Importing Libraries

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

Loading Diabetes Dataset

In [2]:
```
df=pd.read_csv('diabetes.csv')
df
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

768 rows × 9 columns

Reading first five rows of the dataset

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

Out[3]:

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

In [4]:
```
df.head(2)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351

Reading last five rows of the dataset

In [5]:
```
df.tail()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

In [6]:
```
df.tail(3)
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

Display all column names of the dataset

In [9]:
```
df.columns
```

Out[9]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Reading datatypes of each column in dataset

In [10]:
```
df.dtypes
```

Out[10]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

Reading number of rows and columns of the dataset

In [11]:
```
df.shape
```

Out[11]:
```
(768, 9)
```

Apply slicing

In [12]:
```
df[10:20]
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
10	4	110	92	0	0	37.6	0.191
11	10	168	74	0	0	38.0	0.537
12	10	139	80	0	0	27.1	1.441
13	1	189	60	23	846	30.1	0.398
14	5	166	72	19	175	25.8	0.587
15	7	100	0	0	0	30.0	0.484
16	0	118	84	47	230	45.8	0.551
17	7	107	74	0	0	29.6	0.254
18	1	103	30	38	83	43.3	0.183
19	1	115	70	30	96	34.6	0.529

Displaying particular columns with specific number of rows

In [13]:
```
col=['Pregnancies','Age','Outcome']
df[col].head(6)
```

Out[13]:

	Pregnancies	Age	Outcome
0	6	50	1
1	1	31	0
2	8	32	1
3	1	21	0
4	0	33	1
5	5	30	0

Getting rows using specific condition

In [14]:
```
df.loc[df['Outcome']==1]
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
2	8	183	64	0	0	23.3	0.67
4	0	137	40	35	168	43.1	2.28
6	3	78	50	32	88	31.0	0.24
8	2	197	70	45	543	30.5	0.15
...	...	...	...	...	...	...	
755	1	128	88	39	110	36.5	1.05
757	0	123	72	0	0	36.3	0.25
759	6	190	92	0	0	35.5	0.27
761	9	170	74	31	0	44.0	0.40
766	1	126	60	0	0	30.1	0.34

268 rows × 9 columns

In [15]:
```
df.loc[df['Outcome']==1].head(3)
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
2	8	183	64	0	0	23.3	0.672
4	0	137	40	35	168	43.1	2.288

Display number of categories in specific column

In [17]:
```
df['Outcome'].value_counts()
```

Out[17]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

Counting Mean and Median of the particular column

In [18]:
```
df['Outcome'].mean()
```

Out[18]: 0.3489583333333333

In [19]:
```
df['Outcome'].median()
```

Out[19]: 0.0

Finding Minimum and maximum value of the particular column

In [20]:
```
df['Outcome'].min()
```

Out[20]: 0

In [21]:
```
df['Outcome'].max()
```

Out[21]: 1

Creating a new column

In [22]:
```
col=df.columns
df1=df[col]
df['Total']=df1[col].sum(axis=1)
df
```

Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

768 rows × 10 columns

Renaming column names

In [24]:
```
df.rename(columns={'Age':'AGE','Outcome':'OUTCOME','Total':'TOTAL'},inplace=Tr
df
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

768 rows × 10 columns

Apply style to the dataset

In [25]:
```
df.style
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	6	148	72	35	0	33.600000	
1	1	85	66	29	0	26.600000	
2	8	183	64	0	0	23.300000	
3	1	89	66	23	94	28.100000	
4	0	137	40	35	168	43.100000	
5	5	116	74	0	0	25.600000	
6	3	78	50	32	88	31.000000	
7	10	115	0	0	0	35.300000	
8	2	197	70	45	543	30.500000	
9	8	125	96	0	0	0.000000	
10	4	110	92	0	0	37.600000	

Apply colour to the maximum row of the dataset

In [28]:
```
df.head(10).style.highlight_max(color='red',axis=0)
```

Out[28]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.600000	0.62
1	1	85	66	29	0	26.600000	0.35
2	8	183	64	0	0	23.300000	0.67
3	1	89	66	23	94	28.100000	0.16
4	0	137	40	35	168	43.100000	2.28
5	5	116	74	0	0	25.600000	0.20
6	3	78	50	32	88	31.000000	0.24
7	10	115	0	0	0	35.300000	0.13
8	2	197	70	45	543	30.500000	0.15
9	8	125	96	0	0	0.000000	0.23

In [29]:
```
df.head(10).style.highlight_max(color='red',axis=1)
```

Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.600000	0.62
1	1	85	66	29	0	26.600000	0.35
2	8	183	64	0	0	23.300000	0.67
3	1	89	66	23	94	28.100000	0.16
4	0	137	40	35	168	43.100000	2.28
5	5	116	74	0	0	25.600000	0.20
6	3	78	50	32	88	31.000000	0.24
7	10	115	0	0	0	35.300000	0.13
8	2	197	70	45	543	30.500000	0.15
9	8	125	96	0	0	0.000000	0.23

In [30]:
```
df.head(10).style.highlight_max(color='red',axis=None)
```

Out[30]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.600000	0.62
1	1	85	66	29	0	26.600000	0.35
2	8	183	64	0	0	23.300000	0.67
3	1	89	66	23	94	28.100000	0.16
4	0	137	40	35	168	43.100000	2.28
5	5	116	74	0	0	25.600000	0.20
6	3	78	50	32	88	31.000000	0.24
7	10	115	0	0	0	35.300000	0.13
8	2	197	70	45	543	30.500000	0.15
9	8	125	96	0	0	0.000000	0.23

Identifying how many number of null values in each column

In [31]:
```
df.isna().sum()
```

Out[31]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
AGE                         0
OUTCOME                     0
TOTAL                       0
dtype: int64
```

In [32]:
```
df.isnull()
```

Out[32]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
0	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	
763	False	False	False	False	False	False	Fa
764	False	False	False	False	False	False	Fa
765	False	False	False	False	False	False	Fa
766	False	False	False	False	False	False	Fa
767	False	False	False	False	False	False	Fa

768 rows × 10 columns

In [33]:
```
df.isna().sum().sum()
```

Out[33]: 0

Exploratory Data Analysis

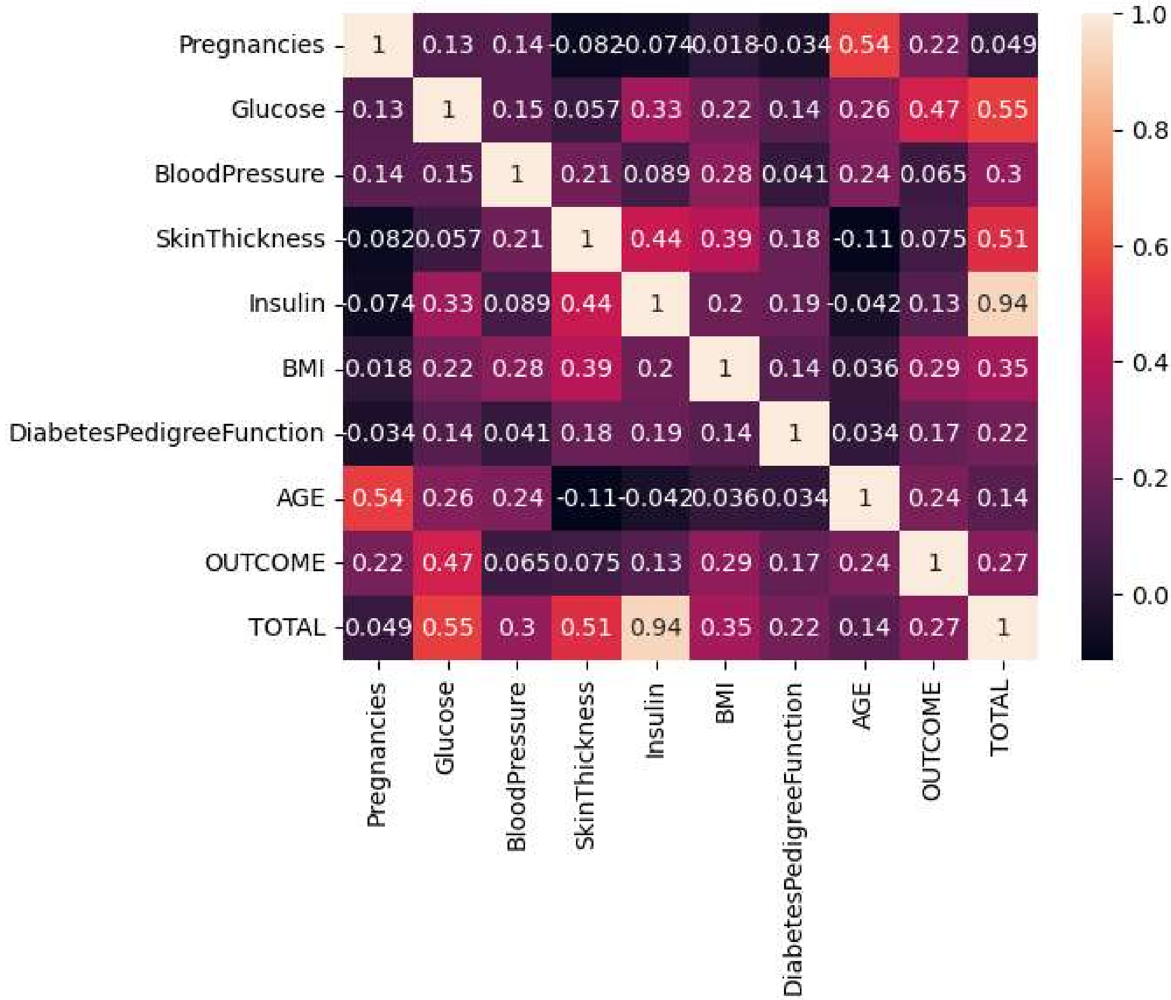
In [ ]:
```
Exploratory data analysis is one of the major step to fine-tune the given data
analysis to understand the insights of the key characteristics of the column,r
numpy and statistical methods.
```

Heat map

In [ ]:
```
A Heat map is a graphical representation of multivariate data that is structur
and columns.
   Heat map is very usefull in describing correlation among several numerical
```

In [36]:
```
sns.heatmap(df.corr(),annot=True)
```

Out[36]: <AxesSubplot:>

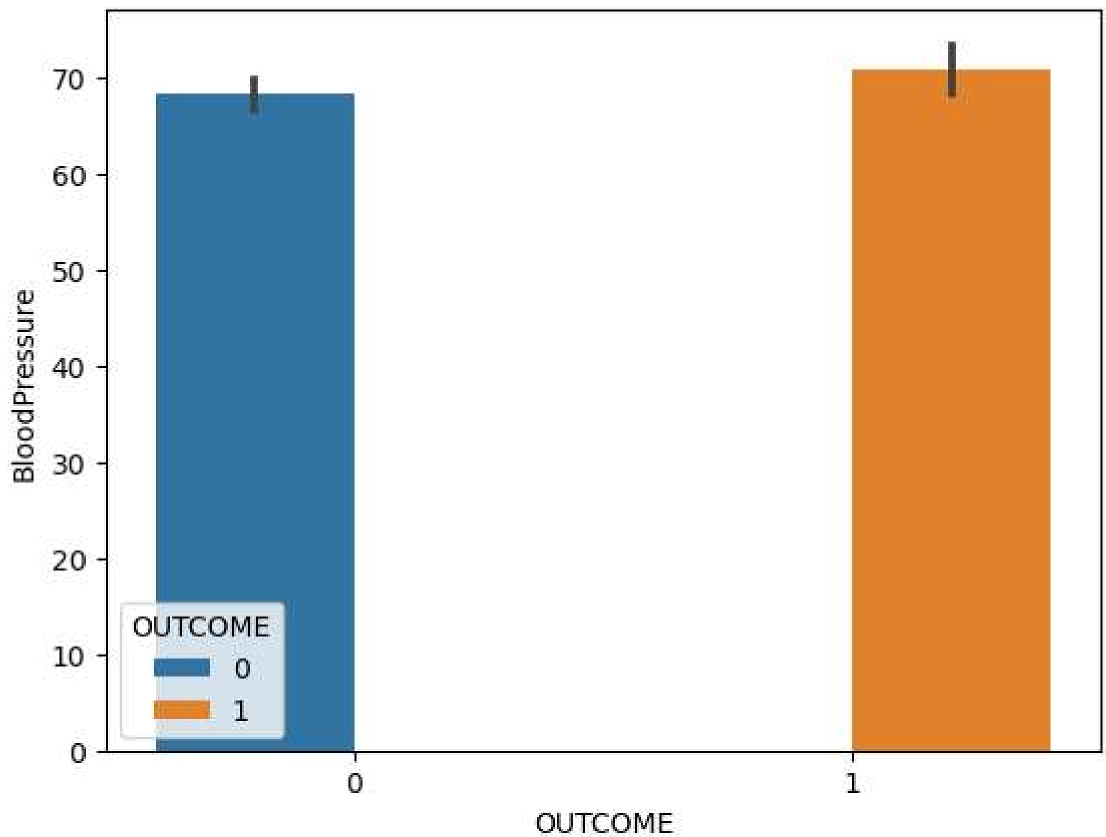


Count plot

In [ ]:
```
Count plot is used for data visualizing.It shows the observational count in di
bins with the help of bars.
```

In [45]:
```
sns.barplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME')
```

Out[45]: <AxesSubplot:xlabel='OUTCOME', ylabel='BloodPressure'>

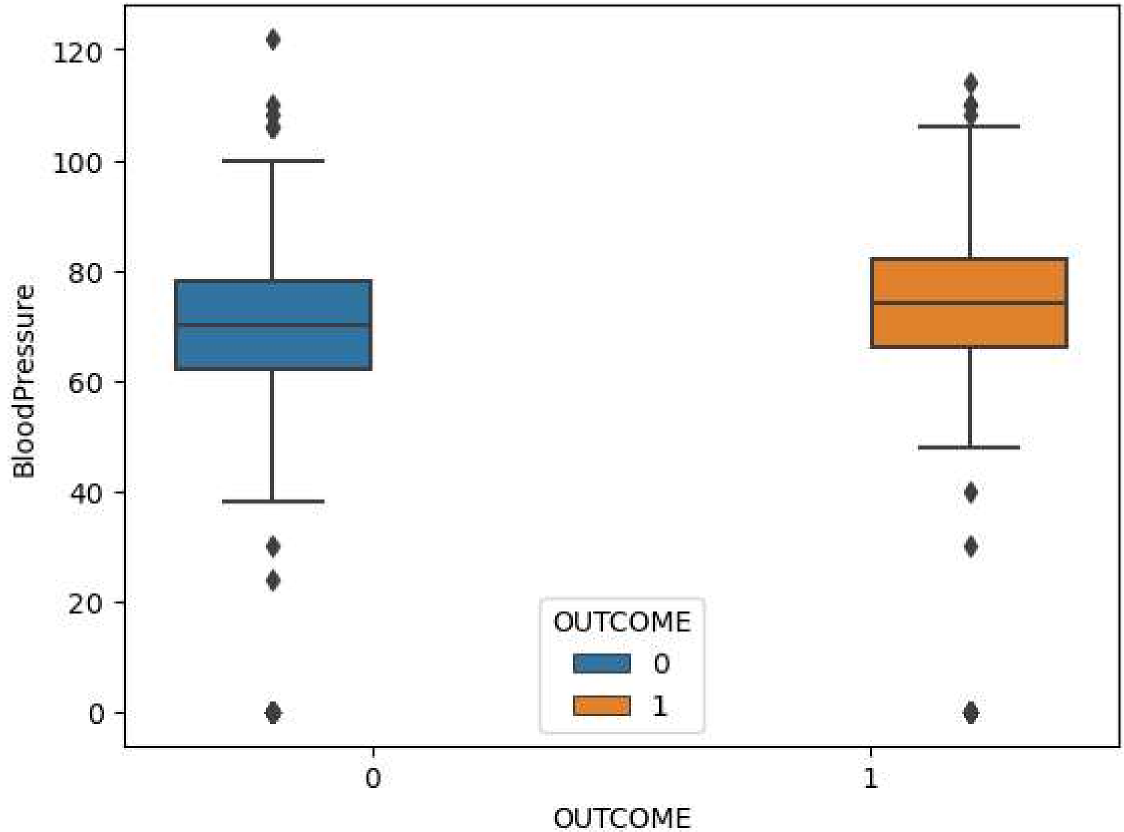


Box plot

In [ ]:
```
Box plot displays the five number summary of a set of data.The five-number summ
first quartile,median,third quartile,and maximum.
     In box plot we draw a box from first quartile to third quatile.A vertical 
box at the median.
      Box plot is mainly used for identifying the outliers.Outliers are the da
above the data limit.
```

In [46]:
```
sns.boxplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME')
```

Out[46]: <AxesSubplot:xlabel='OUTCOME', ylabel='BloodPressure'>

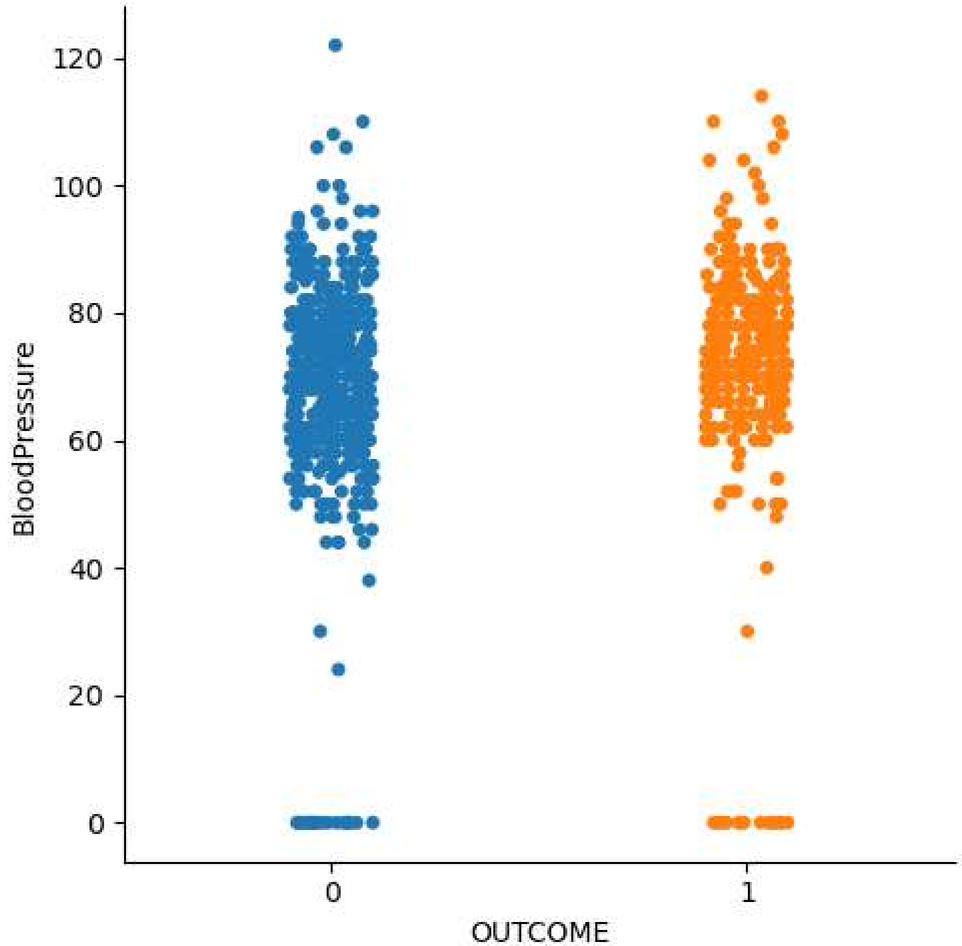


catplot

In [ ]:
```
Cat plot can handle 8 different plots currently available in seaborn.Cat plot 
these types of plots and one needs to specify the type of plot one needs with 
     Cat plot show the relationship between one or more categorical variables 
using one of the several visual representations.
```

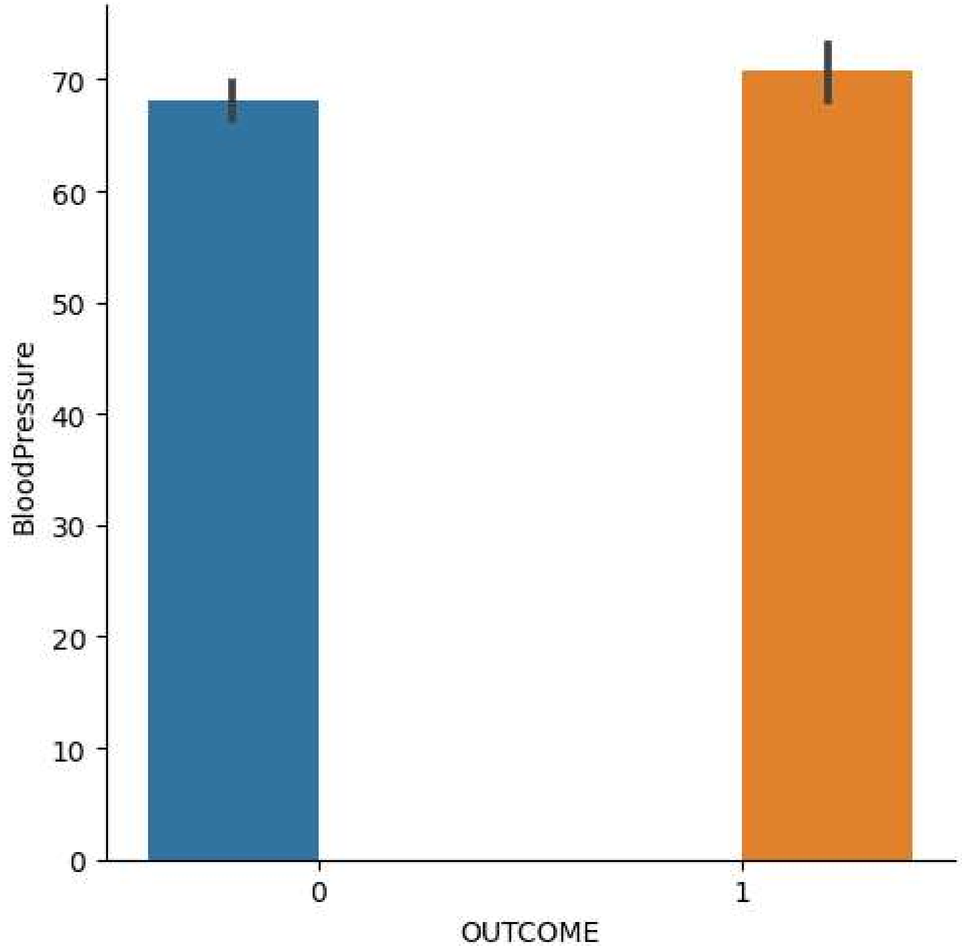
In [47]:
```
sns.catplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME')
```

Out[47]: <seaborn.axisgrid.FacetGrid at 0x2179f6b43a0>



In [48]:
```
sns.catplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME',kind='bar')
```

Out[48]: <seaborn.axisgrid.FacetGrid at 0x2179f7f91f0>

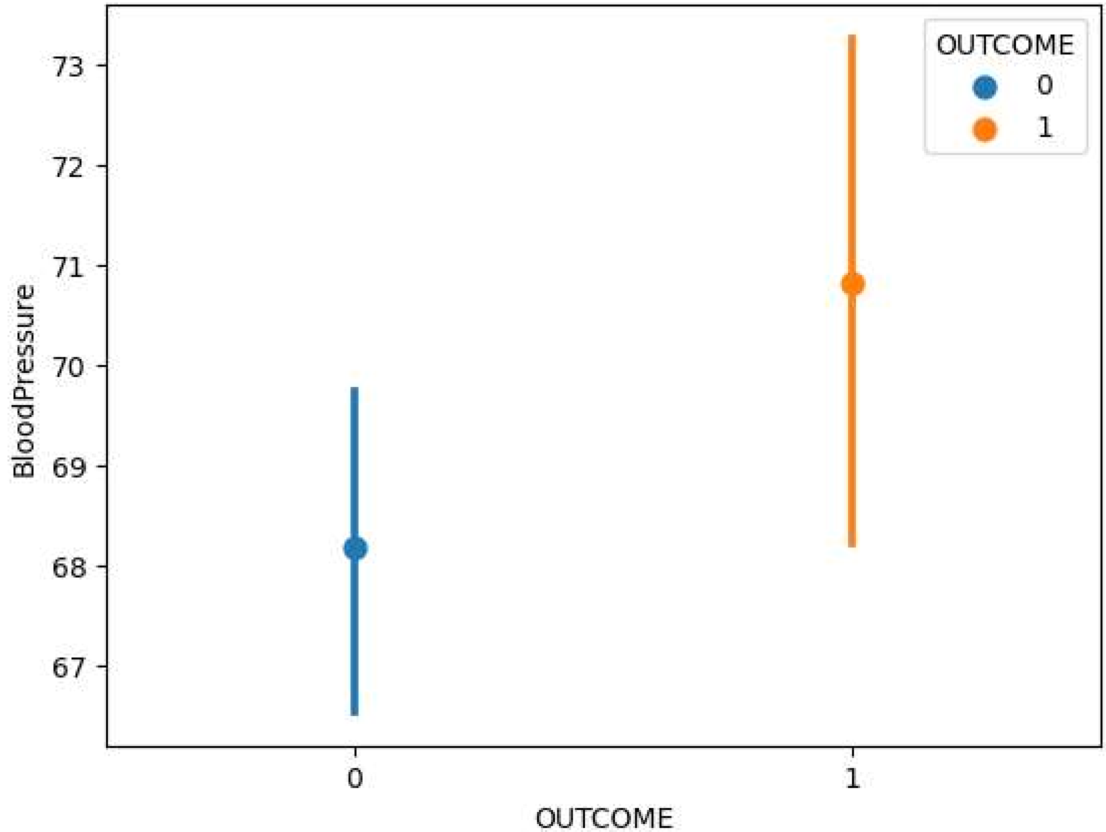


point plot

In [ ]:
```
Point plot represents an estimate of central tendency for numerical variable b
scatter plot points.
```

In [49]:
```
sns.pointplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME')
```

Out[49]:
```
<AxesSubplot:xlabel='OUTCOME', ylabel='BloodPressure'>
```

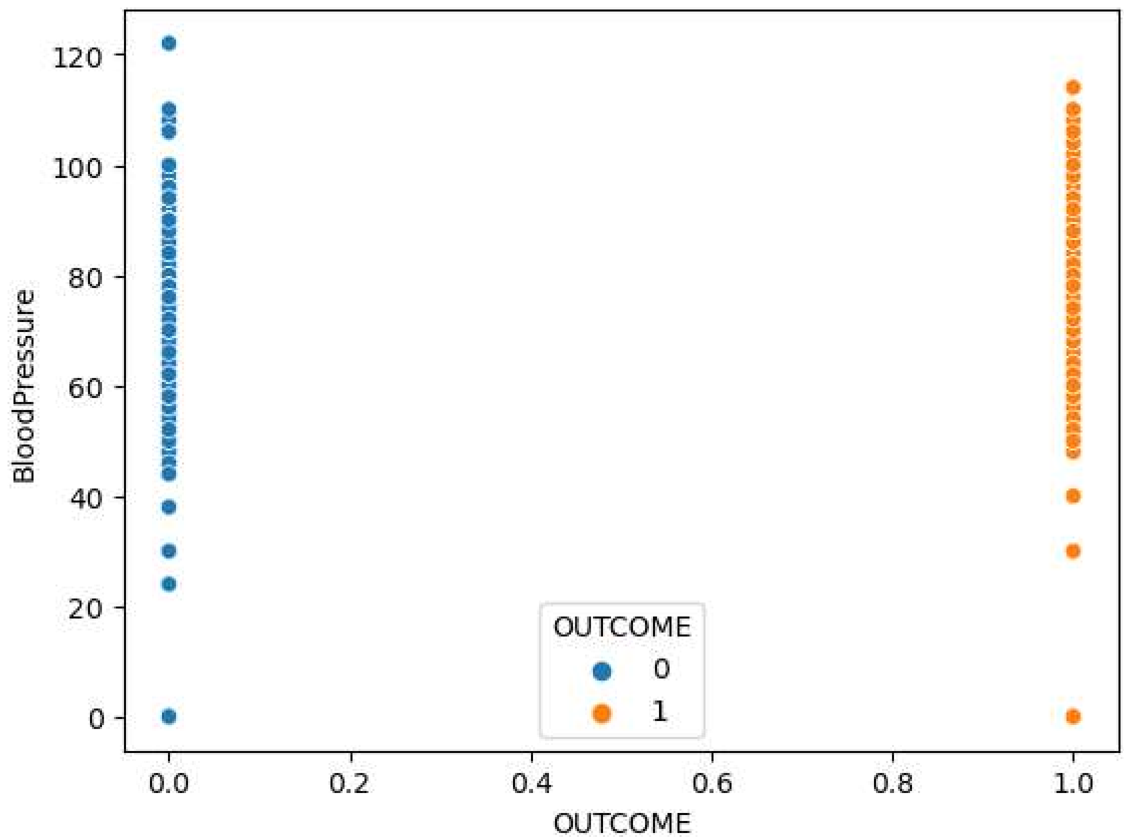


scatter plot

In [ ]:
```
In scatter plot each value of the dataset is represented by a dot.Scatter plot
variable is affected by another.Scatter plots very much like line plots that tl
vertical data points.
```

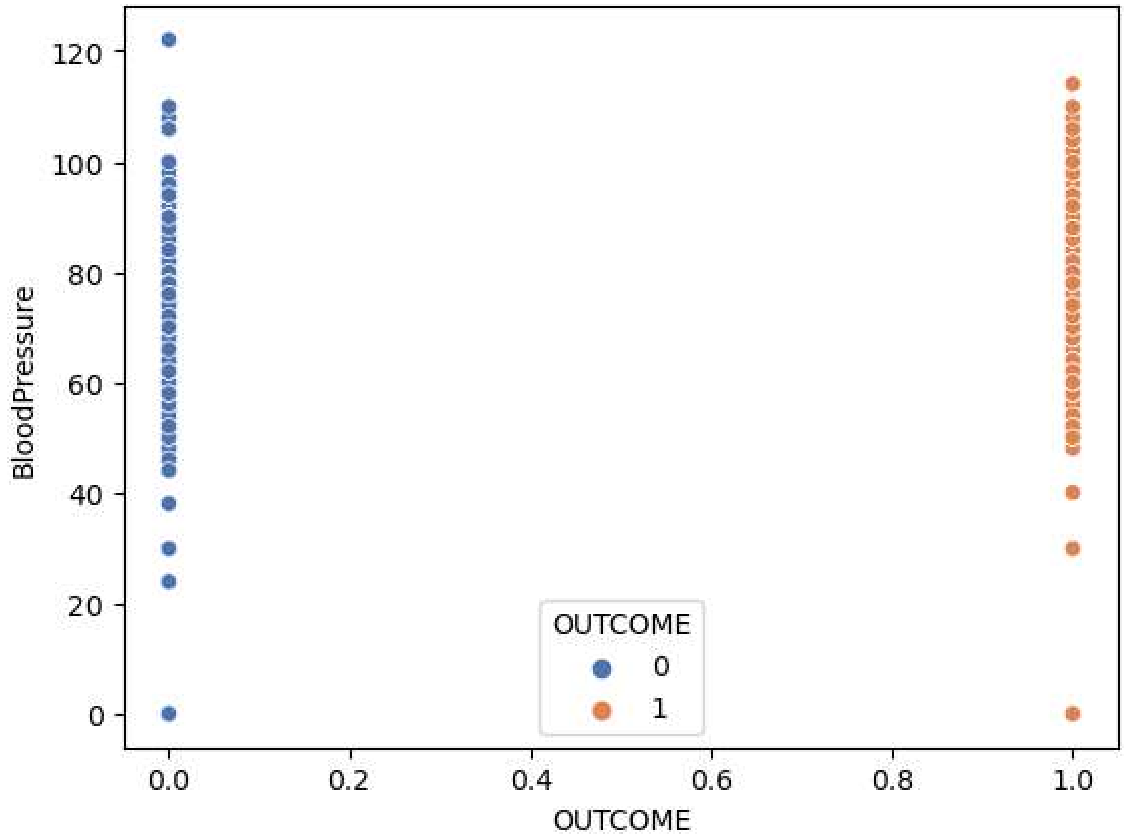
In [50]:
```
sns.scatterplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME')
```

Out[50]: <AxesSubplot:xlabel='OUTCOME', ylabel='BloodPressure'>



In [51]:
```
sns.scatterplot(x='OUTCOME',y='BloodPressure',data=df,hue='OUTCOME',palette='d
```

Out[51]:
```
<AxesSubplot:xlabel='OUTCOME', ylabel='BloodPressure'>
```



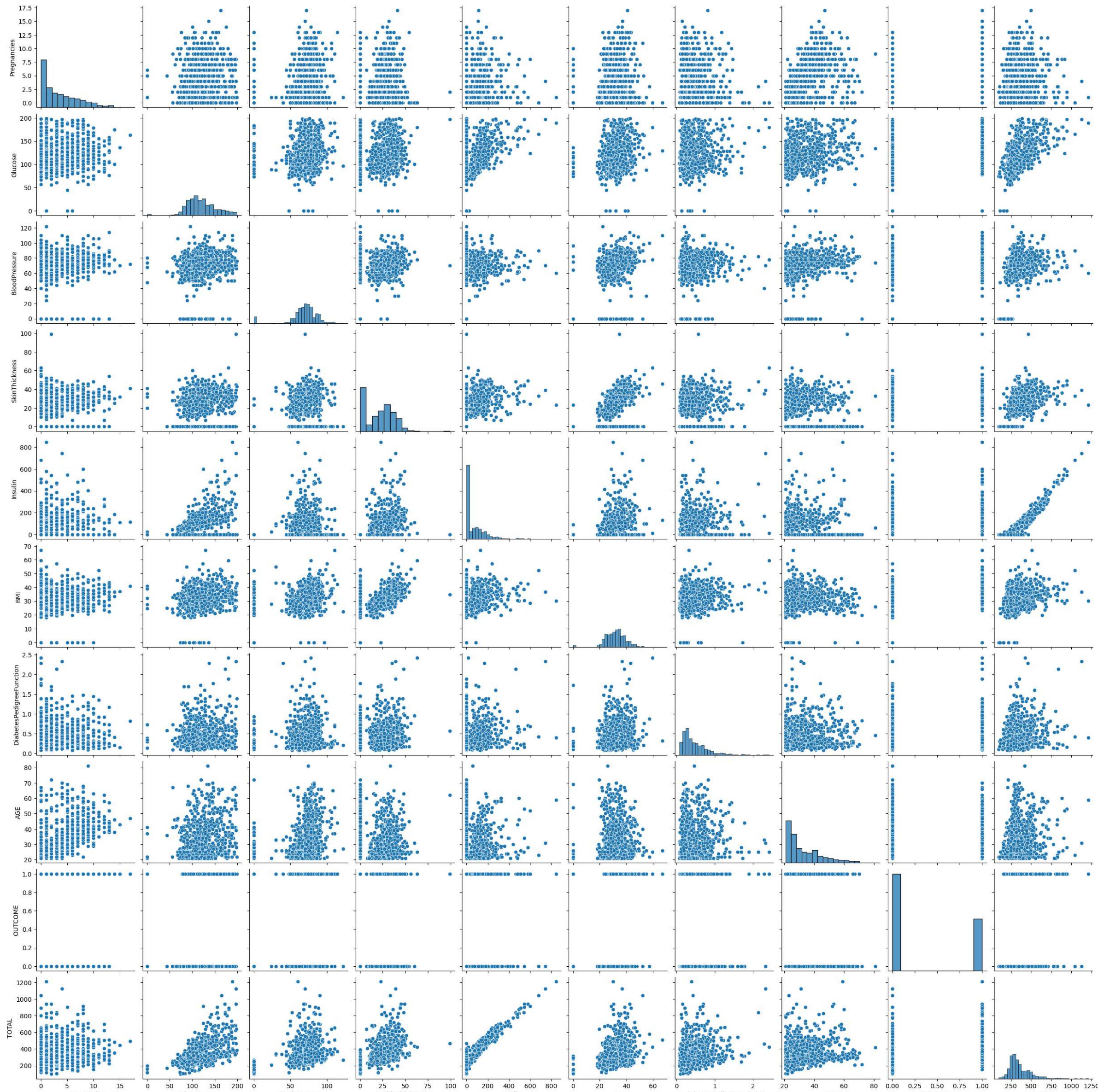
Showing different plots

Pair plot

In [ ]:
```
The default pair plots in seaborn only plots through numerical columns.Pair pl
plots histogram and scatterplot.
```

In [53]:
```
sns.pairplot(df)
```

Out[53]: <seaborn.axisgrid.PairGrid at 0x217a0972eb0>



In [ ]:
```
sns.pairplot(df,hue='OUTCOME')
```

In [3]:
```
import pandas as pd
import numpy as np
df=pd.read_csv('diabetes (1).csv')
df.head()
```

Out[3]:

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

Feature Scaling

In [ ]:
```
Data set contains different features and features contains different values.
Feature Scaling convert all values of the data set into unique.This is
most important to perform our model well.
   There are different feature techniques.
      1.MinMaxScaler
      2.StandardScaler
      3.RobustScaler
```

Standard Scaler

In [ ]:
```
Standard scaler converts the values of the features into unique.Here
 mean=0 and variance=1.
    StandardScaler=(x-mean(x))/variance
```

In [4]:
```
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
scale=st.fit_transform(df)
print(scale)
df1=pd.DataFrame(scale,columns=df.columns)
df1
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.46849198  1.4259954
   1.36589591]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.36506078 -0.19067191
  -0.73212021]
 [ 1.23388019  1.94372388 -0.26394125 ...  0.60439732 -0.10558415
   1.36589591]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.68519336 -0.27575966
  -0.73212021]
 [-0.84488505  0.1597866  -0.47073225 ... -0.37110101  1.17073215
   1.36589591]
 [-0.84488505 -0.8730192   0.04624525 ... -0.47378505 -0.87137393
  -0.73212021]]
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	
...	...	...	...	...	...	...	
763	1.827813	-0.622642	0.356432	1.722735	0.870031	0.115169	
764	-0.547919	0.034598	0.046245	0.405445	-0.692891	0.610154	
765	0.342981	0.003301	0.149641	0.154533	0.279594	-0.735190	
766	-0.844885	0.159787	-0.470732	-1.288212	-0.692891	-0.240205	
767	-0.844885	-0.873019	0.046245	0.656358	-0.692891	-0.202129	

768 rows × 9 columns

Split Features

In [ ]:
```
Split the features into x and y.And y be the Targeted feature.
```

In [5]:
```
x=df.drop(['Outcome'],axis=1)
x
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

768 rows × 8 columns

In [6]:
```
y=df['Outcome']
y
```

Out[6]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Split the data Training and Testing.

In [ ]:
```
Now split the data for training and testing.
```

In [7]:
```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=0.2,random_state=
print('Training data points in x:',train_x.shape)
print('Testing data points in x:',test_x.shape)
print('Training data points in y:',train_y.shape)
print('Testing data points in y:',test_y.shape)
```

```
Training data points in x: (614, 8)
Testing data points in x: (154, 8)
Training data points in y: (614,)
Testing data points in y: (154,)
```

In [8]:
```
print('Training percentage of x:',(train_x.shape[0]/df.shape[0])*100)
print('Testing percentage of x:',(test_x.shape[0]/df.shape[0])*100)
print('Training percentage of y:',(train_y.shape[0]/df.shape[0])*100)
print('Testing percentage of y:',(test_y.shape[0]/df.shape[0])*100)
```

```
Training percentage of x: 79.94791666666666
Testing percentage of x: 20.052083333333336
Training percentage of y: 79.94791666666666
Testing percentage of y: 20.052083333333336
```

Logistic Regression

In [ ]:
```
Logistic Regression is a supervised machine learning technique.It is used
for classification problem to find the accuracy of the model using different
Evaluation metrics such as precision,accuracy,confusion matrix and so on.
```

In [9]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(train_x,train_y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
814: ConvergenceWarning: 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 (https://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

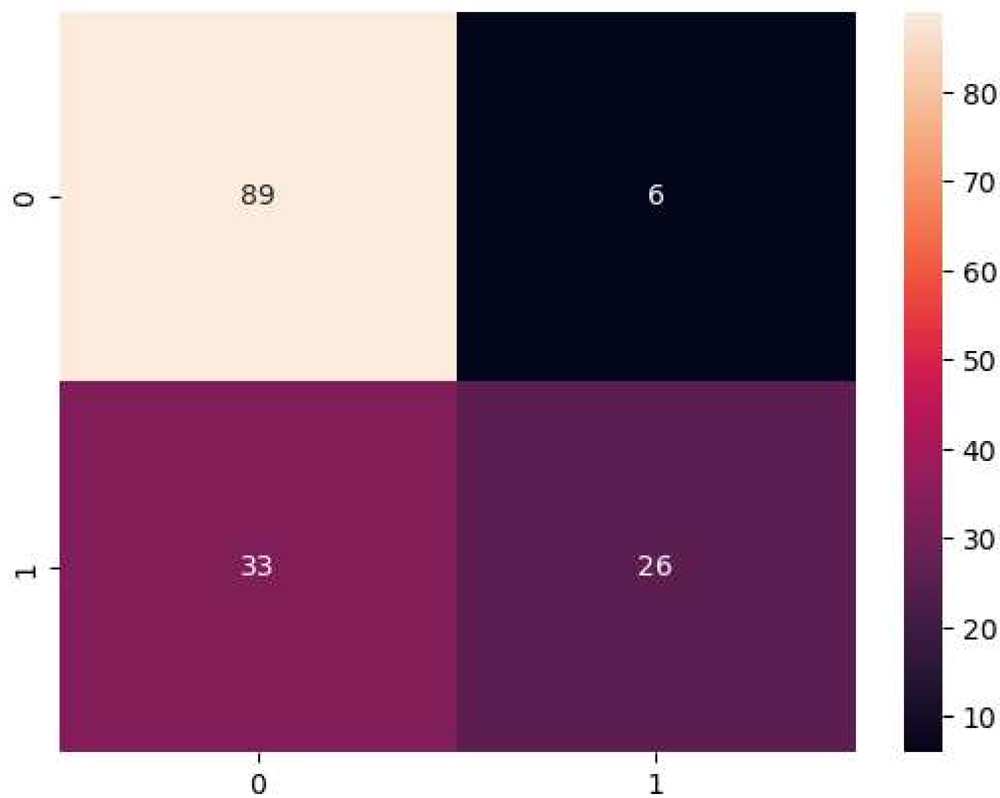
Out[9]:
```
LogisticRegression()
```

In [10]:
```
from sklearn.metrics import accuracy_score
pred=lr.predict(train_x)
acc1=accuracy_score(train_y,pred)
print('accuracy of the model in training:',acc1)
pred1=lr.predict(test_x)
acc2=accuracy_score(test_y,pred1)
print('accuracy of the model in testing:',acc2)
```

```
accuracy of the model in training: 0.7850162866449512
accuracy of the model in testing: 0.7467532467532467
```

In [14]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
label=[1,0]
acc3=confusion_matrix(test_y,pred1)
sns.heatmap(acc3,annot=True,label=label)
```

Out[14]: <AxesSubplot:>



Decision Tree

In [ ]:
```
Decision Tree is a supervised machine learning technique.It is used for
both classification and regression problems.
      once we have to fit the Decision tree on training data it will find
gini impurity of the each decision tree.If lowest gini impurity form good
feature
```

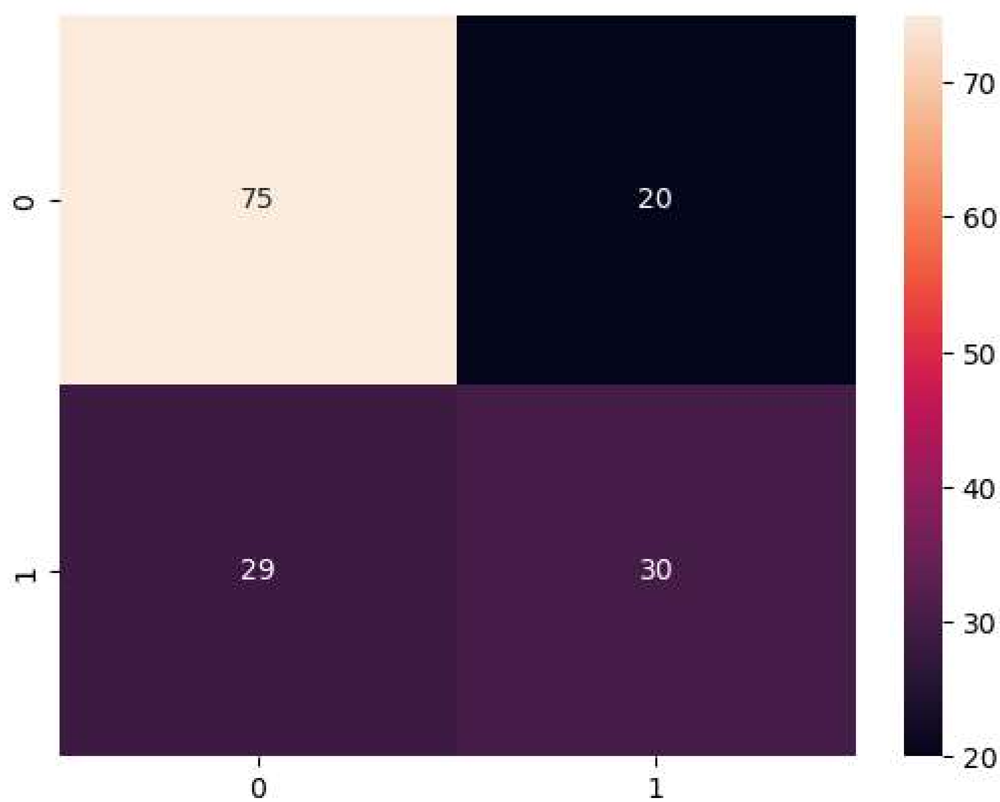
In [16]:
```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(train_x,train_y)
```

Out[16]:
```
DecisionTreeClassifier()
```

In [17]:
```
pred=dt.predict(train_x)
acc1=accuracy_score(train_y,pred)
print('accuracy of the model in training:',acc1)
pred1=dt.predict(test_x)
acc2=accuracy_score(test_y,pred1)
print('accuracy of the model in testing:',acc2)
acc4=confusion_matrix(test_y,pred1)
label=[1,0]
sns.heatmap(acc4,label=label,annot=True)
```

```
accuracy of the model in training: 1.0
accuracy of the model in testing: 0.6818181818181818
```

Out[17]:
```
<AxesSubplot:>
```



In [14]:
```
train_xx,test_x,train_yy,test_y=train_test_split(x,y,test_size=0.2,random_stat
train_x,val_x,train_y,test_y=train_test_split(train_xx,train_yy,test_size=0.2,
lr=LogisticRegression()
lr.fit(train_x,train_y)
pred=lr.predict(train_x)
acc1=accuracy_score(train_y,pred)
print('accuracy of the model in training:',acc1)
```

```
accuracy of the model in training: 0.7963340122199593

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
814: ConvergenceWarning: 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 (https://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

In [15]:
```
x={'classification technique':['Logistic Regression','Decision Tree'],'accurac
acc=pd.DataFrame(x,index=[0,1])
acc
```

Out[15]:

	classification technique	accuracy
0	Logistic Regression	0.79
1	Decision Tree	1.00

Conclusion

In [ ]:
```
Finally I got above 80% accuracy using different classification techniques.
Our model perform well.
```

In [ ]:
```
Thank you sir/mam.
```