Baseline Model

I have done the pre-processing and decided the evaluation metric (RMSLE), will create baseline models for forecasting demand.

Generally, to create baseline model using very basic techniques like mean prediction and then implemented more complex solutions to improve the results that I got from the baseline model. The idea is to spot bugs in the final model as any score which is below baseline is not good enough.

Summary:

- Predict the target using the mean demand from historical data for that particular store and product
- Use Simple Moving Average (a very basic Time Series Model).
- Comparision between a linear Regression based model and Decision Tree model.

In [1]:

```
# importing the required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import mean_squared_log_error as msle

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [2]:

```
# reading the pre-processed dataset
train_data = pd.read_csv('updated_dataset/updated_train_data.csv')
product_data = pd.read_csv('updated_dataset/updated_product_data.csv')
store_data = pd.read_csv('updated_dataset/updated_store_data.csv')
print(train_data.shape, product_data.shape, store_data.shape)
```

```
(232266, 7) (30, 22) (76, 17)
```

In [3]:

```
# view the train data
train_data.head(2)
```

Out[3]:

	WEEK_END_DATE	STORE_NUM	UPC	BASE_PRICE	FEATURE	DISPLAY	UNITS
0	14-Jan-09	367	1111009477	1.57	0	0	13
1	14-Jan-09	367	1111009497	1.39	0	0	20

In [4]:

```
# view the product data
product_data.head(2)
```

Out[4]:

	UPC	MANUFACTURER_1	MANUFACTURER_2	MANUFACTURER_3	MANUFACTURER_4	MANUFACTURER_5	MANUFACTURER_6	MANI
0	1111009477	1	0	0	0	0	0	
1	1111009497	1	0	0	0	0	0	

2 rows × 22 columns

In [5]:

```
# view the store data
store_data.head(2)
```

Out[5]:

	STORE_ID	ADDRESS_STATE_PROV_CODE_1	ADDRESS_STATE_PROV_CODE_2	ADDRESS_STATE_PROV_CODE_3	ADDRESS_STATE_PROV_CODE
0	367	1	0	0	
1	389	1	0	0	

Merging Tables

In [6]:

```
# merge the datasets
merge_data = train_data.merge(product_data, how='left', on= 'UPC')
merge_data = merge_data.merge(store_data, how= 'left', left_on= 'STORE_NUM', right_on= 'STORE_ID')
```

In [7]:

```
merge_data = merge_data.drop(columns=['STORE_ID'])
```

In [8]:

```
# check if there is any null value in the final merged data set.
merge_data.isna().sum().sum()
```

Out[8]:

```
0
```

In [9]:

```
# let's look at a row, how data looks in the merged dataset
merge_data.loc[0]
```

Out[9]:

```
WEEK_END_DATE                 14-Jan-09
STORE_NUM                           367
UPC                          1111009477
BASE_PRICE                         1.57
FEATURE                               0
DISPLAY                               0
UNITS                                13
MANUFACTURER_1                        1
MANUFACTURER_2                        0
MANUFACTURER_3                        0
MANUFACTURER_4                        0
MANUFACTURER_5                        0
MANUFACTURER_6                        0
MANUFACTURER_7                        0
MANUFACTURER_8                        0
MANUFACTURER_9                        0
CATEGORY_1                            1
CATEGORY_2                            0
CATEGORY_3                            0
CATEGORY_4                            0
SUB_CATEGORY_1                        1
SUB_CATEGORY_2                        0
SUB_CATEGORY_3                        0
SUB_CATEGORY_4                        0
SUB_CATEGORY_5                        0
SUB_CATEGORY_6                        0
SUB_CATEGORY_7                        0
PRODUCT_SIZE                          2
ADDRESS_STATE_PROV_CODE_1             1
ADDRESS_STATE_PROV_CODE_2             0
ADDRESS_STATE_PROV_CODE_3             0
ADDRESS_STATE_PROV_CODE_4             0
MSA_CODE_1                            1
MSA_CODE_2                            0
MSA_CODE_3                            0
MSA_CODE_4                            0
MSA_CODE_5                            0
MSA_CODE_6                            0
MSA_CODE_7                            0
MSA_CODE_8                            0
MSA_CODE_9                            0
SEG_VALUE_NAME                        1
SALES_AREA_SIZE_NUM               24721
AVG_WEEKLY_BASKETS                12707
Name: 0, dtype: object
```

Creating the Validation Set

In [10]:

```
# convert the column WEEK_END_DATE to datetime format
merge_data.WEEK_END_DATE = pd.to_datetime(merge_data.WEEK_END_DATE)
```

- We will store the unique `WEEK_END_DATE` in a list so that it would be easier to split the data based on time. It will be used to create and train and validation split.
- We will keep one week gap between the train and validation set and train set will start from the very begining from where the data is available.

In [11]:

```
# store the unique dates as the dates are already sorted in the original dataframe, we can directly use the unique fun
weeks = merge_data.WEEK_END_DATE.unique()
weeks
```

Out[11]:

```
array(['2009-01-14T00:00:00.000000000', '2009-01-21T00:00:00.000000000',
       '2009-01-28T00:00:00.000000000', '2009-02-04T00:00:00.000000000',
       '2009-02-11T00:00:00.000000000', '2009-02-18T00:00:00.000000000',
       '2009-02-25T00:00:00.000000000', '2009-03-04T00:00:00.000000000',
       '2009-03-11T00:00:00.000000000', '2009-03-18T00:00:00.000000000',
       '2009-03-25T00:00:00.000000000', '2009-04-01T00:00:00.000000000',
       '2009-04-08T00:00:00.000000000', '2009-04-15T00:00:00.000000000',
       '2009-04-22T00:00:00.000000000', '2009-04-29T00:00:00.000000000',
       '2009-05-06T00:00:00.000000000', '2009-05-13T00:00:00.000000000',
       '2009-05-20T00:00:00.000000000', '2009-05-27T00:00:00.000000000',
       '2009-06-03T00:00:00.000000000', '2009-06-10T00:00:00.000000000',
       '2009-06-17T00:00:00.000000000', '2009-06-24T00:00:00.000000000',
       '2009-07-01T00:00:00.000000000', '2009-07-08T00:00:00.000000000',
       '2009-07-15T00:00:00.000000000', '2009-07-22T00:00:00.000000000',
       '2009-07-29T00:00:00.000000000', '2009-08-05T00:00:00.000000000',
       '2009-08-12T00:00:00.000000000', '2009-08-19T00:00:00.000000000',
       '2009-08-26T00:00:00.000000000', '2009-09-02T00:00:00.000000000',
       '2009-09-09T00:00:00.000000000', '2009-09-16T00:00:00.000000000',
       '2009-09-23T00:00:00.000000000', '2009-09-30T00:00:00.000000000',
       '2009-10-07T00:00:00.000000000', '2009-10-14T00:00:00.000000000',
       '2009-10-21T00:00:00.000000000', '2009-10-28T00:00:00.000000000',
       '2009-11-04T00:00:00.000000000', '2009-11-11T00:00:00.000000000',
       '2009-11-18T00:00:00.000000000', '2009-11-25T00:00:00.000000000',
       '2009-12-02T00:00:00.000000000', '2009-12-09T00:00:00.000000000',
       '2009-12-16T00:00:00.000000000', '2009-12-23T00:00:00.000000000',
       '2009-12-30T00:00:00.000000000', '2010-01-06T00:00:00.000000000',
       '2010-01-13T00:00:00.000000000', '2010-01-20T00:00:00.000000000',
       '2010-01-27T00:00:00.000000000', '2010-02-03T00:00:00.000000000',
       '2010-02-10T00:00:00.000000000', '2010-02-17T00:00:00.000000000',
       '2010-02-24T00:00:00.000000000', '2010-03-03T00:00:00.000000000',
       '2010-03-10T00:00:00.000000000', '2010-03-17T00:00:00.000000000',
       '2010-03-24T00:00:00.000000000', '2010-03-31T00:00:00.000000000',
       '2010-04-07T00:00:00.000000000', '2010-04-14T00:00:00.000000000',
       '2010-04-21T00:00:00.000000000', '2010-04-28T00:00:00.000000000',
       '2010-05-05T00:00:00.000000000', '2010-05-12T00:00:00.000000000',
       '2010-05-19T00:00:00.000000000', '2010-05-26T00:00:00.000000000',
       '2010-06-02T00:00:00.000000000', '2010-06-09T00:00:00.000000000',
       '2010-06-16T00:00:00.000000000', '2010-06-23T00:00:00.000000000',
       '2010-06-30T00:00:00.000000000', '2010-07-07T00:00:00.000000000',
       '2010-07-14T00:00:00.000000000', '2010-07-21T00:00:00.000000000',
       '2010-07-28T00:00:00.000000000', '2010-08-04T00:00:00.000000000',
       '2010-08-11T00:00:00.000000000', '2010-08-18T00:00:00.000000000',
       '2010-08-25T00:00:00.000000000', '2010-09-01T00:00:00.000000000',
       '2010-09-08T00:00:00.000000000', '2010-09-15T00:00:00.000000000',
       '2010-09-22T00:00:00.000000000', '2010-09-29T00:00:00.000000000',
       '2010-10-06T00:00:00.000000000', '2010-10-13T00:00:00.000000000',
       '2010-10-20T00:00:00.000000000', '2010-10-27T00:00:00.000000000',
       '2010-11-03T00:00:00.000000000', '2010-11-10T00:00:00.000000000',
       '2010-11-17T00:00:00.000000000', '2010-11-24T00:00:00.000000000',
       '2010-12-01T00:00:00.000000000', '2010-12-08T00:00:00.000000000',
       '2010-12-15T00:00:00.000000000', '2010-12-22T00:00:00.000000000',
       '2010-12-29T00:00:00.000000000', '2011-01-05T00:00:00.000000000',
       '2011-01-12T00:00:00.000000000', '2011-01-19T00:00:00.000000000',
       '2011-01-26T00:00:00.000000000', '2011-02-02T00:00:00.000000000',
       '2011-02-09T00:00:00.000000000', '2011-02-16T00:00:00.000000000',
       '2011-02-23T00:00:00.000000000', '2011-03-02T00:00:00.000000000',
       '2011-03-09T00:00:00.000000000', '2011-03-16T00:00:00.000000000',
       '2011-03-23T00:00:00.000000000', '2011-03-30T00:00:00.000000000',
       '2011-04-06T00:00:00.000000000', '2011-04-13T00:00:00.000000000',
       '2011-04-20T00:00:00.000000000', '2011-04-27T00:00:00.000000000',
       '2011-05-04T00:00:00.000000000', '2011-05-11T00:00:00.000000000',
       '2011-05-18T00:00:00.000000000', '2011-05-25T00:00:00.000000000',
       '2011-06-01T00:00:00.000000000', '2011-06-08T00:00:00.000000000',
       '2011-06-15T00:00:00.000000000', '2011-06-22T00:00:00.000000000',
       '2011-06-29T00:00:00.000000000', '2011-07-06T00:00:00.000000000',
       '2011-07-13T00:00:00.000000000', '2011-07-20T00:00:00.000000000',
       '2011-07-27T00:00:00.000000000', '2011-08-03T00:00:00.000000000',
       '2011-08-10T00:00:00.000000000', '2011-08-17T00:00:00.000000000',
       '2011-08-24T00:00:00.000000000', '2011-08-31T00:00:00.000000000',
       '2011-09-07T00:00:00.000000000', '2011-09-14T00:00:00.000000000',
       '2011-09-21T00:00:00.000000000', '2011-09-28T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

In [12]:

```
# define the function that will return a dictionary which contains the keys

"""
[
  {
 "validation_set" : ## validation set week,
 "train_set_end_date" : ## last week of trainind set with one gap with the validation set.
  },
  .
  .
  .
  {
   "validation_set" : ''
   "train_set_end_date" : ''
  }
]

Initially, we will use the same start date of each training data set.


The function will take the parameter number which is the number of train and validation sets required.
"""
def get_train_validation_set(number=1):
    validation_sets = []
    for n in range(number):
        x = {}

        x['validation_set'] = weeks[len(weeks)-n-1]
        x['train_set_end_date'] = weeks[len(weeks)-n-3]
        validation_sets.append(x)

    return validation_sets
```

I just checking with just 1 validation set to check the RMSLE score that are getting from different baseline models. However, taking multiple validation sets also allows us to look at the consistency of scores across multiple subsets of data.

Here, I will take 5 validation sets for starters

In [13]:

```
# we will create our baseline model and test it on 5 different sets
validation_sets = get_train_validation_set(number=5)
```

In [14]:

```
# the dictionary that we got from our function
validation_sets
```

Out[14]:

```
[{'validation_set': numpy.datetime64('2011-09-28T00:00:00.000000000'),
  'train_set_end_date': numpy.datetime64('2011-09-14T00:00:00.000000000')},
 {'validation_set': numpy.datetime64('2011-09-21T00:00:00.000000000'),
  'train_set_end_date': numpy.datetime64('2011-09-07T00:00:00.000000000')},
 {'validation_set': numpy.datetime64('2011-09-14T00:00:00.000000000'),
  'train_set_end_date': numpy.datetime64('2011-08-31T00:00:00.000000000')},
 {'validation_set': numpy.datetime64('2011-09-07T00:00:00.000000000'),
  'train_set_end_date': numpy.datetime64('2011-08-24T00:00:00.000000000')},
 {'validation_set': numpy.datetime64('2011-08-31T00:00:00.000000000'),
  'train_set_end_date': numpy.datetime64('2011-08-17T00:00:00.000000000')}]
```

In [15]:

```
# Now, we will use that dictionary and store the train and validation sets as a list of tuples.


"""
data_set = [ (train_set_1, valid_set_1), ()....... (train_set_n, valid_set_n) ]

"""

data_set = []

for data in validation_sets:

    training_data = merge_data[merge_data.WEEK_END_DATE <= data['train_set_end_date']]
    validation_data = merge_data[merge_data.WEEK_END_DATE == data['validation_set']]

    data_set.append((training_data, validation_data))
```

MEAN PREDICTION

Now, we will create our first baseline model, `MEAN PREDICTION` . We will use the past data to take average on a group of `STORE_NUM` and `UPC` and use this to predict on the validaion set.

`Evaluation Metric:` Root Mean Squared Log Error

In [16]:

```
# define the function to get the RMSLE

def get_msle(true, predicted) :
    return np.sqrt(msle( true, predicted))
```

In [17]:

```
train_rmsle = []
valid_rmsle = []

for i, data in enumerate(data_set):

    # get the train and validation set
    train, valid = data

    # get the mean prediction dataframe by using a groupby on STORE_NUM and UPC
    mean_prediction = train.groupby(['STORE_NUM', 'UPC'])['UNITS'].mean().reset_index()

    # left join the train and validation set with the mean prediction.
    train = train.merge(mean_prediction, how='left', on=['STORE_NUM', 'UPC'])
    valid = valid.merge(mean_prediction, how='left', on=['STORE_NUM', 'UPC'])

    # In the updated dataframe after the left join,
    # column UNITS_x is the original value of the target variable
    # column UNITS_y is the predicted value of the target variable

    # get the rmsle on train and validation set
    t_rmsle = get_msle(train.UNITS_x, train.UNITS_y)
    v_rmsle = get_msle(valid.UNITS_x, valid.UNITS_y)
    train_rmsle.append(t_rmsle)
    valid_rmsle.append(v_rmsle)

    print('RMSLE ON TRAINING SET: ',i+1, ': ', t_rmsle)
    print('RMSLE ON VALIDATION SET: ',i+1, ': ',v_rmsle)
    print('======================================================================')

# get the mean RMSLE on train and validation set.
print('Mean RMSLE on Train: ', np.mean(train_rmsle))
print('Mean RMSLE on Valid: ', np.mean(valid_rmsle))
```

```
RMSLE ON TRAINING SET:  1 :  0.5902592110738579
RMSLE ON VALIDATION SET:  1 :  0.5887804241752373
======================================================================
RMSLE ON TRAINING SET:  2 :  0.5912639141033686
RMSLE ON VALIDATION SET:  2 :  0.6263169979832923
======================================================================
RMSLE ON TRAINING SET:  3 :  0.5917964778574165
RMSLE ON VALIDATION SET:  3 :  0.4783767090098456
======================================================================
RMSLE ON TRAINING SET:  4 :  0.5914356263311358
RMSLE ON VALIDATION SET:  4 :  0.5811810487862565
======================================================================
RMSLE ON TRAINING SET:  5 :  0.5916390592275275
RMSLE ON VALIDATION SET:  5 :  0.7181642414489362
======================================================================
Mean RMSLE on Train:  0.5912788577186613
Mean RMSLE on Valid:  0.5985638842807136
```

Simple Moving Average

- Now, will use the Simple Moving Average, Like earlier I have used the average over the complete training period. Here, average will be taken on a specified period.
- I will use the predicted value as the average number of UNITS sold in last 8 weeks from a particular store of a particular product.
- As, we have one week gap between the train and validation set.

In [18]:

```
def get_sma(i, train, valid, no_of_weeks=2):

    # create a copy of train and validation set
    train_copy = train.copy()
    valid_copy = valid.copy()

    # group the data by STORE_NUM and UPC and use rolling and mean function to calculate the moving average.
    data_copy = train_copy.groupby(['STORE_NUM','UPC'])['UNITS'].rolling(no_of_weeks).mean().reset_index().set_index('

    # add the moving average column to the train data
    train_copy['moving_average'] = data_copy['UNITS']

    # the last prediction on train set will be used as prediction on validation set.
    # calculate the last_average dataframe by groupby using last function.
    last_average = train_copy.groupby(['STORE_NUM', 'UPC'])['moving_average'].last().reset_index()

    train_copy = train_copy[['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS', 'moving_average']]
    valid_copy = valid_copy[['WEEK_END_DATE','STORE_NUM', 'UPC', 'UNITS']]

    # drop the null values in the dataframe
    train_copy.dropna(inplace=True)
    # merge the validation data with the last_average by left join
    valid_copy = valid_copy.merge(last_average, how= 'left', on= ['STORE_NUM', 'UPC'])

    # calculate the rmsle on train and validation data
    t_rmsle = get_msle(train_copy['UNITS'], train_copy['moving_average'])
    v_rmsle = get_msle(valid_copy['UNITS'], valid_copy['moving_average'])


    print('RMSLE ON TRAINING SET: ',i+1, ': ', t_rmsle)
    print('RMSLE ON VALIDATION SET: ',i+1, ': ',v_rmsle)
    print('======================================================================')

    return t_rmsle, v_rmsle
```

In [19]:

```
train_rmsle_ma = []
valid_rmsle_ma = []

for i, data in enumerate(data_set):
    train, valid = data

    t_rmsle, v_rmsle = get_sma(i,train, valid, no_of_weeks=8)
    train_rmsle_ma.append(t_rmsle)
    valid_rmsle_ma.append(v_rmsle)

print('Mean RMSLE on Train: ', np.mean(train_rmsle_ma))
print('Mean RMSLE on Valid: ', np.mean(valid_rmsle_ma))
```

```
RMSLE ON TRAINING SET:  1 :  0.532302643351482
RMSLE ON VALIDATION SET:  1 :  0.5469206496913668
======================================================================
RMSLE ON TRAINING SET:  2 :  0.5332435318948314
RMSLE ON VALIDATION SET:  2 :  0.6421015703332319
======================================================================
RMSLE ON TRAINING SET:  3 :  0.5335704565359952
RMSLE ON VALIDATION SET:  3 :  0.46149085909723564
======================================================================
RMSLE ON TRAINING SET:  4 :  0.5324889809038001
RMSLE ON VALIDATION SET:  4 :  0.5878031103068386
======================================================================
RMSLE ON TRAINING SET:  5 :  0.5318770729185398
RMSLE ON VALIDATION SET:  5 :  0.7558881602487321
======================================================================
Mean RMSLE on Train:  0.5326965371209298
Mean RMSLE on Valid:  0.598840869935481
```

Linear Regresssion

- Now, we will try one Linear Regression Model and see how it performs on our dataset. We will use the same 5 validation sets and compare the results.
- We will drop the columns like `WEEK_END_DATE`, `STORE_NUM` and `UPC` before training the model.

In [20]:

```
train_rmsle_lr = []
valid_rmsle_lr = []

for i, data in enumerate(data_set):

    train, valid = data

    # drop the columns that are not required, separate the target and independent features
    train_x = train.drop(columns=['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS'])
    train_y = train['UNITS']

    valid_x = valid.drop(columns=['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS'])
    valid_y = valid['UNITS']

    # create an Object of the Linear Regression model
    model_LR = LinearRegression(normalize=True)
    # fit the model with  the training data
    model_LR.fit(train_x, train_y)

    # predict on the training data
    # the model can predict some negative values also and RMSLE only supports positive values.
    # So, we will use the clip function. It will convert all the negative predicted values to 0.
    predict_train = model_LR.predict(train_x).clip(min=0)
    predict_valid = model_LR.predict(valid_x).clip(min=0)

    # get the rmsle on the training and validation data.
    t_rmsle = get_msle(train_y, predict_train)
    v_rmsle = get_msle(valid_y, predict_valid)
    train_rmsle_lr.append(t_rmsle)
    valid_rmsle_lr.append(v_rmsle)

    print('RMSLE ON TRAINING SET: ',i+1, ': ', t_rmsle)
    print('RMSLE ON VALIDATION SET: ',i+1, ': ',v_rmsle)
    print('======================================================================')



print('Mean RMSLE on Train: ', np.mean(train_rmsle_lr))
print('Mean RMSLE on Valid: ', np.mean(valid_rmsle_lr))
```

```
RMSLE ON TRAINING SET:  1 :  0.991102330902912
RMSLE ON VALIDATION SET:  1 :  0.9176909713930113
======================================================================
RMSLE ON TRAINING SET:  2 :  0.9920364929863829
RMSLE ON VALIDATION SET:  2 :  0.8987132671889044
======================================================================
RMSLE ON TRAINING SET:  3 :  0.9926413721401044
RMSLE ON VALIDATION SET:  3 :  0.9634434333577667
======================================================================
RMSLE ON TRAINING SET:  4 :  0.9914855503678094
RMSLE ON VALIDATION SET:  4 :  0.9532581576287663
======================================================================
RMSLE ON TRAINING SET:  5 :  0.9932616371893961
RMSLE ON VALIDATION SET:  5 :  0.9753445782222039
======================================================================
Mean RMSLE on Train:  0.992105476717321
Mean RMSLE on Valid:  0.9416900815581306
```

We can see that `Linear Regression` has performed really bad. Even predicting the mean vaues would be better model. So, it is clear the target variable has no linear dependency on the avaiable features.

Decision Tree

- Now, we will try one Tree Based Model. We will use the same 5 validation sets and compare the results.
- We will drop the columns like `WEEK_END_DATE` , `STORE_NUM` and `UPC` before training the model.

In [21]:

```
train_rmsle_dtr = []
valid_rmsle_dtr = []

for i, data in enumerate(data_set):

    # get the train and validation set
    train, valid = data

    # drop the columns that are not required, separate the target and independent features
    train_x = train.drop(columns=['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS'])
    train_y = train['UNITS']

    valid_x = valid.drop(columns=['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS'])
    valid_y = valid['UNITS']

    # create an Object of DecisionTree Regressor
    model_DTR = DecisionTreeRegressor()
    # fit the model with the training data
    model_DTR.fit(train_x, train_y)

    # predict the target and set the minimum value of the predicted target variable to be 0
    predict_train = model_DTR.predict(train_x).clip(min=0)
    predict_valid = model_DTR.predict(valid_x).clip(min=0)

    # get the rmsle on train and validation set.
    t_rmsle = get_msle(train_y, predict_train)
    v_rmsle = get_msle(valid_y, predict_valid)

    train_rmsle_dtr.append(t_rmsle)
    valid_rmsle_dtr.append(v_rmsle)

    print('RMSLE ON TRAINING SET: ',i+1, ': ', t_rmsle)
    print('RMSLE ON VALIDATION SET: ',i+1, ': ',v_rmsle)
    print('=====================================================================')



print('Mean RMSLE on Train: ', np.mean(train_rmsle_dtr))
print('Mean RMSLE on Valid: ', np.mean(valid_rmsle_dtr))
```

```
RMSLE ON TRAINING SET:  1 :  0.41667563384749384
RMSLE ON VALIDATION SET:  1 :  0.45338621389287065
=====================================================================
RMSLE ON TRAINING SET:  2 :  0.41663996700336603
RMSLE ON VALIDATION SET:  2 :  0.4939311811084024
=====================================================================
RMSLE ON TRAINING SET:  3 :  0.4163555433255185
RMSLE ON VALIDATION SET:  3 :  0.4605479370590595
=====================================================================
RMSLE ON TRAINING SET:  4 :  0.4159250978004229
RMSLE ON VALIDATION SET:  4 :  0.5185600581172628
=====================================================================
RMSLE ON TRAINING SET:  5 :  0.4158593590916643
RMSLE ON VALIDATION SET:  5 :  0.589281722516514
=====================================================================
Mean RMSLE on Train:  0.41629112021369313
Mean RMSLE on Valid:  0.5031414225388218
```

So, we can see that Decision Tree performed way better than the LinearRegression and better than the Mean Prediction.

RandomForest

We just saw that the Decision Tree performed better than the Linear Regression Model. So, we will try one Ensemble Model of Decision Trees like RandomForest and compare the results on the same 5 validation sets.

In [*]:

```
train_rmsle_rfr = []
valid_rmsle_rfr = []

for i, data in enumerate(data_set):
    # get the train and vaidation set
    train, valid = data

    # drop the columns that are not required, separate the target and independent features
    train_x = train.drop(columns=['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS'])
    train_y = train['UNITS']

    valid_x = valid.drop(columns=['WEEK_END_DATE', 'STORE_NUM', 'UPC', 'UNITS'])
    valid_y = valid['UNITS']

    # create an object of the Random Forest Regressor
    model_RFR = RandomForestRegressor(random_state=0)

    # fit the model with the training data
    model_RFR.fit(train_x, train_y)

    # predict the target and set the minimum value of the predicted target variable to be 0
    predict_train = model_RFR.predict(train_x).clip(min=0)
    predict_valid = model_RFR.predict(valid_x).clip(min=0)

    # get the rmsle on train and validate
    t_rmsle = get_msle(train_y, predict_train)
    v_rmsle = get_msle(valid_y, predict_valid)

    train_rmsle_rfr.append(t_rmsle)
    valid_rmsle_rfr.append(v_rmsle)

    print('RMSLE ON TRAINING SET: ',i+1, ': ', t_rmsle)
    print('RMSLE ON VALIDATION SET: ',i+1, ': ',v_rmsle)
    print('======================================================================')

print('Mean RMSLE on Train: ', np.mean(train_rmsle_rfr))
print('Mean RMSLE on Valid: ', np.mean(valid_rmsle_rfr))
```

```
RMSLE ON TRAINING SET:  1 :  0.4217605462439632
RMSLE ON VALIDATION SET:  1 :  0.4391385693265205
======================================================================
RMSLE ON TRAINING SET:  2 :  0.42174972277437306
RMSLE ON VALIDATION SET:  2 :  0.46793496690839687
======================================================================
RMSLE ON TRAINING SET:  3 :  0.42146623952292034
RMSLE ON VALIDATION SET:  3 :  0.4437868022855753
======================================================================
RMSLE ON TRAINING SET:  4 :  0.42102647939541493
RMSLE ON VALIDATION SET:  4 :  0.5089890017503047
======================================================================
```

Conclusions

- We have seen that Tree Based Models have a better performance than other models.
- We have a basic idea of what is the baseline.
- But still, we don't know what should be the right number of validation sets required and what should be the size of the training data.

In [ ]: