Table of Contents

- - 0.1 Before we Begin:
  - 0.2 Problem Statement :
  - 0.3 Solution Approach:
    - 0.3.1 Assumptions:
  - 0.4 Objective of this Kernel:
  - 0.5 Credits
- 1 Import Packages
- 2 Custom Functions
- 3 Get Data
- 4 Save Memory Usage
- 5 Data Analysis
  - 5.1 Data Dictionary
  - 5.2 Data Preview
  - 5.3 Difference Columns Attributes - Train Vs Test
  - 5.4 Null Records Analysis
  - 5.5 Visuals
    - 5.5.1 Card Type
    - 5.5.2 ProductCD: Product Category
- 6 Prepare Data
  - 6.1 Drop duplicate TransactionID records
  - 6.2 First Drop: Drop Columns (Nulls > 90%)
  - 6.3 Imputation
    - 6.3.1 More visuals
    - 6.3.2 TransactionAmt
    - 6.3.3 M1-M6 : Matched Card details
    - 6.3.4 C1 to C14 : Addresses associated with payment card
    - 6.3.5 D series : Date related attributes
    - 6.3.6 Vesta Attributes
  - 6.4 Binning
    - 6.4.1 R_emaildomain: Receiver email domain
    - 6.4.2 P_emaildomain: Purchase email domain
    - 6.4.3 Card6 : Card Category (Debit or Credit or Charge Card)
    - 6.4.4 Card Details : card1 to card3, card5
    - 6.4.5 addr1, addr2 : Billing region, Biling country
  - 6.5 Final drop : Columns (having noise)
- 7 Data Split
  - 7.1 Train Data subset
  - 7.2 Train Data spilt: Data vs Label
  - 7.3 Numerical Attributes vs Categorical Attributes
  - 7.4 X -vs- y sets
- 8 Feature Engineering
  - 8.1 Lable Encoding
  - 8.2 Validation set
- 9 Modeling & Validation

    - 9.1 Logistic Regression
    - 9.2 XGBoost Classifier
    - 9.3 GradientBoostingClassifier
    - 9.4 RandomForest Classifier
    - 9.5 CatBoostClassifier
- 10 Model Comparison
- 11 Model Tunning
    - 11.0.1 Tunning n_estimators : Number of trees
    - 11.0.2 General Approach for Parameter Tunning
- 12 Model stacking
- 13 Model Testing
- 14 Conclusion

My Name: Sateesh Babu

Before we Begin:

I would like to thank Lynn Vesta,IEEE-CIS and Kaggle for organizing this exciting competition. This competition gave us a fantastic opportunity to learn how to deal with very large table data and to explore efficient ways so that our ML models are able to detect more accurately both fraud and non-fraud transactions.

https://www.kaggle.com/c/ieee-fraud-detection/leaderboard (https://www.kaggle.com/c/ieee-fraud-detection/leaderboard)

Problem Statement :

This competition is a binary classification problem - i.e. target variable is a binary attribute (Is the card transaction fraudlent or not?). The goal is to classify transaction into "fraudlent" (1) or "not fraudlent"(0) and their predict probabilities with a scalable and robust ML model.

Solution Approach:

1. Exploratory data analysis
2. Prepare data (cleaning & transforrmation)
3. Feature Selection
4. Build, validate & compare ML Models
5. Stack the models
6. Tune the model
7. Finally, predict label probability of test data.

Assumptions:

1. Card category value "Debit or Credit" is treated as "Debit".
2. Attributes (Card1, Card2, Card3 and Card5) helps to construct unique card weight id.
3. One year data is available for analysis.
4. Concatenate Addr1 and Addr2 will provide unique address weight id of transaction.
5. D series attributes represent the Transaction date.

Objective of this Kernel:

Objective is to prepare and normalize data, then run, validate and compare various ML classifier using AUC (Area Under Curve) evaluation metric and performance time, stack the models using Stacknet, and tune the model for better classification prediction. Finally, submitted predict probabilities of the test set.

Credits

All credit goes to the original authors of StackNet and pyStackNet.

- https://github.com/h2oai/pystacknet (https://github.com/h2oai/pystacknet)
- https://github.com/kaz-Anova/StackNet (https://github.com/kaz-Anova/StackNet)

Import Packages

In [2]:
```
# Data Processing
import numpy as np # linear algebra
import pandas as pd # data processing
from numpy import sort

# Visulaization
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches
from pandas.plotting import scatter_matrix


# XGBoost
import xgboost as xgb

# lightGBM
from lightgbm import LGBMRegressor
from lightgbm import LGBMClassifier

# Catboost
from catboost import CatBoostRegressor
from catboost import CatBoostClassifier

# StackNet
import sys
sys.path.append("//Downloads/h2oai-pystacknet-af571e0")
from pystacknet.pystacknet import StackNetClassifier
# https://github.com/h2oai/pystacknet

# Scikit learn Classfiers
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingRegressor
,RandomForestRegressor,GradientBoostingClassifier
from xgboost.sklearn import XGBClassifier
#from sklearn.tree import DecisionTreeClassifier

# Other Scikit Learn
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.model_selection import KFold, StratifiedKFold
from scipy.stats import reciprocal, uniform
from scipy.stats import expon
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_
score, classification_report
from scipy.stats import randint, uniform
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import mean_squared_error

# Others
import time
import collections
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
import gc
```

```python
import shutil

# Reproducibility
seed = 342
np.random.seed(seed)

#import image
from IPython.display import Image
```

Custom Functions

In [2]:

```python
def df_stats(df,columns):
    """
    df info

    """

    stats = []
    for col in columns:
        stats.append((col, df[col].count(), df[col].nunique(), df[col].isnull
().sum(), df[col].isnull().sum() * 100 / df.shape[0], df[col].dtype))

    stats_cols = ['attribute', 'record_count', 'unique_values','missing_record
s', 'percent_missing_records', 'data_type']
    df_stats = pd.DataFrame(stats, columns=stats_cols)

    return(df_stats)
```

In [356]:

```python
def drop_cols(df,cols):
    """
    drop columns which have missing records > 90%

    """
    drop_cols = []
    n = 90
    for col in cols:
        rate = (df[col].isnull().sum() * 100 / df.shape[0])
        if rate > n:
            df = df.drop(col, axis=1)
            drop_cols.append(col)

    return(df,drop_cols)
```

In [4]:
```
def CalcOutliers(df_num):

    # calculating mean and std of the array
    data_mean, data_std = np.mean(df_num), np.std(df_num)

    # seting the cut line to both higher and lower values
    # You can change this value
    cut = data_std * 3

    #Calculating the higher and lower cut values
    lower, upper = data_mean - cut, data_mean + cut

    # creating an array of lower, higher and total outlier values
    outliers_lower = [x for x in df_num if x < lower]
    outliers_higher = [x for x in df_num if x > upper]
    outliers_total = [x for x in df_num if x < lower or x > upper]

    # array without outlier values
    outliers_removed = [x for x in df_num if x > lower and x < upper]

    print('Identified lowest outliers: %d' % len(outliers_lower)) # printing t
otal number of values in lower cut of outliers
    print('Identified upper outliers: %d' % len(outliers_higher)) # printing t
otal number of values in higher cut of outliers
    print('Total outlier observations: %d' % len(outliers_total)) # printing t
otal number of values outliers of both sides
    print('Non-outlier observations: %d' % len(outliers_removed)) # printing t
otal number of non outlier values
    print("Total percentual of Outliers: ", round((len(outliers_total) / len(o
utliers_removed) )*100, 4)) # Percentual of outliers in points

    return
```

In [5]:
```
def ploting_cnt_amt(df, col, lim=2000):
    total = len(df)
    total_amt = df.groupby(['isFraud'])['TransactionAmt'].sum().sum()
    tmp = pd.crosstab(df[col], df['isFraud'], normalize='index') * 100
    tmp = tmp.reset_index()
    tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)

    plt.figure(figsize=(16,14))
    plt.suptitle(f'{col} Distributions ', fontsize=24)

    plt.subplot(211)
    g = sns.countplot( x=col,  data=df, order=list(tmp[col].values))
    gt = g.twinx()
    gt = sns.pointplot(x=col, y='Fraud', data=tmp, order=list(tmp[col].values
),
                       color='black', legend=False, )
    gt.set_ylim(0,tmp['Fraud'].max()*1.1)
    gt.set_ylabel("%Fraud Transactions", fontsize=16)
    g.set_title(f"Most Frequent {col} values and % Fraud Transactions", fontsi
ze=20)
    g.set_xlabel(f"{col} Category Names", fontsize=16)
    g.set_ylabel("Count", fontsize=17)
    g.set_xticklabels(g.get_xticklabels(),rotation=45)
    sizes = []
    for p in g.patches:
        height = p.get_height()
        sizes.append(height)
        g.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(height/total*100),
                ha="center",fontsize=12)

    g.set_ylim(0,max(sizes)*1.15)

    #########################################################################
    perc_amt = (df.groupby(['isFraud',col])['TransactionAmt'].sum() \
                / df.groupby([col])['TransactionAmt'].sum() * 100).unstack('is
Fraud')
    perc_amt = perc_amt.reset_index()
    perc_amt.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
    amt = df.groupby([col])['TransactionAmt'].sum().reset_index()
    perc_amt = perc_amt.fillna(0)
    plt.subplot(212)
    g1 = sns.barplot(x=col, y='TransactionAmt',
                       data=amt,
                       order=list(tmp[col].values))
    g1t = g1.twinx()
    g1t = sns.pointplot(x=col, y='Fraud', data=perc_amt,
                        order=list(tmp[col].values),
                       color='black', legend=False, )
    g1t.set_ylim(0,perc_amt['Fraud'].max()*1.1)
    g1t.set_ylabel("%Fraud Total Amount", fontsize=16)
    g.set_xticklabels(g.get_xticklabels(),rotation=45)
    g1.set_title(f"{col} by Transactions Total + %of total and %Fraud Transact
ions", fontsize=20)
    g1.set_xlabel(f"{col} Category Names", fontsize=16)
```

```
    g1.set_ylabel("Transaction Total Amount(U$)", fontsize=16)
    g1.set_xticklabels(g.get_xticklabels(),rotation=45)

    for p in g1.patches:
        height = p.get_height()
        g1.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(height/total_amt*100),
                ha="center",fontsize=12)

    plt.subplots_adjust(hspace=.4, top = 0.9)
    plt.show()
```

In [6]:
```
def auc_score(y_true, y_pred):
    """
    Calculates the Area Under ROC Curve (AUC)
    """
    return roc_auc_score(y_true, y_pred)
```

In [7]:
```
def plot_curve(y_true_train, y_pred_train, y_true_val, y_pred_val, model_name
):
    """
    Plots the ROC Curve given predictions and labels
    """
    fpr_train, tpr_train, _ = roc_curve(y_true_train, y_pred_train, pos_label=
1)
    fpr_val, tpr_val, _ = roc_curve(y_true_val, y_pred_val, pos_label=1)
    plt.figure(figsize=(8, 8))
    plt.plot(fpr_train, tpr_train, color='black',
             lw=2, label=f"ROC train curve (AUC = {round(roc_auc_score(y_true_
train, y_pred_train), 4)})")
    plt.plot(fpr_val, tpr_val, color='darkorange',
             lw=2, label=f"ROC validation curve (AUC = {round(roc_auc_score(y_
true_val, y_pred_val), 4)})")
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([-0.05, 1.05])
    plt.ylim([-0.05, 1.05])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.title(f'ROC Plot for {model_name}', weight="bold", fontsize=20)
    plt.legend(loc="lower right", fontsize=16)
```

StackNet does not accept missing values (NaN's), Infinity values (inf) or values higher than 32 bytes (for example float64 or int64). Therefore, we have to fill in missing values and compress certain columns as the Pandas standard is 64 bytes. Big thanks to Arjan Groen for creating this convenient function. The function is taken from this Kaggle kernel.

In [8]:
```
def reduce_mem_usage(df):
    """
    Reduces memory usage for all columns in a Pandas DataFrame
    """
    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB")
    NAlist = [] # Keeps track of columns that have missing values filled in.
    for col in df.columns:
        if df[col].dtype != object:  # Exclude strings
            # make variables for Int, max and min
            IsInt = False
            mx = df[col].max()
            mn = df[col].min()
            # Integer does not support NA, therefore, NA needs to be filled
            if not np.isfinite(df[col]).all():
                NAlist.append(col)
                df[col].fillna(mn-1,inplace=True)

            # test if column can be converted to an integer
            asint = df[col].fillna(0).astype(np.int32)
            result = (df[col] - asint)
            result = result.sum()
            if result > -0.01 and result < 0.01:
                IsInt = True
            # Make Integer/unsigned Integer datatypes
            if IsInt:
                if mn >= 0:
                    if mx < 255:
                        df[col] = df[col].astype(np.uint8)
                    elif mx < 65535:
                        df[col] = df[col].astype(np.uint16)
                    else:
                        df[col] = df[col].astype(np.uint32)
                else:
                    if mn > np.iinfo(np.int8).min and mx < np.iinfo(np.int8).m
ax:
                        df[col] = df[col].astype(np.int8)
                    elif mn > np.iinfo(np.int16).min and mx < np.iinfo(np.int1
6).max:
                        df[col] = df[col].astype(np.int16)
                    else:
                        df[col] = df[col].astype(np.int32)
            # Make float datatypes 32 bit
            else:
                df[col] = df[col].astype(np.float32)

    # Print final result
    mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is after reduction is:",mem_us
g," MB")
    return df, NAlist
```

In [161]:
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_styl
e="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regress
ors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

Get Data

In [327]:
```
t1 = time.time()
folder_path = 'SourceData/'

# parse all files
train_identity = pd.read_csv(f'{folder_path}train_identity.csv')
test_identity = pd.read_csv(f'{folder_path}test_identity.csv')
train_transaction = pd.read_csv(f'{folder_path}train_transaction.csv')
test_transaction = pd.read_csv(f'{folder_path}test_transaction.csv')
sub_set = pd.read_csv(f'{folder_path}sample_submission.csv')
t2 = time.time()

print("Time to parse(Seconds): %f" % (t2 - t1))
print('-----------------------------------')

# Lets Join
train_set = pd.merge(train_transaction, train_identity, on='TransactionID', ho
w='left')
test_set = pd.merge(test_transaction, test_identity, on='TransactionID', how=
'left')
t3 = time.time()

print("Time to join(Seconds): %f" % (t3 - t2))
print('-----------------------------------')


print('Train set :', train_set.shape)
print('Test set :',test_set.shape)
print('Submission set :',sub_set.shape)
```

```
Time to parse(Seconds): 143.457898
-----------------------------------
Time to join(Seconds): 51.788023
-----------------------------------
Train set : (590540, 434)
Test set : (506691, 433)
Submission set : (506691, 2)
```

Save Memory Usage

In [328]:
```
start_time = time.time()

train_df,tr=reduce_mem_usage(train_set)
test_df,te=reduce_mem_usage(test_set)
sub_df  = sub_set.copy()

print("Time taken:--- %s Seconds ---" % (time.time() - start_time))
```

```
Memory usage of properties dataframe is : 1959.8762512207031  MB
Memory usage of properties dataframe is after reduction is: 550.7928085327148
MB
Memory usage of properties dataframe is : 1677.7335662841797  MB
Memory usage of properties dataframe is after reduction is: 462.9230289459228
5  MB
Time taken:--- 507.39336347579956 Seconds ---
```

In [329]:
```
del train_identity
del test_identity
del train_transaction
del test_transaction
del sub_set
del train_set
del test_set
```

In [331]:
```
gc.collect()
```

Out[331]: 100

Data Analysis

Data Dictionary

Description	Attribute
Transaction id	TransactionID
Label	isFraud
Transaction date	TransactionDT
Transaction amount	TransactionAmt
Product Category	ProductCD
Card details	card1, card2, card3, card4, card5, card6
Address(region, country)	addr1, addr2
Distance	dist1, dist2
Purchaser & Receiver Email domains	P_emaildomain, R_emaildomain
Card details	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14
Date attributes	D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15
Match records	M1, M2, M3, M4, M5, M6, M7, M8, M9
Vesta attributes	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, V29, V30, V31, V32, V33, V34, V35, V36, V37, V38, V39, V40, V41, V42, V43, V44, V45, V46, V47, V48, V49, V50, V51, V52, V53, V54, V55, V56, V57, V58, V59, V60, V61, V62, V63, V64, V65, V66, V67, V68, V69, V70, V71, V72, V73, V74, V75, V76, V77, V78, V79, V80, V81, V82, V83, V84, V85, V86, V87, V88, V89, V90, V91, V92, V93, V94, V95

In [332]:

```
# Categorical Attributes
print([c for c in train_df.columns if train_df[c].dtype == 'object'])
```

```
['ProductCD', 'card4', 'card6', 'P_emaildomain', 'R_emaildomain', 'M1', 'M2',
'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9', 'id_12', 'id_15', 'id_16', 'id_23',
'id_27', 'id_28', 'id_29', 'id_30', 'id_31', 'id_33', 'id_34', 'id_35', 'id_3
6', 'id_37', 'id_38', 'DeviceType', 'DeviceInfo']
```

In [333]:

```
# Number of Numerical Attributes
print(len([c for c in train_df.columns if train_df[c].dtype != 'object']))
```

```
403
```

- Data and Data Attribute names are masked by vista for security reasons.
- Some attributes like card details, even though categorical by nature, are provided as numerical.

Let's do some data analysis.

Data Preview

In [334]:
```
# Train set
train_df.head(3)
```

Out[334]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	
0	2987000	0	86400	68.5	W	13926	99	150	d
1	2987001	0	86401	29.0	W	2755	404	150	mas
2	2987002	0	86469	59.0	W	4663	490	150	

3 rows × 434 columns

In [335]:
```
# Train set
test_df.head(3)
```

Out[335]:

	TransactionID	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5
0	3663549	18403224	31.950001	W	10409	111	150	visa	226
1	3663550	18403263	49.000000	W	4272	111	150	visa	226
2	3663551	18403310	171.000000	W	4476	574	150	visa	226

3 rows × 433 columns

Difference Columns Attributes - Train Vs Test

In [353]:
```
sub_cols  = list(sub_df.columns.values)
train_cols = list(train_df.columns.values)
test_cols = list(test_df.columns.values)
diff_cols = list(set(train_cols) - set(test_cols))
diff_cols
```

Out[353]: ['isFraud']

- 'isFraud' exists in Train set. But, not in Test set.

Null Records Analysis

In [337]:
```
# dataframe descritpion.
train_stats = df_stats(train_df,train_cols)
test_stats = df_stats(test_df,test_cols)
sub_stats = df_stats(sub_df,sub_cols)
```

In [338]:
```
Q = (train_stats['percent_missing_records']>80)

tmp_df = train_stats[Q].sort_values('percent_missing_records', ascending=False
).head(10)
```

In [339]:
```
tmp_df
```

Out[339]:

	attribute	record_count	unique_values	missing_records	percent_missing_records	data_typ
416	id_23	5169	3	585371	99.124699	objec
420	id_27	5169	2	585371	99.124699	objec
426	id_33	73289	260	517251	87.589494	objec
423	id_30	77565	75	512975	86.865411	objec
427	id_34	77805	4	512735	86.824771	objec

In [340]:
```
test_stats.sort_values('percent_missing_records', ascending=False).head(5)
```

Out[340]:

	attribute	record_count	unique_values	missing_records	percent_missing_records	data_typ
419	id_27	5062	2	501629	99.000969	objec
415	id_23	5062	3	501629	99.000969	objec
422	id_30	70659	86	436032	86.054814	objec
425	id_33	70671	390	436020	86.052446	objec
426	id_34	72175	2	434516	85.755618	objec

In [341]:
```
sub_stats.sort_values('percent_missing_records', ascending=False)
```

Out[341]:

	attribute	record_count	unique_values	missing_records	percent_missing_records	data_ty
0	TransactionID	506691	506691	0	0.0	in
1	isFraud	506691	1	0	0.0	floa

- Most of column attributes, especially [53:], have highest number of missing records.

Visuals

Card Type

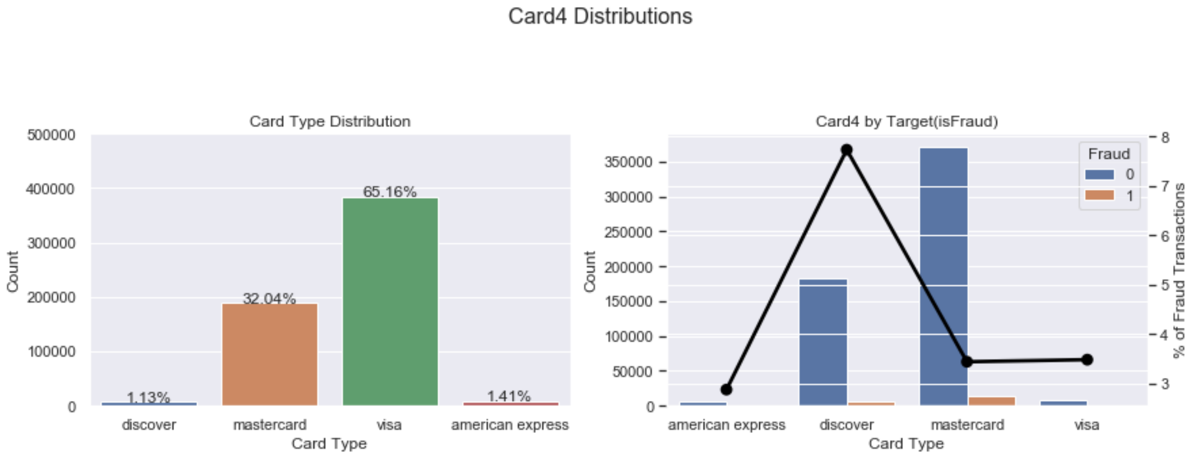
In [342]:

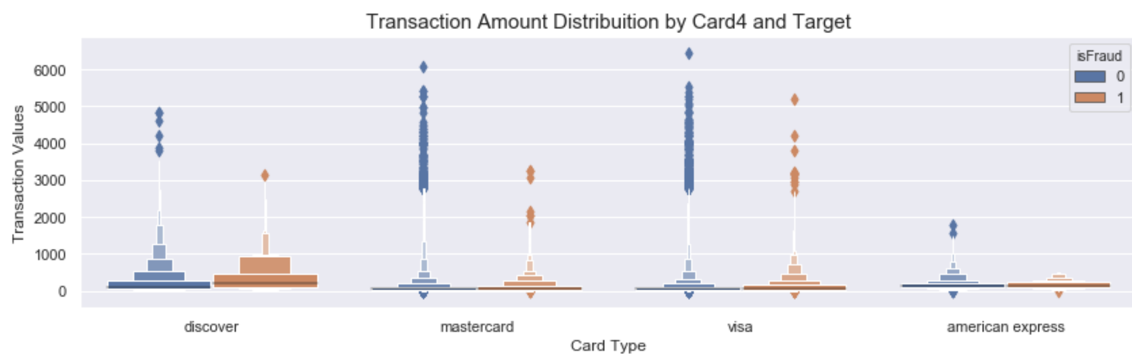
```
tmp = pd.crosstab(train_df['card4'], train_df['isFraud'], normalize='index') *
100
tmp = tmp.reset_index()
tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
tmp
```

Out[342]:

isFraud	card4	NoFraud	Fraud
0	american express	97.130163	2.869837
1	discover	92.271839	7.728161
2	mastercard	96.566905	3.433095
3	visa	96.524390	3.475610

In [343]:
```
# Visualize

total = len(train_df)

plt.figure(figsize=(14,10))
plt.suptitle('Card4 Distributions', fontsize=16)

# Subplot 1 - Count vs card4

plt.subplot(221)
g = sns.countplot(x='card4', data=train_df)

g.set_title("Card Type Distribution", fontsize=12)
g.set_xlabel("Card Type", fontsize=12)
g.set_ylabel("Count", fontsize=12)
g.set_ylim(0,500000)

## Percentage calculation
for p in g.patches:
    height = p.get_height()
    g.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total*100),
            ha="center", fontsize=12)

# Subplot 2 -

unique_values = tmp.card4.unique()

plt.subplot(222)
g1 = sns.countplot(x='card4', hue='isFraud', data=train_df)
plt.legend(title='Fraud', loc='best', labels=[0,1])
gt = g1.twinx()
gt = sns.pointplot(x='card4', y='Fraud', data=tmp, color='black', order=unique
_values, legend=False)
gt.set_ylabel("% of Fraud Transactions", fontsize=12)

g1.set_title("Card4 by Target(isFraud)", fontsize=12)
g1.set_xlabel("Card Type", fontsize=12)
g1.set_ylabel("Count", fontsize=12)


# Subplot 3 -

plt.subplot(212)
g3 = sns.boxenplot(x='card4', y='TransactionAmt', hue='isFraud',
              data=train_df[train_df['TransactionAmt'] <= 10000] )
g3.set_title("Transaction Amount Distribiution by Card4 and Target", fontsize=
15)
g3.set_xlabel("Card Type", fontsize=12)
g3.set_ylabel("Transaction Values", fontsize=12)

plt.subplots_adjust(hspace = 0.6, top = 0.85)

plt.show()
```





'card4' categorical attribute shares information on the type of the card used in the transaction. Noticed above that Visa (65.16%) has more transactions followed by Master card (32.04%). Discover card is used for highest % of fraud transactions in count, where as their values are small amounts. In case of Visa, even though % of fraud transactions are low. But, fraud transaction amount is high.

In [344]:
```
del tmp
del train_stats
del test_stats
del sub_stats
```

In [345]:
```
gc.collect()
```

Out[345]:
```
22914
```

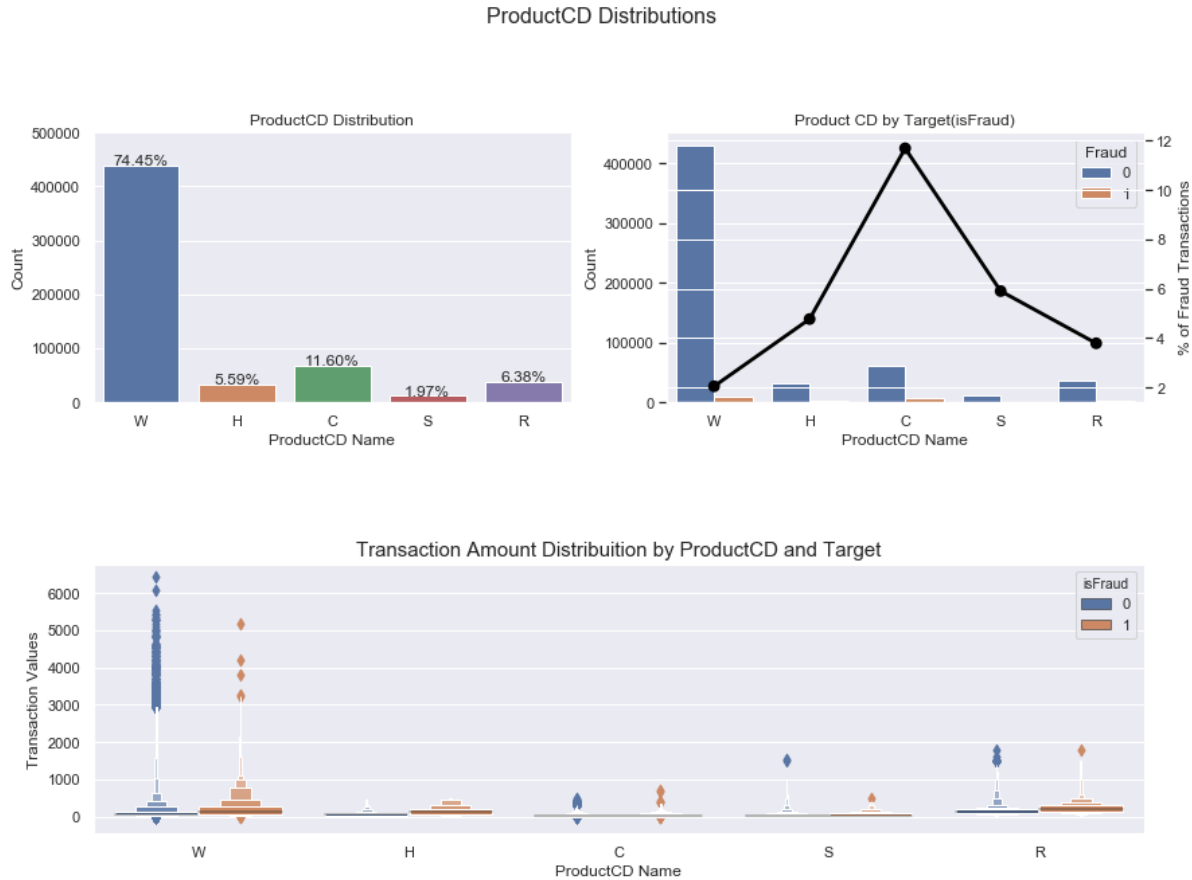
ProductCD: Product Category

In [346]:
```
tmp = pd.crosstab(train_df['ProductCD'], train_df['isFraud'], normalize='inde
x') * 100
tmp = tmp.reset_index()
tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
tmp.sort_values('Fraud',ascending=False)
```

Out[346]:

isFraud	ProductCD	NoFraud	Fraud
0	C	88.312731	11.687269
3	S	94.100447	5.899553
1	H	95.233769	4.766231
2	R	96.217406	3.782594
4	W	97.960061	2.039939

In [347]:
```
# Visualize

total = len(train_df)

plt.figure(figsize=(14,10))
plt.suptitle('ProductCD Distributions', fontsize=16)

# Subplot 1 - Count vs ProductCD

plt.subplot(221)
g = sns.countplot(x='ProductCD', data=train_df)

g.set_title("ProductCD Distribution", fontsize=12)
g.set_xlabel("ProductCD Name", fontsize=12)
g.set_ylabel("Count", fontsize=12)
g.set_ylim(0,500000)

## Percentage calculation
for p in g.patches:
    height = p.get_height()
    g.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total*100),
            ha="center", fontsize=12)

# Subplot 2 -

plt.subplot(222)
g1 = sns.countplot(x='ProductCD', hue='isFraud', data=train_df)
plt.legend(title='Fraud', loc='best', labels=[0,1])
gt = g1.twinx()
gt = sns.pointplot(x='ProductCD', y='Fraud', data=tmp, color='black', order=[
'W', 'H',"C", "S", "R"], legend=False)
gt.set_ylabel("% of Fraud Transactions", fontsize=12)

g1.set_title("Product CD by Target(isFraud)", fontsize=12)
g1.set_xlabel("ProductCD Name", fontsize=12)
g1.set_ylabel("Count", fontsize=12)


# Subplot 3 -

plt.subplot(212)
g3 = sns.boxenplot(x='ProductCD', y='TransactionAmt', hue='isFraud',
              data=train_df[train_df['TransactionAmt'] <= 10000] )
g3.set_title("Transaction Amount Distribuition by ProductCD and Target", fonts
ize=15)
g3.set_xlabel("ProductCD Name", fontsize=12)
g3.set_ylabel("Transaction Values", fontsize=12)

plt.subplots_adjust(hspace = 0.6, top = 0.85)

plt.show()
```



W, C and R are the most frequent values and % of fraud transactions are high in product category 'C' In case of fraud transaction value, noticed that product catageories - W, H and R - are slightly higher.

In [348]:
```
del tmp
```

In [349]:
```
gc.collect()
```

Out[349]:
```
9761
```

Prepare Data

Drop duplicate TransactionID records

In [350]:
```
# As Transaction Id is the primary key. The last record of the duplicate insta
nce is retained. Rest instances are dropped.

train_df = train_df.drop_duplicates(['TransactionID'], keep ='last')
test_df = test_df.drop_duplicates(['TransactionID'], keep ='last')
sub_df = sub_df.drop_duplicates(['TransactionID'], keep ='last')
print("Train TransactionID count",train_df['TransactionID'].count())
print("Test TransactionID count",test_df['TransactionID'].count())
print("Submission TransactionID count",sub_df['TransactionID'].count())
```

```
Train TransactionID count 590540
Test TransactionID count 506691
Submission TransactionID count 506691
```

First Drop: Drop Columns (Nulls > 90%)

In [357]:
```
# drop the attributes which have more than 90% null values.
train_df, train_drop_cols = drop_cols(train_df,train_cols)
print('dropped columns',train_drop_cols )
```

```
dropped columns ['id_23', 'id_27']
```

In [358]:
```
test_df, test_drop_cols = drop_cols(test_df,test_cols)
print('dropped columns',test_drop_cols )
```

```
dropped columns ['id_23', 'id_27']
```

In [359]:
```
# Attributes dropped in Train. But, still Exists in Test. Hence dropped for re
blancing the datasets.
diff_cols_todrop = list(set(train_drop_cols)-set(test_drop_cols))
print(diff_cols_todrop)
```

```
[]
```

In [360]:
```
# Test - instance & attributes
test_df.shape
```

Out[360]:
```
(506691, 431)
```

In [361]:
```
# Train - instance & attributes
train_df.shape
```

Out[361]:
```
(590540, 432)
```

Imputation

Let's fill the null values with -999 for both categorical as well as numerical attributes.

In [362]:
```
train_trans = train_df.fillna(-999)
test_trans = test_df.fillna(-999)
```

In [363]:
```
del train_df
del test_df
```

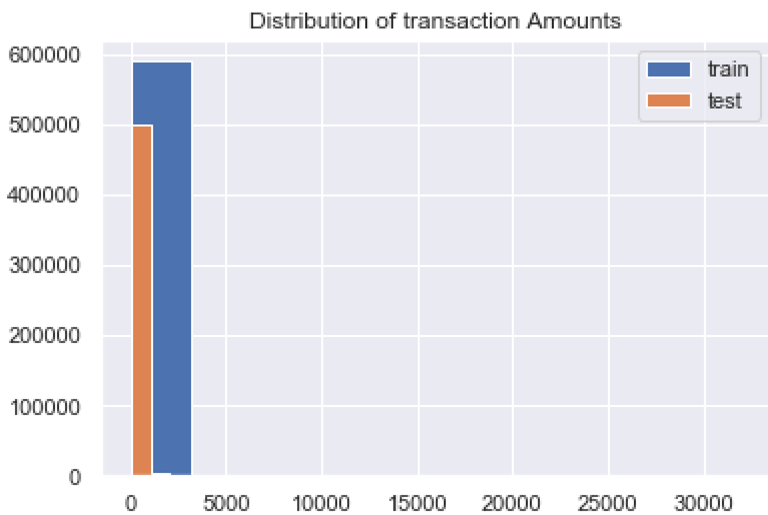
In [364]:
```
gc.collect()
```

Out[364]: 184

More visuals

TransactionAmt

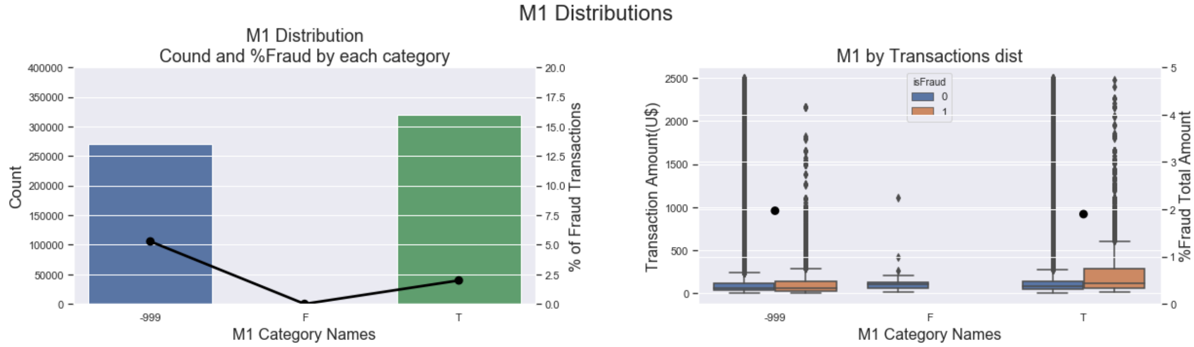
In [381]:
```
plt.hist(train_trans['TransactionAmt'], label='train');
plt.hist(test_trans['TransactionAmt'], label='test');
plt.legend();
plt.title('Distribution of transaction Amounts');
```

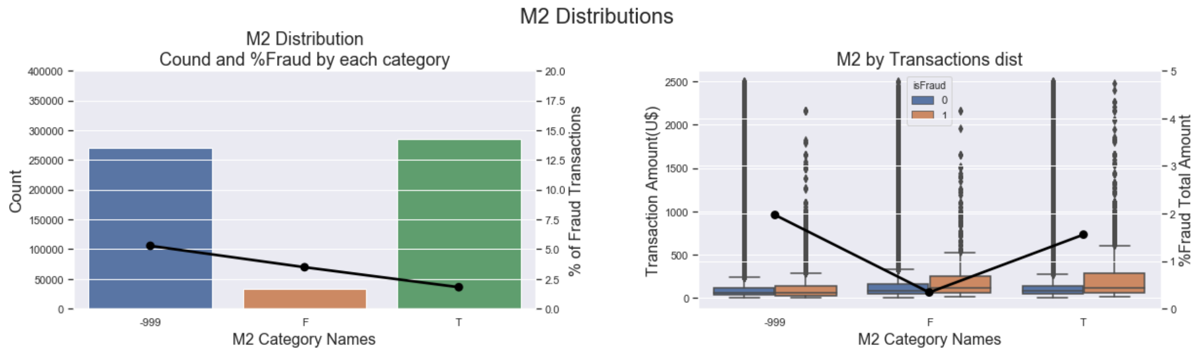


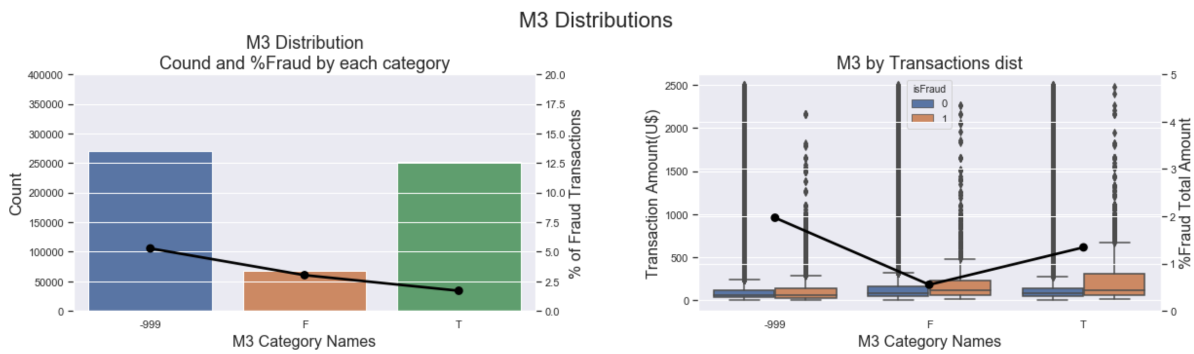
M1-M6 : Matched Card details

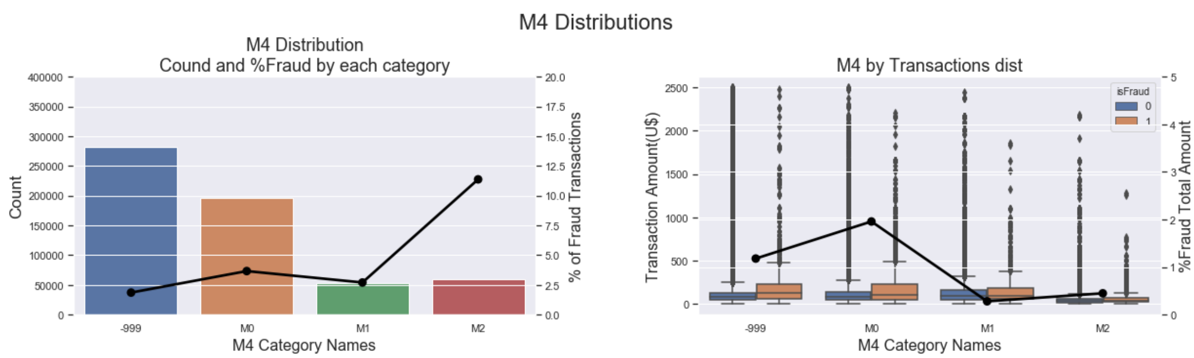
In [365]:
```
def ploting_dist_ratio(df, col, lim=2000):
    tmp = pd.crosstab(df[col], df['isFraud'], normalize='index') * 100
    tmp = tmp.reset_index()
    tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)

    plt.figure(figsize=(20,5))
    plt.suptitle(f'{col} Distributions ', fontsize=22)

    # Plot 1
    plt.subplot(121)
    g = sns.countplot(x=col, data=df, order=list(tmp[col].values))

    g.set_title(f"{col} Distribution\nCound and %Fraud by each category", font
size=18)
    g.set_ylim(0,400000)
    gt = g.twinx()
    gt = sns.pointplot(x=col, y='Fraud', data=tmp, order=list(tmp[col].values
),
                       color='black', legend=False, )
    gt.set_ylim(0,20)
    gt.set_ylabel("% of Fraud Transactions", fontsize=16)
    g.set_xlabel(f"{col} Category Names", fontsize=16)
    g.set_ylabel("Count", fontsize=17)
    for p in gt.patches:
        height = p.get_height()
        gt.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(height/total*100),
                ha="center",fontsize=14)

    total_amt = df.groupby(['isFraud'])['TransactionAmt'].sum().sum()
    perc_amt = (df.groupby(['isFraud',col])['TransactionAmt'].sum() / total_am
t * 100).unstack('isFraud')
    perc_amt = perc_amt.reset_index()
    perc_amt.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)

    plt.subplot(122)
    g1 = sns.boxplot(x=col, y='TransactionAmt', hue='isFraud',
                     data=df[df['TransactionAmt'] <= lim], order=list(tmp[col]
.values))
    g1t = g1.twinx()
    g1t = sns.pointplot(x=col, y='Fraud', data=perc_amt, order=list(tmp[col].v
alues),
                       color='black', legend=False, )
    g1t.set_ylim(0,5)
    g1t.set_ylabel("%Fraud Total Amount", fontsize=16)
    g1.set_title(f"{col} by Transactions dist", fontsize=18)
    g1.set_xlabel(f"{col} Category Names", fontsize=16)
    g1.set_ylabel("Transaction Amount(U$)", fontsize=16)

    plt.subplots_adjust(hspace=.4, wspace = 0.35, top = 0.80)

    plt.show()
```

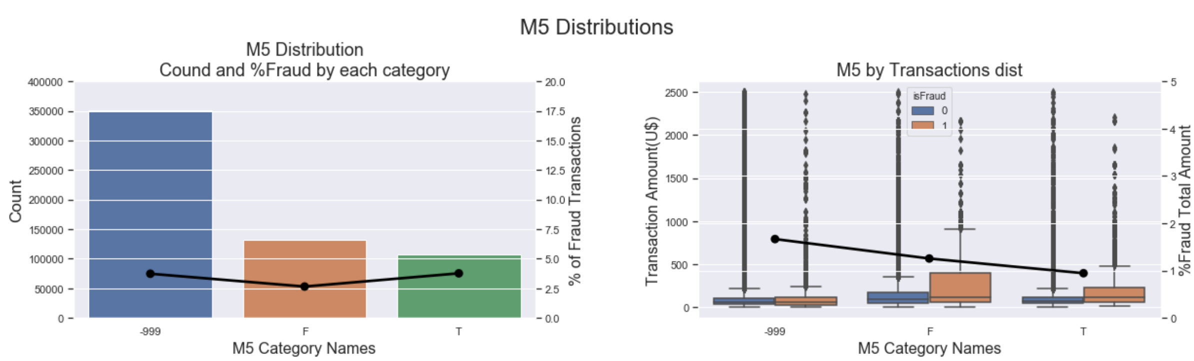
In [366]:
```
m_list = train_trans.columns[train_trans.columns.str.startswith('M')]

for col in m_list:
    ploting_dist_ratio(train_trans, col, lim=2500)
```

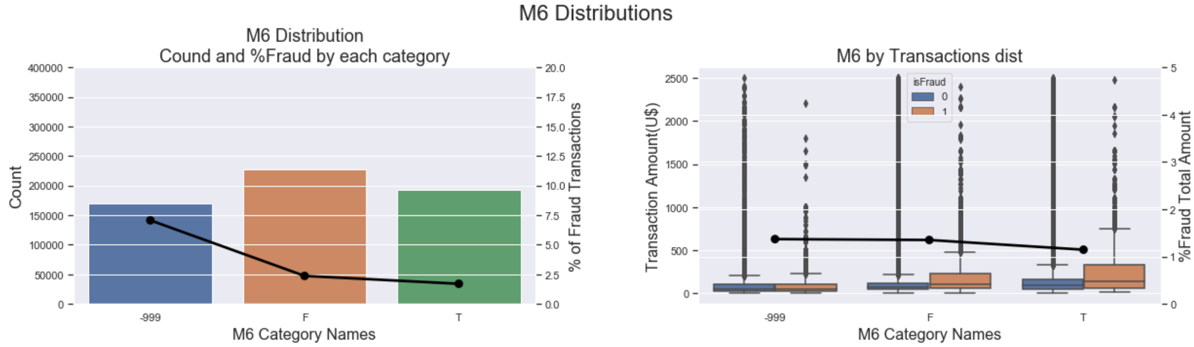

M1 Distributions
M1 Distribution
Cound and %Fraud by each category
Count
% of Fraud Transactions
M1 Category Names
-999
F
T
M1 by Transactions dist
Transaction Amount(U$)
%Fraud Total Amount
isFraud
0
1
M1 Category Names

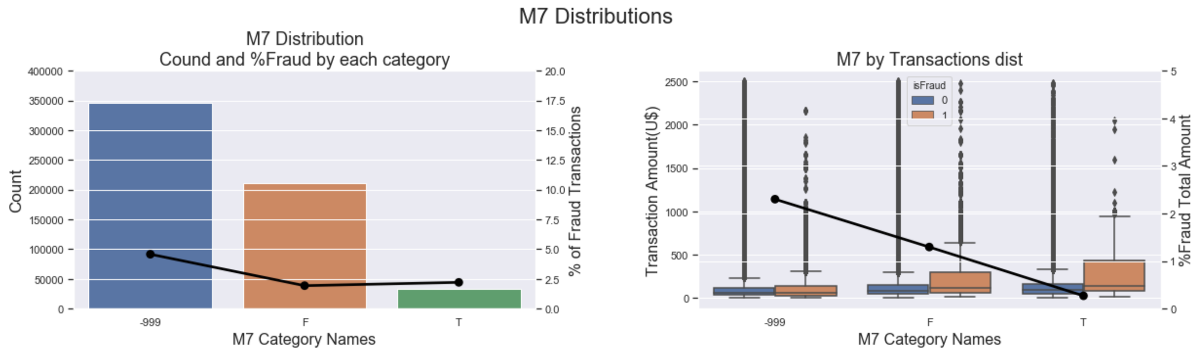

M2 Distributions
M2 Distribution
Cound and %Fraud by each category
Count
% of Fraud Transactions
M2 Category Names
-999
F
T
M2 by Transactions dist
Transaction Amount(U$)
%Fraud Total Amount
isFraud
0
1
M2 Category Names

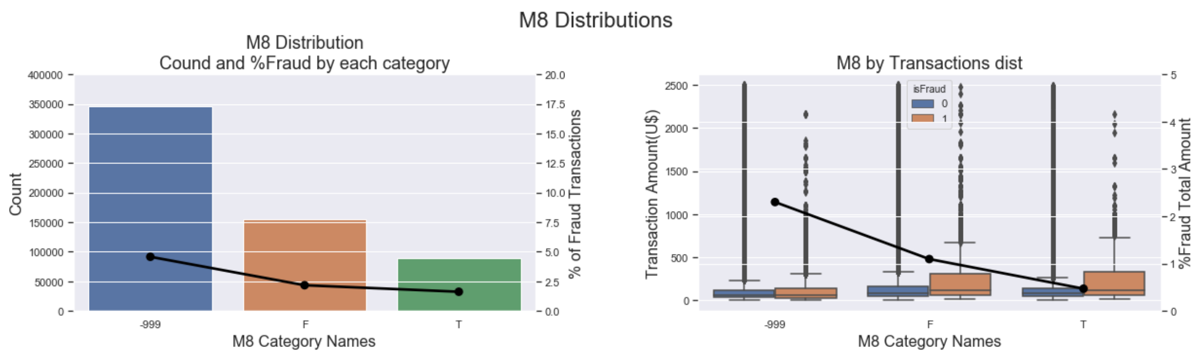

M3 Distributions
M3 Distribution
Cound and %Fraud by each category
Count
% of Fraud Transactions
M3 Category Names
-999
F
T
M3 by Transactions dist
Transaction Amount(U$)
%Fraud Total Amount
isFraud
0
1
M3 Category Names

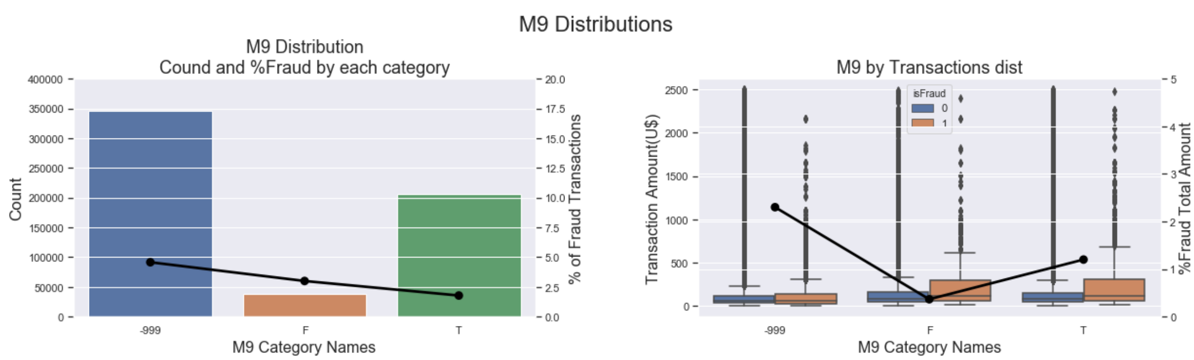

M4 Distributions
M4 Distribution
Cound and %Fraud by each category
Count
% of Fraud Transactions
M4 Category Names
-999
M0
M1
M2
M4 by Transactions dist
Transaction Amount(U$)
%Fraud Total Amount
isFraud
0
1
M4 Category Names


M5 Distributions
M5 Distribution
Cound and %Fraud by each category
Count
% of Fraud Transactions
M5 Category Names
-999
F
T
M5 by Transactions dist
Transaction Amount(U$)
%Fraud Total Amount
isFraud
0
1
M5 Category Names









'-999' means missining values. Its highest in M9, M8, M7. M4 has highest % of fraud transactions.

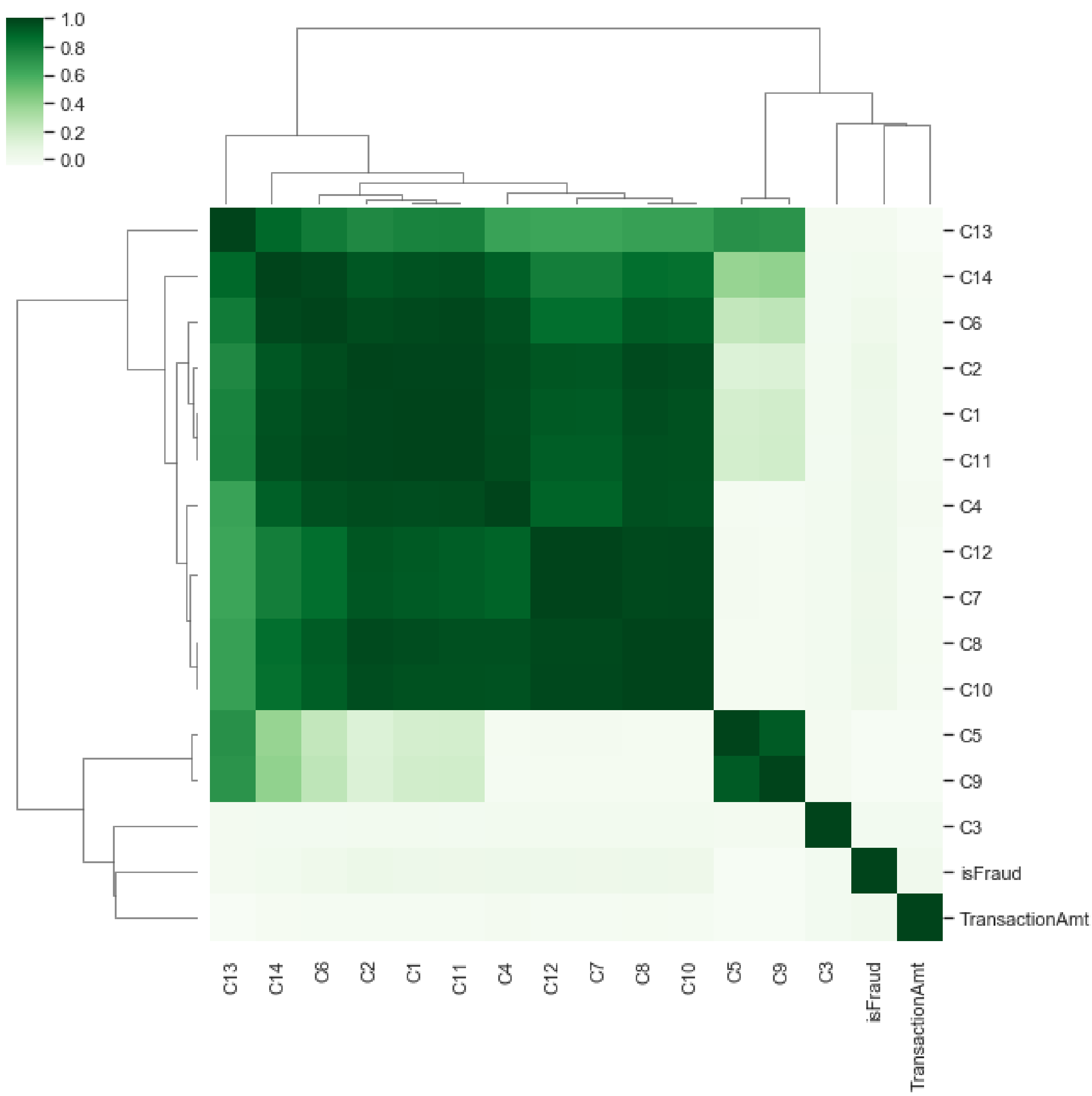
C1 to C14 : Addresses associated with payment card

In [367]:
```
# counting, such as how many addresses are found to be associated with the pay
ment card, etc. Its a masked parameters.
c_list = (list(train_trans.columns[train_trans.columns.str.startswith('C')]))
+ ['isFraud','TransactionAmt']
print(c_list)
```

```
['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12',
'C13', 'C14', 'isFraud', 'TransactionAmt']
```

In [368]:
```
c_df = train_trans.loc[:,c_list]
corr_matrix = c_df.corr()
```

In [369]:
```
# Correlation Matrix - Cluster mapping

k = 16
cols = corr_matrix.nlargest(k , 'isFraud')['isFraud'].index
cm = np.corrcoef(c_df[cols].values.T)
sns.set(font_scale = 1.00)
hm = sns.clustermap(cm , cmap = "Greens",cbar = True,square = True,
                 yticklabels = cols.values, xticklabels = cols.values)
```



C attributes has not much correlation neither with target value "isFraud" nor with Transaction Amount. I think, we can drop this noise data.

In [370]:
```
c_list = list(train_trans.columns[train_trans.columns.str.startswith('C')])
print(c_list)
```

```
['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12',
'C13', 'C14']
```

In [371]:
```
# Train set
train_trans = train_trans.drop(train_trans[c_list], axis=1)
```

In [372]:
```
# Test set
test_trans = test_trans.drop(test_trans[c_list], axis=1)
```

In [373]:
```
train_trans1 = train_trans.copy()
```

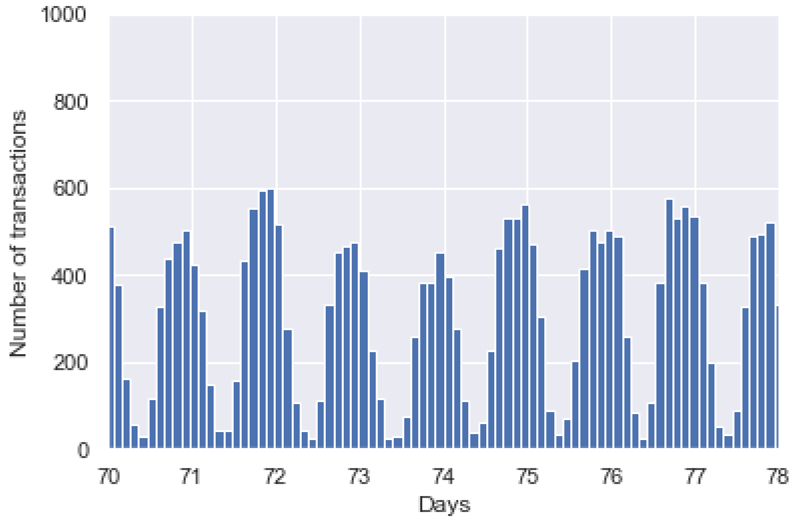
In [374]:
```
test_trans1 = test_trans.copy()
```

D series : Date related attributes

In [446]:
```
# credit to :  https://www.kaggle.com/fchmiel/day-and-time-powerful-predictive
-feature
def make_hour_feature(df, tname='TransactionDT'):
    """
    Creates an hour of the day feature, encoded as 0-23.

    Parameters:
    -----------
    df : pd.DataFrame
        df to manipulate.
    tname : str
        Name of the time column in df.
    """
    hours = df[tname] / (3600)
    encoded_hours = np.floor(hours) % 24
    return encoded_hours
```

In [447]:
```
vals = plt.hist(train_trans['TransactionDT'] / (3600*24), bins=1800)
plt.xlim(70, 78)
plt.xlabel('Days')
plt.ylabel('Number of transactions')
plt.ylim(0,1000)
```

Out[447]:
```
(0, 1000)
```

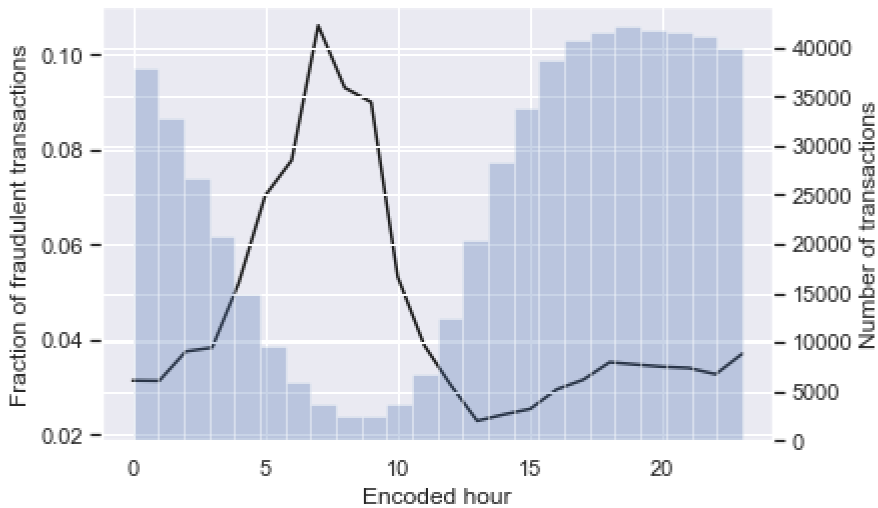


In [448]:
```
train_trans['hours'] = make_hour_feature(train_trans)
```

In [449]:
```
test_trans['hours'] = make_hour_feature(test_trans)
```

In [451]:
```
plt.plot(train_trans.groupby('hours').mean()['isFraud'], color='k')
ax = plt.gca()
ax2 = ax.twinx()
_ = ax2.hist(train_trans['hours'], alpha=0.3, bins=24)
ax.set_xlabel('Encoded hour')
ax.set_ylabel('Fraction of fraudulent transactions')

ax2.set_ylabel('Number of transactions')
```

Out[451]:
```
Text(0, 0.5, 'Number of transactions')
```



In [382]:
```
d_list = list(train_trans.columns[train_trans.columns.str.startswith('D')]) +
['isFraud','TransactionAmt','dist1','TransactionDT']
print(d_list)
```

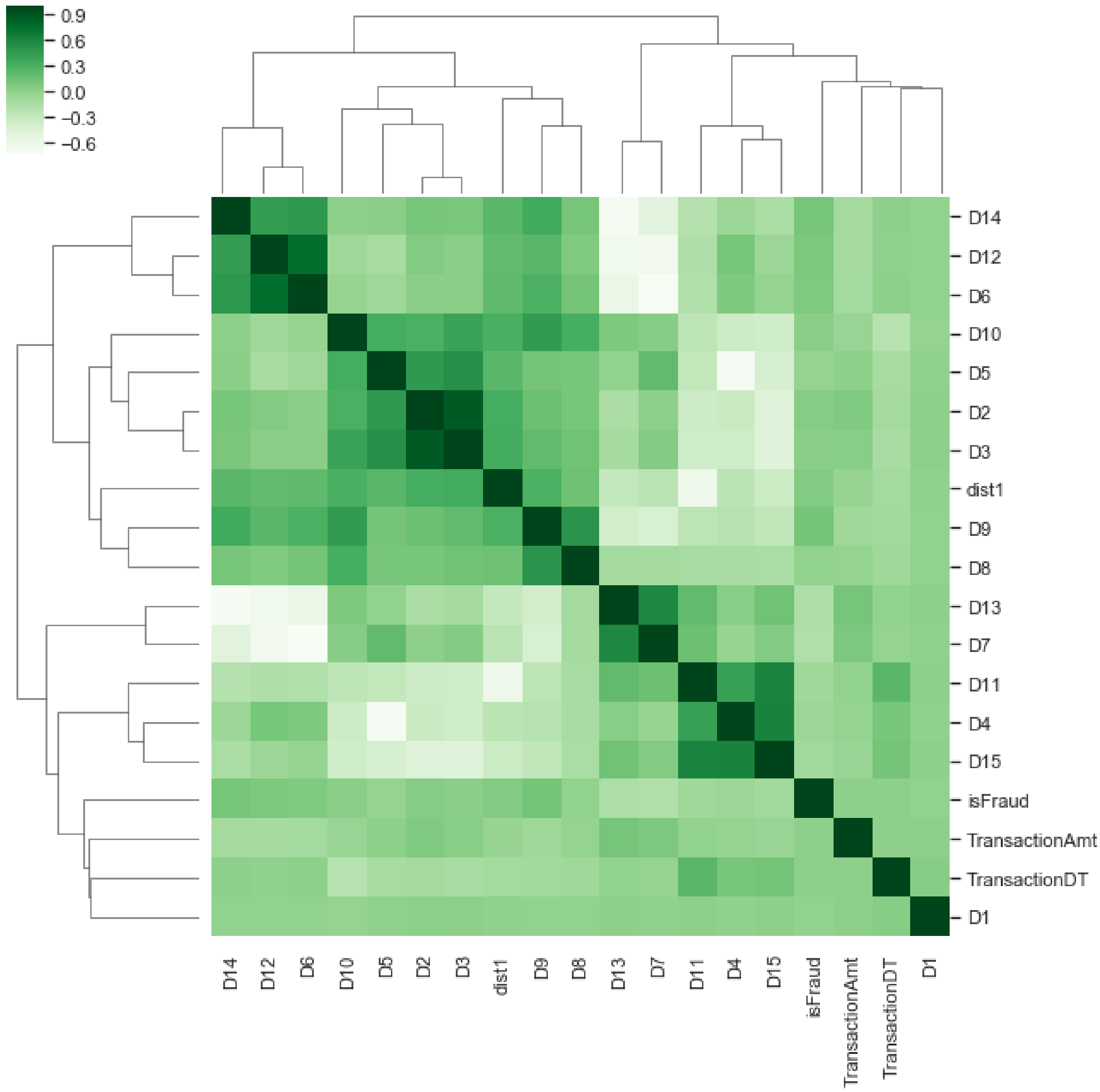
```
['D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'D10', 'D11', 'D12',
'D13', 'D14', 'D15', 'DeviceType', 'DeviceInfo', 'isFraud', 'TransactionAmt',
'dist1', 'TransactionDT']
```

In [383]:
```
d_df = train_trans.loc[:,d_list]
corr_matrix = d_df.corr()
```

In [384]:
```
corr_matrix['isFraud'].sort_values(ascending=False).head(10)
```

Out[384]:
```
isFraud          1.000000
D9               0.135268
D14              0.128808
D12              0.102287
D6               0.078301
dist1            0.067360
D2               0.054562
D3               0.034183
D10              0.033228
TransactionDT    0.013103
Name: isFraud, dtype: float64
```

In [386]:
```
# Correlation Matrix - Cluster mapping

k = 20
cols = corr_matrix.nlargest(k , 'isFraud')['isFraud'].index
cm = np.corrcoef(d_df[cols].values.T)
sns.set(font_scale = 1.00)
hm = sns.clustermap(cm , cmap = "Greens",cbar = True,square = True, yticklabel
s = cols.values, xticklabels = cols.values)
```



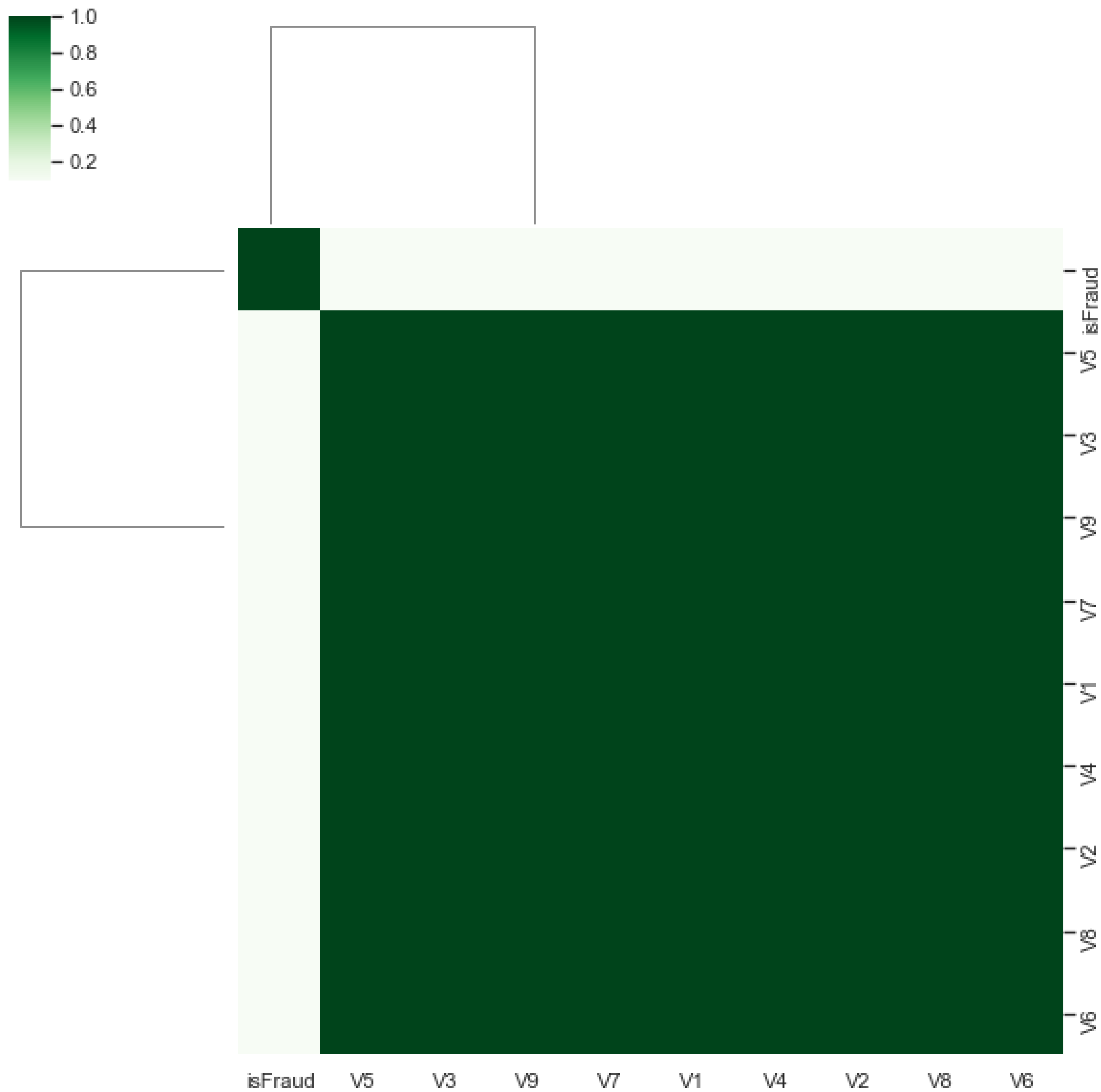
I assume, D attributes are related to date and time of transaction, except 'DeviceType', 'DeviceInfo' and 'dist1'. D14,D6,D12,D2,D3,D10 ,D8 and D9 is highly correlated to the target variable

Vesta Attributes

In [387]:
```
# Vesta engineered rich features, including ranking, counting, and other entit
y relations.
v_list = list(train_trans.columns[train_trans.columns.str.startswith('V')]) +
['isFraud','TransactionAmt']
```

In [388]:
```
v_df = train_trans.loc[:,v_list]
corr_matrix = v_df.corr()
```

In [389]:
```
# Correlation Matrix - Cluster mapping

k = 10
cols = corr_matrix.nlargest(k , 'isFraud')['isFraud'].index
cm = np.corrcoef(v_df[cols].values.T)
sns.set(font_scale = 1.00)
hm = sns.clustermap(cm , cmap = "Greens",cbar = True,square = True,
                 yticklabels = cols.values, xticklabels = cols.values)
```



V (vesta) attributes has no correlation with target as well as with transaction amount. I think, we drop these variables.

In [390]:
```
v_list = list(train_trans.columns[train_trans.columns.str.startswith('V')])
print(len(v_list))
```

```
339
```

In [391]:
```
# Train set
train_trans = train_trans.drop(train_trans[v_list], axis=1)
```

In [392]:
```
# Test set
test_trans = test_trans.drop(test_trans[v_list], axis=1)
```

In [393]:
```
gc.collect()
```

Out[393]:
```
49222
```

Binning

R_emaildomain: Receiver email domain

In [394]:
```
def email_bin(range_value):
    """
    Group mail domains
    
    """
    gmail = ['gmail.com', 'gmail']
    yahoo = ['yahoo.com', 'yahoo.com.mx',  'yahoo.co.uk','yahoo.co.jp', 'yahoo.de', 'yahoo.fr','yahoo.es']
    microsoft = ['hotmail.com','outlook.com','msn.com', 'live.com.mx', 'hotmail.es','hotmail.co.uk', 'hotmail.de', 'outlook.es', 'live.com', 'live.fr', 'hotmail.fr']
    #others = ['mail.com', 'anonymous.com', 'verizon.net', 'aol.com', 'me.com', 'comcast.net', 'optonline.net', 'cox.net', 'charter.net', 'rocketmail.com', 'prodigy.net.mx', 'embarqmail.com', 'icloud.com', 'att.net', 'juno.com', 'ymail.com', 'sbcglobal.net', 'bellsouth.net', 'q.com', 'centurylink.net', 'servicios-ta.com', 'earthlink.net', 'cfl.rr.com', 'roadrunner.com', 'netzero.net', 'gmx.de', 'suddenlink.net', 'frontiernet.net', 'windstream.net', 'frontier.com',  'mac.com', 'netzero.com', 'aim.com', 'web.de', 'twc.com', 'cableone.net','sc.rr.com', 'ptd.net', 'protonmail.com']
    
    
    if range_value in gmail:
        return "gmail"
    elif range_value in yahoo:
        return "yahoo"
    elif range_value in microsoft:
        return "msn"
    elif range_value == -999:
        return "empty"
    else:
        return "Others"
```

In [395]:
```
# Unique values
train_trans['R_emaildomain'].nunique()
```

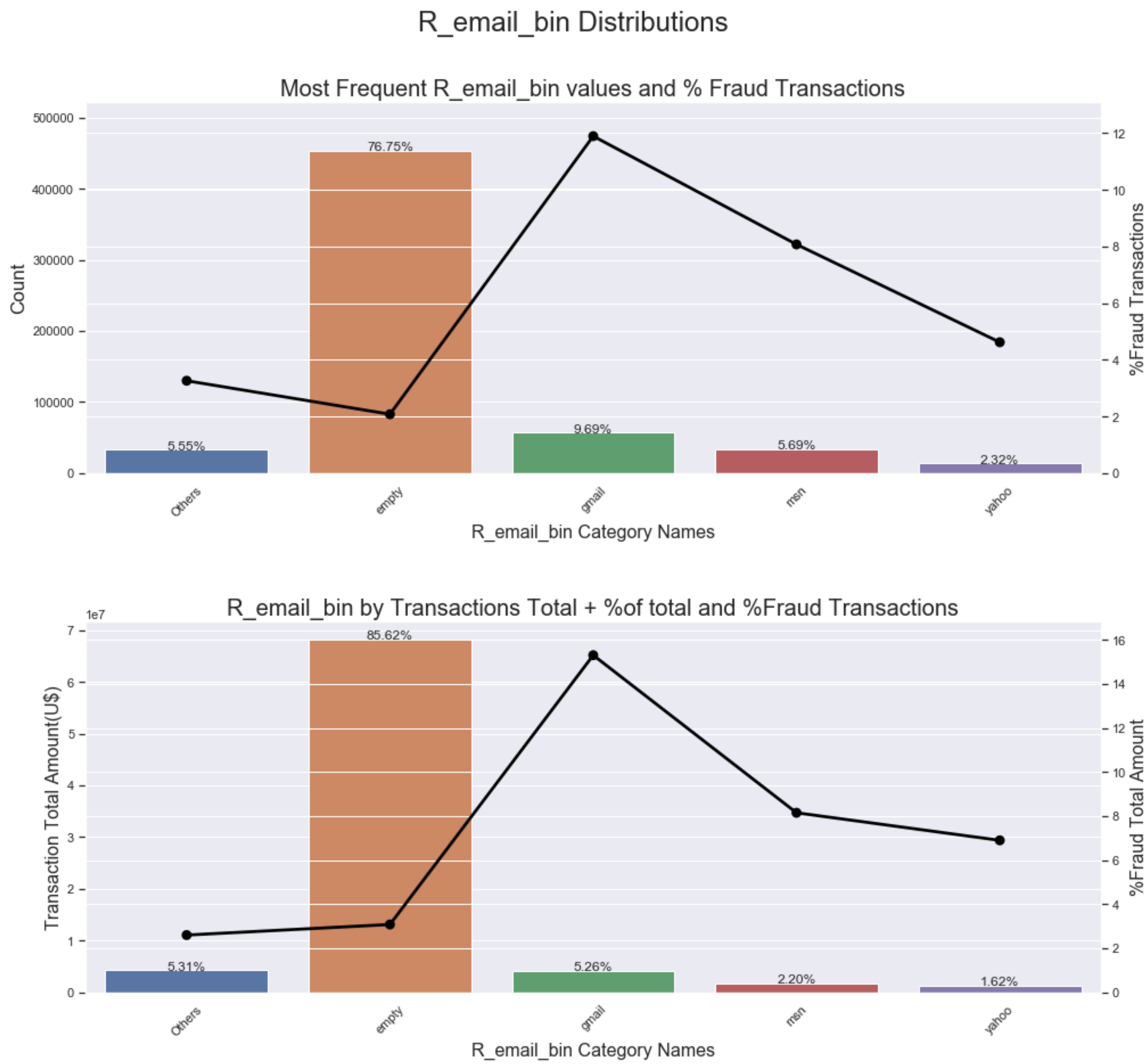
Out[395]: 61

In [396]:
```
# Add new column in Train set
train_trans['R_email_bin'] = train_trans['R_emaildomain'].apply(email_bin)
```

In [397]:
```
# Drop P_emaildomain in Train set
train_trans = train_trans.drop('R_emaildomain', axis=1)
```

In [398]:
```
# Add new column in Train set
test_trans['R_email_bin'] = test_trans['R_emaildomain'].apply(email_bin)

# Drop P_emaildomain in Train set
test_trans = test_trans.drop('R_emaildomain', axis=1)
```

In [399]:
```
ploting_cnt_amt(train_trans, 'R_email_bin')
```



Larger of transaction (76.75%) dont carry receiver emails. Noticed that gmail account is highly used for fraudlent transactions. Due to larger number of nulls(empty), this attribute is not highly important .

P_emaildomain: Purchase email domain

In [400]:
```
# Unique values
train_trans['P_emaildomain'].nunique()
```

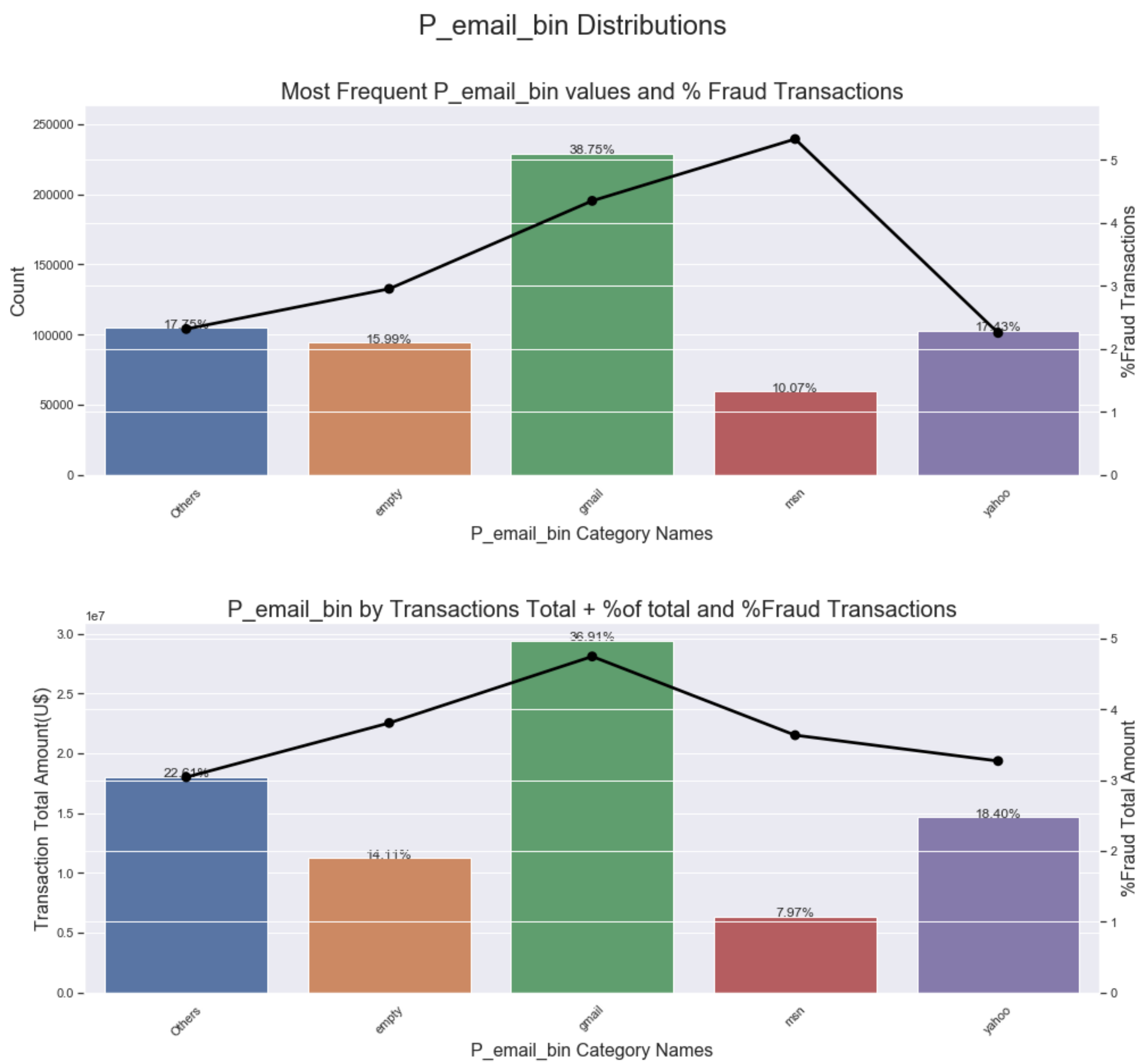
Out[400]: 60

In [401]:
```
# Add new column in Train set
train_trans['P_email_bin'] = train_trans['P_emaildomain'].apply(email_bin)

# Drop P_emaildomain in Train set
train_trans = train_trans.drop('P_emaildomain', axis=1)
```

In [402]:
```
train_trans.shape
```
Out[402]:
```
(590540, 79)
```

In [403]:
```
# Add new column in Train set
test_trans['P_email_bin'] = test_trans['P_emaildomain'].apply(email_bin)

# Drop P_emaildomain in Train set
test_trans = test_trans.drop('P_emaildomain', axis=1)

# test shape
test_trans.shape
```
Out[403]:
```
(506691, 78)
```

In [404]:
```
ploting_cnt_amt(train_trans, 'P_email_bin')
```



It seems that gmail account is highly used for purchases and also has highest fraud transaction value . MSN is highly used for fraud transactions.

Card6 : Card Category (Debit or Credit or Charge Card)

In [405]:
```
# train unique values
print('Train:',train_trans['card6'].unique())

# test unique values
print('Test:',test_trans['card6'].unique())
```

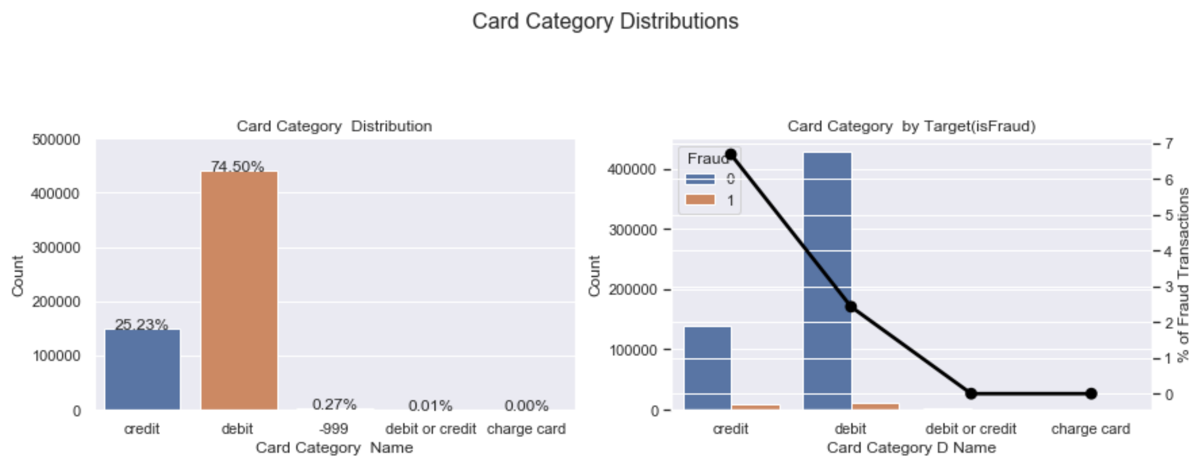
```
Train: ['credit' 'debit' -999 'debit or credit' 'charge card']
Test: ['debit' 'credit' -999 'charge card']
```

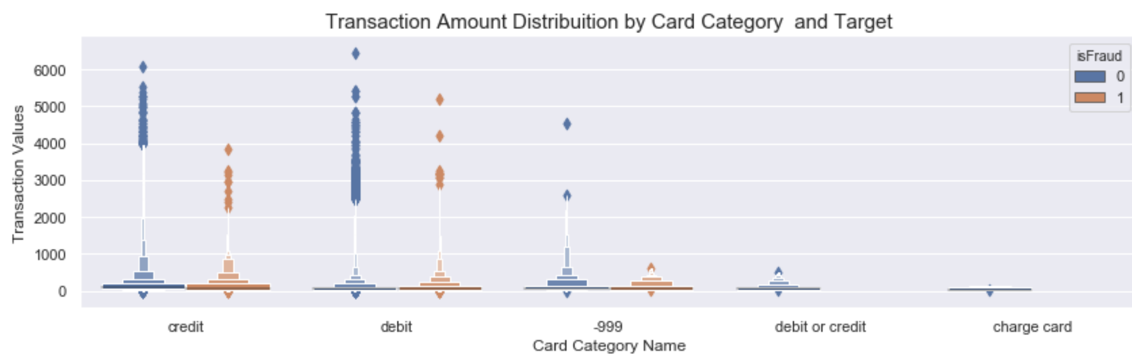
In [406]:
```
tmp = pd.crosstab(train_trans['card6'], train_trans['isFraud'], normalize='ind
ex') * 100
tmp = tmp.reset_index()
tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
tmp.sort_values('Fraud',ascending=False)
```

Out[406]:

isFraud	card6	NoFraud	Fraud
2	credit	93.321520	6.678480
0	-999	97.517505	2.482495
3	debit	97.573749	2.426251
1	charge card	100.000000	0.000000
4	debit or credit	100.000000	0.000000

In [407]:
```
# Visualize

total = len(train_trans)

plt.figure(figsize=(14,10))
plt.suptitle('Card Category Distributions', fontsize=16)

# Subplot 1 - Count vs ProductCD

plt.subplot(221)
g = sns.countplot(x='card6', data=train_trans)

g.set_title("Card Category  Distribution", fontsize=12)
g.set_xlabel("Card Category  Name", fontsize=12)
g.set_ylabel("Count", fontsize=12)
g.set_ylim(0,500000)

## Percentage calculation
for p in g.patches:
    height = p.get_height()
    g.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total*100),
            ha="center", fontsize=12)

# Subplot 2 -

plt.subplot(222)
g1 = sns.countplot(x='card6', hue='isFraud', data=train_trans)
plt.legend(title='Fraud', loc='best', labels=[0,1])
gt = g1.twinx()
gt = sns.pointplot(x='card6', y='Fraud', data=tmp, color='black', order=['cred
it', 'debit', 'debit or credit', 'charge card'], legend=False)
gt.set_ylabel("% of Fraud Transactions", fontsize=12)

g1.set_title("Card Category  by Target(isFraud)", fontsize=12)
g1.set_xlabel("Card Category D Name", fontsize=12)
g1.set_ylabel("Count", fontsize=12)


# Subplot 3 -

plt.subplot(212)
g3 = sns.boxenplot(x='card6', y='TransactionAmt', hue='isFraud',
              data=train_trans[train_trans['TransactionAmt'] <= 10000] )
g3.set_title("Transaction Amount Distribuition by Card Category  and Target",
fontsize=15)
g3.set_xlabel("Card Category Name", fontsize=12)
g3.set_ylabel("Transaction Values", fontsize=12)

plt.subplots_adjust(hspace = 0.6, top = 0.85)

plt.show()
```





Noticed that test set don't have instances with "Debit or Credit" ("card6" 's attribute value)

In [408]:

```
# Treat "Debit or Credit" as "Debit"

# Train set
train_trans['card6'] = train_trans['card6'].apply(lambda x:'debit' if x =='debit or credit' else x)

# Test set
test_trans['card6'] = test_trans['card6'].apply(lambda x:'debit' if x =='debit or credit' else x)
```

In [409]:

```
# train unique values
print('Train:',train_trans['card6'].unique())

# test unique values
print('Test:',test_trans['card6'].unique())
```

```
Train: ['credit' 'debit' -999 'charge card']
Test: ['debit' 'credit' -999 'charge card']
```

Its a categorical value of the card, represents whether the card is "debit", "credit", "charge" or "debit or credit". Noticed there are no transactions for "debit or credit" in test set. I have treated "debit or credit" card as "debit" card. Interestingly, Credit card has highest amout of fraudlent transactions. And its obvious, "debit" (what goes out) card linked to highest number of fraud transactions.

Card Details : card1 to card3, card5

In [410]:
```
# Noticed "country + region" has its own count of "isFraud" transactions.
train_trans.groupby(['card1','card2','card3','card5']).agg({'isFraud':'size'})
.sort_values('isFraud',ascending=False).head(5)
```

Out[410]:

				isFraud
card1	**card2**	**card3**	**card5**	
9500	**321**	**150**	**226**	14112
15885	**545**	**185**	**138**	10332
17188	**321**	**150**	**226**	10312
7919	**194**	**150**	**166**	8844
15066	**170**	**150**	**102**	7918

In [411]:
```
# new column: "card1+ card2 + card3 + card5" as weight
columns = ['card1','card2','card3','card5']

# Train set
train_trans['card_1235'] = train_trans[columns].astype(int).astype(str).sum(ax
is=1)
```

In [412]:
```
# Test set
test_trans['card_1235'] = test_trans[columns].astype(int).astype(str).sum(axis
=1)
```

In [413]:
```
# After concatnate
train_trans.groupby(['card_1235']).agg({'isFraud':'size'}).sort_values('isFrau
d',ascending=False).head(5)
```

Out[413]:

	isFraud
card_1235	
9.500321e+12	14112
1.588555e+13	10332
1.718832e+13	10312
7.919194e+12	8844
1.506617e+13	7918

In [417]:
```
train_trans['TransactionAmt_to_mean_card_1235'] = train_trans['TransactionAmt'
] / train_trans.groupby(['card_1235'])['TransactionAmt'].transform('mean')
```

In [431]:
```
test_trans['TransactionAmt_to_mean_card_1235'] = test_trans['TransactionAmt']
/ test_trans.groupby(['card_1235'])['TransactionAmt'].transform('mean')
```

In [432]:
```
# Drop card_1235 in Train set
train_trans = train_trans.drop('card_1235', axis=1)
```

In [433]:
```
# Drop card_1235 in Train set
test_trans = test_trans.drop('card_1235', axis=1)
```

In [418]:
```
tmp = pd.crosstab(train_trans['TransactionAmt_to_mean_card_1235'], train_trans
['isFraud'], normalize='index') * 100
tmp = tmp.reset_index()
tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
tmp.sort_values('Fraud',ascending=False).head(5)
```

Out[418]:

isFraud	TransactionAmt_to_mean_card_1235	NoFraud	Fraud
123788	1.147689	0.0	100.0
173954	2.909696	0.0	100.0
169188	2.527565	0.0	100.0
24849	0.270525	0.0	100.0
148511	1.660036	0.0	100.0

In [429]:
```
Q = (tmp['Fraud']>75.0)
Q1 = (tmp['TransactionAmt_to_mean_card_1235']>75)
Q3 = Q|Q1
data = list(tmp[Q3]['TransactionAmt_to_mean_card_1235'].values)
```

Concatenate of card1,card2, card3 and card5 will provide a unique card id. Actually, its a categorical value. I think, further categorizing is required. For time being, lets treat them as weight.

addr1, addr2 : Billing region, Biling country

In [434]:
```
# Noticed "country + region" has its own count of "isFraud" transactions
train_trans.groupby(['addr2','addr1']).agg({'isFraud':'size'}).sort_values('is
Fraud',ascending=False).head(5)
```

Out[434]:

addr2	addr1	isFraud
9	**99**	65706
87	**299**	46324
	325	42748
	204	42018
	264	39870

In [435]:
```
# new column: "country + region" as weight
columns = ['addr2', 'addr1']

# Train set
train_trans['addr_21'] = train_trans[columns].astype(int).astype(str).sum(axis
=1)
```

In [436]:
```
# Test set
test_trans['addr_21'] = test_trans[columns].astype(int).astype(str).sum(axis=1
)
```

In [437]:
```
train_trans.groupby(['addr_21']).agg({'isFraud':'size'}).sort_values('isFraud'
,ascending=False).head(5)
```

Out[437]:

	isFraud
addr_21	
999.0	65706
87299.0	46324
87325.0	42748
87204.0	42018
87264.0	39870

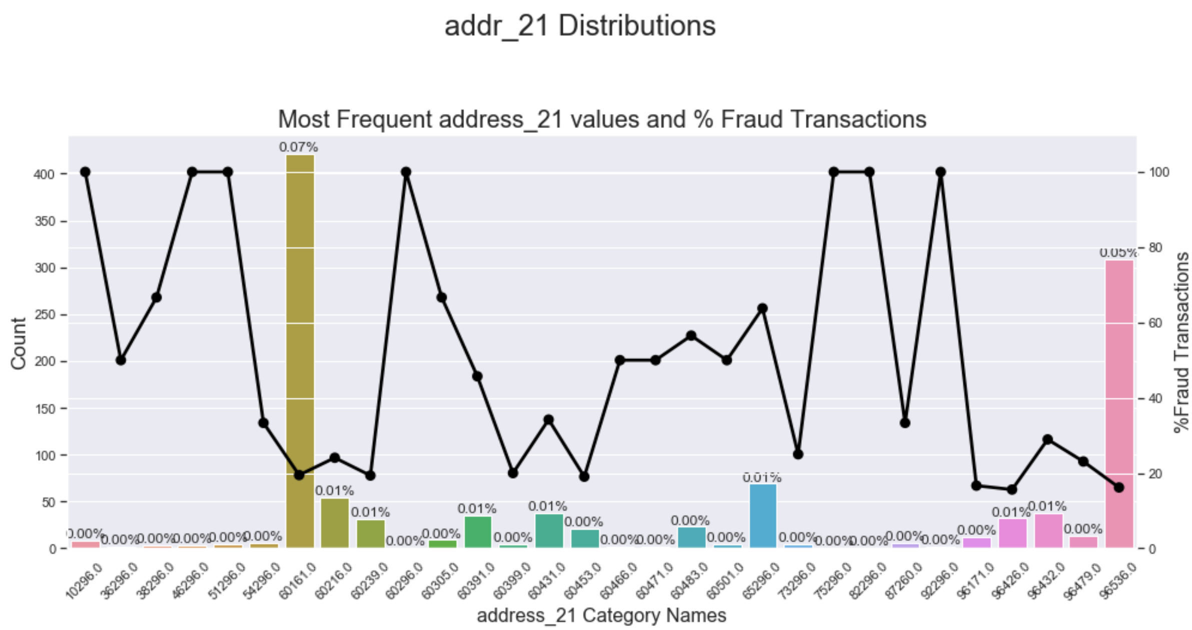
In [438]:
```
tmp = pd.crosstab(train_trans['addr_21'], train_trans['isFraud'], normalize='i
ndex') * 100
tmp = tmp.reset_index()
tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
tmp.sort_values('Fraud',ascending=False).head(5)
```

Out[438]:

isFraud	addr_21	NoFraud	Fraud
1	10296.0	0.0	100.0
151	60296.0	0.0	100.0
113	51296.0	0.0	100.0
228	75296.0	0.0	100.0
108	46296.0	0.0	100.0

In [439]:
```
Q = (tmp['Fraud']>15.0)

data = list(tmp[Q]['addr_21'].values)
```

In [440]:
```
plt.figure(figsize=(16,14))
plt.suptitle('addr_21 Distributions ', fontsize=24)

plt.subplot(211)
g = sns.countplot( x='addr_21',  data=train_trans, order=data)
gt = g.twinx()
gt = sns.pointplot(x='addr_21', y='Fraud', data=tmp, order=data,color='black', 
legend=False, )
gt.set_ylim(0,tmp['Fraud'].max()*1.1)
gt.set_ylabel("%Fraud Transactions", fontsize=16)
g.set_title("Most Frequent address_21 values and % Fraud Transactions", fontsi
ze=20)
g.set_xlabel("address_21 Category Names", fontsize=16)
g.set_ylabel("Count", fontsize=17)
g.set_xticklabels(g.get_xticklabels(),rotation=45)
sizes = []
for p in g.patches:
    height = p.get_height()
    sizes.append(height)
    g.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total*100),
            ha="center",fontsize=12)
```



In [441]:
```
train_trans['TransactionAmt_to_mean_addr_21'] = train_trans['TransactionAmt']
/ train_trans.groupby(['addr_21'])['TransactionAmt'].transform('mean')
```

In [442]:
```
test_trans['TransactionAmt_to_mean_addr_21'] = test_trans['TransactionAmt'] /
test_trans.groupby(['addr_21'])['TransactionAmt'].transform('mean')
```

In [444]:
```
# Drop card_21 in Train set
train_trans = train_trans.drop('addr_21', axis=1)
```

In [445]:
```
# Drop card_21 in Test set
test_trans = test_trans.drop('addr_21', axis=1)
```

Concatenating region and country will provide a unique pincode. Actually, it a categorical value. For time being, lets treat this new column(addr_21) as weight.

Final drop : Columns (having noise)

Lets drop all unwanted columns.

In [452]:
```
drop_cols = ['id_12', 'id_15', 'id_16', 'id_28', 'id_29', 'id_30', 'id_31', 'id_33', 'id_34',
             'id_35', 'id_36', 'id_37', 'id_38', 'DeviceType', 'DeviceInfo', 'id_01', 'id_02',
             'id_03', 'id_04', 'id_05', 'id_06', 'id_07', 'id_08', 'id_09', 'id_10', 'id_11',
             'id_13', 'id_14', 'id_17', 'id_18', 'id_19', 'id_20', 'id_21', 'id_22', 'id_24',
             'id_25', 'id_26', 'id_32']
```

In [453]:
```
# Train set
train_trans = train_trans.drop(train_trans[drop_cols], axis=1)
```

In [454]:
```
# Test set
test_trans = test_trans.drop(test_trans[drop_cols], axis=1)
```

In [455]:
```
train_trans.shape
```

Out[455]: (590540, 44)

In [456]:
```
test_trans.shape
```

Out[456]: (506691, 43)

Data Split

Train Data subset

In [100]:
```
# For reproducibility of the results
np.random.seed(42)
rndperm = np.random.permutation(train_trans.shape[0])
```

In [101]:
```
N = 60000
df_subset = train_trans.loc[rndperm[:N],:].copy()
#data_subset = df_subset[feat_cols].values
#data_subset.shape
df_subset.shape
```

Out[101]:
```
(60000, 37)
```

Train Data spilt: Data vs Label

In [457]:
```
label = 'isFraud'
idcol = 'TransactionID'
predictors = [c for c in train_trans.columns if c not in [label, idcol]]
len(predictors)
```

Out[457]:
```
42
```

Numerical Attributes vs Categorical Attributes

In [458]:
```
num_attribs = [c for c in train_trans[predictors] if train_trans[c].dtype not
in ['object']]
cat_attribs = [c for c in train_trans[predictors] if train_trans[c].dtype ==
'object']
```

In [459]:
```
print(cat_attribs)
```

```
['ProductCD', 'card4', 'card6', 'M1', 'M2', 'M3', 'M4', 'M5', 'M6', 'M7', 'M
8', 'M9', 'R_email_bin', 'P_email_bin']
```

In [460]:
```
print(num_attribs)
```

```
['TransactionDT', 'TransactionAmt', 'card1', 'card2', 'card3', 'card5', 'addr
1', 'addr2', 'dist1', 'dist2', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D
8', 'D9', 'D10', 'D11', 'D12', 'D13', 'D14', 'D15', 'TransactionAmt_to_mean_c
ard_1235', 'TransactionAmt_to_mean_addr_21', 'hours']
```

In [461]:
```
print('Numerical Attributes :',len(num_attribs))
print('Categorical Attributes :',len(cat_attribs))
```

```
Numerical Attributes : 28
Categorical Attributes : 14
```

X -vs- y sets

In [462]:
```
X_train = train_trans[predictors]
```

In [463]:
```
y_train = train_trans[label]
```

In [464]:
```
X_train.shape
```

Out[464]: (590540, 42)

In [465]:
```
X_test = test_trans[predictors]
```

In [466]:
```
X_test.shape
```

Out[466]: (506691, 42)

- Number of predictors of X_train and X_test are matched.

Feature Engineering

Lable Encoding

In [467]:
```
# Label Encoding

def cat_label(df):
    lbl = LabelEncoder()
    for f in df.columns:
        if df[f].dtype == 'object':
            lbl.fit(list(df[f].values))
            df[f] = lbl.transform(list(df[f].values))
    return(df)
```

In [468]:
```
train_prepared = cat_label(X_train)
```

In [469]:
```
train_prepared.shape
```

Out[469]: (590540, 42)

In [470]:
```
test_prepared = cat_label(X_test)
```

In [471]:
```
test_prepared.shape
```

Out[471]: (506691, 42)

Principal Component analysis(PCA) is not helpful for this dataset, as:

```
1. PCA features is 1 with high variance ratio, which mayn't helpful to ensemble mod
els.
2. There is no imporvement in predicting labels with PCA dimensions.
3. Decision trees can handle feature importance.
```

In [472]:
```
gc.collect()
```

Out[472]:
```
94
```

Validation set

In [473]:
```
# Split Train and Validation
X_train_prep, X_val_prep, y_train, y_val = train_test_split(train_prepared,
                                                             y_train,
                                                             test_size=0.2,
                                                             random_state=seed)
```

In [474]:
```
# Data
print(X_train_prep.shape)
print(X_val_prep.shape)
print('-------------------------')
# label
print(y_train.shape)
print(y_val.shape)
```

```
(472432, 42)
(118108, 42)
-------------------------
(472432,)
(118108,)
```

Modeling & Validation

Logistic Regression

In [306]:
```
# Start time
start_time = time.time()

# Model
log_reg_clf = LogisticRegression(random_state = seed)
log_reg_clf.fit(X_train_prep, y_train)

# Time Taken
print("Time taken:--- %s seconds ---" % (time.time() - start_time))
```

```
Time taken:--- 2.3486828804016113 seconds ---
```

In [307]:
```
# Get score on training set and validation set for our StackNetClassifier
train_preds = log_reg_clf.predict_proba(X_train_prep)[:, 1]
val_preds = log_reg_clf.predict_proba(X_val_prep)[:, 1]
train_score = auc_score(y_train, train_preds)
val_score = auc_score(y_val, val_preds)
```

In [308]:
```
print(f" Logistic Regression AUC on training set: {round(train_score, 4)}")
print(f"Logistic Regression AUC on validation set: {round(val_score, 4)}")
```

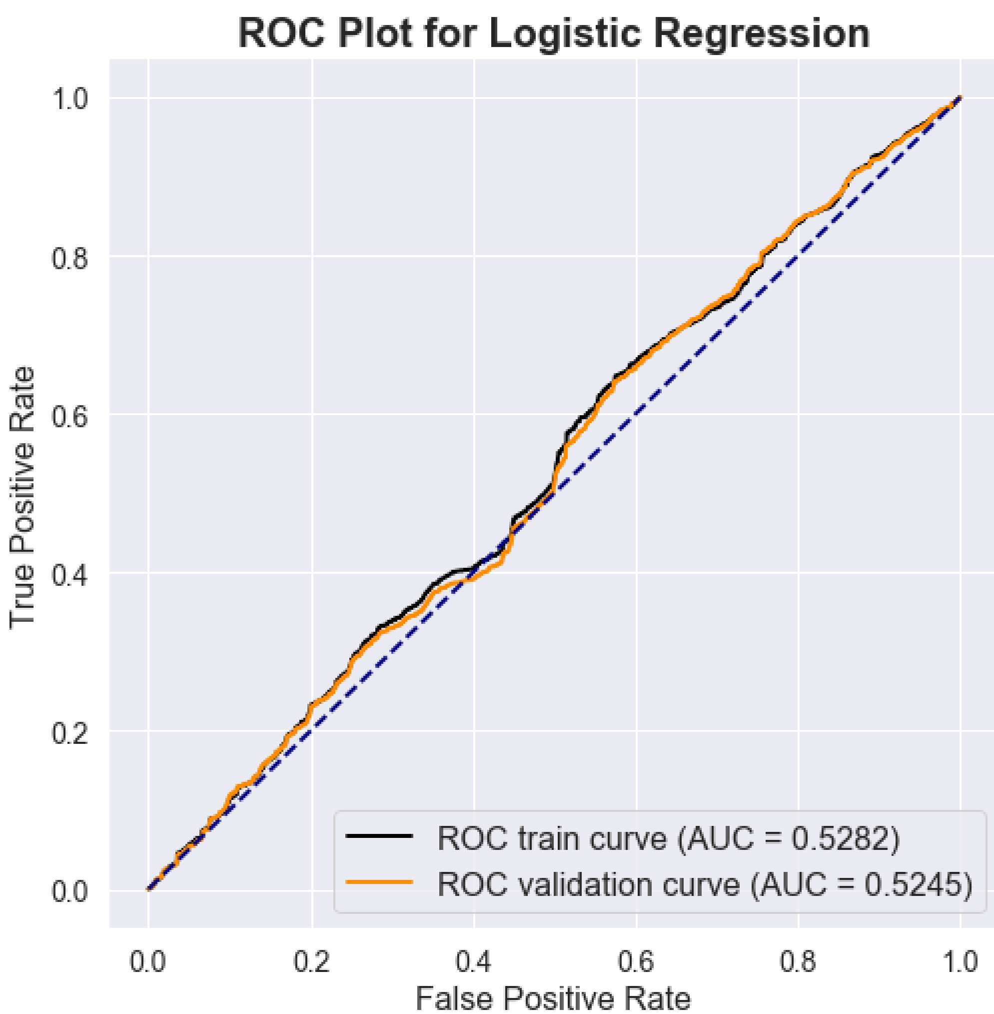
```
 Logistic Regression AUC on training set: 0.5282
Logistic Regression AUC on validation set: 0.5245
```

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection[5] in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm.

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is also called a perfect classification.

When using normalized units, the area under the curve (often referred to as simply the AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative')

In [244]:
```
# Plot ROC curve
plot_curve(y_train, train_preds, y_val, val_preds, "Logistic Regression")
```



XGBoost Classifier

In [297]:
```
# hyper parameters are identified after mutliple random runs as part of tunnin
g.

params = {
    'objective': 'binary:logistic',
    'max_depth': 4,
    'learning_rate': 0.1,
    'n_estimators': 1400,
    'gamma': 0,
    'silent': 1
}
```

In [287]:
```
# Start time
start_time = time.time()

xgb_clf = XGBClassifier(**params).fit(X_train_prep, y_train)
print(xgb_clf)

# Time Taken
print("Time taken:--- %s seconds ---" % (time.time() - start_time))
```

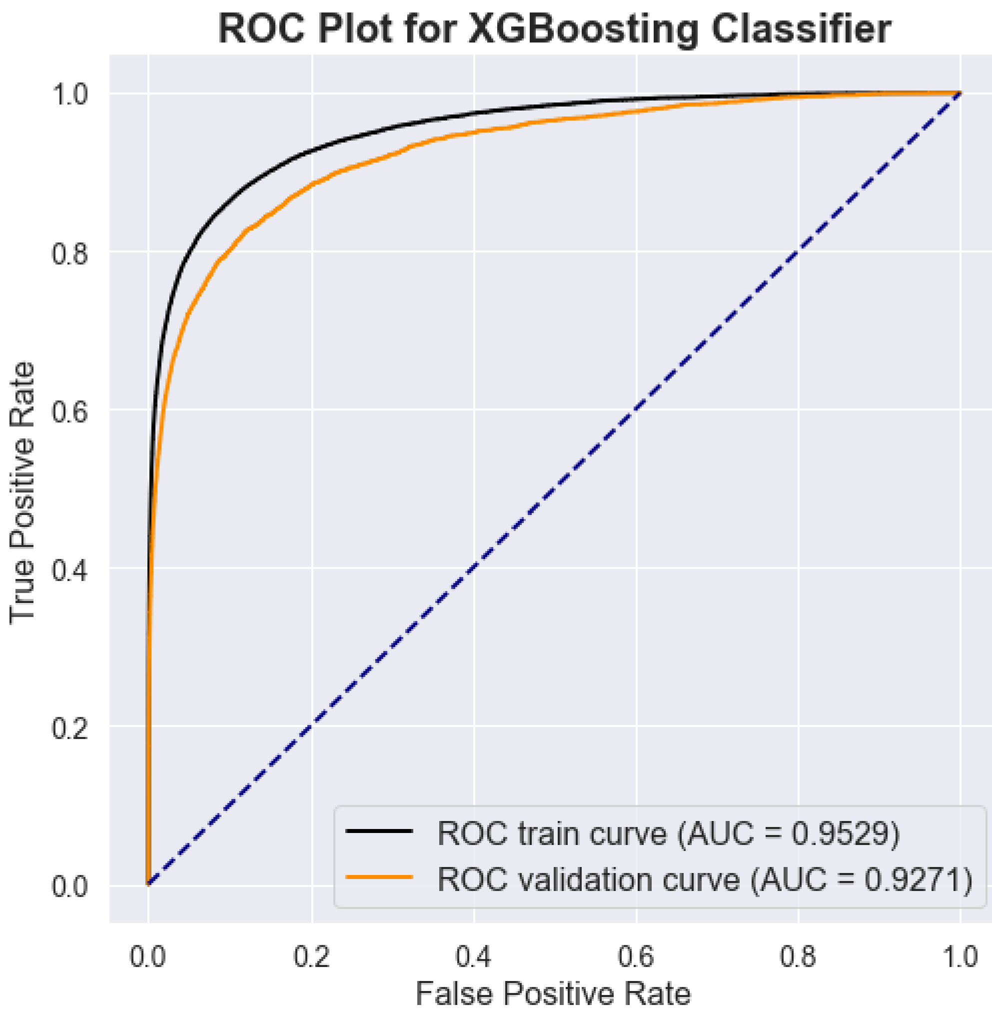
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=4, min_child_weight=1, missing=None,
       n_estimators=1400, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=1, subsample=1,
       verbosity=1)
Time taken:--- 723.4190270900726 seconds ---
```

In [288]:
```
# Get score on training set and validation set for XGB Classifier
train_preds = xgb_clf.predict_proba(X_train_prep)[:, 1]
val_preds = xgb_clf.predict_proba(X_val_prep)[:, 1]
train_score = auc_score(y_train, train_preds)
val_score = auc_score(y_val, val_preds)
```

In [289]:
```
print(f"StackNet AUC on training set: {round(train_score, 4)}")
print(f"StackNet AUC on validation set: {round(val_score, 4)}")
```

```
StackNet AUC on training set: 0.9529
StackNet AUC on validation set: 0.9271
```

In [290]:
```
# Plot ROC curve
plot_curve(y_train, train_preds, y_val, val_preds, "XGBoosting Classifier")
```



GradientBoostingClassifier

In [311]:
```
gbrt_clf = GradientBoostingClassifier(max_depth=2, n_estimators=500, learning_
rate=0.1, random_state=seed)
gbrt_clf
```

Out[311]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=500,
              n_iter_no_change=None, presort='auto', random_state=342,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

In [312]:
```
# Start time
start_time = time.time()

gbrt_clf.fit(X_train_prep, y_train)

# Time Taken
print("Time taken:--- %s seconds ---" % (time.time() - start_time))
```

```
Time taken:--- 258.41605138778687 seconds ---
```

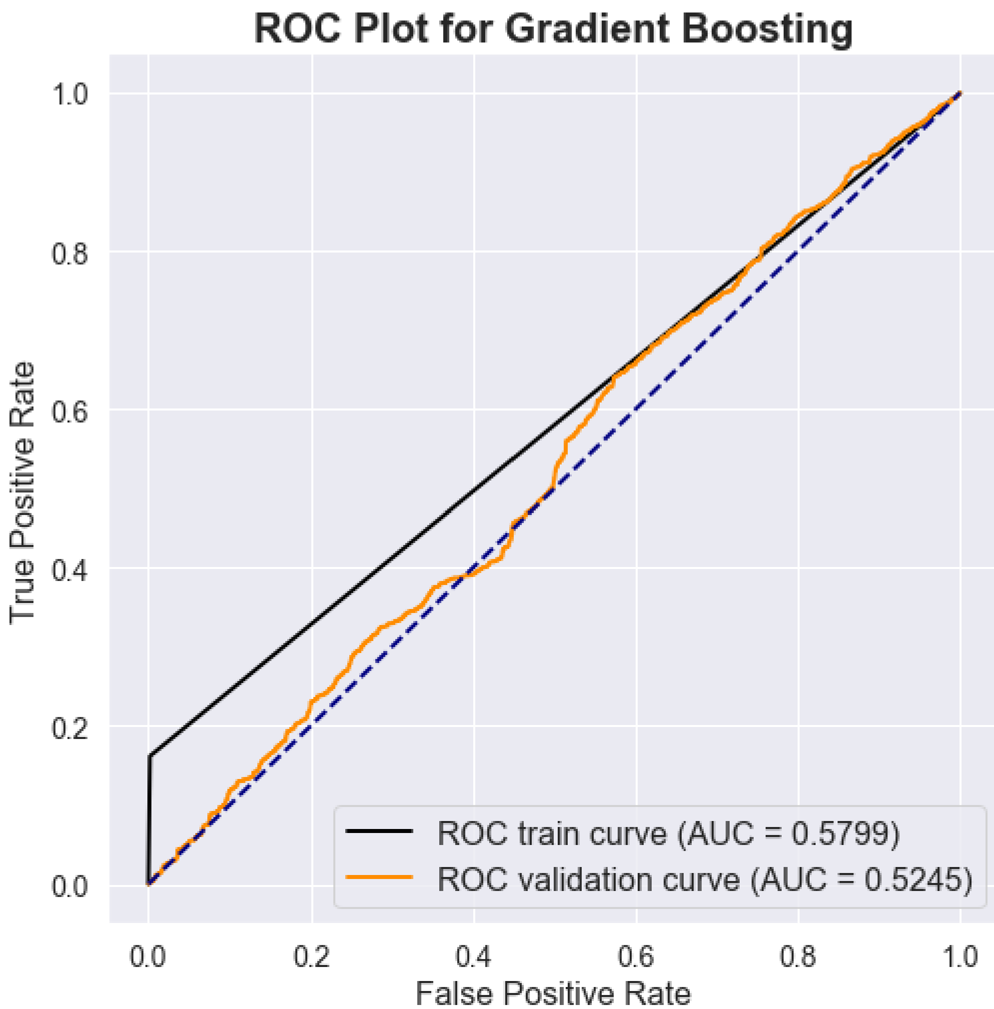
In [313]:
```
train_preds = gbrt_clf.predict(X_train_prep)
test_preds = gbrt_clf.predict(X_val_prep)
```

In [314]:
```
train_score = auc_score(y_train, train_preds)
val_score = auc_score(y_val, val_preds)
```

In [315]:
```
print(f"StackNet AUC on training set: {round(train_score, 4)}")
print(f"StackNet AUC on validation set: {round(val_score, 4)}")
```

```
StackNet AUC on training set: 0.5799
StackNet AUC on validation set: 0.5245
```

In [316]:
```
# Plot ROC curve
plot_curve(y_train, train_preds, y_val, val_preds, "Gradient Boosting")
```



RandomForest Classifier

In [298]:
```
rnd_clf = RandomForestClassifier(n_estimators=1400, max_depth=2,n_jobs=-1, ran
dom_state=seed)
rnd_clf
```

Out[298]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=2, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1400, n_jobs=-1,
            oob_score=False, random_state=342, verbose=0, warm_start=False)
```

In [299]:
```
# Start time
start_time = time.time()

rnd_clf.fit(X_train_prep, y_train)

# Time Taken
print("Time taken:--- %s seconds ---" % (time.time() - start_time))
```

```
Time taken:--- 73.48510527610779 seconds ---
```

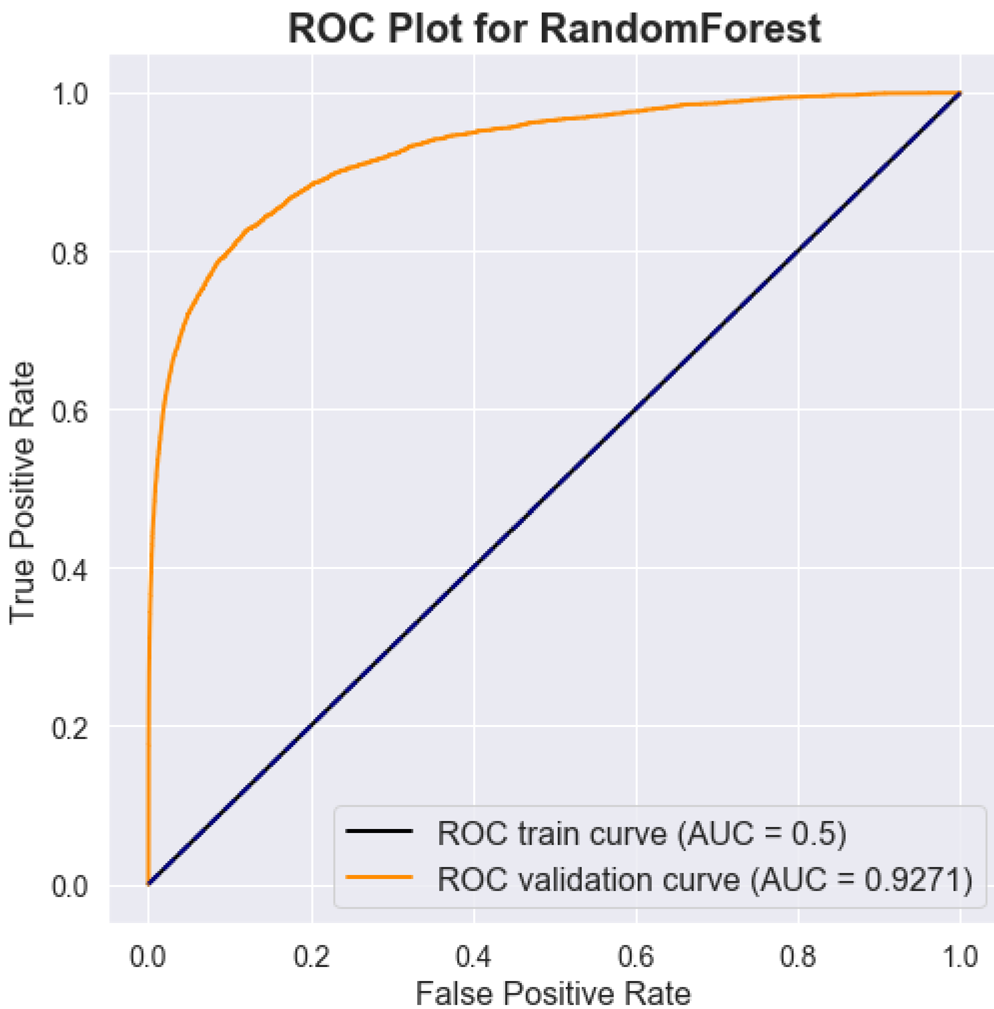
In [300]:
```
train_preds = rnd_clf.predict(X_train_prep)
test_preds = rnd_clf.predict(X_val_prep)
```

In [301]:
```
train_score = auc_score(y_train, train_preds)
val_score = auc_score(y_val, val_preds)
```

In [302]:
```
print(f"StackNet AUC on training set: {round(train_score, 4)}")
print(f"StackNet AUC on validation set: {round(val_score, 4)}")
```

```
StackNet AUC on training set: 0.5
StackNet AUC on validation set: 0.9271
```

In [303]:
```
# Plot ROC curve
plot_curve(y_train, train_preds, y_val, val_preds, "RandomForest")
```



CatBoostClassifier

In [262]:
```
param_cb = {
        'learning_rate': 0.1,
        'l2_leaf_reg': 30,
        'depth': 2,
        'max_bin':255,
        'iterations' : 1000,
        'eval_metric' : "AUC",
        'bootstrap_type' : 'Bayesian',
        'random_seed':42 }


cat_clf = CatBoostClassifier(silent=True, **param_cb)
```

In [264]:
```
# Start time
start_time = time.time()

cat_clf.fit(X_train_prep, y_train)

# Time Taken
print("Time taken:--- %s seconds ---" % (time.time() - start_time))
```

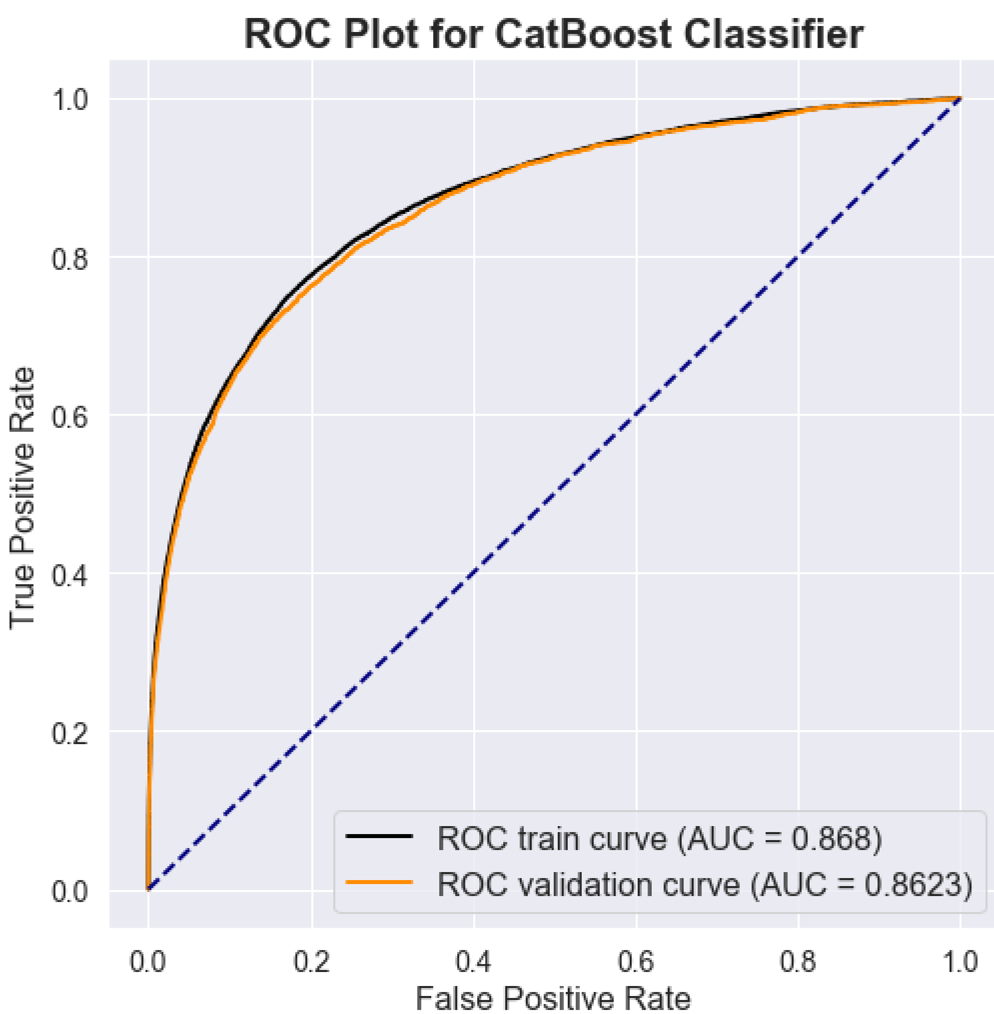
```
Time taken:--- 351.47167921066284 seconds ---
```

In [265]:
```
# Get score on training set and validation set for Catboost Classifier
train_preds = cat_clf.predict_proba(X_train_prep)[:, 1]
val_preds = cat_clf.predict_proba(X_val_prep)[:, 1]
train_score = auc_score(y_train, train_preds)
val_score = auc_score(y_val, val_preds)
```

In [266]:
```
print(f"StackNet AUC on training set: {round(train_score, 4)}")
print(f"StackNet AUC on validation set: {round(val_score, 4)}")
```

```
StackNet AUC on training set: 0.868
StackNet AUC on validation set: 0.8623
```

In [267]:
```
# Plot ROC curve
plot_curve(y_train, train_preds, y_val, val_preds, "CatBoost Classifier")
```



Model Comparison

Model	Train AUC Score	Validation AUC Score	Time Taken
LogisticRegression	0.5282	0.5245	4.417233943939209 seconds
XGBClassifier	0.9529	0.9271	723.4190270900726 seconds
GradientBoostingClassifier	0.5799	0.5245	258.41605138778687 seconds
RandomForestClassifier	0.5	0.9271	73.48510527610779 seconds
CatBoostClassifier	0.868	0.8623	351.47167921066284 seconds

Model Tunning

Tunning n_estimators : Number of trees

I ran the below mutlitple times by adjusting the range and keeping other hyperparameters as constant.

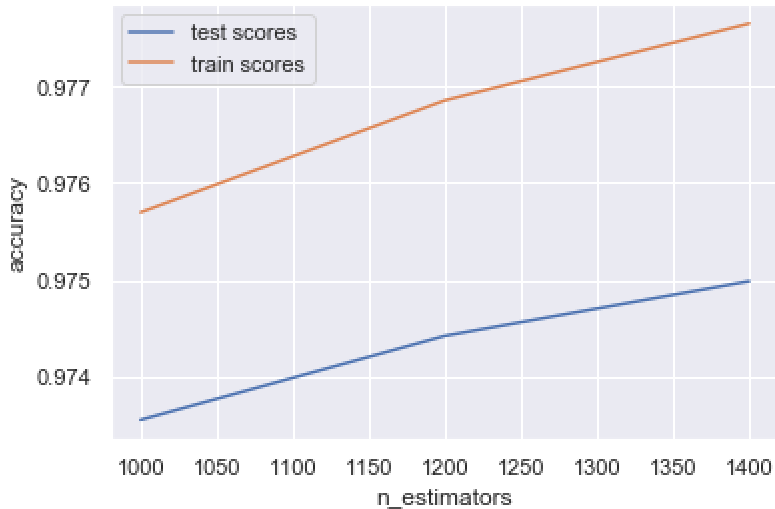
In [295]:

```
train_scores = []
test_scores = []
train_best_score = 0
test_best_score = 0
train_best_estimators = 0
test_best_estimators = 0

start_time = time.time()

# estimator range 1,5,10....100
estimator_range = range(1000, 1500, 200)
for n_estimators in estimator_range:
    xgb_clf.n_estimators = n_estimators

    #fit the model
    xgb_clf.fit(X_train_prep, y_train)

    #Train score
    train_score = xgb_clf.score(X_train_prep, y_train)
    train_scores.append(train_score)
    if train_score>train_best_score:
        train_best_score = train_score
        train_best_estimators = n_estimators


    #Test score
    test_score = xgb_clf.score(X_val_prep, y_val)
    test_scores.append(test_score)
    if test_score > test_best_score:
        test_best_score = test_score
        test_best_estimators = n_estimators

print("Time taken:--- %s seconds ---" % (time.time() - start_time))

print(" Train- best score :%s" %train_best_score)
print(" Train- Estimator:%s " %train_best_estimators)
print('-----------------------------------------------------------')
print(" Validation- best score :%s" %test_best_score)
print(" Validation- Estimator:%s " %test_best_estimators)
```

```
Time taken:--- 4210.579234600067 seconds ---
 Train- best score :0.9776539269143496
 Train- Estimator:1400
-----------------------------------------------------------
 Validation- best score :0.9749889931249365
 Validation- Estimator:1400
```

In [296]:
```
plt.plot(estimator_range, test_scores, label="test scores")
plt.plot(estimator_range, train_scores, label="train scores")
plt.ylabel("accuracy")
plt.xlabel("n_estimators")
plt.legend()
```

Out[296]: <matplotlib.legend.Legend at 0x223bfd02208>



General Approach for Parameter Tunning

- Lower the learning rate, optimal number of trees(n_estimators) and decide the optimal regualizers (lambda, alpha) for xgboost which can help reduce model complexity and enhance performance.
- min_samples_split represents the minimum number of samples required to split an internal node.
- min_samples_leaf is The minimum number of samples required to be at a leaf node
- max_features represents the number of features to consider when looking for the best split.
- n_estimators represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data.
- max_depth. This indicates how deep the built tree can be. The deeper the tree, the more splits it has and it captures more information about how the data

In [217]:
```
min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(900, 1500,100):
    xgb_clf.n_estimators = n_estimators
    xgb_clf.fit(X_train, y_train)
    y_pred = XGBClassifier(**params).predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1
        if error_going_up == 5:
            break  # early stopping
```

In [218]:
```
print(xgb_clf.n_estimators)
```

1400

Model stacking

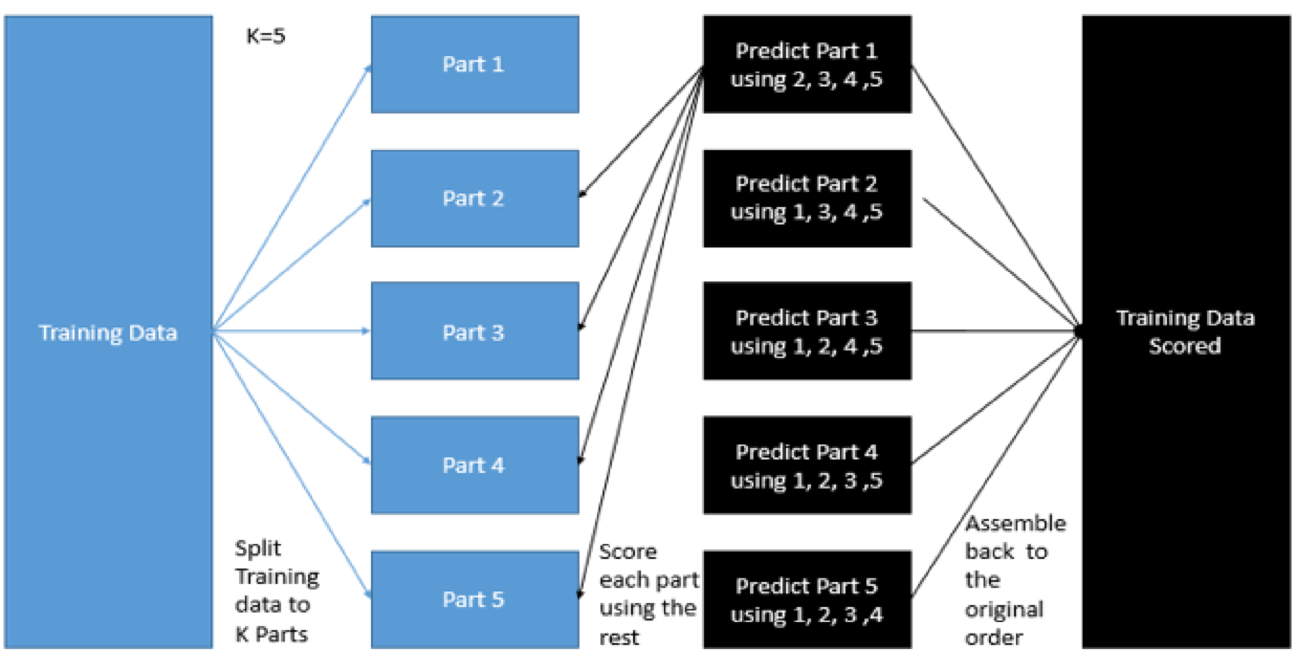
Stacking or Stacked Generalization is the process of combining various machine learning algorithms using holdout data.

StackNet is a computational, scalable and analytical, meta-modeling framework implemented in Java that resembles a feedforward neural network and uses Wolpert's stacked generalization on multiple levels to improve accuracy in machine learning predictive problems.Its created by Marios Michailidis.

It requires a forward training methodology that splits the data into 2 parts (A and B)–one of which is used for training (A) and the other for predictions (B). The reason this split is necessary is to avoid over-fitting. The algorithm utilizes a K-fold cross validation as show in the below diagram.

A good stacking solution is often composed of at least:

- 2 or 3 GBMs (one with low depth, one with medium and one with high)
- 1 or 2 Random Forests (again as diverse as possible–one low depth, one high)
- 1 or 2 NNs (one deeper, one smaller)
- 1 linear model



Stacked as below:

- Level 0 - 2 strong model + 1 Weak Model
- Level 1 - I weak model (Randome forest)

In [475]:
```
# Specify model tree for StackNet
models = [[xgb_clf, log_reg_clf, cat_clf], # Level 0
          [rnd_clf]] # Level 1
```

In [476]:
```
# Specify parameters for stacked model and begin training
model = StackNetClassifier(models,
                           metric="auc",
                           folds=2,
                           restacking=False,
                           use_retraining=True,
                           use_proba=True, # To use predict_proba after traini
ng
                           random_state=seed,
                           n_jobs=-1,
                           verbose=1)
```

In [477]:
```
model.fit(X_train_prep, y_train)
```

```
====================== Start of Level 0 ======================
Input Dimensionality 42 at Level 0
3 models included in Level 0
Fold 1/2 , model 0 , auc===0.922836
Fold 1/2 , model 1 , auc===0.662296
Fold 1/2 , model 2 , auc===0.869111
=========== end of fold 1 in level 0 ===========
Fold 2/2 , model 0 , auc===0.924629
Fold 2/2 , model 1 , auc===0.665026
Fold 2/2 , model 2 , auc===0.868176
=========== end of fold 2 in level 0 ===========
Level 0, model 0 , auc===0.923733
Level 0, model 1 , auc===0.663661
Level 0, model 2 , auc===0.868643
Output dimensionality of level 0 is 3
====================== End of Level 0 ======================
 level 0 lasted 1829.014529 seconds
====================== Start of Level 1 ======================
Input Dimensionality 3 at Level 1
1 models included in Level 1
Fold 1/2 , model 0 , auc===0.867443
=========== end of fold 1 in level 1 ===========
Fold 2/2 , model 0 , auc===0.867577
=========== end of fold 2 in level 1 ===========
Level 1, model 0 , auc===0.867510
Output dimensionality of level 1 is 1
====================== End of Level 1 ======================
 level 1 lasted 150.848905 seconds
====================== End of fit ======================
 fit() lasted 1979.948205 seconds
```

In [478]:
```
# Models stacked in StackNet
model.models
```

Out[478]:
```
[[XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
         colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
         max_delta_step=0, max_depth=4, min_child_weight=1, missing=None,
         n_estimators=1400, n_jobs=1, nthread=None,
         objective='binary:logistic', random_state=0, reg_alpha=0,
         reg_lambda=1, scale_pos_weight=1, seed=None, silent=1, subsample=1,
         verbosity=1),
  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
            intercept_scaling=1, max_iter=100, multi_class='warn',
            n_jobs=None, penalty='l2', random_state=342, solver='warn',
            tol=0.0001, verbose=0, warm_start=False),
  <catboost.core.CatBoostClassifier at 0x223bef885f8>],
 [RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
              max_depth=2, max_features='auto', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=1400, n_jobs=-1,
              oob_score=False, random_state=342, verbose=0, warm_start=Fals
e)]]
```

In [479]:
```
X_matrix = X_train_prep.as_matrix()
X_val_matrix = X_val_prep.as_matrix()
```

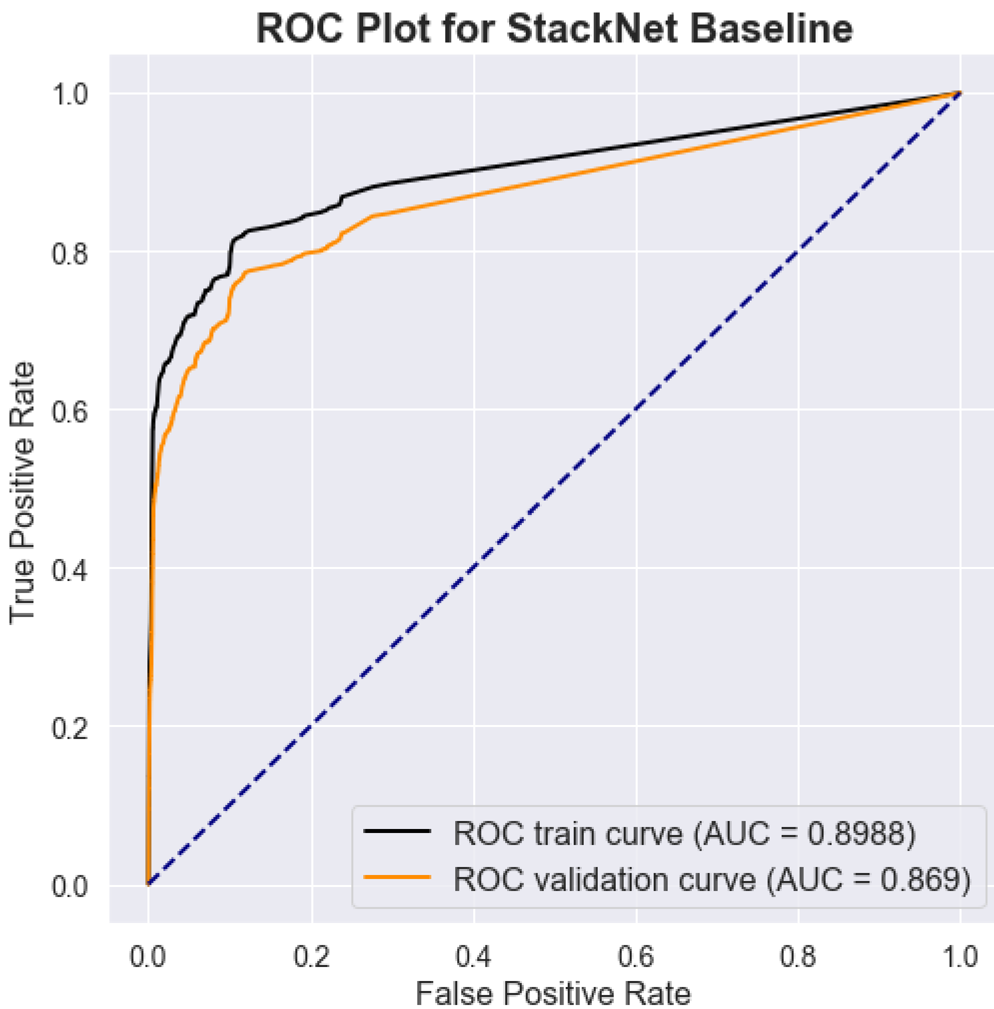
In [480]:
```
# Get score on training set and validation set for StackNet Classifier
train_preds = model.predict_proba(X_matrix)[:, 1]
val_preds = model.predict_proba(X_val_matrix)[:, 1]
train_score = auc_score(y_train, train_preds)
val_score = auc_score(y_val, val_preds)
```

```
====================== Start of Level 0 ======================
1 estimators included in Level 0
====================== Start of Level 1 ======================
1 estimators included in Level 1
====================== Start of Level 0 ======================
1 estimators included in Level 0
====================== Start of Level 1 ======================
1 estimators included in Level 1
```

In [481]:
```
print(f"StackNet AUC on training set: {round(train_score, 4)}")
print(f"StackNet AUC on validation set: {round(val_score, 4)}")
```

```
StackNet AUC on training set: 0.8988
StackNet AUC on validation set: 0.869
```

In [482]:
```
# Plot ROC curve
plot_curve(y_train, train_preds, y_val, val_preds, "StackNet Baseline")
```



Model Testing

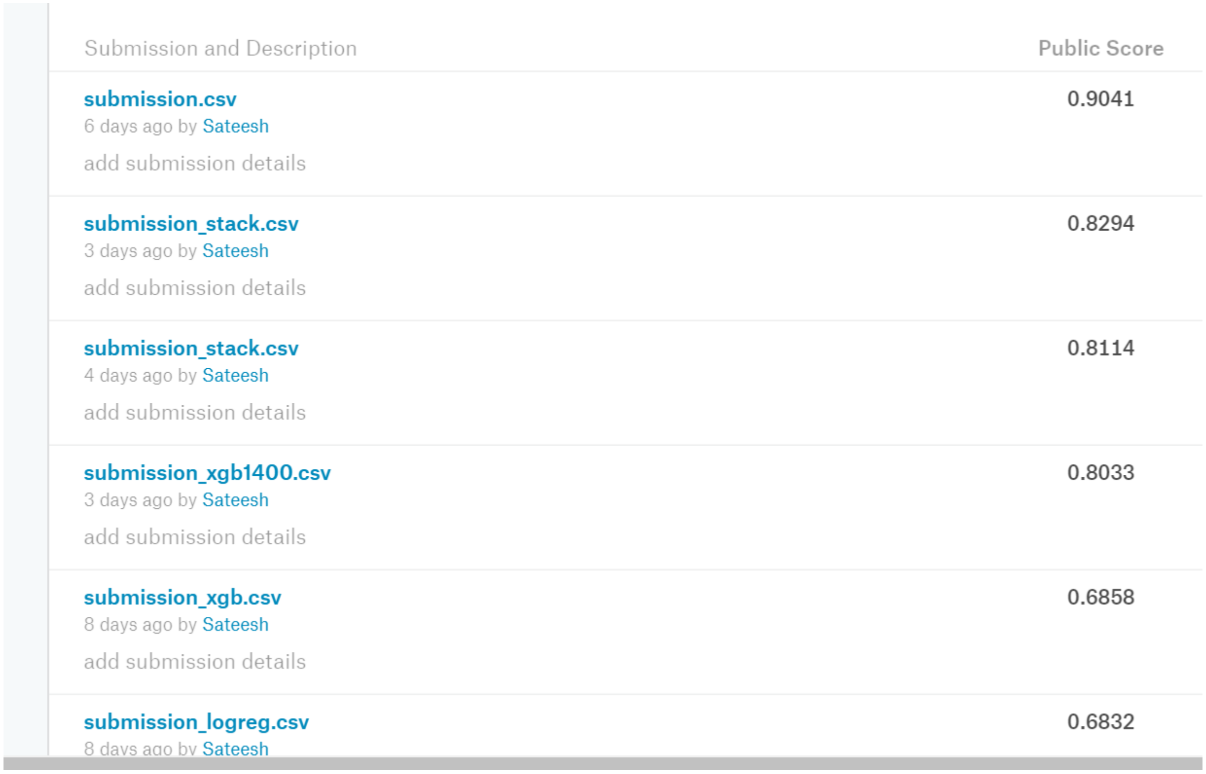
In [483]:
```
# Write predictions to csv
X_test_matrix = test_prepared.as_matrix()
preds = model.predict_proba(X_test_matrix)[:, 1]
sub_df['isFraud'] = preds
sub_df.to_csv(f"submission_stack.csv", index=False)
```

```
====================== Start of Level 0 ======================
1 estimators included in Level 0
====================== Start of Level 1 ======================
1 estimators included in Level 1
```

Kaggle Entry ! your best score 0.90418 . Keep trying!

In [3]:
```
Image('KaggleScore.PNG')
```

Out[3]:



Submission and Description	Public Score
submission.csv 6 days ago by Sateesh add submission details	0.9041
submission_stack.csv 3 days ago by Sateesh add submission details	0.8294
submission_stack.csv 4 days ago by Sateesh add submission details	0.8114
submission_xgb1400.csv 3 days ago by Sateesh add submission details	0.8033
submission_xgb.csv 8 days ago by Sateesh add submission details	0.6858
submission_logreg.csv 8 days ago by Sateesh	0.6832

Conclusion

Initial challenge, we have seen is preparing the data. Values as well as data atttributes are highly masked. It's not so easy to interpret or take subjective decision based on the data content.Its obvious that we will be facing a similar situation with any client. In our opinion, understand the data is highly important before solutioning the problem.

Tried multiple classification alogrithms, especially decision trees - bagging as well as boosting. Even though boosting model took more time to process. But, their results are more accurate. As part of model tunning, we have tunned hyperparameters - n_estimators, learning rate, max_depth. Using PystackNet, We have stacked the models { Level 0: 2 strong and one weak model, Level 1: weak model) based on their AUC metrics. AUC measure on train set and validation set gave good measure of each model as well as stack model performance. Finally, scored 0.9041 on test set.

Data cleaning and feature engineering has key role in model performance along with model selection and tunning. There is lot more to improve to reach the final score, 0.9557.

In [ ]: