Final, Case Study

Sanjay Pillay - Nov 21, 2021

Abstract

The primary aim of this case study is to build a predictive model to reduce the financial loss for our client who makes a loss for every wrong class prediction made.

1 Introduction

The problem statement presented for this study was to reduce the financial loss by making accurate predictions based on a set of masked data. Currently every prediction that misclassifies the positive class (binary 1) incurs a loss of $100 and misclassifying the negative (binary 0) incurs a loss of $25. The goal is to come up with an optimum supervised learning model that reduces the overall monetary loss.

2 Method

The labeled dataset with masked features was analyzed, imputed, scaled and used to train three supervised learning models. First a Random Forest (RF) model was used to establish a base line accuracy followed by XGBoost (xgb) and a Dense Neural Network (NN) model to improve on the baseline accuracy. Model parameters were tuned using roc_auc score for RF and XGB and binary cross entropy loss for NN, appropriate early stopping and patience was used to halt training when the gain stops increasing. The best tuned models were than compared for best f1 score, its confusion matrix (CM) and ROC/AUC. The model with best highest f1 score was used to optimize class threshold calculate financial loss using a saved holdout dataset. A common method was created to calculate the financial loss of a model based on CM results.

2.1 Data

The data consists of 50 masked features of 160,000 rows that had binary labels of 1 for the positive class and 0 for negative class.
The correlation analysis [6] identified two sets of attributes having 100% correlation x2, x6 and x38, x41. We chose to drop columns x2 and x41 as these had a higher number of missing values compared to its corelated column [Table 1].

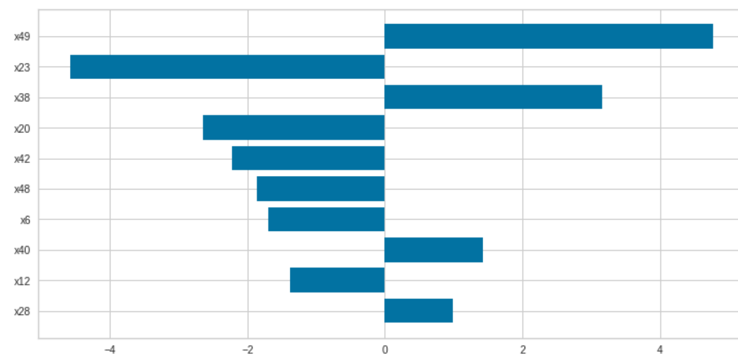
Table 1

Column	**Missing Value Count**	**Corelation %**
X2, x6	X2=38, x6=26	100
X38, x41	X38=31, x41=40	100

Column x37 which apparently was a cost attribute was modified to remove $ sign and column x32 which was a percent attribute was modified to strip % sign. These two columns were then converted to float type. Column x29 which was an attribute having month, one value ‘sept.’ was

the only anomaly with a period at the end so that period was stripped out (This prevented the MICE imputation package from tripping while generating the linear regression formula for imputation).
Column x24 had a region data and had missing values for 28 rows, column x29 had month data with 30 rows of missing values and x30 had day of the week data with 30 rows of missing values. Since these three categorical columns had a total of 88 rows or .055% of missing data we completely removed from the dataset and one hot encoded the remaining.
All other feature columns had some degree of missing values at random in the range of .02 to .03%. We used Multivariate Imputation by Chained Equations (MICE) algorithm to impute the remaining dataset [6]. The shape of the final data was 62 features and 159912 rows.
Figure 1 shows the top 10 feature importance using a Logistic Regression model, we chose to keep all the features.

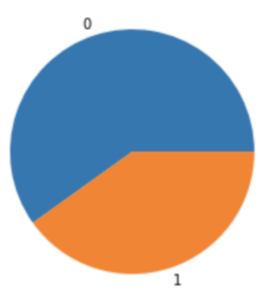
Figure 1 Variable Importance



The target class had a significant imbalance shown in [Figure 2]. Positive class was at about 40% and negative was 60%, the study took into account this imbalance by using stratified shuffle splits and balanced class.

Figure 2 Class Imbalance

```
Total Records 159912
Total Classes: 2
Class Gini Index 0.4804828175501279
Smallest Class Id: 1 Records: 64159
Largest Class Id: 0 Records: 95753
          y  Percentage
0    95753    0.598786
1    64159    0.401214
```



2.2 Models

Three models were evaluated, the first one was RF which served as our base line model followed by XGB and NN. Using Stratified Shuffle Split 10% of the data was reserved as hold out for final testing. The models were trained using 80/20 Train/Test stratified shuffle split and 5-fold cross validation using the remaining 90% of the data. Grid search was used for RF and XGB model for parameter tuning and for NN a common function was created that served to quickly evaluate various NN layer parameters.

The three models with best tuning parameters was compared using the holdout set for the total monetary loss.

Table 2 shows the final shape of our Train/Test/Holdout datasets.

Table 2

Hold Out	Training	Test
(15992, 62) (15992,)	(115136, 62) (115136,)	(28784, 62) (28784,)

2.2.1 Random Forest

Grid search identified the following parameters with highest AUC score of 0.97804
{'class_weight': 'balanced', 'max_features': 25, 'min_samples_leaf': 5, 'n_estimators': 250, 'random_state': 45}

2.2.2 XG Boost

An **early stopping of 5** was used for XGB, Grid search identified the following parameters with highest AUC score of 0.98383
{'booster': 'gbtree', 'colsample_bytree': 0.7, 'eval_metric': 'logloss', 'gamma': 4, 'learning_rate': 0.01, 'max_depth': 12, 'n_estimators': 1000, 'num_classes': 2, 'objective': 'binary:logistic', 'random_state': 45, 'verbose_eval': True}

2.2.3 Dense Neural Network

A **patience** of 50 was used to stop training when no gain is achieved in accuracy for the last 50 epochs. Simple, medium and complex set of NN models [6] were evaluated and the best model was the one with medium complexity with the best score of 0.9712.

Figure 3, shows the model parameters. The model used 800 epoch and stopped training at epoch 209, 'relu' activation and BinaryCrossEntropy Loss, with a total of 78339 trainable parameters.

Figure 3

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               16128

dense_1 (Dense)              (None, 176)               45232

dense_2 (Dense)              (None, 96)                16992

dense_3 (Dense)              (None, 1)                 97

=================================================================
Total params: 78,449
Trainable params: 78,449
Non-trainable params: 0
```

3 Results

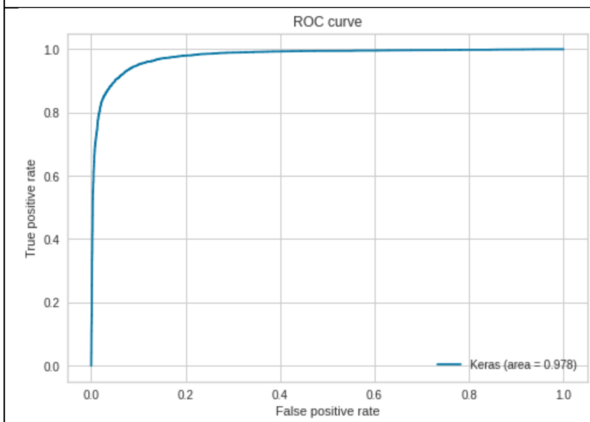
The results are based on a common set of 15992 rows holdout set that was not used in the model training to avoid any bias.

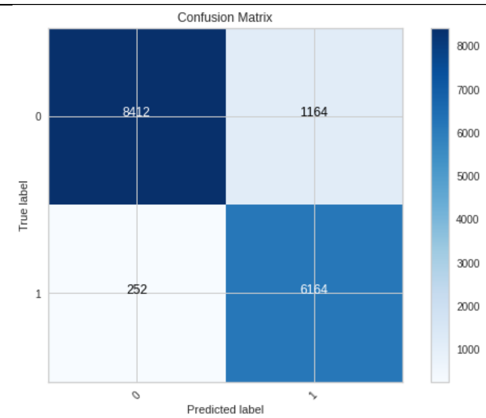
3.1.1 Baseline RF model

The RF model had a total monetary loss of 54300 after adjusting the class threshold at 0.35%. The model's f1 score was 91% with positive class f1 score of 90% and negative of 92%. Table 3 shows the classification report, ROC Curve and CM.

Table 3 RF Results

	precision	recall	f1-score	support
0	0.97	0.88	0.92	9576
1	0.84	0.96	0.90	6416
accuracy			0.91	15992
macro avg	0.91	0.92	0.91	15992
weighted avg	0.92	0.91	0.91	15992





3.1.2 Comparing XGB and NN model

Both XGB and NN model improved on the RF model. The NN model was the one with least financial loss of $45825 which was an improvement of $8475 over RF and was a slight improvement over the XGB by $150. The threshold for NN had to be adjusted to 0.134. The Table 4 XGB/NN Resultssummarizes the detail results for the two models which consists of the financial loss incurred on test data, model's and class f1 score, ROC curve, CM and the training log-loss progression.

Table 4 XGB/NN Results

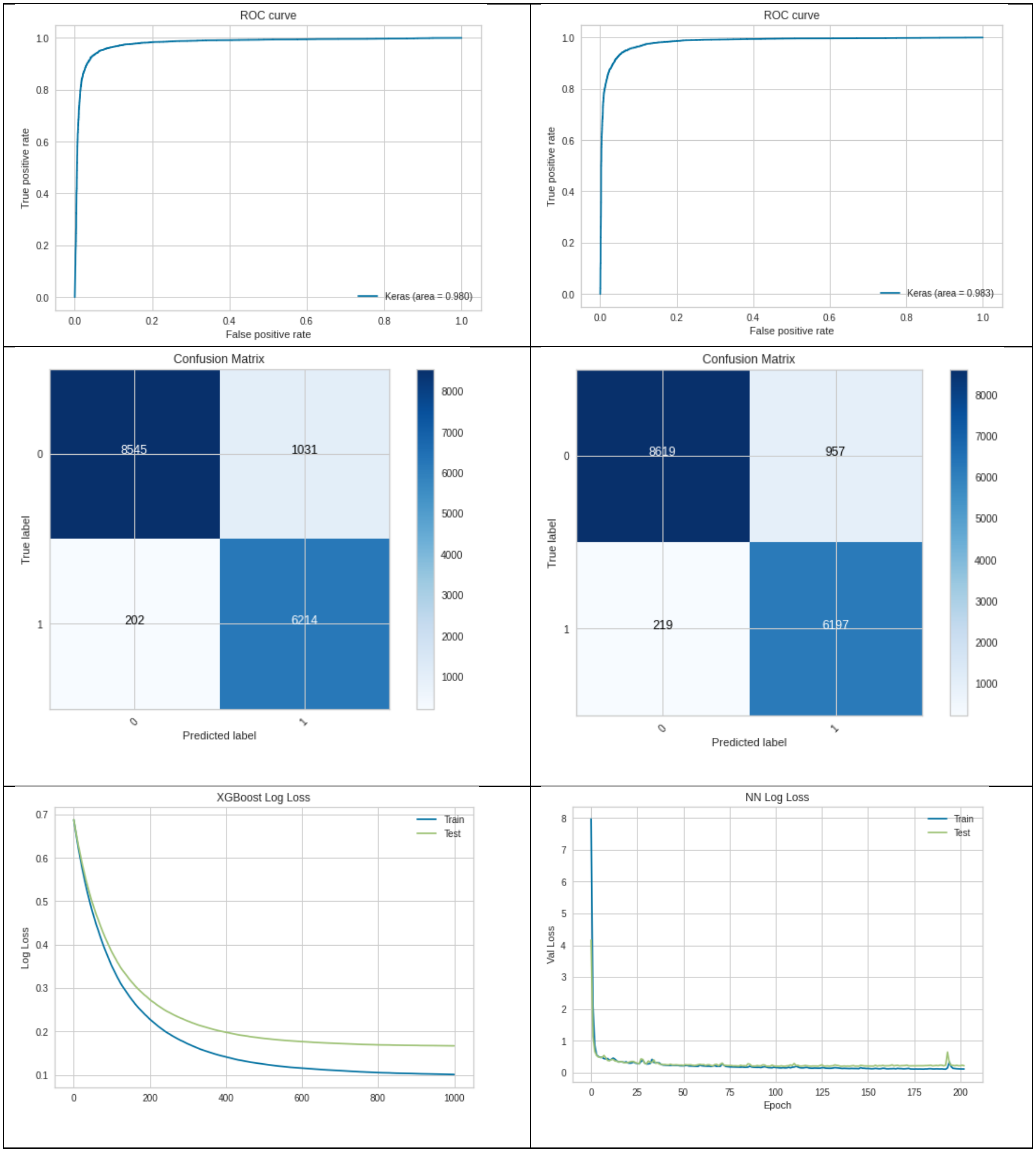
XGB	**NN**
Total Financial Loss: $45975	Total Financial Loss: $45825
Threshold: 0.25	Threshold: 0.134

XGB:

	precision	recall	f1-score	support
0	0.98	0.89	0.93	9576
1	0.86	0.97	0.91	6416
accuracy			0.92	15992
macro avg	0.92	0.93	0.92	15992
weighted avg	0.93	0.92	0.92	15992

NN:

	precision	recall	f1-score	support
0	0.98	0.90	0.94	9576
1	0.87	0.97	0.91	6416
accuracy			0.93	15992
macro avg	0.92	0.93	0.92	15992
weighted avg	0.93	0.93	0.93	15992



4 Conclusion

Both the XGB and NN model did improve significantly over the RF model, while the NN model showed a very miniscule increase in accuracy of $150 over XGB, it is worth noting from the log loss curve the NN model trained significantly faster with less epochs and better test loss, both models had a same f1 score to identify the positive class but NN was a percent point better at identifying the negative class.

5 Appendix

5.1 Code

Some of the output has been cleaned to reduce document.

```
#fimnal CS
import os
import email
import pickle
#All Python module imports
#https://pandas.pydata.org/docs/user_guide/index.html#user-guide
import pandas as pd #Pandas Dataframe module
from imblearn.over_sampling import SMOTE
import numpy as np
from math import pi
#scikit learn

#https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_
model
import sklearn as skl

#https://seaborn.pydata.org
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib

import warnings
#Module for formating table for documentation
#https://pypi.org/project/tabulate/
from tabulate import tabulate

from IPython.display import display, Markdown
#Interactive mode
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
from IPython.display import Image


from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics as mt
from sklearn.metrics import plot_confusion_matrix
```

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.model_selection import GridSearchCV as gridcv
from sklearn import preprocessing
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
import pprint
import re
from sklearn.model_selection import cross_val_predict
from html.parser import HTMLParser
from bs4 import BeautifulSoup
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from scipy.io import arff
from statsmodels.imputation import mice
import statsmodels as sm
from xgboost import XGBClassifier
from numpy import arange
from numpy import argmax
from sklearn.preprocessing import QuantileTransformer
import tensorflow as tf
print(tf.__version__)
import missingno as msno

import math
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: Fut
ureWarning: pandas.util.testing is deprecated. Use the functions in the publ
ic API at pandas.testing instead.
  import pandas.util.testing as tm
2.7.0
```

In [3]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

In [ ]:

```
df = pd.read_csv('./drive/MyDrive/data/final_project.csv')
df.shape
df.head()
df.info(verbose=True, null_counts=True)
```

Out[ ]:

```
(160000, 51)
```

Out[ ]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160000 entries, 0 to 159999
Data columns (total 51 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   x0      159974 non-null  float64
 1   x1      159975 non-null  float64
 2   x2      159962 non-null  float64
 3   x3      159963 non-null  float64
 …
 46  x46     159969 non-null  float64
 47  x47     159963 non-null  float64
 48  x48     159968 non-null  float64
 49  x49     159968 non-null  float64
 50  y       160000 non-null  int64
dtypes: float64(45), int64(1), object(5)
memory usage: 62.3+ MB
```

In [ ]:

```
df['y'].value_counts()
```

Out[ ]:

```
0    95803
1    64197
Name: y, dtype: int64
```

In [ ]:

```
df.describe([.05,.1,.25,.5,.75,.9,.95]).transpose()
```

Out[ ]:

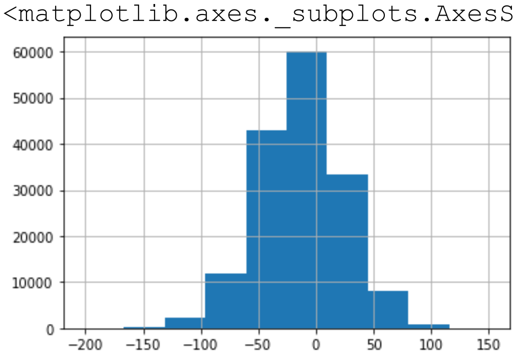
	count	mean	std	min	5%	10%	25%	50%	75%	90%	95%	max
x0	159974.0	-0.001028	0.371137	-1.592635	-0.609244	-0.476793	-0.251641	-0.002047	0.248532	0.476354	0.611374	1.600849
x1	159975.0	0.001358	6.340632	-26.278302	-10.436173	-8.121119	-4.260973	0.004813	4.284220	8.119877	10.422512	27.988178
x3	159963.0	-0.024637	8.065032	-35.476594	-13.286032	-10.367339	-5.454438	-0.031408	5.445179	10.295276	13.191297	38.906025
x4	159974.0	-0.000549	6.382293	-28.467536	-10.490097	-8.173413	-4.313118	0.000857	4.306660	8.191609	10.502674	26.247812
x49	159968.0	-0.674224	15.036738	-65.791191	-25.389774	-20.116675	-10.931753	-0.574410	9.651072	18.574212	23.969346	66.877604
y	160000.0	0.401231	0.490149	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000

In [ ]:

```
df['x46'].hist()
```

Out[ ]:

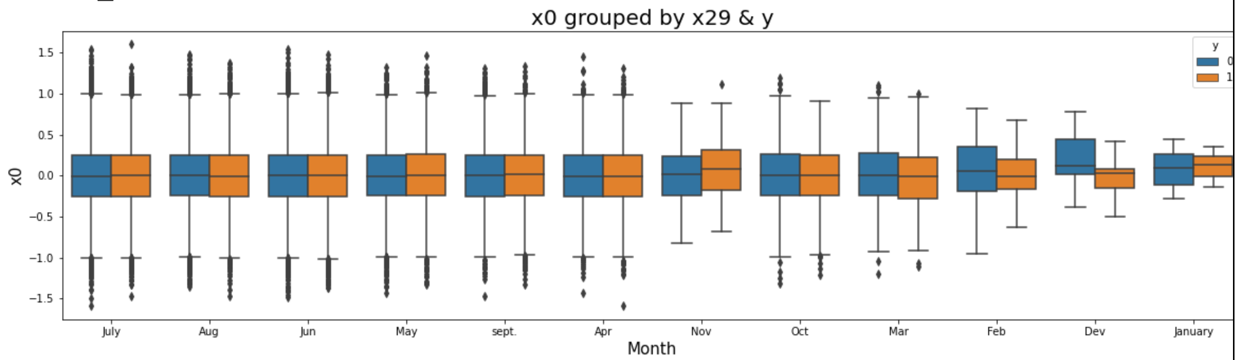
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd50656c890>
```



In [ ]:

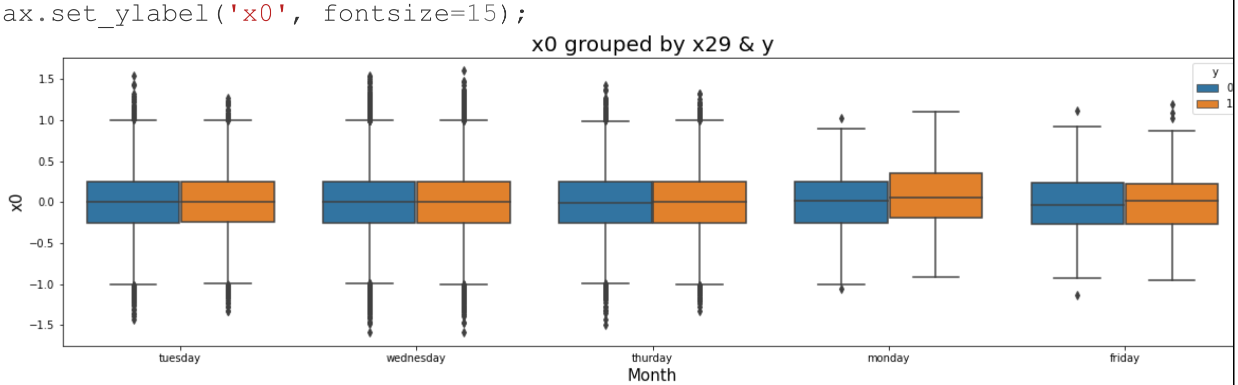
```
#Plotting wages distribution on log scale by position
plt.figure(figsize=(20,5))
ax = sns.boxplot(data=df, y='x0', x='x29', hue='y');
#ax.set_yscale('log');
ax.set_title('x0 grouped by x29 & y', fontsize=20);
```

```
ax.set_xlabel('Month', fontsize=15);
ax.set_ylabel('x0', fontsize=15);
```



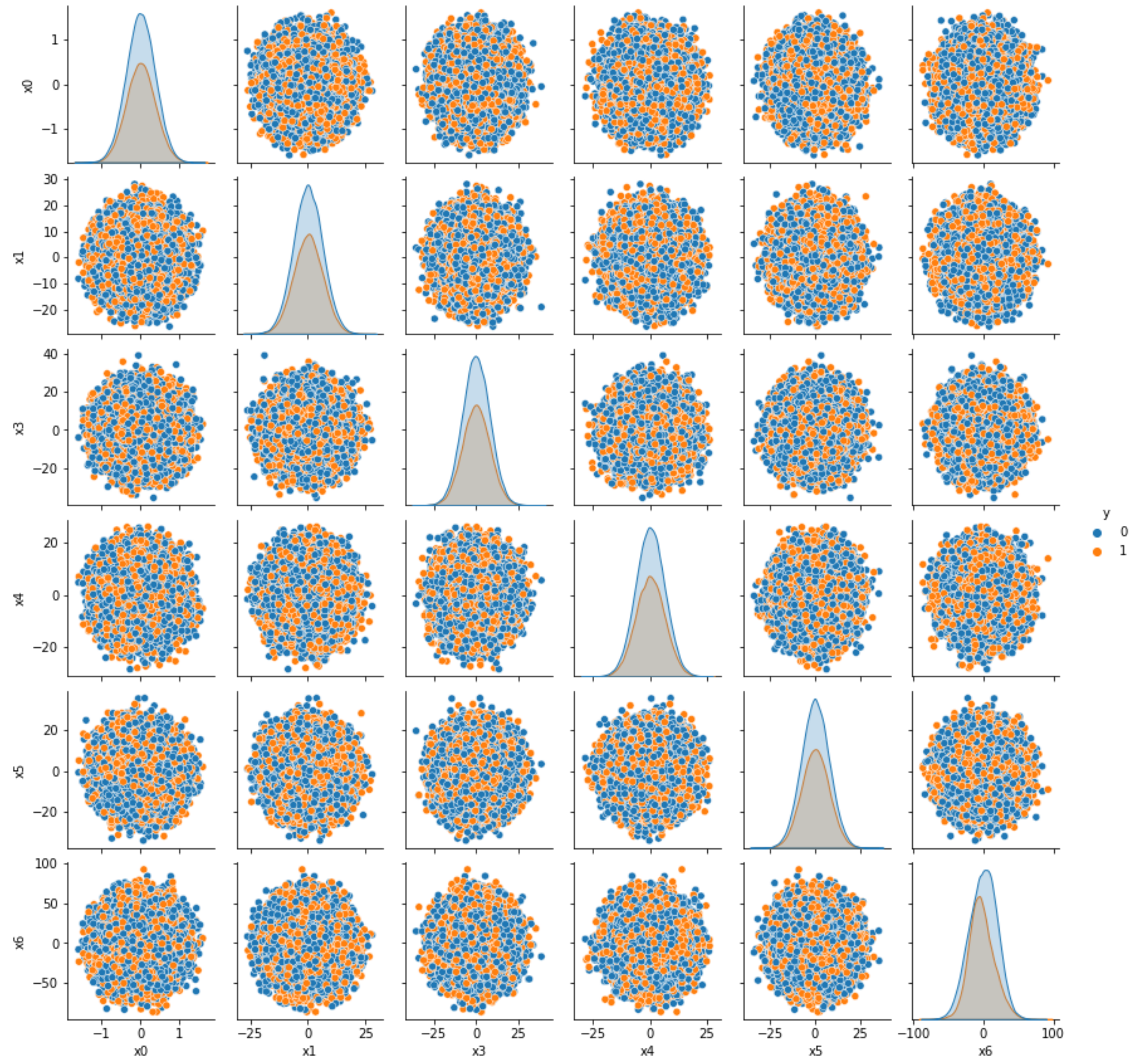
In [ ]:

```
#Plotting wages distribution on log scale by position
plt.figure(figsize=(20,5))
ax = sns.boxplot(data=df, y='x0', x='x30', hue='y');
#ax.set_yscale('log');
ax.set_title('x0 grouped by x29 & y', fontsize=20);
ax.set_xlabel('Month', fontsize=15);
ax.set_ylabel('x0', fontsize=15);
```



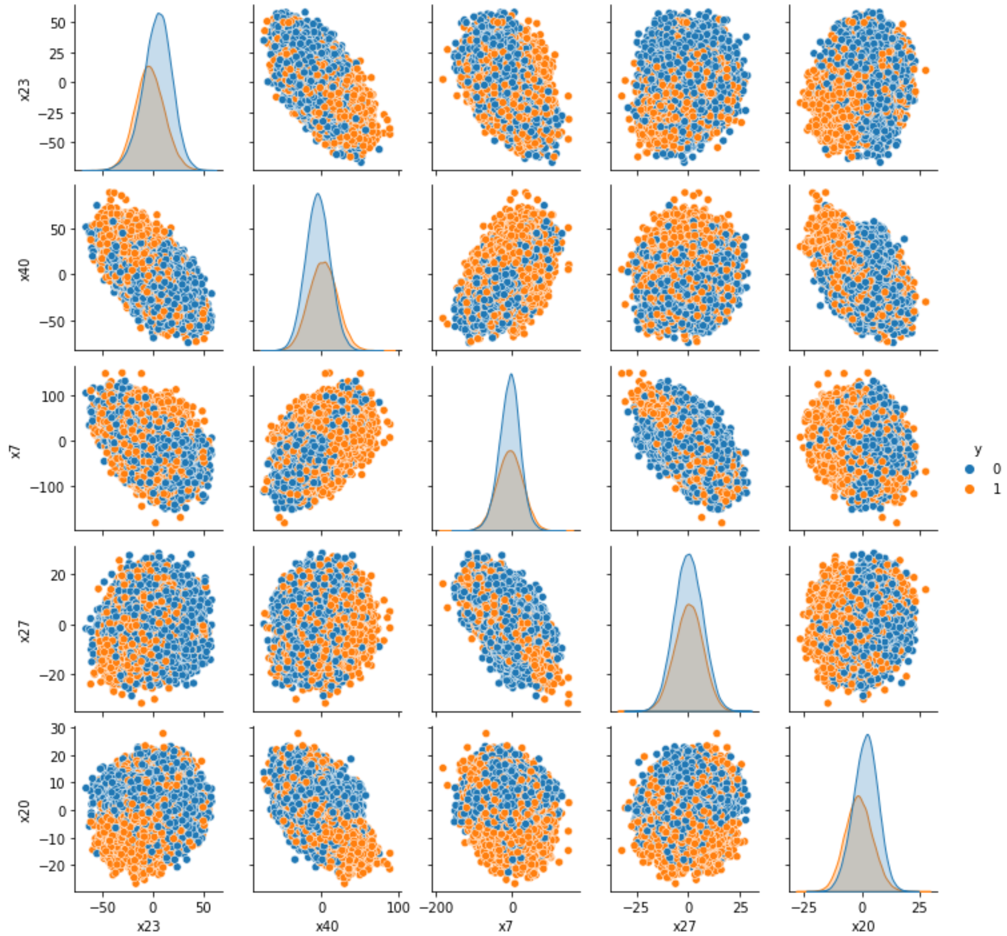
In [ ]:

```
#analyse Technical skills of regular Non GK
l=pd.Series(['x0','x1','x3','x4','x5', 'x6', 'y'])
sns.pairplot(df[l], height=2, hue='y');
```



In [ ]:

```
l=pd.Series(['x23','x40','x7','x27','x20','y'])
sns.pairplot(df[l], height=2, hue='y');
```



In [ ]:

In [ ]:

```
miss_count = df.isnull().sum()
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing, 'count': miss_count})
missing_value_df.sort_values('percent_missing', inplace=True, ascending=False)
missing_value_df.head(50)
```

Out[ ]:

	column_name	percent_missing	count
x23	x23	0.029375	47
x10	x10	0.026875	43

x37	x37	0.014375	23
x39	x39	0.014375	23
x25	x25	0.013750	22
x8	x8	0.013125	21

In [ ]:

In [ ]:

```
def print_highly_correlated(df, features, t=0.8):
    #Method will extractout featuresthat are corelated based on thresh hold
    l = []
    c_df = df[features].corr() # get correlations
    cor_features = np.where(np.abs(c_df) > t) # nparray method
    cor_features = [(c_df.iloc[x,y], x, y) for x, y in zip(*cor_features) if
x != y and x < y]
    #try sorting
    corr_list = sorted(cor_features, key=lambda x: -abs(x[0]))
    if corr_list == []:
        print("Nothing above: ", t)
    else:

        for v, i, j in corr_list:
            cols = df[features].columns
            if c_df.index[i] not in l:
                l.append(c_df.index[i])
            if c_df.index[j] not in l:
                l.append(c_df.index[j])
            print ("%s and %s = %.3f" % (c_df.index[i], c_df.columns[j], v))
    return l

print_highly_correlated(df, df.columns, t=0.80)

#prepare the plot pallete
#cmap = sns.diverging_palette(220, 10, as_cmap=True) # one of the many color
mappings
#sns.set(style="darkgrid") # one of the many styles to plot using
#f, ax = plt.subplots(figsize=(25, 25))
#%time sns.heatmap(df_imputed[print_highly_correlated(df, df.columns, t=0.99
)].corr(), cmap=cmap, fmt=".2f",annot=True);
#f.tight_layout();
```

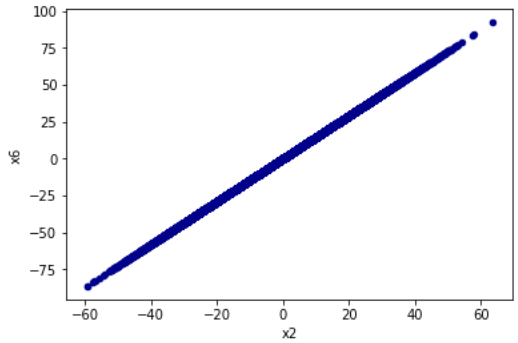
```
x2 and x6 = 1.000
x38 and x41 = 1.000
```

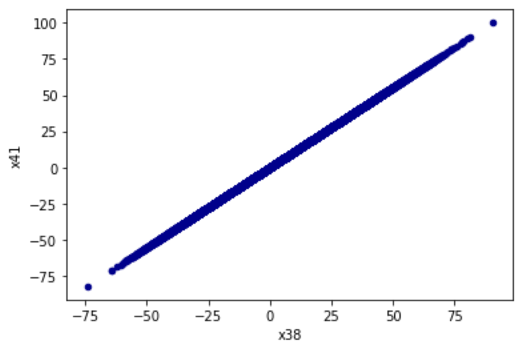
Out[ ]:

```
['x2', 'x6', 'x38', 'x41']
```

In [ ]:

```
_=df.plot.scatter(x='x2', y='x6', c='DarkBlue')
_=df.plot.scatter(x='x38', y='x41', c='DarkBlue')
```





In [ ]:

```
df_imputed = df.drop(['x2','x41'], axis=1)
```

In [ ]:

```
df_imputed['x24'].unique()
#df.plot.bar()
```

Out[ ]:

```
array(['euorpe', 'asia', 'america', nan], dtype=object)
```

In [ ]:

```
df['x29'].value_counts()
```

Out[ ]:

```
July      45569
Jun       41329
Aug       29406
May       21939
sept.     10819
Apr        6761
Oct        2407
Mar        1231
```

```
Nov         337
Feb         140
Dev          23
January       9
Name: x29, dtype: int64
```

In [ ]:

```
#Lets fix some data

#X37 remove leading $
df_imputed['x37'] = df_imputed['x37'].str.lstrip('$')



#x32 remove 10.0%
df_imputed['x32'] = df_imputed['x32'].str.rstrip('%')
df_imputed[['x37','x32']] = df_imputed[['x37','x32']].astype(np.float64)
# replace sept. to sept
df_imputed['x29'] = df_imputed['x29'].str.rstrip('.')

# ??#x29 July, March
#x30 Mon, Tue
#x24 asia europe

#Remove, 88 mutully exclusive rows of Month, day, region
```

In [ ]:

```
#lets remove these rows they are hard to estimate missing values and are ver
y few
df[['x24','x29','x30']].isnull().sum()
```

Out[ ]:

```
x24    28
x29    30
x30    30
dtype: int64
```

In [ ]:

```
df_imputed.dropna(subset=['x24','x29','x30'], inplace=True)
```

In [ ]:

```
df_imputed[['x24','x29','x30']].isnull().sum()
```

Out[ ]:

```
x24    0
x29    0
x30    0
dtype: int64
```

In [ ]:

```
df_imputed.shape
df_imputed.info(verbose=True, null_counts=True)
```

Out[ ]:

```
(159912, 49)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 159912 entries, 0 to 159999
Data columns (total 49 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   x0      159886 non-null  float64
 1   x1      159887 non-null  float64
 2   x3      159875 non-null  float64
 …
 44  x46     159881 non-null  float64
 45  x47     159875 non-null  float64
 46  x48     159880 non-null  float64
 47  x49     159880 non-null  float64
 48  y       159912 non-null  int64
dtypes: float64(45), int64(1), object(3)
memory usage: 61.0+ MB
```

In [ ]:

```
l=pd.Series(['x37','x32','x23','x40','x7','x27','x20','y'])
sns.pairplot(df_imputed[l], height=2, hue='y');
```

In [ ]:

```
#OHE
ohe_list = ['x24','x29','x30']

# get oheed columns and add to imputed and drop original columns
pd_ohe = pd.get_dummies(df_imputed[ohe_list], prefix=ohe_list,drop_first=Tru
e)
#lets seperate response variable
#df_target = df_imputed.iloc[:,-1:]
#df_imputed.drop('y', axis=1, inplace = True)
df_imputed = pd.concat([ pd_ohe, df_imputed], axis=1)
#df_imputed = pd.concat([df_imputed, pd_ohe], axis=1)
df_imputed.drop(ohe_list, axis=1, inplace = True)
```

In [ ]:

```
df_imputed.shape
df_imputed.head()
```

Out[ ]:

```
(159912, 63)
#imput missing data
#MICE imputer
%%time
imp = sm.imputation.mice.MICEData(df_imputed)

def make_fml(col_list):
    out = ''
    for i  in col_list:
        out = out + i + " + "
    return out[:-3]
t = make_fml(df_imputed.columns[~df_imputed.columns.isin(['y'])].tolist())

fml = 'y ~ ' + t
print(fml)
y ~ x24_asia + x24_euorpe + x29_Aug + x29_Dev + x29_Feb + x29_January + x29_
July + x29_Jun + x29_Mar + x29_May + x29_Nov + x29_Oct + x29_sept + x30_mond
ay + x30_thurday + x30_tuesday + x30_wednesday + x0 + x1 + x3 + x4 + x5 + x6
+ x7 + x8 + x9 + x10 + x11 + x12 + x13 + x14 + x15 + x16 + x17 + x18 + x19 +
x20 + x21 + x22 + x23 + x25 + x26 + x27 + x28 + x31 + x32 + x33 + x34 + x35
+ x36 + x37 + x38 + x39 + x40 + x42 + x43 + x44 + x45 + x46 + x47 + x48 + x4
9
CPU times: user 243 ms, sys: 12.5 ms, total: 256 ms
Wall time: 237 ms
```

In [ ]:

```
mice = sm.imputation.mice.MICE(fml, sm.regression.linear_model.OLS, imp)
results = mice.fit(1, 2)

print(results.summary())
                            Results: MICE
=====================================================================
Method:                    MICE        Sample size:            159912
Model:                     OLS         Scale                   0.20
Dependent variable:        y           Num. imputations        2
---------------------------------------------------------------------
               Coef.  Std.Err.    t     P>|t|   [0.025    0.975]  FMI
---------------------------------------------------------------------
Intercept      9.7024  12.4594  0.7787 0.4361 -14.7175 34.1223 0.0007
x24_asia       0.0339   0.0077  4.4218 0.0000   0.0189  0.0489 0.0005
x24_euorpe     0.0380   0.0099  3.8388 0.0001   0.0186  0.0574 0.0002
```

```
x29_Aug       -0.0040   0.0060 -0.6639 0.5068  -0.0157   0.0077 0.0000
x29_Dev        0.0488   0.0924  0.5284 0.5972  -0.1323   0.2300 0.0000
x29_Feb       -0.0255   0.0378 -0.6741 0.5003  -0.0995   0.0486 0.0000
…
x47            0.0001   0.0002  0.2463 0.8054  -0.0004   0.0005 0.0012
x48            7.6361  10.1405  0.7530 0.4514 -12.2388  27.5110 0.0286
x49           -2.3550   5.6532 -0.4166 0.6770 -13.4351   8.7250 0.5947
=====================================================================
```

In [ ]:

```
#mice.data.data[:,df_imputed[df_imputed['Attr37'].isnull()].index.tolist()]
df_imputed = imp.data
df_imputed.info(verbose=True, null_counts=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159912 entries, 0 to 159911
Data columns (total 63 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   x24_asia       159912 non-null  uint8
 1   x24_euorpe     159912 non-null  uint8
 2   x29_Aug        159912 non-null  uint8
 3   x29_Dev        159912 non-null  uint8
 4   x29_Feb        159912 non-null  uint8
 5   x29_January    159912 non-null  uint8
 6   x29_July       159912 non-null  uint8
 7   x29_Jun        159912 non-null  uint8
 …
 60  x48            159912 non-null  float64
 61  x49            159912 non-null  float64
 62  y              159912 non-null  int64
dtypes: float64(45), int64(1), uint8(17)
memory usage: 58.7 MB
```

In [ ]:

```
#scale
```

In [ ]:

```
df_imputed.info(verbose=True, null_counts=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159912 entries, 0 to 159911
Data columns (total 63 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   x24_asia       159912 non-null  uint8
 1   x24_euorpe     159912 non-null  uint8
 2   x29_Aug        159912 non-null  uint8
```

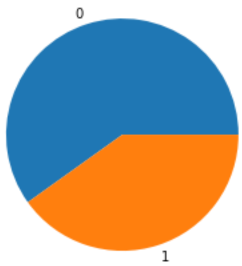
```
 3   x29_Dev         159912 non-null  uint8
 …
 59  x47             159912 non-null  float64
 60  x48             159912 non-null  float64
 61  x49             159912 non-null  float64
 62  y               159912 non-null  int64
dtypes: float64(45), int64(1), uint8(17)
memory usage: 58.7 MB
```

In [ ]:

```
#Check class distribution
%matplotlib inline

# Adapted from:
# https://www.featureranking.com/tutorials/machine-learning-tutorials/inform
ation-gain-computation/
def gini_index(y):
    probs = pd.value_counts(y,normalize=True)
    return 1 - np.sum(np.square(probs))

def plot_class_dist(y):
    class_ct = len(np.unique(y['y']))
    vc = pd.value_counts(y['y'])
    print('Total Records', len(y['y']))
    print('Total Classes:', class_ct)
    print('Class Gini Index', gini_index(y['y']))
    print('Smallest Class Id:',vc.idxmin(),'Records:',vc.min())
    print('Largest Class Id:',vc.idxmax(),'Records:',vc.max())

    position_counts = pd.DataFrame(y['y'].value_counts())
    position_counts['Percentage'] = position_counts['y']/position_counts.sum
()[0]
    print(position_counts)
    plt.figure(figsize=(4,4))
    plt.pie(position_counts['Percentage'],labels = ['0', '1']);


plot_class_dist(df_imputed.iloc[:,-1:])
```

```
Total Records 159912
Total Classes: 2
Class Gini Index 0.4804828175501279
Smallest Class Id: 1 Records: 64159
Largest Class Id: 0 Records: 95753
       y  Percentage
0  95753    0.598786
```

```
1  64159    0.401214
```



In [ ]:

```
#pickle.dump(df_imputed, open('imputed_data.sav', 'wb'))
```

In [4]:

```
with open('./drive/MyDrive/data/imputed_data.sav', 'rb') as f:
  df_imputed = pickle.load(f)
```

In [5]:

```
X = df_imputed.iloc[:,:-1].values
X.shape
y = df_imputed['y'].values
y.shape

#Normalize data
##Scale the transformed data
scl_obj = MinMaxScaler(feature_range=[0, 1]) #StandardScaler()
scl_obj.fit(X)
X_scaled = scl_obj.transform(X)
#QuantileTransformer(output_distribution='uniform').fit_transform(X))
X_scaled.shape
#X_scaled
```

Out[5]:

```
(159912, 62)
```

Out[5]:

```
(159912,)
```

Out[5]:

```
MinMaxScaler(feature_range=[0, 1])
```

Out[5]:

```
(159912, 62)
```

In [6]:

```
# #train/holdout 90/10 stratified
stt = StratifiedShuffleSplit(n_splits=1, test_size=0.1, random_state=111)
train_index_clf, test_index_clf = next(stt.split(X_scaled, y))
X_train = X[train_index_clf]
y_train = y[train_index_clf].ravel()
X_test = X[test_index_clf]
y_test = y[test_index_clf].ravel()
```

```
X_train.shape
y_train.shape
X_test.shape
y_test.shape
```

Out[6]:

```
(143920, 62)
```

Out[6]:

```
(143920,)
```

Out[6]:

```
(15992, 62)
```

Out[6]:

```
(15992,)
```

In [7]:

```
# #train_nn/test_nn 80/20 of X_train stratified
stt = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=111)
train_index_clf, test_index_clf = next(stt.split(X_train, y_train))
X_train_nn = X_train[train_index_clf]
y_train_nn = y_train[train_index_clf].ravel()
X_test_nn = X_train[test_index_clf]
y_test_nn = y_train[test_index_clf].ravel()
X_train_nn.shape
y_train_nn.shape
X_test_nn.shape
y_test_nn.shape
```

Out[7]:

```
(115136, 62)
```

Out[7]:

```
(115136,)
```

Out[7]:

```
(28784, 62)
```

Out[7]:

```
(28784,)
```

In [8]:

```
import warnings
warnings.filterwarnings('ignore')
from yellowbrick.classifier import ROCAUC
def plot_roc(est, X_test, y_test, X_train, y_train):
    visualizer = ROCAUC(est, binary=True ,classes=["No", "Bankrupt"])
    visualizer.fit(X_train, y_train)        # Fit the training data to the v
isualizer
    visualizer.score(X_test, y_test)        # Evaluate the model on the test
data
    visualizer.show()
```

```
def evaluate_xg_model_performance(model_name, params, clf, X_train, y_train,
X_test, y_test, nCV = 5, n_jobs = 10):
    fit_params={"early_stopping_rounds":5,
            "eval_metric" : "logloss",
            "eval_set" : [[X_test, y_test]]}
    # We prepare the grid search object to be passed to GSCV
    sss = StratifiedShuffleSplit(n_splits=nCV, test_size=0.2, random_state=4
5)
    grid = gridcv(clf, params, cv=sss, verbose=1, scoring='roc_auc',n_jobs =
-1, refit=True )
    grid.fit(X_train, y_train, **fit_params)
    model_stat =  pd.DataFrame()
    model_stat['model_name'] =[str(model_name)]

    res = grid.cv_results_
    #print(res)
    # Lets store the scores for t-test validation of models
    #cvscore = cross_val_score(grid.best_estimator_, X_train, y_train, scori
ng='f1_weighted', cv=nCV,n_jobs= n_jobs)

    #model_stat['scores'] = [cvscore]
    #grid.cv_results_.keys()
    #res.keys()
    #res['params']
    grid_scr = pd.DataFrame()
    grid_scr['params'] = res['params']
    grid_scr['mean_test_score'] = res['mean_test_score']
    grid_scr = pd.DataFrame(grid_scr)
    #print(grid_scr)

    grid_scr.plot.bar(color='grey',figsize=(10,6))
    plt.ylabel('Accuracy')
    plt.xlabel('Params')
    plt.grid(color='blue', linestyle='--', linewidth=0.5)
    plt.ylim(0.93,.97)
    plt.show()
    print("Best parameters set found on development set:")
    print()
    print(grid.best_params_)
    #model_stat['score'] = [grid.best_score_]
    print()
    print("Grid scores on development set:")
    print()
```

```
    means = res['mean_test_score']

    stds = res['std_test_score']
    for mean, std, params in zip(means, stds, res['params']):
        print("%0.5f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()
    #plot_roc(grid.best_estimator_, X_test, y_test, X_train, y_train)
    #plt.show()
    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the test set.")
    print()
    #build CM using test/Train
    y_true, y_pred = y_test, grid.best_estimator_.predict(X_test)
    y_predprob = grid.best_estimator_.predict_proba(X_test)


    #y_pred
    print(classification_report(y_true, y_pred, target_names=['0','1']))
    s = classification_report(y_true, y_pred, target_names=['0','1'])
    model_stat['CM'] = s
    plot_confusion_matrix(grid, X_test,y_test,cmap=plt.cm.Blues,values_forma
t='d',display_labels = ['0','1'])
    model_stat['time_refit'] = [grid.refit_time_]
    model_stat['model_param'] = [str(grid.best_params_)]
    model_stat['weighted_f1_score']=round(f1_score(y_true, y_pred, average='
weighted'),2)
    #model_stat['accuracy']=accuracy_score(y_true, y_pred)
    plt.grid(b=None);
    plt.show()
    print()
#     for input, prediction, prob in zip(y_true, y_pred,  y_predprob):
#         if prediction != input:
#             print(input, 'has been classified as ', prediction, 'and shoul
d be ', input, ' proabability:', prob)

    return model_stat, grid.best_estimator_

def evaluate_clf_model_performance(model_name, params, clf, X_train, y_train
, X_test, y_test, nCV = 5, n_jobs = 10):

    # We prepare the grid search object to be passed to GSCV
```

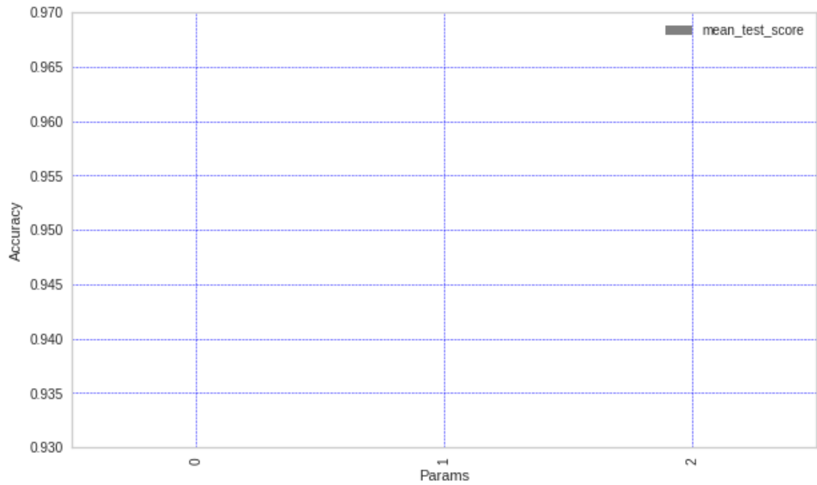
```
    sss = StratifiedShuffleSplit(n_splits=nCV, test_size=0.2, random_state=4
5)
    grid = gridcv(clf, params, cv=sss,scoring='roc_auc',n_jobs =-1, refit=Tr
ue )
    grid.fit(X_train, y_train)
    model_stat =  pd.DataFrame()
    model_stat['model_name'] =[str(model_name)]

    res = grid.cv_results_
    #print(res)
    # Lets store the scores for t-test validation of models
    #cvscore = cross_val_score(grid.best_estimator_, X_train, y_train, scori
ng='f1_weighted', cv=nCV,n_jobs= n_jobs)

    #model_stat['scores'] = [cvscore]
    #grid.cv_results_.keys()
    #res.keys()
    #res['params']
    grid_scr = pd.DataFrame()
    grid_scr['params'] = res['params']
    grid_scr['mean_test_score'] = res['mean_test_score']
    grid_scr = pd.DataFrame(grid_scr)
    #print(grid_scr)

    grid_scr.plot.bar(color='grey',figsize=(10,6))
    plt.ylabel('Accuracy')
    plt.xlabel('Params')
    plt.grid(color='blue', linestyle='--', linewidth=0.5)
    plt.ylim(0.93,.97)
    plt.show()
    print("Best parameters set found on development set:")
    print()
    print(grid.best_params_)
    #model_stat['score'] = [grid.best_score_]
    print()
    print("Grid scores on development set:")
    print()
    means = res['mean_test_score']

    stds = res['std_test_score']
    for mean, std, params in zip(means, stds, res['params']):
        print("%0.5f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()
```

```
    #plot_roc(grid.best_estimator_, X_test, y_test, X_train, y_train)
    #plt.show()
    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the test set.")
    print()
    #build CM using test/Train
    y_true, y_pred = y_test, grid.best_estimator_.predict(X_test)
    y_predprob = grid.best_estimator_.predict_proba(X_test)


    #y_pred
    print(classification_report(y_true, y_pred, target_names=['0','1']))
    s = classification_report(y_true, y_pred, target_names=['0','1'])
    model_stat['CM'] = s
    plot_confusion_matrix(grid, X_test,y_test,cmap=plt.cm.Blues,values_forma
t='d',display_labels = ['0','1'])
    model_stat['time_refit'] = [grid.refit_time_]
    model_stat['model_param'] = [str(grid.best_params_)]
    model_stat['weighted_f1_score']=round(f1_score(y_true, y_pred, average='
weighted'),2)
    #model_stat['accuracy']=accuracy_score(y_true, y_pred)
    plt.grid(b=None);
    plt.show()
    print()
#     for input, prediction, prob in zip(y_true, y_pred,  y_predprob):
#         if prediction != input:
#             print(input, 'has been classified as ', prediction, 'and shoul
d be ', input, ' proabability:', prob)

    return model_stat, grid.best_estimator_

numCVs=5
```

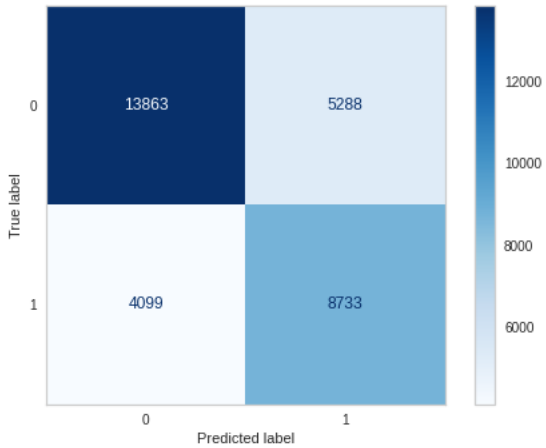
In [ ]:

```
#Logistic regression
params = [{
            'penalty': ['l2'],
            'C':[ .08, .1, .12],
            'class_weight': ['balanced'],
            'solver' : [ 'saga'] # 'newton-cg', 'lbfgs', 'liblinear', 'sag',
'saga'
         }]
logr = LogisticRegression(random_state = 45,max_iter = 5000)
```

```
%time m, mdl = evaluate_clf_model_performance('LogisticRegn', params, logr,
X_scaled, y, numCVs)
```



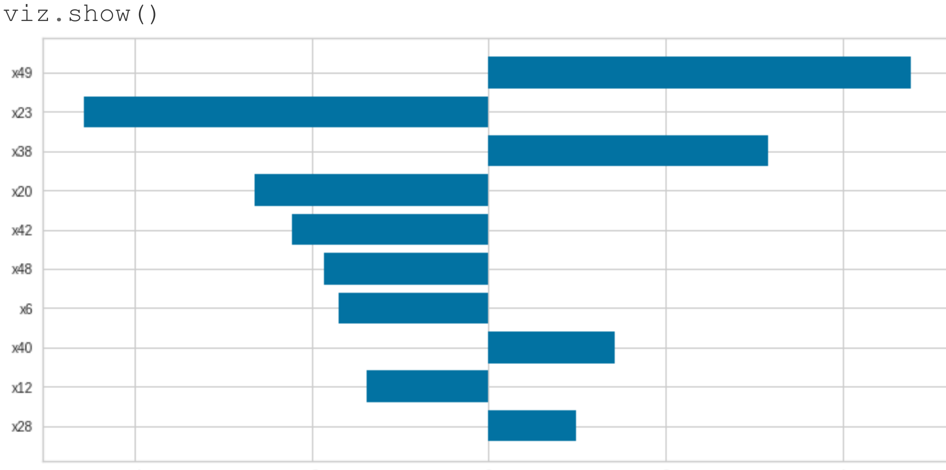

```
Best parameters set found on development set:

{'C': 0.12, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'saga'}

Grid scores on development set:

0.76051 (+/-0.004) for {'C': 0.08, 'class_weight': 'balanced', 'penalty': 'l
2', 'solver': 'saga'}
0.76058 (+/-0.004) for {'C': 0.1, 'class_weight': 'balanced', 'penalty': 'l2
', 'solver': 'saga'}
0.76063 (+/-0.004) for {'C': 0.12, 'class_weight': 'balanced', 'penalty': 'l
2', 'solver': 'saga'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the test set.

              precision    recall  f1-score   support

           0       0.77      0.72      0.75     19151
           1       0.62      0.68      0.65     12832

    accuracy                           0.71     31983
   macro avg       0.70      0.70      0.70     31983
weighted avg       0.71      0.71      0.71     31983
```




```
CPU times: user 3.93 s, sys: 1.06 s, total: 4.99 s
Wall time: 17.3 s
```

In [ ]:

```
#mdl.coef_
#fig, ax = plt.subplots()
#fig.size(10,10)
from yellowbrick.model_selection import FeatureImportances
import matplotlib
matplotlib.rcParams['legend.fontsize'] = 10
labels = df_imputed.columns[:-1]

viz = FeatureImportances(mdl, stack=True, labels=labels, relative=False, top
n = 10, size=(880, 420))
_ = viz.fit(X_scaled, y)
axes = plt.gca()
#axes.set_title('Model Scores For Class <30', fontsize=20)
axes.yaxis.label.set_size(18)
viz.ax.xaxis.label.set_size(14)

viz.show()
```



In [ ]:

```
df_imputed.columns[:-1]
```

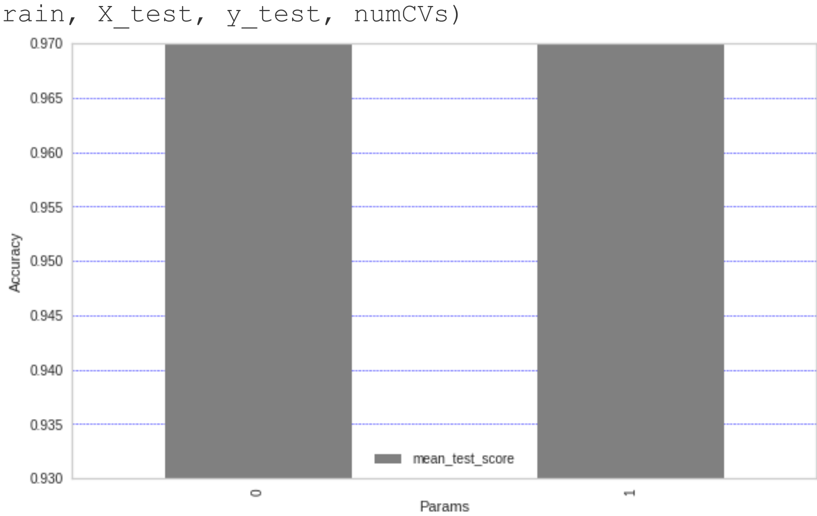
Out[ ]:

```
Index(['x24_asia', 'x24_euorpe', 'x29_Aug', 'x29_Dev', 'x29_Feb',
       'x29_January', 'x29_July', 'x29_Jun', 'x29_Mar', 'x29_May', 'x29_Nov'
,
       'x29_Oct', 'x29_sept', 'x30_monday', 'x30_thurday', 'x30_tuesday',
       'x30_wednesday', 'x0', 'x1', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9'
,
       'x10', 'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19',
       'x20', 'x21', 'x22', 'x23', 'x25', 'x26', 'x27', 'x28', 'x31', 'x32',
       'x33', 'x34', 'x35', 'x36', 'x37', 'x38', 'x39', 'x40', 'x42', 'x43',
       'x44', 'x45', 'x46', 'x47', 'x48', 'x49'],
      dtype='object')
```

In [ ]:

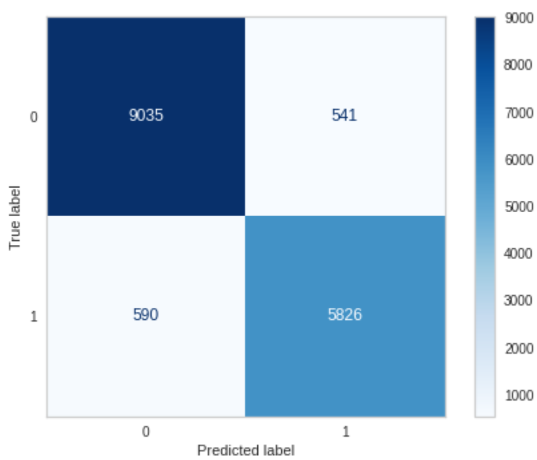
In [ ]:

```
#model1 RF
from sklearn.ensemble import RandomForestClassifier
n_estimators = [250]
params = [{
    'n_estimators' : n_estimators,
    'min_samples_leaf': [10,5],
    'max_features': [25],
    'random_state': [45],
    'class_weight': ['balanced']}]
RF = RandomForestClassifier()

%time m, mdl = evaluate_clf_model_performance('RF', params, RF, X_train, y_t
rain, X_test, y_test, numCVs)
```




```
Best parameters set found on development set:
```

```
{'class_weight': 'balanced', 'max_features': 25, 'min_samples_leaf': 5, 'n_e
stimators': 250, 'random_state': 45}

Grid scores on development set:

0.97673 (+/-0.001) for {'class_weight': 'balanced', 'max_features': 25, 'min
_samples_leaf': 10, 'n_estimators': 250, 'random_state': 45}
0.97804 (+/-0.001) for {'class_weight': 'balanced', 'max_features': 25, 'min
_samples_leaf': 5, 'n_estimators': 250, 'random_state': 45}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the test set.

              precision    recall  f1-score   support

           0       0.94      0.94      0.94      9576
           1       0.92      0.91      0.91      6416

    accuracy                           0.93     15992
   macro avg       0.93      0.93      0.93     15992
weighted avg       0.93      0.93      0.93     15992
```

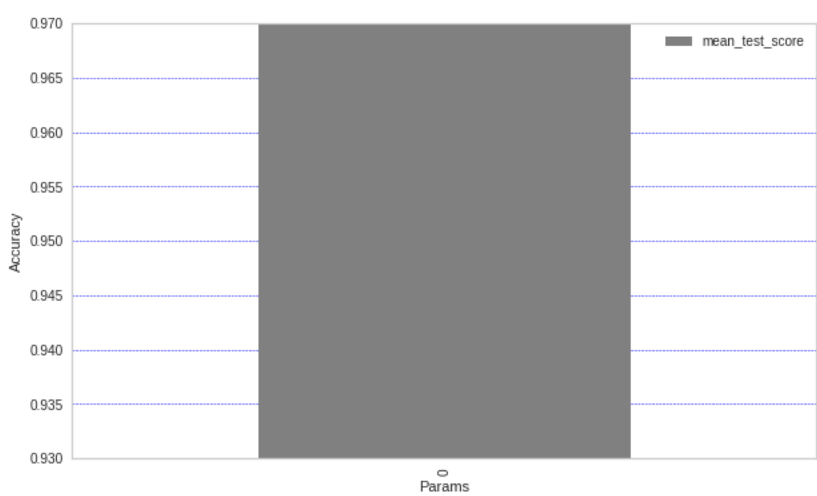



```
CPU times: user 17min 23s, sys: 2.45 s, total: 17min 26s
Wall time: 1h 4min 34s
```

In [9]:

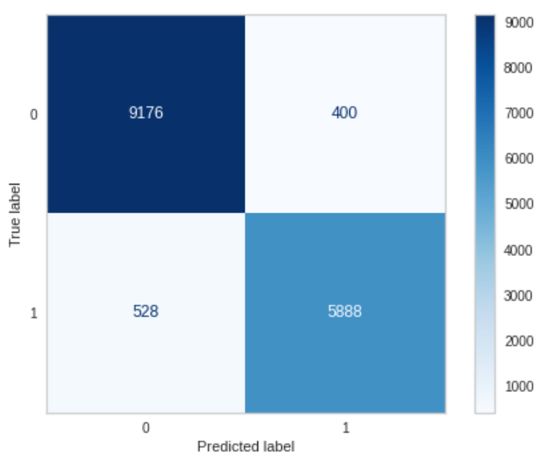
```
#https://xgboost.readthedocs.io/en/stable/python/python_api.html?highlight=x
gbclassifier#xgboost.XGBClassifier
from xgboost import XGBClassifier
n_estimators = [ 1000]
```

```
params = [{
    'n_estimators' : n_estimators, #number of boosting rounds
    'learning_rate' : [.01], #eta
    'objective' : ['binary:logistic'],
    'gamma' : [4], #early stopping/min_split_loss
    'max_depth' : [12], #max depth to traverse
    'colsample_bytree' : [ .7],
    'num_classes' : [2],
    'eval_metric':["logloss"],
    'booster': ['gbtree'], #['gbtree','gblinear'],
    'random_state': [45], 'verbose_eval':[True]
         }]
clf = XGBClassifier(random_state=45)
%time m, mdl1 = evaluate_xg_model_performance('XGBClassifier', params, clf,
X_train, y_train, X_test, y_test, numCVs)
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[0]     validation_0-logloss:0.688
Will train until validation_0-logloss hasn't improved in 5 rounds.
[1]     validation_0-logloss:0.683029
[2]     validation_0-logloss:0.677104
…
[980]   validation_0-logloss:0.16714
[981]   validation_0-logloss:0.167139
Stopping. Best iteration:
[976]   validation_0-logloss:0.167137
```




```
Best parameters set found on development set:

{'booster': 'gbtree', 'colsample_bytree': 0.7, 'eval_metric': 'logloss', 'ga
mma': 4, 'learning_rate': 0.01, 'max_depth': 12, 'n_estimators': 1000, 'num_
```

```
classes': 2, 'objective': 'binary:logistic', 'random_state': 45, 'verbose_ev
al': True}

Grid scores on development set:

0.98383 (+/-0.001) for {'booster': 'gbtree', 'colsample_bytree': 0.7, 'eval_
metric': 'logloss', 'gamma': 4, 'learning_rate': 0.01, 'max_depth': 12, 'n_e
stimators': 1000, 'num_classes': 2, 'objective': 'binary:logistic', 'random_
state': 45, 'verbose_eval': True}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the test set.

              precision    recall  f1-score   support

           0       0.95      0.96      0.95      9576
           1       0.94      0.92      0.93      6416

    accuracy                           0.94     15992
   macro avg       0.94      0.94      0.94     15992
weighted avg       0.94      0.94      0.94     15992
```




```
CPU times: user 29min 17s, sys: 5.8 s, total: 29min 23s
Wall time: 1h 31min 17s
```

In [11]:

```
with open('./drive/MyDrive/data/xgb2_mdl.sav', 'wb') as f:
    pickle.dump(mdl1, f)
```

In [ ]:

```
with open('./drive/MyDrive/data/rf1_mdl.sav', 'wb') as f:
    pickle.dump(mdl, f)
```

```
with open('./drive/MyDrive/data/xgb1_mdl.sav', 'wb') as f:
    pickle.dump(mdl1, f)
```

In [ ]:

```
def FindLayerNodesLinear(n_layers, first_layer_nodes, last_layer_nodes):
    layers = []

    nodes_increment = (last_layer_nodes - first_layer_nodes)/ (n_layers-1)
    nodes = first_layer_nodes
    for i in range(1, n_layers+1):
        layers.append(math.ceil(nodes))
        nodes = nodes + nodes_increment

    return layers
```

In [51]:

```
from tensorflow.keras.callbacks import EarlyStopping
model_clf_stats = pd.DataFrame()

def createmodel(n_layers, first_layer_nodes, last_layer_nodes, activation_fu
nc, loss_func):
    model = Sequential()
    n_nodes = FindLayerNodesLinear(n_layers, first_layer_nodes, last_layer_n
odes)
    for i in range(1, n_layers):

        if i==1:
            print("building node:",i)
            model.add(Dense(first_layer_nodes, input_dim=X_train.shape[1], a
ctivation=activation_func))
        else:
            print("building node:",i)
            model.add(Dense(n_nodes[i-1], activation=activation_func))

    #Finally, the output layer should have a single node in binary classific
ation
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss=loss_func, metrics = ["accuracy"])
#note: metrics could also be 'mse'

    return model
```

In [ ]:

```
from statistics import mean
def test_model(layers, start, end, activation, batch, X_train, y_train, X_te
st, y_test, ver=1):
  #relu, l=5, nodes=600, e_nodes=8, e=500, b=20000
```

```
  print("*************Execution started for**********************")
  print("Activation:",activation," layers:", layers, " nodes:", start," batc
h:", batch)
  safety = EarlyStopping(monitor='val_loss', patience=50)
  seed = 45 #88.27
  m = createmodel(n_layers=layers, first_layer_nodes=start, last_layer_nodes
=end,
                  activation_func=activation, loss_func=tf.keras.losses.Bina
ryCrossentropy()) #tanh
  hist = m.fit(X_train, y_train, epochs=800, batch_size=batch,
          validation_data=(X_test, y_test), callbacks=[safety], verbose=ver)
# add validation left out here

  best_score = max(hist.history['accuracy'])
  print("Best score: ",best_score)
  model_stat =  pd.DataFrame()
  model_stat['Max Accuracy'] = [best_score]
  model_stat['Avg Accuracy'] = [mean(hist.history['accuracy'])]
  model_stat['Model'] = ["Activation:" + activation + " layers:" + str(layer
s) + " nodes:" + str(start) + " batch:" + str(batch)]
  m.summary()
  tf.keras.backend.clear_session()
  del m
  print("*************Execution ended**********************")
  print("*****************************************************\n\n")
  return model_stat
```

In [52]:

```
#small model
p = test_model(3, 64, 15, 'relu', 10000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)

p = test_model(3, 64, 15, 'relu', 25000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)

#medium
p = test_model(4, 128, 15, 'relu', 10000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)

p = test_model(4, 128, 15, 'relu', 25000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)
```

```
p = test_model(4, 256, 15, 'relu', 10000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)

#large
p = test_model(5, 512, 15, 'relu', 10000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)

p = test_model(5, 512, 15, 'relu', 25000, X_train_nn, y_train_nn, X_test_nn,
y_test_nn)
model_clf_stats = model_clf_stats.append(p)


model_clf_stats
Streaming output truncated to the last 5000 lines.
12/12 [==============================] - 0s 6ms/step - loss: 0.1844 - accura
cy: 0.9299 - val_loss: 0.2220 - val_accuracy: 0.9148
…
Epoch 799/800
12/12 [==============================] - 0s 6ms/step - loss: 0.1414 - accura
cy: 0.9493 - val_loss: 0.1881 - val_accuracy: 0.9332
Epoch 800/800
12/12 [==============================] - 0s 5ms/step - loss: 0.1406 - accura
cy: 0.9490 - val_loss: 0.1848 - val_accuracy: 0.9353
Best score:  0.9495726823806763
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_27 (Dense)            (None, 64)                4032

 dense_28 (Dense)            (None, 40)                2600

 dense_29 (Dense)            (None, 1)                 41

=================================================================
Total params: 6,673
Trainable params: 6,673
Non-trainable params: 0
_________________________________________________________________
*************Execution ended**********************
*****************************************************
```

```
**************Execution started for***********************
Activation: relu  layers: 3  nodes: 64  batch: 25000
building node: 1
building node: 2
Epoch 1/800
5/5 [==============================] - 1s 42ms/step - loss: 2.9663 - accurac
y: 0.5084 - val_loss: 2.1329 - val_accuracy: 0.5412
Epoch 2/800
5/5 [==============================] - 0s 13ms/step - loss: 1.7995 - accurac
y: 0.5261 - val_loss: 1.5295 - val_accuracy: 0.55…
Epoch 799/800
5/5 [==============================] - 0s 11ms/step - loss: 0.1690 - accurac
y: 0.9376 - val_loss: 0.1988 - val_accuracy: 0.9261
Epoch 800/800
5/5 [==============================] - 0s 12ms/step - loss: 0.1671 - accurac
y: 0.9381 - val_loss: 0.1958 - val_accuracy: 0.9277
Best score:  0.941608190536499
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                4032

 dense_1 (Dense)             (None, 40)                2600

 dense_2 (Dense)             (None, 1)                 41

=================================================================
Total params: 6,673
Trainable params: 6,673
Non-trainable params: 0
_________________________________________________________________
**************Execution ended***********************
****************************************************



**************Execution started for***********************
Activation: relu  layers: 4  nodes: 128  batch: 10000
building node: 1
building node: 2
building node: 3
Epoch 1/800
```

```
12/12 [==============================] - 1s 17ms/step - loss: 12.1438 - accu
racy: 0.5234 - val_loss: 9.3886 - val_accuracy: 0.5121
Epoch 2/800
12/12 [==============================] - 0s 6ms/step - loss: 4.2087 - accura
cy: 0.5447 - val_loss: 3.1782 - val_accuracy: 0.5…
Epoch 252/800
12/12 [==============================] - 0s 6ms/step - loss: 0.1512 - accura
cy: 0.9447 - val_loss: 0.1992 - val_accuracy: 0.9312
Best score:  0.9457337260246277
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               8064

 dense_1 (Dense)             (None, 91)                11739

 dense_2 (Dense)             (None, 53)                4876

 dense_3 (Dense)             (None, 1)                 54

=================================================================
Total params: 24,733
Trainable params: 24,733
Non-trainable params: 0
_________________________________________________________________
**************Execution ended**********************
*****************************************************


**************Execution started for**********************
Activation: relu  layers: 4  nodes: 128  batch: 25000
building node: 1
building node: 2
building node: 3
Epoch 1/800
5/5 [==============================] - 1s 42ms/step - loss: 14.9136 - accura
cy: 0.5247 - val_loss: 8.9409 - val_accuracy: 0.4893
Epoch 2/800
5/5 [==============================] - 0s 13ms/step - loss: 5.9309 - accurac
y: 0.4904 - val_loss: 5.1469 - val_accuracy: 0.5296
…
Epoch 392/800
```

```
5/5 [==============================] - 0s 12ms/step - loss: 0.1869 - accurac
y: 0.9283 - val_loss: 0.2189 - val_accuracy: 0.9195
Epoch 393/800
5/5 [==============================] - 0s 14ms/step - loss: 0.1865 - accurac
y: 0.9285 - val_loss: 0.2236 - val_accuracy: 0.9164
Best score:  0.9491991996765137
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               8064

 dense_1 (Dense)             (None, 91)                11739

 dense_2 (Dense)             (None, 53)                4876

 dense_3 (Dense)             (None, 1)                 54

=================================================================
Total params: 24,733
Trainable params: 24,733
Non-trainable params: 0
_________________________________________________________________
**************Execution ended***********************
*****************************************************



**************Execution started for***********************
Activation: relu  layers: 4  nodes: 256  batch: 10000
building node: 1
building node: 2
building node: 3
Epoch 1/800
12/12 [==============================] - 1s 19ms/step - loss: 11.6042 - accu
racy: 0.5313 - val_loss: 4.7630 - val_accuracy: 0.4756
Epoch 2/800
12/12 [==============================] - 0s 8ms/step - loss: 2.2674 - accura
cy: 0.6068 - val_loss: 0.9994 - val_accuracy: 0.7…
Epoch 206/800
12/12 [==============================] - 0s 8ms/step - loss: 0.1024 - accura
cy: 0.9638 - val_loss: 0.2058 - val_accuracy: 0.9372
Epoch 207/800
12/12 [==============================] - 0s 8ms/step - loss: 0.0971 - accura
cy: 0.9664 - val_loss: 0.2147 - val_accuracy: 0.9352
```

```
Epoch 208/800
12/12 [==============================] - 0s 8ms/step - loss: 0.0997 - accura
cy: 0.9650 - val_loss: 0.2246 - val_accuracy: 0.9315
Epoch 209/800
12/12 [==============================] - 0s 8ms/step - loss: 0.1024 - accura
cy: 0.9645 - val_loss: 0.2192 - val_accuracy: 0.9337
Best score:  0.9664483666419983
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               16128

 dense_1 (Dense)             (None, 176)               45232

 dense_2 (Dense)             (None, 96)                16992

 dense_3 (Dense)             (None, 1)                 97

=================================================================
Total params: 78,449
Trainable params: 78,449
Non-trainable params: 0
_________________________________________________________________
**************Execution ended***********************
*****************************************************



**************Execution started for***********************
Activation: relu  layers: 5  nodes: 512  batch: 10000
building node: 1
building node: 2
building node: 3
building node: 4
Epoch 1/800
12/12 [==============================] - 1s 24ms/step - loss: 10.6516 - accu
racy: 0.5234 - val_loss: 1.0219 - val_accuracy: 0.5082
Epoch 2/800
12/12 [==============================] - 0s 11ms/step - loss: 0.9978 - accur
acy: 0.5838 - val_loss: 0.6522 - val_accuracy: 0.6597
…
Epoch 118/800
12/12 [==============================] - 0s 11ms/step - loss: 0.1042 - accur
acy: 0.9622 - val_loss: 0.2590 - val_accuracy: 0.9285
```

```
Epoch 119/800
12/12 [==============================] - 0s 11ms/step - loss: 0.1090 - accur
acy: 0.9597 - val_loss: 0.2398 - val_accuracy: 0.9332
Best score:  0.9622272849082947
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               32256

 dense_1 (Dense)             (None, 388)               199044

 dense_2 (Dense)             (None, 264)               102696

 dense_3 (Dense)             (None, 140)               37100

 dense_4 (Dense)             (None, 1)                 141

=================================================================
Total params: 371,237
Trainable params: 371,237
Non-trainable params: 0
_________________________________________________________________
**************Execution ended***********************
*****************************************************



**************Execution started for***********************
Activation: relu  layers: 5  nodes: 512  batch: 25000
building node: 1
building node: 2
building node: 3
building node: 4
Epoch 1/800
5/5 [==============================] - 1s 56ms/step - loss: 19.6319 - accura
cy: 0.5458 - val_loss: 11.9125 - val_accuracy: 0.4046
….
Epoch 207/800
5/5 [==============================] - 0s 24ms/step - loss: 0.1945 - accurac
y: 0.9222 - val_loss: 0.2751 - val_accuracy: 0.9095
Epoch 208/800
5/5 [==============================] - 0s 23ms/step - loss: 0.1710 - accurac
y: 0.9343 - val_loss: 0.2613 - val_accuracy: 0.9197
Epoch 209/800
```

```
5/5 [==============================] - 0s 23ms/step - loss: 0.1568 - accurac
y: 0.9405 - val_loss: 0.2571 - val_accuracy: 0.9197
Epoch 210/800
5/5 [==============================] - 0s 23ms/step - loss: 0.1514 - accurac
y: 0.9425 - val_loss: 0.2608 - val_accuracy: 0.9198
Epoch 211/800
5/5 [==============================] - 0s 24ms/step - loss: 0.1483 - accurac
y: 0.9440 - val_loss: 0.2559 - val_accuracy: 0.9213
Best score:  0.9482612013816833
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               32256

 dense_1 (Dense)             (None, 388)               199044

 dense_2 (Dense)             (None, 264)               102696

 dense_3 (Dense)             (None, 140)               37100

 dense_4 (Dense)             (None, 1)                 141

=================================================================
Total params: 371,237
Trainable params: 371,237
Non-trainable params: 0
_________________________________________________________________
*************Execution ended**********************
**************************************************
```

Out[52]:

	Max Accuracy	Avg Accuracy	Model
0	0.949573	0.922388	Activation:relu layers:3 nodes:64 batch:10000
0	0.941608	0.910226	Activation:relu layers:3 nodes:64 batch:25000
0	0.945734	0.905817	Activation:relu layers:4 nodes:128 batch:10000

0	0.949199	0.894355	Activation:relu layers:4 nodes:128 batch:25000
0	0.966448	0.918571	Activation:relu layers:4 nodes:256 batch:10000
0	0.962227	0.897282	Activation:relu layers:5 nodes:512 batch:10000
0	0.948261	0.870642	Activation:relu layers:5 nodes:512 batch:25000

In [ ]:

```
#Analyze RF
#{'class_weight': 'balanced', 'criterion': 'gini', 'max_features': 15, 'min_
samples_leaf': 5, 'n_estimators': 250, 'random_state': 45}
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators = 250,
      min_samples_leaf = 5, max_features = 25, random_state =45, class_weigh
t ='balanced')
%time RF.fit(X_train, y_train)
CPU times: user 13min 8s, sys: 574 ms, total: 13min 9s
Wall time: 13min 6s
```

Out[ ]:

```
RandomForestClassifier(class_weight='balanced', max_features=25,
                       min_samples_leaf=5, n_estimators=250, random_state=45
)
```

In [35]:

```
from tensorflow.keras.callbacks import EarlyStopping
safety = EarlyStopping(monitor='val_loss', patience=100)
seed = 45 #88.27
nn_m = createmodel(n_layers=4, first_layer_nodes=256, last_layer_nodes=15,
                activation_func='relu', loss_func=tf.keras.losses.BinaryCros
sentropy()) #tanh
hist = nn_m.fit(X_train, y_train, epochs=2000, batch_size=10000,
        validation_data=(X_test, y_test), callbacks=[safety], verbose=1) # a
dd validation left out here

best_score = max(hist.history['accuracy'])
print("Best score: ",best_score)
building node: 1
building node: 2
building node: 3
Epoch 1/2000
15/15 [==============================] - 0s 6ms/step - loss: 0.8424 - accura
cy: 0.7118 - val_loss: 0.5927 - val_accuracy: 0.7469
….
```

```
Epoch 233/2000
15/15 [==============================] - 0s 6ms/step - loss: 0.0892 - accura
cy: 0.9702 - val_loss: 0.2228 - val_accuracy: 0.9427
Epoch 234/2000
15/15 [==============================] - 0s 6ms/step - loss: 0.0943 - accura
cy: 0.9674 - val_loss: 0.2360 - val_accuracy: 0.9381
Epoch 235/2000
15/15 [==============================] - 0s 6ms/step - loss: 0.0931 - accura
cy: 0.9679 - val_loss: 0.2159 - val_accuracy: 0.9425
Epoch 236/2000
15/15 [==============================] - 0s 6ms/step - loss: 0.0871 - accura
cy: 0.9708 - val_loss: 0.2161 - val_accuracy: 0.9444
Best score:  0.9712548851966858
```

In [50]:

```
with open('./drive/MyDrive/data/nn1_mdl.sav', 'wb') as f:
    pickle.dump(nn_m, f)
```

```
INFO:tensorflow:Assets written to: ram://47c9af35-7f43-4f1f-af15-92bb922aaef
8/assets
```

In [14]:

```
with open('./drive/MyDrive/data/rf1_mdl.sav', 'rb') as f:
  mdl_rf = pickle.load(f)
with open('./drive/MyDrive/data/xgb2_mdl.sav', 'rb') as f:
  mdl_xgb = pickle.load(f)
with open('./drive/MyDrive/data/nn1_mdl.sav', 'rb') as f:
  mdl_nn = pickle.load(f)
```

In [13]:

```
from sklearn.metrics import confusion_matrix
import itertools

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`.
    """
    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)
```

```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

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

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

In [37]:

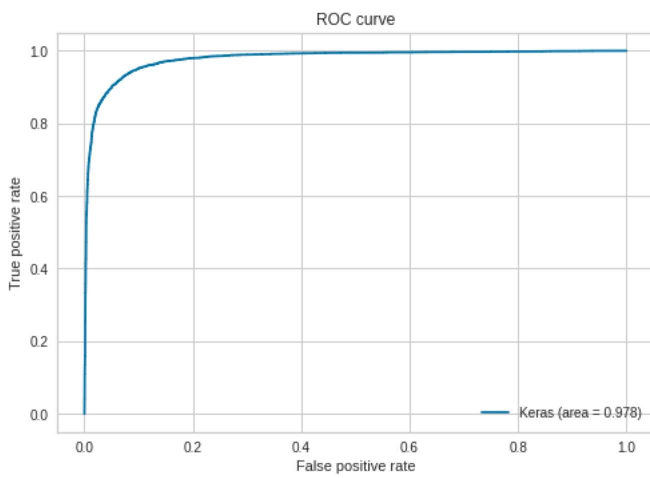
```
from sklearn.metrics import roc_curve
from numpy import sqrt
from sklearn.metrics import auc

def to_labels(pos_probs, threshold):
        return (pos_probs >= threshold).astype('int')

def get_mdl_stats(name, mdl, thresh, X_test, y_test, is_nn=False):
  print("****** stats for ", name, "********")
  if is_nn:
    y_pred_keras = mdl.predict(X_test)
  else:
    y_pred_keras = mdl.predict_proba(X_test)
    y_pred_keras=np.delete(y_pred_keras, 0, 1)
  auc_keras = auc(fpr_keras, tpr_keras)
  plt.figure(1)

  plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_ke
ras))
  plt.xlabel('False positive rate')
  plt.ylabel('True positive rate')
  plt.title('ROC curve')
  #plt.scatter(fpr_keras[ix], tpr_keras[ix], marker='o', color='black', labe
l='Best')
  plt.legend(loc='best')
  plt.show()
```

```
  #print(y_pred_keras)
  y_pred_keras[y_pred_keras <= thresh] = 0.
  y_pred_keras[y_pred_keras > thresh] = 1.
  #print(y_pred_keras)
  cm_plot_labels = ['0','1']
  cm = confusion_matrix(y_true=y_test, y_pred=y_pred_keras)
  print("Total fimnancial loss: ",cm[0,1]*25 + cm[1,0]*100)
  plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matr
ix')
  #np.unique(y_test, return_counts=True)
  #y_pred_keras
  print(classification_report(y_test, y_pred_keras, target_names=['0','1']))
```

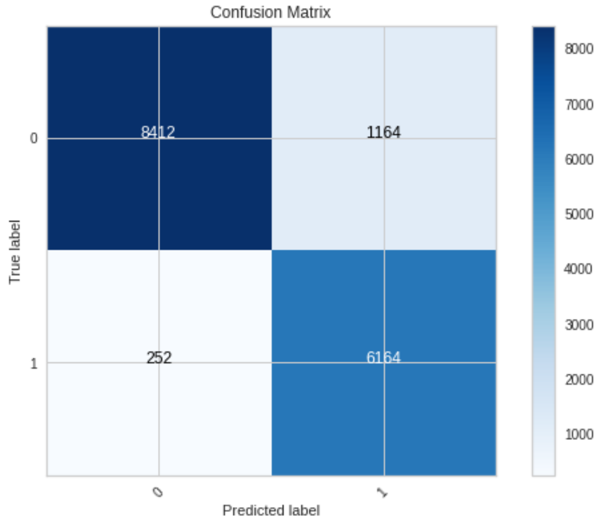
In [38]:

```
get_mdl_stats('RF', mdl_rf, .35, X_test, y_test)
```

```
****** stats for  RF ********
```



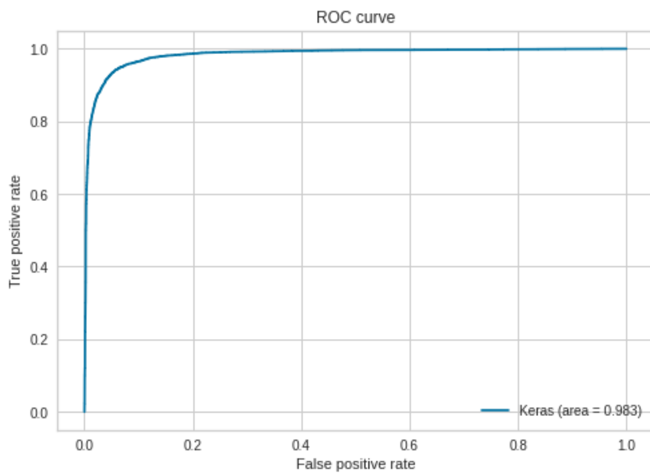

```
Total fimnancial loss:  54300
Confusion matrix, without normalization
[[8412 1164]
 [ 252 6164]]
              precision    recall  f1-score   support

           0       0.97      0.88      0.92      9576
           1       0.84      0.96      0.90      6416

    accuracy                           0.91     15992
   macro avg       0.91      0.92      0.91     15992
weighted avg       0.92      0.91      0.91     15992
```



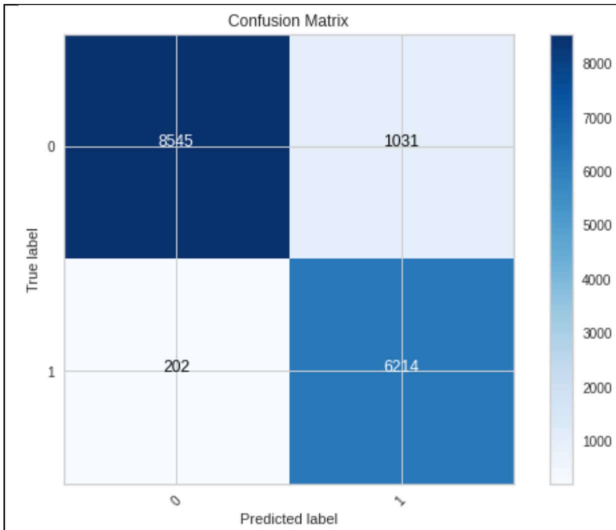
In [39]:

```
get_mdl_stats('XGB', mdl_xgb, .25, X_test, y_test) #.255
```

```
****** stats for  XGB ********
```



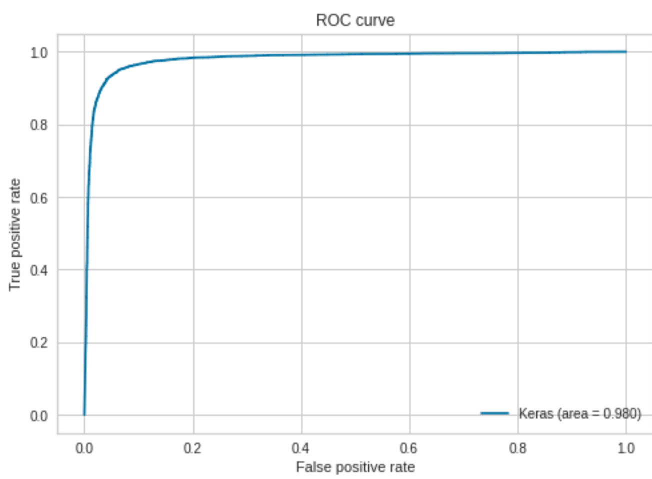

```
Total fimnancial loss:  45975
Confusion matrix, without normalization
[[8545 1031]
 [ 202 6214]]
              precision    recall  f1-score   support

           0       0.98      0.89      0.93      9576
           1       0.86      0.97      0.91      6416

    accuracy                           0.92     15992
   macro avg       0.92      0.93      0.92     15992
weighted avg       0.93      0.92      0.92     15992
```



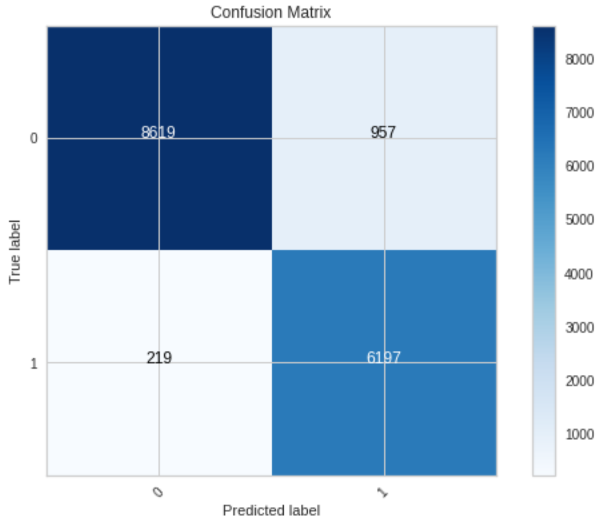
In [40]:

```
get_mdl_stats('NN', mdl_nn, .134, X_test, y_test, True)
```

```
****** stats for  NN ********
```




```
Total fimnancial loss:  45825
Confusion matrix, without normalization
[[8619  957]
 [ 219 6197]]
              precision    recall  f1-score   support

           0       0.98      0.90      0.94      9576
           1       0.87      0.97      0.91      6416

    accuracy                           0.93     15992
   macro avg       0.92      0.93      0.92     15992
weighted avg       0.93      0.93      0.93     15992
```



Confusion Matrix
8619
957
219
6197
0
1
True label
Predicted label
8000
7000
6000
5000
4000
3000
2000
1000

In [ ]: