Finance

[1]:
```
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib import gridspec
import seaborn as sns
import numpy as np
import time
import copy
import sys
from datetime import datetime
from joblib import Parallel, delayed
import os
from scipy.stats import multivariate_normal
import plotly.graph_objects as go
from chart_studio.plotly import iplot
from matplotlib.colors import ListedColormap

import tensorflow as tf
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
import tensorboard
from keras.utils.vis_utils import plot_model
from ann_visualizer.visualize import ann_viz

from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report, roc_curve, auc, f1_score, precision_score,␣
 ↪recall_score
from sklearn.svm import SVC, LinearSVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,␣
 ↪GradientBoostingClassifier, ExtraTreesClassifier, AdaBoostClassifier,␣
 ↪IsolationForest

from imblearn.ensemble import BalancedRandomForestClassifier
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler, Normalizer, MaxAbsScaler,␣
 ↪StandardScaler, StandardScaler, RobustScaler
from sklearn.model_selection import GridSearchCV, cross_val_score,␣
 ↪GridSearchCV, train_test_split, cross_validate, KFold, RandomizedSearchCV
from sklearn.pipeline import Pipeline
import xgboost as xgb
from sklearn.metrics import mean_squared_error, matthews_corrcoef
from sklearn.naive_bayes import GaussianNB, MultinomialNB
plt.style.use('ggplot')
sns.set()
```

```
Using TensorFlow backend.
```

[2]:
```
from dask.distributed import Client

client = Client("tcp://127.0.0.1:51484")
client
```

```
C:\Users\Pritesh\Anaconda3\lib\site-packages\distributed\client.py:1137:
VersionMismatchWarning: Mismatched versions found

+---------+--------+-----------+---------+
| Package | client | scheduler | workers |
+---------+--------+-----------+---------+
| dask    | 2.25.0 | 2.20.0    | 2.25.0  |
+---------+--------+-----------+---------+
  warnings.warn(version_module.VersionMismatchWarning(msg[0]["warning"]))
```

[2]:
```
<Client: 'tcp://127.0.0.1:51484' processes=4 threads=12, memory=17.13 GB>
```

[3]:
```
tf.__version__
```

[3]:
```
'2.1.0'
```

[4]:
```
pd.set_option('precision', 3)
```

[5]:
```
df_train_original = pd.read_csv('Financial/train_data.csv')
df_train_hidden = pd.read_csv('Financial/test_data_hidden.csv')
df_test = pd.read_csv('Financial/test_data.csv')
frame = [df_train_original,df_train_hidden]
df = pd.concat(frame)
```

[6]:
```
df_train = copy.deepcopy(df_train_original)
```

1 EDA

[7]:
```
df.shape
```

[7]:
```
(284807, 31)
```

[8]:
```
df.head()
```

[8]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	\
0	38355.0	1.044	0.319	1.046	2.806	-0.561	-0.368	0.033	-0.042	-0.323	
1	22555.0	-1.665	0.808	1.806	1.903	-0.822	0.935	-0.825	0.976	1.747	
2	2431.0	-0.324	0.602	0.865	-2.138	0.295	-1.252	1.072	-0.335	1.071	
3	86773.0	-0.258	1.218	-0.585	-0.875	1.222	-0.311	1.074	-0.161	0.201	
4	127202.0	2.142	-0.495	-1.937	-0.818	-0.025	-1.027	-0.152	-0.306	-0.869	

	…	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	…	-0.240	-0.680	0.085	0.685	0.319	-0.205	0.002	0.038	49.67	0
1	…	-0.335	-0.511	0.036	0.148	-0.529	-0.567	-0.596	-0.220	16.94	0
2	…	0.012	0.353	-0.342	-0.146	0.094	-0.804	0.229	-0.022	1.00	0
3	…	-0.425	-0.781	0.019	0.179	-0.316	0.097	0.270	-0.021	10.78	0
4	…	0.010	0.022	0.079	-0.481	0.024	-0.279	-0.030	-0.044	39.96	0

[5 rows x 31 columns]

1.1 Perform an EDA on the Dataset.

- ### Check all the latent features and parameters with their mean and standard deviation. Value are close to 0 centered (mean) with unit standard deviation

[9]:
```
df.iloc[:, [x for x in range(1,29)]].describe()
```

[9]:

	V1	V2	V3	V4	V5	V6	\
count	2.848e+05	2.848e+05	2.848e+05	2.848e+05	2.848e+05	2.848e+05	
mean	1.254e-15	3.285e-16	-1.401e-15	2.054e-15	1.021e-15	1.498e-15	
std	1.959e+00	1.651e+00	1.516e+00	1.416e+00	1.380e+00	1.332e+00	
min	-5.641e+01	-7.272e+01	-4.833e+01	-5.683e+00	-1.137e+02	-2.616e+01	
25%	-9.204e-01	-5.985e-01	-8.904e-01	-8.486e-01	-6.916e-01	-7.683e-01	
50%	1.811e-02	6.549e-02	1.798e-01	-1.985e-02	-5.434e-02	-2.742e-01	
75%	1.316e+00	8.037e-01	1.027e+00	7.433e-01	6.119e-01	3.986e-01	
max	2.455e+00	2.206e+01	9.383e+00	1.688e+01	3.480e+01	7.330e+01	

	V7	V8	V9	V10	…	V19	V20	\

```
count  2.848e+05  2.848e+05  2.848e+05  2.848e+05  …  2.848e+05  2.848e+05
mean  -5.773e-16  1.195e-16 -2.420e-15  2.281e-15  …  1.035e-15  6.482e-16
std    1.237e+00  1.194e+00  1.099e+00  1.089e+00  …  8.140e-01  7.709e-01
min   -4.356e+01 -7.322e+01 -1.343e+01 -2.459e+01  … -7.214e+00 -5.450e+01
25%   -5.541e-01 -2.086e-01 -6.431e-01 -5.354e-01  … -4.563e-01 -2.117e-01
50%    4.010e-02  2.236e-02 -5.143e-02 -9.292e-02  …  3.735e-03 -6.248e-02
75%    5.704e-01  3.273e-01  5.971e-01  4.539e-01  …  4.589e-01  1.330e-01
max    1.206e+02  2.001e+01  1.559e+01  2.375e+01  …  5.592e+00  3.942e+01

             V21        V22        V23        V24        V25        V26  \
count  2.848e+05  2.848e+05  2.848e+05  2.848e+05  2.848e+05  2.848e+05
mean   1.622e-16 -3.311e-16  2.561e-16  4.488e-15  5.294e-16  1.679e-15
std    7.345e-01  7.257e-01  6.245e-01  6.056e-01  5.213e-01  4.822e-01
min   -3.483e+01 -1.093e+01 -4.481e+01 -2.837e+00 -1.030e+01 -2.605e+00
25%   -2.284e-01 -5.424e-01 -1.618e-01 -3.546e-01 -3.171e-01 -3.270e-01
50%   -2.945e-02  6.782e-03 -1.119e-02  4.098e-02  1.659e-02 -5.214e-02
75%    1.864e-01  5.286e-01  1.476e-01  4.395e-01  3.507e-01  2.410e-01
max    2.720e+01  1.050e+01  2.253e+01  4.585e+00  7.520e+00  3.517e+00

             V27        V28
count  2.848e+05  2.848e+05
mean  -3.501e-16 -1.357e-16
std    4.036e-01  3.301e-01
min   -2.257e+01 -1.543e+01
25%   -7.084e-02 -5.296e-02
50%    1.342e-03  1.124e-02
75%    9.105e-02  7.828e-02
max    3.161e+01  3.385e+01

[8 rows x 28 columns]
```

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 284807 entries, 0 to 56961
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
```

```
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 69.5 MB
```

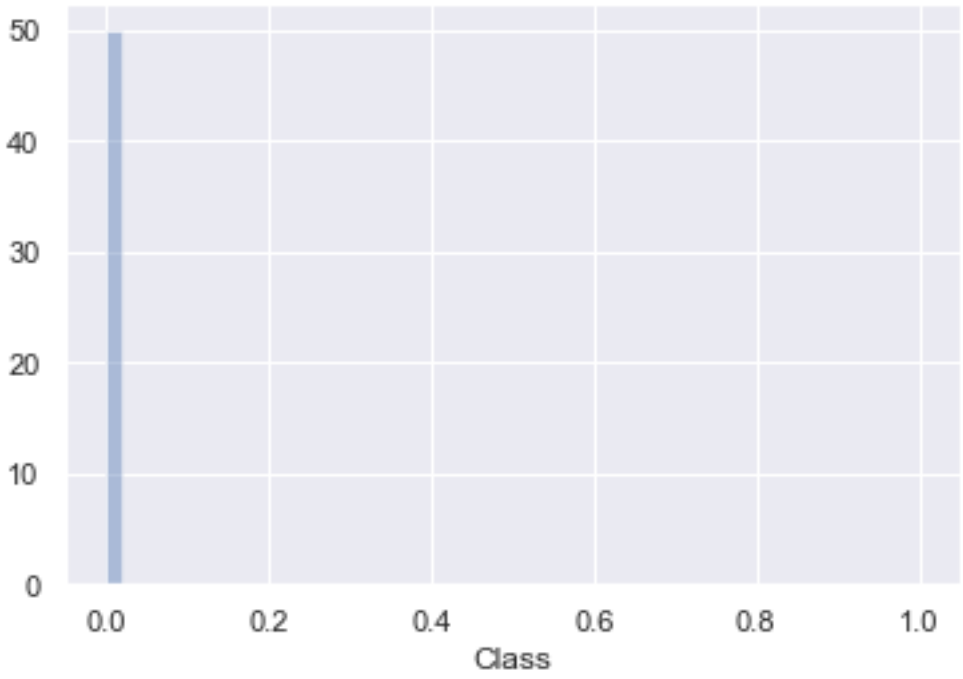
[11]:
```
df.Class.value_counts()
```

[11]:
```
0    284315
1       492
Name: Class, dtype: int64
```

[12]:
```
# df.Class.plot.bar()
sns.distplot(df.Class)
```

```
C:\Users\Pritesh\Anaconda3\lib\site-packages\seaborn\distributions.py:369:
UserWarning: Default bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```

[12]:
```
<AxesSubplot:xlabel='Class'>
```



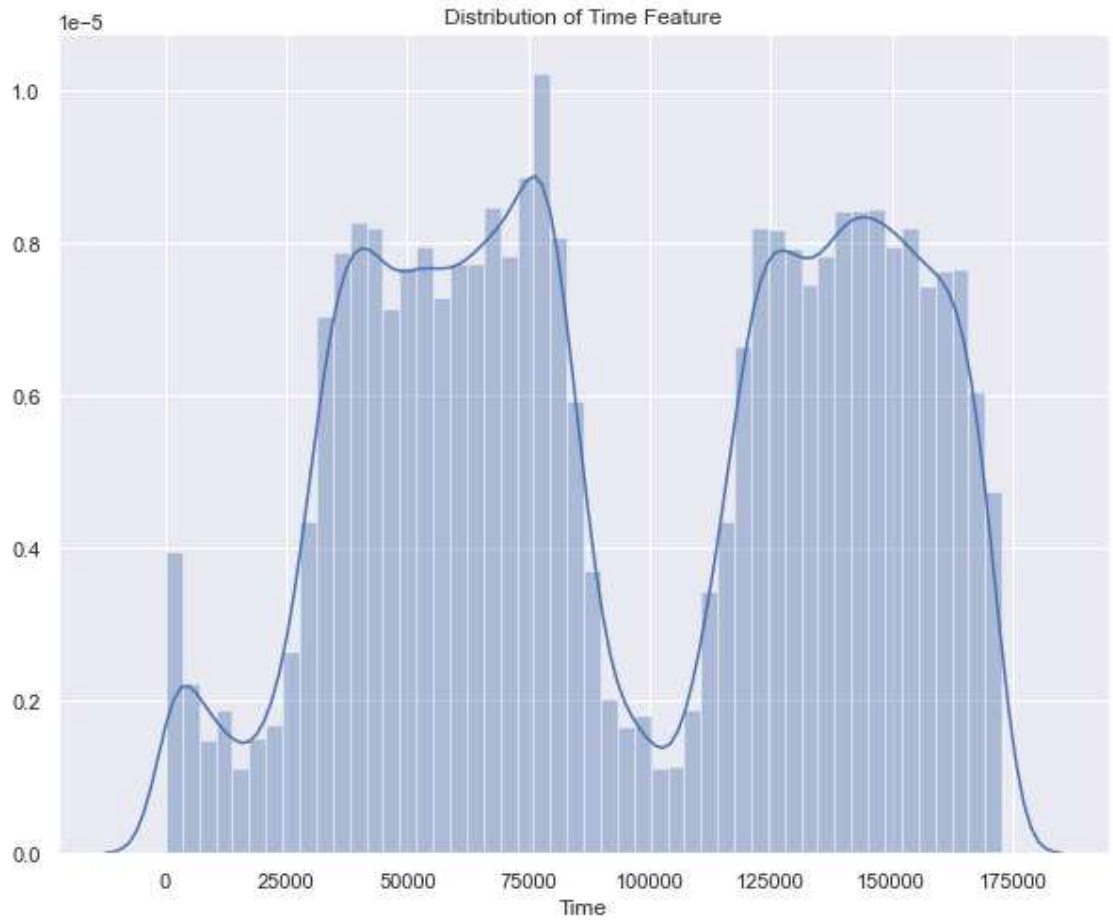
-

1.1.1 Find if there is any connection between Time, Amount, and the transaction being fraudulent.

Time

[13]:
```
#visualizations of time and amount
plt.figure(figsize=(10,8))
plt.title('Distribution of Time Feature')
sns.distplot(df.Time)
```

[13]: <AxesSubplot:title={'center':'Distribution of Time Feature'}, xlabel='Time'>



[14]:
```
# Let's convert the time from seconds to hours to ease the interpretation.
df.loc[:,'Time'] = df.Time / 3600
df['Time'].max() / 24
```

[14]: 1.9999074074074075

[15]:
```
plt.figure(figsize=(12,4), dpi=80)
sns.distplot(df['Time'], bins=48, kde=False)
plt.xlim([0,48])
plt.xticks(np.arange(0,54,6))
plt.xlabel('Time After First Transaction (hr)')
plt.ylabel('Count')
plt.title('Transaction Times')
```

[15]: Text(0.5, 1.0, 'Transaction Times')

[16]:
```
df['Time'].describe()
```

[16]:
```
count    284807.000
mean         26.337
std          13.191
min           0.000
25%          15.056
50%          23.526
75%          38.700
max          47.998
Name: Time, dtype: float64
```

Amount

[17]:
```
df['Amount'].describe()
```

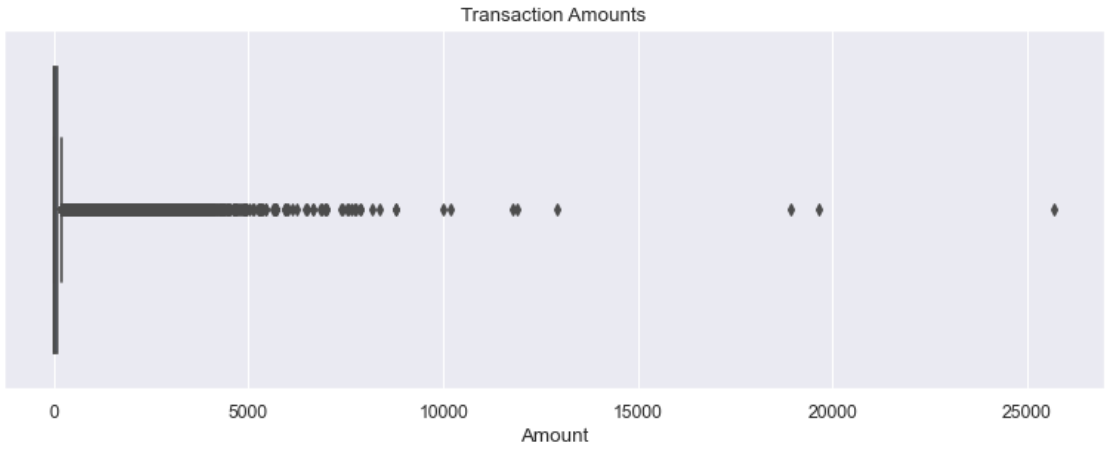
[17]:
```
count    284807.000
mean         88.350
std         250.120
min           0.000
25%           5.600
50%          22.000
75%          77.165
max       25691.160
Name: Amount, dtype: float64
```

[18]:
```
plt.figure(figsize=(12,4), dpi=80)
sns.distplot(df['Amount'], bins=300, kde=False)
plt.ylabel('Count')
plt.title('Transaction Amounts')
```

[18]:
```
Text(0.5, 1.0, 'Transaction Amounts')
```

[19]:
```
plt.figure(figsize=(12,4), dpi=80)
sns.boxplot(df['Amount'])
plt.title('Transaction Amounts')
```

[19]:
```
Text(0.5, 1.0, 'Transaction Amounts')
```



[20]:
```
df['Amount'].skew()
```

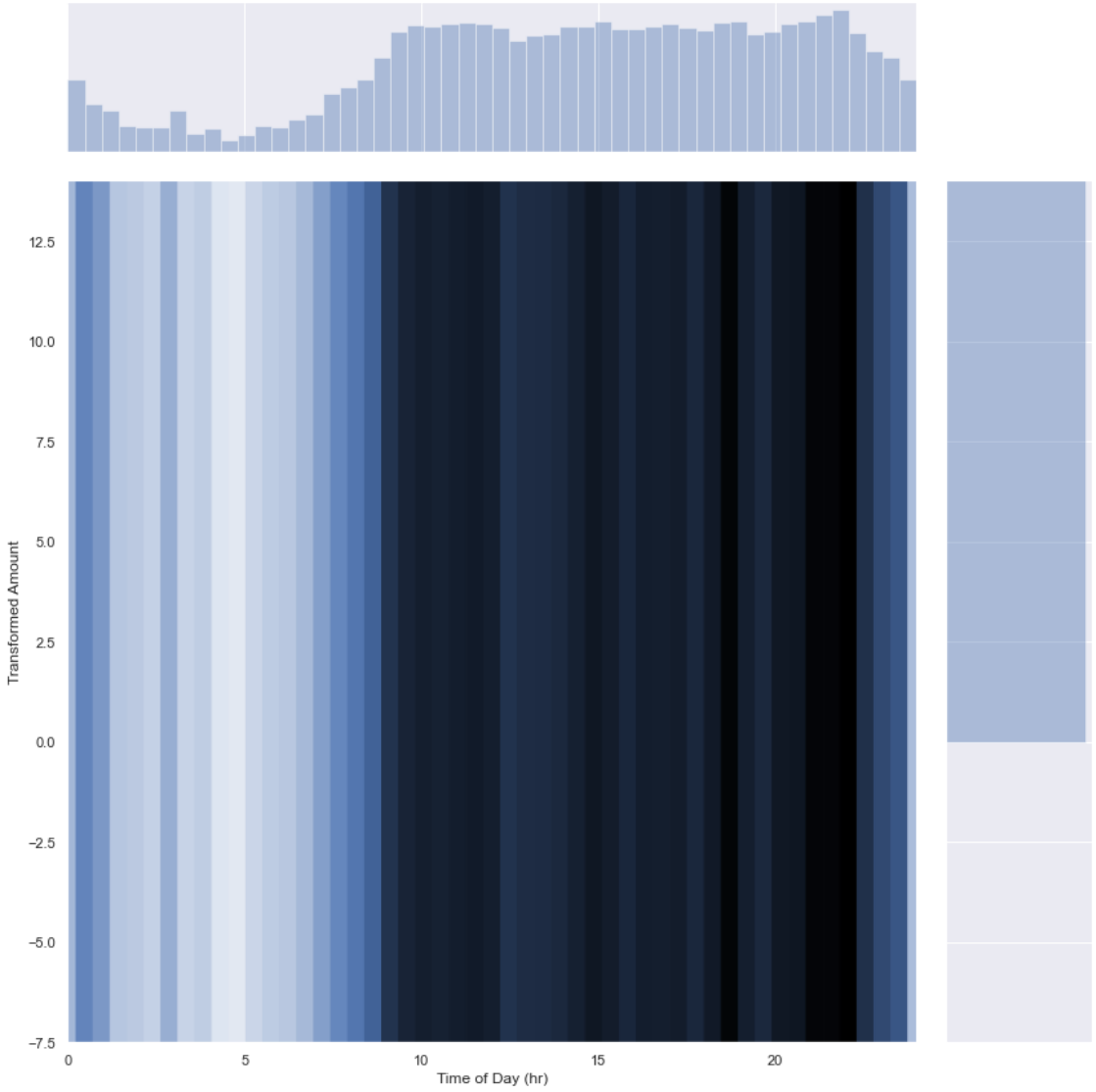
[20]:
```
16.977724453761013
```

Time vs Amount

[21]:
```
sns.jointplot(df['Time'].apply(lambda x: x % 24), df['Amount'], kind='hex',␣
↪stat_func=None, size=12, xlim=(0,24), ylim=(-7.5,14)).set_axis_labels('Time␣
↪of Day (hr)','Transformed Amount')
```

```
C:\Users\Pritesh\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2264:
UserWarning: The `size` parameter has been renamed to `height`; please update
your code.
  warnings.warn(msg, UserWarning)
```

[21]: `<seaborn.axisgrid.JointGrid at 0x15b5b4d4248>`



[22]:
```
df.hist(figsize = (20, 20))
plt.show()
```

[23]:
```
#

# std_scaler = StandardScaler()
# rob_scaler = RobustScaler()

# df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.
 ↪reshape(-1,1))
# df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

# df.drop(['Time','Amount'], axis=1, inplace=True)
# df.head()
```

[24]:
```
# Since our classes are highly skewed we should make them equivalent in order␣
 ↪to have a normal distribution of the classes.
```

```
# Lets shuffle the data before creating the subsamples

df = df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:394]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)

new_df.head()
```

[24]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	\
189937	12.370	-0.235	0.355	1.972	-1.256	-0.681	-0.666	0.059	-0.003	1.122	
181016	23.851	-3.365	2.929	-5.661	3.891	-1.840	-1.801	-5.559	2.402	-2.849	
37300	19.770	-0.426	0.512	1.598	-0.496	-0.053	-0.313	0.415	0.157	-0.316	
24481	0.582	1.229	-0.665	0.450	-1.418	-0.929	-0.404	-0.569	-0.056	1.892	
94916	2.942	1.209	-0.142	0.167	0.155	-0.269	-0.434	-0.163	-0.153	1.782	

	…	V21	V22	V23	V24	V25	V26	V27	V28	\
189937	…	0.221	0.912	-0.286	0.451	1.883e-01	-0.532	0.123	0.040	
181016	…	0.875	-0.103	-0.606	-0.743	9.632e-02	-0.135	1.239	0.100	
37300	…	0.119	0.174	-0.125	-0.003	-8.394e-04	0.203	-0.040	-0.006	
24481	…	0.098	0.469	-0.263	-0.347	6.010e-01	0.158	0.039	0.026	
94916	…	-0.511	-1.465	0.040	-0.694	8.065e-02	0.494	-0.108	0.008	

	Amount	Class
189937	1.00	1
181016	1.00	1
37300	31.90	0
24481	56.75	0
94916	75.00	0

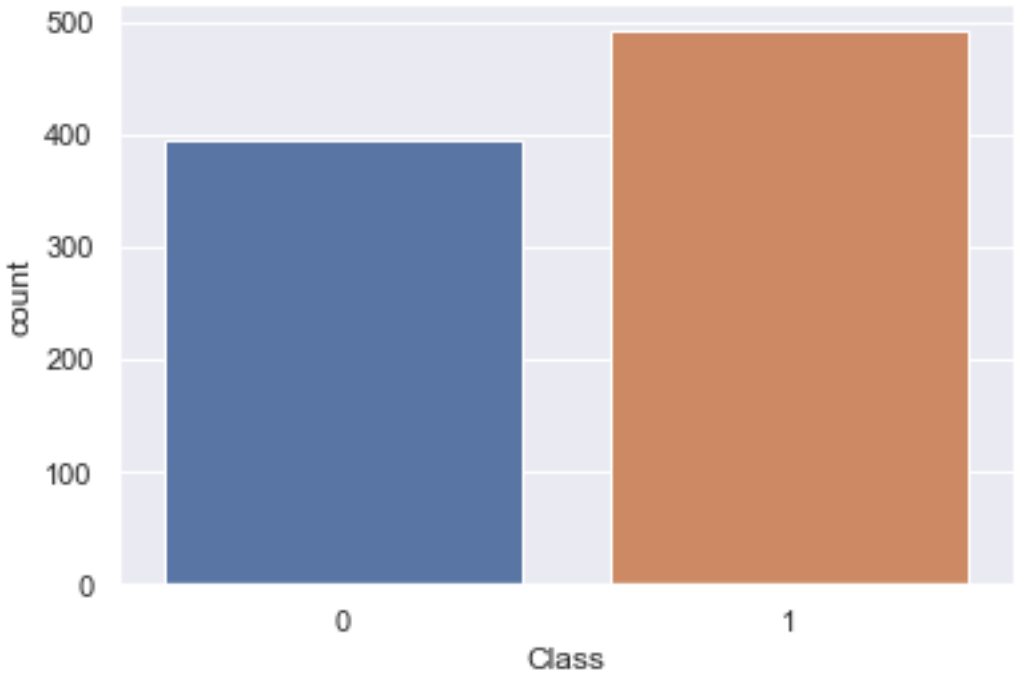
[5 rows x 31 columns]

[25]:
```
new_df.shape
```

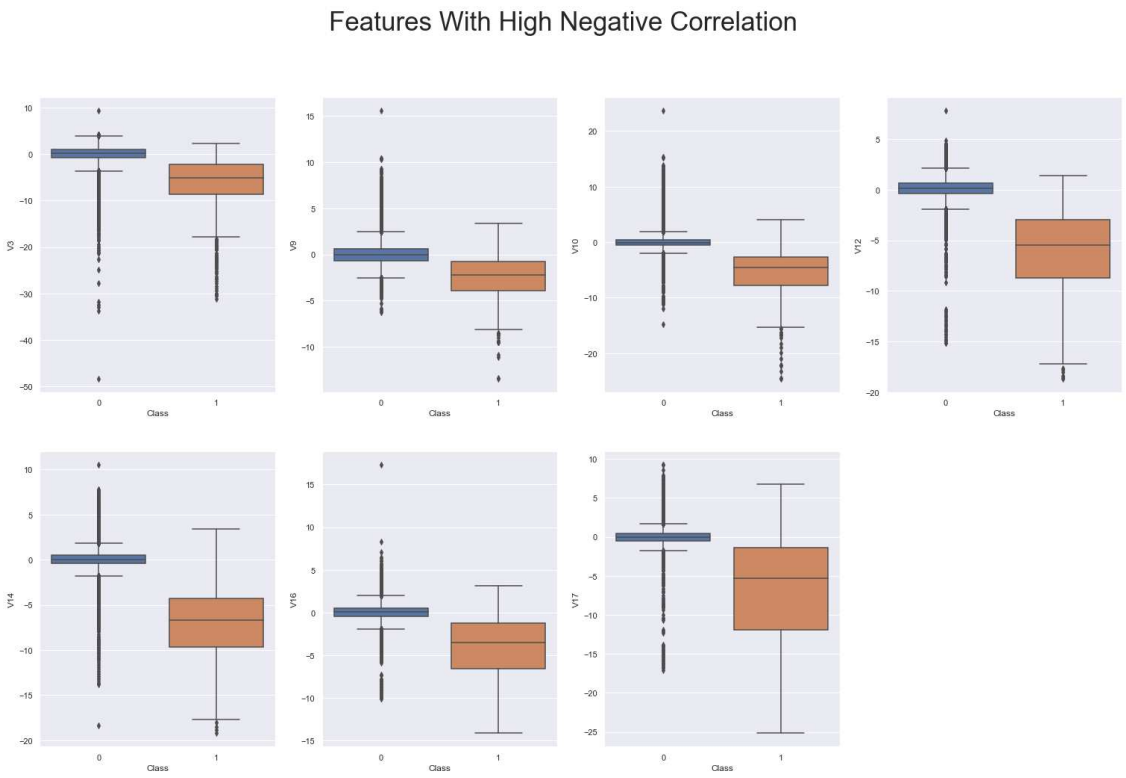
[25]: (886, 31)

[26]:
```
sns.countplot('Class', data=new_df)
```

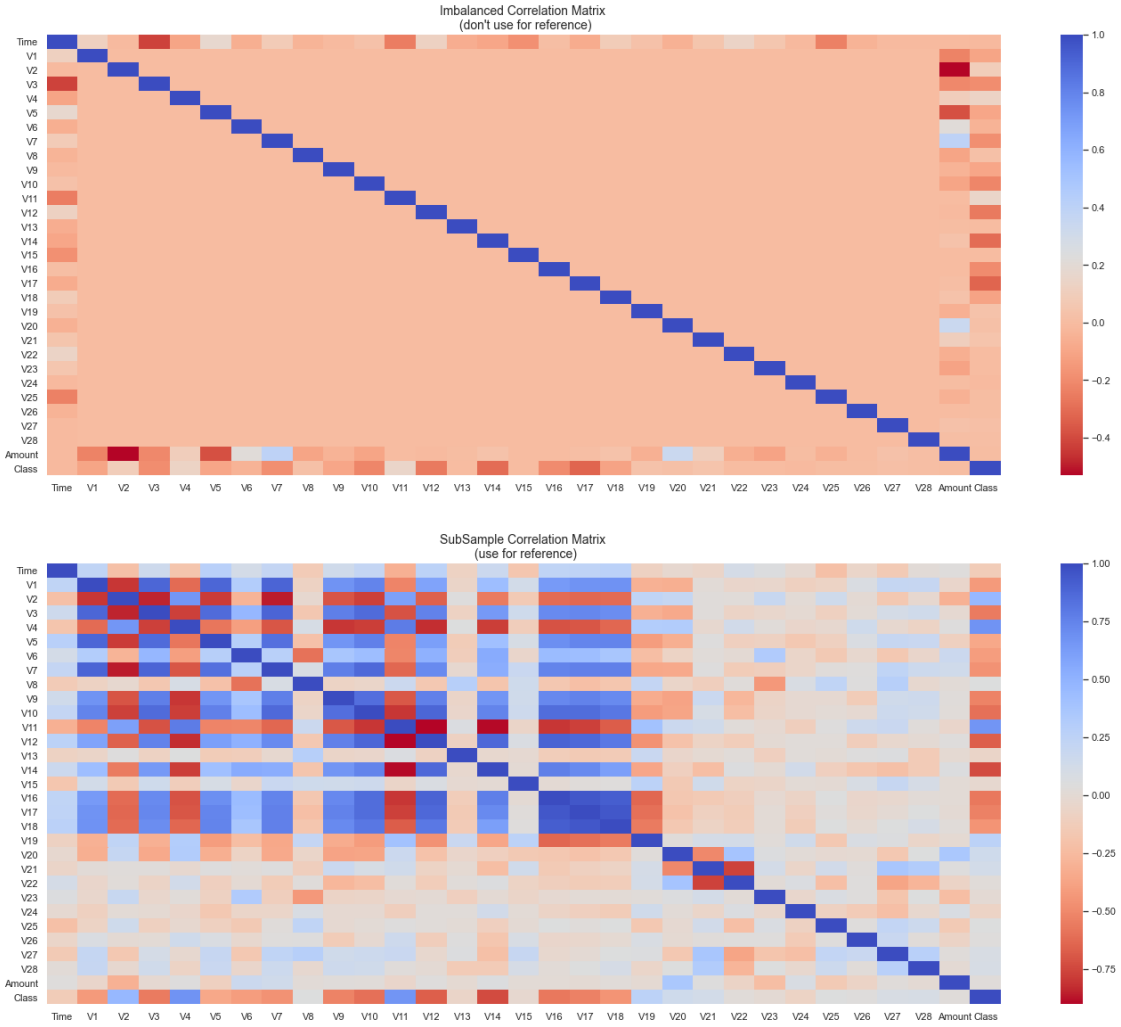
[26]: <AxesSubplot:xlabel='Class', ylabel='count'>



[27]:
```
#visualizing the features w high negative correlation
f, axes = plt.subplots(nrows=2, ncols=4, figsize=(26,16))

f.suptitle('Features With High Negative Correlation', size=35)
sns.boxplot(x="Class", y="V3", data=df, ax=axes[0,0])
sns.boxplot(x="Class", y="V9", data=df, ax=axes[0,1])
sns.boxplot(x="Class", y="V10", data=df, ax=axes[0,2])
sns.boxplot(x="Class", y="V12", data=df, ax=axes[0,3])
sns.boxplot(x="Class", y="V14", data=df, ax=axes[1,0])
sns.boxplot(x="Class", y="V16", data=df, ax=axes[1,1])
sns.boxplot(x="Class", y="V17", data=df, ax=axes[1,2])
f.delaxes(axes[1,3])
```

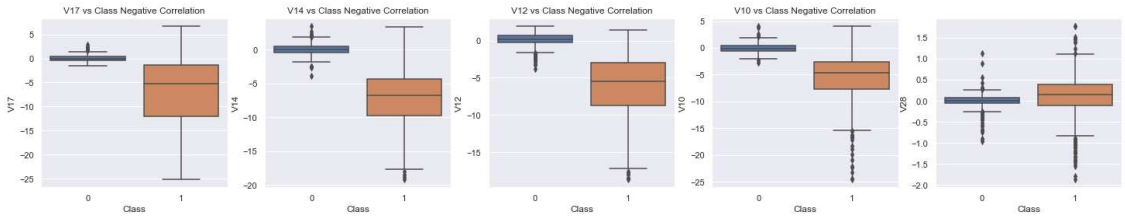


[28]:
```
f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix \n (don't use for reference)",␣
 ↪fontsize=14)


sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix \n (use for reference)',␣
 ↪fontsize=14)
plt.show()
```

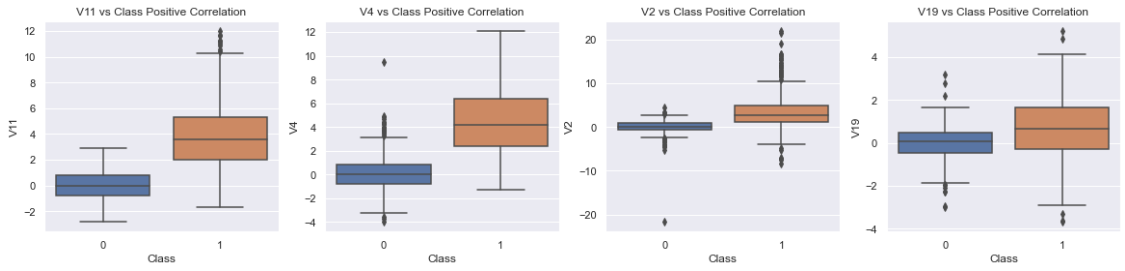


[29]:
```
f, axes = plt.subplots(ncols=5, figsize=(25,4))

# Negative Correlations with our Class (The lower our feature value the more␣
 ↪likely it will be a fraud transaction)
sns.boxplot(x="Class", y="V17", data=new_df, ax=axes[0])
axes[0].set_title('V17 vs Class Negative Correlation')

sns.boxplot(x="Class", y="V14", data=new_df, ax=axes[1])
axes[1].set_title('V14 vs Class Negative Correlation')


sns.boxplot(x="Class", y="V12", data=new_df, ax=axes[2])
axes[2].set_title('V12 vs Class Negative Correlation')
```

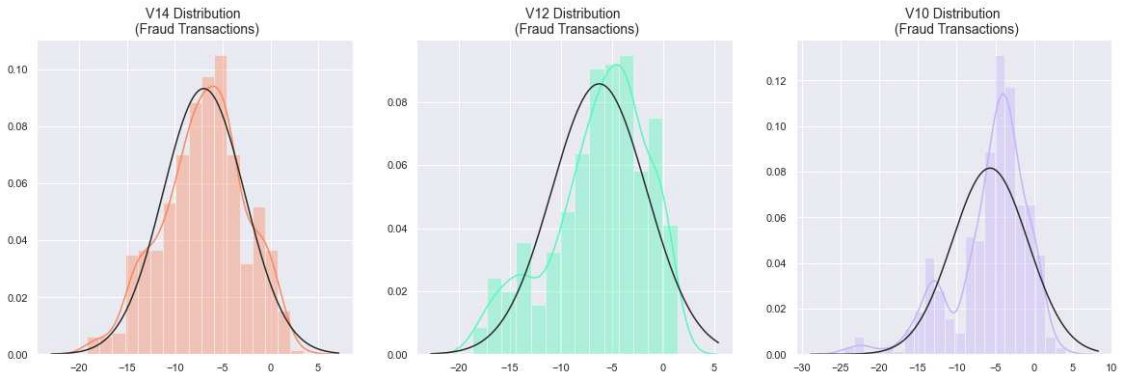
```
sns.boxplot(x="Class", y="V10", data=new_df, ax=axes[3])
axes[3].set_title('V10 vs Class Negative Correlation')

sns.boxplot(x="Class", y="V28", data=new_df, ax=axes[4])
axes[3].set_title('V10 vs Class Negative Correlation')

plt.show()
```



[30]:
```
f, axes = plt.subplots(ncols=4, figsize=(20,4))

# Positive correlations (The higher the feature the probability increases that
 ↪it will be a fraud transaction)
sns.boxplot(x="Class", y="V11", data=new_df, ax=axes[0])
axes[0].set_title('V11 vs Class Positive Correlation')

sns.boxplot(x="Class", y="V4", data=new_df, ax=axes[1])
axes[1].set_title('V4 vs Class Positive Correlation')


sns.boxplot(x="Class", y="V2", data=new_df, ax=axes[2])
axes[2].set_title('V2 vs Class Positive Correlation')


sns.boxplot(x="Class", y="V19", data=new_df, ax=axes[3])
axes[3].set_title('V19 vs Class Positive Correlation')

plt.show()
```



[31]:
```
from scipy.stats import norm

f, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(20, 6))

v14_fraud_dist = new_df['V14'].loc[new_df['Class'] == 1].values
sns.distplot(v14_fraud_dist,ax=ax1, fit=norm, color='#FB8861')
ax1.set_title('V14 Distribution \n (Fraud Transactions)', fontsize=14)

v12_fraud_dist = new_df['V12'].loc[new_df['Class'] == 1].values
sns.distplot(v12_fraud_dist,ax=ax2, fit=norm, color='#56F9BB')
ax2.set_title('V12 Distribution \n (Fraud Transactions)', fontsize=14)


v10_fraud_dist = new_df['V10'].loc[new_df['Class'] == 1].values
sns.distplot(v10_fraud_dist,ax=ax3, fit=norm, color='#C5B3F9')
ax3.set_title('V10 Distribution \n (Fraud Transactions)', fontsize=14)

plt.show()
```



[32]:
```
std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df_train['scaled_amount'] = rob_scaler.fit_transform(df_train['Amount'].values.
 ↪reshape(-1,1))
df_train['scaled_time'] = rob_scaler.fit_transform(df_train['Time'].values.
 ↪reshape(-1,1))

df_test['scaled_amount'] = rob_scaler.fit_transform(df_test['Amount'].values.
 ↪reshape(-1,1))
df_test['scaled_time'] = rob_scaler.fit_transform(df_test['Time'].values.
 ↪reshape(-1,1))
```

```
new_df['scaled_amount'] = rob_scaler.fit_transform(new_df['Amount'].values.
 ↪reshape(-1,1))
new_df['scaled_time'] = rob_scaler.fit_transform(new_df['Time'].values.
 ↪reshape(-1,1))

df_train.drop(['Time','Amount'], axis=1, inplace=True)
df_train.head()
```

[32]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	…	\
0	1.044	0.319	1.046	2.806	-0.561	-0.368	0.033	-0.042	-0.323	0.499	…	
1	-1.665	0.808	1.806	1.903	-0.822	0.935	-0.825	0.976	1.747	-0.659	…	
2	-0.324	0.602	0.865	-2.138	0.295	-1.252	1.072	-0.335	1.071	-1.110	…	
3	-0.258	1.218	-0.585	-0.875	1.222	-0.311	1.074	-0.161	0.201	0.154	…	
4	2.142	-0.495	-1.937	-0.818	-0.025	-1.027	-0.152	-0.306	-0.869	0.429	…	

	V22	V23	V24	V25	V26	V27	V28	Class	scaled_amount	\
0	-0.680	0.085	0.685	0.319	-0.205	0.002	0.038	0	0.387	
1	-0.511	0.036	0.148	-0.529	-0.567	-0.596	-0.220	0	-0.071	
2	0.353	-0.342	-0.146	0.094	-0.804	0.229	-0.022	0	-0.294	
3	-0.781	0.019	0.179	-0.316	0.097	0.270	-0.021	0	-0.157	
4	0.022	0.079	-0.481	0.024	-0.279	-0.030	-0.044	0	0.251	

	scaled_time
0	-0.543
1	-0.729
2	-0.965
3	0.025
4	0.500

[5 rows x 31 columns]

[33]:
```
df_test.drop(['Time','Amount'], axis=1, inplace=True)
df_test.head()
```

[33]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	…	\
0	0.115	0.796	-0.150	-0.823	0.879	-0.553	0.939	-0.109	0.111	-0.391	…	
1	-0.039	0.496	-0.811	0.547	1.986	4.386	-1.345	-1.744	-0.563	-0.616	…	
2	2.276	-1.532	-1.022	-1.602	-1.220	-0.462	-1.196	-0.147	-0.950	1.560	…	
3	1.940	-0.358	-1.211	0.383	0.051	-0.171	-0.109	-0.002	0.869	-0.002	…	
4	1.081	-0.503	1.076	-0.543	-1.473	-1.065	-0.443	-0.143	1.660	-1.131	…	

	V21	V22	V23	V24	V25	V26	V27	V28	scaled_amount	\
0	-0.336	-0.808	-0.056	-1.025	-0.370	0.205	0.243	0.086	-0.292	
1	-1.377	-0.072	-0.198	1.015	1.011	-0.168	0.113	0.257	0.878	
2	-0.193	-0.104	0.151	-0.811	-0.198	-0.128	0.014	-0.051	0.289	
3	0.158	0.650	0.034	0.740	0.224	-0.196	-0.013	-0.057	0.113	
4	0.224	0.821	-0.137	0.986	0.563	-0.574	0.090	0.052	0.641	

```
   scaled_time
0        0.328
1       -0.689
2        0.875
3        0.616
4       -0.257

[5 rows x 30 columns]
```

[34]:
```
y = df_train.pop('Class')
x = df_train
```

[35]:
```
df_test.head()
```

[35]:
```
      V1     V2     V3     V4     V5     V6     V7     V8     V9    V10  …  \
0  0.115  0.796 -0.150 -0.823  0.879 -0.553  0.939 -0.109  0.111 -0.391  …
1 -0.039  0.496 -0.811  0.547  1.986  4.386 -1.345 -1.744 -0.563 -0.616  …
2  2.276 -1.532 -1.022 -1.602 -1.220 -0.462 -1.196 -0.147 -0.950  1.560  …
3  1.940 -0.358 -1.211  0.383  0.051 -0.171 -0.109 -0.002  0.869 -0.002  …
4  1.081 -0.503  1.076 -0.543 -1.473 -1.065 -0.443 -0.143  1.660 -1.131  …

     V21    V22    V23    V24    V25    V26    V27    V28  scaled_amount  \
0 -0.336 -0.808 -0.056 -1.025 -0.370  0.205  0.243  0.086         -0.292
1 -1.377 -0.072 -0.198  1.015  1.011 -0.168  0.113  0.257          0.878
2 -0.193 -0.104  0.151 -0.811 -0.198 -0.128  0.014 -0.051          0.289
3  0.158  0.650  0.034  0.740  0.224 -0.196 -0.013 -0.057          0.113
4  0.224  0.821 -0.137  0.986  0.563 -0.574  0.090  0.052          0.641

   scaled_time
0        0.328
1       -0.689
2        0.875
3        0.616
4       -0.257

[5 rows x 30 columns]
```

[36]:
```
x = x.drop(['scaled_amount','scaled_time'],axis=1)
df_test = df_test.drop(['scaled_amount','scaled_time'],axis=1)
```

[37]:
```
x
```

[37]:
```
           V1     V2     V3     V4     V5     V6     V7     V8     V9    V10  \
0       1.044  0.319  1.046  2.806 -0.561 -0.368  0.033 -0.042 -0.323  0.499
1      -1.665  0.808  1.806  1.903 -0.822  0.935 -0.825  0.976  1.747 -0.659
2      -0.324  0.602  0.865 -2.138  0.295 -1.252  1.072 -0.335  1.071 -1.110
```

```
3      -0.258  1.218 -0.585 -0.875  1.222 -0.311  1.074 -0.161  0.201  0.154
4       2.142 -0.495 -1.937 -0.818 -0.025 -1.027 -0.152 -0.306 -0.869  0.429
…         …     …     …     …     …     …     …     …     …     …
227840 -1.994  1.735 -1.108 -2.672  1.605  3.042 -0.418  1.438  0.945  1.017
227841 -0.440  1.063  1.582 -0.030  0.041 -0.904  0.730 -0.108 -0.513 -0.332
227842  0.828 -2.649 -3.161  0.209 -0.561 -1.570  1.613 -0.930 -1.319  0.684
227843 -1.524 -6.287 -2.638  1.330 -1.672  1.958  1.359  0.082  0.753 -0.702
227844 -1.609  0.133  2.076 -1.937 -1.822 -0.430  0.247  0.684  1.177 -2.064

          …    V19    V20    V21    V22    V23    V24    V25    V26    V27  \
0         … -0.721 -0.085 -0.240 -0.680  0.085  0.685  0.319 -0.205  0.002
1         …  1.326 -0.374 -0.335 -0.511  0.036  0.148 -0.529 -0.567 -0.596
2         …  0.096 -0.040  0.012  0.353 -0.342 -0.146  0.094 -0.804  0.229
3         …  0.136  0.382 -0.425 -0.781  0.019  0.179 -0.316  0.097  0.270
4         …  1.065  0.107  0.010  0.022  0.079 -0.481  0.024 -0.279 -0.030
…       …    …     …     …     …     …     …     …     …     …
227840    … -0.575  0.776 -0.304 -0.708  0.047  1.008  0.234  0.769  0.698
227841    …  0.010  0.159 -0.216 -0.532 -0.025  0.383 -0.165  0.069  0.269
227842    … -0.911  0.893  0.350  0.002 -0.747  0.172  0.248  0.937 -0.258
227843    … -1.477  3.299  1.329  0.001 -1.360 -1.508 -1.184  0.578 -0.329
227844    … -1.255  0.186  0.465  1.017  0.173  0.570  0.505 -0.660  0.175

            V28
0         0.038
1        -0.220
2        -0.022
3        -0.021
4        -0.044
…           …
227840    0.355
227841    0.123
227842    0.038
227843    0.230
227844    0.092

[227845 rows x 28 columns]
```

[38]:
```
x.shape
```

[38]:
```
(227845, 28)
```

[39]:
```
y.shape
```

[39]:
```
(227845,)
```

[40]:
```
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(x, y)
```

```
clf.feature_importances_
```

[40]:
```
array([0.00000000e+00, 6.64079693e-04, 1.64016769e-02, 1.10726952e-02,
       1.12833261e-03, 8.12053307e-06, 1.54202198e-02, 2.80607340e-03,
       5.20390042e-02, 8.13971343e-02, 9.74799949e-02, 2.41933240e-01,
       0.00000000e+00, 1.11869504e-01, 0.00000000e+00, 5.89654101e-02,
       2.45464979e-01, 5.09558577e-02, 1.16757491e-03, 6.82818632e-04,
       2.84711088e-03, 4.82174136e-04, 0.00000000e+00, 0.00000000e+00,
       1.21449817e-04, 6.36458207e-03, 1.37594009e-05, 7.14208201e-04])
```

[41]:
```
xb = xgb.XGBRFRegressor()
xb.fit(x, y)
xb.feature_importances_
```

```
[13:31:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
```

[41]:
```
array([0.00687269, 0.0076968 , 0.01050094, 0.01611375, 0.00325623,
       0.01119005, 0.01446964, 0.0103796 , 0.03653619, 0.07195353,
       0.02403417, 0.1196127 , 0.        , 0.04612156, 0.        ,
       0.00892835, 0.5283889 , 0.        , 0.00280011, 0.01193771,
       0.00858623, 0.        , 0.        , 0.        , 0.02091379,
       0.02481328, 0.01489371, 0.        ], dtype=float32)
```

[42]:
```
def roc_curve_plots(y_test,y_predict_wrf,X_test,model):
    print(classification_report(y_test,y_predict_wrf),"\n")
    neigh_prob_linear=model.predict_proba(X_test)
    neigh_prob_linear1=neigh_prob_linear[:,1]
    fpr,tpr,thresh=roc_curve(y_test,neigh_prob_linear1)
    roc_auc_neigh=auc(fpr,tpr)

    plt.figure(dpi=80)
    plt.title("ROC Curve")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_neigh)
    plt.plot(fpr,fpr,'r--',color='red')
    plt.legend()
```

[43]:
```
def model_accuracies(model,x_feature,y_label,X_test,df_test_len):
    # X_train, X_test, y_train, y_test = train_test_split(x_feature, y_label,␣
 ↪test_size=0.33, random_state=42)
    print(x_feature.shape,y_label.shape,X_test.shape)
    y_pred = model.fit(x_feature, y_label).predict(X_test)
    print("df_test_len", df_test_len)
    cm_wrf = confusion_matrix(y_pred,y_label.iloc[:df_test_len])
```

```
    print("f1_score       : ", f1_score(y_label.iloc[:df_test_len], y_pred,␣
↪average="macro"))
    print("precision_score: ", precision_score(y_label.iloc[:df_test_len],␣
↪y_pred, average="macro"))
    print("recall_score   : ", recall_score(y_label.iloc[:df_test_len], y_pred,␣
↪average="macro"))
    print("\nAccuracy Score :",accuracy_score(y_pred,y_label.iloc[:
↪df_test_len]))
    roc_curve_plots(y_label.iloc[:df_test_len],y_pred,X_test,model)
```

1.2 Use techniques like undersampling or oversampling before running Naïve Bayes, Logistic Regression or SVM.

- ### Oversampling or undersampling can be used to tackle the class imbalance problem
- ### Oversampling increases the prior probability of imbalanced class and in case of other classifiers, error gets multiplied as the low-proportionate class is mimicked multiple times.

[44]:
```
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(x, y)
```

[45]:
```
print(X_ros.shape, y_ros.shape)
```

```
(454902, 28) (454902,)
```

[46]:
```
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(x, y)
```

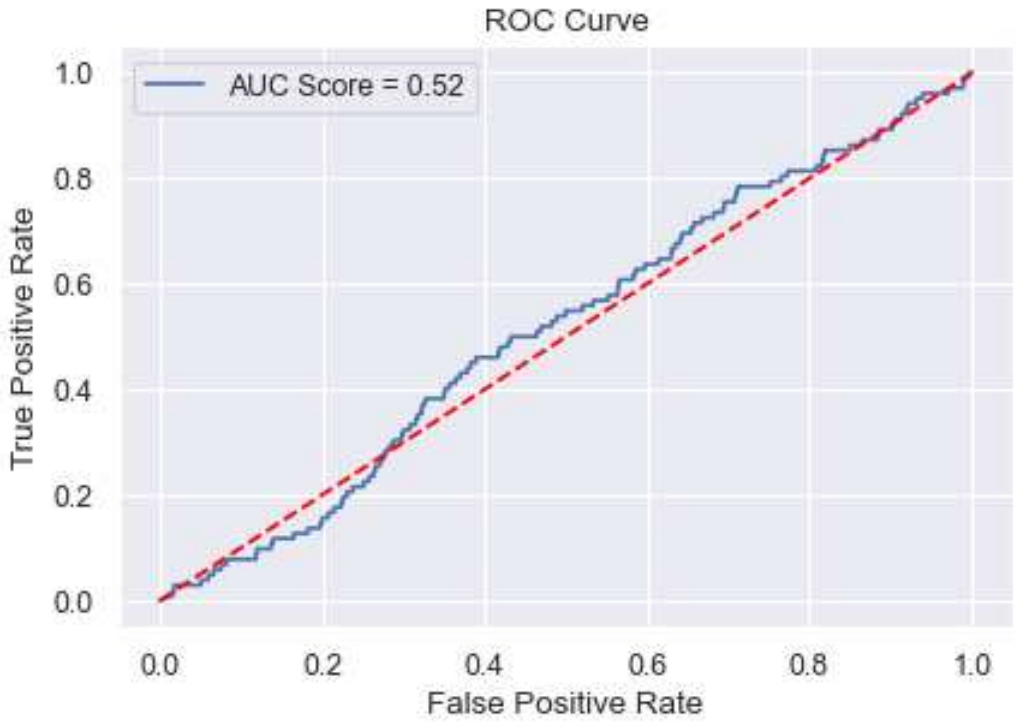
[47]:
```
print(X_rus.shape, y_rus.shape)
```

```
(788, 28) (788,)
```

[48]:
```
gnb = GaussianNB()
df_test_len = df_test.shape[0]
# print(df_test.sample(n=df_test_len))
model_accuracies(model = gnb, x_feature=X_ros,y_label= y_ros, X_test=df_test,␣
↪df_test_len=df_test_len)
```

```
(454902, 28) (454902,) (56962, 28)
df_test_len 56962
f1_score       :  0.49453055317909606
precision_score:  0.5000795832882818
recall_score   :  0.5011726428172394

Accuracy Score : 0.9712439872195499
              precision    recall  f1-score   support
```

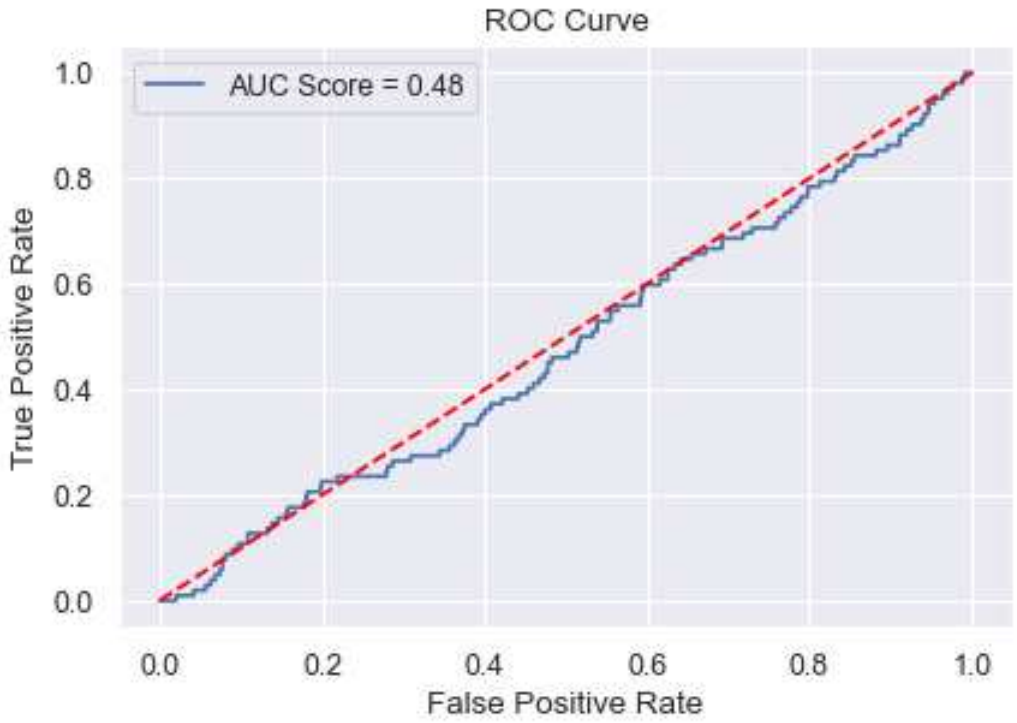
```
              0       1.00      0.97      0.99     56860
              1       0.00      0.03      0.00       102

    accuracy                            0.97     56962
   macro avg       0.50      0.50      0.49     56962
weighted avg       1.00      0.97      0.98     56962
```



[49]:
```
lr = LogisticRegression(class_weight ='balanced')
model_accuracies(model = lr, x_feature=X_ros,y_label=␣
 ↪y_ros,X_test=df_test,df_test_len=df_test.shape[0])
```

```
(454902, 28) (454902,) (56962, 28)
df_test_len 56962
f1_score        :  0.4942775108100917
precision_score:  0.49946433771292903
recall_score    :  0.49310104625740553

Accuracy Score : 0.9746673220743653
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56860
           1       0.00      0.01      0.00       102
```

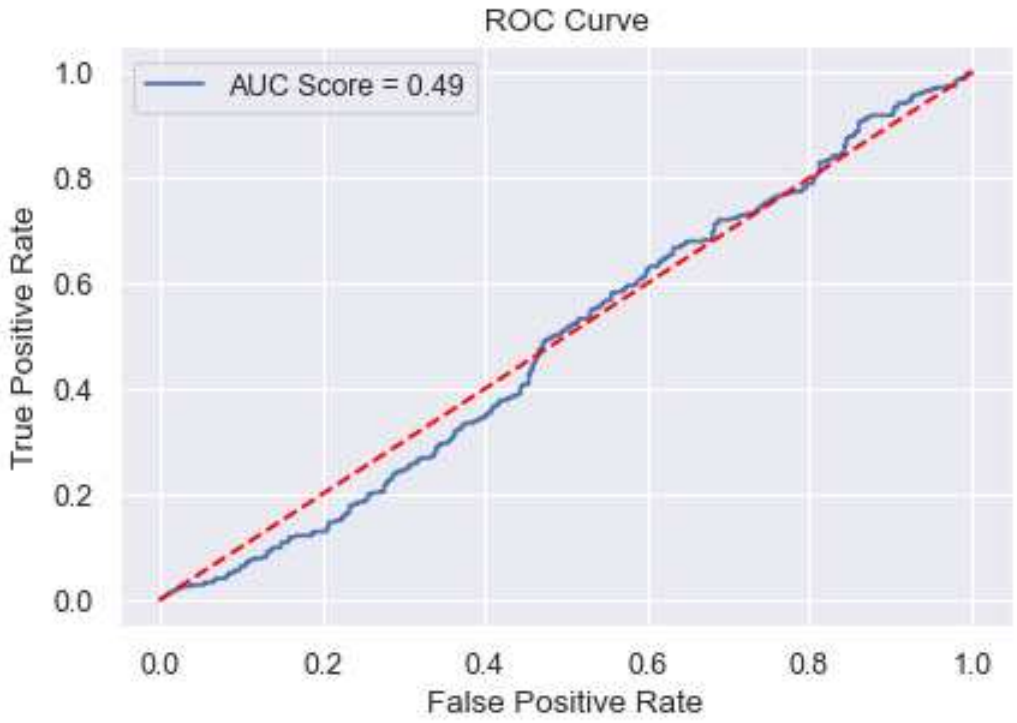
```
    accuracy                           0.97     56962
   macro avg       0.50      0.49      0.49     56962
weighted avg       1.00      0.97      0.99     56962
```



[50]:
```
gnb = GaussianNB()
model_accuracies(model = gnb, x_feature=X_rus,y_label= y_rus, X_test=df_test.
 ↪iloc[:788],df_test_len = 788)
```

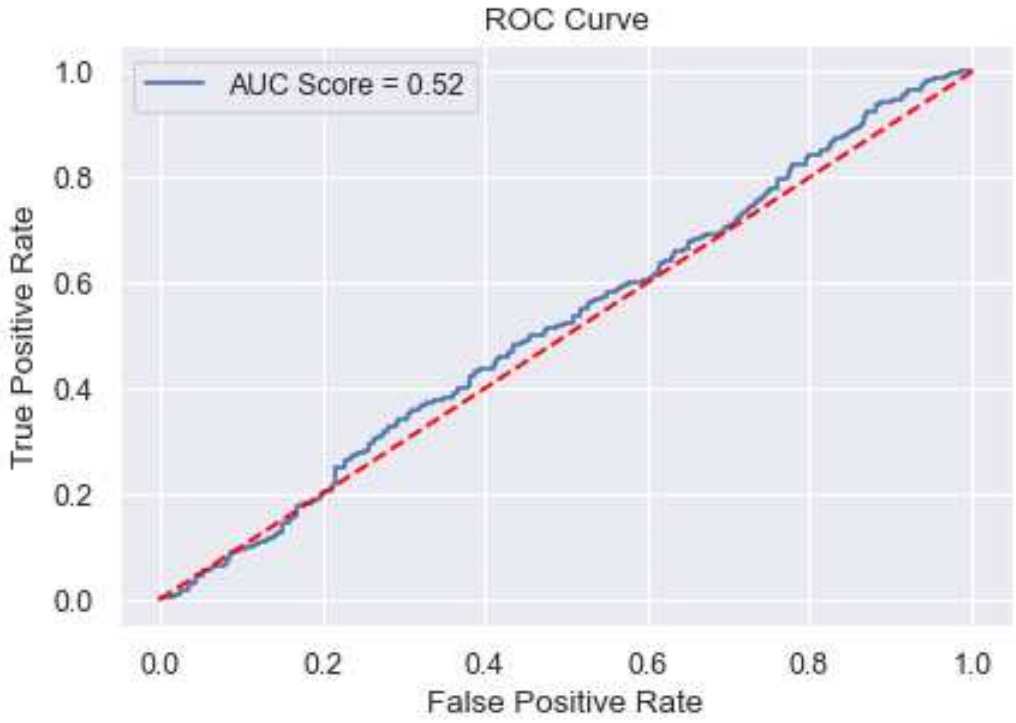
```
(788, 28) (788,) (788, 28)
df_test_len 788
f1_score       :  0.35340099330598146
precision_score:  0.45206112814522803
recall_score   :  0.4936548223350254

Accuracy Score : 0.4936548223350254
              precision    recall  f1-score   support

           0       0.50      0.96      0.65       394
           1       0.41      0.03      0.05       394

    accuracy                           0.49       788
```

```
   macro avg       0.45      0.49      0.35       788
weighted avg       0.45      0.49      0.35       788
```



[51]:
```
lr = LogisticRegression(class_weight ='balanced')
model_accuracies(model = lr, x_feature=X_rus,y_label= y_rus, X_test=df_test.
 ↪iloc[:788],df_test_len = 788)
```

```
(788, 28) (788,) (788, 28)
df_test_len 788
f1_score       :  0.3699581963553715
precision_score:  0.5
recall_score   :  0.5

Accuracy Score : 0.5
              precision    recall  f1-score   support

           0       0.50      0.95      0.66       394
           1       0.50      0.05      0.08       394

    accuracy                           0.50       788
   macro avg       0.50      0.50      0.37       788
weighted avg       0.50      0.50      0.37       788
```



1.3 Following are the matrices for evaluating the model performance: Precision, Recall, F1-Score, AUC-ROC curve. Use F1-Score as the evaluation criteria for this project.

- Accuracy can be used when the class distribution is similar while F1-score is a better metric when there are imbalanced classes as in the above case.
- Accuracy is used when the True Positives and True negatives are more important while F1-score is used when the False Negatives and False Positives are crucial
- F1 score for undersampled data for Model Logistic Regression and Nave bayes is more.

1.3.1 * Try out models like Naive Bayes, Logistic Regression or SVM. Find out which one performs the best

[52]:
```
def run_parllel_job_cross_val(pipeline,x,y):
    big_future_x = client.scatter(x)
    big_future_y = client.scatter(y)
    a = client.submit(cross_val_score,pipeline, big_future_x, big_future_y,␣
↪cv=5, scoring='f1', n_jobs=5)
```

```
    return a.result().mean()
```

[53]:
```
pipe_lr = Pipeline(steps=[('scaler', StandardScaler()), ('lr',␣
 ↪LogisticRegression())])
output = run_parllel_job_cross_val(pipe_lr,x,y)
output
```

[53]: 0.7333977375000338

[54]:
```
pipe_svc = Pipeline(steps=[('scaler', StandardScaler()), ('svc', SVC())])
output = run_parllel_job_cross_val(pipe_svc,x,y)
output
```

[54]: 0.7736833523239918

[55]:
```
pipe_nb = Pipeline(steps=[('nb', GaussianNB())])
output = run_parllel_job_cross_val(pipe_nb,x,y)
output
```

[55]: 0.11600483849746838

[56]:
```
# scoring = 'f1'
# models = []
# results = []
# names = []
# models.append(('SVM', SVC()))
# models.append(('LR', LogisticRegression()))
# models.append(('LDA', LinearDiscriminantAnalysis()))
# models.append(('KNN', KNeighborsClassifier()))
# models.append(('CART', DecisionTreeClassifier()))
# models.append(('NB', GaussianNB()))

# for name, model in models:
#     start_time = time.time()
#     kfold = KFold(n_splits=10, random_state=42, shuffle=True)
#     cv_results = cross_val_score(model, x, y, cv=kfold, scoring=scoring)
#     elapsed_time = time.time() - start_time
#     results.append(cv_results)
#     names.append(name)
#     msg = "{:3.2f} ({:3.2f})  Time elapsed: {:6.2f}".format(cv_results.
 ↪mean(), cv_results.std(), elapsed_time)
#     msg = "%s "%(name) + msg
#     print(msg)
```

[57]:
```
# pipe = Pipeline([('scaler', StandardScaler()),('classifier',␣
 ↪RandomForestClassifier())])
# estimators_range  =range(5,20)
```

```
# # Create space of candidate learning algorithms and their hyperparameters
# search_space = [{
#                       'scaler':[StandardScaler(),MinMaxScaler(), Normalizer(),␣
 ↪MaxAbsScaler()],
#                       'classifier': [LogisticRegression()],
#                       'classifier__penalty' : ['l2'],'classifier__dual':
 ↪[False], 'classifier__solver':['newton-cg', 'saga'],
#                       'classifier__C': np.logspace(0, 4, 10)
#                 },
#                 {
#                       'scaler':[StandardScaler(),MinMaxScaler(), Normalizer(),␣
 ↪MaxAbsScaler()],
#                       'classifier': [RandomForestClassifier()],
#                       'classifier__n_estimators':␣
 ↪[10,100,200],'classifier__criterion':['gini', 'entropy'],
#                       'classifier__max_features': [1, 2, 3]
#                 },
#                 {
#                       'scaler':[StandardScaler(),MinMaxScaler(), Normalizer(),␣
 ↪MaxAbsScaler()],
#                       'classifier': [xgb.XGBRFRegressor()],
#                       'classifier__learning_rate ': [0.1],
#                       'classifier__max_depth' : [5],␣
 ↪'classifier__n_estimators' : estimators_range, 'classifier__booster':
 ↪['gbtree'], 'classifier__verbosity' : [0]
#                 },
#                 {
#                     'scaler':[StandardScaler(),MinMaxScaler(), Normalizer(),␣
 ↪MaxAbsScaler()],
#                       'classifier': [GaussianNB()],
#                 },
#                ]
```

[58]:
```
# clf = GridSearchCV(pipe, search_space, cv=5, verbose=0, n_jobs=5)
# best_model = clf.fit(x, y)

# print("best_score_ :",best_model.best_score_,"\n")
# best_model.best_estimator_.get_params()
```

1.3.2 * Ensemble Learning

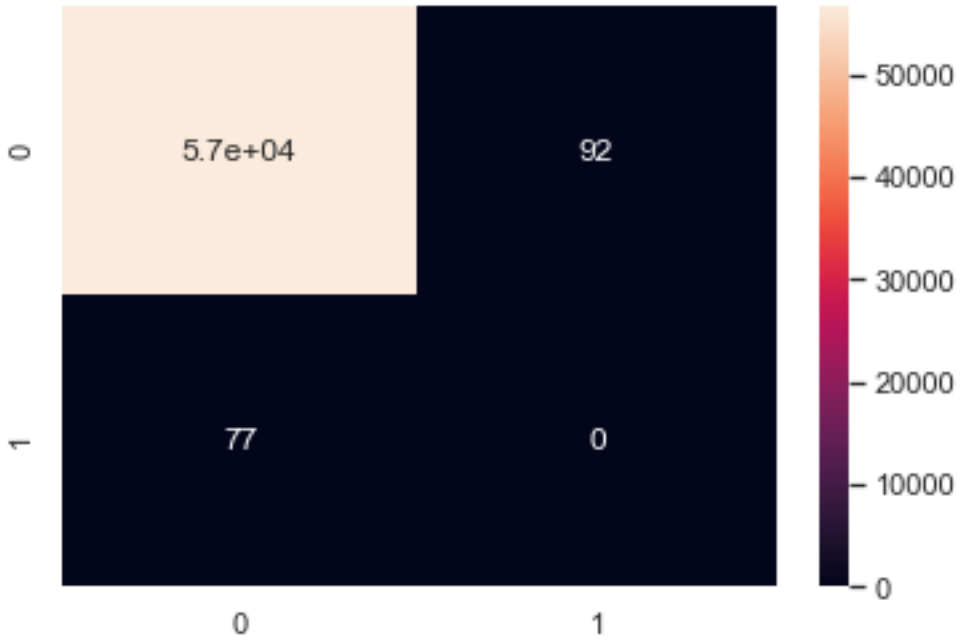
RandomForest

[59]:
```
wrf = RandomForestClassifier(class_weight='balanced_subsample',␣
 ↪random_state=42, n_jobs=5)
wrf.fit(x, y)
```

[59]:
```
RandomForestClassifier(class_weight='balanced_subsample', n_jobs=5,
                       random_state=42)
```

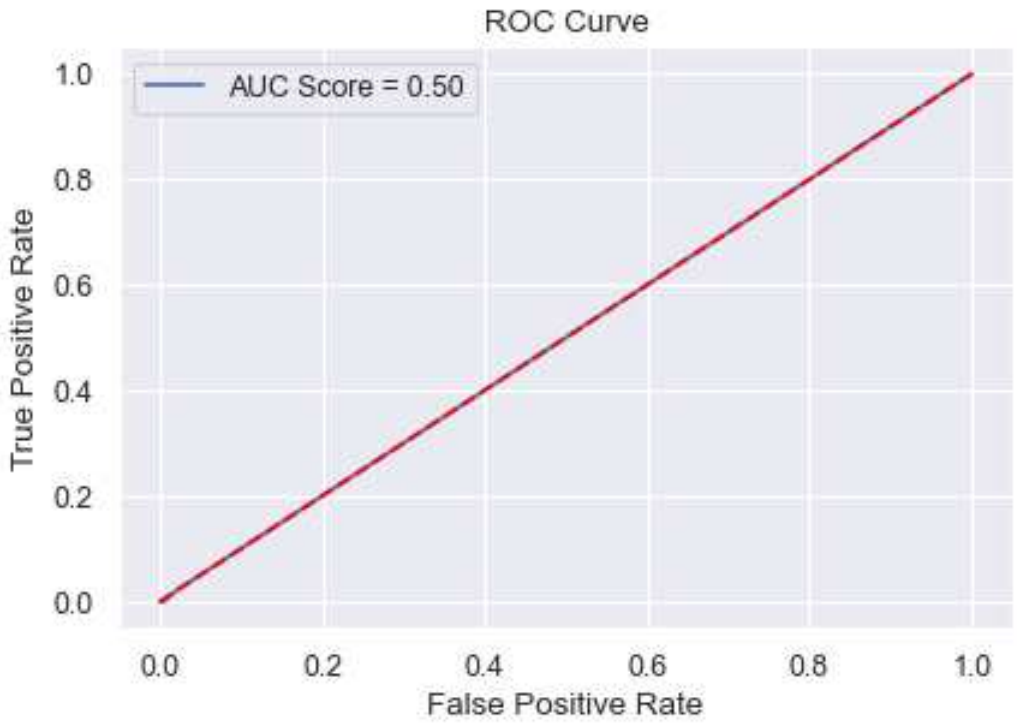
[60]:
```
y_predict_wrf = wrf.predict(df_test)
cm_wrf = confusion_matrix(y_predict_wrf,y.sample(n = 56962))
sns.heatmap(cm_wrf,annot=True)
print("Accuracy Score :",accuracy_score(y_predict_wrf,y.sample(n = 56962)))
```

```
Accuracy Score : 0.9969979986657772
```



[61]:
```
roc_curve_plots(y.sample(n = 56962),y_predict_wrf,df_test,wrf)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.00      0.00      0.00        99

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962
```



[62]:
```
y_label = y.iloc[:df_test_len]
```

[63]:
```
print("f1_score       : ", f1_score(y_label, y_predict_wrf, average="macro"))
print("precision_score: ", precision_score(y_label, y_predict_wrf,␣
 ↪average="macro"))
print("recall_score   : ", recall_score(y_label, y_predict_wrf,␣
 ↪average="macro"))
```

```
f1_score       :  0.4992131522264715
precision_score:  0.49910345433769887
recall_score   :  0.49932289834681676
```

XGBoost

*** Suppose, the dataset has 90 observations of negative class and 10 observations of positive class, then ideal value of scale_pos_weight should be 9.**

[64]:
```
y.value_counts(normalize=True) * 100
```

[64]:
```
0    99.827
1     0.173
Name: Class, dtype: float64
```

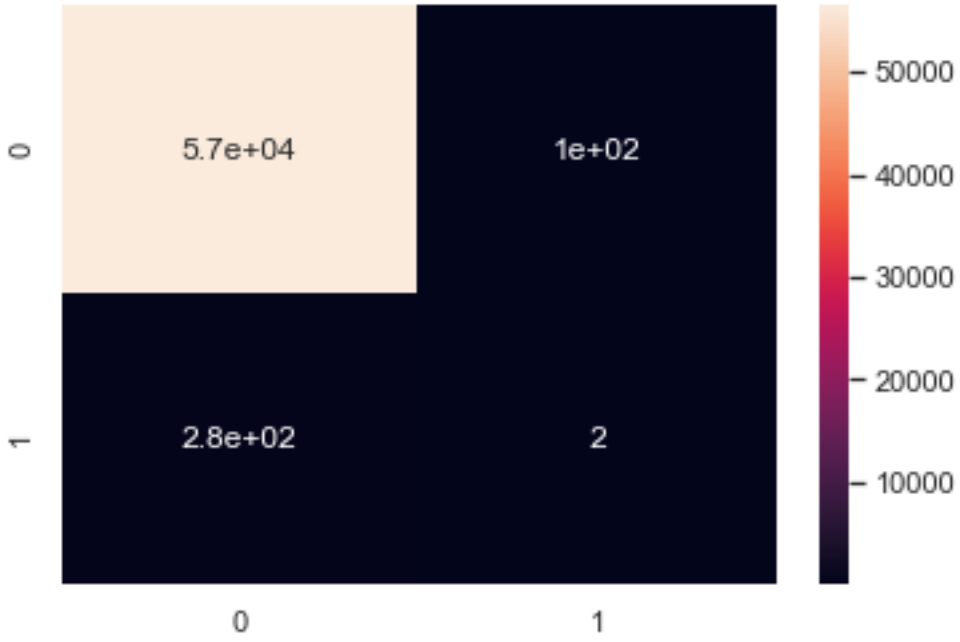
[65]:
```
xb = xgb.XGBClassifier(learning_rate=0.001, max_depth=1, n_estimators=1,␣
↪scale_pos_weight=99.827, n_jobs=5)
```

[66]:
```
xb.fit(x, y,eval_metric='logloss',verbose=True)
```

[66]:
```
XGBClassifier(learning_rate=0.001, max_depth=1, n_estimators=1, n_jobs=5,
              scale_pos_weight=99.827)
```

[67]:
```
xgb_predict=xb.predict(df_test)
cm_xgb = confusion_matrix(xgb_predict,y.sample(n = 56962))
sns.heatmap(cm_xgb,annot=True)
print("Accuracy Score :",accuracy_score(xgb_predict,y.sample(n = 56962)))
print("f1_score       : ", f1_score(y.sample(n = 56962), xgb_predict,␣
↪average="macro"))
```

```
Accuracy Score : 0.9935395526842457
f1_score       :  0.5009670259987318
```

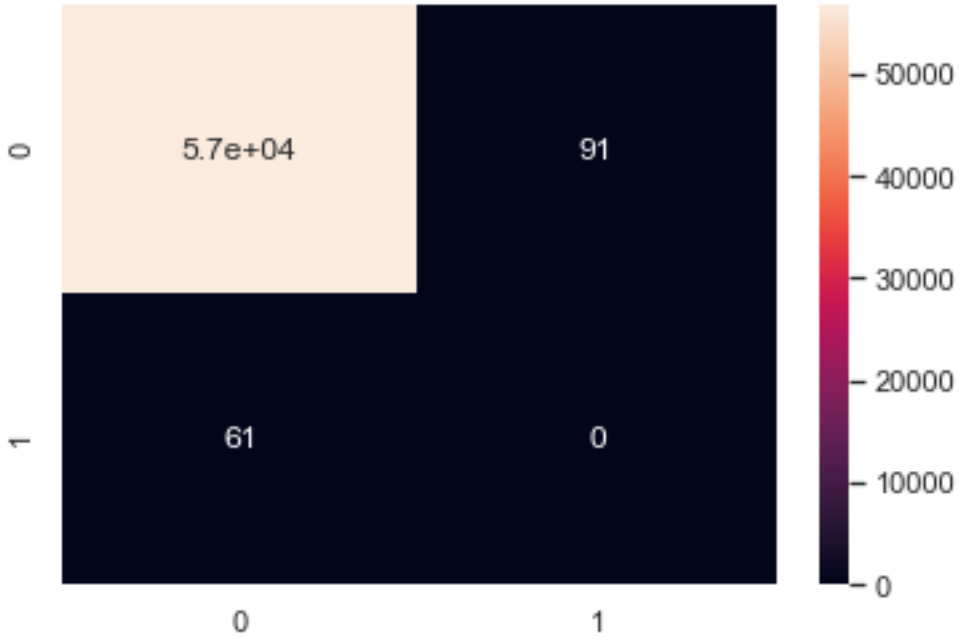


BaggingClassifier

[68]:
```
BC_clf = BaggingClassifier(base_estimator=SVC(),
                           n_estimators=10, random_state=0, n_jobs=5).fit(x, y)
BC_pred = BC_clf.predict(df_test)
cm_bc = confusion_matrix(BC_pred,y.sample(n = 56962))
sns.heatmap(cm_bc,annot=True)
```

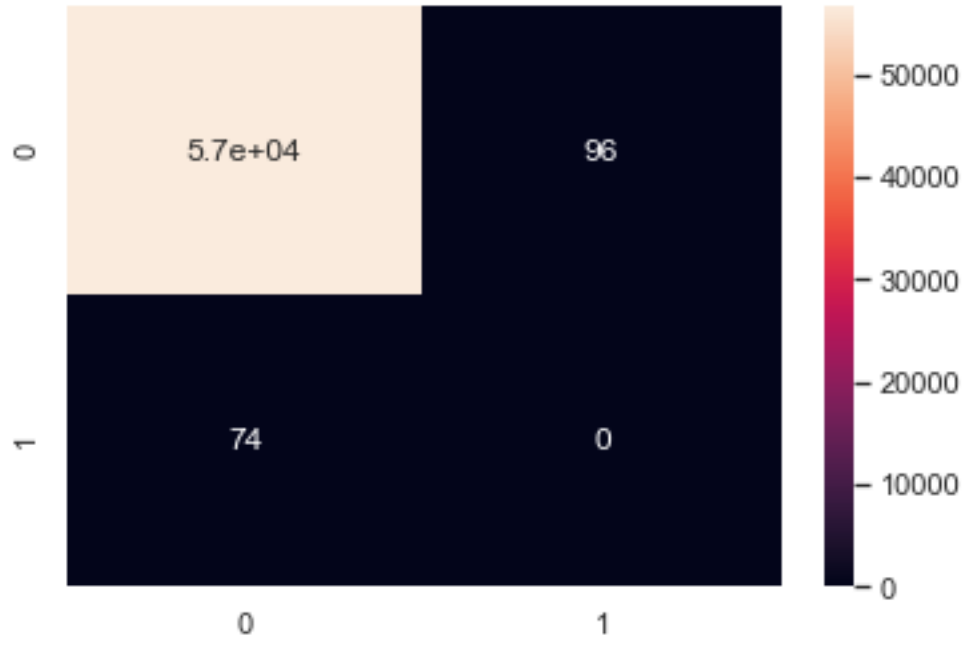
```
print("Accuracy Score :",accuracy_score(BC_pred,y.sample(n = 56962)))
print("f1_score       : ", f1_score(y.sample(n = 56962), BC_pred,␣
↪average="macro"))
```

```
Accuracy Score : 0.997559776693234
f1_score       :  0.4992967898456454
```



[69]:
```
BC_RFC =␣
↪BaggingClassifier(base_estimator=RandomForestClassifier(class_weight='balanced_subsample'),
                        n_estimators=10, random_state=0, n_jobs=5).fit(x, y)
BC_rfc_pred = BC_RFC.predict(df_test)
cm_BC_rf = confusion_matrix(BC_rfc_pred,y.sample(n = 56962))
sns.heatmap(cm_BC_rf,annot=True)
print("Accuracy Score :",accuracy_score(BC_rfc_pred,y.sample(n = 56962)))
print("f1_score       : ", f1_score(y.sample(n = 56962), BC_rfc_pred,␣
↪average="macro"))
```

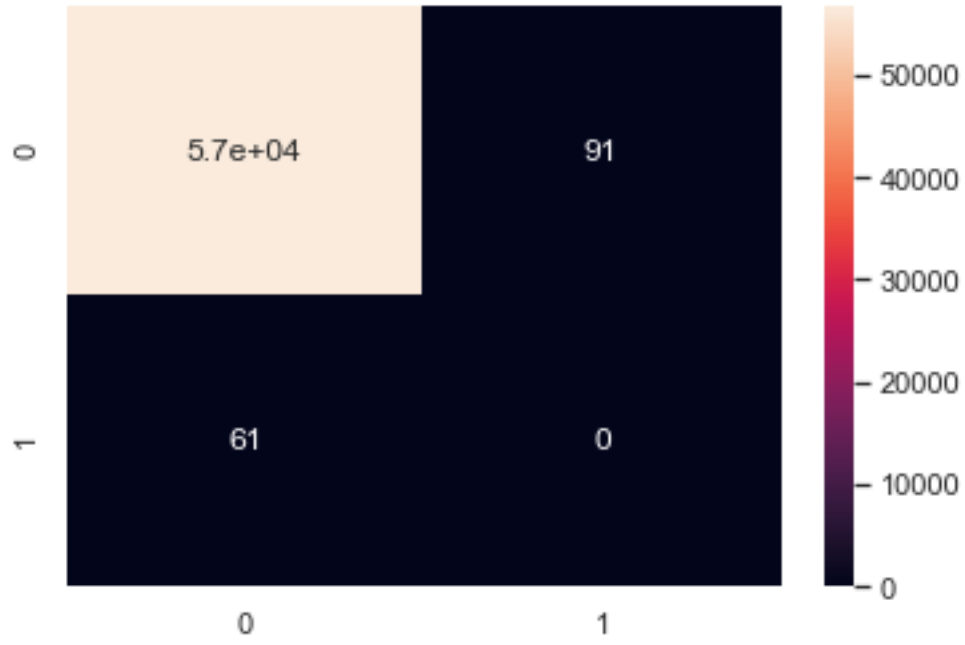
```
Accuracy Score : 0.9969277764123451
f1_score       :  0.4992703809841946
```



GradientBoostingClassifier

[70]:
```
GBC = GradientBoostingClassifier(random_state=0, loss='deviance',␣
 ↪learning_rate=0.01, criterion='mse')
GBC.fit(x, y)
BC_gbc_pred = GBC.predict(df_test)
cm_gbc = confusion_matrix(BC_gbc_pred,y.sample(n = 56962))
sns.heatmap(cm_bc,annot=True)
print("Accuracy Score :",accuracy_score(BC_gbc_pred,y.sample(n = 56962)))
print("f1_score       : ", f1_score(y.sample(n = 56962), BC_gbc_pred,␣
 ↪average="macro"))
```

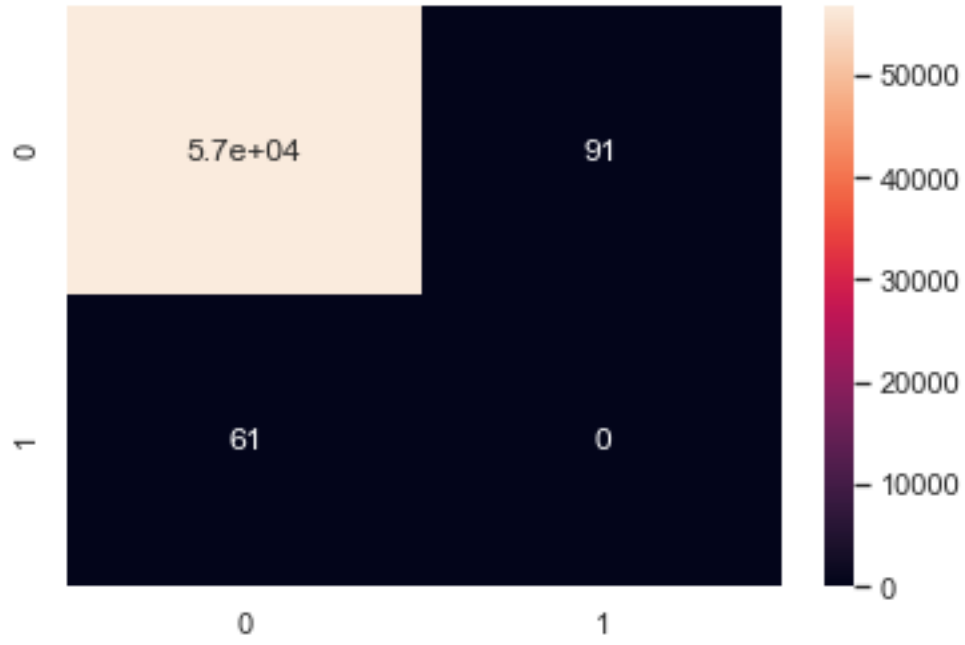
```
Accuracy Score : 0.9971033320459254
f1_score       :  0.4992615774113014
```



ExtraTreesClassifier

[71]:
```
GBC = ExtraTreesClassifier(n_estimators=100, random_state=0,␣
 ↪class_weight='balanced', n_jobs = 6)
GBC.fit(x, y)
BC_gbc_pred = GBC.predict(df_test)
cm_gbc = confusion_matrix(BC_gbc_pred,y.sample(n = 56962))
sns.heatmap(cm_bc,annot=True)
print("Accuracy Score :",accuracy_score(BC_gbc_pred,y.sample(n = 56962)))
print("f1_score       : ", f1_score(y.sample(n = 56962), BC_gbc_pred,␣
 ↪average="macro"))
```

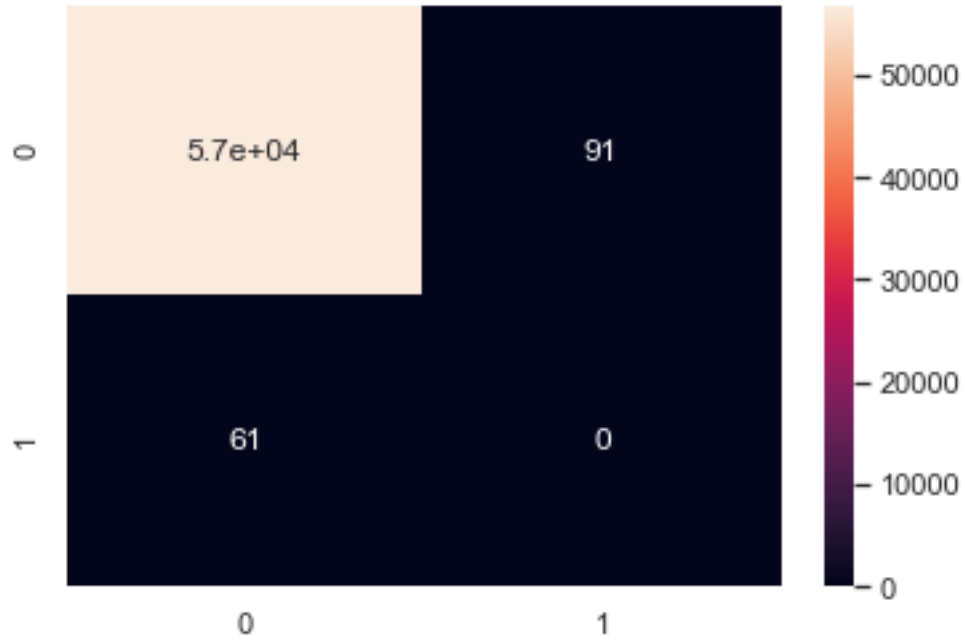
```
Accuracy Score : 0.996892665285629
f1_score       :  0.4992439693368029
```



AdaBoostClassifier

[72]:
```
GBC = AdaBoostClassifier(n_estimators=100, random_state=0)
GBC.fit(x, y)
BC_gbc_pred = GBC.predict(df_test)
cm_gbc = confusion_matrix(BC_gbc_pred,y.sample(n = 56962))
sns.heatmap(cm_bc,annot=True)
print("Accuracy Score :",accuracy_score(BC_gbc_pred,y.sample(n = 56962)))
print("f1_score       : ", f1_score(y.sample(n = 56962), BC_gbc_pred,␣
↪average="macro"))
```

```
Accuracy Score : 0.9968399985955549
f1_score       :  0.49924837147152157
```



1.3.3 ANN

[73]:
```
new_df.drop(['Time','Amount'], axis=1, inplace=True)
new_df.head()
```

[73]:
```
           V1     V2     V3     V4     V5     V6     V7     V8     V9    V10  \
189937 -0.235  0.355  1.972 -1.256 -0.681 -0.666  0.059 -0.003  1.122 -1.481
181016 -3.365  2.929 -5.661  3.891 -1.840 -1.801 -5.559  2.402 -2.849 -5.996
37300  -0.426  0.512  1.598 -0.496 -0.053 -0.313  0.415  0.157 -0.316 -0.512
24481   1.229 -0.665  0.450 -1.418 -0.929 -0.404 -0.569 -0.056  1.892 -1.202
94916   1.209 -0.142  0.167  0.155 -0.269 -0.434 -0.163 -0.153  1.782 -0.559

        …    V22    V23    V24        V25    V26    V27    V28  Class  \
189937  …  0.912 -0.286  0.451  1.883e-01 -0.532  0.123  0.040      1
181016  … -0.103 -0.606 -0.743  9.632e-02 -0.135  1.239  0.100      1
37300   …  0.174 -0.125 -0.003 -8.394e-04  0.203 -0.040 -0.006      0
24481   …  0.469 -0.263 -0.347  6.010e-01  0.158  0.039  0.026      0
94916   … -1.465  0.040 -0.694  8.065e-02  0.494 -0.108  0.008      0

        scaled_amount  scaled_time
189937         -0.172       -0.372
181016         -0.172        0.088
37300           0.141       -0.075
24481           0.393       -0.845
```

```
94916           0.578       -0.750

[5 rows x 31 columns]
```

[74]:
```
new_df.head()
```

[74]:
```
           V1     V2     V3     V4     V5     V6     V7     V8     V9    V10  \
189937 -0.235  0.355  1.972 -1.256 -0.681 -0.666  0.059 -0.003  1.122 -1.481
181016 -3.365  2.929 -5.661  3.891 -1.840 -1.801 -5.559  2.402 -2.849 -5.996
37300  -0.426  0.512  1.598 -0.496 -0.053 -0.313  0.415  0.157 -0.316 -0.512
24481   1.229 -0.665  0.450 -1.418 -0.929 -0.404 -0.569 -0.056  1.892 -1.202
94916   1.209 -0.142  0.167  0.155 -0.269 -0.434 -0.163 -0.153  1.782 -0.559

        …    V22    V23    V24        V25    V26    V27    V28  Class  \
189937  …  0.912 -0.286  0.451  1.883e-01 -0.532  0.123  0.040      1
181016  … -0.103 -0.606 -0.743  9.632e-02 -0.135  1.239  0.100      1
37300   …  0.174 -0.125 -0.003 -8.394e-04  0.203 -0.040 -0.006      0
24481   …  0.469 -0.263 -0.347  6.010e-01  0.158  0.039  0.026      0
94916   … -1.465  0.040 -0.694  8.065e-02  0.494 -0.108  0.008      0

        scaled_amount  scaled_time
189937         -0.172       -0.372
181016         -0.172        0.088
37300           0.141       -0.075
24481           0.393       -0.845
94916           0.578       -0.750

[5 rows x 31 columns]
```

[75]:
```
sns.pairplot(new_df, diag_kind="kde")
```

[75]:
```
<seaborn.axisgrid.PairGrid at 0x15b5c0b75c8>
```

[76]:
```
new_df.shape
```

[76]:
```
(886, 31)
```

[77]:
```
y_label = new_df.pop('Class')
x_feat = new_df
```

[78]:
```
y_label.head()
```

[78]:
```
189937    1
181016    1
37300     0
24481     0
```

```
94916     0
Name: Class, dtype: int64
```

[79]:
```
x_feat.head()
```

[79]:
```
           V1     V2     V3     V4     V5     V6     V7     V8     V9    V10  \
189937 -0.235  0.355  1.972 -1.256 -0.681 -0.666  0.059 -0.003  1.122 -1.481
181016 -3.365  2.929 -5.661  3.891 -1.840 -1.801 -5.559  2.402 -2.849 -5.996
37300  -0.426  0.512  1.598 -0.496 -0.053 -0.313  0.415  0.157 -0.316 -0.512
24481   1.229 -0.665  0.450 -1.418 -0.929 -0.404 -0.569 -0.056  1.892 -1.202
94916   1.209 -0.142  0.167  0.155 -0.269 -0.434 -0.163 -0.153  1.782 -0.559

        …    V21    V22    V23    V24        V25    V26    V27    V28  \
189937  …  0.221  0.912 -0.286  0.451  1.883e-01 -0.532  0.123  0.040
181016  …  0.875 -0.103 -0.606 -0.743  9.632e-02 -0.135  1.239  0.100
37300   …  0.119  0.174 -0.125 -0.003 -8.394e-04  0.203 -0.040 -0.006
24481   …  0.098  0.469 -0.263 -0.347  6.010e-01  0.158  0.039  0.026
94916   … -0.511 -1.465  0.040 -0.694  8.065e-02  0.494 -0.108  0.008

        scaled_amount  scaled_time
189937         -0.172       -0.372
181016         -0.172        0.088
37300           0.141       -0.075
24481           0.393       -0.845
94916           0.578       -0.750

[5 rows x 30 columns]
```

[80]:
```
x_train, x_test, y_train, y_test = train_test_split(x_feat, y_label,␣
↪test_size=0.30, random_state=42, shuffle=True)
```

[81]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

[81]:
```
((620, 30), (266, 30), (620,), (266,))
```

a) Fine-tune number of layers

b) Number of Neurons in each layers

Sigmoid is used for binary classification neural network. Because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. The function is differentiable.

ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. ReLU provides just enough non-linearity so that it is nearly as simple as a linear activation, but this non-linearity opens the door for extremely complex representations. Because unlike in the linear case, the more you stack non-linear ReLUs, the more it becomes non-linear.

[82]:
```
def create_model(optimizer='Adam',learn_rate=0.01, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(units =256,activation='relu',input_shape=(30,)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(Dense(units=256,kernel_initializer='normal',activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=128,kernel_initializer='normal',activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=64,kernel_initializer='normal',activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=32,kernel_initializer='normal',activation='relu'))
    model.add(Dense(units=16,kernel_initializer='normal',activation='relu'))
    model.add(Dense(units=1,activation='sigmoid'))
#     if not optimizer:
#         optimizer = tf.keras.optimizers.Adam(lr=learn_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy',tf.keras.metrics.AUC()])
    return model

create_model().summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               7936
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
```

```
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 118,529
Trainable params: 118,017
Non-trainable params: 512
_________________________________________________________________
```

[83]:
```
# tf.keras.metrics.AUC(),tf.keras.metrics.Precision(),tf.keras.metrics.Recall()
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
create_model().compile(optimizer=adam_optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[84]:
```
# print(os.getcwd())
logdir="logs/fit/"+ datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
history = create_model().
 ↪fit(x_train,y_train,epochs=20,batch_size=45,validation_data=(x_test, y_test))
```

```
Train on 620 samples, validate on 266 samples
Epoch 1/20
620/620 [==============================] - 1s 2ms/sample - loss: 0.6147 -
accuracy: 0.8242 - auc_2: 0.9225 - val_loss: 0.3690 - val_accuracy: 0.7143 -
val_auc_2: 0.9853
Epoch 2/20
620/620 [==============================] - 0s 110us/sample - loss: 0.2899 -
accuracy: 0.9274 - auc_2: 0.9482 - val_loss: 0.2641 - val_accuracy: 0.9098 -
val_auc_2: 0.9866
Epoch 3/20
620/620 [==============================] - 0s 105us/sample - loss: 0.1885 -
accuracy: 0.9323 - auc_2: 0.9752 - val_loss: 0.2079 - val_accuracy: 0.9361 -
val_auc_2: 0.9882
Epoch 4/20
620/620 [==============================] - 0s 102us/sample - loss: 0.1613 -
accuracy: 0.9339 - auc_2: 0.9815 - val_loss: 0.1754 - val_accuracy: 0.9436 -
val_auc_2: 0.9881
Epoch 5/20
620/620 [==============================] - 0s 106us/sample - loss: 0.1239 -
accuracy: 0.9500 - auc_2: 0.9899 - val_loss: 0.1617 - val_accuracy: 0.9474 -
val_auc_2: 0.9874
Epoch 6/20
620/620 [==============================] - 0s 108us/sample - loss: 0.1353 -
accuracy: 0.9516 - auc_2: 0.9848 - val_loss: 0.1541 - val_accuracy: 0.9474 -
val_auc_2: 0.9879
Epoch 7/20
```

```
620/620 [==============================] - 0s 106us/sample - loss: 0.1032 -
accuracy: 0.9597 - auc_2: 0.9917 - val_loss: 0.1439 - val_accuracy: 0.9511 -
val_auc_2: 0.9881
Epoch 8/20
620/620 [==============================] - 0s 102us/sample - loss: 0.1071 -
accuracy: 0.9661 - auc_2: 0.9900 - val_loss: 0.1471 - val_accuracy: 0.9436 -
val_auc_2: 0.9871
Epoch 9/20
620/620 [==============================] - 0s 102us/sample - loss: 0.0959 -
accuracy: 0.9661 - auc_2: 0.9933 - val_loss: 0.1511 - val_accuracy: 0.9361 -
val_auc_2: 0.9884
Epoch 10/20
620/620 [==============================] - 0s 102us/sample - loss: 0.0909 -
accuracy: 0.9629 - auc_2: 0.9933 - val_loss: 0.1454 - val_accuracy: 0.9436 -
val_auc_2: 0.9881
Epoch 11/20
620/620 [==============================] - 0s 102us/sample - loss: 0.0736 -
accuracy: 0.9694 - auc_2: 0.9965 - val_loss: 0.1503 - val_accuracy: 0.9361 -
val_auc_2: 0.9879
Epoch 12/20
620/620 [==============================] - 0s 103us/sample - loss: 0.0547 -
accuracy: 0.9806 - auc_2: 0.9984 - val_loss: 0.1576 - val_accuracy: 0.9436 -
val_auc_2: 0.9863
Epoch 13/20
620/620 [==============================] - 0s 105us/sample - loss: 0.0674 -
accuracy: 0.9774 - auc_2: 0.9970 - val_loss: 0.1420 - val_accuracy: 0.9436 -
val_auc_2: 0.9861
Epoch 14/20
620/620 [==============================] - 0s 163us/sample - loss: 0.0522 -
accuracy: 0.9806 - auc_2: 0.9987 - val_loss: 0.1971 - val_accuracy: 0.9286 -
val_auc_2: 0.9888
Epoch 15/20
620/620 [==============================] - 0s 103us/sample - loss: 0.0358 -
accuracy: 0.9887 - auc_2: 0.9993 - val_loss: 0.1831 - val_accuracy: 0.9436 -
val_auc_2: 0.9881
Epoch 16/20
620/620 [==============================] - 0s 100us/sample - loss: 0.0489 -
accuracy: 0.9903 - auc_2: 0.9975 - val_loss: 0.2090 - val_accuracy: 0.9398 -
val_auc_2: 0.9832
Epoch 17/20
620/620 [==============================] - 0s 102us/sample - loss: 0.0368 -
accuracy: 0.9855 - auc_2: 0.9993 - val_loss: 0.2075 - val_accuracy: 0.9398 -
val_auc_2: 0.9840
Epoch 18/20
620/620 [==============================] - 0s 102us/sample - loss: 0.0403 -
accuracy: 0.9806 - auc_2: 0.9991 - val_loss: 0.2232 - val_accuracy: 0.9436 -
val_auc_2: 0.9816
Epoch 19/20
```

```
620/620 [==============================] - 0s 98us/sample - loss: 0.0421 -
accuracy: 0.9871 - auc_2: 0.9976 - val_loss: 0.1836 - val_accuracy: 0.9511 -
val_auc_2: 0.9827
Epoch 20/20
620/620 [==============================] - 0s 102us/sample - loss: 0.0281 -
accuracy: 0.9887 - auc_2: 0.9997 - val_loss: 0.2152 - val_accuracy: 0.9436 -
val_auc_2: 0.9835
```

[85]:
```
df_test = pd.read_csv('Financial/test_data.csv')
std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df_test['scaled_amount'] = rob_scaler.fit_transform(df_test['Amount'].values.
 ↪reshape(-1,1))
df_test['scaled_time'] = rob_scaler.fit_transform(df_test['Time'].values.
 ↪reshape(-1,1))

df_test.drop(['Time','Amount'], axis=1, inplace=True)
df_test.head()
```

[85]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	…	\
0	0.115	0.796	-0.150	-0.823	0.879	-0.553	0.939	-0.109	0.111	-0.391	…	
1	-0.039	0.496	-0.811	0.547	1.986	4.386	-1.345	-1.744	-0.563	-0.616	…	
2	2.276	-1.532	-1.022	-1.602	-1.220	-0.462	-1.196	-0.147	-0.950	1.560	…	
3	1.940	-0.358	-1.211	0.383	0.051	-0.171	-0.109	-0.002	0.869	-0.002	…	
4	1.081	-0.503	1.076	-0.543	-1.473	-1.065	-0.443	-0.143	1.660	-1.131	…	

	V21	V22	V23	V24	V25	V26	V27	V28	scaled_amount	\
0	-0.336	-0.808	-0.056	-1.025	-0.370	0.205	0.243	0.086	-0.292	
1	-1.377	-0.072	-0.198	1.015	1.011	-0.168	0.113	0.257	0.878	
2	-0.193	-0.104	0.151	-0.811	-0.198	-0.128	0.014	-0.051	0.289	
3	0.158	0.650	0.034	0.740	0.224	-0.196	-0.013	-0.057	0.113	
4	0.224	0.821	-0.137	0.986	0.563	-0.574	0.090	0.052	0.641	

	scaled_time
0	0.328
1	-0.689
2	0.875
3	0.616
4	-0.257

[5 rows x 30 columns]

[86]:
```
# df_train_hidden
std_scaler = StandardScaler()
rob_scaler = RobustScaler()
```

```
df_train_hidden['scaled_amount'] = rob_scaler.
 ↪fit_transform(df_train_hidden['Amount'].values.reshape(-1,1))
df_train_hidden['scaled_time'] = rob_scaler.
 ↪fit_transform(df_train_hidden['Time'].values.reshape(-1,1))

df_train_hidden.drop(['Time','Amount'], axis=1, inplace=True)
df_train_hidden.head()
```

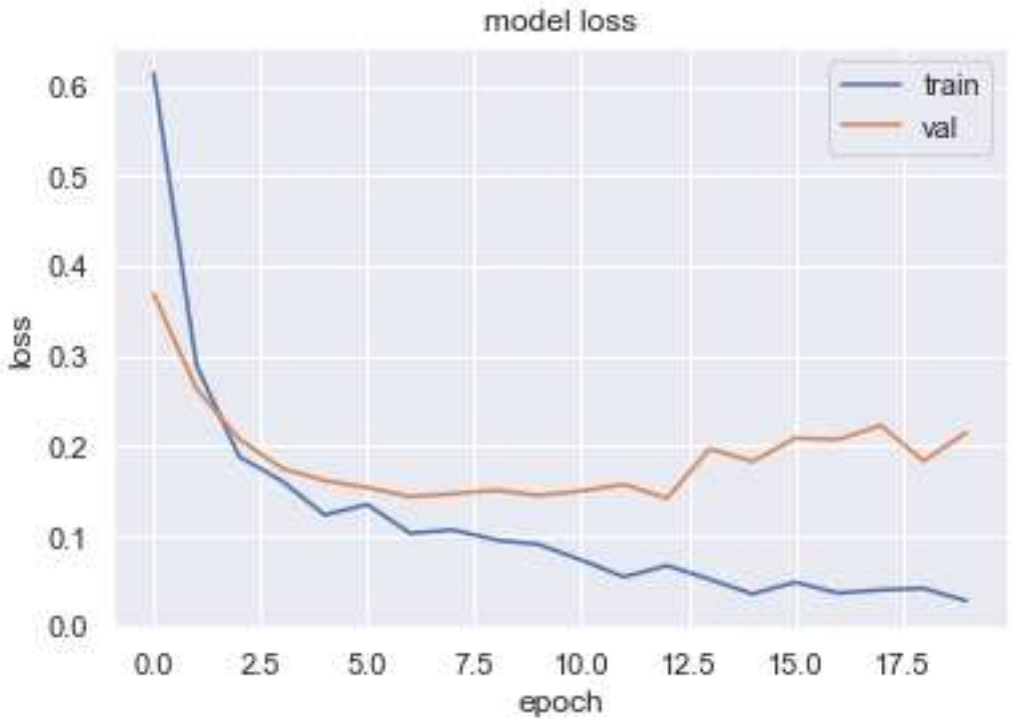
[86]:
```
      V1     V2     V3     V4     V5     V6     V7     V8     V9    V10  …  \
0  0.115  0.796 -0.150 -0.823  0.879 -0.553  0.939 -0.109  0.111 -0.391  …
1 -0.039  0.496 -0.811  0.547  1.986  4.386 -1.345 -1.744 -0.563 -0.616  …
2  2.276 -1.532 -1.022 -1.602 -1.220 -0.462 -1.196 -0.147 -0.950  1.560  …
3  1.940 -0.358 -1.211  0.383  0.051 -0.171 -0.109 -0.002  0.869 -0.002  …
4  1.081 -0.503  1.076 -0.543 -1.473 -1.065 -0.443 -0.143  1.660 -1.131  …

     V22    V23    V24    V25    V26    V27    V28  Class  scaled_amount  \
0 -0.808 -0.056 -1.025 -0.370  0.205  0.243  0.086      0         -0.292
1 -0.072 -0.198  1.015  1.011 -0.168  0.113  0.257      0          0.878
2 -0.104  0.151 -0.811 -0.198 -0.128  0.014 -0.051      0          0.289
3  0.650  0.034  0.740  0.224 -0.196 -0.013 -0.057      0          0.113
4  0.821 -0.137  0.986  0.563 -0.574  0.090  0.052      0          0.641

   scaled_time
0        0.328
1       -0.689
2        0.875
3        0.616
4       -0.257

[5 rows x 31 columns]
```

1.3.4 Predictions

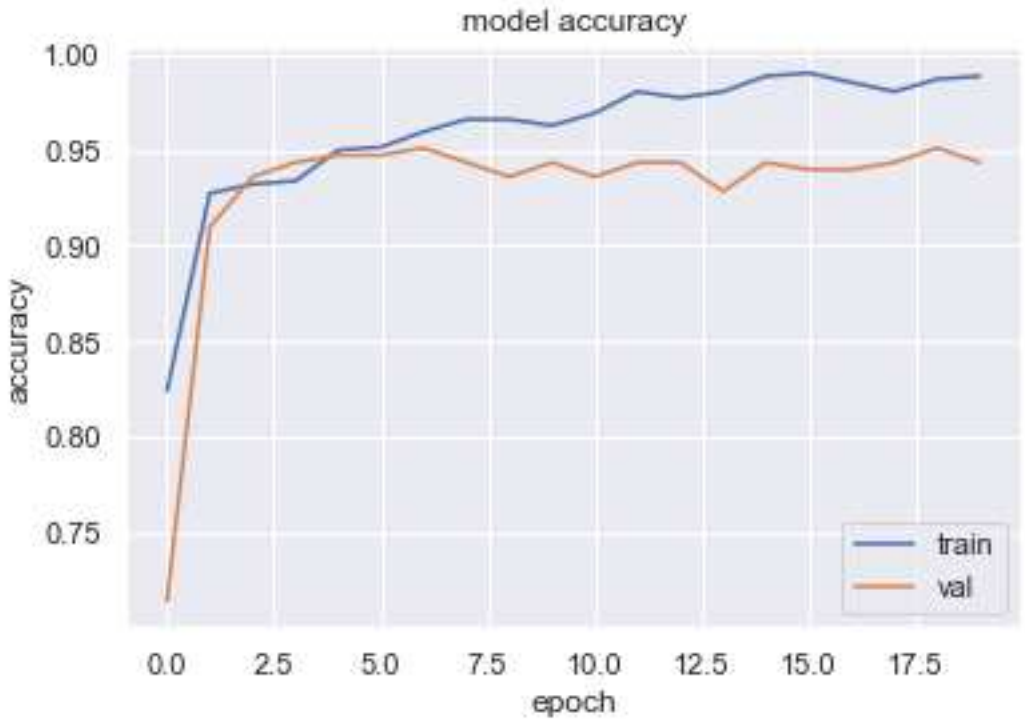
[87]:
```
ann_predictions = create_model().predict(df_test)
```

[88]:
```
df_train_hidden_pred = df_train_hidden.drop(['Class'], axis=1)
ann_predictions_hidden = create_model().predict(df_train_hidden_pred)
```

[89]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```



[90]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```



[91]:
```
df_test.head()
```

[91]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	…	\
0	0.115	0.796	-0.150	-0.823	0.879	-0.553	0.939	-0.109	0.111	-0.391	…	
1	-0.039	0.496	-0.811	0.547	1.986	4.386	-1.345	-1.744	-0.563	-0.616	…	
2	2.276	-1.532	-1.022	-1.602	-1.220	-0.462	-1.196	-0.147	-0.950	1.560	…	
3	1.940	-0.358	-1.211	0.383	0.051	-0.171	-0.109	-0.002	0.869	-0.002	…	
4	1.081	-0.503	1.076	-0.543	-1.473	-1.065	-0.443	-0.143	1.660	-1.131	…	

	V21	V22	V23	V24	V25	V26	V27	V28	scaled_amount	\
0	-0.336	-0.808	-0.056	-1.025	-0.370	0.205	0.243	0.086	-0.292	
1	-1.377	-0.072	-0.198	1.015	1.011	-0.168	0.113	0.257	0.878	
2	-0.193	-0.104	0.151	-0.811	-0.198	-0.128	0.014	-0.051	0.289	
3	0.158	0.650	0.034	0.740	0.224	-0.196	-0.013	-0.057	0.113	
4	0.224	0.821	-0.137	0.986	0.563	-0.574	0.090	0.052	0.641	

	scaled_time
0	0.328
1	-0.689
2	0.875
3	0.616
4	-0.257

[5 rows x 30 columns]

1.3.5 RanomSearch

In Random Search, we create a grid of hyperparameters and train/test our model on just some random combination of these hyperparameters.

[92]:
```
sklearn_model = KerasClassifier(build_fn=create_model, epochs=30)
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
learn_rate = [0.01, 0.02, 0.2]
dropout_rate = [0.0, 0.1, 0.2, 0.4]
param_grid = dict(batch_size=batch_size, epochs=epochs,learn_rate=learn_rate,␣
 ↪dropout_rate=dropout_rate)
```

[93]:
```
grid = RandomizedSearchCV(estimator=sklearn_model, param_distributions␣
 ↪=param_grid, n_jobs=2, cv=3)
grid_result = grid.fit(x_train,y_train,validation_data=(x_test, y_test))
print("\nBest: %f using %s" % (grid_result.best_score_, grid_result.
 ↪best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("\n%f (%f) with: %r" % (mean, stdev, param))
```

```
Train on 620 samples, validate on 266 samples
Epoch 1/10
620/620 [==============================] - 1s 2ms/sample - loss: 0.4140 -
accuracy: 0.8419 - auc_5: 0.9176 - val_loss: 0.1594 - val_accuracy: 0.9436 -
val_auc_5: 0.9876
Epoch 2/10
620/620 [==============================] - 0s 262us/sample - loss: 0.2066 -
accuracy: 0.9403 - auc_5: 0.9662 - val_loss: 0.1854 - val_accuracy: 0.9436 -
val_auc_5: 0.9878
Epoch 3/10
620/620 [==============================] - 0s 290us/sample - loss: 0.2015 -
accuracy: 0.9355 - auc_5: 0.9679 - val_loss: 0.1591 - val_accuracy: 0.9398 -
val_auc_5: 0.9864
Epoch 4/10
620/620 [==============================] - 0s 260us/sample - loss: 0.1545 -
accuracy: 0.9435 - auc_5: 0.9804 - val_loss: 0.1340 - val_accuracy: 0.9474 -
val_auc_5: 0.9893
Epoch 5/10
620/620 [==============================] - 0s 263us/sample - loss: 0.2234 -
accuracy: 0.9210 - auc_5: 0.9699 - val_loss: 0.1474 - val_accuracy: 0.9436 -
val_auc_5: 0.9876
```

```
Epoch 6/10
620/620 [==============================] - 0s 260us/sample - loss: 0.1427 -
accuracy: 0.9532 - auc_5: 0.9838 - val_loss: 0.1498 - val_accuracy: 0.9398 -
val_auc_5: 0.9866
Epoch 7/10
620/620 [==============================] - 0s 279us/sample - loss: 0.1261 -
accuracy: 0.9613 - auc_5: 0.9900 - val_loss: 0.1526 - val_accuracy: 0.9436 -
val_auc_5: 0.9857
Epoch 8/10
620/620 [==============================] - 0s 272us/sample - loss: 0.1332 -
accuracy: 0.9548 - auc_5: 0.9869 - val_loss: 0.1516 - val_accuracy: 0.9248 -
val_auc_5: 0.9836
Epoch 9/10
620/620 [==============================] - 0s 269us/sample - loss: 0.1263 -
accuracy: 0.9516 - auc_5: 0.9904 - val_loss: 0.1734 - val_accuracy: 0.9398 -
val_auc_5: 0.9843
Epoch 10/10
620/620 [==============================] - 0s 263us/sample - loss: 0.1229 -
accuracy: 0.9484 - auc_5: 0.9896 - val_loss: 0.1508 - val_accuracy: 0.9361 -
val_auc_5: 0.9869

Best: 0.933845 using {'learn_rate': 0.02, 'epochs': 10, 'dropout_rate': 0.4,
'batch_size': 10}

0.927419 (0.003948) with: {'learn_rate': 0.01, 'epochs': 50, 'dropout_rate':
0.1, 'batch_size': 10}

0.924183 (0.006169) with: {'learn_rate': 0.2, 'epochs': 50, 'dropout_rate': 0.0,
'batch_size': 20}

0.924206 (0.005879) with: {'learn_rate': 0.02, 'epochs': 10, 'dropout_rate':
0.2, 'batch_size': 60}

0.909667 (0.006194) with: {'learn_rate': 0.01, 'epochs': 100, 'dropout_rate':
0.1, 'batch_size': 80}

0.919352 (0.002377) with: {'learn_rate': 0.02, 'epochs': 10, 'dropout_rate':
0.2, 'batch_size': 20}

0.928998 (0.017962) with: {'learn_rate': 0.01, 'epochs': 10, 'dropout_rate':
0.4, 'batch_size': 20}

0.921001 (0.015818) with: {'learn_rate': 0.02, 'epochs': 10, 'dropout_rate':
0.0, 'batch_size': 60}

0.927403 (0.010560) with: {'learn_rate': 0.2, 'epochs': 10, 'dropout_rate': 0.1,
'batch_size': 40}
```

```
0.933845 (0.014007) with: {'learn_rate': 0.02, 'epochs': 10, 'dropout_rate':
0.4, 'batch_size': 10}

0.912879 (0.012035) with: {'learn_rate': 0.2, 'epochs': 100, 'dropout_rate':
0.0, 'batch_size': 100}
```

1.3.6 Cross-Validation

[94]:
```
scores = cross_val_score(sklearn_model, x_train, y_train, cv=5, scoring='f1')
print("f1 score is: ", scores.mean())
```

```
Train on 496 samples
Epoch 1/30
496/496 [==============================] - 1s 2ms/sample - loss: 0.6078 -
accuracy: 0.7742 - auc_6: 0.8941
Epoch 2/30
496/496 [==============================] - 0s 113us/sample - loss: 0.3223 -
accuracy: 0.9133 - auc_6: 0.9413
Epoch 3/30
496/496 [==============================] - 0s 113us/sample - loss: 0.2116 -
accuracy: 0.9194 - auc_6: 0.9644
Epoch 4/30
496/496 [==============================] - 0s 113us/sample - loss: 0.1729 -
accuracy: 0.9355 - auc_6: 0.9810
Epoch 5/30
496/496 [==============================] - 0s 111us/sample - loss: 0.1412 -
accuracy: 0.9435 - auc_6: 0.9848
Epoch 6/30
496/496 [==============================] - 0s 109us/sample - loss: 0.1214 -
accuracy: 0.9597 - auc_6: 0.9879
Epoch 7/30
496/496 [==============================] - 0s 107us/sample - loss: 0.1174 -
accuracy: 0.9536 - auc_6: 0.9903
Epoch 8/30
496/496 [==============================] - 0s 107us/sample - loss: 0.1220 -
accuracy: 0.9476 - auc_6: 0.9904
Epoch 9/30
496/496 [==============================] - 0s 104us/sample - loss: 0.1093 -
accuracy: 0.9637 - auc_6: 0.9919
Epoch 10/30
496/496 [==============================] - 0s 85us/sample - loss: 0.0877 -
accuracy: 0.9597 - auc_6: 0.9963
Epoch 11/30
496/496 [==============================] - 0s 95us/sample - loss: 0.0912 -
accuracy: 0.9617 - auc_6: 0.9946
Epoch 12/30
496/496 [==============================] - 0s 112us/sample - loss: 0.0821 -
```

```
accuracy: 0.9657 - auc_6: 0.9954
Epoch 13/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0887 -
accuracy: 0.9657 - auc_6: 0.9965
Epoch 14/30
496/496 [==============================] - 0s 102us/sample - loss: 0.0688 -
accuracy: 0.9738 - auc_6: 0.9973
Epoch 15/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0498 -
accuracy: 0.9778 - auc_6: 0.9986
Epoch 16/30
496/496 [==============================] - 0s 102us/sample - loss: 0.0482 -
accuracy: 0.9778 - auc_6: 0.9987
Epoch 17/30
496/496 [==============================] - 0s 118us/sample - loss: 0.0255 -
accuracy: 0.9899 - auc_6: 0.9997
Epoch 18/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0407 -
accuracy: 0.9859 - auc_6: 0.9988
Epoch 19/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0446 -
accuracy: 0.9819 - auc_6: 0.9990
Epoch 20/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0619 -
accuracy: 0.9758 - auc_6: 0.9987
Epoch 21/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0409 -
accuracy: 0.9819 - auc_6: 0.9993
Epoch 22/30
496/496 [==============================] - 0s 117us/sample - loss: 0.0253 -
accuracy: 0.9919 - auc_6: 0.9998
Epoch 23/30
496/496 [==============================] - 0s 117us/sample - loss: 0.0100 -
accuracy: 0.9980 - auc_6: 1.0000
Epoch 24/30
496/496 [==============================] - 0s 117us/sample - loss: 0.0139 -
accuracy: 0.9960 - auc_6: 0.9999
Epoch 25/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0421 -
accuracy: 0.9899 - auc_6: 0.9979
Epoch 26/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0562 -
accuracy: 0.9859 - auc_6: 0.9951
Epoch 27/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0280 -
accuracy: 0.9879 - auc_6: 0.9995
Epoch 28/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0256 -
```

```
accuracy: 0.9919 - auc_6: 0.9997
Epoch 29/30
496/496 [==============================] - 0s 97us/sample - loss: 0.0256 -
accuracy: 0.9879 - auc_6: 0.9999
Epoch 30/30
496/496 [==============================] - 0s 98us/sample - loss: 0.0359 -
accuracy: 0.9879 - auc_6: 0.9995
Train on 496 samples
Epoch 1/30
496/496 [==============================] - 1s 2ms/sample - loss: 0.5988 -
accuracy: 0.7419 - auc_7: 0.8996
Epoch 2/30
496/496 [==============================] - 0s 101us/sample - loss: 0.2783 -
accuracy: 0.9315 - auc_7: 0.9509
Epoch 3/30
496/496 [==============================] - 0s 107us/sample - loss: 0.1805 -
accuracy: 0.9395 - auc_7: 0.9685
Epoch 4/30
496/496 [==============================] - 0s 113us/sample - loss: 0.1714 -
accuracy: 0.9375 - auc_7: 0.9826
Epoch 5/30
496/496 [==============================] - 0s 107us/sample - loss: 0.1271 -
accuracy: 0.9516 - auc_7: 0.9886
Epoch 6/30
496/496 [==============================] - 0s 113us/sample - loss: 0.1046 -
accuracy: 0.9516 - auc_7: 0.9938
Epoch 7/30
496/496 [==============================] - 0s 103us/sample - loss: 0.1088 -
accuracy: 0.9577 - auc_7: 0.9927
Epoch 8/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0831 -
accuracy: 0.9677 - auc_7: 0.9959
Epoch 9/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0706 -
accuracy: 0.9718 - auc_7: 0.9971
Epoch 10/30
496/496 [==============================] - 0s 97us/sample - loss: 0.0596 -
accuracy: 0.9819 - auc_7: 0.9960
Epoch 11/30
496/496 [==============================] - 0s 99us/sample - loss: 0.0800 -
accuracy: 0.9698 - auc_7: 0.9971
Epoch 12/30
496/496 [==============================] - 0s 118us/sample - loss: 0.0750 -
accuracy: 0.9778 - auc_7: 0.9934
Epoch 13/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0536 -
accuracy: 0.9859 - auc_7: 0.9982
Epoch 14/30
```

```
496/496 [==============================] - 0s 109us/sample - loss: 0.0318 -
accuracy: 0.9899 - auc_7: 0.9994
Epoch 15/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0410 -
accuracy: 0.9859 - auc_7: 0.9989
Epoch 16/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0267 -
accuracy: 0.9919 - auc_7: 0.9997
Epoch 17/30
496/496 [==============================] - 0s 99us/sample - loss: 0.0254 -
accuracy: 0.9899 - auc_7: 0.9996
Epoch 18/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0674 -
accuracy: 0.9758 - auc_7: 0.9965
Epoch 19/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0444 -
accuracy: 0.9859 - auc_7: 0.9971
Epoch 20/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0721 -
accuracy: 0.9718 - auc_7: 0.9953
Epoch 21/30
496/496 [==============================] - 0s 97us/sample - loss: 0.0385 -
accuracy: 0.9879 - auc_7: 0.9991
Epoch 22/30
496/496 [==============================] - 0s 110us/sample - loss: 0.0224 -
accuracy: 0.9899 - auc_7: 0.9997
Epoch 23/30
496/496 [==============================] - 0s 91us/sample - loss: 0.0157 -
accuracy: 0.9940 - auc_7: 0.9999
Epoch 24/30
496/496 [==============================] - 0s 120us/sample - loss: 0.0176 -
accuracy: 0.9899 - auc_7: 0.9999
Epoch 25/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0042 -
accuracy: 1.0000 - auc_7: 1.0000
Epoch 26/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0223 -
accuracy: 0.9940 - auc_7: 0.9977
Epoch 27/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0419 -
accuracy: 0.9879 - auc_7: 0.9975
Epoch 28/30
496/496 [==============================] - 0s 111us/sample - loss: 0.0198 -
accuracy: 0.9940 - auc_7: 0.9998
Epoch 29/30
496/496 [==============================] - 0s 113us/sample - loss: 0.0066 -
accuracy: 0.9980 - auc_7: 1.0000
Epoch 30/30
```

```
496/496 [==============================] - 0s 109us/sample - loss: 0.0024 -
accuracy: 1.0000 - auc_7: 1.0000
Train on 496 samples
Epoch 1/30
496/496 [==============================] - 1s 2ms/sample - loss: 0.6216 -
accuracy: 0.8488 - auc_8: 0.9102
Epoch 2/30
496/496 [==============================] - 0s 108us/sample - loss: 0.2679 -
accuracy: 0.9214 - auc_8: 0.9613
Epoch 3/30
496/496 [==============================] - 0s 94us/sample - loss: 0.1811 -
accuracy: 0.9254 - auc_8: 0.9786
Epoch 4/30
496/496 [==============================] - 0s 108us/sample - loss: 0.1578 -
accuracy: 0.9456 - auc_8: 0.9791
Epoch 5/30
496/496 [==============================] - 0s 125us/sample - loss: 0.1336 -
accuracy: 0.9496 - auc_8: 0.9873
Epoch 6/30
496/496 [==============================] - 0s 117us/sample - loss: 0.1065 -
accuracy: 0.9637 - auc_8: 0.9917
Epoch 7/30
496/496 [==============================] - 0s 119us/sample - loss: 0.1107 -
accuracy: 0.9456 - auc_8: 0.9919
Epoch 8/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0762 -
accuracy: 0.9738 - auc_8: 0.9954
Epoch 9/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0739 -
accuracy: 0.9657 - auc_8: 0.9965
Epoch 10/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0727 -
accuracy: 0.9718 - auc_8: 0.9968
Epoch 11/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0570 -
accuracy: 0.9758 - auc_8: 0.9982
Epoch 12/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0584 -
accuracy: 0.9758 - auc_8: 0.9983
Epoch 13/30
496/496 [==============================] - 0s 100us/sample - loss: 0.0727 -
accuracy: 0.9738 - auc_8: 0.9975
Epoch 14/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0844 -
accuracy: 0.9677 - auc_8: 0.9952
Epoch 15/30
496/496 [==============================] - 0s 87us/sample - loss: 0.0569 -
accuracy: 0.9778 - auc_8: 0.9980
```

```
Epoch 16/30
496/496 [==============================] - 0s 129us/sample - loss: 0.0322 -
accuracy: 0.9899 - auc_8: 0.9996
Epoch 17/30
496/496 [==============================] - 0s 106us/sample - loss: 0.0169 -
accuracy: 0.9940 - auc_8: 0.9999
Epoch 18/30
496/496 [==============================] - 0s 86us/sample - loss: 0.0339 -
accuracy: 0.9819 - auc_8: 0.9994
Epoch 19/30
496/496 [==============================] - 0s 95us/sample - loss: 0.0406 -
accuracy: 0.9778 - auc_8: 0.9991
Epoch 20/30
496/496 [==============================] - 0s 122us/sample - loss: 0.0226 -
accuracy: 0.9899 - auc_8: 0.9997
Epoch 21/30
496/496 [==============================] - 0s 89us/sample - loss: 0.0297 -
accuracy: 0.9899 - auc_8: 0.9996
Epoch 22/30
496/496 [==============================] - 0s 119us/sample - loss: 0.0125 -
accuracy: 0.9960 - auc_8: 0.9999
Epoch 23/30
496/496 [==============================] - 0s 99us/sample - loss: 0.0116 -
accuracy: 0.9940 - auc_8: 1.0000
Epoch 24/30
496/496 [==============================] - 0s 106us/sample - loss: 0.0314 -
accuracy: 0.9899 - auc_8: 0.9997
Epoch 25/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0187 -
accuracy: 0.9960 - auc_8: 0.9998
Epoch 26/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0224 -
accuracy: 0.9919 - auc_8: 0.9997
Epoch 27/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0277 -
accuracy: 0.9919 - auc_8: 0.9974
Epoch 28/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0269 -
accuracy: 0.9879 - auc_8: 0.9998
Epoch 29/30
496/496 [==============================] - 0s 102us/sample - loss: 0.0174 -
accuracy: 0.9919 - auc_8: 1.0000
Epoch 30/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0143 -
accuracy: 0.9960 - auc_8: 0.9998
Train on 496 samples
Epoch 1/30
496/496 [==============================] - 1s 2ms/sample - loss: 0.6225 -
```

```
accuracy: 0.8528 - auc_9: 0.9017
Epoch 2/30
496/496 [==============================] - 0s 114us/sample - loss: 0.2611 -
accuracy: 0.9214 - auc_9: 0.9504
Epoch 3/30
496/496 [==============================] - 0s 105us/sample - loss: 0.1920 -
accuracy: 0.9375 - auc_9: 0.9686
Epoch 4/30
496/496 [==============================] - 0s 81us/sample - loss: 0.1697 -
accuracy: 0.9315 - auc_9: 0.9785
Epoch 5/30
496/496 [==============================] - 0s 129us/sample - loss: 0.1282 -
accuracy: 0.9435 - auc_9: 0.9894
Epoch 6/30
496/496 [==============================] - 0s 103us/sample - loss: 0.1229 -
accuracy: 0.9496 - auc_9: 0.9893
Epoch 7/30
496/496 [==============================] - 0s 98us/sample - loss: 0.1122 -
accuracy: 0.9597 - auc_9: 0.9905
Epoch 8/30
496/496 [==============================] - 0s 94us/sample - loss: 0.0853 -
accuracy: 0.9677 - auc_9: 0.9941
Epoch 9/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0703 -
accuracy: 0.9718 - auc_9: 0.9967
Epoch 10/30
496/496 [==============================] - 0s 119us/sample - loss: 0.0789 -
accuracy: 0.9738 - auc_9: 0.9945
Epoch 11/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0773 -
accuracy: 0.9718 - auc_9: 0.9971
Epoch 12/30
496/496 [==============================] - 0s 176us/sample - loss: 0.0539 -
accuracy: 0.9879 - auc_9: 0.9969
Epoch 13/30
496/496 [==============================] - 0s 155us/sample - loss: 0.0440 -
accuracy: 0.9798 - auc_9: 0.9988
Epoch 14/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0524 -
accuracy: 0.9798 - auc_9: 0.9983
Epoch 15/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0334 -
accuracy: 0.9859 - auc_9: 0.9993
Epoch 16/30
496/496 [==============================] - 0s 111us/sample - loss: 0.0563 -
accuracy: 0.9738 - auc_9: 0.9983
Epoch 17/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0334 -
```

```
accuracy: 0.9859 - auc_9: 0.9994
Epoch 18/30
496/496 [==============================] - 0s 121us/sample - loss: 0.0483 -
accuracy: 0.9778 - auc_9: 0.9985
Epoch 19/30
496/496 [==============================] - 0s 125us/sample - loss: 0.0453 -
accuracy: 0.9899 - auc_9: 0.9973
Epoch 20/30
496/496 [==============================] - 0s 113us/sample - loss: 0.0432 -
accuracy: 0.9899 - auc_9: 0.9972
Epoch 21/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0196 -
accuracy: 0.9940 - auc_9: 0.9999
Epoch 22/30
496/496 [==============================] - 0s 137us/sample - loss: 0.0202 -
accuracy: 0.9919 - auc_9: 0.9998
Epoch 23/30
496/496 [==============================] - 0s 137us/sample - loss: 0.0162 -
accuracy: 0.9919 - auc_9: 0.9999
Epoch 24/30
496/496 [==============================] - 0s 180us/sample - loss: 0.0200 -
accuracy: 0.9879 - auc_9: 0.9998
Epoch 25/30
496/496 [==============================] - 0s 157us/sample - loss: 0.0118 -
accuracy: 0.9960 - auc_9: 0.9999
Epoch 26/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0315 -
accuracy: 0.9879 - auc_9: 0.9995
Epoch 27/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0373 -
accuracy: 0.9879 - auc_9: 0.9992
Epoch 28/30
496/496 [==============================] - 0s 113us/sample - loss: 0.0287 -
accuracy: 0.9940 - auc_9: 0.9996
Epoch 29/30
496/496 [==============================] - 0s 111us/sample - loss: 0.0217 -
accuracy: 0.9940 - auc_9: 0.9998
Epoch 30/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0097 -
accuracy: 0.9960 - auc_9: 1.0000
Train on 496 samples
Epoch 1/30
496/496 [==============================] - 1s 2ms/sample - loss: 0.6537 -
accuracy: 0.7681 - auc_10: 0.8853
Epoch 2/30
496/496 [==============================] - 0s 131us/sample - loss: 0.3304 -
accuracy: 0.9052 - auc_10: 0.9583
Epoch 3/30
```

```
496/496 [==============================] - 0s 127us/sample - loss: 0.1609 -
accuracy: 0.9496 - auc_10: 0.9745
Epoch 4/30
496/496 [==============================] - 0s 115us/sample - loss: 0.1199 -
accuracy: 0.9556 - auc_10: 0.9891
Epoch 5/30
496/496 [==============================] - 0s 117us/sample - loss: 0.1105 -
accuracy: 0.9577 - auc_10: 0.9903
Epoch 6/30
496/496 [==============================] - 0s 142us/sample - loss: 0.0993 -
accuracy: 0.9657 - auc_10: 0.9905
Epoch 7/30
496/496 [==============================] - 0s 146us/sample - loss: 0.0904 -
accuracy: 0.9677 - auc_10: 0.9940
Epoch 8/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0741 -
accuracy: 0.9718 - auc_10: 0.9963
Epoch 9/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0669 -
accuracy: 0.9698 - auc_10: 0.9954
Epoch 10/30
496/496 [==============================] - 0s 109us/sample - loss: 0.0615 -
accuracy: 0.9798 - auc_10: 0.9963
Epoch 11/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0560 -
accuracy: 0.9778 - auc_10: 0.9978
Epoch 12/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0324 -
accuracy: 0.9859 - auc_10: 0.9996
Epoch 13/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0506 -
accuracy: 0.9819 - auc_10: 0.9968
Epoch 14/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0592 -
accuracy: 0.9798 - auc_10: 0.9976
Epoch 15/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0395 -
accuracy: 0.9879 - auc_10: 0.9991
Epoch 16/30
496/496 [==============================] - 0s 117us/sample - loss: 0.0198 -
accuracy: 0.9940 - auc_10: 0.9999
Epoch 17/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0138 -
accuracy: 0.9960 - auc_10: 0.9999
Epoch 18/30
496/496 [==============================] - 0s 101us/sample - loss: 0.0204 -
accuracy: 0.9980 - auc_10: 0.9977
Epoch 19/30
```

```
496/496 [==============================] - 0s 107us/sample - loss: 0.0050 -
accuracy: 0.9980 - auc_10: 1.0000
Epoch 20/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0317 -
accuracy: 0.9919 - auc_10: 0.9977
Epoch 21/30
496/496 [==============================] - 0s 104us/sample - loss: 0.0600 -
accuracy: 0.9798 - auc_10: 0.9966
Epoch 22/30
496/496 [==============================] - 0s 99us/sample - loss: 0.0225 -
accuracy: 0.9919 - auc_10: 0.9997
Epoch 23/30
496/496 [==============================] - 0s 102us/sample - loss: 0.0280 -
accuracy: 0.9879 - auc_10: 0.9997
Epoch 24/30
496/496 [==============================] - 0s 102us/sample - loss: 0.0294 -
accuracy: 0.9919 - auc_10: 0.9994
Epoch 25/30
496/496 [==============================] - 0s 102us/sample - loss: 0.0518 -
accuracy: 0.9899 - auc_10: 0.9955
Epoch 26/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0530 -
accuracy: 0.9798 - auc_10: 0.9971
Epoch 27/30
496/496 [==============================] - 0s 115us/sample - loss: 0.0261 -
accuracy: 0.9940 - auc_10: 0.9998
Epoch 28/30
496/496 [==============================] - 0s 103us/sample - loss: 0.0128 -
accuracy: 0.9980 - auc_10: 1.0000
Epoch 29/30
496/496 [==============================] - 0s 107us/sample - loss: 0.0160 -
accuracy: 0.9919 - auc_10: 0.9999
Epoch 30/30
496/496 [==============================] - 0s 105us/sample - loss: 0.0035 -
accuracy: 1.0000 - auc_10: 1.0000
f1 score is:  0.9203886495190844
```

- F1 score for deep learning model is 0.9182353216277704

Experiment with number of epochs. Check the observations in loss and accuracy

[95]:
```
grid = GridSearchCV(estimator=sklearn_model, param_grid=param_grid, n_jobs=2,␣
 ↪cv=3)
grid_result = grid.fit(x_train,y_train,validation_data=(x_test, y_test))
print("\nBest: %f using %s" % (grid_result.best_score_, grid_result.
 ↪best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
```

```
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("\n%f (%f) with: %r" % (mean, stdev, param))
```

```
C:\Users\Pritesh\Anaconda3\lib\site-
packages\joblib\externals\loky\process_executor.py:691: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning

Train on 620 samples, validate on 266 samples
Epoch 1/10
620/620 [==============================] - 1s 2ms/sample - loss: 0.6138 -
accuracy: 0.7984 - auc_11: 0.8918 - val_loss: 0.4239 - val_accuracy: 0.8421 -
val_auc_11: 0.9794
Epoch 2/10
620/620 [==============================] - 0s 111us/sample - loss: 0.2577 -
accuracy: 0.9145 - auc_11: 0.9647 - val_loss: 0.2386 - val_accuracy: 0.9098 -
val_auc_11: 0.9847
Epoch 3/10
620/620 [==============================] - 0s 110us/sample - loss: 0.1890 -
accuracy: 0.9210 - auc_11: 0.9750 - val_loss: 0.2047 - val_accuracy: 0.9398 -
val_auc_11: 0.9857
Epoch 4/10
620/620 [==============================] - 0s 111us/sample - loss: 0.1634 -
accuracy: 0.9403 - auc_11: 0.9787 - val_loss: 0.2017 - val_accuracy: 0.9549 -
val_auc_11: 0.9863
Epoch 5/10
620/620 [==============================] - 0s 111us/sample - loss: 0.1310 -
accuracy: 0.9500 - auc_11: 0.9882 - val_loss: 0.1684 - val_accuracy: 0.9474 -
val_auc_11: 0.9855
Epoch 6/10
620/620 [==============================] - 0s 113us/sample - loss: 0.1458 -
accuracy: 0.9468 - auc_11: 0.9847 - val_loss: 0.1514 - val_accuracy: 0.9511 -
val_auc_11: 0.9834
Epoch 7/10
620/620 [==============================] - 0s 108us/sample - loss: 0.1066 -
accuracy: 0.9597 - auc_11: 0.9919 - val_loss: 0.1407 - val_accuracy: 0.9549 -
val_auc_11: 0.9866
Epoch 8/10
620/620 [==============================] - 0s 113us/sample - loss: 0.0966 -
accuracy: 0.9629 - auc_11: 0.9930 - val_loss: 0.1406 - val_accuracy: 0.9474 -
val_auc_11: 0.9863
Epoch 9/10
620/620 [==============================] - 0s 111us/sample - loss: 0.0789 -
accuracy: 0.9774 - auc_11: 0.9941 - val_loss: 0.1332 - val_accuracy: 0.9474 -
val_auc_11: 0.9867
Epoch 10/10
```

```
620/620 [==============================] - 0s 108us/sample - loss: 0.0847 -
accuracy: 0.9710 - auc_11: 0.9949 - val_loss: 0.1358 - val_accuracy: 0.9474 -
val_auc_11: 0.9861

Best: 0.938675 using {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.920986 (0.008061) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.01}

0.927419 (0.015779) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.02}

0.909651 (0.016592) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.2}

0.914529 (0.008089) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.01}

0.917726 (0.007042) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.02}

0.904851 (0.014859) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.2}

0.900013 (0.005827) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.01}

0.914505 (0.006186) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.02}

0.890257 (0.028877) with: {'batch_size': 10, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.2}

0.924175 (0.008404) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.01}

0.925785 (0.010109) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.02}

0.922580 (0.000177) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.2}

0.916116 (0.006224) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.01}

0.914513 (0.002383) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.02}
```

```
0.904859 (0.009754) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.2}

0.900005 (0.002056) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.01}

0.917741 (0.003949) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.02}

0.909674 (0.009953) with: {'batch_size': 10, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.2}

0.914537 (0.021655) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.01}

0.906430 (0.010150) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.920939 (0.012898) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.2}

0.904867 (0.013693) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.01}

0.909659 (0.010114) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.02}

0.904836 (0.009954) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.2}

0.917749 (0.006745) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.01}

0.914521 (0.012016) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.02}

0.898363 (0.020630) with: {'batch_size': 10, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.2}

0.922565 (0.007031) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.01}

0.908033 (0.015998) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.02}

0.929014 (0.010078) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.2}
```

```
0.916123 (0.004657) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.01}

0.909690 (0.014863) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.02}

0.906462 (0.005876) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.2}

0.898410 (0.023909) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.01}

0.911300 (0.005884) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.02}

0.909659 (0.010114) with: {'batch_size': 10, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.2}

0.919328 (0.022885) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.01}

0.912871 (0.014456) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.02}

0.911285 (0.015973) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.2}

0.914474 (0.020087) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.01}

0.916123 (0.008268) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.02}

0.912903 (0.000199) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.2}

0.904828 (0.006203) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.01}

0.919344 (0.009210) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.02}

0.917741 (0.003949) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.2}

0.919367 (0.008097) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.01}
```

```
0.930624 (0.012193) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.02}

0.933845 (0.016553) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.2}

0.906422 (0.012931) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.01}

0.919360 (0.004469) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.02}

0.912895 (0.004128) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.2}

0.908048 (0.008091) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.01}

0.919352 (0.002377) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.02}

0.904820 (0.008447) with: {'batch_size': 20, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.2}

0.920970 (0.002194) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.01}

0.911269 (0.012227) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.922557 (0.020602) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.2}

0.916108 (0.010130) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.01}

0.924183 (0.006169) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.02}

0.911261 (0.012920) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.2}

0.909682 (0.012014) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.01}

0.919344 (0.004754) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.02}
```

```
0.908048 (0.008091) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.2}

0.928990 (0.021910) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.01}

0.933845 (0.016553) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.02}

0.929021 (0.012744) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.2}

0.898402 (0.010300) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.01}

0.903249 (0.014087) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.02}

0.911246 (0.021961) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.2}

0.919344 (0.012752) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.01}

0.911292 (0.005991) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.02}

0.916123 (0.002476) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.2}

0.935478 (0.012081) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.01}

0.929037 (0.008169) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.02}

0.933845 (0.014007) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.2}

0.917726 (0.010575) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.01}

0.917741 (0.011835) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.02}

0.903249 (0.011671) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.2}
```

```
0.914521 (0.002089) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.01}

0.920954 (0.015010) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.02}

0.909682 (0.002078) with: {'batch_size': 40, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.2}

0.920954 (0.006214) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.01}

0.929045 (0.008112) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.02}

0.922580 (0.003948) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.2}

0.917741 (0.015779) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.01}

0.929029 (0.006059) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.02}

0.916139 (0.009022) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.2}

0.919352 (0.006064) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.01}

0.916123 (0.004657) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.02}

0.901623 (0.005868) with: {'batch_size': 40, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.2}

0.924206 (0.005879) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.01}

0.938675 (0.020379) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.922580 (0.007891) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.2}

0.922557 (0.014374) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.01}
```

```
0.916131 (0.005993) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.02}

0.909698 (0.009765) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.2}

0.917749 (0.018043) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.01}

0.924206 (0.008104) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.02}

0.914513 (0.009951) with: {'batch_size': 40, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.2}

0.925801 (0.004646) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.01}

0.929021 (0.004732) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.02}

0.925801 (0.004646) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.2}

0.922580 (0.003948) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.01}

0.920954 (0.011487) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.02}

0.920954 (0.006214) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.2}

0.911300 (0.004367) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.01}

0.909651 (0.014053) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.02}

0.914497 (0.012820) with: {'batch_size': 40, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.2}

0.925832 (0.011925) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.01}

0.916155 (0.011904) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.02}
```

```
0.914537 (0.011926) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.2}

0.924183 (0.009205) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.01}

0.925785 (0.010109) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.02}

0.927403 (0.008052) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.2}

0.909667 (0.006194) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.01}

0.916108 (0.010130) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.02}

0.917718 (0.012025) with: {'batch_size': 60, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.2}

0.917757 (0.013582) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.01}

0.927435 (0.010342) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.02}

0.914552 (0.015798) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.2}

0.920962 (0.008265) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.01}

0.924191 (0.009948) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.02}

0.920954 (0.011487) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.2}

0.906446 (0.002498) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.01}

0.911277 (0.006235) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.02}

0.911285 (0.012092) with: {'batch_size': 60, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.2}
```

```
0.920986 (0.008061) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.01}

0.917765 (0.011700) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.925832 (0.011925) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.2}

0.924183 (0.009205) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.01}

0.908056 (0.006945) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.02}

0.922557 (0.010642) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.2}

0.916116 (0.006224) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.01}

0.904828 (0.006203) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.02}

0.917741 (0.003949) with: {'batch_size': 60, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.2}

0.927442 (0.014127) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.01}

0.930647 (0.005998) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.02}

0.914537 (0.009775) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.2}

0.911285 (0.004663) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.01}

0.925793 (0.008342) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.02}

0.919352 (0.009950) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.2}

0.912911 (0.003783) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.01}
```

```
0.906446 (0.004669) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.02}

0.922580 (0.003948) with: {'batch_size': 60, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.2}

0.916155 (0.011904) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.01}

0.904859 (0.011909) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.02}

0.916170 (0.019732) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.2}

0.919344 (0.009210) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.01}

0.925785 (0.010109) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.02}

0.916100 (0.014063) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.2}

0.920954 (0.011487) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.01}

0.912903 (0.000199) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.02}

0.917726 (0.007042) with: {'batch_size': 80, 'dropout_rate': 0.0, 'epochs': 100,
'learn_rate': 0.2}

0.916155 (0.013725) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.01}

0.919360 (0.004469) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.02}

0.917757 (0.006666) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.2}

0.925762 (0.026435) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.01}

0.925801 (0.012085) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.02}
```

```
0.917741 (0.011835) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.2}

0.911285 (0.002487) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.01}

0.916116 (0.006224) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.02}

0.914505 (0.006186) with: {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 100,
'learn_rate': 0.2}

0.932273 (0.013598) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.01}

0.911316 (0.013716) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.927435 (0.006688) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.2}

0.919352 (0.002377) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.01}

0.924167 (0.012249) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.02}

0.919344 (0.009210) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.2}

0.917726 (0.013770) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.01}

0.914529 (0.014867) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.02}

0.914505 (0.006186) with: {'batch_size': 80, 'dropout_rate': 0.2, 'epochs': 100,
'learn_rate': 0.2}

0.922611 (0.014094) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.01}

0.917781 (0.024541) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.02}

0.919375 (0.008954) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.2}
```

```
0.930624 (0.012193) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.01}

0.927419 (0.000166) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.02}

0.920954 (0.015010) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.2}

0.917749 (0.006745) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.01}

0.922572 (0.004108) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.02}

0.920962 (0.004651) with: {'batch_size': 80, 'dropout_rate': 0.4, 'epochs': 100,
'learn_rate': 0.2}

0.904859 (0.011909) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.01}

0.914529 (0.005858) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.02}

0.916131 (0.009907) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs': 10,
'learn_rate': 0.2}

0.927380 (0.019914) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.01}

0.922541 (0.017424) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.02}

0.911285 (0.004663) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs': 50,
'learn_rate': 0.2}

0.912918 (0.007736) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs':
100, 'learn_rate': 0.01}

0.919344 (0.006177) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs':
100, 'learn_rate': 0.02}

0.922596 (0.006677) with: {'batch_size': 100, 'dropout_rate': 0.0, 'epochs':
100, 'learn_rate': 0.2}

0.900028 (0.028453) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.01}
```

```
0.914529 (0.014867) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.02}

0.912926 (0.010273) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs': 10,
'learn_rate': 0.2}

0.920939 (0.012898) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.01}

0.924198 (0.015918) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.02}

0.922572 (0.004108) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs': 50,
'learn_rate': 0.2}

0.911285 (0.008272) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs':
100, 'learn_rate': 0.01}

0.911261 (0.014073) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs':
100, 'learn_rate': 0.02}

0.919313 (0.020077) with: {'batch_size': 100, 'dropout_rate': 0.1, 'epochs':
100, 'learn_rate': 0.2}

0.925809 (0.002199) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.01}

0.912957 (0.025753) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.02}

0.903233 (0.003764) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs': 10,
'learn_rate': 0.2}

0.924183 (0.012748) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.01}

0.914513 (0.006066) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.02}

0.924167 (0.014026) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs': 50,
'learn_rate': 0.2}

0.917710 (0.014445) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs':
100, 'learn_rate': 0.01}

0.929068 (0.022351) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs':
100, 'learn_rate': 0.02}
```

```
0.919344 (0.012752) with: {'batch_size': 100, 'dropout_rate': 0.2, 'epochs':
100, 'learn_rate': 0.2}

0.912942 (0.017940) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.01}

0.912926 (0.010273) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.02}

0.922604 (0.011709) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs': 10,
'learn_rate': 0.2}

0.917718 (0.014382) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.01}

0.919360 (0.002100) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.02}

0.932234 (0.017306) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs': 50,
'learn_rate': 0.2}

0.922580 (0.011835) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs':
100, 'learn_rate': 0.01}

0.912895 (0.006939) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs':
100, 'learn_rate': 0.02}

0.916123 (0.004657) with: {'batch_size': 100, 'dropout_rate': 0.4, 'epochs':
100, 'learn_rate': 0.2}
```

- Best parameters are
- Best: 0.937065 using {'batch_size': 80, 'dropout_rate': 0.1, 'epochs': 50, 'learn_rate': 0.01}

*** Play with different Learning Rate variants of Gradient Descent like Adam, SGD, RMS-prop**

[96]:
```
optimizers = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax',␣
↪'Nadam']
param_grid = dict(optimizer=optimizers)
grid = GridSearchCV(estimator=sklearn_model, param_grid=param_grid, n_jobs=3,␣
↪cv=3)
grid_result = grid.fit(x_train, y_train)
print("Best optimizers With Best score: ",grid_result.best_score_, grid_result.
↪best_params_)
```

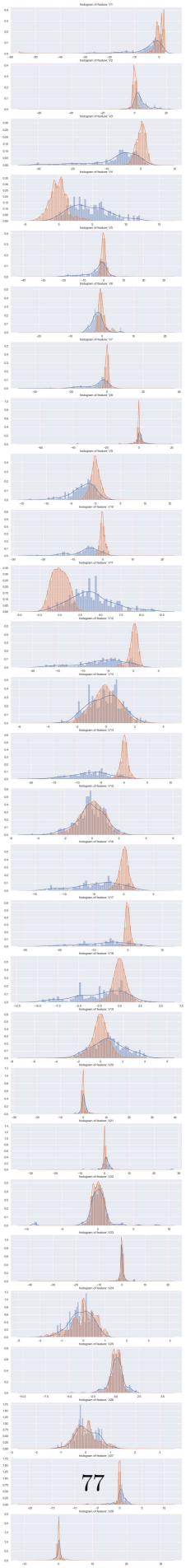
```
Train on 620 samples
Epoch 1/30
```

```
620/620 [==============================] - 1s 1ms/sample - loss: 0.6519 -
accuracy: 0.8387 - auc_12: 0.8975
Epoch 2/30
620/620 [==============================] - 0s 100us/sample - loss: 0.3832 -
accuracy: 0.9242 - auc_12: 0.9473
Epoch 3/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1999 -
accuracy: 0.9323 - auc_12: 0.9680
Epoch 4/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1885 -
accuracy: 0.9339 - auc_12: 0.9753
Epoch 5/30
620/620 [==============================] - 0s 102us/sample - loss: 0.1660 -
accuracy: 0.9355 - auc_12: 0.9800
Epoch 6/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1544 -
accuracy: 0.9403 - auc_12: 0.9826
Epoch 7/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1607 -
accuracy: 0.9484 - auc_12: 0.9796
Epoch 8/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1475 -
accuracy: 0.9403 - auc_12: 0.9848
Epoch 9/30
620/620 [==============================] - 0s 97us/sample - loss: 0.1501 -
accuracy: 0.9484 - auc_12: 0.9804
Epoch 10/30
620/620 [==============================] - 0s 97us/sample - loss: 0.1179 -
accuracy: 0.9548 - auc_12: 0.9897
Epoch 11/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1408 -
accuracy: 0.9403 - auc_12: 0.98680s - loss: 0.1396 - accuracy: 0.9408 - auc_12:
0.987
Epoch 12/30
620/620 [==============================] - 0s 97us/sample - loss: 0.1362 -
accuracy: 0.9484 - auc_12: 0.9846
Epoch 13/30
620/620 [==============================] - 0s 97us/sample - loss: 0.1186 -
accuracy: 0.9597 - auc_12: 0.9888
Epoch 14/30
620/620 [==============================] - 0s 100us/sample - loss: 0.1111 -
accuracy: 0.9613 - auc_12: 0.9911
Epoch 15/30
620/620 [==============================] - 0s 108us/sample - loss: 0.0942 -
accuracy: 0.9629 - auc_12: 0.9938
Epoch 16/30
620/620 [==============================] - 0s 98us/sample - loss: 0.1134 -
accuracy: 0.9613 - auc_12: 0.9889
```

```
Epoch 17/30
620/620 [==============================] - 0s 95us/sample - loss: 0.1051 -
accuracy: 0.9629 - auc_12: 0.9906
Epoch 18/30
620/620 [==============================] - 0s 100us/sample - loss: 0.0949 -
accuracy: 0.9661 - auc_12: 0.9935
Epoch 19/30
620/620 [==============================] - 0s 102us/sample - loss: 0.1030 -
accuracy: 0.9629 - auc_12: 0.9914
Epoch 20/30
620/620 [==============================] - 0s 98us/sample - loss: 0.0832 -
accuracy: 0.9742 - auc_12: 0.9953
Epoch 21/30
620/620 [==============================] - 0s 103us/sample - loss: 0.0786 -
accuracy: 0.9645 - auc_12: 0.9954
Epoch 22/30
620/620 [==============================] - 0s 100us/sample - loss: 0.0782 -
accuracy: 0.9677 - auc_12: 0.9955
Epoch 23/30
620/620 [==============================] - 0s 98us/sample - loss: 0.0645 -
accuracy: 0.9790 - auc_12: 0.9976
Epoch 24/30
620/620 [==============================] - 0s 105us/sample - loss: 0.0876 -
accuracy: 0.9694 - auc_12: 0.9948
Epoch 25/30
620/620 [==============================] - 0s 100us/sample - loss: 0.0656 -
accuracy: 0.9758 - auc_12: 0.9972
Epoch 26/30
620/620 [==============================] - 0s 100us/sample - loss: 0.0819 -
accuracy: 0.9710 - auc_12: 0.9934
Epoch 27/30
620/620 [==============================] - 0s 100us/sample - loss: 0.0887 -
accuracy: 0.9694 - auc_12: 0.9937
Epoch 28/30
620/620 [==============================] - 0s 97us/sample - loss: 0.0675 -
accuracy: 0.9774 - auc_12: 0.9964
Epoch 29/30
620/620 [==============================] - 0s 98us/sample - loss: 0.0703 -
accuracy: 0.9774 - auc_12: 0.9950
Epoch 30/30
620/620 [==============================] - 0s 102us/sample - loss: 0.0568 -
accuracy: 0.9774 - auc_12: 0.9982
Best optimizers With Best score:  0.9306473930676779 {'optimizer': 'Adamax'}
```

1.3.7 Anomaly Detection

[97]:
```
df_train = copy.deepcopy(df_train_original)
features = df_train.iloc[:, [x for x in range(1,29)]].columns
plt.figure(figsize=(12,28*4))

gs = gridspec.GridSpec(28, 1)
for i, c in enumerate(df_train[features]):
 ax = plt.subplot(gs[i])
 sns.distplot(df_train[c][df_train.Class == 1], bins=50)
 sns.distplot(df_train[c][df_train.Class == 0], bins=50)
 ax.set_xlabel('')
 ax.set_title('histogram of feature: ' + str(c))
plt.show()
```


Histogram of feature: V1
Histogram of feature: V2
Histogram of feature: V3
Histogram of feature: V4
Histogram of feature: V5
Histogram of feature: V6
Histogram of feature: V7
Histogram of feature: V8
Histogram of feature: V9
Histogram of feature: V10
Histogram of feature: V11
Histogram of feature: V12
Histogram of feature: V13
Histogram of feature: V14
Histogram of feature: V15
Histogram of feature: V16
Histogram of feature: V17
Histogram of feature: V18
Histogram of feature: V19
Histogram of feature: V20
Histogram of feature: V21
Histogram of feature: V22
Histogram of feature: V23
Histogram of feature: V24
Histogram of feature: V25
Histogram of feature: V26
Histogram of feature: V27
Histogram of feature: V28

[98]:
```
Fraud = df_train[df_train['Class'] == 1]
Valid = df_train[df_train['Class'] == 0]
outlier_fraction = len(Fraud)/float(len(Valid))
print(outlier_fraction)
print('Fraud Cases: {}'.format(len(df_train[df_train['Class'] == 1])))
print('Valid Transactions: {}'.format(len(df_train[df_train['Class'] == 0])))
```

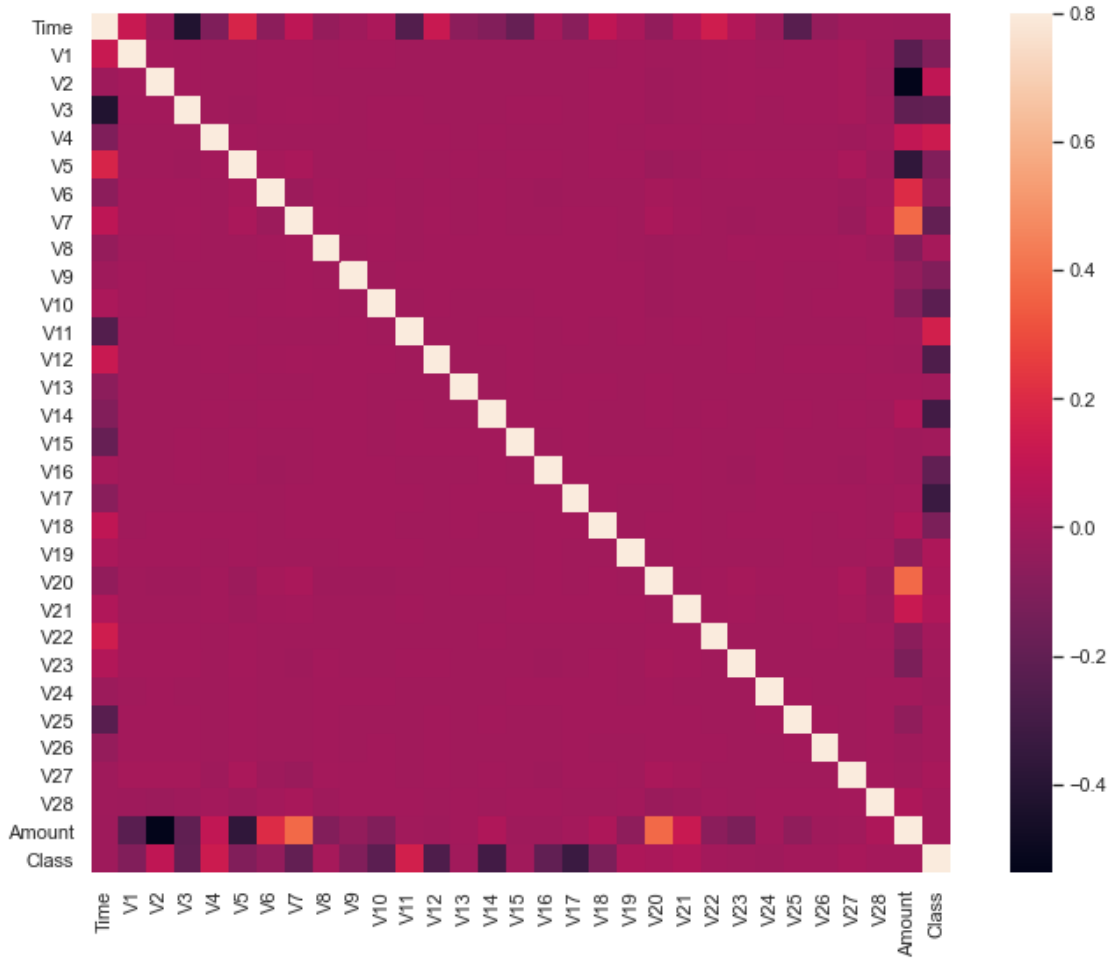
```
0.0017322412299792043
Fraud Cases: 394
Valid Transactions: 227451
```

[99]:
```
print('Amount details of fraudulent transaction')
Fraud.Amount.describe()
```

```
Amount details of fraudulent transaction
```

[99]:
```
count     394.000
mean      127.307
std       264.534
min         0.000
25%         1.000
50%        11.395
75%       106.385
max      2125.870
Name: Amount, dtype: float64
```

[100]:
```
# Correlation matrix
corrmat = df_train.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```



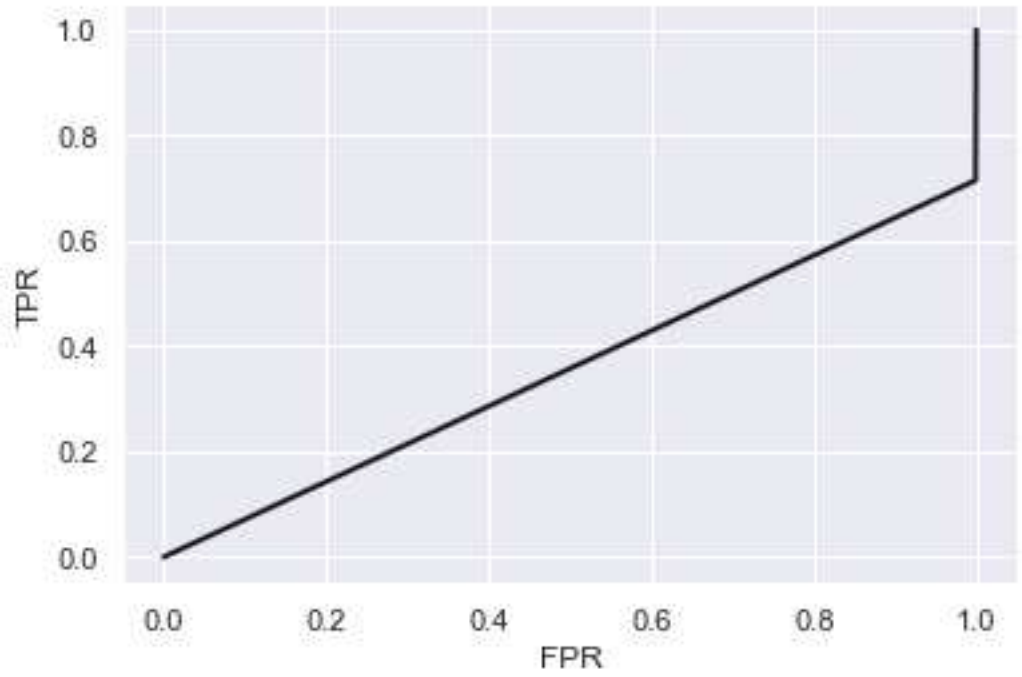
Anomalies Detection using Isolation forest algorithm

[101]:
```
y_data = df_train.pop('Class')
x_data = df_train
X_train, X_test, Y_train, Y_test = train_test_split(x_data, y_data, test_size =␣
↪0.2, random_state = 42)
```

[102]:
```
ifc=IsolationForest(max_samples=len(X_train),contamination=outlier_fraction,random_state=1,␣
↪n_jobs=4)
ifc.fit(X_train)
scores_pred = ifc.decision_function(X_train)
y_pred = ifc.predict(X_test)
```

[103]:
```
fpr, tpr, thresholds = roc_curve(Y_test,y_pred)
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'k-', lw=2)
plt.xlabel('FPR')
```

```
plt.ylabel('TPR')
plt.show()
```



[104]:
```
# np.set_printoptions(threshold=sys.maxsize)
ifc.score_samples(X_train)
```

[104]:
```
array([-0.33111238, -0.34219728, -0.33014527, …, -0.32617074,
       -0.32454384, -0.3225194 ])
```

[105]:
```
ifc.decision_function(X_train)
```

[105]:
```
array([0.19927989, 0.18819499, 0.200247  , …, 0.20422153, 0.20584843,
       0.20787287])
```

[106]:
```
y_score = -ifc.decision_function(X_train)
```

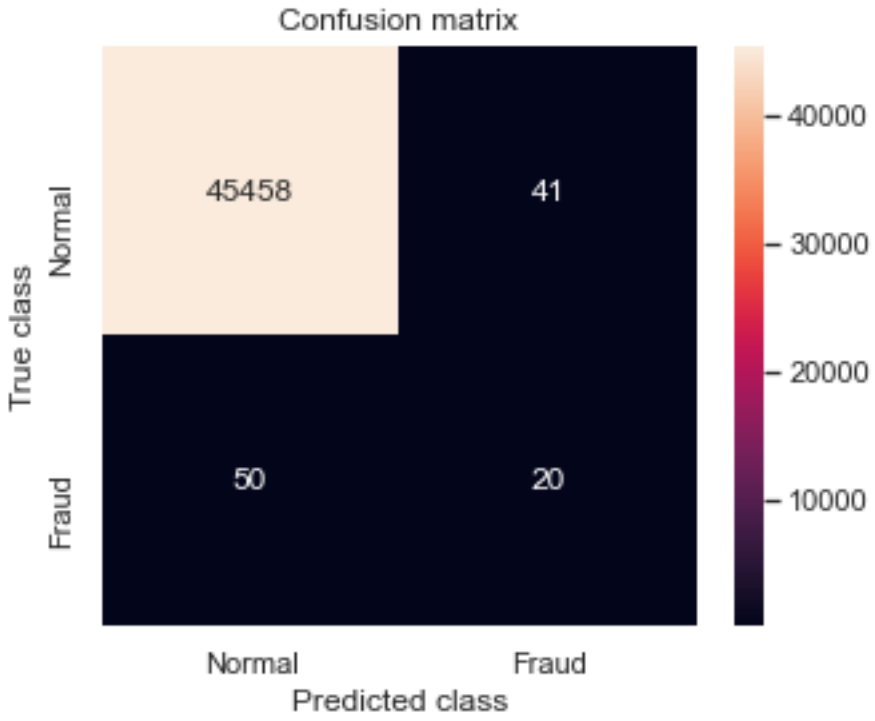
[107]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(Y_train, y_score)
```

[107]:
```
0.9565597852696466
```

95.6% score is thresold for Anomaly Detection

[108]:
```
# Reshape the prediction values to 0 for valid, 1 for fraud.
y_pred[y_pred == 1] = 0
y_pred[y_pred == -1] = 1
n_errors = (y_pred != Y_test).sum()
```

[109]:
```
#printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(Y_test, y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS,
 yticklabels=LABELS, annot=True, fmt='d');
plt.title('Confusion matrix')
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



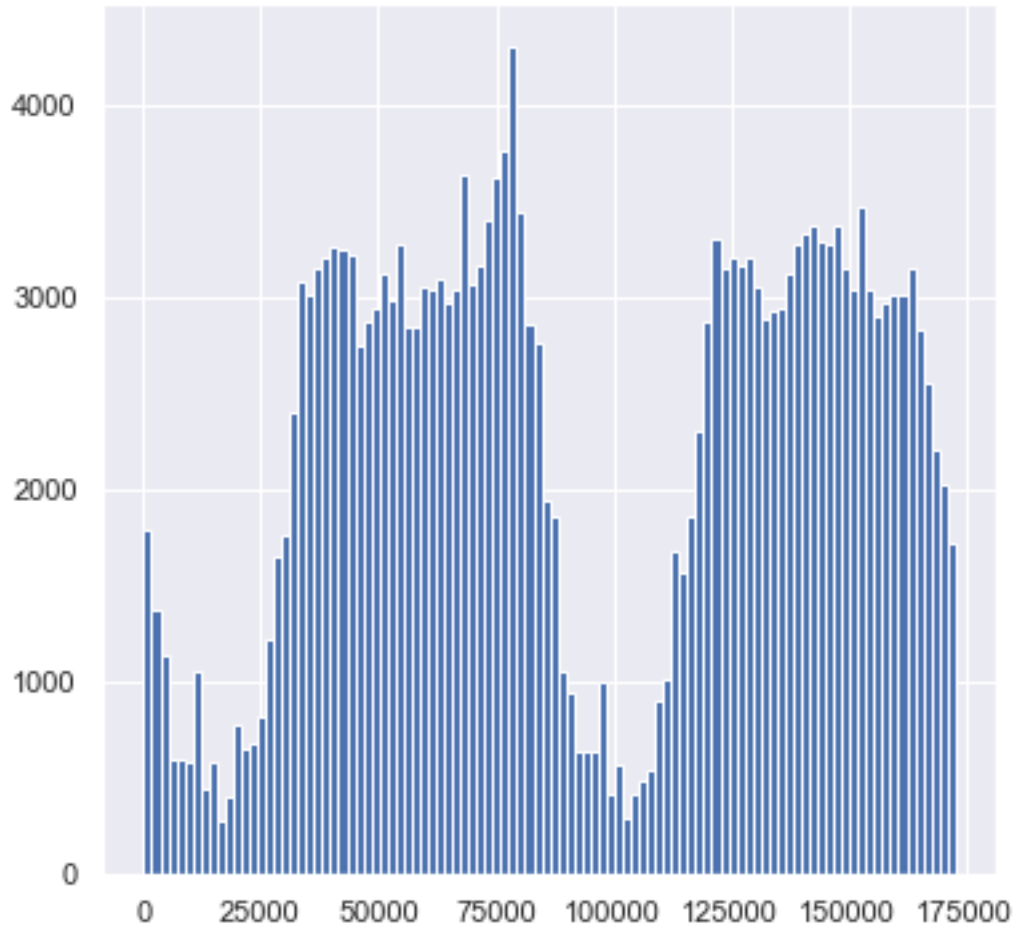
[110]:
```
#evaluation of the model
#printing every score of the classifier
#scoring in any thing
from sklearn.metrics import confusion_matrix
n_outliers = len(Fraud)
print('The Model used is {}'.format('Isolation Forest'))
acc= accuracy_score(Y_test,y_pred)
```

```
print('The accuracy is {}'.format(acc))
prec= precision_score(Y_test,y_pred)
print('The precision is {}'.format(prec))
rec= recall_score(Y_test,y_pred)
print('The recall is {}'.format(rec))
f1= f1_score(Y_test,y_pred)
print('The F1-Score is {}'.format(f1))
MCC=matthews_corrcoef(Y_test,y_pred)
print('The Matthews correlation coefficient is {}'.format(MCC))
```

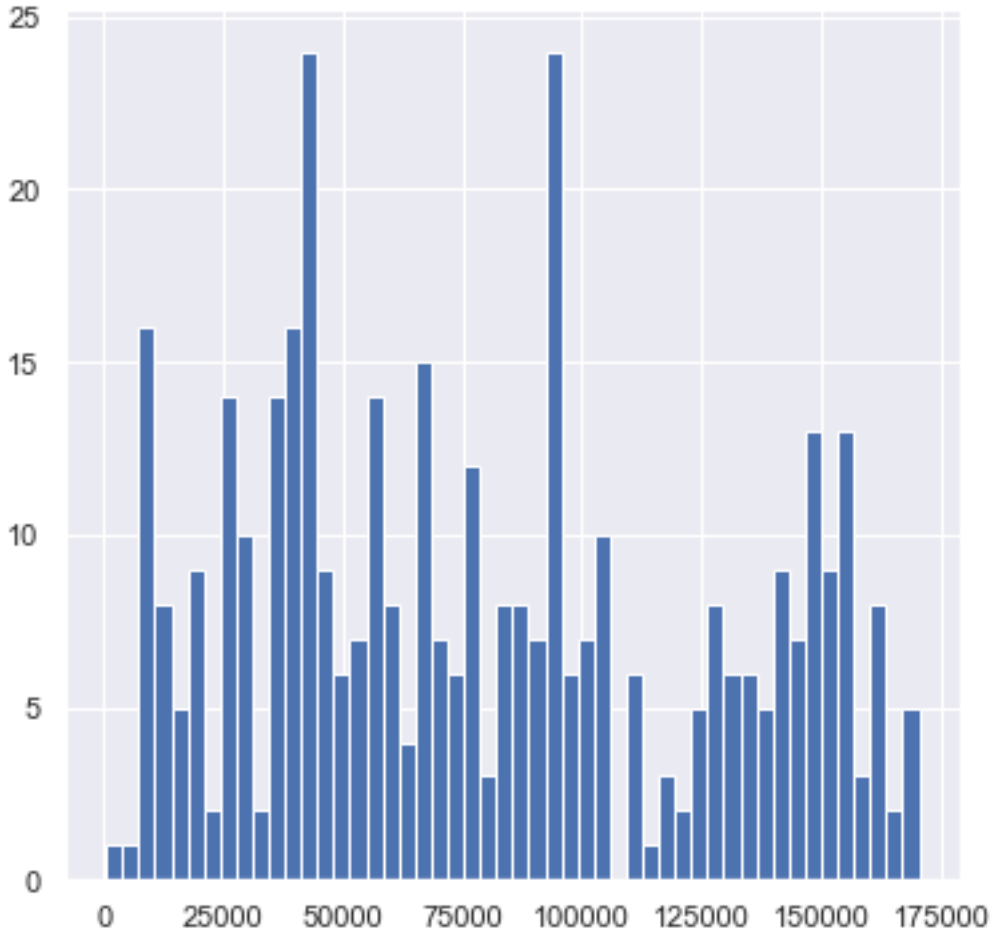
```
The Model used is Isolation Forest
The accuracy is 0.9980030283745529
The precision is 0.32786885245901637
The recall is 0.2857142857142857
The F1-Score is 0.30534351145038163
The Matthews correlation coefficient is 0.30507120439348184
```

[111]:
```
Valid["Time"].hist(bins=100, figsize=(6,6))
```

[111]: <AxesSubplot:>



[112]:
```
Fraud["Time"].hist(bins=50, figsize=(6,6));
```



Anomalies Detection using quartiles

[113]:
```
for x in df_train.columns.to_list()[1:29]:
    Q1 = np.percentile(df_train[x], 25, interpolation = 'midpoint')
    # Third quartile (Q3)
    Q3 = np.percentile(df_train[x], 75, interpolation = 'midpoint')
    # Interquaritle range (IQR)
    IQR = Q3 - Q1
    # print("IQR:",IQR*1.5, "|| Q3:",Q3,"|| Q1:",Q1)
    if IQR*1.5>Q3 or IQR*1.5<Q1:
        outliers_no = ((df_train[x] < (Q1 - 1.5 * IQR)) | (df_train[x] > (Q3 +␣
↪1.5 * IQR))).sum()
        print(x," have Anomalies and total no are: ",outliers_no )
```

```
    else:
        print(x," No Anomalies")
```

```
V1  have Anomalies and total no are:  5661
V2  have Anomalies and total no are:  10877
V3  have Anomalies and total no are:  2672
V4  have Anomalies and total no are:  8872
V5  have Anomalies and total no are:  9933
V6  have Anomalies and total no are:  18386
V7  have Anomalies and total no are:  7166
V8  have Anomalies and total no are:  19338
V9  have Anomalies and total no are:  6535
V10  have Anomalies and total no are:  7597
V11  have Anomalies and total no are:  619
V12  have Anomalies and total no are:  12277
V13  have Anomalies and total no are:  2737
V14  have Anomalies and total no are:  11252
V15  have Anomalies and total no are:  2291
V16  have Anomalies and total no are:  6486
V17  have Anomalies and total no are:  5968
V18  have Anomalies and total no are:  6033
V19  have Anomalies and total no are:  8054
V20  have Anomalies and total no are:  22193
V21  have Anomalies and total no are:  11614
V22  have Anomalies and total no are:  1033
V23  have Anomalies and total no are:  14888
V24  have Anomalies and total no are:  3813
V25  have Anomalies and total no are:  4286
V26  have Anomalies and total no are:  4523
V27  have Anomalies and total no are:  31365
V28  have Anomalies and total no are:  24275
```

1.4 Week 4

Visualize the scores for Fraudulent and Non-Fraudulent transactions.

[114]:
```
ann_predictions
```

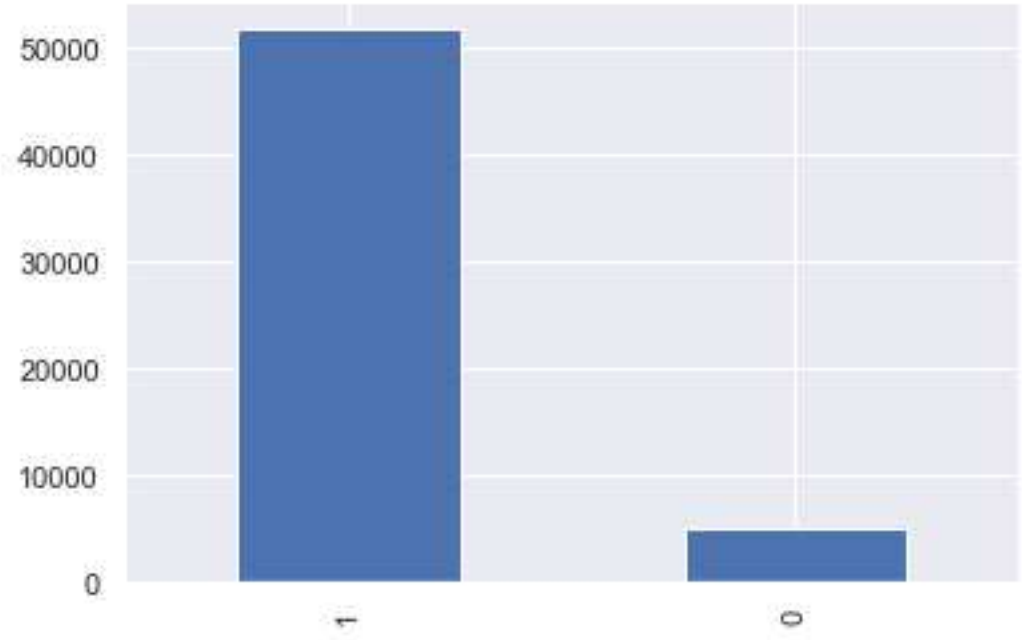
[114]:
```
array([[0.5008916 ],
       [0.50068694],
       [0.5001906 ],
       …,
       [0.5005281 ],
       [0.49995697],
       [0.50735885]], dtype=float32)
```

[115]:
```
ann_prediction_copy = [1 if x>=0.5 else 0 for x in ann_predictions]
ann_prediction_series = pd.Series(ann_prediction_copy)
ann_prediction_series
```

[115]:
```
0        1
1        1
2        1
3        1
4        1
        ..
56957    1
56958    0
56959    1
56960    0
56961    1
Length: 56962, dtype: int64
```

[116]:
```
# ann_prediction_series.plot(kind='bar')
ann_prediction_series.value_counts().plot(kind='bar')
```

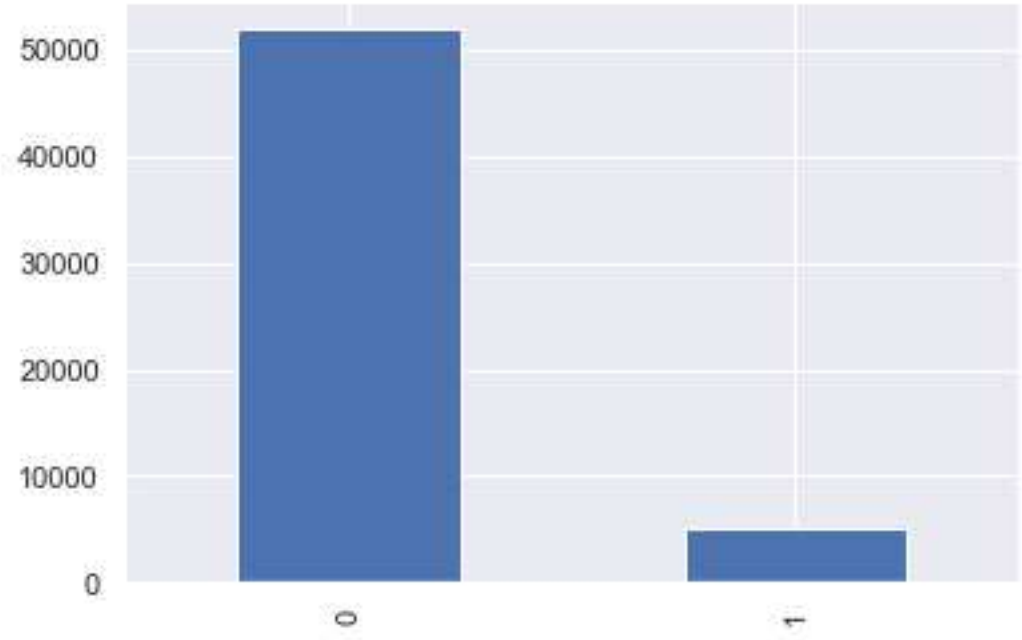
[116]:
```
<AxesSubplot:>
```



Predicted Hidden values

[117]:
```
ann_prediction_hidden = [1 if x>=0.5 else 0 for x in ann_predictions_hidden]
ann_prediction_series_hidden = pd.Series(ann_prediction_hidden)
ann_prediction_series_hidden.value_counts().plot(kind='bar')
```

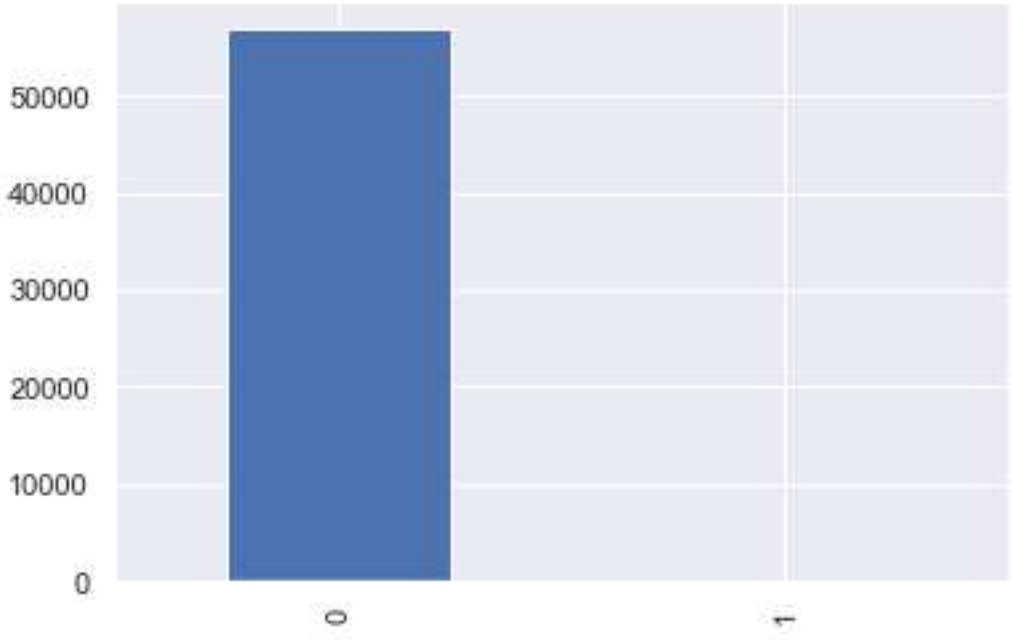
[117]: <AxesSubplot:>



1.4.1 Original hidden values

[118]:
```
df_train_hidden['Class'].value_counts().plot(kind='bar')
```

[118]: <AxesSubplot:>



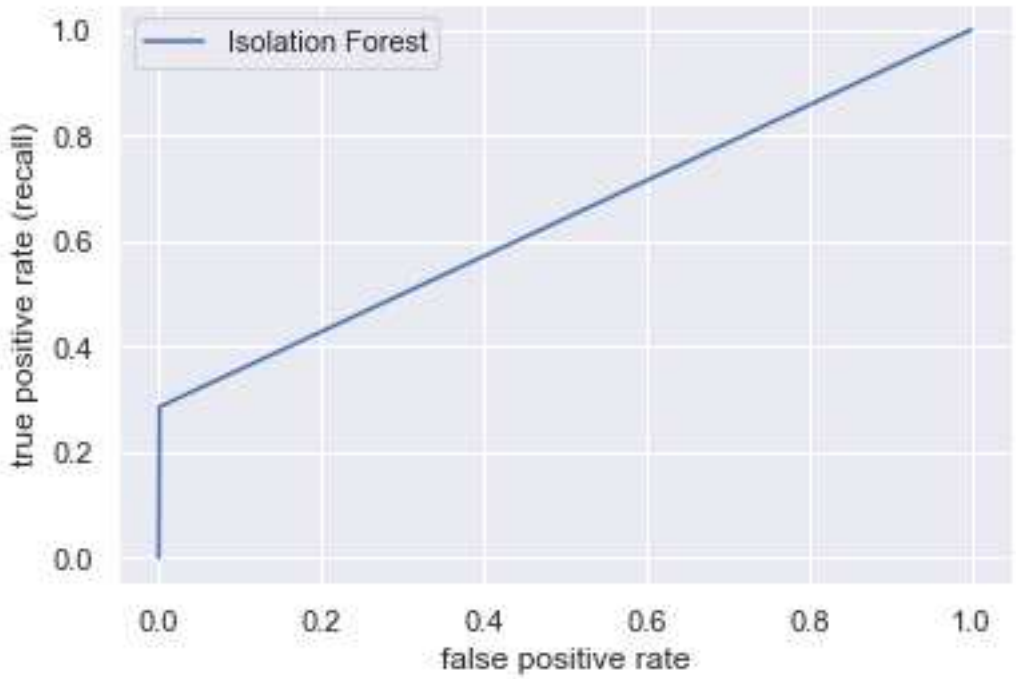
Find out the threshold value for marking or reporting a transaction as fraudulent in your anomaly detection system.

For this kind of unbalanced dataset, it is more reasonable to look at Area Under ROC curve, since it has the nice property to be 0.5 for random scoring and 1.0 for perfect scoring whatever the balancing.

*** 95.6% is auc score as threshold for Anoamly detection. Generated by Isolation forest.**

[119]:
```
fp, tp, thres = roc_curve(Y_test,y_pred)
plt.plot(fp, tp, label="Isolation Forest")

# fp, tp, thres = roc_curve(y, np.random.rand(len(y)))
# plt.plot(fp, tp, label="Random")

plt.xlabel("false positive rate")
plt.ylabel("true positive rate (recall)")
plt.legend()
```

[119]: <matplotlib.legend.Legend at 0x15b4260b688>



[120]:
```
print("\n",fp,"\n\n",tp)
```

```
[0.00000000e+00 9.01118706e-04 1.00000000e+00]

[0.         0.28571429 1.        ]
```

1.4.2 Can this score be used as an engineered feature in the models developed previously? Are there any incremental gains in F1-Score? Why or Why not?

*** F1 takes both false positives and false negatives into consideration. In imbalanced classes such as this, F1 is much more effective than accuracy at determining the performance of the model.**

*** F1 score for neural network model is better than traditional Machine Learning model.**

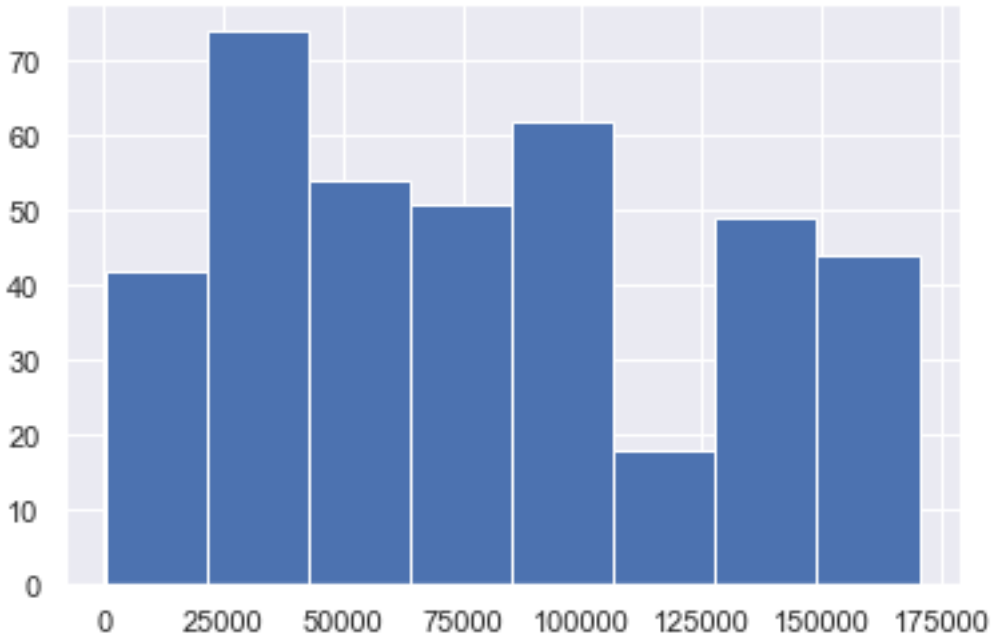
1.4.3 Be as creative as possible in finding other interesting insights.

[121]:
```
df_train = copy.deepcopy(df_train_original)
df_train
```

[121]:
```
            Time      V1      V2      V3      V4      V5      V6      V7      V8  \
0        38355.0   1.044   0.319   1.046   2.806  -0.561  -0.368   0.033  -0.042
1        22555.0  -1.665   0.808   1.806   1.903  -0.822   0.935  -0.825   0.976
2         2431.0  -0.324   0.602   0.865  -2.138   0.295  -1.252   1.072  -0.335
3        86773.0  -0.258   1.218  -0.585  -0.875   1.222  -0.311   1.074  -0.161
4       127202.0   2.142  -0.495  -1.937  -0.818  -0.025  -1.027  -0.152  -0.306
…            …       …       …       …       …       …       …       …       …
227840   62074.0  -1.994   1.735  -1.108  -2.672   1.605   3.042  -0.418   1.438
227841   32193.0  -0.440   1.063   1.582  -0.030   0.041  -0.904   0.730  -0.108
227842  163864.0   0.828  -2.649  -3.161   0.209  -0.561  -1.570   1.613  -0.930
227843  122571.0  -1.524  -6.287  -2.638   1.330  -1.672   1.958   1.359   0.082
227844   43440.0  -1.609   0.133   2.076  -1.937  -1.822  -0.430   0.247   0.684

              V9  …     V21     V22     V23     V24     V25     V26     V27     V28  \
0         -0.323  … -0.240  -0.680   0.085   0.685   0.319  -0.205   0.002   0.038
1          1.747  … -0.335  -0.511   0.036   0.148  -0.529  -0.567  -0.596  -0.220
2          1.071  …  0.012   0.353  -0.342  -0.146   0.094  -0.804   0.229  -0.022
3          0.201  … -0.425  -0.781   0.019   0.179  -0.316   0.097   0.270  -0.021
4         -0.869  …  0.010   0.022   0.079  -0.481   0.024  -0.279  -0.030  -0.044
…            …  …    …       …       …       …       …       …       …       …
227840   0.945  … -0.304  -0.708   0.047   1.008   0.234   0.769   0.698   0.355
227841  -0.513  … -0.216  -0.532  -0.025   0.383  -0.165   0.069   0.269   0.123
227842  -1.319  …  0.350   0.002  -0.747   0.172   0.248   0.937  -0.258   0.038
227843   0.753  …  1.329   0.001  -1.360  -1.508  -1.184   0.578  -0.329   0.230
227844   1.177  …  0.465   1.017   0.173   0.570   0.505  -0.660   0.175   0.092

          Amount  Class
0          49.67      0
1          16.94      0
2           1.00      0
3          10.78      0
4          39.96      0
…            …    …
227840     14.83      0
227841      2.58      0
227842    748.04      0
227843   1771.50      0
227844    191.80      0

[227845 rows x 31 columns]
```

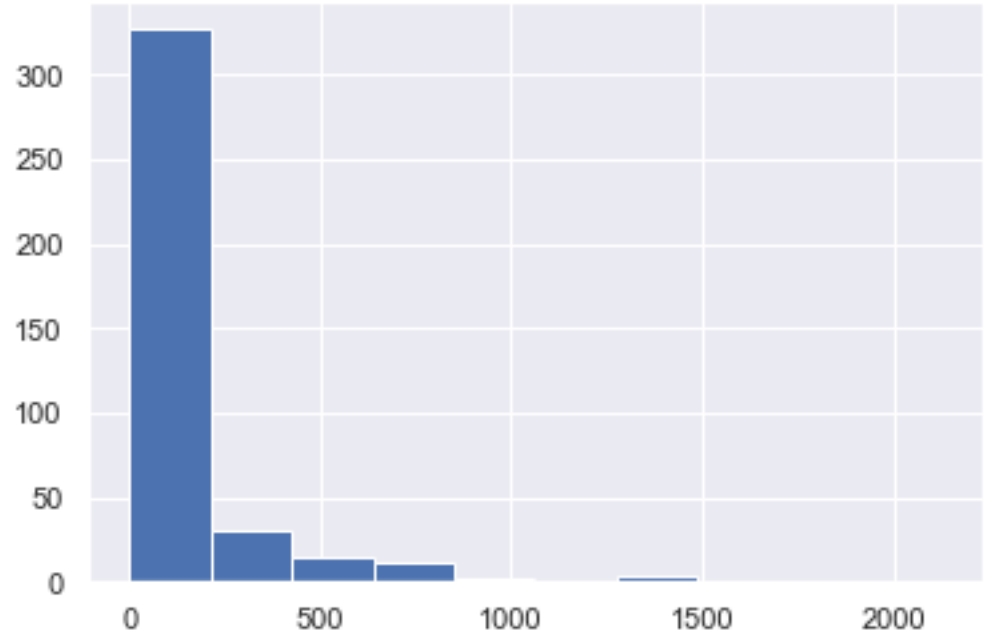
[122]:
```
df_train[df_train['Class']==1].Time.hist(bins=8)
```

[122]:
```
<AxesSubplot:>
```



[123]:
```
df_train[df_train['Class']==1].Amount.hist(bins=10)
```

[123]: <AxesSubplot:>

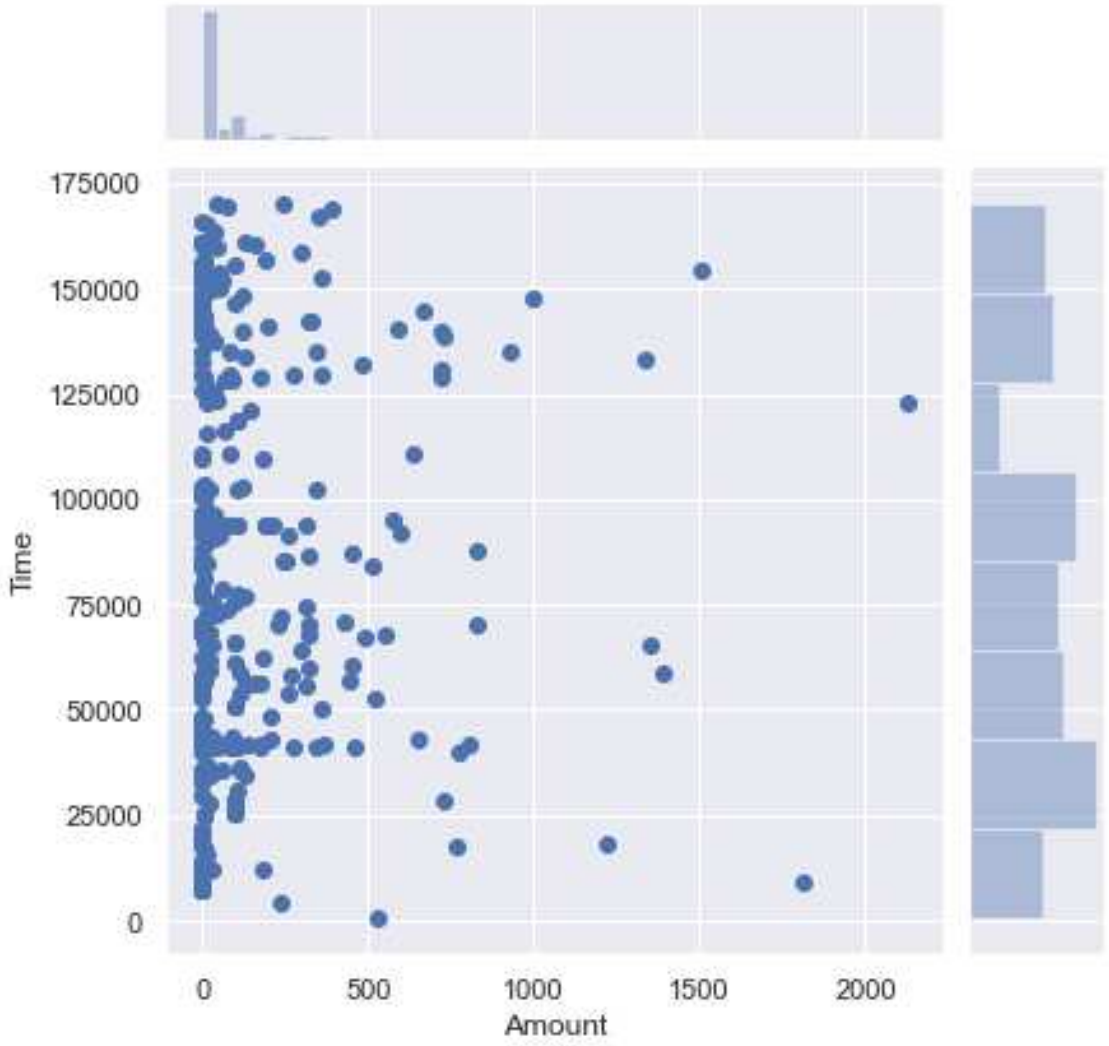


[124]:
```
df_train[df_train['Class']==1].Amount
```

[124]:
```
266        11.38
360       316.06
421         0.01
734         0.00
854         1.00
           ...
226409    175.90
226814    362.55
226817      1.00
226957      1.00
227531      1.00
Name: Amount, Length: 394, dtype: float64
```

[125]:
```
sns.jointplot(x="Amount", y="Time", data=df_train[df_train['Class']==1])
```

[125]:
```
<seaborn.axisgrid.JointGrid at 0x15b41c32048>
```



most of Fraud Transaction had less Amount

[ ]: