Vijeet Sharma

Credit Card Fraud Detection

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
import category_encoders as ce
import plotly.graph_objects as go
import plotly.express as px
```

In [2]:
```
# data read

df = pd.read_json('transactions.txt', lines = True)
```

In [3]:
```
# data

df.head()
```

Out[3]:

	accountNumber	customerId	creditLimit	availableMoney	transactionDateTime	transactionAr
0	737265056	737265056	5000	5000.0	2016-08-13T14:27:32	
1	737265056	737265056	5000	5000.0	2016-10-11T05:05:54	
2	737265056	737265056	5000	5000.0	2016-11-08T09:18:39	
3	737265056	737265056	5000	5000.0	2016-12-10T02:14:50	
4	830329091	830329091	5000	5000.0	2016-03-24T21:04:46	

5 rows × 29 columns

Summary Statistics

In [4]:
```
# Structure of the data

print("Total no of records are: ",df['accountNumber'].count())
print("Total no of features are: ",df.shape[1])
```

```
Total no of records are:  786363
Total no of features are:  29
```

In [5]:
```
# data type of each column

df.dtypes
```

Out[5]:
```
accountNumber                  int64
customerId                     int64
creditLimit                    int64
availableMoney               float64
transactionDateTime           object
transactionAmount            float64
merchantName                  object
acqCountry                    object
merchantCountryCode           object
posEntryMode                  object
posConditionCode              object
merchantCategoryCode          object
currentExpDate                object
accountOpenDate               object
dateOfLastAddressChange       object
cardCVV                        int64
enteredCVV                     int64
cardLast4Digits                int64
transactionType               object
echoBuffer                    object
currentBalance               float64
merchantCity                  object
merchantState                 object
merchantZip                   object
cardPresent                     bool
posOnPremises                 object
recurringAuthInd              object
expirationDateKeyInMatch        bool
isFraud                         bool
dtype: object
```

- From data Frame eyeballing we can see columns like merchantState, merchantZip have empty records. Hence converting empty records into "Null values". This is essential to get an estimate of null values.

In [6]:
```
# Checking null values

df = df.replace(r'', np.NaN)
df.isna().sum()
```

Out[6]:
```
accountNumber                    0
customerId                       0
creditLimit                      0
availableMoney                   0
transactionDateTime              0
transactionAmount                0
merchantName                     0
acqCountry                    4562
merchantCountryCode            724
posEntryMode                  4054
posConditionCode               409
merchantCategoryCode             0
currentExpDate                   0
accountOpenDate                  0
dateOfLastAddressChange          0
cardCVV                          0
enteredCVV                       0
cardLast4Digits                  0
transactionType                698
echoBuffer                  786363
currentBalance                   0
merchantCity                786363
merchantState               786363
merchantZip                 786363
cardPresent                      0
posOnPremises               786363
recurringAuthInd            786363
expirationDateKeyInMatch         0
isFraud                          0
dtype: int64
```

In [7]:
```
# no of unique values in each feature

df.nunique()
```

Out[7]:
```
accountNumber                 5000
customerId                    5000
creditLimit                     10
availableMoney              521915
transactionDateTime         776637
transactionAmount            66038
merchantName                  2490
acqCountry                       4
merchantCountryCode              4
posEntryMode                     5
posConditionCode                 3
merchantCategoryCode            19
currentExpDate                 165
accountOpenDate               1820
dateOfLastAddressChange       2184
cardCVV                        899
enteredCVV                     976
cardLast4Digits               5245
transactionType                  3
echoBuffer                       0
currentBalance              487318
merchantCity                     0
merchantState                    0
merchantZip                      0
cardPresent                      2
posOnPremises                    0
recurringAuthInd                 0
expirationDateKeyInMatch         2
isFraud                          2
dtype: int64
```

Data Pre-processing and general findings

In [8]:
```
# Drop non-meaningfull features (Features which only contain null-values)

df = df.drop(columns=['echoBuffer', 'merchantCity','merchantState', 'merchant
df.head()
```

Out[8]:

	accountNumber	customerId	creditLimit	availableMoney	transactionDateTime	transactionAr
0	737265056	737265056	5000	5000.0	2016-08-13T14:27:32	
1	737265056	737265056	5000	5000.0	2016-10-11T05:05:54	
2	737265056	737265056	5000	5000.0	2016-11-08T09:18:39	
3	737265056	737265056	5000	5000.0	2016-12-10T02:14:50	
4	830329091	830329091	5000	5000.0	2016-03-24T21:04:46	

5 rows × 23 columns

In [9]:
```
# convert data columns to appropriate data types

#convert below columns to string
toString = ['accountNumber','customerId','cardCVV','enteredCVV','cardLast4Dig
df[toString] = df[toString].astype(str)


# convert below columns to DateTime Format
df['transactionDateTime'] =  pd.to_datetime(df['transactionDateTime'], infer_
df['currentExpDate'] =  pd.to_datetime(df['currentExpDate'], infer_datetime_f
df['accountOpenDate'] =  pd.to_datetime(df['accountOpenDate'], infer_datetime
df['dateOfLastAddressChange'] = pd.to_datetime(df['dateOfLastAddressChange'],
```

In [10]:
```
# summary of numerical data types

df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%
creditLimit	786363.0	10759.464459	11636.174890	250.00	5000.00	7500.00	15000.000
availableMoney	786363.0	6250.725369	8880.783989	-1005.63	1077.42	3184.86	7500.000
transactionAmount	786363.0	136.985791	147.725569	0.00	33.65	87.90	191.480
currentBalance	786363.0	4508.739089	6457.442068	0.00	689.91	2451.76	5291.095

In [11]:
```
# summary of categorical and boolean Variables

df.describe(include = ['object', 'bool'])
```

Out[11]:

	accountNumber	customerId	merchantName	acqCountry	merchantCountryCode	posEr
count	786363	786363	786363	781801	785639	
unique	5000	5000	2490	4	4	
top	380680241	380680241	Uber	US	US	
freq	32850	32850	25613	774709	778511	

In [12]:
```
# summary of date column

df.describe(include = ['datetime']).T
```

Out[12]:

	count	unique	top	freq	first	last
transactionDateTime	786363	776637	2016-05-28 14:24:41	4	2016-01-01 00:01:02	2016-12-30 23:59:45
currentExpDate	786363	165	2029-03-01 00:00:00	5103	2019-12-01 00:00:00	2033-08-01 00:00:00
accountOpenDate	786363	1820	2014-06-21 00:00:00	33623	1989-08-22 00:00:00	2015-12-31 00:00:00
dateOfLastAddressChange	786363	2184	2016-03-15 00:00:00	3819	1989-08-22 00:00:00	2016-12-30 00:00:00

Reversal Transactions

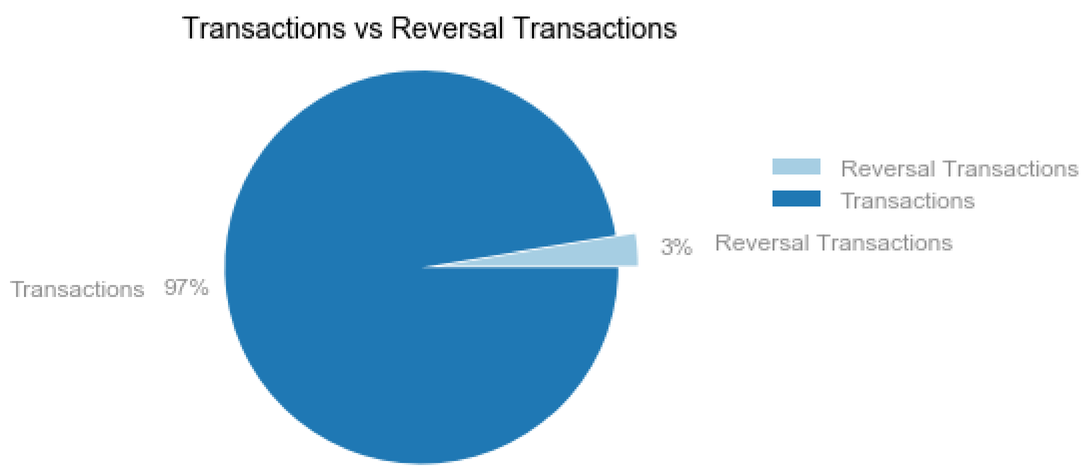
I found the reverse transactions by sub-setting the data with transactionType as Reversal.

In [13]:
```
# Reversals

df_Reversal = df[df['transactionType'] == "REVERSAL"]
print("Total no of Reversal Transactions are:", df_Reversal.shape[0])
print("Total Dollar Amount of Reversal Transactions:", df_Reversal['transacti
```

```
Total no of Reversal Transactions are: 20303
Total Dollar Amount of Reversal Transactions: 2821792.5
```

In [14]:

```
# Transactions vs Reversal Transactions

Total_Transaction = df['customerId'].count()
Reversal_Transaction = df_Reversal.shape[0]
color_palette_list= sns.color_palette("Paired")
fig, ax = plt.subplots()
plt.rcParams['font.sans-serif'] = 'Arial'
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['text.color'] = '#909090'
plt.rcParams['axes.labelcolor']= '#909090'
plt.rcParams['xtick.color'] = '#909090'
plt.rcParams['ytick.color'] = '#909090'
plt.rcParams['font.size']=12
labels = ['Reversal Transactions',
         'Transactions']
percentages = [(Reversal_Transaction/Total_Transaction) * 100, 100 - (Reversa
explode=(0.1,0)
ax.pie(percentages, explode=explode, labels=labels,
       colors=color_palette_list[0:2], autopct='%1.0f%%',
       shadow=False, startangle=0,
       pctdistance=1.2,labeldistance=1.4)
ax.axis('equal')
ax.set_title("Transactions vs Reversal Transactions")
ax.legend(frameon=False, bbox_to_anchor=(1.5,0.8))
plt.show()
```



Identifying multi-swipe transactions

For identifying multiple swipe transactions there will be 2 or more transaction types with exact same entries.

Some undertaken assumptions:

1) The transaction type feature contains NAN values. For this purpose the NANs are ignored. Later they are included in the calculation to get an interesting observation.

2) Muti-swipe scenario has been assumed to be a situation where the seller swipes the card multiple times. This swiping window has been assumed to be of 120 seconds (since itss a short span).

3) Multi swipe has been assumed for other pos-methods as well (like online payment- the amount gets debited twice).

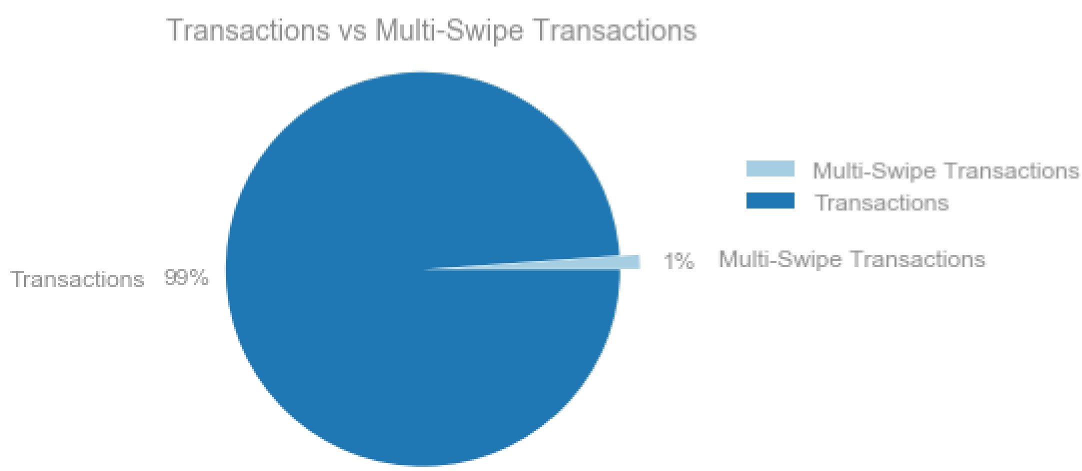
In [15]:
```
# Multi-Swipe Transactions

m2 = pd.Timedelta(minutes = 2)
df['MultiSwipe'] = df.groupby(['customerId', 'transactionAmount', 'merchantNa
```

In [16]:
```
print("Total no of Multi-Swipe Transactions are:", df['MultiSwipe'].sum())
print("Total Dollar Amount of Multi-Swipe Transactions:", df['transactionAmou
```

```
Total no of Multi-Swipe Transactions are: 8952
Total Dollar Amount of Multi-Swipe Transactions: 1314481.8900000001
```

In [17]:

```
# Transactions vs Multi-Swipe Transactions

Total_Transaction = df['customerId'].count()
MultiSwipe_Transaction = df['MultiSwipe'].sum()
color_palette_list= sns.color_palette("Paired")
fig, ax = plt.subplots()
plt.rcParams['font.sans-serif'] = 'Arial'
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['text.color'] = '#909090'
plt.rcParams['axes.labelcolor']= '#909090'
plt.rcParams['xtick.color'] = '#909090'
plt.rcParams['ytick.color'] = '#909090'
plt.rcParams['font.size']=12
labels = ['Multi-Swipe Transactions',
         'Transactions']
percentages = [(MultiSwipe_Transaction/Total_Transaction) * 100, 100 - (Multi
explode=(0.1,0)
ax.pie(percentages, explode=explode, labels=labels,
       colors=color_palette_list[0:2], autopct='%1.0f%%',
       shadow=False, startangle=0,
       pctdistance=1.2,labeldistance=1.4)
ax.axis('equal')
ax.set_title("Transactions vs Multi-Swipe Transactions")
ax.legend(frameon=False, bbox_to_anchor=(1.5,0.8))
plt.show()
```



In [18]:

```
# Droping the Multi-Swipe Column

df.drop(['MultiSwipe'], axis = 1, inplace = True)
```

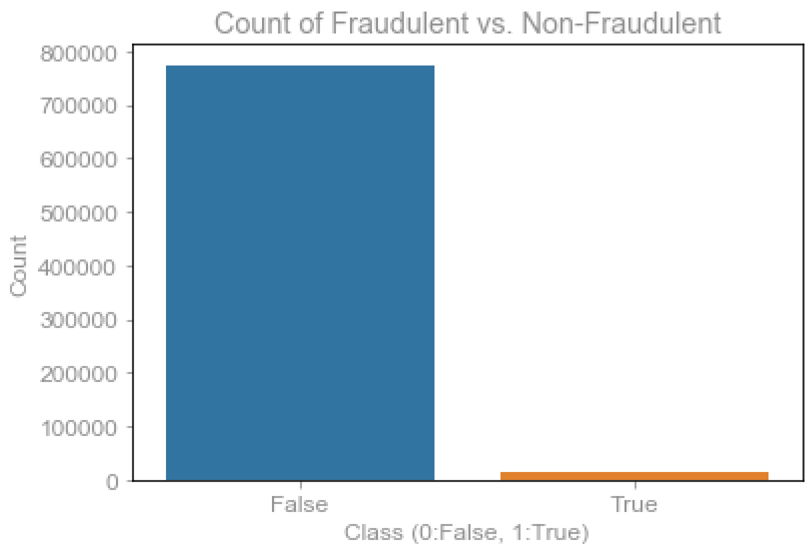
In [19]:
```
# Check the Data Distribution

fig = px.histogram(df, x = 'transactionAmount', nbins = 500)
fig.show()
```

- The data is highly right skewed as expected in case of data which relates to monetary values. As it can be observed from the graph that the frequency of low amount transactions is very high and as the transaction amount increases its frequency decreases.
- A reason for such a behaviour could be that people use credit cards daily for small amount purchases such as meals, snacks, groceries and also people do not hesitate to buy things which are not that expensive.
- But also people do not buy expensive things regularly. Consider the example, how often do you think a person would buy a cell phone ? Once a year probably.
- So this behaviour of the 'transactionAmount' variable can be clearly justified by normal human behaviour and actions.

In [20]:

```
# Fraudulent vs Non-Fraudulent

counts = df.isFraud.value_counts()
print(counts)
sns.barplot(x = counts.index, y= counts)
plt.title('Count of Fraudulent vs. Non-Fraudulent')
plt.ylabel('Count')
plt.xlabel('Class (0:False, 1:True)')
plt.show()
```

```
False    773946
True      12417
Name: isFraud, dtype: int64
```

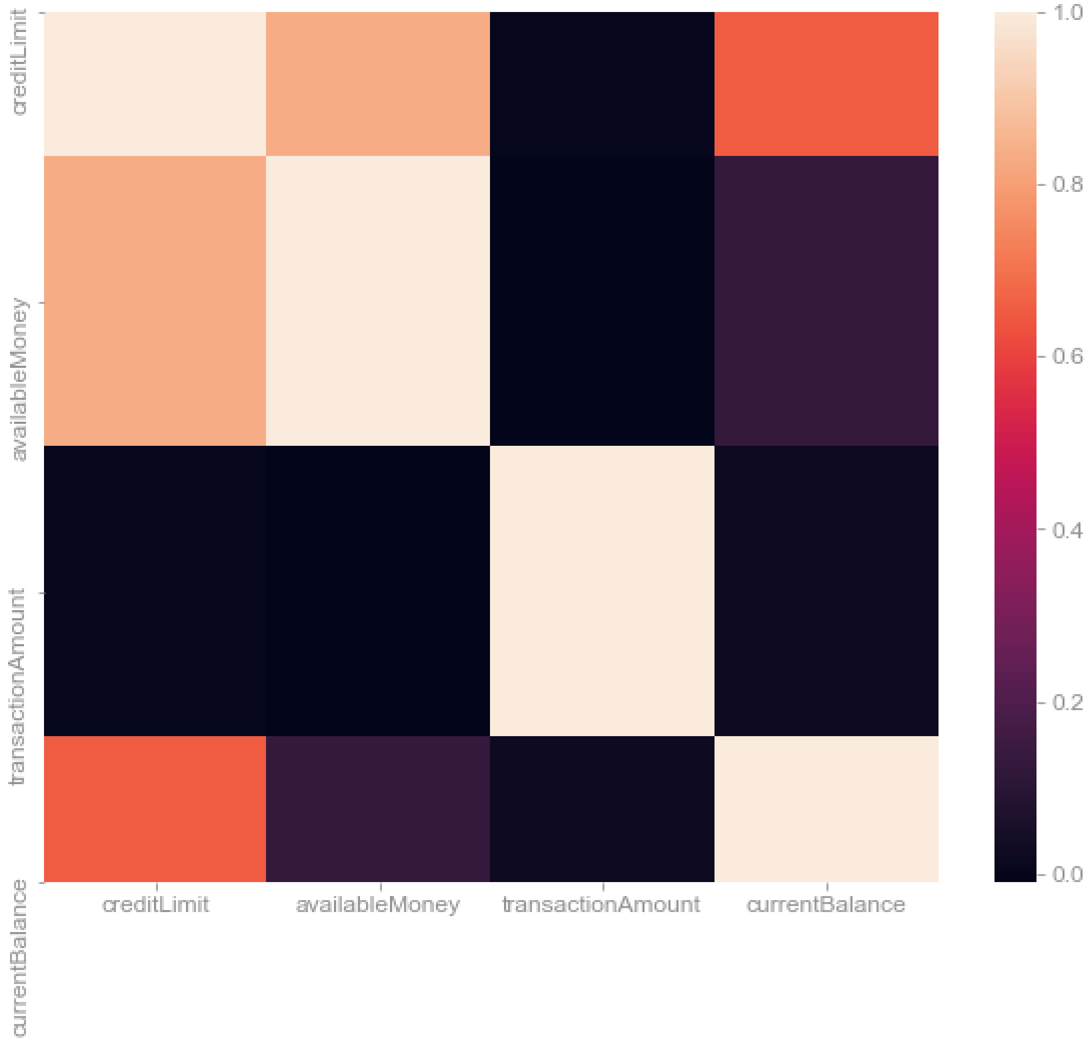


- The class distribution is highly imbalanced. Only 1.58% of the data points represent the TRUE class and since, detecting fraud is very important, we focus on minimizing Type I & Type II errors.
- Therefore, instead of using accuracy as an evaluation metric we would be using F1 score as our model evaluation metric.
- We performed under sampling to make the data set balanced.

I checked the Co-relation between the numeric variables because we can drop one of the co-linear features if there is too much co-linearity between the two features.

In [21]:

```
# heatmap to check correlation - multi colinearity between numerical variable

numeric = df.select_dtypes(include = ['int64', 'float64']).columns
corr = df[numeric].corr()
plt.figure(figsize = (10, 8))
sns.heatmap(corr)
plt.show()
```



In [22]:
```
# correlation between numeric features

corr
```

Out[22]:

	creditLimit	availableMoney	transactionAmount	currentBalance
creditLimit	1.000000	0.834977	0.005581	0.653652
availableMoney	0.834977	1.000000	-0.010070	0.129332
transactionAmount	0.005581	-0.010070	1.000000	0.023905
currentBalance	0.653652	0.129332	0.023905	1.000000

In [23]:
```
categorical = list(df.select_dtypes(include = ['object']).columns)
numeric = list(df.select_dtypes(include = ['int64', 'float64']).columns)
boolean = list(df.select_dtypes(include = ['bool']).columns)
```

In [24]:
```
# replacing Nan's with "NULL" giving it a category

df['merchantCountryCode'] = df['merchantCountryCode'].replace(np.NaN, 'NULL')
df['acqCountry'] = df['acqCountry'].replace(np.NaN, 'NULL')
df['posEntryMode'] = df['posEntryMode'].replace(np.NaN, 'NULL')
df['posConditionCode'] = df['posConditionCode'].replace(np.NaN, 'NULL')
df['transactionType'] = df['transactionType'].replace(np.NaN, 'NULL')
```

To make the data reasonable or in a human-readable form, the training data is frequently named in words. Label Encoding alludes to changing over the marks into the numeric structure in order to change over it into the machine-lucid structure

In [25]:
```
# Encode categorical features

encoders = {}
for each in categorical:
    encode = LabelEncoder()
    #encoder =  LabelEncoder(cols=[each])
    df[each] = encode.fit_transform(df[each])
    encoders[each] = encode
```

In [26]:
```
# scale Boolean features

boolean_encoders = {}
for each in boolean:
    encode = LabelEncoder()
    df[each] = encode.fit_transform(df[each])
    boolean_encoders[each] = encode
```

I also scaled the numeric variables to change the values of numeric variables in the dataset to a typical scale, without misshaping contrasts in the scopes of qualities. I performed Min-Max scaling on the numeric columns.

In [27]:
```
# scale Numeric features

scaler = MinMaxScaler()
df[numeric] = scaler.fit_transform(df[numeric])
df.head()
```

Out[27]:

	accountNumber	customerId	creditLimit	availableMoney	transactionDateTime	transactionAr
0	3554	3554	0.095477	0.117744	2016-08-13 14:27:32	0.0
1	3554	3554	0.095477	0.117744	2016-10-11 05:05:54	0.0
2	3554	3554	0.095477	0.117744	2016-11-08 09:18:39	0.0
3	3554	3554	0.095477	0.117744	2016-12-10 02:14:50	0.0
4	4082	4082	0.095477	0.117744	2016-03-24 21:04:46	0.0

5 rows × 23 columns

Min-Max scaling is done so that the range of each feature's values lies between 0 and 1. This will make reaching global minima of the model much easier hence training the model more efficient.

Why not Standard Scaling??

This transformation will change the distribution of each feature (in order to make mean 0 and standard deviation 1). Also it will have an effect on outliers as reducing standard deviation will increase outliers. I didn't want to change the outliers distributions as it will have an effect on fraud prediction. (I consider Fraud as an outlier).

Data Modelling with K-fold Cross Validation

In [28]:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from sklearn.linear_model import LogisticRegression, ElasticNetCV, SGDClassif
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import RandomizedSearchCV
import scikitplot as skplt
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc
```

In [29]:

```
# removing some non-meaningfull features

df = df.drop(columns=['transactionDateTime','currentExpDate','accountOpenDate
```

The data is highly skewed. This predisposition in the training dataset can impact many machine learning calculations, driving some to overlook the minority class totally. This is an issue as it is normally the minority class on which expectations are generally significant. I reduced the no of observations of the majority class by using the imblearn package. This is called Under-Sampling.

In [30]:

```
# undersampling

from imblearn.under_sampling import RandomUnderSampler
X = df.copy()
y = df['isFraud']
X = X.drop(columns = ['isFraud'])
ros = RandomUnderSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)
df.undersampled = pd.concat([X_resampled,y_resampled], axis = 1)
```

```
C:\Users\vijee\Anaconda\lib\site-packages\ipykernel_launcher.py:9: UserWarn
ing:

Pandas doesn't allow columns to be created via a new attribute name - see h
ttps://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
 (https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-acce
ss)
```

In [31]:

```
# Split data into training and testing datasets

y = df.undersampled['isFraud']
df.undersampled = df.undersampled.drop(columns=['isFraud'])
X_train, X_test, y_train, y_test = train_test_split(df.undersampled, y, test_
```

In [32]:

```
# Model Training

models = {}
models['LR'] = LogisticRegression(max_iter = 10000, solver = 'liblinear')
models['LDA'] = LinearDiscriminantAnalysis()
models['KNN'] = KNeighborsClassifier()
models['CART'] = DecisionTreeClassifier()
models['XGB'] = XGBClassifier()
models['RF'] = RandomForestClassifier(n_estimators=100)
models['ADA'] = AdaBoostClassifier()
#models['RidgeRegression'] = LogisticRegression(penalty = 'elasticnet', max_i


results = []
names = []

scoring = {'acc':'accuracy', 'precision':'precision', 'recall':'recall', 'f1'

for name, model in models.items():
    kfold = KFold(n_splits = 10)
    #cv_results = cross_val_score(model, X_train, y_train, cv = kfold, scorin
    scores = cross_validate(model, X_train, y_train, cv = kfold, scoring = sc
    results.append(scores)
    names.append(name)
    #print(name, ":", scores)#, "(", scores, ")"
    print(name)
    for key, value in scores.items():
        if(key == 'estimator'):
            print('Model Fitted')
            models[name] = value
        else:
            print(key, ':', value.mean())
    print("****************************************")
```

```
LR
fit_time : 0.5103176832199097
score_time : 0.008764982223510742
Model Fitted
test_acc : 0.6471028198902128
train_acc : 0.647841464519286
test_precision : 0.6557486764295303
train_precision : 0.6568334060190099
test_recall : 0.6178990497327824
train_recall : 0.6176359286014239
test_f1 : 0.6361033138513054
train_f1 : 0.6366213441938368
****************************************
LDA
fit_time : 0.07587335109710694
score_time : 0.008289980888366699
Model Fitted
test_acc : 0.6465491454778315
train_acc : 0.6477128319875608
test_precision : 0.6566638782271221
train_precision : 0.658146735017992
```

```
test_recall : 0.6127367245949845
train_recall : 0.6131892343263833
test_f1 : 0.6337834973390782
train_f1 : 0.6348620234816985
****************************************
KNN
fit_time : 0.12965595722198486
score_time : 0.11086702346801758
Model Fitted
test_acc : 0.6705589351935617
train_acc : 0.7801770742118508
test_precision : 0.6708329726800345
train_precision : 0.7848333837316959
test_recall : 0.6687781856085065
train_recall : 0.7713672921237718
test_f1 : 0.6696353928761257
train_f1 : 0.7780387309412349
****************************************
CART
fit_time : 0.24265656471252442
score_time : 0.006871771812438965
Model Fitted
test_acc : 0.6545019464383548
train_acc : 1.0
test_precision : 0.6508512979067168
train_precision : 1.0
test_recall : 0.6656256346347614
train_recall : 1.0
test_f1 : 0.6579877109640716
train_f1 : 1.0
****************************************
XGB
fit_time : 3.0592129230499268
score_time : 0.023024535179138182
Model Fitted
test_acc : 0.7527550934042069
train_acc : 0.8816798138000831
test_precision : 0.7503759347500553
train_precision : 0.8774459744486993
test_recall : 0.7569556304325663
train_recall : 0.8870083508806796
test_f1 : 0.7535515090402007
train_f1 : 0.8821971221229159
****************************************
RF
fit_time : 4.205443620681763
score_time : 0.06676383018493652
Model Fitted
test_acc : 0.7557248753351974
train_acc : 1.0
test_precision : 0.7574364135064495
train_precision : 1.0
test_recall : 0.751892779391927
train_recall : 1.0
test_f1 : 0.7545075002042962
train_f1 : 1.0
****************************************
```

```
ADA
fit_time : 1.3162132263183595
score_time : 0.03293435573577881
Model Fitted
test_acc : 0.6907925432734728
train_acc : 0.6939648630324742
test_precision : 0.6827400876523341
train_precision : 0.6856493743901068
test_recall : 0.7117373724311427
train_recall : 0.7151821262720912
test_f1 : 0.6968273398316304
train_f1 : 0.7000982096265899
***************************************
```

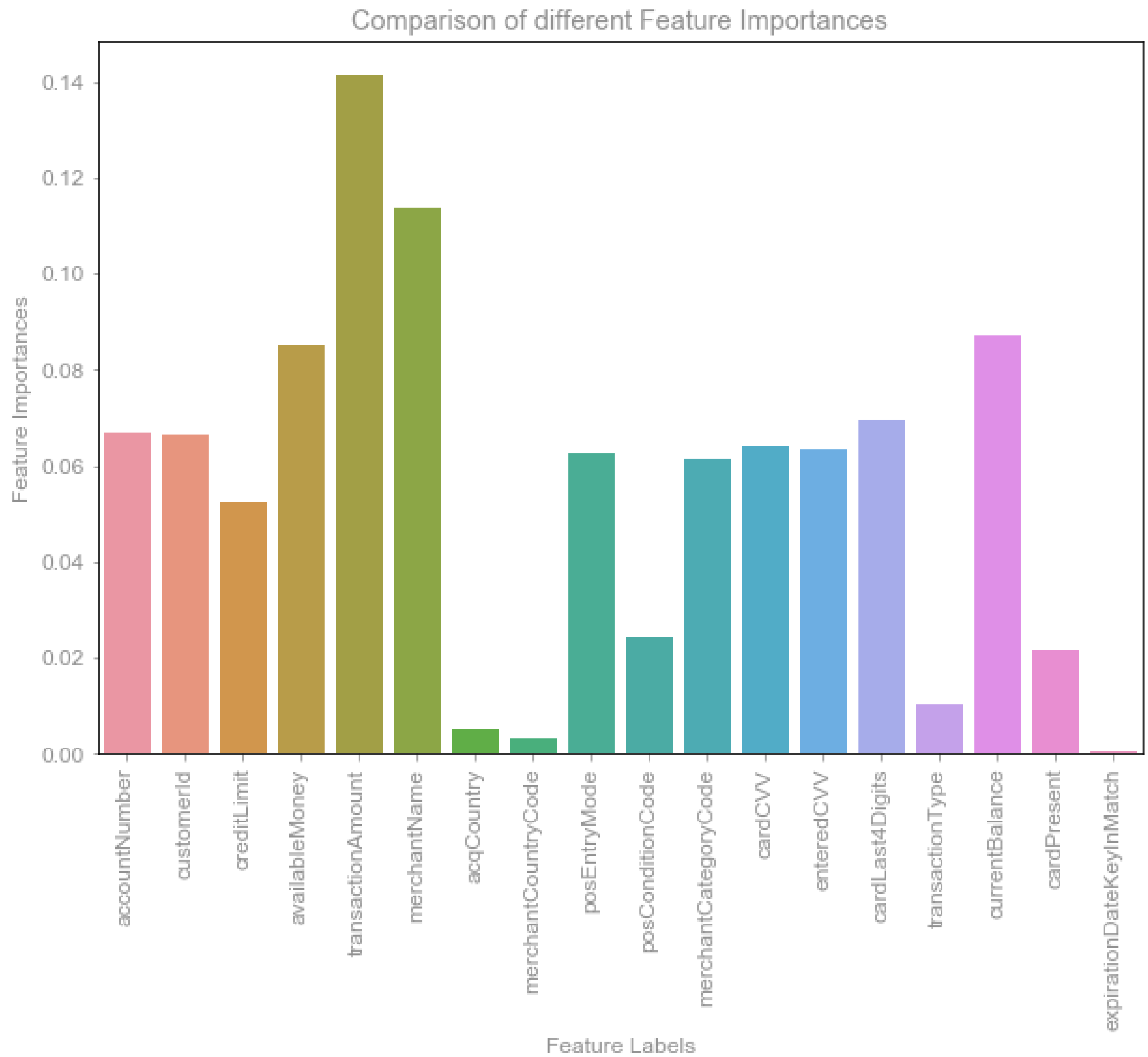
I used F1 score as an evaluation metric because if we consider Accuracy as the evaluation metrics, we would get great accuracy by training on the entire dataset. But we would not get a lot of False Negatives ie Type II error. This is an undesired case in our scenario since the inability to identify fraud transactions will lead to loss in the business and identifying frauds correctly is the whole aim of this project. Therefore, we consider F1 score as the evaluation metrics which gives equal weightage to precision and recall.

Random Forest and XGBoost performs better than all other models.

Extra - Tree Classifier for Feature Selection

In [33]:

```
from sklearn.ensemble import ExtraTreesClassifier

extra_tree_forest = ExtraTreesClassifier(n_estimators = 10, criterion ='entro


extra_tree_forest.fit(X_train, y_train)


cols = list(df.undersampled.columns)
feature_importance = extra_tree_forest.feature_importances_
plt.figure(figsize = (10, 7))
fig = sns.barplot(cols, feature_importance)
plt.xlabel('Feature Labels')
plt.ylabel('Feature Importances')
plt.title('Comparison of different Feature Importances')
fig.set_xticklabels(cols, rotation = 90)
plt.show()
```



Backward Selection for Dimensionality Reduction

In [34]:

```
# Backward selection

to_remove = ['acqCountry', 'merchantCountryCode', 'expirationDateKeyInMatch',
             'posConditionCode', 'transactionType','customerId','cardPresent'

columns = list(df.undersampled.columns)

for each in to_remove:
    try:
        columns.remove(each)
    except:
        pass

X_train, X_test, y_train, y_test = train_test_split(df.undersampled[columns],
```

I performed the Backward Selection- it starts with the full least squares model containing all features, and afterward iteratively expels the least valuable feature, each in turn. The Model performs better with fewer and important features, so I calculated feature importance using the Tree Classifier and removed the least important features.

In [35]:

```
# Model Training

models = {}
models['LR'] = LogisticRegression(max_iter = 10000, solver = 'liblinear')
models['LDA'] = LinearDiscriminantAnalysis()
models['KNN'] = KNeighborsClassifier()
models['CART'] = DecisionTreeClassifier()
models['XGB'] = XGBClassifier()
models['RF'] = RandomForestClassifier(n_estimators=100)
models['ADA'] = AdaBoostClassifier()
#models['RidgeRegression'] = LogisticRegression(penalty = 'elasticnet', max_i


results = []
names = []

scoring = {'acc':'accuracy', 'precision':'precision', 'recall':'recall', 'f1'

for name, model in models.items():
    kfold = KFold(n_splits = 10)
    #cv_results = cross_val_score(model, X_train, y_train, cv = kfold, scorin
    scores = cross_validate(model, X_train, y_train, cv = kfold, scoring = sc
    results.append(scores)
    names.append(name)
    #print(name, ":", scores)#, "(", scores, ")"
    print(name)
    for key, value in scores.items():
        if(key == 'estimator'):
            print('Model Fitted')
            models[name] = value
        else:
            print(key, ':', value.mean())
    print("****************************************")
```

```
LR
fit_time : 0.21617352962493896
score_time : 0.007982683181762696
Model Fitted
test_acc : 0.6335116069152412
train_acc : 0.6338148525292804
test_precision : 0.6543381072923243
train_precision : 0.6545640586831855
test_recall : 0.5643534085278465
train_recall : 0.5650522079124671
test_f1 : 0.6058023912369755
train_f1 : 0.6065140029661474
****************************************
LDA
fit_time : 0.03545601367950439
score_time : 0.007683086395263672
Model Fitted
test_acc : 0.63250498836597
train_acc : 0.6333450635677804
test_precision : 0.6565462189277879
```

```
train_precision : 0.6575838877746529
test_recall : 0.553971285941091
train_recall : 0.5548179316125105
test_f1 : 0.6006744163774076
train_f1 : 0.6018359634458178
***************************************
KNN
fit_time : 0.07132742404937745
score_time : 0.07952659130096436
Model Fitted
test_acc : 0.6744852872979502
train_acc : 0.7844890674828005
test_precision : 0.6737800467613992
train_precision : 0.7877732189032043
test_recall : 0.6759973510362399
train_recall : 0.778167541689817
test_f1 : 0.6746637328433169
train_f1 : 0.7829383660236197
***************************************
CART
fit_time : 0.17746036052703856
score_time : 0.006778860092163086
Model Fitted
test_acc : 0.6516840835014706
train_acc : 1.0
test_precision : 0.6470430028837363
train_precision : 1.0
test_recall : 0.6659882393433325
train_recall : 1.0
test_f1 : 0.6562894863146067
train_f1 : 1.0
***************************************
XGB
fit_time : 2.0736161708831786
score_time : 0.018578147888183592
Model Fitted
test_acc : 0.7441978347678846
train_acc : 0.8747224588933525
test_precision : 0.7400096817522901
train_precision : 0.868310831837644
test_recall : 0.7522803517605523
train_recall : 0.8831153544531753
test_f1 : 0.7459831122619202
train_f1 : 0.8756460573901039
***************************************
RF
fit_time : 4.170279955863952
score_time : 0.06816678047180176
Model Fitted
test_acc : 0.7459595122576708
train_acc : 0.9999832221020725
test_precision : 0.7448165170374751
train_precision : 0.9999775633605701
test_recall : 0.7480242608916127
train_recall : 0.9999888293118856
test_f1 : 0.7462015982240954
train_f1 : 0.9999831953949604
```

```
***************************************
ADA
fit_time : 0.9954020977020264
score_time : 0.025522255897521974
Model Fitted
test_acc : 0.6831920068562474
train_acc : 0.6871528819599922
test_precision : 0.6739212898745184
train_precision : 0.6779985245578649
test_recall : 0.7085632575445864
train_recall : 0.7116537974008262
test_f1 : 0.6906937245265741
train_f1 : 0.6944092143827401
***************************************
```

As, Random Forest performs better than other models, I will interpret Random Forest by plotting ROC Curve. Also, to increase the credibility of the model, I will do hyper-paramter tuning for the Random Forest.

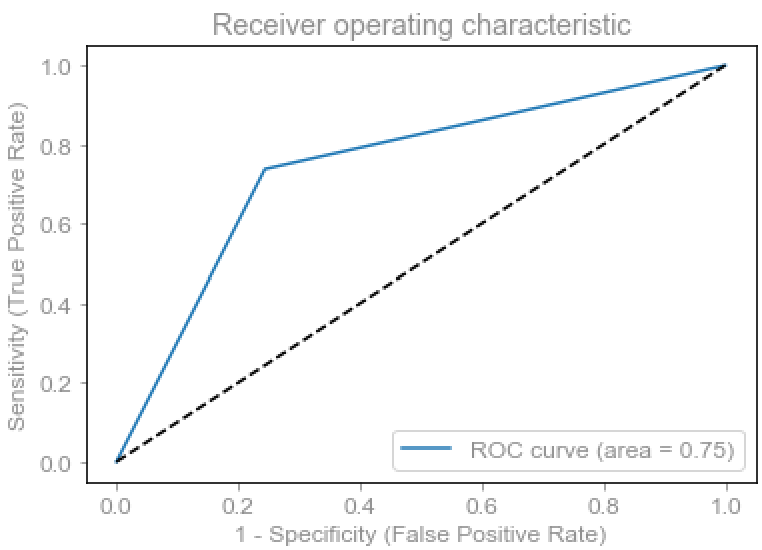
ROC Curve

In [36]:

```
## predicting on X_test ##
RF=RandomForestClassifier()
y_pred=RF.fit(X_train,y_train).predict(X_test)
```

In [37]:

```
## plotting ROC curve on  random forest trained model ##
fpr, tpr, thres = roc_curve(y_test,y_pred)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('1 - Specificity (False Positive Rate)')
plt.ylabel('Sensitivity (True Positive Rate)')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



Hyper- Parameter tuning for Random Forest to increase the performance of the model

In [38]:

```
# # Hyper- Parameter tuning for Random Forest

RFC= RandomForestClassifier(n_jobs=-1) # model is defined

#DEFINING PARAMETER VALUES TO BE SEARCHED
n_estimators1 = range(50,500,50)    # No. of trees
criterion1 = ['gini','entropy']
min_samples_split1 = range(2,5,1)  #  number of samples required to split an
max_depth1 = range(10,60,10)       # number of layers or the depth of the tre

param_grid=dict(n_estimators=n_estimators1, criterion=criterion1, min_samples
print(param_grid)

grid=RandomizedSearchCV(RFC, param_grid, cv=5,scoring='f1_weighted',n_jobs=-1
grid.fit(X_train,y_train)

params=grid.best_params_   # getting parameters for best score


print("Best F1 score",grid.best_score_)
print("Pararmeters of the best score",params)
```

```
{'n_estimators': range(50, 500, 50), 'criterion': ['gini', 'entropy'], 'min
_samples_split': range(2, 5), 'max_depth': range(10, 60, 10)}
Best F1 score 0.7465924366589791
Pararmeters of the best score {'n_estimators': 400, 'min_samples_split': 2,
'max_depth': 20, 'criterion': 'entropy'}
```

In [ ]: