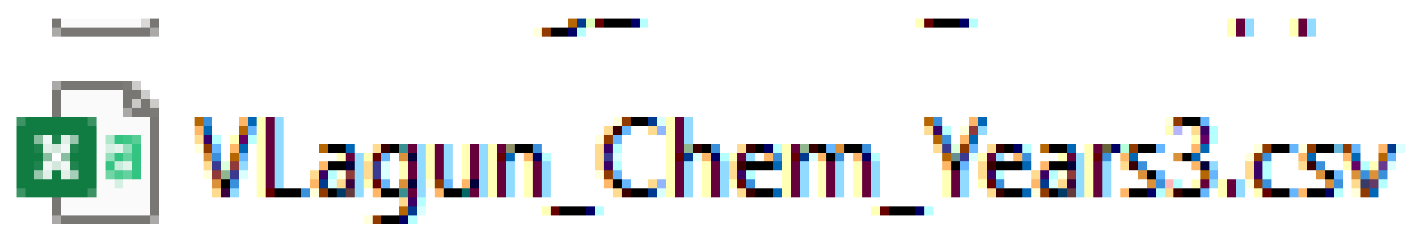
ZADANIE:

Rozszerzona analiza modelowania z EDA i Stacking

Wstaw tą bazę danych zamiast diabetes.csv i wykonaj pełny kod oraz zapisz wnioski ze stackingu.



VLagun_Chem_Years3.csv

Całość odeślij do TEAMS

In [1]:

```python
# Load the dependencies

import numpy as np
import pandas as pd
import os
```

In [2]:

```python
#Import sklearn classes
from sklearn.model_selection import train_test_split,RepeatedKFold, cross_val_score,KFold, RepeatedStratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier,export_graphviz
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, BaggingClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# sklearn utility to compare algorithms
from sklearn import model_selection
```

In [3]:

```python
#Visualisation Libraries
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz
```

In [4]:

```python
import warnings
warnings.filterwarnings('ignore')
```

Eli5
Yellobrick
Install in prompt

In [5]:

```python
from eli5 import explain_weights,show_weights
from yellowbrick import ROCAUC
from yellowbrick.classifier import ClassificationReport
```

In [7]:

```python
print("Imported all libraries successfully")
print(os.listdir("C:/Users/user/Desktop/Stack_Class"))
```

Your directory

```
Imported all libraries successfully
['diabetes.csv', 'vlaCopep_Stack.csv', 'vlaPhyto_Stack.csv']
```

In [8]:

```python
CV_N_REPEATS=20
BINS=10
```

Change CV-20 to CV-5

In [9]:

```python
df = pd.read_csv("C:/Users/user/Desktop/Stack_Class/diabetes.csv")
df.head()
```

Your .csv file

Out[9]:

	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	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [10]:

```python
print('Shape of the dataset', df.shape)
```

```
Shape of the dataset (768, 9)
```

In [11]:

```python
#ELEMENTARY DATA ANALYSIS (EDA)
# We can use the pandas_profiling library to automate most of the EDA process for us.
#It has been commented out for keeping the notebook concise.
#import pandas_profiling
#report=pandas_profiling.ProfileReport(df,check_correlation =True);
#report.to_file(outputfile="eda_report.html")
```

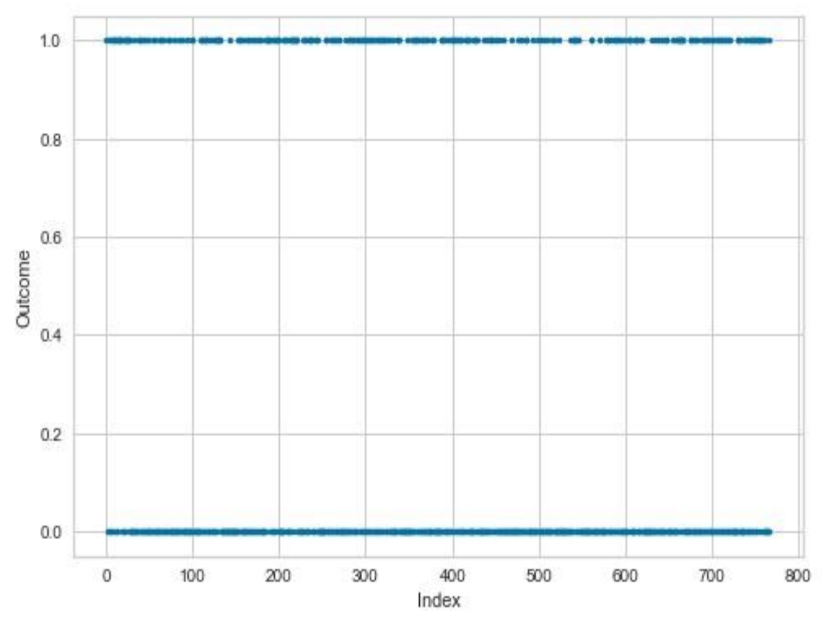
In [12]:

```python
plt.figure()
plt.plot(df.Outcome,'.')
plt.xlabel('Index')
plt.ylabel('Outcome')
```

Your target name,
(change elsewhere)

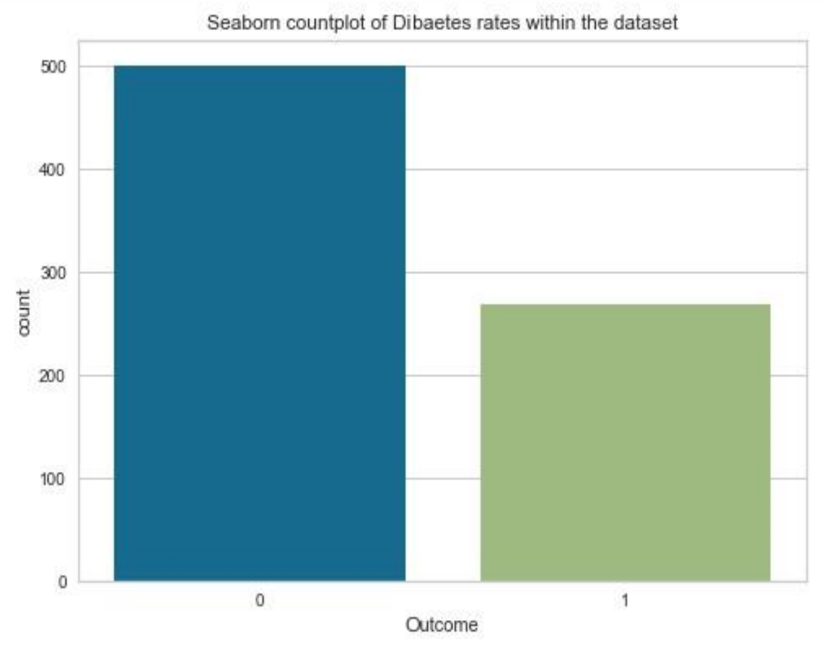
Out[12]:

```
Text(0, 0.5, 'Outcome')
```



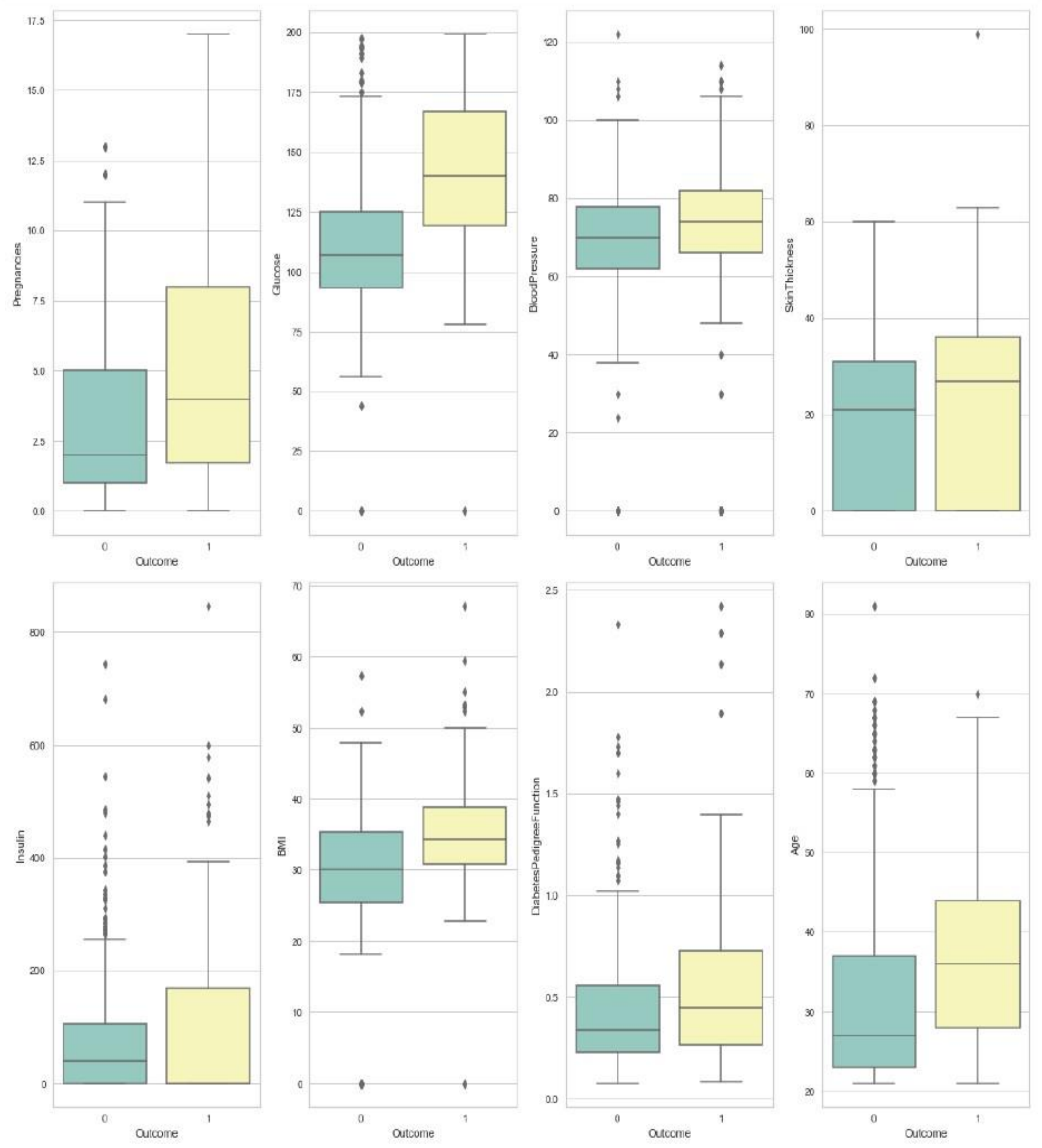
In [13]:

```python
plt.figure()
ax = sns.countplot(data=df, x='Outcome');
ax.set_title("Seaborn countplot of Dibaetes rates within the dataset");
```



```
f, axes = plt.subplots(2, 4,figsize=(15,15))
sns.set(style="white", palette="Set3", color_codes=True)
sns.boxplot(  y="Pregnancies", x= "Outcome", data=df,  orient='v', ax=axes[0,0])
sns.boxplot(  y="Glucose", x= "Outcome", data=df,  orient='v' , ax=axes[0,1])
sns.boxplot(  y="BloodPressure", x= "Outcome", data=df,  orient='v' ,ax=axes[0,2])
sns.boxplot(  y="SkinThickness", x= "Outcome", data=df,  orient='v' , ax=axes[0,3])
sns.boxplot(  y="Insulin", x= "Outcome", data=df,  orient='v' ,  ax=axes[1,0])
sns.boxplot(  y="BMI", x= "Outcome", data=df,  orient='v' ,  ax=axes[1,1])
sns.boxplot(  y="DiabetesPedigreeFunction", x= "Outcome", data=df,  orient='v' , ax=axes[1,2])
sns.boxplot(  y="Age", x= "Outcome", data=df,  orient='v' ,  ax=axes[1,3])

f.subplots_adjust(left=0.08, right=0.98, bottom=0.05, top=0.9,
            hspace=0.4, wspace=0.3)
#f.suptitle('Distribution of data')
plt.tight_layout()
```

Change variables (elsewhere)



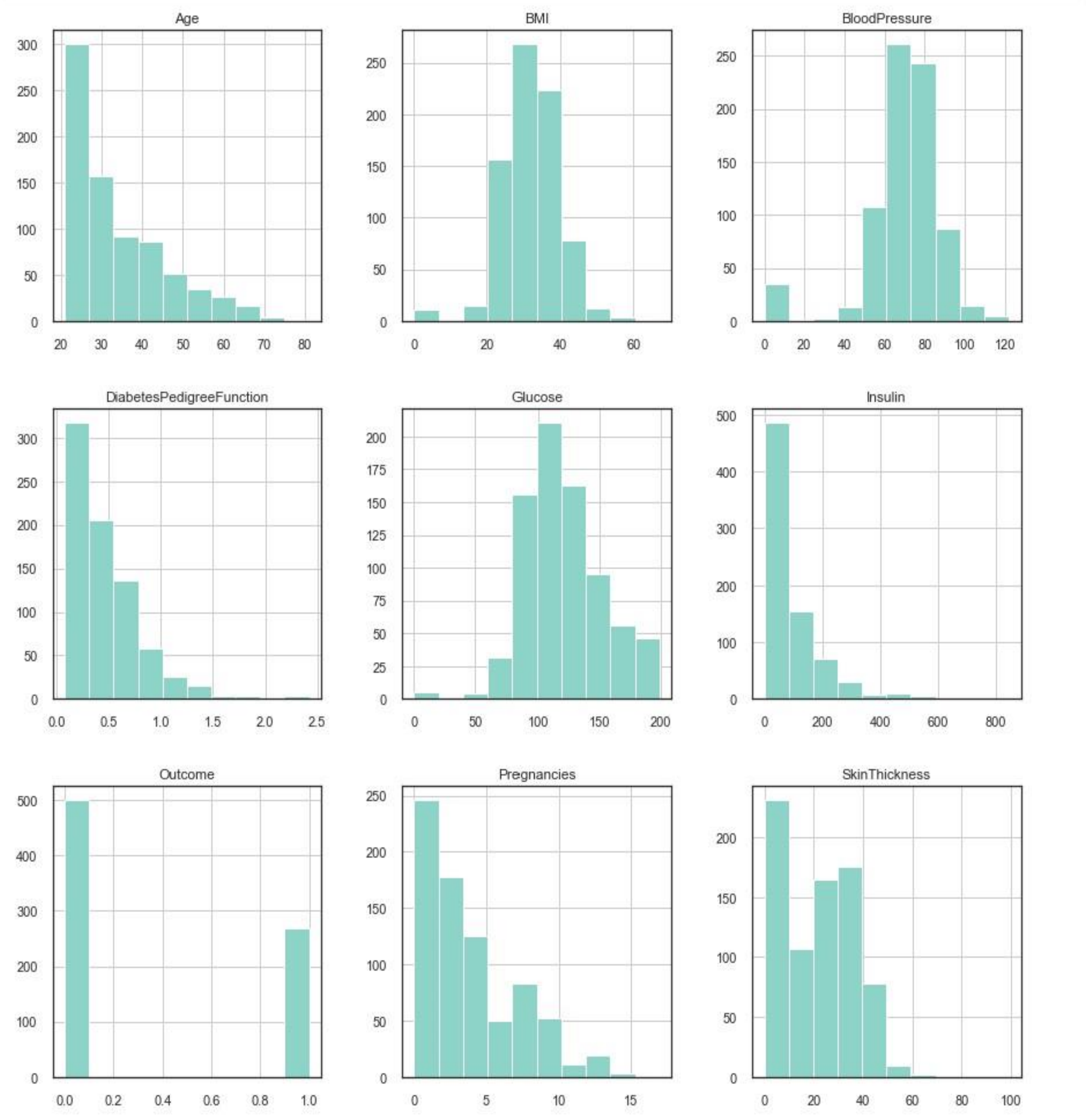
In [15]:

```
df_copy = df.copy(deep = True)
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.
NaN)
print('Number of zero entries in each attribute:\n')
print(df_copy.isnull().sum())
```

```
Insulin              374
BMI                   11
DiabetesPedigreeFunction     0
Age                    0
Outcome                0
dtype: int64
```

In [16]:

```
p = df.hist(figsize = (15,15))
```



In [42]:

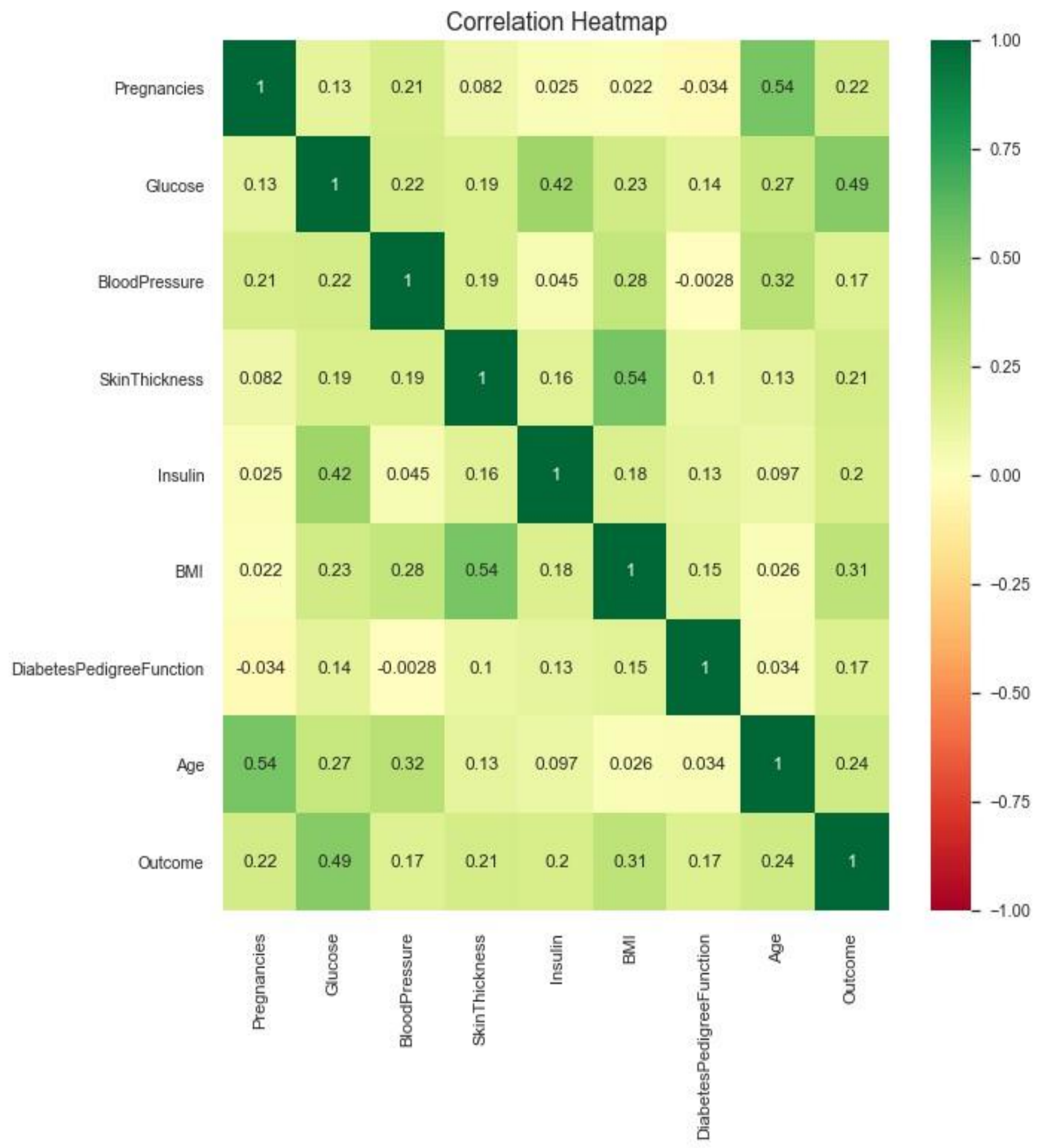
```
#Impute NaN values using mean and meadian values
print('Imputing NaN values')
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace = True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace = True)
df_copy['BMI'].fillna(df_copy['BMI'].mean(), inplace = True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace = True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace = True)
print('Number of zero entries in each attribute:\n')
```

```
print(df_copy.isnull().sum())
print(df_copy.isnull().sum())
```

```
Imputing NaN values
Number of zero entries in each attribute:

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

In [43]:

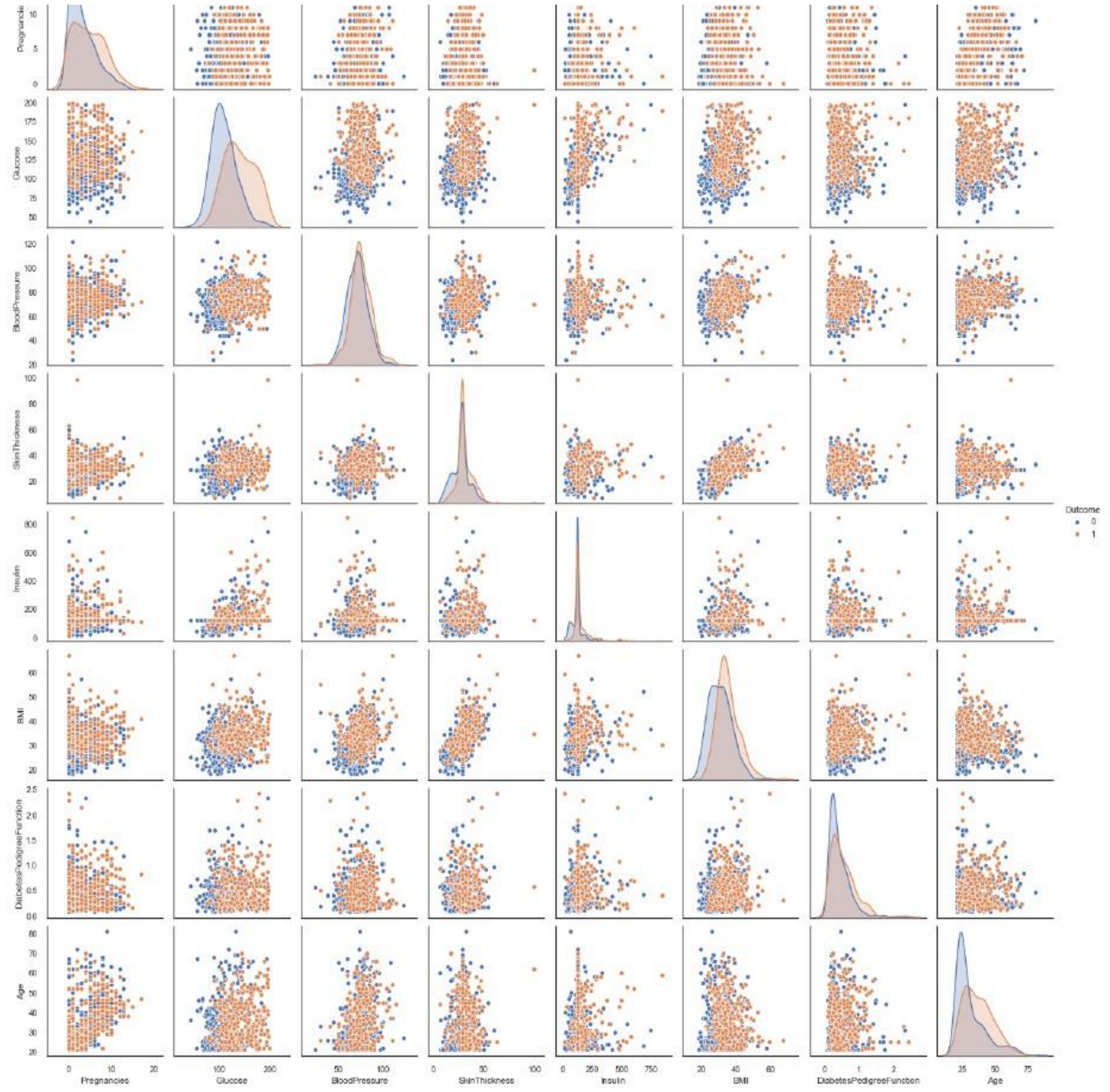
```
#plot the correlation map of the dataset
plt.figure(figsize=(10,10))
corr = df_copy.corr()
corr.index = df_copy.columns
sns.heatmap(corr, annot = True, cmap='RdYlGn', vmin=-1, vmax=1)
plt.title("Correlation Heatmap", fontsize=16)
plt.show()
```



In [44]:

```
plt.figure()
sns.pairplot(data=df_copy,hue='Outcome',diag_kind='kde', palette='deep');
```

```
<Figure size 576x396 with 0 Axes>
```



In [45]:

```
#Principal Component Analysis (PCA)

# Separating the features and the target (Y)
X = df_copy.iloc[:,0:8]
Y = df_copy.iloc[:,8]

#Standardizing the features
X= StandardScaler().fit_transform(X)
# Fit PCA and transform X.
pca=PCA(n_components=.90)
pca.fit(X)
print('Variance explained by the principal components(in decreasing order): ',pca.explained_variance_ratio_)
#print('PCA singular values: ',pca.singular_values_)
X1=pca.transform(X)
print('Shape of transformed X: ',X1.shape)
```

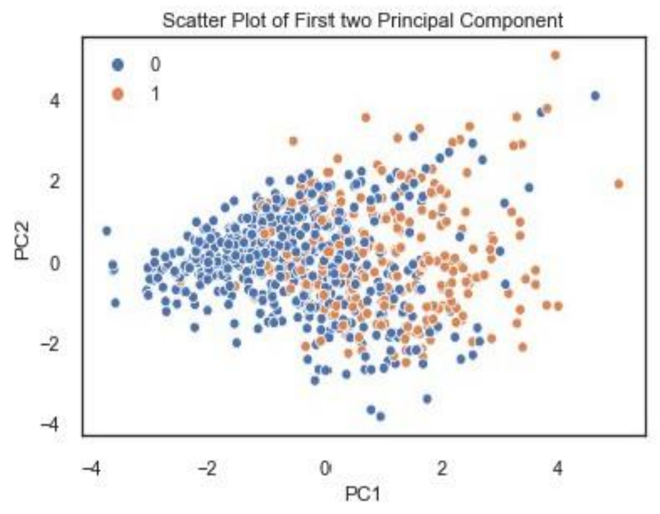
```
Variance explained by the principal components(in decreasing order):  [0.2852704  0.1869508  0.14269862 0.11458318 0.09610493 0.06797424
 0.05856217]
Shape of transformed X:  (768, 7)
```

In [54]:

```
# import the dataset
df = pd.read_csv('C:/Users/user/Desktop/Stack_Class/diabetes.csv')
X = df.iloc[: , 0:8].values     # feature metric
Y = df.iloc[: , 8].values     # dependent variable
```

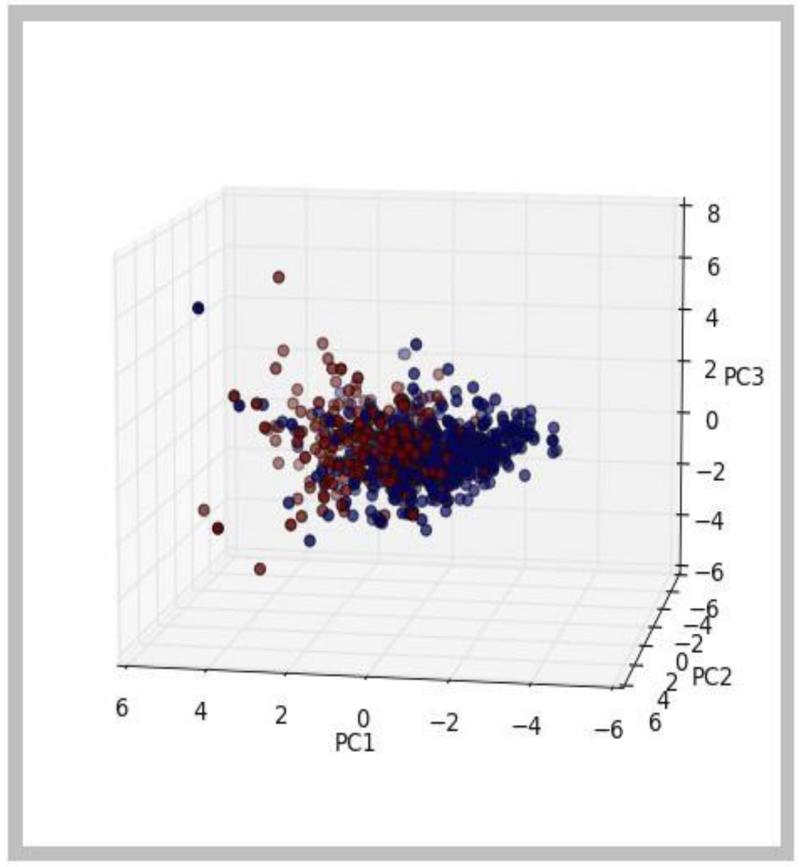
In [55]:

```
plt.figure()
sns.scatterplot(X1[:,0],X1[:,1], hue=Y, palette='deep')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Scatter Plot of First two Principal Component')
plt.show()
```



In [56]:

```
from mpl_toolkits.mplot3d import Axes3D
plt.style.use('classic')
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X1[:,0],X1[:,1], X1[:,2], c=df.Outcome, s=30)
ax.view_init(10, 100)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
plt.tight_layout()
plt.show()
```



In [57]:

```
#Model Building

#DummyClassifier
#We start with the most basic, a dummy classifier which predicts the most frequent class at all times.
#This would serve as our baseline. Split X,y into train and test sets;
#We use the original data instead of the PCA transformed data.
```

X = df_copy.iloc[:, :-1]
Y = df_copy.iloc[;, -1]

```
seed = 7
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
```

```
Shape of X_train: (614, 8)
Shape of X_test: (154, 8)
```

In [58]:

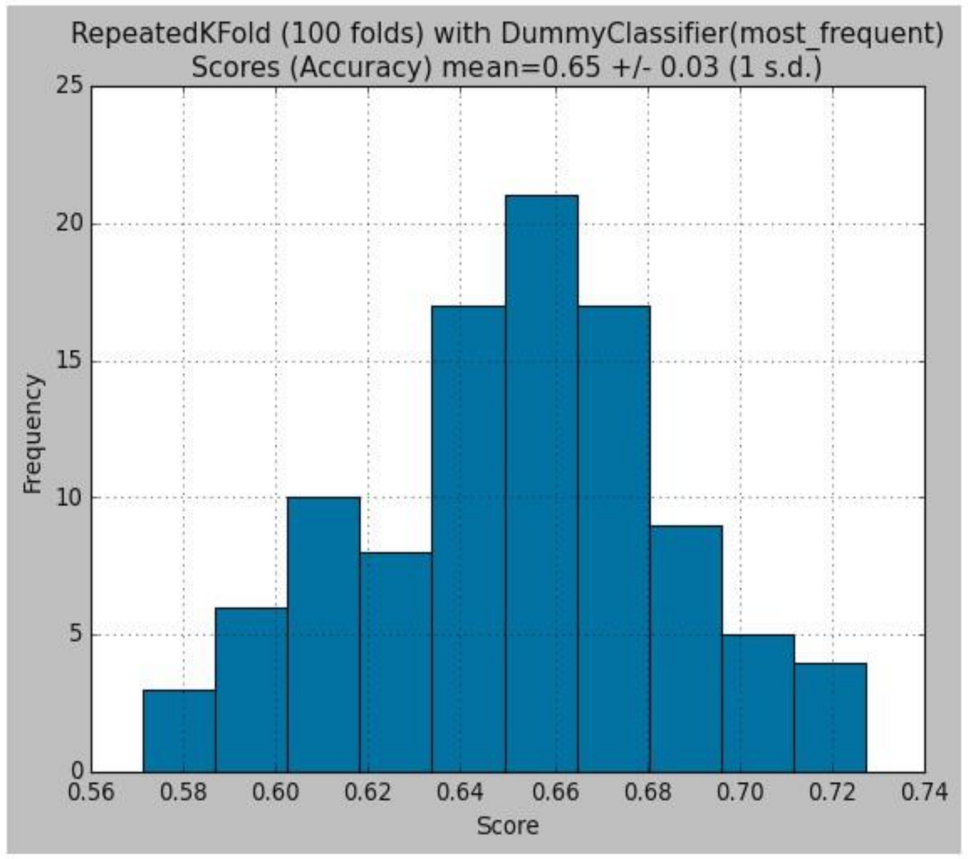
```
dum=DummyClassifier(strategy='most_frequent')
dum=dum.fit(X_train,y_train)

#compute accuracy
score=dum.score(X_test, y_test)
print("Dummy Classifier Accuracy: %.2f%%" % (score * 100.0))
```

```
Dummy Classifier Accuracy: 62.99%
```

In [59]:

```
strategy = "most_frequent"

scores = cross_val_score(dum,X, Y,
                cv=RepeatedKFold(n_repeats=CV_N_REPEATS),
                scoring=None)
scores_dummy = scores.copy()

score_line = "Scores (Accuracy) mean={0:.2f} +/- {1:.2f} (1 s.d.)".format(scores.mean(),scores.std())
plt.figure(figsize=(7,7))
fig, ax = plt.subplots()
pd.Series(scores).hist(ax=ax, bins=BINS)
ax.set_title(f"RepeatedKFold ({len(scores)} folds) with DummyClassifier({strategy})\n" + score_line);
ax.set_xlabel("Score")
ax.set_ylabel("Frequency");
```

```
<Figure size 560x560 with 0 Axes>
```



In [60]:

```
# Helper functions for graphical plotting of decision trees and to plot confusion matrix
def plot_tree_graph(model,columns,class_names):
    #This function plots the constructed decision tree
    dot_data = export_graphviz(model,feature_names=columns,class_names=class_names)
    graph = graphviz.Source(dot_data)
    return graph
```

return graph

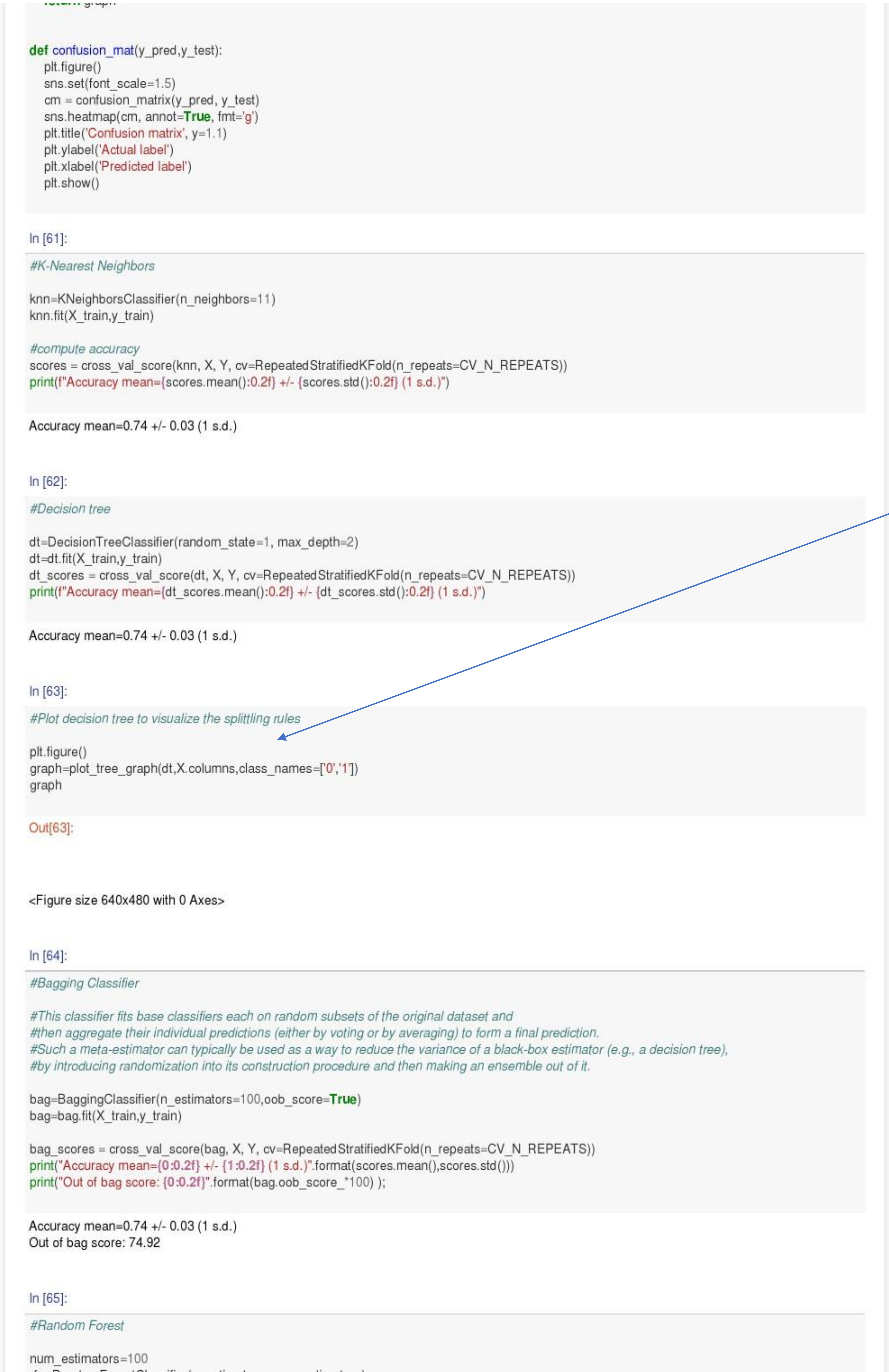
ML | Dummy classifiers using sklearn

Last Updated : 28 Nov, 2019

A dummy classifier is a type of classifier which does not generate any insight about the data and classifies the given data using only simple rules. The classifier's behavior is completely independent of the training data as the trends in the training data are completely ignored and instead uses one of the strategies to predict the class label.
It is used only as a simple baseline for the other classifiers i.e. any other classifier is expected to perform better on the given dataset. It is especially useful for datasets where are sure of a class imbalance. It is based on the philosophy that any analytic approach for a classification problem should be better than a random guessing approach.

Below are a few strategies used by the dummy classifier to predict a class label –

1. **Most Frequent:** The classifier always predicts the most frequent class label in the training data.
2. **Stratified:** It generates predictions by respecting the class distribution of the training data. It is different from the "most frequent" strategy as it instead associates a probability with each data point of being the most frequent class label.
3. **Uniform:** It generates predictions uniformly at random.
4. **Constant:** The classifier always predicts a constant label and is primarily used when classifying non-majority class labels.



```
def confusion_mat(y_pred,y_test):
    plt.figure()
    sns.set(font_scale=1.5)
    cm = confusion_matrix(y_pred, y_test)
    sns.heatmap(cm, annot=True, fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
```

In [61]:

```
#K-Nearest Neighbors

knn=KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train,y_train)

#compute accuracy
scores = cross_val_score(knn, X, Y, cv=RepeatedStratifiedKFold(n_repeats=CV_N_REPEATS))
print(f"Accuracy mean={scores.mean():0.2f} +/- {scores.std():0.2f} (1 s.d.)")
```

```
Accuracy mean=0.74 +/- 0.03 (1 s.d.)
```

In [62]:

```
#Decision tree

dt=DecisionTreeClassifier(random_state=1, max_depth=2)
dt=dt.fit(X_train,y_train)
dt_scores = cross_val_score(dt, X, Y, cv=RepeatedStratifiedKFold(n_repeats=CV_N_REPEATS))
print(f"Accuracy mean={dt_scores.mean():0.2f} +/- {dt_scores.std():0.2f} (1 s.d.)")
```

```
Accuracy mean=0.74 +/- 0.03 (1 s.d.)
```

In [63]:

```
#Plot decision tree to visualize the splittling rules

plt.figure()
graph=plot_tree_graph(dt,X.columns,class_names=['0','1'])
graph
```

Jeśli nie działa opuść

Out[63]:

```
<Figure size 640x480 with 0 Axes>
```

In [64]:

```
#Bagging Classifier

#This classifier fits base classifiers each on random subsets of the original dataset and
#then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.
#Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree),
#by introducing randomization into its construction procedure and then making an ensemble out of it.

bag=BaggingClassifier(n_estimators=100,oob_score=True)
bag=bag.fit(X_train,y_train)

bag_scores = cross_val_score(bag, X, Y, cv=RepeatedStratifiedKFold(n_repeats=CV_N_REPEATS))
print("Accuracy mean={0:0.2f} +/- {1:0.2f} (1 s.d.)".format(scores.mean(),scores.std()))
print("Out of bag score: {0:0.2f}".format(bag.oob_score_*100) );
```

```
Accuracy mean=0.74 +/- 0.03 (1 s.d.)
Out of bag score: 74.92
```

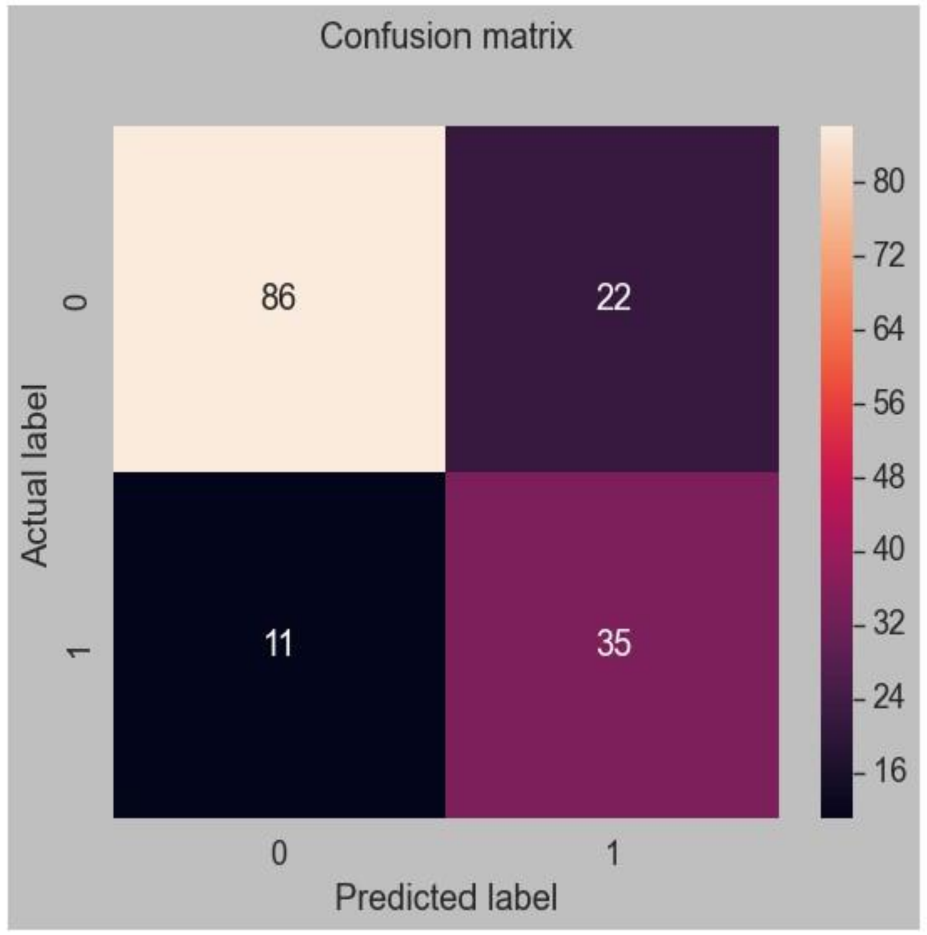
In [65]:

```
#Random Forest

num_estimators=100
```

```
rf = RandomForestClassifier(n_estimators=num_estimators)
rf.fit(X_train, y_train)

rf_score=rf.score(X_test, y_test)
print("Accuracy of Random Forest Classifier: {0:0.2f}".format(rf_score * 100.0));

#Make Predictions
y_pred = rf.predict(X_test)
#Plot the confusion matrix
confusion_mat(y_pred, y_test)
```

rf = RandomForestClassifier(n_estimators=num_estimators)

```
Accuracy of Random Forest Classifier: 78.57
```



In [66]:

```
#From the random forest model, we have 35 incorrectly labeled samples. In medical data analaysis, as it is usually the case,
#we are more concerned about the False Negatives (or Misses),
#i.e. diabetic samples who have been incorrectly labelled as non-diabetic. This model results in 15 such cases.

#We use the eli5 library to analyse which are the most important features for our learned RF model

feature_names=X_train.columns.values
show_weights(rf,feature_names=feature_names)
```

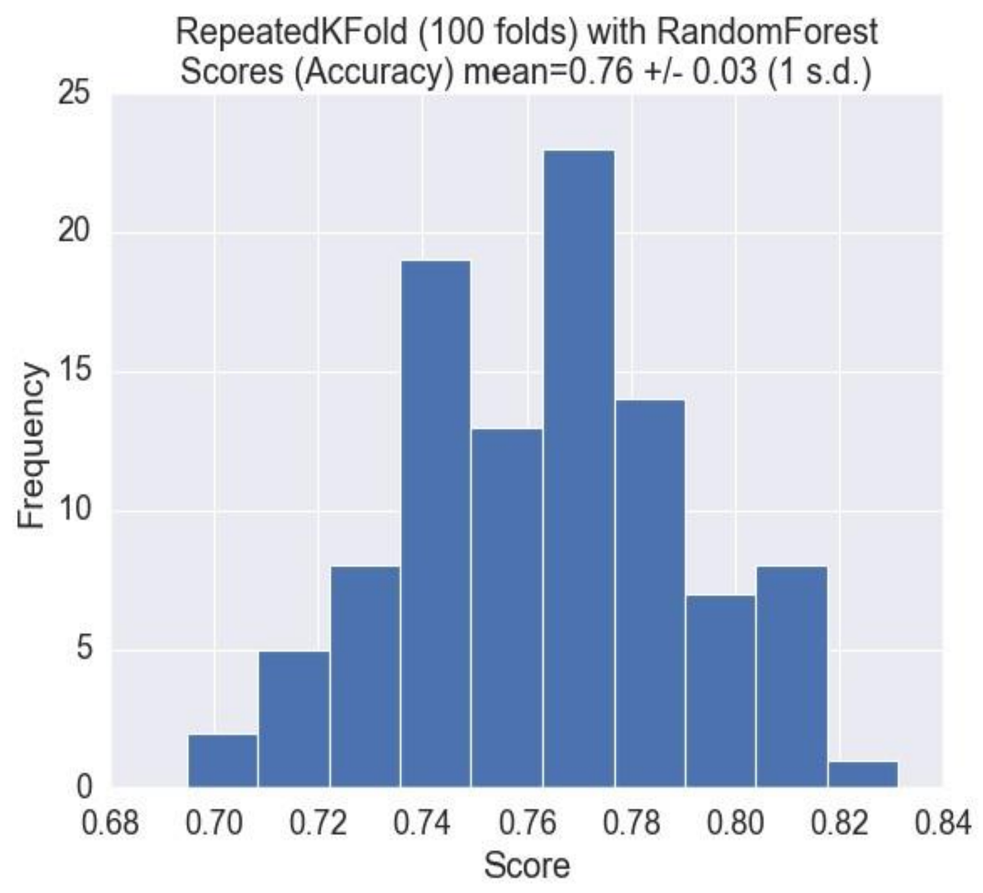
Out[66]:

Weight	Feature
0.2538 ± 0.1238	Glucose
0.1663 ± 0.0998	BMI
0.1312 ± 0.0686	DiabetesPedigreeFunction
0.1254 ± 0.0726	Age
0.0922 ± 0.0754	Insulin
0.0843 ± 0.0590	BloodPressure
0.0745 ± 0.0592	Pregnancies
0.0725 ± 0.0539	SkinThickness

In [67]:

```
#Check variance in RF prediction quality

scores = cross_val_score(rf, X, Y, cv=RepeatedStratifiedKFold(n_repeats=CV_N_REPEATS))
scores_est = scores.copy()
print(f"Scores mean={scores.mean():0.2f} +/- {scores.std():0.2f} (1 s.d.)")

score_line = f"Scores (Accuracy) mean={scores.mean():0.2f} +/- {scores.std():0.2f} (1 s.d.)"
plt.figure()
fig, ax = plt.subplots()
pd.Series(scores).hist(ax=ax, bins=BINS)
ax.set_title(f"RepeatedKFold ({len(scores)} folds) with RandomForest\n" + score_line);
ax.set_xlabel("Score")
ax.set_ylabel("Frequency");
```

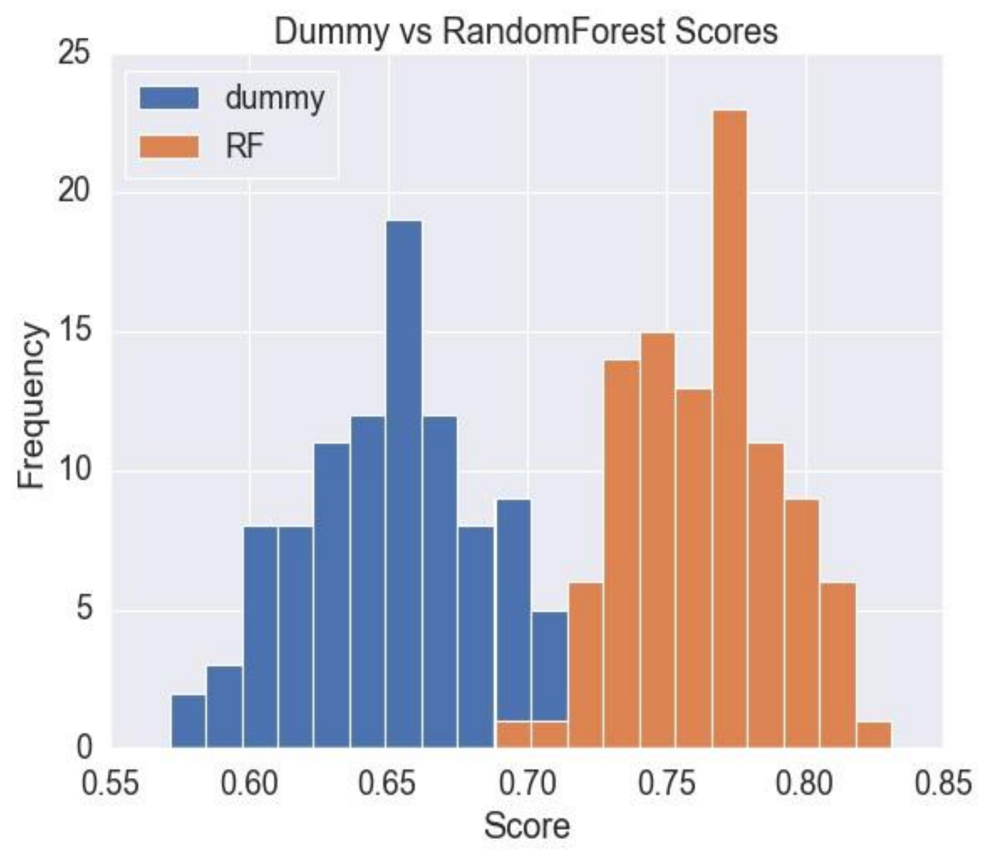
```
Scores mean=0.76 +/- 0.03 (1 s.d.)

<Figure size 640x480 with 0 Axes>
```



In [68]:

```
#We compare the prediction performance of our random forest classifier against the Dummy classifier.

plt.figure()
fig, ax = plt.subplots()
df_dummy_est_scores = pd.DataFrame({'dummy': scores_dummy, 'RF': scores_est})
df_dummy_est_scores.plot(kind='hist', ax=ax, bins=20)
ax.set_xlabel("Score")
ax.set_title("Dummy vs RandomForest Scores");
```

```
<Figure size 640x480 with 0 Axes>
```



In [69]:

```
#GradientBoostingClassifier (Sklearn)
#This is an ensemble model based on the boosting paradigm, i.e. sequential model building using several weak classifiers.
#We start with 500 estimators and a decision tree classifier of depth 4 as our weak learner.

from sklearn.metrics import mean_squared_error
params={'n_estimators': 500,'learning_rate': 0.01,'max_depth': 4, 'loss':'deviance'}
gbm=GradientBoostingClassifier(**params)
gbm.fit(X_train, y_train)
```

Out[69]:

```
GradientBoostingClassifier(learning_rate=0.01, max_depth=4, n_estimators=500)
```

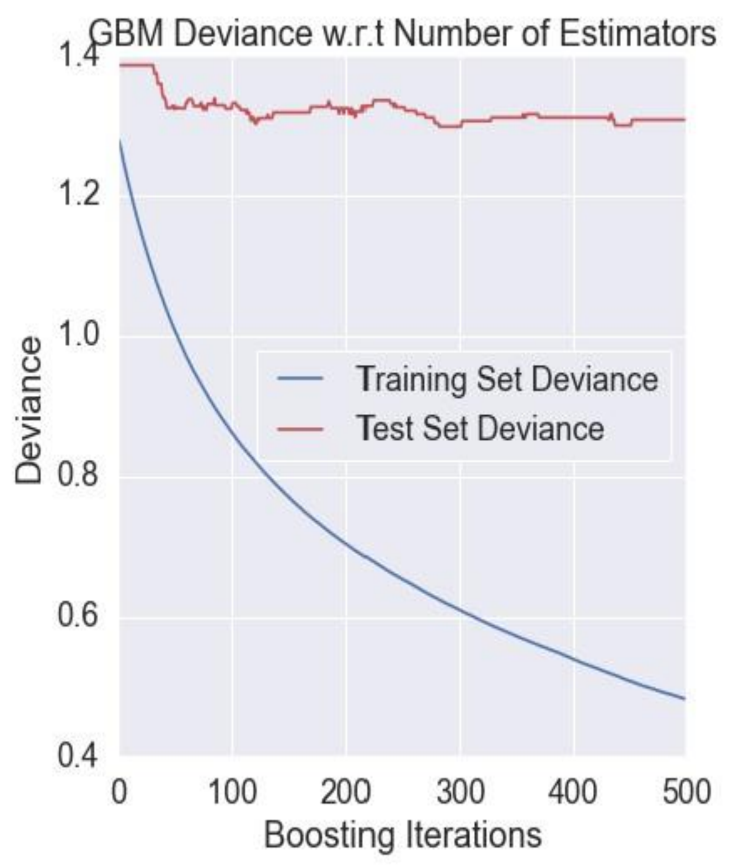
In [70]:

```
#Let's see how the model deviance performs w.r.t the number of estimators.
#Deviance is the logistic loss function used in all implementation of GBM and
#is the default loss function for classification problems.

# compute test set deviance
test_score = np.zeros((params['n_estimators'],), dtype=np.float64)

for i, y_pred in enumerate(gbm.staged_predict(X_test)):
    test_score[i] = gbm.loss_(y_test, y_pred)

#plot train and test set deviance against the number of estimators
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('GBM Deviance w.r.t Number of Estimators')
plt.plot(np.arange(params['n_estimators']) + 1, gbm.train_score_, 'b-',label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
        label='Test Set Deviance')
plt.legend(loc='best')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
```

Out[70]:

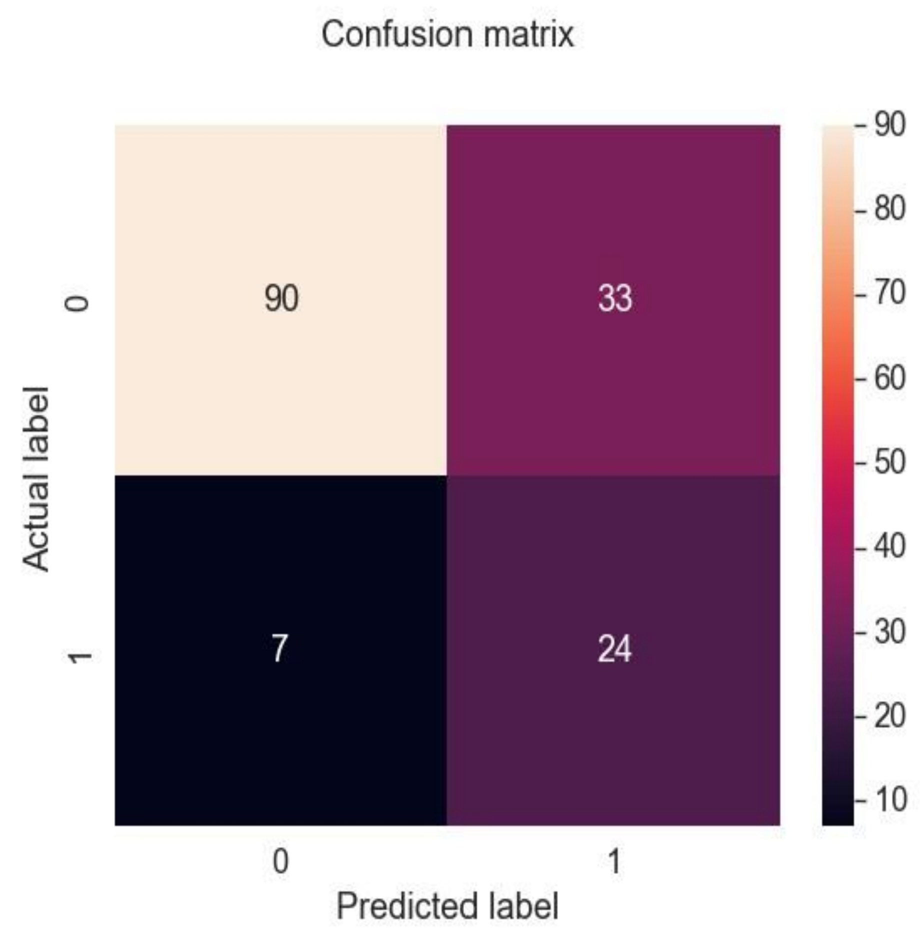
```
Text(0, 0.5, 'Deviance')
```

Alternatively change n-estimators



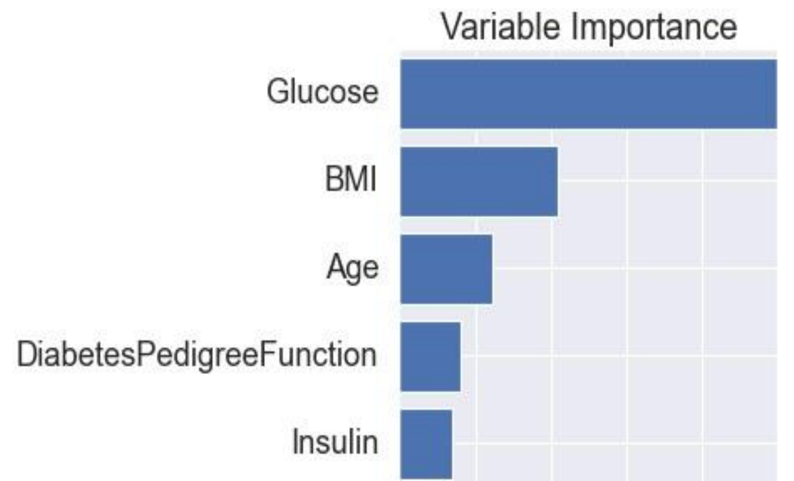
```
params={'n_estimators': 100,'learning_rate': 0.01,'max_depth': 4, 'loss':'deviance'}
gbm=GradientBoostingClassifier(**params)
gbm.fit(X_train, y_train)

# make predictions for test data
y_pred = gbm.predict(X_test)

# evaluate predictions
gbm_score = accuracy_score(y_test, y_pred)
print("Accuracy of GBM Classifier: {0:0.2f}".format(gbm_score * 100.0));

#Plot the confusion matrix
confusion_mat(y_pred, y_test)
```

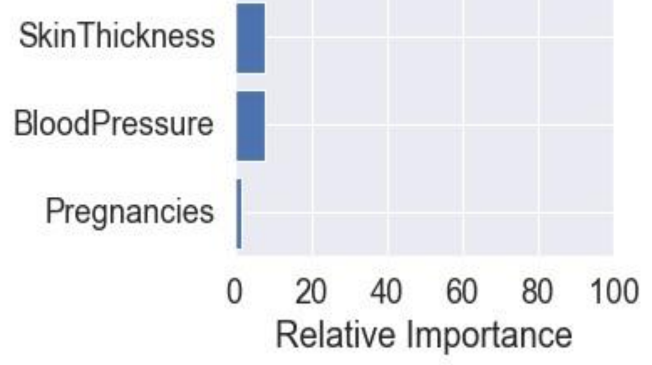
```
Accuracy of GBM Classifier: 74.03
```



In [72]:

```
#Plot Feature Importance for the GBM model

feature_importance = gbm.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, df.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```





In [73]:

```
#XgBoost
#This model is an optimized variant of the Gradient boosting models,
#which at its core does the same work as the previous Gradient Boosting machine does.
#The difference is that XgBoost algorithm is developed with both deep consideration in terms of systems optimization
#and principles in machine learning. The goal of the library is to push the extreme of the computation limits of machines
#to provide a scalable, portable and accurate library.

from xgboost import XGBClassifier, plot_importance,to_graphviz

# fit model on training data
param = {'max_depth': 3, 'eta': 0.8, 'subsample':1, 'objective': 'binary:logistic'}
xgb = XGBClassifier(**param)
xgb.fit(X_train, y_train)
```

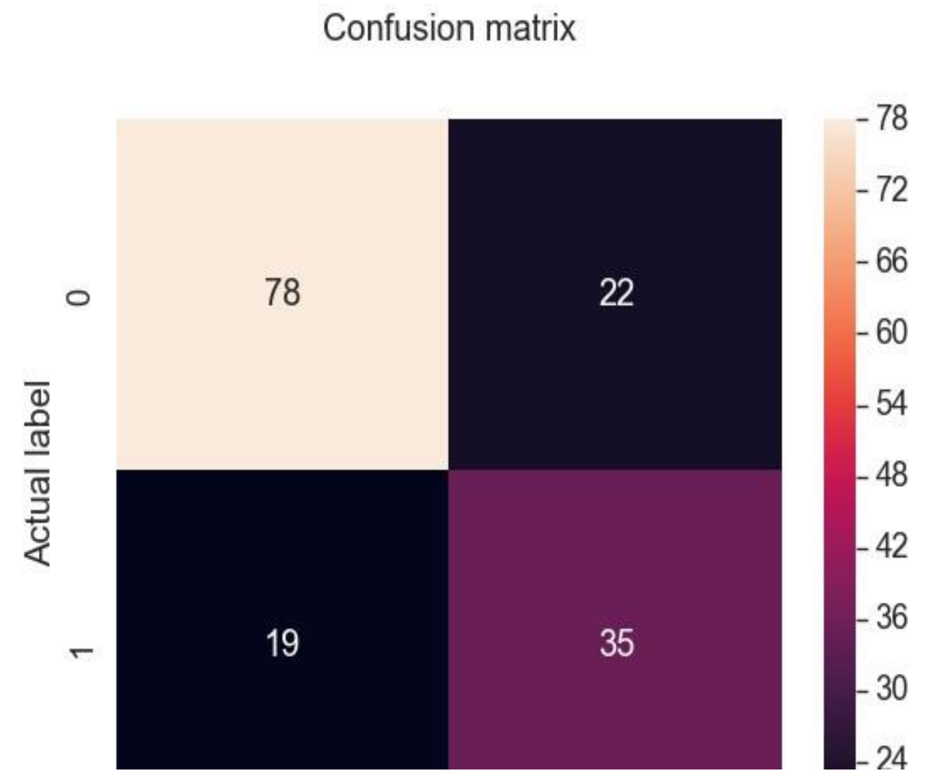
Out[73]:

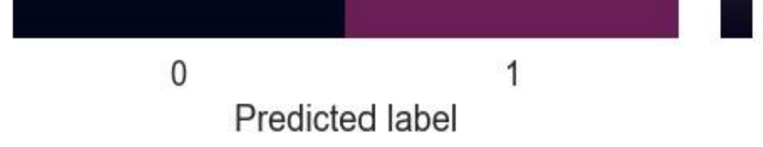
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.8, gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraints='',
              learning_rate=0.800000012, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [74]:

```
# make predictions for test data
y_pred = xgb.predict(X_test)

# evaluate predictions
xgb_score = accuracy_score(y_test, y_pred)
print("Accuracy of XGB Classifier: {0:0.2f}".format(xgb_score * 100.0));

#Plot the confusion matrix
confusion_mat(y_pred, y_test)
```

```
Accuracy of XGB Classifier: 73.38
```

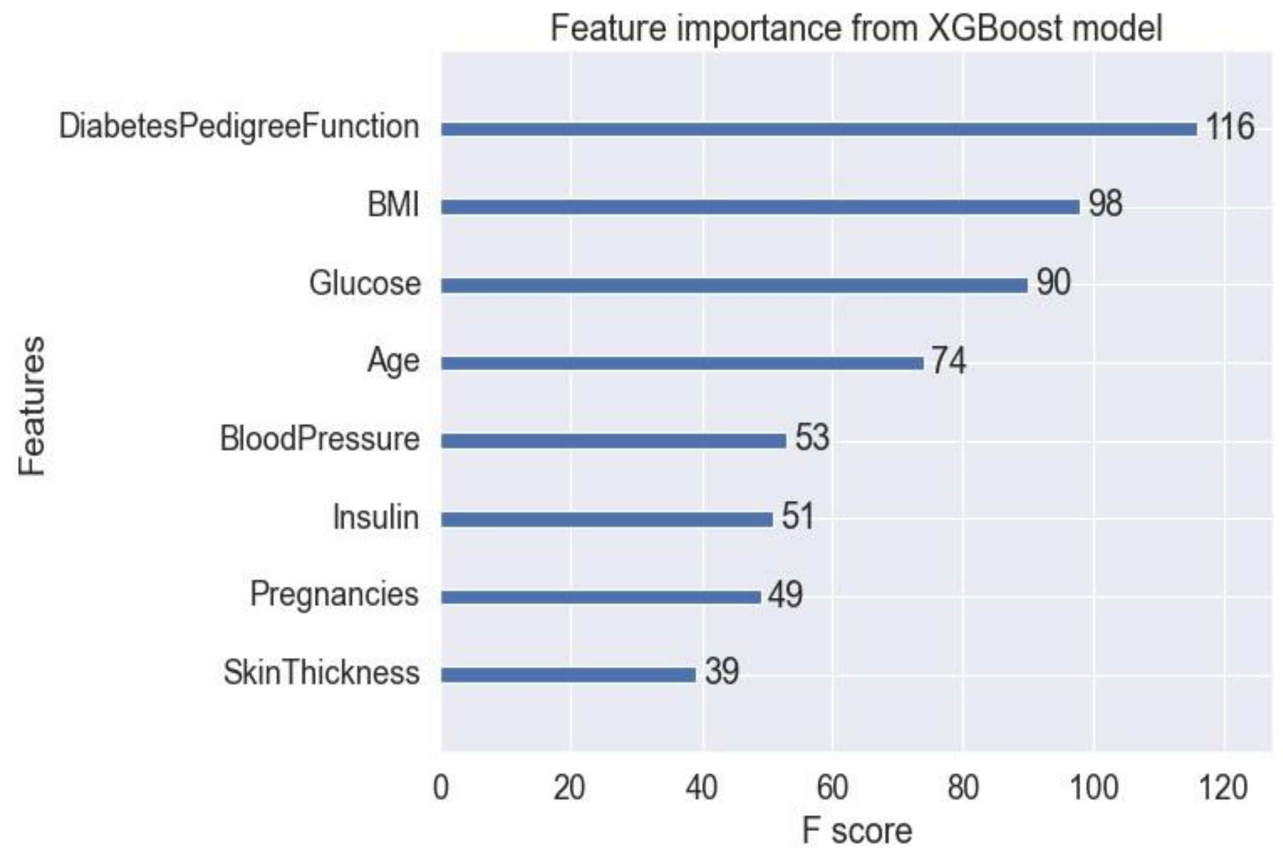




In [75]:

```
#XGBoost in most cases performs better than GBM. As we can see here the classification accuracy has increased to 83.12% than
#compared to GBM's 74%, but the Misses have increased from 7 in GBM to 12 with Xgboost model.

#Next we plot Feature Importance based on the Xgboost model.
#From the feature importance graphs for all models plotted until now, we see Glucose is the most important feature.
#Importance order for the other features is more or less same.

# plot feature importance using built-in function
plt.figure()
plot_importance(xgb,title="Feature importance from XGBoost model")
plt.show()
```

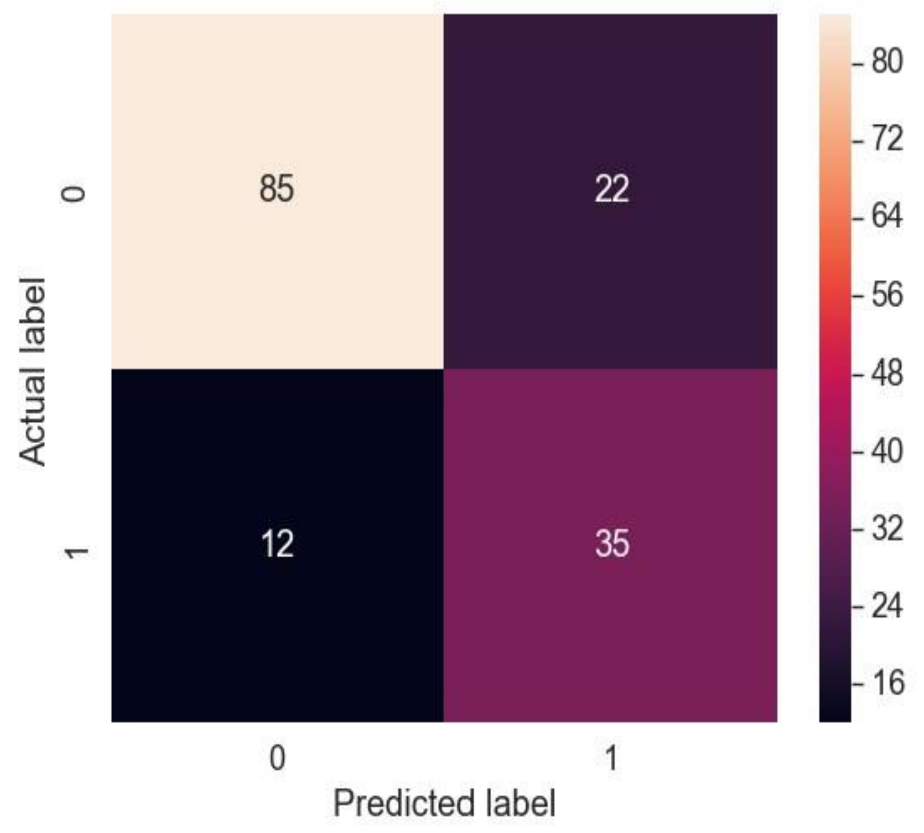
```
<Figure size 640x480 with 0 Axes>
```



In [76]:

```
#Voting Classifier
#The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and
#use a majority vote or the average predicted probabilities (soft vote) to predict the class labels.
#Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses.

#We create a voting classifier using three models: KNN, Random Forest, and XGBoost model.
#Results don't show any improvement over the three models.

from sklearn.ensemble import VotingClassifier

ensemble_knn_rf_xgb=VotingClassifier(estimators= [('KNN', knn), ('Random Forest', rf),('XGBoost',xgb)], voting='hard')
ensemble_knn_rf_xgb.fit(X_train,y_train)

#compute accuracy
print('The ensembled model with all the 3 classifiers is:',ensemble_knn_rf_xgb.score(X_test,y_test))

#make predictions
y_pred = ensemble_knn_rf_xgb.predict(X_test)
#Plot the confusion matrix
confusion_mat(y_pred, y_test)
```

```
The ensembled model with all the 3 classifiers is: 0.7792207792207793
```



In [82]:

```
pip install mlxtend
```

```
Collecting mlxtendNote: you may need to restart the kernel to use updated packages.

  Downloading mlxtend-0.17.3-py2.py3-none-any.whl (1.3 MB)
Requirement already satisfied: scikit-learn>=0.20.3 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (0.23.2)
Requirement already satisfied: joblib>=0.13.2 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (0.16.0)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (3.3.1)
Requirement already satisfied: scipy>=1.2.1 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (1.5.2)
Requirement already satisfied: pandas>=0.24.2 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (1.0.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (1.19.1)
Requirement already satisfied: setuptools in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (49.6.0.post20200814)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn>=0.20.3->mlxtend) (2.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2020.6.20)
Requirement already satisfied: pillow>=6.2.0 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (7.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxte
nd) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.2.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\user\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend) (2020.1)
Requirement already satisfied: six in c:\users\user\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.15.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.17.3
```

In [83]:

```
#Stacking

#Stacking is a way of combining multiple models, that introduces the concept of a meta learner.
#It is less widely used than bagging and boosting. Unlike bagging and boosting,
#stacking may be (and normally is) used to combine models of different types.

#The point of stacking is to explore a space of different models for the same problem.
#The idea is that you can attack a learning problem with different types of models which are capable to learn some part
#of the problem, but not the whole space of the problem.
#So you can build multiple different learners and you use them to build an intermediate prediction,
#one prediction for each learned model. Then you add a new model which learns from the intermediate predictions the same target.
#This final model is said to be stacked on the top of the others, hence the name.
#Thus you might improve your overall performance, and often you end up with a model which is better than
#any individual intermediate model.

from mlxtend.classifier import StackingCVClassifier

sclf = StackingCVClassifier(classifiers=[knn, rf, xgb, gbm],
                    meta_classifier=rf)

print('10-fold cross validation:\n')
```

```
for clf, label in zip([knn, rf, xgb, gbm, rf],
                  ['KNearest Neighbors',
                   'Random Forest',
                    'XGB','GBM',
                   'MetaClassifier']):

    sclf_scores = model_selection.cross_val_score(clf, X, Y,
                                    cv=10, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (sclf_scores.mean(), sclf_scores.std(), label))
```

```
10-fold cross validation:

Accuracy: 0.74 (+/- 0.05) [KNearest Neighbors]
Accuracy: 0.76 (+/- 0.05) [Random Forest]
Accuracy: 0.71 (+/- 0.04) [XGB]
Accuracy: 0.75 (+/- 0.04) [GBM]
Accuracy: 0.77 (+/- 0.05) [MetaClassifier]
```

In [84]:

```
#Summarize results

#We run the algorithms once again using StratifiedK-fold cross-validation and summarize our findings.

models = []
#models.append(('LR', LogisticRegression()))
models.append(('KNN', knn))
models.append(('DT', dt))
models.append(('RF', rf))
models.append(('GBM', gbm))
models.append(('XGB', xgb))
models.append(('Voting',ensemble_knn_rf_xgb))
```

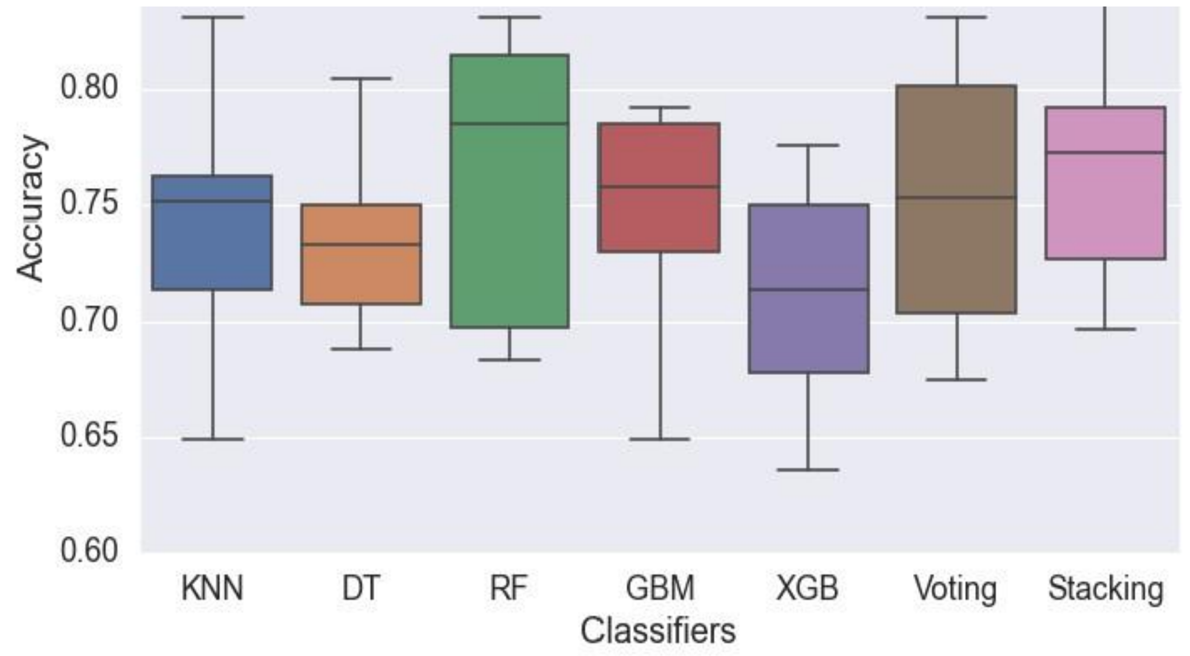
In [85]:

```
#Every algorithm is tested and
#results are collected and printed. We then visualise the variation in the predictions of each algorithm using a boxplot.

results = []
names = []

for name, model in models:
   kfold = model_selection.StratifiedKFold(n_splits=10, random_state=7)
   cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring='accuracy')
   results.append(cv_results)
   names.append(name)
   msg = "{}: {} ({})".format(name, cv_results.mean(), cv_results.std())
   print(msg)

#Add stacking results that we got previously
results.append(np.asarray(sclf_scores))
names.append('Stacking')
```

```
KNN: 0.7422761449077239 (0.050530447112230446)
DT: 0.7369958988380041 (0.0361859969732375)
RF: 0.7643028024606972 (0.05793239373200191)
GBM: 0.7486671223513329 (0.04251329136200218)
XGB: 0.7109706083390294 (0.044846299666432325)
Voting: 0.7539131920710869 (0.05381046766538876)
```

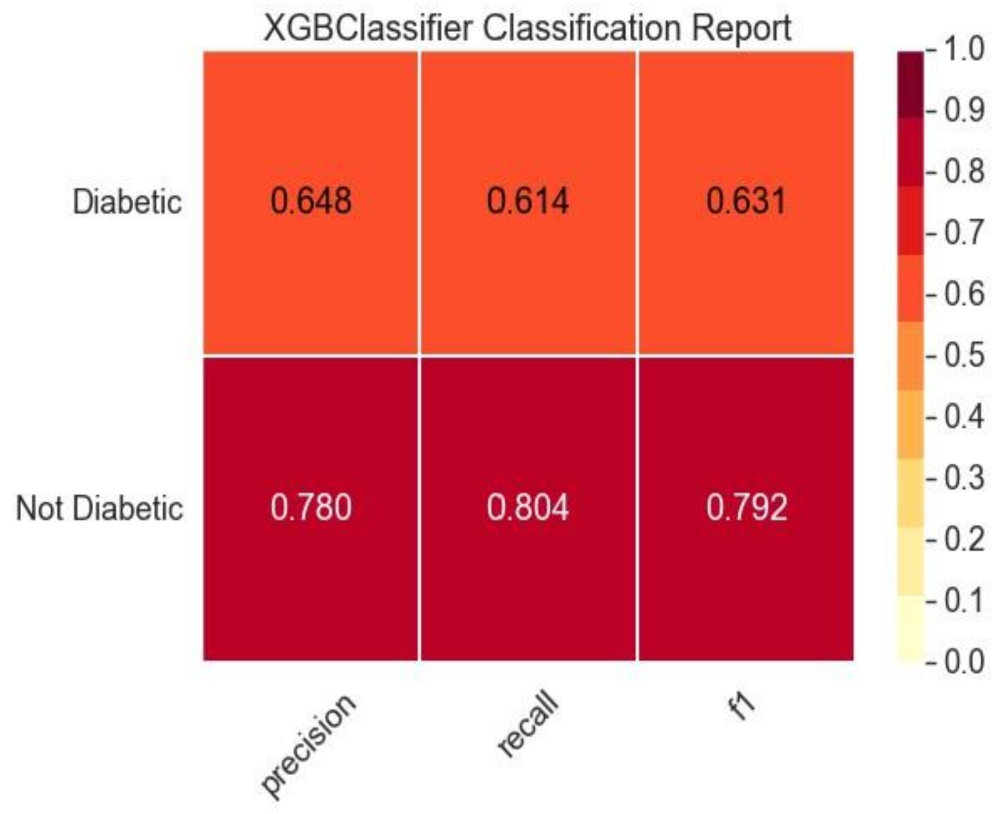
In [86]:

```
# boxplot algorithm comparison
fig = plt.figure(figsize=(10,6))
fig.suptitle('Algorithm Comparison')
ax = sns.boxplot(x=names, y=results)
plt.xlabel('Classifiers')
plt.ylabel('Accuracy')
plt.show()
```



In [90]:

```
"""We obtained a best on average classification accuracy of approximately 77% using XgBoost and the Voting Classifier. We noted earlier from the c
onfusion matrix of XGboost classifier that we there are 26 misclassified samples with 12 misses. One can further analyse those misclassified sample
s to better understand the model behaviorb. This is an extension that is not part of this notebook.

We end the analysis with classification report of our two best performing models.The classification report shows a representation of the main classific
ation metrics on a per-class basis. This gives a deeper intuition of the classifier behavior over global accuracy which can mask functional weaknesse
s in one class of a multiclass problem. Visual classification reports are used to compare classification models to select models that are "redder", e.g.
have stronger classification metrics or that are more balanced.

Precision is the ability of a classiifer not to label an instance positive that is actually negative. For each class it is defined as as the ratio of true positiv
es to the sum of true and false positives.

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives an
d false negatives.

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.

Based on the weighted F-1 score from the below two reports, XgBoost is a better classifier. So was the case when we considered number of False n
egatives samples....

visualizer = ClassificationReport(xgb,classes=['Not Diabetic','Diabetic'])"""

visualizer = ClassificationReport(xgb,classes=['Not Diabetic','Diabetic'])

#visualizer.fit(X_train, y_train)  # Fit the visualizer and the model
visualizer.score(X_test, y_test)   # Evaluate the model on the test data
visualizer.poof()                  # Draw/show/poof the data
```

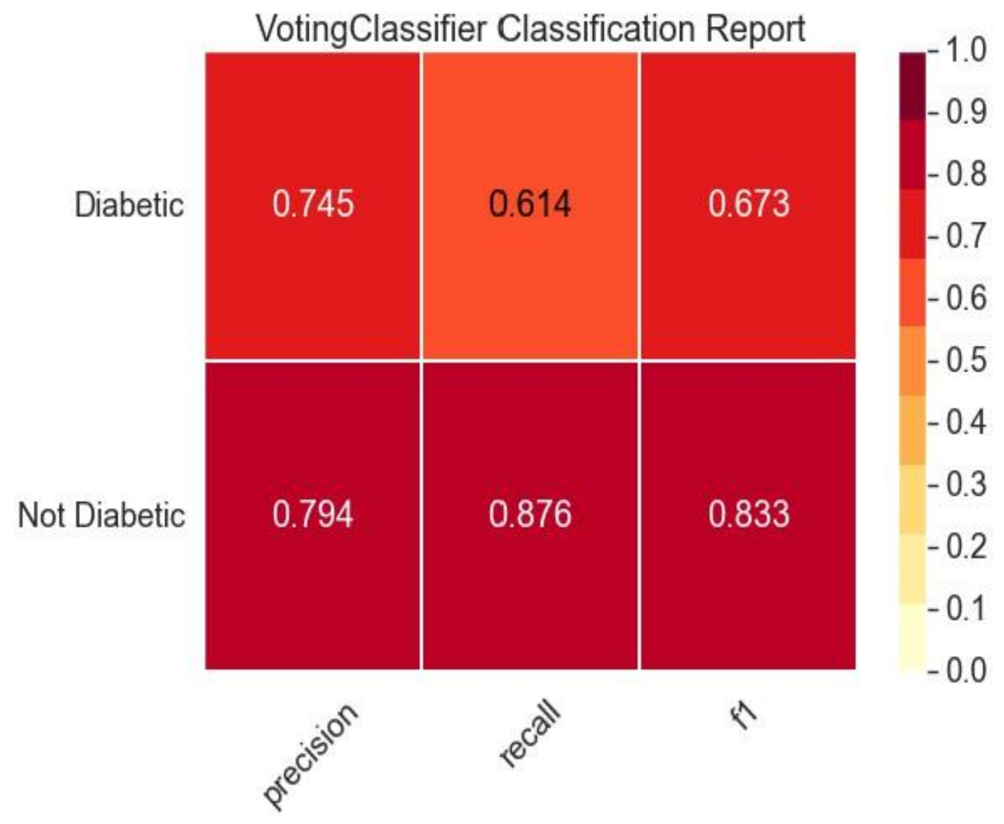


Out[90]:

```
<AxesSubplot:title={'center':'XGBClassifier Classification Report'}>
```

In [91]:

```
visualizer = ClassificationReport(ensemble_knn_rf_xgb,classes=['Not Diabetic','Diabetic'])

visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.poof()                 # Draw/show/poof the data
```



Out[91]:

```
<AxesSubplot:title={'center':'VotingClassifier Classification Report'}>
```

In [ ]:

What is the F-score?

The F-score, also called the F1-score, is a measure of a model's accuracy on a dataset. It is used to evaluate binary classification systems, which classify examples into 'positive' or 'negative'.

The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall.

The F-score is commonly used for evaluating information retrieval systems such as search engines, and also for many kinds of machine learning models, in particular in natural language processing.

It is possible to adjust the F-score to give more importance to precision over recall, or vice-versa. Common adjusted F-scores are the F0.5-score and the F2-score, as well as the standard F1-score.

F-score Formula

The formula for the standard F1-score is the harmonic mean of the precision and recall. A perfect model has an F-score of 1.

$$\begin{aligned} F_1 &= \frac{2}{\frac{1}{\mathrm{recall}} \times \frac{1}{\mathrm{precision}}} = 2 \times \frac{\mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \\ &= \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})} \end{aligned}$$

Mathematical definition of the F-score

F-score Formula Symbols Explained

precision	Precision is the fraction of true positive examples among the examples that the model classified as positive. In other words, the number of true positives divided by the number of false positives plus true positives.
recall	Recall, also known as sensitivity, is the fraction of examples classified as positive, among the total number of positive examples. In other words, the number of true positives divided by the number of true positives plus false negatives.
tp	The number of true positives classified by the model.
fn	The number of false negatives classified by the model.
fp	The number of false positives classified by the model.