Detection of a fraudulent insurance claim project

Insurance fraud is any act committed to defraud an insurance process.It occurs when a claimant attempts to obtain some benefit or advantage they are not entitled to, or when an insurer knowingly denies some benefit that is due.

Every year Fraud insurance represent major losses for insurance companies ($40 billion per year according to the FBI).

The dataset : The dataset is fromm kaggle and is composed of 1000 auto incidents and auto insurance claims from Illinois, Ohio and Indiande from 01 january 2015 to 01 march 2015 and we do not know if it is from multiple company or just one company.

Goal of the project : The main of this project is to build the best model to detect auto insurance fraud.We will use pycaret library to know which model fits the best to be used at the end of the project.

Ressources :

-How to use special machine learning techniques to detect insurance claims: fraud
https://www.youtube.com/watch?v=4Ru3330TnYw&t=1076s

For Real? Auto Insurance Fraud Claim Detection with Machine Learning :
https://towardsdatascience.com/for-real-auto-insurance-fraud-claim-detection-with-machine-learning-efcf957b38f3

Automate Anomaly Detection Using Pycaret -Data Science And Machine Learning :
https://www.youtube.com/watch?v=kbssBdFf764

In [1]:

```
#Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sweetviz as sv
import pandas_profiling
import plotly.express as px
sns.set_theme(style="ticks", color_codes=True)
```

In [2]:

```
pip install shap
```

```
Requirement already satisfied: shap in c:\users\iscar\anacond
a3\envs\gputest\lib\site-packages (0.37.0)Note: you may need
to restart the kernel to use updated packages.
Requirement already satisfied: scipy in c:\users\iscar\anacon
da3\envs\gputest\lib\site-packages (from shap) (1.5.2)
Requirement already satisfied: numpy in c:\users\iscar\anacon
da3\envs\gputest\lib\site-packages (from shap) (1.19.2)
Requirement already satisfied: numba in c:\users\iscar\anacon
```

```
da3\envs\gputest\lib\site-packages (from shap) (0.52.0)
Requirement already satisfied: scikit-learn in c:\users\iscar
\anaconda3\envs\gputest\lib\site-packages (from shap) (0.23.
2)
Requirement already satisfied: slicer==0.0.3 in c:\users\isca
r\anaconda3\envs\gputest\lib\site-packages (from shap) (0.0.
3)
Requirement already satisfied: tqdm>4.25.0 in c:\users\iscar
\anaconda3\envs\gputest\lib\site-packages (from shap) (4.54.
1)
Requirement already satisfied: pandas in c:\users\iscar\anaco
nda3\envs\gputest\lib\site-packages (from shap) (1.1.3)
Requirement already satisfied: setuptools in c:\users\iscar\a
naconda3\envs\gputest\lib\site-packages (from numba->shap) (5
0.3.1.post20201107)
Requirement already satisfied: llvmlite<0.36,>=0.35.0 in c:\u
sers\iscar\anaconda3\envs\gputest\lib\site-packages (from num
ba->shap) (0.35.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\use
rs\iscar\anaconda3\envs\gputest\lib\site-packages (from sciki
t-learn->shap) (2.1.0)
Requirement already satisfied: joblib>=0.11 in c:\users\iscar
\anaconda3\envs\gputest\lib\site-packages (from scikit-learn-
>shap) (0.17.0)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\u
sers\iscar\anaconda3\envs\gputest\lib\site-packages (from pan
das->shap) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\iscar
\anaconda3\envs\gputest\lib\site-packages (from pandas->shap)
(2020.1)
Requirement already satisfied: six>=1.5 in c:\users\iscar\ana
conda3\envs\gputest\lib\site-packages (from python-dateutil>=
2.7.3->pandas->shap) (1.15.0)
```

In [3]:

```
data = pd.read_csv('insurance_claims.csv')
```

EXPLORATORY DATA ANALYSIS

In [4]:

```
data
```

Out[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	poli
0	328	48	521585	17-10-2014	OH	2
1	228	42	342868	27-06-2006	IN	2
2	134	29	687698	06-09-2000	OH	1
3	256	41	227811	25-05-1990	IL	2
4	228	44	367455	06-06-2014	IL	50
...	...	...	...	...	...	

995	3	38	941851	16-07-1991	OH	50
996	285	41	186934	05-01-2014	IL	1
997	130	34	918516	17-02-2003	OH	2
998	458	62	533940	18-11-2011	IL	50
999	456	60	556080	11-11-1996	OH	2

1000 rows × 39 columns

In [5]:

```
#Number of lines and column
data.shape
```

Out[5]:

```
(1000, 39)
```

Before advancing more in the data analyis of the dataset let's make a quick overview of different trends , relationship... of the dataset with pandas_profiling

In [6]:

```
data.profile_report()
```

Pandas Profiling Report

Overview

Overview | Warnings 17 | Reproduction

Dataset statistics

Number of variables	39
Number of observations	1000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	304.8 KiB
Average record size in memory	312.1 B

Variable types

CAT	24
NUM	14
BOOL	1

Variables

Out[6]:

The pandas profiling report allows us to have a quick overview of the dataset from the correlation to

the trends of the dataset.

-There is 1000 insurance claim and in those insurance 247 are fraudulent

- Oldest person : 64 years Youngest person : 19 years (medium >>> 39 years)
- Insured sex : 537 women and 463 men -Insured hobbies , occupation , relationships etc... -Month as customer and age have a correlation of 0.92. Probably because drivers buy auto insurance when they own a car and this time measure only increases with age. -Other correlations are pretty clear and apart of that there seems not to have much correlation in the data

In the following let's see if there is any relationship between the gender , the hobbies , occupation , relationships and a fraudulent insurance claim

In [7]:

```
#Number of fraudulent insurance claim
data.fraud_reported.value_counts()
```

Out[7]:

```
N    753
Y    247
Name: fraud_reported, dtype: int64
```

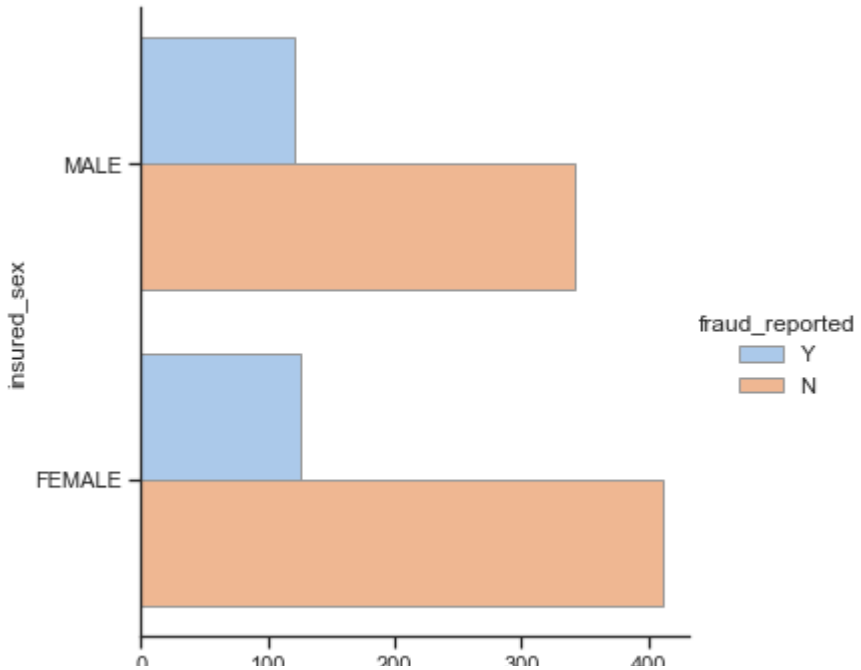
In [8]:

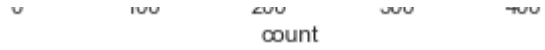
```
#Creating a dataframe with gender and fraud reported

Gender = data[['insured_sex','fraud_reported']]
Gender.head()
#Plotting gender and fraud reported
sns.catplot(y="insured_sex", hue="fraud_reported", kind="count",
            palette="pastel", edgecolor=".6",
            data=Gender)
```

Out[8]:

```
<seaborn.axisgrid.FacetGrid at 0x23efb7dc848>
```





HOBBIES AND FRAUD REPORTED

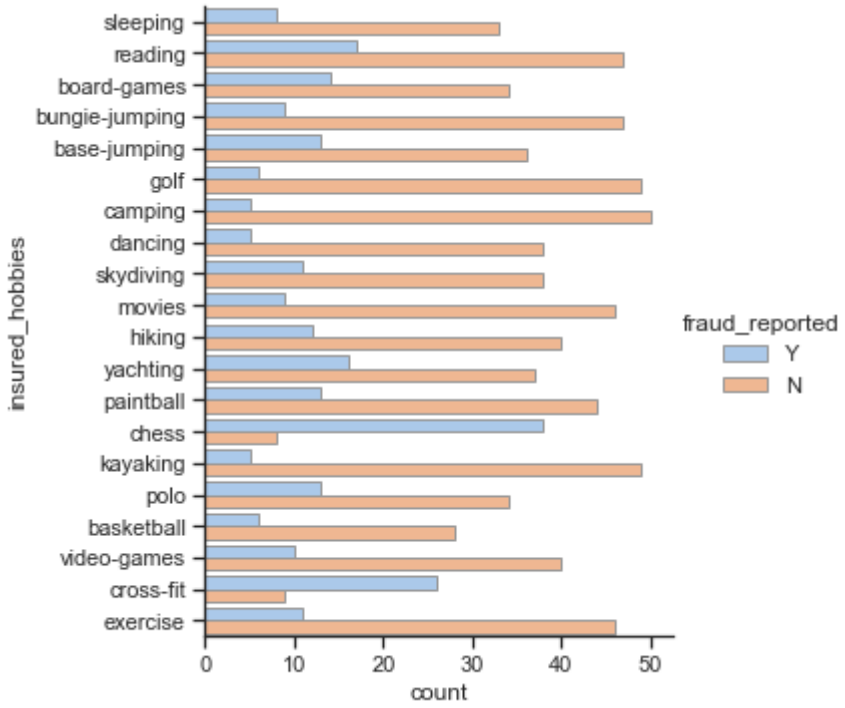
In [9]:

```
#Creating a dataframe with hobbies and fraud reported

Hobbies = data[['insured_hobbies','fraud_reported']]
Hobbies.head()
#Plotting hobbies and fraud reported
sns.catplot(y="insured_hobbies", hue="fraud_reported", kind="count",
            palette="pastel", edgecolor=".6",
            data=Hobbies)
```

Out[9]:

```
<seaborn.axisgrid.FacetGrid at 0x23efc8ee688>
```



Even though hobbies are not necessarily a factor of a fraudulent claim it appears that chess players tend to have more often fraudulent claim than other insured who have other hobbies.

Education level and fraud reported

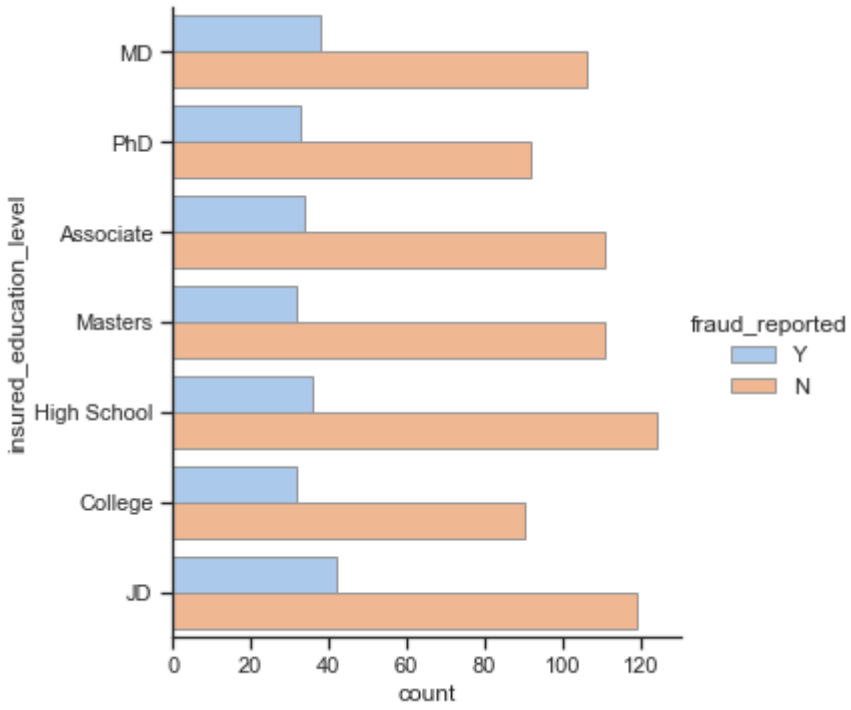
In [10]:

```
#Creating a dataframe with education level  and fraud reported

Education = data[['insured_education_level','fraud_reported']]
Education.head()
#Plottingeducation level  and fraud reported
sns.catplot(y="insured_education_level", hue="fraud_reported", kind="coun
t",
            palette="pastel", edgecolor=".6",
            data=Education)
```

Out[10]:

```
<seaborn.axisgrid.FacetGrid at 0x23e800516c8>
```



In general the more a person is highly educated , the less he tends to make a fraudulent claim.

Fraud reported and total claim amount

In [11]:

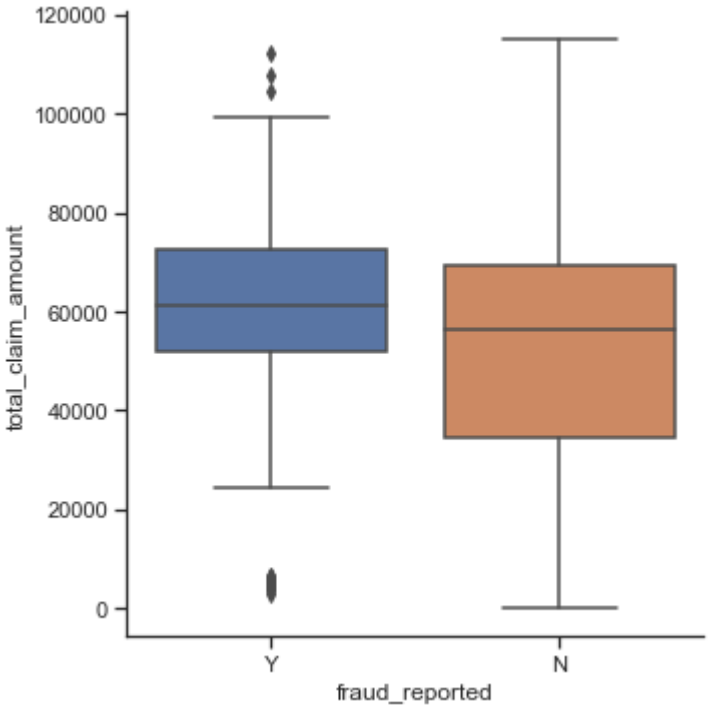
```
#Creating a dataframe with hobbies and fraud reported

Claim_amount = data[['total_claim_amount','fraud_reported']]
Claim_amount.head()

sns.catplot(x="fraud_reported", y="total_claim_amount", hue="fraud_reported",
            kind="box", dodge=False, data=Claim_amount)
```

Out[11]:

```
<seaborn.axisgrid.FacetGrid at 0x23e8018b348>
```



For further exploratory data analysis you can check the html report :

In [12]:

```
#Creating a quick  report to view the different relationship between diffe
rent variables  and the fraud_reported
my_report = sv.analyze([data, "Train"],target_feat='fraud_reported')
my_report.show_html()
```

```
Report SWEETVIZ_REPORT.html was generated! NOTEBOOK/COLAB USE
RS: the web browser MAY not pop up, regardless, the report IS
saved in your notebook/colab files.
```

The best model to predict a fraudulent insurance claim

The model that we will choose will have to be able to classify if a claim is a fraud or not on a data set that it has not seen, accurately.

To find the best model we will use the Pycaret's classification Module. **Pycaret's classification Module** is a supervised machine learning module which is used for classifying elements into groups.

The goal is to predict the categorical class labels which are discrete and unordered. Some common use cases include predicting customer default (Yes or No), predicting customer churn (customer will leave or stay), disease found (positive or negative). This module can be used for binary or multiclass problems. It provides several pre-processing features that prepare the data for modeling through setup function. It has over 18 ready-to-use algorithms and several plots to analyze the performance of trained models.So, it appears that it is the perfect module for our case.

This module provide several pre-processing features that prepares the data for modeling through setup function.

In [13]:

```
#Importing the module
from pycaret.classification import *
#Initializing the setup
exp_clf = setup(data, target = 'fraud_reported')
```

	Description	Value
0	session_id	5542
1	Target	fraud_reported
2	Target Type	Binary
3	Label Encoded	N: 0, Y: 1
4	Original Data	(1000, 39)
5	Missing Values	False
6	Numeric Features	13
7	Categorical Features	23
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(699, 912)
12	Transformed Test Set	(301, 912)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	2ec5
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None

26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	False
37	Combine Rare Levels	False
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Clustering	False
45	Clustering Iteration	None
46	Polynomial Features	False
47	Polynomial Degree	None
48	Trignometry Features	False
49	Polynomial Threshold	None
50	Group Features	False
51	Feature Selection	False
52	Features Selection Threshold	None
53	Feature Interaction	False
54	Feature Ratio	False
55	Interaction Threshold	None
56	Fix Imbalance	False
57	Fix Imbalance Method	SMOTE

The setup with pycaret allows us to preprocess way more quickly data than if we had used sklearn. Note that the data have been split here 70/30 the 30% will be used to predict .

In [22]:

```
#Getting the accuracy , recall , F1 score of the different models
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (S
catboost	CatBoost Classifier	0.8240	0.8641	0.6137	0.6703	0.6349	0.5201	0.5252	8.7
lightgbm	Light Gradient Boosting Machine	0.8212	0.8707	0.5912	0.6739	0.6218	0.5067	0.5139	0.2
gbc	Gradient Boosting Classifier	0.8198	0.8684	0.6438	0.6524	0.6408	0.5219	0.5271	0.5
dt	Decision Tree Classifier	0.8169	0.7478	0.6082	0.6569	0.6247	0.5053	0.5102	0.0
xgboost	Extreme Gradient Boosting	0.8140	0.8559	0.5801	0.6649	0.6121	0.4912	0.4984	0.9
ada	Ada Boost Classifier	0.8070	0.8018	0.5542	0.6434	0.5844	0.4620	0.4712	0.2
et	Extra Trees Classifier	0.7539	0.8487	0.1595	0.5339	0.2387	0.1452	0.1829	0.3
lr	Logistic Regression	0.7525	0.5962	0.0278	0.3500	0.0511	0.0341	0.0728	0.4
lda	Linear Discriminant Analysis	0.7525	0.8276	0.0451	0.4000	0.0802	0.0496	0.0851	0.6
rf	Random Forest Classifier	0.7511	0.8427	0.0578	0.5567	0.1013	0.0599	0.1117	0.3
ridge	Ridge Classifier	0.7496	0.0000	0.0056	0.1000	0.0105	0.0080	0.0205	0.1
qda	Quadratic Discriminant Analysis	0.7482	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.4
knn	K Neighbors Classifier	0.7025	0.5024	0.1301	0.3320	0.1783	0.0330	0.0454	0.1
nb	Naive Bayes	0.6723	0.5769	0.2052	0.2948	0.2330	0.0380	0.0408	0.0
svm	SVM - Linear Kernel	0.6611	0.0000	0.2337	0.1928	0.1371	0.0324	0.0441	0.1

Out[22]:

```
<catboost.core.CatBoostClassifier at 0x23edd9919c8>
```

Well, it appears that by executing various machine learning models the Gradient catboost classifier is the best model. Usually to find out this we should have take much more time.But, thanks to pycaret we can quickly see the model that fits well to our dataset.

Gradient Boost Classifier had an accuracy of 0.8240 , an F1-score of 0.6349

If you want to learn more about the gradient boost classifier you can check the excellent channel of **StatQuest with Josh Starmer** : https://www.youtube.com/watch?v=jxuNLH5dXCs

In [23]:

```
#Creating the model
Gradient = create_model('gbc')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8000	0.8213	0.7647	0.5652	0.6500	0.5144	0.5259
1	0.8571	0.9478	0.6471	0.7333	0.6875	0.5954	0.5973
2	0.7857	0.8529	0.7059	0.5455	0.6154	0.4702	0.4777
3	0.8000	0.8622	0.5000	0.6429	0.5625	0.4355	0.4413
4	0.8143	0.9156	0.6667	0.6316	0.6486	0.5226	0.5229
5	0.8286	0.9156	0.5000	0.7500	0.6000	0.4964	0.5129
6	0.7857	0.7559	0.5556	0.5882	0.5714	0.4287	0.4290
7	0.8429	0.9199	0.6111	0.7333	0.6667	0.5650	0.5690
8	0.8429	0.8809	0.7222	0.6842	0.7027	0.5960	0.5964
9	0.8406	0.8117	0.7647	0.6500	0.7027	0.5948	0.5984
Mean	0.8198	0.8684	0.6438	0.6524	0.6408	0.5219	0.5271
SD	0.0247	0.0565	0.0949	0.0688	0.0489	0.0613	0.0602

Hyperparameter Tuning in PyCaret can also be done in a single line of code. This is done through random grid search through predefined grids which can be customized.

In [24]:

```
#Tuning the model
tune_model(Gradient)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8000	0.8873	0.7647	0.5652	0.6500	0.5144	0.5259
1	0.8286	0.9279	0.6471	0.6471	0.6471	0.5339	0.5339
2	0.8143	0.8935	0.7059	0.6000	0.6486	0.5236	0.5268
3	0.8571	0.8702	0.6111	0.7857	0.6875	0.5968	0.6047
4	0.8143	0.9124	0.7222	0.6190	0.6667	0.5390	0.5421
5	0.8571	0.9322	0.6111	0.7857	0.6875	0.5968	0.6047
6	0.8000	0.7703	0.6667	0.6000	0.6316	0.4948	0.4961
7	0.8143	0.8515	0.3889	0.7778	0.5185	0.4189	0.4576
8	0.8571	0.8996	0.7222	0.7222	0.7222	0.6261	0.6261
9	0.8261	0.8360	0.7059	0.6316	0.6667	0.5495	0.5511
Mean	0.8269	0.8781	0.6546	0.6734	0.6526	0.5394	0.5469
SD	0.0216	0.0464	0.1006	0.0815	0.0511	0.0562	0.0496

Out[24]:

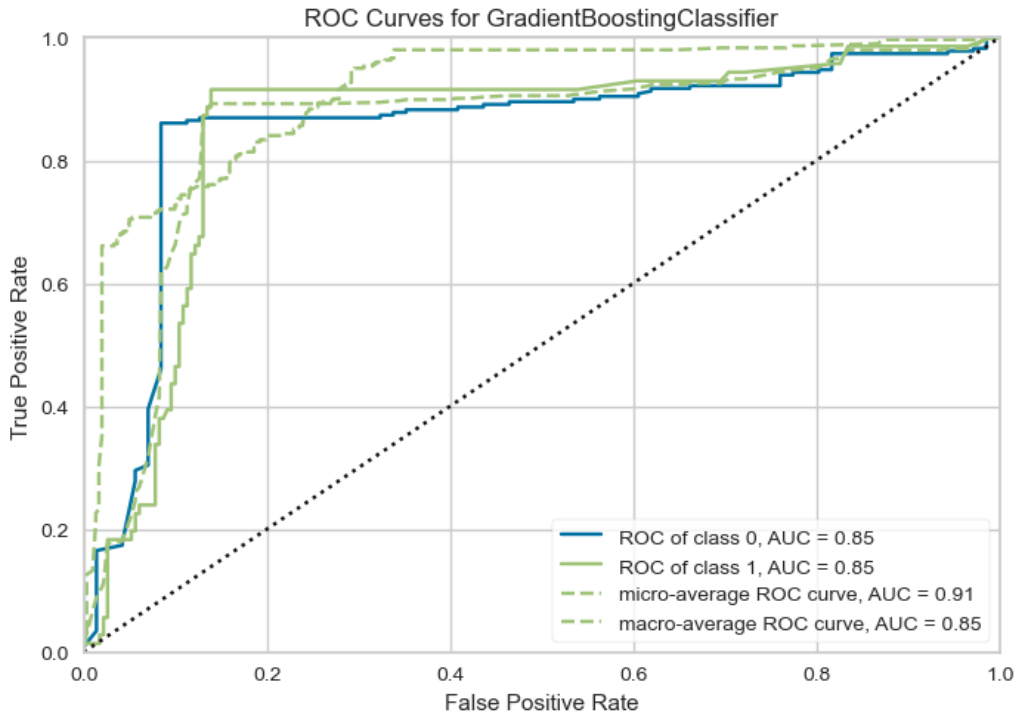
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman
```

```
_mse', init=None,
                           learning_rate=0.47, loss='devianc
e', max_depth=9,
                           max_features=1.0, max_leaf_nodes=N
one,
                           min_impurity_decrease=0, min_impur
ity_split=None,
                           min_samples_leaf=1, min_samples_sp
lit=9,
                           min_weight_fraction_leaf=0.0, n_es
timators=160,
                           n_iter_no_change=None, presort='de
precated',
                           random_state=5542, subsample=0.95,
tol=0.0001,
                           validation_fraction=0.1, verbose=
0,
                           warm_start=False)
```

Well the results has deteriorated so it is better that we continue without tuning.

In [25]:

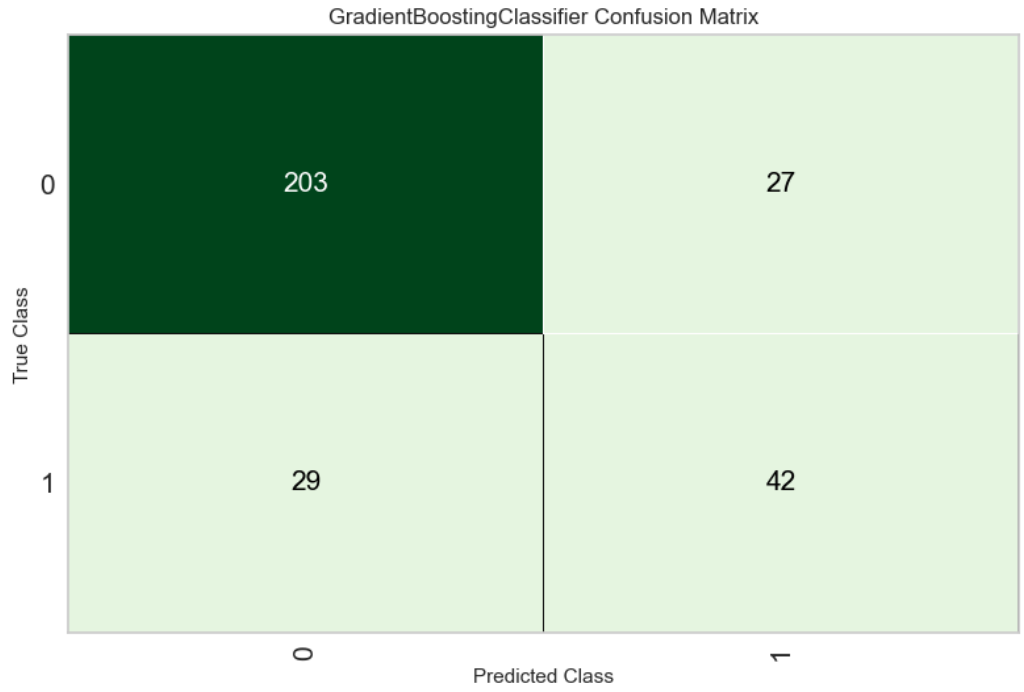
```
#Vizualizing the model
plot_model(Gradient)
```



Area under ROC (Receiver Operating Characteristics) curve tells us how good the model is at distinguishing between classes - predicting fraudulent as fraudulent, and not fraudulent as not fraudulent

In [26]:

```
#plotting the confusion matrix
plot_model(Gradient, 'confusion_matrix')
```



Again with the confusion matrix we can see that the model seems to perform pretty well.Of course the model can be better if we had for example a larger dataset.

Predictions and saving the model

As mentioned previously 70% of the data was used in training the model.Now let's use the rest of the data for testing the model and predicting our model result.

In [27]:

```
predictions = predict_model(Gradient)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Gradient Boosting Classifier	0.8140	0.8521	0.5915	0.6087	0.6000	0.4788	0.4789

So we get an accuracy of 0.8173 and recall of 0.7864 , Precision 0.6087

Actually the problem of the precision of the model can be explained by the fact that we really had a very tiny dataset and as we know statistical models perform better when data sets are larger. Also , the period is only from january 2015 to march 2015 , then we do no have a better review of the entire year.

In [28]:

```
#Saving the model
save_model(Gradient, model_name = 'deployment_28042020')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[28]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=
[],
                                       display_types=True, fe
atures_todrop=[],
                                       id_columns=[],
                                       ml_usecase='classifica
tion',
                                       numerical_features=[],
                                       target='fraud_reporte
d',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='not_av
ailable',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None,
                                 numer...
                                             learning_rate=0.
1, loss='deviance',
                                             max_depth=3, max
_features=None,
                                             max_leaf_nodes=N
one,
                                             min_impurity_dec
rease=0.0,
                                             min_impurity_spl
it=None,
                                             min_samples_leaf
=1,
                                             min_samples_spli
t=2,
                                             min_weight_fract
ion_leaf=0.0,
                                             n_estimators=10
0,
                                             n_iter_no_change
=None,
                                             presort='depreca
ted',
                                             random_state=554
2, subsample=1.0,
                                             tol=0.0001, vali
dation_fraction=0.1,
                                             verbose=0, warm_
start=False)]],
          verbose=False),
 'deployment 28042020.pkl')
```