In [ ]:

```
a = []
while(1):
    a.append(1)
```

In [ ]:

```
# Import the necessary libraries
import numpy as np
import pandas as pd
import os
import time
import warnings
import gc
gc.collect()
import os
from six.moves import urllib
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
warnings.filterwarnings('ignore')
%matplotlib inline
plt.style.use('seaborn')
from scipy.stats import norm, skew
from sklearn.preprocessing import StandardScaler
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: p
andas.util.testing is deprecated. Use the functions in the public API at pandas.testing i
nstead.
  import pandas.util.testing as tm
```

In [ ]:

```
#Add All the Models Libraries

# Scalers
from sklearn.utils import shuffle
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion

# Models

from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_log_error,mean_squared_error, r2_score,mean_abso
lute_error

from sklearn.model_selection import train_test_split #training and testing data split
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
from scipy.stats import reciprocal, uniform

from sklearn.model_selection import StratifiedKFold, RepeatedKFold

# Cross-validation
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
from sklearn.model_selection import cross_validate

# GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

#Common data processors
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn import feature_selection
```

```
from sklearn import model_selection
from sklearn import metrics
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils import check_array
from scipy import sparse
```

In [ ]:

```
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max
:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).m
ax:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(en
d_mem, 100 * (start_mem - end_mem) / start_mem))
    return df
```

Train Test split

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=94731898
9803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%
3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.co
m%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fww
w.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth
%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [ ]:

```
p_d = reduce_mem_usage(pd.read_csv('/content/drive/My Drive/hist_2.csv', index_col=0))
#print('Number of data points : ', elo_train.shape[0])
print('Number of data points : ', p_d.shape[0])
print('Number of features : ', p_d.shape[1])
print('Features : ', p_d.columns.values)
```

```
Mem. usage decreased to 26.57 Mb (47.7% reduction)
Number of data points :  201917
Number of features :  32
Features :  ['first_active_month' 'card_id' 'feature_1' 'feature_2' 'feature_3'
 'target' 'authorized_flag_x' 'city_id_x' 'category_1_x' 'installments_x'
 'category_3_x' 'merchant_category_id_x' 'merchant_id_x' 'month_lag_x'
```

```
 'category_3_x' 'merchant_category_id_x' 'merchant_id_x' 'month_lag_x'
 'purchase_amount_x' 'purchase_date_x' 'category_2_x' 'state_id_x'
 'subsector_id_x' 'authorized_flag_y' 'city_id_y' 'category_1_y'
 'installments_y' 'category_3_y' 'merchant_category_id_y' 'merchant_id_y'
 'month_lag_y' 'purchase_amount_y' 'purchase_date_y' 'category_2_y'
 'state_id_y' 'subsector_id_y']
```

In [ ]:

```
%time
p_d.isnull().sum()
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.68 µs
```

Out[ ]:

```
first_active_month       0
card_id                  0
feature_1                0
feature_2                0
feature_3                0
target                   0
authorized_flag_x        0
city_id_x                0
category_1_x             0
installments_x           0
category_3_x             0
merchant_category_id_x   0
merchant_id_x            0
month_lag_x              0
purchase_amount_x        0
purchase_date_x          0
category_2_x             0
state_id_x               0
subsector_id_x           0
authorized_flag_y        0
city_id_y                0
category_1_y             0
installments_y           0
category_3_y             0
merchant_category_id_y   0
merchant_id_y            0
month_lag_y              0
purchase_amount_y        0
purchase_date_y          0
category_2_y             0
state_id_y               0
subsector_id_y           0
dtype: int64
```

In [ ]:

```
y = p_d["target"].values
X = p_d.drop("target",axis = 1)
```

In [ ]:

```
# train test split
np.random.seed(10)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3)
```

Root Mean Square Error

We'll be using the root mean squared error as our evaluation metric:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N}}$$

$$\sqrt{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

In [ ]:

```
def root_mean_squared_error(y_true, y_pred):
    """Root mean squared error regression loss"""
    return np.sqrt(np.mean(np.square(y_true-y_pred)))
```

In [ ]:

```
root_mean_squared_error(np.mean(y_train), y_train)
```

Out[ ]:

```
3.818
```

OK, so our models should for sure be getting RMSE values lower than 3.887

To apply model on top of it ... Let us convert all the features either in to Numerical

Data pre-processing

Converting Boolean in to Numerical

In [ ]:

```
#converting boolean features in to Numerical

p_d['category_1_x'] = p_d['category_1_x'].map({'Y': 1, 'N': 0})
p_d['category_1_y'] = p_d['category_1_y'].map({'Y': 1, 'N': 0})

p_d['category_3_x'] = p_d['category_3_x'].map({'A':0, 'B':1, 'C':2})
p_d['category_3_y'] = p_d['category_3_y'].map({'A':0, 'B':1, 'C':2})
```

In [ ]:

```
#converting boolean features in to Numerical

X_train['category_1_x'] = X_train['category_1_x'].map({'Y': 1, 'N': 0})
X_test['category_1_x'] = X_test['category_1_x'].map({'Y': 1, 'N': 0})
X_train['category_1_y'] = X_train['category_1_y'].map({'Y': 1, 'N': 0})
X_test['category_1_y'] = X_test['category_1_y'].map({'Y': 1, 'N': 0})


X_train['category_3_x'] = X_train['category_3_x'].map({'A':0, 'B':1, 'C':2})
X_test['category_3_x'] = X_test['category_3_x'].map({'A':0, 'B':1, 'C':2})
X_train['category_3_y'] = X_train['category_3_y'].map({'A':0, 'B':1, 'C':2})
X_test['category_3_y'] = X_test['category_3_y'].map({'A':0, 'B':1, 'C':2})
```

Feature Engineering

After combining all the csv files, we got 32 features. But this 32 features are enough in predicting the model. But to predict better output, Feature engineering comes in rescue.

Since, all the features in this problem are in Numerical/Categorical. It is quite simple to go for a Feature Engineering.

So, I have gone for Aggregation technique for Numerical features.

In [ ]:

```
p_d['merch_purchase_date'] = pd.to_datetime(p_d['purchase_date_x'])
p_d['merch_weekofyear'] = p_d['merch_purchase_date'].dt.weekofyear
p_d['merch_month'] = p_d['merch_purchase_date'].dt.month
p_d['merch_day'] = p_d['merch_purchase_date'].dt.day
p_d['merch_weekday'] = p_d.merch_purchase_date.dt.weekday
p_d['merch_weekend'] = (p_d.merch_purchase_date.dt.weekday >=5).astype(int)
p_d['merch_hour'] = p_d['merch_purchase_date'].dt.hour
p_d['merch_month_diff'] = ((datetime.datetime.today() - p_d['merch_purchase_date']).dt.d
ays)//30
p_d['merch_month_diff'] += p_d['month_lag_x']

# additional features
p_d['merch_duration'] = p_d['purchase_amount_x']*p_d['merch_month_diff']
p_d['merch_amount_month_ratio'] = p_d['purchase_amount_x']/p_d['merch_month_diff']
p_d['merch_price'] = p_d['purchase_amount_x'] / p_d['installments_x']

gc.collect()
```

Out[ ]:

```
15
```

In [ ]:

```
X_train['merch_purchase_date'] = pd.to_datetime(X_train['purchase_date_x'])
X_train['merch_weekofyear'] = X_train['merch_purchase_date'].dt.weekofyear
X_train['merch_month'] = X_train['merch_purchase_date'].dt.month
X_train['merch_day'] = X_train['merch_purchase_date'].dt.day
X_train['merch_weekday'] = X_train.merch_purchase_date.dt.weekday
X_train['merch_weekend'] = (X_train.merch_purchase_date.dt.weekday >=5).astype(int)
X_train['merch_hour'] = X_train['merch_purchase_date'].dt.hour
X_train['merch_month_diff'] = ((datetime.datetime.today() - X_train['merch_purchase_date
']).dt.days)//30
X_train['merch_month_diff'] += X_train['month_lag_x']

# additional features
X_train['merch_duration'] = X_train['purchase_amount_x']*X_train['merch_month_diff']
X_train['merch_amount_month_ratio'] = X_train['purchase_amount_x']/X_train['merch_month_
diff']
X_train['merch_price'] = X_train['purchase_amount_x'] / X_train['installments_x']

X_test['merch_purchase_date'] = pd.to_datetime(X_test['purchase_date_x'])
X_test['merch_weekofyear'] = X_test['merch_purchase_date'].dt.weekofyear
X_test['merch_month'] = X_test['merch_purchase_date'].dt.month
X_test['merch_day'] = X_test['merch_purchase_date'].dt.day
X_test['merch_weekday'] = X_test.merch_purchase_date.dt.weekday
X_test['merch_weekend'] = (X_test.merch_purchase_date.dt.weekday >=5).astype(int)
X_test['merch_hour'] = X_test['merch_purchase_date'].dt.hour
X_test['merch_month_diff'] = ((datetime.datetime.today() - X_test['merch_purchase_date']
).dt.days)//30
X_test['merch_month_diff'] += X_test['month_lag_x']

# additional features
X_test['merch_duration'] = X_test['purchase_amount_x']*X_test['merch_month_diff']
X_test['merch_amount_month_ratio'] = X_test['purchase_amount_x']/X_test['merch_month_dif
f']
X_test['merch_price'] = X_test['purchase_amount_x'] / X_test['installments_x']

gc.collect()
```

Out[ ]:

```
154
```

In [ ]:

```
p_d['new_purchase_date'] = pd.to_datetime(p_d['purchase_date_y'])
p_d['new_weekofyear'] = p_d['new_purchase_date'].dt.weekofyear
p_d['new_month'] = p_d['new_purchase_date'].dt.month
p_d['new_day'] = p_d['new_purchase_date'].dt.day
p_d['new_weekday'] = p_d.new_purchase_date.dt.weekday
```

```
p_d['new_weekend'] = (p_d.new_purchase_date.dt.weekday >=5).astype(int)
p_d['new_hour'] = p_d['new_purchase_date'].dt.hour
p_d['new_month_diff'] = ((datetime.datetime.today() - p_d['new_purchase_date']).dt.days)
//30
p_d['new_month_diff'] += p_d['month_lag_y']

# additional features
p_d['new_duration'] = p_d['purchase_amount_y']*p_d['new_month_diff']
p_d['new_amount_month_ratio'] = p_d['purchase_amount_y']/p_d['new_month_diff']
p_d['new_price'] = p_d['purchase_amount_y'] / p_d['installments_y']
gc.collect()
```

Out[ ]:

```
77
```

In [ ]:

```
X_train['new_purchase_date'] = pd.to_datetime(X_train['purchase_date_y'])
X_train['new_weekofyear'] = X_train['new_purchase_date'].dt.weekofyear
X_train['new_month'] = X_train['new_purchase_date'].dt.month
X_train['new_day'] = X_train['new_purchase_date'].dt.day
X_train['new_weekday'] = X_train.new_purchase_date.dt.weekday
X_train['new_weekend'] = (X_train.new_purchase_date.dt.weekday >=5).astype(int)
X_train['new_hour'] = X_train['new_purchase_date'].dt.hour
X_train['new_month_diff'] = ((datetime.datetime.today() - X_train['new_purchase_date']).
dt.days)//30
X_train['new_month_diff'] += X_train['month_lag_y']

# additional features
X_train['new_duration'] = X_train['purchase_amount_y']*X_train['new_month_diff']
X_train['new_amount_month_ratio'] = X_train['purchase_amount_y']/X_train['new_month_diff
']
X_train['new_price'] = X_train['purchase_amount_y'] / X_train['installments_y']

X_test['new_purchase_date'] = pd.to_datetime(X_test['purchase_date_y'])
X_test['new_weekofyear'] = X_test['new_purchase_date'].dt.weekofyear
X_test['new_month'] = X_test['new_purchase_date'].dt.month
X_test['new_day'] = X_test['new_purchase_date'].dt.day
X_test['new_weekday'] = X_test.new_purchase_date.dt.weekday
X_test['new_weekend'] = (X_test.new_purchase_date.dt.weekday >=5).astype(int)
X_test['new_hour'] = X_test['new_purchase_date'].dt.hour
X_test['new_month_diff'] = ((datetime.datetime.today() - X_test['new_purchase_date']).dt
.days)//30
X_test['new_month_diff'] += X_test['month_lag_y']

# additional features
X_test['new_duration'] = X_test['purchase_amount_y']*X_test['new_month_diff']
X_test['new_amount_month_ratio'] = X_test['purchase_amount_y']/X_test['new_month_diff']
X_test['new_price'] = X_test['purchase_amount_y'] / X_test['installments_y']

gc.collect()
```

Out[ ]:

```
154
```

In [ ]:

```
#https://www.kaggle.com/chauhuynh/my-first-kernel-3-699
p_d['merch_purchase_date_max'] = pd.to_datetime(p_d['purchase_date_x'].max())
p_d['merch_purchase_date_min'] = pd.to_datetime(p_d['purchase_date_x'].min())
p_d['merch_purchase_date_diff'] = (p_d['merch_purchase_date_max'] - p_d['merch_purchase_d
ate_min']).dt.days
p_d['merch_purchase_date_uptonow'] = (datetime.datetime.today() - p_d['merch_purchase_dat
e_max']).dt.days
p_d['merch_purchase_date_uptomin'] = (datetime.datetime.today() - p_d['merch_purchase_dat
e_min']).dt.days

p_d['new_purchase_date_max'] = pd.to_datetime(p_d['purchase_date_y'].max())
p_d['new_purchase_date_min'] = pd.to_datetime(p_d['purchase_date_y'].min())
p_d['new_purchase_date_diff'] = (p_d['new_purchase_date_max'] - p_d['new_purchase_date_mi
```

```
n']).dt.days
p_d['new_purchase_date_uptonow'] = (datetime.datetime.today() - p_d['new_purchase_date_max']).dt.days
p_d['new_purchase_date_uptomin'] = (datetime.datetime.today() - p_d['new_purchase_date_min']).dt.days
```

In [ ]:

```
#https://www.kaggle.com/chauhuynh/my-first-kernel-3-699
X_train['merch_purchase_date_max'] = pd.to_datetime(X_train['purchase_date_x'].max())
X_train['merch_purchase_date_min'] = pd.to_datetime(X_train['purchase_date_x'].min())
X_train['merch_purchase_date_diff'] = (X_train['merch_purchase_date_max'] - X_train['merch_purchase_date_min']).dt.days
X_train['merch_purchase_date_uptonow'] = (datetime.datetime.today() - X_train['merch_purchase_date_max']).dt.days
X_train['merch_purchase_date_uptomin'] = (datetime.datetime.today() - X_train['merch_purchase_date_min']).dt.days


X_test['merch_purchase_date_max'] = pd.to_datetime(X_test['purchase_date_x'].max())
X_test['merch_purchase_date_min'] = pd.to_datetime(X_test['purchase_date_x'].min())
X_test['merch_purchase_date_diff'] = (X_test['merch_purchase_date_max'] - X_test['merch_purchase_date_min']).dt.days
X_test['merch_purchase_date_uptonow'] = (datetime.datetime.today() - X_test['merch_purchase_date_max']).dt.days
X_test['merch_purchase_date_uptomin'] = (datetime.datetime.today() - X_test['merch_purchase_date_min']).dt.days


X_train['new_purchase_date_max'] = pd.to_datetime(X_train['purchase_date_y'].max())
X_train['new_purchase_date_min'] = pd.to_datetime(X_train['purchase_date_y'].min())
X_train['new_purchase_date_diff'] = (X_train['new_purchase_date_max'] - X_train['new_purchase_date_min']).dt.days
X_train['new_purchase_date_uptonow'] = (datetime.datetime.today() - X_train['new_purchase_date_max']).dt.days
X_train['new_purchase_date_uptomin'] = (datetime.datetime.today() - X_train['new_purchase_date_min']).dt.days


X_test['new_purchase_date_max'] = pd.to_datetime(X_test['purchase_date_y'].max())
X_test['new_purchase_date_min'] = pd.to_datetime(X_test['purchase_date_y'].min())
X_test['new_purchase_date_diff'] = (X_test['new_purchase_date_max'] - X_test['new_purchase_date_min']).dt.days
X_test['new_purchase_date_uptonow'] = (datetime.datetime.today() - X_test['new_purchase_date_max']).dt.days
X_test['new_purchase_date_uptomin'] = (datetime.datetime.today() - X_test['new_purchase_date_min']).dt.days
```

In [ ]:

```
p_d = p_d.drop(['first_active_month', 'card_id', 'merchant_id_x', 'merchant_id_y', 'authorized_flag_x', 'authorized_flag_y', 'merch_purchase_date', 'new_purchase_date', 'purchase_date_x', 'merch_purchase_date_max', 'merch_purchase_date_min', 'purchase_date_y', 'new_purchase_date_max', 'new_purchase_date_min'], axis = 1)
```

In [ ]:

```
X_train = X_train.drop(['first_active_month', 'card_id', 'merchant_id_x', 'merchant_id_y', 'authorized_flag_x', 'authorized_flag_y', 'merch_purchase_date', 'new_purchase_date', 'purchase_date_x', 'merch_purchase_date_max', 'merch_purchase_date_min', 'purchase_date_y', 'new_purchase_date_max', 'new_purchase_date_min'], axis = 1)


X_test = X_test.drop(['first_active_month', 'card_id', 'merchant_id_x', 'merchant_id_y', 'authorized_flag_x', 'authorized_flag_y', 'merch_purchase_date', 'new_purchase_date', 'purchase_date_x', 'merch_purchase_date_max', 'merch_purchase_date_min', 'purchase_date_y', 'new_purchase_date_max', 'new_purchase_date_min'], axis = 1)
```

In [ ]:

```
p_d.shape, X_train.shape, X_test.shape
```

Out[ ]:

```
((201917, 50), (134611, 49), (67306, 49))
```

since, we have inf, -inf values in new features such as new_price, merch_price.we are converting this in to numeric values to apply regresion models on top of it.

In [ ]:

```
p_d = p_d.replace([np.inf, -np.inf], np.nan)
```

In [ ]:

```
X_train = X_train.replace([np.inf, -np.inf], np.nan)
X_test = X_test.replace([np.inf, -np.inf], np.nan)
```

In [ ]:

```
def nan_impute(df, col):
  p = df[col].value_counts(normalize=True)   # Series of probabilities
  m = df[col].isnull()

  np.random.seed(42)
  rand_fill = np.random.choice(p.index, size=m.sum(), p=p)

  df.loc[m, col] = rand_fill
```

In [ ]:

```
nan_impute(p_d, 'merch_price')
```

In [ ]:

```
nan_impute(X_train, 'merch_price')
nan_impute(X_test, 'merch_price')

nan_impute(X_train, 'new_price')
nan_impute(X_test, 'new_price')
```

In [ ]:

```
X_train
X_test
```

Out[ ]:

	card_id	feature_1	feature_2	feature_3	city_id_x	category_1_x	installments_x	category_3_x	merchant_category_id_
1117370	3539	3	3	1	261	0	1	1	70
147280	49761	4	1	0	166	0	2	2	8
753785	60741	5	1	1	-1	0	3	2	4
324272	10536	5	2	1	137	0	0	0	4
326356	62780	2	1	0	11	0	1	1	27
...	...	...	...	...	...	...	...	...	.
482634	33293	3	1	1	106	0	0	0	43
1087583	17130	3	1	1	149	0	0	0	70
679621	61542	2	2	0	69	0	0	0	87
497699	3222	4	1	0	158	0	1	1	27
1235709	16984	3	1	1	125	0	0	0	70

67306 rows × 52 columns

In [ ]:

```
#Saving the features in to pickle file
import pickle
with open('fe_train.pickle', "wb") as f:
    pickle.dump(X_train, f)
```

In [ ]:

```
#Saving the features in to pickle file
with open('fe_test.pickle', "wb") as file:
    pickle.dump(X_test, file)
```

In [ ]:

```
# capture all variables in a list
# except the target and the ID

train_vars = [var for var in X_train.columns]

# count number of variables
len(train_vars)
```

Out[ ]:

```
49
```

In [ ]:

```
# create scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

#  fit  the scaler to the train set
scaler.fit(X_train[train_vars])

# transform the train and test set
X_train[train_vars] = scaler.transform(X_train[train_vars])

X_test[train_vars] = scaler.transform(X_test[train_vars])
```

Applying Machine Learning models

1) Random Forest

Hyperparameter Tuning

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
#from scipy.stats import randint as sp_randint

param_dist = {'n_estimators': [10,50,100,150,160,200,300,350,400,500],
               'min_samples_split': [2,3,5,6,7,8],


              "max_depth":[None,10,20,40,60,80,100,120]

              }
regr1 = RandomForestRegressor()
regr1 = RandomizedSearchCV(regr1, param_distributions=param_dist,
                                   n_jobs=-1, scoring="neg_mean_squared_error", cv=3)

regr1.fit(X_train, y_train)
```

Out[ ]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=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=100,
                                                   n_jobs=None, oob_score=False,
                                                   random_state=None, verbose=0,
                                                   warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [None, 10, 20, 40, 60, 80,
                                                      100, 120],
                                        'min_samples_split': [2, 3, 5, 6, 7, 8],
                                        'n_estimators': [10, 50, 100, 150, 160,
                                                         200, 300, 350, 400,
                                                         500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
m = RandomForestRegressor(n_jobs=-1,min_samples_split=2,n_estimators=100,max_depth=None)
m.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=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=100, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = m.predict(X_train)
y_test_pred = m.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  2.5402640114019146

---------------------------------------------
Test RMSE :  1.2747932611460415
```

In [ ]:

```
from sklearn.externals import joblib
joblib.dump(m, 'elo_rf.pkl')
```

Out[ ]:

```
['elo_rf.pkl']
```

In [ ]:

```
joblib.load('elo_rf.pkl')
```

Out[ ]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=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=100, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

-There is a quite difference between Train and Test RMSE values, seems like an overfitting.

2) LightGBM

Hyperparameter Tuning

In [ ]:

```
from lightgbm import LGBMRegressor
from sklearn.model_selection import GridSearchCV
```

In [ ]:

```
from lightgbm import LGBMRegressor
from sklearn.model_selection import GridSearchCV

gridParams = {
    'learning_rate': [ 0.1,0.2,0.3,0.4,0.5],
    'n_estimators': [100,150, 200,250,300,400,500],
    'num_leaves': [20,30,63,80,100,120],
    'boosting_type' : ['gbdt', 'goss'],
    'max_depth' : [2,3,4,5,6,7,8]
}
lgbm_params ={'subsample': 0.9855232997390695, 'colsample_bytree': 0.5665320670155495, 'o
bjective': 'regression', 'eval_metric':'rmse'
        }
model = LGBMRegressor(**lgbm_params)
# Create the grid
grid = GridSearchCV(model, gridParams, verbose=1, cv=3, n_jobs=-1)
# Run the grid
grid.fit(X_train, y_train,
         eval_set = (X_test, y_test),

         early_stopping_rounds=100,
         verbose=True)
```

In [ ]:

```
print('Best parameters found by grid search are:', grid.best_params_)
print('Best score found by grid search is:', grid.best_score_)
```

```
Best parameters found by grid search are: {'boosting_type': 'gbdt', 'learning_rate': 0.1,
'max_depth': 4, 'n_estimators': 100, 'num_leaves': 20}
Best score found by grid search is: 0.03583999592568913
```

In [ ]:

```
lgbm_params ={'subsample': 0.9855232997390695, 'colsample_bytree': 0.5665320670155495, 'm
in_child_samples': 50, 'objective': 'regression','boosting_type': 'gbdt','learning_rate'
: 0.1,'max_depth': 4,'n_estimators': 100,'num_leaves': 20,
       }
```

```
model = LGBMRegressor(**lgbm_params)
model.fit(X_train, y_train,

         eval_set = (X_test, y_test),
         early_stopping_rounds=100,
         verbose=True)


preds2 = model.predict(X_test)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  0.6987800670863433

---------------------------------------------
Test RMSE :  0.699828943240799
```

In [ ]:

```
from sklearn.externals import joblib
joblib.dump(model, 'elo_lgb.pkl')
```

Out[ ]:

```
['elo_lgb.pkl']
```

LGBM, does pretty well job. Looks like no overfitting.

3) Ridge Regression

Hyperparameter Tuning

In [ ]:

```
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
#from sklearn import linear_model

from sklearn.model_selection import GridSearchCV

parameters = {"alpha":[0.00000001,0.00001,0.0001,0.001,0.01,0.1,0,1,10,100,1000,10000,10
0000]}
ridgeReg = Ridge(solver = "lsqr", fit_intercept=False)

lr_reg = GridSearchCV(ridgeReg,param_grid =parameters,n_jobs=-1)
lr_reg.fit(X_train, y_train)
```

Out[ ]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=False,
                             max_iter=None, normalize=False, random_state=None,
                             solver='lsqr', tol=0.001),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [1e-08, 1e-05, 0.0001, 0.001, 0.01, 0.1, 0, 1,
                                   10, 100, 1000, 10000, 100000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [ ]:

```
lr_reg.best_params_
```

Out[ ]:

```
{'alpha': 10}
```

In [ ]:

```
from sklearn.linear_model import Ridge
ridgeReg = Ridge(alpha=10000,solver = "lsqr", fit_intercept=False )
ridgeReg.fit(X_train, y_train)
#y_pred = ridgeReg.predict(X_test)
```

Out[ ]:

```
Ridge(alpha=10000, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='lsqr', tol=0.001)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = ridgeReg.predict(X_train)
y_test_pred = ridgeReg.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  0.7229935013120617

---------------------------------------------
Test RMSE :  0.7254309954273938
```

In [ ]:

```
from sklearn.externals import joblib
joblib.dump(ridgeReg, 'elo_rr.pkl')
```

Out[ ]:

```
['elo_rr.pkl']
```

Ridge Regression has a very less RMSE value as compared to other models.But actually it is overfitting.

4)SGD Regressor

Hyper Parameter Tuning

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDRegressor, Ridge
```

In [ ]:

```
parameters = {"alpha":[0.00000001,0.00001,0.0001,0.001,0.01,0.1,0,1,10,100,1000,10000,10
0000],
             "l1_ratio" : [0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
             }
model = SGDRegressor(
                         loss='squared_loss',
                     learning_rate='invscaling',
                         max_iter=200,
                         penalty='l2',
                         fit_intercept=False
                         )
lr_reg = GridSearchCV(model,param_grid =parameters,n_jobs=-1)
lr_reg.fit(X_train, y_train)
```

Out[ ]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDRegressor(alpha=0.0001, average=False,
                                    early_stopping=False, epsilon=0.1,
                                    eta0=0.01, fit_intercept=False,
                                    l1_ratio=0.15, learning_rate='invscaling',
                                    loss='squared_loss', max_iter=200,
                                    n_iter_no_change=5, penalty='l2',
                                    power_t=0.25, random_state=None,
                                    shuffle=True, tol=0.001,
                                    validation_fraction=0.1, verbose=0,
                                    warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [1e-08, 1e-05, 0.0001, 0.001, 0.01, 0.1, 0, 1,
                                   10, 100, 1000, 10000, 100000],
                         'l1_ratio': [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [ ]:

```
lr_reg.best_params_
```

Out[ ]:

```
{'alpha': 0, 'l1_ratio': 0.9}
```

In [ ]:

```
lr_reg = SGDRegressor(penalty='l2',
                         loss='squared_loss',
                     learning_rate='invscaling',
                         max_iter=200,

                         fit_intercept=False,
                     alpha=0.0001,
                     l1_ratio=0.9
                         )
lr_reg.fit(X_train, y_train)
```

Out[ ]:

```
SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=False, l1_ratio=0.9,
             learning_rate='invscaling', loss='squared_loss', max_iter=200,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = lr_reg.predict(X_train)
y_test_pred = lr_reg.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  0.5149267557847675

---------------------------------------------
Test RMSE :  0.5252051052579404
```

In [ ]:

```
from sklearn.externals import joblib
joblib.dump(lr_reg, 'elo_sgd.pkl')
```

Out[ ]:

```
['elo_sgd.pkl']
```

MOdel looks overfitting.

5) XgBoost

Hyperparameter Tuning

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
from xgboost.sklearn import XGBRegressor

parameters2 = {'n_estimators': [5,10,50,100,200,500,1000] ,
              'max_depth' :  [2,3,4,5,6,7,8,9,10]}


XGB_rg = xgb.XGBRegressor(random_state=11,class_weight='balanced')

XGB_rg2=RandomizedSearchCV(XGB_rg ,param_distributions = parameters2, scoring="neg_mean_
squared_error", cv=5)
XGB_rg2.fit(X_train,y_train)
```

```
[06:39:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:40:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:40:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:40:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:41:20] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:41:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:44:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:46:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
```

```
[06:49:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:51:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:54:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:54:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:55:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:55:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:56:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:56:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:56:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:56:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:56:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:57:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:57:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:57:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:58:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:58:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[06:59:28] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:00:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:01:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:03:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:04:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:06:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:08:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:08:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:08:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:08:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:08:44] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:08:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:17:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:26:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:35:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:44:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:53:28] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:54:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:55:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:56:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
```

```
[07:57:44] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[07:58:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[08:09:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[08:19:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[08:30:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[08:40:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[08:50:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
```

Out[ ]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          class_weight='balanced',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=1, nthread=None,
                                          objective='reg:linear',
                                          r...1, reg_alpha=0,
                                          reg_lambda=1, scale_pos_weight=1,
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9,
                                                       10],
                                        'n_estimators': [5, 10, 50, 100, 200,
                                                         500, 1000]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [ ]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
tml
a2=XGB_rg2.best_params_['n_estimators']
p2 = XGB_rg2.best_params_['max_depth']
print(XGB_rg2.best_score_)
print(a2)
print(p2)
```

```
-14.049633407592774
100
4
```

In [ ]:

```
# initialize Our first XGBoost model...
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
first_xgb = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=50,
max_depth=3)
first_xgb.fit(X_train,y_train)
```

```
[10:34:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
```

Out[ ]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
```

```
             max_depth=3, min_child_weight=1, missing=None, n_estimators=50,
             n_jobs=13, nthread=None, objective='reg:linear', random_state=15,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=False, subsample=1, verbosity=1)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = XGB_rg2.predict(X_train)
y_test_pred = XGB_rg2.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  0.73648345

---------------------------------------------
Test RMSE :  0.7351688
```

In [ ]:

```
from sklearn.externals import joblib
joblib.dump(first_xgb, 'elo_xgb.pkl')
```

Out[ ]:

```
['elo_xgb.pkl']
```

Even XGBoost does pretty well job. Looks like ther is no overfitting. But we prefer LGBM over Xgboost becuase of it's fast computation and does extremely well in predicting the Traget variables.

Lasso Regression

In [ ]:

```
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [ ]:

```
#from sklearn.linear_model import Ridge
lassoReg = Lasso(alpha=0.005)
lassoReg.fit(X_train, y_train)
```

Out[ ]:

```
Lasso(alpha=0.005, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = lassoReg.predict(X_train)
y_test_pred = lassoReg.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
```

```
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  0.4225996007596389

---------------------------------------------
Test RMSE :  0.4207608311641669
```

Selecting the top contributing features from the model because having many features will lead to clutter during productionisation.

In [ ]:

```
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel

sel_ = SelectFromModel(Lasso(alpha=0.005, random_state=0))

sel_.fit(X_train, y_train)
```

Out[ ]:

```
SelectFromModel(estimator=Lasso(alpha=0.005, copy_X=True, fit_intercept=True,
                                max_iter=1000, normalize=False, positive=False,
                                precompute=False, random_state=0,
                                selection='cyclic', tol=0.0001,
                                warm_start=False),
                max_features=None, norm_order=1, prefit=False, threshold=None)
```

In [ ]:

```
# let's print the number of total and selected features

# this is how we can make a list of the selected features
selected_feats = X_train.columns[(sel_.get_support())]

# let's print some stats
print('total features: {}'.format((X_train.shape[1])))
print('selected features: {}'.format(len(selected_feats)))
print('features with coefficients shrank to zero: {}'.format(
    np.sum(sel_.estimator_.coef_ == 0)))
```

```
total features: 49
selected features: 17
features with coefficients shrank to zero: 32
```

In [ ]:

```
# print the selected features
selected_feats
```

Out[ ]:

```
Index(['feature_2', 'feature_3', 'category_1_x', 'category_3_x', 'month_lag_x',
       'category_2_x', 'category_1_y', 'category_3_y', 'month_lag_y',
       'state_id_y', 'subsector_id_y', 'merch_month', 'merch_day',
       'merch_month_diff', 'new_month', 'new_day', 'new_weekday'],
      dtype='object')
```

In [ ]:

```
selected_feats = X_train.columns[(sel_.estimator_.coef_ != 0).ravel().tolist()]

selected_feats
```

Out[ ]:

```
Index(['feature_2', 'feature_3', 'category_1_x', 'category_3_x', 'month_lag_x',
       'category_2_x', 'category_1_y', 'category_3_y', 'month_lag_y',
       'state_id_y', 'subsector_id_y', 'merch_month', 'merch_day',
       'merch_month_diff', 'new_month', 'new_day', 'new_weekday'],
      dtype='object')
```

In [ ]:

```
pd.Series(selected_feats).to_csv('selected_features.csv', index=False)
```

In [ ]:

```
features = pd.read_csv('selected_features.csv')
features = features['0'].to_list()
```

In [ ]:

```
# reduce the train and test set to the selected features

X_train = X_train[features]
X_test = X_test[features]
```

In [ ]:

```
# set up the model
# remember to set the random_state / seed

lin_model = Lasso(alpha=0.005, random_state=0)

# train the model

lin_model.fit(X_train, y_train)
```

Out[ ]:

```
Lasso(alpha=0.005, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=0,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [ ]:

```
#Calculating y_train_pred and y_test_pred
y_train_pred = lin_model.predict(X_train)
y_test_pred = lin_model.predict(X_test)
```

In [ ]:

```
#Calculating rsme and mape scores by using the utility function
rmse_train = root_mean_squared_error(np.mean(y_train), y_train_pred)
rmse_test = root_mean_squared_error(np.mean(y_test), y_test_pred)
```

In [ ]:

```
print('Train RMSE : ', rmse_train)
print('\n'+'-'*45)
print('Test RMSE : ', rmse_test)
```

```
Train RMSE :  0.4229866704932273

---------------------------------------------
Test RMSE :  0.42121332169034087
```

Saving Machine Learning Model : Serialization & Deserialization

In [ ]:

```
# we persist the model for future use
from sklearn.externals import joblib
```

```
joblib.dump(lin_model, 'lasso_regression.pkl')
```

Out[ ]:

```
['lasso_regression.pkl']
```

6) Ensemble Model

In [ ]:

```
from sklearn.ensemble import StackingRegressor
vstack = StackingRegressor(estimators=[('rf', m), ('lgb', model), ('xgb', first_xgb), ('
sgd', lr_reg), ('rr', ridgeReg), ('lr',lassoReg)])
vstack.fit(X_train, y_train)

print("RMSE (train) on the StackedRegressor :", root_mean_squared_error(np.mean(y_train),
vstack.predict(X_train)))
print("RMSE (test) on the StackedRegressor :", root_mean_squared_error(np.mean(y_test), v
stack.predict(X_test)))
```

```
RMSE (train) on the StackedRegressor : 0.46799606908291375
RMSE (test) on the StackedRegressor : 0.47390167441968317
```

In [ ]:

```
from sklearn.externals import joblib
joblib.dump(vstack, 'elo_ensemble.pkl')
```

Out[ ]:

```
['elo_ensemble.pkl']
```

In [1]:

```
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names= ("Model", "Test- MSE")
tb.add_row(["Randomised Model", "3.887"])
tb.add_row(["Random Forest", "1.274"])
tb.add_row(["SGD Regression", "0.525"])
tb.add_row(["Ridge Regression", "0.725"])
tb.add_row(["XGBoost", "0.735",])
tb.add_row(["LightGBM", "0.699"])
tb.add_row(["Lasso regression ", "0.421"])
tb.add_row(["Ensemble", "0.473"])
print(tb.get_string(titles = "Regression Models- Observations"))
#print(tb)
```

```
+-------------------+-----------+
|       Model       | Test- MSE |
+-------------------+-----------+
|  Randomised Model |   3.887   |
|   Random Forest   |   1.274   |
|   SGD Regression  |   0.525   |
|  Ridge Regression |   0.725   |
|      XGBoost      |   0.735   |
|      LightGBM     |   0.699   |
| Lasso regression  |   0.421   |
|      Ensemble     |   0.473   |
+-------------------+-----------+
```

Since, the best model found out from the above is "Lasso Regression". Now, we deploy this final model in to production.