In [142…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
import os

from sklearn.preprocessing import QuantileTransformer
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, lea
```

In [143…
```
df = pd.read_csv (r'C:\Users\jn405\Downloads\test_diabetes.csv',delimiter=';')   #read
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               731 non-null    float64
 1   Glucose                   730 non-null    float64
 2   BloodPressure             734 non-null    float64
 3   SkinThickness             734 non-null    float64
 4   Insulin                   717 non-null    object 
 5   BMI                       733 non-null    float64
 6   DiabetesPedigreeFunction  728 non-null    float64
 7   Age                       717 non-null    float64
 8   Outcome                   768 non-null    object 
dtypes: float64(7), object(2)
memory usage: 54.1+ KB
```

In [144…
```
df.isnull().sum()
```

Out[144…
```
Pregnancies                37
Glucose                    38
BloodPressure              34
SkinThickness              34
Insulin                    51
BMI                        35
DiabetesPedigreeFunction   40
Age                        51
Outcome                     0
dtype: int64
```

In [145…
```
#fill all empty with 0
df = df.fillna(0)
```

In [146…
```
#change the zero to number and change the type from object to interger
df['Insulin'].astype(str)
df['Insulin'] = df['Insulin'].replace("Zero", 0)
df['Insulin'] = df['Insulin'].astype('int')
```

In [147…
```
#change the N to 0, Y to 1 and change the type from object to interger
df['Outcome'].astype(str)
df['Outcome'] = df['Outcome'].replace("N", 0)
df['Outcome'] = df['Outcome'].replace("Y", 1)
df['Outcome'] = df['Outcome'].astype('int')
```

In [148…
```
df.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    float64
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    int32  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    float64
 8   Outcome                   768 non-null    int32  
dtypes: float64(7), int32(2)
memory usage: 48.1 KB
```

In [149…
```
#fill the 0 value with mean or median
df['Glucose'] = df['Glucose'].replace(0,df['Glucose'].mean())
```

In [150…
```
df[df['BloodPressure'] == 0]['BloodPressure'].value_counts()
df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

In [151…
```
df[df['BMI'] == 0]['BMI'].value_counts()
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())
```

In [152…
```
df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())
```

In [153…
```
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())
```

In [154…
```
#description of the data
df.describe()
```

Out[154…

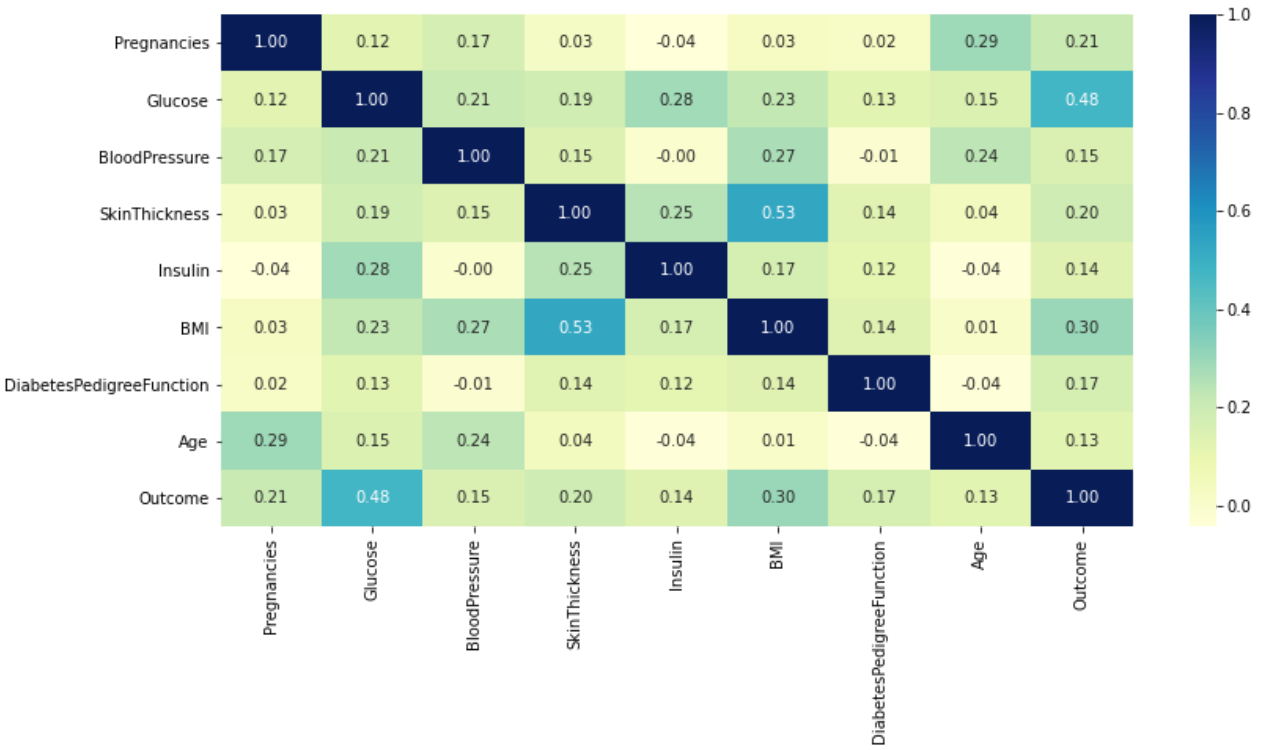
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.640625	121.455231	71.878601	26.886719	74.682292	32.414844	
std	3.387763	29.748040	11.769829	9.293361	111.565431	6.707937	
min	0.000000	44.000000	30.000000	7.000000	0.000000	18.200000	
25%	1.000000	100.000000	65.000000	22.000000	0.000000	27.775000	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreel
50%	3.000000	115.157552	70.000000	22.000000	0.000000	31.600000	
75%	6.000000	139.000000	80.000000	32.000000	122.000000	36.225000	
max	17.000000	198.000000	122.000000	99.000000	846.000000	67.100000	

In [155…

```
#This is the correlation graph
plt.figure(figsize=(13,6))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap="YlGnBu")
```

Out[155… <AxesSubplot:>

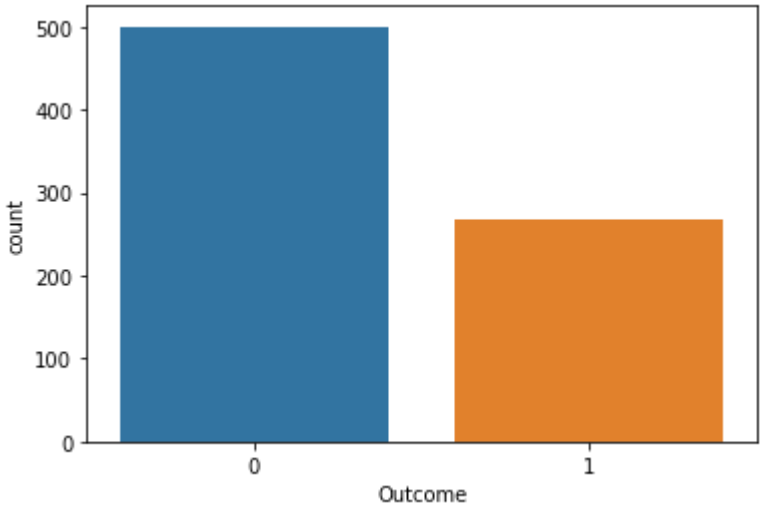


In [156…

```
#show the outcome with bar plot
sns.countplot('Outcome',data=df)
```

```
C:\Users\jn405\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

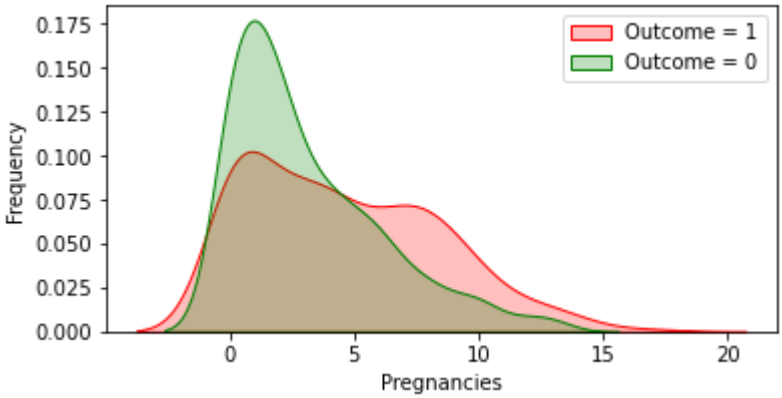
Out[156… <AxesSubplot:xlabel='Outcome', ylabel='count'>



In [157…

```
# Explore Pregnancies vs Outcome
plt.figure(figsize=(6,3))
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1], color="Red", shade = True)
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0], ax =g, color="Green", shade= Tru
g.set_xlabel("Pregnancies")
g.set_ylabel("Frequency")
g.legend(["Outcome = 1","Outcome = 0"])
```

Out[157… <matplotlib.legend.Legend at 0x275a8fb07f0>

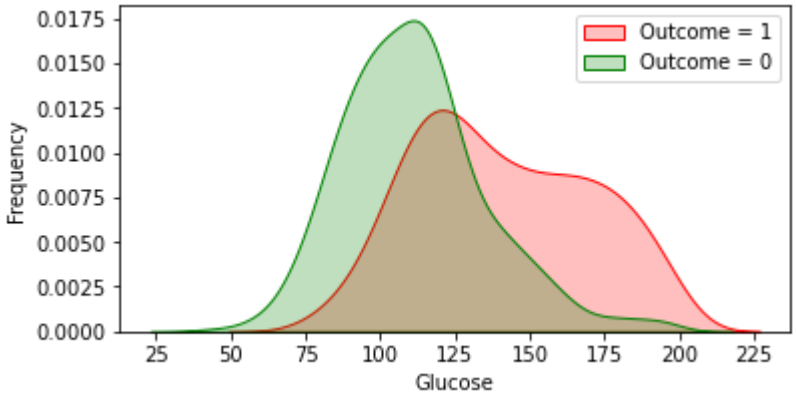


In [158…

```
# Explore Glucose vs Outcome

plt.figure(figsize=(6,3))
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
g.set_xlabel("Glucose")
g.set_ylabel("Frequency")
g.legend(["Outcome = 1","Outcome = 0"])
```

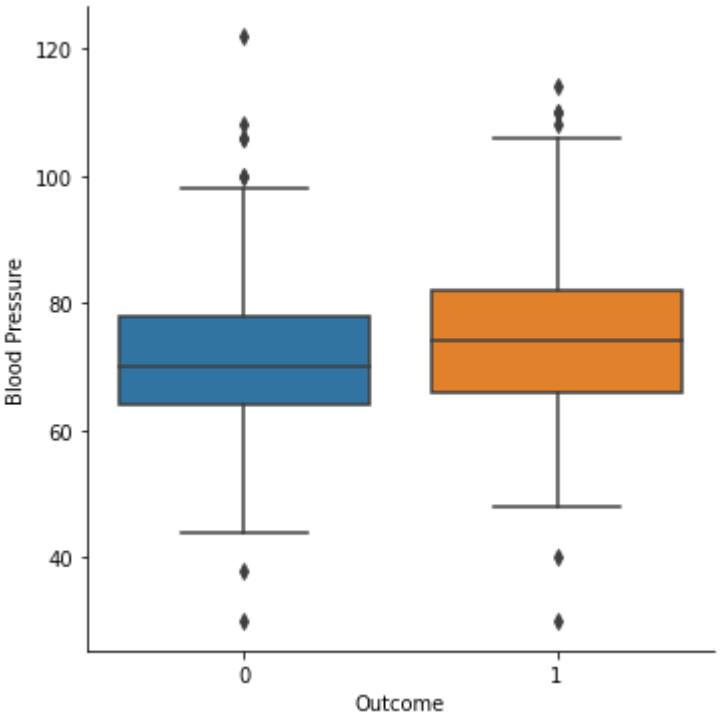
Out[158… <matplotlib.legend.Legend at 0x275a91136a0>



In [159…

```
# boxplot for bloodpressure, Age and DiabitepedigreeFunction
g = sns.catplot(y="BloodPressure",x="Outcome",data=df,kind="box")
g.set_ylabels("Blood Pressure")
g.set_xlabels("Outcome")
```

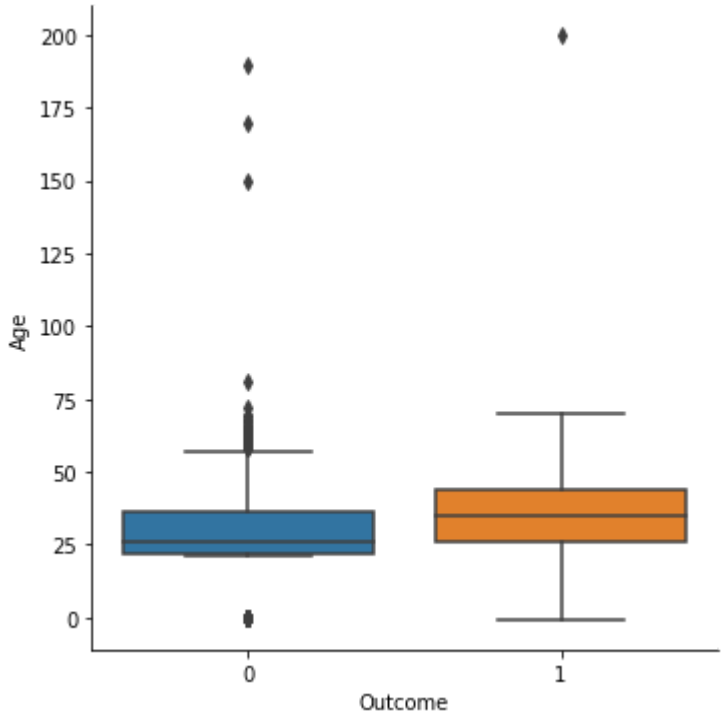
Out[159… <seaborn.axisgrid.FacetGrid at 0x275a9103820>



In [160…

```
g = sns.catplot(y="Age",x="Outcome",data=df,kind="box")
g.set_ylabels("Age")
g.set_xlabels("Outcome")
```

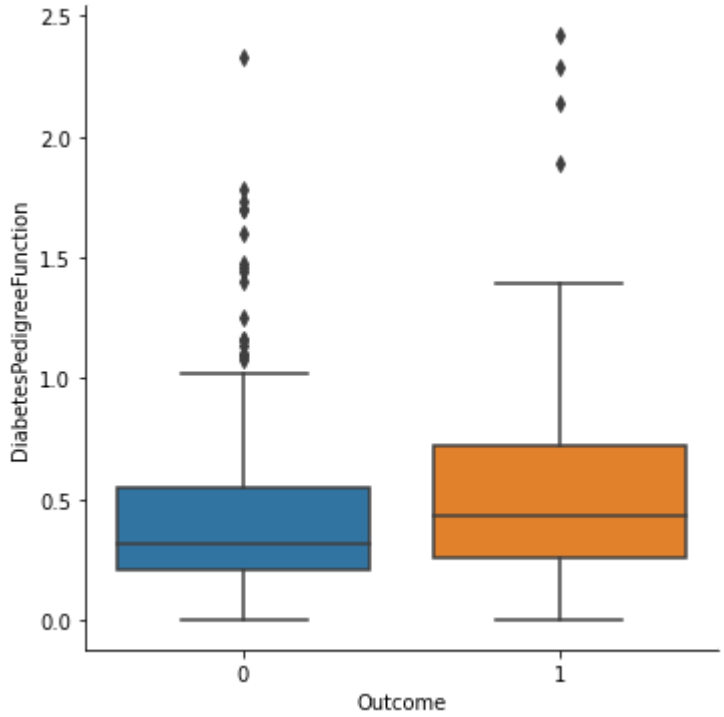
Out[160… <seaborn.axisgrid.FacetGrid at 0x275a68b2a60>



In [161…

```
g = sns.catplot(y="DiabetesPedigreeFunction",x="Outcome",data=df,kind="box")
g.set_ylabels("DiabetesPedigreeFunction")
g.set_xlabels("Outcome")
```

Out[161… <seaborn.axisgrid.FacetGrid at 0x275a909dfa0>



In [162…

```
def detect_outliers(df,n,features):
    outlier_indices = []
    """
    Detect outliers from given list of features. It returns a list of the indices
    according to the observations containing more than n outliers according
```

```
    to the standard diviation method using 95% intervel
    """
    # iterate over features(columns)
    for var in features:
        dfmean = np.mean(df[var])
        dfstd = np.std(df[var])
        cut_off = 1.96*dfstd
        lower = dfmean-cut_off
        upper = dfmean + cut_off
        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[var] < lower) | (df[var] > upper)].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers

# detect outliers from numeric features
outliers_to_drop = detect_outliers(df, 2 ,["Pregnancies", 'Glucose', 'BloodPressure', '
```

In [163…

```
df.loc[outliers_to_drop] # Show the outliers rows
```

Out[163…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
445	0.0	180.0	78.0	63.0	14	59.4	2.420	25.0
177	0.0	129.0	110.0	46.0	130	67.1	0.319	26.0
370	3.0	173.0	82.0	48.0	465	38.4	2.137	25.0

In [164…

```
#drop the outlier
df.drop(df.loc[outliers_to_drop].index, inplace=True)
```

In [165…

```
## Separate train dataset and test dataset
var = df.drop(["Outcome"], axis=1)
output = df["Outcome"]
x_train, x_test, y_train, y_test = train_test_split(var, output, test_size=0.40, random
```

In [166…

```
import statsmodels.api as sm
logit_model=sm.Logit(y_train,x_train)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.576327
         Iterations 5
                          Results: Logit
=================================================================
Model:                 Logit             Pseudo R-squared:   0.090
Dependent Variable:    Outcome           AIC:                545.0682
```

```
Date:                  2021-12-14 10:08   BIC:                578.1006
No. Observations:      459                Log-Likelihood:     -264.53
Df Model:              7                  LL-Null:            -290.75
Df Residuals:          451                LLR p-value:        4.7816e-09
Converged:             1.0000             Scale:              1.0000
No. Iterations:        5.0000
--------------------------------------------------------------------------
                          Coef.  Std.Err.    z    P>|z|   [0.025  0.975]
--------------------------------------------------------------------------
Pregnancies               0.0999   0.0332  3.0101 0.0026  0.0349  0.1650
Glucose                   0.0213   0.0039  5.4917 0.0000  0.0137  0.0289
BloodPressure            -0.0587   0.0091 -6.4806 0.0000 -0.0764 -0.0409
SkinThickness             0.0012   0.0133  0.0936 0.9254 -0.0248  0.0273
Insulin                  -0.0002   0.0010 -0.2195 0.8263 -0.0023  0.0018
BMI                       0.0136   0.0191  0.7113 0.4769 -0.0239  0.0511
DiabetesPedigreeFunction  0.2504   0.3264  0.7670 0.4431 -0.3894  0.8901
Age                      -0.0006   0.0070 -0.0823 0.9344 -0.0143  0.0131
==========================================================================
```

In [167…

```
var2 = var.drop(['SkinThickness'], axis=1)
var2 = var2.drop(["Insulin"], axis=1)
var2 = var2.drop(["BMI"], axis=1)
var2 = var2.drop(["DiabetesPedigreeFunction"], axis=1)
var2 = var2.drop(["Age"], axis=1)
logit_model2=sm.Logit(output,var2)
result2=logit_model2.fit()
print(result2.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.582766
         Iterations 5
                         Results: Logit
================================================================
Model:              Logit            Pseudo R-squared: 0.097
Dependent Variable: Outcome          AIC:              897.6323
Date:               2021-12-14 10:08 BIC:              911.5519
No. Observations:   765              Log-Likelihood:   -445.82
Df Model:           2                LL-Null:          -493.57
Df Residuals:       762              LLR p-value:      1.8182e-21
Converged:          1.0000           Scale:            1.0000
No. Iterations:     5.0000
----------------------------------------------------------------
                 Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
----------------------------------------------------------------
Pregnancies      0.1135   0.0244   4.6455 0.0000  0.0656  0.1614
Glucose          0.0232   0.0028   8.2961 0.0000  0.0177  0.0286
BloodPressure   -0.0534   0.0052 -10.3573 0.0000 -0.0635 -0.0433
================================================================
```

In [168…

```
x_train2, x_test2, y_train2, y_test2 = train_test_split(var2, output, test_size=0.40, r
```

In [169…

```
# Import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

In [170…

```
# Define models and parameters for LogisticRegression with random state 0
model1 = LogisticRegression(solver='liblinear',random_state=0)
```

```
model1.fit(x_train2,y_train2)
y_pred12 = model1.predict(x_test2)
print(classification_report(y_test2, y_pred12))
```

```
              precision    recall  f1-score   support

           0       0.75      0.92      0.82       192
           1       0.77      0.47      0.59       114

    accuracy                           0.75       306
   macro avg       0.76      0.70      0.70       306
weighted avg       0.76      0.75      0.73       306
```

In [171…

```
model2 = RandomForestClassifier()
model2.fit(x_train,y_train)
y_pred2 = model2.predict(x_test)
print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.76      0.88      0.82       192
           1       0.72      0.54      0.62       114

    accuracy                           0.75       306
   macro avg       0.74      0.71      0.72       306
weighted avg       0.75      0.75      0.74       306
```

In [172…

```
model3 = RandomForestClassifier()
model3.fit(x_train2,y_train2)
y_pred3 = model3.predict(x_test2)
print(classification_report(y_test2, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.76      0.88      0.82       192
           1       0.72      0.54      0.62       114

    accuracy                           0.75       306
   macro avg       0.74      0.71      0.72       306
weighted avg       0.75      0.75      0.74       306
```

In [173…

```
x_test['pred'] = y_pred2
x_test
```

```
<ipython-input-173-0a71362a2e7f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  x_test['pred'] = y_pred2
```

Out[173…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
369	1.0	133.0	102.000000	28.0	140	32.8	0.234	45.0
388	0.0	144.0	82.000000	26.0	285	32.0	0.452	58.0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
479	0.0	132.0	86.000000	31.0	0	28.0	0.419	63.0
499	6.0	154.0	74.000000	32.0	193	29.3	0.839	39.0
233	4.0	122.0	68.000000	22.0	0	35.0	0.394	29.0
...	...	...	...	...	...	...	...	...
74	1.0	79.0	75.000000	30.0	0	32.0	0.396	22.0
382	1.0	109.0	60.000000	8.0	182	25.4	0.947	21.0
535	4.0	132.0	65.953125	22.0	0	32.9	0.302	23.0
431	3.0	89.0	74.000000	16.0	85	30.4	0.551	38.0
102	0.0	125.0	96.000000	22.0	0	22.5	0.262	21.0

306 rows × 9 columns