Final submission

In [ ]:

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /opt/conda/lib/python3.
7/site-packages (2.2.2)
Requirement already satisfied: statsmodels>=0.9.0 in /opt/conda/lib/python3.
7/site-packages (from category_encoders) (0.11.1)
Requirement already satisfied: numpy>=1.14.0 in /opt/conda/lib/python3.7/sit
e-packages (from category_encoders) (1.18.1)
Requirement already satisfied: scikit-learn>=0.20.0 in /opt/conda/lib/python
3.7/site-packages (from category_encoders) (0.23.1)
Requirement already satisfied: patsy>=0.5.1 in /opt/conda/lib/python3.7/site
-packages (from category_encoders) (0.5.1)
Requirement already satisfied: scipy>=1.0.0 in /opt/conda/lib/python3.7/site
-packages (from category_encoders) (1.4.1)
Requirement already satisfied: pandas>=0.21.1 in /opt/conda/lib/python3.7/si
te-packages (from category_encoders) (1.0.4)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site
-packages (from scikit-learn>=0.20.0->category_encoders) (0.15.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python
3.7/site-packages (from scikit-learn>=0.20.0->category_encoders) (2.1.0)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages
(from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/site
-packages (from pandas>=0.21.1->category_encoders) (2020.1)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/conda/lib/pyth
on3.7/site-packages (from pandas>=0.21.1->category_encoders) (2.8.1)
```

In [ ]:

```
import pandas as pd
import numpy as np
import datetime
from datetime import date
from datetime import timedelta
import calendar
import math
import time
from tqdm import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
import xgboost as xgb
from joblib import dump, load
from sklearn import preprocessing
import category_encoders as ce
```

In [ ]:

```
#Helper Functions
def cat_encoding(cat_data, category):
  '''
  This function takes a df and the category and generate
  binary encoded vectors for the same
  '''
  encoder = ce.BinaryEncoder()
  return encoder.fit_transform(cat_data[category]).values

def generate_cat_features(sales_data):
  '''
  This function uses cat_encoding function and does binary encoding for all the catego
  '''
  items_df = pd.read_csv('items.csv')
  stores_df = pd.read_csv('stores.csv')

  class_family_df = pd.DataFrame(sales_data['item_nbr']).merge(items_df[['item_nbr', '
  class_family_df['class'] = class_family_df['class'].astype('str')
  class_family_df['item_nbr'] = class_family_df['item_nbr'].astype('str')

  store_detail_df = pd.DataFrame(sales_data['store_nbr']).merge(stores_df[['store_nbr'
  store_detail_df['store_nbr'] = store_detail_df['store_nbr'].astype('str')
  store_detail_df['cluster'] = store_detail_df['cluster'].astype('str')

  class_array = cat_encoding(class_family_df, 'class')
  family_array = cat_encoding(class_family_df, 'family')
  item_array = cat_encoding(class_family_df, 'item_nbr')


  store_array = cat_encoding(store_detail_df, 'store_nbr')
  store_state_array = cat_encoding(store_detail_df, 'state')
  store_city_array = cat_encoding(store_detail_df, 'city')
  store_type_array = cat_encoding(store_detail_df, 'type')
  store_cluster_array = cat_encoding(store_detail_df, 'cluster')

  return class_array, family_array, item_array, store_array, store_state_array, store_

def get_data(data, dt_end, days, period, freq='D'):
  '''
  This function gives us the selected columns based on a range of dates passed.
  '''
  return data[[str(col)[0:10] for col in pd.date_range(dt_end - datetime.timedelta(day

def average(data):
  '''
  Here we are calculating simple average
  '''
  return np.mean(data, axis = 1)

def weighted_moving_average(data):
  '''
  This function computes weighted moving average,
  higher weights are given to recent observations.
  '''
  data = data.values
  weight_len = data.shape[1]
  denom = (weight_len *(weight_len + 1))/2
  weights = [i+1/denom for i in range(weight_len)]
  data = average(data * weights)
  return data

def feature_engg_sales(data, end_date, prefix):
  '''
  This function generates feature dictionary for train, cv, test
  Features generated are:
  moving average, weighted moving average, standard deviation observed,
  moving average of DOW, weighted moving average of DOW, having total sales day,
  last sales day in n days, first sales day in n days
  '''
  days_list = [3, 7, 16, 30, 60, 120] # These are the list of days used for extracting
  #feature_dict = {}
  feature_dict = {'{}_average_{}_days'.format(prefix, days): average(get_data(data, en
  feature_dict.update({'{}_WMA_{}_days'.format(prefix, days): weighted_moving_average(
  feature_dict.update({'{}_std_{}_days'.format(prefix, days) : get_data(data, end_date
  feature_dict.update({'{}_6avgdow_{}_days'.format(prefix, day) : get_data(data, end_d
  feature_dict.update({'{}_20avgdow_{}_days'.format(prefix, day) : get_data(data, end_
  feature_dict.update({'{}_6WMAdow_{}_days'.format(prefix, day) : weighted_moving_aver
  feature_dict.update({'{}_20WMAdow_{}_days'.format(prefix, day) : weighted_moving_ave
  feature_dict.update({'{}_has_sale_day_{}'.format(prefix, days) : (get_data(data, end
  feature_dict.update({'{}_last_has_sale_day_{}'.format(prefix, days) : days - ((get_d
  feature_dict.update({'{}_first_has_sale_day_{}'.format(prefix, days) : ((get_data(da

  return feature_dict

def feature_engg_promo(data, class_array, family_array, item_array, store_array, store
    '''
    This function uses promo information and categorical array to create features
    features created are---
    promo: total_promo, future promo information, promo days in 15 days, last promo in
    categorical: class, item, store, family, city, state, clsuter, type
    '''
    days_list = [16, 30, 60, 120]
    feature_dict = {'{}_totalpromo_{}_days'.format(prefix, days) : get_data(data, end_
    feature_dict.update({'{}_totalpromoafter_{}_days'.format(prefix, days) : get_data(
    feature_dict.update({'{}_promo_{}_day'.format(prefix, abs(day - 1)): get_data(data
    feature_dict.update({'promo_day_in_15_days' : (get_data(data, end_date + timedelta
    feature_dict.update({'last_promo_day_in_15_days' : 15 - ((get_data(data, end_date 
    feature_dict.update({'firt_promo_day_in_15_days' : ((get_data(data, end_date + tim
    feature_dict.update({'class_{}'.format(i+1) : class_array[:, i] for i in range(cla
    feature_dict.update({'item_{}'.format(i+1) : item_array[:, i] for i in range(item_
    feature_dict.update({'store_{}'.format(i+1) : store_array[:, i] for i in range(sto
    feature_dict.update({'family_{}'.format(i+1) : family_array[:, i] for i in range(f
    feature_dict.update({'city_{}'.format(i+1) : store_city_array[:, i] for i in range
    feature_dict.update({'state_{}'.format(i+1) : store_state_array[:, i] for i in ran
    feature_dict.update({'cluster_{}'.format(i+1) : store_cluster_array[:, i] for i in
    feature_dict.update({'type_{}'.format(i+1) : store_type_array[:, i] for i in range
    feature_dict.update({'perishable' : class_family_df['perishable'].values})

    return feature_dict
```

In [ ]:

```
def final_fun_1(X):
  '''
  This function takes raw input, generate features using the raw input and make predict
  '''
  print('Generating sales and promo data for feature engg')
  X.loc[(X.unit_sales<0),'unit_sales'] = 0
  X['unit_sales'] =  X['unit_sales'].apply(lambda x : np.log1p(x))
  X = X.replace(to_replace = [False, True], value = [0, 1])

  sales_data = X.set_index(["store_nbr", "item_nbr", "date"])[["unit_sales"]].unstack(
  sales_data.columns = sales_data.columns.get_level_values(1)
  sales_data = sales_data.reset_index()

  train_promo = X.set_index(["store_nbr", "item_nbr", "date"])[["onpromotion"]].unstac
  train_promo.columns = train_promo.columns.get_level_values(1)

  test = pd.read_csv('test.csv')
  test = test.replace(to_replace = [False, True], value = [0, 1])

  test_promo = test.set_index(['store_nbr', 'item_nbr', 'date'])[["onpromotion"]].unst
  test_promo.columns = test_promo.columns.get_level_values(1)
  test_promo = test_promo.reindex(train_promo.index).fillna(0)

  promo_data = pd.concat([train_promo, test_promo], axis=1)
  promo_data = promo_data.reset_index()
  del test, train_promo, test_promo
  print('Data Collected!!!')
  print('Shape of sales and promo data is: {} and {}'.format(sales_data.shape, promo_d

  print('Generating categorical variables features')
  class_array, family_array, item_array, store_array, store_state_array, store_city_ar
  print('Categorical variables features generated')

  print('Extracting features for training using sales information')
  x_lst, y_lst = [], []
  num_of_intervals = 8
  dates = [date(2017, 5, 31) + timedelta(days=7 * interval) for interval in range(num_
  for train_date in dates:
    train_dict = feature_engg_sales(sales_data, train_date,'item_store')
    x_lst.append(pd.DataFrame(train_dict, index = [i for i in range(len(list(train_dic
    y_lst.append(sales_data[[str(col)[0:10] for col in pd.date_range(train_date, perio

  train_item_store_x = pd.concat(x_lst, axis=0)
  train_y = np.concatenate(y_lst, axis=0)
  del x_lst, y_lst
  #print(train_item_store_x.shape, train_y.shape)

  print('Extracting features for training using promo information')
  x_lst = []
  num_of_intervals = 8
  dates = [date(2017, 5, 31) + timedelta(days=7 * interval) for interval in range(num_
  for train_date in dates:
    train_dict = feature_engg_promo(promo_data, class_array, family_array, item_array,
    x_lst.append(pd.DataFrame(train_dict, index = [i for i in range(len(list(train_dic

  train_item_store_x1 = pd.concat(x_lst, axis=0)
  del x_lst
  #print(train_item_store_x1.shape)
  train_x = train_item_store_x.reset_index(drop = True).merge(train_item_store_x1.rese
  del train_item_store_x, train_item_store_x1
  [train_x[col].update((train_x[col] - train_x[col].min()) / (train_x[col].max() - tra
  print('Shape of train_x and corresponding train_y is {} & {}'.format(train_x.shape, 

  print('Extracting features for prediction on test data using sales information')
  test_date = date(2017, 8, 16)
  test_dict = feature_engg_sales(sales_data, test_date, 'item_store')
  test_item_store_x = pd.DataFrame(test_dict, index = [i for i in range(len(list(test_

  print('Extracting features for prediction on test data using promo information')
  test_dict = feature_engg_promo(promo_data, class_array, family_array, item_array, st
  test_item_store_x1 = pd.DataFrame(test_dict, index = [i for i in range(len(list(test_
  test_x = test_item_store_x.reset_index(drop = True).merge(test_item_store_x1.reset_i
  [test_x[col].update((test_x[col] - test_x[col].min()) / (test_x[col].max() - test_x[
  print('Shape of test_x is {}'.format(test_x.shape))

  print('Making predictions using the pre trained model')
  test_pred = []
  dtest = xgb.DMatrix(test_x)
  for i in range(16):
    #print('Generating results for forecasting step{}'.format(i+1))
    model = xgb.Booster()
    filename = 'step{}_model'.format(i+1)
    model.load_model(filename)
    test_pred.append(model.predict(dtest))

  print('Prediction done on test data... generating final output')
  y_test = np.array(test_pred).transpose()
  pred_df = pd.DataFrame(y_test, columns = pd.date_range('2017-08-16', periods = 16))
  pred_df = sales_data[['item_nbr', 'store_nbr']].merge(pred_df, left_index=True, righ
  pred_df = pred_df.melt(id_vars=['item_nbr', 'store_nbr'], var_name='date', value_nam
  pred_df['unit_sales'] = pred_df['unit_sales'].apply(lambda x : np.expm1(x))
  print('Prediction df generated, loading test file and merging results with test file
  test_df = pd.read_csv('test.csv')
  test_df['date'] = pd.to_datetime(test_df['date'])
  test_df = test_df.merge(pred_df[['item_nbr', 'store_nbr', 'date', 'unit_sales']], on
  test_df['unit_sales'] = test_df['unit_sales'].clip(lower = 0)
  test_df = test_df.fillna(0)

  return test_df
```

In [ ]:

```
def final_fun_2(X):
  '''
  This function takes raw input, generate features using the raw input and generate sc
  and the predicted one by the model.
  '''
  print('Generating sales and promo data for feature engg')
  X.loc[(X.unit_sales<0),'unit_sales'] = 0
  X['unit_sales'] =  X['unit_sales'].apply(lambda x : np.log1p(x))
  X = X.replace(to_replace = [False, True], value = [0, 1])

  sales_data = X.set_index(["store_nbr", "item_nbr", "date"])[["unit_sales"]].unstack(
  sales_data.columns = sales_data.columns.get_level_values(1)
  sales_data = sales_data.reset_index()

  train_promo = X.set_index(["store_nbr", "item_nbr", "date"])[["onpromotion"]].unstac
  train_promo.columns = train_promo.columns.get_level_values(1)

  test = pd.read_csv('test.csv')
  test = test.replace(to_replace = [False, True], value = [0, 1])

  test_promo = test.set_index(['store_nbr', 'item_nbr', 'date'])[["onpromotion"]].unst
  test_promo.columns = test_promo.columns.get_level_values(1)
  test_promo = test_promo.reindex(train_promo.index).fillna(0)

  promo_data = pd.concat([train_promo, test_promo], axis=1)
  promo_data = promo_data.reset_index()
  del test, train_promo, test_promo
  print('Data Collected!!!')
  print('Shape of sales and promo data is: {} and {}'.format(sales_data.shape, promo_d

  print('Generating categorical variables features')
  class_array, family_array, item_array, store_array, store_state_array, store_city_ar
  print('Categorical variables features generated')

  print('Extracting features for training using sales information')
  x_lst, y_lst = [], []
  num_of_intervals = 8
  dates = [date(2017, 5, 31) + timedelta(days=7 * interval) for interval in range(num_
  for train_date in dates:
    train_dict = feature_engg_sales(sales_data, train_date,'item_store')
    x_lst.append(pd.DataFrame(train_dict, index = [i for i in range(len(list(train_dic
    y_lst.append(sales_data[[str(col)[0:10] for col in pd.date_range(train_date, perio

  train_item_store_x = pd.concat(x_lst, axis=0)
  train_y = np.concatenate(y_lst, axis=0)
  del x_lst, y_lst
  #print(train_item_store_x.shape, train_y.shape)

  print('Extracting features for training using promo information')
  x_lst = []
  num_of_intervals = 8
  dates = [date(2017, 5, 31) + timedelta(days=7 * interval) for interval in range(num_
  for train_date in dates:
    train_dict = feature_engg_promo(promo_data, class_array, family_array, item_array,
    x_lst.append(pd.DataFrame(train_dict, index = [i for i in range(len(list(train_dic

  train_item_store_x1 = pd.concat(x_lst, axis=0)
  del x_lst
  #print(train_item_store_x1.shape)
  train_x = train_item_store_x.reset_index(drop = True).merge(train_item_store_x1.rese
  del train_item_store_x, train_item_store_x1
  [train_x[col].update((train_x[col] - train_x[col].min()) / (train_x[col].max() - tra
  print('Shape of train_x and corresponding train_y is {} & {}'.format(train_x.shape, 

  print('Extracting features for prediction on data using sales information')
  cv_date = date(2017, 7, 26)
  cv_dict = feature_engg_sales(sales_data, cv_date, 'item_store')
  cv_item_store_x = pd.DataFrame(cv_dict, index = [i for i in range(len(list(cv_dict.v

  print('Extracting features for prediction on data using promo information')
  cv_dict = feature_engg_promo(promo_data, class_array, family_array, item_array, stor
  cv_item_store_x1 = pd.DataFrame(cv_dict, index = [i for i in range(len(list(cv_dict.
  cv_x = cv_item_store_x.reset_index(drop = True).merge(cv_item_store_x1.reset_index(d
  [cv_x[col].update((cv_x[col] - cv_x[col].min()) / (cv_x[col].max() - cv_x[col].min()
  print('Shape of data on which we will predict is {}'.format(cv_x.shape))
  print('Generating true labels for the data....')
  cv_y = sales_data[[str(col)[0:10] for col in pd.date_range(cv_date, periods = 16)]].

  print('Making predictions using the pre trained model')
  cv_pred = []
  dcv = xgb.DMatrix(cv_x)
  for i in range(16):
   # print('Generating results for forecasting step{}'.format(i+1))
    model = xgb.Booster()
    filename = 'step{}_model'.format(i+1)
    model.load_model(filename)
    cv_pred.append(model.predict(dcv))

  print('Predition done, calculating Normalized Weighted Root Mean Squared Log Error!!
  items_df = pd.read_csv('items.csv')
  cv_weights = pd.DataFrame(sales_data['item_nbr']).merge(items_df[['item_nbr', 'peris
  cv_yhat = np.array(cv_pred).transpose()
  log_error = (np.log1p(cv_yhat) - np.log1p(cv_y)) ** 2
  error = log_error.sum(axis = 1) * cv_weights
  rmsle = np.sqrt(error.sum() / cv_weights.sum())

  return rmsle
```

In [ ]:

```
def flow():
  print('Calling Function 1 which will return predictions on test file!!!!')
  print('*'*75)
  print('Loading raw data!!!')
  train_df = pd.read_csv('train.csv', skiprows=range(1, 101688780))
  predictions = final_fun_1(train_df)
  print(predictions.head())
  predictions[['id', 'unit_sales']].to_csv('final_submission.csv', index = False)
  print('\n\n')
  print('Calling Function 2 which will return NWRMSLE!!!!')
  print('*'*75)
  print('Loading raw data!!!')
  train_df = pd.read_csv('train.csv', skiprows=range(1, 101688780))
  score = final_fun_2(train_df)
  print('score returned is: {}'.format(score))
  print('*'*75)
```

In [ ]:

```python
if __name__ == '__main__':
    flow()
```

```
Calling Function 1 which will return predictions on test file!!!!
***************************************************************************
Loading raw data!!!
Generating sales and promo data for feature engg
Data Collected!!!
Shape of sales and promo data is: (167515, 229) and (167515, 245)
Generating categorical variables features
Categorical variables features generated
Extracting features for training using sales information
Extracting features for training using promo information
Shape of train_x and corresponding train_y is (1340120, 149) & (1340120, 16)
Extracting features for prediction on test data using sales information
Extracting features for prediction on test data using promo information
Shape of test_x is (167515, 149)
Making predictions using the pre trained model
Prediction done on test data... generating final output
Prediction df generated, loading test file and merging results with test fil
e
          id       date  store_nbr  item_nbr  onpromotion  unit_sales
0  125497040 2017-08-16          1     96995        False    0.236070
1  125497041 2017-08-16          1     99197        False    0.375312
2  125497042 2017-08-16          1    103501        False    0.000000
3  125497043 2017-08-16          1    103520        False    1.205339
4  125497044 2017-08-16          1    103665        False    2.255238



Calling Function 2 which will return NWRMSLE!!!!
***************************************************************************
Loading raw data!!!
Generating sales and promo data for feature engg
Data Collected!!!
Shape of sales and promo data is: (167515, 229) and (167515, 245)
Generating categorical variables features
Categorical variables features generated
Extracting features for training using sales information
Extracting features for training using promo information
Shape of train_x and corresponding train_y is (1340120, 149) & (1340120, 16)
Extracting features for prediction on data using sales information
Extracting features for prediction on data using promo information
Shape of data on which we will predict is (167515, 149)
Generating true labels for the data....
Making predictions using the pre trained model
Predition done, calculating Normalized Weighted Root Mean Squared Log Erro
r!!!
score returned is: 0.5208827097261451
***************************************************************************
```

In [ ]:

```
 
```