CapstoneProject1

August 16, 2020

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import StandardScaler, normalize
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

[13]:
```
df_db1 = pd.read_csv('health care diabetes.csv')
```

[14]:
```
df_db1.head(10)
```

[14]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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
5            5      116             74              0        0  25.6
6            3       78             50             32       88  31.0
7           10      115              0              0        0  35.3
8            2      197             70             45      543  30.5
9            8      125             96              0        0   0.0

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
9                     0.232   54        1
```

[15]:
```
df_db1.shape
```

[15]:
```
(768, 9)
```

[16]:
```
df_db1.dtypes
```

[16]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

[17]:
```
df_db1.describe()
```

[17]:
```
       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
```

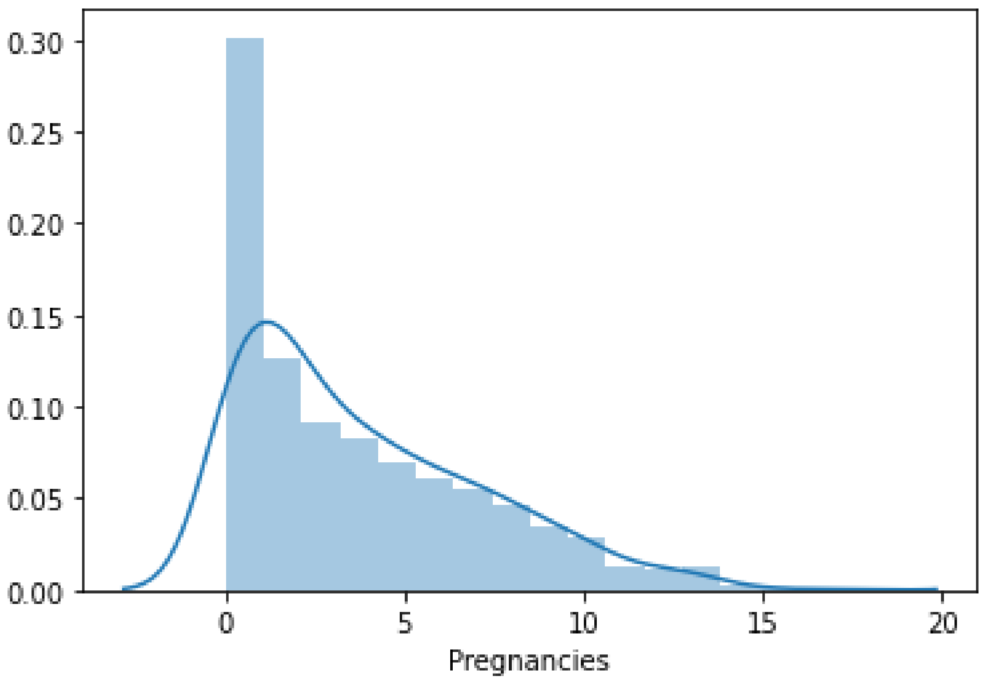
[18]:
```
df_db1.info()
```

```
<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
```

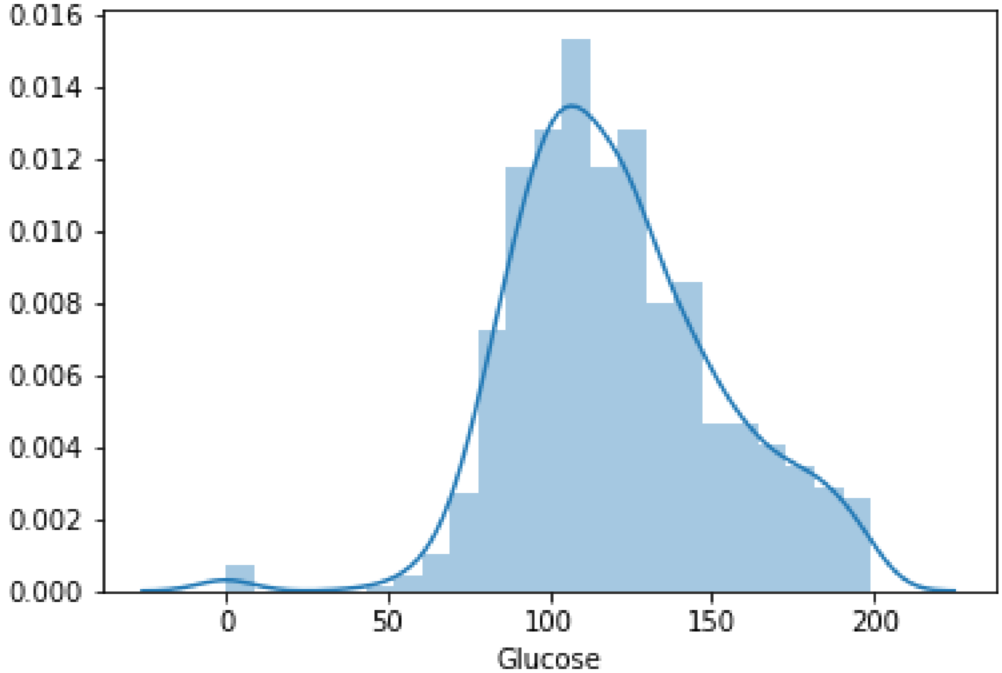
[19]:
```python
sns.distplot(df_db1['Pregnancies'])
```

[19]: <AxesSubplot:xlabel='Pregnancies'>



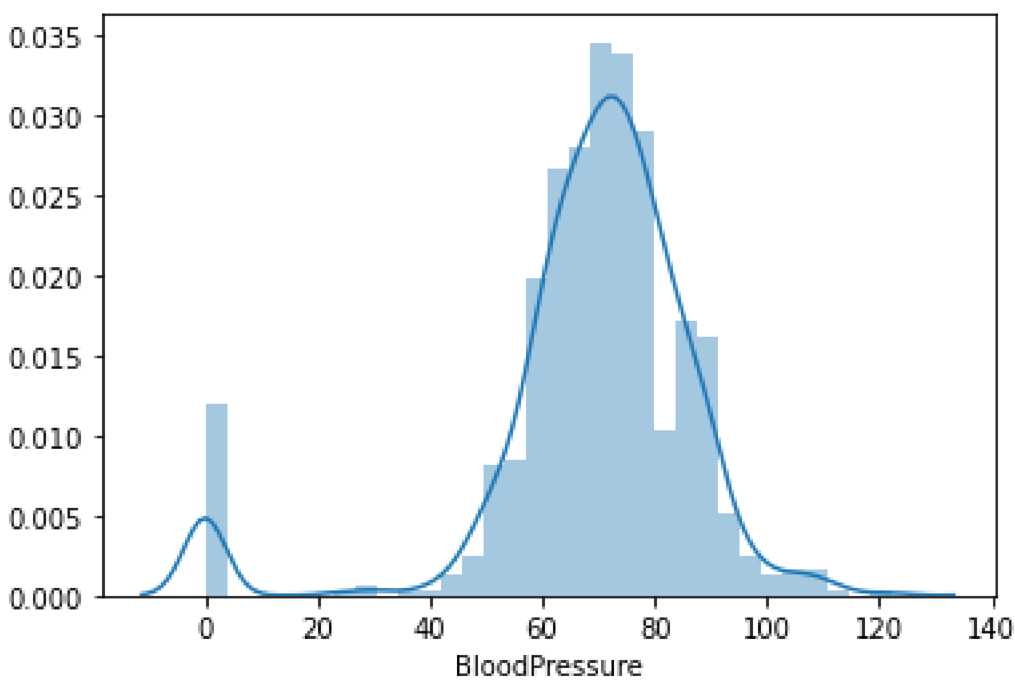
[20]:
```python
sns.distplot(df_db1['Glucose'])
```

[20]: <AxesSubplot:xlabel='Glucose'>



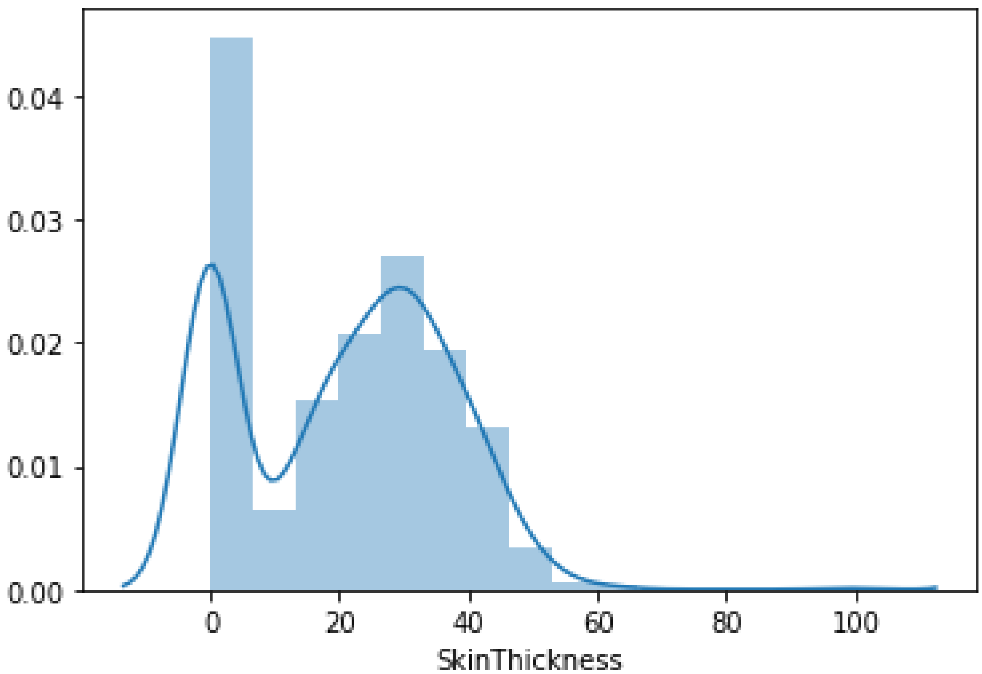

```
[21]: sns.distplot(df_db1['BloodPressure'])
```

[21]: <AxesSubplot:xlabel='BloodPressure'>



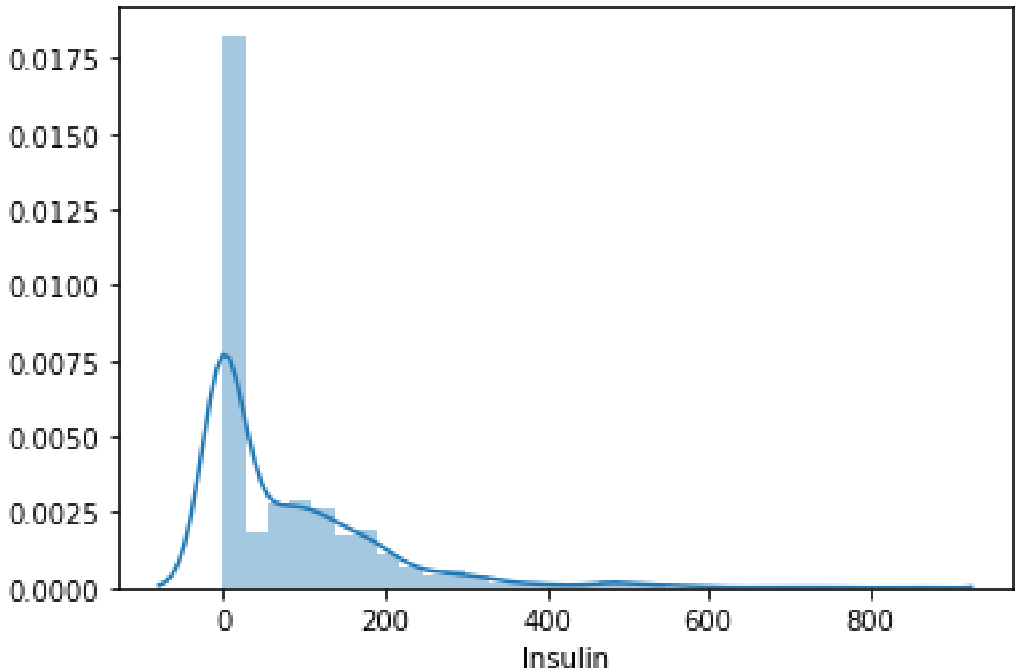
[22]:
```
sns.distplot(df_db1['SkinThickness'])
```

[22]: <AxesSubplot:xlabel='SkinThickness'>



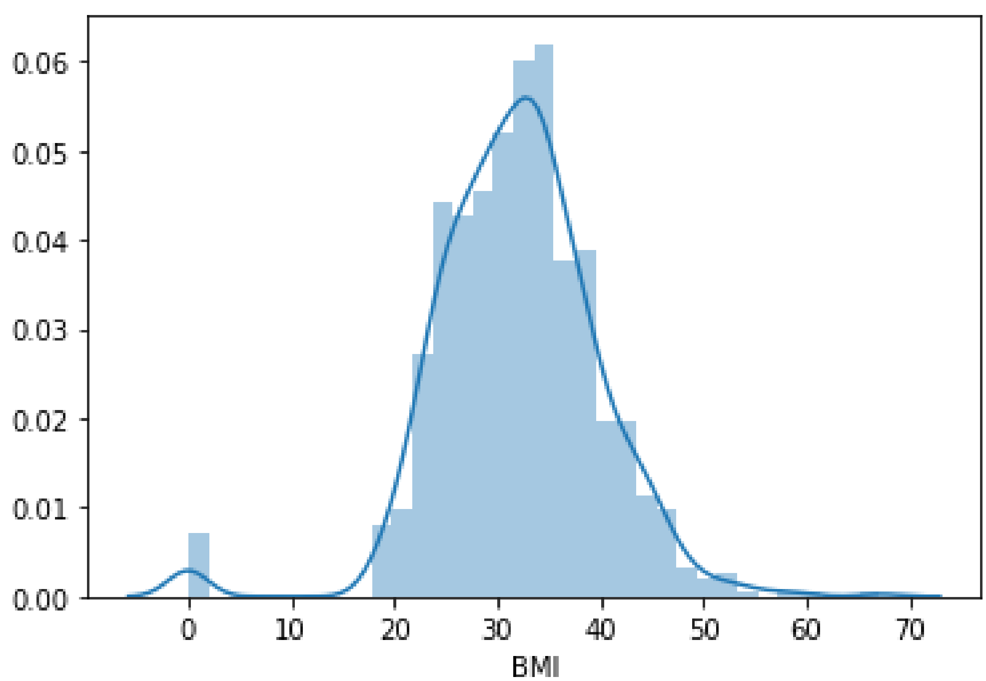
[23]:
```
sns.distplot(df_db1['Insulin'])
```

[23]: <AxesSubplot:xlabel='Insulin'>



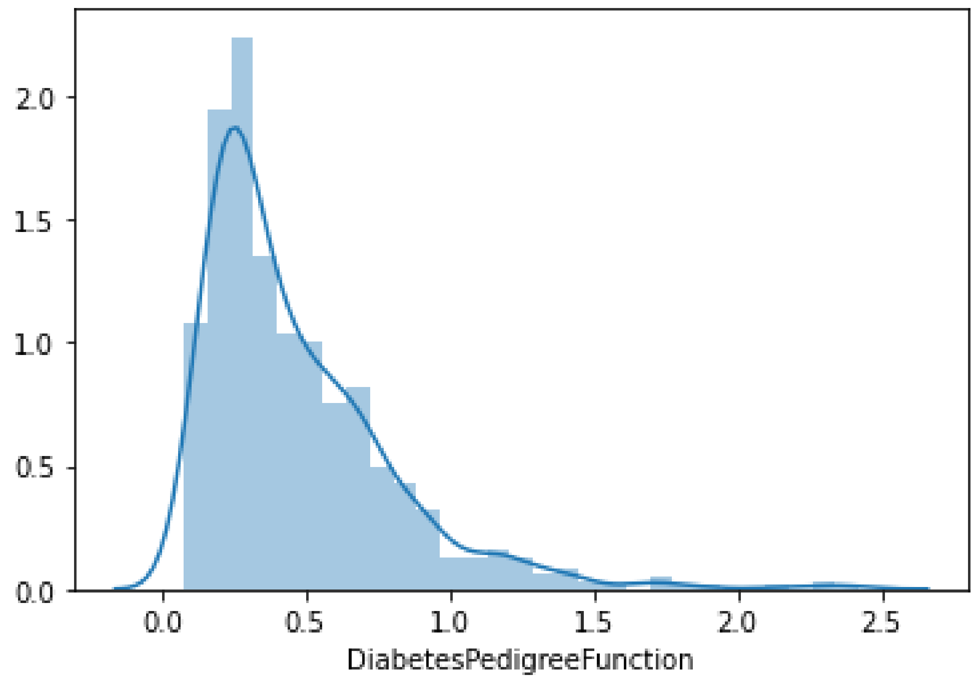
[24]:
```
sns.distplot(df_db1['BMI'])
```

[24]: <AxesSubplot:xlabel='BMI'>



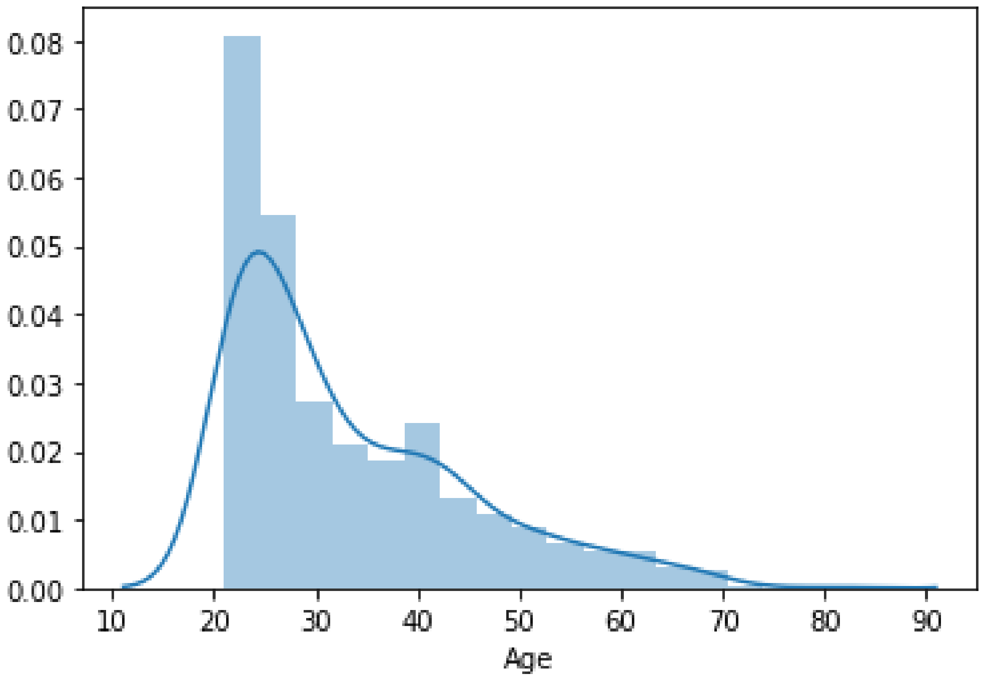
[25]:
```
sns.distplot(df_db1['DiabetesPedigreeFunction'])
```

[25]: <AxesSubplot:xlabel='DiabetesPedigreeFunction'>



[26]:
```
sns.distplot(df_db1['Age'])
```

[26]: <AxesSubplot:xlabel='Age'>



[28]:
```
print("Skewness: %f" % df_db1['Pregnancies'].skew())
print("Kurtosis: %f" % df_db1['Pregnancies'].kurt())
```

```
Skewness: 0.901674
Kurtosis: 0.159220
```

[29]:
```
print("Skewness: %f" % df_db1['Glucose'].skew())
print("Kurtosis: %f" % df_db1['Glucose'].kurt())
```

```
Skewness: 0.173754
Kurtosis: 0.640780
```

[30]:
```
print("Skewness: %f" % df_db1['BloodPressure'].skew())
print("Kurtosis: %f" % df_db1['BloodPressure'].kurt())
```

```
Skewness: -1.843608
Kurtosis: 5.180157
```

[31]:
```
print("Skewness: %f" % df_db1['SkinThickness'].skew())
print("Kurtosis: %f" % df_db1['SkinThickness'].kurt())
```

```
Skewness: 0.109372
Kurtosis: -0.520072
```

[32]:
```
print("Skewness: %f" % df_db1['Insulin'].skew())
print("Kurtosis: %f" % df_db1['Insulin'].kurt())
```

```
Skewness: 2.272251
Kurtosis: 7.214260
```

[33]:
```
print("Skewness: %f" % df_db1['BMI'].skew())
print("Kurtosis: %f" % df_db1['BMI'].kurt())
```

```
Skewness: -0.428982
Kurtosis: 3.290443
```

[34]:
```
print("Skewness: %f" % df_db1['DiabetesPedigreeFunction'].skew())
print("Kurtosis: %f" % df_db1['DiabetesPedigreeFunction'].kurt())
```

```
Skewness: 1.919911
Kurtosis: 5.594954
```

[35]:
```
print("Skewness: %f" % df_db1['Age'].skew())
print("Kurtosis: %f" % df_db1['Age'].kurt())
```

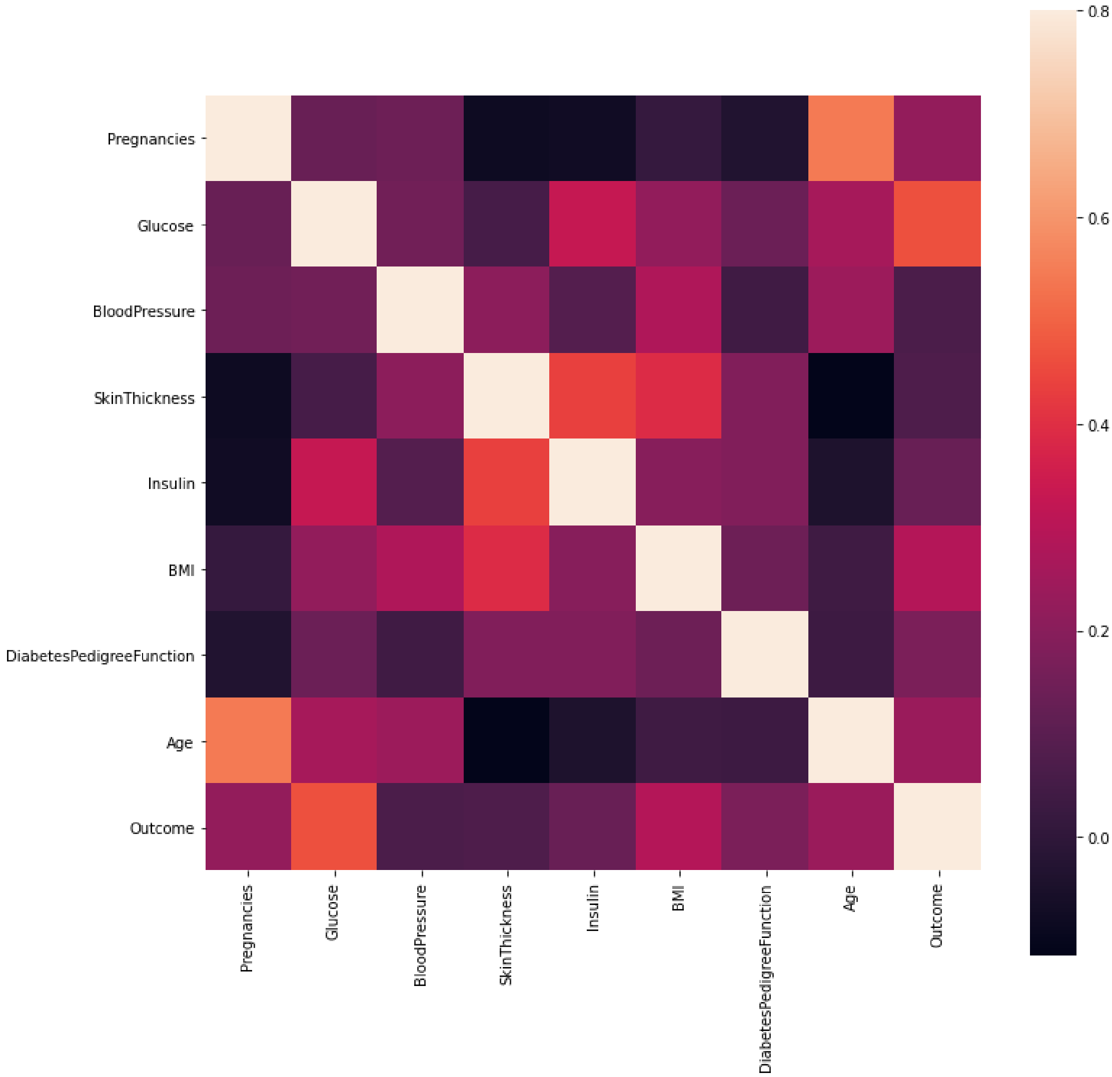
```
Skewness: 1.129597
Kurtosis: 0.643159
```

[37]:
```
# First, let's count the number of null values
total = df_db.isnull().sum().sort_values(ascending=False)
# Then, let's calculate the percentage of missing data per feature
percent = (df_db1.isnull().sum()/df_db.isnull().count()).
↪sort_values(ascending=False)
# Finally, let's concatenate Total and Percent into another dataframe
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

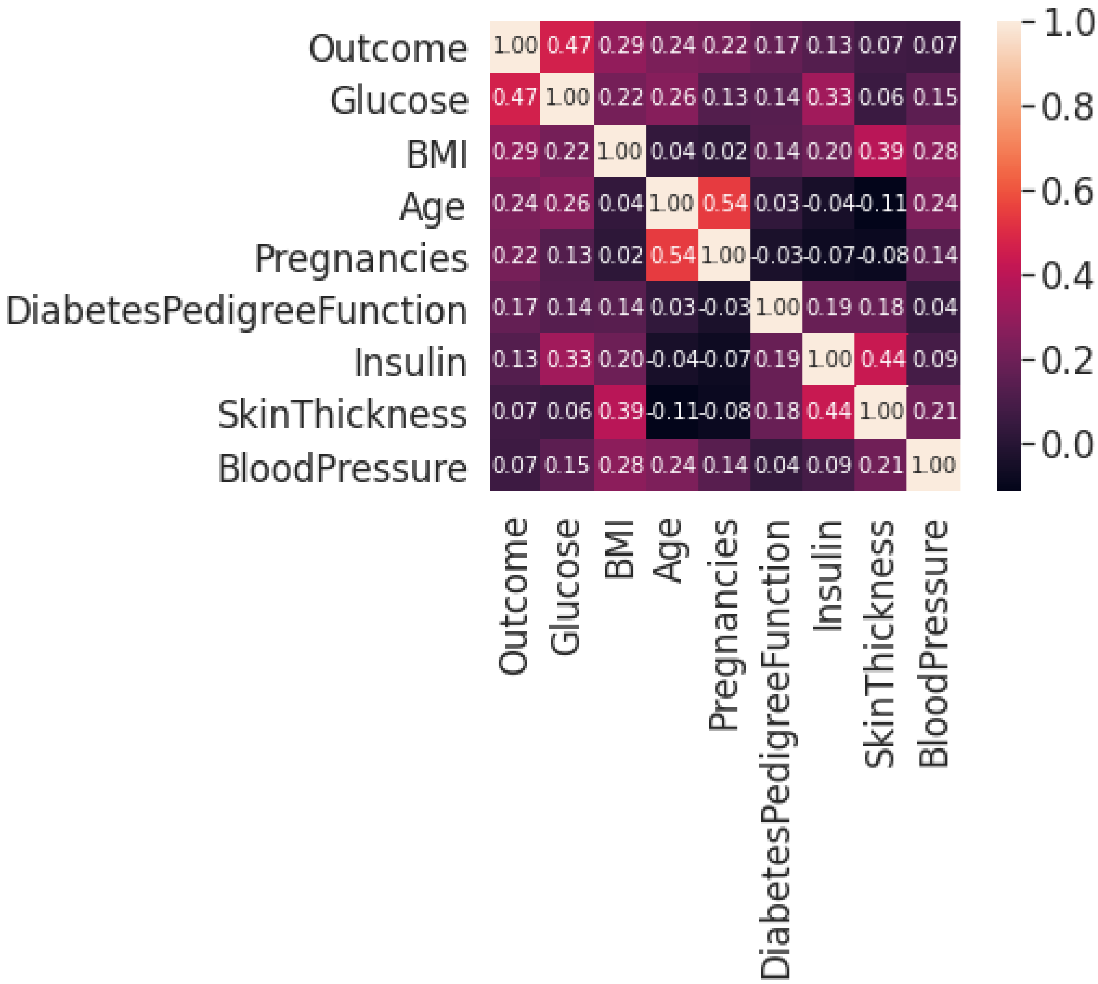
[37]:

	Total	Percent
Outcome	0	0.0
Age	0	0.0
DiabetesPedigreeFunction	0	0.0
BMI	0	0.0
Insulin	0	0.0
SkinThickness	0	0.0
BloodPressure	0	0.0
Glucose	0	0.0
Pregnancies	0	0.0

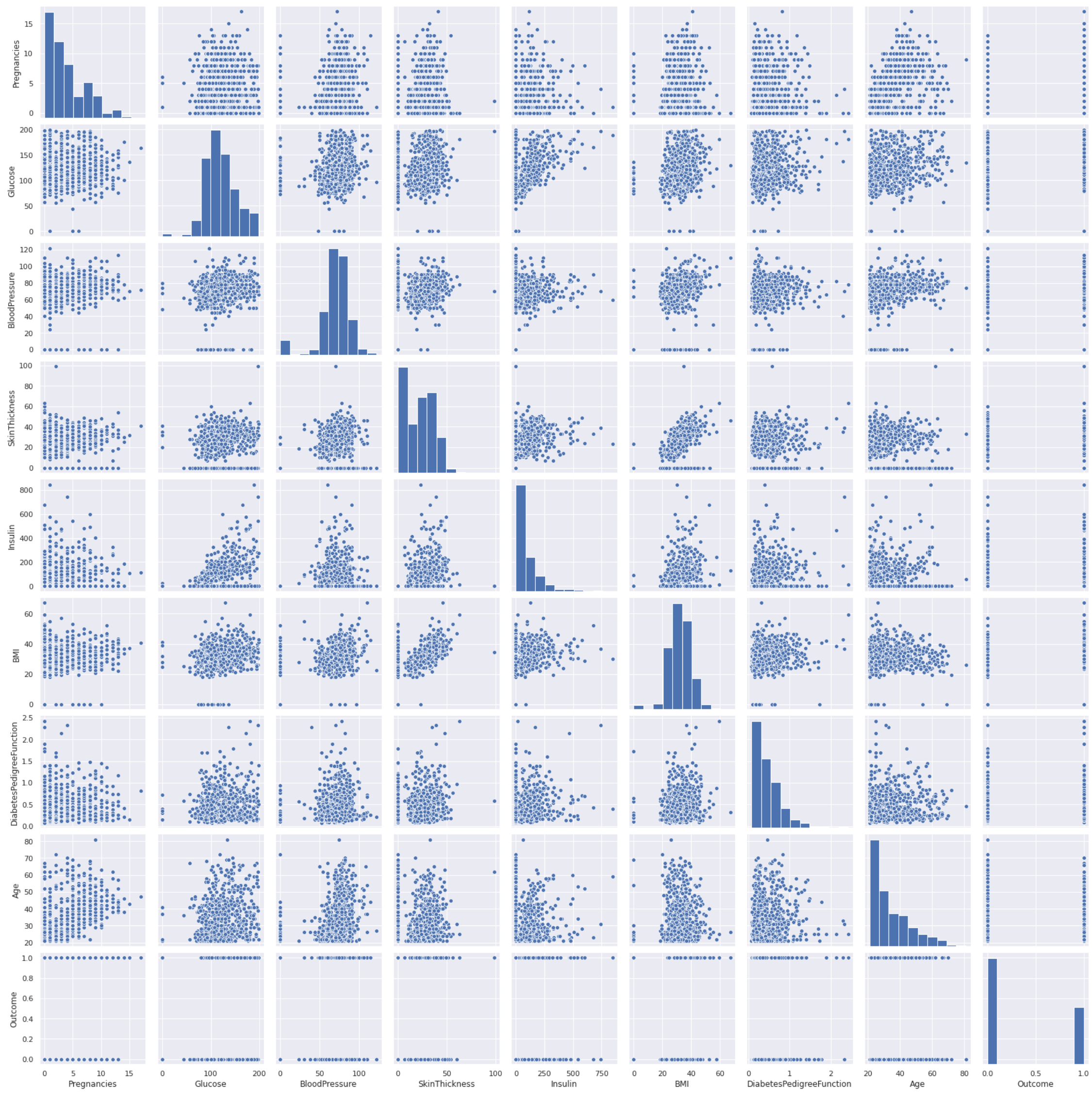
[38]:
```
corrmat = df_db.corr()
f, ax = plt.subplots(figsize=(12, 12))
sns.heatmap(corrmat, vmax=.8, square=True);
```



[39]:
```
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'Outcome')['Outcome'].index
cm = np.corrcoef(df_db1[cols].values.T)
sns.set(font_scale=1.5)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',␣
↪annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```

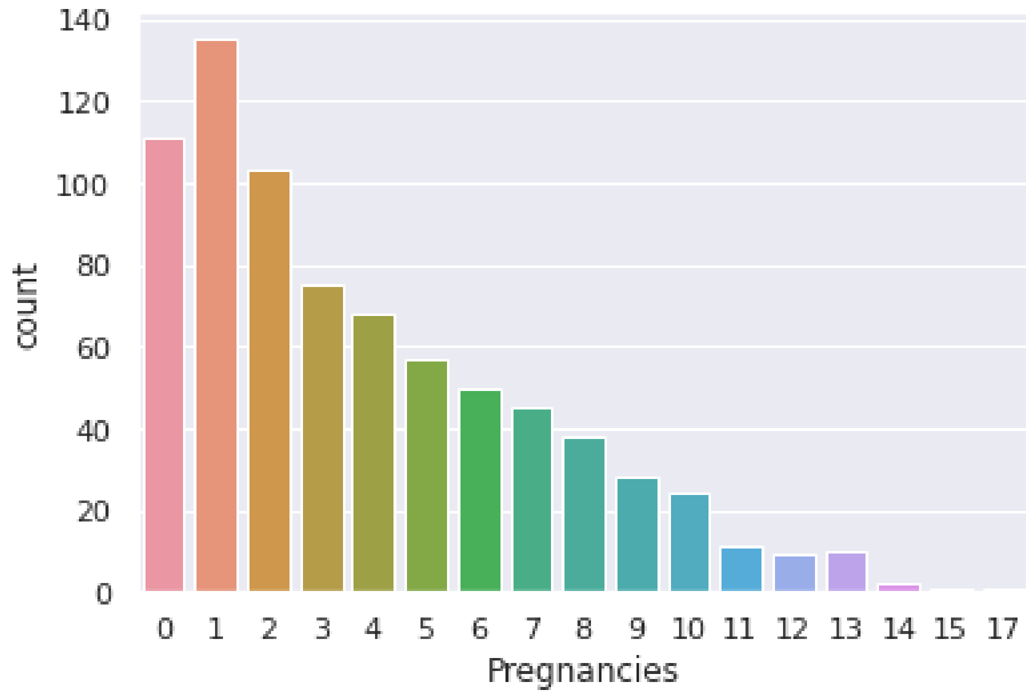


[42]:
```
#pairplot
sns.set()
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',␣
 ↪'BMI', 'DiabetesPedigreeFunction','Age','Outcome']
sns.pairplot(df_db[cols], size = 2.5)
plt.show();
```



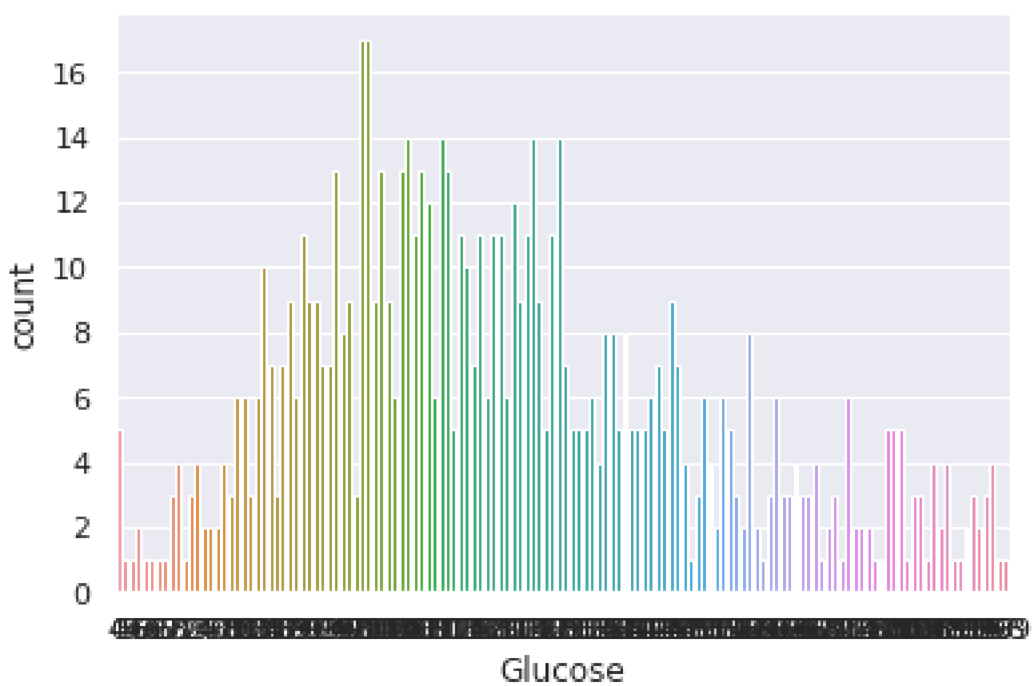
[43]:
```
sns.countplot(df_db1['Pregnancies'])
```

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



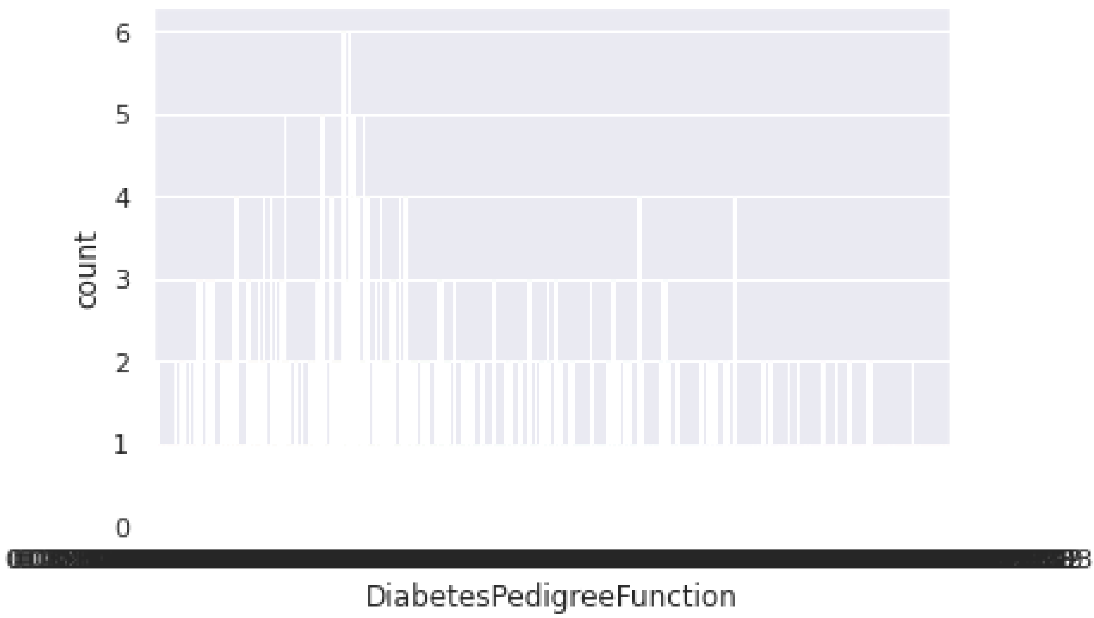

```
[45]: sns.countplot(df_db1['Glucose'])
```

[45]: <AxesSubplot:xlabel='Glucose', ylabel='count'>



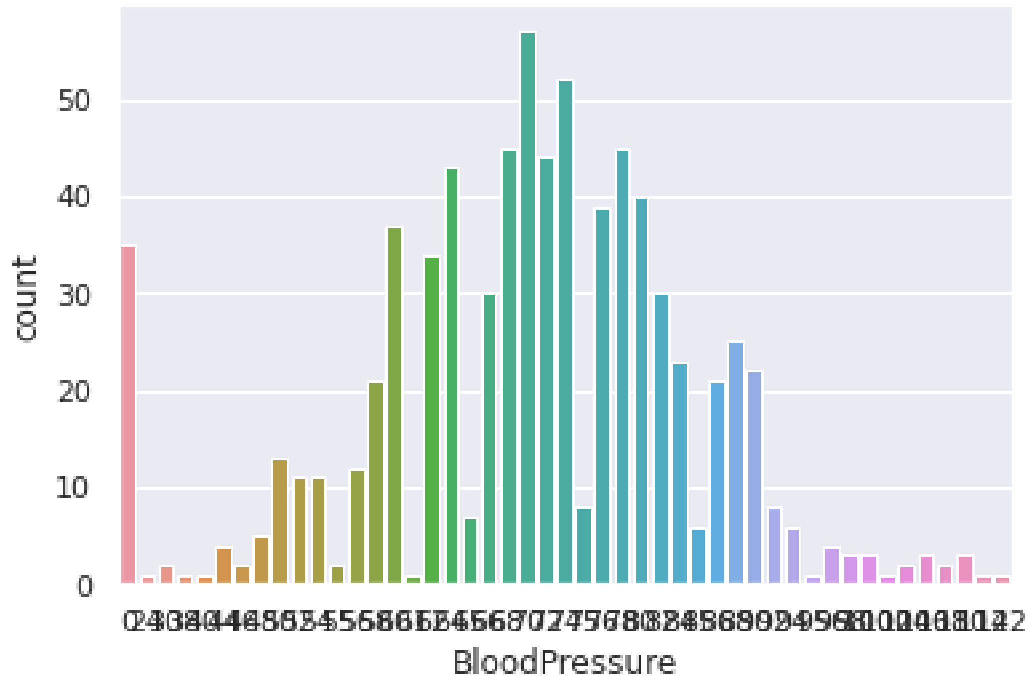
[44]:
```
sns.countplot(df_db1['DiabetesPedigreeFunction'])
```

[44]: <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='count'>



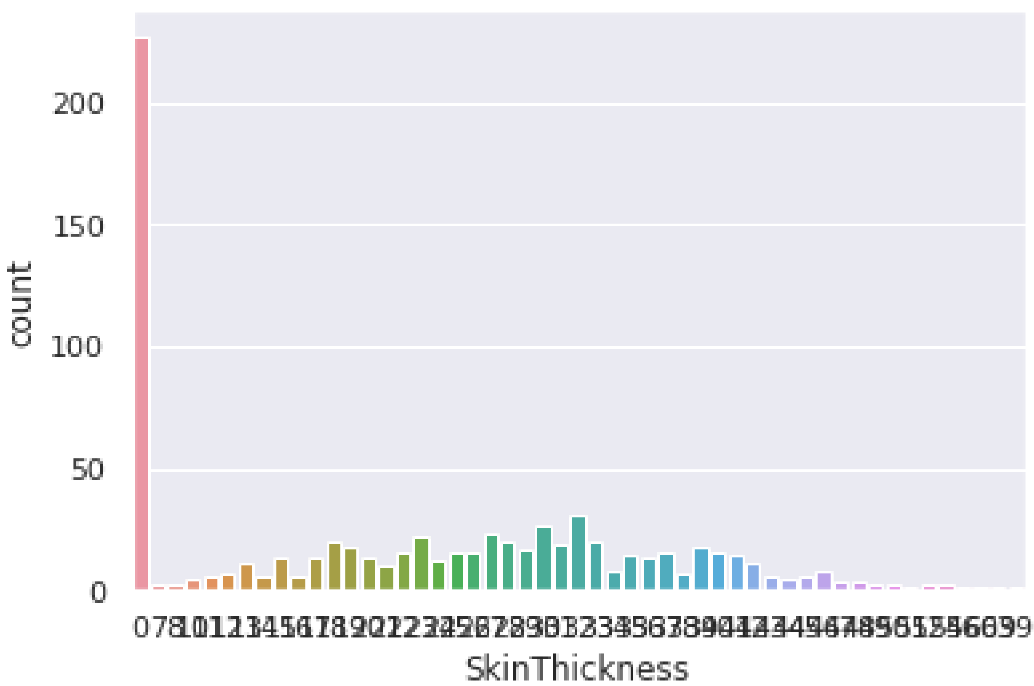
[46]:
```
sns.countplot(df_db1['BloodPressure'])
```

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



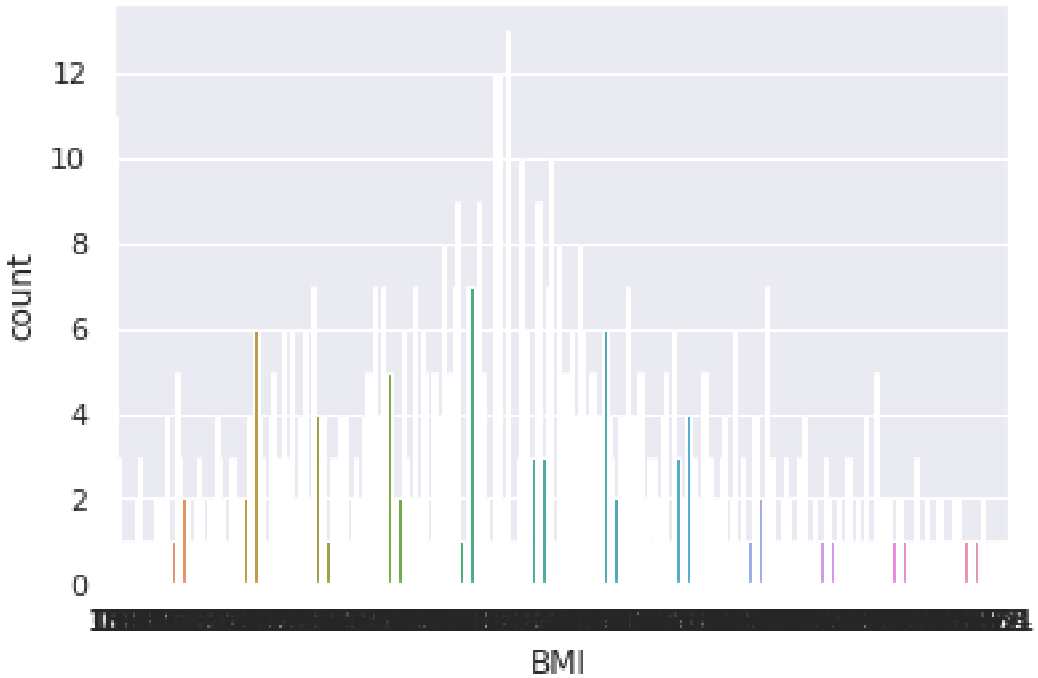

```
[47]: sns.countplot(df_db1['SkinThickness'])
```

[47]: <AxesSubplot:xlabel='SkinThickness', ylabel='count'>



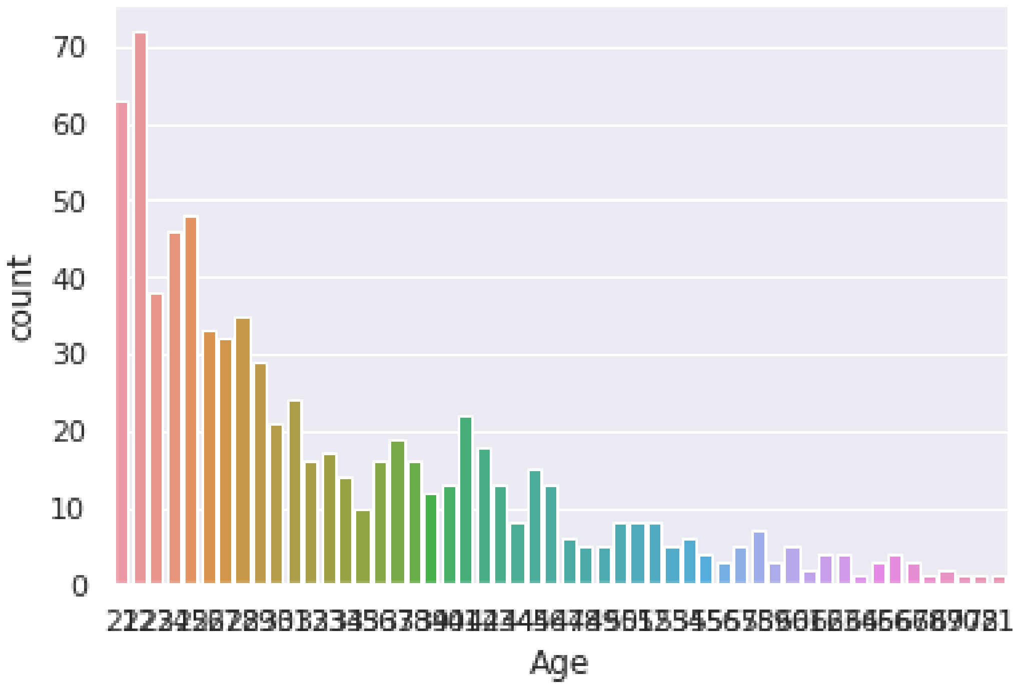
[67]:
```
sns.countplot(df_db1['BMI'])
```

[67]: <AxesSubplot:xlabel='BMI', ylabel='count'>



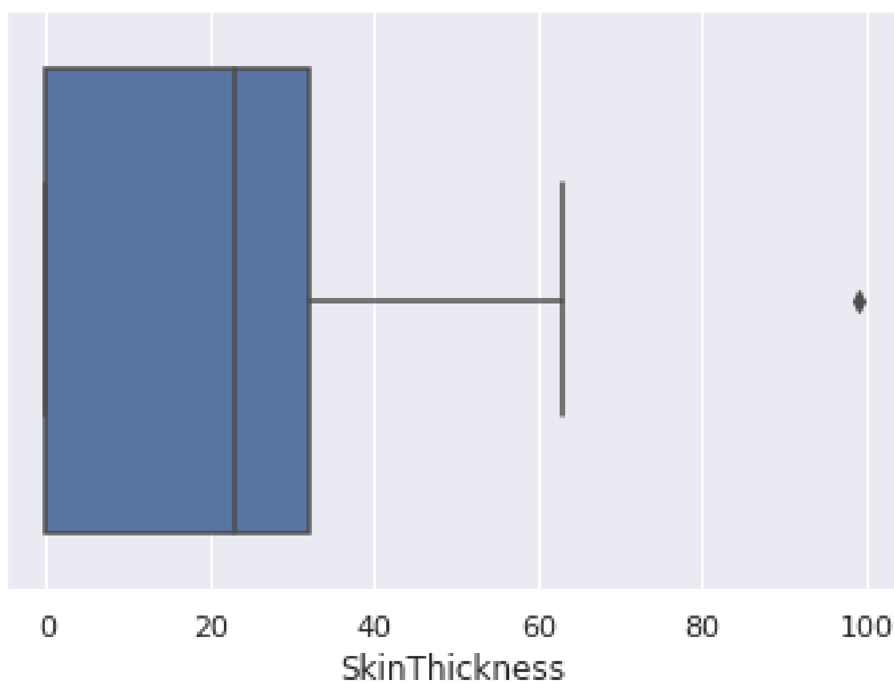
[68]:
```
sns.countplot(df_db1['Age'])
```

[68]: <AxesSubplot:xlabel='Age', ylabel='count'>



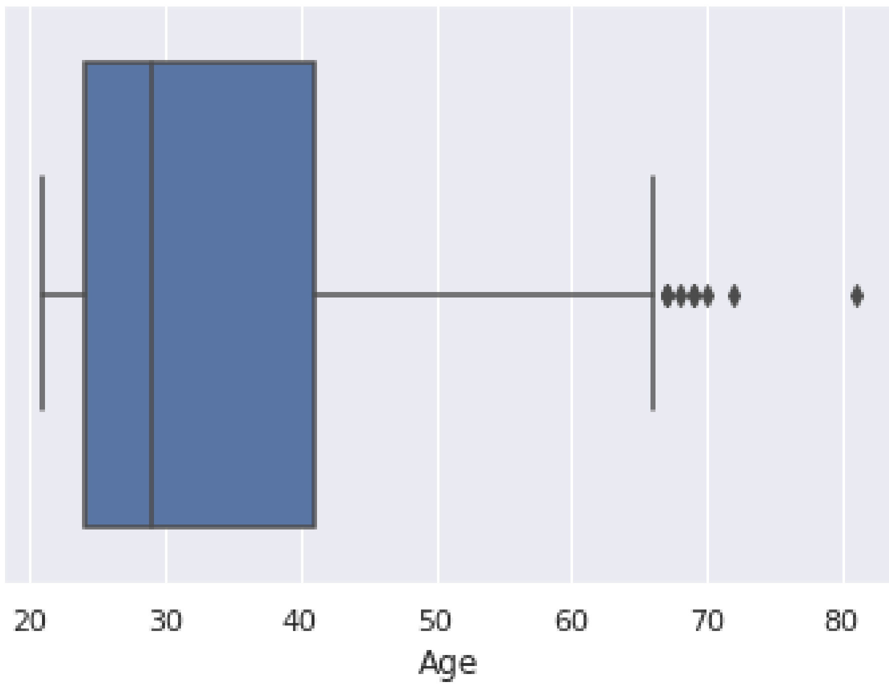
[69]:
```
sns.boxplot(df_db1['SkinThickness'])
```

[69]: <AxesSubplot:xlabel='SkinThickness'>



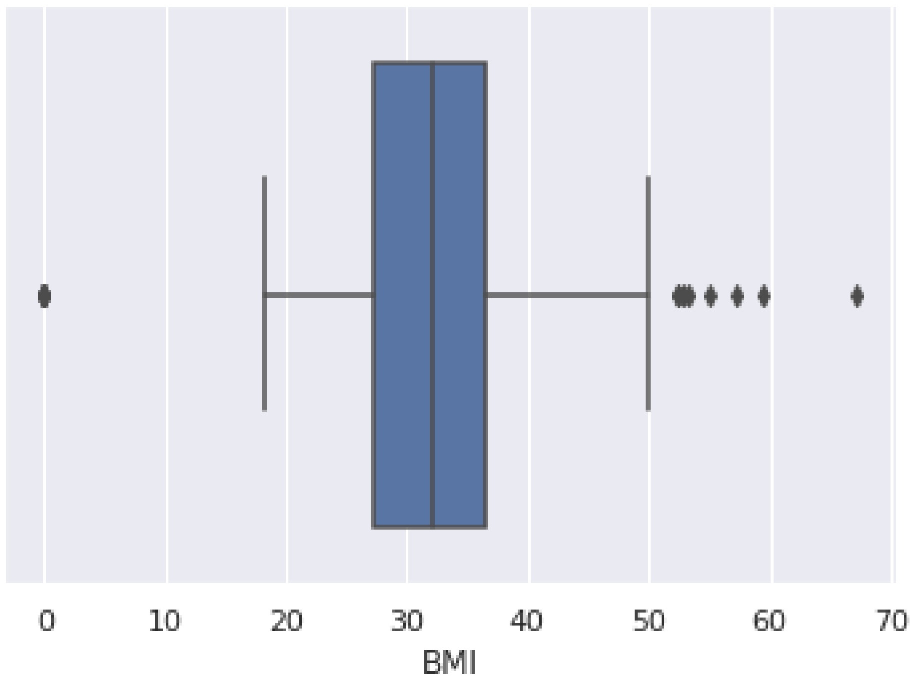
[70]:
```
sns.boxplot(df_db1['Age'])
```

[70]: <AxesSubplot:xlabel='Age'>



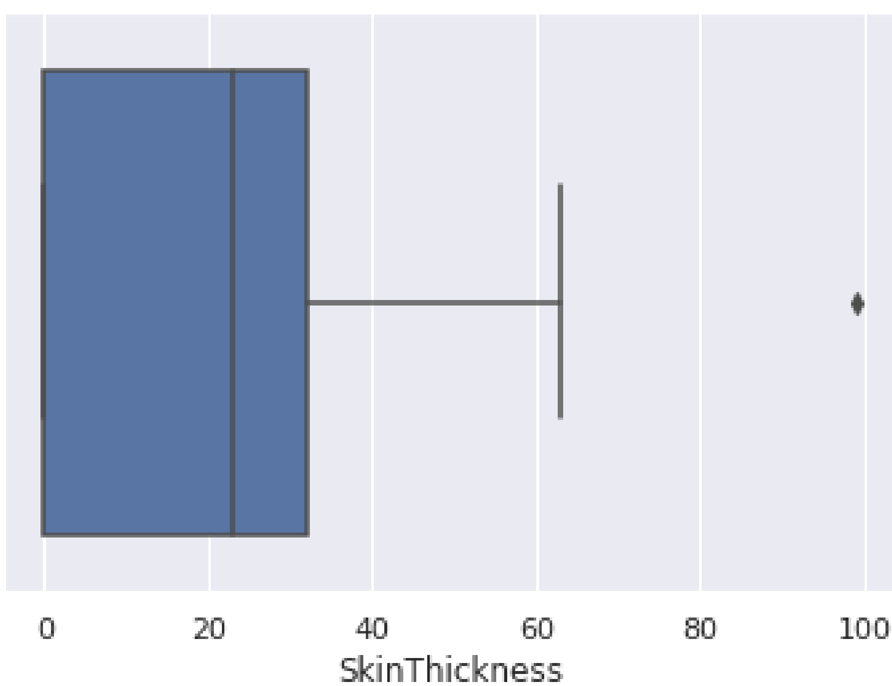
[71]:
```
sns.boxplot(df_db1['BMI'])
```

[71]: <AxesSubplot:xlabel='BMI'>



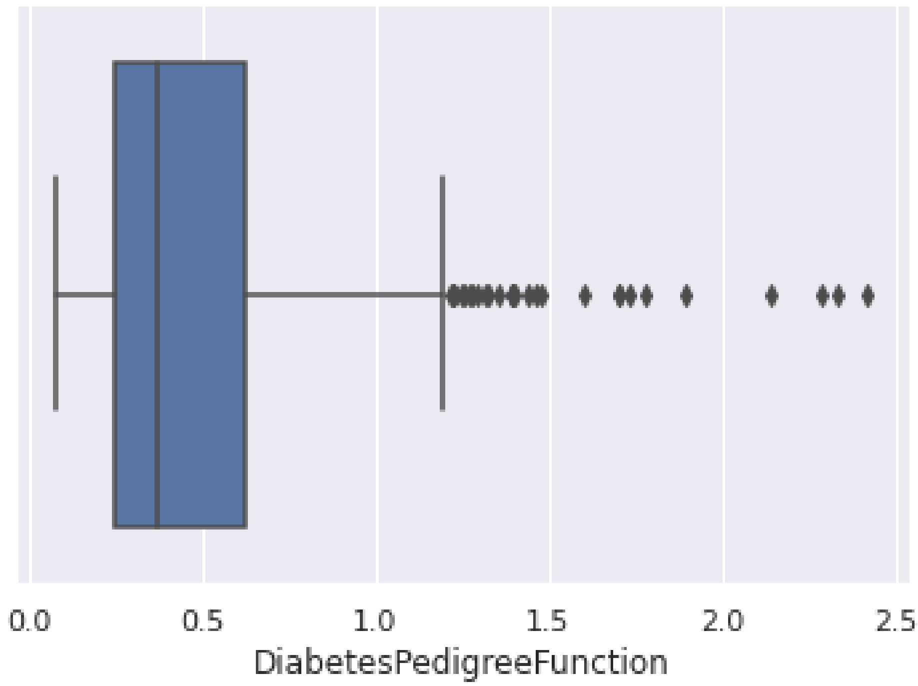
[72]:
```
sns.boxplot(df_db1['SkinThickness'])
```

[72]: <AxesSubplot:xlabel='SkinThickness'>



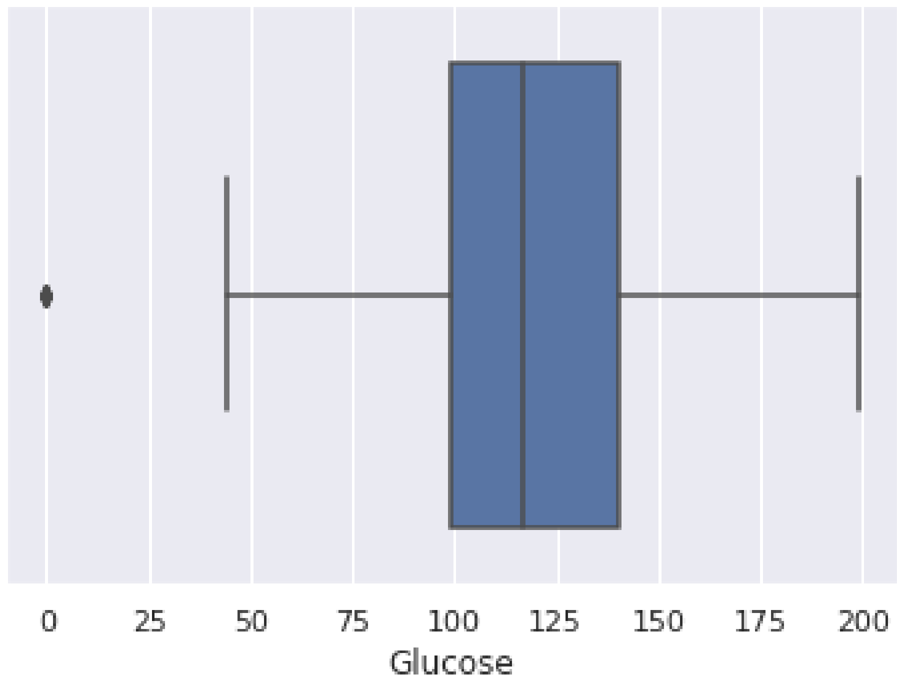
[74]:
```
sns.boxplot(df_db1['DiabetesPedigreeFunction'])
```

[74]: <AxesSubplot:xlabel='DiabetesPedigreeFunction'>



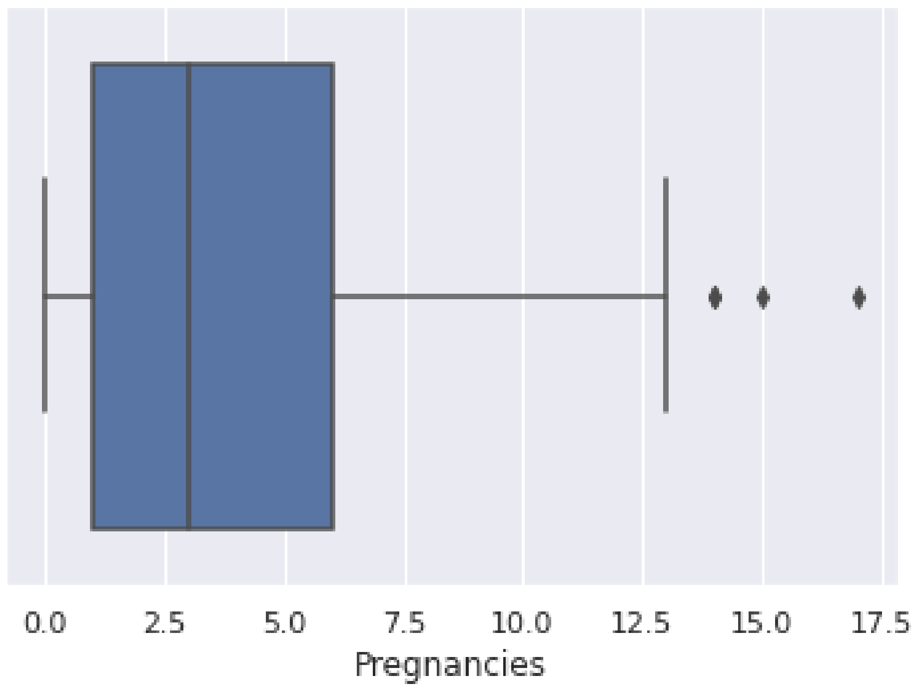
[75]:
```
sns.boxplot(df_db1['Glucose'])
```

[75]: <AxesSubplot:xlabel='Glucose'>



[76]:
```
sns.boxplot(df_db1['Pregnancies'])
```

[76]: <AxesSubplot:xlabel='Pregnancies'>



[79]:
```
from sklearn.model_selection import train_test_split
X = df_db.drop(columns = 'Outcome')
y = df_db['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.
↪2,random_state = 42)
```

[92]:
```
print(X_train)
```

```
[[-0.52639686 -1.15139792 -3.75268255 … -4.13525578 -0.49073479
  -1.03594038]
 [ 1.58804586 -0.27664283  0.68034485 … -0.48916881  2.41502991
   1.48710085]
 [-0.82846011  0.56687102 -1.2658623  … -0.42452187  0.54916055
  -0.94893896]
 …
 [ 1.8901091  -0.62029661  0.89659009 …  1.76054443  1.981245
   0.44308379]
 [-1.13052335  0.62935353 -3.75268255 …  1.34680407 -0.78487662
  -0.33992901]
 [-1.13052335  0.12949347  1.43720319 … -1.22614383 -0.61552223
  -1.03594038]]
```

[89]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
train_t = scalar.fit_transform (X_train) #fit and transform
test_t = scalar.transform (X_test) # only transform
print(train_t)
```

```
[[-0.52639686 -1.15139792 -3.75268255 … -4.13525578 -0.49073479
  -1.03594038]
 [ 1.58804586 -0.27664283  0.68034485 … -0.48916881  2.41502991
   1.48710085]
 [-0.82846011  0.56687102 -1.2658623  … -0.42452187  0.54916055
  -0.94893896]
 …
 [ 1.8901091  -0.62029661  0.89659009 …  1.76054443  1.981245
   0.44308379]
 [-1.13052335  0.62935353 -3.75268255 …  1.34680407 -0.78487662
  -0.33992901]
 [-1.13052335  0.12949347  1.43720319 … -1.22614383 -0.61552223
  -1.03594038]]
```

[93]:
```
#Fitting the Model to the training data
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import xgboost
```

[94]:
```
from sklearn.model_selection import KFold,cross_val_score
for model in [
    DummyClassifier,
    LogisticRegression,
    DecisionTreeClassifier,
    KNeighborsClassifier,
    GaussianNB,
    SVC,
    RandomForestClassifier,
    xgboost.XGBClassifier,
    ]:
    cls = model()
    kf = KFold(n_splits = 5, random_state = 45)
    score = cross_val_score(cls, train_t, y_train, cv = kf, scoring = "roc_auc")
    print(f" {model.__name__:22} AUC:"
          f"\t {score.mean():.3f} STD: {score.std():.2f}")
```

```
DummyClassifier        AUC:     0.481 STD: 0.02
LogisticRegression     AUC:     0.832 STD: 0.02
DecisionTreeClassifier AUC:     0.674 STD: 0.03
KNeighborsClassifier   AUC:     0.786 STD: 0.04
GaussianNB             AUC:     0.803 STD: 0.04
SVC                    AUC:     0.833 STD: 0.04
RandomForestClassifier AUC:     0.835 STD: 0.03
XGBClassifier          AUC:     0.809 STD: 0.03
```

[95]:
```
#Without any Hyper parameters we see "RandomForestClassifier" model has the␣
↪best accuracy
#making the model work
rfe = RandomForestClassifier(n_estimators = 1000, random_state = 42)
```

[96]:
```
rfe.fit(train_t, y_train)
```

[96]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
```

```
                        warm_start=False)
```

[97]:
```
#Evaluating the model
print("Accuracy on test set is", round(rfe.score(test_t, y_test)*100,2),"%")
```

```
Accuracy on test set is 74.03 %
```

[98]:
```
from sklearn.metrics import precision_score
print("Precision score is", round(precision_score(y_test, rfe.
 ↪predict(test_t))*100,2))
```

```
Precision score is 63.16
```

[105]:
```
print("feature importance is \n")
for i,j in zip(X.columns.to_list(), rfe.feature_importances_.tolist()):
    if(i=="DiabetesPedigreeFunction"):
        print(i, "\t\t",j)
    elif(i=="Age" or i=="BMI"):
        print(i, "\t\t\t\t\t\t", j)
    else:
        print(i, "\t\t\t\t", j)
```

```
feature importance is

Pregnancies                         0.07870476973503568
Glucose                             0.257006690994012
BloodPressure                       0.08838587879978824
SkinThickness                       0.0682424578137327
Insulin                             0.07746893645810259
BMI                                         0.16419653560403633
DiabetesPedigreeFunction            0.12046534553435637
Age                                         0.14552938506093616
```

[106]:
```
#lets do hyperparam tuning
from sklearn.model_selection import GridSearchCV
new_rfe = RandomForestClassifier()
params = {
    "max_features":[0.4,"auto"],
    "n_estimators":[15,200,500,1000],
    "min_samples_leaf":[1,0.1],
    "random_state":[42],
}
cvs = GridSearchCV(new_rfe, params, n_jobs = -1).fit(train_t, y_train)
```

[107]:
```
print(cvs.best_score_)
```

```
0.7785285885645742
```

[108]:
```
print(cvs.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

[109]:
```
print(cvs.best_params_)
```

```
{'max_features': 'auto', 'min_samples_leaf': 1, 'n_estimators': 200,
'random_state': 42}
```

[115]:
```
#fitting the model with best params
rfe_final = RandomForestClassifier()
params = {
    "max_features":["auto"],
    "n_estimators":[200],
    "min_samples_leaf":[1],
    "random_state":[42],
}
```

[116]:
```
rfe_final.fit(train_t, y_train)
```

[116]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[117]:
```
y_pred = rfe_final.predict(test_t)
```

[118]:
```
print(precision_score(y_test, y_pred))
```

```
0.6363636363636364
```

[120]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        99
```

```
             1       0.64      0.64      0.64        55

      accuracy                           0.74       154
     macro avg       0.72      0.72      0.72       154
  weighted avg       0.74      0.74      0.74       154
```

[121]:
```
#print Roc_auc_score
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test, y_pred))
```

```
0.7171717171717171
```

[ ]:
```
```