In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
import matplotlib
import math
from tqdm import tqdm
import statistics as st
from datetime import date
from sklearn.neighbors import KNeighborsRegressor

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```

```

In [5]:

```
#Loading the train data
train_data = pd.read_csv('train.csv')
train_data.head()
```

Out[5]:

	date	store_nbr	item_nbr	units
0	2012-01-01	1	1	0
1	2012-01-01	1	2	0
2	2012-01-01	1	3	0
3	2012-01-01	1	4	0
4	2012-01-01	1	5	0

In [6]:

```
# removing outliers (look into EDA)
train_data = train_data[train_data.units < 600]
```

In [7]:

```
# log transformation of target variable
def LogTransform(x):
    return math.log(1+x)

train_data['units_new'] = train_data['units'].apply(LogTransform)
train_data.drop(columns=['units'],inplace=True)
train_data.rename(columns={"units_new":"units"},inplace=True)
train_data.head()
```

Out[7]:

	date	store_nbr	item_nbr	units
0	2012-01-01	1	1	0.0
1	2012-01-01	1	2	0.0
2	2012-01-01	1	3	0.0
3	2012-01-01	1	4	0.0
4	2012-01-01	1	5	0.0

In [8]:

```
# https://www.dataquest.io/blog/python-datetime-tutorial/
# Function to get year,month,day,weekday,numeric_value of date
def date_numeric(df):
    print(df.shape)
    f_date = date(2012, 1, 1)
    dateNumeric = []
    year = []
    month = []
    day = []
    weekday = []
    for i in tqdm(range(df.shape[0])):
        date_str = df.iloc[i].date.split("-")
        y = int(date_str[0])
        m = int(date_str[1])
        d = int(date_str[2])
        year.append(y)
        month.append(m)
        day.append(d)
        l_date = date(y,m,d)
        weekday.append(l_date.weekday())
        delta = l_date - f_date
        dateNumeric.append(delta.days)
    return dateNumeric,year,month,day,weekday
```

In [9]:

```
dateNumeric,year,month,day,weekday = date_numeric(train_data)
train_data['date_numeric'] = dateNumeric
train_data['year'] = year
train_data['month'] = month
train_data['day'] = day
train_data['weekday'] = weekday
train_data.head()
```

```
  0%|          | 401/4617598 [00:00<19:13, 4001.89it/s]

(4617598, 4)

100%|██████████| 4617598/4617598 [16:15<00:00, 4733.21it/s]
```

Out[9]:

	date	store_nbr	item_nbr	units	date_numeric	year	month	day	weekday
0	2012-01-01	1	1	0.0	0	2012	1	1	6
1	2012-01-01	1	2	0.0	0	2012	1	1	6
2	2012-01-01	1	3	0.0	0	2012	1	1	6
3	2012-01-01	1	4	0.0	0	2012	1	1	6
4	2012-01-01	1	5	0.0	0	2012	1	1	6

In [11]:

```
# logic to get the list of unique (store,item) combination for which sales are greater than 0.
df = train_data[train_data.units > 0]
df_group = df[['date','store_nbr','item_nbr']].groupby(['store_nbr','item_nbr'],as_index=False).count()
df_group.rename(columns={"date":"count"},inplace=True)

store_item_list = []
for i in range(df_group.shape[0]):
    store = df_group.iloc[i].store_nbr
    item = df_group.iloc[i].item_nbr
    s_i = [store,item]
    store_item_list.append(s_i)
```

In [14]:

```
#function that implements KNN regression to predict sales.
def KNN(datapoint):
    df = train_data[(train_data.store_nbr==datapoint[0]) & (train_data.item_nbr==datapoint[1])]
    if df.shape[0] == 0 :
        return 0.0
    X = df.drop(columns=['date','units'])
    y = df['units'].values
    neigh = KNeighborsRegressor(n_neighbors=14,weights='distance',metric='euclidean')
    neigh.fit(X, y)
    d_n = neigh.predict([datapoint])
    return d_n[0]
```

In [15]:

```
#This function takes raw data as input(as a set of points), does preprocessing and returns final predictions
.
def compute_predictions(test):
    #preprocessing test data
    dateNumeric,year,month,day,weekday = date_numeric(test)
    test['date_numeric'] = dateNumeric
    test['year'] = year
    test['month'] = month
    test['day'] = day
    test['weekday'] = weekday

    print("shape of test data: ",test.shape)

    y_test_pred = []

    # logic to predict sales of test data
    for i in tqdm(range(test.shape[0])):
        dateNumeric= test.iloc[i].date_numeric
        store = test.iloc[i].store_nbr
        item = test.iloc[i].item_nbr
        year = test.iloc[i].year
        month = test.iloc[i].month
        day = test.iloc[i].day
        weekday = test.iloc[i].weekday

        datapoint = [store,item,dateNumeric,year,month,day,weekday]

        s_i = [store,item]
        if s_i not in store_item_list:
            y_test_pred.append(0.0)
        else:
            d = KNN(datapoint)
            y_test_pred.append(d)

    y_test_pred_exp = []
    for i in y_test_pred:
        val = ((math.exp(i)) - 1)
        y_test_pred_exp.append(val)

    return y_test_pred_exp
```

In [ ]:

```

```

In [16]:

```
# function to calculate the metric. The metric used is Root Mean Squared Logarithmic Error.
def rmsle(y_true,y_pred):
    s=0
    for i in range(0,y_true.shape[0]):
        a = math.log(y_pred[i]+1) - math.log(y[i]+1)
        a = y_pred[i] - y_true[i]
        s = s+(a*a)
    s = s/len(y_true)
    result = math.sqrt(s)
    return result
```

In [17]:

```
#This function takes raw data as input(set of points) along with target values
#does preprocessing and makes final predictions through compute_predictions function
#It also returns the metric value which is Root Mean Squared Logarithmic Error.
def compute_predictions_and_metric(test_x,y_test):
    y_test_pred = compute_predictions(test_x)
    return rmsle(y_test,y_test_pred)
```

In [ ]:

```

```

In [20]:

```
#testing on a sample train data
train = pd.read_csv('train.csv')
test = train.sample(100000)
X = test.drop(columns=['units'])
y = test.units.values
test_rmsle = compute_predictions_and_metric(X,y)
print("\n")
print("rmsle value on a sample train data: ",test_rmsle)
```

```
  0%|          | 492/100000 [00:00<00:20, 4917.19it/s]

(100000, 3)

100%|██████████| 100000/100000 [00:20<00:00, 4942.88it/s]
  0%|          | 33/100000 [00:00<05:05, 327.01it/s]

shape of test data:  (100000, 8)

100%|██████████| 100000/100000 [04:05<00:00, 407.75it/s]


rmsle value on a sample train data:  2.2265044867901985e-15
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```