Capstone Healthcare

November 7, 2022

DESCRIPTION NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. • The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. • Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description Pregnancies Number of times pregnant Glucose Plasma glucose concentration in an oral glucose tolerance test BloodPressure Diastolic blood pressure (mm Hg) SkinThickness Triceps skinfold thickness (mm) Insulin Two hour serum insulin BMI Body Mass Index DiabetesPedigreeFunction Diabetes pedigree function Age Age in years Outcome Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

[2]:
```
#Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
#loading the data
diabetes = pd.read_csv('healthcare.csv')
diabetes.head()
```

[3]:

	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			
2	8	183	64	0	0	23.3			
3	1	89	66	23	94	28.1			
4	0	137	40	35	168	43.1			

```
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
```

[4]:
```
diabetes.shape
```

[4]:
```
(768, 9)
```

[5]:
```
diabetes.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
```

[6]:
```
#See columns in data
diabetes.columns
```

[6]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

[7]:
```
diabetes.describe()
```

[7]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000   
mean      3.845052  120.894531      69.105469      20.536458   79.799479   
std       3.369578   31.972618      19.355807      15.952218  115.244002   
min       0.000000    0.000000       0.000000       0.000000    0.000000   
25%       1.000000   99.000000      62.000000       0.000000    0.000000   
50%       3.000000  117.000000      72.000000      23.000000   30.500000   
75%       6.000000  140.250000      80.000000      32.000000  127.250000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   
```

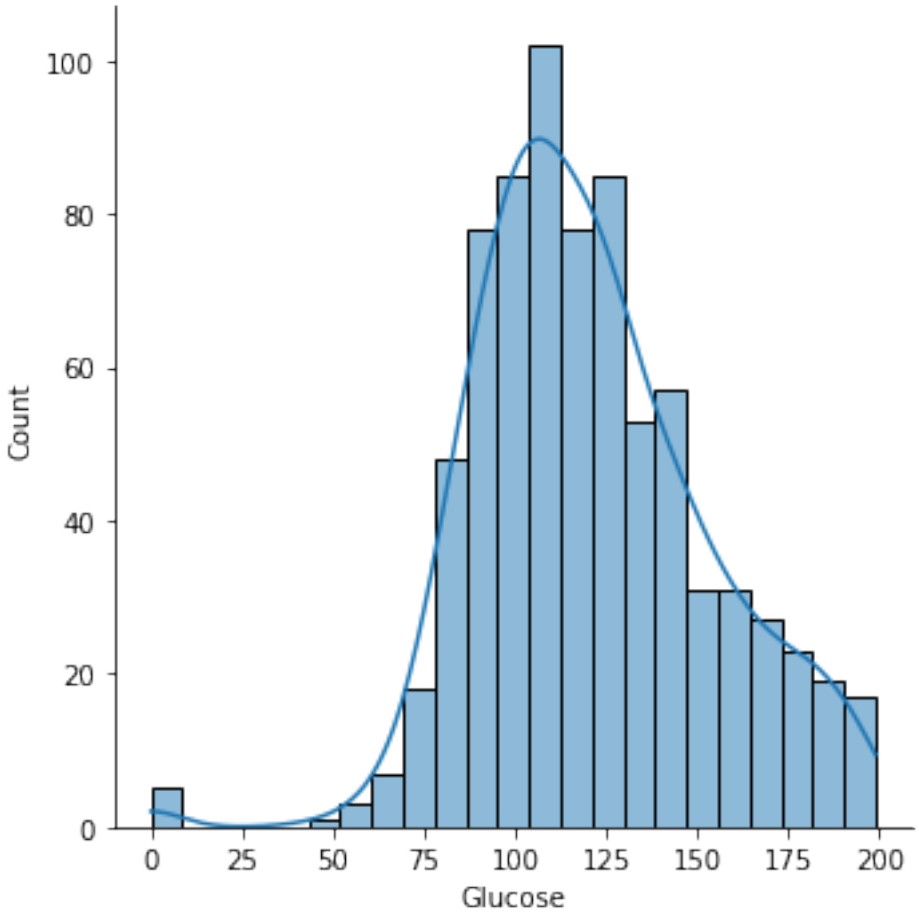
```
                BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[8]:
```
diabetes.isnull().sum()
```

[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

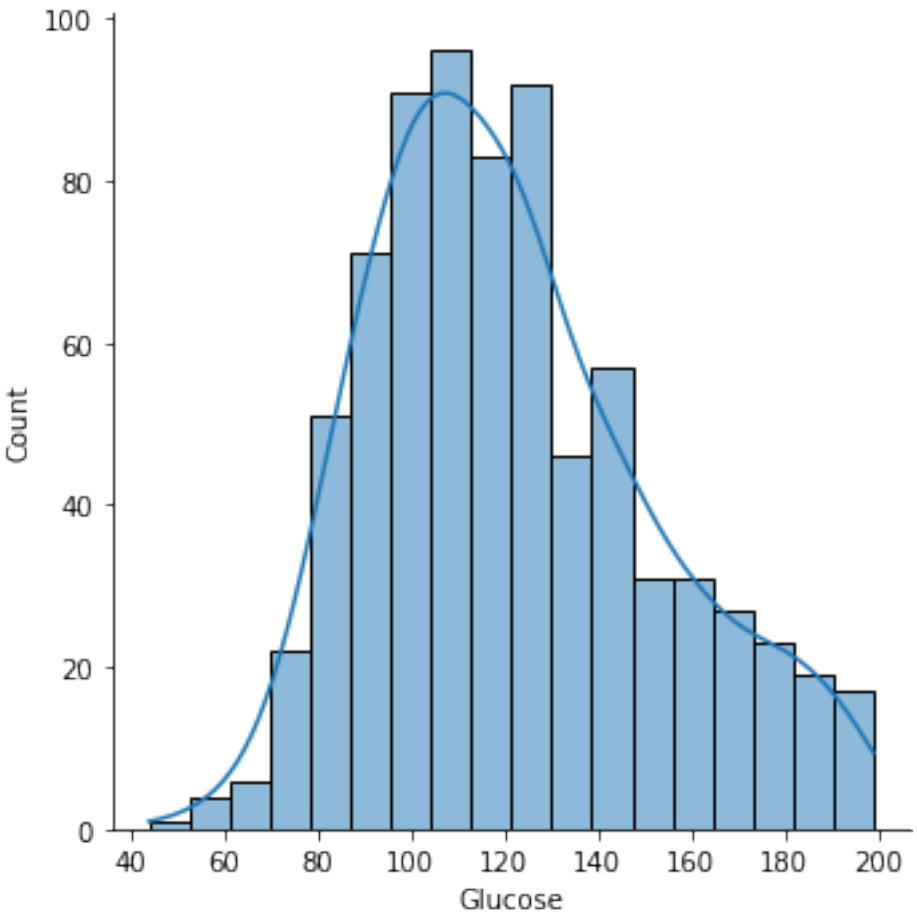
There are no null values per say, however On the columns below, a value of zero does not make sense and thus indicates missing value: Glucose, BloodPressure, SkinThickness, Insulin, BMI

[10]:
```
#Treating missing values in Glucose
sns.displot(diabetes.Glucose, kde = True);
```

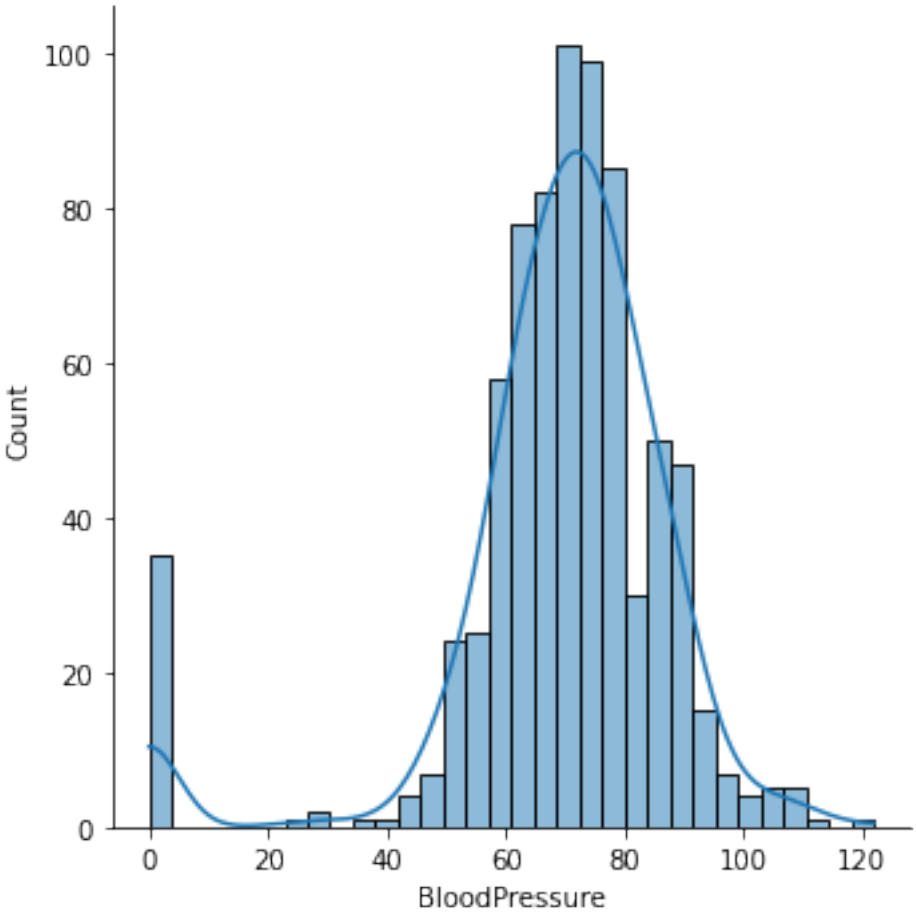


The histogram for Glucose is negativaly skewed, We will replace the zeros with the median.

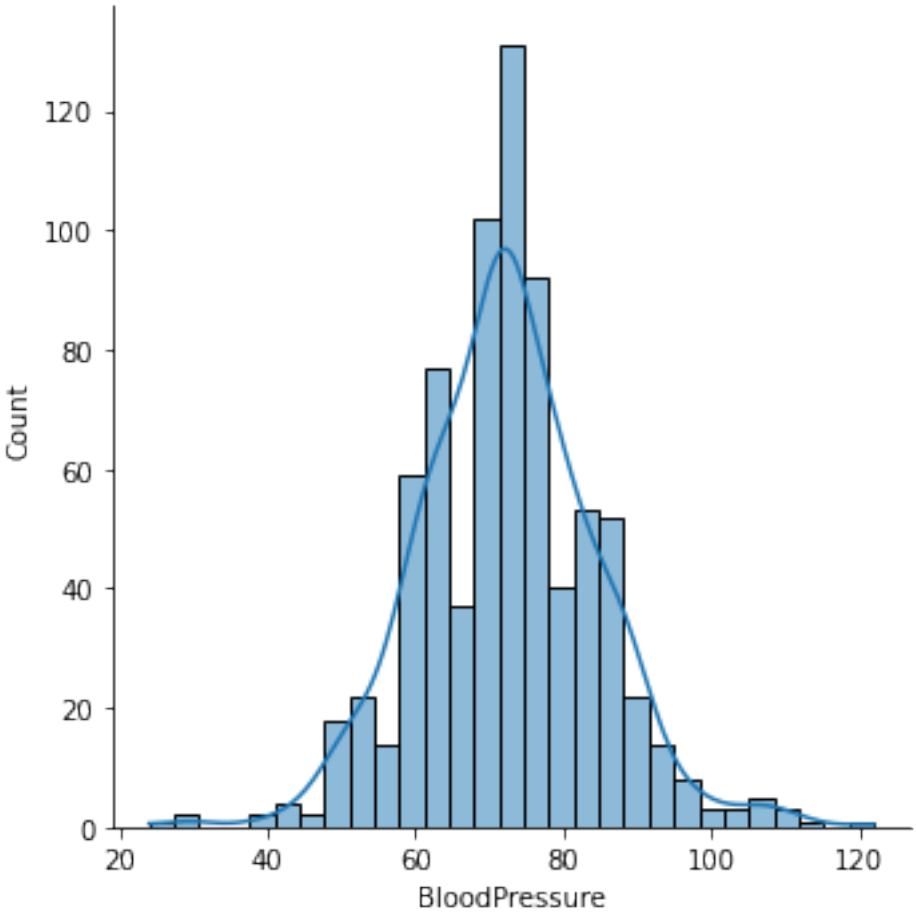
[12]:
```
#replacing zeroes by median since histogram is skewed
diabetes['Glucose']=diabetes['Glucose'].replace(0,diabetes['Glucose'].median())
sns.displot(diabetes.Glucose, kde = True);
```



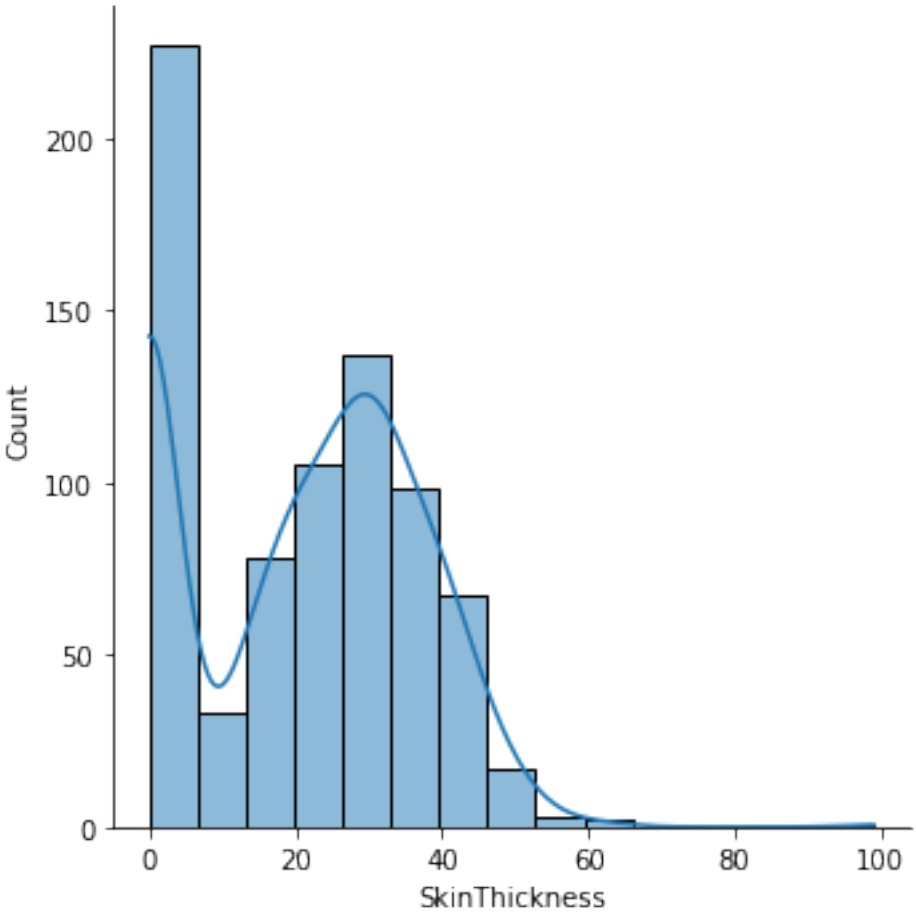
[13]:
```
#Treating missing values in BloodPressure
sns.displot(diabetes.BloodPressure, kde = True);
```



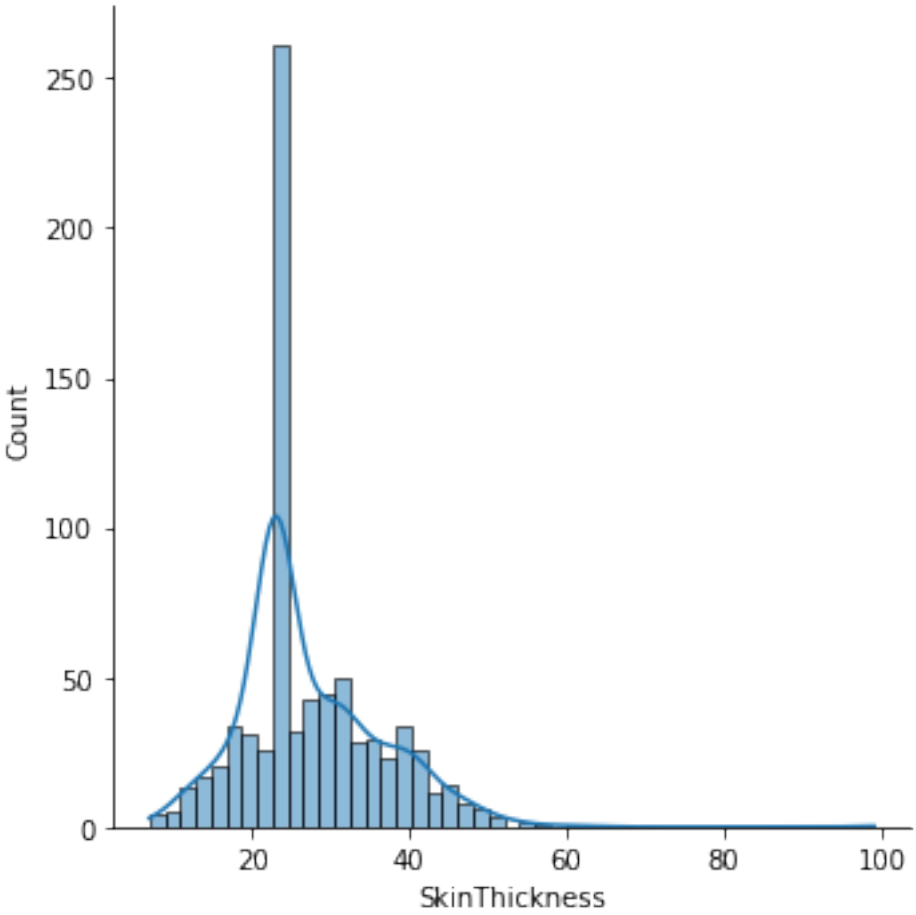
[14]:
```
#replacing zeroes by median since histogram is skewed
diabetes['BloodPressure']=diabetes['BloodPressure'].
 ↪replace(0,diabetes['BloodPressure'].median())
sns.displot(diabetes.BloodPressure, kde = True);
```



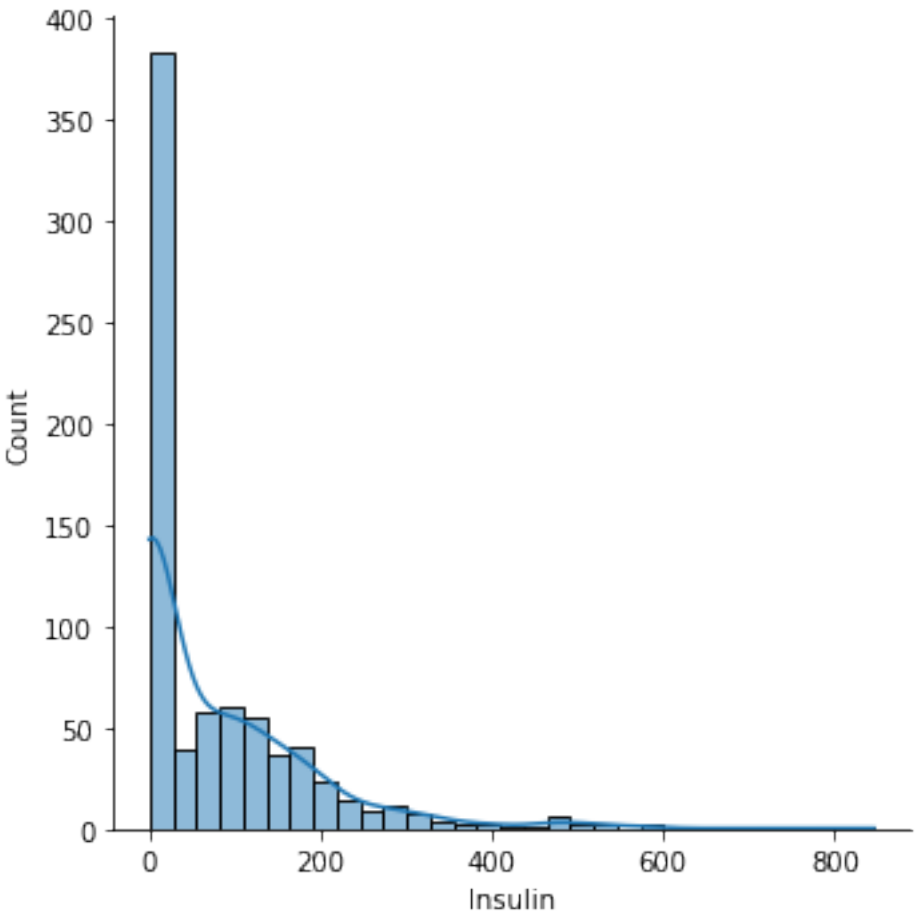
[15]:
```
#Treating missing values in SkinThickness
sns.displot(diabetes.SkinThickness, kde = True);
```



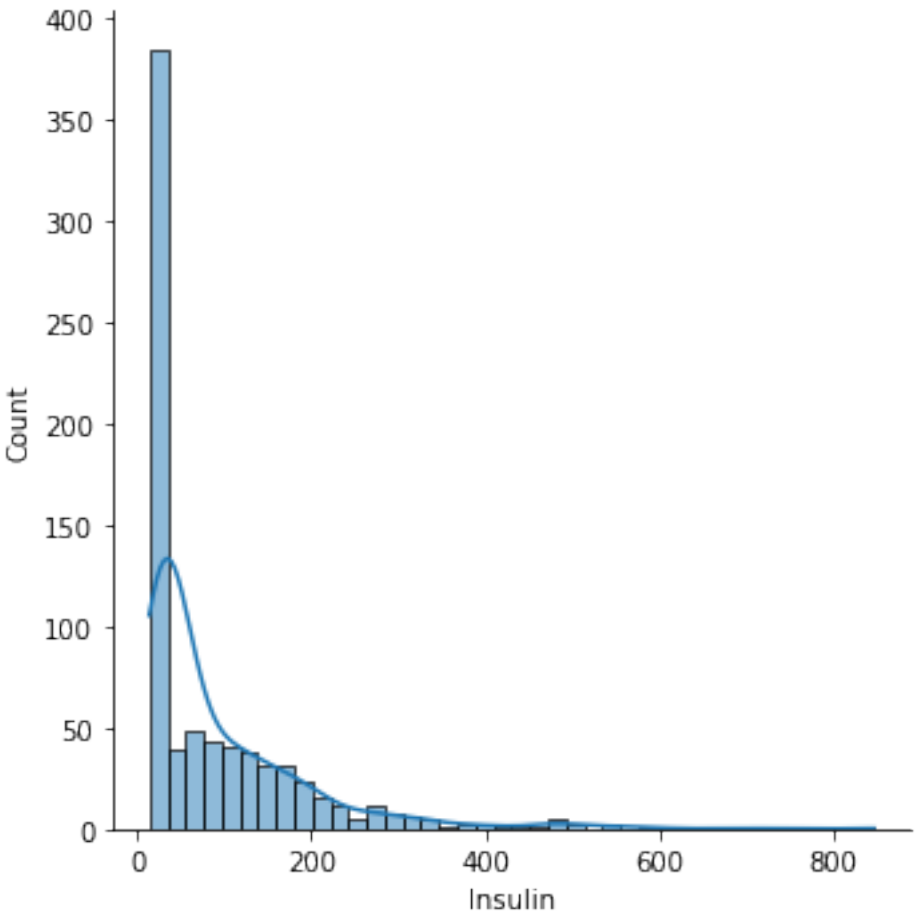
[16]:
```
#replacing zeroes by median since histogram is skewed
diabetes['SkinThickness']=diabetes['SkinThickness'].
 ↪replace(0,diabetes['SkinThickness'].median())
sns.displot(diabetes.SkinThickness, kde = True);
```



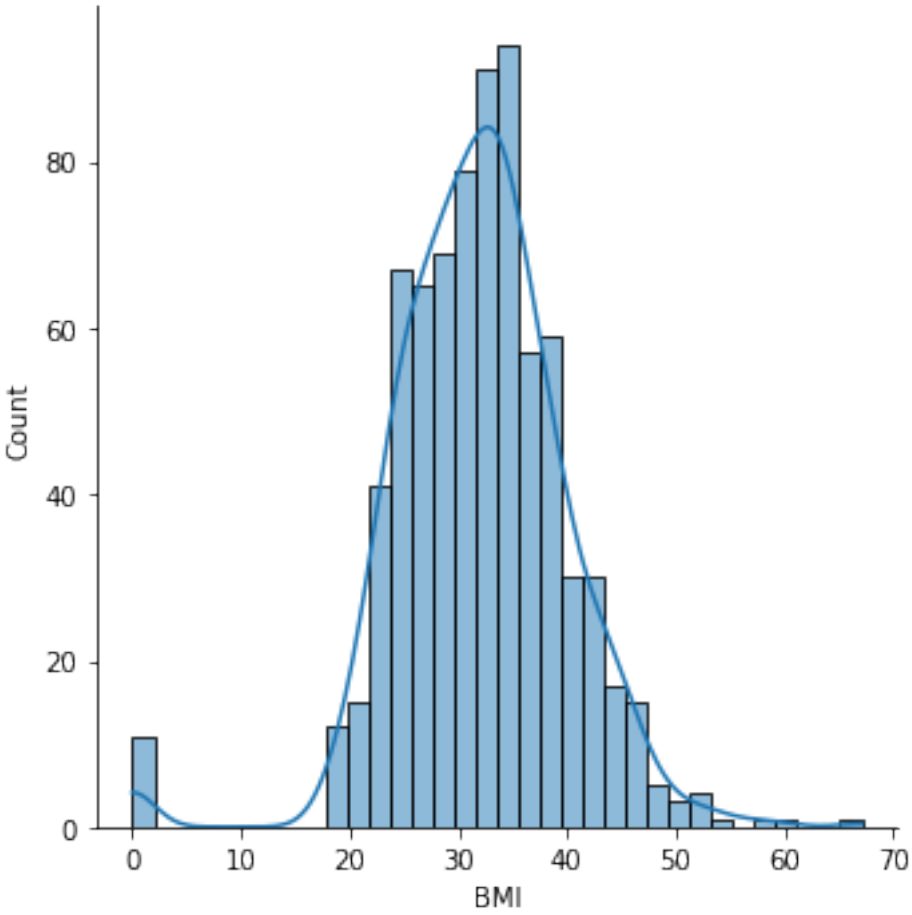
[17]:
```
#Treating missing values in Insulin
sns.displot(diabetes.Insulin, kde = True);
```



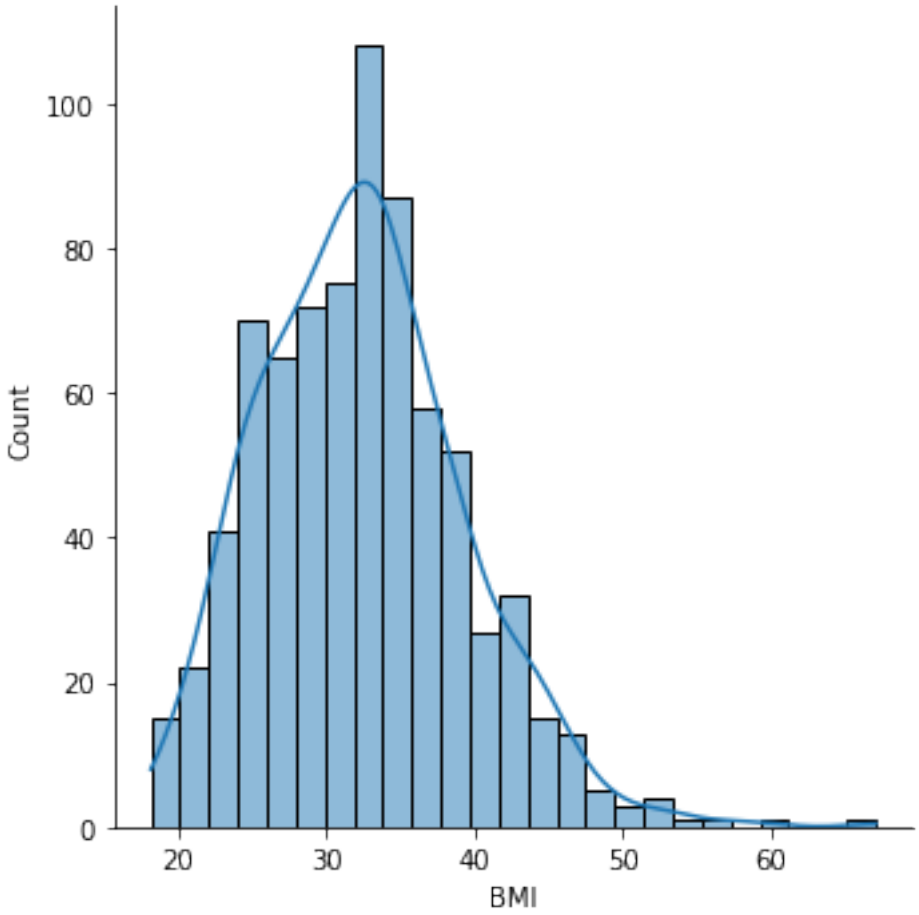
[20]:
```
#replacing zeroes by median since histogram is skewed
diabetes['Insulin']=diabetes['Insulin'].replace(0,diabetes['Insulin'].median())
sns.displot(diabetes.Insulin, kde = True);
```



[21]:
```
#Treating missing values in BMI
sns.displot(diabetes.BMI, kde = True);
```



[24]:
```
#replacing zeroes by median since histogram is skewed
diabetes['BMI']=diabetes['BMI'].replace(0,diabetes['BMI'].median())
sns.displot(diabetes.BMI, kde = True);
```



[26]:
```
diabetes.describe()
```

[26]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.656250	72.386719	27.334635	94.652344	
std	3.369578	30.438286	12.096642	9.229014	105.547598	
min	0.000000	44.000000	24.000000	7.000000	14.000000	
25%	1.000000	99.750000	64.000000	23.000000	30.500000	
50%	3.000000	117.000000	72.000000	23.000000	31.250000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	

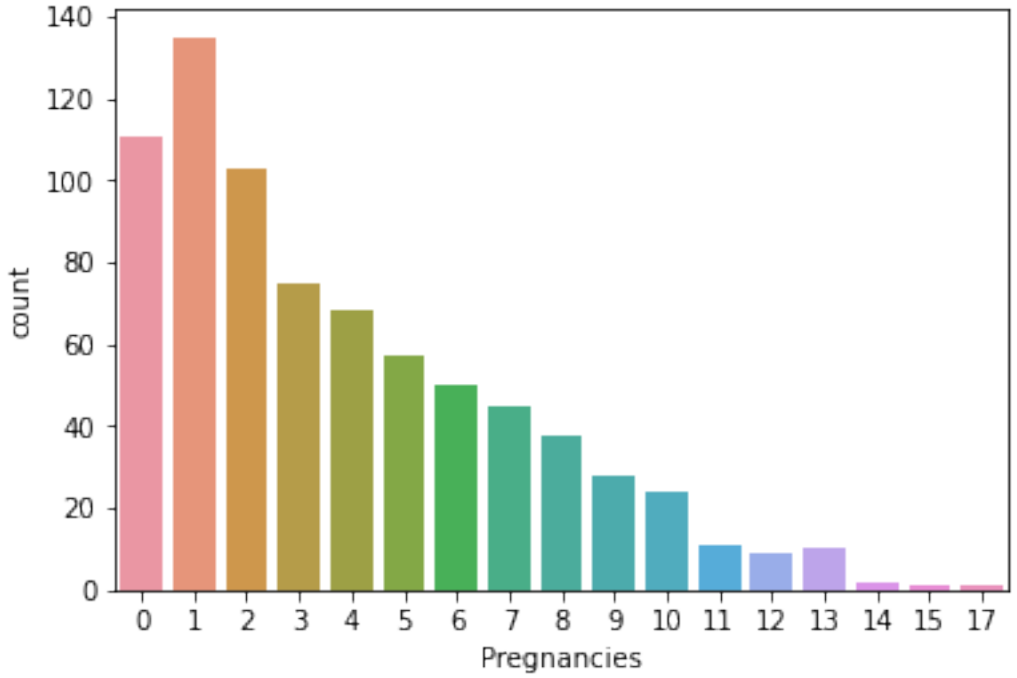
	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	32.450911	0.471876	33.240885	0.348958
std	6.875366	0.331329	11.760232	0.476951
min	18.200000	0.078000	21.000000	0.000000
25%	27.500000	0.243750	24.000000	0.000000

```
50%     32.000000                 0.372500   29.000000    0.000000
75%     36.600000                 0.626250   41.000000    1.000000
max     67.100000                 2.420000   81.000000    1.000000
```

VISUALIZATION

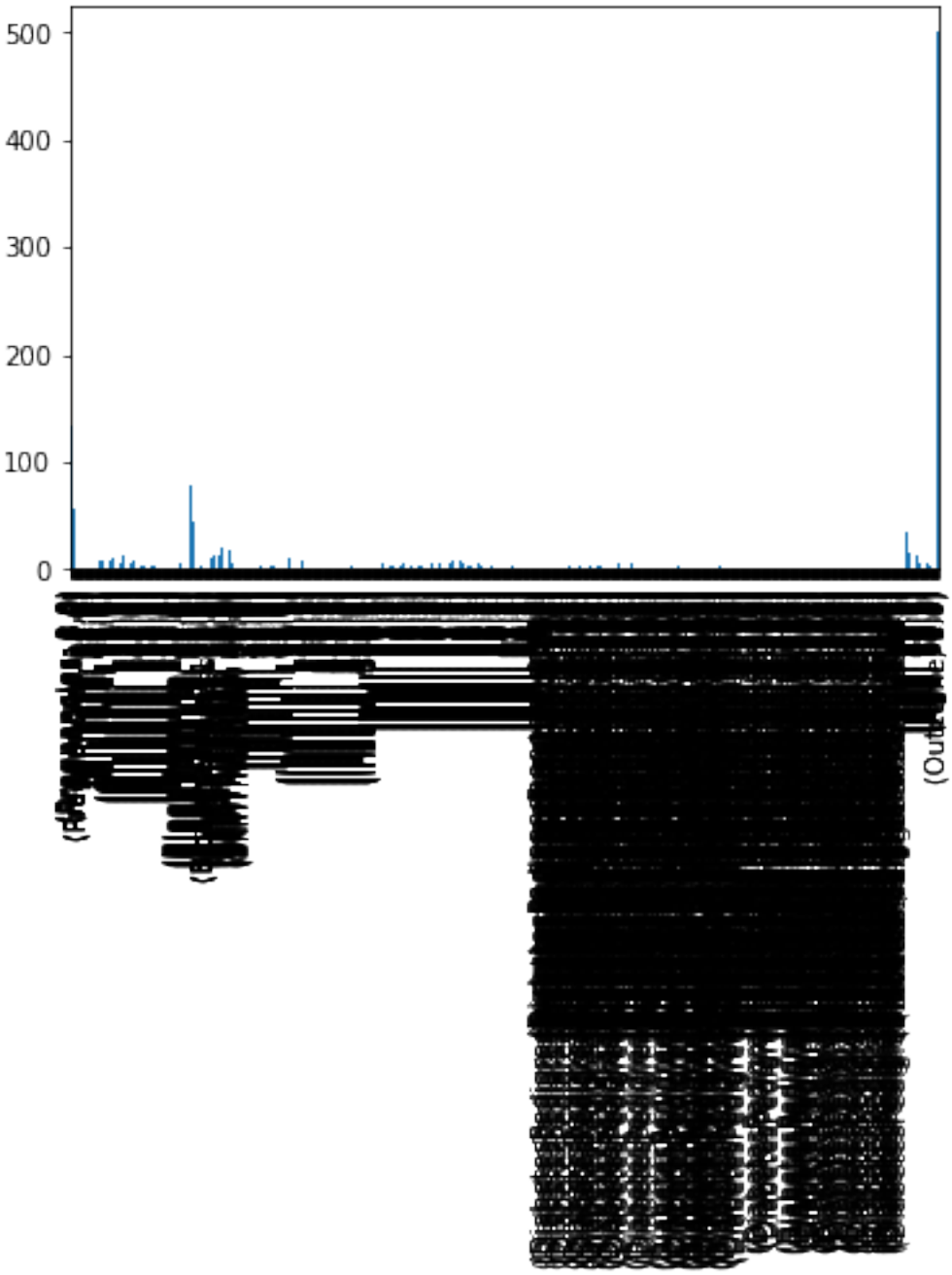
[9]:
```
sns.countplot(diabetes['Pregnancies'])
```

[9]:
```
<AxesSubplot:xlabel='Pregnancies', ylabel='count'>
```



[28]:
```
#Create a count (frequency) plot describing the data types and the count of␣
↪variables.
diabetes.apply(lambda x: x.value_counts()).T.stack().plot(kind='bar')
```

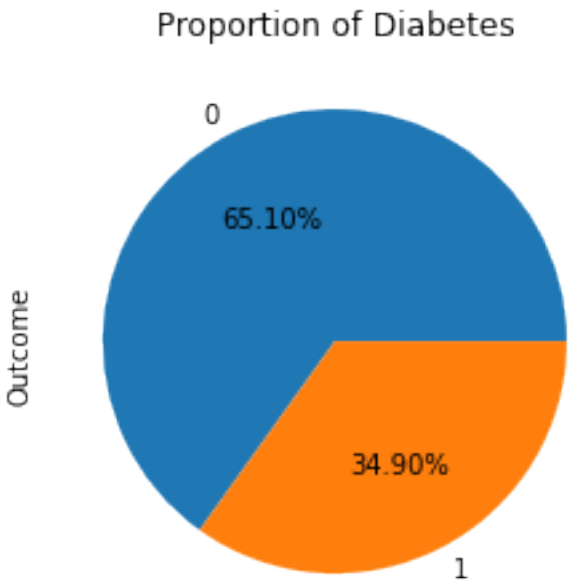
[28]:
```
<AxesSubplot:>
```



DATA EXPLORATION

[29]:
```
#Visualisation of target column relative frequency
diabetes['Outcome'].value_counts().plot(kind='pie',autopct='%1.2f%%',␣
↪title='Proportion of Diabetes')
```

[29]: <AxesSubplot:title={'center':'Proportion of Diabetes'}, ylabel='Outcome'>

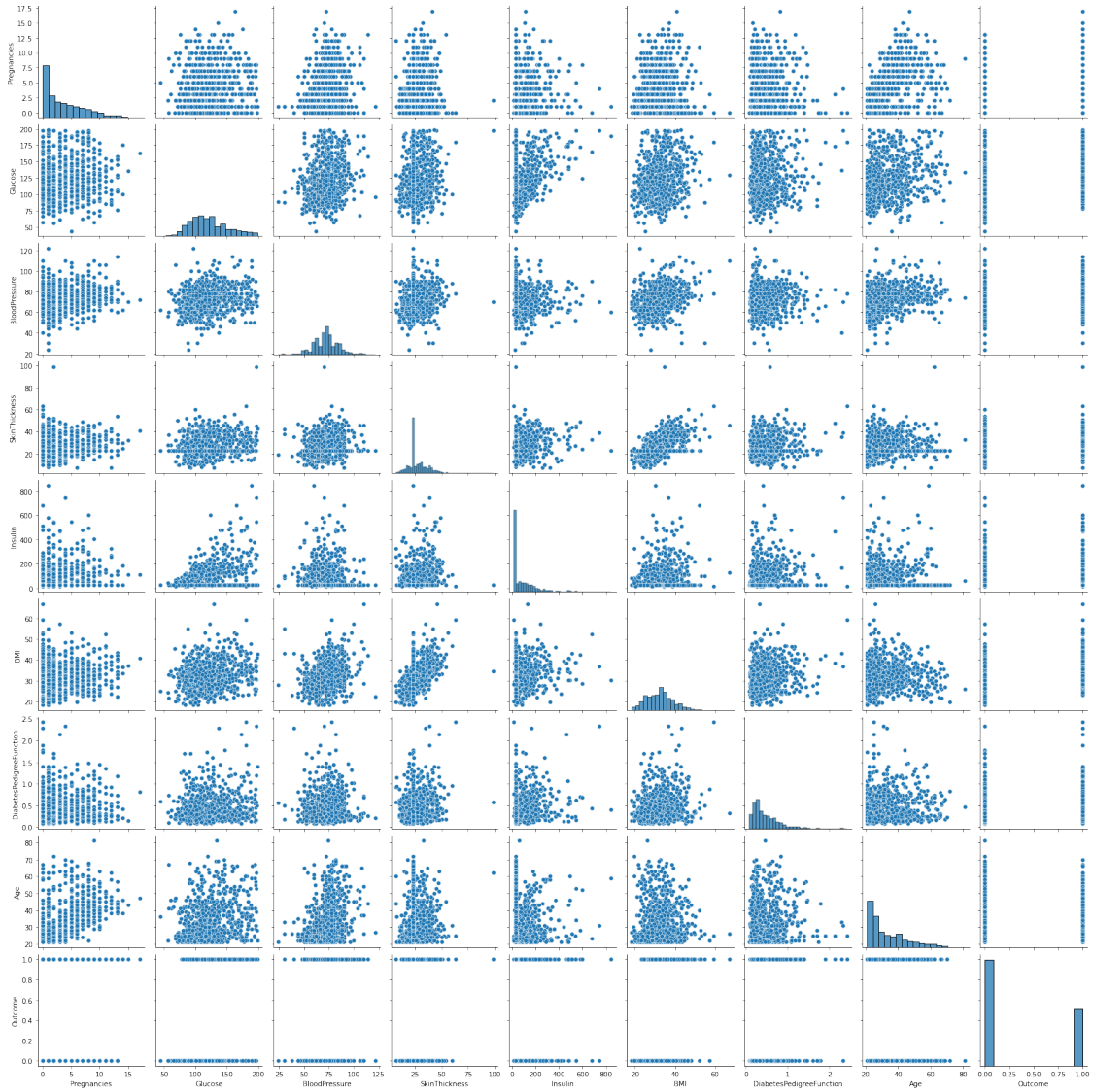


The dataset is not highly imbalanced but it is not balanced either. We will balance the dataset before modeling.

[30]:
```
#Pairplot, scatterplots of all the variables
plt.figure(figsize=(10,10))
sns.pairplot(diabetes)
```

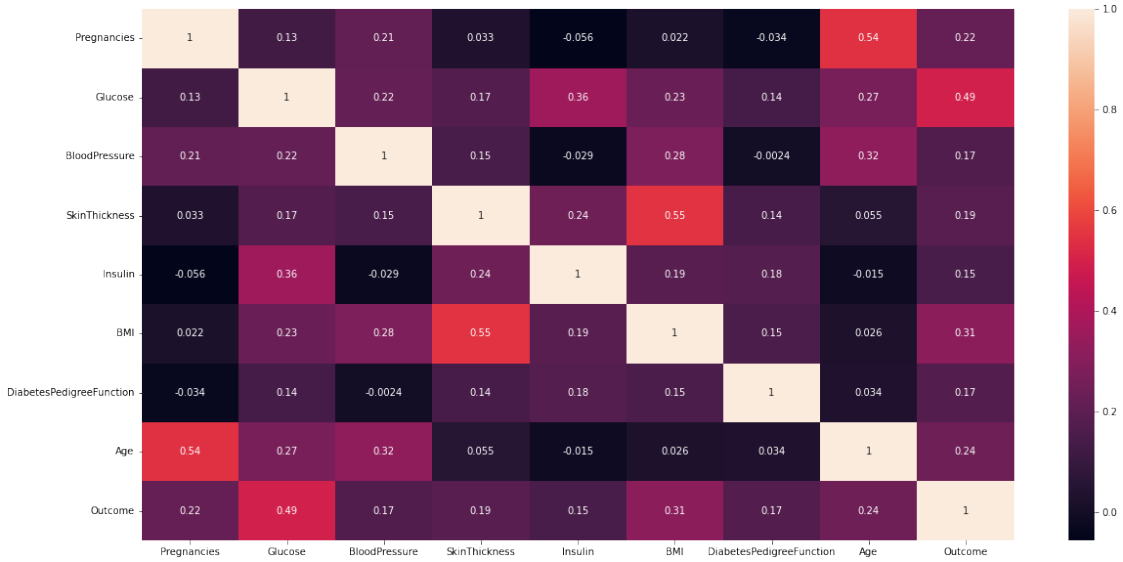
[30]:
```
<seaborn.axisgrid.PairGrid at 0x7ff3ed30b790>

<Figure size 720x720 with 0 Axes>
```



[31]:
```
#Heatmap (negative values are darker, positive values are lighter)
plt.figure(figsize=(20,10))
sns.heatmap(diabetes.corr(),annot=True)
```

[31]: <AxesSubplot:>



[63]:
```
diabetes.corr()
```

[63]:

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.128213	0.208615	0.032568
Glucose	0.128213	1.000000	0.218937	0.172143
BloodPressure	0.208615	0.218937	1.000000	0.147809
SkinThickness	0.032568	0.172143	0.147809	1.000000
Insulin	-0.055697	0.357573	-0.028721	0.238188
BMI	0.021546	0.231400	0.281132	0.546951
DiabetesPedigreeFunction	-0.033523	0.137327	-0.002378	0.142977
Age	0.544341	0.266909	0.324915	0.054514
Outcome	0.221898	0.492782	0.165723	0.189065

	Insulin	BMI	DiabetesPedigreeFunction
Pregnancies	-0.055697	0.021546	-0.033523
Glucose	0.357573	0.231400	0.137327
BloodPressure	-0.028721	0.281132	-0.002378
SkinThickness	0.238188	0.546951	0.142977
Insulin	1.000000	0.189022	0.178029
BMI	0.189022	1.000000	0.153506
DiabetesPedigreeFunction	0.178029	0.153506	1.000000
Age	-0.015413	0.025744	0.033561
Outcome	0.148457	0.312249	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.266909	0.492782
BloodPressure	0.324915	0.165723

```
SkinThickness              0.054514  0.189065
Insulin                   -0.015413  0.148457
BMI                        0.025744  0.312249
DiabetesPedigreeFunction   0.033561  0.173844
Age                        1.000000  0.238356
Outcome                    0.238356  1.000000
```

There is no multicollinearity.

DATA MODELING

[47]:
```
#Importing Logistic Regression model
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression(random_state=0, solver="liblinear")
```

[48]:
```
#Importing "train_test-split" function to test the model
from sklearn.model_selection import train_test_split
```

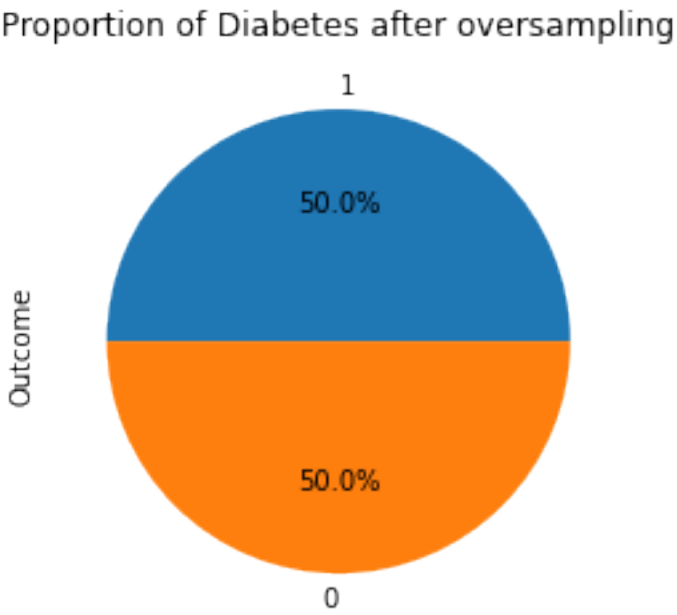
[49]:
```
#Splitting the data
X=diabetes.drop(['Outcome'],axis=1)
y=diabetes['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

[50]:
```
#Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[51]:
```
#balancing the dataset
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X_train, y_train = oversample.fit_resample(X_train, y_train)

y_train.value_counts().plot(kind='pie',autopct="%1.1f%%", title='Proportion of␣
 ↪Diabetes after oversampling')
```

[51]:
```
<AxesSubplot:title={'center':'Proportion of Diabetes after oversampling'},
ylabel='Outcome'>
```



[52]:
```
#Fit the model in train and test data
lr.fit(X_train,y_train).score(X_train,y_train)
```

[52]: 0.769774011299435

[69]:
```
#Now fitting the model in test set
y_pred=lr.predict(X_test)
y_pred_prob=lr.predict_proba(X_test)
```

[70]:
```
#Confusion matrix
from sklearn.metrics import confusion_matrix, recall_score,␣
 ↪precision_score,f1_score, roc_auc_score,roc_curve
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy score:",accuracy)
precision = metrics.precision_score(y_test, y_pred)
print("Precision score:",precision)
recall = metrics.recall_score(y_test, y_pred)
print("Recall score:",recall)
```

```
[[110  36]
 [ 33  52]]
Accuracy score: 0.7012987012987013
Precision score: 0.5909090909090909
Recall score: 0.611764705882353
```

The model has a 70 % accuracy score, an 59 % precision score, and an 61 % recall score, indicating that it doesn't really work effectively.

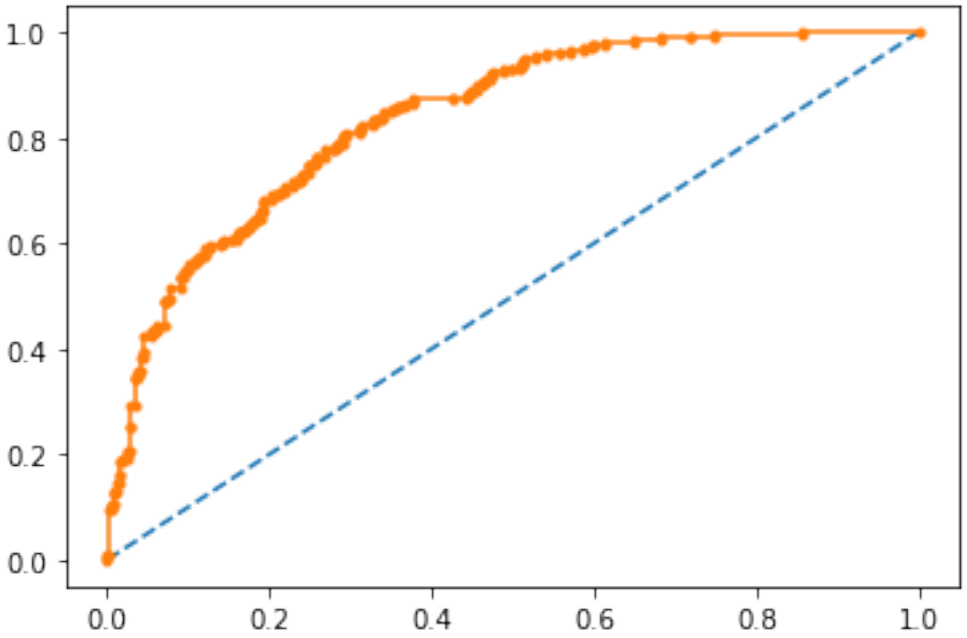
ROC CURVE

[73]:
```
#Preparing ROC Curve (Receiver Operating Characteristics Curve)

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

```
AUC: 0.839
```

[73]:
```
[<matplotlib.lines.Line2D at 0x7ff3dfe31e90>]
```



[58]:
```
#Feature scaling on X
X = sc.fit_transform(X)
```

Model Comparison

[59]:
```
# Compare classification algorithms
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

[77]:
```
classification_models = []
classification_models.append(('Logistic Regression',␣
 ↪LogisticRegression(solver="liblinear")))
classification_models.append(('K Nearest Neighbor',␣
 ↪KNeighborsClassifier(n_neighbors=7, metric="minkowski",p=2)))
classification_models.append(('Kernel SVM', SVC(kernel = 'rbf',gamma='auto')))
classification_models.append(('Naive Bayes', GaussianNB()))
classification_models.append(('Decision Tree',␣
 ↪DecisionTreeClassifier(max_depth=5)))
classification_models.append(('Random Forest',␣
 ↪RandomForestClassifier(n_estimators=100,criterion="entropy")))
```

[78]:
```
for name, model in classification_models:
    kfold = KFold(n_splits=10, random_state=(7), shuffle=(True))
    result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
    print("%s: Mean Accuracy = %.2f%% - SD Accuracy = %.2f%%" % (name, result.
 ↪mean()*100,result.std()*100))
```

```
Logistic Regression: Mean Accuracy = 76.57% - SD Accuracy = 4.80%
K Nearest Neighbor: Mean Accuracy = 75.92% - SD Accuracy = 5.18%
Kernel SVM: Mean Accuracy = 76.30% - SD Accuracy = 6.42%
Naive Bayes: Mean Accuracy = 74.88% - SD Accuracy = 3.39%
Decision Tree: Mean Accuracy = 73.19% - SD Accuracy = 4.06%
Random Forest: Mean Accuracy = 76.83% - SD Accuracy = 4.92%
```

From the results we can see that the most accurate model is Random Forest, followed by Logistic Regression and Kernel SVM.

[ ]: