FETAL HEALTH REPORT

March 25, 2022

Reduction of child mortality is reflected in several of the United Nations' Sustainable Development Goals and is a key indicator of human progress. * The UN expects that by 2030, countries end preventable deaths of newborns and children under 5 years of age, with all countries aiming to reduce under-5 mortality to at least as low as 25 per 1,000 live births. Parallel to notion of child mortality is of course maternal mortality, which accounts for 295.000 deaths during and following pregnancy and childbirth (as of 2017). The vast majority of these deaths (94%) occurred in low-resource settings, and most could have been prevented. In light of what was mentioned above, Cardiotocograms (CTGs) are a simple and cost accessible option to assess fetal health, allowing healthcare professionals to take action in order to prevent child and maternal mortality. The equipment itself works by sending ultrasound pulses and reading its response, thus shedding light on fetal heart rate (FHR), fetal movements, uterine contractions and more.

Introduction: One of the important data insight extracting methods in biomedical research are classification algorithms. Classification algorithms can be divided into two categories: binary and multi-class classifiers. The interest of this project is focused on a multi-class problem identifying and classifying Normal, Suspect, and Pathological fetuses.

Background: Reduction of child mortality is reflected in several of the United Nations' Sustainable Development Goals and is a key indicator of human progress. The UN expects that by 2030, countries end preventable deaths of newborns and children under 5 years of age, with all countries aiming to reduce under-5 mortality to at least as low as 25 per 1,000 live births. Parallel to notion of child mortality is maternal mortality, which accounts for 295,000 deaths during and following pregnancy and childbirth (as of 2017). The vast majority of these deaths (94%) occurred in low-resource settings, and most could have been prevented. In light of what was mentioned above, Cardiotocograms (CTGs) are a simple and cost accessible option to assess fetal health, allowing healthcare professionals to take action in order to prevent child and maternal mortality. The equipment itself works by sending ultrasound pulses and reading its response, thus shedding light on fetal heart rate (FHR), fetal movements, uterine contractions and more.

Purposes: The purpose of this analysis is to satisfy Coursera's/IBM Supervised Learning: Classification course final project requirements.

Attributes: This dataset contains 2,216 records/observations of patient Cardiotocogram exams, which were then classified by three expert obstetricians into one of the following three classes: 1) Normal, 2) Suspect, and 3) Pathological. Moreover, the data was tidy and did not necessitate cleaning. Additionally, there was a significant class imbalance, but that was remedied by utilizing the Synthetic Minority Oversampling Technique (SMOTE) on the training set. Furthermore, the dataset contained 22 feature columns including; baseline values, fetal movements, histogram features, etc.

Limitations: A glaring limitation with this dataset is the small amount of observations, 2,126, available. To give context the UN estimates that roughly 385,000 children are born daily, thus the observations in this dataset account for 0.55% of possible observations on a daily basis. Therefore, with more data it is possible that new trends could be uncovered or that particular features become more/less important in reference to what was observed during my analysis.

Methods: The following models were fit to classify the data into the three aforementioned classes: Logistic Regression, K-Nearest Neighbors, Decision Tree, Random Forest, XGBoost, ADABoost, and a Voting Classifier.

Data

The dataset contains 2126 records of features extracted from Cardiotocogram exams, which were then classified by three expert obstetricians into 3 classes:

- 1 = Normal
- 2 = Suspect
- 3 = Pathological

Questions to be answered

1. Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation and the benefits that your analysis provides to the business or stakeholders of this data.
2. Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.
3. Brief summary of data exploration and actions taken for data cleaning and feature engineering.
4. Summary of training at least three different classifier models, preferably of different nature in explainability and predictability. For example, you can start with a simple logistic regression as a baseline, adding other models or ensemble models. Preferably, all your models use the same training and test splits, or the same cross-validation method.
5. A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.
6. Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classifier model.
7. Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model after adding specific data features that may help you achieve a better explanation or a better prediction.

Executive: A Voting Classifier model, that was Grid Searched and incorporates Synthetic Minority Oversampling Technique (SMOTE), provided the best overall metrics regarding classifying the three classes; Normal, Suspect, and Pathological. The overall metrics are as follows: Accuracy: 92%, Precision: 93%, Recall: 92%, and F1: 92%.

Results: Important to note that while the Random Forest model, that was Grid Searched and incorporated Synthetic Minority Oversampling Technique (SMOTE), provided the best raw metrics for Accuracy, Recall, and F1 Score the model still resulted in classifying 2 Pathological fetuses as Normal. Given that this is healthcare related data a False Negative, or type 2 error, is egregious and not acceptable when preforming diagnostic testing of this nature. Thus, the following visualizations will provide a ROC-AUC graph, Confusion Matrix, and Accuracy/ Precision/Recall/F1 scores for the best overall model.

Recommendations: For future analysis it would be interesting to compare Principle Component Analysis and Linear Discriminant Analysis dimension reduction techniques in attempt to further separate the data. Then train the models again on dimensionally reduced data for comparison. Furthermore, as previously mentioned obtaining more data could result in uncovering new trends and or resulting in particular features become more/less important.

Prevent_child_and_maternal_mortality

March 25, 2022

0.0.1 Import Libraries/Data

[81]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import itertools
import xgboost as xgb
import shap
import warnings
%matplotlib inline

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold,␣
 ↪train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier,␣
 ↪GradientBoostingClassifier, AdaBoostClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, f1_score, recall_score
from yellowbrick.classifier import ROCAUC
from xgboost import plot_importance
from imblearn.over_sampling import SMOTE

warnings.filterwarnings('ignore')
```

[2]:
```
df = pd.read_csv('fetal_health.csv')
```

0.0.2 EDA

[3]:
```
df.describe().T
```

[3]:
```
                                                          count        mean  \
baseline value                                           2126.0  133.303857
```

accelerations	2126.0	0.003178
fetal_movement	2126.0	0.009481
uterine_contractions	2126.0	0.004366
light_decelerations	2126.0	0.001889
severe_decelerations	2126.0	0.000003
prolongued_decelerations	2126.0	0.000159
abnormal_short_term_variability	2126.0	46.990122
mean_value_of_short_term_variability	2126.0	1.332785
percentage_of_time_with_abnormal_long_term_vari…	2126.0	9.846660
mean_value_of_long_term_variability	2126.0	8.187629
histogram_width	2126.0	70.445908
histogram_min	2126.0	93.579492
histogram_max	2126.0	164.025400
histogram_number_of_peaks	2126.0	4.068203
histogram_number_of_zeroes	2126.0	0.323612
histogram_mode	2126.0	137.452023
histogram_mean	2126.0	134.610536
histogram_median	2126.0	138.090310
histogram_variance	2126.0	18.808090
histogram_tendency	2126.0	0.320320
fetal_health	2126.0	1.304327

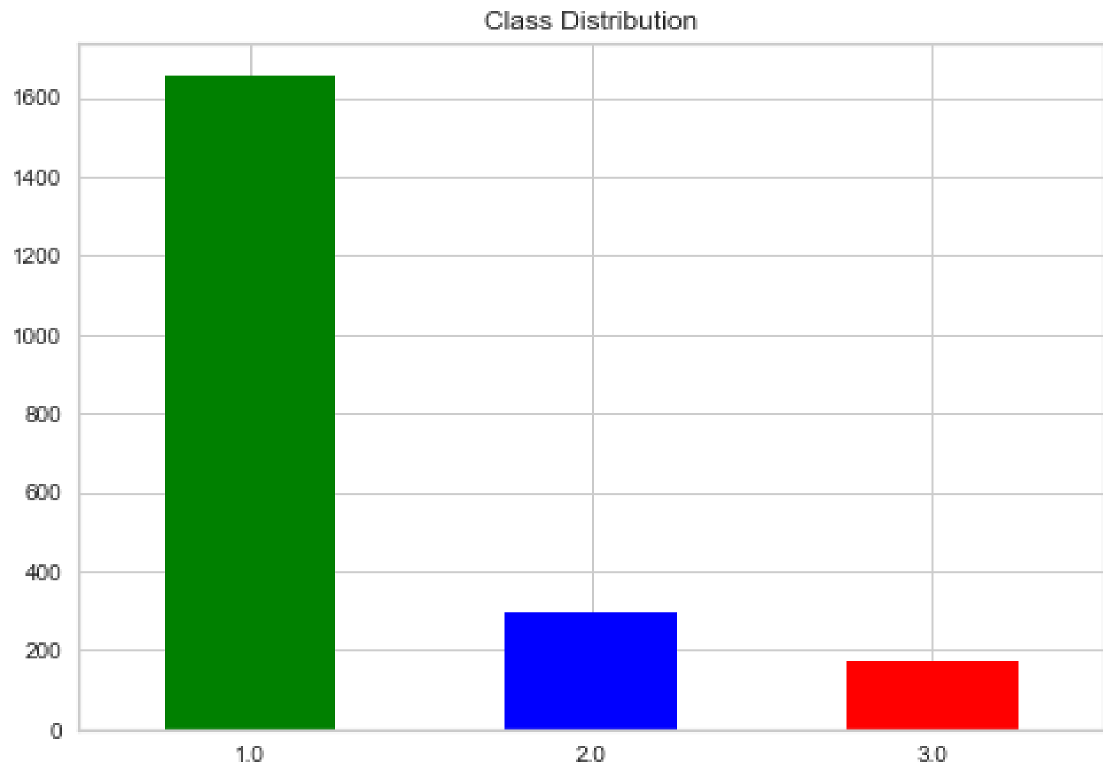
	std	min	25%	\
baseline value	9.840844	106.0	126.000	
accelerations	0.003866	0.0	0.000	
fetal_movement	0.046666	0.0	0.000	
uterine_contractions	0.002946	0.0	0.002	
light_decelerations	0.002960	0.0	0.000	
severe_decelerations	0.000057	0.0	0.000	
prolongued_decelerations	0.000590	0.0	0.000	
abnormal_short_term_variability	17.192814	12.0	32.000	
mean_value_of_short_term_variability	0.883241	0.2	0.700	
percentage_of_time_with_abnormal_long_term_vari…	18.396880	0.0	0.000	
mean_value_of_long_term_variability	5.628247	0.0	4.600	
histogram_width	38.955693	3.0	37.000	
histogram_min	29.560212	50.0	67.000	
histogram_max	17.944183	122.0	152.000	
histogram_number_of_peaks	2.949386	0.0	2.000	
histogram_number_of_zeroes	0.706059	0.0	0.000	
histogram_mode	16.381289	60.0	129.000	
histogram_mean	15.593596	73.0	125.000	
histogram_median	14.466589	77.0	129.000	
histogram_variance	28.977636	0.0	2.000	
histogram_tendency	0.610829	-1.0	0.000	
fetal_health	0.614377	1.0	1.000	

	50%	75%	max

```
baseline value                                    133.000  140.000  160.000
accelerations                                       0.002    0.006    0.019
fetal_movement                                      0.000    0.003    0.481
uterine_contractions                                0.004    0.007    0.015
light_decelerations                                 0.000    0.003    0.015
severe_decelerations                                0.000    0.000    0.001
prolongued_decelerations                            0.000    0.000    0.005
abnormal_short_term_variability                    49.000   61.000   87.000
mean_value_of_short_term_variability                1.200    1.700    7.000
percentage_of_time_with_abnormal_long_term_vari…  0.000   11.000   91.000
mean_value_of_long_term_variability                 7.400   10.800   50.700
histogram_width                                    67.500  100.000  180.000
histogram_min                                      93.000  120.000  159.000
histogram_max                                     162.000  174.000  238.000
histogram_number_of_peaks                           3.000    6.000   18.000
histogram_number_of_zeroes                          0.000    0.000   10.000
histogram_mode                                    139.000  148.000  187.000
histogram_mean                                    136.000  145.000  182.000
histogram_median                                  139.000  148.000  186.000
histogram_variance                                  7.000   24.000  269.000
histogram_tendency                                  0.000    1.000    1.000
fetal_health                                        1.000    1.000    3.000
```

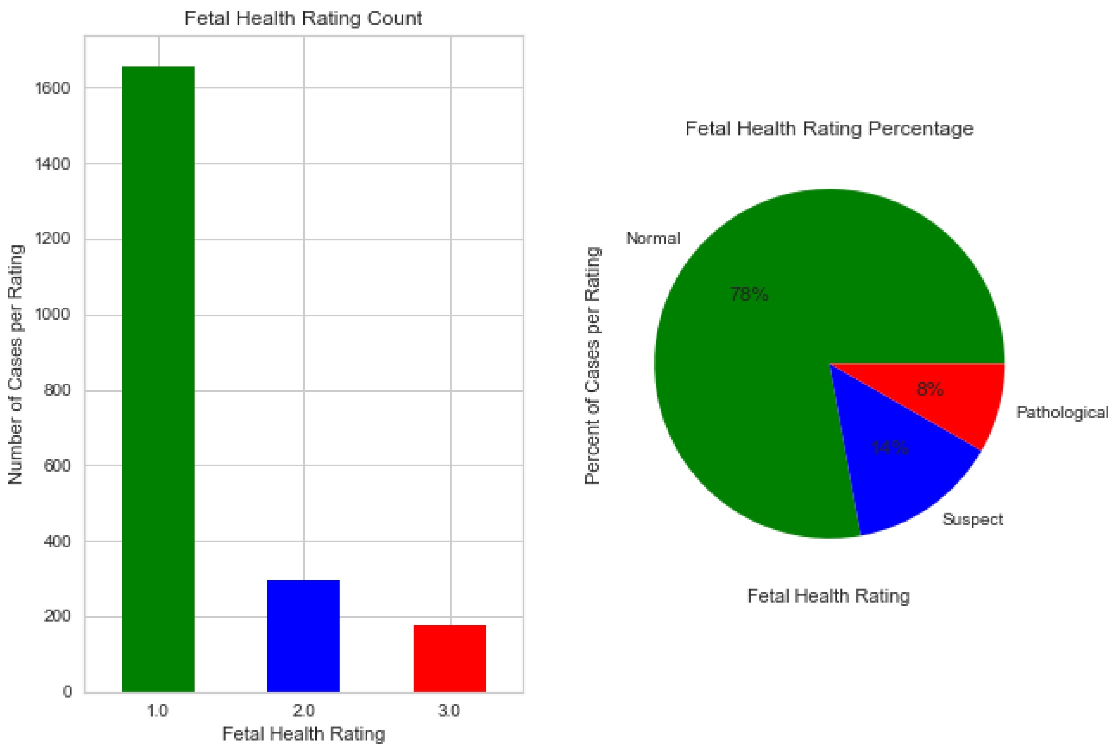
[4]:

```
normal = df[df['fetal_health'] == 1]['fetal_health'].count()
suspect = df[df['fetal_health'] == 2]['fetal_health'].count()
pathological = df[df['fetal_health'] == 3]['fetal_health'].count()

df1=df
df1['fetal_health'].replace(to_replace=1, value="normal")
df1['fetal_health'].replace(to_replace=2, value="suspect")
df1['fetal_health'].replace(to_replace=3, value="pathological")

df1['fetal_health'].value_counts().plot.bar(rot=0,title="Class␣
↪Distribution",color=['green', 'blue','red'])
plt.show()
```

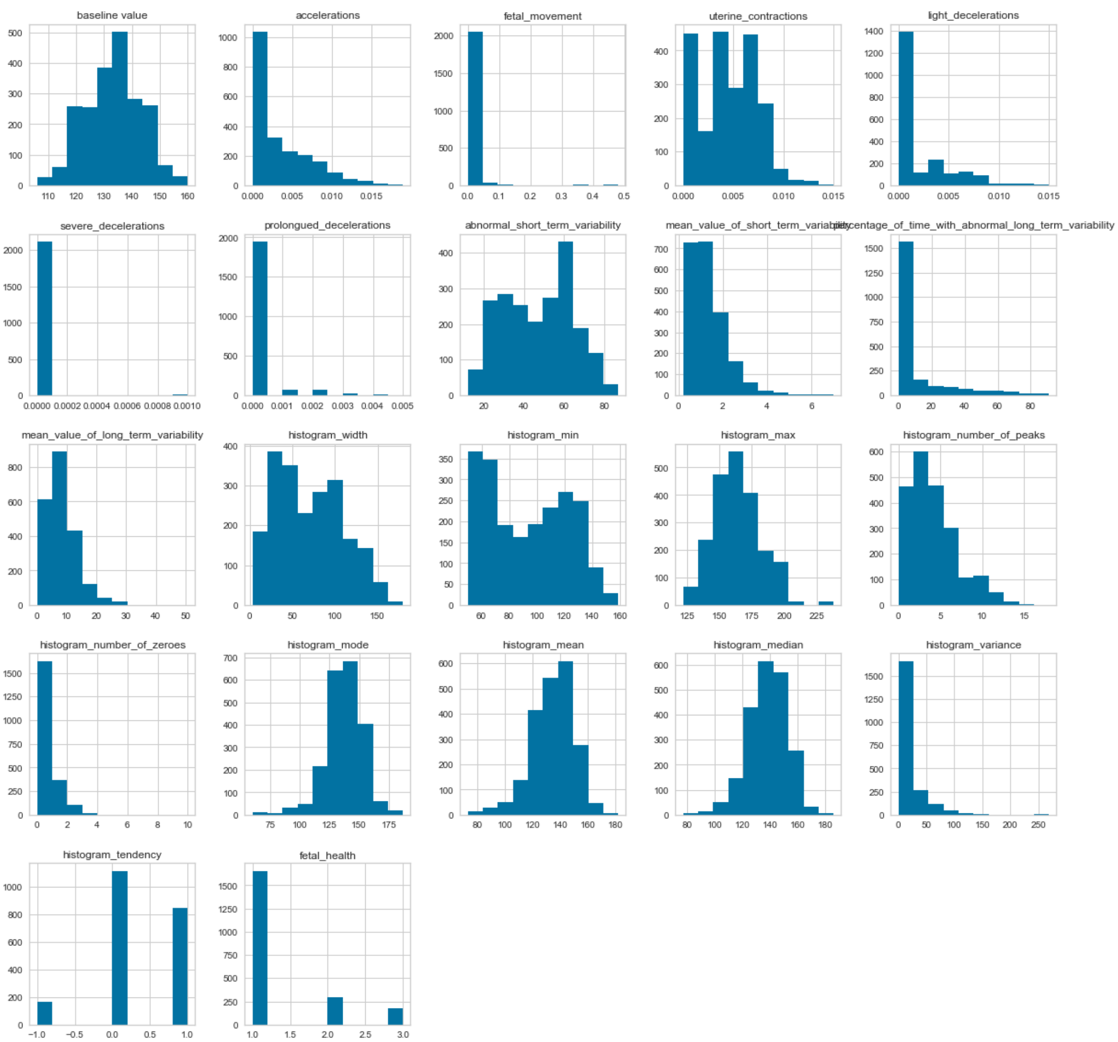


[5]:
```
# Barplot and Piechart to visual Fetal Health Rating counts

plt.figure(figsize = (10,7))
plt.subplot(121)

df_visual_bar = df1.fetal_health.value_counts().plot(rot=0,
    figsize=(10, 7), kind='bar', color = ['green', 'blue','red'])
plt.title('Fetal Health Rating Count')
plt.xlabel('Fetal Health Rating')
plt.ylabel('Number of Cases per Rating')


plt.subplot(122)
plt.title('Fetal State')

df_visual_pie = plt.pie(
    [normal, suspect, pathological], labels=[
    'Normal', 'Suspect', 'Pathological'], colors = ['green', 'blue','red'],␣
 ↪autopct='%1.0f%%')
plt.title('Fetal Health Rating Percentage')
plt.xlabel('Fetal Health Rating')
plt.ylabel('Percent of Cases per Rating')
```

```
plt.show()
```



[6]:
```
# Histogram to visualize data
df_visual_hist = df.hist(figsize = (20,20))
```



[7]:
```
#Heatmap to visualize correlations
sns.set(style='white')
corr = df.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.75, center=0,
            square=True, linewidths=.5, cbar_kws={'shrink': .5});
```

0.0.3 Train/Test Split

[8]:
```
X = df.drop(['fetal_health'], axis = 1)
y = df['fetal_health']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[8]: ((1700, 21), (426, 21), (1700,), (426,))

0.0.4 Scaling Data

[9]:
```
scaler= StandardScaler()
scaler.fit(X_train)
X_train = pd.DataFrame(data = scaler.transform(X_train), columns = X.columns)
X_test = pd.DataFrame(data = scaler.transform(X_test), columns = X.columns)
```

0.0.5 Confusion Matrix

[10]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion Matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

0.0.6 Logistic Regression

[11]:

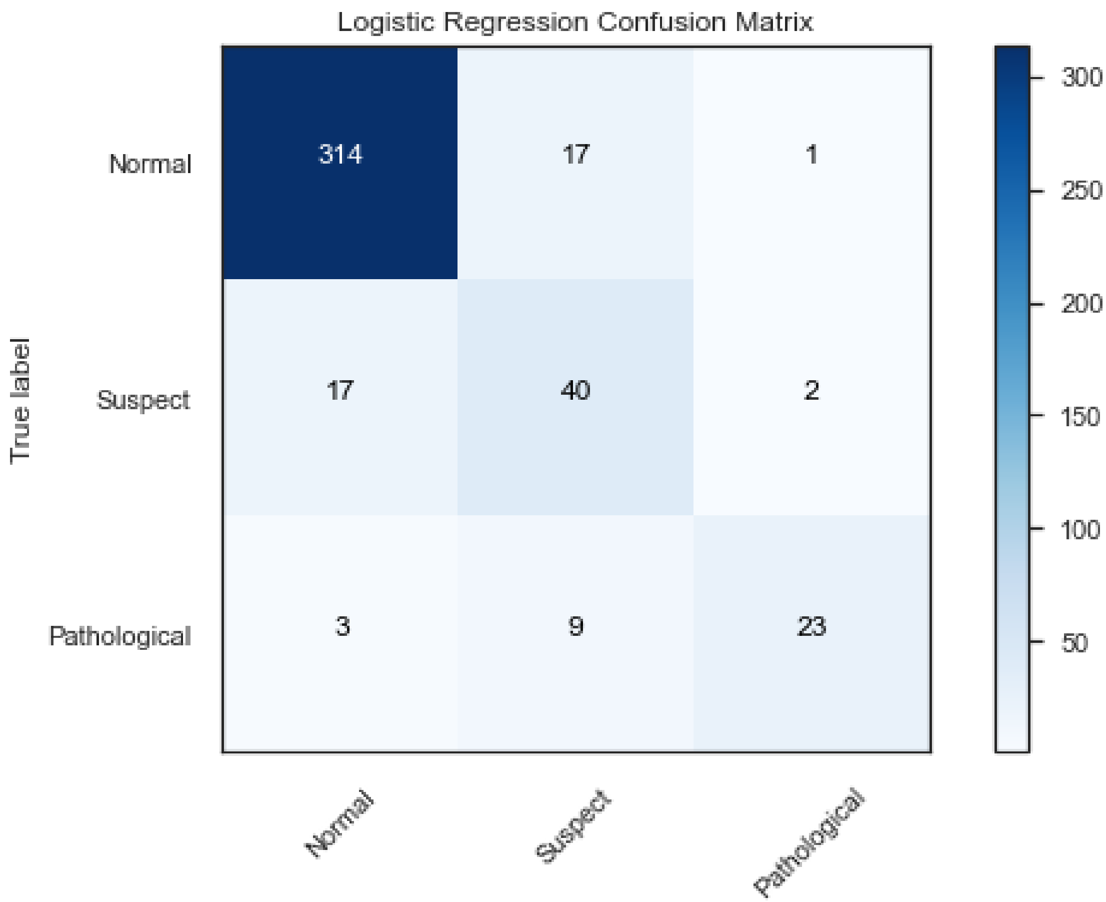
```
logreg = LogisticRegression(multi_class='multinomial', random_state=42)
logreg.fit(X_train, y_train)
logreg_preds = logreg.predict(X_test)

logreg_f1 = metrics.f1_score(y_test, logreg_preds, average='weighted')
logreg_recall = metrics.recall_score(y_test, logreg_preds, average='weighted')
logreg_acc = metrics.accuracy_score(y_test, logreg_preds)

# Checking Accuracy, F1, and Recall scores
print('Test F1:' , logreg_f1)
print('Test Accuracy:' , logreg_acc)
print('Test Recall:' , logreg_recall)
```

```
Test F1: 0.8854706565258401
Test Accuracy: 0.8849765258215962
Test Recall: 0.8849765258215962
```

[12]:

```
classes = ['Normal', 'Suspect', 'Pathological']
cm_lr = metrics.confusion_matrix(y_test, logreg_preds)
plot_confusion_matrix(cm_lr, classes=classes, title = 'Logistic Regression␣
↪Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[314  17   1]
 [ 17  40   2]
 [  3   9  23]]
```



[13]:
```
print(classification_report(y_test, logreg_preds))
```

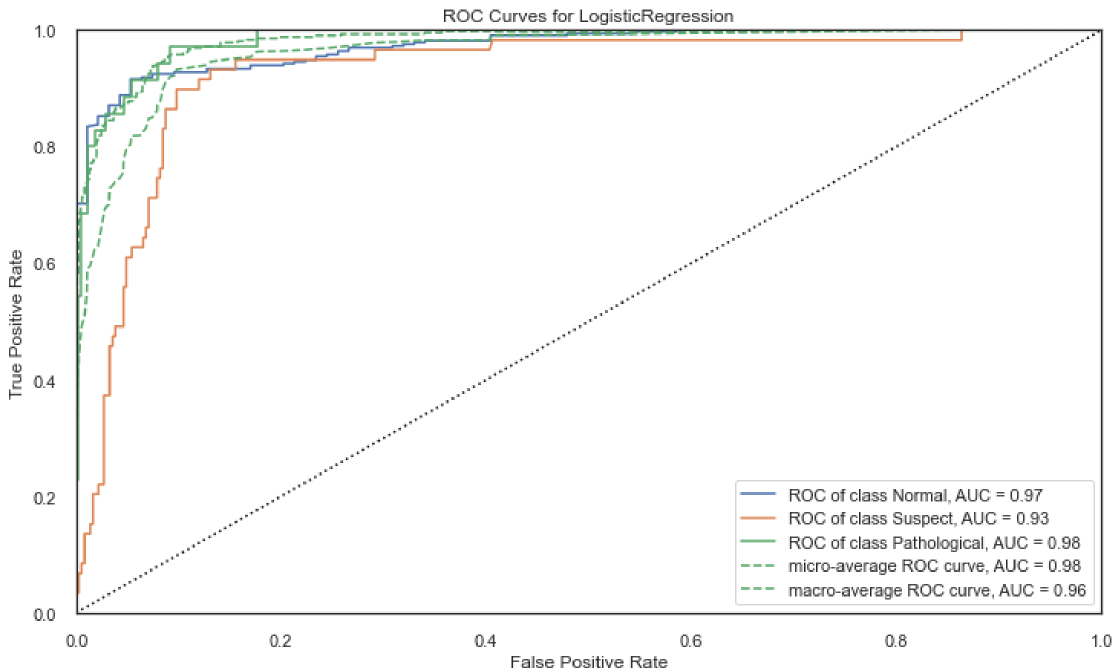
```
              precision    recall  f1-score   support

         1.0       0.94      0.95      0.94       332
         2.0       0.61      0.68      0.64        59
         3.0       0.88      0.66      0.75        35

    accuracy                           0.88       426
   macro avg       0.81      0.76      0.78       426
weighted avg       0.89      0.88      0.89       426
```

[14]:
```
y_test.value_counts()
```

[14]:
```
1.0    332
2.0     59
3.0     35
```

```
Name: fetal_health, dtype: int64
```

[15]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(logreg, classes=classes, ax=ax)
roc.fit(X_train, y_train)        # Fit the training data to the visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[15]:
```
<AxesSubplot:title={'center':'ROC Curves for LogisticRegression'}, xlabel='False
Positive Rate', ylabel='True Positive Rate'>
```

0.0.7 Synthetic Minority Oversampling TEchnique (SMOTE) Logistic Regression

[16]:
```
sm = SMOTE(random_state=42)
X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)
```

[17]:
```
y_train_smote.value_counts()
```

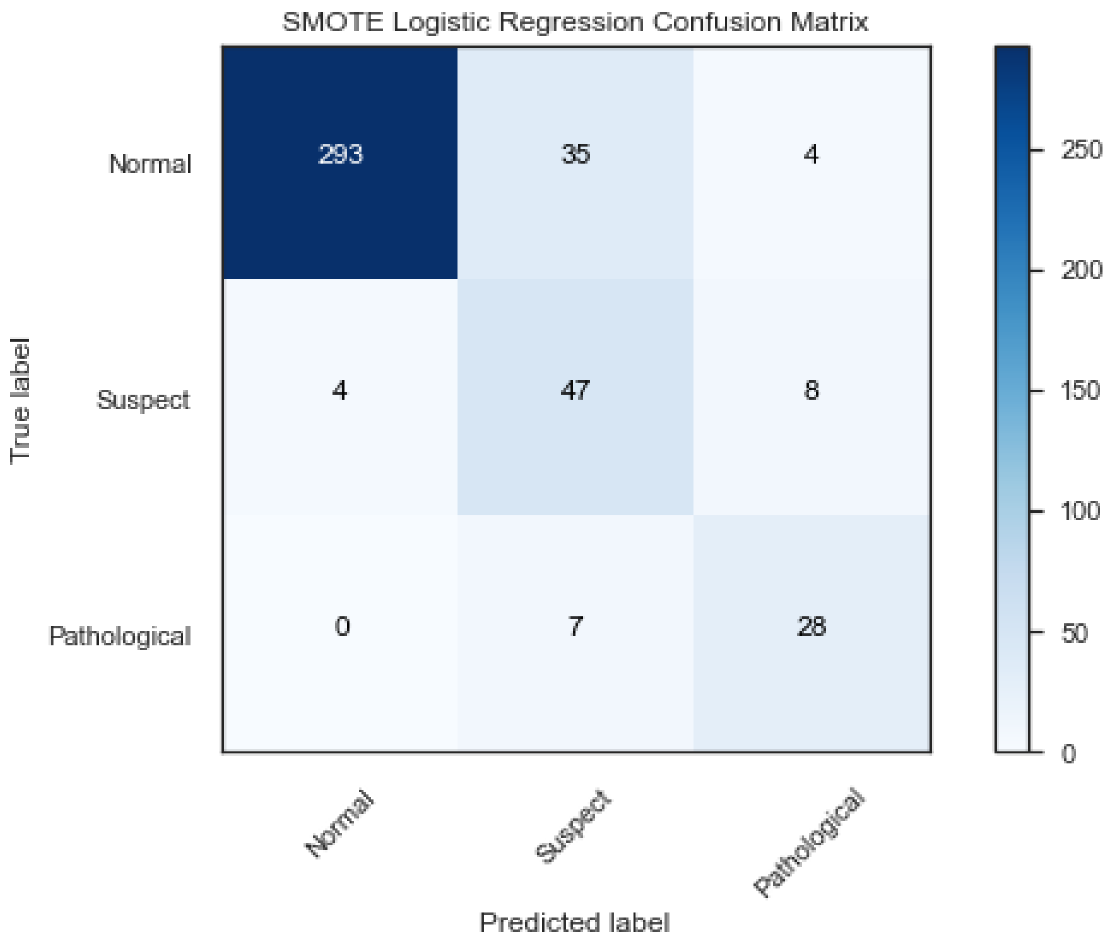
[17]:
```
2.0    1323
1.0    1323
3.0    1323
Name: fetal_health, dtype: int64
```

[18]:
```
smote = LogisticRegression(multi_class='multinomial', random_state = 42).
 ↪fit(X_train_smote, y_train_smote)

smote_preds = smote.predict(X_test)

# Checking Accuracy, F1, and Recall scores
print('Test Accuracy:' , accuracy_score(y_test, smote_preds))
# F1 score
print('Test F1:' , f1_score(y_test, smote_preds, average='weighted'))
# Recall
print('Test Recall:' , recall_score(y_test, smote_preds, average='weighted'))
```

```
Test Accuracy: 0.863849765258216
Test F1: 0.875375499774837
Test Recall: 0.863849765258216
```

[19]:
```
cm_lr_smote=metrics.confusion_matrix(y_test, smote_preds)
plot_confusion_matrix(cm_lr_smote, classes=['Normal', 'Suspect',␣
 ↪'Pathological'],
                      title='SMOTE Logistic Regression Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[293  35   4]
 [  4  47   8]
 [  0   7  28]]
```

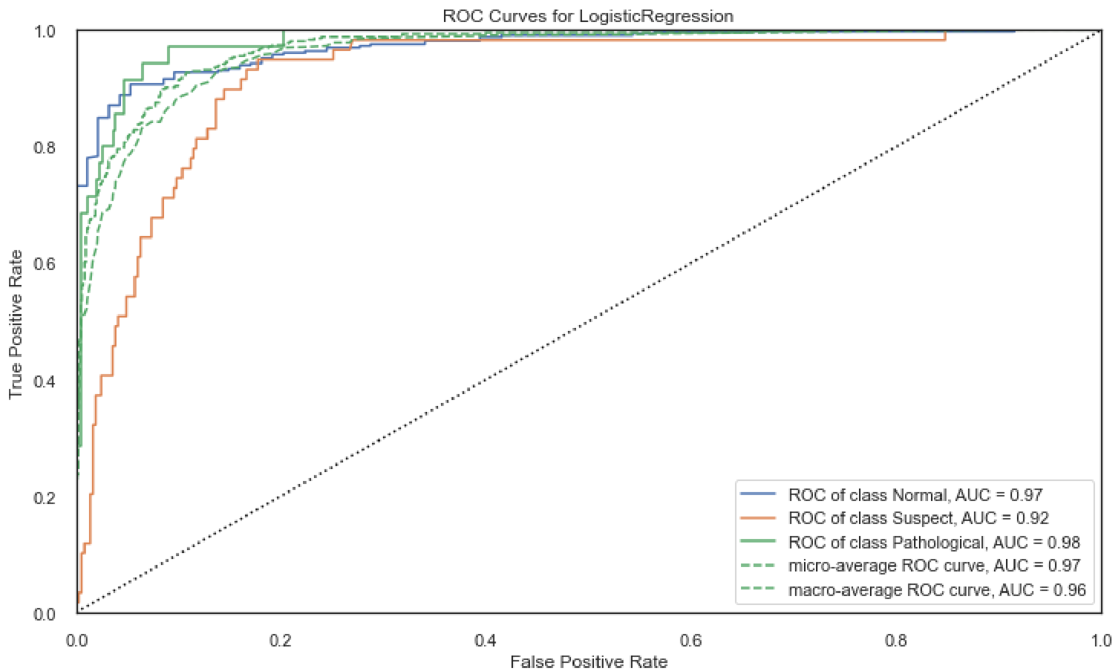


[20]:
```
print(classification_report(y_test, smote_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.88      0.93       332
         2.0       0.53      0.80      0.64        59
         3.0       0.70      0.80      0.75        35

    accuracy                           0.86       426
   macro avg       0.74      0.83      0.77       426
weighted avg       0.90      0.86      0.88       426
```

[21]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(smote, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
 ↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
```

```
roc.show()
```



[21]: <AxesSubplot:title={'center':'ROC Curves for LogisticRegression'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>

0.0.8 Cross Validate SMOTE Logistic Regression

[22]:
```
cv_method = StratifiedKFold(n_splits=3)
```

[23]:
```
# Cross validate SMOTE Logistic Regression model
scores_lr = cross_val_score(logreg, X_train_smote, y_train_smote, cv␣
↪=cv_method, scoring = 'accuracy')

print(f'Scores(Cross Validation) for Logistic Regression model:\n{scores_lr}')
print(f'CrossValMeans: {round(scores_lr.mean(), 3)}')
print(f'CrossValStandard Deviation: {round(scores_lr.std(), 3)}')
```

```
Scores(Cross Validation) for Logistic Regression model:
[0.87452759 0.89569161 0.89417989]
CrossValMeans: 0.888
CrossValStandard Deviation: 0.01
```

0.0.9 GridCV - SMOTE Logistic Regression

[24]:
```
#Creating Dictionary of Parameters to Tune
params_lr = {'tol': [0.0001,0.0002,0.0003],
            'C': [80, 100, 120, 140],
             'penalty':['l1', 'l2']
             }
estimator_cv = LogisticRegression(multi_class='multinomial', random_state = 42)
```

[25]:
```
GridSearchCV_LR = GridSearchCV(estimator=estimator_cv,
                                param_grid=params_lr,
                                cv=cv_method,
                                verbose=1,
                                n_jobs=5
                              )
```

[26]:
```
# Fit model with train data
GridSearchCV_LR.fit(X_train_smote, y_train_smote)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

[26]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=False),
             estimator=LogisticRegression(multi_class='multinomial',
                                          random_state=42),
             n_jobs=5,
             param_grid={'C': [80, 100, 120, 140], 'penalty': ['l1', 'l2'],
                         'tol': [0.0001, 0.0002, 0.0003]},
             verbose=1)
```

[27]:
```
#Identifying Best Parameters
print(GridSearchCV_LR.best_params_)
print(GridSearchCV_LR.best_estimator_)
#Identifying Best Score During Fitting with Cross-Validation
print(GridSearchCV_LR.best_score_)
```

```
{'C': 100, 'penalty': 'l2', 'tol': 0.0001}
LogisticRegression(C=100, multi_class='multinomial', random_state=42)
0.888888888888889
```

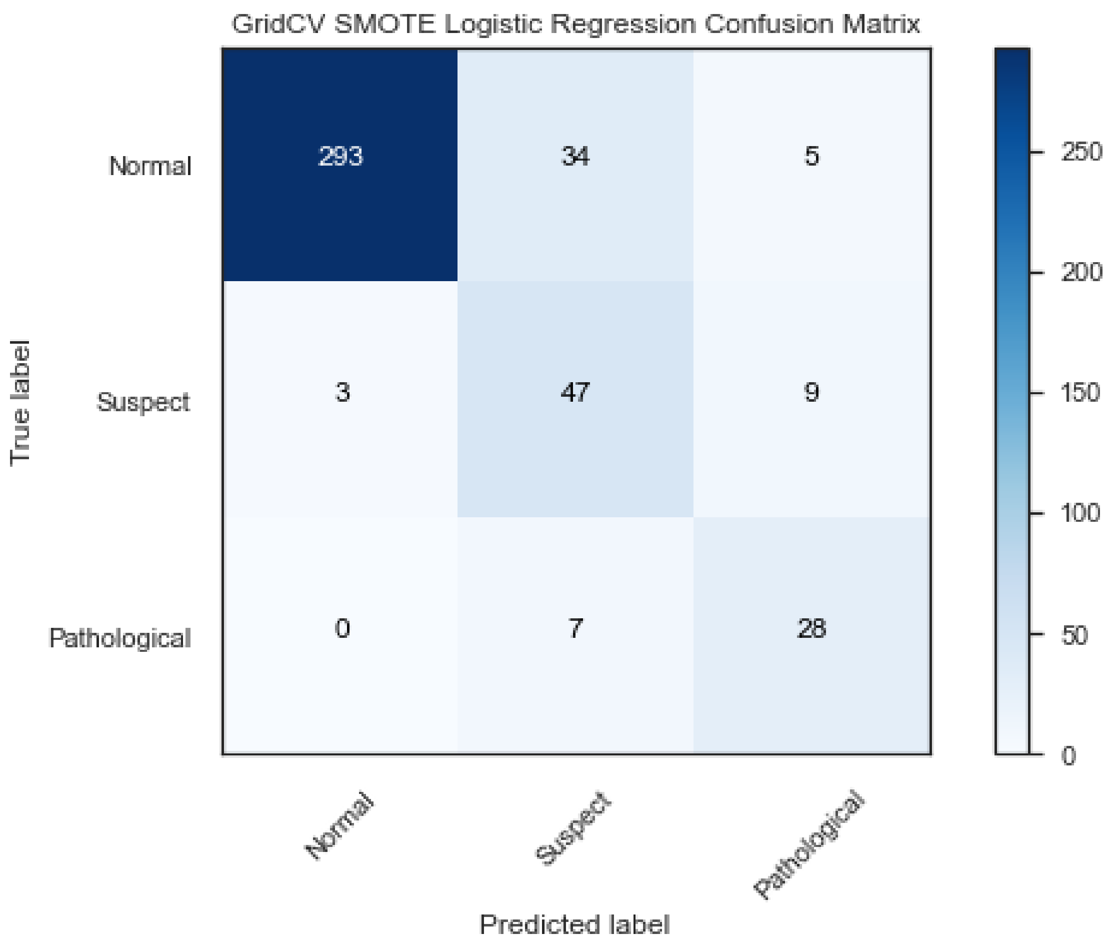
[28]:
```
#Predicting Test Set
GridSearchCV_LR_preds = GridSearchCV_LR.best_estimator_.predict(X_test)

# Checking Accuracy, F1, Recall Scores
print('Test Accuracy:' , accuracy_score(y_test, GridSearchCV_LR_preds))
# F1 score
print('Test F1:' , f1_score(y_test, GridSearchCV_LR_preds, average='weighted'))
# Recall
```

```
print('Test Recall:' , recall_score(y_test, GridSearchCV_LR_preds,␣
 ↪average='weighted'))
```

```
Test Accuracy: 0.863849765258216
Test F1: 0.8755366548238429
Test Recall: 0.863849765258216
```

[29]:
```
cm_lr_smote_cv=metrics.confusion_matrix(y_test,GridSearchCV_LR_preds)
plot_confusion_matrix(cm_lr_smote_cv, classes=['Normal', 'Suspect',␣
 ↪'Pathological'],
                      title='GridCV SMOTE Logistic Regression Confusion Matrix')
```

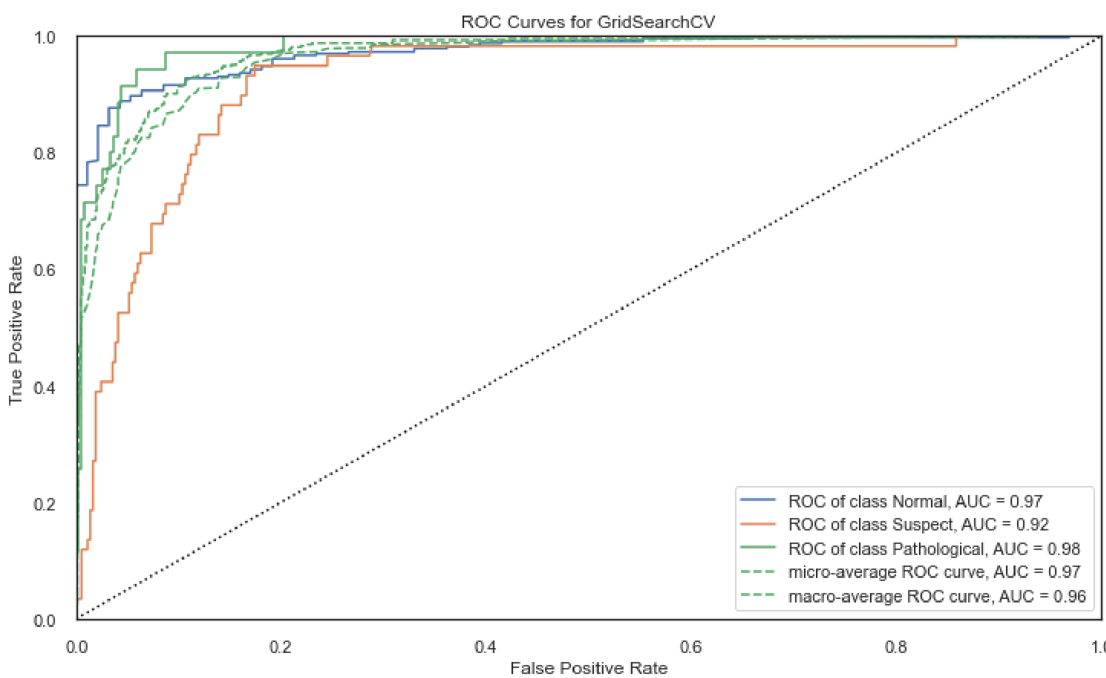
```
Confusion Matrix, without normalization
[[293  34   5]
 [  3  47   9]
 [  0   7  28]]
```



[30]:
```
print(classification_report(y_test, GridSearchCV_LR_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.99      0.88      0.93       332
         2.0       0.53      0.80      0.64        59
         3.0       0.67      0.80      0.73        35

    accuracy                           0.86       426
   macro avg       0.73      0.83      0.77       426
weighted avg       0.90      0.86      0.88       426
```

[31]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(GridSearchCV_LR, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
 ↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[31]:
```
<AxesSubplot:title={'center':'ROC Curves for GridSearchCV'}, xlabel='False
Positive Rate', ylabel='True Positive Rate'>
```

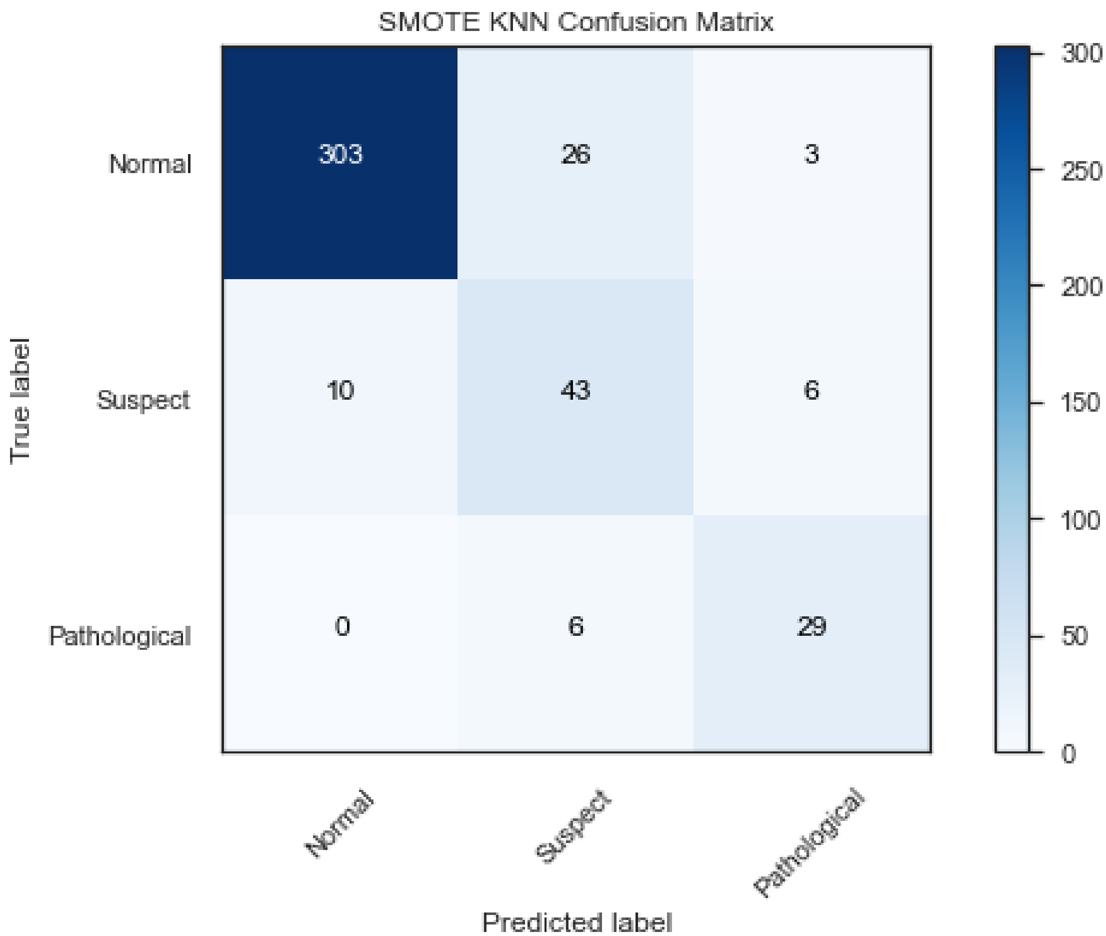
0.0.10 SMOTE KNN

[32]:
```
knn = KNeighborsClassifier(weights='distance')
knn.fit(X_train_smote, y_train_smote)
knn_preds = knn.predict(X_test)

knn_f1 = metrics.f1_score(y_test, knn_preds, average='weighted')
knn_acc = metrics.accuracy_score(y_test, knn_preds)
knn_recall = metrics.recall_score(y_test, knn_preds, average='weighted')

# Checking Accuracy, F1, and Recall scores
print('Test F1:' , knn_f1)
print('Test Accuracy:' , knn_acc)
print('Test Recall:' , knn_recall)
```

```
Test F1: 0.886383737594418
Test Accuracy: 0.8802816901408451
Test Recall: 0.8802816901408451
```

[33]:
```
cm_knn=metrics.confusion_matrix(y_test, knn_preds)
plot_confusion_matrix(cm_knn, classes=['Normal', 'Suspect', 'Pathological'],
                      title='SMOTE KNN Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[303  26   3]
 [ 10  43   6]
 [  0   6  29]]
```

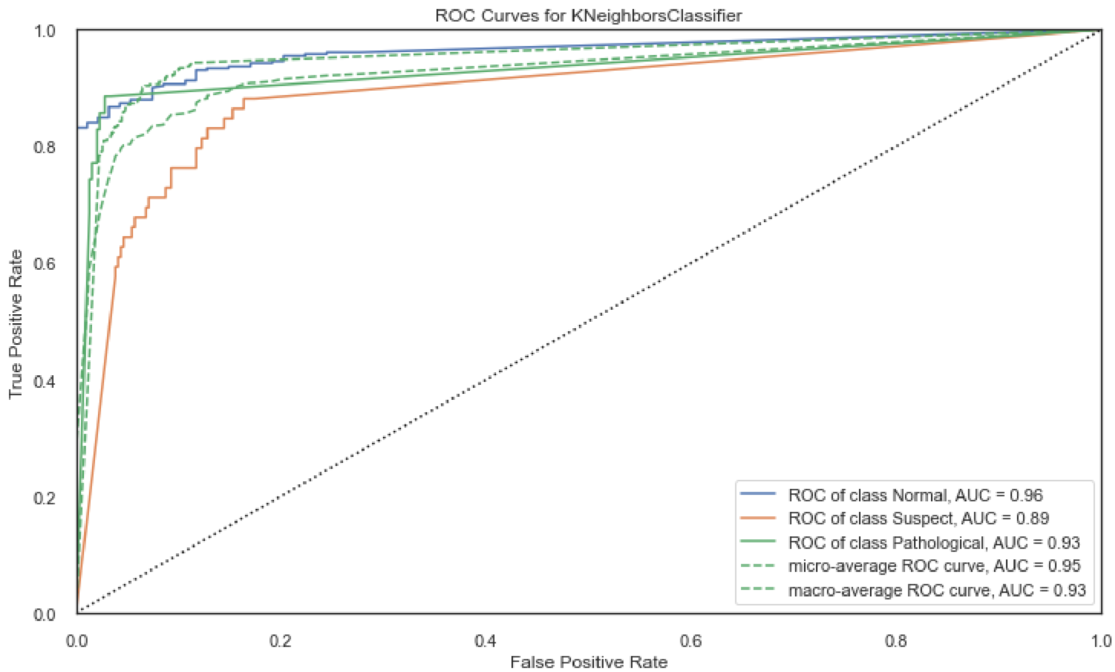


[34]:
```
# Cross validate SMOTE K-Nearest Neighbors model
scores_knn = cross_val_score(knn, X_train_smote, y_train_smote, cv = cv_method,␣
↪n_jobs=-1, scoring="accuracy")

print(f"Scores(Cross validate) for K-Nearest Neighbors model:\n{scores_knn}")
print(f"CrossValMeans: {round(scores_knn.mean(), 3)}")
print(f"CrossValStandard Deviation: {round(scores_knn.std(), 3)}")
```

```
Scores(Cross validate) for K-Nearest Neighbors model:
[0.95313681 0.96674225 0.96900983]
CrossValMeans: 0.963
CrossValStandard Deviation: 0.007
```

[35]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(knn, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
↪visualizer
```

```
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[35]: <AxesSubplot:title={'center':'ROC Curves for KNeighborsClassifier'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>

0.0.11 GridCV - SMOTE KNN

[36]:
```
params_knn = {"n_neighbors": [3,5,11,15],
              "p": [1,2],
              'metric': ['euclidean', 'manhattan']
             }
```

[37]:
```
GridSearchCV_knn = GridSearchCV(estimator=knn,
                                param_grid=params_knn,
                                cv=cv_method,
                                verbose=1,
                                n_jobs=-1,
                                )
```

[38]:
```
# Fit model with train data
GridSearchCV_knn.fit(X_train_smote, y_train_smote)
```

Fitting 3 folds for each of 16 candidates, totalling 48 fits

[38]: GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=False),
             estimator=KNeighborsClassifier(weights='distance'), n_jobs=-1,
             param_grid={'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': [3, 5, 11, 15], 'p': [1, 2]},
             verbose=1)

[39]:
```
#Identifying Best Parameters
print(GridSearchCV_knn.best_params_)
print(GridSearchCV_knn.best_estimator_)
#Identifying Best Score During Fitting with Cross-Validation
print(GridSearchCV_knn.best_score_)
```

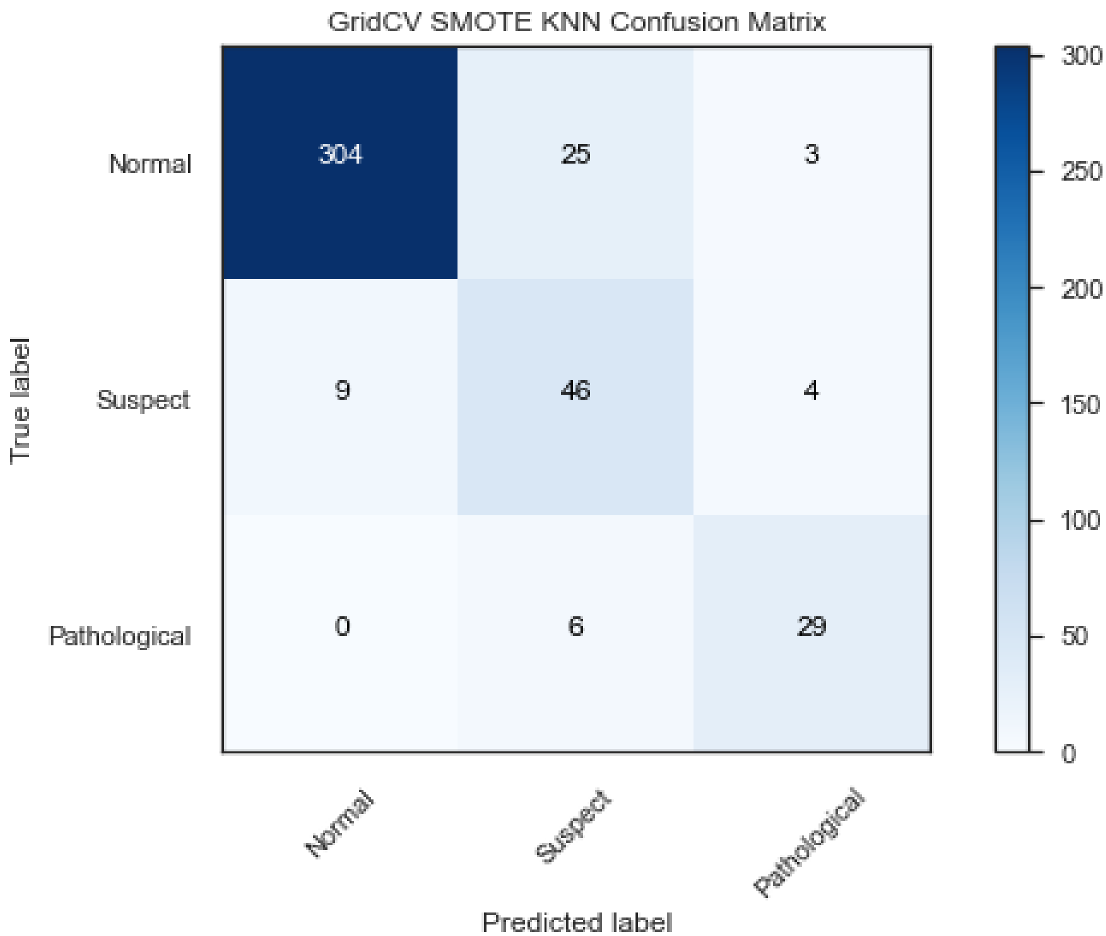
```
{'metric': 'manhattan', 'n_neighbors': 3, 'p': 1}
KNeighborsClassifier(metric='manhattan', n_neighbors=3, p=1, weights='distance')
0.9705215419501134
```

[40]:
```
#Predicting Test Set
GridSearchCV_knn_preds = GridSearchCV_knn.best_estimator_.predict(X_test)

# Checking Accuracy, F1, Recall Scores
print('Test Accuracy:' , accuracy_score(y_test, GridSearchCV_knn_preds))
# F1 score
print('Test F1:' , f1_score(y_test, GridSearchCV_knn_preds, average='weighted'))
# Recall
print('Test Recall:' , recall_score(y_test, GridSearchCV_knn_preds,␣
 ↪average='weighted'))
```

```
Test Accuracy: 0.8896713615023474
Test F1: 0.8954421426322698
Test Recall: 0.8896713615023474
```

[41]:
```
cm_knn_smote_cv=metrics.confusion_matrix(y_test,GridSearchCV_knn_preds)
plot_confusion_matrix(cm_knn_smote_cv, classes=['Normal', 'Suspect',␣
 ↪'Pathological'],
                      title='GridCV SMOTE KNN Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[304  25   3]
 [  9  46   4]
 [  0   6  29]]
```

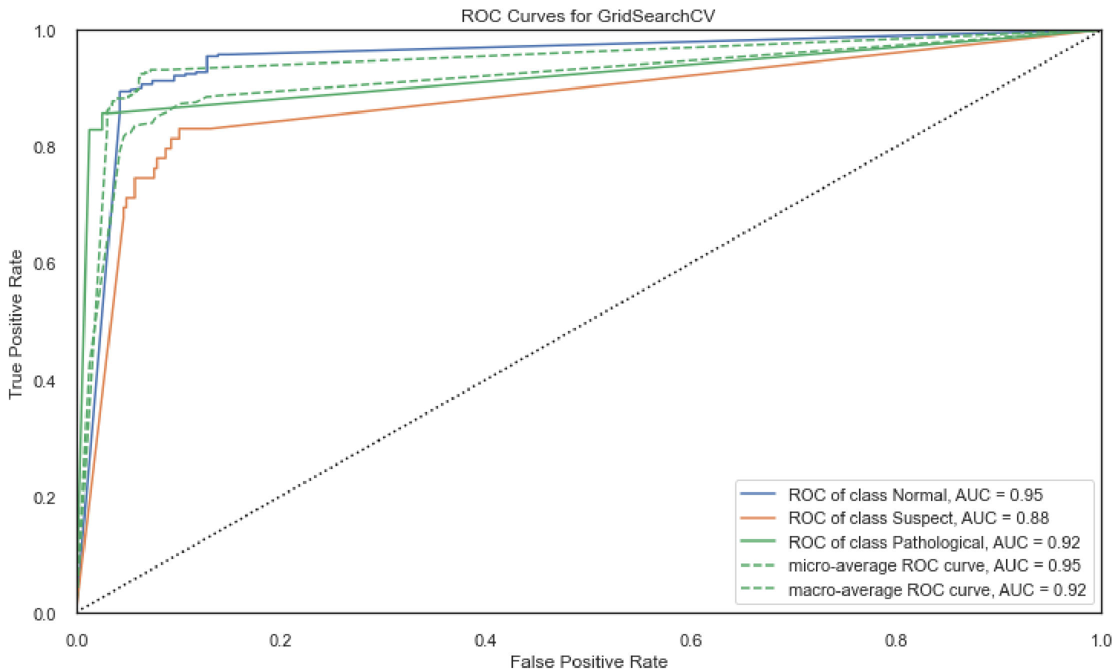


[42]:
```
print(classification_report(y_test, GridSearchCV_knn_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.97      0.92      0.94       332
         2.0       0.60      0.78      0.68        59
         3.0       0.81      0.83      0.82        35

    accuracy                           0.89       426
   macro avg       0.79      0.84      0.81       426
weighted avg       0.91      0.89      0.90       426
```

[43]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(GridSearchCV_knn, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
 ↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
```

```
roc.show()
```



[43]: <AxesSubplot:title={'center':'ROC Curves for GridSearchCV'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>

0.0.12 SMOTE Decision Tree

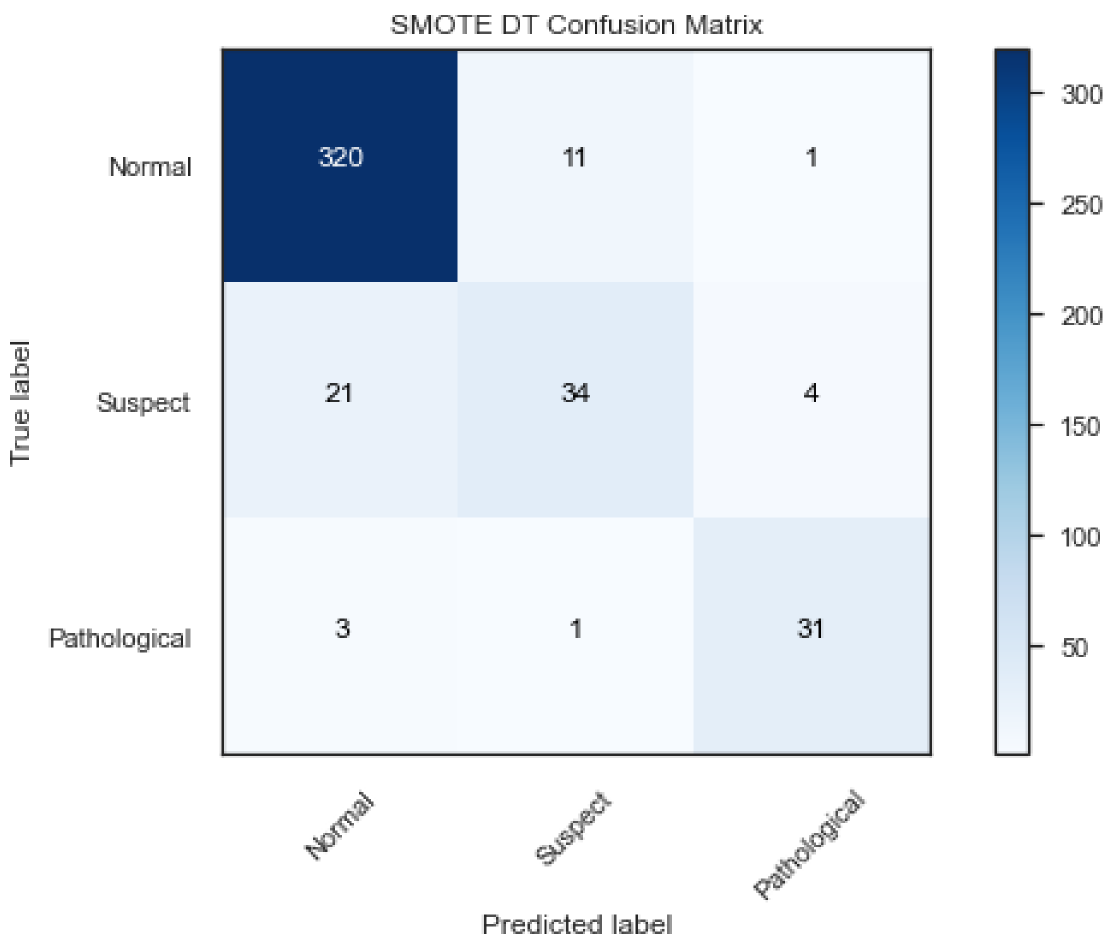
[44]:
```
dt = DecisionTreeClassifier(random_state = 42)
dt.fit(X_train_smote, y_train_smote)
dt_preds = dt.predict(X_test)

dt_f1 = metrics.f1_score(y_test, dt_preds, average='weighted')
dt_acc = metrics.accuracy_score(y_test, dt_preds)
dt_recall = metrics.recall_score(y_test, dt_preds, average='weighted')

# Checking Accuracy, F1, and Recall scores
print('Test F1:' , dt_f1)
print('Test Accuracy:' , dt_acc)
print('Test Recall:' , dt_recall)
```

```
Test F1: 0.8992780063812981
Test Accuracy: 0.903755868544601
Test Recall: 0.903755868544601
```

[45]:

```
cm_dt=metrics.confusion_matrix(y_test,dt_preds)
plot_confusion_matrix(cm_dt, classes=['Normal', 'Suspect', 'Pathological'],
                      title='SMOTE DT Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[320  11   1]
 [ 21  34   4]
 [  3   1  31]]
```



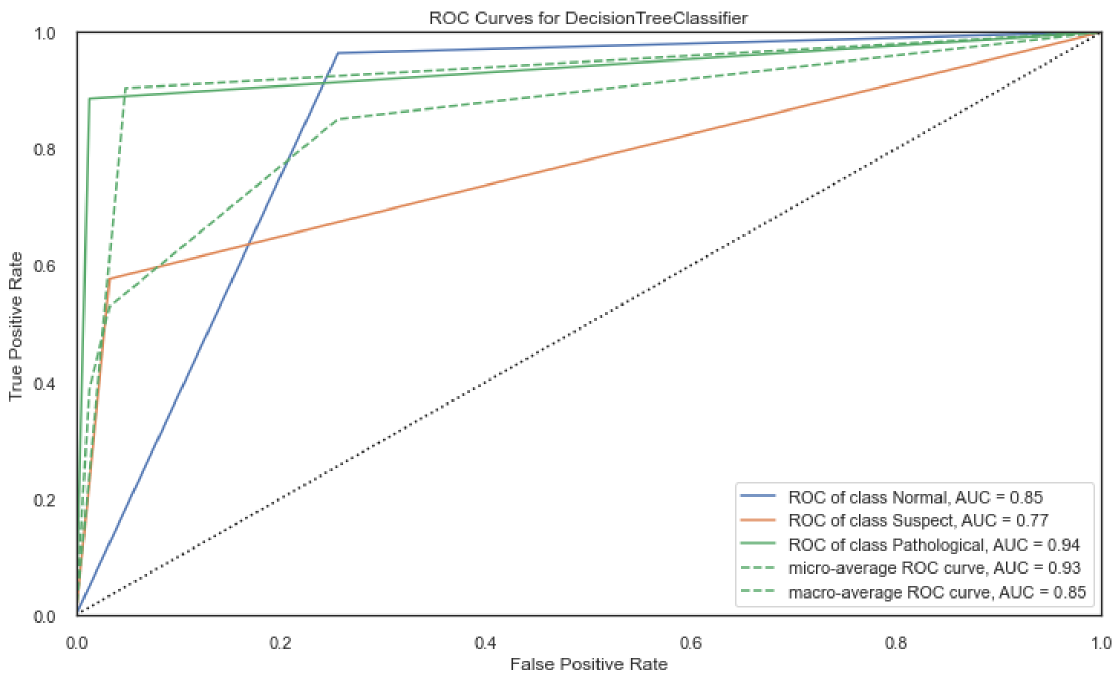
[46]:

```
# Cross validate DT model
scores_dt = cross_val_score(dt, X_train_smote, y_train_smote, cv = cv_method,␣
 ↪n_jobs=-1, scoring="accuracy")

print(f"Scores(Cross validate) for Decision Tree model:\n{scores_dt}")
print(f"CrossValMeans: {round(scores_dt.mean(), 3)}")
print(f"CrossValStandard Deviation: {round(scores_dt.std(), 3)}")
```

```
Scores(Cross validate) for Decision Tree model:
```

```
[0.93953137 0.96598639 0.96523054]
CrossValMeans: 0.957
CrossValStandard Deviation: 0.012
```

[47]:
```python
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(dt, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
 ↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[47]:
```
<AxesSubplot:title={'center':'ROC Curves for DecisionTreeClassifier'},
xlabel='False Positive Rate', ylabel='True Positive Rate'>
```

0.0.13 SMOTE RF

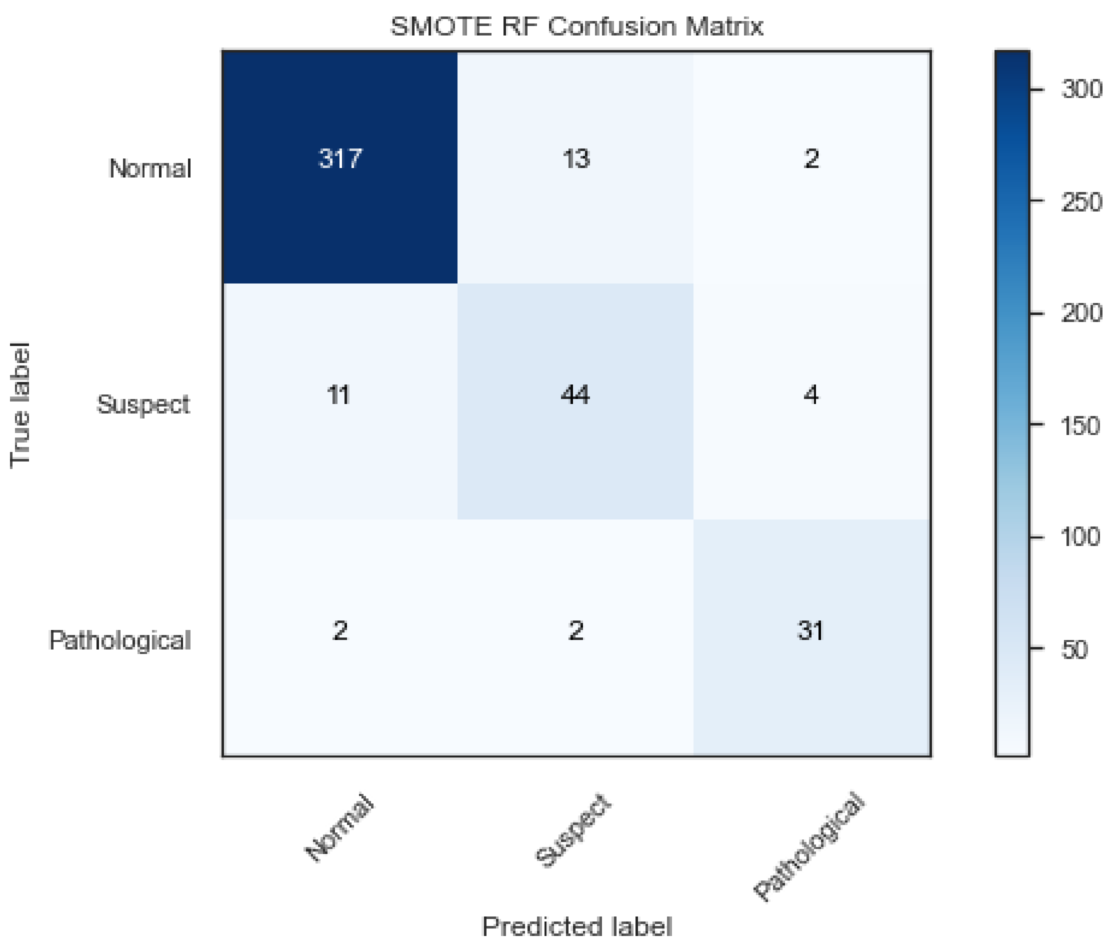
[48]:
```python
rf = RandomForestClassifier(random_state=42, n_jobs=-1, n_estimators = 100)
rf.fit(X_train_smote, y_train_smote)
rf_preds = rf.predict(X_test)

rf_f1 = metrics.f1_score(y_test, rf_preds, average='weighted')
rf_acc = metrics.accuracy_score(y_test, rf_preds)
rf_recall = metrics.recall_score(y_test, rf_preds, average='weighted')

# Checking Accuracy, F1, and Recall scores
```

```
print('Test F1:' , rf_f1)
print('Test Accuracy:' , rf_acc)
print('Test Recall:' , rf_recall)
```

```
Test F1: 0.9204145371276556
Test Accuracy: 0.92018779342723
Test Recall: 0.92018779342723
```

[49]:
```
cm_rf=metrics.confusion_matrix(y_test,rf_preds)
plot_confusion_matrix(cm_rf, classes=['Normal', 'Suspect', 'Pathological'],
                      title='SMOTE RF Confusion Matrix')
```

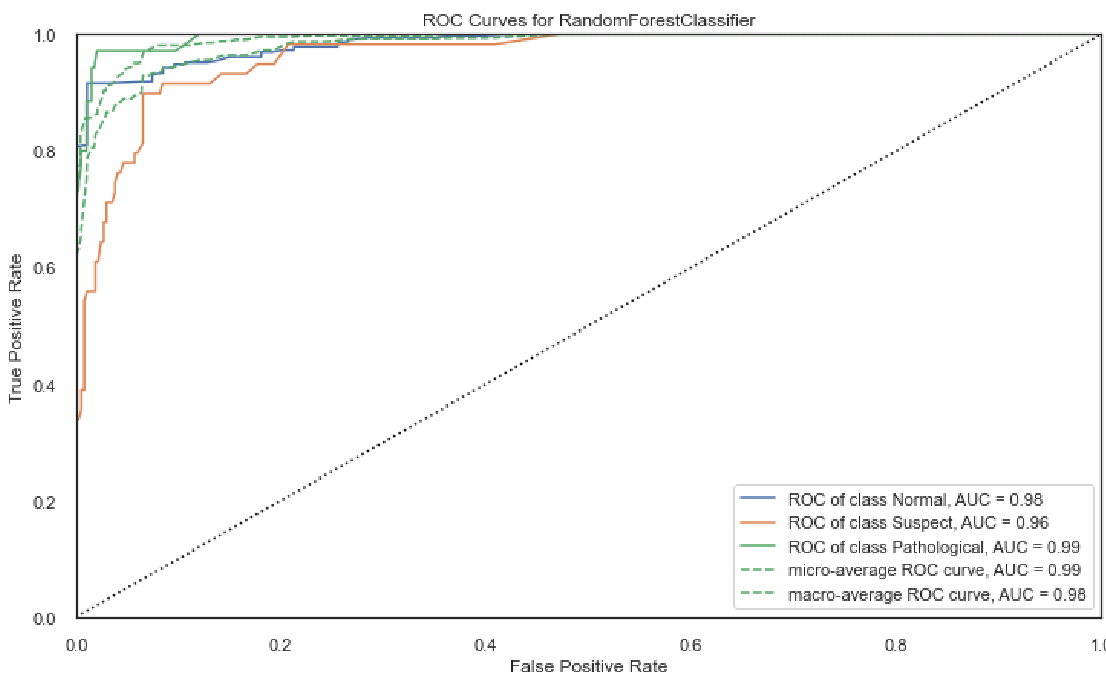
```
Confusion Matrix, without normalization
[[317  13   2]
 [ 11  44   4]
 [  2   2  31]]
```



[50]:
```
# Cross validate SMOTE RF model
scores_rf = cross_val_score(rf, X_train_smote, y_train_smote, cv = cv_method,␣
↪n_jobs=-1, scoring="accuracy")

print(f"Scores(Cross validate) for Random Forest model:\n{scores_rf}")
print(f"CrossValMeans: {round(scores_rf.mean(), 3)}")
print(f"CrossValStandard Deviation: {round(scores_rf.std(), 3)}")
```

```
Scores(Cross validate) for Random Forest model:
[0.96069539 0.9856387  0.9856387 ]
CrossValMeans: 0.977
CrossValStandard Deviation: 0.012
```

[51]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(rf, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[51]:
```
<AxesSubplot:title={'center':'ROC Curves for RandomForestClassifier'},
xlabel='False Positive Rate', ylabel='True Positive Rate'>
```

0.0.14 SMOTE GridCV - RF

[52]:
```
rf_params = {
    'n_estimators': [75, 100, 125],
    'max_features': [.25, .35, 'auto'],
    'max_depth' : [7, 9, 11],
    'criterion' : ['entropy']
}
```

[53]:
```
GridSearchCV_rf = GridSearchCV(estimator=rf,
                               param_grid=rf_params,
                               cv=cv_method,
                               scoring='accuracy',
                               verbose=1,
                               n_jobs=-1
                              )
```

[54]:
```
# Fit model with train data
GridSearchCV_rf.fit(X_train_smote, y_train_smote)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
```

[54]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=False),
             estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
             n_jobs=-1,
             param_grid={'criterion': ['entropy'], 'max_depth': [7, 9, 11],
                         'max_features': [0.25, 0.35, 'auto'],
                         'n_estimators': [75, 100, 125]},
             scoring='accuracy', verbose=1)
```

[55]:
```
#Identifying Best Parameters
print(GridSearchCV_rf.best_params_)
print(GridSearchCV_rf.best_estimator_)
#Identifying Best Score During Fitting with Cross-Validation
print(GridSearchCV_rf.best_score_)
```

```
{'criterion': 'entropy', 'max_depth': 11, 'max_features': 0.25, 'n_estimators':
100}
RandomForestClassifier(criterion='entropy', max_depth=11, max_features=0.25,
                       n_jobs=-1, random_state=42)
0.9768203577727387
```

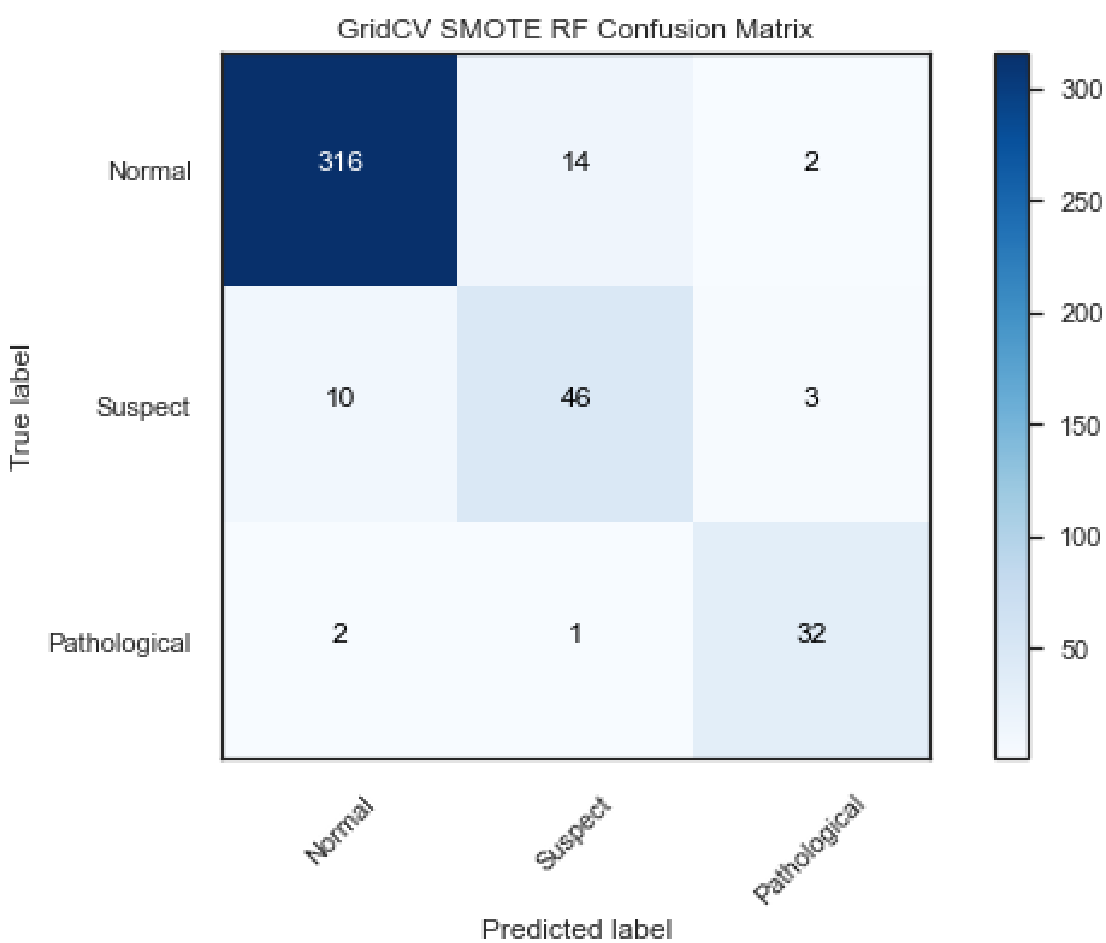
[56]:
```
#Predicting Test Set
GridSearchCV_rf_preds = GridSearchCV_rf.best_estimator_.predict(X_test)

# Checking Accuracy, F1, Recall Scores
print('Test Accuracy:' , accuracy_score(y_test, GridSearchCV_rf_preds))
# F1 score
```

```
print('Test F1:' , f1_score(y_test, GridSearchCV_rf_preds, average='weighted'))
# Recall
print('Test Recall:' , recall_score(y_test, GridSearchCV_rf_preds,␣
↪average='weighted'))
```

```
Test Accuracy: 0.9248826291079812
Test F1: 0.9254920092948263
Test Recall: 0.9248826291079812
```

[57]:
```
cm_rf_smote_cv=metrics.confusion_matrix(y_test,GridSearchCV_rf_preds)
plot_confusion_matrix(cm_rf_smote_cv, classes=['Normal', 'Suspect',␣
↪'Pathological'],
                      title='GridCV SMOTE RF Confusion Matrix')
```

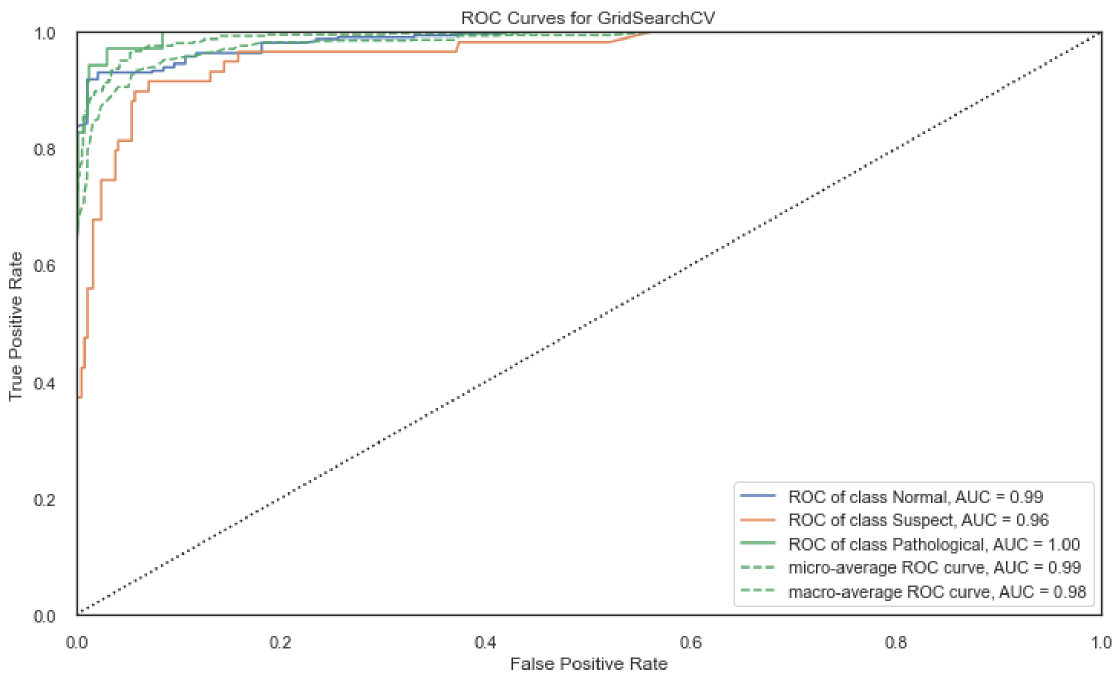
```
Confusion Matrix, without normalization
[[316  14   2]
 [ 10  46   3]
 [  2   1  32]]
```



[58]:
```
print(classification_report(y_test, GridSearchCV_rf_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.96      0.95      0.96       332
         2.0       0.75      0.78      0.77        59
         3.0       0.86      0.91      0.89        35

    accuracy                           0.92       426
   macro avg       0.86      0.88      0.87       426
weighted avg       0.93      0.92      0.93       426
```

[59]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(GridSearchCV_rf, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
 ↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[59]:
```
<AxesSubplot:title={'center':'ROC Curves for GridSearchCV'}, xlabel='False
Positive Rate', ylabel='True Positive Rate'>
```

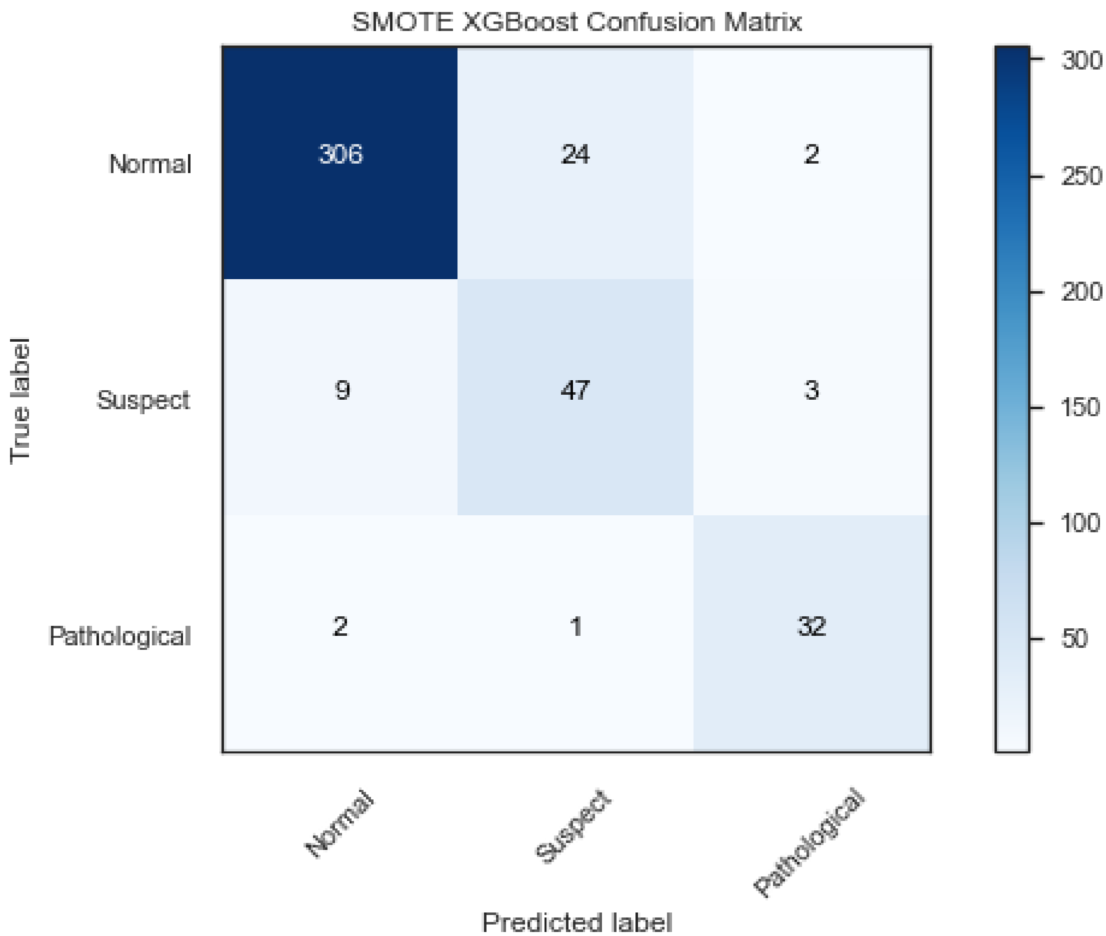
0.0.15 SMOTE XGBoost

[60]:
```
xgb = xgb.XGBClassifier(random_state=42,
                        n_jobs=-1,
                        n_estimators=100,
                        learning_rate=0.01)
xgb.fit(X_train_smote, y_train_smote)
xgb_preds = xgb.predict(X_test)

xgb_f1 = metrics.f1_score(y_test, xgb_preds, average='weighted')
xgb_acc = metrics.accuracy_score(y_test, xgb_preds)
xgb_recall = metrics.recall_score(y_test, xgb_preds, average='weighted')

# Checking Accuracy, F1, and Recall scores
print('Test F1:' , xgb_f1)
print('Test Accuracy:' , xgb_acc)
print('Test Recall:' , xgb_recall)
```

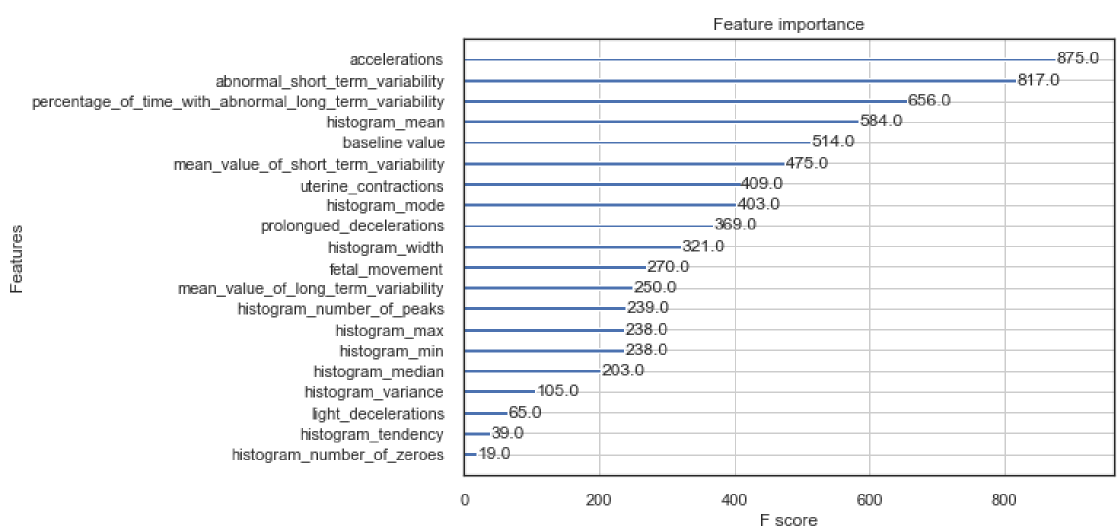
```
[02:31:21] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.0/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'multi:softprob' was changed from
'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
Test F1: 0.90732255417846
Test Accuracy: 0.903755868544601
Test Recall: 0.903755868544601
```

[61]:
```
cm_xgb_smote_cv=metrics.confusion_matrix(y_test, xgb_preds)
plot_confusion_matrix(cm_xgb_smote_cv, classes=['Normal', 'Suspect',␣
↪'Pathological'],
                      title='SMOTE XGBoost Confusion Matrix')
```

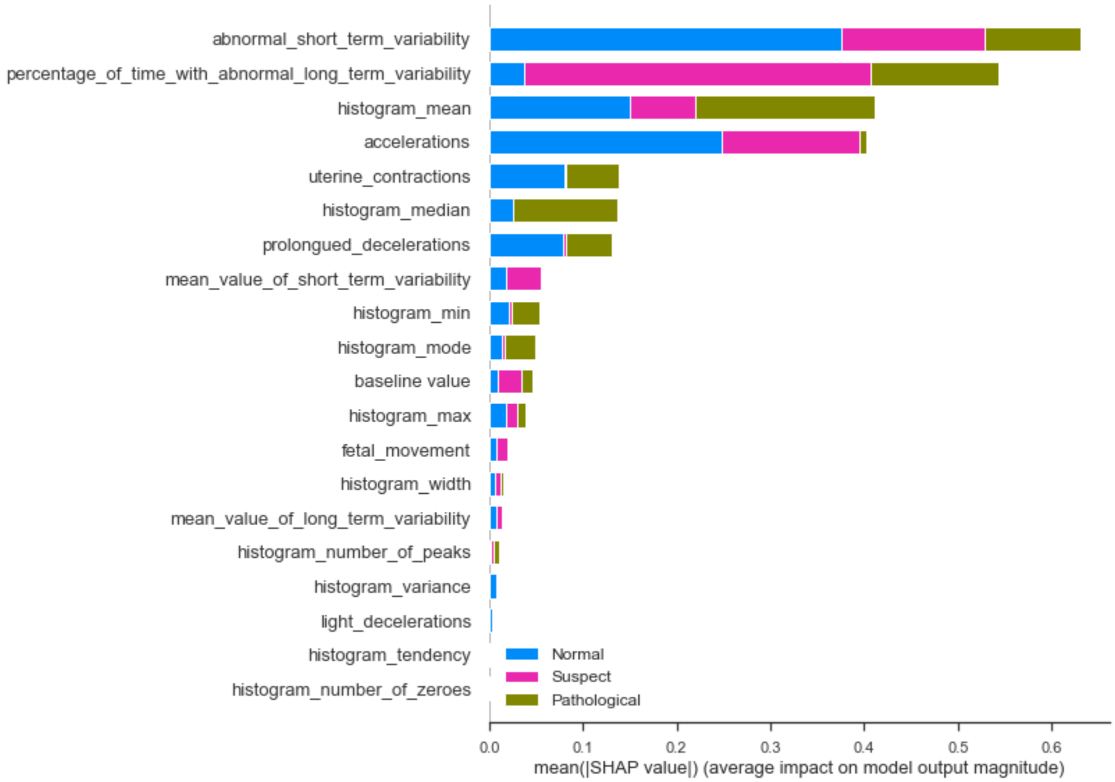
```
Confusion Matrix, without normalization
[[306  24   2]
 [  9  47   3]
 [  2   1  32]]
```



[62]:
```
plot_importance(xgb)
plt.show()
```



[63]:
```
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)

classes=['Normal', 'Suspect', 'Pathological']

shap.summary_plot(shap_values, X_test, plot_type="bar", class_names=classes)
```



0.0.16 SMOTE ADABoost

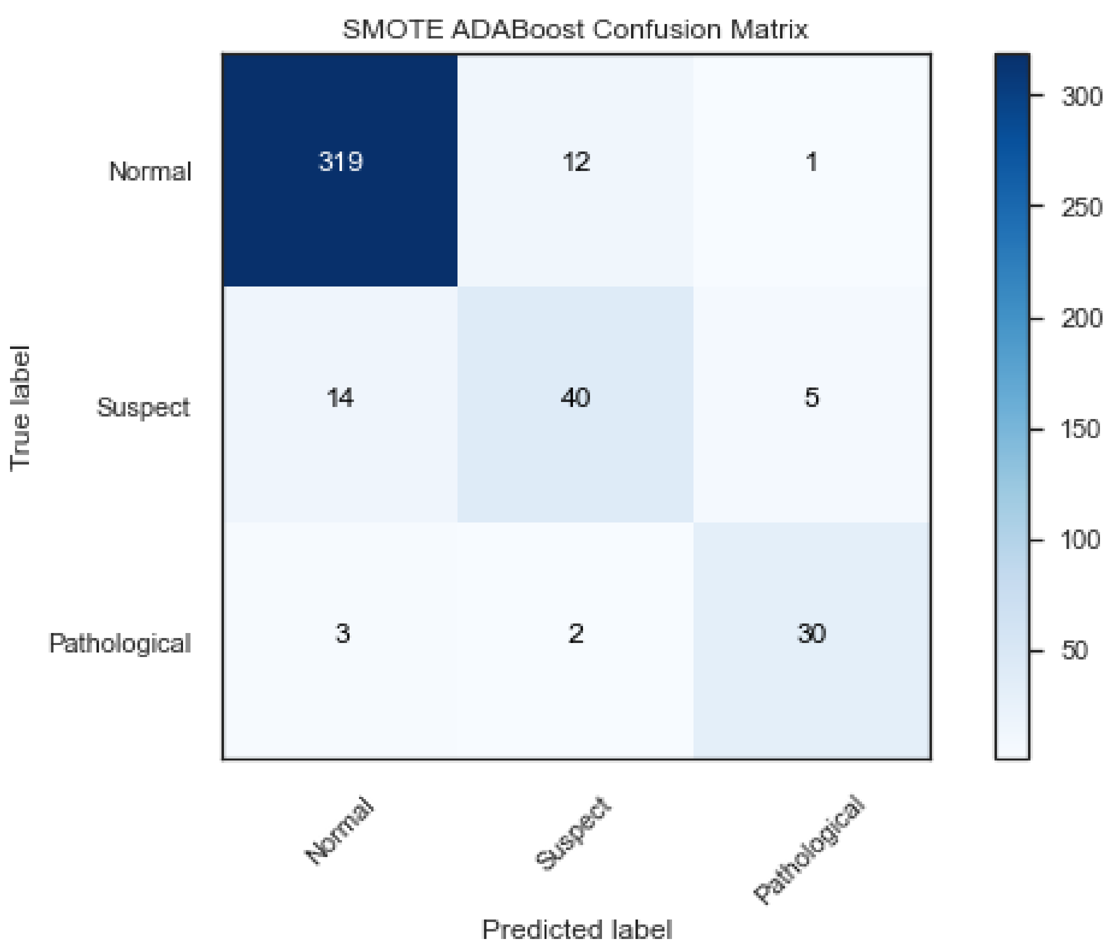
[64]:
```
ada = AdaBoostClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=200,
    algorithm='SAMME.R', learning_rate=0.01, random_state=42)
ada.fit(X_train_smote, y_train_smote)
ada_preds = ada.predict(X_test)

ada_f1 = metrics.f1_score(y_test, ada_preds, average='weighted')
ada_acc = metrics.accuracy_score(y_test, ada_preds)
ada_recall = metrics.recall_score(y_test, ada_preds, average='weighted')
```

```
# Checking Accuracy, F1, and Recall scores
print('Test F1:' , ada_f1)
print('Test Accuracy:' , ada_acc)
print('Test Recall:' , ada_recall)
```

```
Test F1: 0.9118244052874147
Test Accuracy: 0.9131455399061033
Test Recall: 0.9131455399061033
```

[65]:
```
cm_ada_smote_cv=metrics.confusion_matrix(y_test, ada_preds)
plot_confusion_matrix(cm_ada_smote_cv, classes=['Normal', 'Suspect',␣
↪'Pathological'],
                      title='SMOTE ADABoost Confusion Matrix')
```

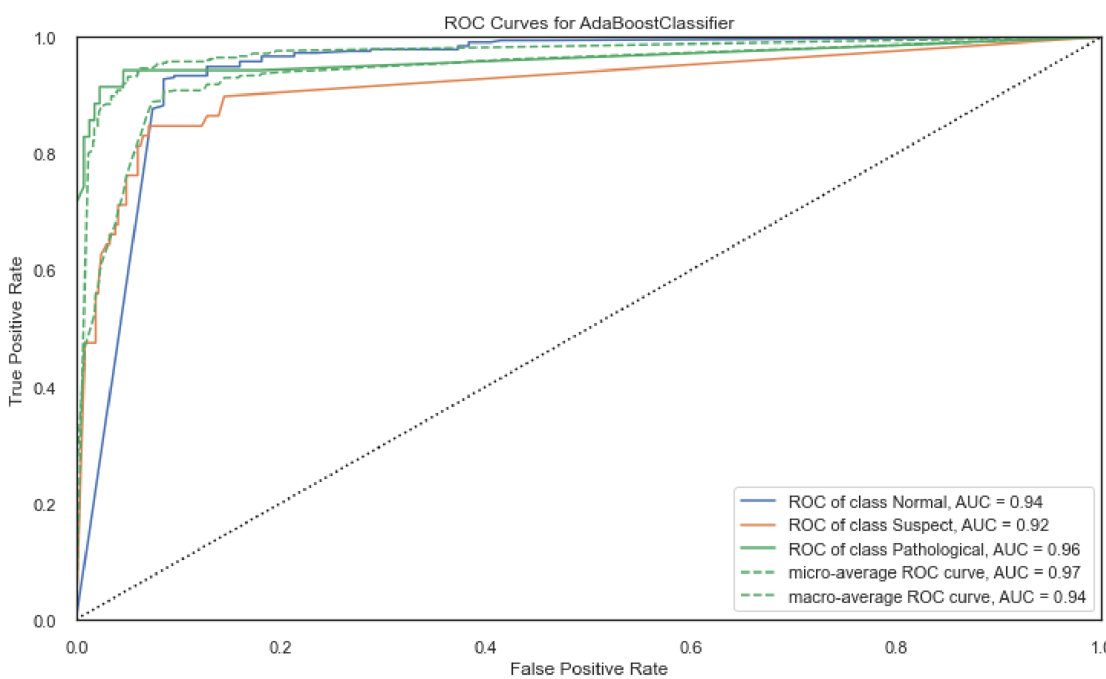
```
Confusion Matrix, without normalization
[[319  12   1]
 [ 14  40   5]
 [  3   2  30]]
```



[66]:
```
print(classification_report(y_test, ada_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.95      0.96      0.96       332
         2.0       0.74      0.68      0.71        59
         3.0       0.83      0.86      0.85        35

    accuracy                           0.91       426
   macro avg       0.84      0.83      0.84       426
weighted avg       0.91      0.91      0.91       426
```

[67]:
```
fig, ax = plt.subplots(figsize=(12, 7))
roc = ROCAUC(ada, classes=classes, ax=ax)
roc.fit(X_train_smote, y_train_smote)        # Fit the training data to the␣
 ↪visualizer
roc.score(X_test, y_test)        # Evaluate the model on the test data
roc.show()
```



[67]:
```
<AxesSubplot:title={'center':'ROC Curves for AdaBoostClassifier'}, xlabel='False
Positive Rate', ylabel='True Positive Rate'>
```

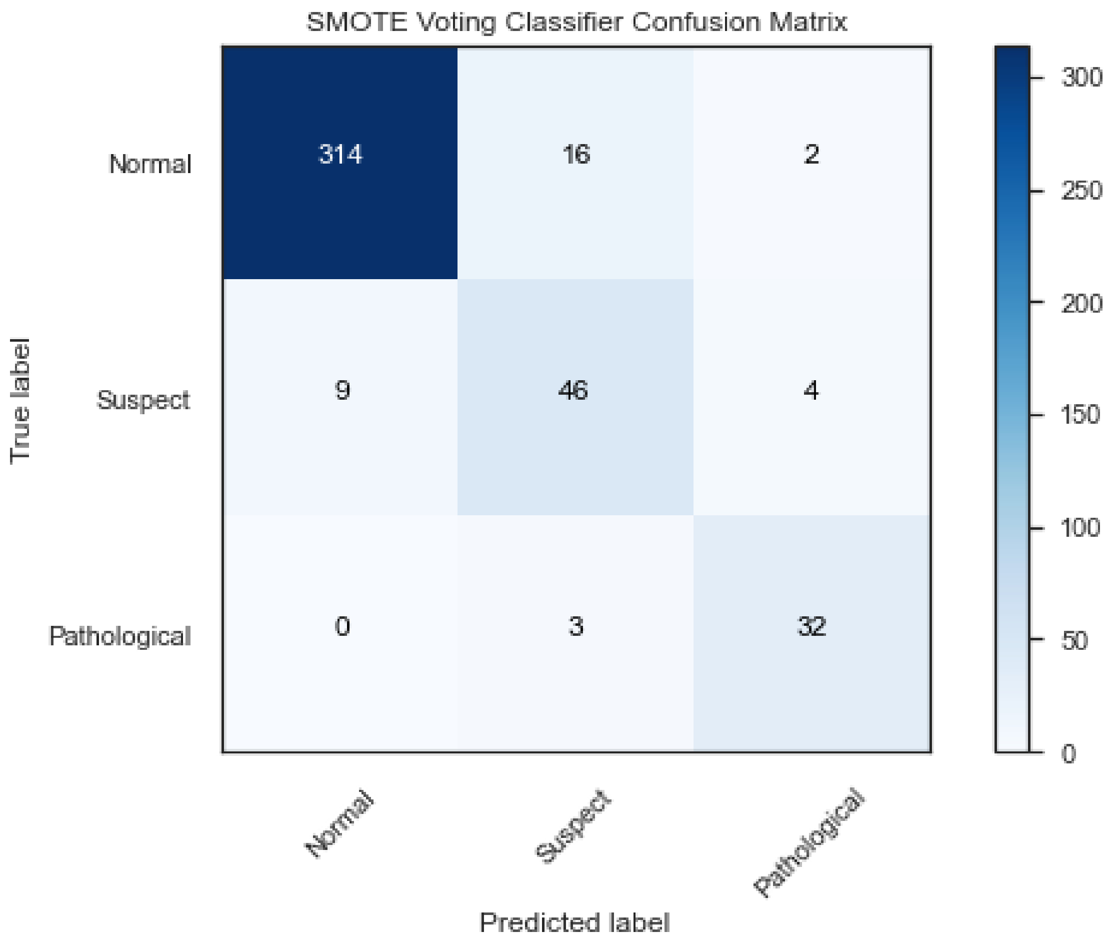
0.0.17 SMOTE VotingClassifier

[68]:
```
vc = VotingClassifier(estimators=[('Base_LR', logreg), ('SMOTE_LR', smote),
                                  ('GridSearchCV_SMOTE_LR', GridSearchCV_LR),
                                  ('SMOTE_KNN', knn),␣
 ↪('GridSearchCV_SMOTE_KNN', GridSearchCV_knn),
                                  ('SMOTE_DT', dt), ('SMOTE_RF', rf),␣
 ↪('GridSearchCV_SMOTE_RF', GridSearchCV_rf),
                                  ('SMOTE_XGBoost', xgb), ('SMOTE_ADABoost',␣
 ↪ada)],
                      voting='soft',
                      n_jobs=5)
vc.fit(X_train_smote, y_train_smote)
vc_preds = vc.predict(X_test)

vc_f1 = metrics.f1_score(y_test, vc_preds, average='weighted')
vc_acc = metrics.accuracy_score(y_test, vc_preds)
vc_recall = metrics.recall_score(y_test, vc_preds, average='weighted')

# Checking Accuracy, F1, and Recall scores
print('Test F1:' , vc_f1)
print('Test Accuracy:' , vc_acc)
print('Test Recall:' , vc_recall)
```

```
Test F1: 0.9220038236432635
Test Accuracy: 0.92018779342723
Test Recall: 0.92018779342723
```

[69]:
```
cm_vc_smote_cv=metrics.confusion_matrix(y_test, vc_preds)
plot_confusion_matrix(cm_vc_smote_cv, classes=['Normal', 'Suspect',␣
 ↪'Pathological'],
                      title='SMOTE Voting Classifier Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[314  16   2]
 [  9  46   4]
 [  0   3  32]]
```



[70]:
```
print(classification_report(y_test, vc_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.97      0.95      0.96       332
         2.0       0.71      0.78      0.74        59
         3.0       0.84      0.91      0.88        35

    accuracy                           0.92       426
   macro avg       0.84      0.88      0.86       426
weighted avg       0.92      0.92      0.92       426
```

0.0.18 SMOTE GridCV - VotingClassifier

[71]:
```
vc_params = {'weights': [[1,2,2,2,2,1,1,1,1,1], [1,3,3,3,3,1,1,1,1,1],
                         [0.5,1,1,1,1,0.5,0.5,0.5,0.5,0.5], [0.75,1,1,1,1,0.
↪75,0.75,0.75,0.75,0.75]],
             'voting':['soft']}
```

[72]:
```
GridSearchCV_vc = GridSearchCV(estimator=vc,
                               param_grid=vc_params,
                               cv=cv_method,
                               scoring='accuracy',
                               verbose=1,
                               n_jobs=5)
```

[73]:
```
GridSearchCV_vc.fit(X_train_smote, y_train_smote)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
```

[73]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None, shuffle=False),
             estimator=VotingClassifier(estimators=[('Base_LR',
LogisticRegression(multi_class='multinomial',
random_state=42)),
                                                     ('SMOTE_LR',
LogisticRegression(multi_class='multinomial',
random_state=42)),
                                                     ('GridSearchCV_SMOTE_LR',
GridSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None,…
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(random_state=42),
learning_rate=0.01,
n_estimators=200,
random_state=42))],
                                        n_jobs=5, voting='soft'),
             n_jobs=5,
             param_grid={'voting': ['soft'],
                         'weights': [[1, 2, 2, 2, 2, 1, 1, 1, 1, 1],
                                     [1, 3, 3, 3, 3, 1, 1, 1, 1, 1],
                                     [0.5, 1, 1, 1, 1, 0.5, 0.5, 0.5, 0.5, 0.5],
                                     [0.75, 1, 1, 1, 1, 0.75, 0.75, 0.75, 0.75,
                                      0.75]]},
             scoring='accuracy', verbose=1)
```

[74]:
```
#Identifying Best Parameters
print(GridSearchCV_vc.best_params_)
print(GridSearchCV_vc.best_estimator_)
#Identifying Best Score During Fitting with Cross-Validation
print(GridSearchCV_vc.best_score_)
```

```
{'voting': 'soft', 'weights': [0.75, 1, 1, 1, 1, 0.75, 0.75, 0.75, 0.75, 0.75]}
```

```
VotingClassifier(estimators=[('Base_LR',
                              LogisticRegression(multi_class='multinomial',
                                                 random_state=42)),
                             ('SMOTE_LR',
                              LogisticRegression(multi_class='multinomial',
                                                 random_state=42)),
                             ('GridSearchCV_SMOTE_LR',
                              GridSearchCV(cv=StratifiedKFold(n_splits=3,
random_state=None, shuffle=False),
estimator=LogisticRegression(multi_class='multinomial',
random_state=42)…
                                            predictor='auto', random_state=42,
                                            reg_alpha=0, reg_lambda=1,
                                            scale_pos_weight=None, subsample=1,
                                            tree_method='exact',
                                            validate_parameters=1,
                                            verbosity=None)),
                             ('SMOTE_ADABoost',
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(random_state=42),
                                                 learning_rate=0.01,
                                                 n_estimators=200,
                                                 random_state=42))],
                 n_jobs=5, voting='soft',
                 weights=[0.75, 1, 1, 1, 1, 0.75, 0.75, 0.75, 0.75, 0.75])
0.9753086419753085
```

[75]:

```python
#Predicting Test Set
GridSearchCV_vc_preds = GridSearchCV_vc.best_estimator_.predict(X_test)

# Checking Accuracy, F1, Recall Scores
print('Test Accuracy:' , accuracy_score(y_test, GridSearchCV_vc_preds))
# F1 score
print('Test F1:' , f1_score(y_test, GridSearchCV_vc_preds, average='weighted'))
# Recall
print('Test Recall:' , recall_score(y_test, GridSearchCV_vc_preds,␣
 ↪average='weighted'))
```

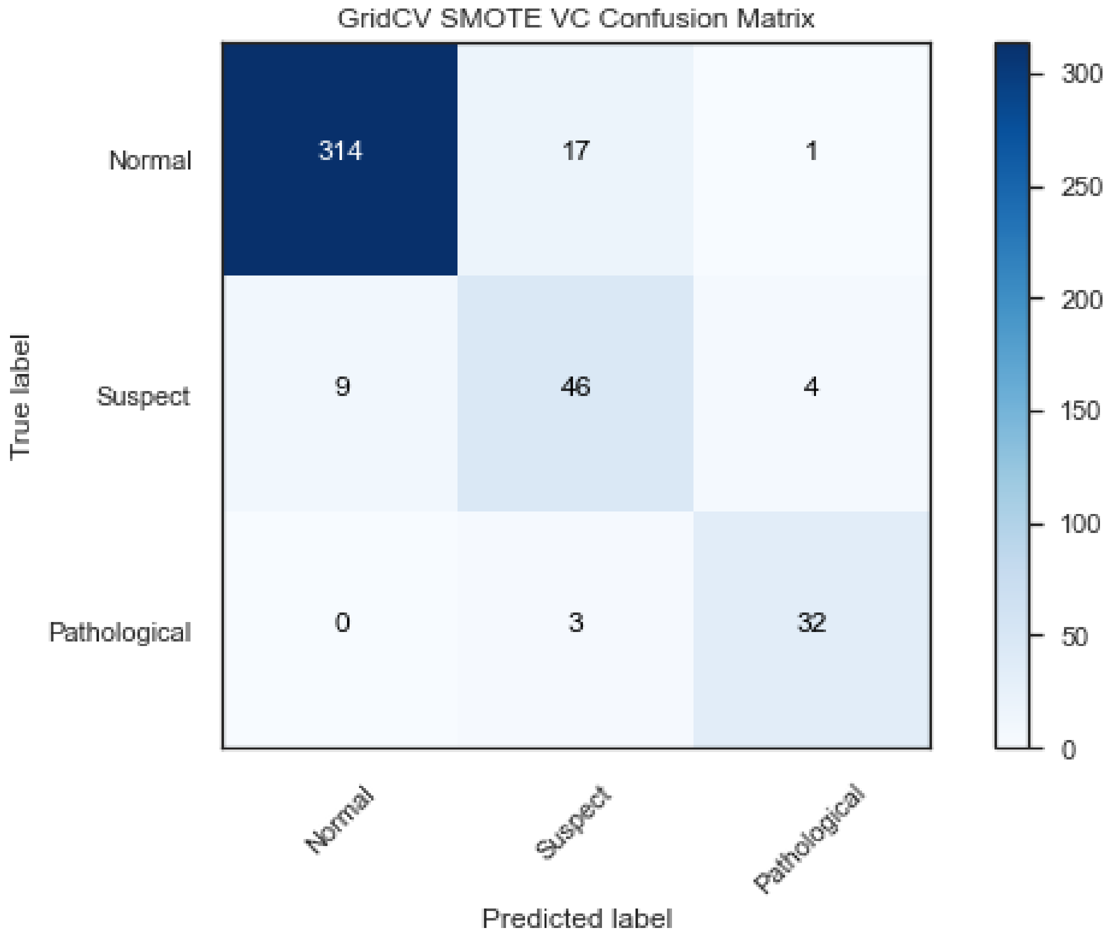
```
Test Accuracy: 0.92018779342723
Test F1: 0.9221821946664437
Test Recall: 0.92018779342723
```

[76]:

```python
cm_gcv_vc_smote_cv=metrics.confusion_matrix(y_test,GridSearchCV_vc_preds)
plot_confusion_matrix(cm_gcv_vc_smote_cv, classes=['Normal', 'Suspect',␣
 ↪'Pathological'],
                      title='GridCV SMOTE VC Confusion Matrix')
```

```
Confusion Matrix, without normalization
[[314  17   1]
```

```
 [  9  46   4]
 [  0   3  32]]
```



[77]:
```python
print(classification_report(y_test, GridSearchCV_vc_preds))
```

```
              precision    recall  f1-score   support

         1.0       0.97      0.95      0.96       332
         2.0       0.70      0.78      0.74        59
         3.0       0.86      0.91      0.89        35

    accuracy                           0.92       426
   macro avg       0.84      0.88      0.86       426
weighted avg       0.93      0.92      0.92       426
```

0.0.19 Model Comparison

Accuracy Score

[78]:
```python
acc_results = pd.DataFrame({
                    'Model': ['Baseline Logistic Regression', 'SMOTE␣
 ↪Logistic Regression',
                              'GridSearchCV SMOTE Logistic␣
 ↪Regression','SMOTE KNN',
                              'GridSearchCV SMOTE KNN', 'SMOTE Decision␣
 ↪Tree',
                              'SMOTE Random Forest','GridSearchCV SMOTE␣
 ↪Random Forest',
                              'SMOTE XGBoost', 'SMOTE ADABoost', 'SMOTE␣
 ↪Voting Classifier',
                              'GridSearchCV SMOTE Voting Classifier'
                             ],
                    'Accuracy Score': [logreg_acc, accuracy_score(y_test,␣
 ↪smote_preds),
                                        accuracy_score(y_test,␣
 ↪GridSearchCV_LR_preds),
                                        knn_acc, accuracy_score(y_test,␣
 ↪GridSearchCV_knn_preds),
                                        dt_acc, rf_acc, accuracy_score(y_test,␣
 ↪GridSearchCV_rf_preds),
                                        xgb_acc, ada_acc,vc_acc,
                                        accuracy_score(y_test,␣
 ↪GridSearchCV_vc_preds)
                                       ]
                    })
acc_result_df = acc_results.sort_values(by="Accuracy Score", ascending=False)
acc_result_df = acc_result_df.set_index("Accuracy Score")
acc_result_df
```

[78]:

Accuracy Score	Model
0.924883	GridSearchCV SMOTE Random Forest
0.920188	SMOTE Random Forest
0.920188	SMOTE Voting Classifier
0.920188	GridSearchCV SMOTE Voting Classifier
0.913146	SMOTE ADABoost
0.903756	SMOTE Decision Tree
0.903756	SMOTE XGBoost
0.889671	GridSearchCV SMOTE KNN
0.884977	Baseline Logistic Regression
0.880282	SMOTE KNN
0.863850	SMOTE Logistic Regression
0.863850	GridSearchCV SMOTE Logistic Regression

Recall Score

[79]:
```
recall_results = pd.DataFrame({
                        'Model': ['Baseline Logistic Regression', 'SMOTE␣
↪Logistic Regression',
                                  'GridSearchCV SMOTE Logistic␣
↪Regression','SMOTE KNN',
                                  'GridSearchCV SMOTE KNN', 'SMOTE Decision␣
↪Tree',
                                  'SMOTE Random Forest','GridSearchCV SMOTE␣
↪Random Forest',
                                  'SMOTE XGBoost', 'SMOTE ADABoost', 'SMOTE␣
↪Voting Classifier',
                                  'GridSearchCV SMOTE Voting Classifier'
                                 ],
                        'Recall Score': [logreg_recall, recall_score(y_test,␣
↪smote_preds, average='weighted'),
                                         recall_score(y_test,␣
↪GridSearchCV_LR_preds, average='weighted'),
                                         knn_recall,recall_score(y_test,␣
↪GridSearchCV_knn_preds, average='weighted'),
                                         dt_recall, rf_recall,
                                         recall_score(y_test,␣
↪GridSearchCV_rf_preds, average='weighted'),
                                         xgb_recall, ada_recall, vc_recall,
                                         recall_score(y_test,␣
↪GridSearchCV_vc_preds, average='weighted'),

                                        ]
                        })
recall_result_df = recall_results.sort_values(by="Recall Score",␣
↪ascending=False)
recall_result_df = recall_result_df.set_index("Recall Score")
recall_result_df
```

[79]:
```
                                               Model
Recall Score
0.924883          GridSearchCV SMOTE Random Forest
0.920188                       SMOTE Random Forest
0.920188                   SMOTE Voting Classifier
0.920188      GridSearchCV SMOTE Voting Classifier
0.913146                            SMOTE ADABoost
0.903756                       SMOTE Decision Tree
0.903756                             SMOTE XGBoost
0.889671                    GridSearchCV SMOTE KNN
0.884977              Baseline Logistic Regression
0.880282                                 SMOTE KNN
0.863850                 SMOTE Logistic Regression
```

```
0.863850      GridSearchCV SMOTE Logistic Regression
```

F1 Score

[80]:
```python
f1_results = pd.DataFrame({
                        'Model': ['Baseline Logistic Regression', 'SMOTE␣
  ↪Logistic Regression',
                                  'GridSearchCV SMOTE Logistic␣
  ↪Regression','SMOTE KNN',
                                  'GridSearchCV SMOTE KNN', 'SMOTE Decision␣
  ↪Tree',
                                  'SMOTE Random Forest','GridSearchCV SMOTE␣
  ↪Random Forest',
                                  'SMOTE XGBoost', 'SMOTE ADABoost', 'SMOTE␣
  ↪Voting Classifier',
                                  'GridSearchCV SMOTE Voting Classifier'
                                 ],
                        'F1 Score': [logreg_f1, f1_score(y_test, smote_preds,␣
  ↪average='weighted'),
                                     f1_score(y_test, GridSearchCV_LR_preds,␣
  ↪average='weighted'),
                                     knn_f1,f1_score(y_test,␣
  ↪GridSearchCV_knn_preds, average='weighted'),
                                     dt_f1, rf_f1, f1_score(y_test,␣
  ↪GridSearchCV_rf_preds, average='weighted'),
                                     xgb_f1, ada_f1,vc_f1,
                                     f1_score(y_test, GridSearchCV_vc_preds,␣
  ↪average='weighted'),

                                    ]
                        })
f1_result_df = f1_results.sort_values(by="F1 Score", ascending=False)
f1_result_df = f1_result_df.set_index("F1 Score")
f1_result_df
```

[80]:

F1 Score	Model
0.925492	GridSearchCV SMOTE Random Forest
0.922182	GridSearchCV SMOTE Voting Classifier
0.922004	SMOTE Voting Classifier
0.920415	SMOTE Random Forest
0.911824	SMOTE ADABoost
0.907323	SMOTE XGBoost
0.899278	SMOTE Decision Tree
0.895442	GridSearchCV SMOTE KNN
0.886384	SMOTE KNN
0.885471	Baseline Logistic Regression

```
0.875537  GridSearchCV SMOTE Logistic Regression
0.875375               SMOTE Logistic Regression
```