**Import the pandas library using the import statement and alias it as pd. Then, I have use the pd.read_csv() function to read in the CSV file located at the specified path and assign the resulting dataframe to the variable dataset.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
dataset=pd.read_csv(".//creditcard.csv")
```

**To display the first few rows of a dataset

In [3]:
```
dataset.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

5 rows × 31 columns

**To display the columns of a dataset

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

Out[4]:
```
Index(['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'],
      dtype='object')
```

**To display the value counts of the 'Class' column

In [5]:
```
dataset['Class'].value_counts()
```

Out[5]:
```
0    284315
1       492
Name: Class, dtype: int64
```

**To check the number of missing values in each column of the dataset

In [6]:
```
dataset.isna().sum()
```

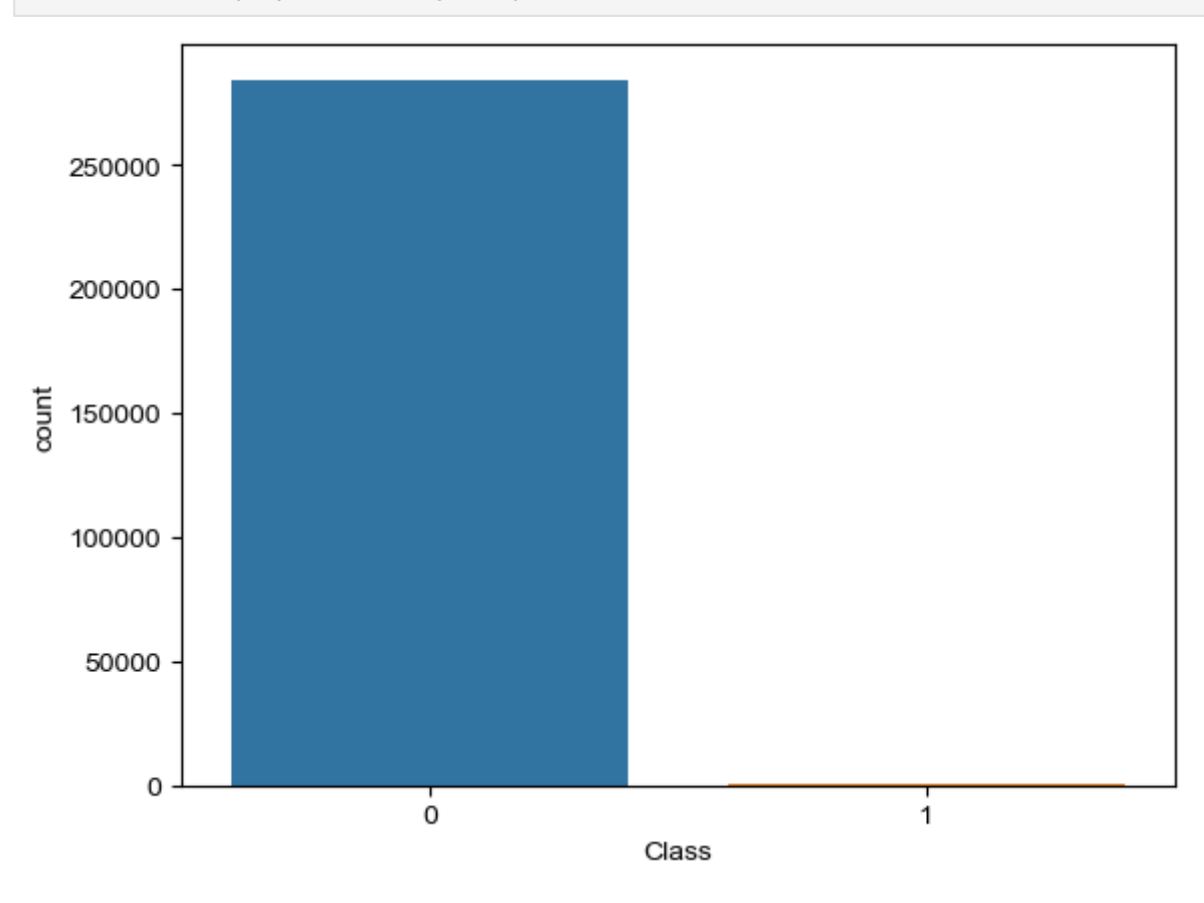
Out[6]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

**The seaborn module is a visualization library based on matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. The matplotlib.pyplot module is a collection of functions that make matplotlib work like MATLAB. It provides a convenient way to create plots and charts. Once these modules are imported, we can use their functions to create various types of plots and visualizations in the Jupyter Notebook.

In [7]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

**plotting count of classes

In [8]:
```
sns.countplot(x=dataset["Class"])
sns.set_theme(style="whitegrid")
```



**Checking duplicacy in the given dataset

In [12]:
```
dataset.duplicated().sum()
```

Out[12]:
```
1081
```

**Drop the dupicate data

In [13]:
```
dataset=dataset.drop_duplicates()
```

In [14]:
```
dataset.duplicated().sum()
```

Out[14]:
```
0
```

In [15]:
```
dataset['Class'].value_counts()
```

Out[15]:
```
0    283253
1       473
Name: Class, dtype: int64
```

**Standard scaler uses mean and standard deviation where as robust scaler used median and IQR (inter quartile range) and also it is less prone to outliers as compared to Standard Scaler

In [16]:
```
from sklearn.preprocessing import RobustScaler
rbslr=RobustScaler()
dataset['Amount']=rbslr.fit_transform(dataset['Amount'].values.reshape(-1,1))
dataset['Time']=rbslr.fit_transform(dataset['Time'].values.reshape(-1,1))
```

In [17]:
```
dataset.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	-0.995290	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	-0.995290	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	-0.995279	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	-0.995279	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	-0.995267	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

**splitting the data

In [18]:
```
X=dataset.drop(['Class'],axis=1)
Y=dataset['Class']
```

**random under sampling

**shufle before creating subsamples

In [19]:
```
dataset=dataset.sample(frac=1)
```

In [20]:
```
# fraud-classes 473 rows
fraud_cl=dataset.loc[dataset['Class']==1]
n_fraud_cls=dataset.loc[dataset['Class']==0][:473]

df=pd.concat([fraud_cl,n_fraud_cls])

#shuffle rows
balanced_df=df.sample(frac=1,random_state=42)
```

In [21]:
```
balanced_df.head()
```

Out[21]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
208651	0.617188	0.630579	1.183631	-5.066283	2.179903	-0.703376	-0.103614	-3.490350	1.094734
221018	0.678097	-3.367770	0.099249	-6.148487	3.401955	0.458307	-1.571630	-1.358708	0.672409
249852	0.821634	1.859722	-0.385551	-0.045093	1.279660	-0.575175	0.244463	-0.768012	0.179253
151103	0.116725	1.707857	0.024881	-0.488140	3.787548	1.139451	2.914673	-0.743358	0.699136
39183	-0.528403	-0.964567	-1.643541	-0.187727	1.158253	-2.458336	0.852222	2.785163	-0.303609

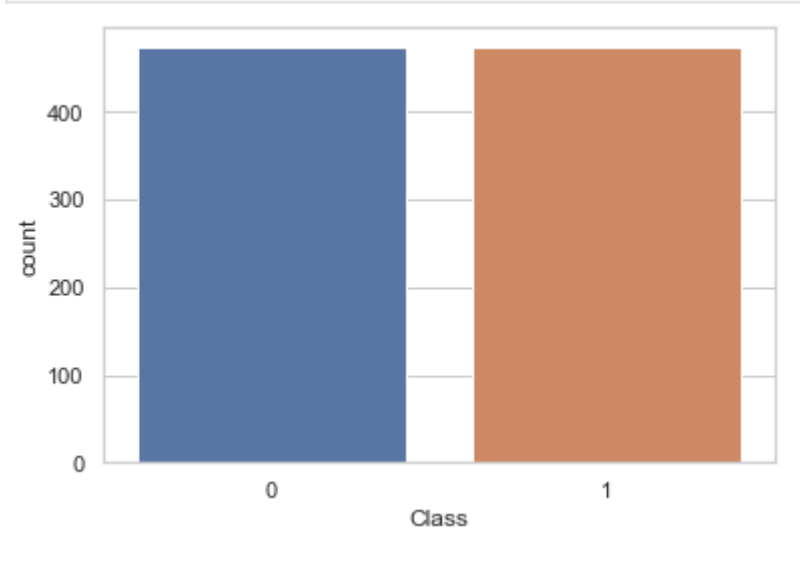
5 rows × 31 columns

In [22]:
```
balanced_df['Class'].value_counts()
```

Out[22]:
```
1    473
0    473
Name: Class, dtype: int64
```

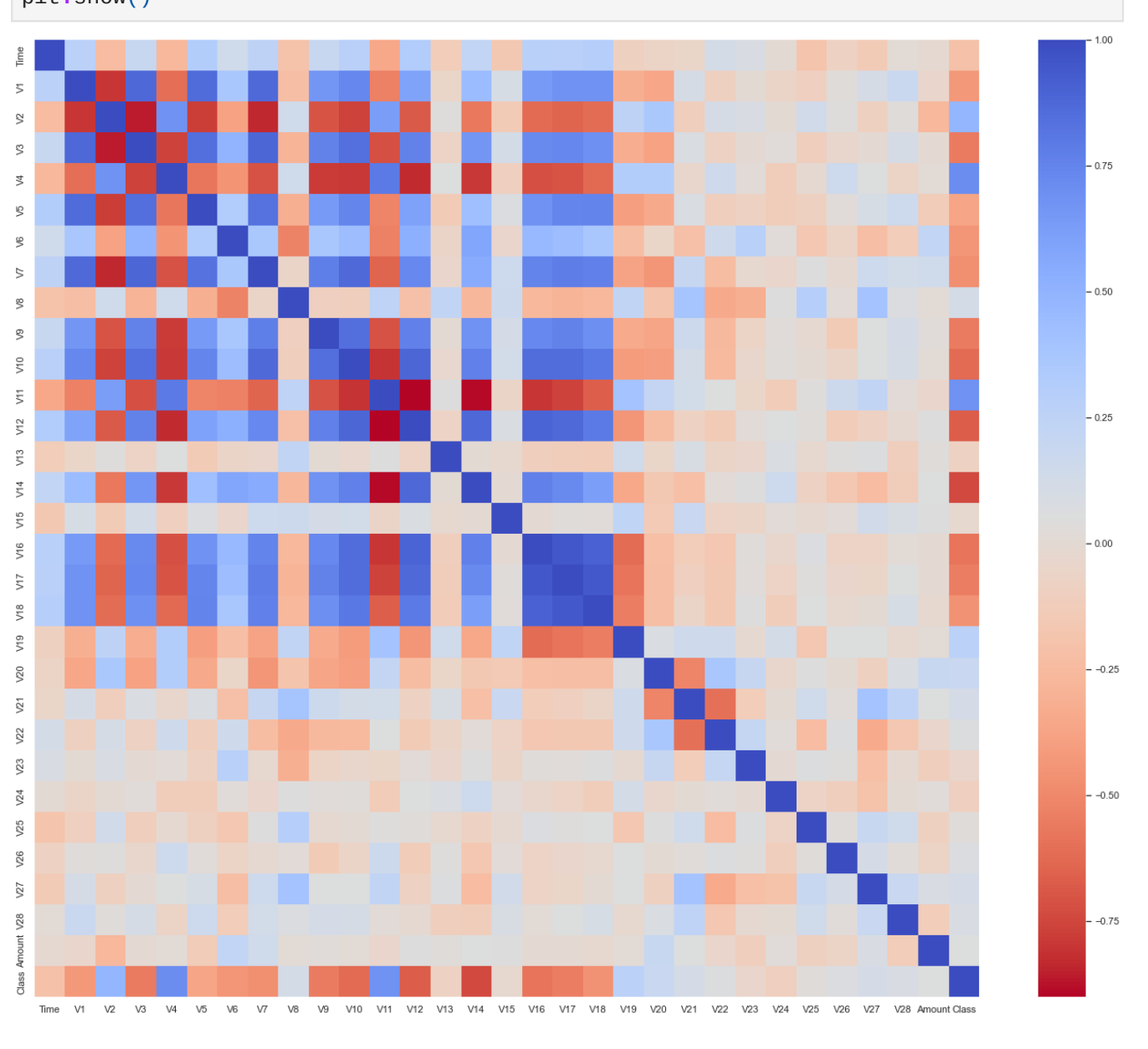
Plot the balanced data

In [23]:
```
sns.countplot(x=balanced_df["Class"])
sns.set_theme(style="whitegrid")
```



Plot coorelation matrix

In [25]:
```
#coorelation matrices
plt.subplots(figsize=(24,20))
sub_sample_corr = balanced_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20})
plt.show()
```

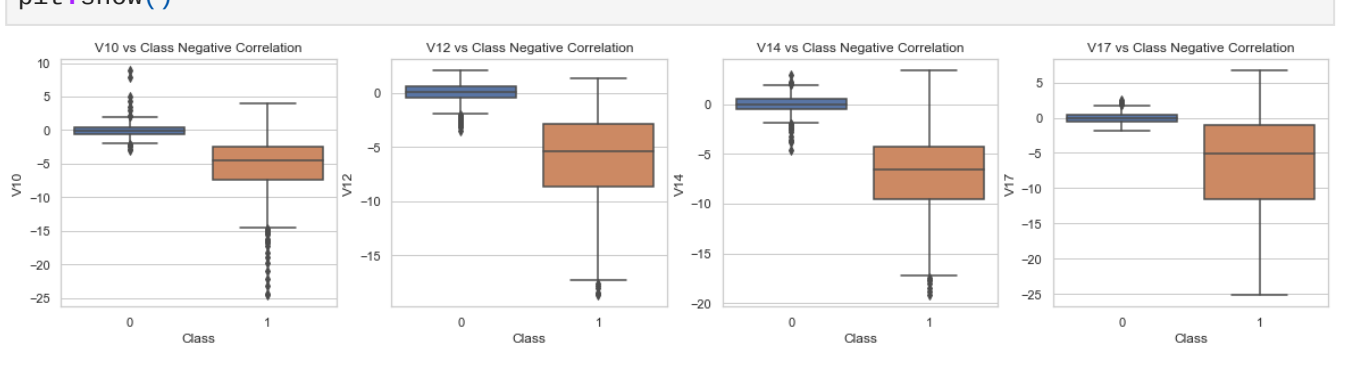


**From the above Plotting

**V10,V12,V14,V17 are highle negatively correlated

**V2,V4,V11,V19 are higly postitive correlated

In [26]:
```
# boxplot
# Negative Correlations vs Class
f, axes = plt.subplots(ncols=4, figsize=(20,4))

sns.boxplot(x="Class", y="V10", data=balanced_df, ax=axes[0])
axes[0].set_title('V10 vs Class Negative Correlation')

sns.boxplot(x="Class", y="V12", data=balanced_df, ax=axes[1])
axes[1].set_title('V12 vs Class Negative Correlation')

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

sns.boxplot(x="Class", y="V17", data=balanced_df,ax=axes[3])
axes[3].set_title('V17 vs Class Negative Correlation')

plt.show()
```



**As we can see there are outliers but not extreme ouliers. In fraud detection,ouliers are important beacuse to detect fraud we need alot of information. And removing ouliers will result in loss of information which may can affect the accuracy. So we are not removing the outliers

Balanced Dataset information

In [27]:
```
balanced_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 946 entries, 208651 to 261473
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    946 non-null    float64
 1   V1      946 non-null    float64
 2   V2      946 non-null    float64
 3   V3      946 non-null    float64
 4   V4      946 non-null    float64
 5   V5      946 non-null    float64
 6   V6      946 non-null    float64
 7   V7      946 non-null    float64
 8   V8      946 non-null    float64
 9   V9      946 non-null    float64
 10  V10     946 non-null    float64
 11  V11     946 non-null    float64
 12  V12     946 non-null    float64
 13  V13     946 non-null    float64
 14  V14     946 non-null    float64
 15  V15     946 non-null    float64
 16  V16     946 non-null    float64
 17  V17     946 non-null    float64
 18  V18     946 non-null    float64
 19  V19     946 non-null    float64
 20  V20     946 non-null    float64
 21  V21     946 non-null    float64
 22  V22     946 non-null    float64
 23  V23     946 non-null    float64
 24  V24     946 non-null    float64
 25  V25     946 non-null    float64
 26  V26     946 non-null    float64
 27  V27     946 non-null    float64
 28  V28     946 non-null    float64
 29  Amount  946 non-null    float64
 30  Class   946 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 236.5 KB
```

In [28]:
```
x=balanced_df.drop('Class',axis=1)
y=balanced_df['Class']
```

In [29]:
```
x.head()
```

Out[29]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
208651	0.617188	0.630579	1.183631	-5.066283	2.179903	-0.703376	-0.103614	-3.490350	1.094734
221018	0.678097	-3.367770	0.099249	-6.148487	3.401955	0.458307	-1.571630	-1.358708	0.672409
249852	0.821634	1.859722	-0.385551	-0.045093	1.279660	-0.575175	0.244463	-0.768012	0.179253
151103	0.116725	1.707857	0.024881	-0.488140	3.787548	1.139451	2.914673	-0.743358	0.699136
39183	-0.528403	-0.964567	-1.643541	-0.187727	1.158253	-2.458336	0.852222	2.785163	-0.303609

5 rows × 30 columns

**The train_test_split function takes three arguments: the data we want to split (in this case x and y), the size of the test set (here set to 0.3 or 30%), and an optional random state for reproducibility.

**The function returns four arrays: x_train and y_train, which contain the training data, and x_test and y_test, which contain the testing data. These arrays can then be used to train and evaluate a machine learning model.

In [30]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

conversion of X from dataframe to array so that we can use it for prediction

In [31]:
```
x_test,x_train=x_test.values,x_train.values
```

**Import the LogisticRegression class from the linear_model module, RandomForestClassifier class from the ensemble module, SVC class from the svm module, and various functions from the metrics module. These functions include accuracy_score, classification_report, ConfusionMatrixDisplay, precision_score, recall_score, f1_score, roc_auc_score, and confusion_matrix.

We can use these modules and functions to perform various machine learning tasks such as classification and evaluation of classification models.

In [32]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report,ConfusionMatrixDisp
```

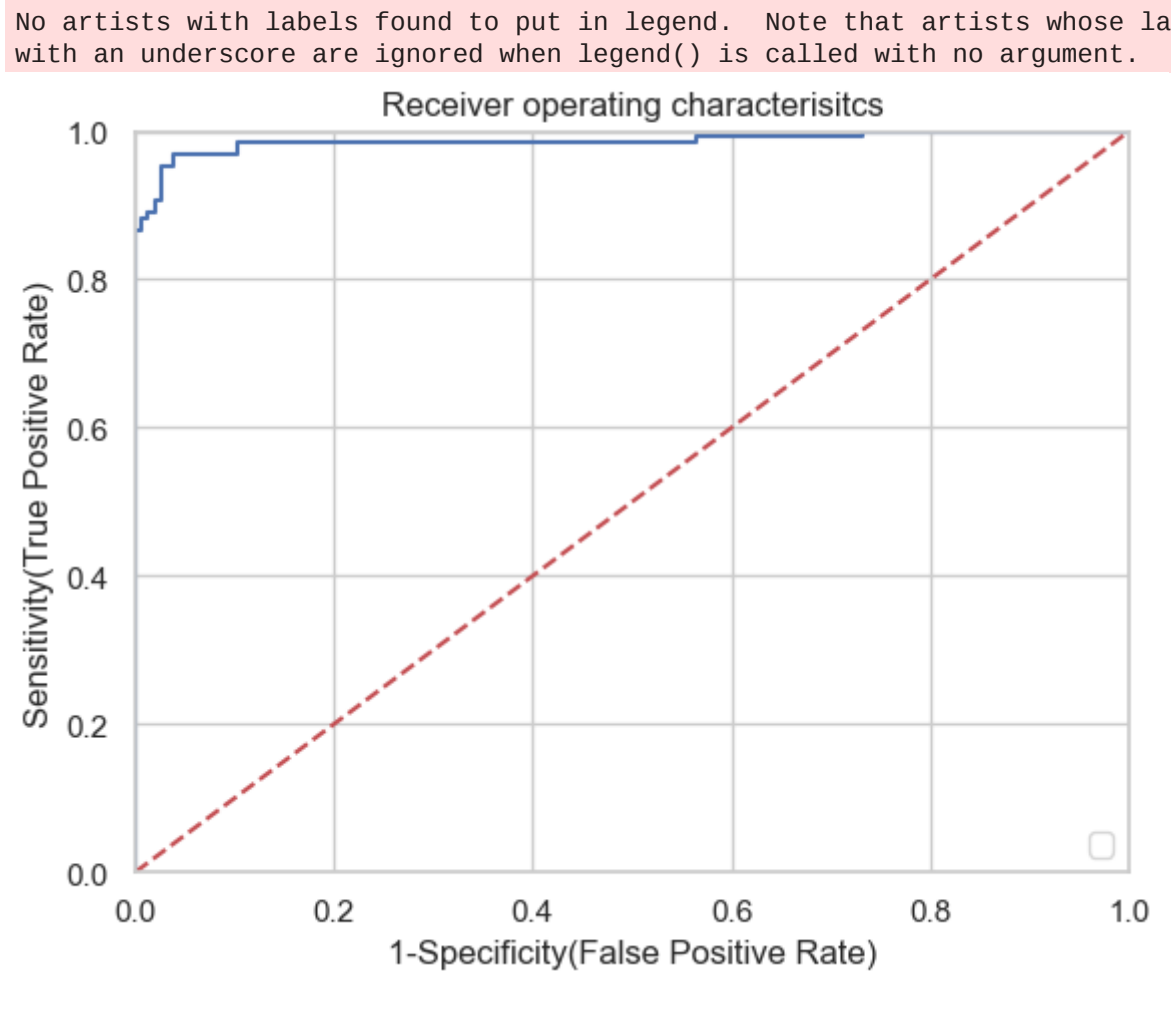
In [33]:
```
models={
    "Logistic Regression":LogisticRegression(),
    "Rnadom Forest":RandomForestClassifier(),
    "SVC":SVC()
}

for i in range(len(list(models))):
               model=list(models.values())[i]
               model.fit(x_train,y_train)

               #prediction
               y_train_pred=model.predict(x_train)
               y_test_pred=model.predict(x_test)

               #training performance
               model_train_accuracy=accuracy_score(y_train,y_train_pred)
               model_train_f1=f1_score(y_train,y_train_pred,average='weighted')
               model_train_precision=precision_score(y_train,y_train_pred)
               model_train_recall=recall_score(y_train,y_train_pred)
               model_train_rocauc_score=roc_auc_score(y_train,y_train_pred)


               #test performance
               model_test_accuracy=accuracy_score(y_test,y_test_pred)
               model_test_f1=f1_score(y_test,y_test_pred,average='weighted')
               model_test_precision=precision_score(y_test,y_test_pred)
               model_test_recall=recall_score(y_test,y_test_pred)
               model_test_rocauc_score=roc_auc_score(y_test,y_test_pred)


               print(list(models.keys())[i])

               print('Model performance for training set')
               print("- Accuracy: {:.4f}" .format(model_train_accuracy))
               print("- F1 score: {:.4f}" .format(model_train_f1))

               print("- Precisiom: {:.4f}" .format(model_train_precision))
               print("- Recall: {:.4f}" .format(model_train_recall))
               print("- Roc Auc Score:{:.4f}" .format(model_train_rocauc_score))

               print('---------')
               print('Model performance for test set')
               print("- Accuracy: {:.4f}" .format(model_test_accuracy))
               print("- F1 score: {:.4f}" .format(model_test_f1))

               print("- Precisiom: {:.4f}" .format(model_test_precision))
               print("- Recall: {:.4f}" .format(model_test_recall))
               print("- Roc Auc Score:{:.4f}".format(model_test_rocauc_score))
               print('='*10)
               print(confusion_matrix(y_test,y_test_pred))
               print(accuracy_score(y_test,y_test_pred))

               print('='*35)
               print('\n')
```

```
Logistic Regression
Model performance for training set
- Accuracy: 0.9532
- F1 score: 0.9531
- Precisiom: 0.9807
- Recall: 0.9242
- Roc Auc Score:0.9531
---------
Model performance for test set
- Accuracy: 0.9437
- F1 score: 0.9436
- Precisiom: 0.9704
- Recall: 0.9161
- Roc Auc Score:0.9439
==========
[[137   4]
 [ 12 131]]
0.9436619718309859
===================================


Rnadom Forest
Model performance for training set
- Accuracy: 1.0000
- F1 score: 1.0000
- Precisiom: 1.0000
- Recall: 1.0000
- Roc Auc Score:1.0000
---------
Model performance for test set
- Accuracy: 0.9401
- F1 score: 0.9401
- Precisiom: 0.9773
- Recall: 0.9021
- Roc Auc Score:0.9404
==========
[[138   3]
 [ 14 129]]
0.9401408450704225
===================================


SVC
Model performance for training set
- Accuracy: 0.9471
- F1 score: 0.9470
- Precisiom: 0.9836
- Recall: 0.9091
- Roc Auc Score:0.9470
---------
Model performance for test set
- Accuracy: 0.9296
- F1 score: 0.9295
- Precisiom: 0.9695
- Recall: 0.8881
- Roc Auc Score:0.9299
==========
[[137   4]
 [ 16 127]]
0.9295774647887324
===================================
```

From the above we can see that Logistic Regression performing better,so creating the ROC curve to check how better the model is trained

In [284…
```
from sklearn.metrics import roc_auc_score,roc_curve
plt.figure()

auc_models=[
    {
        'label':'Logistic Regression',
        'model': LogisticRegression(random_state=0),
         'auc':  94.4
    },
]

for algo in auc_models:
    model=algo['model']
    model.fit(x_train,y_train)

    # tpr and fpr
    fpr,tpr,threshold=roc_curve(y_test,model.predict_proba(x_test)[:,1])

    # area under curve
    plt.plot(fpr,tpr)

    plt.plot([0,1],[0,1],'r--')
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.xlabel('1-Specificity(False Positive Rate)')
    plt.ylabel('Sensitivity(True Positive Rate)')
    plt.title('Receiver operating characterisitcs')
    plt.legend(loc='lower right')
    plt.savefig('auc.png')
    plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start
with an underscore are ignored when legend() is called with no argument.
```



predicting the test set

In [34]:
```
y_pred=model.predict(x_test)
```

In [35]:
```
y_pred
```

Out[35]:
```
array([0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1])
```

Creating the pickle file

In [36]:
```
import pickle
filename='credit_Card'
pickle.dump(models,open(filename,'wb'))
```

Size of test and train set

In [37]:
```
x_test.shape
```

Out[37]:
```
(284, 30)
```

In [38]:
```
y_test.shape
```

Out[38]:
```
(284,)
```

In [41]:
```
x_train.shape
```

Out[41]:
```
(662, 30)
```

In [42]:
```
y_train.shape
```

Out[42]:
```
(662,)
```

Checking if the model is correctly predicting

In [44]:
```
y_pred = model.predict([[-0.906846,1.234235046,3.019740421,-4.304596885,4.73279513,3
y_pred
```

Out[44]:
```
array([1])
```