Exetreme Boosting Model and Predictions

In [75]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot
from xgboost import XGBRegressor
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
import warnings
warnings.filterwarnings('ignore')
import random
```

In [76]:

```
# Generate n-s ramdom numbers in the range (2,n)
# where n is our train dataset size
n = 41390267
s = 100000
skip = sorted(random.sample(range(2,n),n-s))
```

In [77]:

```
train_modified = pd.read_csv('train_modified.csv',skiprows = skip)
```

In [78]:
```
# Replace all NAs with their mean value
train_modified.fillna(train_modified.mean(),inplace = True)
train_modified.isnull().any()
```

Out[78]:
```
Semana                      False
Agencia_ID                  False
Canal_ID                    False
Ruta_SAK                    False
Cliente_ID                  False
Producto_ID                 False
Client_Type                 False
Producto_name               False
weight                      False
pieces                      False
weight_per_piece            False
Demanda_uni_equil           False
Demanda_uni_equil_tminus1   False
Demanda_uni_equil_tminus2   False
Demanda_uni_equil_tminus3   False
Demanda_uni_equil_tminus4   False
Demanda_uni_equil_tminus5   False
Agencia_ID_count            False
Canal_ID_count              False
Ruta_SAK_count              False
Cliente_ID_count            False
Producto_ID_count           False
Client_Type_count           False
dtype: bool
```

In [79]:
```
train_modified.columns
```

Out[79]:
```
Index(['Semana', 'Agencia_ID', 'Canal_ID', 'Ruta_SAK', 'Cliente_ID',
       'Producto_ID', 'Client_Type', 'Producto_name', 'weight', 'pieces',
       'weight_per_piece', 'Demanda_uni_equil', 'Demanda_uni_equil_tminus1',
       'Demanda_uni_equil_tminus2', 'Demanda_uni_equil_tminus3',
       'Demanda_uni_equil_tminus4', 'Demanda_uni_equil_tminus5',
       'Agencia_ID_count', 'Canal_ID_count', 'Ruta_SAK_count',
       'Cliente_ID_count', 'Producto_ID_count', 'Client_Type_count'],
      dtype='object')
```

In [80]:
```
# Use MinMaxScaler to scale numerical data
data_num_scaled = MinMaxScaler().fit(train_modified.drop(columns =[ 'Semana',
'Agencia_ID','Canal_ID','Ruta_SAK','Cliente_ID','Producto_ID','Client_Type',
'Producto_name'])).transform(train_modified.drop(columns =[ 'Semana','Agencia_
ID','Canal_ID','Ruta_SAK','Cliente_ID','Producto_ID','Client_Type', 'Producto_
name']))

data_cat = train_modified[ ['Semana','Agencia_ID','Canal_ID',
                            'Ruta_SAK','Cliente_ID','Producto_ID',
                            'Client_Type', 'Producto_name']]

# Transfer to dataframe and add column names
data_num_scaled = pd.DataFrame(data_num_scaled,columns = ['weight', 'pieces',
       'weight_per_piece', 'Demanda_uni_equil', 'Demanda_uni_equil_tminus1',
       'Demanda_uni_equil_tminus2', 'Demanda_uni_equil_tminus3',
       'Demanda_uni_equil_tminus4', 'Demanda_uni_equil_tminus5',
       'Agencia_ID_count', 'Canal_ID_count', 'Ruta_SAK_count',
       'Cliente_ID_count', 'Producto_ID_count', 'Client_Type_count'])

data_cat = pd.DataFrame(data_cat)

# Join two dataframes together
data_scaled = data_cat.join(data_num_scaled)
```

In [81]:
```
X = data_scaled.drop(columns = ['Demanda_uni_equil'])
Y = train_modified['Demanda_uni_equil']
```

In [82]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,train_size = 0.75,random_
state = 1)
```

In [83]:
```
# Define a function to evaluate our predictions
def RMSLE(actuals, predictions):
    """ Takes true values and predictions.
        Returns their Root Mean Squared Logarithmic Error.
    """
    result = 0.0
    actuals = np.asarray(actuals) * 1.0
    predictions = np.asarray(predictions)
    if(len(actuals) == len(predictions)):
        result = np.sqrt(sum(((np.log(predictions + 1.0) -
                               np.log(actuals + 1.0)) ** 2) / len(actuals)))
        return result
    else:
        return "Error!"
```

In [15]:
```
# Gridsearch with stratified KFold CV to find the best params
n_estimators = [30,40,50]
depth_value = [5,10,15]
alpha = [0.1,0.3,0.5]

# Use make_scorer() function to help use our evaluation matrics
# in GridsearchCV
from sklearn.metrics import make_scorer
scorer = make_scorer(RMSLE,greater_is_better = False)

param_grid = dict(n_estimators = n_estimators,max_depth = depth_value,learning
_rate = alpha)
kfold = StratifiedKFold(n_splits=10,shuffle=True,random_state=1)
grid_search = GridSearchCV(XGBRegressor(objective ='reg:squarederror'),
                           param_grid,cv = kfold,scoring = scorer)
grid_result = grid_search.fit(X_train,Y_train)
print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_
))
```

```
Best: -0.496802 using {'learning_rate': 0.1, 'max_depth': 15, 'n_estimators':
30}
```

In [84]:
```
# Fit the XGBRegressor model with the best params
xgbmodel = XGBRegressor(objective ='reg:squarederror',n_estimaors = 30,
                        max_depth=15,learning_rate=0.1).fit(X_train,Y_train)
prediction = xgbmodel.predict(X_test)
prediction_pos = np.where(prediction<0,0,prediction)
score = RMSLE(Y_test,prediction_pos)
```

In [85]:
```
score
```

Out[85]:
```
0.49445419919165134
```

Feature Importance

In [86]:

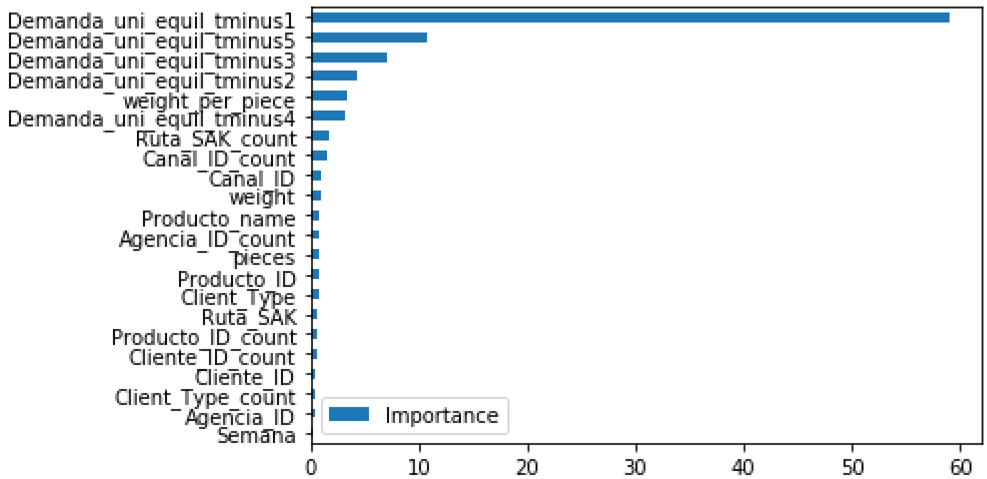
```
# calculate feature importance for each feature
importance = pd.DataFrame({'Importance':xgbmodel.feature_importances_*100},ind
ex = X.columns)
importance.sort_values(by='Importance',axis=0,ascending=False)
```

Out[86]:

	Importance
Demanda_uni_equil_tminus1	58.978928
Demanda_uni_equil_tminus5	10.807762
Demanda_uni_equil_tminus3	7.161037
Demanda_uni_equil_tminus2	4.261444
weight_per_piece	3.320200
Demanda_uni_equil_tminus4	3.189065
Ruta_SAK_count	1.765322
Canal_ID_count	1.490104
Canal_ID	0.932841
weight	0.902859
Producto_name	0.870707
Agencia_ID_count	0.849580
pieces	0.800243
Producto_ID	0.749049
Client_Type	0.705741
Ruta_SAK	0.662346
Producto_ID_count	0.585941
Cliente_ID_count	0.523772
Cliente_ID	0.464204
Client_Type_count	0.392494
Agencia_ID	0.342910
Semana	0.243445

In [87]:
```
importance.sort_values(by='Importance',axis=0,ascending=True).plot(kind='barh'
)
```

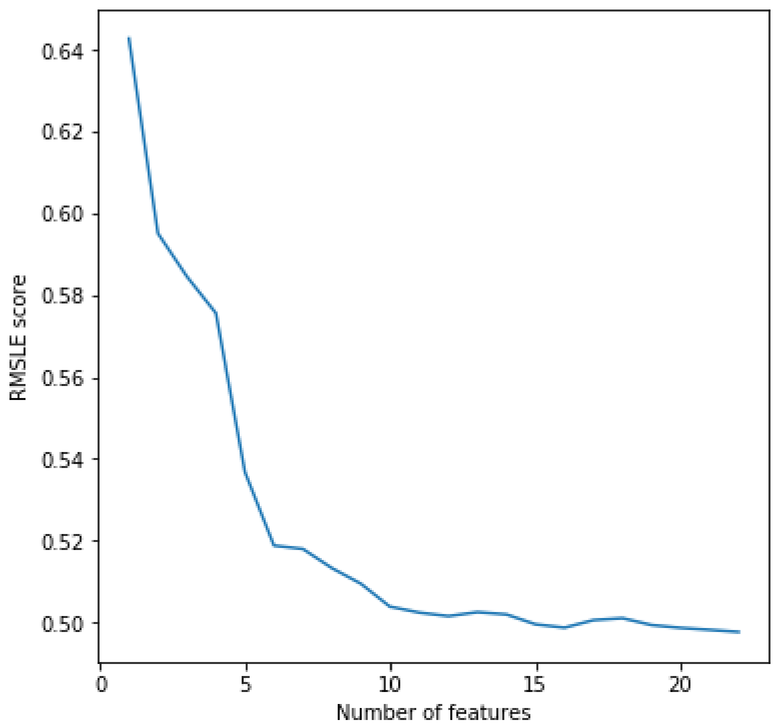
Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x2a0add096d8>



The plot shows that three to four features have great feature importance. In order to figure out how many features used returns the best prediction, we implement a for loop. We plot a RMSLE score for all models using one feature up to twenty two (all) features.

In [24]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_log_error

# Calculate thresholds and save them into a list
thresholds = sorted(xgbmodel.feature_importances_,reverse=True)

# Create two empty lists
RMSLElist =[]
nlist =[]
for thresh in thresholds:
    selection = SelectFromModel(xgbmodel,threshold = thresh, prefit = True)
    X_train_selected = selection.transform(X_train)
    X_test_selected = selection.transform(X_test)

    model2 = XGBRegressor(objective ='reg:squarederror',n_estimaors = 30,
                          max_depth=15,learning_rate=0.1)
    model2.fit(X_train_selected,Y_train)

    pred = model2.predict(X_test_selected)

    # Replace all negative predictions with zero since the demand can't be neg
ative
    pred = np.where(pred<0,0,pred)
    RMSLE_score = RMSLE(Y_test,pred)
    n_features = X_train_selected.shape[1]
    print ('Thresh = %.3f,n = %d, RMSLE:%.4f' % (thresh, n_features,RMSLE_scor
e))
    # Save the RMSLE score into a list
    RMSLElist.append(RMSLE_score)
    nlist.append(n_features)

# Plot the figure (RMSLE score v.s. Number of features)
pyplot.figure(figsize=(6,6))
pyplot.xlabel('Number of features')
pyplot.ylabel('RMSLE score')
pyplot.plot(nlist,RMSLElist)
pyplot.show()
```

```
Thresh = 0.565,n = 1, RMSLE:0.6427
Thresh = 0.115,n = 2, RMSLE:0.5951
Thresh = 0.102,n = 3, RMSLE:0.5845
Thresh = 0.073,n = 4, RMSLE:0.5755
Thresh = 0.036,n = 5, RMSLE:0.5368
Thresh = 0.021,n = 6, RMSLE:0.5188
Thresh = 0.011,n = 7, RMSLE:0.5180
Thresh = 0.009,n = 8, RMSLE:0.5133
Thresh = 0.008,n = 9, RMSLE:0.5095
Thresh = 0.008,n = 10, RMSLE:0.5039
Thresh = 0.006,n = 11, RMSLE:0.5024
Thresh = 0.005,n = 12, RMSLE:0.5016
Thresh = 0.005,n = 13, RMSLE:0.5026
Thresh = 0.005,n = 14, RMSLE:0.5020
Thresh = 0.005,n = 15, RMSLE:0.4996
Thresh = 0.004,n = 16, RMSLE:0.4988
Thresh = 0.004,n = 17, RMSLE:0.5006
Thresh = 0.004,n = 18, RMSLE:0.5011
Thresh = 0.003,n = 19, RMSLE:0.4994
Thresh = 0.003,n = 20, RMSLE:0.4987
Thresh = 0.003,n = 21, RMSLE:0.4982
Thresh = 0.003,n = 22, RMSLE:0.4977
```



Based on the figure, sixteen features should be used in the XGBRegressor model since it returns a relatively low RMSLE score while using less features.

In [88]:
```
# transform X_train and X_test to datasets with selected features
from sklearn.feature_selection import SelectFromModel
select_best = SelectFromModel(xgbmodel,max_features = 16,prefit = True,thresho
ld=-np.inf)
X_train_selected = select_best.transform(X_train)
X_test_selected = select_best.transform(X_test)
```

In [89]:
```
# Put all feature importance values into a dataframe in descending order.
# The 16 features selected are the top 16 ones
importance = pd.DataFrame({'Importance':xgbmodel.feature_importances_*100}, in
dex = X.columns)
importance.sort_values(by='Importance',axis=0,ascending=False)
```

Out[89]:

	Importance
Demanda_uni_equil_tminus1	58.978928
Demanda_uni_equil_tminus5	10.807762
Demanda_uni_equil_tminus3	7.161037
Demanda_uni_equil_tminus2	4.261444
weight_per_piece	3.320200
Demanda_uni_equil_tminus4	3.189065
Ruta_SAK_count	1.765322
Canal_ID_count	1.490104
Canal_ID	0.932841
weight	0.902859
Producto_name	0.870707
Agencia_ID_count	0.849580
pieces	0.800243
Producto_ID	0.749049
Client_Type	0.705741
Ruta_SAK	0.662346
Producto_ID_count	0.585941
Cliente_ID_count	0.523772
Cliente_ID	0.464204
Client_Type_count	0.392494
Agencia_ID	0.342910
Semana	0.243445

In [90]:
```
# Put the numpy array into a dataframe and name all columns
df_X_train_selected = pd.DataFrame(X_train_selected,columns = [
'Canal_ID',
'Producto_ID',
'Producto_name',
'weight',
'pieces',
'weight_per_piece',
'Demanda_uni_equil_tminus1',
'Demanda_uni_equil_tminus2',
'Demanda_uni_equil_tminus3',
'Demanda_uni_equil_tminus4',
'Demanda_uni_equil_tminus5',
'Agencia_ID_count',
'Canal_ID_count',
'Ruta_SAK_count',
'Cliente_ID_count',
'Producto_ID_count'
] )
```

In [91]:
```
# fit a new model using the 16 selected features
model_best = XGBRegressor(objective ='reg:squarederror',n_estimaors = 30,
                          max_depth=15,learning_rate=0.1).fit(df_X_train_selec
ted,Y_train)
```

In [92]:
```
# Read our kaggle test dataset
kaggle_test = pd.read_csv('test_modified.csv')
```

In [93]:
```
# Deal with NAs. Replace them with mean value
kaggle_test.fillna(kaggle_test.mean(),inplace = True)
kaggle_test.isnull().any()
```

Out[93]:
```
Semana                       False
Agencia_ID                   False
Canal_ID                     False
Ruta_SAK                     False
Cliente_ID                   False
Producto_ID                  False
Client_Type                  False
Producto_name                False
weight                       False
pieces                       False
weight_per_piece             False
Demanda_uni_equil_tminus1    False
Demanda_uni_equil_tminus2    False
Demanda_uni_equil_tminus3    False
Demanda_uni_equil_tminus4    False
Demanda_uni_equil_tminus5    False
Agencia_ID_count             False
Canal_ID_count               False
Ruta_SAK_count               False
Cliente_ID_count             False
Producto_ID_count            False
Client_Type_count            False
dtype: bool
```

In [94]:
```
# Scale numerical features

kaggle_num_scaled = MinMaxScaler().fit(train_modified.drop(columns =[ 'Semana'
,'Agencia_ID','Canal_ID','Ruta_SAK','Cliente_ID','Producto_ID','Client_Type',
'Producto_name', 'Demanda_uni_equil'])).transform(kaggle_test.drop(columns =[
'Semana','Agencia_ID','Canal_ID','Ruta_SAK','Cliente_ID','Producto_ID','Client
_Type', 'Producto_name']))
kaggle_cat = kaggle_test[['Semana','Agencia_ID','Canal_ID','Ruta_SAK','Cliente
_ID','Producto_ID','Client_Type', 'Producto_name']]
kaggle_num_scaled = pd.DataFrame(kaggle_num_scaled,columns = ['weight', 'piece
s',
       'weight_per_piece', 'Demanda_uni_equil_tminus1',
       'Demanda_uni_equil_tminus2', 'Demanda_uni_equil_tminus3',
       'Demanda_uni_equil_tminus4', 'Demanda_uni_equil_tminus5',
       'Agencia_ID_count', 'Canal_ID_count', 'Ruta_SAK_count',
       'Cliente_ID_count', 'Producto_ID_count', 'Client_Type_count'])
kaggle_cat = pd.DataFrame(kaggle_cat)
kaggle_scaled = kaggle_cat.join(kaggle_num_scaled)
```

In [95]:
```
# Separate week 10 test data and week 11 test data
# since we need to predict two weeks inventory demand separately
# i.e. use week4 - week9 data to predict week10
#      use week5 - week10(prediction) data to predict week 11

kaggle_week10 = kaggle_scaled[kaggle_scaled['Semana']==10]
kaggle_week11 = kaggle_scaled[kaggle_scaled['Semana']==11]
```

In [96]:
```
# Drop the 6 features with least feature importance values
kaggle_week10_select = kaggle_week10.drop(columns = [
'Cliente_ID',
'Ruta_SAK',
'Client_Type',
'Client_Type_count',
'Agencia_ID',
'Semana',
])
```

In [97]:
```
# Predict week 10 inventroy demand
week10_pred = model_best.predict(kaggle_week10_select)
week10_pred = np.where(week10_pred<0,0,week10_pred)
week10_pred_df = pd.DataFrame(week10_pred, dtype= float, columns = ['Demanda_u
ni_equil'])
```

In [98]:
```
week10_pred_df[:5]
```

Out[98]:

	Demanda_uni_equil
0	1167.73645
1	1167.73645
2	1167.73645
3	1167.73645
4	1167.73645

In [99]:
```
# Add our prediction to the week10 test dataset
week10_match = kaggle_week10.copy()
week10_match['Demanda_uni_equil'] = week10_pred
```

In [100]:
```
# Calculate the column 'Demanda_uni_equil_tminus1' for week11 prediction
# Demanda_uni_equil_tminus1 refers to an average demand of a product of a clie
nt.
columns = [ 'Cliente_ID', 'Producto_ID']
week10_tminus = pd.DataFrame({'Demanda_uni_equil_tminus1' : week10_match.group
by(columns)
                                ['Demanda_uni_equil_tminus1'].mean()}).reset_ind
ex()
```

In [101]:
```
week10_tminus[:5]
```

Out[101]:

	Cliente_ID	Producto_ID	Demanda_uni_equil_tminus1
0	26	31518	0.000000
1	26	34210	0.026000
2	26	34785	0.016000
3	26	34786	0.050000
4	26	35142	0.025333

In [102]:
```
# Replace the original 'Demanda_uni_equil_tminus1' column in week11 test datas
et
# with the one we just calculated above
week11_match = kaggle_week11.copy().drop(columns = 'Demanda_uni_equil_tminus1'
)
week11_match = week11_match.merge(week10_tminus, how = 'left',
                                  on = [ 'Cliente_ID', 'Producto_ID'])
```

In [103]:
```
# Drop the six features with least feature importance values
week11_select = week11_match.drop(columns = [
'Cliente_ID',
'Ruta_SAK',
'Client_Type',
'Client_Type_count',
'Agencia_ID',
'Semana',
])
week11_select.columns
```

Out[103]:
```
Index(['Canal_ID', 'Producto_ID', 'Producto_name', 'weight', 'pieces',
       'weight_per_piece', 'Demanda_uni_equil_tminus2',
       'Demanda_uni_equil_tminus3', 'Demanda_uni_equil_tminus4',
       'Demanda_uni_equil_tminus5', 'Agencia_ID_count', 'Canal_ID_count',
       'Ruta_SAK_count', 'Cliente_ID_count', 'Producto_ID_count',
       'Demanda_uni_equil_tminus1'],
      dtype='object')
```

In [104]:
```
# reorder the columns so that it matches the X_train dataset
columns_reorder = [
            'Canal_ID', 'Producto_ID', 'Producto_name', 'weight', 'pieces',
        'weight_per_piece','Demanda_uni_equil_tminus1', 'Demanda_uni_equil_tmin
us2',
        'Demanda_uni_equil_tminus3', 'Demanda_uni_equil_tminus4',
        'Demanda_uni_equil_tminus5', 'Agencia_ID_count', 'Canal_ID_count',
        'Ruta_SAK_count', 'Cliente_ID_count', 'Producto_ID_count'
           ]

week11_select = week11_select.reindex(columns = columns_reorder)
week11_select.columns
```

Out[104]:
```
Index(['Canal_ID', 'Producto_ID', 'Producto_name', 'weight', 'pieces',
       'weight_per_piece', 'Demanda_uni_equil_tminus1',
       'Demanda_uni_equil_tminus2', 'Demanda_uni_equil_tminus3',
       'Demanda_uni_equil_tminus4', 'Demanda_uni_equil_tminus5',
       'Agencia_ID_count', 'Canal_ID_count', 'Ruta_SAK_count',
       'Cliente_ID_count', 'Producto_ID_count'],
      dtype='object')
```

In [105]:
```
# Use data from week6 to week10 to predict the inventory demand of week11
week11_pred = model_best.predict(week11_select)
# Replace all negative predictions with zero
week11_pred = np.where(week11_pred<0,0,week11_pred)
# Save it into a dataframe
week11_pred_df = pd.DataFrame(week11_pred, dtype= float, columns = ['Demanda_u
ni_equil'])
```

In [106]:
```
week11_pred_df[:5]
```

Out[106]:

	Demanda_uni_equil
0	1167.736450
1	1167.736450
2	1167.736450
3	1167.736450
4	1159.280029

In [107]:
```
# Now we put our week11 predictions into our week11 test dataset
week11_match2 = kaggle_week11.copy()
week11_match2['Demanda_uni_equil'] = week11_pred
```

In [108]:
```
kaggle_test.columns
```

Out[108]:
```
Index(['Semana', 'Agencia_ID', 'Canal_ID', 'Ruta_SAK', 'Cliente_ID',
       'Producto_ID', 'Client_Type', 'Producto_name', 'weight', 'pieces',
       'weight_per_piece', 'Demanda_uni_equil_tminus1',
       'Demanda_uni_equil_tminus2', 'Demanda_uni_equil_tminus3',
       'Demanda_uni_equil_tminus4', 'Demanda_uni_equil_tminus5',
       'Agencia_ID_count', 'Canal_ID_count', 'Ruta_SAK_count',
       'Cliente_ID_count', 'Producto_ID_count', 'Client_Type_count'],
      dtype='object')
```

In [109]:
```
week10_match.columns
```

Out[109]:
```
Index(['Semana', 'Agencia_ID', 'Canal_ID', 'Ruta_SAK', 'Cliente_ID',
       'Producto_ID', 'Client_Type', 'Producto_name', 'weight', 'pieces',
       'weight_per_piece', 'Demanda_uni_equil_tminus1',
       'Demanda_uni_equil_tminus2', 'Demanda_uni_equil_tminus3',
       'Demanda_uni_equil_tminus4', 'Demanda_uni_equil_tminus5',
       'Agencia_ID_count', 'Canal_ID_count', 'Ruta_SAK_count',
       'Cliente_ID_count', 'Producto_ID_count', 'Client_Type_count',
       'Demanda_uni_equil'],
      dtype='object')
```

In [110]:
```
# Since we normalize our numerical features, we cannot match our week10_match
 dataset
# with kaggle_test dataset based on those features. Thus, we choose to match t
hem up
# according to all categorical features

all_pred10 = kaggle_test.merge(week10_match, how = 'left',
                               on = [ 'Semana', 'Agencia_ID', 'Canal_ID', 'Rut
a_SAK', 'Cliente_ID',
                                      'Producto_ID', 'Client_Type', 'Producto
_name'])
```

In [111]:
```
# Select common columns and also rename our prediction column
all_pred10 = all_pred10[[ 'Semana', 'Agencia_ID', 'Canal_ID', 'Ruta_SAK', 'Cli
ente_ID',
    'Producto_ID', 'Client_Type', 'Producto_name','Demanda_uni_equil']]
all_pred10 = all_pred10.rename(columns={'Demanda_uni_equil': 'pred10'})
```

In [112]:
```
# Do the same thing for week11 test dataset
all_pred11 = kaggle_test.merge(week11_match2, how = 'left', on = [ 'Semana',
'Agencia_ID', 'Canal_ID', 'Ruta_SAK', 'Cliente_ID',
       'Producto_ID', 'Client_Type', 'Producto_name'])
```

In [113]:
```
all_pred11 = all_pred11[[ 'Semana', 'Agencia_ID', 'Canal_ID', 'Ruta_SAK', 'Cli
ente_ID',
       'Producto_ID', 'Client_Type', 'Producto_name','Demanda_uni_equil']]
all_pred11 = all_pred11.rename(columns={'Demanda_uni_equil': 'pred11'})
```

In [114]:
```
# Create a new dataframe called all_pred where there are two columns:
# 1) pred10 and 2) pred11
all_pred = pd.DataFrame(all_pred10['pred10'],columns = ['pred10'])
```

In [115]:
```
all_pred['pred11'] = all_pred11['pred11']
```

In [116]:
```
all_pred[:10]
```

Out[116]:

	pred10	pred11
0	NaN	1167.73645
1	1167.73645	NaN
2	1167.73645	NaN
3	NaN	1167.73645
4	1167.73645	NaN
5	1167.73645	NaN
6	1167.73645	NaN
7	1167.73645	NaN
8	1167.73645	NaN
9	NaN	1167.73645

In [117]:
```
# Replace all NA's with zero
all_pred.fillna(0,inplace = True)
all_pred.isnull().any()
```

Out[117]:
```
pred10    False
pred11    False
dtype: bool
```

In [118]:
```
# create a new column and add pred10 and pre11 up.
# Thus, the new column we obtained happens to be the prediction
# of week 10 or week 11, and in this new column there is no NA.

all_pred['Demanda_uni_equil'] = all_pred['pred10']+all_pred['pred11']
all_pred[:5]
```

Out[118]:

	pred10	pred11	Demanda_uni_equil
0	0.00000	1167.73645	1167.73645
1	1167.73645	0.00000	1167.73645
2	1167.73645	0.00000	1167.73645
3	0.00000	1167.73645	1167.73645
4	1167.73645	0.00000	1167.73645

In [119]:
```
# Write our kaggle submission file and save it as a CSV file.

kaggle_df = pd.DataFrame(all_pred['Demanda_uni_equil'],dtype = float, columns
= ['Demanda_uni_equil'])
kaggle_df.to_csv('kaggle_result.csv',index = True,header = True,index_label =
'id')
```

In [120]:
```
# Test if our prediction covers all rows in our kaggle_test set.

kaggle_df.shape, kaggle_test.shape
```

Out[120]:
```
((6999251, 1), (6999251, 22))
```

Kaggle Score: 1.13564

This kaggle score is not as good as we expected. Based on our time and device used, in order to finish this project on time, we used only 100,000 * 0.75 = 75,000 data (2% of the total training dataset) to train our model and predict 7 million data using that.Our training set was too small, and that caused the major deviation (error). However, our model well predicted our small-size-test dataset, which contains 100,000 * 0.25 = 25,000 data, with a 0.49 RMSLE score.Therefore, it is reasonable for us to believe that if we have more time and more computing resources,we could possibly decrease our kaggle score to 0.5 which will place us in top 50%.

Furthermore,we used the Google Colab tool to reproduce the above training process with all 40 million datasets, and used the above results of gridsearch to train and fit the xgbmodel. In the end, we re-predicted the kaggle test data and uploaded it to the kaggle website to score again. My kaggle score obtained from training with the full data set is 0.58. Compared with the previous smallsize results, our ranking has improved significantly.

Final Kaggle Score:0.58796