Healthcare

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

1. Visually explore these variables using histograms. Treat the missing values accordingly.
2. 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 [63]:
```
### import libraries
import numpy as np
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
```

In [64]:
```
data = pd.read_csv('health care diabetes.csv')
```

In [65]:
```
data.head()
```

Out[65]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [66]:
```
# replace missing values with their mean

data['Glucose']=data.Glucose.mask(data.Glucose == 0,data['Glucose'].mean(skipna=True))

data['BloodPressure']=data.BloodPressure.mask(data.BloodPressure == 0,data['BloodPressur
e'].mean(skipna=True))

data['SkinThickness']=data.SkinThickness.mask(data.SkinThickness == 0,data['SkinThicknes
s'].mean(skipna=True))

data['Insulin']=data.Insulin.mask(data.Insulin == 0,data['Insulin'].mean(skipna=True))

data['BMI']=data.BMI.mask(data.BMI == 0,data['BMI'].mean(skipna=True))

# data = data[data['Glucose'] != 0]
# data = data[data['BloodPressure'] != 0]
# data = data[data['SkinThickness'] != 0]
# data = data[data['Insulin'] != 0]
# data = data[data['BMI'] != 0]
```

In [67]:
```
# Checking if any data is null or not
data.isnull().any()
```

Out[67]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [68]:
```
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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

In [69]:
```
Positive = data[data['Outcome']==1]
Positive.head(5)
```

Out[69]:

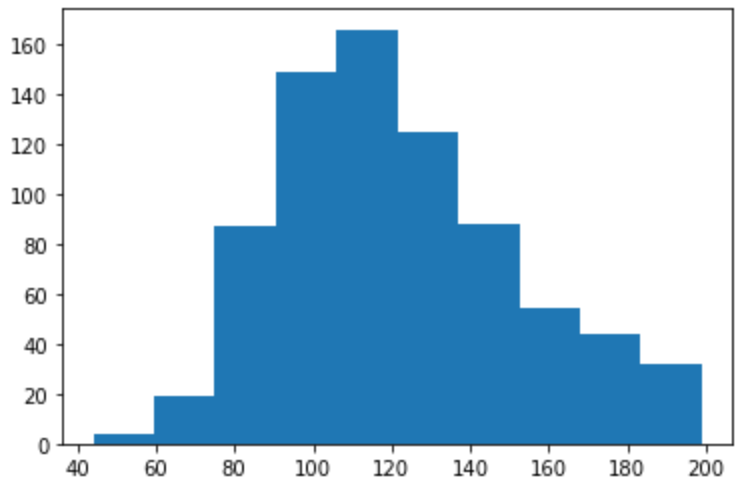
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	C
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	
6	3	78.0	50.0	32.000000	88.000000	31.0	0.248	26	
8	2	197.0	70.0	45.000000	543.000000	30.5	0.158	53	

In [70]:
```
data['Glucose'].value_counts().head(8)
```

Out[70]:
```
100.0    17
99.0     17
125.0    14
106.0    14
111.0    14
129.0    14
108.0    13
102.0    13
Name: Glucose, dtype: int64
```

In [71]:
```
plt.hist(data['Glucose'])
```

Out[71]:
```
(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. ]),
 <BarContainer object of 10 artists>)
```

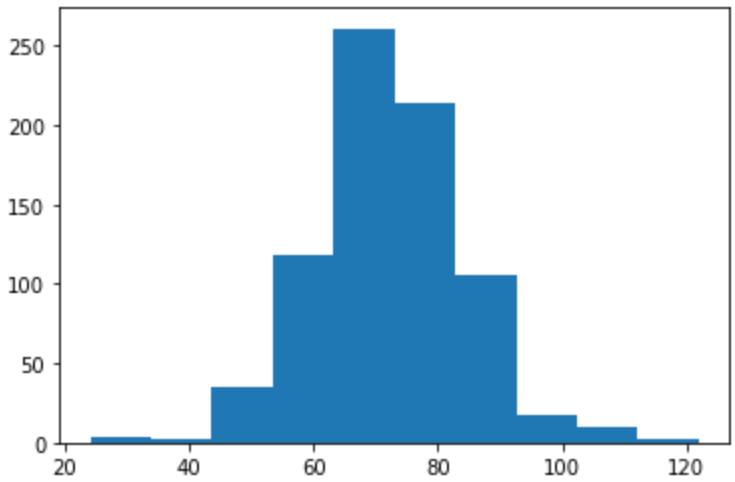


In [72]:
```
data['BloodPressure'].value_counts().head(7)
```

Out[72]:
```
70.0    57
74.0    52
68.0    45
78.0    45
72.0    44
64.0    43
80.0    40
Name: BloodPressure, dtype: int64
```

In [73]:
```
plt.hist(data['BloodPressure'])
```

Out[73]:
```
(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. ]),
 <BarContainer object of 10 artists>)
```

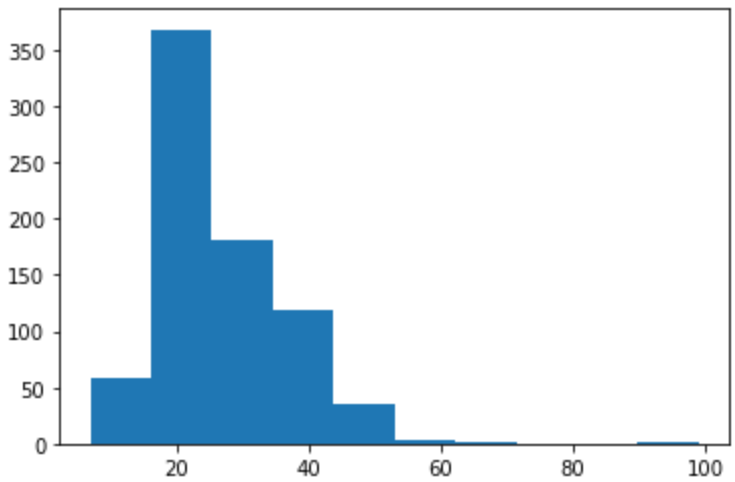


In [74]:
```
data['SkinThickness'].value_counts().head(7)
```

Out[74]:
```
20.536458    227
32.000000     31
30.000000     27
27.000000     23
23.000000     22
33.000000     20
18.000000     20
Name: SkinThickness, dtype: int64
```

In [75]:
```
plt.hist(data['SkinThickness'])
```

Out[75]:
```
(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. ]),
 <BarContainer object of 10 artists>)
```

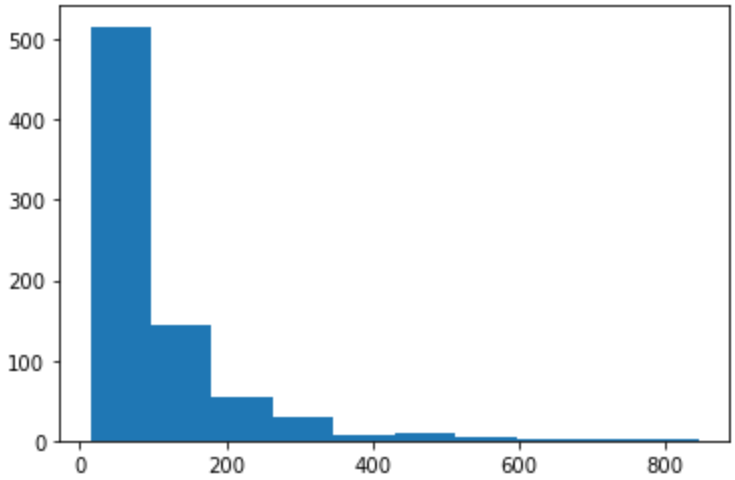


In [76]:
```
data['Insulin'].value_counts().head(7)
```

Out[76]:
```
79.799479     374
105.000000     11
130.000000      9
140.000000      9
120.000000      8
94.000000       7
180.000000      7
Name: Insulin, dtype: int64
```

In [77]:
```
plt.hist(data['Insulin'])
```

Out[77]:
```
(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. ]),
 <BarContainer object of 10 artists>)
```

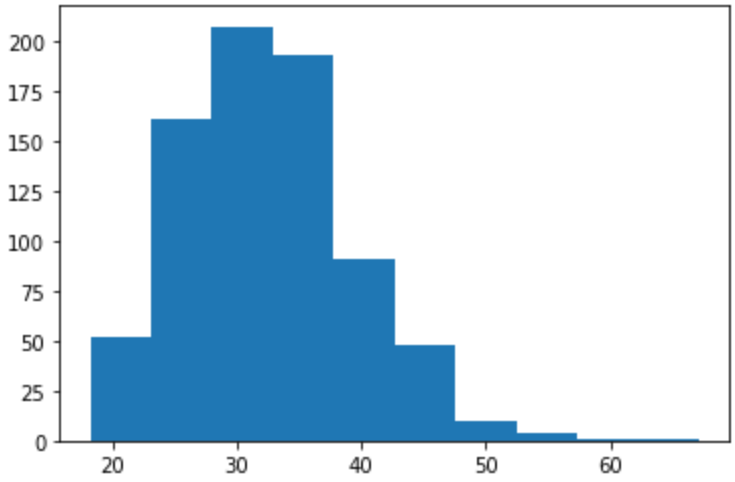


In [78]:
```
data['BMI'].value_counts().head(7)
```

Out[78]:
```
32.000000    13
31.600000    12
31.200000    12
31.992578    11
33.300000    10
32.400000    10
32.900000     9
Name: BMI, dtype: int64
```

In [79]:
```
plt.hist(data['BMI'])
```

Out[79]:
```
(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 ]),
 <BarContainer object of 10 artists>)
```



In [80]:
```
data.describe().transpose()
```

Out[80]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.000000	3.000000	6.00000	17.00
Glucose	768.0	121.681605	30.436016	44.000	99.750000	117.000000	140.25000	199.00
BloodPressure	768.0	72.254807	12.115932	24.000	64.000000	72.000000	80.00000	122.00
SkinThickness	768.0	26.606479	9.631241	7.000	20.536458	23.000000	32.00000	99.00
Insulin	768.0	118.660163	93.080358	14.000	79.799479	79.799479	127.25000	846.00
BMI	768.0	32.450805	6.875374	18.200	27.500000	32.000000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.243750	0.372500	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.000000	29.000000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.000000	0.000000	1.00000	1.00

Thank You

In [ ]:
```
```

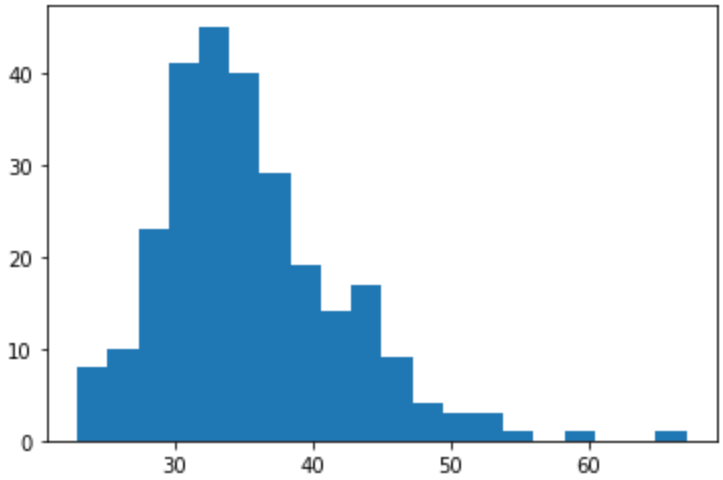
Project Task: Week 2

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 [81]:
```
plt.hist(Positive['BMI'],histtype='stepfilled',bins=20)
```

Out[81]:
```
(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 ]),
 [<matplotlib.patches.Polygon at 0x19014037888>])
```

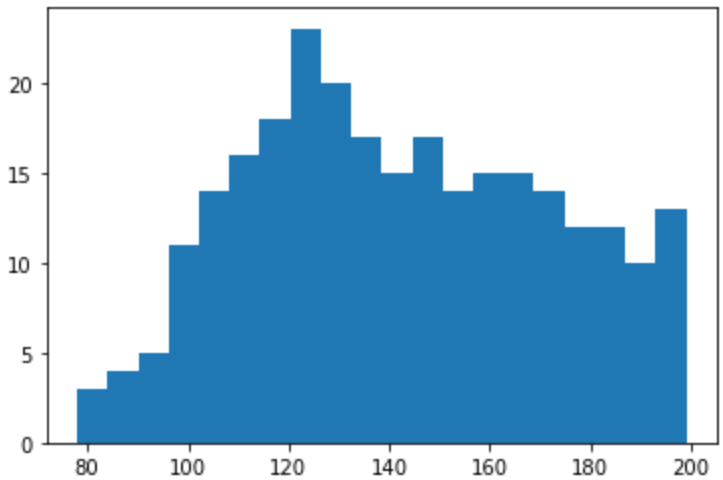


In [82]:
```
Positive['BMI'].value_counts().head(7)
```

Out[82]:
```
32.9    8
31.6    7
33.3    6
32.0    5
30.5    5
31.2    5
30.0    4
Name: BMI, dtype: int64
```

In [83]:
```
plt.hist(Positive['Glucose'],histtype='stepfilled',bins=20)
```

Out[83]:
```
(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.  ]),
 [<matplotlib.patches.Polygon at 0x190140b1fc8>])
```

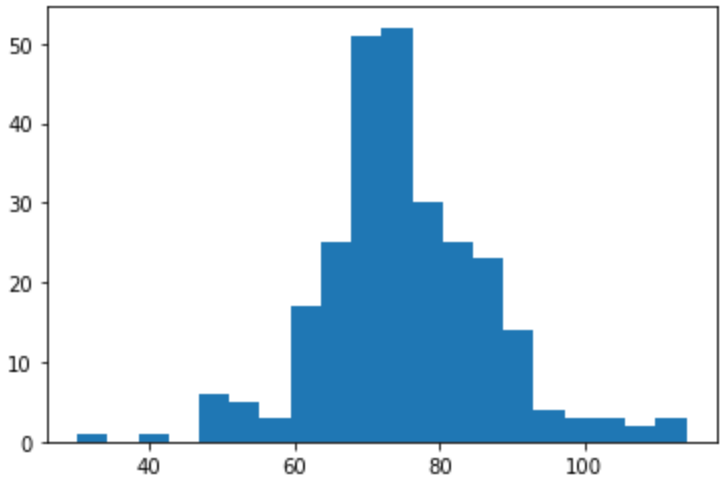


In [84]:
```
Positive['Glucose'].value_counts().head(7)
```

Out[84]:
```
125.0    7
128.0    6
129.0    6
158.0    6
115.0    6
181.0    5
173.0    5
Name: Glucose, dtype: int64
```

In [85]:
```
plt.hist(Positive['BloodPressure'],histtype='stepfilled',bins=20)
```

Out[85]:
```
(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. ]),
 [<matplotlib.patches.Polygon at 0x19014126948>])
```

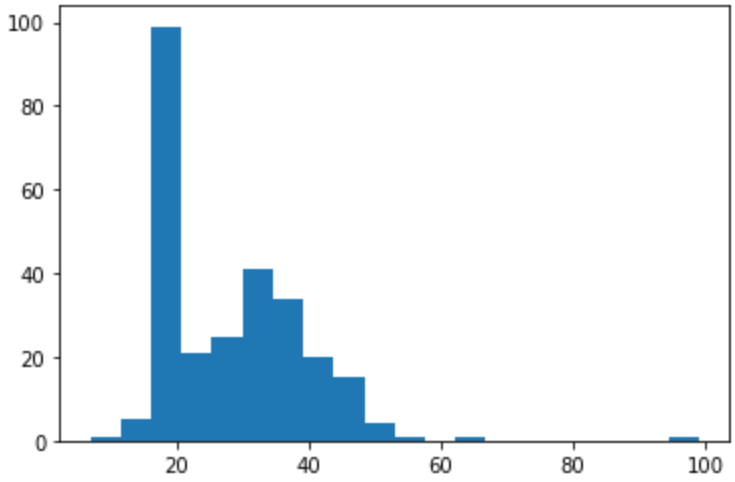


In [86]:
```
Positive['BloodPressure'].value_counts().head(7)
```

Out[86]:
```
70.000000    23
76.000000    18
78.000000    17
74.000000    17
69.105469    16
72.000000    16
64.000000    13
Name: BloodPressure, dtype: int64
```

In [87]:
```
plt.hist(Positive['SkinThickness'],histtype='stepfilled',bins=20)
```

Out[87]:
```
(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. ]),
 [<matplotlib.patches.Polygon at 0x1901417f2c8>])
```

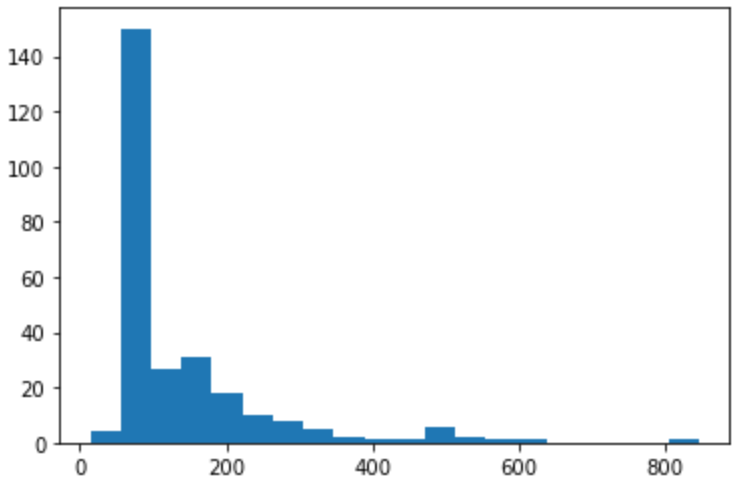


In [88]:
```
Positive['SkinThickness'].value_counts().head(7)
```

Out[88]:
```
20.536458    88
32.000000    14
30.000000     9
33.000000     9
39.000000     8
36.000000     8
37.000000     8
Name: SkinThickness, dtype: int64
```

In [89]:
```
plt.hist(Positive['Insulin'],histtype='stepfilled',bins=20)
```

Out[89]:
```
(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. ]),
 [<matplotlib.patches.Polygon at 0x190151acf88>])
```



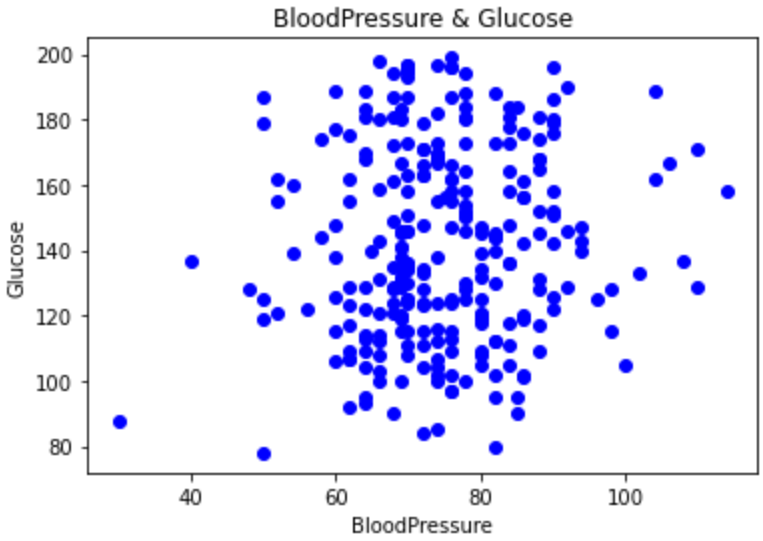
In [90]:
```
Positive['Insulin'].value_counts().head(7)
```

Out[90]:
```
79.799479     138
130.000000      6
180.000000      4
156.000000      3
175.000000      3
168.000000      2
145.000000      2
Name: Insulin, dtype: int64
```

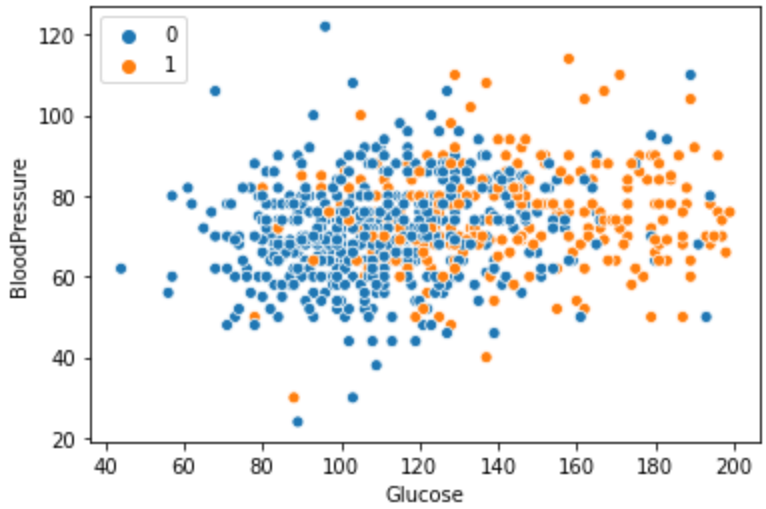
Scatter plot

In [91]:
```
BloodPressure = Positive['BloodPressure']
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
Insulin = Positive['Insulin']
BMI = Positive['BMI']
```

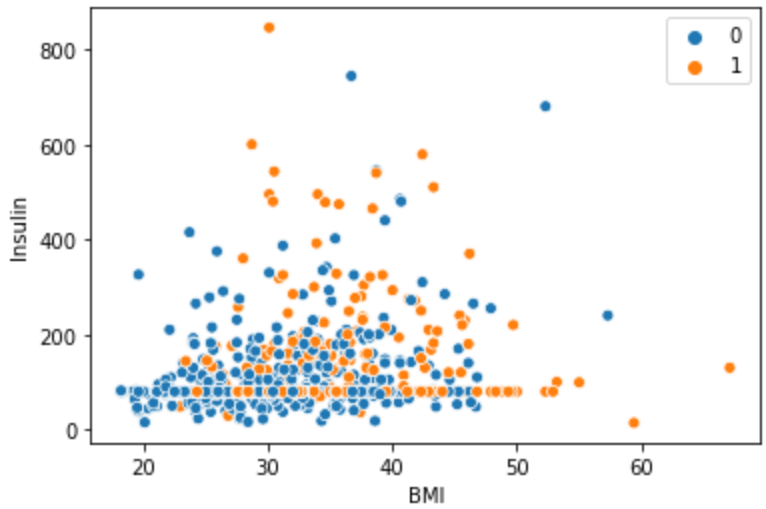
In [92]:
```
plt.scatter(BloodPressure, Glucose, color=['b'])
plt.xlabel('BloodPressure')
plt.ylabel('Glucose')
plt.title('BloodPressure & Glucose')
plt.show()
```



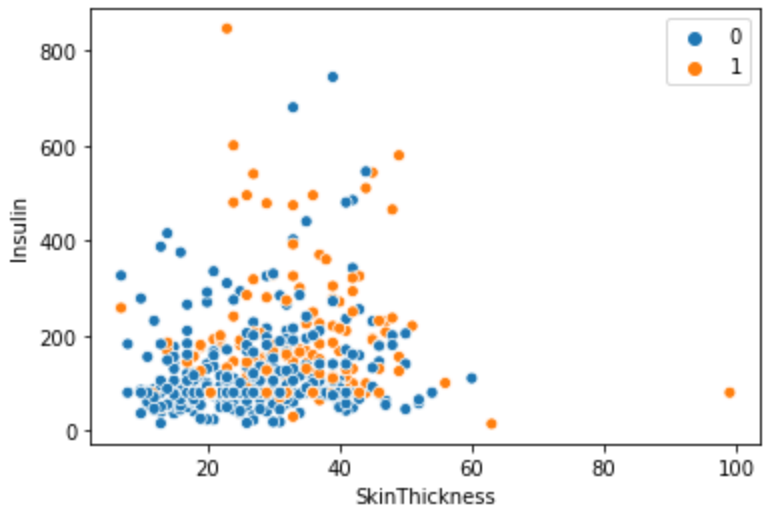
In [93]:
```
g =sns.scatterplot(x= "Glucose" ,y= "BloodPressure",
                   hue= data.Outcome.tolist(),
                   data=data);
```



In [94]:
```
B =sns.scatterplot(x= "BMI" ,y= "Insulin",
                   hue= data.Outcome.tolist(),
                   data=data);
```



In [95]:
```
S =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",
                   hue= data.Outcome.tolist(),
                   data=data);
```



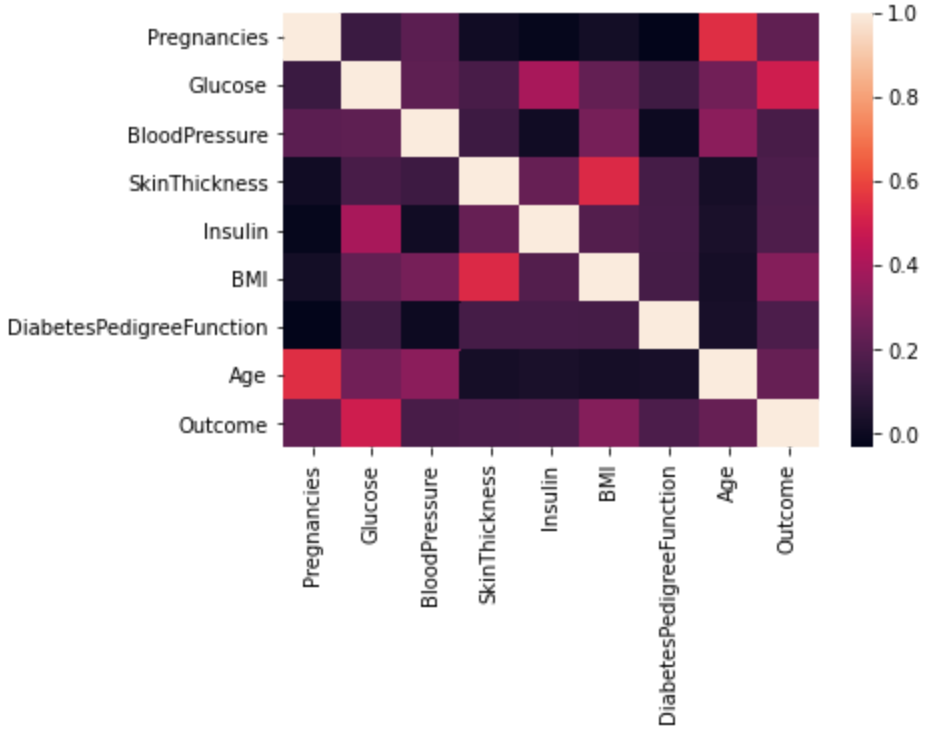
In [96]:
```
### correlation matrix
data.corr()
```

Out[96]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
Pregnancies	1.000000	0.127964	0.208984	0.013376	-0.018082	0.021546	
Glucose	0.127964	1.000000	0.219666	0.160766	0.396597	0.231478	
BloodPressure	0.208984	0.219666	1.000000	0.134155	0.010926	0.281231	
SkinThickness	0.013376	0.160766	0.134155	1.000000	0.240361	0.535703	
Insulin	-0.018082	0.396597	0.010926	0.240361	1.000000	0.189856	
BMI	0.021546	0.231478	0.281231	0.535703	0.189856	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.154961	0.157806	0.153508	
Age	0.544341	0.266600	0.326740	0.026423	0.038652	0.025748	
Outcome	0.221898	0.492908	0.162986	0.175026	0.179185	0.312254	

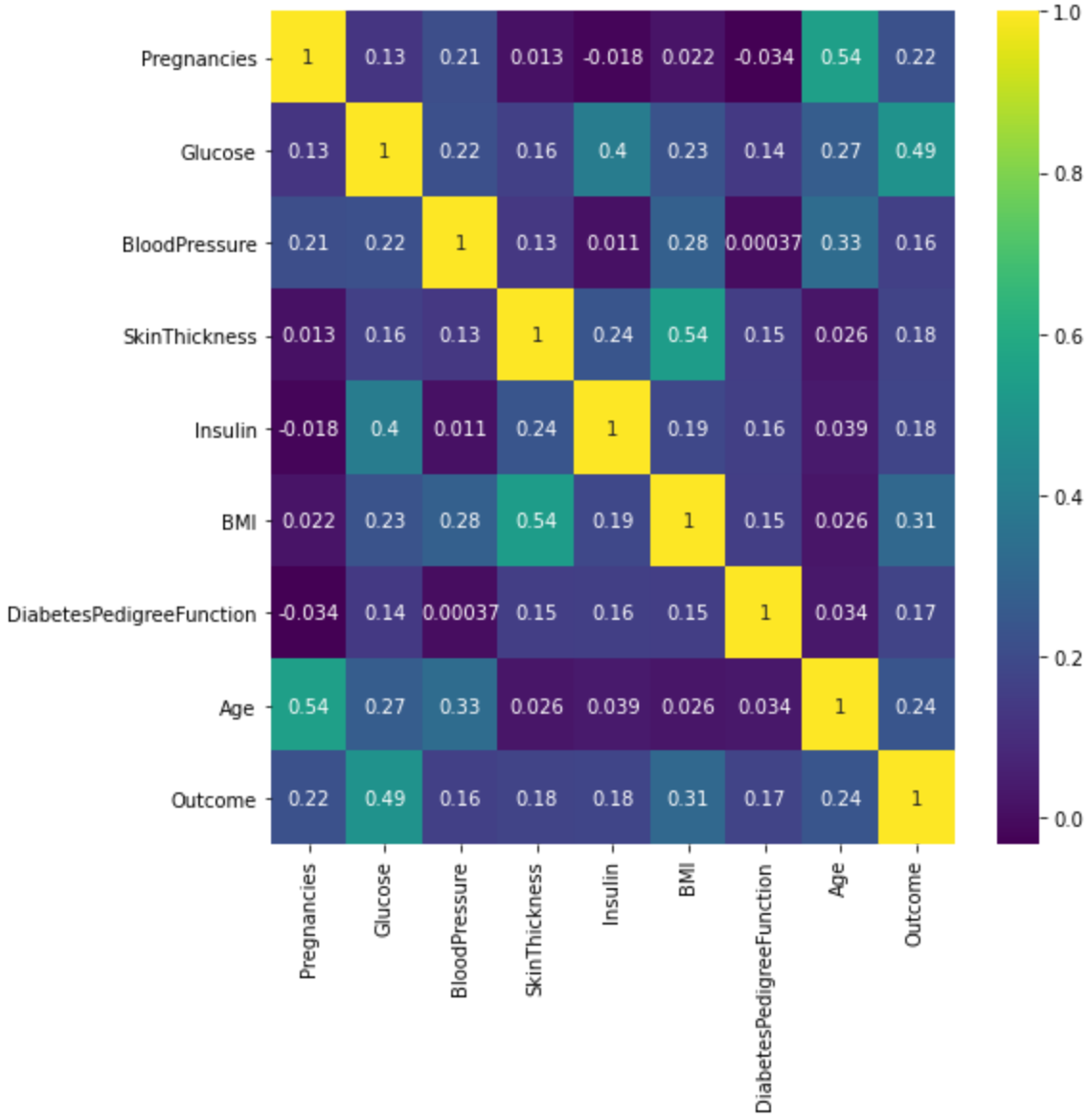
In [97]:
```
### create correlation heat map
sns.heatmap(data.corr())
```

Out[97]: <AxesSubplot:>



In [98]:
```
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True,cmap='viridis')  ### gives correlation value
```

Out[98]: <AxesSubplot:>



Thank You

In [ ]:
```
```

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 [99]:
```
# Logistic Regreation and model building
```

In [100]:
```
data.head(5)
```

Out[100]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	C
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	

In [101]:
```
features = data.iloc[:,[0,1,2,3,4,5,6,7]].values
label = data.iloc[:,8].values
```

In [102]:
```
#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 [103]:
```
#Create model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\anike\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:764: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Out[103]:
```
LogisticRegression()
```

In [104]:
```
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.7703583061889251
0.7337662337662337
```

In [105]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(label,model.predict(features))
cm
```

Out[105]:
```
array([[441,  59],
       [123, 145]], dtype=int64)
```

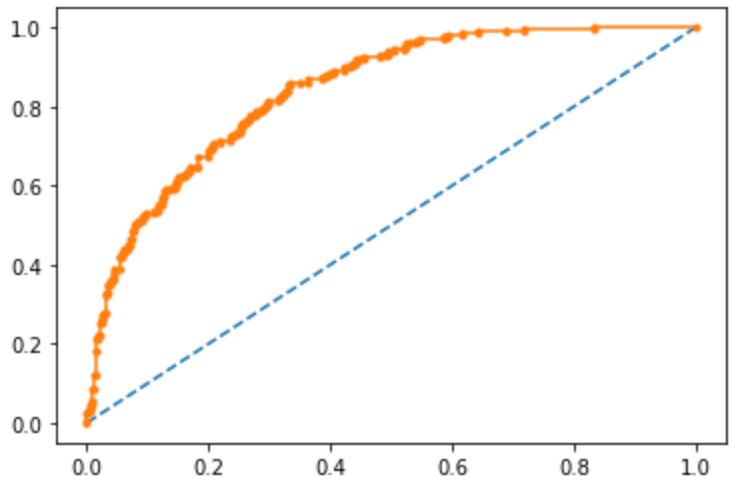
In [106]:
```
from sklearn.metrics import classification_report
print(classification_report(label,model.predict(features)))
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       500
           1       0.71      0.54      0.61       268

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

In [107]:
```
#Preparing ROC Curve (Receiver Operating Characteristics Curve)
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: 0.842
```

Out[107]:
```
[<matplotlib.lines.Line2D at 0x19015758b48>]
```



In [108]:
```
#Applying Decission Tree Classifier
from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier(max_depth=5)
model3.fit(X_train,y_train)
```

Out[108]:
```
DecisionTreeClassifier(max_depth=5)
```

In [109]:
```
model3.score(X_train,y_train)
```

Out[109]:
```
0.8273615635179153
```

In [110]:
```
model3.score(X_test,y_test)
```

Out[110]:
```
0.7467532467532467
```

In [111]:
```
#Applying Random Forest
from sklearn.ensemble import RandomForestClassifier
model4 = RandomForestClassifier(n_estimators=11)
model4.fit(X_train,y_train)
```

Out[111]:
```
RandomForestClassifier(n_estimators=11)
```

In [112]:
```
model3.score(X_train,y_train)
```

Out[112]:
```
0.8273615635179153
```

In [113]:
```
model4.score(X_test,y_test)
```

Out[113]:
```
0.6948051948051948
```

In [114]:
```
#Support Vector Classifier

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

Out[114]:
```
SVC(gamma='auto')
```

In [115]:
```
model5.score(X_train,y_train)
```

Out[115]:
```
1.0
```

In [116]:
```
model5.score(X_test,y_test)
```

Out[116]:
```
0.6168831168831169
```

In [117]:
```
#Applying K-NN
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=7,
                              metric='minkowski',
                              p = 2)
model2.fit(X_train,y_train)
```

Out[117]:
```
KNeighborsClassifier(n_neighbors=7)
```

In [118]:
```
model2.score(X_train, y_train)
```

Out[118]:
```
0.8078175895765473
```

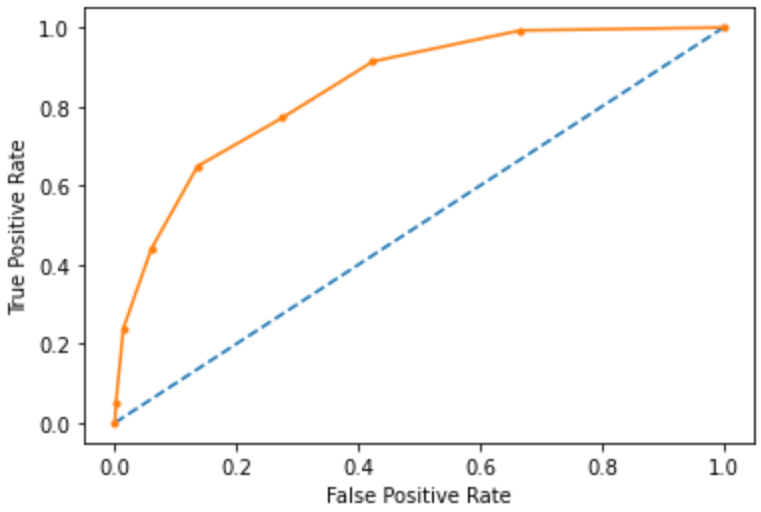
In [119]:
```
model2.score(X_test, y_test)
```

Out[119]:
```
0.7142857142857143
```

In [120]:
```
#Preparing ROC Curve (Receiver Operating Characteristics Curve)
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict probabilities
probs = model2.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)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr,fpr
,thresholds))
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.847
True Positive Rate - [0.         0.05223881 0.23880597 0.44029851 0.64925373 0.77238806
 0.9141791  0.99253731 1.        ], False Positive Rate - [0.    0.002 0.014 0.06  0.13
6 0.276 0.424 0.666 1.   ] Thresholds - [2.         1.         0.85714286 0.71428571 0.
57142857 0.42857143
 0.28571429 0.14285714 0.        ]
```

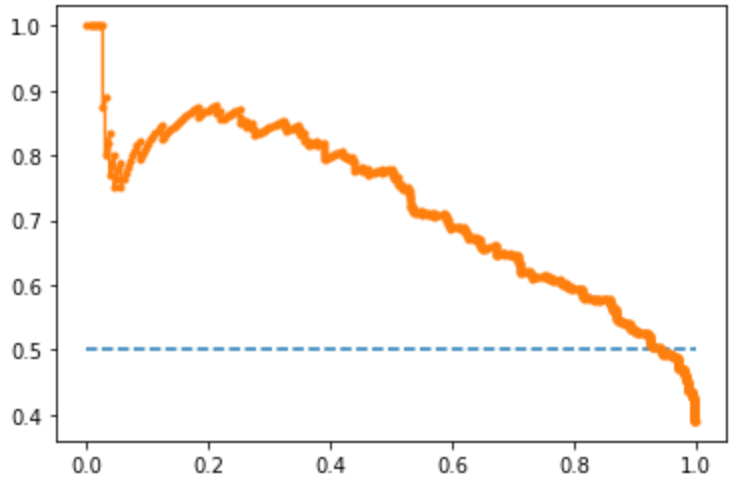
Out[120]:
```
Text(0, 0.5, 'True Positive Rate')
```



In [121]:
```
#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=0.614 auc=0.723 ap=0.724
```

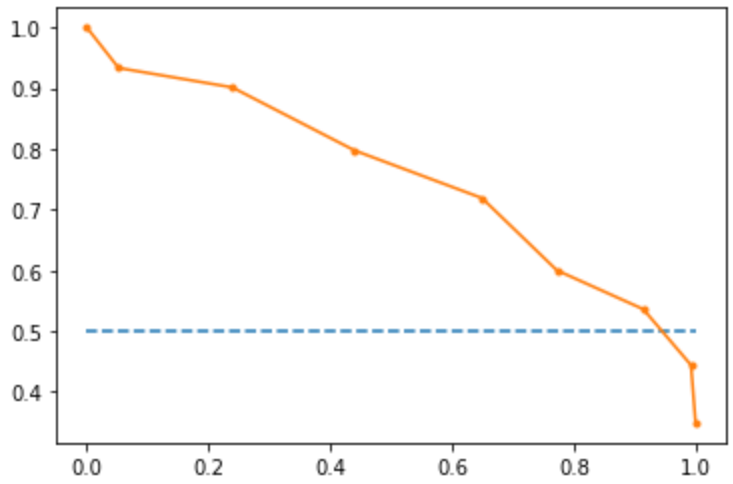
Out[121]:
```
[<matplotlib.lines.Line2D at 0x190155b05c8>]
```



In [122]:
```
#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
```

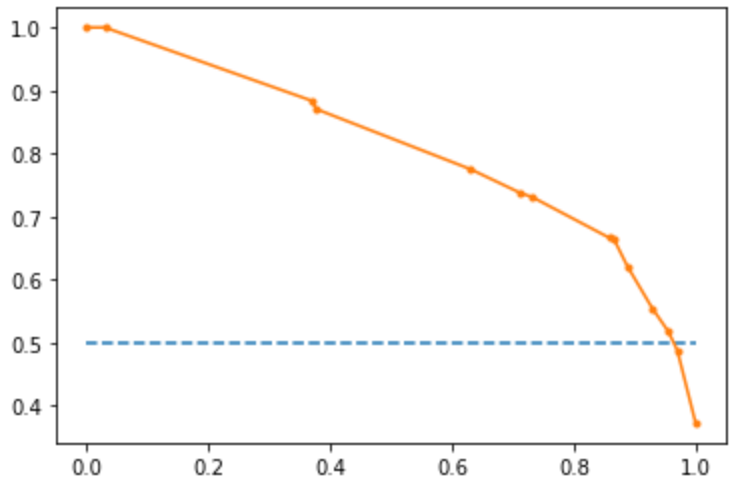
Out[122]:
```
[<matplotlib.lines.Line2D at 0x19015460f48>]
```



In [123]:
```
#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=0.725 auc=0.807 ap=0.766
```

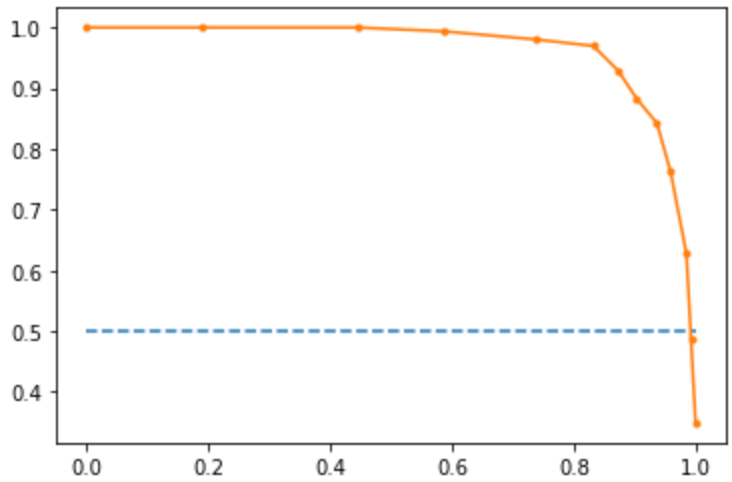
Out[123]:
```
[<matplotlib.lines.Line2D at 0x19011f2bdc8>]
```



In [124]:
```
#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=0.900 auc=0.966 ap=0.958
```

Out[124]:
```
[<matplotlib.lines.Line2D at 0x19011daa288>]
```



In [ ]: