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

%matplotlib inline


data = pd.read_csv('/content/health care diabetes.csv')


data.head(5)   #Printing First Five Rows
```

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

```
data.isnull().any() #Finding Null Values
```

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```



```
data.info()   #Information of Dataset
```

```
<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
```

```
Positive = data[data['Outcome']==1]    #selecting only positive Outcome Only
Positive.head(5)
```

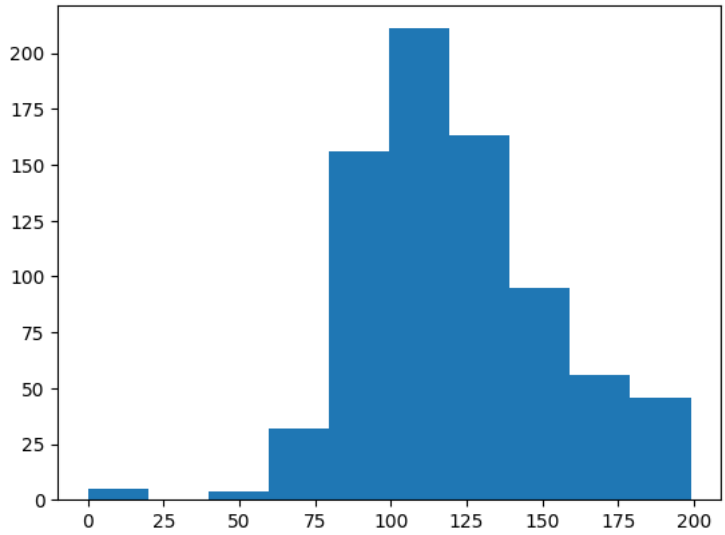
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
0	6	148	72	35	0	33.6	
2	8	183	64	0	0	23.3	
4	0	137	40	35	168	43.1	
6	3	78	50	32	88	31.0	
8	2	197	70	45	543	30.5	

```
data['Glucose'].value_counts().head()   #Getting count for Glucose Only
```

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

```
plt.hist(data['Glucose'])   # Histogram for Glucose
```

```
(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. ]),
 <BarContainer object of 10 artists>)
```



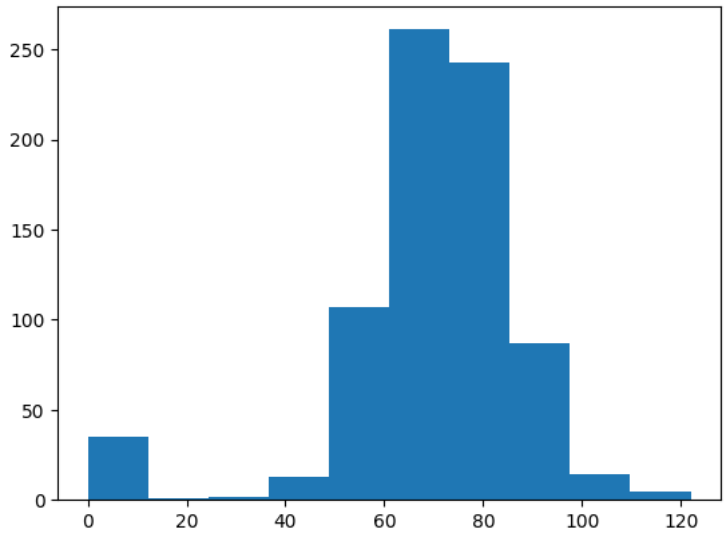

```
data['BloodPressure'].value_counts().head()  #Getting count for BloodPressure Only
```

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



```
plt.hist(data['BloodPressure'])         # Histogram for BloodPressure
```

```
(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. ]),
 <BarContainer object of 10 artists>)
```

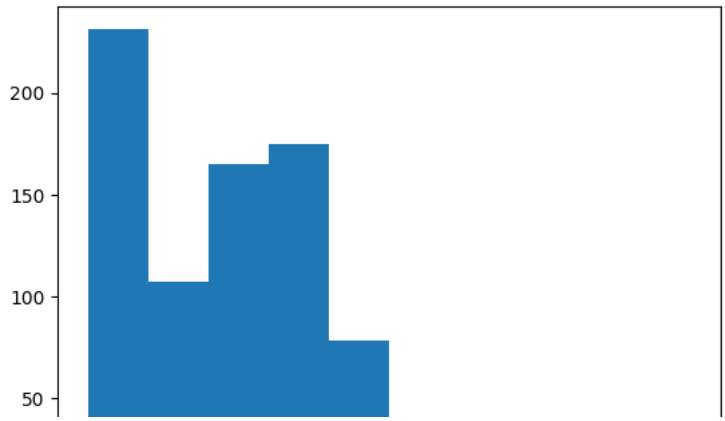



```
data['SkinThickness'].value_counts().head()    #Getting count for SkinThickness Only
```

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

```
plt.hist(data['SkinThickness'])        # Histogram for SkinThickness
```

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



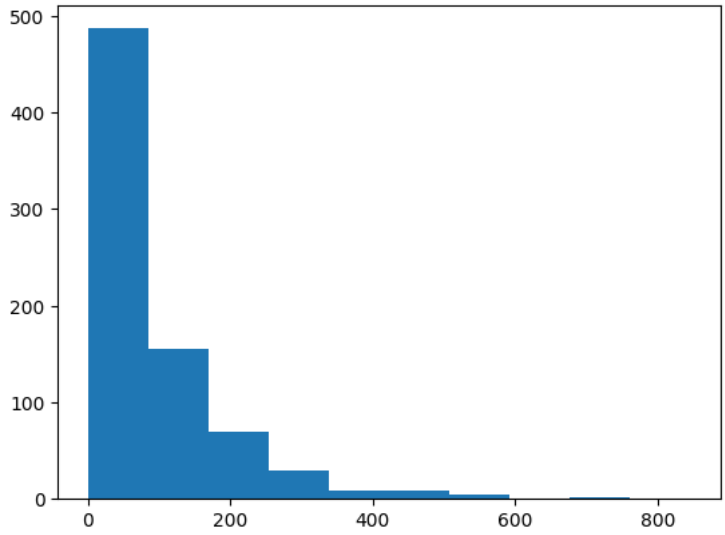

```
data['Insulin'].value_counts().head()     #Getting count for Insulin Only
```

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

```
plt.hist(data['Insulin'])         # Histogram for Insulin
```



```
2.,   1.]),
92.2, 676.8,
       761.4, 846. ]),
<BarContainer object of 10 artists>)
```

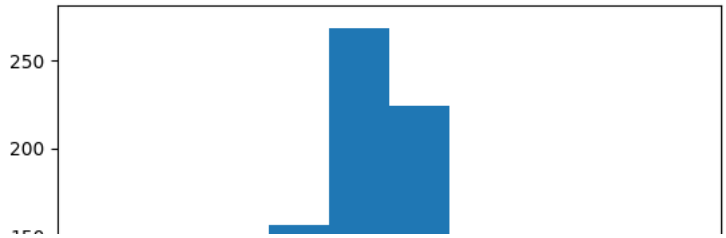



```
data['BMI'].value_counts().head()     #Getting count for BMI Only
```

```
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
Name: BMI, dtype: int64
```

```
plt.hist(data['BMI'])       # Histogram for BMI
```

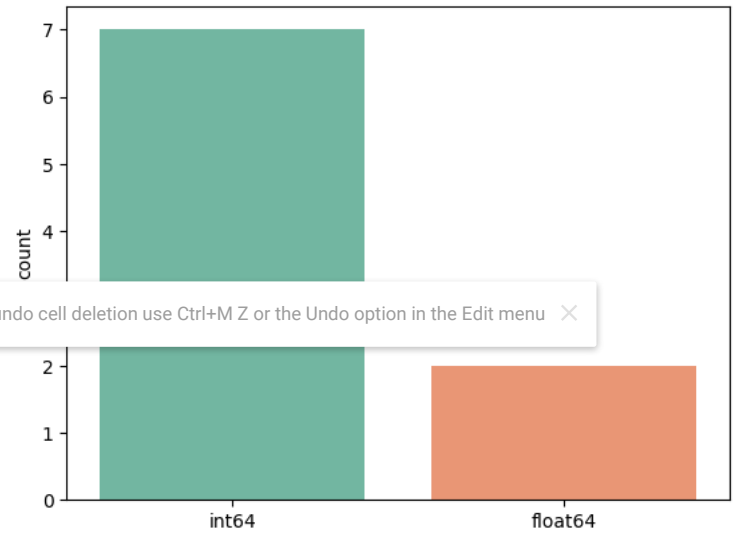
```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <BarContainer object of 10 artists>)
```




```
values = data.dtypes     #Assigning datatype tp values
```

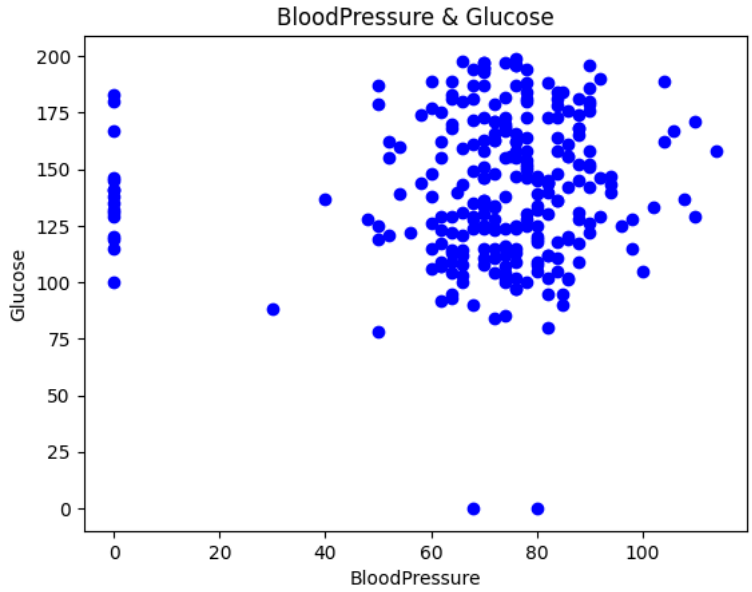
```
sns.countplot(x = values , data = data, palette = "Set2")      #a count  plot describing the data types and the count of variables.
```

```
<Axes: ylabel='count'>
```



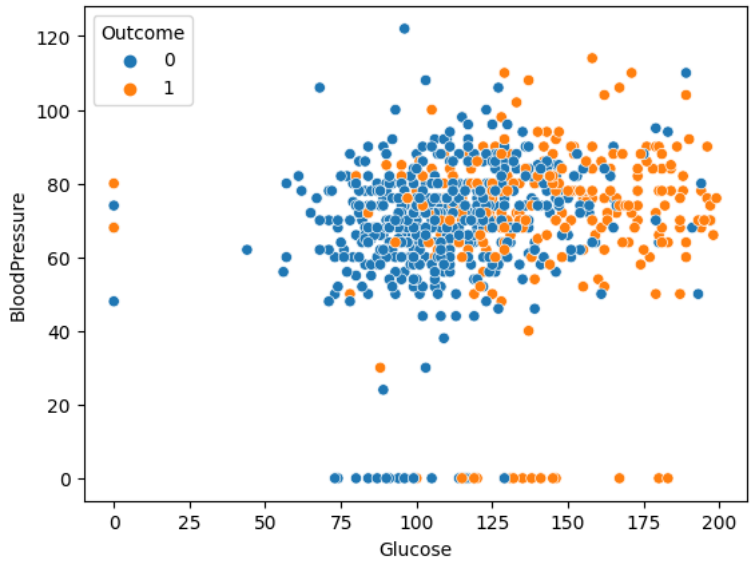

```
BloodPressure = Positive['BloodPressure']    #Selecting all Positive
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
Insulin = Positive['Insulin']
BMI = Positive['BMI']
```

```
plt.scatter(BloodPressure, Glucose, color=['b'])    #Scatter PLot for BloodPressure and glucose for Positive
plt.xlabel('BloodPressure')
plt.ylabel('Glucose')
plt.title('BloodPressure & Glucose')
plt.show()
```




```
======================================================================
```

```
g =sns.scatterplot(x= "Glucose" ,y= "BloodPressure",hue="Outcome",data=data);        #Scatter PLot for BloodPressure and glucose
```




```
========================================================================
```

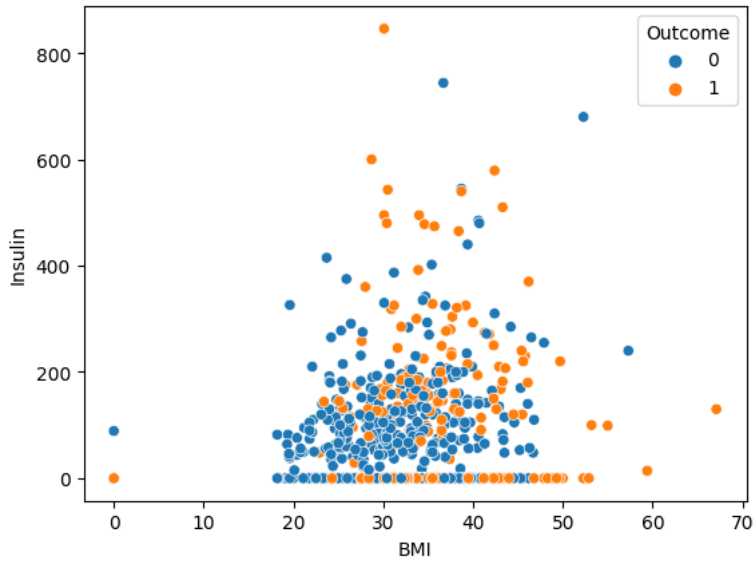
```
========================================================================
```



```
========================================================================
```

```
========================================================================
```

```
g =sns.scatterplot(x= "BMI" ,y= "Insulin",hue="Outcome",data=data);     #Scatter PLot for Insulin and BMI
```

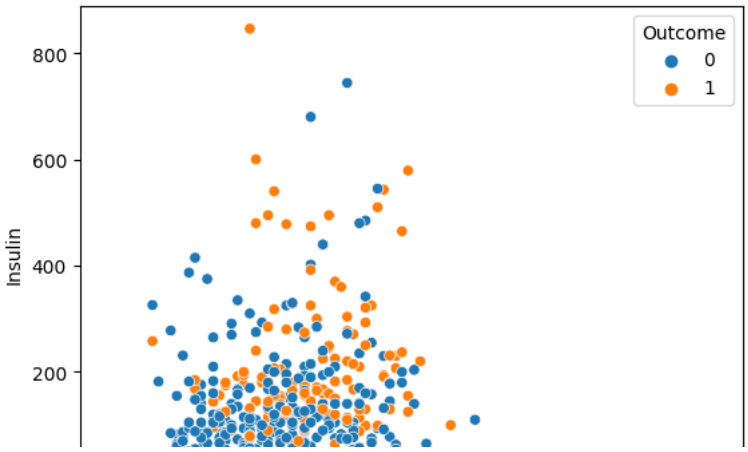



```
========================================================================
```

```
========================================================================
```

```
========================================================================
```

```
g =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",hue="Outcome",data=data);      #Scatter PLot for SkinThickness and Insulin
```



==========

```
data.corr()     #Finding Correlation
```

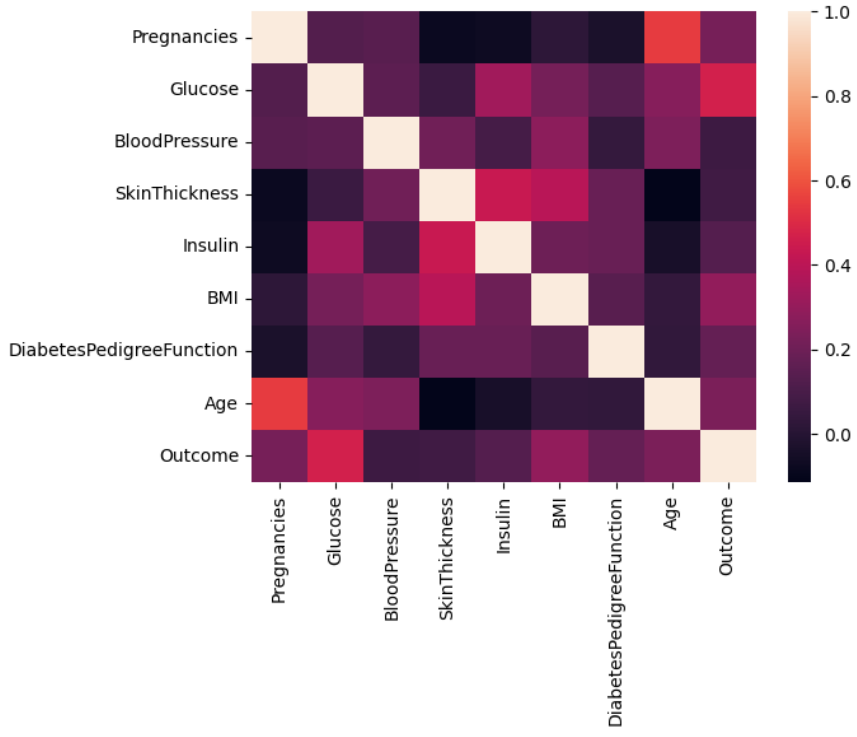
	Pregnancies	Glucose	BloodPressure	SkinThickness	Ins
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.07
Glucose	0.129459	1.000000	0.152590	0.057328	0.33
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.08
SkinThickness	0.081672	0.057328	0.207371	1.000000	0.43
Insulin	-0.073535	0.331357	0.088933	0.436783	1.00
BMI	0.017683	0.221071	0.281805	0.392573	0.19
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.18
Age	0.544341	0.263514	0.239528	-0.113970	-0.04
Outcome	0.221898	0.466581	0.065068	0.074752	0.13

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

==========

```
sns.heatmap(data.corr())      #HeatMap for Correlation
```

```
<Axes: >
```

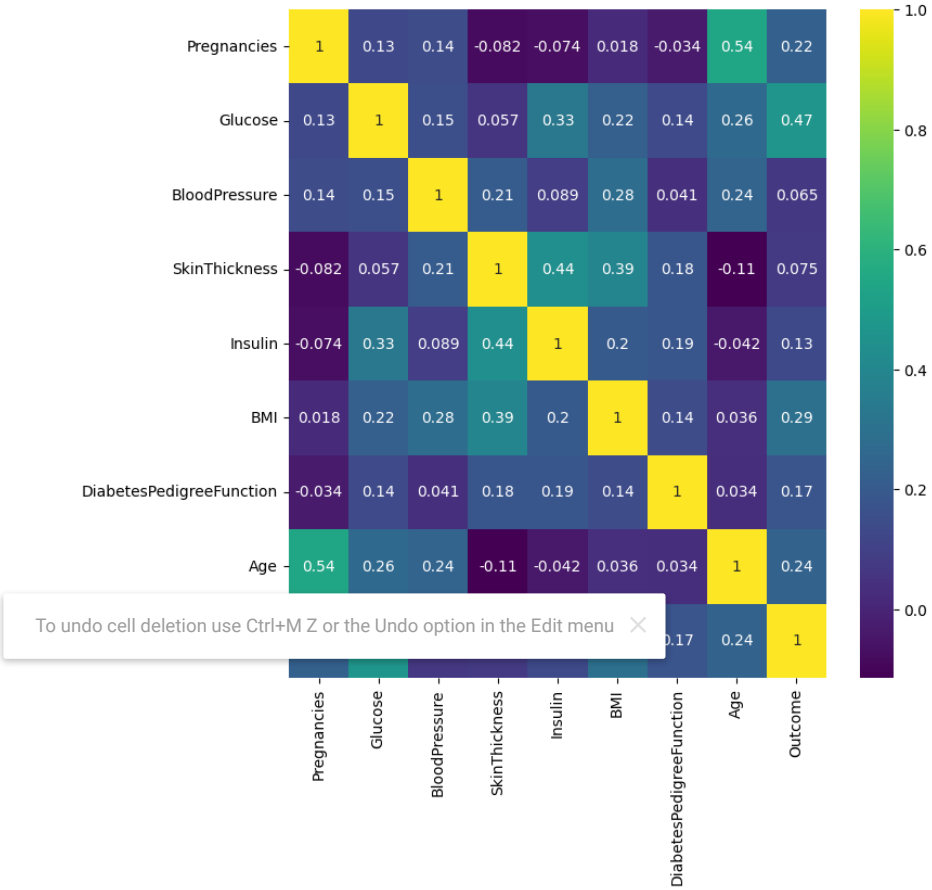


==========

```
========================================================================
```

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

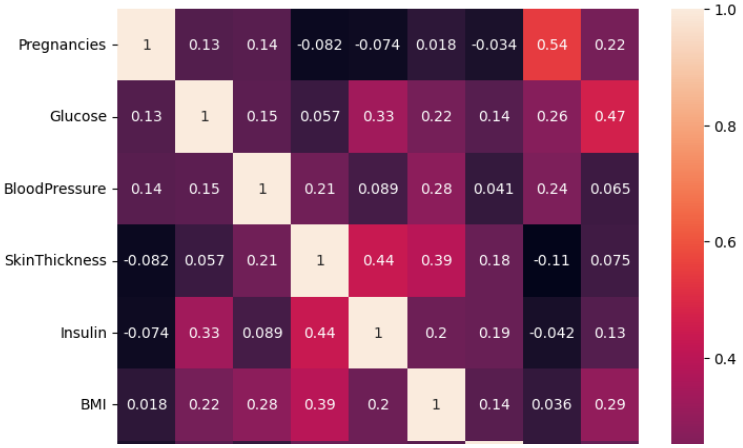
```
<Axes: >
```



To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
plt.subplots(figsize=(8,8))
sns.heatmap(data.corr(),annot=True)     #HeatMaps With Values
```

```
<Axes: >
```




```
data.head(5)
```

	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

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

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


from sklearn.model_selection import train_test_split     #Splitting data into Training and testing
X_train , X_test , y_train , y_test =  train_test_split(features, label,test_size = 0.2 , random_state = 42)


from sklearn.linear_model import LogisticRegression    #Fitting LogisticRegression on Training Dataset
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status
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
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

```
print('The values for training data set is : ',model.score(X_train,y_train))
print('The values for testing data set is : ',model.score(X_test, y_test))
```

```
The values for training data set is :  0.7719869706840391
The values for testing data set is :  0.7467532467532467
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(label,model.predict(features))     #Using features for Confusion Matrix for prediction
f'The Confusion Matrix is : {cm}'
```

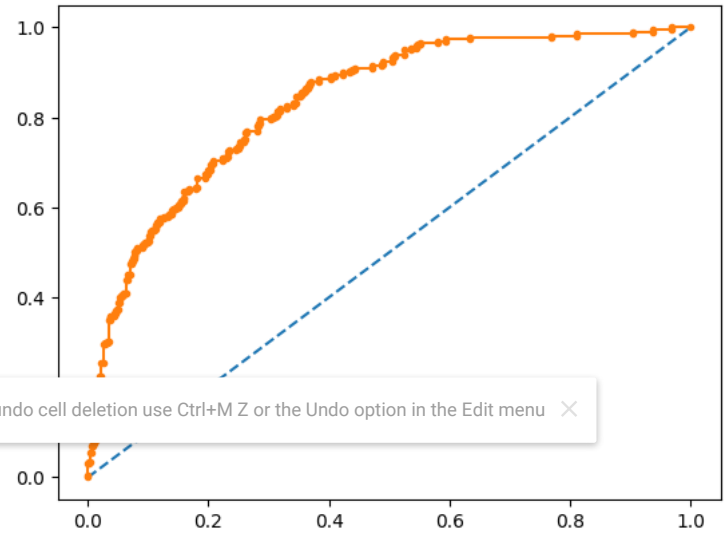
```
'The Confusion Matrix is : [[432  68]\n [111 157]]'
```

```
from sklearn.metrics import classification_report
print(classification_report(label,model.predict(features)))     #Using features for Classification Report  for prediction
```

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       500
           1       0.70      0.59      0.64       268

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

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

probs = model.predict_proba(features)     #predicting probabilities
probs = probs[:, 1]   #for positive Outcome

auc = roc_auc_score(label, probs) #calculating AUC
print('AUC: %.3f' % auc)

fpr, tpr, thresholds = roc_curve(label, probs)  #calculating ROC

plt.plot([0, 1], [0, 1], linestyle='--')   #[0,1] Plot on graph using --
plt.plot(fpr, tpr, marker='.')   # ROC Curve
```

```
AUC: 0.835
[<matplotlib.lines.Line2D at 0x79ecd39a50c0>]
```




```
from sklearn.neighbors import KNeighborsClassifier   #Using KNeighbour Classifier
model2 = KNeighborsClassifier(n_neighbors=7,metric='minkowski',p = 2)
Ktrain = model2.fit(X_train,y_train)
Ktest = model2.fit(X_test , y_test)
f'Value using KNeighbour Classifier Classifer For Training is : {Ktrain} and For Testing is : {Ktest}'
```

```
'Value using KNeighbour Classifier Classifer For Training is : KNeighborsClassifier(n_neighbors=7) and For Testing is : KNeighbors
Classifier(n_neighbors=7)'
```

```
from sklearn.tree import DecisionTreeClassifier   #Using DecisionTree Classifier
model3 = DecisionTreeClassifier(max_depth=5)
model3.fit(X_train,y_train)
Dtrain = model3.score(X_train,y_train)
Dtest = model3.score(X_test,y_test)
f'Value using DecisionTree Classifer For Training is : {Dtrain} and For Testing is : {Dtest}'
```

```
'Value using DecisionTree Classifer For Training is : 0.8420195439739414 and For Testing is : 0.7922077922077922'
```

```
from sklearn.ensemble import RandomForestClassifier   #Using RandomForest Classifier
model4 = RandomForestClassifier(n_estimators=11)
model4.fit(X_train,y_train)
Rtrain = model4.score(X_train,y_train)
Rtest = model4.score(X_test,y_test)
f'Value using RandomForest Classifer For Training is : {Rtrain} and For Testing is : {Rtest}'
```

```
'Value using RandomForest Classifer For Training is : 0.993485342019544 and For Testing is : 0.7337662337662337'
```

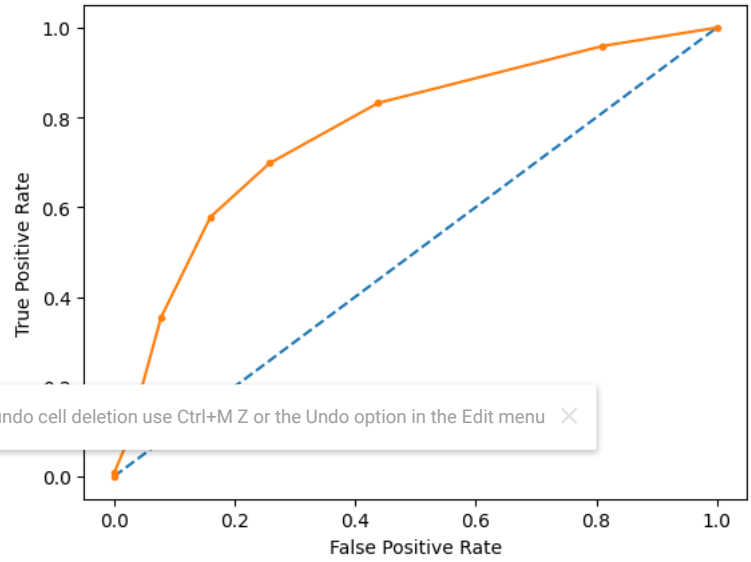
```
from sklearn.svm import SVC    #Using Support Vector Classifier
model5 = SVC(kernel='rbf',gamma='auto')
model5.fit(X_train,y_train)
Strain = model5.score(X_train,y_train)
Stest = model5.score(X_test,y_test)
f'Value using Support Vector Classifer For Training is : {Strain} and For Testing is : {Stest}'
```

```
'Value using Support Vector Classifer For Training is : 1.0 and For Testing is : 0.6428571428571429'
```

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

probs = model2.predict_proba(features)
probs = probs[:, 1]
```

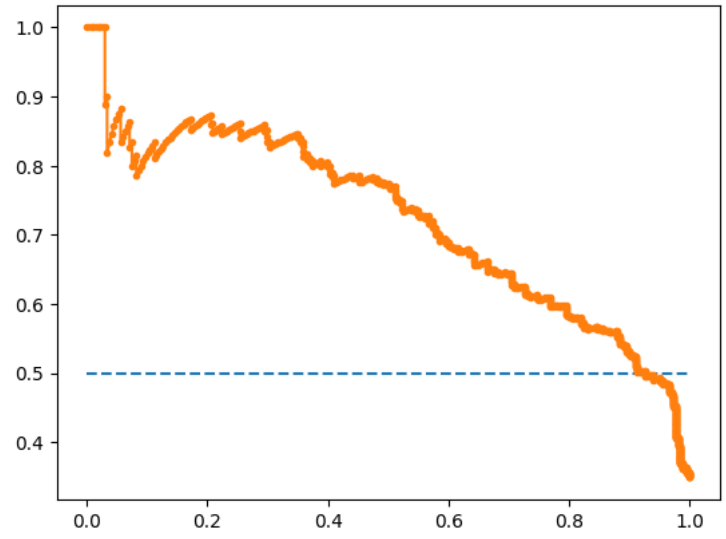
```
auc = roc_auc_score(label, probs)  # calculate AUC
print('AUC: %.3f' % auc)

fpr, tpr, thresholds = roc_curve(label, probs)# calculate roc curve
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr,fpr,thresholds))

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

```
AUC: 0.771
True Positive Rate - [0.         0.00746269 0.07089552 0.35447761 0.57835821 0.697
 0.83208955 0.95895522 1.        ], False Positive Rate - [0.    0.    0.02  0.078
 0.28571429 0.14285714 0.        ]
Text(0, 0.5, 'True Positive Rate')
```




```
from sklearn.metrics import precision_recall_curve , f1_score , auc , average_precision_score  #For Logistic Regression

probs = model.predict_proba(features)
probs = probs[:, 1]
yhat = model.predict(features)
precision, recall, thresholds = precision_recall_curve(label, probs)
f1 = f1_score(label, yhat)
auc = auc(recall, precision)
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

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

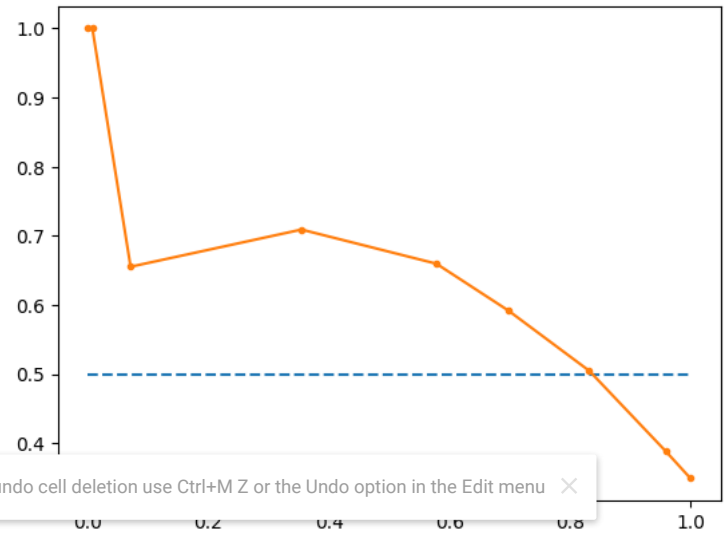
```
f1=0.637 auc=0.722 ap=0.723
[<matplotlib.lines.Line2D at 0x79ecd1c4dcf0>]
```




```
from sklearn.metrics import precision_recall_curve , f1_score , auc , average_precision_score
probs = model2.predict_proba(features)
```

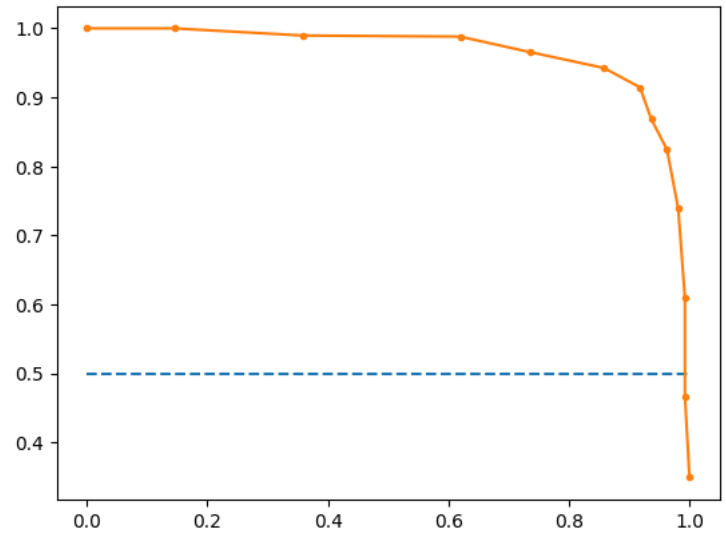
```
probs = probs[:,1]
yhat = model2.predict(features)
precision, recall , thresholds = precision_recall_curve(label,probs)
f1 = f1_score(label,yhat)
auc = auc(recall, precision)
ap = average_precision_score(label,probs)
print('f1 = {} , AUC = {} , Average Precision = {}'.format(f1,auc,ap))

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

```
f1 = 0.6163021868787275 , AUC = 0.6266704749836112 , Average Precision = 0.5997408147758778
[<matplotlib.lines.Line2D at 0x79ecceb8a620>]
```




```
from sklearn.metrics import precision_recall_curve , f1_score , auc , average_precision_score  #For Logistic Regression
# Random Forest
probs = model4.predict_proba(features)
probs = probs[:, 1]
yhat = model4.predict(features)
precision, recall, thresholds = precision_recall_curve(label, probs)
f1 = f1_score(label, yhat)
auc = auc(recall, precision)
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

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

```
f1=0.916 auc=0.965 ap=0.957
[<matplotlib.lines.Line2D at 0x79ecceb30340>]
```



✓ 0s completed at 11:58 AM

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu