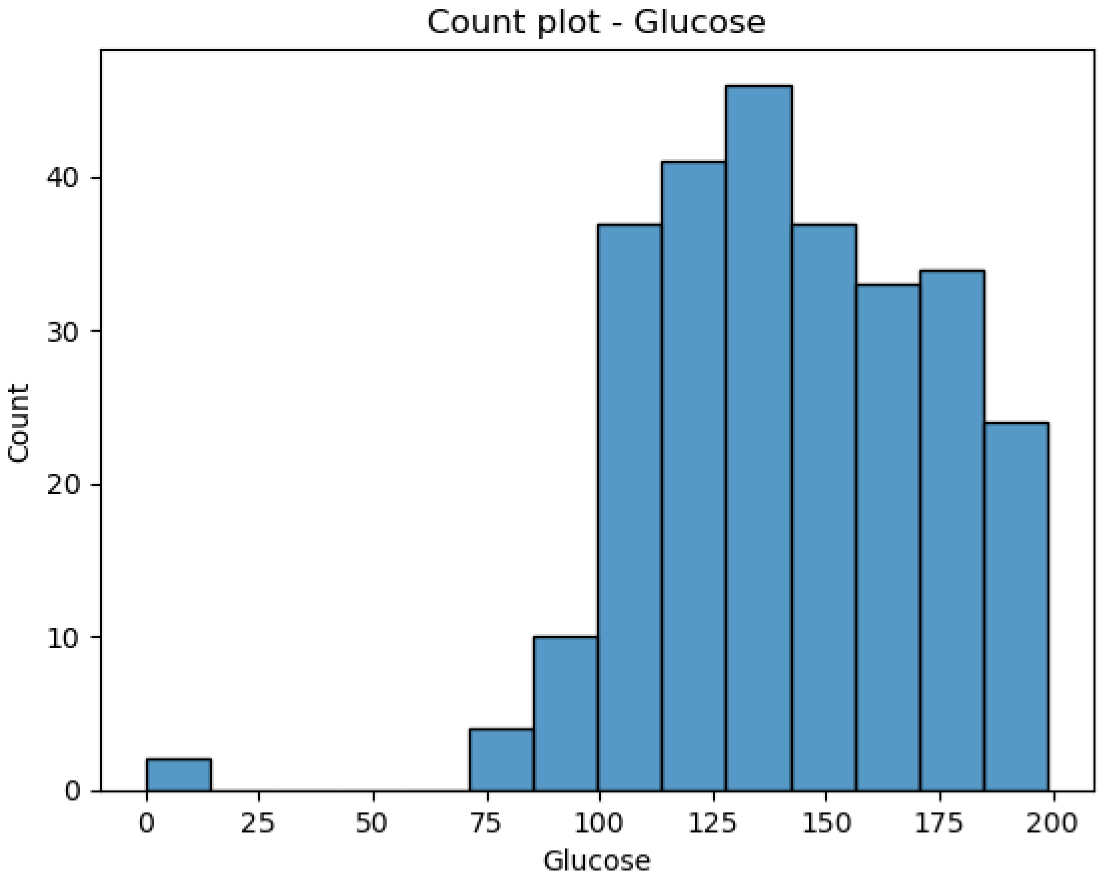
Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1

In [9]:
```
df.shape
```
Out[9]:
```
(268, 9)
```

In [10]:
```
data['Glucose'].value_counts()
```
Out[10]:
```
99     17
100    17
111    14
129    14
125    14
       ..
191     1
177     1
44      1
62      1
190     1
Name: Glucose, Length: 136, dtype: int64
```

In [11]:
```
sns.histplot(df['Glucose'])
plt.title('Count plot - Glucose')
plt.show()
```

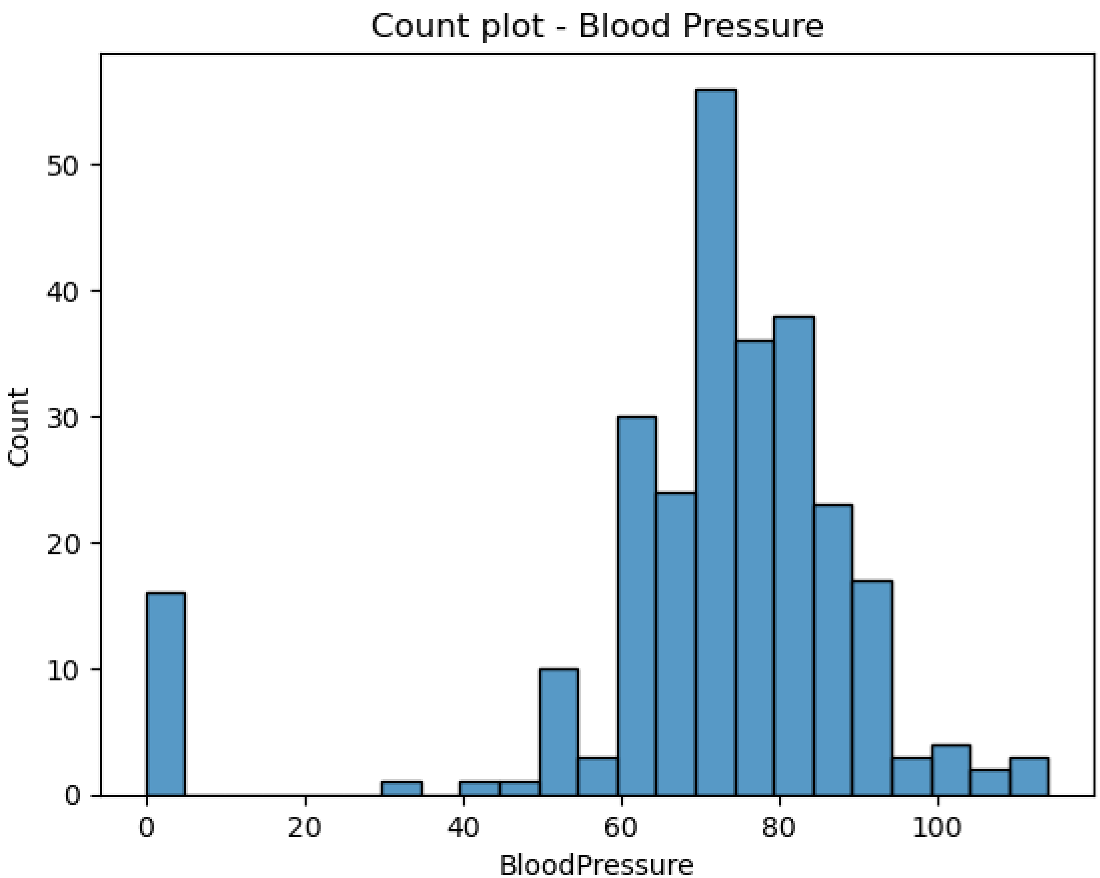


In [12]:
```
data['BloodPressure'].value_counts()
```

Out[12]:
```
70     57
74     52
78     45
68     45
72     44
64     43
80     40
76     39
60     37
0      35
62     34
66     30
82     30
88     25
84     23
90     22
86     21
58     21
50     13
56     12
52     11
54     11
75      8
92      8
65      7
85      6
94      6
48      5
96      4
44      4
100     3
106     3
98      3
110     3
55      2
108     2
104     2
46      2
30      2
122     1
95      1
```

```
102     1
61      1
24      1
38      1
40      1
114     1
Name: BloodPressure, dtype: int64
```

In [13]:
```
sns.histplot(df['BloodPressure'])
plt.title('Count plot - Blood Pressure')
plt.show()
```

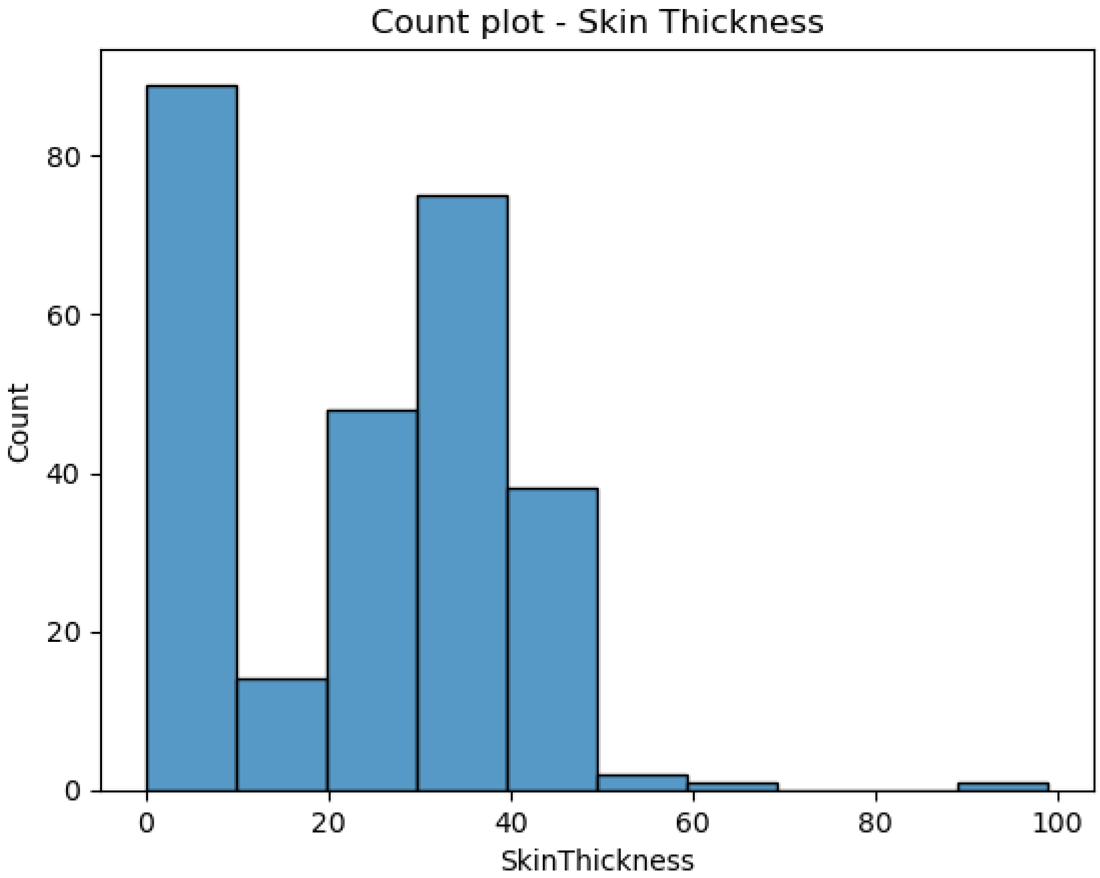


In [14]:
```
data['SkinThickness'].value_counts()
```

Out[14]:
```
0     227
32     31
30     27
27     23
23     22
33     20
28     20
18     20
31     19
19     18
39     18
29     17
40     16
25     16
26     16
22     16
37     16
41     15
35     15
36     14
15     14
17     14
20     13
24     12
42     11
13     11
21     10
46      8
34      8
12      7
38      7
11      6
43      6
16      6
45      6
14      6
44      5
10      5
48      4
47      4
49      3
```

```
50      3
8       2
7       2
52      2
54      2
63      1
60      1
56      1
51      1
99      1
Name: SkinThickness, dtype: int64
```

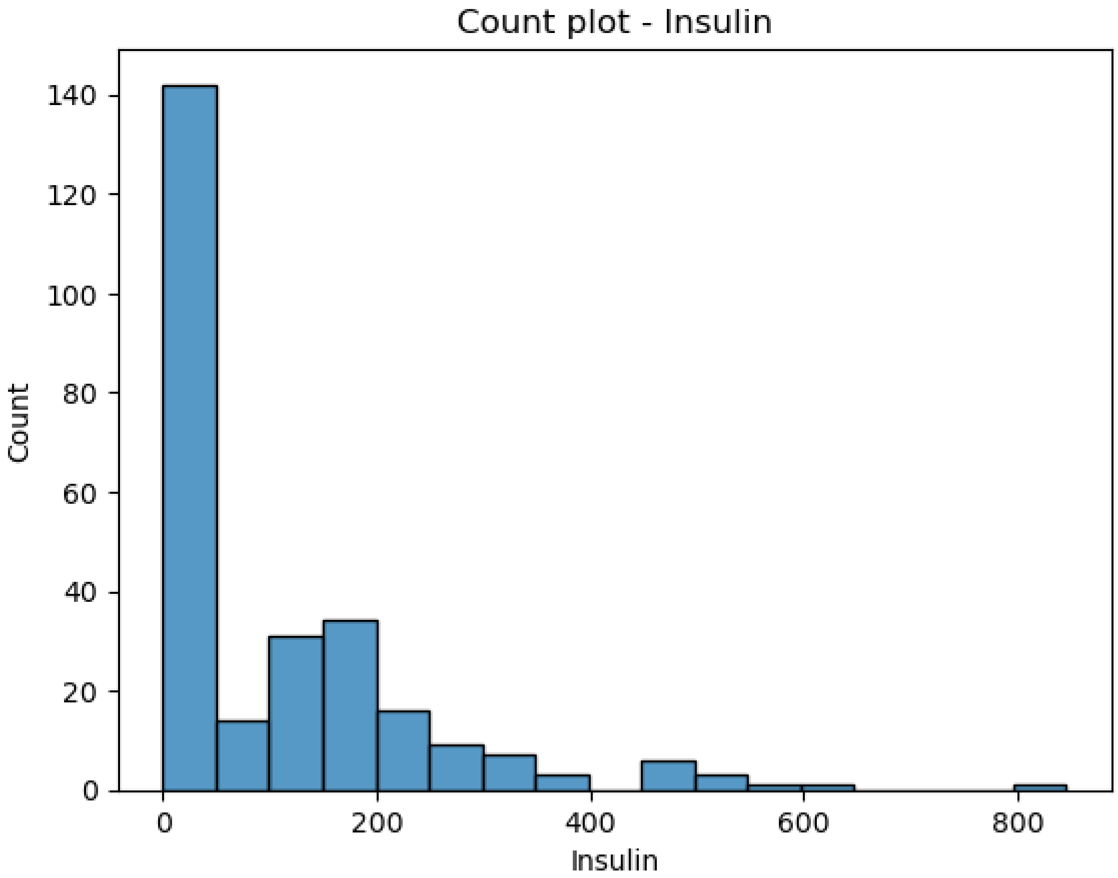
In [15]:
```
sns.histplot(df['SkinThickness'])
plt.title('Count plot - Skin Thickness')
plt.show()
```



In [16]:
```
data['Insulin'].value_counts()
```

Out[16]:
```
0      374
105     11
130      9
140      9
120      8
      ... 
73       1
171      1
255      1
52       1
112      1
Name: Insulin, Length: 186, dtype: int64
```

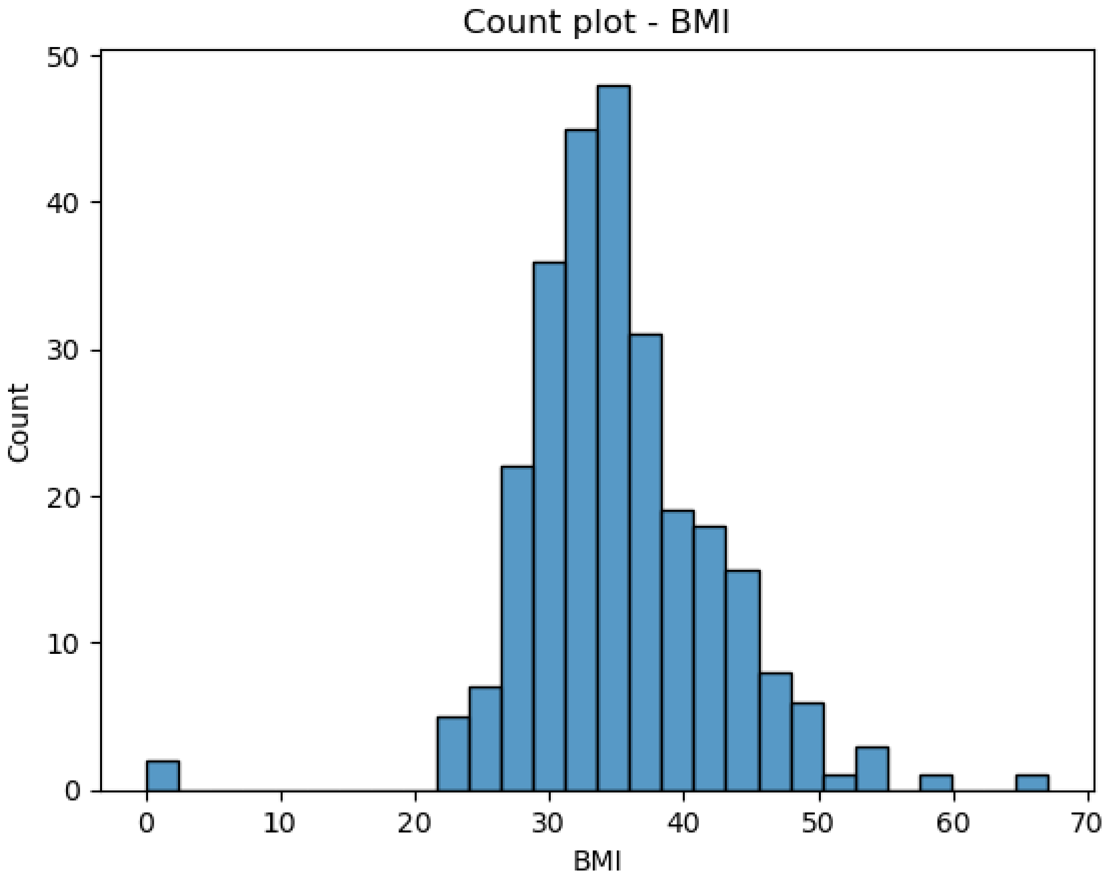
In [17]:
```
sns.histplot(df['Insulin'])
plt.title('Count plot - Insulin')
plt.show()
```



In [18]:
```
data['BMI'].value_counts()
```

Out[18]:
```
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
        ..
36.7     1
41.8     1
42.6     1
42.8     1
46.3     1
Name: BMI, Length: 248, dtype: int64
```

In [19]:
```
sns.histplot(df['BMI'])
plt.title('Count plot - BMI')
plt.show()
```



In [20]:
```
data.describe()
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Project Task: Week 2

Data Exploration:

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

In [21]:
```
df.describe()
```

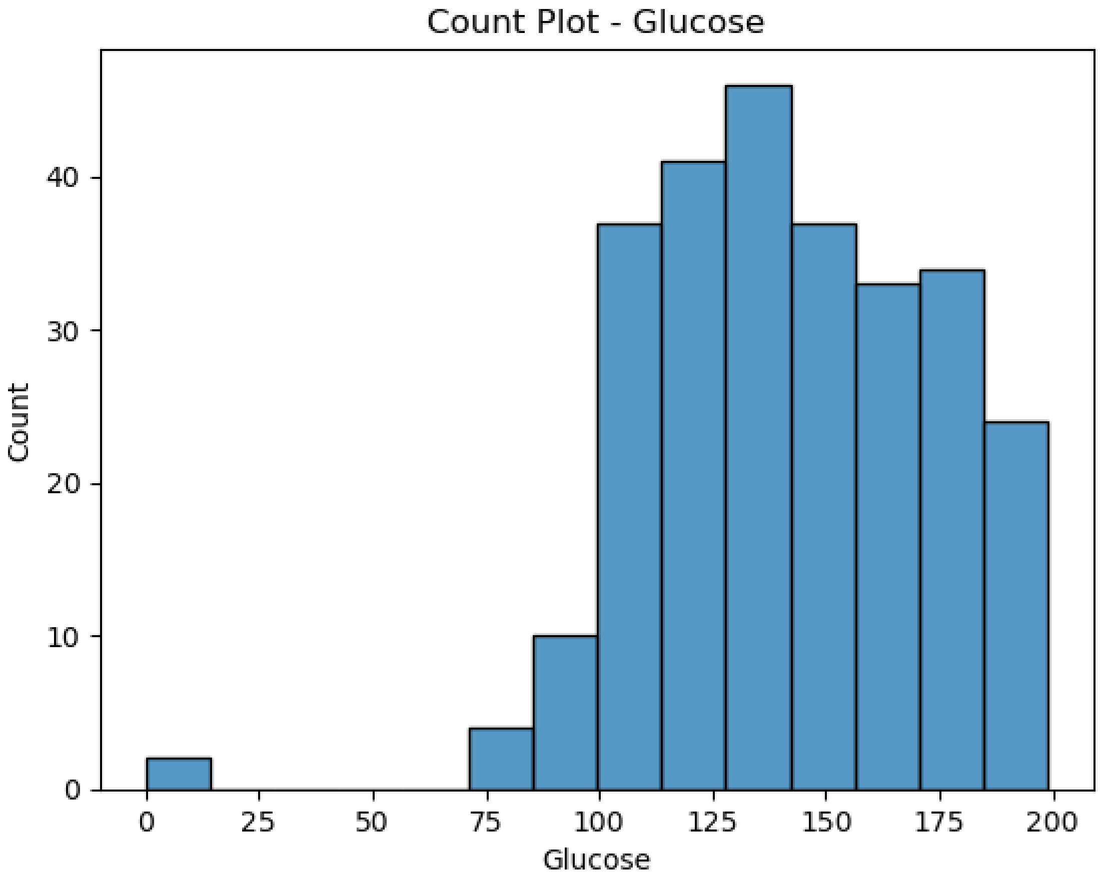
Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	268.000000	268.000000	268.000000	268.000000	268.000000	268.000000	268.000000	268.000000	268.0
mean	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164	1.0
std	3.741239	31.939622	21.491812	17.679711	138.689125	7.262967	0.372354	10.968254	0.0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.088000	21.000000	1.0
25%	1.750000	119.000000	66.000000	0.000000	0.000000	30.800000	0.262500	28.000000	1.0
50%	4.000000	140.000000	74.000000	27.000000	0.000000	34.250000	0.449000	36.000000	1.0
75%	8.000000	167.000000	82.000000	36.000000	167.250000	38.775000	0.728000	44.000000	1.0
max	17.000000	199.000000	114.000000	99.000000	846.000000	67.100000	2.420000	70.000000	1.0

In [22]:
```
df['Glucose'].value_counts()
```

Out[22]:
```
125    7
128    6
129    6
115    6
158    6
      ..
165    1
116    1
193    1
172    1
190    1
Name: Glucose, Length: 104, dtype: int64
```

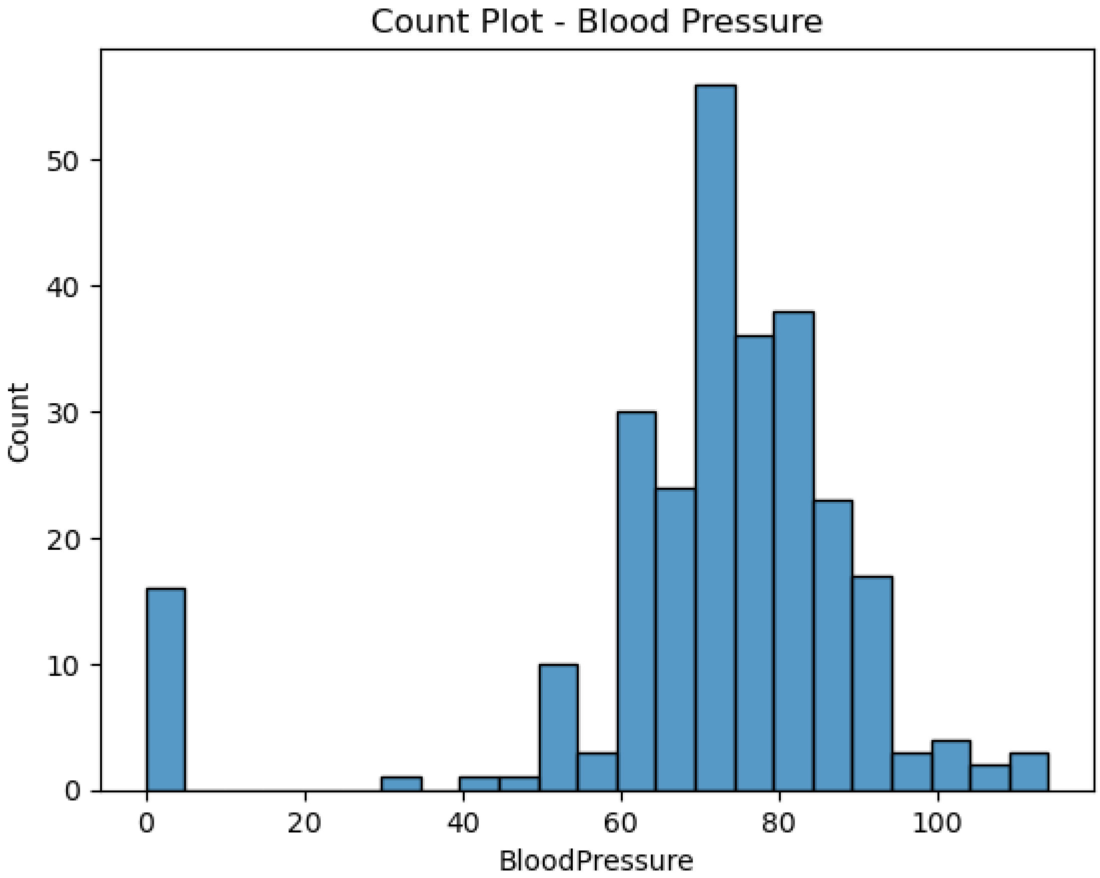
In [23]:
```
sns.histplot(df['Glucose'])
plt.title('Count Plot - Glucose')
plt.show()
```



In [24]:
```
df['BloodPressure'].value_counts()
```

Out[24]:
```
70     23
76     18
78     17
74     17
72     16
0      16
80     13
64     13
82     13
84     12
68     12
66     11
88     11
90     11
62     10
86      9
60      7
50      5
52      3
92      3
85      3
94      3
54      2
110     2
98      2
104     2
58      2
48      1
106     1
100     1
75      1
102     1
65      1
40      1
108     1
96      1
56      1
30      1
114     1
Name: BloodPressure, dtype: int64
```

In [25]:
```
sns.histplot(df['BloodPressure'])
plt.title('Count Plot - Blood Pressure')
plt.show()
```



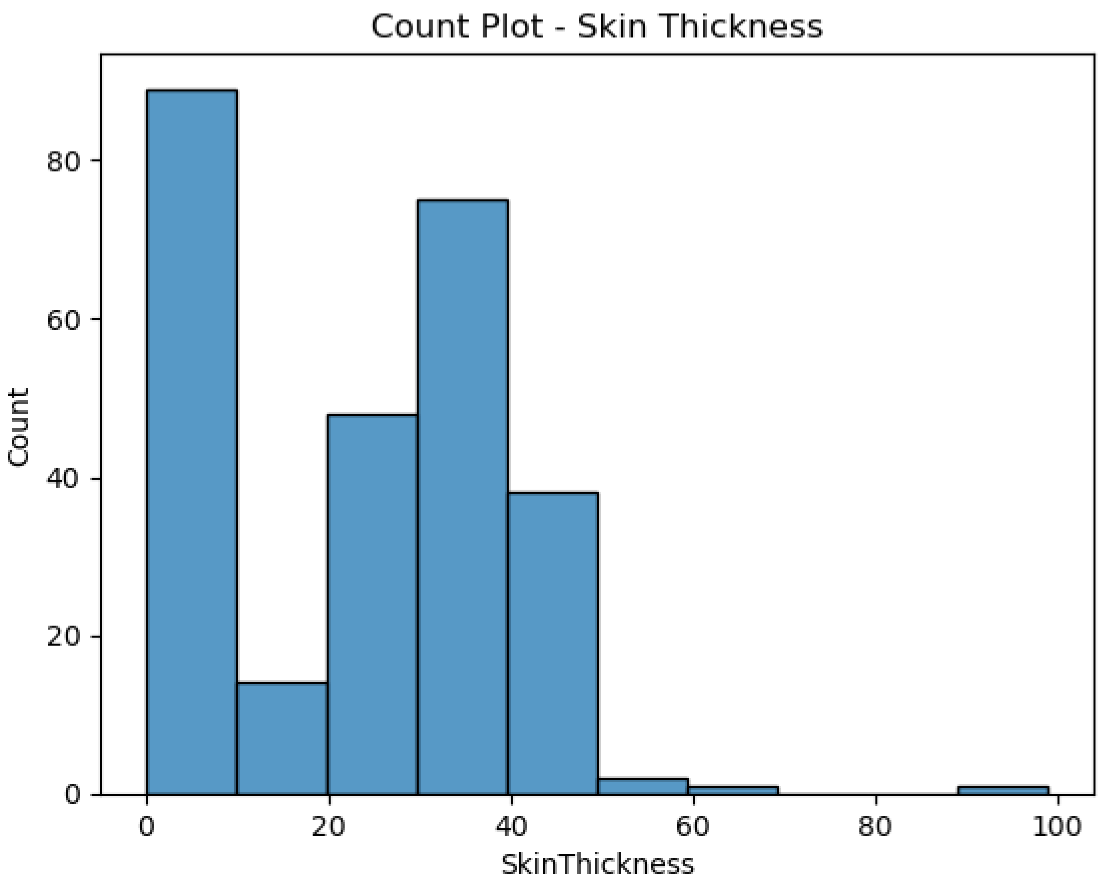
In [26]:
```
df['SkinThickness'].value_counts()
```

Out[26]:
```
0     88
32    14
30     9
33     9
39     8
37     8
36     8
35     8
27     7
29     7
41     7
42     6
24     6
31     6
26     6
25     5
40     5
46     5
28     5
22     4
23     4
18     4
19     3
34     3
49     3
45     3
44     3
47     2
48     2
20     2
14     2
21     2
43     2
17     2
38     2
12     1
63     1
56     1
7      1
15     1
13     1
```

```
51     1
99     1
Name: SkinThickness, dtype: int64
```

In [27]:

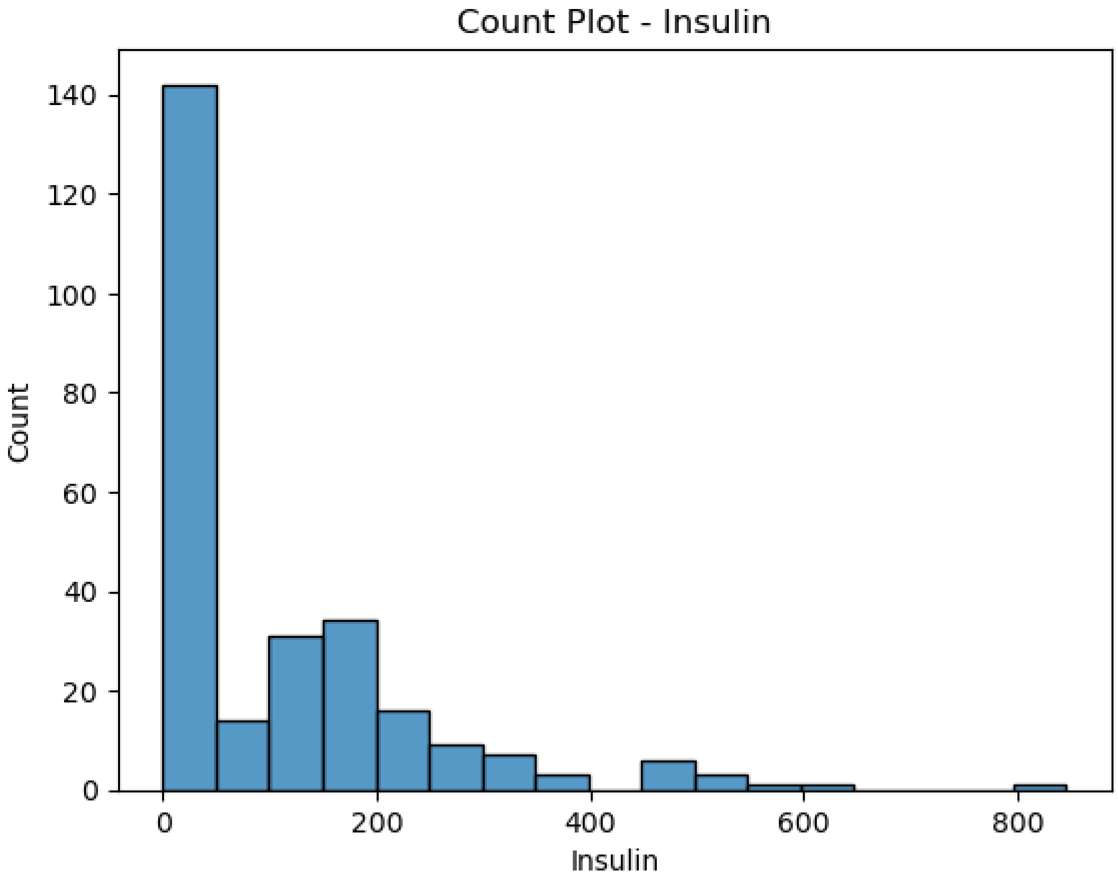
```
sns.histplot(df['SkinThickness'])
plt.title('Count Plot - Skin Thickness')
plt.show()
```



In [28]:
```
df['Insulin'].value_counts()
```

Out[28]:
```
0      138
130      6
180      4
175      3
156      3
      ... 
29       1
171      1
249      1
140      1
510      1
Name: Insulin, Length: 93, dtype: int64
```

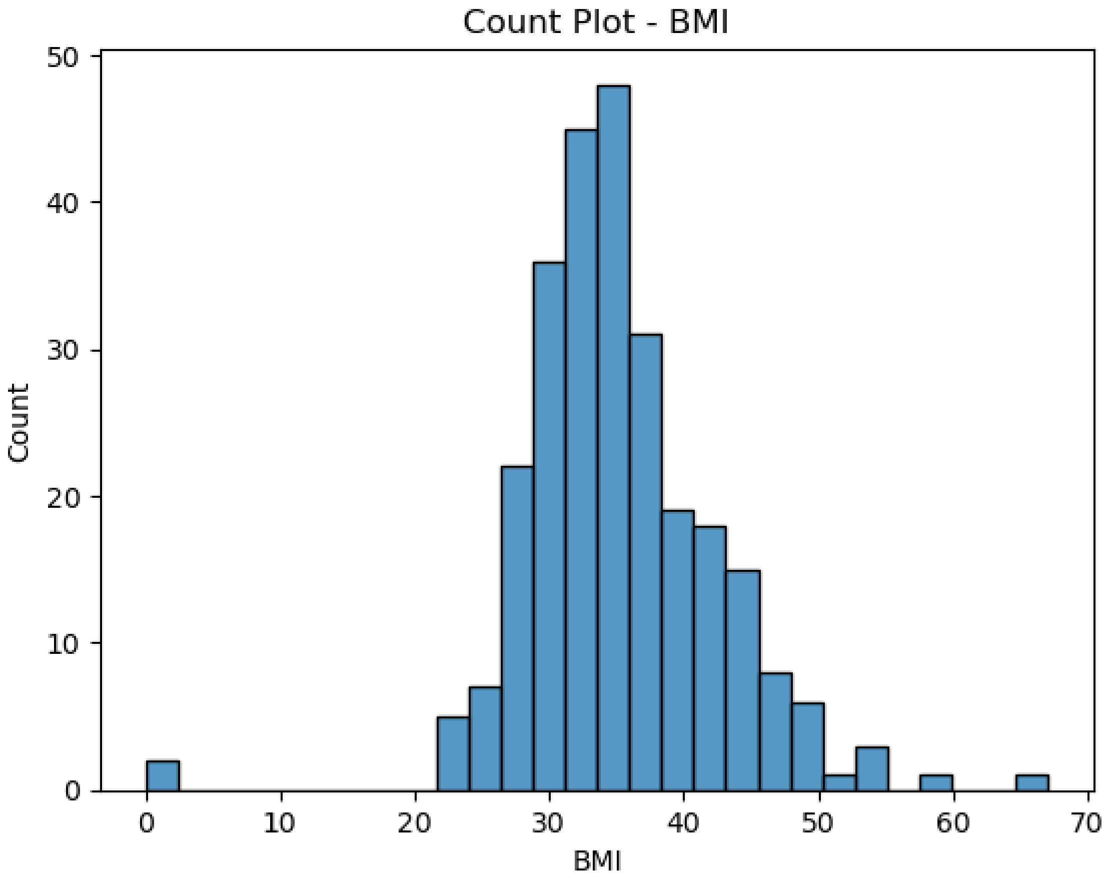
In [29]:
```
sns.histplot(df['Insulin'])
plt.title('Count Plot - Insulin')
plt.show()
```



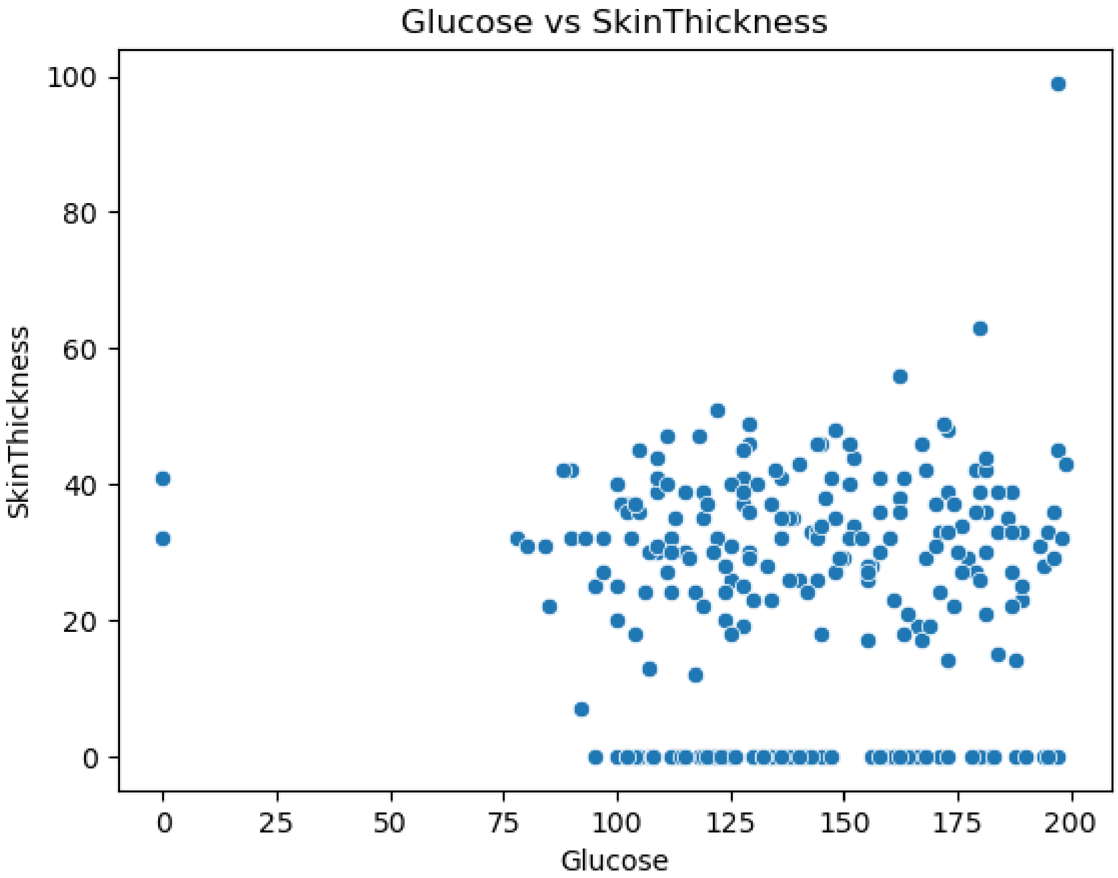
In [30]:
```
df['BMI'].value_counts()
```

Out[30]:
```
32.9    8
31.6    7
33.3    6
31.2    5
30.5    5
       ..
37.2    1
35.8    1
41.8    1
42.6    1
24.3    1
Name: BMI, Length: 148, dtype: int64
```

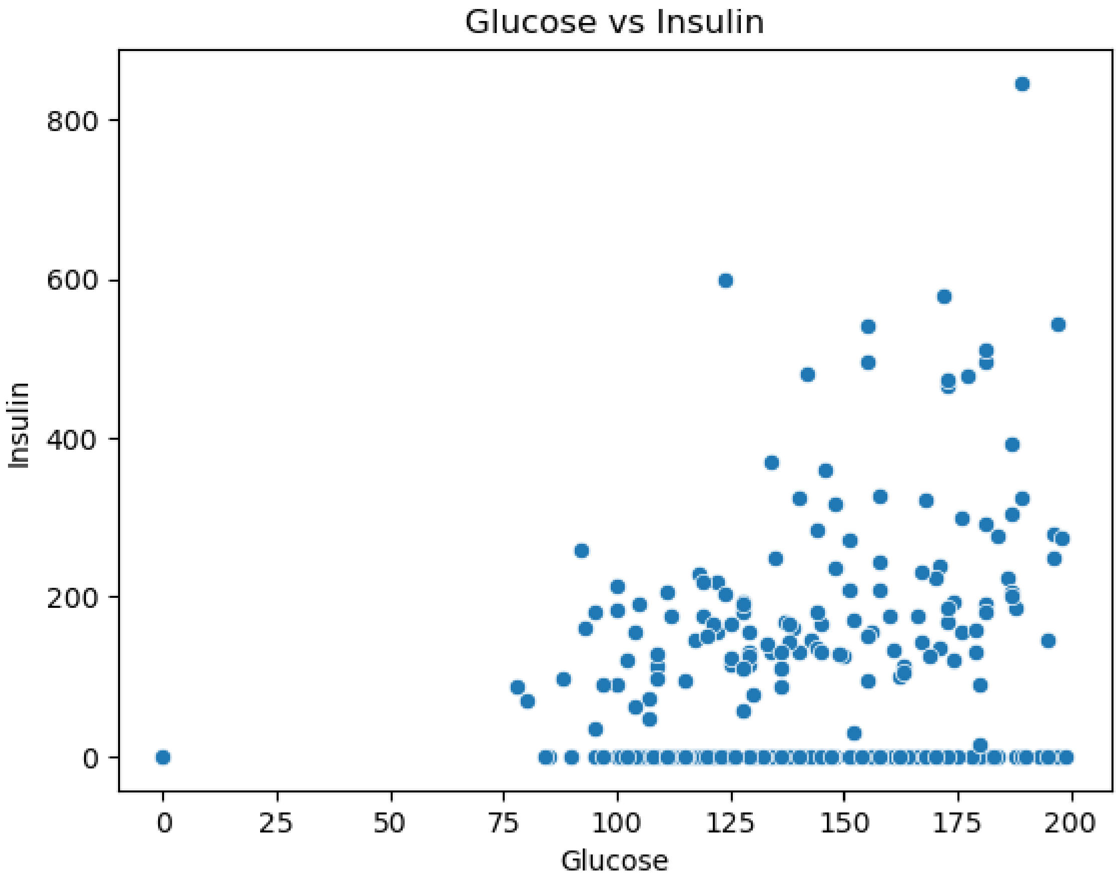
In [31]:
```
sns.histplot(df['BMI'])
plt.title('Count Plot - BMI')
plt.show()
```



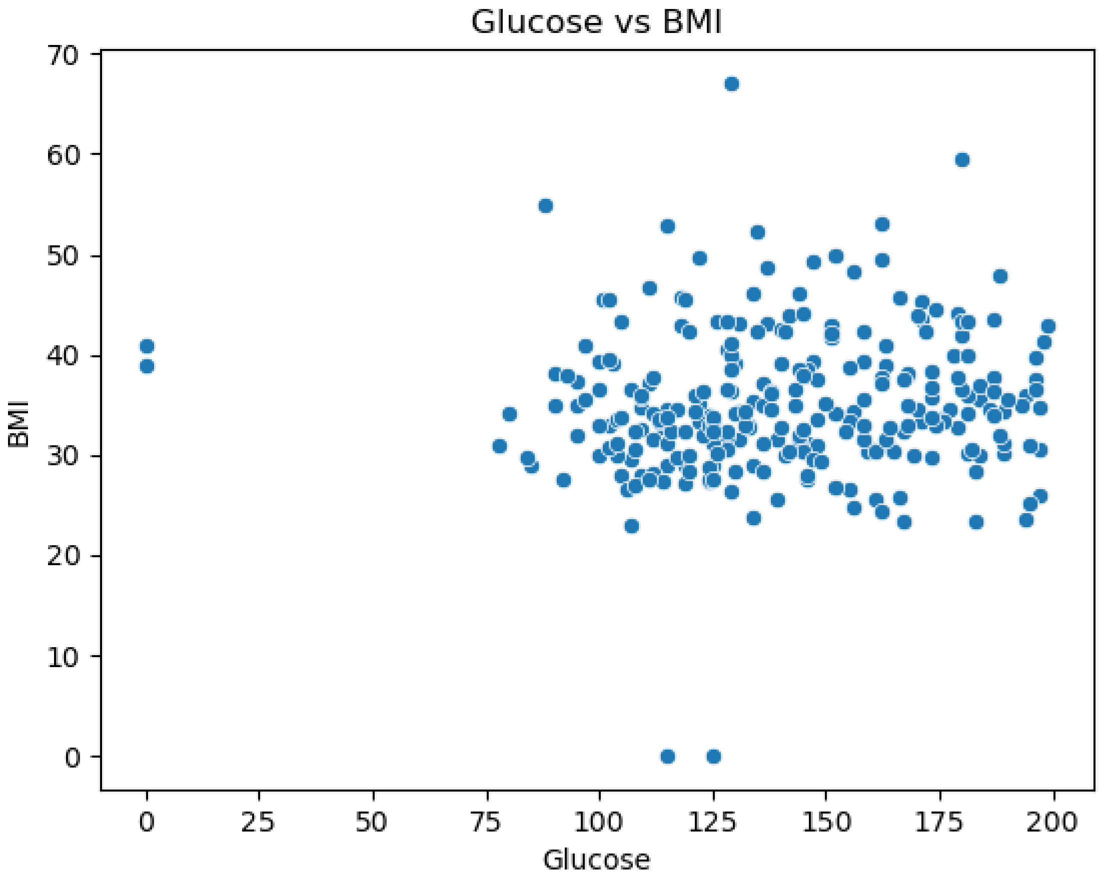
In [32]:
```
sns.scatterplot(x=df['Glucose'],y=df['SkinThickness'])
plt.title('Glucose vs SkinThickness')
plt.show()
```



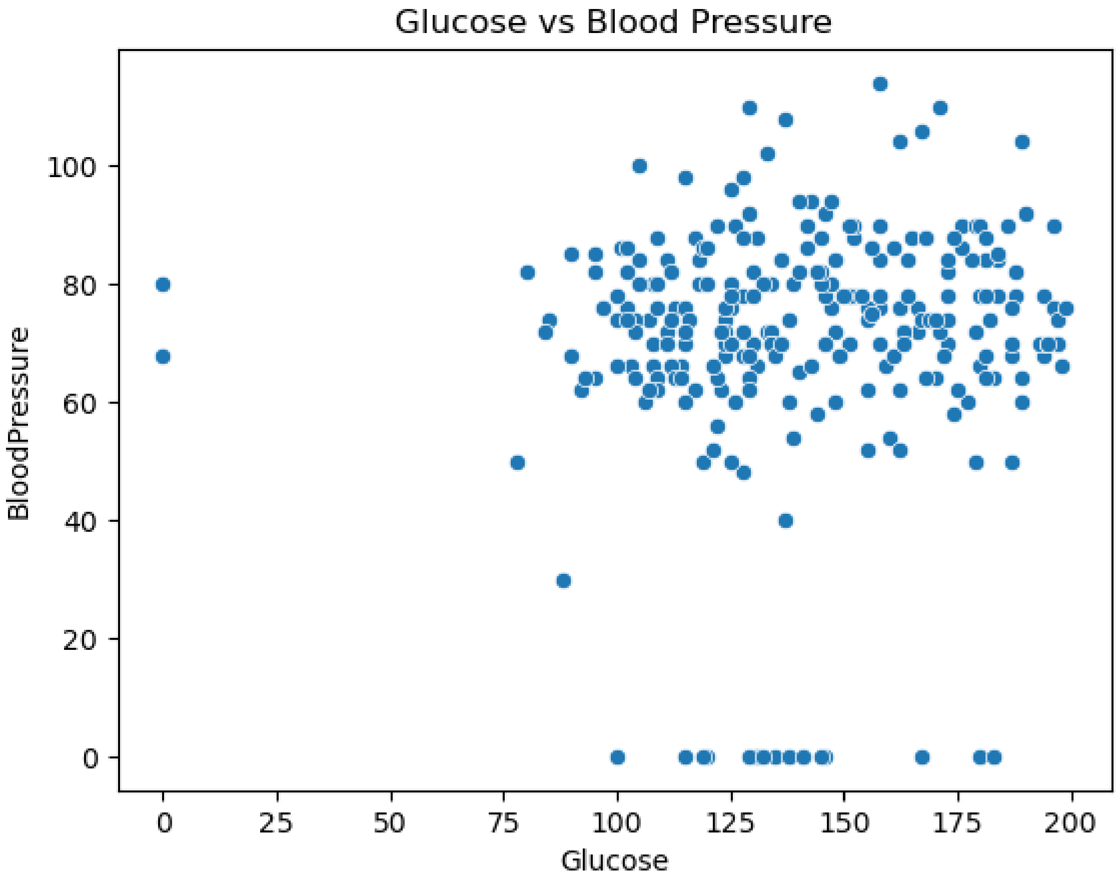
In [33]:
```
sns.scatterplot(x=df['Glucose'],y=df['Insulin'])
plt.title('Glucose vs Insulin')
plt.show()
```



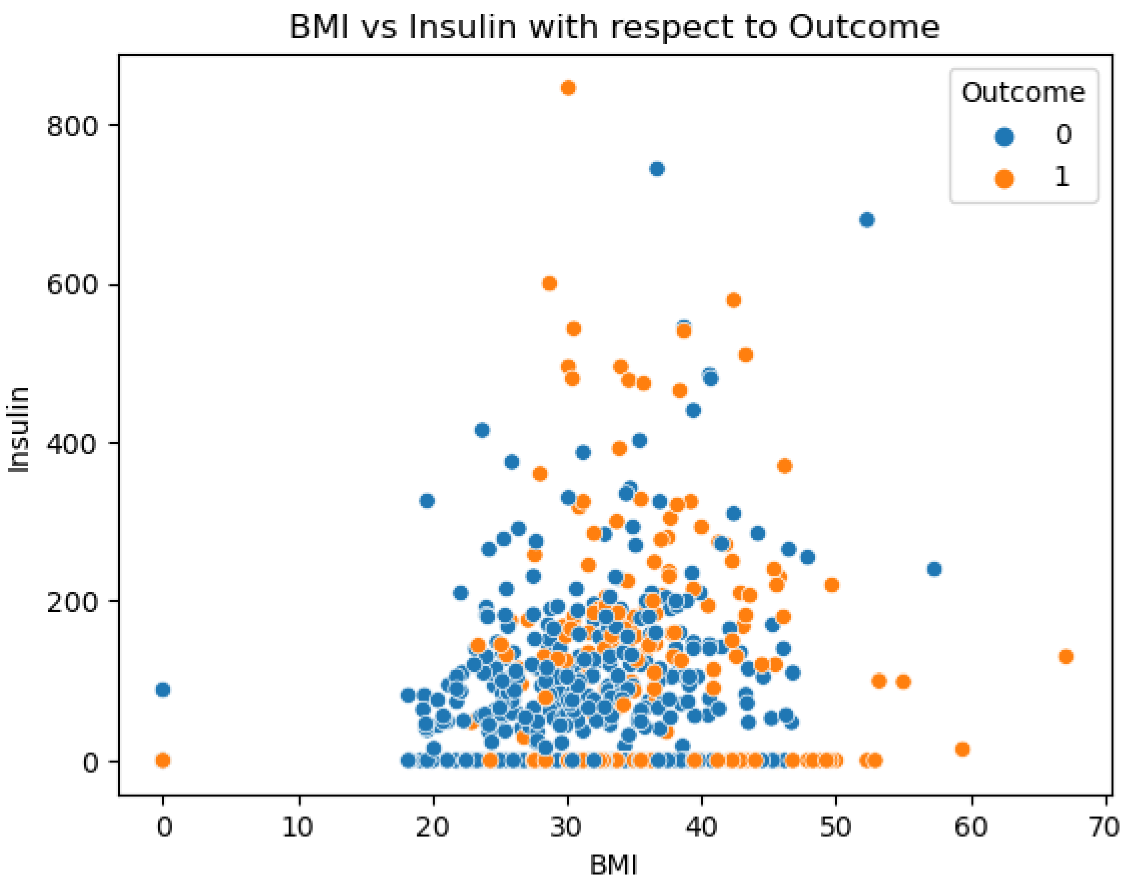
In [34]:
```
sns.scatterplot(x=df['Glucose'],y=df['BMI'])
plt.title('Glucose vs BMI')
plt.show()
```



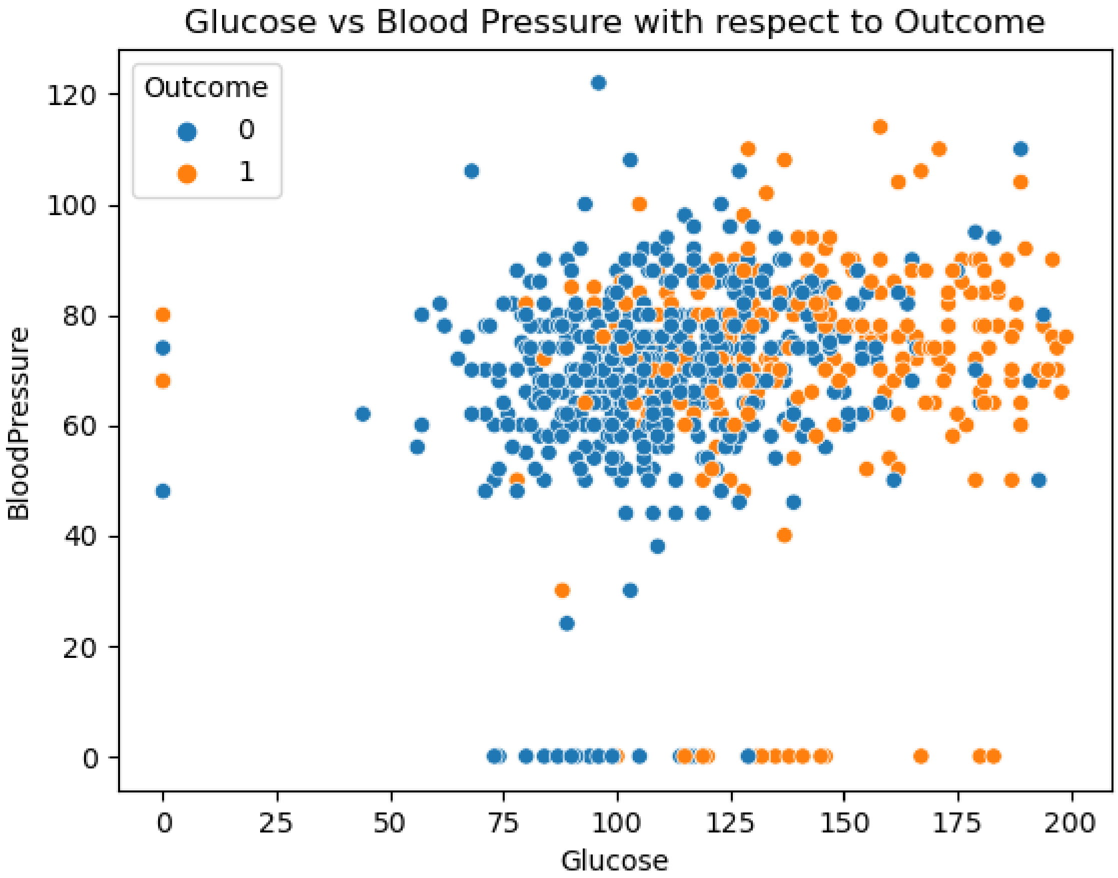
In [35]:
```
sns.scatterplot(x=df['Glucose'],y=df['BloodPressure'])
plt.title('Glucose vs Blood Pressure')
plt.show()
```



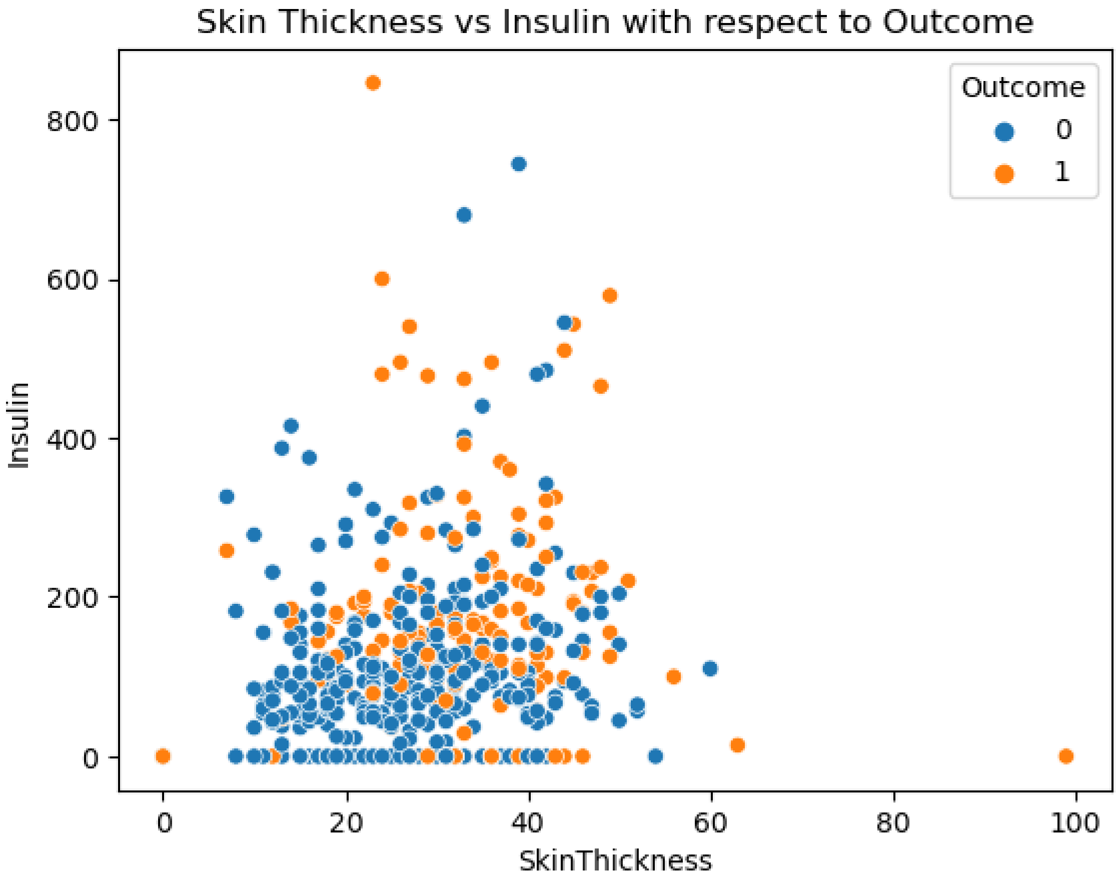
In [36]:
```
sns.scatterplot(x=data['BMI'],y=data['Insulin'],hue=data['Outcome'])
plt.title('BMI vs Insulin with respect to Outcome')
plt.show()
```



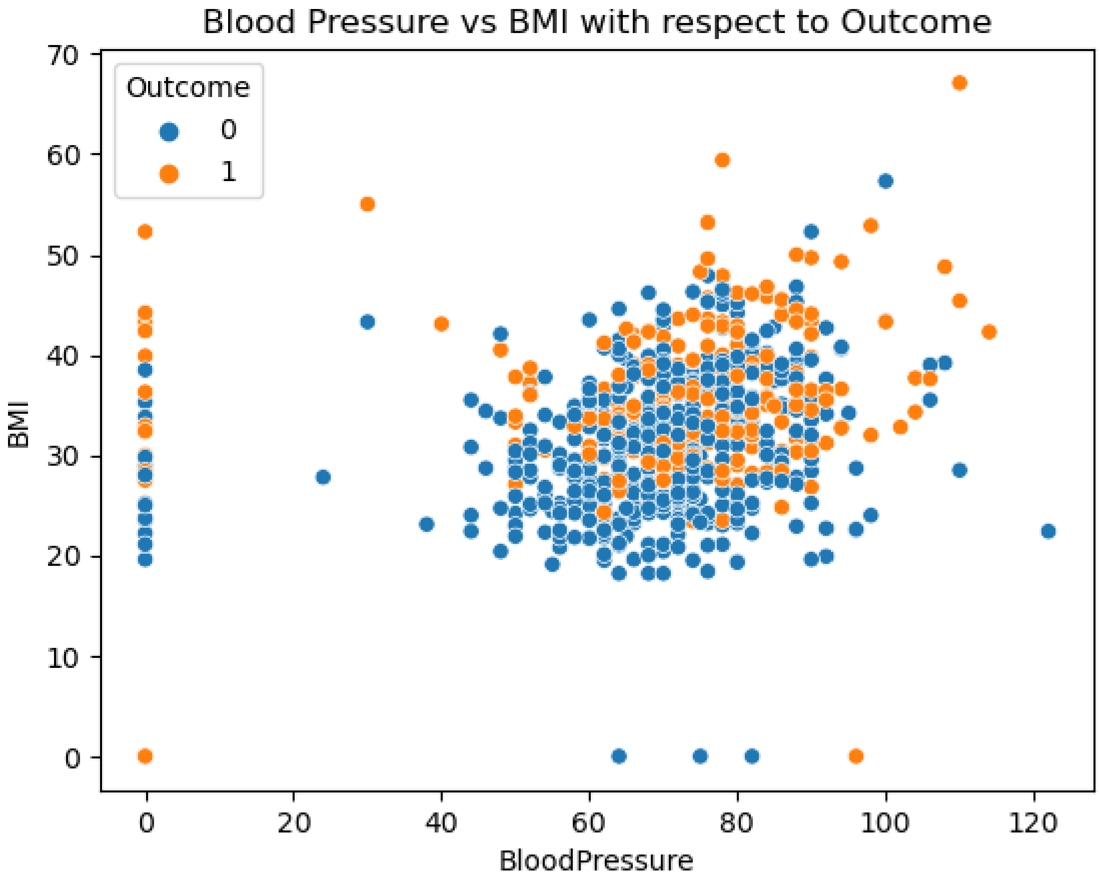
In [37]:
```
sns.scatterplot(x=data['Glucose'],y=data['BloodPressure'],hue=data['Outcome'])
plt.title('Glucose vs Blood Pressure with respect to Outcome')
plt.show()
```



In [38]:
```
sns.scatterplot(x=data['SkinThickness'],y=data['Insulin'],hue=data['Outcome'])
plt.title('Skin Thickness vs Insulin with respect to Outcome')
plt.show()
```



In [39]:
```
sns.scatterplot(x=data['BloodPressure'],y=data['BMI'],hue=data['Outcome'])
plt.title('Blood Pressure vs BMI with respect to Outcome')
plt.show()
```



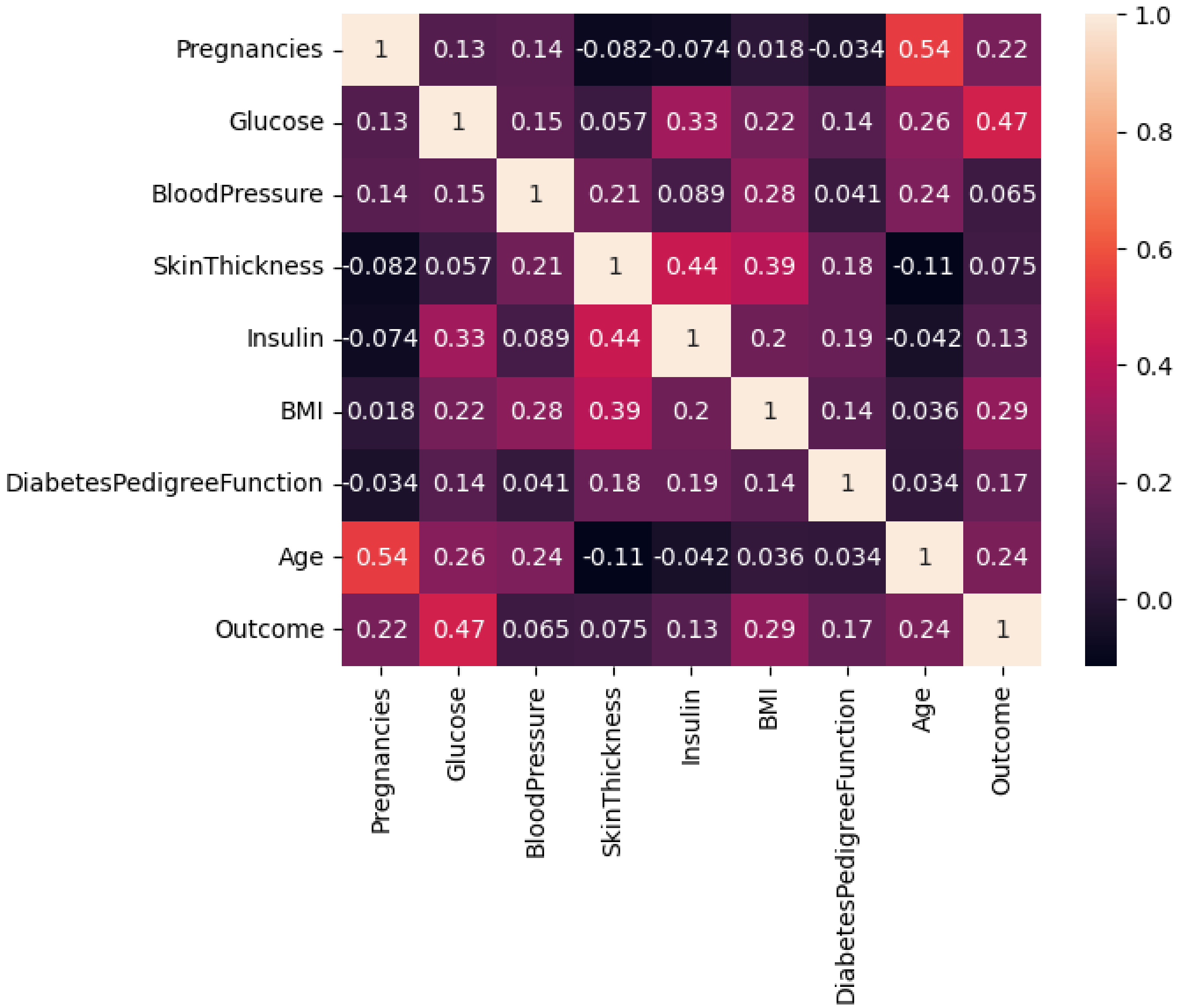
In [40]:
```
data.corr()
```

Out[40]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcor
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.2218
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.4665
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.0650
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.0747
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.1305
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.2926
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.1738
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.2383
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.0000

In [41]:
```
sns.heatmap(data.corr(),annot=True)
```

Out[41]: <AxesSubplot:>



Project Task: Week 3

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

In [42]:
```
data.head()
```

Out[42]:

	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 [43]:
```
features = data.iloc[:,[0,1,2,3,4,5,6,7]].values
```

In [44]:
```
target = data.iloc[:,-1].values
```

In [45]:
```
features
```

Out[45]:
```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

In [46]:
```
target
```

Out[46]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

In [47]:
```
from sklearn.linear_model import LogisticRegression
```

In [48]:
```
from sklearn.model_selection import train_test_split
```

In [49]:
```
X_train,X_test,y_train,y_test = train_test_split(features,target,test_size=0.20,random_state=10)
```

In [50]:
```
lr = LogisticRegression()
```

In [51]:
```
lr.fit(X_train,y_train)
```

Out[51]:
```
LogisticRegression()
```

In [52]:
```
y_pred = lr.predict(X_test)
```

In [53]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [54]:
```
accuracy_score(y_test,y_pred)
```

Out[54]:
```
0.7662337662337663
```

In [55]:
```
confusion_matrix(y_test,y_pred)
```

Out[55]:
```
array([[88,  7],
       [29, 30]], dtype=int64)
```

In [56]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.93      0.83        95
           1       0.81      0.51      0.62        59

    accuracy                           0.77       154
   macro avg       0.78      0.72      0.73       154
weighted avg       0.77      0.77      0.75       154
```

Accuracy Score from Logistic Regression : 76.6%

In [57]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [58]:
```
dtc = DecisionTreeClassifier(max_depth=5)
```

In [59]:
```
dtc.fit(X_train,y_train)
```

Out[59]:
```
DecisionTreeClassifier(max_depth=5)
```

In [60]:
```
y_pred_dtc = dtc.predict(X_test)
```

In [61]:
```
accuracy_score(y_test,y_pred_dtc)
```

Out[61]:
```
0.7597402597402597
```

In [62]:
```
confusion_matrix(y_test,y_pred_dtc)
```

Out[62]:
```
array([[84, 11],
       [26, 33]], dtype=int64)
```

In [63]:
```
print(classification_report(y_test,y_pred_dtc))
```

```
              precision    recall  f1-score   support

           0       0.76      0.88      0.82        95
           1       0.75      0.56      0.64        59

    accuracy                           0.76       154
   macro avg       0.76      0.72      0.73       154
weighted avg       0.76      0.76      0.75       154
```

Accuracy Score from Decision Tree Classifier : 76%

In [64]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [65]:
```
rfc = RandomForestClassifier(n_estimators=10)
```

In [66]:
```
rfc.fit(X_train,y_train)
```

Out[66]:
```
RandomForestClassifier(n_estimators=10)
```

In [67]:
```
y_pred_rfc = rfc.predict(X_test)
```

In [68]:
```
accuracy_score(y_test,y_pred_rfc)
```

Out[68]:
```
0.7402597402597403
```

In [69]:
```
confusion_matrix(y_test,y_pred_rfc)
```

Out[69]:
```
array([[84, 11],
       [29, 30]], dtype=int64)
```

In [70]:
```
print(classification_report(y_test,y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.74      0.88      0.81        95
           1       0.73      0.51      0.60        59

    accuracy                           0.74       154
   macro avg       0.74      0.70      0.70       154
weighted avg       0.74      0.74      0.73       154
```

Accuracy Score from Random Forest Classifier : 71.4%

In [71]:
```
from sklearn.svm import SVC
```

In [72]:
```
svc = SVC(kernel='rbf',gamma=0.1)
```

In [73]:
```
svc.fit(X_train,y_train)
```

Out[73]:
```
SVC(gamma=0.1)
```

In [74]:
```
y_pred_svc = svc.predict(X_test)
```

In [75]:
```
accuracy_score(y_test,y_pred_svc)
```

Out[75]:
```
0.6168831168831169
```

In [76]:
```
confusion_matrix(y_test,y_pred_svc)
```

Out[76]:
```
array([[95,  0],
       [59,  0]], dtype=int64)
```

In [77]:
```
print(classification_report(y_test,y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.62      1.00      0.76        95
           1       0.00      0.00      0.00        59

    accuracy                           0.62       154
   macro avg       0.31      0.50      0.38       154
weighted avg       0.38      0.62      0.47       154
```

```
C:\Users\Vinosh\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precisi
on and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Vinosh\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precisi
on and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Vinosh\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precisi
on and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Accuracy Score from Support Vector Machines(SVM) : 61.7%

In [78]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [79]:
```
knn = KNeighborsClassifier(n_neighbors=7,metric='minkowski',p=2)
```

In [80]:
```
knn.fit(X_train,y_train)
```

Out[80]:
```
KNeighborsClassifier(n_neighbors=7)
```

In [81]:
```
y_pred_knn = knn.predict(X_test)
```

```
C:\Users\Vinosh\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other re
duction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over w
hich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to Tru
e or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [82]:
```
accuracy_score(y_test,y_pred_knn)
```

Out[82]:
```
0.6948051948051948
```

In [83]:
```
confusion_matrix(y_test,y_pred_knn)
```

Out[83]:
```
array([[76, 19],
       [28, 31]], dtype=int64)
```

In [84]:
```
print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.73      0.80      0.76        95
           1       0.62      0.53      0.57        59

    accuracy                           0.69       154
   macro avg       0.68      0.66      0.67       154
weighted avg       0.69      0.69      0.69       154
```

Accuracy Score from K-Neighbours Classifier(KNN) : 69.5%

Project Task: Week 4

Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used

In [85]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

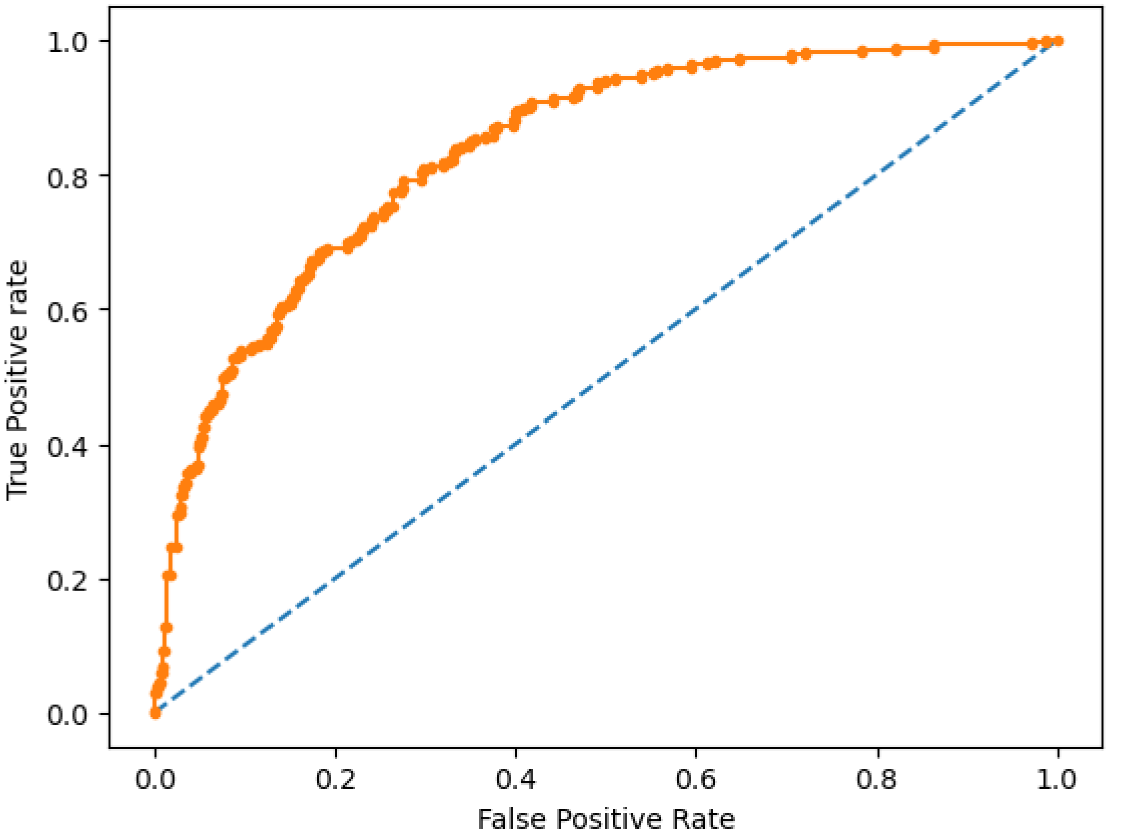
ROC curve(Receiver Operating Characteristics Curve) for Logistic Regression

In [86]:
```
probs = lr.predict_proba(features)
probs = probs[:,1]

auc = roc_auc_score(target,probs)
print('AUC :',auc)

fpr, tpr, threshold = roc_curve(target,probs)
print('True Positive Rate - {}, False Positive Rate - {}, Threshold - {}'.format(tpr,fpr,threshold))

plt.plot([0,1],[0,1],linestyle='--')
plt.plot(fpr,tpr,marker='.')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.show()
```

```
AUC : 0.8365149253731343
True Positive Rate - [0.         0.00373134 0.02985075 0.02985075 0.03731343 0.03731343
 0.04477612 0.04477612 0.05970149 0.05970149 0.06716418 0.06716418
 0.09328358 0.09328358 0.12686567 0.12686567 0.20522388 0.20522388
 0.24626866 0.24626866 0.29477612 0.29477612 0.29850746 0.29850746
 0.30597015 0.30597015 0.32462687 0.32462687 0.3358209  0.3358209
 0.34328358 0.34328358 0.35820896 0.35820896 0.3619403  0.3619403
 0.36940299 0.36940299 0.39552239 0.39552239 0.40298507 0.40298507
 0.41044776 0.41044776 0.42537313 0.42537313 0.44029851 0.44029851
 0.44402985 0.44402985 0.44776119 0.44776119 0.45895522 0.45895522
 0.46268657 0.46268657 0.4738806  0.4738806  0.49626866 0.49626866
 0.5        0.5        0.50373134 0.50373134 0.50746269 0.50746269
 0.5261194  0.5261194  0.52985075 0.52985075 0.53731343 0.53731343
 0.54104478 0.54104478 0.54477612 0.54477612 0.54850746 0.54850746
 0.55597015 0.55597015 0.56716418 0.56716418 0.57462687 0.57462687
 0.59328358 0.59328358 0.59701493 0.59701493 0.60447761 0.60447761
 0.60820896 0.60820896 0.6119403  0.6119403  0.61567164 0.61567164
 0.61940299 0.61940299 0.62686567 0.62686567 0.63059701 0.63059701
 0.64179104 0.64179104 0.64552239 0.64552239 0.64925373 0.64925373
 0.65298507 0.65298507 0.6641791  0.6641791  0.67164179 0.67164179
 0.67537313 0.67537313 0.68283582 0.68283582 0.68656716 0.68656716
 0.69029851 0.69029851 0.69776119 0.69776119 0.70149254 0.70149254
 0.70895522 0.70895522 0.71641791 0.71641791 0.7238806  0.7238806
 0.73134328 0.73134328 0.73880597 0.73880597 0.74626866 0.74626866
 0.75373134 0.75373134 0.77238806 0.77238806 0.77985075 0.77985075
 0.79104478 0.79104478 0.80223881 0.80223881 0.80970149 0.80970149
 0.81343284 0.81343284 0.81716418 0.81716418 0.82089552 0.82089552
 0.83208955 0.83208955 0.83955224 0.83955224 0.84328358 0.84328358
 0.84701493 0.84701493 0.85074627 0.85074627 0.85447761 0.85447761
 0.85820896 0.85820896 0.86940299 0.86940299 0.87313433 0.87313433
 0.88059701 0.88059701 0.89179104 0.89179104 0.89552239 0.89552239
 0.89925373 0.89925373 0.90298507 0.90298507 0.90671642 0.90671642
 0.9141791  0.9141791  0.91791045 0.91791045 0.92537313 0.92537313
 0.92910448 0.92910448 0.93656716 0.93656716 0.94029851 0.94029851
 0.94402985 0.94402985 0.94776119 0.94776119 0.95149254 0.95149254
 0.95522388 0.95522388 0.95895522 0.95895522 0.96268657 0.96268657
 0.96641791 0.96641791 0.97014925 0.97014925 0.9738806  0.9738806
 0.97761194 0.97761194 0.98134328 0.98134328 0.98507463 0.98507463
 0.98880597 0.98880597 0.99253731 0.99253731 0.99626866 0.99626866
 1.         1.        ], False Positive Rate - [0.    0.    0.    0.002 0.002 0.004 0.004 0.006 0.006 0.008 0.008 0.
01
```

```
0.01  0.012 0.012 0.014 0.014 0.018 0.018 0.024 0.024 0.026 0.026 0.028
0.028 0.03  0.03  0.032 0.032 0.034 0.034 0.036 0.036 0.04  0.04  0.046
0.046 0.048 0.048 0.05  0.05  0.052 0.052 0.054 0.054 0.056 0.056 0.058
0.058 0.06  0.06  0.064 0.064 0.072 0.072 0.074 0.074 0.076 0.076 0.078
0.078 0.08  0.08  0.084 0.084 0.086 0.086 0.09  0.09  0.096 0.096 0.106
0.106 0.108 0.108 0.116 0.116 0.124 0.124 0.128 0.128 0.134 0.134 0.136
0.136 0.138 0.138 0.14  0.14  0.146 0.146 0.15  0.15  0.152 0.152 0.154
0.154 0.156 0.156 0.158 0.158 0.16  0.16  0.164 0.164 0.166 0.166 0.168
0.168 0.17  0.17  0.174 0.174 0.18  0.18  0.182 0.182 0.186 0.186 0.19
0.19  0.214 0.214 0.218 0.218 0.224 0.224 0.228 0.228 0.23  0.23  0.24
0.24  0.242 0.242 0.254 0.254 0.258 0.258 0.264 0.264 0.274 0.274 0.276
0.276 0.294 0.294 0.298 0.298 0.306 0.306 0.32  0.32  0.326 0.326 0.33
0.33  0.332 0.332 0.34  0.34  0.348 0.348 0.35  0.35  0.356 0.356 0.366
0.366 0.374 0.374 0.38  0.38  0.398 0.398 0.4   0.4   0.402 0.402 0.408
0.408 0.414 0.414 0.416 0.416 0.442 0.442 0.464 0.464 0.468 0.468 0.47
0.47  0.49  0.49  0.498 0.498 0.51  0.51  0.54  0.54  0.552 0.552 0.556
0.556 0.568 0.568 0.594 0.594 0.612 0.612 0.622 0.622 0.648 0.648 0.706
0.706 0.72  0.72  0.782 0.782 0.82  0.82  0.862 0.862 0.972 0.972 0.986
0.986 1.   ], Threshold - [1.98701958 0.98701958 0.94267813 0.93864767 0.93299793 0.92965234
0.92449809 0.92295419 0.9109033  0.9089279  0.89986778 0.89430642
0.8751992  0.87398928 0.85595595 0.85073959 0.80122753 0.80058422
0.76977526 0.76759245 0.7439569  0.74038711 0.73746476 0.73738287
0.73576096 0.73482429 0.72598562 0.72549377 0.7229558  0.72267404
0.71273193 0.71256592 0.70019619 0.69929725 0.69694969 0.69041877
0.68611012 0.68590784 0.6762251  0.67580406 0.67034738 0.66457106
0.6606839  0.66066835 0.64618493 0.64376411 0.63513039 0.63454764
0.62955203 0.62873801 0.62389708 0.62185462 0.6109426  0.59956486
0.59909079 0.59824388 0.59301877 0.59300751 0.58164358 0.57809436
0.57775975 0.57536941 0.57418329 0.56811984 0.56563107 0.56548073
0.55489933 0.55184235 0.54886894 0.53716694 0.52127787 0.51026704
0.50875682 0.50060969 0.5003371  0.49221375 0.49115548 0.48690113
0.48374831 0.48059014 0.47838555 0.47267881 0.46910984 0.46868878
0.45606003 0.45538138 0.45216607 0.45199098 0.45144526 0.44536687
0.44477876 0.44111613 0.43921584 0.43913741 0.43809249 0.43311663
0.42895914 0.4254682  0.42213722 0.42185714 0.41998808 0.41860876
0.41607864 0.41484523 0.41378442 0.41103317 0.40738591 0.40592932
0.40592529 0.40278561 0.39789206 0.39448681 0.39368984 0.38902675
0.38857132 0.38755247 0.38301324 0.3796726  0.37942035 0.37639384
0.37324015 0.3608211  0.36076678 0.35557766 0.35481391 0.35424796
0.35051158 0.34707509 0.34387902 0.34349274 0.33632515 0.32565849
0.3241703  0.32362376 0.32287564 0.3193176  0.31809647 0.31651008
0.31472705 0.31079557 0.30211361 0.29743473 0.29608229 0.2954971
```

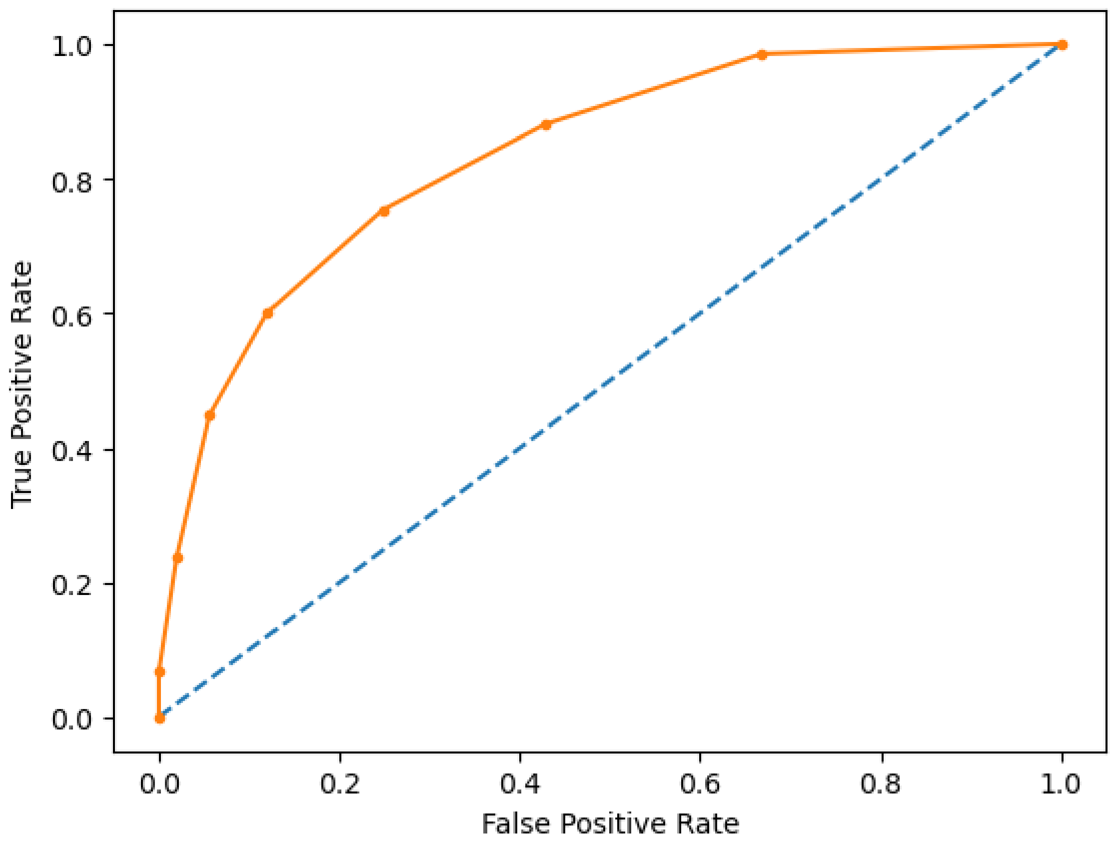
```
0.28975046 0.28250323 0.28119305 0.27998668 0.27822654 0.27677575
0.27657858 0.26868302 0.26805709 0.26570691 0.26533752 0.26308727
0.25965733 0.25933212 0.25550478 0.24956107 0.24953643 0.24664982
0.24448216 0.24308441 0.24270423 0.23698716 0.23594263 0.23432217
0.23362581 0.22943678 0.2251244  0.22382226 0.22301707 0.21872426
0.21758256 0.21632704 0.21549192 0.21515184 0.21511625 0.21296899
0.21239997 0.20870119 0.20755782 0.20653257 0.20450608 0.18984373
0.18597385 0.17906852 0.17872236 0.17782626 0.17689093 0.17679299
0.17553486 0.16960362 0.1686741  0.16388822 0.16339877 0.15857575
0.15580608 0.14780034 0.14774528 0.14575832 0.14570642 0.1449563
0.14469839 0.13924984 0.1384298  0.12990157 0.12868798 0.12120243
0.12113263 0.11690265 0.11626777 0.11000648 0.10971483 0.09905693
0.09882209 0.09580951 0.09574561 0.08191707 0.0809058  0.06912687
0.06908708 0.0598058  0.05764699 0.02224674 0.02194656 0.01486693
0.01315679 0.00210894]
```



ROC Curve (Receiver Operating Characteristics Curve) for K-Neighbors Classifiers(KNN)

In [87]:
```
probs = knn.predict_proba(features)
probs = probs[:,1]

auc = roc_auc_score(target,probs)
print('AUC :',auc)

fpr,tpr,threshold = roc_curve(target,probs)
print('True Positive Rate - {}, False Positive Rate - {}, Threshold - {}'.format(tpr,fpr,threshold))

plt.plot([0,1],[0,1],linestyle='--')
plt.plot(fpr,tpr,marker='.')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

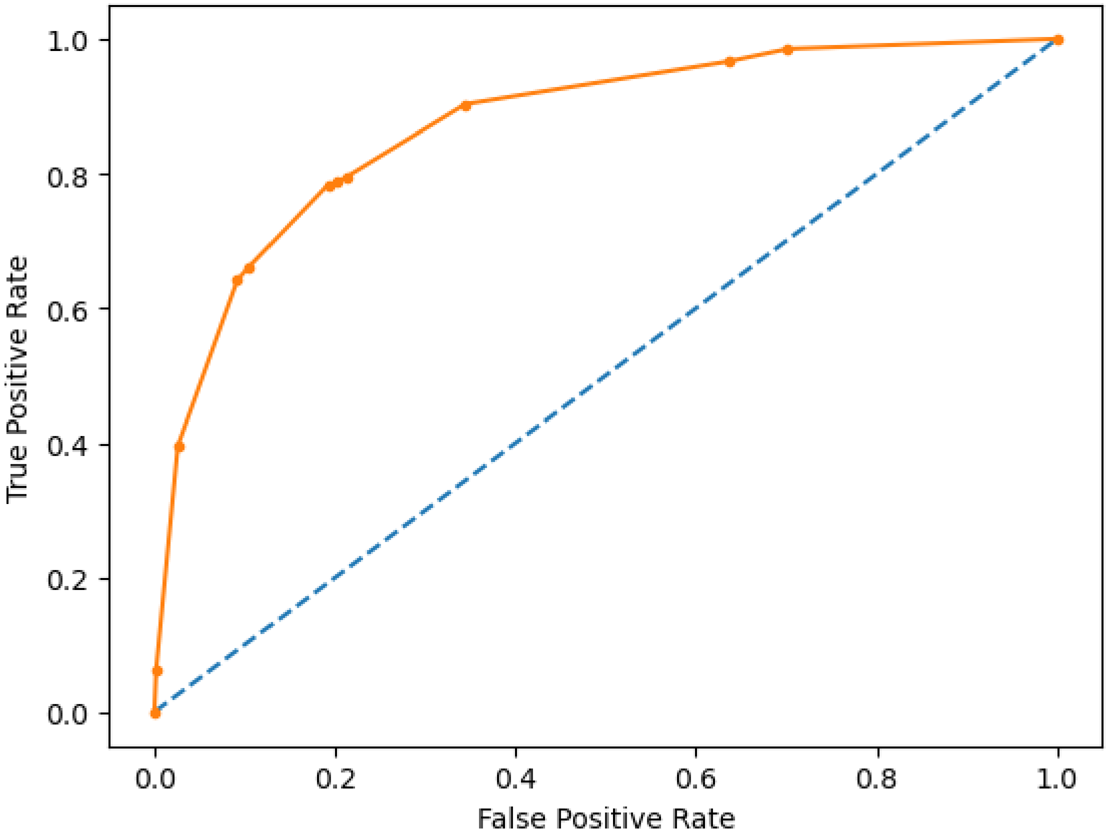
```
AUC : 0.8361492537313433
True Positive Rate - [0.         0.06716418 0.23880597 0.44776119 0.60074627 0.75373134
 0.88059701 0.98507463 1.        ], False Positive Rate - [0.    0.    0.02  0.056 0.12  0.248 0.428 0.668 1.   ], T
hreshold - [2.         1.         0.85714286 0.71428571 0.57142857 0.42857143
 0.28571429 0.14285714 0.        ]
```



ROC Curve (Receiver Operating Characteristics Curve) for Decision Tree Classifier

In [88]:
```
probs = dtc.predict_proba(features)
probs = probs[:,1]

auc = roc_auc_score(target,probs)
print('AUC :',auc)

fpr,tpr,threshold = roc_curve(target,probs)
print('True Positive Rate - {}, False Positive Rate - {}, Threshold - {}'.format(tpr,fpr,threshold))

plt.plot([0,1],[0,1],linestyle='--')
plt.plot(fpr,tpr,marker='.')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

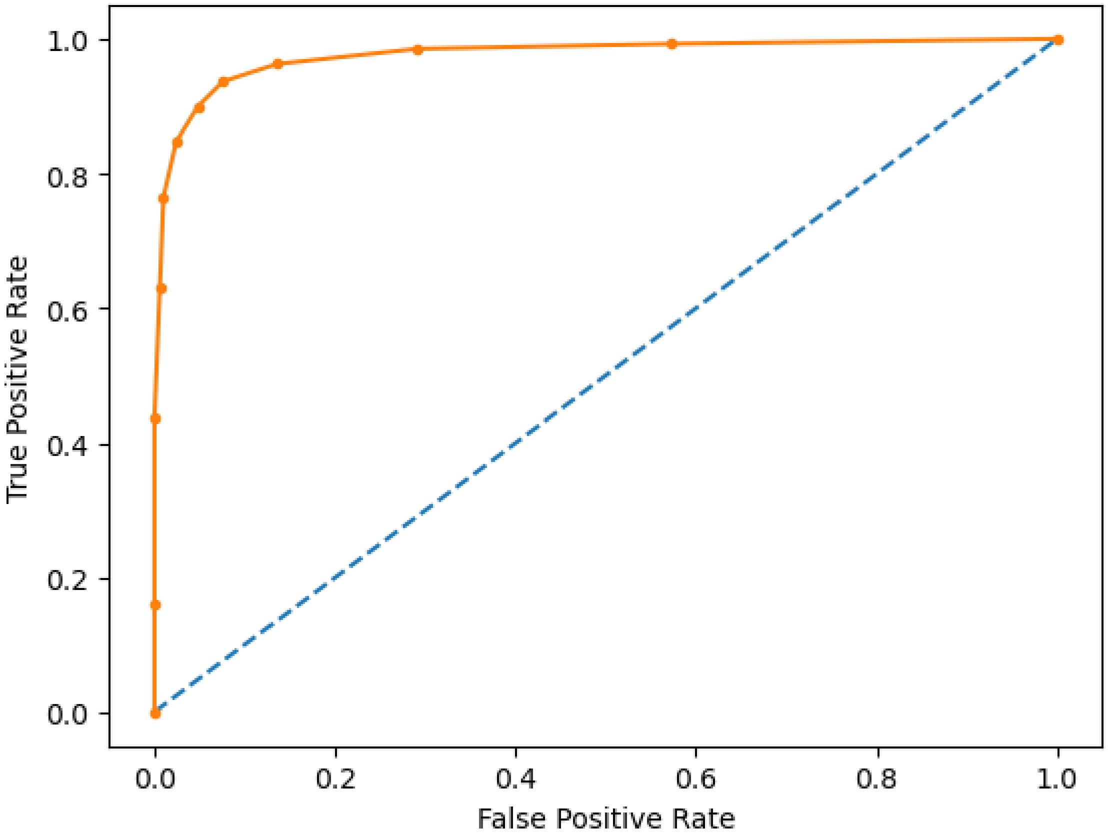
```
AUC : 0.8720783582089552
True Positive Rate - [0.         0.06343284 0.39552239 0.64179104 0.66044776 0.78358209
 0.78731343 0.79477612 0.90298507 0.96641791 0.98507463 1.        ], False Positive Rate - [0.    0.002 0.026 0.092
0.104 0.192 0.202 0.212 0.344 0.636 0.702 1.   ], Threshold - [2.         1.         0.89285714 0.65789474 0.4444444
4 0.40983607
 0.33333333 0.28571429 0.28205128 0.10606061 0.07692308 0.        ]
```



ROC Curve (Receiver Operating Characteristics Curve) for Random Forest Classifier

In [89]:
```
probs = rfc.predict_proba(features)
probs = probs[:,1]

auc = roc_auc_score(target,probs)
print('AUC :',auc)

fpr,tpr,threshold = roc_curve(target,probs)
print('True Positive Rate - {}, False Positive Rate - {}, Threshold - {}'.format(tpr,fpr,threshold))

plt.plot([0,1],[0,1],linestyle='--')
plt.plot(fpr,tpr,marker='.')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
AUC : 0.9761343283582089
True Positive Rate - [0.         0.16044776 0.43656716 0.63059701 0.76492537 0.84701493
 0.89925373 0.93656716 0.96268657 0.98507463 0.99253731 1.        ], False Positive Rate - [0.    0.    0.    0.006
0.01  0.024 0.048 0.076 0.136 0.29  0.572 1.   ], Threshold - [2.  1.  0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0. ]
```

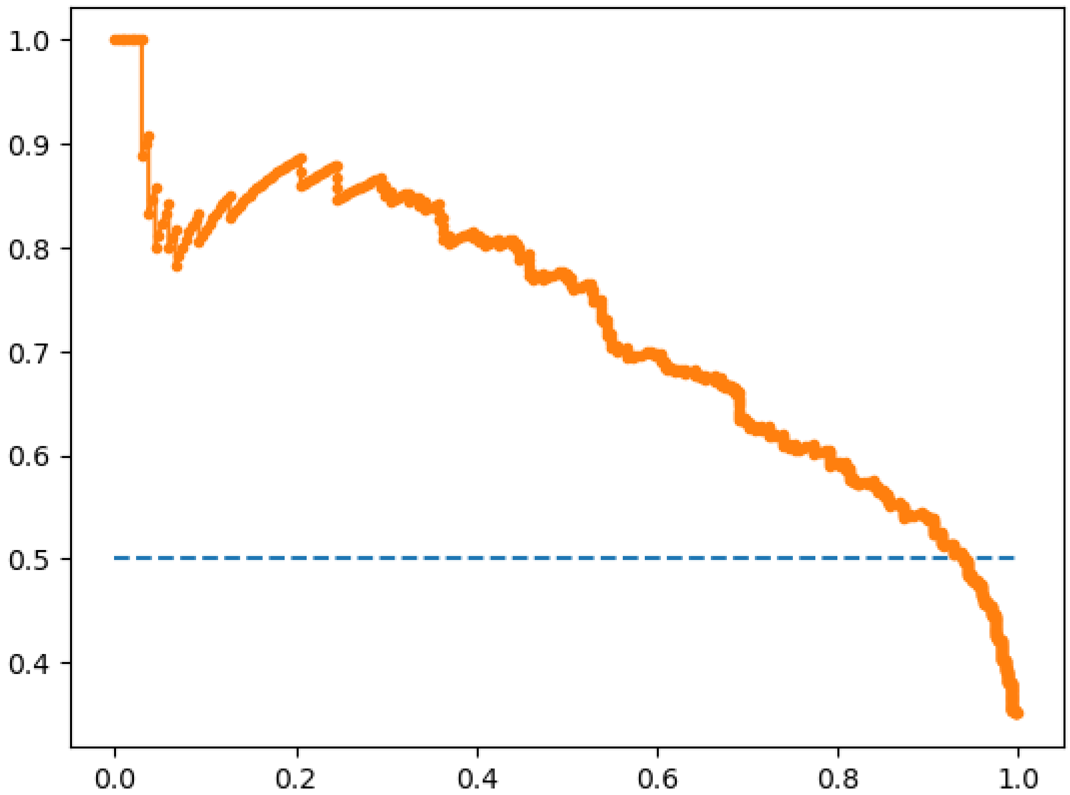


In [90]:
```
from sklearn.metrics import f1_score, precision_recall_curve, auc, average_precision_score
```

Precision Recall Curve for Logistic Regression

In [91]:
```
probs = lr.predict_proba(features)
probs = probs[:,1]

precision, recall, threshold = precision_recall_curve(target, probs)

pred_lr = lr.predict(features)

f1 = f1_score(target,pred_lr)

auc = auc(recall,precision)

ap = average_precision_score(target, probs)

print('F1 score - {}, AUC - {}, AP - {}'.format(f1,auc,ap))

plt.plot([0,1], [0.5,0.5], linestyle='--')
plt.plot(recall, precision, marker='.')
plt.show()
```

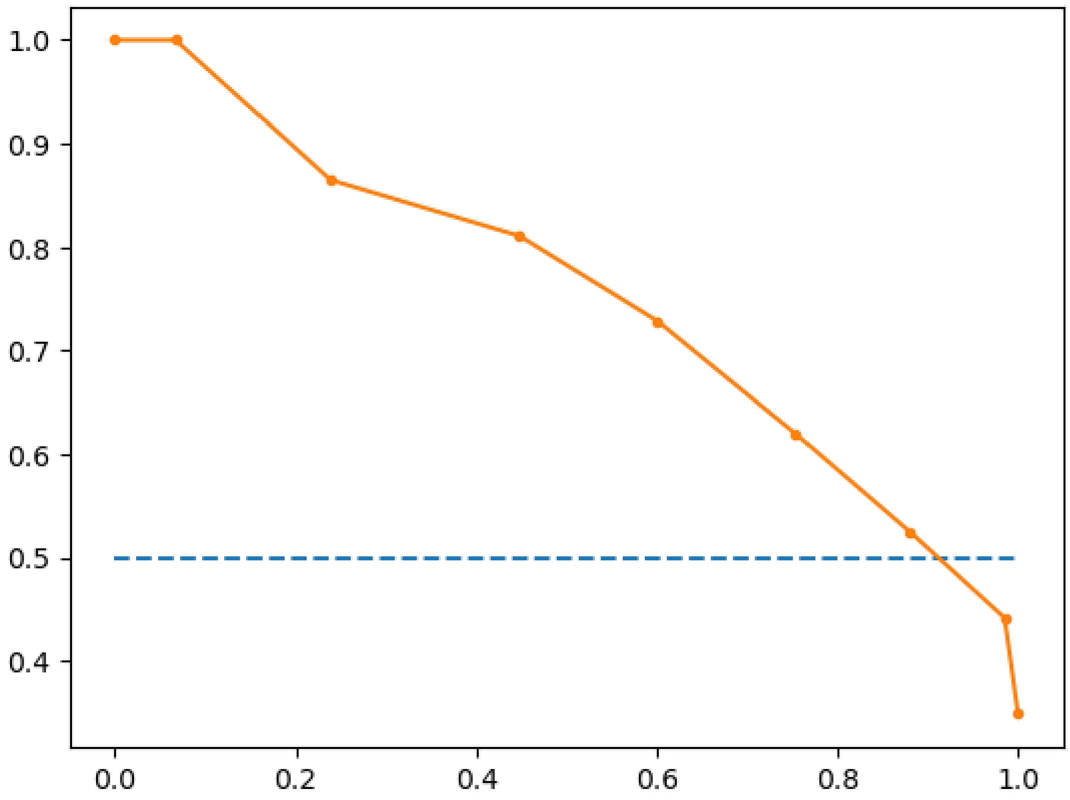
```
F1 score - 0.6239316239316239, AUC - 0.7259547659317013, AP - 0.7269516907977814
```



Precision Curve for K-Neighbors Classifier(KNN)

In [92]:
```
from sklearn.metrics import precision_recall_curve, f1_score, auc, average_precision_score

probs = knn.predict_proba(features)
probs = probs[:,1]

precision, recall, threshold = precision_recall_curve(target,probs)

pred_knn = knn.predict(features)

f1_knn = f1_score(target,pred_knn)

auc_knn = auc(recall, precision)

ap_knn = average_precision_score(target,probs)

print('F1 score - {}, AUC - {}, AP - {}'.format(f1_knn,auc_knn,ap_knn))

plt.plot([0,1],[0.5,0.5],linestyle='--')
plt.plot(recall, precision, marker='.')
plt.show()
```

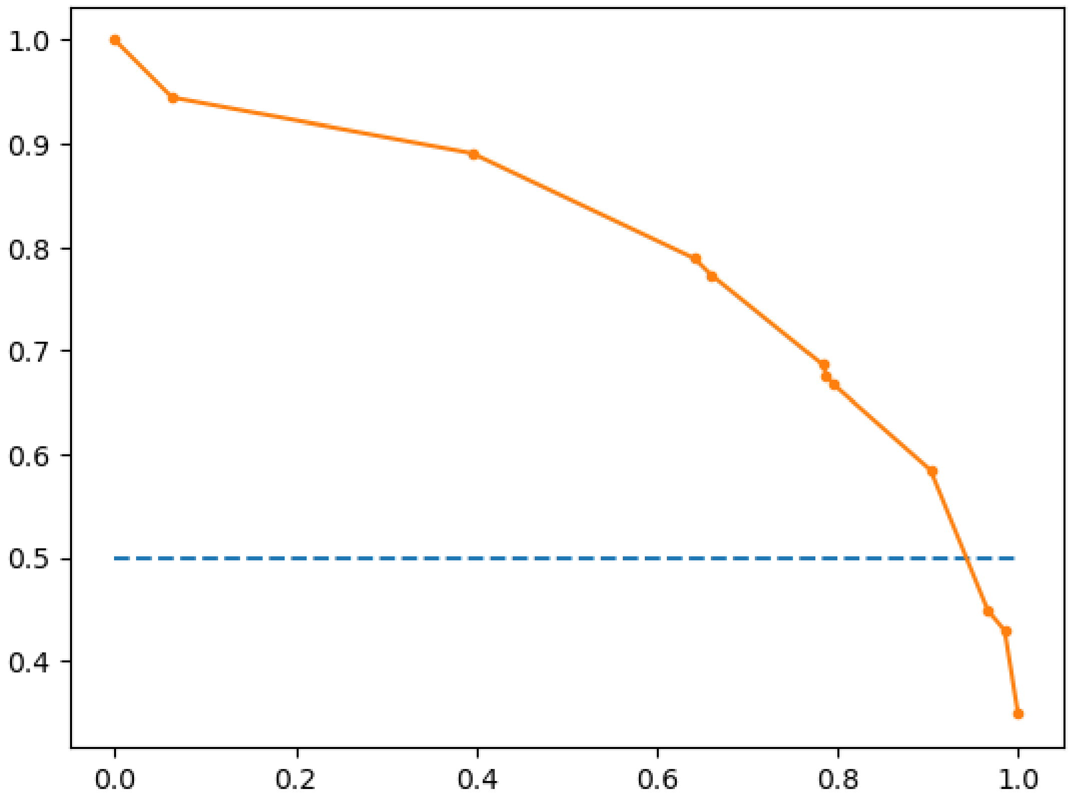
```
F1 score - 0.6584867075664621, AUC - 0.752077039652761, AP - 0.7091456115784129

C:\Users\Vinosh\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other re
duction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over w
hich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to Tru
e or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```



Precision Curve for Decision Tree Classifier

In [93]:
```
from sklearn.metrics import f1_score, precision_recall_curve, auc, average_precision_score

probs = dtc.predict_proba(features)
probs = probs[:,1]

precision, recall, threshold = precision_recall_curve(target,probs)

pred_dtc = dtc.predict(features)

f1_dtc = f1_score(target,pred_dtc)

auc_dtc = auc(recall,precision)

ap_dtc = average_precision_score(target,probs)

print('F1 Score - {}, AUC - {}, AP - {}'.format(f1_dtc,auc_dtc,ap_dtc))

plt.plot([0,1],[0.5,0.5],linestyle = '--')
plt.plot(recall, precision,marker='.')
plt.show()
```

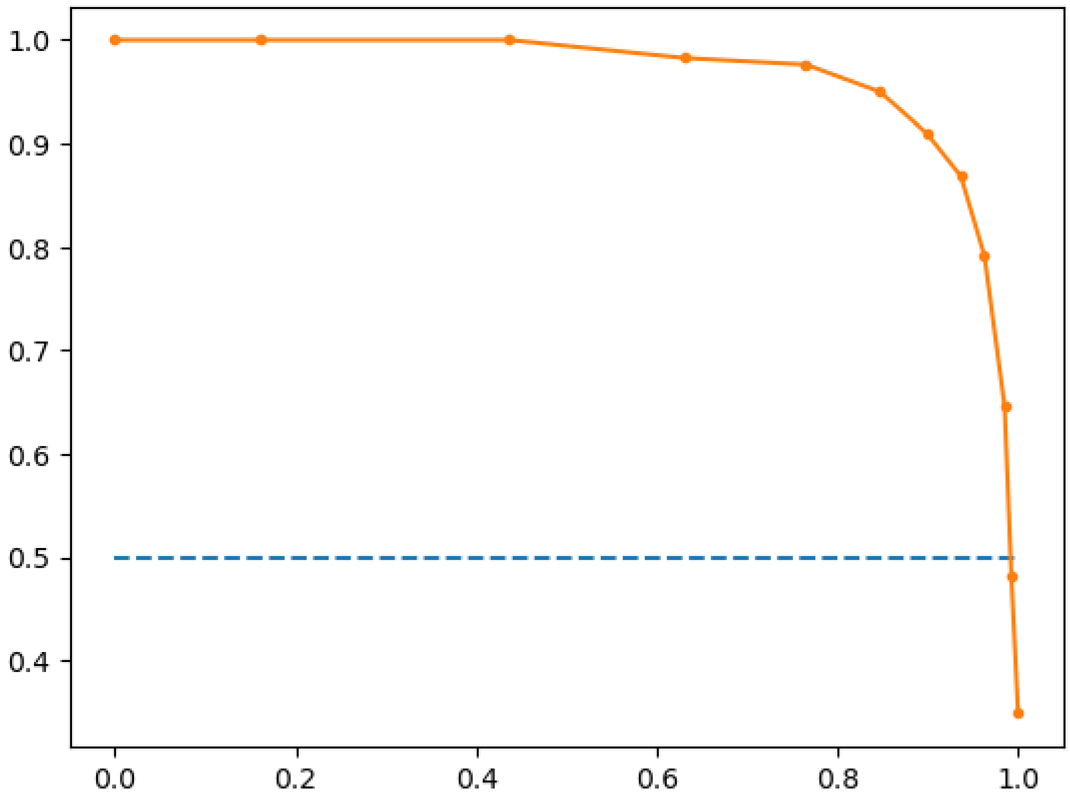
```
F1 Score - 0.7078189300411523, AUC - 0.7997247354617167, AP - 0.7613968981593674
```



Precision Curve for Random Forest Classifier

In [94]:
```
from sklearn.metrics import f1_score, precision_recall_curve, auc, average_precision_score

probs = rfc.predict_proba(features)
probs = probs[:,1]

precision, recall, threshold = precision_recall_curve(target,probs)

pred_rfc = rfc.predict(features)

f1_rfc = f1_score(target,pred_rfc)

auc_rfc = auc(recall, precision)

ap_rfc = average_precision_score(target, probs)

print('F1 Score - {}, AUC - {}, AP - {}'.format(f1_rfc,auc_rfc,ap_rfc))

plt.plot([0,1],[0.5,0.5],linestyle='--')
plt.plot(recall,precision,marker='.')
plt.show()
```

```
F1 Score - 0.8954635108481261, AUC - 0.9663159778578709, AP - 0.9575481238287105
```



In [ ]: