In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
%matplotlib inline
```

In [43]:

```
data = pd.read_csv('health care diabetes.csv')
data.head()
```

Out[43]:

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

In [44]:

```
data.isnull().any()
```

Out[44]:

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [45]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [ ]:

```

```

In [46]:

```
data.describe().transpose()
```

Out[46]:

	count	mean	std	min	25%	50%	7
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00

In [47]:

```
data['Glucose'].value_counts().head(5)
```

Out[47]:

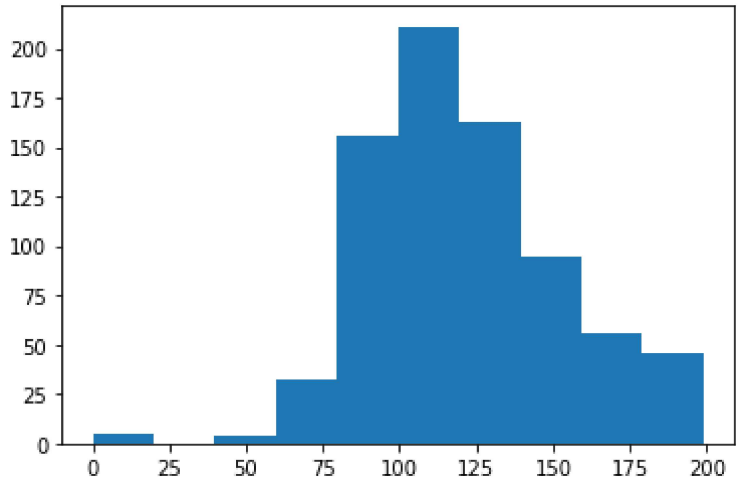
```
100    17
99     17
129    14
125    14
111    14
Name: Glucose, dtype: int64
```

In [48]:

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

Out[48]:

```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <a list of 10 Patch objects>)
```



In [49]:

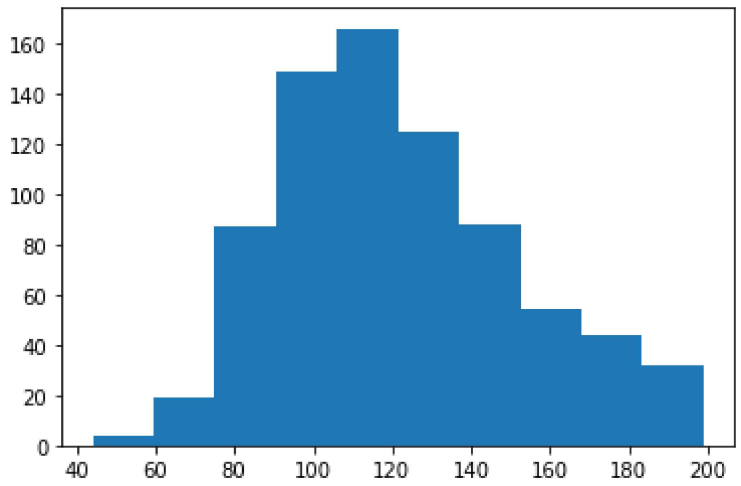
```
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
```

In [50]:

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

Out[50]:

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



In [51]:

```
data['BloodPressure'].value_counts().head(5)
```

Out[51]:

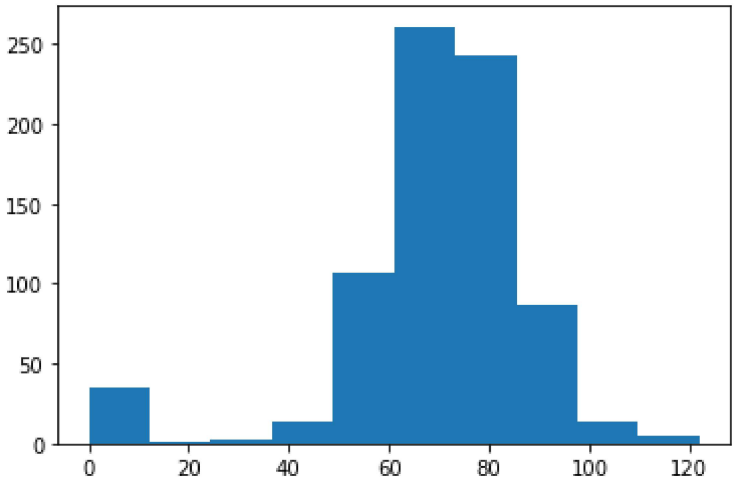
```
70    57
74    52
68    45
78    45
72    44
Name: BloodPressure, dtype: int64
```

In [52]:

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

Out[52]:

```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
 <a list of 10 Patch objects>)
```



In [53]:

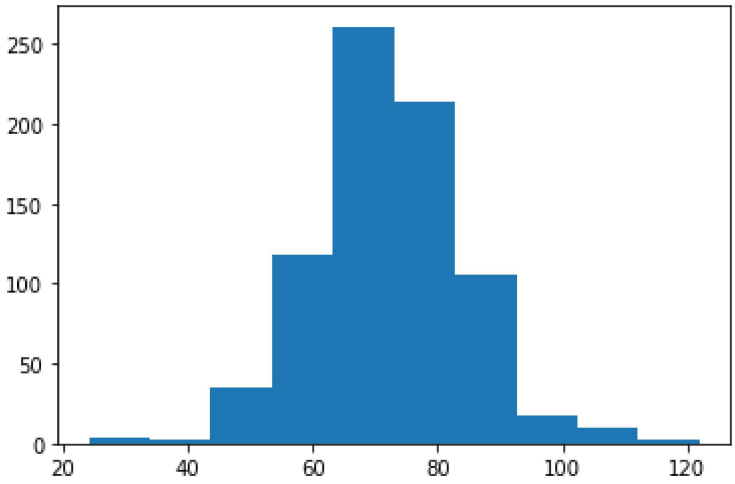
```
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
```

In [54]:

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

Out[54]:

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



In [55]:

```
data['SkinThickness'].value_counts().head(5)
```

Out[55]:

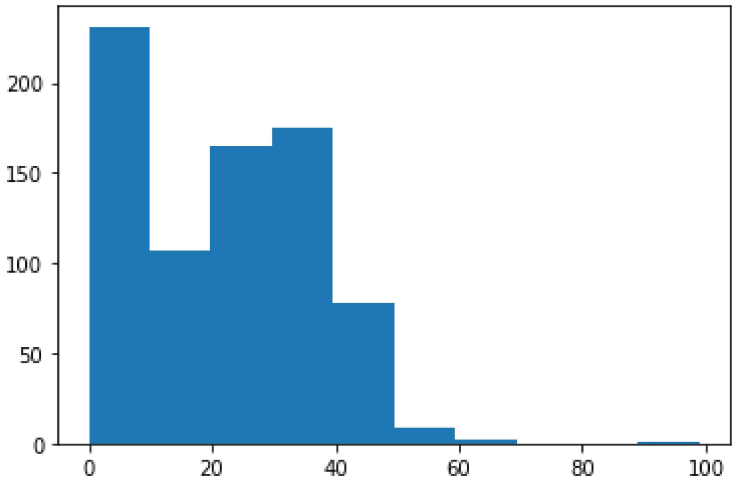
```
0     227
32     31
30     27
27     23
23     22
Name: SkinThickness, dtype: int64
```

In [56]:

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

Out[56]:

```
(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.
]),
 <a list of 10 Patch objects>)
```



In [57]:

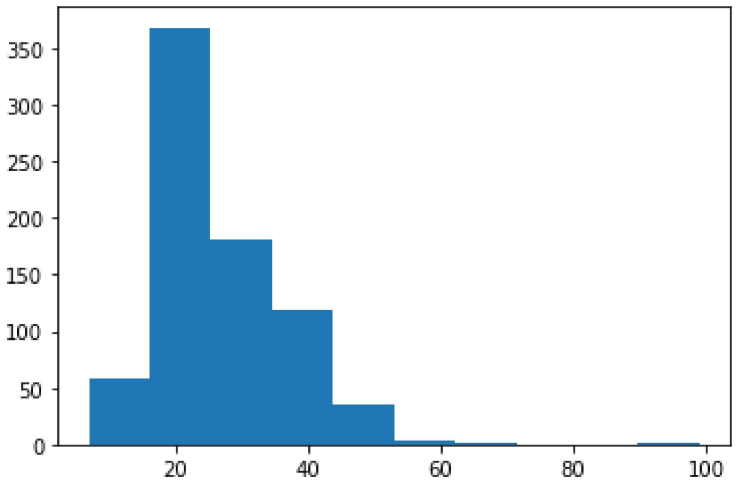
```
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [58]:

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

Out[58]:

```
(array([ 59., 368., 181., 118.,  36.,   4.,   1.,   0.,   0.,   1.]),
 array([ 7. , 16.2, 25.4, 34.6, 43.8, 53. , 62.2, 71.4, 80.6, 89.8, 99.
]),
 <a list of 10 Patch objects>)
```



In [59]:

```
data['Insulin'].value_counts().head(5)
```

Out[59]:

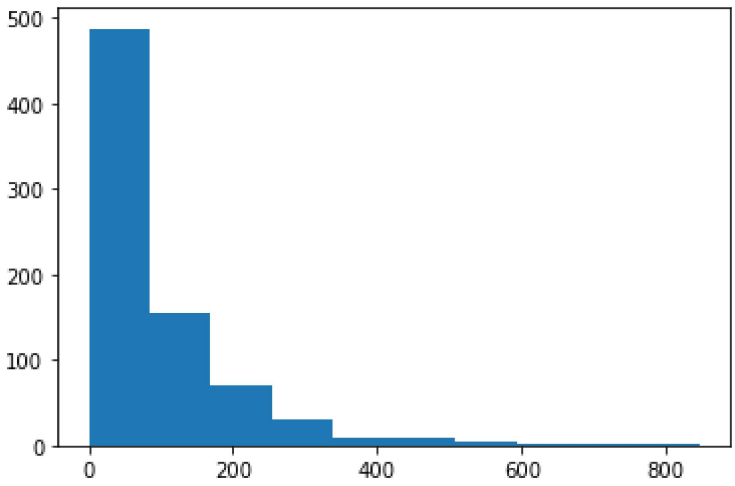
```
0      374
105     11
140      9
130      9
120      8
Name: Insulin, dtype: int64
```

In [60]:

```
plt.hist(data['Insulin'])
```

Out[60]:

```
(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. ]),
 <a list of 10 Patch objects>)
```



In [61]:

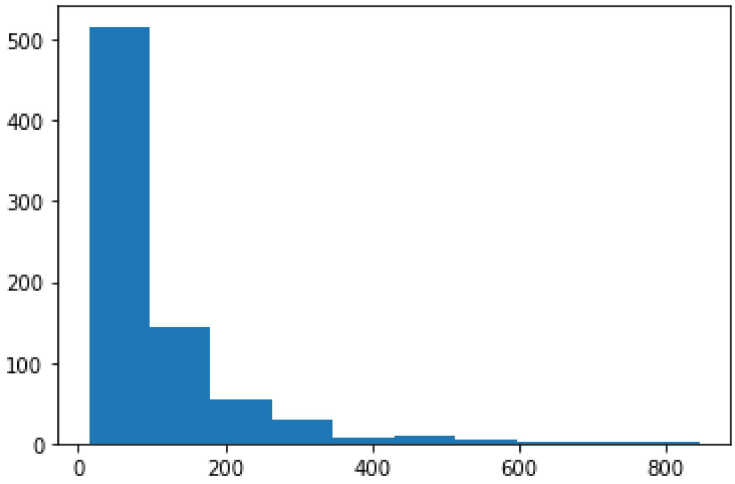
```
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
```

In [62]:

```
plt.hist(data['Insulin'])
```

Out[62]:

```
(array([516., 143.,  55.,  29.,   7.,  10.,   4.,   1.,   2.,   1.]),
 array([ 14. ,  97.2, 180.4, 263.6, 346.8, 430. , 513.2, 596.4, 679.6,
        762.8, 846. ]),
 <a list of 10 Patch objects>)
```



In [63]:

```
data['BMI'].value_counts().head(5)
```

Out[63]:

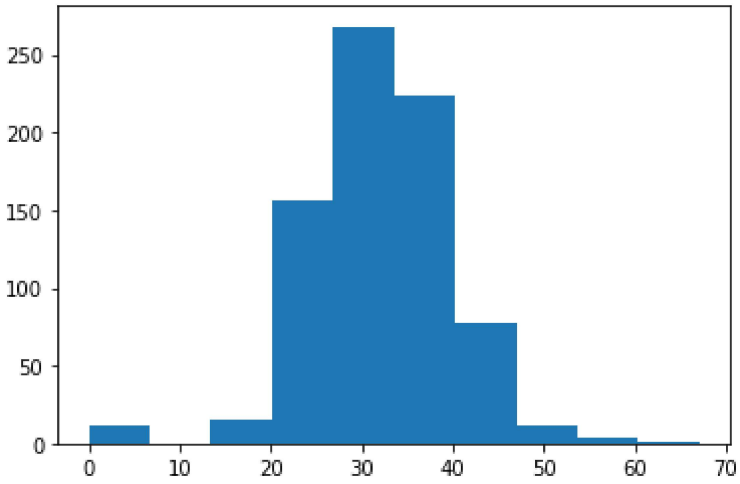
```
32.0    13
31.6    12
31.2    12
0.0     11
33.3    10
Name: BMI, dtype: int64
```

In [64]:

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

Out[64]:

```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <a list of 10 Patch objects>)
```



In [65]:

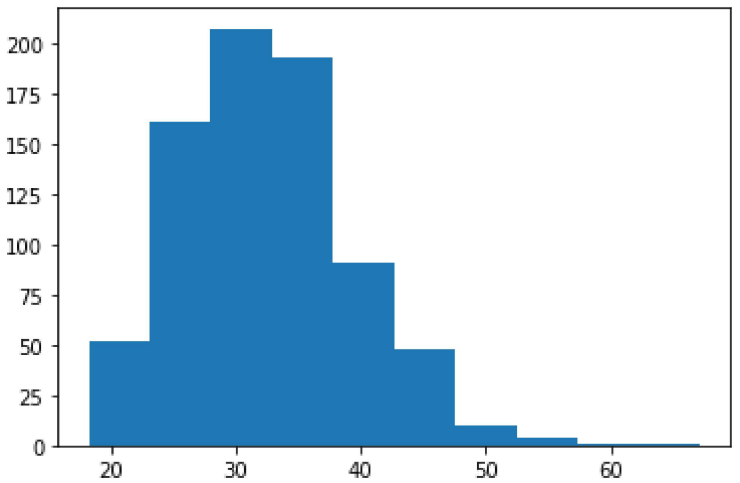
```
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
```

In [66]:

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

Out[66]:

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



In [ ]:

```
```

In [67]:

```
Positive = data[data['Outcome']==1]
Positive.head(5)
```

Out[67]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148.0	72.0	35.000000	79.799479	33.6	
2	8	183.0	64.0	20.536458	79.799479	23.3	
4	0	137.0	40.0	35.000000	168.000000	43.1	
6	3	78.0	50.0	32.000000	88.000000	31.0	
8	2	197.0	70.0	45.000000	543.000000	30.5	

In [68]:

```
Positive['BMI'].value_counts().head(5)
```

Out[68]:

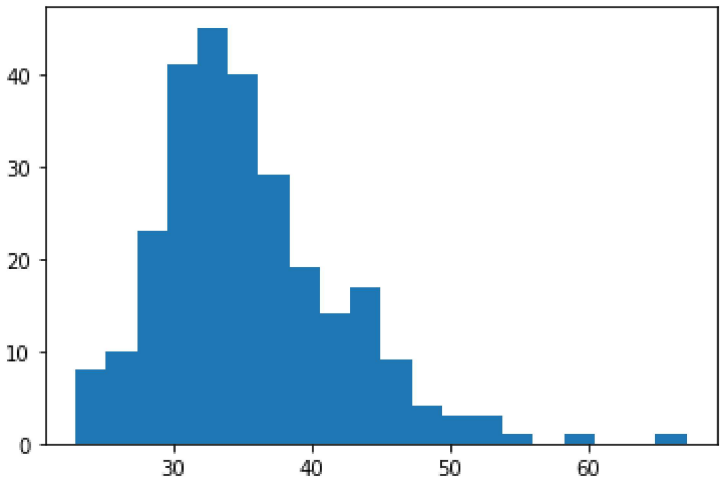
```
32.9    8
31.6    7
33.3    6
32.0    5
30.5    5
Name: BMI, dtype: int64
```

In [69]:

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

Out[69]:

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



In [70]:

```
Positive['Glucose'].value_counts().head(5)
```

Out[70]:

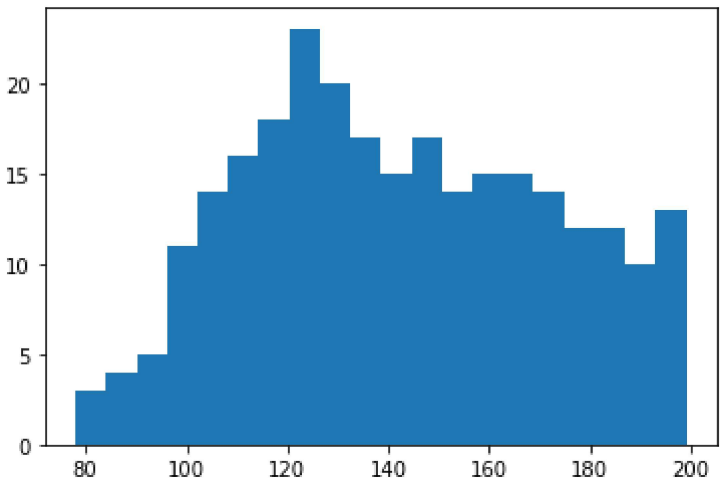
```
125.0    7
128.0    6
129.0    6
158.0    6
115.0    6
Name: Glucose, dtype: int64
```

In [71]:

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

Out[71]:

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



In [72]:

```
Positive['BloodPressure'].value_counts().head(5)
```

Out[72]:

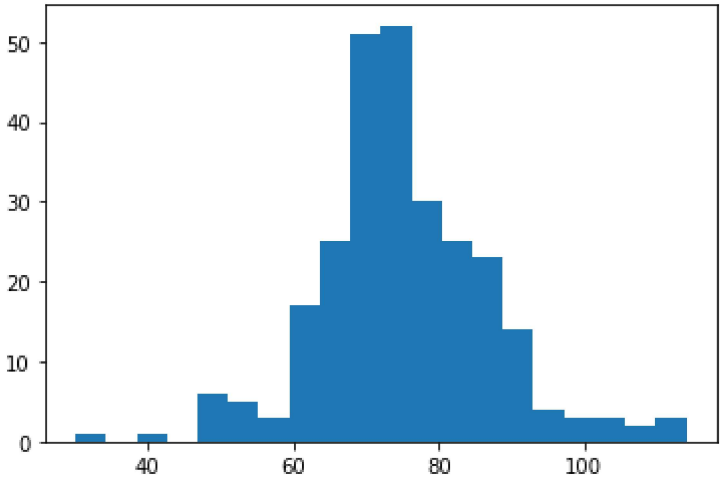
```
70.000000    23
76.000000    18
78.000000    17
74.000000    17
69.105469    16
Name: BloodPressure, dtype: int64
```

In [73]:

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

Out[73]:

```
(array([ 1.,  0.,  1.,  0.,  6.,  5.,  3., 17., 25., 51., 52., 30., 25.,
        23., 14.,  4.,  3.,  3.,  2.,  3.]),
 array([ 30. ,  34.2,  38.4,  42.6,  46.8,  51. ,  55.2,  59.4,  63.6,
         67.8,  72. ,  76.2,  80.4,  84.6,  88.8,  93. ,  97.2, 101.4,
        105.6, 109.8, 114. ]),
 <a list of 1 Patch objects>)
```



In [74]:

```
Positive['SkinThickness'].value_counts().head(5)
```

Out[74]:

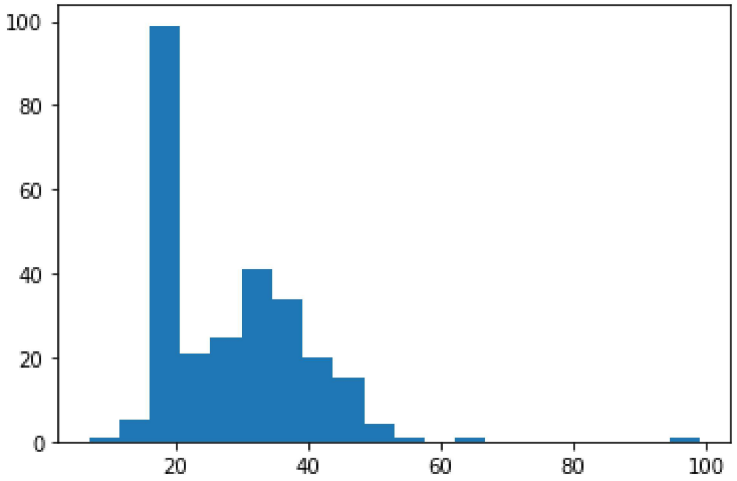
```
20.536458    88
32.000000    14
30.000000     9
33.000000     9
39.000000     8
Name: SkinThickness, dtype: int64
```

In [75]:

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

Out[75]:

```
(array([ 1.,  5., 99., 21., 25., 41., 34., 20., 15.,  4.,  1.,  0.,  1.,
         0.,  0.,  0.,  0.,  0.,  0.,  1.]),
 array([ 7. , 11.6, 16.2, 20.8, 25.4, 30. , 34.6, 39.2, 43.8, 48.4, 53. ,
        57.6, 62.2, 66.8, 71.4, 76. , 80.6, 85.2, 89.8, 94.4, 99. ]),
 <a list of 1 Patch objects>)
```



In [76]:

```
Positive['Insulin'].value_counts().head(5)
```

Out[76]:

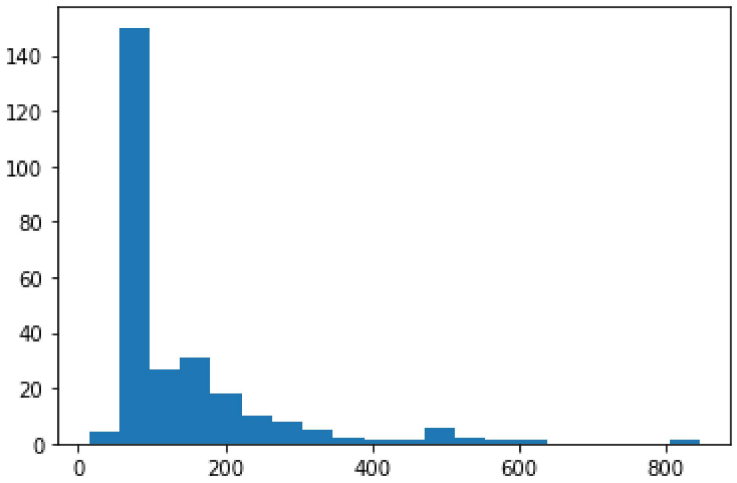
```
79.799479     138
130.000000      6
180.000000      4
156.000000      3
175.000000      3
Name: Insulin, dtype: int64
```

In [77]:

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

Out[77]:

```
(array([  4., 150.,  27.,  31.,  18.,  10.,   8.,   5.,   2.,   1.,   1.,
          6.,   2.,   1.,   1.,   0.,   0.,   0.,   0.,   1.]),
 array([ 14. ,  55.6,  97.2, 138.8, 180.4, 222. , 263.6, 305.2, 346.8,
        388.4, 430. , 471.6, 513.2, 554.8, 596.4, 638. , 679.6, 721.2,
        762.8, 804.4, 846. ]),
 <a list of 1 Patch objects>)
```



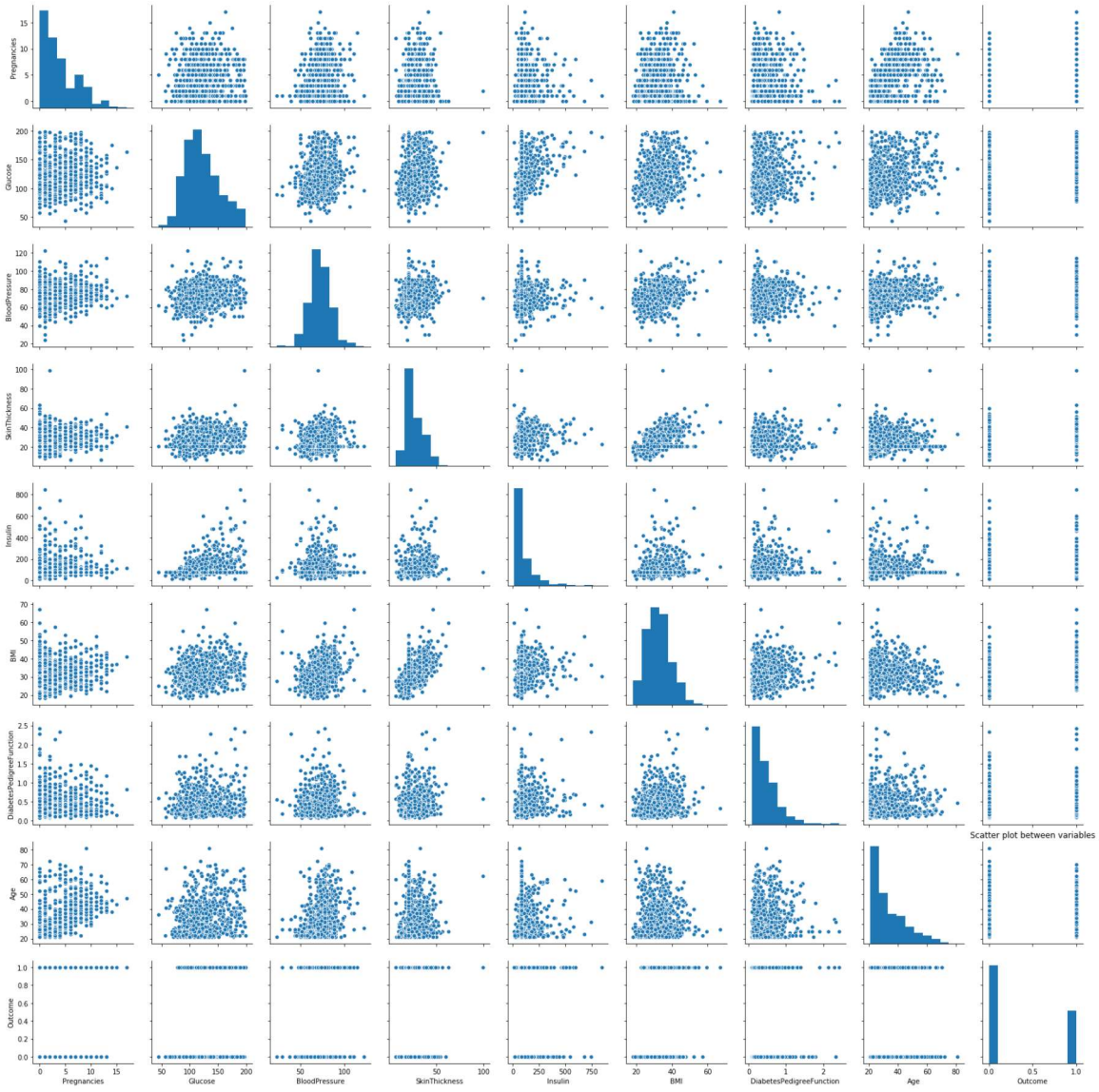
In [ ]:

In [78]:

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

Out[78]:

```
Text(0.5, 1, 'Scatter plot between variables')
```



We can see from scatter plot that there is no strong multicolinearity among features, but between skin thickness and BMI, Pregnancies and age it looks like there is small chance of positive correlation.

In [ ]:

```
```

In [79]:

```
### correlation matrix
data.corr()
```

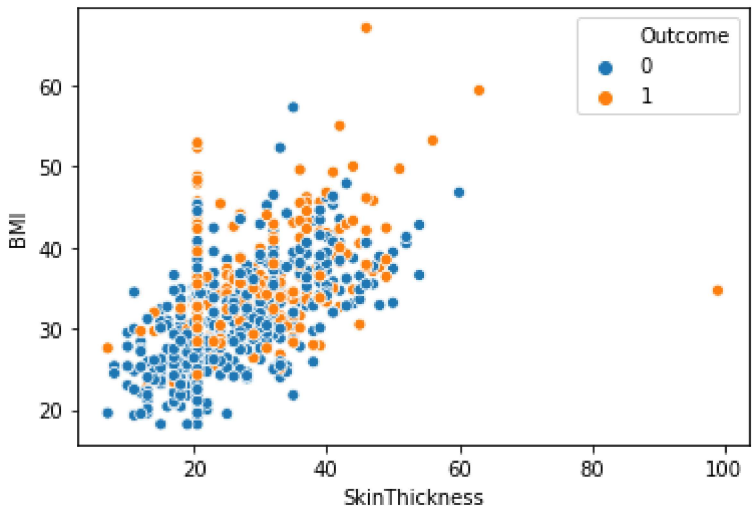
Out[79]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.127964	0.208984	0.013376	-0.018082	
Glucose	0.127964	1.000000	0.219666	0.160766	0.396597	
BloodPressure	0.208984	0.219666	1.000000	0.134155	0.010926	
SkinThickness	0.013376	0.160766	0.134155	1.000000	0.240361	
Insulin	-0.018082	0.396597	0.010926	0.240361	1.000000	
BMI	0.021546	0.231478	0.281231	0.535703	0.189856	
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.154961	0.157806	
Age	0.544341	0.266600	0.326740	0.026423	0.038652	
Outcome	0.221898	0.492908	0.162986	0.175026	0.179185	

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

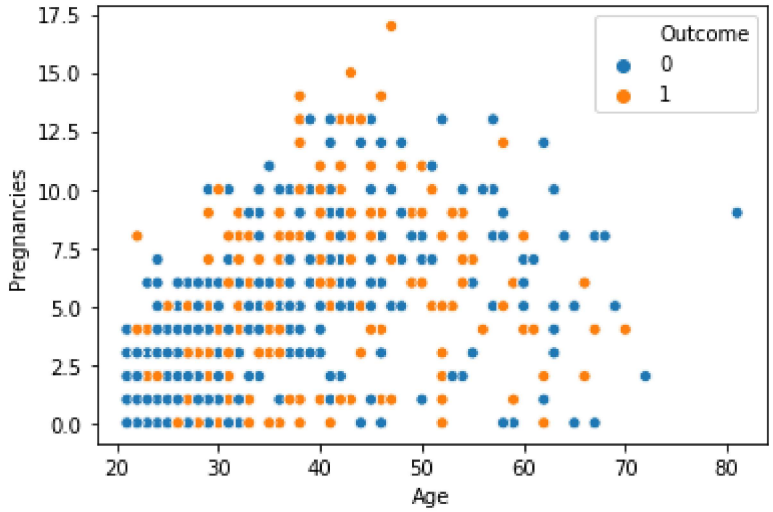
In [80]:

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



In [81]:

```
g =sns.scatterplot(x= "Age" ,y= "Pregnancies",
                   hue="Outcome",
                   data=data);
```

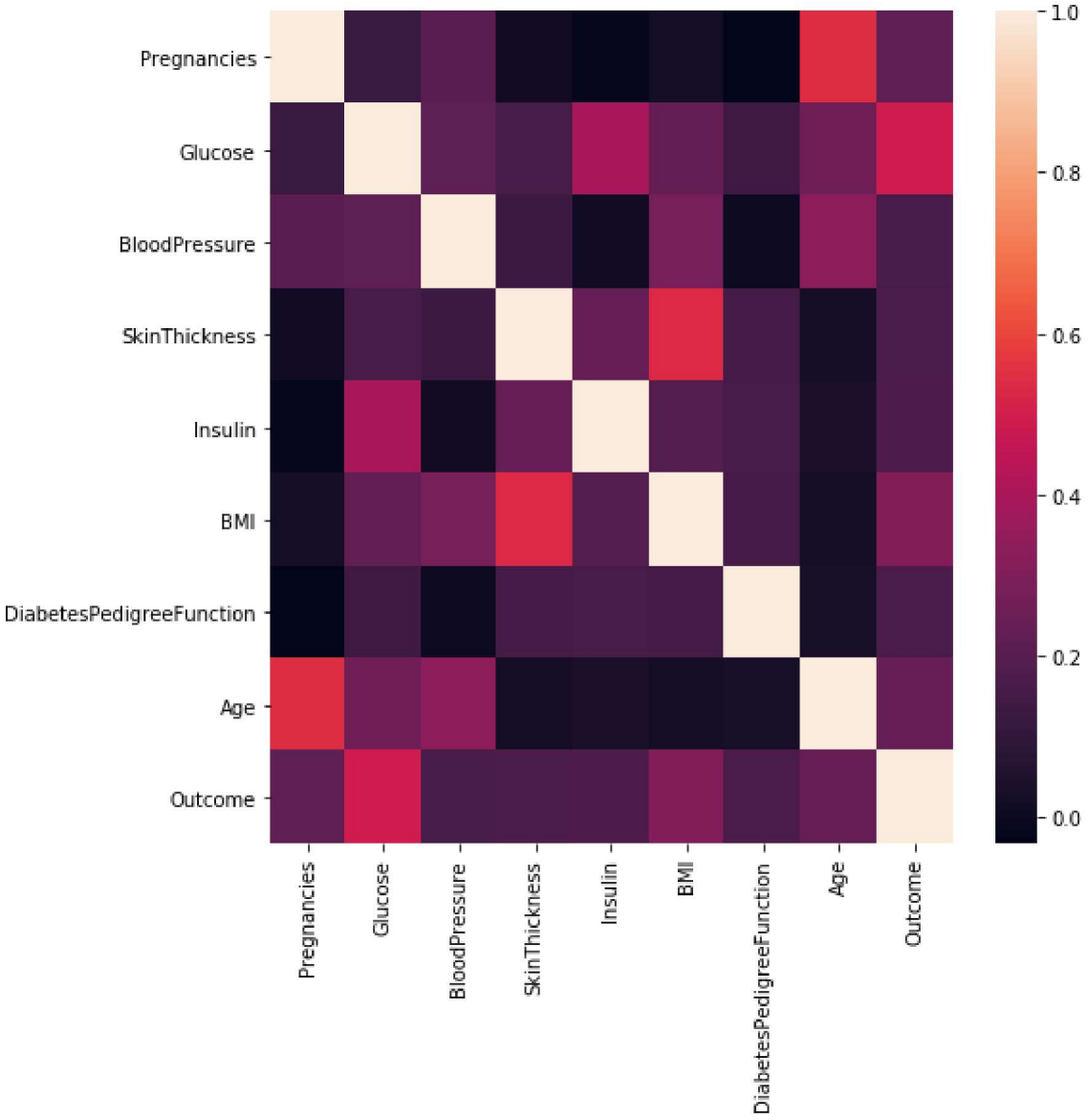


In [82]:

```
### create correlation heat map
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr())
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd7f65bd250>
```

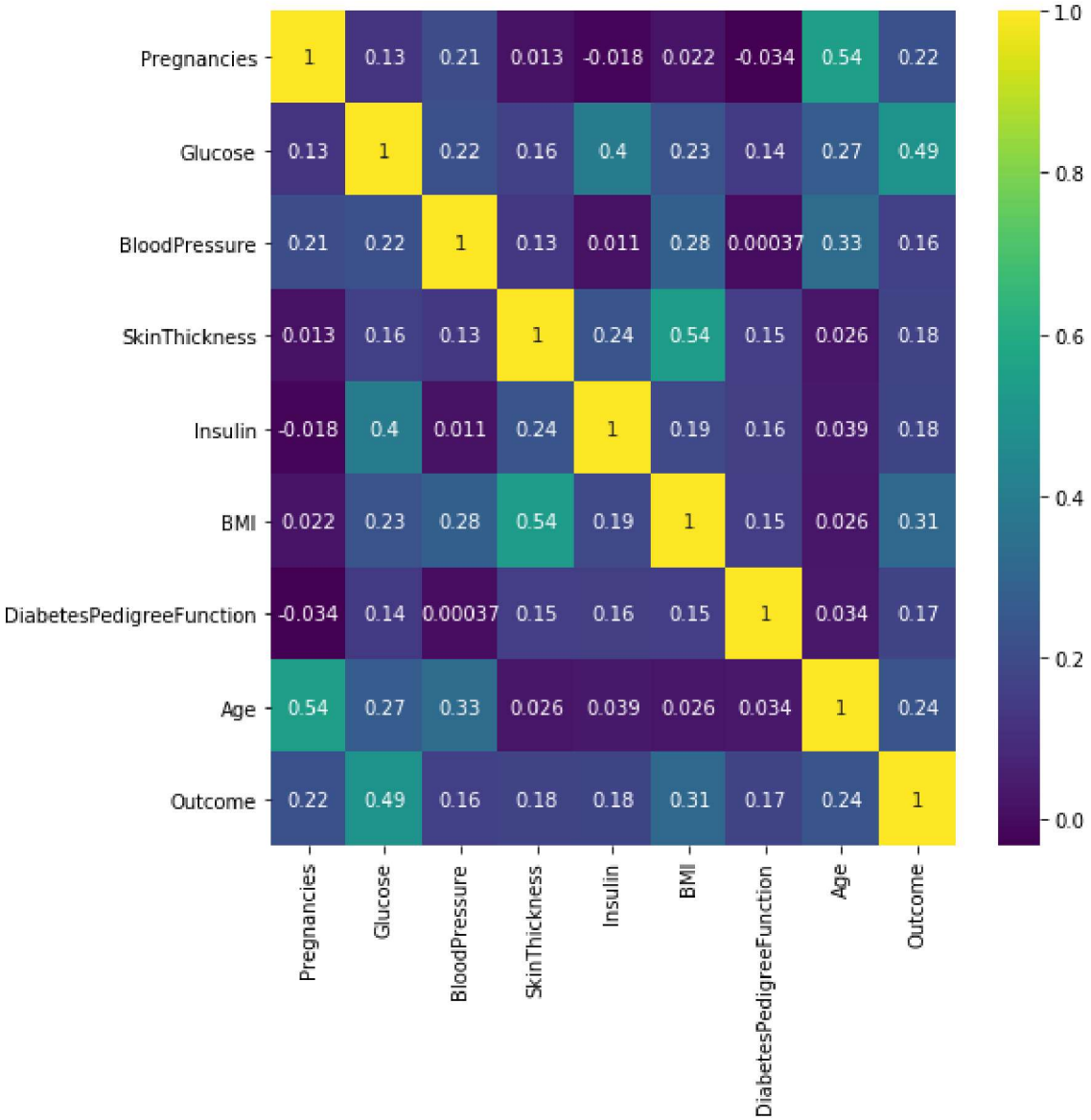

Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

In [83]:

```
### gives correlation value
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True,cmap='viridis')
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd7f6508dd0>
```



Logistic Regreation and model building

In [84]:

```
features = data.iloc[:,0:8].values
label = data.iloc[:,8].values
```

In [85]:

```
features
```

Out[85]:

```
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 [86]:

```
#Train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,
                                                 label,
                                                 test_size=0.2,
                                                 random_state =10)
```

In [87]:

```
from sklearn.preprocessing import StandardScaler
Scale = StandardScaler()
features_std = Scale.fit_transform(features)
print(features_std)
```

```
[[ 0.63994726  0.86527574 -0.0210444  ...  0.16725546  0.46849198
   1.4259954 ]
 [-0.84488505 -1.20598931 -0.51658286 ... -0.85153454 -0.36506078
  -0.19067191]
 [ 1.23388019  2.01597855 -0.68176235 ... -1.33182125  0.60439732
  -0.10558415]
 ...
 [ 0.3429808  -0.02240928 -0.0210444  ... -0.90975111 -0.68519336
  -0.27575966]
 [-0.84488505  0.14197684 -1.01212132 ... -0.34213954 -0.37110101
   1.17073215]
 [-0.84488505 -0.94297153 -0.18622389 ... -0.29847711 -0.47378505
  -0.87137393]]
```

In [88]:

```
X_train_std,X_test_std,y_train,y_test = train_test_split(features_std,
                                                 label,
                                                 test_size=0.2,
                                                 random_state =10)
```

In [89]:

```
print(X_train.shape)
print(X_train)
```

```
(614, 8)
[[0.00000000e+00 1.62000000e+02 7.60000000e+01 ... 5.32000000e+01
  7.59000000e-01 2.50000000e+01]
 [2.00000000e+00 8.70000000e+01 6.91054688e+01 ... 2.89000000e+01
  7.73000000e-01 2.50000000e+01]
 [0.00000000e+00 1.37000000e+02 6.80000000e+01 ... 2.48000000e+01
  1.43000000e-01 2.10000000e+01]
 ...
 [3.00000000e+00 1.16000000e+02 7.40000000e+01 ... 2.63000000e+01
  1.07000000e-01 2.40000000e+01]
 [1.00000000e+00 8.80000000e+01 3.00000000e+01 ... 5.50000000e+01
  4.96000000e-01 2.60000000e+01]
 [5.00000000e+00 9.60000000e+01 7.40000000e+01 ... 3.36000000e+01
  9.97000000e-01 4.30000000e+01]]
```

In [90]:

```
print(X_test.shape)
print(X_test)
```

```
(154, 8)
[[4.00e+00 1.54e+02 7.20e+01 ... 3.13e+01 3.38e-01 3.70e+01]
 [2.00e+00 1.12e+02 8.60e+01 ... 3.84e+01 2.46e-01 2.80e+01]
 [1.00e+00 1.35e+02 5.40e+01 ... 2.67e+01 6.87e-01 6.20e+01]
 ...
 [8.00e+00 8.50e+01 5.50e+01 ... 2.44e+01 1.36e-01 4.20e+01]
 [5.00e+00 1.30e+02 8.20e+01 ... 3.91e+01 9.56e-01 3.70e+01]
 [3.00e+00 9.90e+01 5.40e+01 ... 2.56e+01 1.54e-01 2.40e+01]]
```

In [ ]:

```

```

In [91]:

```
print(X_train_std)
print(X_train_std.shape)
```

```
[[-1.14185152  1.32555687  0.30931457 ...  3.01986745  0.86714764
  -0.70119842]
 [-0.54791859 -1.14023487 -0.260103   ... -0.51678925  0.9094293
  -0.70119842]
 [-1.14185152  0.50362629 -0.35140338 ... -1.11350911 -0.99324546
  -1.04154944]
 ...
 [-0.25095213 -0.1867954   0.14413508 ... -0.89519697 -1.10196973
  -0.78628618]
 [-0.84488505 -1.10735764 -3.48981362 ...  3.28184202  0.07285643
  -0.61611067]
 [ 0.3429808  -0.84433986  0.14413508 ...  0.16725546  1.58593589
   0.83038113]]
(614, 8)
```

In [92]:

```
print(X_test_std)
print(X_test_std.shape)
```

```
[[ 0.04601433  1.06253908 -0.0210444  ... -0.16748983 -0.40432232
   0.31985461]
 [-0.54791859 -0.31830429  1.135212   ...  0.86585431 -0.68217324
  -0.44593516]
 [-0.84488505  0.43787184 -1.50765978 ... -0.83698039  0.6496991
   2.44704844]
 ...
 [ 1.23388019 -1.20598931 -1.42507004 ... -1.17172568 -1.01438629
   0.74529338]
 [ 0.3429808   0.27348573  0.80485303 ...  0.96773331  1.46211102
   0.31985461]
 [-0.25095213 -0.74570819 -1.50765978 ... -0.99707597 -0.96002416
  -0.78628618]]
(154, 8)
```

#Apply Logistic Regression Classifier

In [93]:

```
#Create Logistic Regression model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/sklearn/linear_model/
logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs'
in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[93]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [94]:

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

```
0.7736156351791531
0.7012987012987013
```

In [95]:

```
model_std = LogisticRegression()
model_std.fit(X_train_std,y_train)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/sklearn/linear_model/
logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs'
in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[95]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [96]:

```
print(model_std.score(X_train_std,y_train))
print(model_std.score(X_test_std,y_test))
```

```
0.7719869706840391
0.7337662337662337
```

In [97]:

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

Out[97]:

```
array([[451,  49],
       [136, 132]])
```

In [98]:

```
from sklearn.metrics import confusion_matrix
cm_std= confusion_matrix(label,model_std.predict(features_std))
cm_std
```

Out[98]:

```
array([[442,  58],
       [123, 145]])
```

Preparing ROC Curve (Receiver Operating Characteristics Curve)

In [99]:

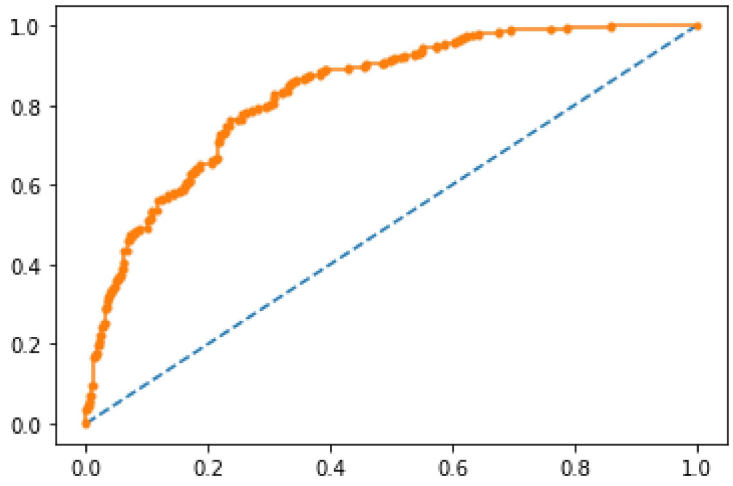
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict probabilities
probs = model.predict_proba(features)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(label, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(label, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
```

```
AUC: 0.834
```

Out[99]:

```
[<matplotlib.lines.Line2D at 0x7fd7f5e4e350>]
```



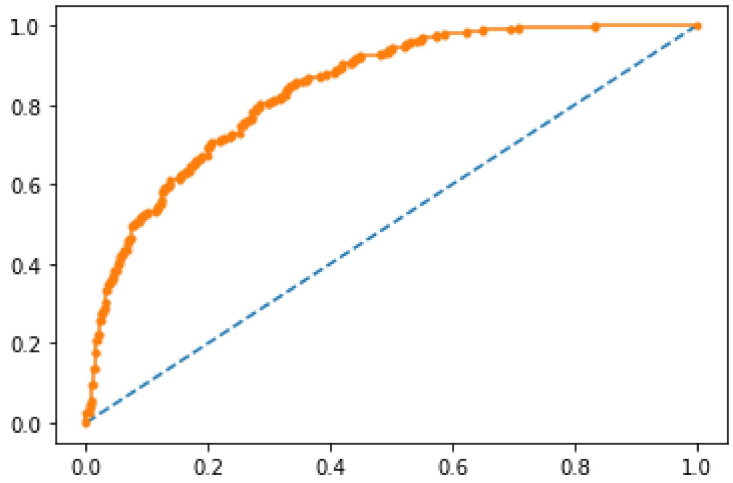
In [100]:

```
# predict probabilities
probs_std = model_std.predict_proba(features_std)
# keep probabilities for the positive outcome only
probs_std = probs_std[:, 1]
# calculate AUC
auc_std = roc_auc_score(label, probs_std)
print('AUC: %.3f' % auc_std)
# calculate roc curve
fpr_std, tpr_std, thresholds_std = roc_curve(label, probs_std)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr_std, tpr_std, marker='.')
```

```
AUC: 0.842
```

Out[100]:

```
[<matplotlib.lines.Line2D at 0x7fd7f5ecbf10>]
```



#Applying Decission Tree Classifier

In [101]:

```
from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier(max_depth=5)
model3_std = DecisionTreeClassifier(max_depth=5)
model3.fit(X_train,y_train)
model3_std.fit(X_train_std,y_train)
```

Out[101]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [102]:

```
print(model3.score(X_train,y_train))
print(model3.score(X_test,y_test))
```

```
0.8257328990228013
0.7402597402597403
```

In [103]:

```
print(model3_std.score(X_train_std,y_train))
print(model3_std.score(X_test_std,y_test))
```

```
0.8273615635179153
0.7532467532467533
```

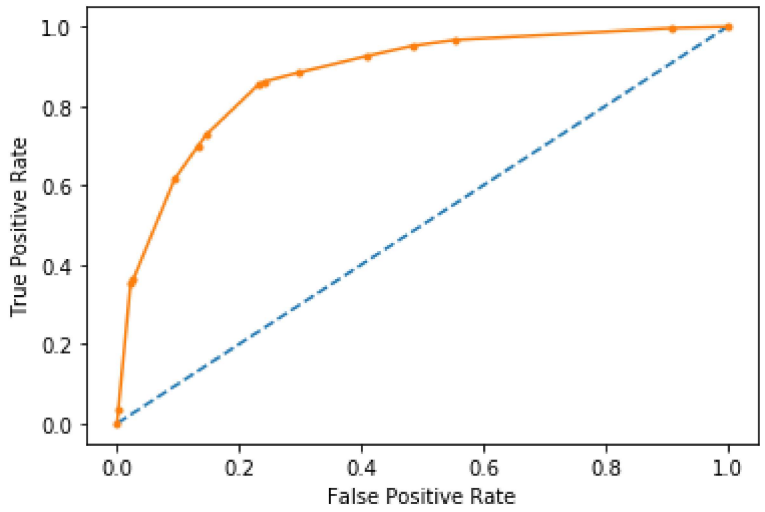
In [104]:

```
#Preparing ROC Curve for Decission Tree Classifier
# predict probabilities
probs3 = model3.predict_proba(features)
# keep probabilities for the positive outcome only
probs3 = probs3[:, 1]
# calculate AUC
auc3 = roc_auc_score(label, probs3)
print('AUC: %.3f' % auc3)
# calculate roc curve
fpr3, tpr3, thresholds3 = roc_curve(label, probs3)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr3,f
pr3,thresholds3))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr3, tpr3, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.876
True Positive Rate - [0.         0.03358209 0.35447761 0.3619403  0.615671
64 0.69776119
 0.72761194 0.85447761 0.8619403  0.88432836 0.92537313 0.95149254
 0.96641791 0.99626866 1.        ], False Positive Rate - [0.    0.002 0.0
22 0.026 0.094 0.132 0.146 0.232 0.242 0.298 0.408 0.484
 0.554 0.908 1.    ] Thresholds - [2.         1.         0.91139241 0.66666
667 0.66233766 0.55172414
 0.5        0.41935484 0.28571429 0.19354839 0.18181818 0.08108108
 0.07407407 0.02702703 0.        ]
```

Out[104]:

```
Text(0, 0.5, 'True Positive Rate')
```



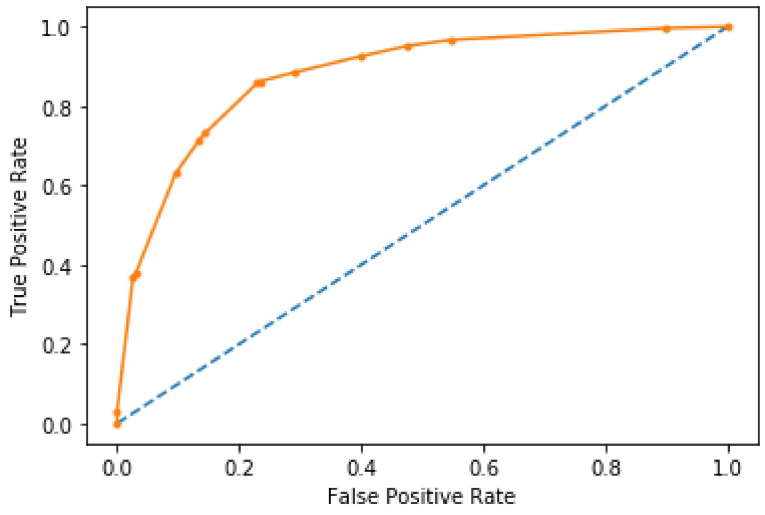
In [105]:

```
#Preparing ROC Curve for Decission Tree Classifier
# predict probabilities
probs3_std = model3_std.predict_proba(features_std)
# keep probabilities for the positive outcome only
probs3_std = probs3_std[:, 1]
# calculate AUC
auc3_std = roc_auc_score(label, probs3_std)
print('AUC: %.3f' % auc3_std)
# calculate roc curve
fpr3_std, tpr3_std, thresholds3_std = roc_curve(label, probs3_std)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr3_s
td,fpr3_std,thresholds3_std))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr3_std, tpr3_std, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.879
True Positive Rate - [0.         0.02985075 0.36940299 0.37686567 0.630597
01 0.71268657
 0.73134328 0.85820896 0.8619403  0.88432836 0.92537313 0.95149254
 0.96641791 0.99626866 1.        ], False Positive Rate - [0.    0.    0.0
26 0.03  0.096 0.134 0.144 0.23  0.234 0.29  0.4   0.476
 0.546 0.9   1.    ] Thresholds - [2.         1.         0.88505747 0.66666
667 0.66233766 0.55172414
 0.5        0.41935484 0.33333333 0.19354839 0.18181818 0.08108108
 0.07407407 0.02702703 0.        ]
```

Out[105]:

```
Text(0, 0.5, 'True Positive Rate')
```



#Applying Random Forest

In [106]:

```
from sklearn.ensemble import RandomForestClassifier
model4 = RandomForestClassifier(n_estimators=11)
model4_std = RandomForestClassifier(n_estimators=11)
model4.fit(X_train,y_train)
model4_std.fit(X_train_std,y_train)
```

Out[106]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
                       max_depth=None, max_features='auto', max_leaf_nodes
=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=11,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [107]:

```
print(model4.score(X_train,y_train))
print(model4.score(X_test,y_test))
```

```
0.993485342019544
0.7337662337662337
```

In [108]:

```
print(model4_std.score(X_train_std,y_train))
print(model4_std.score(X_test_std,y_test))
```

```
0.990228013029316
0.7402597402597403
```

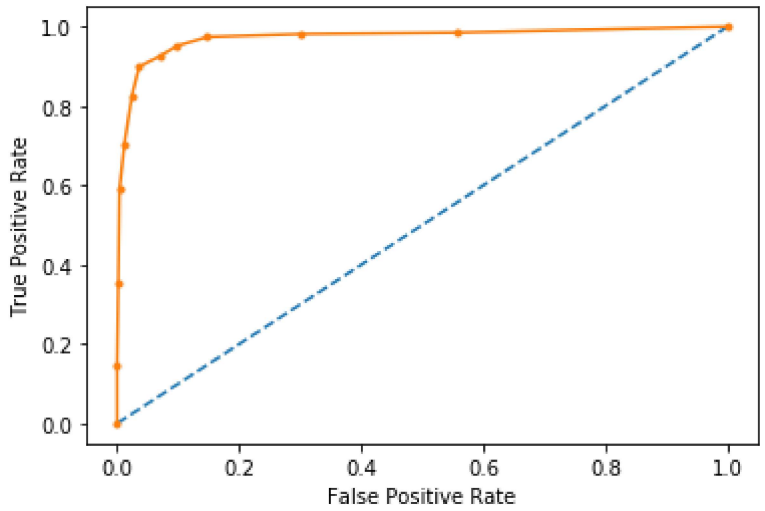
In [109]:

```
#Preparing ROC Curve for Random Forest Classifier
# predict probabilities
probs4 = model4.predict_proba(features)
# keep probabilities for the positive outcome only
probs4 = probs4[:, 1]
# calculate AUC
auc4 = roc_auc_score(label, probs4)
print('AUC: %.3f' % auc4)
# calculate roc curve
fpr4, tpr4, thresholds4 = roc_curve(label, probs4)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr4,f
pr4,thresholds4))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr4, tpr4, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.973
True Positive Rate - [0.         0.14552239 0.35447761 0.58955224 0.705223
88 0.82462687
 0.89925373 0.92537313 0.95149254 0.9738806  0.98134328 0.98507463
 1.        ], False Positive Rate - [0.    0.    0.002 0.004 0.012 0.024
0.036 0.07  0.098 0.148 0.3   0.558
 1.   ] Thresholds - [2.         1.         0.90909091 0.81818182 0.727272
73 0.63636364
 0.54545455 0.45454545 0.36363636 0.27272727 0.18181818 0.09090909
 0.        ]
```

Out[109]:

```
Text(0, 0.5, 'True Positive Rate')
```



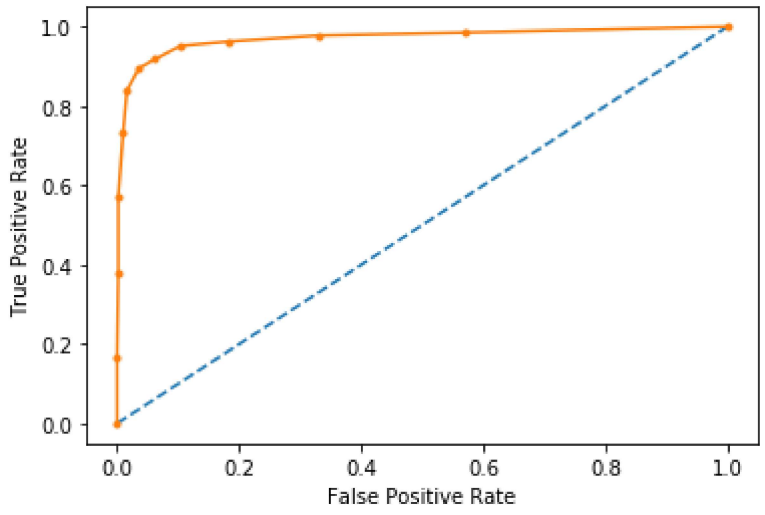
In [110]:

```
#Preparing ROC Curve for Random Forest Classifier
# predict probabilities
probs4_std = model4_std.predict_proba(features_std)
# keep probabilities for the positive outcome only
probs4_std = probs4_std[:, 1]
# calculate AUC
auc4_std = roc_auc_score(label, probs4_std)
print('AUC: %.3f' % auc4_std)
# calculate roc curve
fpr4_std, tpr4_std, thresholds4_std = roc_curve(label, probs4_std)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr4_s
td,fpr4_std,thresholds4_std))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr4_std, tpr4_std, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.971
True Positive Rate - [0.         0.16791045 0.38059701 0.57089552 0.735074
63 0.83955224
 0.89552239 0.91791045 0.95149254 0.96268657 0.97761194 0.98507463
 1.        ], False Positive Rate - [0.    0.    0.002 0.002 0.01  0.016
0.036 0.062 0.104 0.184 0.33  0.57
 1.   ] Thresholds - [2.         1.         0.90909091 0.81818182 0.727272
73 0.63636364
 0.54545455 0.45454545 0.36363636 0.27272727 0.18181818 0.09090909
 0.        ]
```

Out[110]:

```
Text(0, 0.5, 'True Positive Rate')
```



#Support Vector Machine Classifier

In [111]:

```
from sklearn.svm import SVC
model5 = SVC(kernel='rbf',
             gamma='auto',
            probability=True,
              C=0.01)
model5.fit(X_train,y_train)

model5_std = SVC(kernel='rbf',
             gamma='auto',
          probability=True,
              C=0.01)
model5_std.fit(X_train_std,y_train)
```

Out[111]:

```
SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=
0.001,
    verbose=False)
```

In [112]:

```
print(model5.score(X_train,y_train))
print(model5.score(X_test,y_test))
```

```
0.6596091205211726
0.6168831168831169
```

In [113]:

```
print(model5_std.score(X_train_std,y_train))
print(model5_std.score(X_test_std,y_test))
```

```
0.6596091205211726
0.6168831168831169
```

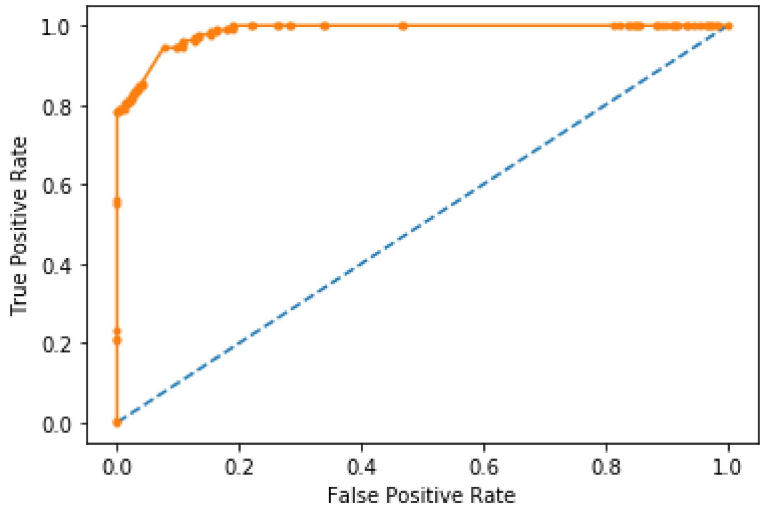
In [114]:

```
#Preparing ROC Curve for SVC
# predict probabilities
probs5 = model5.predict_proba(features)
# keep probabilities for the positive outcome only
probs5 = probs5[:, 1]
# calculate AUC
auc5 = roc_auc_score(label, probs5)
print('AUC: %.3f' % auc5)
# calculate roc curve
fpr5, tpr5, thresholds5 = roc_curve(label, probs5)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr5,f
pr5,thresholds5))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr5, tpr5, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.985
True Positive Rate - [0.         0.00373134 0.20522388 0.21268657 0.231343
28 0.54850746
 0.55970149 0.78358209 0.78358209 0.78731343 0.78731343 0.79104478
 0.79104478 0.80223881 0.80223881 0.80597015 0.80597015 0.80970149
 0.80970149 0.81343284 0.81343284 0.81716418 0.81716418 0.82835821
 0.82835821 0.8358209  0.8358209  0.83955224 0.83955224 0.84701493
 0.84701493 0.85447761 0.94402985 0.94402985 0.94776119 0.94776119
 0.95522388 0.95522388 0.96268657 0.96268657 0.96641791 0.96641791
 0.97014925 0.97014925 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.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.        ], False Positive Rate - [0.
0.    0.    0.    0.    0.    0.    0.    0.002 0.002 0.004 0.004
 0.012 0.012 0.014 0.014 0.016 0.016 0.02  0.02  0.022 0.022 0.024 0.024
 0.028 0.028 0.032 0.032 0.036 0.036 0.04  0.04  0.076 0.098 0.098 0.106
 0.106 0.108 0.108 0.126 0.126 0.128 0.128 0.134 0.134 0.152 0.152 0.154
 0.154 0.164 0.164 0.18  0.18  0.188 0.188 0.19  0.19  0.218 0.222 0.262
 0.266 0.28  0.284 0.336 0.34  0.464 0.468 0.812 0.824 0.836 0.84  0.846
 0.85  0.852 0.856 0.882 0.886 0.892 0.9   0.908 0.912 0.914 0.918 0.932
 0.936 0.944 0.954 0.964 0.968 0.972 0.976 0.98  0.984 1.    ] Thresholds -
[1.98788406 0.98788406 0.98499468 0.98499468 0.98499468 0.98499468
 0.98499468 0.614445   0.4245071  0.40276772 0.36060239 0.3605665
 0.34661406 0.34567346 0.34557384 0.34554316 0.34554069 0.3455201
 0.345474   0.34547184 0.34546963 0.34546856 0.34546846 0.34546803
 0.34546785 0.3454677  0.34546768 0.34546766 0.34546765 0.34546765
 0.34546765 0.34546765 0.34546765 0.34546765 0.34546765 0.34546764
 0.34546763 0.34546762 0.34546759 0.34546559 0.34546481 0.34546336
 0.34546265 0.34546093 0.34546024 0.34535235 0.34483476 0.34482692
 0.34451958 0.3425674  0.34240585 0.33584199 0.33189054 0.30123197
 0.28968381 0.28625857 0.22594237 0.04605443 0.04605443 0.04605443
 0.04605443 0.04605443 0.04605443 0.0460544  0.0460544  0.04605286
 0.04605244 0.03499126 0.03498408 0.0347773  0.0347773  0.03475263
 0.03475263 0.03475263 0.03475094 0.03432844 0.03432844 0.03427075
 0.03427044 0.03424309 0.03424309 0.03424308 0.03424308 0.03423309
 0.03423309 0.03422806 0.03422806 0.03422391 0.03422371 0.03420914
 0.03420886 0.03420859 0.03420859 0.02942379]
```

Out[114]:

```
Text(0, 0.5, 'True Positive Rate')
```


True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

In [115]:

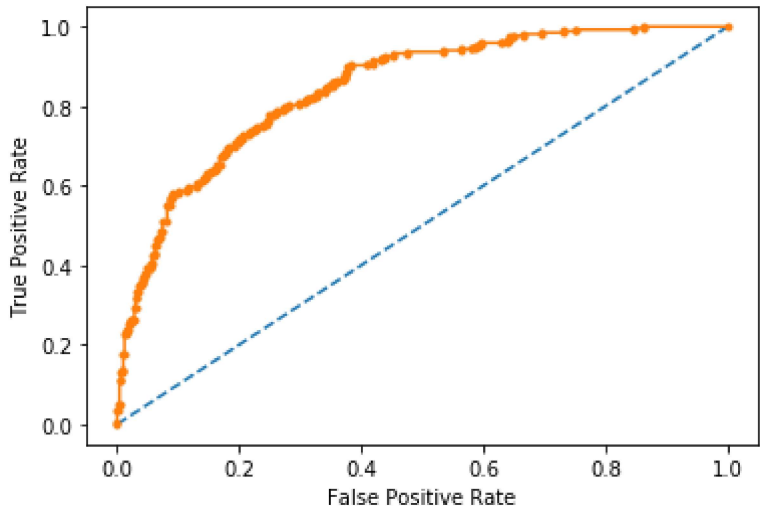
```
#Preparing ROC Curve for SVC
# predict probabilities
probs5_std = model5_std.predict_proba(features_std)
# keep probabilities for the positive outcome only
probs5_std = probs5_std[:, 1]
# calculate AUC
auc5_std = roc_auc_score(label, probs5_std)
print('AUC: %.3f' % auc5_std)
# calculate roc curve
fpr5_std, tpr5_std, thresholds5_std = roc_curve(label, probs5_std)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr5_s
td,fpr5_std,thresholds5_std))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr5_std, tpr5_std, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.846
True Positive Rate - [0.         0.00373134 0.03358209 0.03358209 0.048507
46 0.04850746
 0.1119403  0.1119403  0.13059701 0.13059701 0.13432836 0.13432836
 0.17537313 0.17537313 0.22761194 0.22761194 0.23134328 0.23134328
 0.23507463 0.23507463 0.25373134 0.25373134 0.25746269 0.25746269
 0.26119403 0.26119403 0.26492537 0.26492537 0.29477612 0.29477612
 0.31716418 0.31716418 0.33208955 0.33208955 0.35074627 0.35074627
 0.35447761 0.35447761 0.35820896 0.35820896 0.36940299 0.36940299
 0.38059701 0.38059701 0.39179104 0.39179104 0.39552239 0.39552239
 0.40671642 0.40671642 0.42537313 0.42537313 0.42910448 0.42910448
 0.44776119 0.44776119 0.46641791 0.46641791 0.47014925 0.47014925
 0.48507463 0.48507463 0.51119403 0.51119403 0.54850746 0.54850746
 0.55223881 0.55223881 0.56716418 0.56716418 0.57089552 0.57089552
 0.57835821 0.57835821 0.58208955 0.58208955 0.5858209  0.5858209
 0.59328358 0.59328358 0.59701493 0.59701493 0.60447761 0.60447761
 0.6119403  0.61567164 0.61567164 0.61940299 0.61940299 0.63059701
 0.63059701 0.6380597  0.6380597  0.64179104 0.64179104 0.65298507
 0.65298507 0.67164179 0.67164179 0.67537313 0.67537313 0.68283582
 0.68283582 0.69402985 0.69402985 0.69776119 0.69776119 0.70895522
 0.70895522 0.71268657 0.71268657 0.71641791 0.71641791 0.72014925
 0.72014925 0.72761194 0.72761194 0.73134328 0.73134328 0.73880597
 0.73880597 0.74253731 0.74253731 0.75       0.75       0.75373134
 0.75373134 0.76119403 0.76119403 0.7761194  0.7761194  0.77985075
 0.77985075 0.78358209 0.78358209 0.78731343 0.78731343 0.79477612
 0.79477612 0.79850746 0.79850746 0.80223881 0.80223881 0.80970149
 0.80970149 0.81343284 0.81343284 0.81716418 0.81716418 0.82089552
 0.82089552 0.82462687 0.82462687 0.83208955 0.83208955 0.8358209
 0.8358209  0.84328358 0.84328358 0.84701493 0.84701493 0.85074627
 0.85074627 0.85820896 0.85820896 0.8619403  0.8619403  0.86567164
 0.86567164 0.86940299 0.86940299 0.87686567 0.87686567 0.88432836
 0.88432836 0.89925373 0.89925373 0.90298507 0.90298507 0.90671642
 0.90671642 0.9141791  0.9141791  0.91791045 0.91791045 0.92164179
 0.92164179 0.92537313 0.92537313 0.93283582 0.93283582 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 0.9738806
 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 0.016 0.016 0.018 0.018 0.02  0.02  0.022
 0.022 0.024 0.024 0.028 0.028 0.03  0.03  0.032 0.032 0.034 0.034 0.038
 0.038 0.04  0.04  0.042 0.042 0.044 0.044 0.048 0.048 0.054 0.054 0.056
 0.056 0.058 0.058 0.06  0.06  0.062 0.062 0.064 0.064 0.068 0.068 0.07
 0.07  0.074 0.074 0.082 0.082 0.084 0.084 0.086 0.086 0.088 0.088 0.09
 0.09  0.092 0.092 0.102 0.102 0.112 0.112 0.118 0.118 0.13  0.13  0.134
 0.14  0.14  0.144 0.144 0.148 0.148 0.152 0.152 0.158 0.158 0.164 0.164
 0.168 0.168 0.174 0.174 0.176 0.176 0.18  0.18  0.184 0.184 0.194 0.194
 0.198 0.198 0.2   0.2   0.202 0.202 0.206 0.206 0.216 0.216 0.22  0.22
 0.224 0.224 0.23  0.23  0.24  0.24  0.246 0.246 0.248 0.248 0.254 0.254
 0.258 0.258 0.26  0.26  0.272 0.272 0.274 0.274 0.28  0.28  0.296 0.296
 0.308 0.308 0.31  0.31  0.314 0.314 0.32  0.32  0.328 0.328 0.332 0.332
 0.34  0.34  0.342 0.342 0.346 0.346 0.352 0.352 0.354 0.354 0.36  0.36
 0.37  0.37  0.372 0.372 0.374 0.374 0.378 0.378 0.384 0.384 0.41  0.41
 0.418 0.418 0.432 0.432 0.434 0.434 0.44  0.44  0.452 0.452 0.476 0.476
 0.534 0.534 0.564 0.564 0.58  0.58  0.588 0.588 0.594 0.594 0.596 0.596
 0.63  0.63  0.638 0.638 0.642 0.642 0.65  0.65  0.664 0.664 0.696 0.696
 0.732 0.732 0.75  0.75  0.846 0.846 0.862 0.862 1.   ] Thresholds - [1.98
564012 0.98564012 0.96490284 0.96291139 0.95001984 0.94669933
 0.90806531 0.90705592 0.90363741 0.90353866 0.9031077  0.90045801
```

```
0.88850277 0.88494191 0.86614526 0.86320817 0.8593209  0.85624909
0.8530289  0.85013069 0.84670592 0.84532797 0.8437096  0.84011748
0.83588362 0.83212657 0.83184498 0.82971116 0.81645697 0.8100872
0.80157656 0.80041596 0.79367073 0.79325988 0.78687887 0.78352636
0.78094588 0.77714601 0.77597376 0.77528723 0.76377309 0.76163715
0.75464675 0.75278284 0.73777192 0.73108559 0.73025515 0.72933405
0.72393004 0.72266681 0.71030586 0.70612245 0.70439735 0.70416885
0.68652816 0.68344738 0.67475388 0.65629785 0.65285292 0.64766636
0.63259978 0.63133826 0.61786535 0.61086046 0.59463939 0.59434967
0.59282608 0.58961955 0.58351005 0.58225596 0.57912276 0.57112902
0.56967345 0.56920751 0.56812658 0.55563943 0.55501599 0.53527709
0.53179009 0.52699286 0.52601944 0.5164182  0.50992409 0.50502232
0.5        0.49377889 0.49232705 0.49200029 0.48921867 0.48500123
0.47628156 0.47389883 0.46863274 0.46801399 0.4629532  0.45427631
0.44933427 0.44137658 0.44015102 0.43853077 0.43570019 0.43464385
0.4299516  0.42346182 0.42132739 0.41742992 0.40694035 0.40581384
0.40246134 0.40203329 0.40177122 0.40161525 0.39863133 0.39808066
0.39594494 0.39316866 0.3813083  0.38041085 0.37933064 0.37694526
0.37444178 0.37042879 0.36705477 0.36449332 0.35932141 0.35547852
0.34925145 0.34666    0.34557223 0.33849103 0.33699632 0.33603173
0.3309685  0.32984133 0.32957127 0.32357838 0.31495779 0.30900313
0.30828017 0.30813554 0.30582648 0.30476676 0.29781141 0.29689491
0.2843877  0.28301812 0.28252938 0.28199924 0.27761652 0.27750251
0.2746569  0.27410537 0.27282926 0.26932886 0.26530637 0.26237134
0.25707941 0.25550892 0.25506668 0.254274   0.25156172 0.25139121
0.24864105 0.24699026 0.24686088 0.24634378 0.24452525 0.24173283
0.23583522 0.2354071  0.2350383  0.23364342 0.23224268 0.22963113
0.22665241 0.22006498 0.21871699 0.21784349 0.21049043 0.21039684
0.20366015 0.2023459  0.19007396 0.18961664 0.188247   0.18683289
0.18443343 0.1837697  0.17731921 0.17642823 0.16881122 0.16834049
0.15807796 0.1569068  0.15146482 0.15035928 0.1467797  0.14635311
0.14425083 0.14378177 0.14274225 0.14271753 0.14144103 0.14136563
0.13691925 0.13611328 0.13210116 0.1312427  0.13103862 0.13077583
0.12889413 0.12861837 0.12471651 0.12416263 0.11921876 0.11917417
0.11335655 0.11333674 0.11058189 0.11055484 0.0973823  0.09690351
0.09501145 0.09497472 0.0718217 ]
```

Out[115]:

```
Text(0, 0.5, 'True Positive Rate')
```



#Applying K-NN

In [116]:

```
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=7,
                              metric='minkowski',
                              p = 2)
model2_std = KNeighborsClassifier(n_neighbors=7,
                              metric='minkowski',
                              p = 2)
model2.fit(X_train,y_train)
model2_std.fit(X_train_std,y_train)
```

Out[116]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

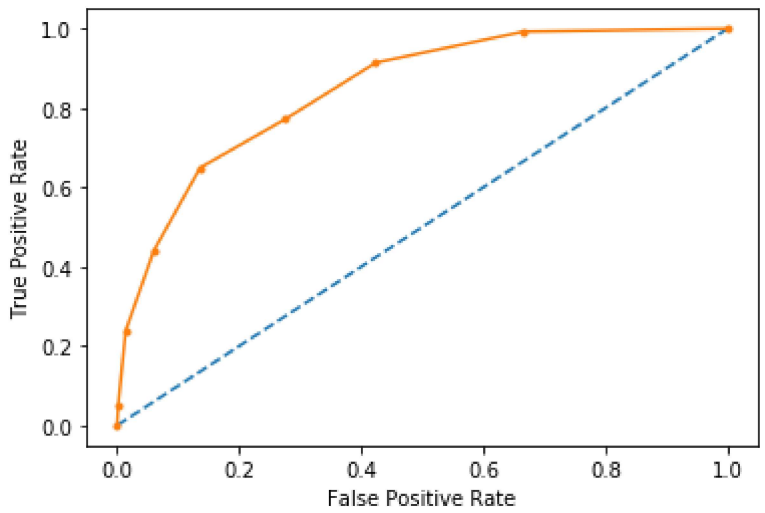
In [117]:

```
#Preparing ROC Curve for K-NN
# predict probabilities
probs2 = model2.predict_proba(features)
# keep probabilities for the positive outcome only
probs2 = probs2[:, 1]
# calculate AUC
auc2 = roc_auc_score(label, probs2)
print('AUC: %.3f' % auc2)
# calculate roc curve
fpr2, tpr2, thresholds2 = roc_curve(label, probs2)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr2,f
pr2,thresholds2))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr2, tpr2, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.847
True Positive Rate - [0.         0.05223881 0.23880597 0.44029851 0.649253
73 0.77238806
 0.9141791  0.99253731 1.        ], False Positive Rate - [0.    0.002 0.0
14 0.06  0.136 0.276 0.424 0.666 1.   ] Thresholds - [2.         1.
0.85714286 0.71428571 0.57142857 0.42857143
 0.28571429 0.14285714 0.        ]
```

Out[117]:

```
Text(0, 0.5, 'True Positive Rate')
```



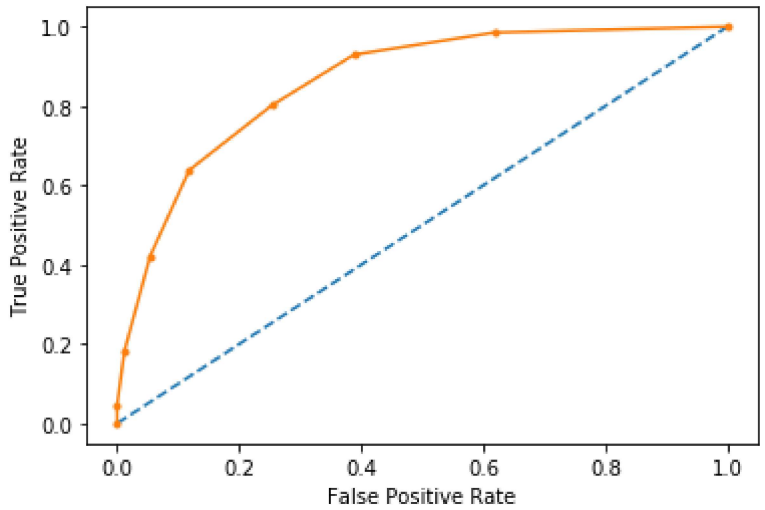
In [118]:

```
#Preparing ROC Curve for K-NN
# predict probabilities
probs2_std = model2_std.predict_proba(features_std)
# keep probabilities for the positive outcome only
probs2_std = probs2_std[:, 1]
# calculate AUC
auc2_std = roc_auc_score(label, probs2_std)
print('AUC: %.3f' % auc2_std)
# calculate roc curve
fpr2_std, tpr2_std, thresholds2_std = roc_curve(label, probs2_std)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr2_s
td,fpr2_std,thresholds2_std))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr2_std, tpr2_std, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.861
True Positive Rate - [0.         0.04477612 0.18283582 0.42164179 0.638059
7  0.80223881
 0.92910448 0.98507463 1.        ], False Positive Rate - [0.    0.    0.0
12 0.054 0.118 0.254 0.388 0.618 1.   ] Thresholds - [2.         1.
0.85714286 0.71428571 0.57142857 0.42857143
 0.28571429 0.14285714 0.        ]
```

Out[118]:

```
Text(0, 0.5, 'True Positive Rate')
```



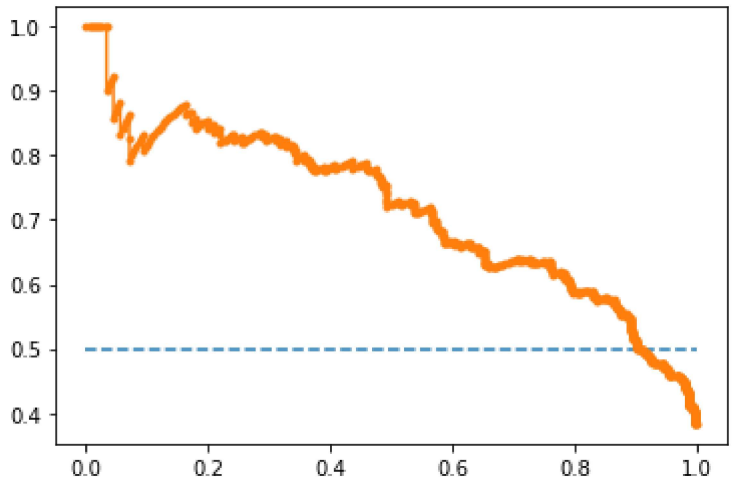
In [119]:

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

```
f1=0.588 auc=0.717 ap=0.718
```

Out[119]:

```
[<matplotlib.lines.Line2D at 0x7fd7f56470d0>]
```



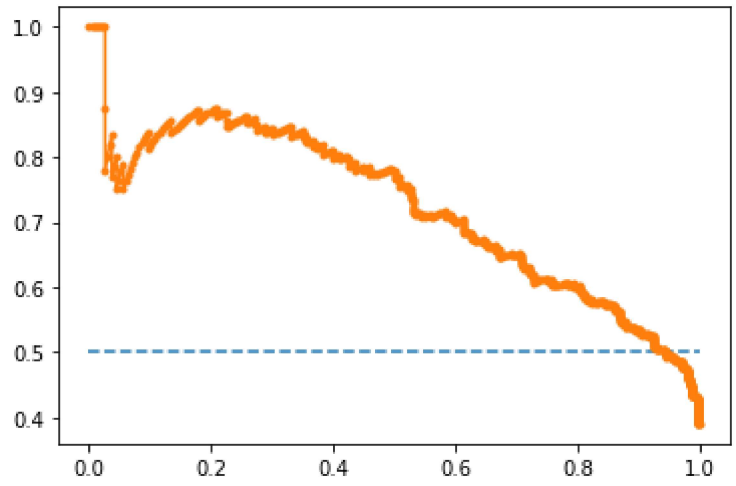
In [120]:

```
#Precision Recall Curve for Logistic Regression
from sklearn.metrics import auc
# predict probabilities
probs_std_rc = model_std.predict_proba(features_std)
# keep probabilities for the positive outcome only
probs_std_rc = probs_std_rc[:, 1]
# predict class values
yhat_std_rc = model_std.predict(features_std)
# calculate precision-recall curve
precision_std_rc, recall_std_rc, thresholds_std_rc = precision_recall_curve(label, prob
s_std_rc)
# calculate F1 score
f1_std_rc = f1_score(label, yhat_std_rc)
# calculate precision-recall AUC
auc_std_rc = auc(recall_std_rc, precision_std_rc)
# calculate average precision score
ap_std_rc = average_precision_score(label, probs_std_rc)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1_std_rc, auc_std_rc, ap_std_rc))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall_std_rc, precision_std_rc, marker='.')
```

```
f1=0.616 auc=0.723 ap=0.724
```

Out[120]:

```
[<matplotlib.lines.Line2D at 0x7fd7f5633250>]
```



In [ ]:

In [121]:

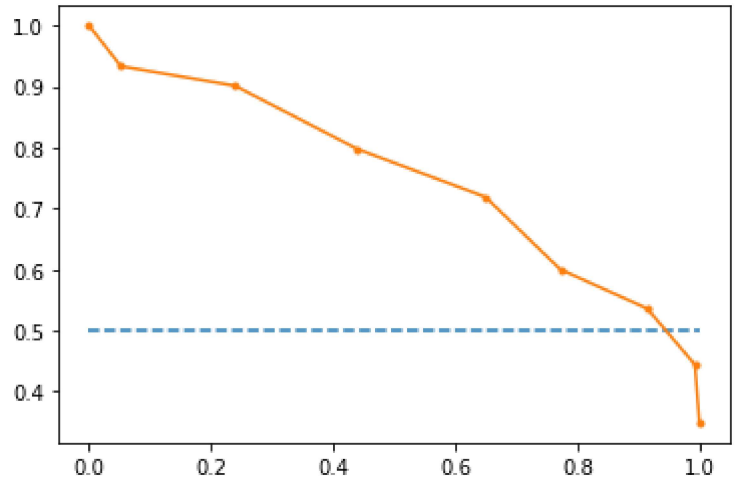
```
#Precision Recall Curve for KNN

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 = model2.predict_proba(features)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model2.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, probs)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.682 auc=0.754 ap=0.715
```

Out[121]:

```
[<matplotlib.lines.Line2D at 0x7fd7f5590a50>]
```



In [122]:

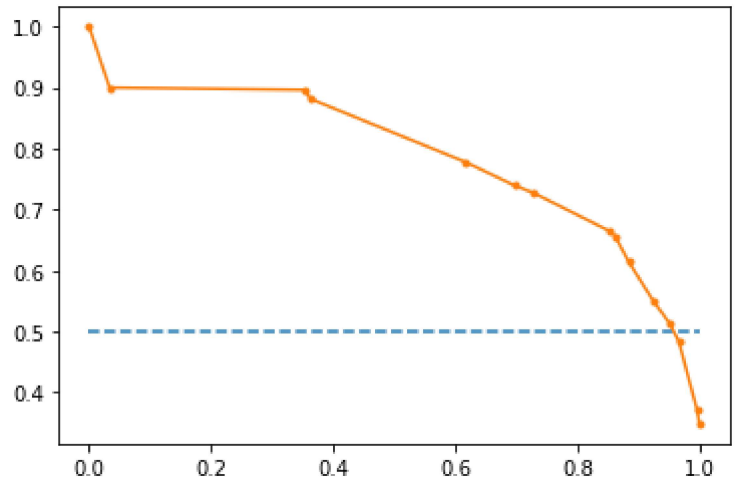
```
#Precision Recall Curve for Decission Tree Classifier

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

```
f1=0.718 auc=0.788 ap=0.763
```

Out[122]:

```
[<matplotlib.lines.Line2D at 0x7fd7f55a54d0>]
```



In [123]:

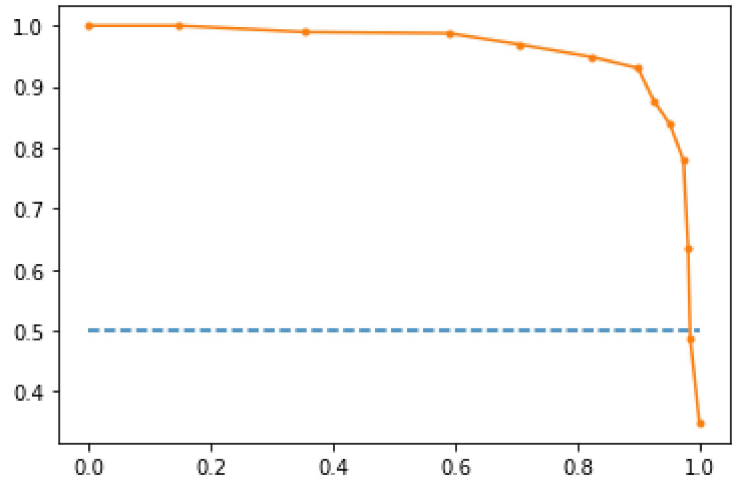
```
#Precision Recall Curve for Random Forest

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

```
f1=0.915 auc=0.961 ap=0.953
```

Out[123]:

```
[<matplotlib.lines.Line2D at 0x7fd7f5494350>]
```



In [124]:

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

```
              precision    recall  f1-score   support

           0       0.77      0.90      0.83       500
           1       0.73      0.49      0.59       268

    accuracy                           0.76       768
   macro avg       0.75      0.70      0.71       768
weighted avg       0.75      0.76      0.75       768

              precision    recall  f1-score   support

           0       0.78      0.88      0.83       500
           1       0.71      0.54      0.62       268

    accuracy                           0.76       768
   macro avg       0.75      0.71      0.72       768
weighted avg       0.76      0.76      0.76       768
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: