Dealing with imbalanced dataset

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[2]:

	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	0.36
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

5 rows × 31 columns

In [3]:
```
df.isnull().sum().max()
```

Out[3]: 0

In [4]:
```
df.columns
```

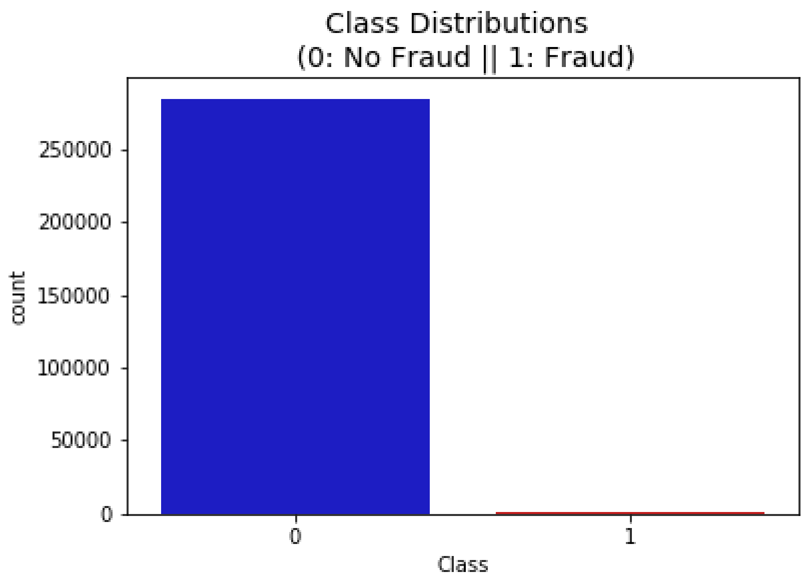
Out[4]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [5]:
```
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '%
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of

print('\nNo Frauds amount', df['Class'].value_counts()[0])
print('Frauds amount', df['Class'].value_counts()[1])
```

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

No Frauds amount 284315
Frauds amount 492
```

In [5]:
```
colors = ["#0101DF", "#DF0101"]

sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

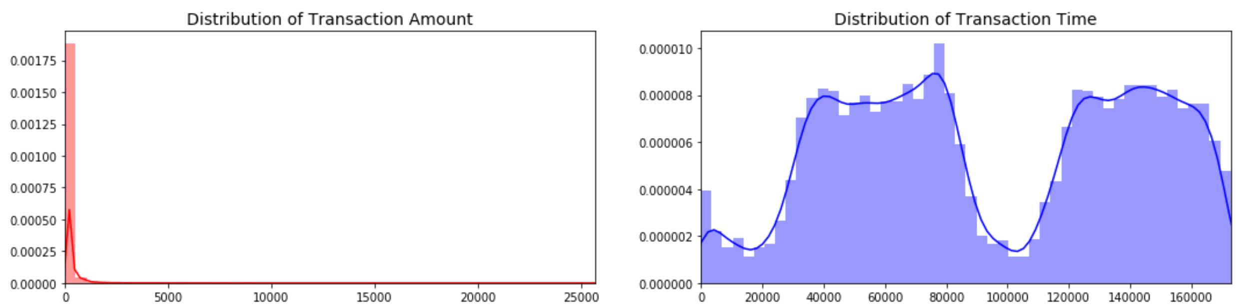
Out[5]:
```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```



So the dataset is too unbalanced to ignore this fact.

In [6]:
```
%%time
fig, ax = plt.subplots(1, 2, figsize=(18,4))

# .values returns a numpy representation of the dataframe
amount_val = df['Amount'].values
print(type(amount_val))
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()
```

```
<class 'numpy.ndarray'>
```




```
CPU times: user 1.89 s, sys: 148 ms, total: 2.04 s
Wall time: 1.13 s
```

Scaling the data

In [7]:

```
%%time
# Since most of our data has already been scaled
# we should 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(
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1

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

df.head()
```

```
CPU times: user 136 ms, sys: 60.6 ms, total: 196 ms
Wall time: 208 ms
```

Out[7]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 31 columns

Splitting the data

In this scenario our subsample will be a dataframw with a 50/50 ratio of fraud and non-fraud transactions.

Stratified K-Folds cross-validator provides train/test indices to split data in train/test sets.

This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

In [9]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import KFold, StratifiedKFold


X = df.drop('Class', axis=1)
y = df['Class']

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

i = 0
for train_index, test_index in sss.split(X, y):
    print('Nubmer of fold:', i)
    print("Train:", train_index, '\n', "Test:", test_index)
    print("Len(train_index):", len(train_index), '\n',
          "Len(test_index):", len(test_index), '\n\n')
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_inde
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_inde
    i+=1

# We already have X_train and y_train for undersample data thats why I am u
# 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 distribut
train_unique_label, train_counts_label = np.unique(original_ytrain, return_
test_unique_label, test_counts_label = np.unique(original_ytest, return_cou
print('-' * 100)

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

```
sss type <class 'sklearn.model_selection._split.StratifiedKFold'>
Nubmer of fold: 0
Train: [ 30473  30496  31002 ... 284804 284805 284806]
 Test: [     0      1      2 ... 57017 57018 57019]
Len(train_index): 227845
 Len(test_index): 56962


Nubmer of fold: 1
Train: [     0      1      2 ... 284804 284805 284806]
 Test: [ 30473  30496  31002 ... 113964 113965 113966]
Len(train_index): 227845
```

```
 Len(test_index): 56962


Nubmer of fold: 2
Train: [     0      1      2 ... 284804 284805 284806]
 Test: [ 81609  82400  83053 ... 170946 170947 170948]
Len(train_index): 227846
 Len(test_index): 56961


Nubmer of fold: 3
Train: [     0      1      2 ... 284804 284805 284806]
 Test: [150654 150660 150661 ... 227866 227867 227868]
Len(train_index): 227846
 Len(test_index): 56961


Nubmer of fold: 4
Train: [     0      1      2 ... 227866 227867 227868]
 Test: [212516 212644 213092 ... 284804 284805 284806]
Len(train_index): 227846
 Len(test_index): 56961


----------------------------------------------------------------------
-----------------------------
Label Distributions:

[0.99827076 0.00172924]
[0.99827952 0.00172048]
```

Manual undersampling

In [11]:
```
# 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]

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[11]:

	V1	V2	V3	V4	V5	V6	V7	V8	
197195	-2.488810	0.320152	-1.406537	-0.948898	0.807181	0.477675	-0.164258	1.504650	-0.60
191074	-1.836940	-1.646764	-3.381168	0.473354	0.074243	-0.446751	3.791907	-1.351045	0.09
74967	1.278550	-1.210571	-0.233822	-1.723942	-0.312205	1.240504	-0.983253	0.449612	-2.26
204064	0.232512	0.938944	-4.647780	3.079844	-1.902655	-1.041408	-1.020407	0.547069	-1.10
93788	1.080433	0.962831	-0.278065	2.743318	0.412364	-0.320778	0.041290	0.176170	-0.96

5 rows × 31 columns

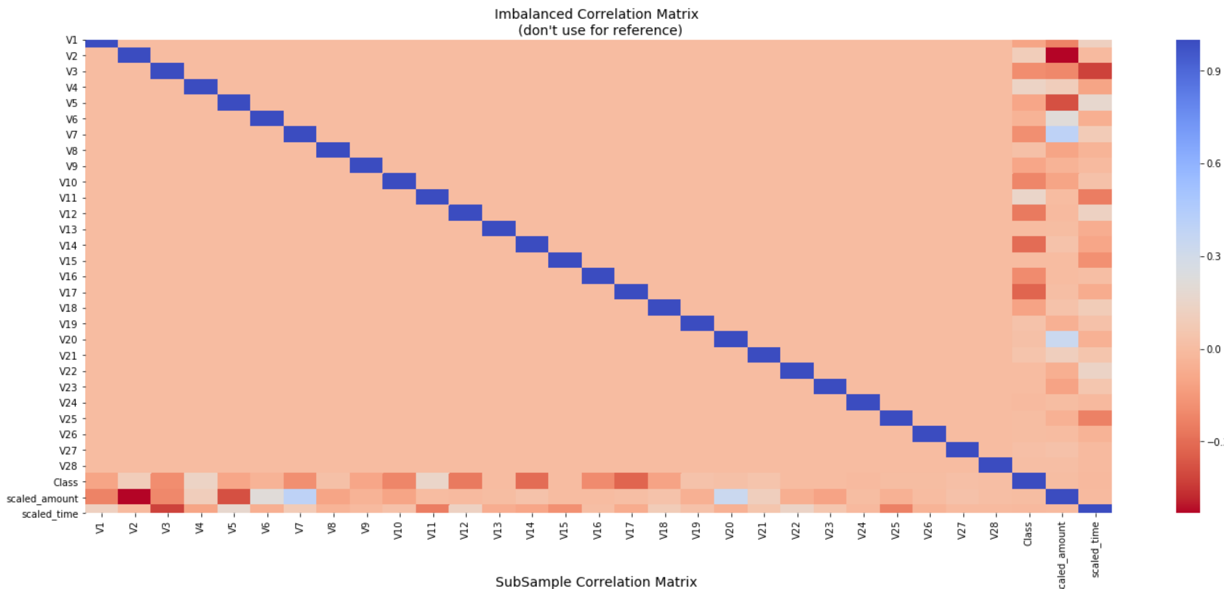
In [13]:
```
new_df.shape
```

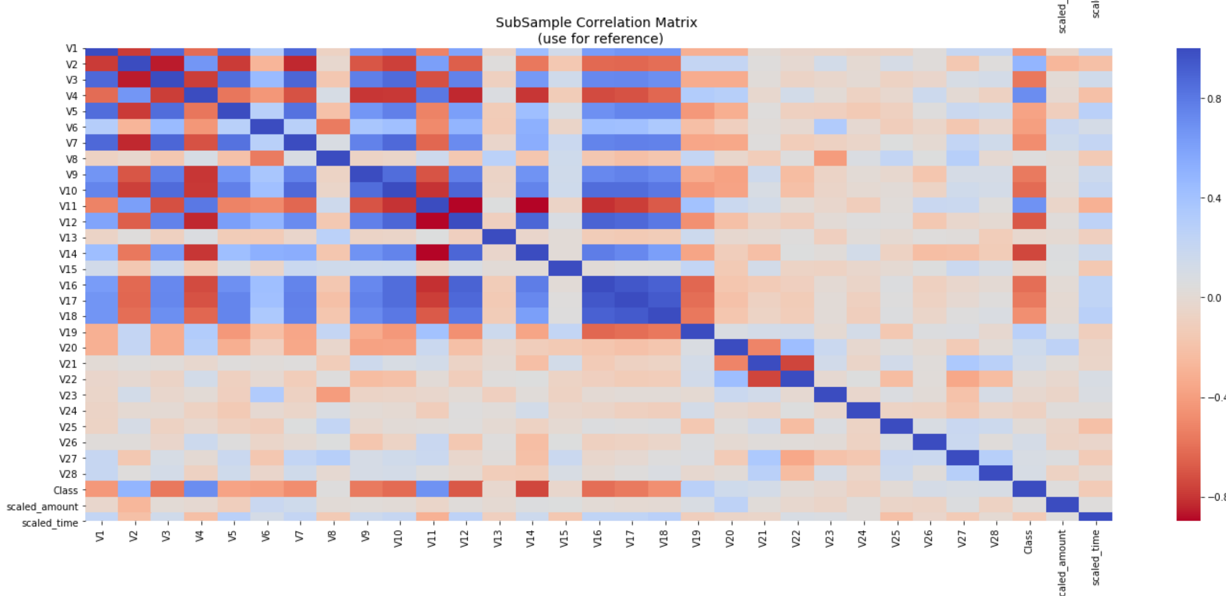
Out[13]:
```
(984, 31)
```

In [14]:
```
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))

sns.countplot('Class', data=new_df, palette=colors)
plt.title('Equally Distributed Classes', fontsize=14)
plt.show()
```

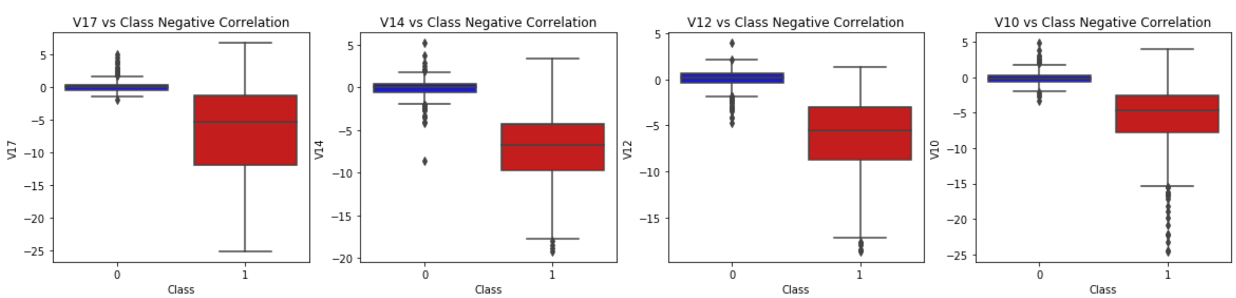
```
Distribution of the Classes in the subsample dataset
1    0.5
0    0.5
Name: Class, dtype: float64
```

In [15]:
```
f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame
# df.corr() computes pairwise correlation of columns, excluding NA/null val
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)",


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



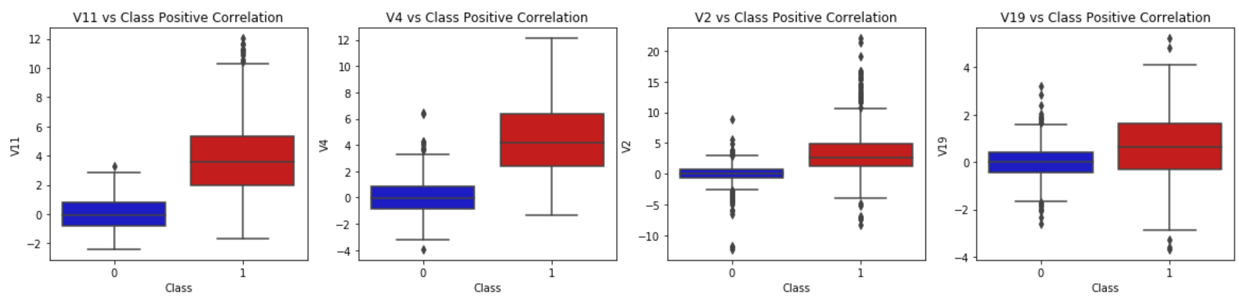


In [16]:
```
f, axes = plt.subplots(ncols=4, figsize=(20,4))

# Negative Correlations with our Class (The lower our feature value the mor
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="V14", data=new_df, palette=colors, ax=axes[1])
axes[1].set_title('V14 vs Class Negative Correlation')


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


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

plt.show()
```



The imbalanced correlation matrix does not tell much because the classes in the dataset are highly imbalanced. But balanced chart does show some positive and negative correlation. For example, V17, V14, V12 and V10 are negatively correlated. The lower the value of this parameters - the higher the chance of a transaction to be fraud.

In [17]:
```
f, axes = plt.subplots(ncols=4, figsize=(20,4))

# Positive correlations (The higher the feature the probability increases t
sns.boxplot(x="Class", y="V11", data=new_df, palette=colors, ax=axes[0])
axes[0].set_title('V11 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="V2", data=new_df, palette=colors, ax=axes[2])
axes[2].set_title('V2 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()
```



V2, V4, V11, and V19 are positively correlated. Notice how the higher these values are, the more likely the end result will be a fraud transaction.

In [ ]: