Credit Card Fraud Detection Project

Library Imports

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly import tools
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
```

Data Importing

In [32]:

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

In [34]:

```

```

Out[34]:

```
(94, 9)
```

PART 1 : EDA

Column and Row details

In [3]:

```
# column names
print(data.columns)
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V1
9', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amou
nt',
       'Class'],
      dtype='object')
```

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284806 entries, 0 to 284805
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284806 non-null  float64
 1   V1      284806 non-null  float64
 2   V2      284806 non-null  float64
 3   V3      284806 non-null  float64
 4   V4      284806 non-null  float64
 5   V5      284806 non-null  float64
 6   V6      284806 non-null  float64
 7   V7      284806 non-null  float64
 8   V8      284806 non-null  float64
 9   V9      284806 non-null  float64
 10  V10     284806 non-null  float64
 11  V11     284806 non-null  float64
 12  V12     284806 non-null  float64
 13  V13     284806 non-null  float64
 14  V14     284806 non-null  float64
 15  V15     284806 non-null  float64
 16  V16     284806 non-null  float64
 17  V17     284806 non-null  float64
 18  V18     284806 non-null  float64
 19  V19     284806 non-null  float64
 20  V20     284806 non-null  float64
 21  V21     284806 non-null  float64
 22  V22     284806 non-null  float64
 23  V23     284806 non-null  float64
 24  V24     284806 non-null  float64
 25  V25     284806 non-null  float64
 26  V26     284806 non-null  float64
 27  V27     284806 non-null  float64
 28  V28     284806 non-null  float64
 29  Amount  284806 non-null  float64
 30  Class   284806 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [5]:

```
# data dimensions
data.shape
```

Out[5]:

```
(284806, 31)
```

In [6]:

```
# summary stats on columns
data.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	
count	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000	2.848060
mean	94813.585781	0.000002	6.661837e-07	-0.000002	0.000002	4.405007
std	47488.004530	1.958699	1.651311e+00	1.516257	1.415871	1.380249
min	0.000000	-56.407510	-7.271573e+01	-48.325589	-5.683171	-1.137433
25%	54201.250000	-0.920374	-5.985522e-01	-0.890368	-0.848642	-6.915995
50%	84691.500000	0.018109	6.549621e-02	0.179846	-0.019845	-5.433621
75%	139320.000000	1.315645	8.037257e-01	1.027198	0.743348	6.119267
max	172788.000000	2.454930	2.205773e+01	9.382558	16.875344	3.480167

8 rows × 31 columns

In [7]:

```
# capture numeric columns and non numeric columns list
num_vars = data.columns[data.dtypes != object]
cat_vars = data.columns[data.dtypes == object]
```

In [8]:

```
# print numeric columns
print(num_vars)
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V1
9', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amou
nt',
       'Class'],
      dtype='object')
```

In [9]:

```
# print categorical columns
print(cat_vars)
```

```
Index([], dtype='object')
```

In [10]:

```
# count of  nulls in columns
data.isnull().sum()
```

Out[10]:

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [11]:

```
# function to get unique count per column in a data frame
def count_unique_values(df):

    total = df.count()
    temp = pd.DataFrame(total)
    temp.columns = ['Total']            # Count total number of non-null values

    uniques = []
    for col in df.columns:
        unique = df[col].nunique()    # Get unique values for each column
        uniques.append(unique)
    temp['Uniques'] = uniques

    return(np.transpose(temp))
```

In [12]:

```
# get count of unique values for dataframe
count_unique_values(data)
```

Out[12]:

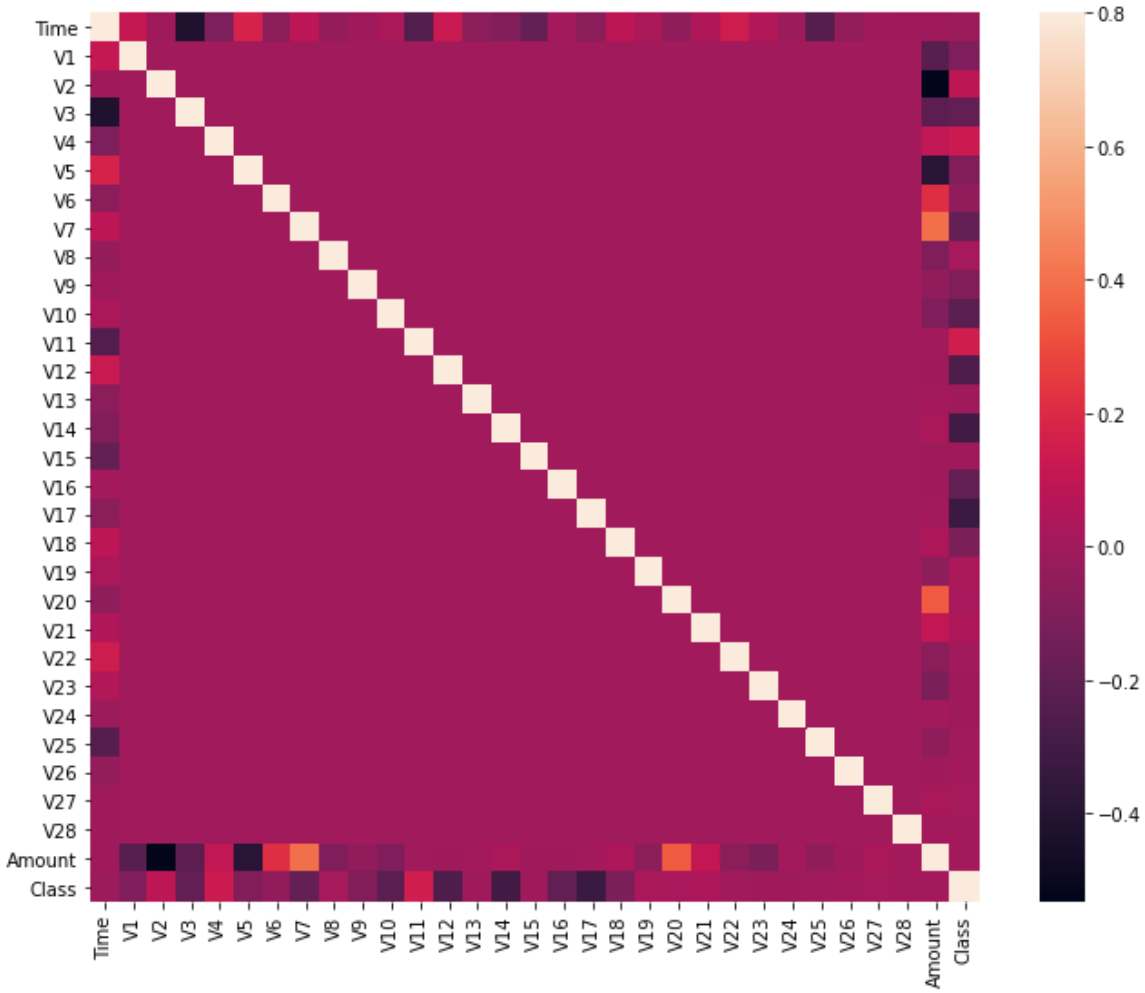
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
Total	284806	284806	284806	284806	284806	284806	284806	284806	284806	284806
Uniques	124591	275652	275654	275656	275653	275656	275651	275650	275642	275655

2 rows × 31 columns

Distributions and Correlations

In [13]:

```
# correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))

sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```



Plot shows mostly that the columns are not correlated

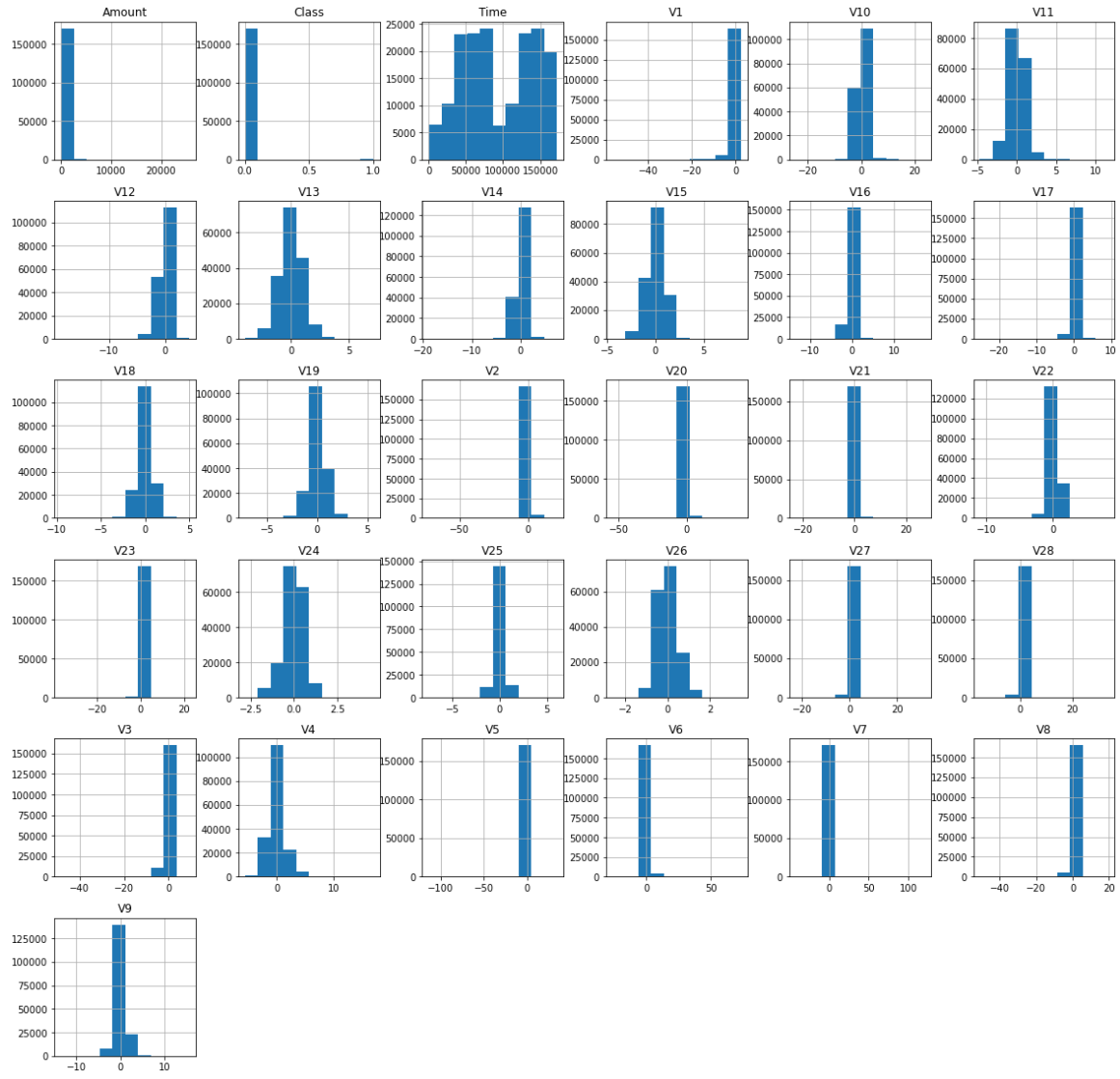
In [14]:

```
# Histograms of each variable :  0.2 random sample of rows selected
# ------------------------------------------------------------------

# random_state helps assure that you always get the same output when you split t
he data
# this helps create reproducible results and it does not actually matter what th
e number is
# frac is percentage of the data that will be returned
data_part = data.sample(frac = 0.6, random_state = 1)
print(data_part.shape)
```

```
(170884, 31)
```

In [15]:

```
# plot the histogram of each parameter
data_part.hist(figsize = (20, 20))
plt.show()
```



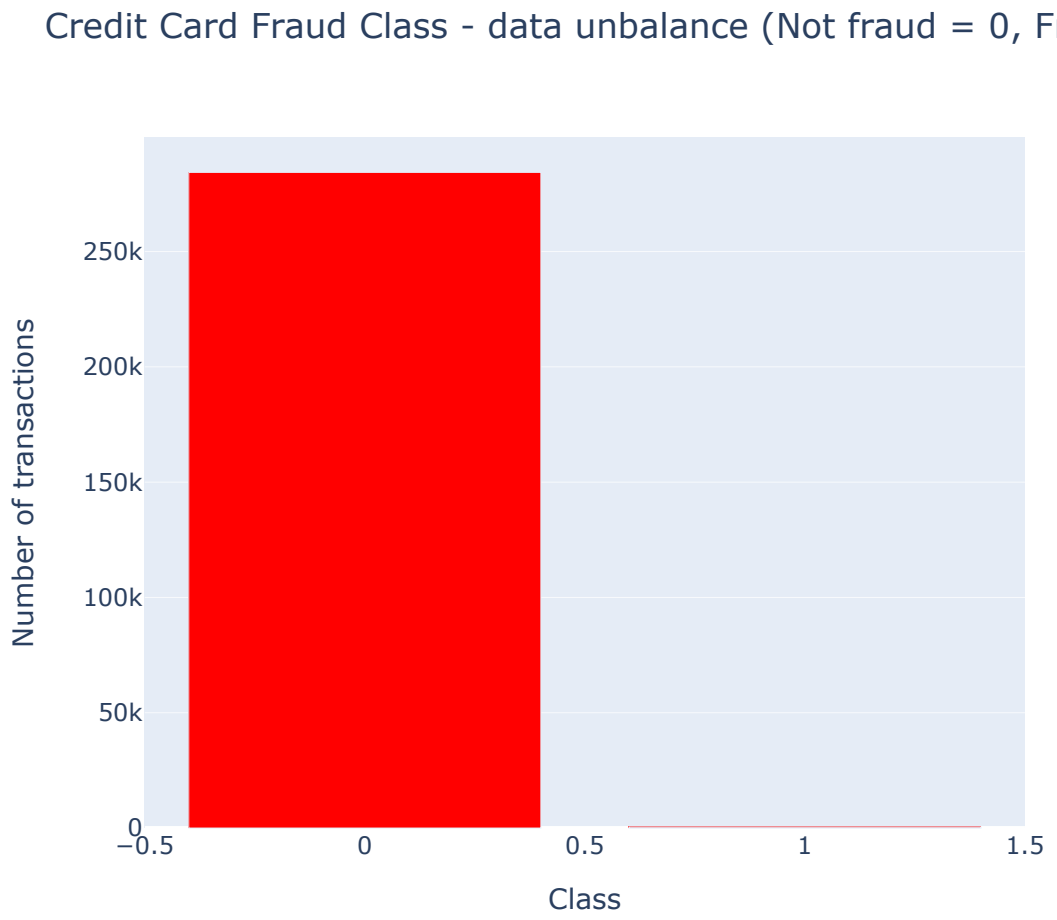
You can see most of the V's are clustered around 0 with some or no outliers. Notice we have very few fraudulent cases over valid cases in our class histogram.

Feature Explorations

Target Variable distribution studies

In [16]:

```
temp = data["Class"].value_counts()
df = pd.DataFrame({'Class': temp.index,'values': temp.values})

trace = go.Bar(
    x = df['Class'],y = df['values'],
    name="Credit Card Fraud Class - data unbalance (Not fraud = 0, Fraud = 1)",
    marker=dict(color="Red"),
    text=df['values']
)
data_fig = [trace]
layout = dict(title = 'Credit Card Fraud Class - data unbalance (Not fraud = 0,
 Fraud = 1)',
          xaxis = dict(title = 'Class', showticklabels=True),
          yaxis = dict(title = 'Number of transactions'),
          hovermode = 'closest',width=600
         )
fig = dict(data=data_fig, layout=layout)
iplot(fig, filename='class')
```



In [17]:

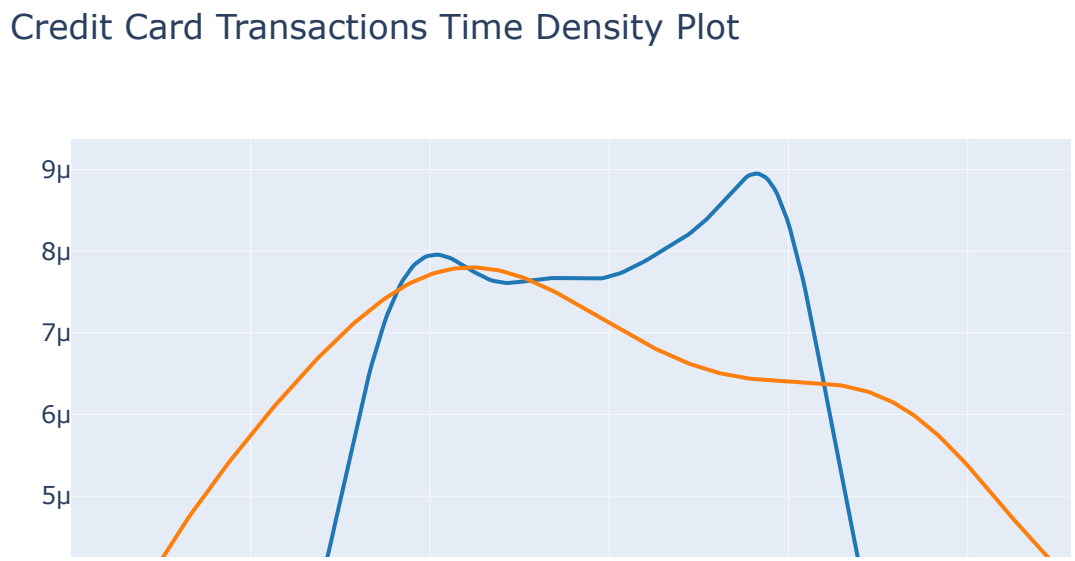
```
# determine the number of fraud vs not fraud cases
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]

outlier_fraction = len(fraud) / float(len(valid))

# ------------------------
print('outlier_fraction : {}'.format(outlier_fraction))
print('Fraud Cases: {}'.format(len(fraud)))
print('Valid Cases: {}'.format(len(valid)))
```

```
outlier_fraction : 0.0017304810878113635
Fraud Cases: 492
Valid Cases: 284314
```

Explorations in Time : Fraud Vs Non Fraud

In [18]:

```
# Separate Time col for both the classes
class_0 = data.loc[data['Class'] == 0]["Time"]
class_1 = data.loc[data['Class'] == 1]["Time"]

hist_data = [class_0, class_1]
group_labels = ['Not Fraud', 'Fraud']

fig = ff.create_distplot(hist_data, group_labels, show_hist=False, show_rug=Fals
e)
fig['layout'].update(title='Credit Card Transactions Time Density Plot', xaxis=d
ict(title='Time [s]'))
iplot(fig, filename='dist_only')#
```



Observations from the above plot :

1. Fraud transactions show a more even spread than the Non Fraudulent transactions

In [19]:

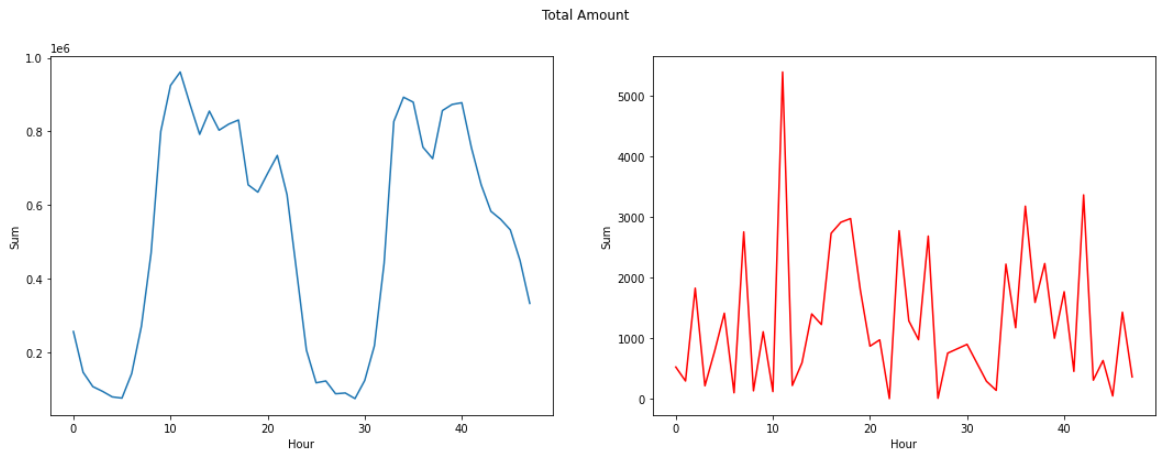
```
# Study Hourly Transaction Amount patterns for Fraud vs Non Fraud
data['Hour'] = data['Time'].apply(lambda x: np.floor(x / 3600))

# Create a temp data frame With Hour , Class
tmp = data.groupby(['Hour', 'Class'])['Amount'].aggregate(['min', 'max', 'count'
, 'sum', 'mean', 'median', 'var']).reset_index()
df = pd.DataFrame(tmp)
df.columns = ['Hour', 'Class', 'Min', 'Max', 'Transactions', 'Sum', 'Mean', 'Med
ian', 'Var']
df.head()
```

Out[19]:

	Hour	Class	Min	Max	Transactions	Sum	Mean	Median	Var
0	0.0	0	0.0	7712.43	3961	256572.87	64.774772	12.990	45615.821201
1	0.0	1	0.0	529.00	2	529.00	264.500000	264.500	139920.500000
2	1.0	0	0.0	1769.69	2215	145806.76	65.826980	22.820	20053.615770
3	1.0	1	59.0	239.93	2	298.93	149.465000	149.465	16367.832450
4	2.0	0	0.0	4002.88	1555	106989.39	68.803466	17.900	45355.430437

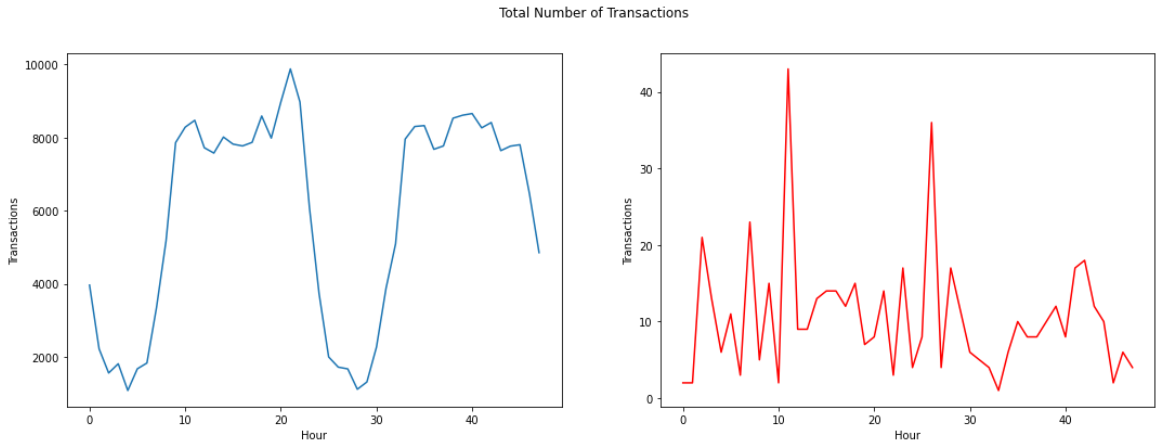
In [20]:

```
# Plot Total  Tran Amount Vs Hour : For Normal and Fraud Transactions

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Sum", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Sum", data=df.loc[df.Class==1], color="r
ed")
plt.suptitle("Total Amount")
plt.show();
```



In [21]:

```
# Plot Count for Transactions  Vs Hour : For Normal and Fraud Transactions

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Transactions", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Transactions", data=df.loc[df.Class==1],
color="red")
plt.suptitle("Total Number of Transactions")
plt.show();
```

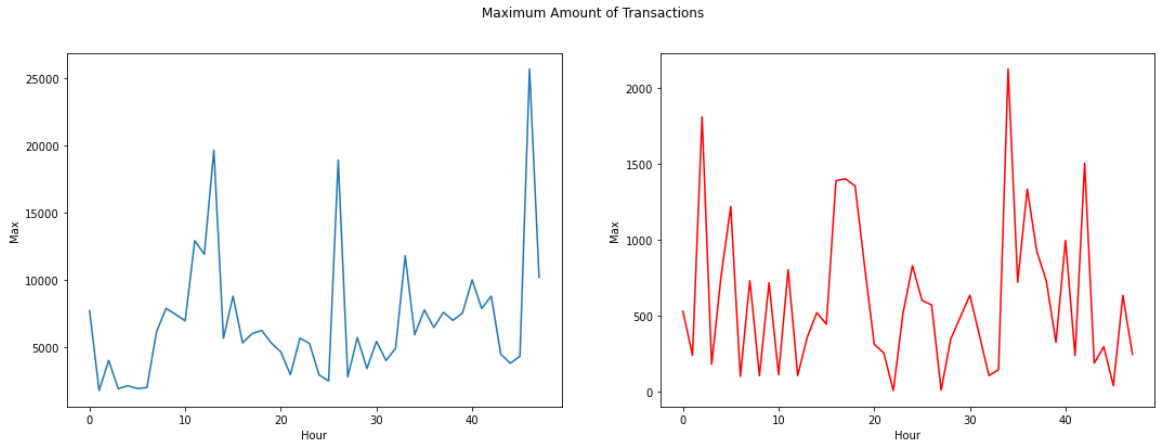


In [22]:

```
# Plot Total  Avg  Amount Vs Hour : For Normal and Fraud Transactions

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Mean", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Mean", data=df.loc[df.Class==1], color=
"red")
plt.suptitle("Average Amount of Transactions")
plt.show();
```

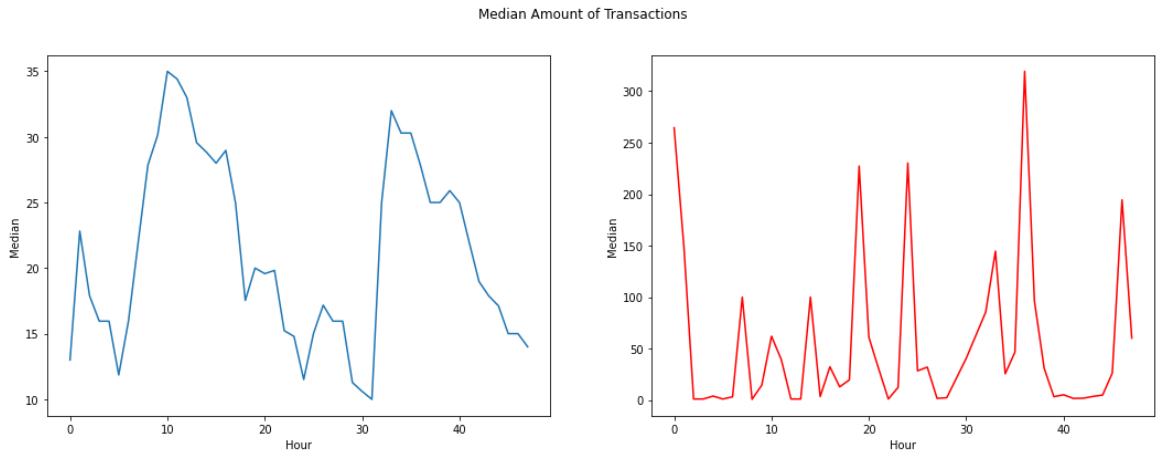
In [23]:

```
# Plot Max Amt of Transactions vs Hour for good Transactions Vs Fraud ones

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Max", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Max", data=df.loc[df.Class==1], color="r
ed")
plt.suptitle("Maximum Amount of Transactions")
plt.show();
```



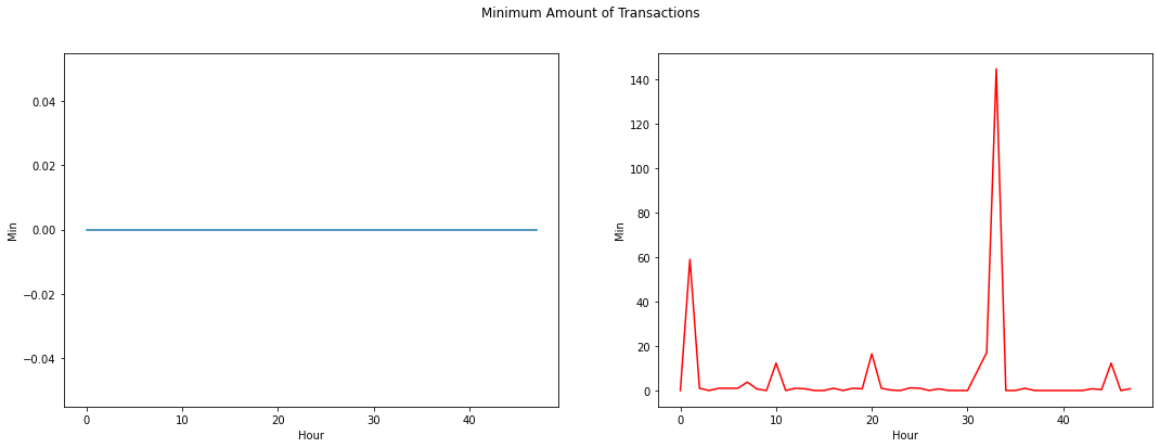
In [24]:

```
# Plot Median Amt of Transactions vs Hour

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Median", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Median", data=df.loc[df.Class==1], color
="red")
plt.suptitle("Median Amount of Transactions")
plt.show();
```



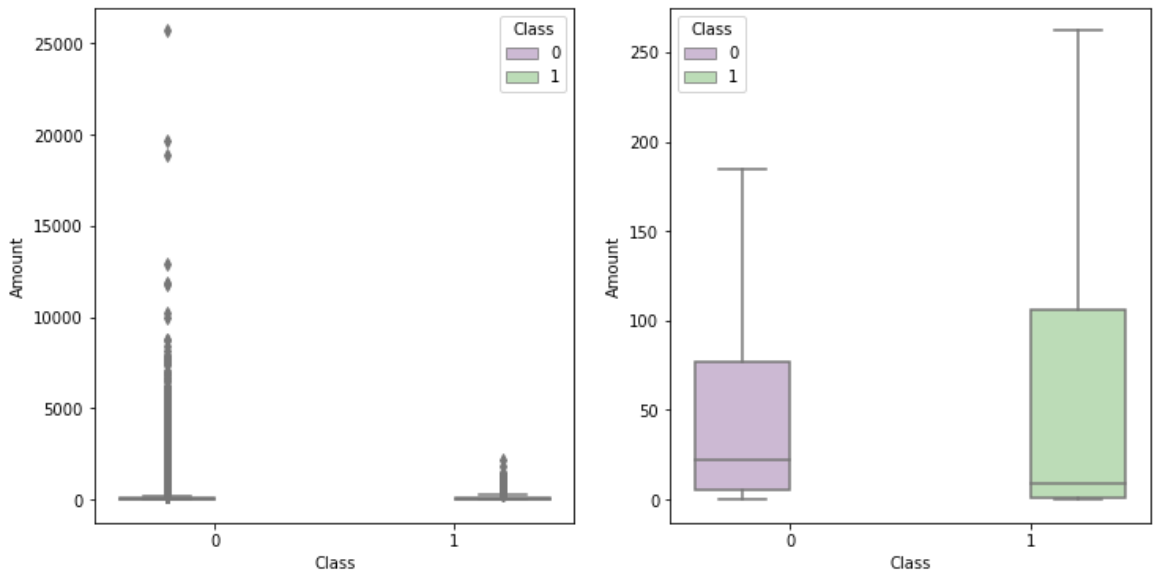
In [25]:

```
# Plot Minimum Amt of Transactions vs Hour

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Min", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Min", data=df.loc[df.Class==1], color="r
ed")
plt.suptitle("Minimum Amount of Transactions")
plt.show();
```



In [26]:

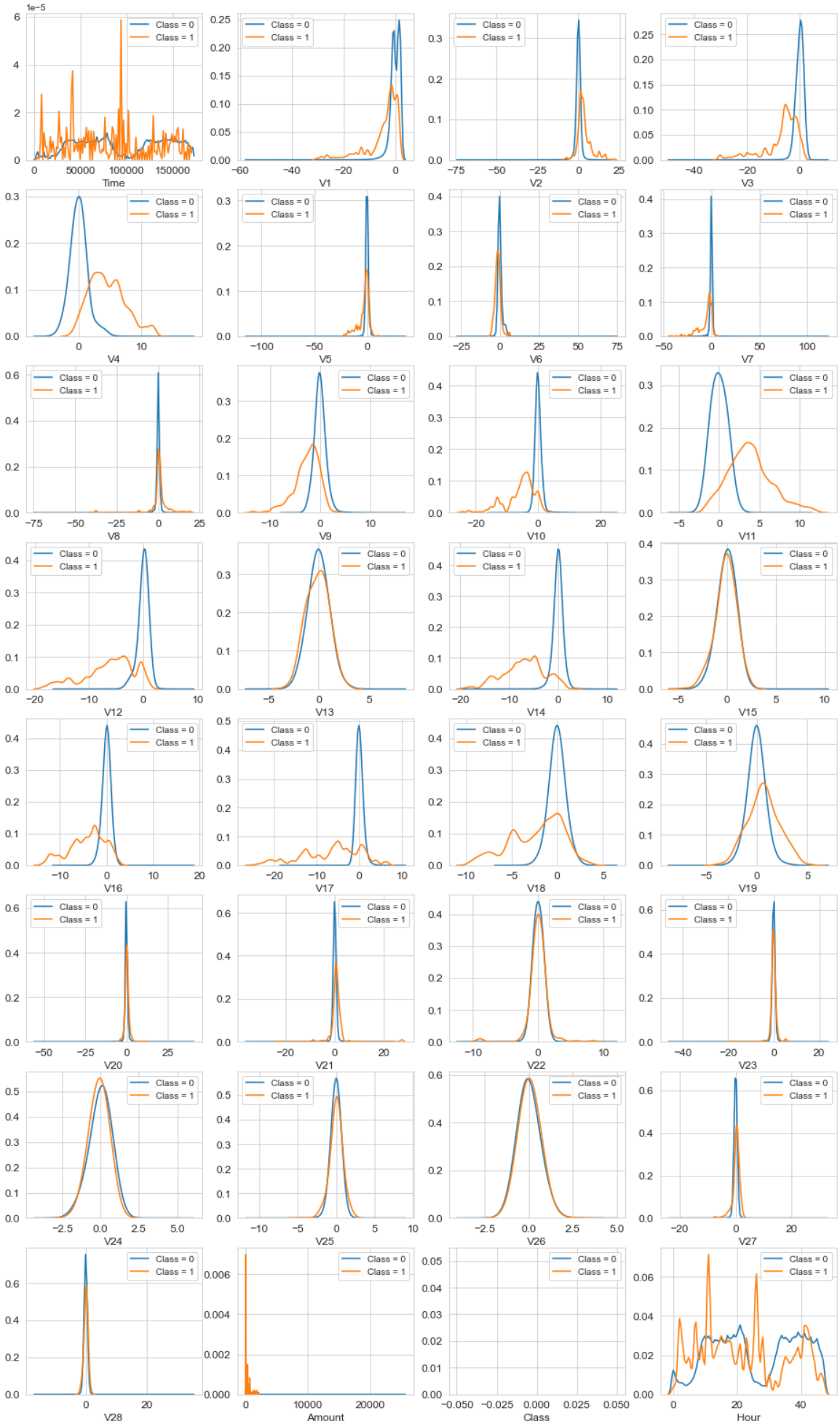
```
# Box plots on Amount Vs class

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))
s = sns.boxplot(ax = ax1, x="Class", y="Amount", hue="Class",data=data, palette=
"PRGn",showfliers=True)
s = sns.boxplot(ax = ax2, x="Class", y="Amount", hue="Class",data=data, palette=
"PRGn",showfliers=False)
plt.show();
```



In [27]:

```
var = data.columns.values

i = 0
t0 = data.loc[data['Class'] == 0]
t1 = data.loc[data['Class'] == 1]

sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(8,4,figsize=(16,28))

for feature in var:
    i += 1
    plt.subplot(8,4,i)
    sns.kdeplot(t0[feature], bw=0.5,label="Class = 0")
    sns.kdeplot(t1[feature], bw=0.5,label="Class = 1")
    plt.xlabel(feature, fontsize=12)
    locs, labels = plt.xticks()
    plt.tick_params(axis='both', which='major', labelsize=12)
plt.show()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:
283: UserWarning:

Data must have variance to compute a kernel density estimate.

/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:
283: UserWarning:

Data must have variance to compute a kernel density estimate.


<Figure size 432x288 with 0 Axes>
```


Class = 0
Class = 1
Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class
Hour

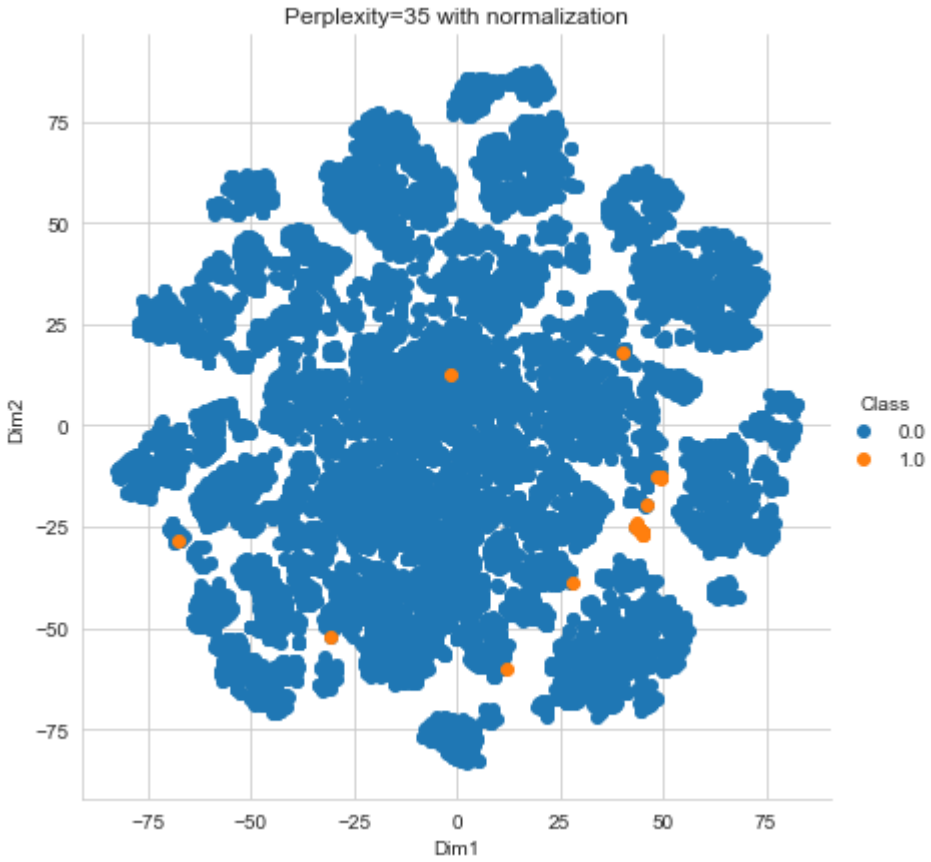
Obeservation. V4 and V11 have a clear separation of classes

Checking Separation of classes with TSNA Plot

In [37]:

```
from sklearn.manifold import TSNE
from time import time

from sklearn.preprocessing import StandardScaler

sample_features = data.sample(20000)
# sample_features = df
sample_class = sample_features.Class
sample_class = sample_class[:,np.newaxis]
sample_features = sample_features.drop('Class',axis=1)
scr = StandardScaler()
sample_features = scr.fit_transform(sample_features)
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=35)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))

final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legen
d()
plt.title("Perplexity=35 with normalization")
plt.show()
```

```
(20000, 30) (20000, 1)
TSNE done in 86.256s.
(20000, 3)

/opt/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py:243:
UserWarning:

The `size` parameter has been renamed to `height`; please update you
r code.
```



In [ ]:

CREDIT CARD FRAUD ANALYSIS

PART 2 : Handling Class Imbalance

Imports

In [1]:

```
# Imported Libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
#import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import matplotlib.patches as mpatches
import time

# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import collections


# Other Libraries
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_sco
re, accuracy_score, classification_report
from collections import Counter
from sklearn.model_selection import KFold, StratifiedKFold
import warnings
warnings.filterwarnings("ignore")
```

Data Reads

In [2]:

```
df = pd.read_csv('creditcard_data.csv')
```

PART 2 : Handling Class Imbalance

Check for class imbalance

In [3]:

```
# Check for class Imbalance

# The classes are heavily skewed we need to solve this issue later.
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of t
he dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the
 dataset')
```

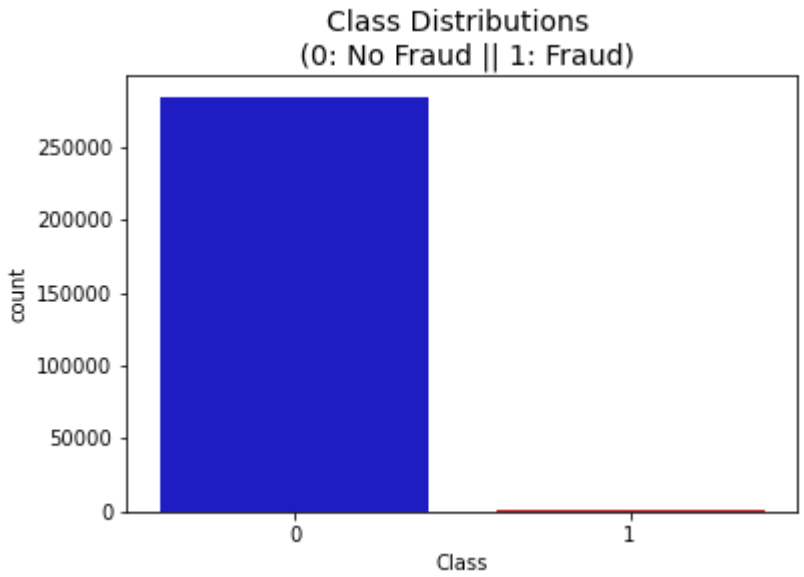
```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
```

In [4]:

```
# Check Visual for the class distribution

colors = ["#0101DF", "#DF0101"]
sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

Out[4]:

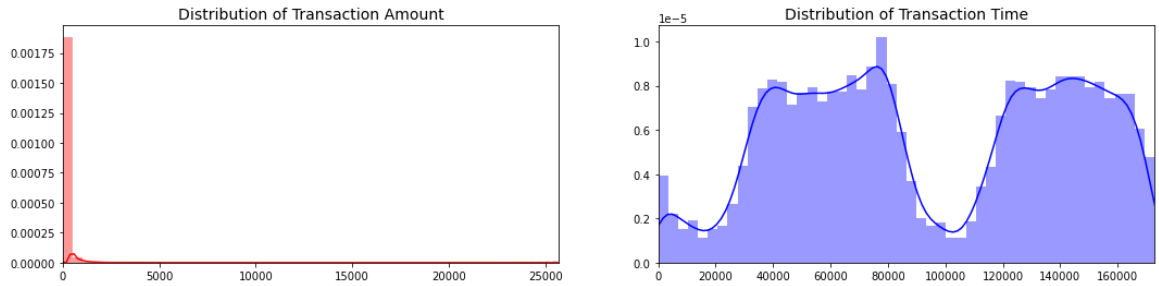
```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```



Check distribution for Amount and Time field

In [5]:

```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = df['Amount'].values
time_val = df['Time'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])

plt.show()
```



Observation : Most of the Transactions are smaller amount : The ferquency of Transactions have a clear peak pattern depending upon the time of the day

Scale Variables Time and Amount

In [6]:

```
# Scale using Robust scaler as it is less prone to outliers

# Scale the columns that are left to scale (Amount and Time)
from sklearn.preprocessing import StandardScaler, RobustScaler

# RobustScaler is less prone to outliers.

#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()
```

Out[6]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81

5 rows × 31 columns

Split original data set for testing

In [7]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit


# Create X an y Data
X = df.drop('Class', axis=1)
y = df['Class']

sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for train_index, test_index in sss.split(X, y):
    print("Train:", train_index, "Test:", test_index)
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

# We already have X_train and y_train for undersample data thats why I am using
 original to distinguish and to not overwrite these variables.
# original_Xtrain, original_Xtest, original_ytrain, original_ytest = train_test_
split(X, y, test_size=0.2, random_state=42)

# Check the Distribution of the labels


# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values


# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_count
s=True)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=T
rue)
print('-' * 100)

print('Label Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
```

```
Train: [ 30473  30496  31002 ... 284803 284804 284805] Test: [    0
1     2 ... 57017 57018 57019]
Train: [     0      1      2 ... 284803 284804 284805] Test: [ 30473
30496  31002 ... 113964 113965 113966]
Train: [     0      1      2 ... 284803 284804 284805] Test: [ 81186
81609  82400 ... 170946 170947 170948]
Train: [     0      1      2 ... 284803 284804 284805] Test: [150647
150654 150660 ... 227866 227867 227868]
Train: [     0      1      2 ... 227866 227867 227868] Test: [208651
212516 212644 ... 284803 284804 284805]
----------------------------------------------------------------------
--------------------------------
Label Distributions:

[0.99827514 0.00172486]
[0.99826197 0.00173803]
```

CLASS IMBALANCE HANDLING APPROACHES

1. Random Undersampling

Steps:

- The first thing we have to do is determine how **imbalanced** is our class (use "value_counts()" on the class column to determine the amount for each label)
- Once we determine how many instances are considered **fraud transactions** (Fraud = "1") , we should bring the **non-fraud transactions** to the same amount as fraud transactions (assuming we want a 50/50 ratio), this will be equivalent to 492 cases of fraud and 492 cases of non-fraud transactions.
- After implementing this technique, we have a sub-sample of our dataframe with a 50/50 ratio with regards to our classes. Then the next step we will implement is to **shuffle the data** to see if our models can maintain a certain accuracy everytime we run this script.

Note: The main issue with "Random Under-Sampling" is that we run the risk that our classification models will not perform as accurate as we would like to since there is a great deal of **information loss** (bringing 492 non-fraud transaction from 284,315 non-fraud transaction)

In [8]:

```
# Create Undersample and Equally represented dataset
# 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][:492]

# combine the fraud and non fraud rows
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()
```

Out[8]:

	V1	V2	V3	V4	V5	V6	V7	
260781	-1.164409	-1.279381	-1.304220	-1.200687	-0.779834	0.229462	0.985915	0.0
154720	-5.552122	5.678134	-9.775528	8.416295	-4.409844	-1.506235	-6.899839	3.7
108353	0.826988	-0.317576	0.261861	1.125783	-0.209647	0.191533	0.100275	0.1
151006	-26.457745	16.497472	-30.177317	8.904157	-17.892600	-1.227904	-31.197329	-11.4
163586	0.949241	1.333519	-4.855402	1.835006	-1.053245	-2.562826	-2.286986	0.2

5 rows × 31 columns

Undersampling using imblearn

TBD

Check for class distribution after Under Sampling

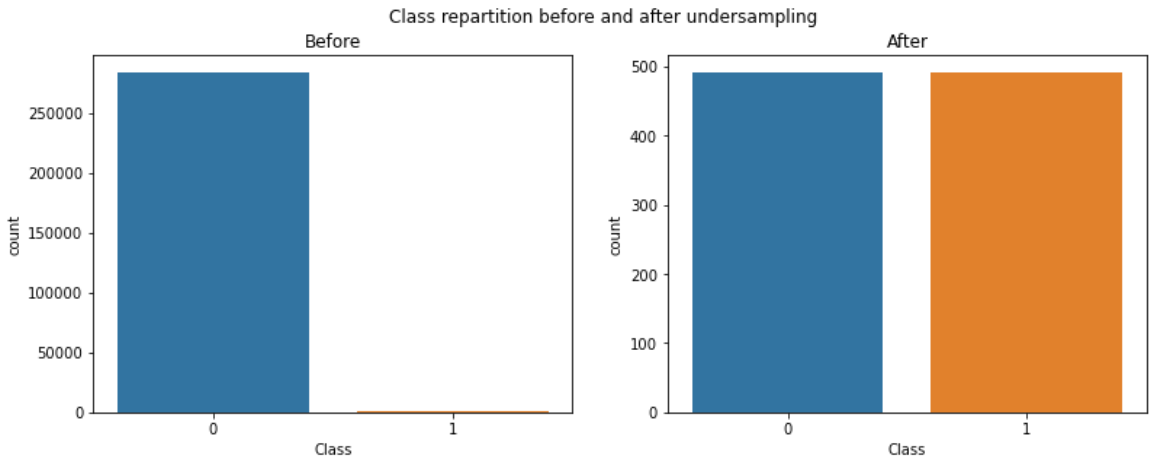
In [9]:

```
# Check class distribution after undersampling
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))


#visualizing undersampling results
fig, axs = plt.subplots(ncols=2, figsize=(13,4.5))
sns.countplot(x="Class", data=df, ax=axs[0])
sns.countplot(x="Class", data=new_df, ax=axs[1])

fig.suptitle("Class repartition before and after undersampling")
a1=fig.axes[0]
a1.set_title("Before")
a2=fig.axes[1]
a2.set_title("After")
```

```
Distribution of the Classes in the subsample dataset
1    0.5
0    0.5
Name: Class, dtype: float64
```

Out[9]:

```
Text(0.5, 1.0, 'After')
```

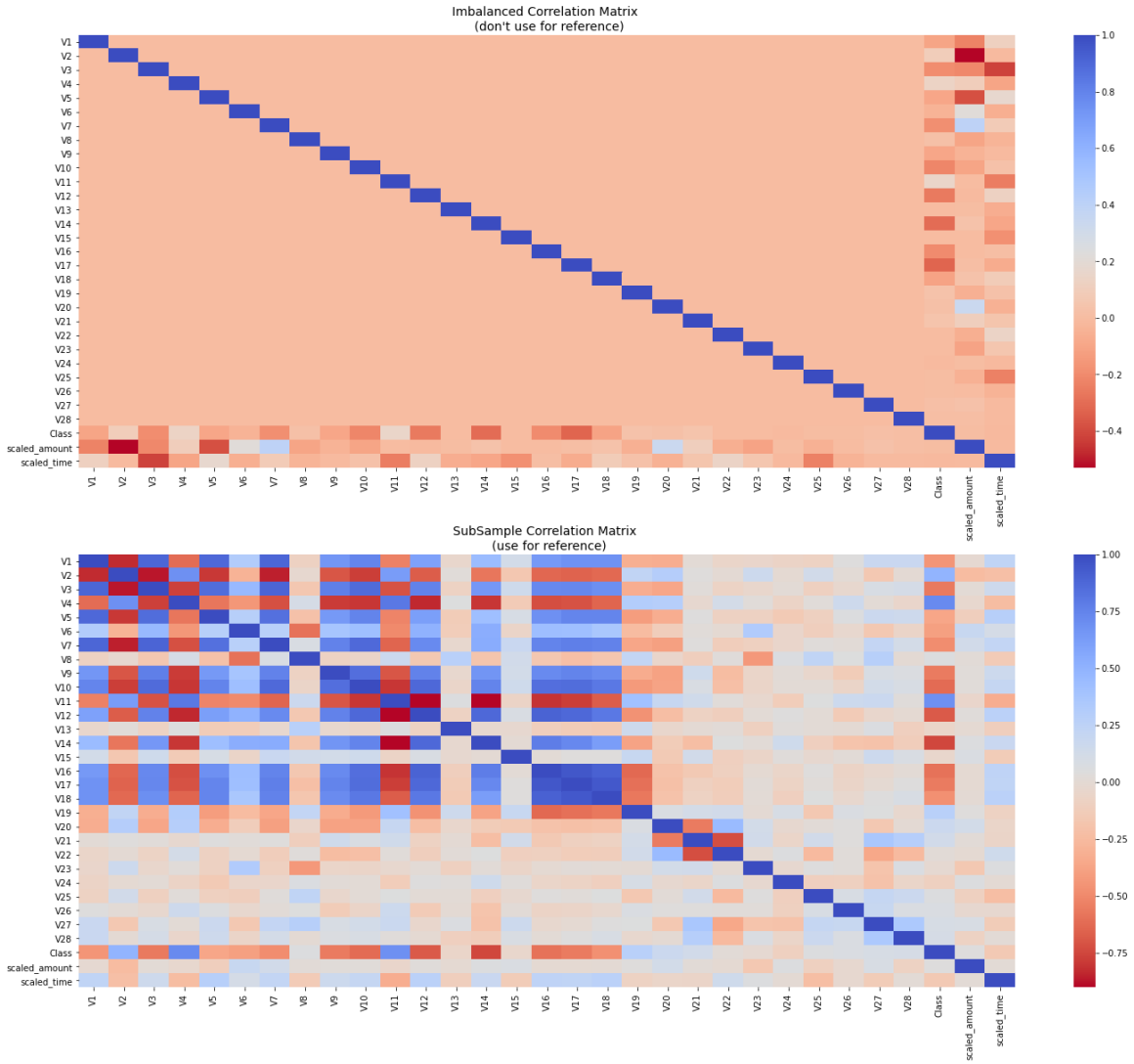


Check correlation matrix with balanced data set

https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb (https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb)

In [10]:

```
# Make sure we use the subsample in our correlation

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)", font
size=14)

#. Balanced Sub Set
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()
```



Observations from the Reference Heatmap

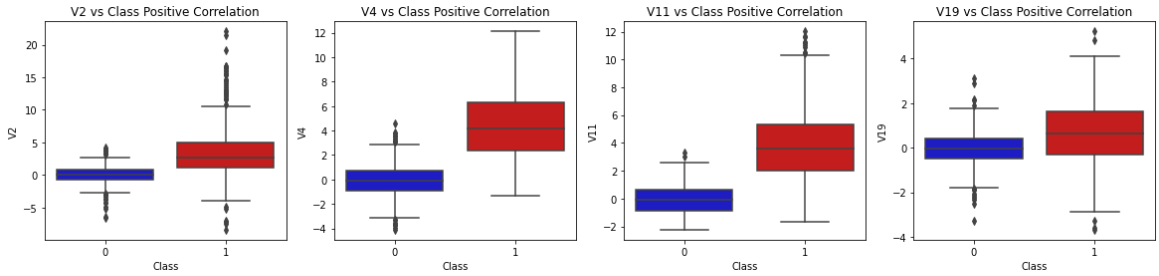
Positive correlation with Y Variable V2 ;V4; V11 ; V19

Negative correlation with Y Variable V17, V16, V14, V12 and V10

Box plots to Study Distributions by class for Highly correlated variables

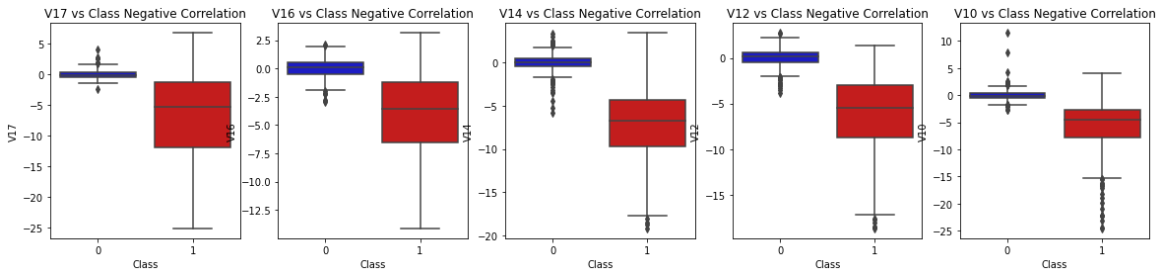
In [11]:

```
## Positively correlated variables
f, axes = plt.subplots(ncols=4, figsize=(20,4))

# Positive correlations (The higher the feature the probability increases that i
t will be a fraud transaction)
sns.boxplot(x="Class", y="V2", data=new_df, palette=colors, ax=axes[0])
axes[0].set_title('V2 vs Class Positive Correlation')

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

sns.boxplot(x="Class", y="V11", data=new_df, palette=colors, ax=axes[2])
axes[2].set_title('V11 vs Class Positive Correlation')


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

plt.show()
```



In [12]:

```
## Negatively correlated variables
#---------------------------------
# Negative Correlations with our Class (The lower our feature value the more lik
ely it will be a fraud transaction)

f, axes = plt.subplots(ncols=5, figsize=(20,4))

sns.boxplot(x="Class", y="V17", data=new_df, palette=colors, ax=axes[0])
axes[0].set_title('V17 vs Class Negative Correlation')

sns.boxplot(x="Class", y="V16", data=new_df, palette=colors, ax=axes[1])
axes[1].set_title('V16 vs Class Negative Correlation')

sns.boxplot(x="Class", y="V14", data=new_df, palette=colors, ax=axes[2])
axes[2].set_title('V14 vs Class Negative Correlation')


sns.boxplot(x="Class", y="V12", data=new_df, palette=colors, ax=axes[3])
axes[3].set_title('V12 vs Class Negative Correlation')


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

plt.show()
```



Observation :. For the variable studied the low values clearly result in fraud

Visualize and remove extreme outliers for high correlated variables

Steps:

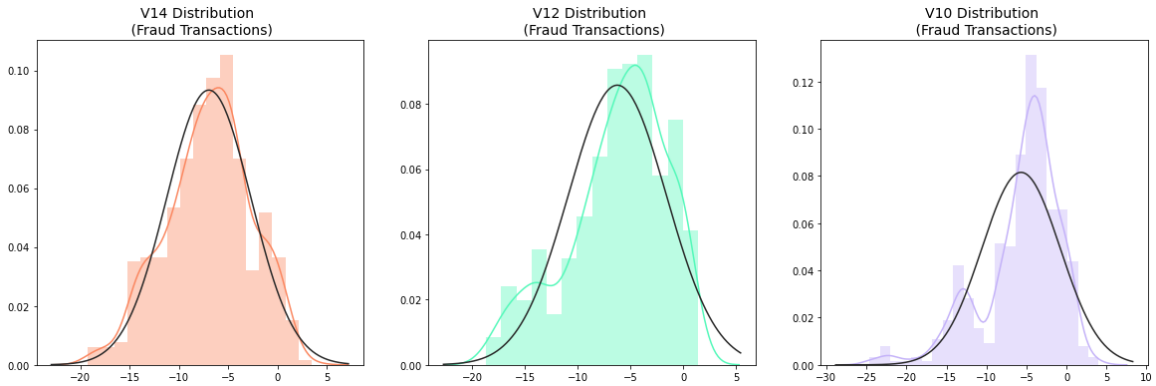
- **Visualize Distributions:** We first start by visualizing the distribution of the feature we are going to use to eliminate some of the outliers.
- **Determining the threshold:** After we decide which number we will use to multiply with the iqr (the lower more outliers removed), we will proceed in determining the upper and lower thresholds by substrating q25 - threshold (lower extreme threshold) and adding q75 + threshold (upper extreme threshold).
- **Conditional Dropping:** Lastly, we create a conditional dropping stating that if the "threshold" is exceeded in both extremes, the instances will be removed.
- **Boxplot Representation:** Visualize through the boxplot that the number of "extreme outliers" have been reduced to a considerable amount.

Visualize the Top Negative correlated Variables with Y

V10, V12 , V14

In [13]:

```
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()
```



In [14]:

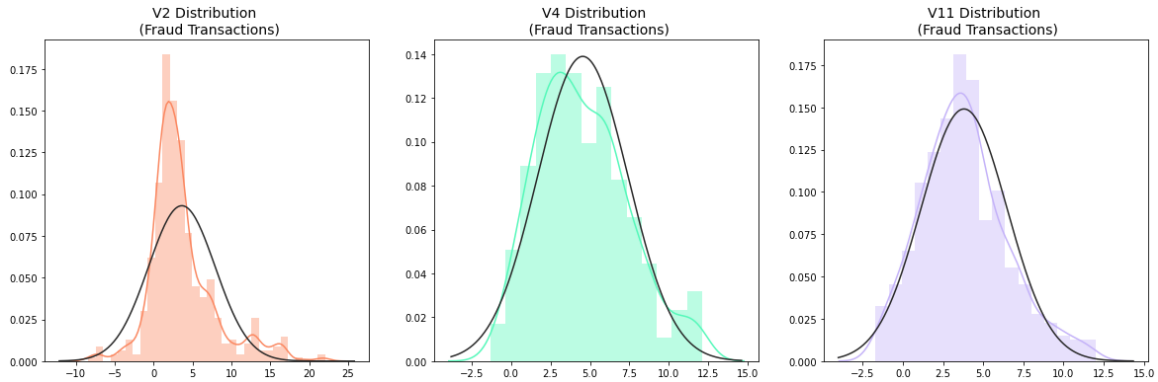
```
#### Observations :
```

Visualize the Top positive correlated variables with Y

V2,V4,V11

In [15]:

```
from scipy.stats import norm

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

v14_fraud_dist = new_df['V2'].loc[new_df['Class'] == 1].values
sns.distplot(v14_fraud_dist,ax=ax1, fit=norm, color='#FB8861')
ax1.set_title('V2 Distribution \n (Fraud Transactions)', fontsize=14)

v12_fraud_dist = new_df['V4'].loc[new_df['Class'] == 1].values
sns.distplot(v12_fraud_dist,ax=ax2, fit=norm, color='#56F9BB')
ax2.set_title('V4 Distribution \n (Fraud Transactions)', fontsize=14)


v10_fraud_dist = new_df['V11'].loc[new_df['Class'] == 1].values
sns.distplot(v10_fraud_dist,ax=ax3, fit=norm, color='#C5B3F9')
ax3.set_title('V11 Distribution \n (Fraud Transactions)', fontsize=14)

plt.show()
```



Remove Extreme Outliers From Negative correlated variables

Remove outliers from V14

In [16]:

```
 # -----> V14 Removing Outliers (Highest Negative Correlated with Labels)
# Isolate the fraud rows
v14_fraud = new_df['V14'].loc[new_df['Class'] == 1].values

# Get the lower quartile and upper quartile
q25, q75 = np.percentile(v14_fraud, 25), np.percentile(v14_fraud, 75)
print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))

# Get the interquartile range
v14_iqr = q75 - q25
print('iqr: {}'.format(v14_iqr))

# define cut offs
v14_cut_off = v14_iqr * 1.5

# define lower and upper cut offs
v14_lower, v14_upper = q25 - v14_cut_off, q75 + v14_cut_off
print('Cut Off: {}'.format(v14_cut_off))
print('V14 Lower: {}'.format(v14_lower))
print('V14 Upper: {}'.format(v14_upper))

# Get the outliers for Fraud Rows
outliers = [x for x in v14_fraud if x < v14_lower or x > v14_upper]
print('Feature V14 Outliers for Fraud Cases: {}'.format(len(outliers)))
print('V14 outliers:{}'.format(outliers))


# Drop outliers from Fraud rows
print("row count before outlier drop : {}".format(new_df.shape[0]))
new_df = new_df.drop(new_df[((new_df['V14'] > v14_upper) | (new_df['V14'] < v14_
lower))&(new_df['Class'] ==1 )].index)
print("row count after outlier drop : {}".format(new_df.shape[0]))
print('----' * 44)
```

```
Quartile 25: -9.69272296475 | Quartile 75: -4.2828208495
iqr: 5.40990211525
Cut Off: 8.114853172875002
V14 Lower: -17.807576137625002
V14 Upper: 3.8320323233750013
Feature V14 Outliers for Fraud Cases: 4
V14 outliers:[-18.49377336, -18.04999769, -18.82208674, -19.2143254
9]
row count before outlier drop : 984
row count after outlier drop : 980
------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------
```

Remove outliers from V12

In [17]:

```
# -----> V12 Removing Outliers (Highest Negative Correlated with Labels)
# Isolate the fraud rows
v12_fraud = new_df['V12'].loc[new_df['Class'] == 1].values

# Get the lower quartile and upper quartile
q25, q75 = np.percentile(v12_fraud, 25), np.percentile(v12_fraud, 75)
print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))

# Get the interquartile range
v12_iqr = q75 - q25
print('iqr: {}'.format(v12_iqr))

# define cut offs
v12_cut_off = v12_iqr * 1.5

# define lower and upper cut offs
v12_lower, v12_upper = q25 - v12_cut_off, q75 + v12_cut_off
print('Cut Off: {}'.format(v12_cut_off))
print('V12 Lower: {}'.format(v12_lower))
print('V12 Upper: {}'.format(v12_upper))

# Get the outliers for Fraud Rows
outliers = [x for x in v12_fraud if x < v12_lower or x > v12_upper]
print('Feature V12 Outliers for Fraud Cases: {}'.format(len(outliers)))
print('V12 outliers:{}'.format(outliers))


# Drop outliers from Fraud rows
print("row count before outlier drop : {}".format(new_df.shape[0]))
new_df = new_df.drop(new_df[((new_df['V12'] > v12_upper) | (new_df['V12'] < v12_
lower))&(new_df['Class'] ==1 )].index)
print("row count after outlier drop : {}".format(new_df.shape[0]))
print('----' * 44)
```

```
Quartile 25: -8.6730332045 | Quartile 75: -2.8930305682500004
iqr: 5.780002636249999
Cut Off: 8.670003954374998
V12 Lower: -17.343037158875
V12 Upper: 5.776973386124998
Feature V12 Outliers for Fraud Cases: 4
V12 outliers:[-18.43113103, -18.55369701, -18.68371463, -18.0475965
7]
row count before outlier drop : 980
row count after outlier drop : 976
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------
```

In [18]:

```
#### Remove Outliers from V10

# -----> V10 Removing Outliers (Highest Negative Correlated with Labels)
# Isolate the fraud rows
v10_fraud = new_df['V10'].loc[new_df['Class'] == 1].values

# Get the lower quartile and upper quartile
q25, q75 = np.percentile(v10_fraud, 25), np.percentile(v10_fraud, 75)
print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))

# Get the interquartile range
v10_iqr = q75 - q25
print('iqr: {}'.format(v10_iqr))

# define cut offs
v10_cut_off = v10_iqr * 1.5

# define lower and upper cut offs
v10_lower, v10_upper = q25 - v10_cut_off, q75 + v10_cut_off
print('Cut Off: {}'.format(v10_cut_off))
print('V10 Lower: {}'.format(v10_lower))
print('V10 Upper: {}'.format(v10_upper))

# Get the outliers for Fraud Rows
outliers = [x for x in v10_fraud if x < v10_lower or x > v10_upper]
print('Feature V10 Outliers for Fraud Cases: {}'.format(len(outliers)))
print('V10 outliers:{}'.format(outliers))


# Drop outliers from Fraud rows
print("row count before outlier drop : {}".format(new_df.shape[0]))
new_df = new_df.drop(new_df[((new_df['V10'] > v10_upper) | (new_df['V10'] < v10_
lower))&(new_df['Class'] ==1 )].index)
print("row count after outlier drop : {}".format(new_df.shape[0]))
print('----' * 44)
```

```
Quartile 25: -7.466658536000001 | Quartile 75: -2.51186113825
iqr: 4.954797397750001
Cut Off: 7.432196096625002
V10 Lower: -14.898854632625003
V10 Upper: 4.920334958375001
Feature V10 Outliers for Fraud Cases: 27
V10 outliers:[-22.18708856, -15.23996196, -16.60119697, -15.1241628
1, -19.83614885, -24.58826244, -15.34609885, -14.92465477, -16.64962
816, -15.56379134, -16.30353766, -22.18708856, -22.18708856, -15.231
83337, -16.74604411, -20.94919155, -24.40318497, -16.25561175, -23.2
2825484, -18.91324333, -15.23996196, -22.18708856, -17.14151364, -1
5.56379134, -14.92465477, -15.12375218, -18.27116817]
row count before outlier drop : 976
row count after outlier drop : 949
------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------
```

Removing Extreme Outliers from Highly Positive correlated variables

Remove Outliers from V2

In [19]:

```
#### Remove Outliers from V2

# -----> V2 Removing Outliers (Highest Negative Correlated with Labels)
# Isolate the fraud rows
v2_fraud = new_df['V2'].loc[new_df['Class'] == 1].values

# Get the lower quartile and upper quartile
q25, q75 = np.percentile(v2_fraud, 25), np.percentile(v2_fraud, 75)
print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))

# Get the interquartile range
v2_iqr = q75 - q25
print('iqr: {}'.format(v2_iqr))

# define cut offs
v2_cut_off = v2_iqr * 1.5

# define lower and upper cut offs
v2_lower, v2_upper = q25 - v2_cut_off, q75 + v2_cut_off
print('Cut Off: {}'.format(v2_cut_off))
print('V2 Lower: {}'.format(v2_lower))
print('V2 Upper: {}'.format(v2_upper))

# Get the outliers for Fraud Rows
outliers = [x for x in v2_fraud if x < v2_lower or x > v2_upper]
print('Feature V2 Outliers for Fraud Cases: {}'.format(len(outliers)))
print('V2 outliers:{}'.format(outliers))


# Drop outliers from Fraud rows
print("row count before outlier drop : {}".format(new_df.shape[0]))
new_df = new_df.drop(new_df[((new_df['V2'] > v2_upper) | (new_df['V2'] < v2_lowe
r))&(new_df['Class'] ==1 )].index)
print("row count after outlier drop : {}".format(new_df.shape[0]))
print('----' * 44)
```

```
Quartile 25: 1.133138588 | Quartile 75: 4.141986232
iqr: 3.008847644
Cut Off: 4.513271466
V2 Lower: -3.380132878
V2 Upper: 8.655257698
Feature V2 Outliers for Fraud Cases: 46
V2 outliers:[12.78597064, 13.76594216, 14.60199804, 10.5417508, -3.9
359189239999997, 16.15570143, 10.39391714, 12.65219683, 12.37398914,
12.78597064, 15.59819266, 12.78597064, 12.78597064, 9.06761342700000
1, -4.81446074, 12.78597064, -7.159041717000001, -6.976420008, 12.78
597064, 11.81792199, 11.58638052, 12.09589323, 14.32325381, -3.42046
79839999996, 12.93050512, -7.1969796310000005, -8.402153678, 14.0445
6678, 8.775997152999999, 10.11481572, -7.449015159, 10.81966537, -5.
198360199, 13.20890428, 16.43452455, 16.71338924, 8.713250171, 9.669
900173, -3.4881301810000003, 13.48738579, 15.87692299, 10.55860019,
9.223691949, 15.36580438, -3.9523200860000003, -3.930731396]
row count before outlier drop : 949
row count after outlier drop : 903
--------------------------------------------------------------------
--------------------------------------------------------------------
----------------------------------------
```

Remove Outliers for V11

In [20]:

```
# -----> V11 Removing Outliers (Highest Negative Correlated with Labels)
# Isolate the fraud rows
v11_fraud = new_df['V11'].loc[new_df['Class'] == 1].values

# Get the lower quartile and upper quartile
q25, q75 = np.percentile(v11_fraud, 25), np.percentile(v11_fraud, 75)
print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))

# Get the interquartile range
v11_iqr = q75 - q25
print('iqr: {}'.format(v11_iqr))

# define cut offs
v11_cut_off = v11_iqr * 1.5

# define lower and upper cut offs
v11_lower, v11_upper = q25 - v11_cut_off, q75 + v11_cut_off
print('Cut Off: {}'.format(v11_cut_off))
print('V11 Lower: {}'.format(v11_lower))
print('V11 Upper: {}'.format(v11_upper))

# Get the outliers for Fraud Rows
outliers = [x for x in v11_fraud if x < v11_lower or x > v11_upper]
print('Feature V11 Outliers for Fraud Cases: {}'.format(len(outliers)))
print('V11 outliers:{}'.format(outliers))


# Drop outliers from Fraud rows
print("row count before outlier drop : {}".format(new_df.shape[0]))
new_df = new_df.drop(new_df[((new_df['V11'] > v11_upper) | (new_df['V11'] < v11_
lower))&(new_df['Class'] ==1 )].index)
print("row count after outlier drop : {}".format(new_df.shape[0]))
print('----' * 44)
```

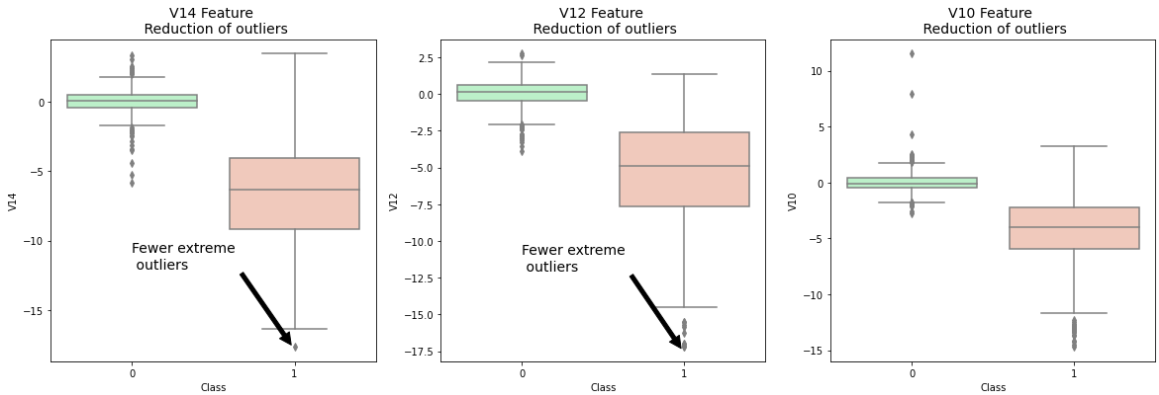
```
Quartile 25: 1.8450858435 | Quartile 75: 4.775434035
iqr: 2.9303481915000003
Cut Off: 4.39552228725
V11 Lower: -2.5504364437500007
V11 Upper: 9.17095632225
Feature V11 Outliers for Fraud Cases: 11
V11 outliers:[9.369079057999999, 11.27792073, 10.27776886, 10.187587
32, 9.939819742000001, 10.54526295, 11.1524906, 10.06378975, 9.32879
9257, 10.85301165, 10.44684681]
row count before outlier drop : 903
row count after outlier drop : 892
------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------
```

Check outlier removal with Box Plots

Box plots for highly correlated -ve features : V10, V12 , V14

In [21]:

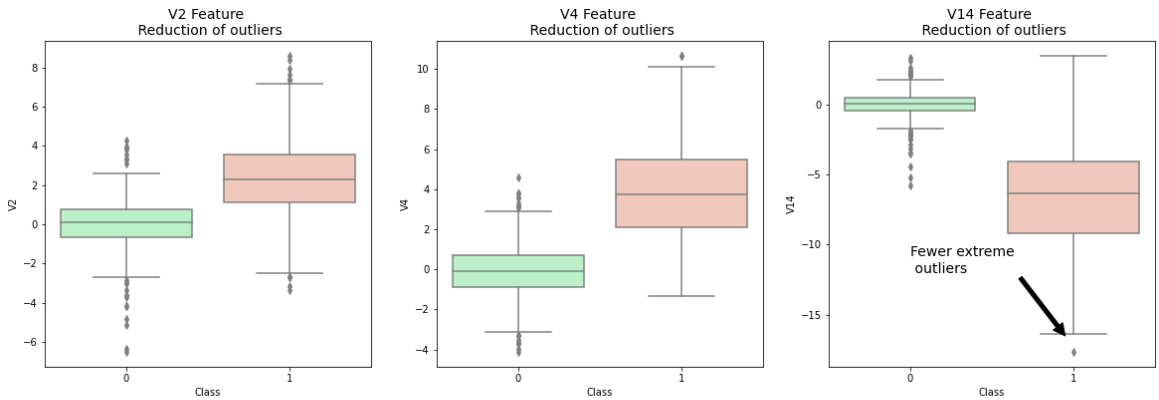
```
f,(ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))

colors = ['#B3F9C5', '#f9c5b3']
# Boxplots with outliers removed
# Feature V14
sns.boxplot(x="Class", y="V14", data=new_df,ax=ax1, palette=colors)
ax1.set_title("V14 Feature \n Reduction of outliers", fontsize=14)
ax1.annotate('Fewer extreme \n outliers', xy=(0.98, -17.5), xytext=(0, -12),
            arrowprops=dict(facecolor='black'),
            fontsize=14)

# Feature 12
sns.boxplot(x="Class", y="V12", data=new_df, ax=ax2, palette=colors)
ax2.set_title("V12 Feature \n Reduction of outliers", fontsize=14)
ax2.annotate('Fewer extreme \n outliers', xy=(0.98, -17.3), xytext=(0, -12),
            arrowprops=dict(facecolor='black'),
            fontsize=14)

# Feature V10
sns.boxplot(x="Class", y="V10", data=new_df, ax=ax3, palette=colors)
ax3.set_title("V10 Feature \n Reduction of outliers", fontsize=14)
ax3.annotate('Fewer extreme \n outliers', xy=(0.95, -16.5), xytext=(0, -12),
            arrowprops=dict(facecolor='black'),
            fontsize=14)


plt.show()
```



Box plots for highly correlated +ve features : V2, V4, V11

In [22]:

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

colors = ['#B3F9C5', '#f9c5b3']
# Boxplots with outliers removed
# Feature V2
sns.boxplot(x="Class", y="V2", data=new_df,ax=ax1, palette=colors)
ax1.set_title("V2 Feature \n Reduction of outliers", fontsize=14)
ax1.annotate('Fewer extreme \n outliers', xy=(0.98, -17.5), xytext=(0, -12),
            arrowprops=dict(facecolor='black'),
            fontsize=14)

# Feature 12
sns.boxplot(x="Class", y="V4", data=new_df, ax=ax2, palette=colors)
ax2.set_title("V4 Feature \n Reduction of outliers", fontsize=14)
ax2.annotate('Fewer extreme \n outliers', xy=(0.98, -17.3), xytext=(0, -12),
            arrowprops=dict(facecolor='black'),
            fontsize=14)

# Feature V10
sns.boxplot(x="Class", y="V14", data=new_df, ax=ax3, palette=colors)
ax3.set_title("V14 Feature \n Reduction of outliers", fontsize=14)
ax3.annotate('Fewer extreme \n outliers', xy=(0.95, -16.5), xytext=(0, -12),
            arrowprops=dict(facecolor='black'),
            fontsize=14)


plt.show()
```



Dimensionality Reduction and Clustering Visualization using t-SNE

Summary:

- t-SNE algorithm can pretty accurately cluster the cases that were fraud and non-fraud in our dataset.
- Although the subsample is pretty small, the t-SNE algorithm is able to detect clusters pretty accurately in every scenario (I shuffle the dataset before running t-SNE)
- This gives us an indication that further predictive models will perform pretty well in separating fraud cases from non-fraud cases.

In [23]:

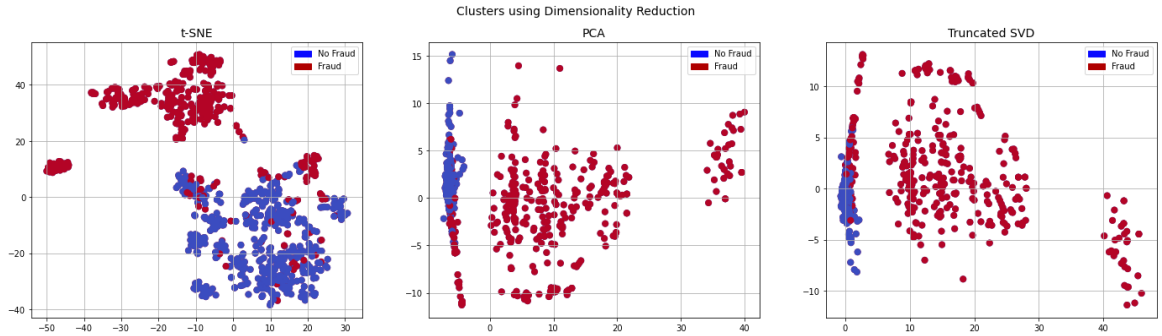
```
# New_df is from the random undersample data (fewer instances)
X = new_df.drop('Class', axis=1)
y = new_df['Class']


# T-SNE Implementation
t0 = time.time()
X_reduced_tsne = TSNE(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()
print("T-SNE took {:.2} s".format(t1 - t0))

# PCA Implementation
t0 = time.time()
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()
print("PCA took {:.2} s".format(t1 - t0))

# TruncatedSVD
t0 = time.time()
X_reduced_svd = TruncatedSVD(n_components=2, algorithm='randomized', random_stat
e=42).fit_transform(X.values)
t1 = time.time()
print("Truncated SVD took {:.2} s".format(t1 - t0))
```

```
T-SNE took 2.2 s
PCA took 0.018 s
Truncated SVD took 0.0034 s
```

Plot Scatter from the various reduced techniques

In [24]:

```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24,6))
# labels = ['No Fraud', 'Fraud']
f.suptitle('Clusters using Dimensionality Reduction', fontsize=14)


blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')


# t-SNE scatter plot
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 0), cmap='coolwar
m', label='No Fraud', linewidths=2)
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 1), cmap='coolwar
m', label='Fraud', linewidths=2)
ax1.set_title('t-SNE', fontsize=14)

ax1.grid(True)

ax1.legend(handles=[blue_patch, red_patch])


# PCA scatter plot
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 0), cmap='coolwarm',
label='No Fraud', linewidths=2)
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 1), cmap='coolwarm',
label='Fraud', linewidths=2)
ax2.set_title('PCA', fontsize=14)

ax2.grid(True)

ax2.legend(handles=[blue_patch, red_patch])

# TruncatedSVD scatter plot
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 0), cmap='coolwarm',
label='No Fraud', linewidths=2)
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 1), cmap='coolwarm',
label='Fraud', linewidths=2)
ax3.set_title('Truncated SVD', fontsize=14)

ax3.grid(True)

ax3.legend(handles=[blue_patch, red_patch])

plt.show()
```



Observations : We find that t-SNE offers a good separation - Thus the classes are separable

Classification with - Under Sampling

In [25]:

```
# Preparing data for modelling
#----------------------------

# Undersampling before cross validating (prone to overfit)
X = new_df.drop('Class', axis=1)
y = new_df['Class']

# Our data is already scaled we should split our training and test sets
from sklearn.model_selection import train_test_split

# This is explicitly used for undersampling.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state=42)
```

In [26]:

```
X_train1 = X_train.values
X_train1.shape
```

Out[26]:

```
(713, 30)
```

In [27]:

```
# Let's implement simple classifiers

classifiers = {
    "LogisticRegression": LogisticRegression(),
    "KNN": KNeighborsClassifier(),
    "SupportVectorClassifier": SVC(),
    "DecisionTreeClassifier": DecisionTreeClassifier()
}
```

Cross Validation employed

In [28]:

```
from sklearn.model_selection import cross_val_score


for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    print("Classifiers: ", classifier.__class__.__name__, "Has a training score
 of", round(training_score.mean(), 2) * 100, "% accuracy score")
```

```
Classifiers:  LogisticRegression Has a training score of 93.0 % accu
racy score
Classifiers:  KNeighborsClassifier Has a training score of 93.0 % ac
curacy score
Classifiers:  SVC Has a training score of 93.0 % accuracy score
Classifiers:  DecisionTreeClassifier Has a training score of 89.0 %
accuracy score
```

Observation

LR has the best training accuracy with CV

Grid Search Employed - Hyperparameter tuned individually for each classifiers

In [29]:

```
# Use GridSearchCV to find the best parameters.
from sklearn.model_selection import GridSearchCV


# Logistic Regression
#-----------------------------------
# penalty is regularization L1 or L2
# C parameter inverse of Reg strength
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1
000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train, y_train)
# We automatically get the logistic regression with the best parameters.
log_reg = grid_log_reg.best_estimator_
#-------------------------------------

# K nearest neighbours
#-----------------------------------
# Number of nearest neighbours
# Algo to find distance
# auto
# ball_tree
# kd_tree
# brute
knears_params = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['auto', 'ball_
tree', 'kd_tree', 'brute']}
grid_knears = GridSearchCV(KNeighborsClassifier(), knears_params)
grid_knears.fit(X_train, y_train)
# KNears best estimator
knears_neighbors = grid_knears.best_estimator_
#-----------------------------------------------

# Support Vector Classifier
#---------------------------------------
# Small value of C will cause the optimizer to look for alarger margin hence mor
e misclassification
svc_params = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly', 'sigmoid', 'lin
ear']}
grid_svc = GridSearchCV(SVC(), svc_params)
grid_svc.fit(X_train, y_train)
# SVC best estimator
svc = grid_svc.best_estimator_
#-------------------------------------


# DecisionTree Classifier
#---------------------------------------
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1
)),
              "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train, y_train)
# tree best estimator
tree_clf = grid_tree.best_estimator_
#-------------------------------------
```

Get the Cross Validated Score for each of the best models chosen

In [30]:

```
# Overfitting Case

log_reg_score = cross_val_score(log_reg, X_train, y_train, cv=5)
print('Logistic Regression Cross Validation Score: ', round(log_reg_score.mean()
* 100, 2).astype(str) + '%')


knears_score = cross_val_score(knears_neighbors, X_train, y_train, cv=5)
print('Knears Neighbors Cross Validation Score', round(knears_score.mean() * 100
, 2).astype(str) + '%')

svc_score = cross_val_score(svc, X_train, y_train, cv=5)
print('Support Vector Classifier Cross Validation Score', round(svc_score.mean()
* 100, 2).astype(str) + '%')

tree_score = cross_val_score(tree_clf, X_train, y_train, cv=5)
print('DecisionTree Classifier Cross Validation Score', round(tree_score.mean()
* 100, 2).astype(str) + '%')
```

```
Logistic Regression Cross Validation Score:  93.41%
Knears Neighbors Cross Validation Score 93.69%
Support Vector Classifier Cross Validation Score 93.27%
DecisionTree Classifier Cross Validation Score 92.56%
```

Get the ROC Score for All the models

In [31]:

```
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
# Capture the predictions from all the models

log_reg_pred = cross_val_predict(log_reg, X_train, y_train, cv=5,
                                 method="decision_function")

knears_pred = cross_val_predict(knears_neighbors, X_train, y_train, cv=5)

svc_pred = cross_val_predict(svc, X_train, y_train, cv=5,
                             method="decision_function")

tree_pred = cross_val_predict(tree_clf, X_train, y_train, cv=5)
```

In [32]:

```
# Get the ROC Score for all the models

from sklearn.metrics import roc_auc_score

print('Logistic Regression: ', roc_auc_score(y_train, log_reg_pred))
print('KNears Neighbors: ', roc_auc_score(y_train, knears_pred))
print('Support Vector Classifier: ', roc_auc_score(y_train, svc_pred))
print('Decision Tree Classifier: ', roc_auc_score(y_train, tree_pred))
```

```
Logistic Regression:  0.9801644202275117
KNears Neighbors:  0.9331126724659847
Support Vector Classifier:  0.9799095051460982
Decision Tree Classifier:  0.9220756460504095
```

Observation Logistics Regression shows the best ROC Score

Plot ROC Curve

In [33]:

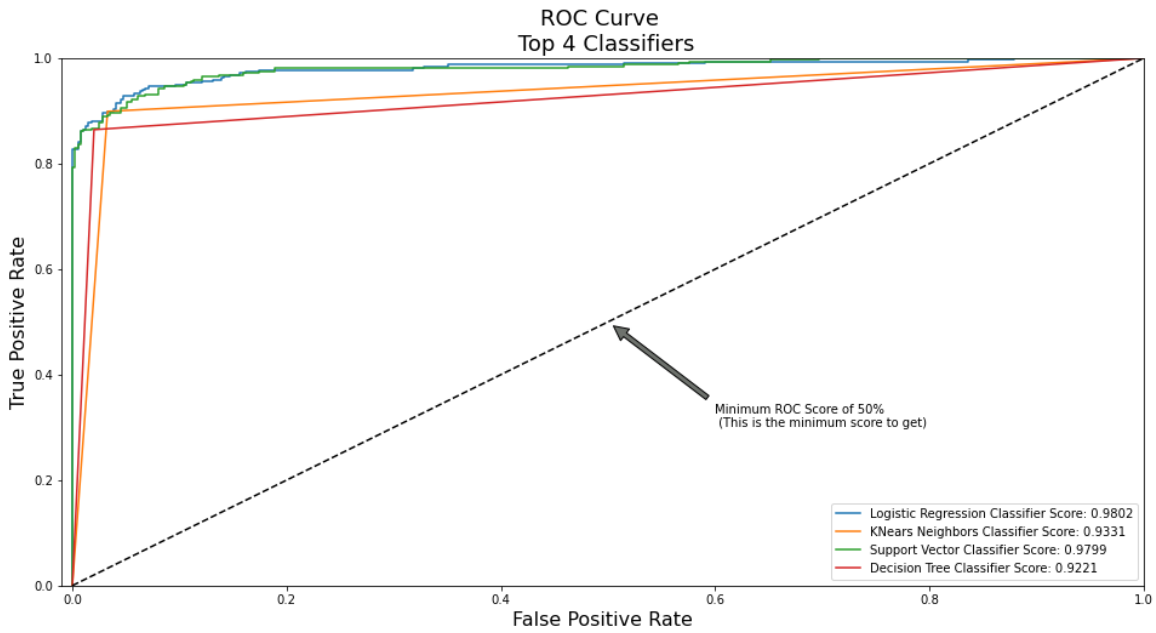
```
# Get ROC curve parms for Logit
log_fpr, log_tpr, log_threshold = roc_curve(y_train, log_reg_pred)

# Get ROC curve parms for Logit
knear_fpr, knear_tpr, knear_threshold = roc_curve(y_train, knears_pred)

# Get ROC curve parms for Logit
svc_fpr, svc_tpr, svc_threshold = roc_curve(y_train, svc_pred)

# Get ROC curve parms for Logit
tree_fpr, tree_tpr, tree_threshold = roc_curve(y_train, tree_pred)
```

In [ ]:

```

```

In [34]:

```
# plot curve


def graph_roc_curve_multiple(log_fpr, log_tpr, knear_fpr, knear_tpr, svc_fpr, sv
c_tpr, tree_fpr, tree_tpr):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
    plt.plot(log_fpr, log_tpr, label='Logistic Regression Classifier Score: {:.4
f}'.format(roc_auc_score(y_train, log_reg_pred)))
    plt.plot(knear_fpr, knear_tpr, label='KNears Neighbors Classifier Score: {:.
4f}'.format(roc_auc_score(y_train, knears_pred)))
    plt.plot(svc_fpr, svc_tpr, label='Support Vector Classifier Score: {:.4f}'.f
ormat(roc_auc_score(y_train, svc_pred)))
    plt.plot(tree_fpr, tree_tpr, label='Decision Tree Classifier Score: {:.4f}'.
format(roc_auc_score(y_train, tree_pred)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to ge
t)', xy=(0.5, 0.5), xytext=(0.6, 0.3),
                 arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                 )
    plt.legend()

graph_roc_curve_multiple(log_fpr, log_tpr, knear_fpr, knear_tpr, svc_fpr, svc_tp
r, tree_fpr, tree_tpr)
plt.show()
```



In [59]:

```
labels = ['No Fraud', 'Fraud']
undersample_pred = log_reg.predict(original_Xtest)
print(classification_report(original_ytest, undersample_pred, target_names=label
s))
```

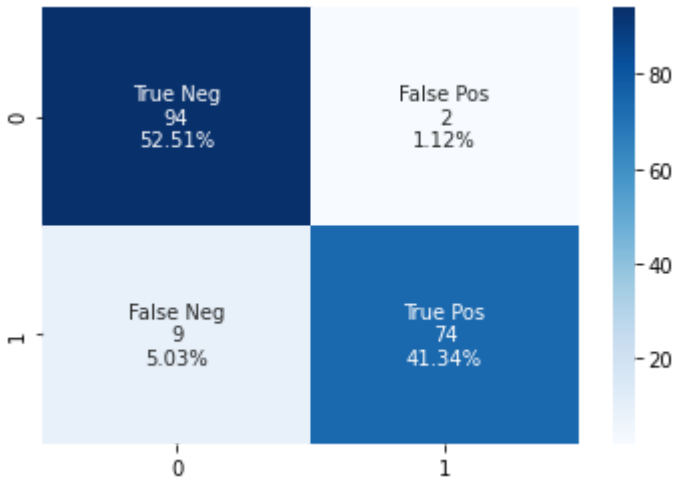
```
              precision    recall  f1-score   support

    No Fraud       1.00      0.99      1.00     56862
       Fraud       0.15      0.88      0.26        99

    accuracy                           0.99     56961
   macro avg       0.57      0.94      0.63     56961
weighted avg       1.00      0.99      0.99     56961
```

In [60]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns

# Logistic Regression fitted using SMOTE technique
y_pred_log_reg = log_reg.predict(X_test)

# Confusion Matrix
log_reg_cf = confusion_matrix(y_test, y_pred_log_reg)

group_names = ["True Neg","False Pos","False Neg","True Pos"]
group_counts = ["{0:0.0f}".format(value) for value in log_reg_cf.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     log_reg_cf.flatten()/np.sum(log_reg_cf)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)



sns.heatmap(log_reg_cf, annot=labels, fmt='', cmap='Blues')
```

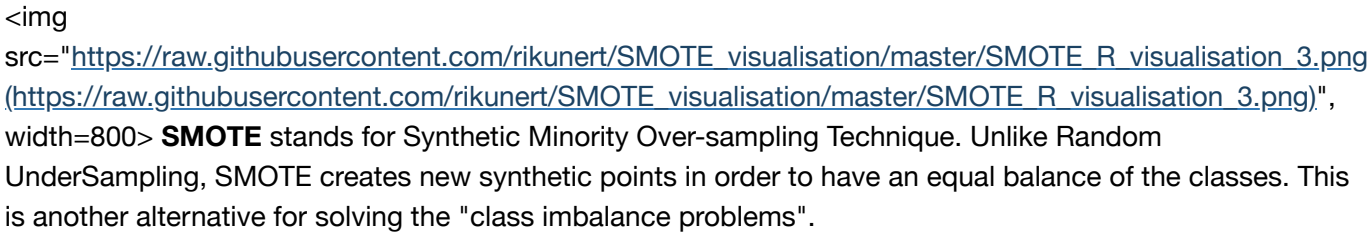
Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f87655a3670>
```



Classification with Over Sampling

SMOTE Technique (Over-Sampling):



SMOTE stands for Synthetic Minority Over-sampling Technique. Unlike Random UnderSampling, SMOTE creates new synthetic points in order to have an equal balance of the classes. This is another alternative for solving the "class imbalance problems".

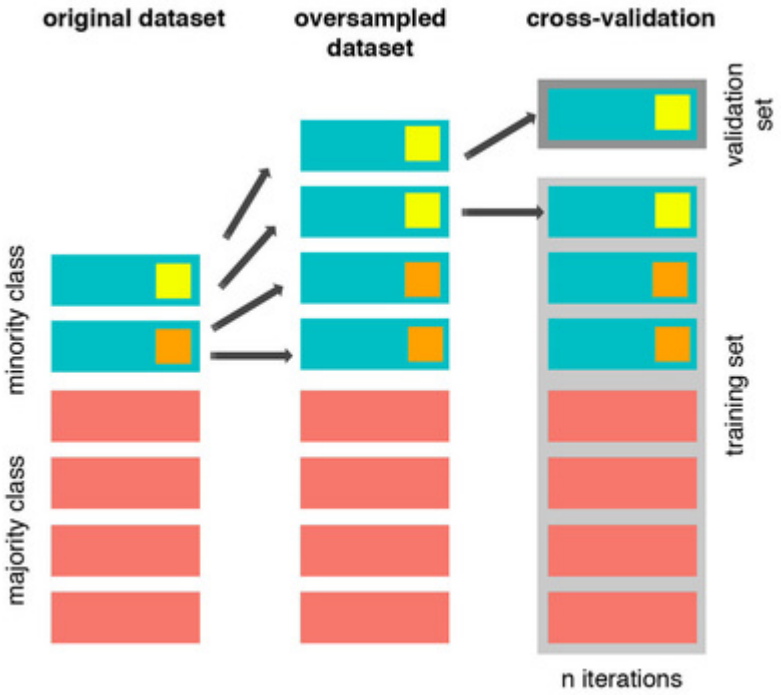
Understanding SMOTE:

- **Solving the Class Imbalance:** SMOTE creates synthetic points from the minority class in order to reach an equal balance between the minority and majority class.
- **Location of the synthetic points:** SMOTE picks the distance between the closest neighbors of the minority class, in between these distances it creates synthetic points.
- **Final Effect:** More information is retained since we didn't have to delete any rows unlike in random undersampling.
- **Accuracy || Time Tradeoff:** Although it is likely that SMOTE will be more accurate than random under-sampling, it will take more time to train since no rows are eliminated as previously stated.

Cautionary Note on Cross Validation for SMOTE

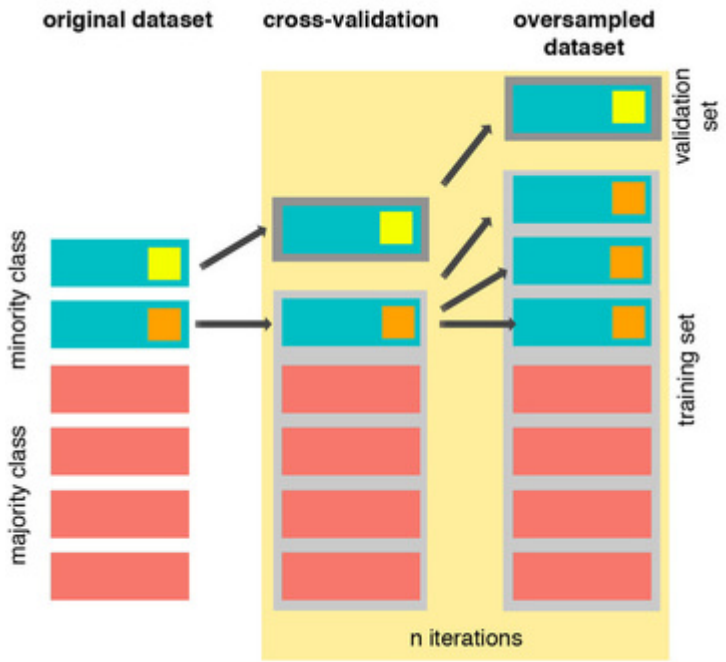
Overfitting during Cross Validation:

The Wrong Way:



As mentioned previously, if we get the minority class ("Fraud) in our case, and create the synthetic points before cross validating we have a certain influence on the "validation set" of the cross validation process. Remember how cross validation works, let's assume we are splitting the data into 5 batches, 4/5 of the dataset will be the training set while 1/5 will be the validation set. The test set should not be touched! For that reason, we have to do the creation of synthetic datapoints "during" cross-validation and not before, just like below:

The Right Way:



As you see above, SMOTE occurs "during" cross validation and not "prior" to the cross validation process. Synthetic data are created only for the training set without affecting the validation set.

References:

- DEALING WITH IMBALANCED DATA: UNDERSAMPLING, OVERSAMPLING AND PROPER CROSS-VALIDATION
- SMOTE explained for noobs

SMOTE Implementation for OverSampling with CV for Logistics Regression

MODEL TRAIN

In [35]:

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, RandomizedSearchCV

# List to append the score and then find the average
accuracy_lst = []
precision_lst = []
recall_lst = []
f1_lst = []
auc_lst = []

# Instantiate LR
log_reg_sm = LogisticRegression()

# LR parameters
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1
000]}

# Instantiate Random Search CV
rand_log_reg = RandomizedSearchCV(LogisticRegression(), log_reg_params, n_iter=4
)

# Implementing SMOTE Technique
# Cross Validating the right way

for train, test in sss.split(original_Xtrain, original_ytrain):
    pipeline = imbalanced_make_pipeline(SMOTE(sampling_strategy='minority'), ran
d_log_reg) # SMOTE happens during Cross Validation not before..
    model = pipeline.fit(original_Xtrain[train], original_ytrain[train])
    best_est = rand_log_reg.best_estimator_
    prediction = best_est.predict(original_Xtrain[test])

    accuracy_lst.append(pipeline.score(original_Xtrain[test], original_ytrain[te
st]))
    precision_lst.append(precision_score(original_ytrain[test], prediction))
    recall_lst.append(recall_score(original_ytrain[test], prediction))
    f1_lst.append(f1_score(original_ytrain[test], prediction))
    auc_lst.append(roc_auc_score(original_ytrain[test], prediction))

# Get the MEAN Values of the CV accuracies
#-------------------------------------------------------
print('---' * 45)
print('')
print("accuracy: {}".format(np.mean(accuracy_lst)))
print("precision: {}".format(np.mean(precision_lst)))
print("recall: {}".format(np.mean(recall_lst)))
print("f1: {}".format(np.mean(f1_lst)))
print('---' * 45)
```

```
---------------------------------------------------------------------------
---------------------------------------------------------------------------

accuracy: 0.9425706072110426
precision: 0.0622266391310668
recall: 0.9137617656604998
f1: 0.11454090998118364
---------------------------------------------------------------------------
---------------------------------------------------------------------------
```

MODEL TEST

Classification report on test data

In [36]:

```
labels = ['No Fraud', 'Fraud']
smote_prediction = best_est.predict(original_Xtest)
print(classification_report(original_ytest, smote_prediction, target_names=label
s))
```

```
              precision    recall  f1-score   support

    No Fraud       1.00      0.99      0.99     56862
       Fraud       0.10      0.86      0.18        99

    accuracy                           0.99     56961
   macro avg       0.55      0.92      0.58     56961
weighted avg       1.00      0.99      0.99     56961
```

Precision and Recall Studies

In [37]:

```
from sklearn.metrics import average_precision_score
y_score = best_est.decision_function(original_Xtest)
average_precision = average_precision_score(original_ytest, y_score)

print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
```

```
Average precision-recall score: 0.70
```

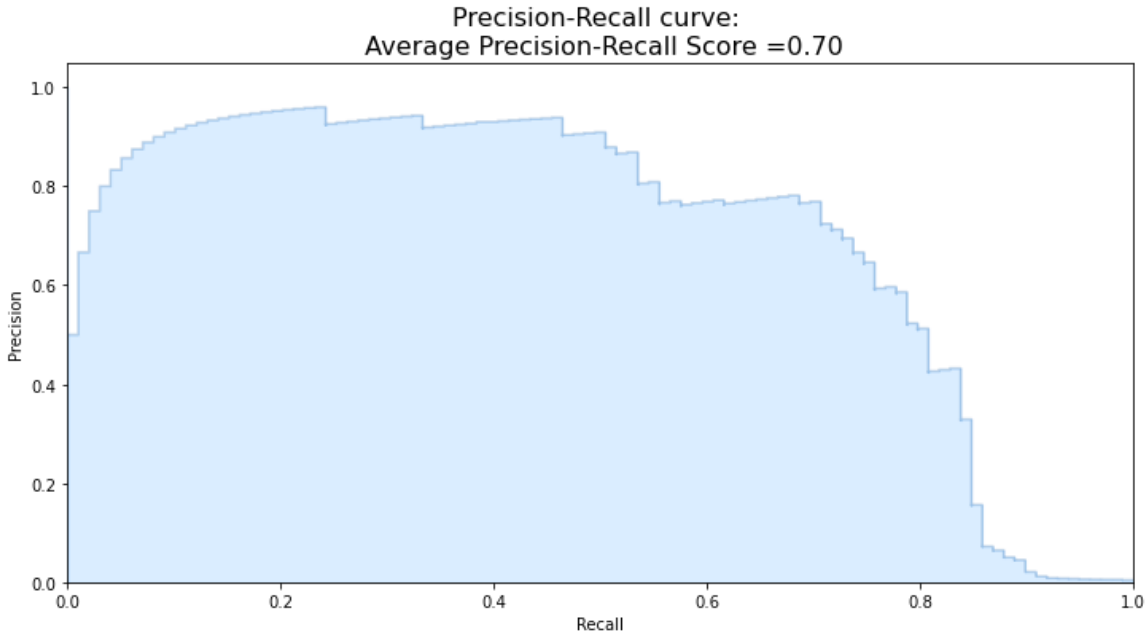
Plot Precision Recall Curve

In [38]:

```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12,6))

precision, recall, _ = precision_recall_curve(original_ytest, y_score)

plt.step(recall, precision, color='#004a93', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='#48a6ff')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve: \n Average Precision-Recall Score ={0:0.2f}'.
format(
          average_precision), fontsize=16)
```

Out[38]:

```
Text(0.5, 1.0, 'Precision-Recall curve: \n Average Precision-Recall
Score =0.70')
```



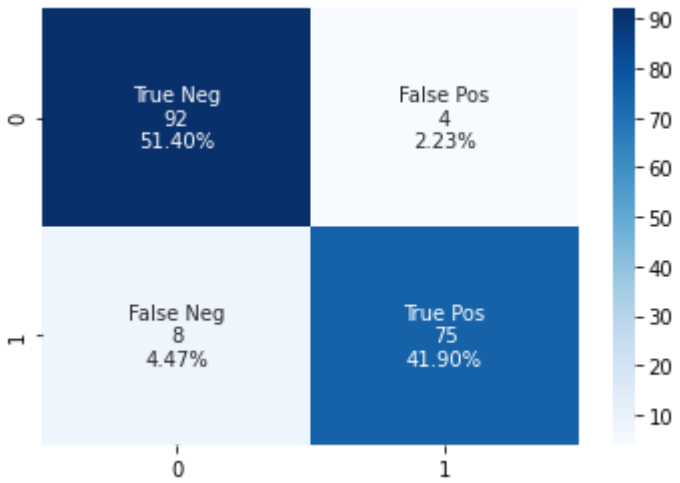
Confusion Matrix

In [57]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns

# Logistic Regression fitted using SMOTE technique
y_pred_log_reg = best_est.predict(X_test)

# Confusion Matrix
log_reg_cf = confusion_matrix(y_test, y_pred_log_reg)

group_names = ["True Neg","False Pos","False Neg","True Pos"]
group_counts = ["{0:0.0f}".format(value) for value in log_reg_cf.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     log_reg_cf.flatten()/np.sum(log_reg_cf)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)



sns.heatmap(log_reg_cf, annot=labels, fmt='', cmap='Blues')
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8764678460>
```



In [ ]:

CREDIT CARD FRAUD ANALYSIS

PART 3 : Anomaly Detection Methods

INTRODUCTION

Anomaly Detection

Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers. It has many applications in business, from intrusion detection (identifying strange patterns in network traffic that could signal a hack) to system health monitoring (spotting a malignant tumor in an MRI scan), and from fraud detection in credit card transactions to fault detection in operating environments.

What Are Anomalies? Anomalies can be broadly categorized as:

- Point anomalies: A single instance of data is anomalous if it's too far off from the rest. Business use case: Detecting credit card fraud based on "amount spent."
- Contextual anomalies: The abnormality is context specific. This type of anomaly is common in time-series data. Business use case: Spending $100 on food every day during the holiday season is normal, but may be odd otherwise.
- Collective anomalies: A set of data instances collectively helps in detecting anomalies. Business use case: Someone is trying to copy data form a remote machine to a local host unexpectedly, an anomaly that would be flagged as a potential cyber attack.

Anomaly Detection Techniques

Simple Statistical Methods

The simplest approach to identifying irregularities in data is to flag the data points that deviate from common statistical properties of a distribution, including mean, median, mode, and quantiles. Let's say the definition of an anomalous data point is one that deviates by a certain standard deviation from the mean. Traversing mean over time-series data isn't exactly trivial, as it's not static. You would need a rolling window to compute the average across the data points. Technically, this is called a rolling average or a moving average, and it's intended to smooth short-term fluctuations and highlight long-term ones. Mathematically, an n-period simple moving average can also be defined as a "low pass filter."

Challenges with Simple Statistical Methods The low pass filter allows you to identify anomalies in simple use cases, but there are certain situations where this technique won't work. Here are a few:

The data contains noise which might be similar to abnormal behavior, because the boundary between normal and abnormal behavior is often not precise.

The definition of abnormal or normal may frequently change, as malicious adversaries constantly adapt themselves. Therefore, the threshold based on moving average may not always apply.

The pattern is based on seasonality. This involves more sophisticated methods, such as decomposing the data into multiple trends in order to identify the change in seasonality.

Machine Learning based Approaches

- **Density-Based Anomaly Detection** Density-based anomaly detection is based on the k-nearest neighbors algorithm. Assumption: Normal data points occur around a dense neighborhood and abnormalities are far away. The nearest set of data points are evaluated using a score, which could be Eucledian distance or a similar measure dependent on the type of the data (categorical or numerical). They could be broadly classified into two algorithms:
  - **K-nearest neighbor:** k-NN is a simple, non-parametric lazy learning technique used to classify data based on similarities in distance metrics such as Eucledian, Manhattan, Minkowski, or Hamming distance.
  - **Relative density of data:** This is better known as local outlier factor (LOF). This concept is based on a distance metric called reachability distance.

- **Clustering-Based Anomaly Detection** Clustering is one of the most popular concepts in the domain of unsupervised learning. Assumption: Data points that are similar tend to belong to similar groups or clusters, as determined by their distance from local centroids. K-means is a widely used clustering algorithm. It creates 'k' similar clusters of data points. Data instances that fall outside of these groups could potentially be marked as anomalies.
- **Support Vector Machine-Based Anomaly Detection** A support vector machine is another effective technique for detecting anomalies. A SVM is typically associated with supervised learning, but there are extensions (OneClassCVM, for instance) that can be used to identify anomalies as an unsupervised problems (in which training data are not labeled). The algorithm learns a soft boundary in order to cluster the normal data instances using the training set, and then, using the testing instance, it tunes itself to identify the abnormalities that fall outside the learned region. Depending on the use case, the output of an anomaly detector could be numeric scalar values for filtering on domain-specific thresholds or textual labels (such as binary/multi labels).

Problem Statement

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect 100 % of the fraudulent transactions while minimizing the incorrect fraud classifications.

Approaches

- Isolation Forest Anomaly Detection Algorithm
- Density-Based Anomaly Detection (Local Outlier Factor)Algorithm
- Support Vector Machine Anomaly Detection Algorithm

Import Libraries

In [7]:

```
```

1. Isolation Forest

REFERENCES

[1] https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e (https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e)

[2] https://heartbeat.fritz.ai/isolation-forest-algorithm-for-anomaly-detection-2a4abd347a5 (https://heartbeat.fritz.ai/isolation-forest-algorithm-for-anomaly-detection-2a4abd347a5)

[3] https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf (https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf)

What is the Isolation Forest

Isolation Forest, like any tree ensemble method, is built on the basis of decision trees. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature. In principle, outliers are less frequent than regular observations and are different from them in terms of values (they lie further away from the regular observations in the feature space). That is why by using such random partitioning they should be identified closer to the root of the tree (shorter average path length, i.e., the number of edges an observation must pass in the tree going from the root to the terminal node), with fewer splits necessary.

Anomaly Score computation in Isolation forest

As with other outlier detection methods, an anomaly score is required for decision making. In the case of Isolation Forest, it is defined as:

$$s(x, n) = 2^{\frac{-E(h(x)}{c(n)}}$$

where

- $h(x)$ is the path length of the observation x
- $c(n)$ is average path length of unsuccessful search in a binary tree$x$
- n is the number of external nodes

also $c(n)$ is given as follows

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

where $H(i)$ is the harmonic number and it can be estimated by $ln(i) + 0.5772156649$ (Euler's constant)

- A score close to 1 indicates anomalies
- Score much smaller than 0.5 indicates normal observations
- If all scores are close to 0.5 then the entire sample does not seem to have clearly distinct anomalies

2. Local Outlier Factor

REFERENCES

[4] https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843 (https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843)

[5] https://towardsdatascience.com/local-outlier-factor-for-anomaly-detection-cc0c770d2ebe (https://towardsdatascience.com/local-outlier-factor-for-anomaly-detection-cc0c770d2ebe)

[6] https://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf (https://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf)

What is the LOF method?

When a point is considered as an outlier based on its local neighborhood, it is a local outlier. LOF will identify an outlier considering the density of the neighborhood. LOF performs well when the density of the data is not the same throughout the dataset. To understand LOF, we have to learn a few concepts sequentially:

- K-distance and K-neighbors
- Reachability distance (RD)
- Local reachability density (LRD)
- Local Outlier Factor (LOF)

- **K Distance and K Neighbours**

  K-distance is the distance between the point, and it's K^{th} nearest neighbor. K-neighbors denoted by $N_k(A)$ includes a set of points that lie in or on the circle of radius K-distance. K-neighbors can be more than or equal to the value of K.

- **Reachability distance (RD)**

  It is defined as the maximum of K-distance of Xj and the distance between Xi and Xj. The distance measure is problem-specific (Euclidean, Manhattan, etc.)

  In layman terms, if a point Xi lies within the K-neighbors of Xj, the reachability distance will be K-distance of Xj , else reachability distance will be the distance between Xi and Xj. Thus mathematicallywe have the following:

$$RD(X_i, X_j) = max(K - Distance(X_j), distance(X_i, X_j))$$

- **Local Reachability Density**

  LRD is inverse of the average reachability distance of A from its neighbors. Intuitively according to LRD formula, more the average reachability distance (i.e., neighbors are far from the point), less density of points are present around a particular point. This tells how far a point is from the nearest cluster of points. Low values of LRD implies that the closest cluster is far from the point.

$$LRD_K(A) = \frac{1}{\sum_{X_j \in N_k(A)} \frac{RD(A,X_j)}{||N_k(A)||}}$$

- **LOCAL OUTLIER FACTOR**

  RD of each point is used to compare with the average LRD of its K neighbors. LOF is the ratio of the average LRD of the K neighbors of A to the LRD of A.

  Intuitively, if the point is not an outlier (inlier), the ratio of average LRD of neighbors is approximately equal to the LRD of a point (because the density of a point and its neighbors are roughly equal). In that case, LOF is nearly equal to 1. On the other hand, if the point is an outlier, the LRD of a point is less than the average LRD of neighbors. Then LOF value will be high.

  Generally, if LOF> 1, it is considered as an outlier, but that is not always true. Let's say we know that we only have one outlier in the data, then we take the maximum LOF value among all the LOF values, and the point corresponding to the maximum LOF value will be considered as an outlier.

$$LOF_K(A) = \frac{\sum_{X_j \in N_k(A)} LRD_k(X_j)}{||N_k(A)||} \times \frac{1}{LRD_k(A)}$$

3. One Class SVM

REFERENCES

[7] https://towardsdatascience.com/support-vector-machine-svm-for-anomaly-detection-73a8d676c331 (https://towardsdatascience.com/support-vector-machine-svm-for-anomaly-detection-73a8d676c331)

[8]http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/ (http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/)

RECAP of Two Class SVM

Let us first take a look at the traditional two-class support vector machine. Consider a data set Ω={(x1,y1), (x2,y2),…,(xn,yn)}; points xi∈ℝ d in a (for instance two-dimensional) space where xi is the i-th input data point and yi∈{−1,1} is the i-th output pattern, indicating the class membership.

A very nice property of SVMs is that it can create a non-linear decision boundary by projecting the data through a non-linear function ϕ to a space with a higher dimension. This means that data points which can't be separated by a straight line in their original space I are "lifted" to a feature space F where there can be a "straight" hyperplane that separates the data points of one class from an other.

The hyperplane is represented with the equation wTx+b=0, with w∈F and b∈R. The hyperplane that is constructed determines the margin between the classes; all the data points for the class −1 are on one side, and all the data points for class 1 on the other. The distance from the closest point from each class to the hyperplane is equal; thus the constructed hyperplane searches for the maximal margin ("separating power") between the classes. To prevent the SVM classifier from over-fitting with noisy data (or to create a soft margin), slack variables ξi are introduced to allow some data points to lie within the margin, and the constant C>0 determines the trade-off between maximizing the margin and the number of training data points within that margin (and thus training errors). The objective function of the SVM classifier is the following minimization formulation:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^t w + C \sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(w\phi(x_i + b)) \geq 1 - \xi_i \text{ for all i= 1,...,n} \\ & \xi \geq 0 \text{ for all i= 1,...,n} \end{aligned}$$

One-Class SVM according to Schölkopf

REFERENCE [9] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.675.575&rep=rep1&type=pdf (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.675.575&rep=rep1&type=pdf)

The Support Vector Method For Novelty Detection by Schölkopf et al. basically separates all the data points from the origin (in feature space F) and maximizes the distance from this hyperplane to the origin. This results in a binary function which captures regions in the input space where the probability density of the data lives. Thus the function returns +1 in a "small" region (capturing the training data points) and −1 elsewhere.

The quadratic programming minimization function is slightly different from the original stated above, but the similarity is still clear:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^t w + \frac{1}{v\nu} \sum_{i=1}^{N} \xi_i - \rho \\ \text{s.t.} \quad & y_i(w\phi(x_i)) \geq \rho - \xi_i \text{ for all i= 1,...,n} \\ & \xi \geq 0 \text{ for all i= 1,...,n} \end{aligned}$$

In the previous formulation the parameter C decided the smoothness. In this formula it is the parameter ν that characterizes the solution;

it sets an upper bound on the fraction of outliers (training examples regarded out-of-class) and, it is a lower bound on the number of training examples used as Support Vector. Due to the importance of this parameter, this approach is often referred to as ν-SVM.

CODE IMPLEMENTATION

REFERENCE [10]. https://www.kaggle.com/naveengowda16/anomaly-detection-credit-card-fraud-analysis (https://www.kaggle.com/naveengowda16/anomaly-detection-credit-card-fraud-analysis)

Imports

In [ ]:

```
#Import the required libraries
import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

Data Reads

In [8]:

```
data = pd.read_csv('creditcard_data.csv')
data.head()
```

Out[8]:

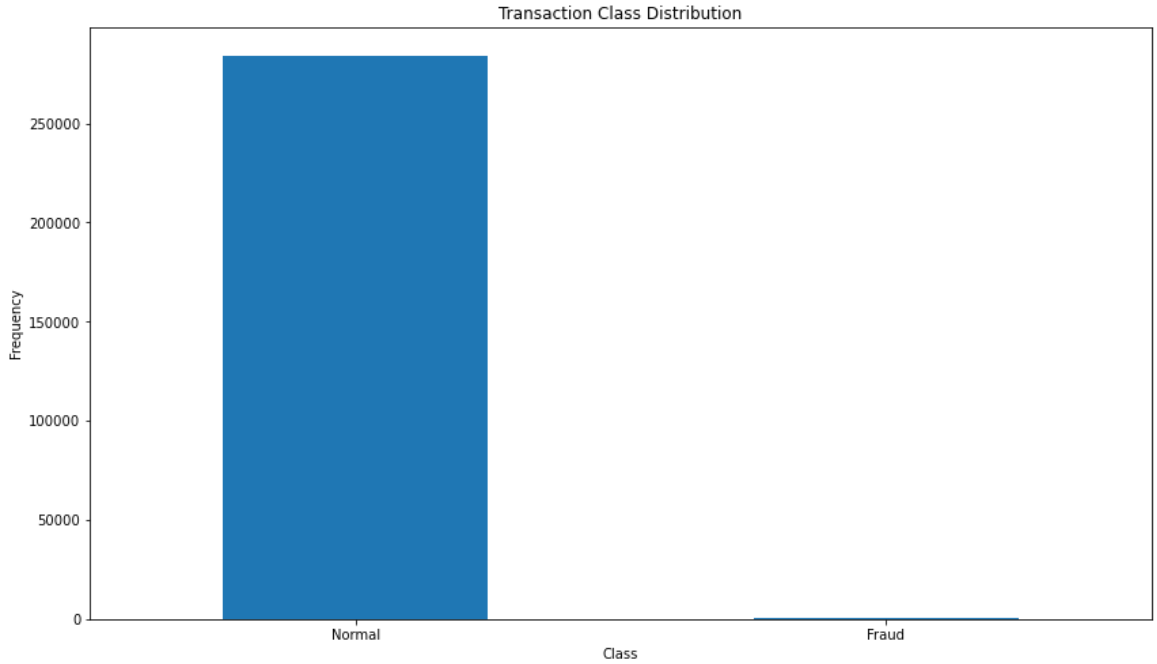
	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

Class Distribution

In [9]:

```
#Determine the number of fraud and valid transactions in the entire dataset

count_classes = pd.value_counts(data['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");
```



In [10]:

```
#Assigning the transaction class "0 = NORMAL  & 1 = FRAUD"
Normal = data[data['Class']==0]
Fraud = data[data['Class']==1]
```

In [12]:

```
### Print outlier fraction

Fraud = data[data['Class']==1]
Valid = data[data['Class']==0]
outlier_fraction = len(Fraud)/float(len(Valid))

print(outlier_fraction)
print("Fraud Cases : {}".format(len(Fraud)))
print("Valid Cases : {}".format(len(Valid)))
```

```
0.0017304810878113635
Fraud Cases : 492
Valid Cases : 284314
```

Subset data

We choose a 10% fraction of the orginal data set and build models

In [13]:

```
data1= data.sample(frac = 0.1,random_state=1)

Fraud1 = data1[data1['Class']==1]
Valid1 = data1[data1['Class']==0]
outlier_fraction1 = len(Fraud1)/float(len(Valid1))

print(outlier_fraction1)
print("Fraud Cases : {}".format(len(Fraud1)))
print("Valid Cases : {}".format(len(Valid1)))
```

```
0.0016529506928325245
Fraud Cases : 47
Valid Cases : 28434
```

MODELLING

Prepare X, Y Data

In [14]:

```
#Get all the columns from the dataframe

columns = data1.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]
# Store the variable we are predicting
target = "Class"
# Define a random state
state = np.random.RandomState(42)
X = data1[columns]
Y = data1[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(28481, 30)
(28481,)
```

Define Classifiers

In [18]:

```
#Define the outlier detection methods

classifiers = {
    "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(X),
                                       contamination=outlier_fraction,random_sta
te=state, verbose=0),
    "Local Outlier Factor":LocalOutlierFactor(n_neighbors=20, algorithm='auto',
                                              leaf_size=30, metric='minkowski',
                                              p=2, metric_params=None, contamina
tion=outlier_fraction),
    "Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.
05, max_iter=-1)


}
```

In [19]:

```
#Fit the model

n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    elif clf_name == "Support Vector Machine":
        clf.fit(X)
        y_pred = clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud tra
nsactions
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_errors = (y_pred != Y).sum()
    # Run Classification Metrics
    print("{}: {}".format(clf_name,n_errors))
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
    print("Classification Report :")
    print(classification_report(Y,y_pred))
```

```
Isolation Forest: 69
Accuracy Score :
0.9975773322565921
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28434
           1       0.28      0.30      0.29        47

    accuracy                           1.00     28481
   macro avg       0.64      0.65      0.64     28481
weighted avg       1.00      1.00      1.00     28481

Local Outlier Factor: 95
Accuracy Score :
0.9966644429619747
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28434
           1       0.02      0.02      0.02        47

    accuracy                           1.00     28481
   macro avg       0.51      0.51      0.51     28481
weighted avg       1.00      1.00      1.00     28481

Support Vector Machine: 8411
Accuracy Score :
0.7046803131912504
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.71      0.83     28434
           1       0.00      0.34      0.00        47

    accuracy                           0.70     28481
   macro avg       0.50      0.52      0.42     28481
weighted avg       1.00      0.70      0.83     28481
```

OBSERVATIONS

- Isolation Forest detected 69 errors versus Local Outlier Factor detecting 93 errors vs. SVM detecting 8411 errors
- Isolation Forest has a 99.75% more accurate than LOF of 99.67% and SVM of 70.46
- When comparing error precision & recall for 3 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 27 % versus LOF detection rate of just 2 % and SVM of 0
- So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.
- We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

In [ ]: