In [1]:
```
import seaborn as sns
```

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn as sk
import matplotlib.pyplot as plt
import random
pallete = ['Accent_r', 'Blues', 'BrBG', 'BrBG_r', 'BuPu', 'CMRmap', 'CMRmap_r', 'Dark2'
           'GnBu', 'GnBu_r', 'OrRd', 'Oranges', 'Paired', 'PuBu', 'PuBuGn', 'PuRd', 'Pu
           'RdPu', 'Reds', 'autumn', 'cool', 'coolwarm', 'flag', 'flare', 'gist_rainbow
           'mako', 'plasma', 'prism', 'rainbow', 'rocket', 'seismic', 'spring', 'summer
           'turbo', 'twilight']
```

In [3]:
```
db=pd.read_csv(r'C:\Users\laksh\OneDrive\Desktop\diabetes.csv')
```

In [4]:
```
db
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
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
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 9 columns

In [5]:
```
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
```

```
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [13]:
```
db.head(10)
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	O
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	
5	5	116	74	0	0	25.6	0.201	30	
6	3	78	50	32	88	31.0	0.248	26	
7	10	115	0	0	0	35.3	0.134	29	
8	2	197	70	45	543	30.5	0.158	53	
9	8	125	96	0	0	0.0	0.232	54	

In [6]:
```
db.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [7]:
```
db.isnull().sum()
```

Out[7]:
```
Pregnancies                 0
Glucose                     0
```

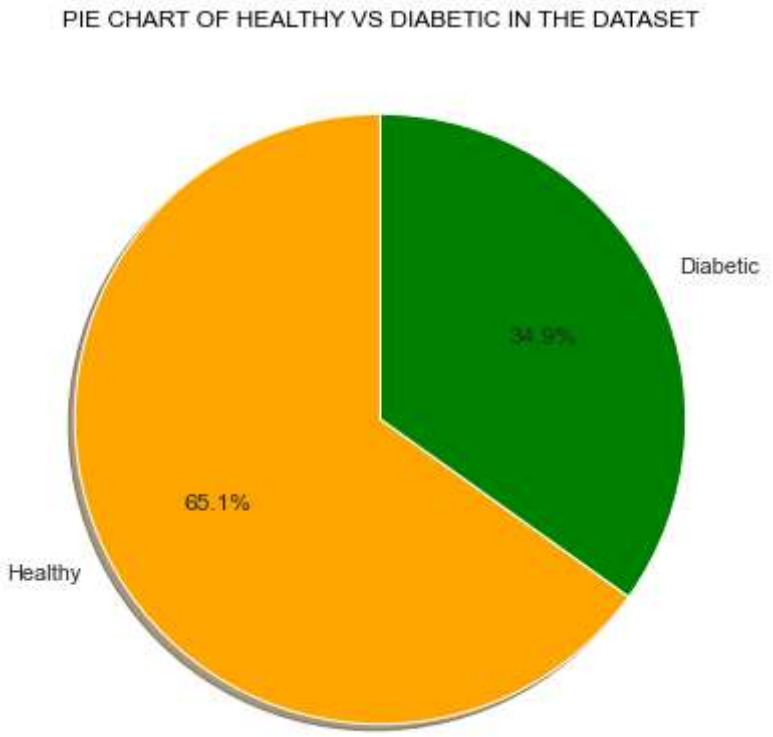
```
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Exporatory Data Analysis

Pie Chart

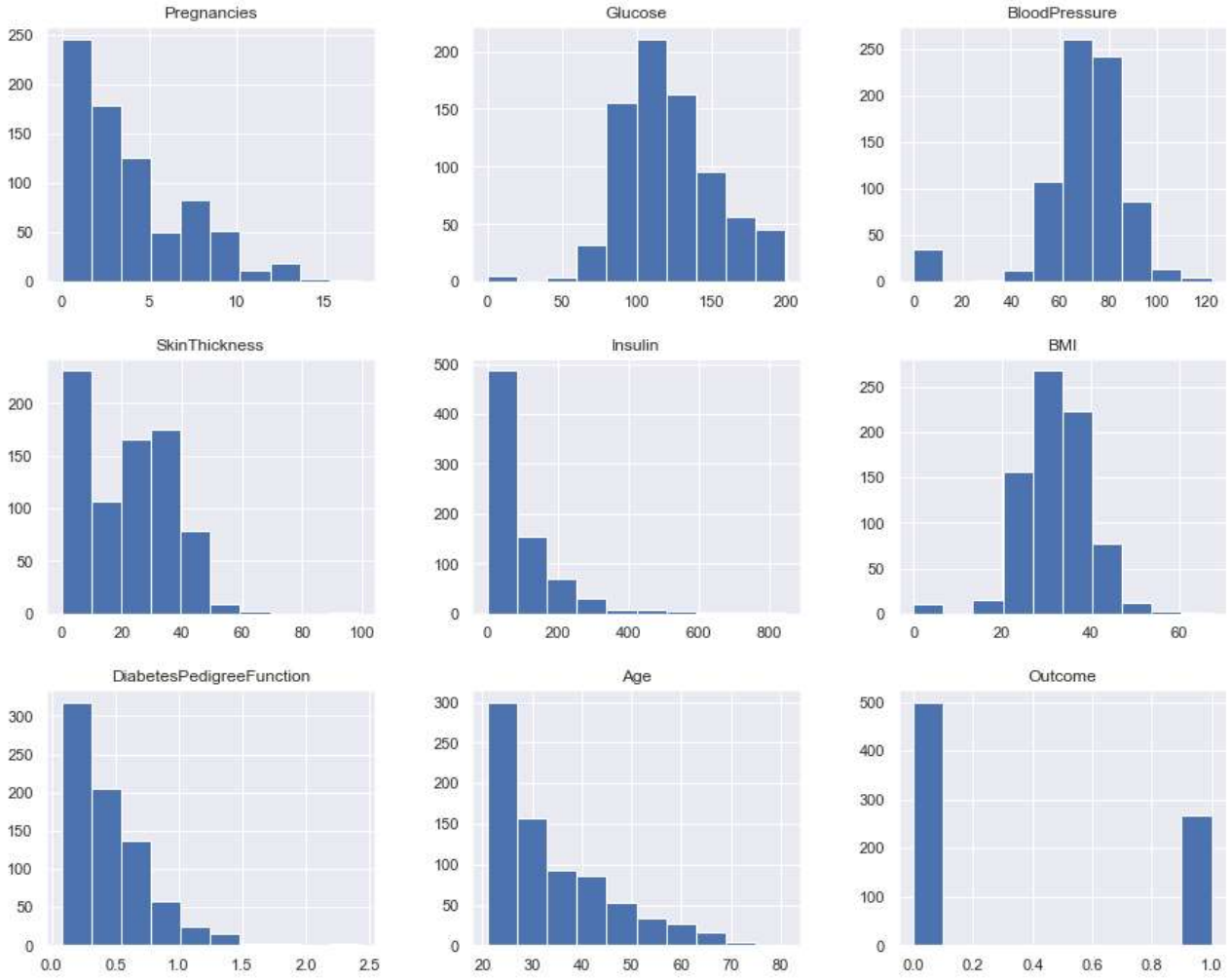
In [91]:

```
sns.set(style='darkgrid')
labels=['Healthy','Diabetic']
sizes=db['Outcome'].value_counts(sort=True)
colors=["orange","green"]
explode=(0,0)
plt.figure(figsize=(8,7))
plt.pie(sizes,explode=explode,labels=labels,colors=colors,autopct='%1.1f%%',shadow=True
plt.title("PIE CHART OF HEALTHY VS DIABETIC IN THE DATASET",color='black')
plt.show()
```



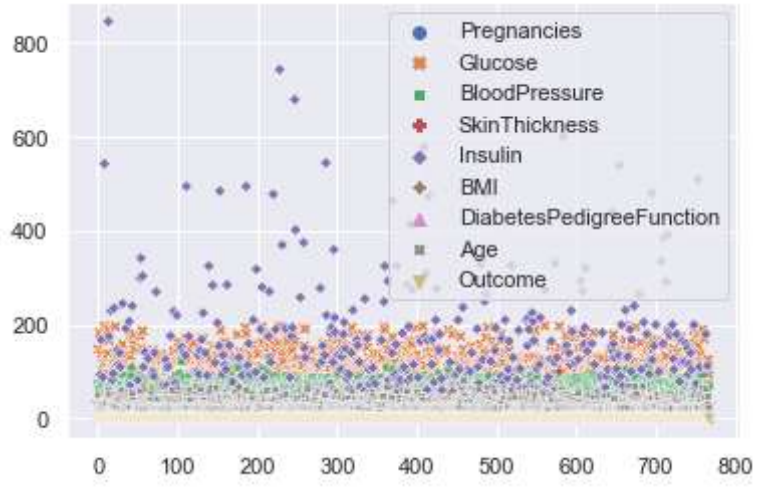
In [9]:

```
db.hist(figsize=(15,12));
```



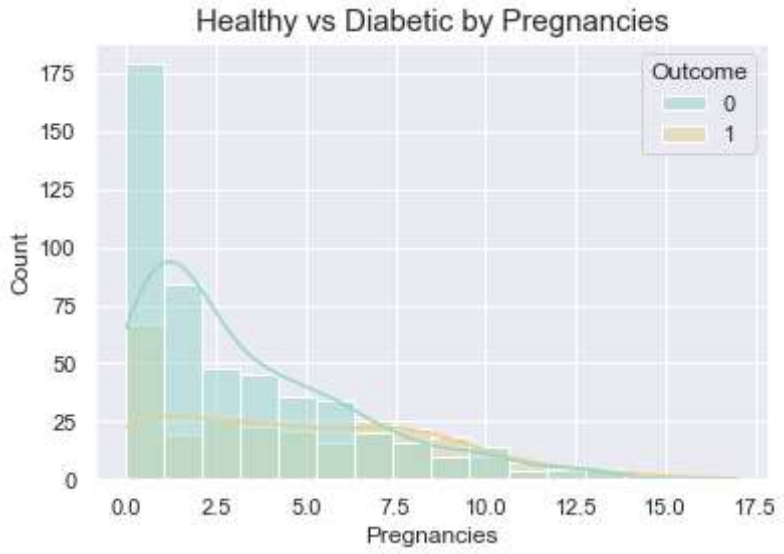
In [10]:
```
sns.scatterplot(data=db,legend='auto')
```

Out[10]: <AxesSubplot:>



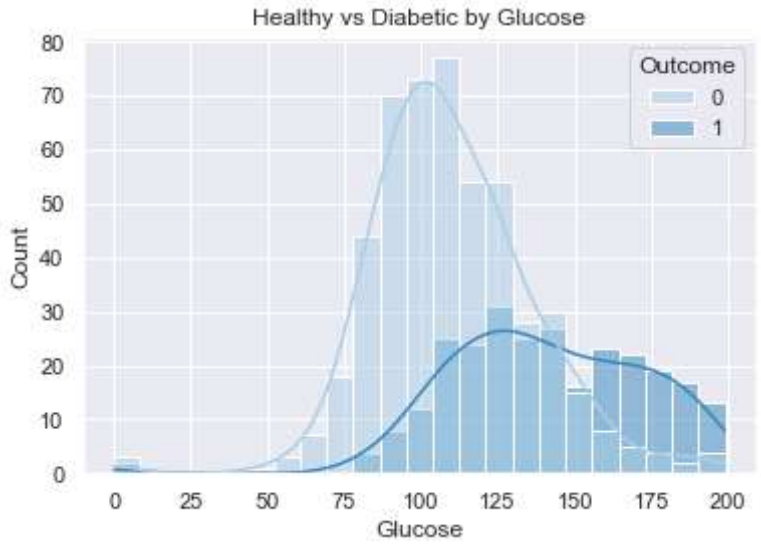
In [11]:
```
sns.histplot(x="Pregnancies", hue="Outcome", data=db, kde=True, palette=random.choice(p
plt.title("Healthy vs Diabetic by Pregnancies",fontsize=15)
```

Out[11]: Text(0.5, 1.0, 'Healthy vs Diabetic by Pregnancies')



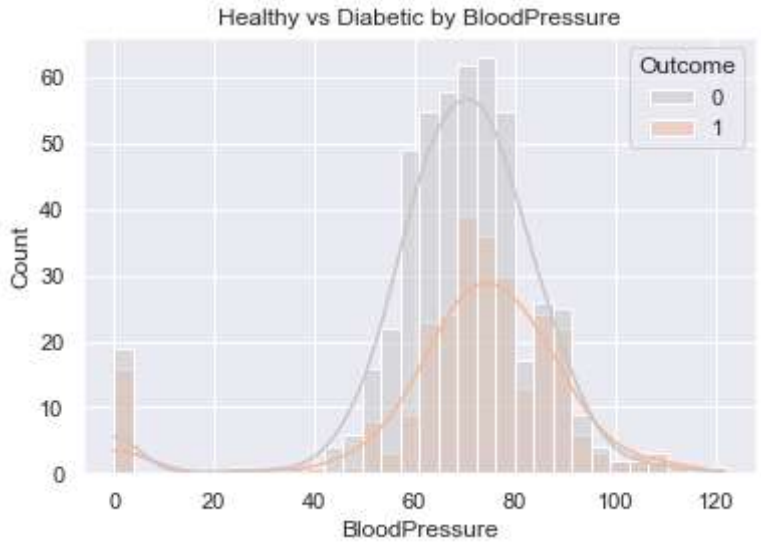
In [12]:
```
sns.histplot(x="Glucose", hue="Outcome", data=db, kde=True, palette=random.choice(palle
plt.title("Healthy vs Diabetic by Glucose")
```

Out[12]: Text(0.5, 1.0, 'Healthy vs Diabetic by Glucose')



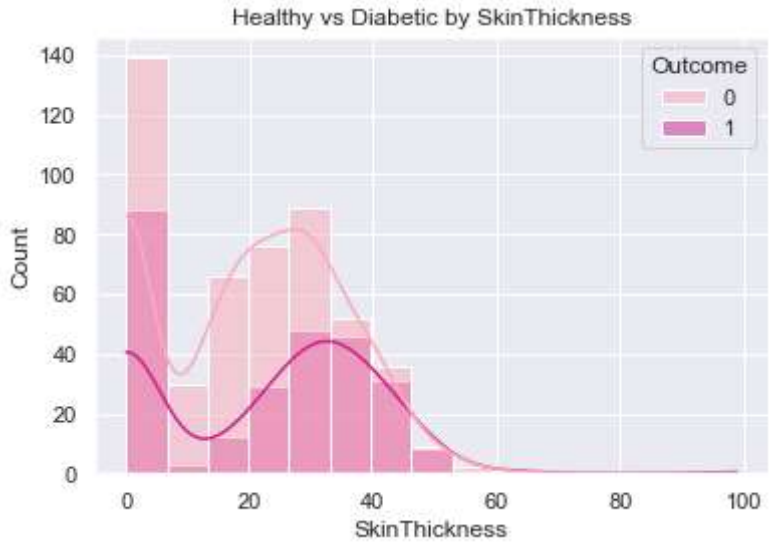
In [13]:
```
sns.histplot(x="BloodPressure", hue="Outcome", data=db, kde=True, palette=random.choice
plt.title("Healthy vs Diabetic by BloodPressure")
```

Out[13]: Text(0.5, 1.0, 'Healthy vs Diabetic by BloodPressure')



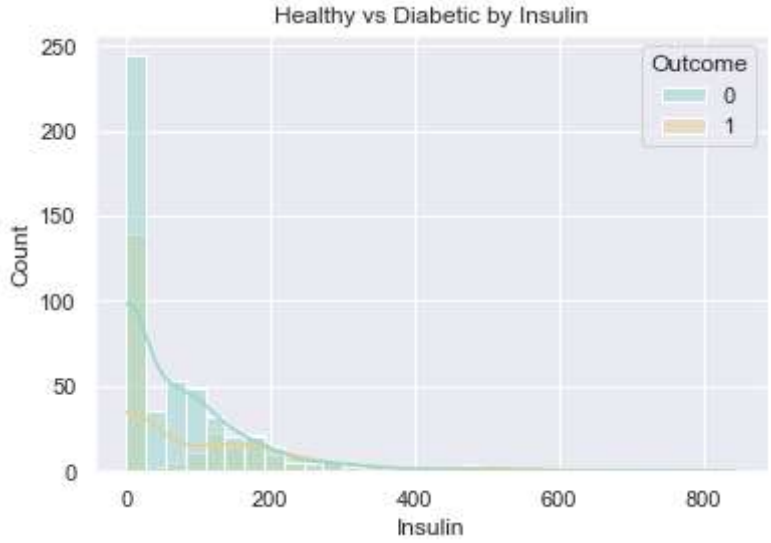
In [14]:
```
sns.histplot(x="SkinThickness", hue="Outcome", data=db, kde=True, palette=random.choice
plt.title("Healthy vs Diabetic by SkinThickness")
```

Out[14]: Text(0.5, 1.0, 'Healthy vs Diabetic by SkinThickness')



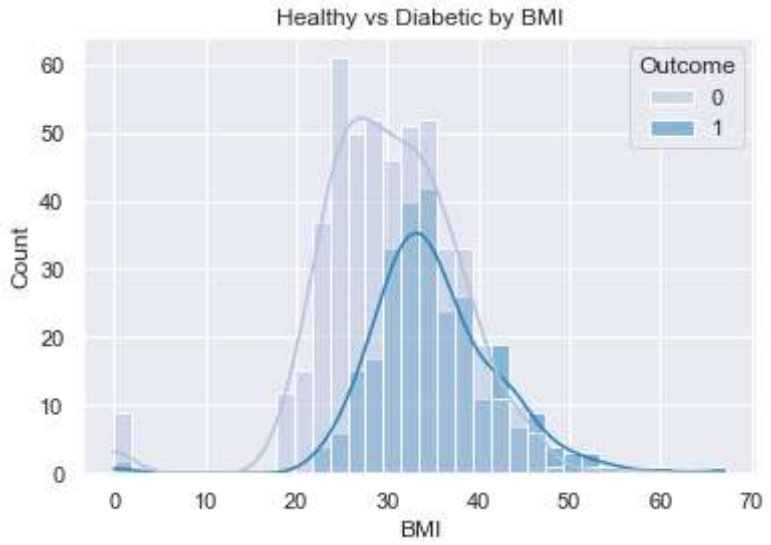
In [15]:
```
sns.histplot(x="Insulin", hue="Outcome", data=db, kde=True, palette=random.choice(palle
plt.title("Healthy vs Diabetic by Insulin")
```

Out[15]: Text(0.5, 1.0, 'Healthy vs Diabetic by Insulin')



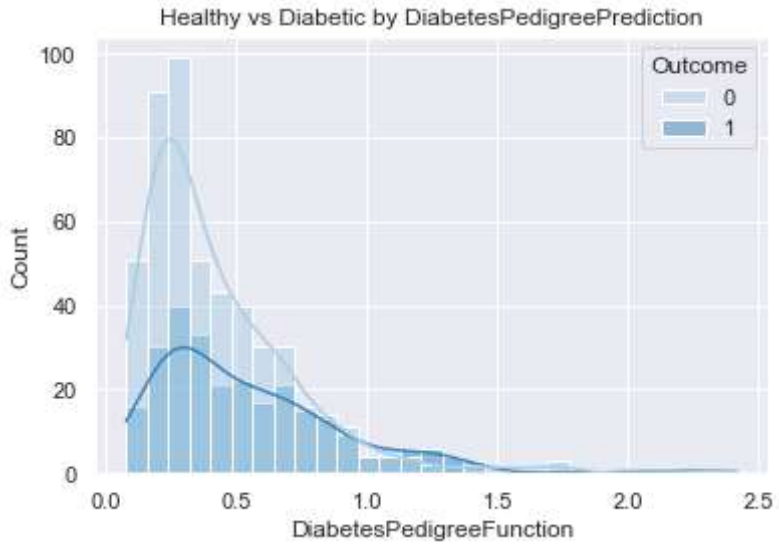
In [16]:
```
sns.histplot(x="BMI", hue="Outcome", data=db, kde=True, palette=random.choice(pallete),
plt.title("Healthy vs Diabetic by BMI")
```

Out[16]: Text(0.5, 1.0, 'Healthy vs Diabetic by BMI')



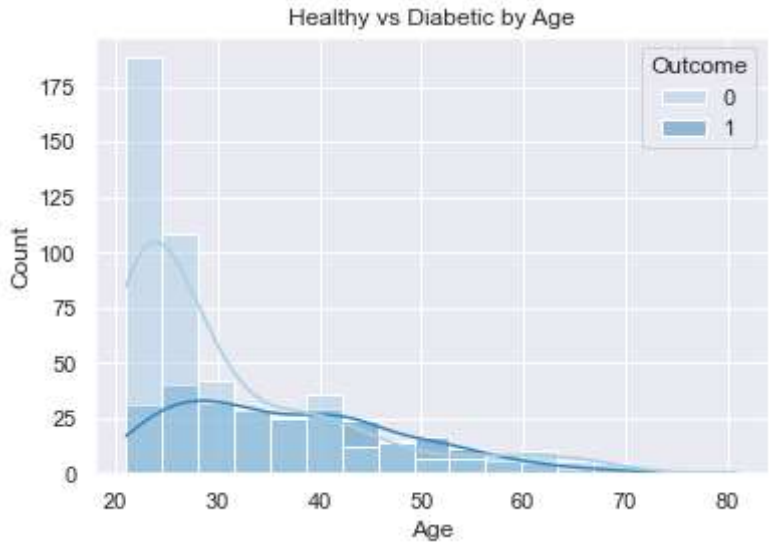
In [17]:
```
sns.histplot(x="DiabetesPedigreeFunction", hue="Outcome", data=db, kde=True, palette=ra
plt.title("Healthy vs Diabetic by DiabetesPedigreePrediction")
```

Out[17]: Text(0.5, 1.0, 'Healthy vs Diabetic by DiabetesPedigreePrediction')



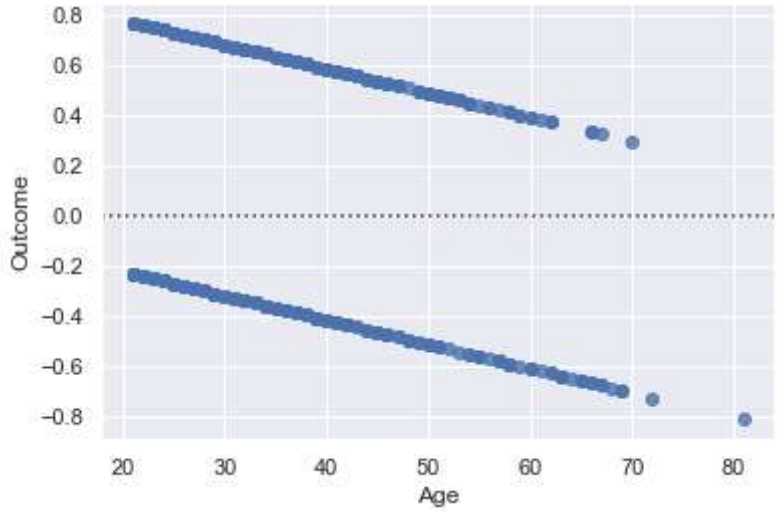
In [18]:
```
sns.histplot(x="Age", hue="Outcome", data=db, kde=True, palette=random.choice(pallete),
plt.title("Healthy vs Diabetic by Age")
```

Out[18]: Text(0.5, 1.0, 'Healthy vs Diabetic by Age')



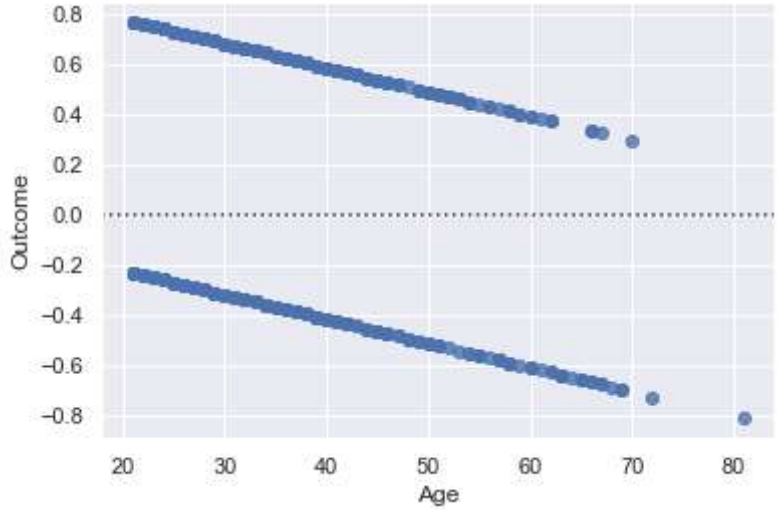
In [19]:
```
sns.residplot(x=db['Age'],y=db['Outcome'])
```

Out[19]: <AxesSubplot:xlabel='Age', ylabel='Outcome'>



In [20]:
```
sns.residplot(x=db['Age'],y=db['Outcome'])
```

Out[20]: <AxesSubplot:xlabel='Age', ylabel='Outcome'>



In [21]:
```
sns.pairplot(db,hue='Outcome',palette=random.choice(pallete))
```

Out[21]: <seaborn.axisgrid.PairGrid at 0x2312e461130>

Data Preprocessing

CLEANING THE DATASET

In [22]:
```
db.isnull().sum()
```

Out[22]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [23]:
```
db.describe()
```

Out[23]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [24]:

```
db.corr()
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	

In [25]:

```
corr=db.corr()
corr[corr>0.5]
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
Pregnancies	1.000000	NaN	NaN	NaN	NaN	NaN	
Glucose	NaN	1.0	NaN	NaN	NaN	NaN	
BloodPressure	NaN	NaN	1.0	NaN	NaN	NaN	
SkinThickness	NaN	NaN	NaN	1.0	NaN	NaN	
Insulin	NaN	NaN	NaN	NaN	1.0	NaN	
BMI	NaN	NaN	NaN	NaN	NaN	1.0	
DiabetesPedigreeFunction	NaN	NaN	NaN	NaN	NaN	NaN	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
Age	0.544341	NaN	NaN	NaN	NaN	NaN	
Outcome	NaN	NaN	NaN	NaN	NaN	NaN	

SKEW HANDLING

Dataset before Skew Handling

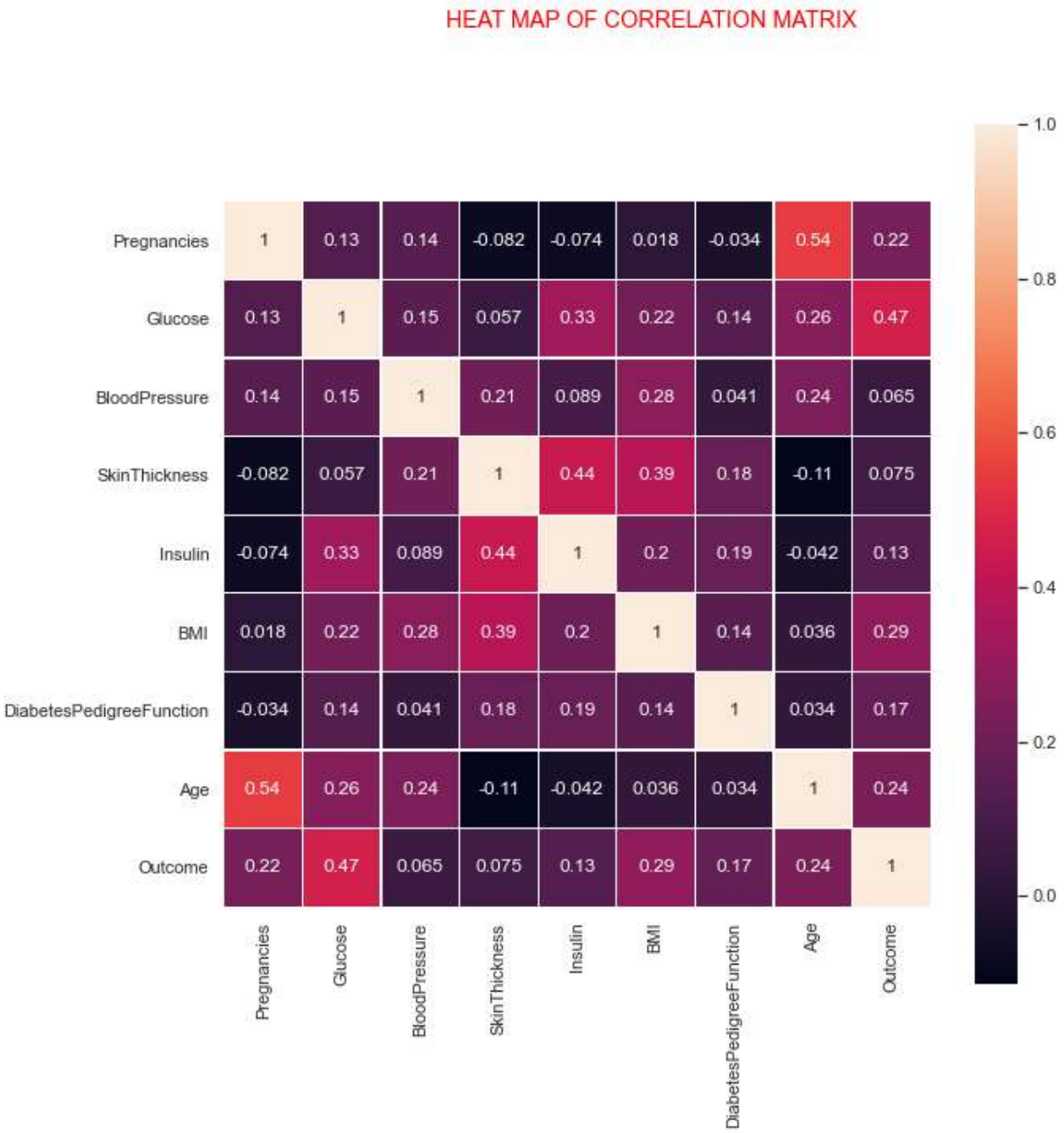
In [26]:
```
db.skew()
```

Out[26]:
```
Pregnancies                0.901674
Glucose                    0.173754
BloodPressure             -1.843608
SkinThickness              0.109372
Insulin                    2.272251
BMI                       -0.428982
DiabetesPedigreeFunction   1.919911
Age                        1.129597
Outcome                    0.635017
dtype: float64
```

Heatmap before skew handling

In [27]:
```
plt.figure(figsize=(10,10))
s=db.corr()
plt.suptitle("HEAT MAP OF CORRELATION MATRIX",color='red',ha='center')
sns.heatmap(data=s,linecolor='white',linewidths=0.1,square=True,annot=True)
```

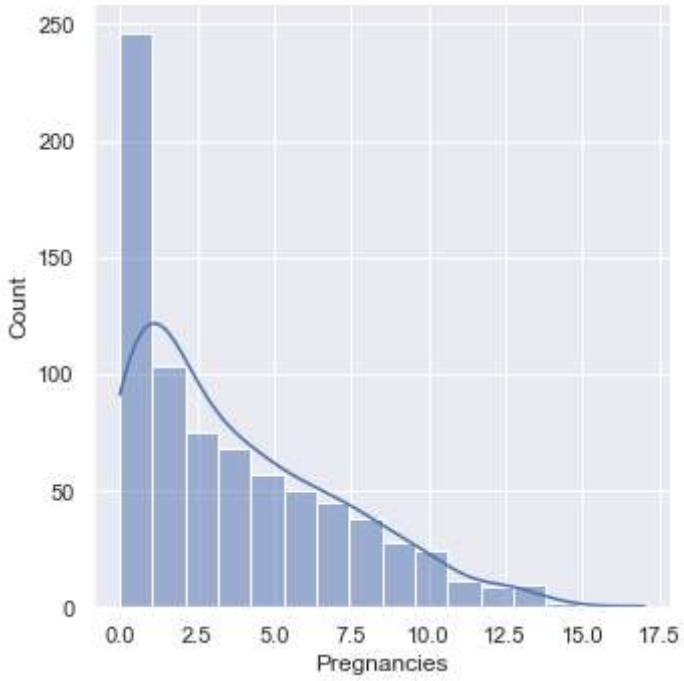
Out[27]:
```
<AxesSubplot:>
```



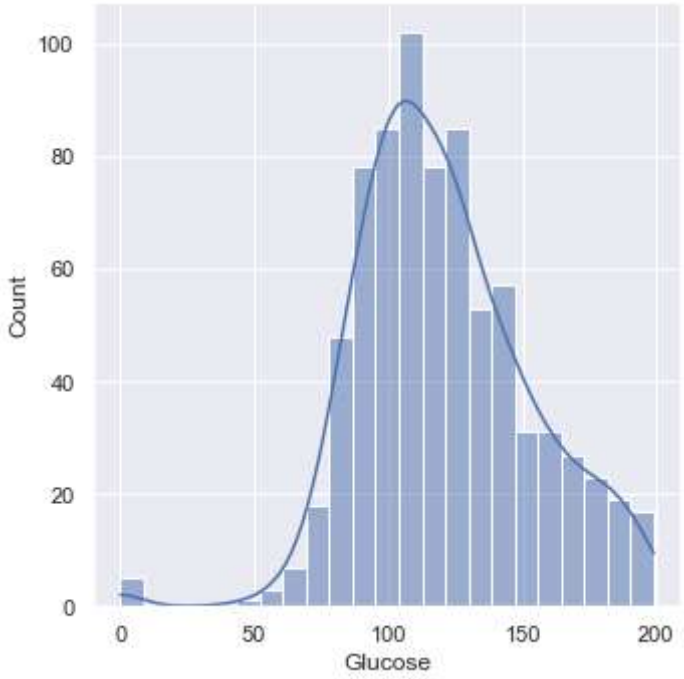
In [28]:

```
from scipy.stats import skew
for col in db:
    print(col)
    print(skew(db[col]))
    plt.figure()
    sns.displot((db[col]),kde=True)
    plt.show()
```

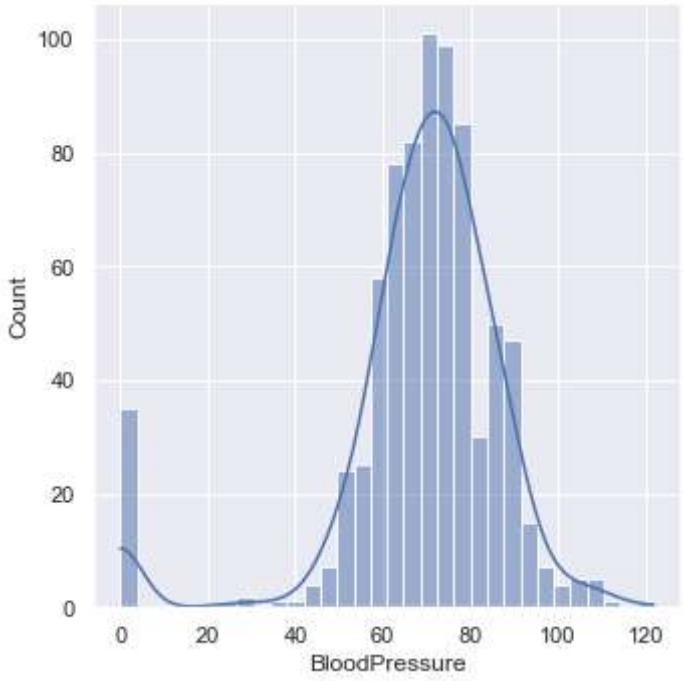
```
Pregnancies
0.8999119408414357
<Figure size 432x288 with 0 Axes>
```



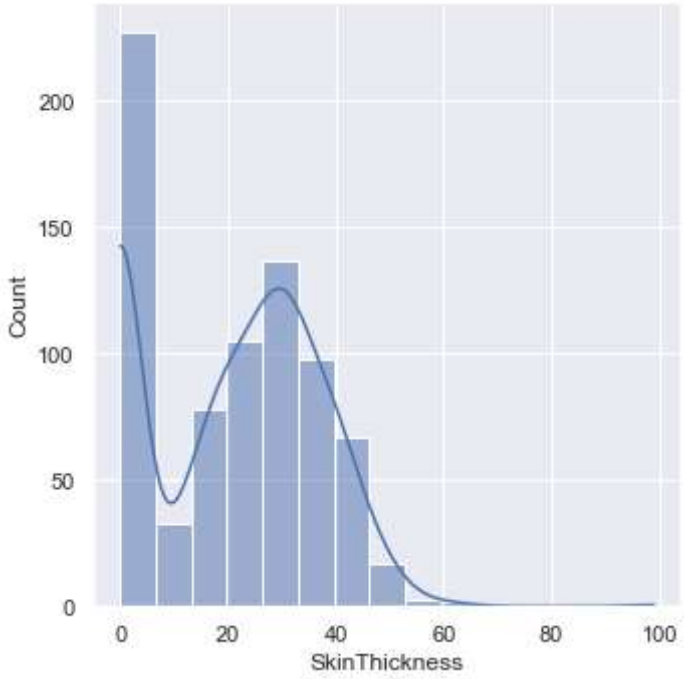

```
Glucose
0.17341395519987735
<Figure size 432x288 with 0 Axes>
```



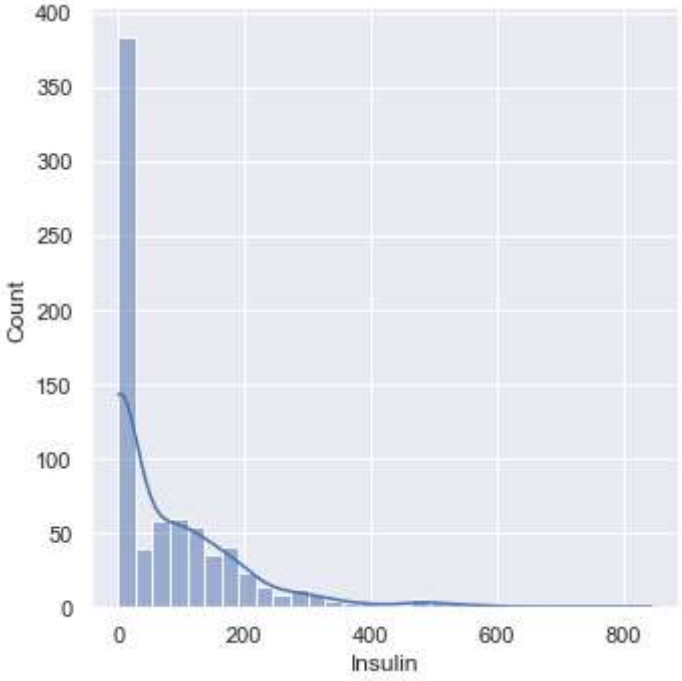

```
BloodPressure
-1.8400052311728738
<Figure size 432x288 with 0 Axes>
```



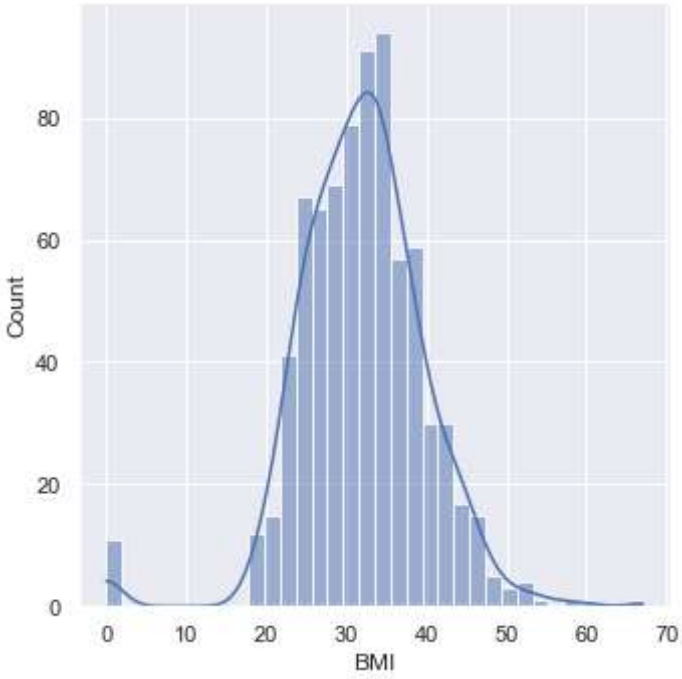

```
SkinThickness
0.109158762323673
<Figure size 432x288 with 0 Axes>
```



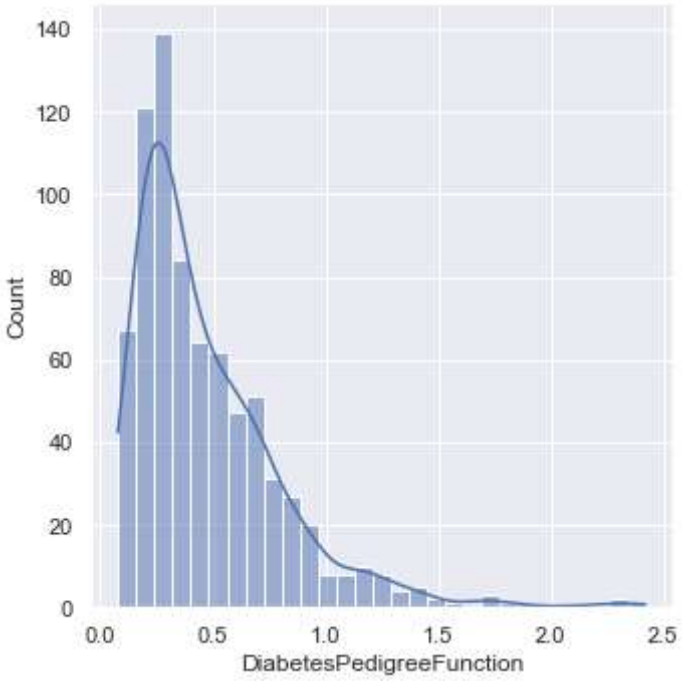

```
Insulin
2.2678104585131753
<Figure size 432x288 with 0 Axes>
```



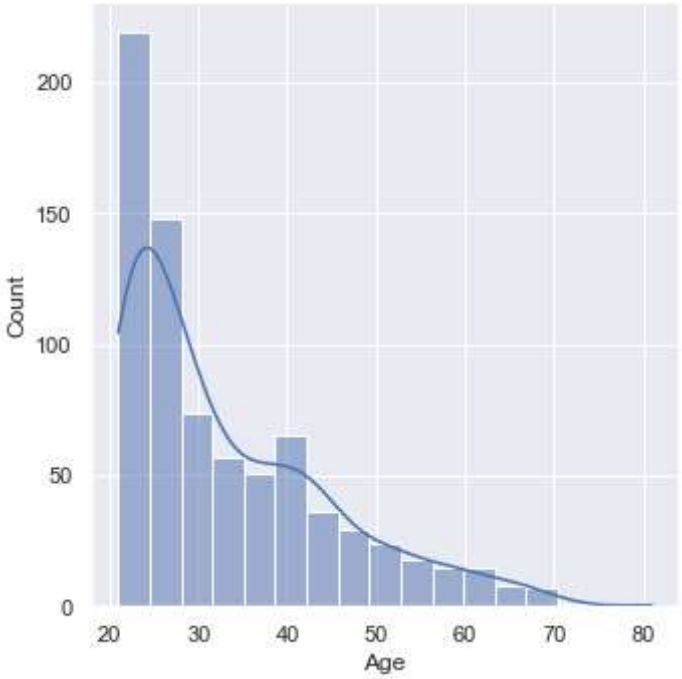

```
BMI
-0.42814327880861786
<Figure size 432x288 with 0 Axes>
```



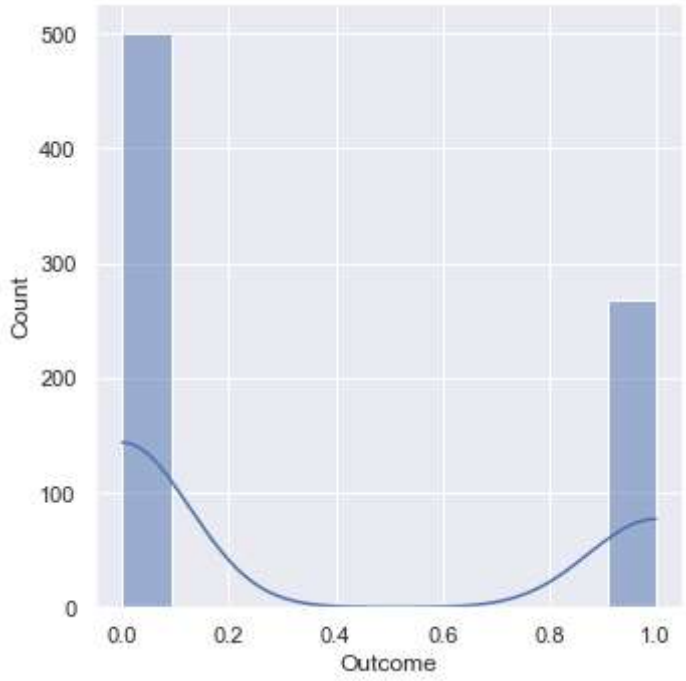

```
DiabetesPedigreeFunction
1.9161592037386292
<Figure size 432x288 with 0 Axes>
```




```
Age
1.127389259531697
<Figure size 432x288 with 0 Axes>
```




```
Outcome
0.6337757030614577
<Figure size 432x288 with 0 Axes>
```

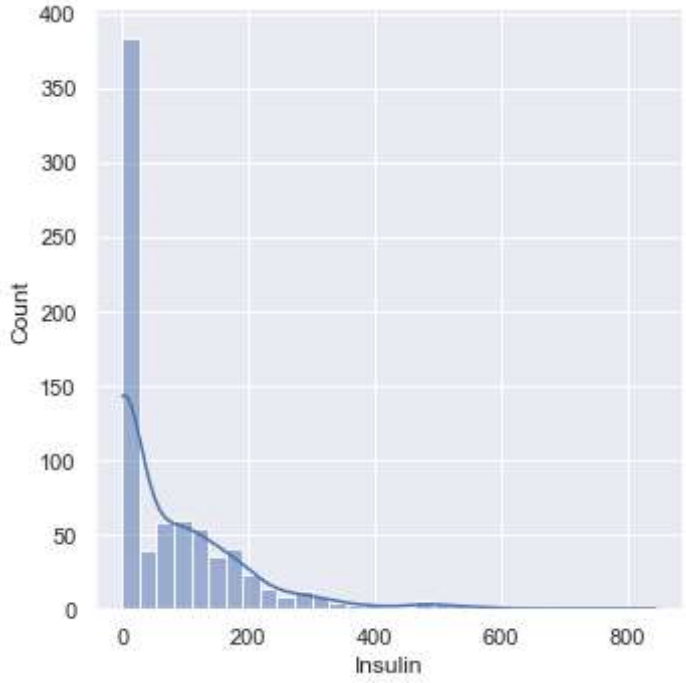


Dataset after Skew Handling

Here we are handling the skewed data by using "sqrt" function which we need to import from scipy.stats

In [29]:
```
skew(db['Insulin'])
plt.figure()
sns.displot(db['Insulin'],kde=True)
plt.show()
```

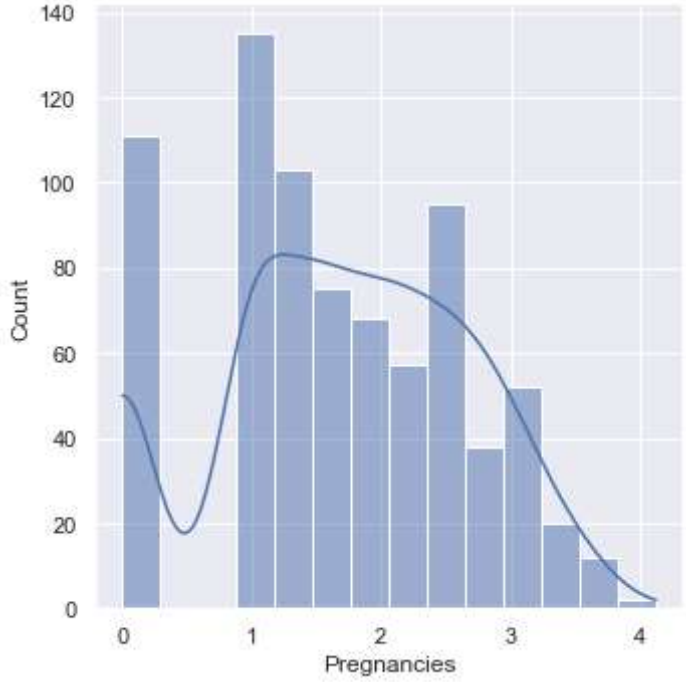
```
<Figure size 432x288 with 0 Axes>
```



In [30]:
```
db['Pregnancies']=np.sqrt(db['Pregnancies'])
```

In [31]:
```
skew(db['Pregnancies'])
print(skew(db['Pregnancies']))
plt.figure()
sns.displot((db['Pregnancies']),kde=True)
plt.show()
```

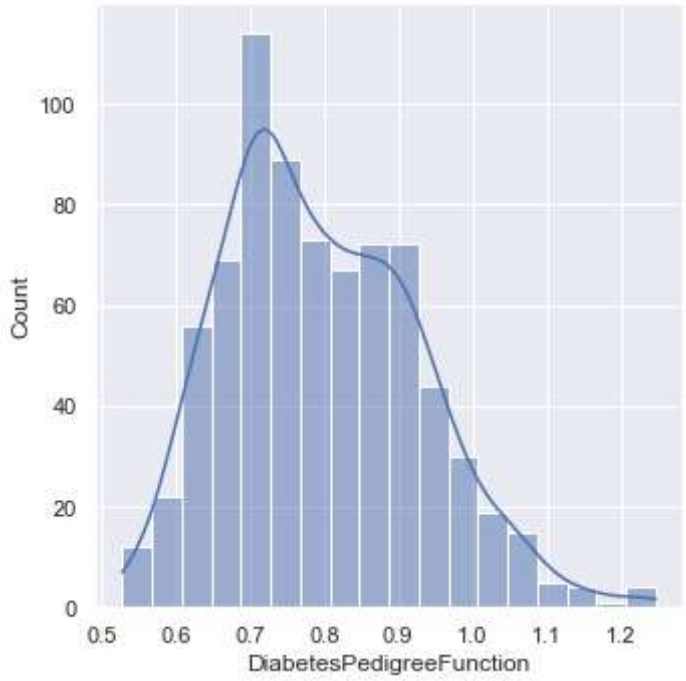
```
-0.15862732650798175
<Figure size 432x288 with 0 Axes>
```



In [32]:
```
db['DiabetesPedigreeFunction']=np.sqrt(np.sqrt(db['DiabetesPedigreeFunction']))
```

In [33]:
```
skew(db['DiabetesPedigreeFunction'])
print(skew(db['DiabetesPedigreeFunction']))
plt.figure()
sns.displot((db['DiabetesPedigreeFunction']),kde=True)
plt.show()
```

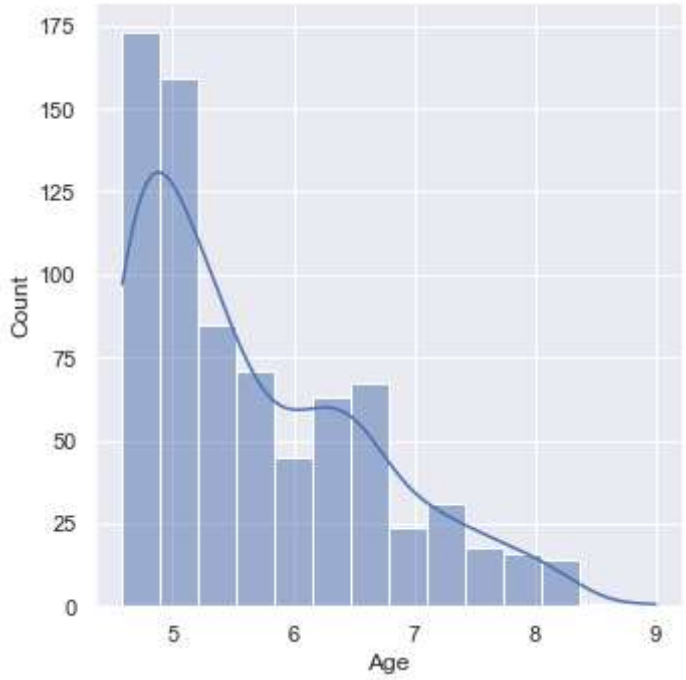
```
0.4890956246034101
<Figure size 432x288 with 0 Axes>
```



In [34]:
```
db['Age']=np.sqrt(db['Age'])
```

In [35]:
```
skew(db['Age'])
print(skew(db['Age']))
plt.figure()
sns.displot((db['Age']),kde=True)
plt.show()
```

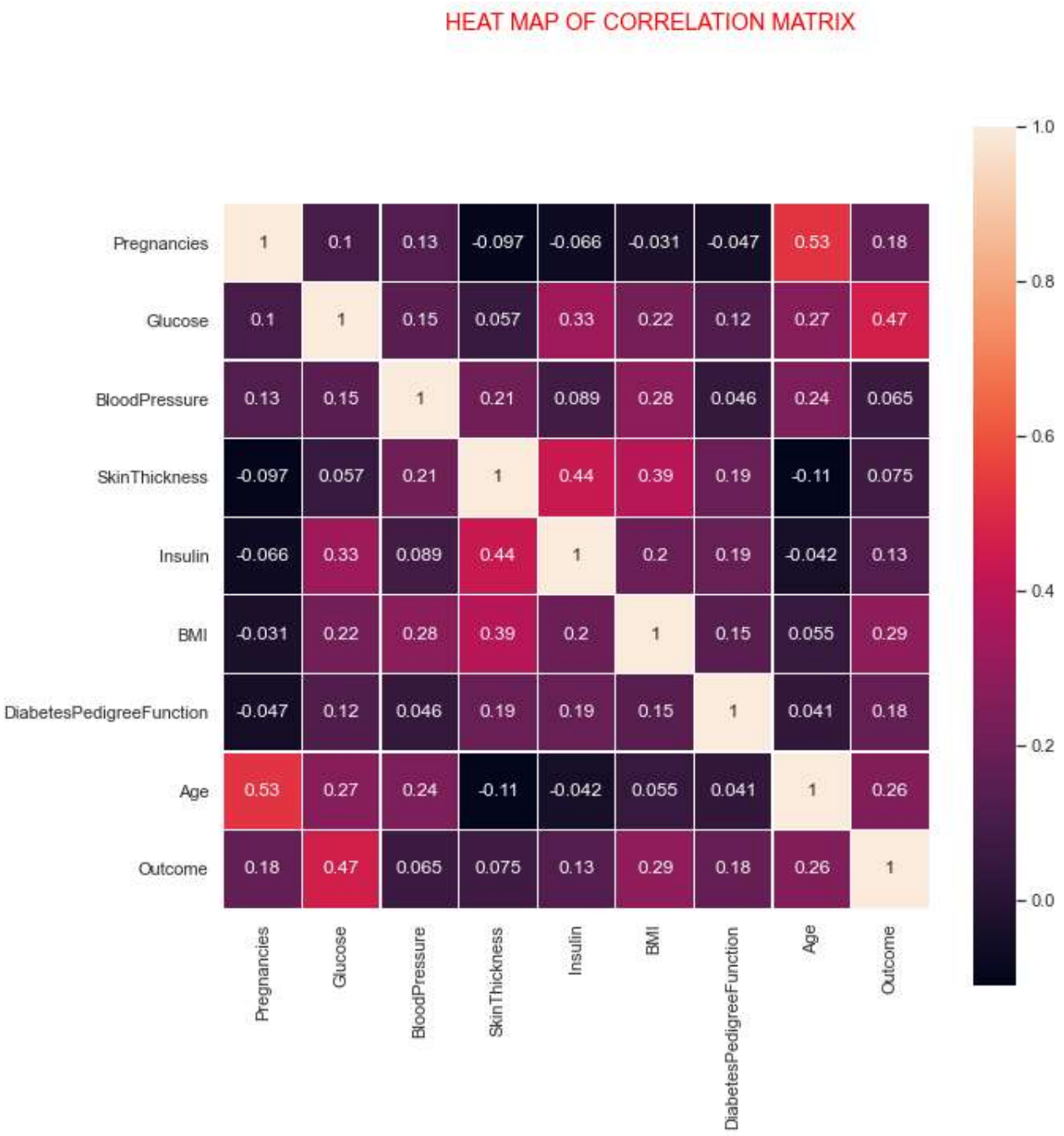
```
0.8489360850576027
<Figure size 432x288 with 0 Axes>
```



Heat map of Dataset after Skew Handling

In [36]:
```
plt.figure(figsize=(10,10))
s=db.corr()
plt.suptitle("HEAT MAP OF CORRELATION MATRIX",color='red',ha='center')
sns.heatmap(data=s,linecolor='white',linewidths=0.1,square=True,annot=True)
```

Out[36]: <AxesSubplot:>



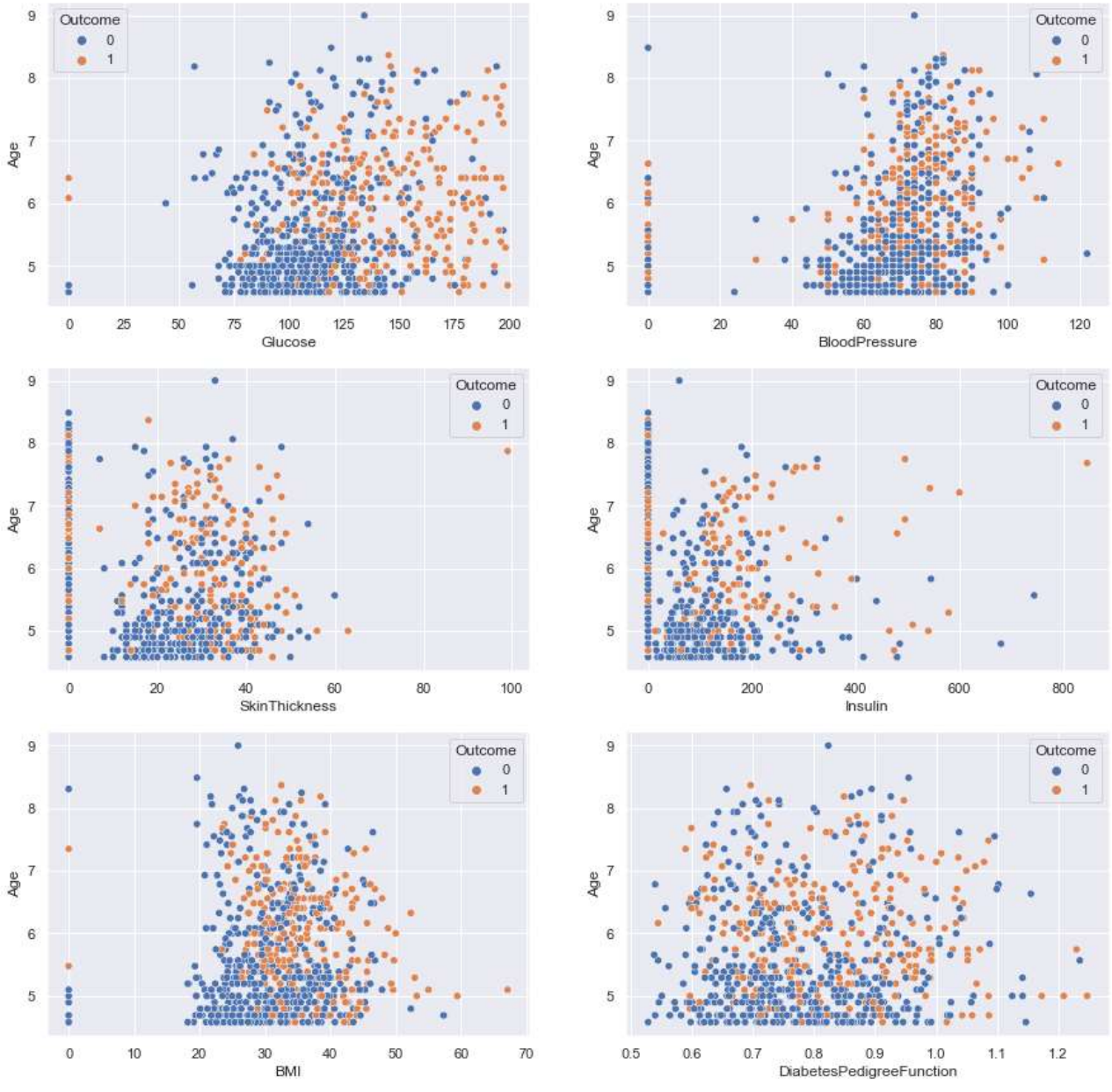
In [37]:
```
db.groupby('Outcome').size()
```

Out[37]:
```
Outcome
0    500
1    268
dtype: int64
```

Scatter plots

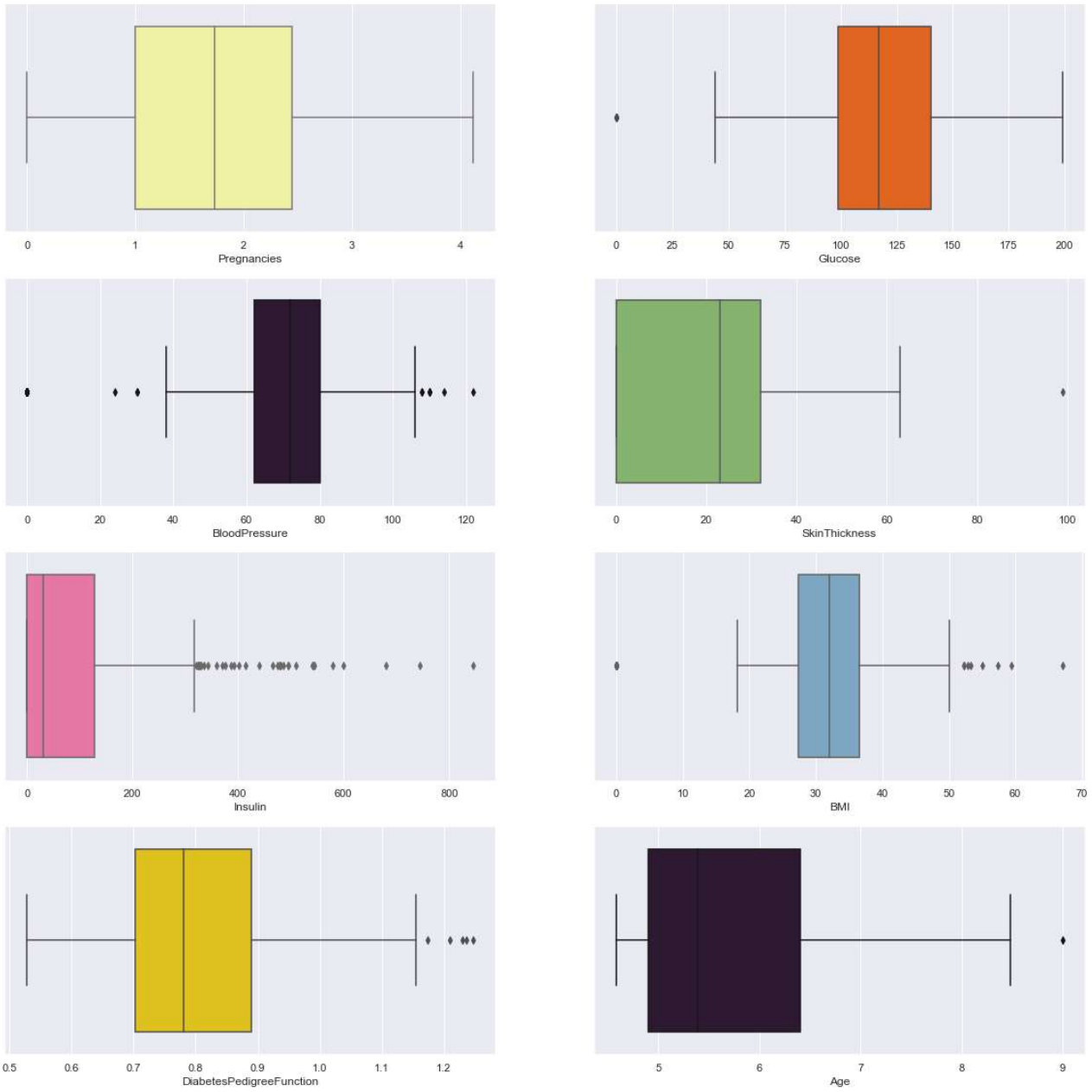
In [38]:
```
def ScatterPlot(x , y, axis):
    return sns.scatterplot(x = x, y = y,hue = 'Outcome',ax = axis,data = db)
```

In [39]:
```
fig, ax = plt.subplots(3, 2, figsize=(15, 15))
ax1 = ax[0, 0]
ax2 = ax[0, 1]
ax3 = ax[1, 0]
ax4 = ax[1, 1]
ax5 = ax[2, 0]
ax6 = ax[2, 1]
ScatterPlot('Glucose','Age',ax1)
ScatterPlot('BloodPressure','Age',ax2)
ScatterPlot('SkinThickness','Age',ax3)
ScatterPlot('Insulin','Age',ax4)
ScatterPlot('BMI','Age',ax5)
ScatterPlot('DiabetesPedigreeFunction','Age',ax6)
plt.show()
```



CHECKING OUTLIERS OF THE DATASET USING THE BOXPLOTS

In [40]:
```
fig,axs=plt.subplots(4,2,figsize=(20,20))
axs=axs.flatten()
for i in range(len(db.columns)-1):
    sns.boxplot(data=db,x=db.columns[i],ax=axs[i],palette=random.choice(pallete))
```



In [41]:
```
db.isnull().sum()
```

Out[41]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
```

```
Age                         0
Outcome                     0
dtype: int64
```

In [42]:
```
db.iloc[:,:-1]
```

Out[42]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	2.449490	148	72	35	0	33.6	0.889850	7.071
1	1.000000	85	66	29	0	26.6	0.769709	5.567
2	2.828427	183	64	0	0	23.3	0.905404	5.656
3	1.000000	89	66	23	94	28.1	0.639262	4.582
4	0.000000	137	40	35	168	43.1	1.229884	5.744
...	...	...	...	...	...	...	...	
763	3.162278	101	76	48	180	32.9	0.643056	7.937
764	1.414214	122	70	27	0	36.8	0.763607	5.196
765	2.236068	121	72	23	112	26.2	0.703544	5.477
766	1.000000	126	60	0	0	30.1	0.768611	6.855
767	1.000000	93	70	31	0	30.4	0.749165	4.795

768 rows × 8 columns

Scaling the Dataset

In [43]:
```
from sklearn.preprocessing import MinMaxScaler
mmscaler=MinMaxScaler()
mscalerdb=mmscaler.fit_transform(db)
mscalerdb
```

Out[43]:
```
array([[0.59408853, 0.74371859, 0.59016393, ..., 0.50276536, 0.56333554,
        1.        ],
       [0.24253563, 0.42713568, 0.54098361, ..., 0.33561922, 0.22302333,
        0.        ],
       [0.68599434, 0.91959799, 0.52459016, ..., 0.52440441, 0.24319116,
        1.        ],
       ...,
       [0.54232614, 0.6080402 , 0.59016393, ..., 0.24356707, 0.2025275 ,
        0.        ],
       [0.24253563, 0.63316583, 0.49180328, ..., 0.33409051, 0.5145711 ,
        1.        ],
       [0.24253563, 0.46733668, 0.57377049, ..., 0.30703719, 0.04827606,
        0.        ]])
```

Standardize the Dataset

In [44]:
```
from sklearn.preprocessing import StandardScaler
sscaler=StandardScaler()
```

```
sscalerdb=sscaler.fit_transform(db.iloc[:,:-1])
sscalerdb
```

Out[44]:
```
array([[ 0.76542237,  0.84832379,  0.14964075, ...,  0.20401277,
         0.71549208,  1.44236451],
       [-0.70600148, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.20980428, -0.12179477],
       [ 1.15009394,  1.94372388, -0.26394125, ..., -1.10325546,
         0.83528265, -0.0290984 ],
       ...,
       [ 0.54877104,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.71939143, -0.2159987 ],
       [-0.70600148,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.21826701,  1.21823107],
       [-0.70600148, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.36803021, -0.92497654]])
```

Normalizing the Dataset

In [45]:
```
from sklearn.preprocessing import Normalizer
nr=Normalizer()
nr1=nr.fit_transform(db)
nr1
```

Out[45]:
```
array([[0.01426154, 0.86169294, 0.41920197, ..., 0.00518093, 0.04116952,
        0.00582225],
       [0.00871639, 0.74089296, 0.57528159, ..., 0.00670909, 0.04853079,
        0.        ],
       [0.01447723, 0.93668089, 0.32758239, ..., 0.00463429, 0.02895447,
        0.00511847],
       ...,
       [0.01219513, 0.65991338, 0.39267573, ..., 0.00383701, 0.02987186,
        0.        ],
       [0.00699598, 0.88149397, 0.41975904, ..., 0.00537719, 0.04796205,
        0.00699598],
       [0.00804291, 0.74799064, 0.56300371, ..., 0.00602547, 0.03857244,
        0.        ]])
```

In [46]:
```
db.isnull().sum()
```

Out[46]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Classification

Feature Selection

In [92]:
```
x = db.iloc[:, :-1].values
y = db.iloc[:, -1].values
```

SPLITTING DATA INTO TRAIN AND TEST SET

In [93]:
```
from sklearn.model_selection import train_test_split
```

In [94]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=
```

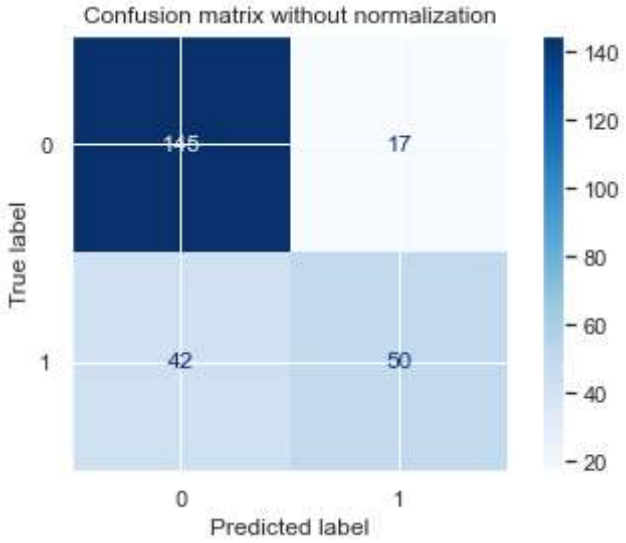
In [50]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn import linear_model
from sklearn.metrics import accuracy_score,classification_report
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import plot_confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
```

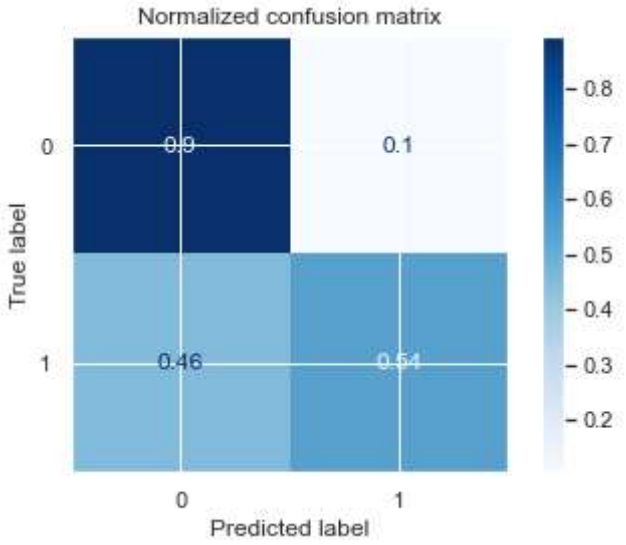
RandomForestClassifier

In [51]:
```
rf=RandomForestClassifier()
rf1= rf.fit(x_train,y_train)
yhat=rf1.predict(x_test)
acc_scores=dict()
acc_scores["RandomForestClassifier"]=accuracy_score(y_test,yhat)
print("Classification report--Random Forest Classifier")
print("**************************************************")
print(classification_report(y_test,yhat));
print("==========================================")
print("                                          ")
titles_options = [("Confusion matrix without normalization", None),("Normalized confusi
for title, normalize in titles_options:
    disp = plot_confusion_matrix(rf1, x_test, y_test,display_labels=(0,1),cmap=plt.cm.B
    disp.ax_.set_title(title)
    print(title)
    print(disp.confusion_matrix)
    print("******************************************")
```

```
Classification report--Random Forest Classifier
**************************************************
              precision    recall  f1-score   support

           0       0.78      0.90      0.83       162
           1       0.75      0.54      0.63        92

    accuracy                           0.77       254
   macro avg       0.76      0.72      0.73       254
weighted avg       0.76      0.77      0.76       254

==========================================

Confusion matrix without normalization
[[145  17]
```

```
 [ 42  50]]
****************************************
Normalized confusion matrix
[[0.89506173 0.10493827]
 [0.45652174 0.54347826]]
****************************************
```





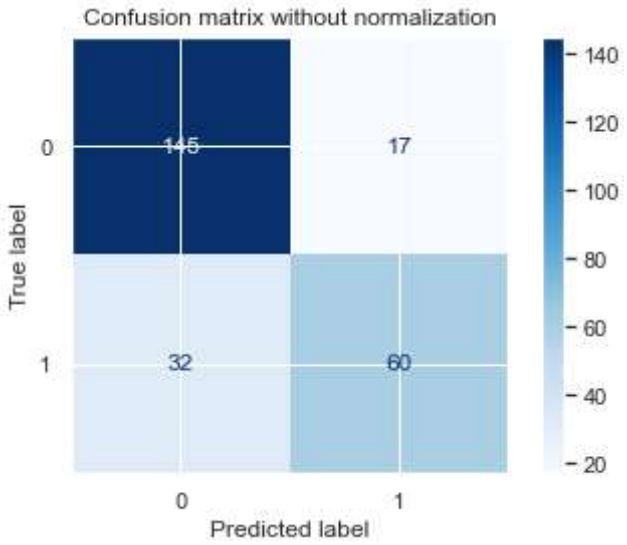
GradientBoostingClassifier

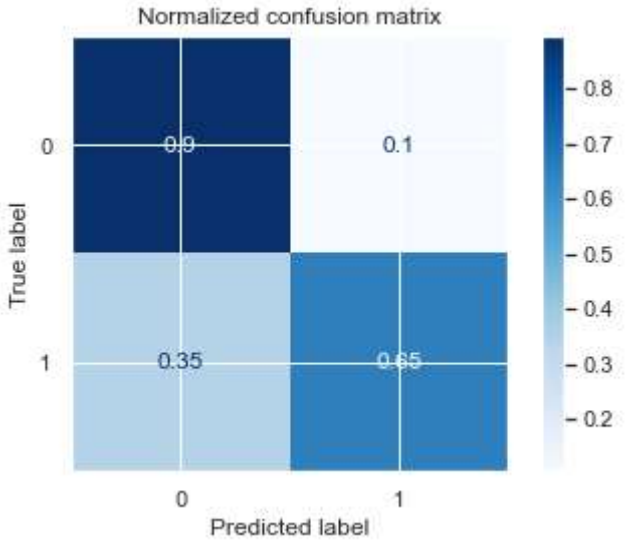
In [52]:

```
gcb=GradientBoostingClassifier()
grcb= gcb.fit(x_train,y_train)
yhat=grcb.predict(x_test)
acc_scores=dict()
acc_scores["GradientBoostingClassifier"]=accuracy_score(y_test,yhat)
print("Classification report--GradientBoostingClassifier")
print("**************************************************")
print(classification_report(y_test,yhat));
print("===============================================")
print("                                               ")
titles_options = [("Confusion matrix without normalization", None),("Normalized confusi
for title, normalize in titles_options:
    disp = plot_confusion_matrix(grcb, x_test, y_test,display_labels=(0,1),cmap=plt.cm.
    disp.ax_.set_title(title)
    print(title)
```

```
    print(disp.confusion_matrix)
    print("*****************************************")
```

```
Classification report--GradientBoostingClassifier
**************************************************
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       162
           1       0.78      0.65      0.71        92

    accuracy                           0.81       254
   macro avg       0.80      0.77      0.78       254
weighted avg       0.80      0.81      0.80       254

==========================================

Confusion matrix without normalization
[[145  17]
 [ 32  60]]
****************************************
Normalized confusion matrix
[[0.89506173 0.10493827]
 [0.34782609 0.65217391]]
****************************************
```



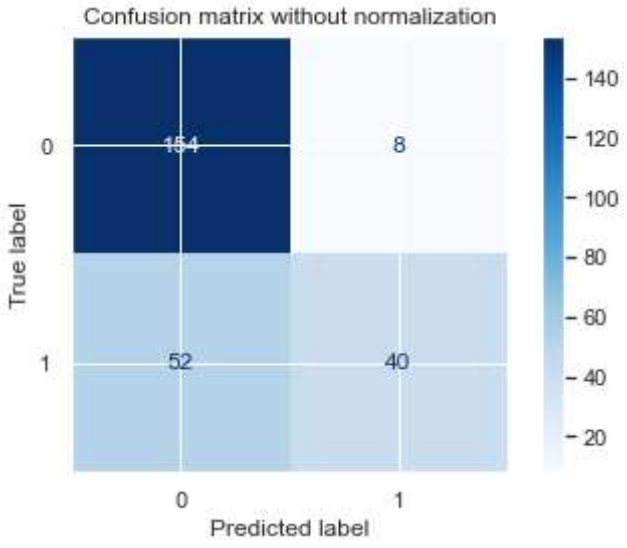


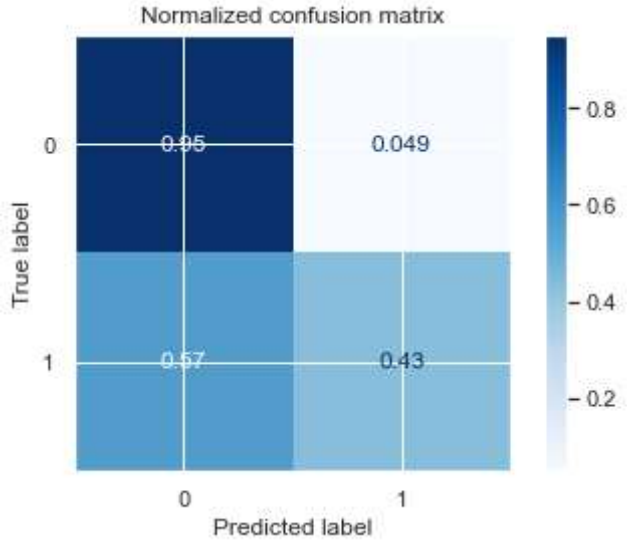
Support Vector Machine Classifier

In [53]:

```
svc=SVC()
svc1= svc.fit(x_train,y_train)
yhat=svc1.predict(x_test)
acc_scores=dict()
acc_scores["SVM Classifier"]=accuracy_score(y_test,yhat)
print("Classification report--SVM Classifier")
print("**************************************************")
print(classification_report(y_test,yhat));
print("==========================================")
print("                                          ")
titles_options = [("Confusion matrix without normalization", None),("Normalized confusi
for title, normalize in titles_options:
    disp = plot_confusion_matrix(svc1, x_test, y_test,display_labels=(0,1),cmap=plt.cm.
    disp.ax_.set_title(title)
    print(title)
    print(disp.confusion_matrix)
    print("******************************************")
```

```
Classification report--SVM Classifier
**************************************************
              precision    recall  f1-score   support

           0       0.75      0.95      0.84       162
           1       0.83      0.43      0.57        92

    accuracy                           0.76       254
   macro avg       0.79      0.69      0.70       254
weighted avg       0.78      0.76      0.74       254

==========================================

Confusion matrix without normalization
[[154   8]
 [ 52  40]]
******************************************
Normalized confusion matrix
[[0.95061728 0.04938272]
 [0.56521739 0.43478261]]
******************************************
```



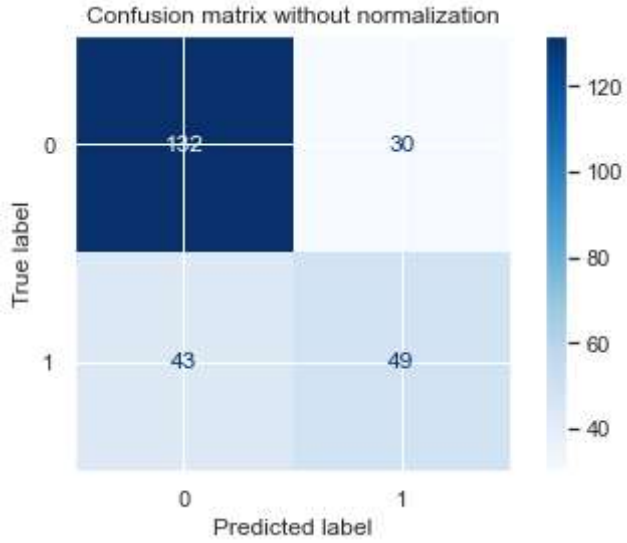


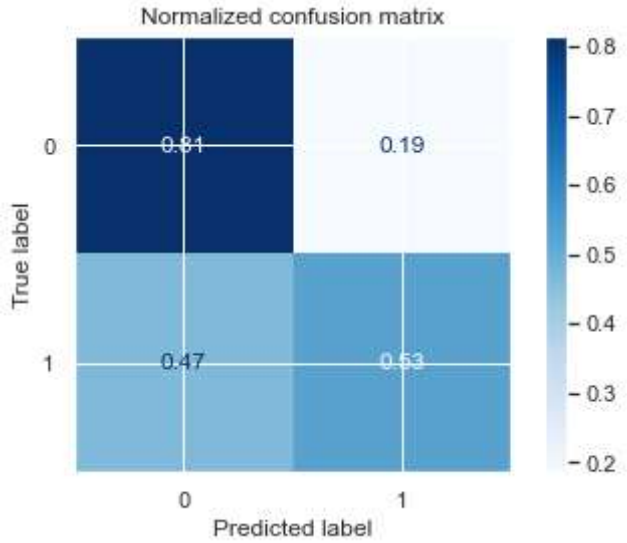
DecisionTreeClassifier

In [54]:

```
dt=DecisionTreeClassifier()
dt1=dt.fit(x_train,y_train)
yhat=dt1.predict(x_test)
acc_scores = dict()
acc_scores["DecisionTreeClassifier"] = accuracy_score(y_test,yhat)
print("Classification report - DecisionTreeClassifier")
print("===========================================")
print(classification_report(y_test,yhat))
print("===========================================")
print("                                           ")
titles_options = [("Confusion matrix without normalization", None),("Normalized confusi
for title, normalize in titles_options:
    disp = plot_confusion_matrix(dt1, x_test, y_test,display_labels=(0,1),cmap=plt.cm.B
    disp.ax_.set_title(title)
    print(title)
    print(disp.confusion_matrix)
    print("*****************************************")
```

```
Classification report - DecisionTreeClassifier
===========================================
              precision    recall  f1-score   support

           0       0.75      0.81      0.78       162
           1       0.62      0.53      0.57        92

    accuracy                           0.71       254
   macro avg       0.69      0.67      0.68       254
weighted avg       0.71      0.71      0.71       254

===========================================

Confusion matrix without normalization
[[132  30]
 [ 43  49]]
*****************************************
Normalized confusion matrix
[[0.81481481 0.18518519]
 [0.4673913  0.5326087 ]]
*****************************************
```





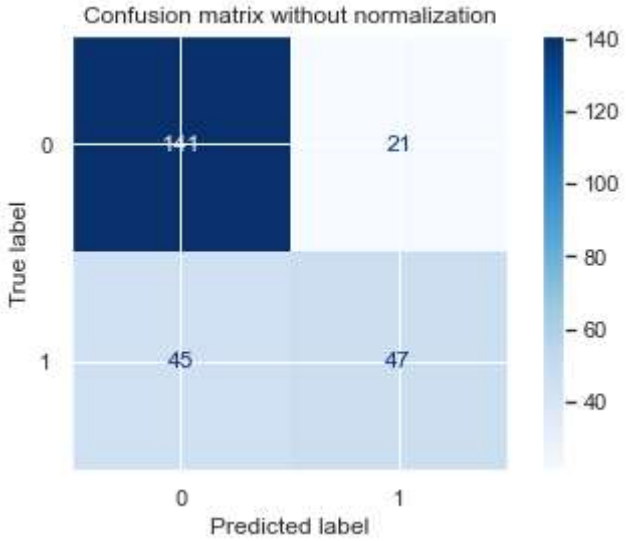
K-Nearest NeighbourClassifiers

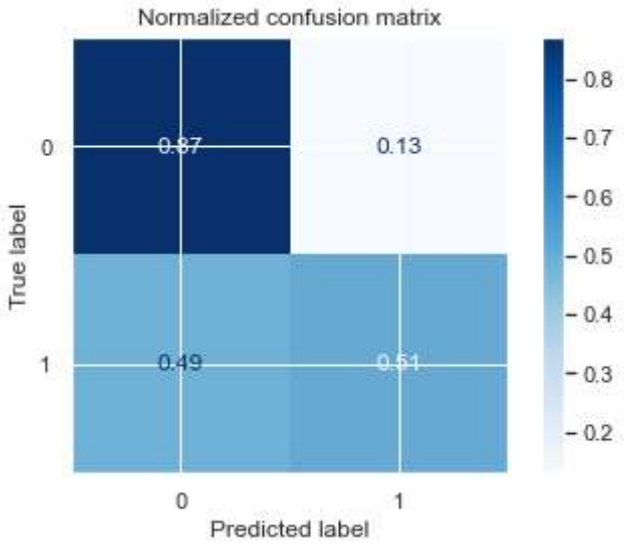
In [55]:

```
kn=KNeighborsClassifier()
knn=kn.fit(x_train,y_train)
yhat =knn.predict(x_test)
acc_scores=dict()
acc_scores["KNeighborsClassifier"]=accuracy_score(y_test,yhat)
print("Classification report - KNeighborsClassifier")
print("============================================")
print(classification_report(y_test,yhat))
print("============================================")
print("                                            ")
titles_options = [("Confusion matrix without normalization", None),("Normalized confusi
for title, normalize in titles_options:
    disp = plot_confusion_matrix(knn, x_test, y_test,display_labels=(0,1),cmap=plt.cm.B
    disp.ax_.set_title(title)
    print(title)
    print(disp.confusion_matrix)
    print("****************************************")
```

```
Classification report - KNeighborsClassifier
============================================
              precision    recall  f1-score   support
```

```
              0       0.76      0.87      0.81       162
              1       0.69      0.51      0.59        92

    accuracy                            0.74       254
   macro avg       0.72      0.69      0.70       254
weighted avg       0.73      0.74      0.73       254

==========================================

Confusion matrix without normalization
[[141  21]
 [ 45  47]]
****************************************
Normalized confusion matrix
[[0.87037037 0.12962963]
 [0.48913043 0.51086957]]
****************************************
```





Logistic Regression Model

In [7]:

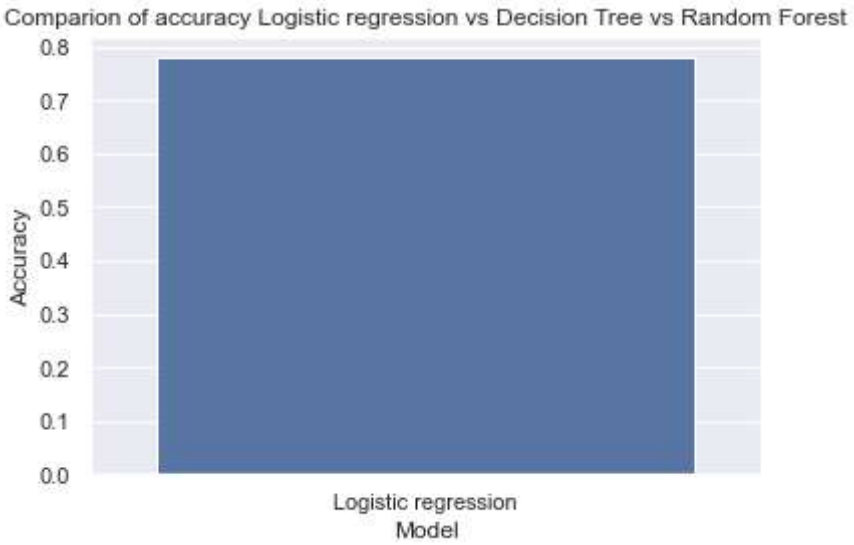
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
lm=LogisticRegression(max_iter=10000)
lm1=lm.fit(x_train,y_train)
```

```
yhat=lm1.predict(x_test)
acc_scores=dict()
acc_scores["Logistic regression"] = accuracy_score(y_test,yhat)
print("Classification report - Logistic regression ")
print("============================================")
print(classification_report(y_test,yhat))
print("============================================")
print("                                            ")
titles_options=[("Confusion matrix without normalization", None),("Normalized confusion
for title, normalize in titles_options:
    disp=plot_confusion_matrix(lm1, x_test, y_test,display_labels=(0,1),cmap=plt.cm.Blu
    disp.ax_.set_title(title)
    print(title)
    print(disp.confusion_matrix)
    print("****************************************")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-8b6007394866> in <module>
      2 from sklearn.metrics import accuracy_score,classification_report
      3 lm=LogisticRegression(max_iter=10000)
----> 4 lm1=lm.fit(x_train,y_train)
      5 yhat=lm1.predict(x_test)
      6 acc_scores=dict()

NameError: name 'x_train' is not defined
```

In [74]:

```
from sklearn.metrics import accuracy_score
plt.figure()
accuracy_score_df = pd.DataFrame()
accuracy_score_df["Model"] = acc_scores.keys()
accuracy_score_df["Accuracy"] = acc_scores.values()
ax = sns.barplot(x="Model",y="Accuracy",data=accuracy_score_df)
tx = ax.set_title("Comparion of accuracy Logistic regression vs Decision Tree vs Random
plt.show()
```



In [87]:

```
import numpy as np
input_data=[2,197,70,45,543,30.5,0.158,53]
input_data_as_numpyarray=np.asarray(input_data)
input_datareshaped=input_data_as_numpyarray.reshape(1,-1)
print(input_datareshaped)
```

```
std_data=sscaler.transform(input_datareshaped)
print(std_data)
prediction=lm.predict(std_data)
print(prediction)
```

```
[[2.00e+00 1.97e+02 7.00e+01 4.50e+01 5.43e+02 3.05e+01 1.58e-01 5.30e+01]]
[[ 3.09130814e-01  2.38188392e+00  4.62452528e-02  1.53455054e+00
   4.02192191e+00 -1.89436889e-01 -4.92104758e+00  4.92305644e+01]]
[1]
```

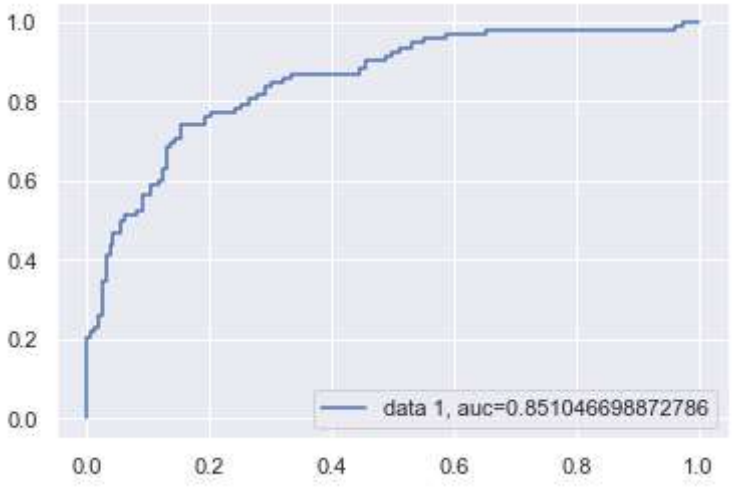
In [105…

```
print("Accuracy:",metrics.accuracy_score(y_test, yhat))
print("Precision:",metrics.precision_score(y_test, yhat))
print("Recall:",metrics.recall_score(y_test, yhat))
```

```
Accuracy: 0.7795275590551181
Precision: 0.7727272727272727
Recall: 0.5543478260869565
```

In [103…

```
from sklearn import metrics
y_pred_proba = lm.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
print("AUC score is",auc)
```




```
AUC score is 0.851046698872786
```

In [5]:

```
import pickle
# Save trained model to file
pickle.dump(lm1, open("Diabetes.pkl", "wb"))
loaded_model = pickle.load(open("Diabetes.pkl", "rb"))
loaded_model.predict(X_test)
loaded_model.score(X_test,y_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-557b05280124> in <module>
      1 import pickle
      2 # Save trained model to file
----> 3 pickle.dump(lm1, open("Diabetes.pkl", "wb"))
      4 loaded_model = pickle.load(open("Diabetes.pkl", "rb"))
```

```
      5 loaded_model.predict(X_test)

NameError: name 'lm1' is not defined
```

In [ ]:

In [ ]: