1.Importing Libraries

In [ ]:

```
import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
import seaborn as sns; sns.set()
import lightgbm as lgb
from sklearn import preprocessing, metrics
from sklearn.preprocessing import LabelEncoder
import gc
import os
import time
from scipy.sparse import csr_matrix
from scipy.stats import poisson
from joblib import Parallel, delayed
from tqdm import tqdm_notebook as tqdm
from math import ceil
%env JOBLIB_TEMP_FOLDER=/tmp
```

Useful links

- https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163216
- https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/174371

2. Memory reduction function

In [ ]:

```
# Helper function to reduce the memory usage for all the given data sets

def mem_usage_reduction(df):

    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.float32)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
```

```
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    print('Mem. usage decreased to {:5.2f} Mb from {:5.2f} ({:.1f}%
reduction)'.format(end_mem,start_mem, 100 * (start_mem - end_mem) / start_mem))
    return df
```

3. Melting and Merging the data

In [ ]:

```
def changing_data(calendar,sell_prices,sales_train, sample_submission,nrows):

    # We are melting the sales_train data based on days('d_x')
    sales_train = pd.melt(sales_train, id_vars = ['id','item_id','dept_id','cat_id','store_id','sta
te_id'],
                            var_name = 'd',value_name = 'unit_sales')


    # Creating the final evaluation data
    evaluation_rows = [row for row in sample_submission['id'] if 'evaluation' in row]
    evaluation_data = sample_submission[sample_submission['id'].isin(evaluation_rows)]

    # Changing the column names to the respective daywise representations
    id2 = ['id']
    day_eval_columns = [f'd_{row}' for row in range(1942,1970)]
    id2.extend(day_eval_columns)
    evaluation_data.columns = id2


    # Product id's table
    product_ids =
sales_train[['id','item_id','dept_id','cat_id','store_id','state_id']].drop_duplicates()


    # merging evaluation data with product_ids columns
    evaluation_data = evaluation_data.merge(product_ids, how ='left', on='id')


    # Melting the evaluation data

    evaluation_data = pd.melt(evaluation_data, id_vars = ['id', 'item_id', 'dept_id','cat_id', 'sto
re_id','state_id'],
                          var_name = 'd', value_name = 'unit_sales')

    # Adding a columns that separates train and evaluation
    sales_train['data'] = 'train'
    evaluation_data['data'] = 'evaluation'


    # Concatenating train and evaluation
    data = pd.concat([sales_train,evaluation_data],axis=0)

    del sales_train, evaluation_data


    data = mem_usage_reduction(data)

    # Taking only a subset of data for fast training

    data = data.iloc[nrows:]

    # Adding sell_prices data to the train data along with new columns

    sell_prices['price_max']     =sell_prices.groupby(['store_id','item_id'])['sell_price'].transfo
rm('max')
    sell_prices['price_min']     =sell_prices.groupby(['store_id','item_id'])['sell_price'].transfo
rm('min')
    sell_prices['price_std']     =sell_prices.groupby(['store_id','item_id'])['sell_price'].transfo
rm('std')
    sell_prices['price_mean']    =sell_prices.groupby(['store_id','item_id'])['sell_price'].transfo
rm('mean')
    sell_prices['price_max']     =sell_prices['sell_price']/sell_prices['price_max']
    sell_prices['price_nunique'] =sell_prices.groupby(['store_id','item_id'])
```

```
    sell_prices['price_nunique'] =sell_prices.groupby(['store_id','item_id'])
['sell_price'].transform('nunique')
    sell_prices['item_nunique']  =sell_prices.groupby(['store_id','sell_price'])['item_id'].transfo
rm('nunique')

    calendar_prices = calendar[['wm_yr_wk','month','year']]
    calendar_prices = calendar_prices.drop_duplicates(subset=['wm_yr_wk'])
    sell_prices = sell_prices.merge(calendar_prices[['wm_yr_wk','month','year']],on=['wm_yr_wk'],ho
w='left')

    del calendar_prices

    #sell_prices['price_momentum']=
sell_prices['sell_price']/sell_prices.groupby(['store_id','item_id'])
['sell_price'].transform(lambda x:x.shift(1))
    #sell_prices['price_momentum_m']=
sell_prices['sell_price']/sell_prices.groupby(['store_id','item_id','month'])
['sell_price'].transform('mean')
    #sell_prices['price_momentum_y']=
sell_prices['sell_price']/sell_prices.groupby(['store_id','item_id','year'])
['sell_price'].transform('mean')

    #sell_prices[['store_id','item_id','release']] = sell_prices.groupby(['store_id','item_id'])['
wm_yr_wk'].agg(['min']).reset_index()
    #release_df.columns = ['store_id','item_id','release']
    #d = data[['store_id','item_id']]
    #d = d.merge(sell_prices[['store_id','item_id','release']],on=
['store_id','item_id'],how='left')
    #new_columns = [col for col in list(d) if col not in ['store_id','item_id']]
    #data = pd.concat([data,d[new_columns]])
    #del d,new_columns

    # Dropping few features from calender
    calendar.drop(["weekday","wday","month","year"], inplace = True, axis =1)

    data = pd.merge(data, calendar, how = 'left',on = ['d'])
    data.drop(['d'], inplace = True, axis = 1)

    data = data.merge(sell_prices, on=['store_id', 'item_id','wm_yr_wk'], how = 'left')

    del calendar,sell_prices

    gc.collect()
    return data


#4390560   for 5 years data
#15519410  for 4 years data
#26648260  for 3 years data
#37777110  for 2 years data
#48905960  for 1 years data
#60034810 total
```

4. Vectorization - Label Encoding

In [ ]:

```
def transforming_data(data):
    nan_features = ['event_name_1','event_type_1','event_name_2','event_type_2']
    for i in nan_features:
        data[i].fillna('no_event',inplace=True)

    encoder = preprocessing.LabelEncoder()
    data['encoded_id']=encoder.fit_transform(data['id'])

    cat_feat = ['item_id', 'dept_id', 'cat_id', 'store_id', 'state_id',
                'event_name_1', 'event_type_1', 'event_name_2', 'event_type_2']

    for fe in cat_feat:
        data[fe]=encoder.fit_transform(data[fe])

    return data
```

5. Feature Engineering

In [ ]:

```
def feature_engineering(df):

    # Adding shift and rolling_mean features
    lag_list = [7,8,9,14,15,16,21,22,23,28,29,30]
    rolling_list = [7,14,30]

    for val in lag_list:
        df[f"lag_d_{val}"]=df.groupby(['id'])['unit_sales'].transform(lambda x:x.shift(val))
    #print('done1')
    for val in rolling_list:
        df[f"r_std_d{val}"] = df.groupby(["id"])["unit_sales"].transform(lambda x:x.shift(28).rolli
ng(val).std())
    #print('done2')
    for val in rolling_list:
        df[f"r_mean_d{val}"] = df.groupby(['id'])['unit_sales'].transform(lambda x:x.shift(28).roll
ing(val).mean())
    #print('done3')


    # time features
    df['date'] = pd.to_datetime(df['date'])


    df['tm_d']=df['date'].dt.day.astype(np.int8)
    df['tm_w']=df['date'].dt.week.astype(np.int8)
    df['tm_m']=df['date'].dt.month.astype(np.int8)
    df['tm_y']=df['date'].dt.year
    df['tm_y']=(df['tm_y']-df['tm_y'].min()).astype(np.int8)
    df['tm_wm']=df['tm_d'].apply(lambda x: ceil(x/7)).astype(np.int8)

    df['tm_dw']=df['date'].dt.dayofweek.astype(np.int8)
    df['tm_w_end']=(df['tm_dw']>=5).astype(np.int8)

    return df
```

6.Reading the Data and Memory Reduction

In [ ]:

```
%%time
calendar = pd.read_csv("../input/m5-forecasting-accuracy/calendar.csv")
calendar['date']=pd.to_datetime(calendar['date'])
calendar = mem_usage_reduction(calendar)
sales_train = pd.read_csv("../input/m5-forecasting-accuracy/sales_train_evaluation.csv")
sales_train = mem_usage_reduction(sales_train)
sell_prices = pd.read_csv("../input/m5-forecasting-accuracy/sell_prices.csv")
sell_prices = mem_usage_reduction(sell_prices)
sample_submission = pd.read_csv('../input/m5-forecasting-accuracy/sample_submission.csv')
```

In [ ]:

```
%%time
data = changing_data(calendar,sell_prices,sales_train,sample_submission,nrows=26648260)
gc.collect()

# Memory reduction function
data = mem_usage_reduction(data)

# Data transformation
data = transforming_data(data)
gc.collect()

# Memory reduction function
data = mem_usage_reduction(data)

# Addition of new features
data = feature_engineering(data)
gc.collect()
```

```
# Memory reduction function
data = mem_usage_reduction(data)
```

In [ ]:

```
data = data.drop(['month','year','data'],axis=1)
```

In [ ]:

```
data.info()
```

7. Missing value percentage

In [ ]:

```
def nan_values(data):
    total = data.isnull().sum().sort_values(ascending = False)
    percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascending = False)
    missing_train_data  = pd.concat([total, percent], axis=1, keys=['Total', 'Missing_Percentage'])
    print(missing_train_data.head(29))
nan_values(data)
```

8. Saving the data

In [ ]:

```
data.reset_index(drop=True,inplace=True)
data.to_hdf('data.h5',key='data')
```

Summary

3. Melting and Merging the data

- Melting the data based on 'd'(days) - sales_train
- Creating evaluation data for 28 days and melted it based on 'd'(days) - evaluation_data
- Concatenating the evaluation data to the melted sales_data. (data)
- Taken 3 years data starting from 26648260 till the end.(Saving the memory)
- Creating new features in sell_prices and merging with calendar.
- Merging calendar and sell_prices features with the data.

4. Vectorization - Label Encoding

- Removing nan values and replacing them with a category.
- Label Encoding all the categorical features.

5. Feature Engineering

- Adding new features based on unit_sales.
- lag features with multiple shifts and rolling features of mean and standard deviation with a 28 days shift.
- New time features- day,week,month,year and weekend are formed based on the 'date' feature.

6.Reading the Data and Memory Reduction

- Reading the data of calendar,sales_train,sell_prices and sample_submission from the competition data.
- Reducing the memory usage with mem_usage_reduction function.

8. Saving the data

- Saving the final data to hdf file for further usage of modelling.