### Fetching the data

In [ ]:

```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:

```
{'kaggle.json': b'{"username":"sidd1996","key":"411008e9f7f47955a0869b805c7a724
0"}'}
```

In [ ]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!ls ~/.kaggle
!chmod 600 /root/.kaggle/kaggle.json
```

```
kaggle.json
```

In [ ]:

```
!kaggle competitions download -c m5-forecasting-accuracy
```

```
Warning: Looks like you're using an outdated API Version, please consider updat
ing (server 1.5.10 / client 1.5.4)
Downloading sales_train_validation.csv.zip to /content
 32% 5.00M/15.5M [00:00<00:00, 49.5MB/s]
100% 15.5M/15.5M [00:00<00:00, 99.4MB/s]
Downloading sales_train_evaluation.csv.zip to /content
 57% 9.00M/15.8M [00:00<00:00, 42.2MB/s]
100% 15.8M/15.8M [00:00<00:00, 52.7MB/s]
Downloading sample_submission.csv.zip to /content
  0% 0.00/163k [00:00<?, ?B/s]
100% 163k/163k [00:00<00:00, 51.3MB/s]
Downloading calendar.csv to /content
  0% 0.00/101k [00:00<?, ?B/s]
100% 101k/101k [00:00<00:00, 91.2MB/s]
Downloading sell_prices.csv.zip to /content
 77% 11.0M/14.2M [00:00<00:00, 43.8MB/s]
100% 14.2M/14.2M [00:00<00:00, 47.4MB/s]
```

In [ ]:

```
!unzip -q /content/sales_train_validation.csv.zip
!unzip -q /content/sell_prices.csv.zip
!unzip -q /content/sales_train_evaluation.csv.zip
!unzip -q /content/sample_submission.csv.zip
```

In [ ]:

```
!pip install dask
!pip install 'fsspec>=0.3.3'
!pip install partd
```

```
Requirement already satisfied: dask in /usr/local/lib/python3.7/dist-packages
(2.12.0)
Collecting fsspec>=0.3.3
  Downloading https://files.pythonhosted.org/packages/91/0d/a6bfee0ddf47b254286
b9bd574e6f50978c69897647ae15b14230711806e/fsspec-0.8.7-py3-none-any.whl (http
s://files.pythonhosted.org/packages/91/0d/a6bfee0ddf47b254286b9bd574e6f50978c69
897647ae15b14230711806e/fsspec-0.8.7-py3-none-any.whl) (103kB)
     |████████████████████████████████| 112kB 5.1MB/s
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /u
sr/local/lib/python3.7/dist-packages (from fsspec>=0.3.3) (3.7.0)
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8"
in /usr/local/lib/python3.7/dist-packages (from importlib-metadata; python_vers
ion < "3.8"->fsspec>=0.3.3) (3.7.4.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packa
ges (from importlib-metadata; python_version < "3.8"->fsspec>=0.3.3) (3.4.0)
Installing collected packages: fsspec
Successfully installed fsspec-0.8.7
Collecting partd
  Downloading https://files.pythonhosted.org/packages/44/e1/68dbe731c9c067655bf
f1eca5b7d40c20ca4b23fd5ec9f3d17e201a6f36b/partd-1.1.0-py3-none-any.whl (http
s://files.pythonhosted.org/packages/44/e1/68dbe731c9c067655bff1eca5b7d40c20ca4b
23fd5ec9f3d17e201a6f36b/partd-1.1.0-py3-none-any.whl)
Collecting locket
  Downloading https://files.pythonhosted.org/packages/50/b8/e789e45b9b9c2db75e9
d9e6ceb022c8d1d7e49b2c085ce8c05600f90a96b/locket-0.2.1-py2.py3-none-any.whl (ht
tps://files.pythonhosted.org/packages/50/b8/e789e45b9b9c2db75e9d9e6ceb022c8d1d7
e49b2c085ce8c05600f90a96b/locket-0.2.1-py2.py3-none-any.whl)
Requirement already satisfied: toolz in /usr/local/lib/python3.7/dist-packages
 (from partd) (0.11.1)
Installing collected packages: locket, partd
Successfully installed locket-0.2.1 partd-1.1.0
```

In [ ]:

```
import os
# import gc
import time
import math
import datetime
from math import log, floor
# from sklearn.neighbors import KDTree

import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.utils import shuffle
from tqdm.notebook import tqdm as tqdm
from sklearn.externals import joblib
import pickle

import seaborn as sns
from matplotlib import colors
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder

# import pywt
from statsmodels.robust import mad

import scipy
import statsmodels
from scipy import signal
import statsmodels.api as sm
from fbprophet import Prophet
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import gc

import warnings
warnings.filterwarnings("ignore")

plt.style.use('seaborn')
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/joblib/__init__.py:15:
FutureWarning: sklearn.externals.joblib is deprecated in 0.21 and will be remov
ed in 0.23. Please import this functionality directly from joblib, which can be
installed with: pip install joblib. If this warning is raised when loading pick
led models, you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: Future
Warning:

pandas.util.testing is deprecated. Use the functions in the public API at panda
s.testing instead.
```

In [ ]:

```

```

In [ ]:

```
sales = pd.read_csv('/content/sales_train_evaluation.csv')
calendar = pd.read_csv('/content/calendar.csv')
sell_prices = pd.read_csv('/content/sell_prices.csv')
```

In [ ]:

```
# Ref. link :- https://www.kaggle.com/anshuls235/time-series-forecasting-eda-fe-m
#Add zero sales for the remaining days 1942-1969
for d in range(1942,1970):
    col = 'd_' + str(d)
    sales[col] = 0
    sales[col] = sales[col].astype(np.int16)
```

# Business Problem :-

M5 Forecasting Accuracy is a competetion in which we have to forecast future sales of each product in each store based on the hierarchical sales data provided by Walmart. In this competetion we have to forecast daily sales for next 28 days. Here we have the data for 3 states in US(California, Texas, and Wisconsin). The data files (.csv files) provided for the competetion consists of item level, department, product categories,items sold on a day, store details, price, promotions, day of the week, and special events. So by using this data we will forecast daily sales for next 28 days as accurately as possible.

# ML formulation :-

We will do some data preprocessing and feature engineering to get desired format and some new features respectively . Once the data is ready we will pass it through different machine learning and deep learning models . After the model is trained we will predict the values for test dataset. We will pose this as a supervised machine learning regression problem. In this problem we will be using LGBMRegressor, Facebook Prophet and a deep learning model.

# Metrics :-

The performance measures are first computed for each series separately by averaging their values across the forecasting horizon and then averaged again across the series in a weighted fashion .

Forcasting horizon or number of days for which forecast is required is 28 days.

The metric used for evaluating the accuracy of the each series is Root Mean Squared Scaled Error (RMSSE).

After estimating the RMSSE for all the 42,840 time series of the competition, we will calculate Weighted RMSSE (WRMSSE) which will be used as our final metric .

The formulas for RMSSE and WRMSSE are given below :-

$$RMSSE = \sqrt{\frac{1}{h}\frac{\sum_{t=n+1}^{n+h}(Y_t - \widehat{Y}_t)^2}{\frac{1}{n-1}\sum_{t=2}^{n}(Y_t - Y_{t-1})^2}}, \qquad WRMSSE = \sum_{i=1}^{42,840} w_i * RMSSE$$

RMSSE variables :- Y_t is the actual future value of the examined time series at point t, (Y_t^)the generated forecast, n the length of the training sample (number of historical observations), and h the forecasting horizon.

WRMSSE variables :- w_i is the weight of the i_th series of the competition. A lower WRMSSE score is better.Explaination on how to calculate w_i is given in the pdf present in M5 Participants Guide :- https://mofc.unic.ac.cy/m5-competition/ (https://mofc.unic.ac.cy/m5-competition/) .

## Downcasting

In [ ]:
```
### Ref link :- https://www.kaggle.com/anshuls235/time-series-forecasting-eda-fe-


#Downcast in order to save memory
def downcast(df):
    cols = df.dtypes.index.tolist()
    types = df.dtypes.values.tolist()
    for i,t in enumerate(types):
        if 'int' in str(t):
            if df[cols[i]].min() > np.iinfo(np.int8).min and df[cols[i]].max() <
                df[cols[i]] = df[cols[i]].astype(np.int8)
            elif df[cols[i]].min() > np.iinfo(np.int16).min and df[cols[i]].max()
                df[cols[i]] = df[cols[i]].astype(np.int16)
            elif df[cols[i]].min() > np.iinfo(np.int32).min and df[cols[i]].max()
                df[cols[i]] = df[cols[i]].astype(np.int32)
            else:
                df[cols[i]] = df[cols[i]].astype(np.int64)
        elif 'float' in str(t):
            if df[cols[i]].min() > np.finfo(np.float16).min and df[cols[i]].max()
                df[cols[i]] = df[cols[i]].astype(np.float16)
            elif df[cols[i]].min() > np.finfo(np.float32).min and df[cols[i]].max
                df[cols[i]] = df[cols[i]].astype(np.float32)
            else:
                df[cols[i]] = df[cols[i]].astype(np.float64)
        elif t == np.object:
            if cols[i] == 'date':
                df[cols[i]] = pd.to_datetime(df[cols[i]], format='%Y-%m-%d')
            else:
                df[cols[i]] = df[cols[i]].astype('category')
    return df
```

In [ ]:
```
sales = downcast(sales)
sell_prices = downcast(sell_prices)
calendar = downcast(calendar)
```

In [ ]:

In [ ]:
```
sales.head()
```

Out[11]:

| | id | item_id | dept_id | cat_id | store_id | state_id | d_1 |
|---|---|---|---|---|---|---|---|
| **0** | HOBBIES_1_001_CA_1_evaluation | HOBBIES_1_001 | HOBBIES_1 | HOBBIES | CA_1 | CA | 0 |
| **1** | HOBBIES_1_002_CA_1_evaluation | HOBBIES_1_002 | HOBBIES_1 | HOBBIES | CA_1 | CA | 0 |
| **2** | HOBBIES_1_003_CA_1_evaluation | HOBBIES_1_003 | HOBBIES_1 | HOBBIES | CA_1 | CA | 0 |
| **3** | HOBBIES_1_004_CA_1_evaluation | HOBBIES_1_004 | HOBBIES_1 | HOBBIES | CA_1 | CA | 0 |
| **4** | HOBBIES_1_005_CA_1_evaluation | HOBBIES_1_005 | HOBBIES_1 | HOBBIES | CA_1 | CA | 0 |

5 rows × 1975 columns

In [ ]:
```
sales = pd.melt(sales, id_vars=['id', 'item_id', 'dept_id', 'cat_id', 'store_id',
```

In [ ]:
```
sales
```

Out[13]:

| | id | item_id | dept_id | cat_id | store_id | state_ |
|---|---|---|---|---|---|---|
| **0** | HOBBIES_1_001_CA_1_evaluation | HOBBIES_1_001 | HOBBIES_1 | HOBBIES | CA_1 | C |
| **1** | HOBBIES_1_002_CA_1_evaluation | HOBBIES_1_002 | HOBBIES_1 | HOBBIES | CA_1 | C |
| **2** | HOBBIES_1_003_CA_1_evaluation | HOBBIES_1_003 | HOBBIES_1 | HOBBIES | CA_1 | C |
| **3** | HOBBIES_1_004_CA_1_evaluation | HOBBIES_1_004 | HOBBIES_1 | HOBBIES | CA_1 | C |
| **4** | HOBBIES_1_005_CA_1_evaluation | HOBBIES_1_005 | HOBBIES_1 | HOBBIES | CA_1 | C |
| **...** | ... | ... | ... | ... | ... | |
| **60034805** | FOODS_3_823_WI_3_evaluation | FOODS_3_823 | FOODS_3 | FOODS | WI_3 | W |
| **60034806** | FOODS_3_824_WI_3_evaluation | FOODS_3_824 | FOODS_3 | FOODS | WI_3 | W |
| **60034807** | FOODS_3_825_WI_3_evaluation | FOODS_3_825 | FOODS_3 | FOODS | WI_3 | W |
| **60034808** | FOODS_3_826_WI_3_evaluation | FOODS_3_826 | FOODS_3 | FOODS | WI_3 | W |
| **60034809** | FOODS_3_827_WI_3_evaluation | FOODS_3_827 | FOODS_3 | FOODS | WI_3 | W |

60034810 rows × 8 columns

In [ ]:

```
sell_prices
```

Out[14]:

```
        store_id       item_id  wm_yr_wk  sell_price
      0     CA_1   HOBBIES_1_001     11325    9.578125
      1     CA_1   HOBBIES_1_001     11326    9.578125
      2     CA_1   HOBBIES_1_001     11327    8.257812
      3     CA_1   HOBBIES_1_001     11328    8.257812
      4     CA_1   HOBBIES_1_001     11329    8.257812
    ...      ...             ...       ...         ...
6841116     WI_3     FOODS_3_827     11617    1.000000
6841117     WI_3     FOODS_3_827     11618    1.000000
6841118     WI_3     FOODS_3_827     11619    1.000000
6841119     WI_3     FOODS_3_827     11620    1.000000
6841120     WI_3     FOODS_3_827     11621    1.000000

6841121 rows × 4 columns
```

In [ ]:
```
calendar
```

Out[15]:

|  | date | wm_yr_wk | weekday | wday | month | year | d | event_name_1 | event_type_1 | ev |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2011-01-29 | 11101 | Saturday | 1 | 1 | 2011 | d_1 | NaN | NaN |  |
| **1** | 2011-01-30 | 11101 | Sunday | 2 | 1 | 2011 | d_2 | NaN | NaN |  |
| **2** | 2011-01-31 | 11101 | Monday | 3 | 1 | 2011 | d_3 | NaN | NaN |  |
| **3** | 2011-02-01 | 11101 | Tuesday | 4 | 2 | 2011 | d_4 | NaN | NaN |  |
| **4** | 2011-02-02 | 11101 | Wednesday | 5 | 2 | 2011 | d_5 | NaN | NaN |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **1964** | 2016-06-15 | 11620 | Wednesday | 5 | 6 | 2016 | d_1965 | NaN | NaN |  |
| **1965** | 2016-06-16 | 11620 | Thursday | 6 | 6 | 2016 | d_1966 | NaN | NaN |  |
| **1966** | 2016-06-17 | 11620 | Friday | 7 | 6 | 2016 | d_1967 | NaN | NaN |  |
| **1967** | 2016-06-18 | 11621 | Saturday | 1 | 6 | 2016 | d_1968 | NaN | NaN |  |
| **1968** | 2016-06-19 | 11621 | Sunday | 2 | 6 | 2016 | d_1969 | NBAFinalsEnd | Sporting |  |

1969 rows × 14 columns

In [ ]:
```
df = pd.merge(sales, calendar,how = "left",on = 'd')
```

In [ ]:
```
df = pd.merge(df, sell_prices, how = 'left', on = ['store_id','item_id','wm_yr_wk
```

In [ ]:
```
df['sell_price'].fillna(0 , inplace = True)
```

In [ ]:
```
df['revenue'] = df['sold'] * df['sell_price']
```

In [ ]:
```
df
```

Out[20]:
```
                                   id        item_id    dept_id   cat_id  store_id  state_
        0  HOBBIES_1_001_CA_1_evaluation  HOBBIES_1_001  HOBBIES_1  HOBBIES      CA_1       (
        1  HOBBIES_1_002_CA_1_evaluation  HOBBIES_1_002  HOBBIES_1  HOBBIES      CA_1       (
        2  HOBBIES_1_003_CA_1_evaluation  HOBBIES_1_003  HOBBIES_1  HOBBIES      CA_1       (
        3  HOBBIES_1_004_CA_1_evaluation  HOBBIES_1_004  HOBBIES_1  HOBBIES      CA_1       (
        4  HOBBIES_1_005_CA_1_evaluation  HOBBIES_1_005  HOBBIES_1  HOBBIES      CA_1       (
      ...                            ...            ...        ...      ...       ...
 60034805    FOODS_3_823_WI_3_evaluation    FOODS_3_823    FOODS_3    FOODS      WI_3       \
 60034806    FOODS_3_824_WI_3_evaluation    FOODS_3_824    FOODS_3    FOODS      WI_3       \
 60034807    FOODS_3_825_WI_3_evaluation    FOODS_3_825    FOODS_3    FOODS      WI_3       \
 60034808    FOODS_3_826_WI_3_evaluation    FOODS_3_826    FOODS_3    FOODS      WI_3       \
 60034809    FOODS_3_827_WI_3_evaluation    FOODS_3_827    FOODS_3    FOODS      WI_3       \

60034810 rows × 23 columns
```

In [ ]:

In [ ]:

## Feature Engineering

In [ ]:
```
df = downcast(df)
```

In [ ]:

```
In [ ]: df
```

Out[26]:

| | id | item_id | dept_id | cat_id | store_id | state_ |
|---|---|---|---|---|---|---|
| **0** | HOBBIES_1_001_CA_1_evaluation | HOBBIES_1_001 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **1** | HOBBIES_1_002_CA_1_evaluation | HOBBIES_1_002 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **2** | HOBBIES_1_003_CA_1_evaluation | HOBBIES_1_003 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **3** | HOBBIES_1_004_CA_1_evaluation | HOBBIES_1_004 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **4** | HOBBIES_1_005_CA_1_evaluation | HOBBIES_1_005 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **...** | ... | ... | ... | ... | ... | |
| **60034805** | FOODS_3_823_WI_3_evaluation | FOODS_3_823 | FOODS_3 | FOODS | WI_3 | \ |
| **60034806** | FOODS_3_824_WI_3_evaluation | FOODS_3_824 | FOODS_3 | FOODS | WI_3 | \ |
| **60034807** | FOODS_3_825_WI_3_evaluation | FOODS_3_825 | FOODS_3 | FOODS | WI_3 | \ |
| **60034808** | FOODS_3_826_WI_3_evaluation | FOODS_3_826 | FOODS_3 | FOODS | WI_3 | \ |
| **60034809** | FOODS_3_827_WI_3_evaluation | FOODS_3_827 | FOODS_3 | FOODS | WI_3 | \ |

60034810 rows × 23 columns

## Lags :-

The past values are known as lags. We use lags to get the correlation between present values and past values, as past values are important to make predictions.

We will get Lag values for 1 day , 7 days and 30 days to get daily , weekly and monthly lags respectively.

In [ ]:

```
# Ref. link :- https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/15025

df['lag_7'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(7))

df['lag_21'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(21))

df['lag_28'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(28))

df['lag_35'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(35))

df['lag_42'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(42))
```

In [ ]:

```
## Ref.link :- https://www.kaggle.com/nemuritarinai/m5-accuracy

df['price_lag'] = df.groupby(['id', 'item_id', 'dept_id', 'cat_id', 'store_id', '

df['price-diff']=df['price_lag']-df['sell_price']

df.drop(['price_lag'], axis=1, inplace=True)
```

### Rolling / Sliding Window :-

In this we use the past values to calculate statistical values. This is called rolling/sliding window because the window which is used to calculate these statistical value changes for every datapoint.

In [ ]:

```
window = 28
```

For every item in every store

In [ ]:

```
# Mean
df['rolling_sold_mean_7'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(wi

df['rolling_sold_mean_14'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(w

df['rolling_sold_mean_28'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(w

df['rolling_sold_mean_90'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(w


# Max
df['rolling_sold_max_7'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(win

df['rolling_sold_max_28'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(wi

# df['rolling_sold_max_365'] = df.groupby(['id'])['sold'].apply(lambda x: x.rolli

# STD
df['rolling_sold_std_7'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(win

df['rolling_sold_std_28'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(wi

df['rolling_sold_std_90'] = df.groupby(['id'])['sold'].apply(lambda x: x.shift(wi
```

In [ ]:

```
# # Mean
# df['rolling_revenue_mean_7'] = df.groupby(['id'])['revenue'].apply(lambda x: x.

df['rolling_revenue_mean_28'] = df.groupby(['id'])['revenue'].apply(lambda x: x.s

# df['rolling_revenue_mean_90'] = df.groupby(['id'])['revenue'].apply(lambda x: x


# # Max
# df['rolling_revenue_max_7'] = df.groupby(['id'])['revenue'].apply(lambda x: x.r

df['rolling_revenue_max_28'] = df.groupby(['id'])['revenue'].apply(lambda x: x.sh

# df['rolling_revenue_max_365'] = df.groupby(['id'])['revenue'].apply(lambda x: x

# # STD
# df['rolling_revenue_std_7'] = df.groupby(['id'])['revenue'].apply(lambda x: x.r

df['rolling_revenue_std_28'] = df.groupby(['id'])['revenue'].apply(lambda x: x.sh

# df['rolling_revenue_std_90'] = df.groupby(['id'])['revenue'].apply(lambda x: x.
```

In [ ]:
```
# # Mean

# df['rolling_item_revenue_mean_7'] = df.groupby(['item_id'])['revenue'].apply(la

df['rolling_item_revenue_mean_28'] = df.groupby(['item_id'])['revenue'].apply(lam
```

In [ ]:
```
# # Mean

df['rolling_item_sold_mean_7'] = df.groupby(['item_id'])['sold'].apply(lambda x:

df['rolling_item_sold_mean_28'] = df.groupby(['item_id'])['sold'].apply(lambda x:
```

In [ ]:
```
# # Mean

df['rolling_dept_sold_mean_28'] = df.groupby(['dept_id'])['sold'].apply(lambda x:

df['rolling_store_sold_mean_28'] = df.groupby(['store_id'])['sold'].apply(lambda
```

Some other statistical features

In [ ]:
```
# Average

df['item_sold_avg'] = df.groupby('item_id')['sold'].transform('mean').astype(np.f
df['store_sold_avg'] = df.groupby('store_id')['sold'].transform('mean').astype(np
df['state_sold_avg'] = df.groupby('state_id')['sold'].transform('mean').astype(np


df['store_item_sold_avg'] = df.groupby(['store_id','item_id'])['sold'].transform(

df['cat_item_sold_avg'] = df.groupby(['cat_id','item_id'])['sold'].transform('mea

df['state_item_sold_avg'] = df.groupby(['state_id','item_id'])['sold'].transform(

df['store_weekday_sold_avg'] = df.groupby(['store_id','weekday'])['sold'].transfo
```

In [ ]:
```
# Ref. link :- https://www.kaggle.com/kyakovlev/m5-simple-fe
df['max_price'] = df.groupby(['item_id'])['sell_price'].transform('max').astype(n
df['min_price'] = df.groupby(['item_id'])['sell_price'].transform('min').astype(n

df['price_mean'] = df.groupby(['item_id'])['sell_price'].transform('mean').astype

df['price_std'] = df.groupby(['item_id'])['sell_price'].transform('std').astype(n

df['price_norm'] =  df['sell_price']/df['max_price']
```

In [ ]:

```
## Ref.link :- https://www.kaggle.com/nemuritarinai/m5-accuracy

df['daily_avg_sold'] = df.groupby(['id', 'item_id', 'dept_id', 'cat_id', 'store_i

df['avg_sold'] = df.groupby(['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 's


df['selling_trend'] = (df['daily_avg_sold'] - df['avg_sold']).astype(np.float16)
```

In [ ]:

## Expanding window :-

This is an advanced version of rolling windows. In this the window which is used to calculate statistical values increases. The idea behind expanding window is that it takes all the past values into account.

In [ ]:

```
df['expanding_sold_mean'] = df.groupby(['id'])['sold'].apply(lambda x: x.expandin
```

In [ ]:

```
df['expanding_revenue_mean'] = df.groupby(['id'])['revenue'].apply(lambda x: x.ex
```

In [ ]:

In [ ]:

```
df
```

Out[42]:

| | id | item_id | dept_id | cat_id | store_id | state_ |
|---|---|---|---|---|---|---|
| **0** | HOBBIES_1_001_CA_1_evaluation | HOBBIES_1_001 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **1** | HOBBIES_1_002_CA_1_evaluation | HOBBIES_1_002 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **2** | HOBBIES_1_003_CA_1_evaluation | HOBBIES_1_003 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **3** | HOBBIES_1_004_CA_1_evaluation | HOBBIES_1_004 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **4** | HOBBIES_1_005_CA_1_evaluation | HOBBIES_1_005 | HOBBIES_1 | HOBBIES | CA_1 | ( |
| **...** | ... | ... | ... | ... | ... | |
| **60034805** | FOODS_3_823_WI_3_evaluation | FOODS_3_823 | FOODS_3 | FOODS | WI_3 | \ |
| **60034806** | FOODS_3_824_WI_3_evaluation | FOODS_3_824 | FOODS_3 | FOODS | WI_3 | \ |
| **60034807** | FOODS_3_825_WI_3_evaluation | FOODS_3_825 | FOODS_3 | FOODS | WI_3 | \ |
| **60034808** | FOODS_3_826_WI_3_evaluation | FOODS_3_826 | FOODS_3 | FOODS | WI_3 | \ |
| **60034809** | FOODS_3_827_WI_3_evaluation | FOODS_3_827 | FOODS_3 | FOODS | WI_3 | \ |

60034810 rows × 63 columns

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 60034810 entries, 0 to 60034809
Data columns (total 63 columns):
 #   Column                         Dtype
---  ------                         -----
 0   id                             category
 1   item_id                        category
 2   dept_id                        category
 3   cat_id                         category
 4   store_id                       category
 5   state_id                       category
 6   d                              category
 7   sold                           int16
 8   date                           datetime64[ns]
 9   wm_yr_wk                       int16
 10  weekday                        category
 11  wday                           int8
 12  month                          int8
 13  year                           int16
 14  event_name_1                   category
 15  event_type_1                   category
 16  event_name_2                   category
 17  event_type_2                   category
 18  snap_CA                        int8
 19  snap_TX                        int8
 20  snap_WI                        int8
 21  sell_price                     float16
 22  revenue                        float16
 23  lag_7                          float64
 24  lag_21                         float64
 25  lag_28                         float64
 26  lag_35                         float64
 27  lag_42                         float64
 28  price-diff                     float16
 29  rolling_sold_mean_7            float16
 30  rolling_sold_mean_14           float16
 31  rolling_sold_mean_28           float16
 32  rolling_sold_mean_90           float16
 33  rolling_sold_max_7             float16
 34  rolling_sold_max_28            float16
 35  rolling_sold_std_7             float16
 36  rolling_sold_std_28            float16
 37  rolling_sold_std_90            float16
 38  rolling_revenue_mean_28        float16
 39  rolling_revenue_max_28         float16
 40  rolling_revenue_std_28         float16
 41  rolling_item_revenue_mean_28   float16
 42  rolling_item_sold_mean_7       float16
 43  rolling_item_sold_mean_28      float16
 44  rolling_dept_sold_mean_28      float16
 45  rolling_store_sold_mean_28     float16
 46  item_sold_avg                  float16
 47  store_sold_avg                 float16
 48  state_sold_avg                 float16
 49  store_item_sold_avg            float16
 50  cat_item_sold_avg             float16
 51  state_item_sold_avg           float16
 52  store_weekday_sold_avg        float16
 53  max_price                     float16
 54  min_price                     float16
 55  price_mean                    float16
 56  price_std                     float16
 57  price_norm                    float16
 58  daily_avg_sold                float16
 59  avg_sold                      float16
 60  selling_trend                 float16
 61  expanding_sold_mean           float16
 62  expanding_revenue_mean        float16
dtypes: category(12), datetime64[ns](1), float16(37), float64(5), int16(3), int
8(5)
memory usage: 11.2 GB
```

In [ ]:

## Handling categorical features :-

In [ ]:

```
# Also need to make categorical features category  needs to be removed
```

In [ ]:

```
# Changing datatype of days to numeric
df['d'] = df['d'].str[2:]
df['d'] = df['d'].astype(int)
```

In [ ]:

```
# We will apply ordinal encoding.
```

In [ ]:

```
# Replacing NAN values in cat_cols with No_event
cat_cols =[ "event_name_1", "event_type_1", "event_name_2", "event_type_2"]

for i in cat_cols:
  df[i] = df[i].cat.add_categories('No_event')
  df[i].fillna('No_event' , inplace = True)
```

In [ ]:

In [ ]:

```
df
```

Out[7]:

| | id | item_id | dept_id | cat_id | store_id | state_id | d | sold |
|---|---|---|---|---|---|---|---|---|
| **0** | HOBBIES_1_001_CA_1_evaluation | 1437 | 3 | 1 | 0 | 0 | 1 | 0 |
| **1** | HOBBIES_1_002_CA_1_evaluation | 1438 | 3 | 1 | 0 | 0 | 1 | 0 |
| **2** | HOBBIES_1_003_CA_1_evaluation | 1439 | 3 | 1 | 0 | 0 | 1 | 0 |
| **3** | HOBBIES_1_004_CA_1_evaluation | 1440 | 3 | 1 | 0 | 0 | 1 | 0 |
| **4** | HOBBIES_1_005_CA_1_evaluation | 1441 | 3 | 1 | 0 | 0 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **60034805** | FOODS_3_823_WI_3_evaluation | 1432 | 2 | 0 | 9 | 2 | 1969 | 0 |
| **60034806** | FOODS_3_824_WI_3_evaluation | 1433 | 2 | 0 | 9 | 2 | 1969 | 0 |
| **60034807** | FOODS_3_825_WI_3_evaluation | 1434 | 2 | 0 | 9 | 2 | 1969 | 0 |
| **60034808** | FOODS_3_826_WI_3_evaluation | 1435 | 2 | 0 | 9 | 2 | 1969 | 0 |
| **60034809** | FOODS_3_827_WI_3_evaluation | 1436 | 2 | 0 | 9 | 2 | 1969 | 0 |

60034810 rows × 67 columns

In [ ]:

In [ ]:

```
del sales , sell_prices , calendar
gc.collect()
```

Out[46]:

```
815
```

In [ ]:

## Date features

In [ ]:

In [ ]:

```
# Getting features related to date

df['date'] = pd.to_datetime(df['date'])
time_features = ['year', 'month', 'week', 'day', 'dayofweek', 'dayofyear']
dtype = np.int16
for time_feature in time_features:
    df[time_feature] = getattr(df['date'].dt, time_feature).astype(dtype)
```

In [ ]:

```
df['weekends'] = np.where((df['date'].dt.dayofweek) < 5, 0, 1)
```

In [ ]:

## Saving the file

We are saving the file and performing some the preprocessing later to avoid the crashing of ram.

In [ ]:

```
# Saving the data
from sklearn.externals import joblib
import pickle
filename = 'features_1.pkl'
joblib.dump(df, filename)
```

Out[49]:

```
['features_1.pkl']
```

In [ ]:

```
filename = 'features_1.pkl'

df = joblib.load(filename)
```

### Handling Categorical Features II

In [ ]:

```
# Ordinal coding and changing the column type to category

category_cols = ["item_id", "dept_id", "store_id", "cat_id", "state_id" , "event_


for i in category_cols:

  df[i] = OrdinalEncoder(dtype="int").fit_transform(df[[i]])

  df[i] = df[i].astype('category')
```

In [ ]:

### Handling nan values

In [ ]:
```
# Taking the data after 91st day to remove all the null values created by lags ar

# df.isnull().sum()
df = df[df['d'] > 91]
```

In [ ]:
```
df = downcast(df)
```

In [ ]:
```
```

In [ ]:
```
```

# Modelling

Trying out various models and techniques to get predictions for validation and test datasets.

In [ ]:
```
filename = 'features_1.pkl'
df = joblib.load(filename)
```

In [ ]:
```
df
```

Out[9]:
```
                                    id  item_id  dept_id  cat_id  store_id  state_id     d  sold
2774590  HOBBIES_1_001_CA_1_evaluation     1437        3       1         0         0    92     0
2774591  HOBBIES_1_002_CA_1_evaluation     1438        3       1         0         0    92     0
2774592  HOBBIES_1_003_CA_1_evaluation     1439        3       1         0         0    92     0
2774593  HOBBIES_1_004_CA_1_evaluation     1440        3       1         0         0    92     1
2774594  HOBBIES_1_005_CA_1_evaluation     1441        3       1         0         0    92     0
    ...                            ...      ...      ...     ...       ...       ...   ...   ...
60034805   FOODS_3_823_WI_3_evaluation     1432        2       0         9         2  1969     0
60034806   FOODS_3_824_WI_3_evaluation     1433        2       0         9         2  1969     0
60034807   FOODS_3_825_WI_3_evaluation     1434        2       0         9         2  1969     0
60034808   FOODS_3_826_WI_3_evaluation     1435        2       0         9         2  1969     0
60034809   FOODS_3_827_WI_3_evaluation     1436        2       0         9         2  1969     0

57260220 rows × 67 columns
```

In [ ]:

```

```

In [ ]:

```
# # Changing column type to category

# Encoding id
df["id_encoded"] = OrdinalEncoder(dtype="int").fit_transform(df[["id"]])
df["id_encoded"] = df["id_encoded"].astype('category')
```

In [ ]:

```
category_cols = ['wday' ,'month']
for i in category_cols:
  df[i] = df[i].astype('category')
```

Only choosing few columns for better performance

In [ ]:

```
df_final = df[['id_encoded','item_id', 'dept_id', 'cat_id', 'store_id', 'state_ic
               'year', 'month', 'week', 'day', 'dayofweek', 'dayofyear' , 'rollir
                'rolling_sold_std_7', 'rolling_sold_std_28','rolling_sold_std_90'
           'cat_item_sold_avg','weekends','d','sold' ]]
```

In [ ]:

```

```

In [ ]:

```
# items = df_final['item_id'].unique()
items = df['item_id'].unique()
```

In [ ]:

```
departments = df['dept_id'].unique()
```

In [ ]:

```
stores = df['store_id'].unique()
```

In [ ]:

```
states = df['state_id'].unique()
```

In [ ]:

```
categories = df['cat_id'].unique()
```

In [ ]:

```
categories
```

Out[18]:

```
[1, 2, 0]
Categories (3, int64): [1, 2, 0]
```

In [ ]:

```
len(stores)
```

Out[19]:

```
10
```

In [ ]:

```

# Getting valid and test sets
df_valid = df_final[(df_final['d'] >= 1914) & (df_final['d']<1942)]
df_test = df_final[df_final['d']>=1942]
```

In [ ]:

```
X_valid = df_valid.drop('sold',axis=1)
y_valid_pred = df_valid['sold']

X_test = df_test.drop('sold',axis=1)
y_test = df_test['sold']
```

In [ ]:

In [ ]:

```
# Converting all elements of y_valid_pred to 0 so that it could be used in final

y_valid_pred[X_valid.index] = 0
```

In [ ]:

In [ ]:

## LGBMRegressor

Type *Markdown* and LaTeX: $\alpha^2$

In [ ]:

```

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform

from sklearn.model_selection import StratifiedKFold, KFold, RepeatedKFold, GroupK


import lightgbm as lgb
```

In [ ]:

In [ ]:

In [ ]:

```
# To avoid ram crashing
df_final_train = df_final[df_final['d'] > 600 ]
```

In [ ]:
```
df_final_train
```

Out[23]:
```
                id  item_id  dept_id  cat_id  store_id  state_id     d  sold  wday  month  year  eve
30490000     14370     1437        3       1         0         0  1001     2     7     10  2013
30490001     14380     1438        3       1         0         0  1001     0     7     10  2013
30490002     14390     1439        3       1         0         0  1001     0     7     10  2013
30490003     14400     1440        3       1         0         0  1001     0     7     10  2013
30490004     14410     1441        3       1         0         0  1001     1     7     10  2013
     ...       ...      ...      ...     ...       ...       ...   ...   ...   ...    ...   ...
60034805     14329     1432        2       0         9         2  1969     0     2      6  2016
60034806     14339     1433        2       0         9         2  1969     0     2      6  2016
60034807     14349     1434        2       0         9         2  1969     0     2      6  2016
60034808     14359     1435        2       0         9         2  1969     0     2      6  2016
60034809     14369     1436        2       0         9         2  1969     0     2      6  2016

29544810 rows × 45 columns
```

In [ ]:
```
df_final_train = downcast(df_final_train)
```

In [ ]:
```
del df,df_final,df_valid , df_test

gc.collect()
```

Out[29]:
```
325
```

In [ ]:
```

```

In [ ]:

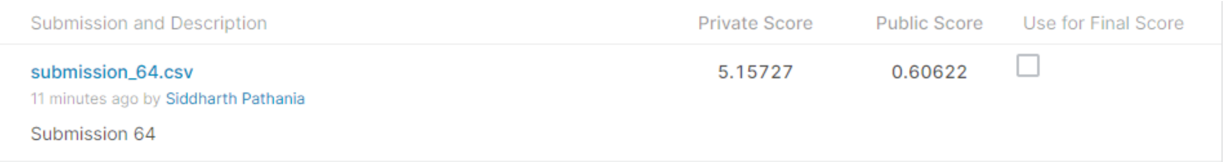
```
category_cols = ['item_id','dept_id','cat_id','store_id','state_id','event_name_1


df_train, df_valid = train_test_split(df_final_train, test_size=0.30, random_stat


train = lgb.Dataset(df_train.drop('sold' , axis =1 ), df_train['sold'] )

valid = lgb.Dataset(df_valid.drop('sold' , axis =1 ), df_valid['sold'] )

params = {
        'boosting_type': 'gbdt',
        'metric': 'rmse',
        'objective': 'poisson',
        'max_depth': 100, # max depth of decision trees
        'num_leaves': 100,  #  number of leaves
         'learning_rate' : 0.05 }

model = lgb.train( params  , train_set = train , early_stopping_rounds = 50,
                          valid_sets = valid, verbose_eval = 50, num_boost_round


lgb.plot_importance(model, importance_type = 'gain', precision = 0,
                            figsize = (6, 10),
                            title = 'feature importance')


y_valid_pred[X_valid.index] = model.predict(X_valid)

y_test[X_test.index] = model.predict(X_test)



gc.collect()
```

In [ ]:

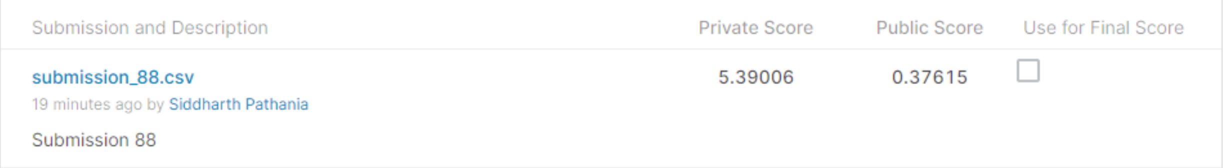
```
gc.collect()
```

Out[27]:

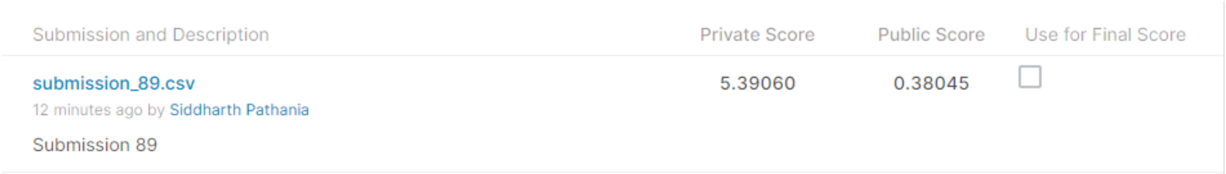
```
7639
```



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_64.csv<br>11 minutes ago by Siddharth Pathania<br>Submission 64 | 5.15727 | 0.60622 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_88.csv<br>19 minutes ago by Siddharth Pathania<br>Submission 88 | 5.39006 | 0.37615 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_89.csv<br>12 minutes ago by Siddharth Pathania<br>Submission 89 | 5.39060 | 0.38045 | □ |

In [ ]:

In [ ]:

## LGBM on different categories

In [ ]:

In [ ]:

In [ ]:
```
# X_valid
```

In [ ]:
```


# df_final_train = downcast(df_final)

# df_final_train = df_final[df_final['d'] > 900 ]
df_final_train = df_final

df_final_train = downcast(df_final_train)
# df_final_train = df_final_train[df_final_train['d'] > 500 ]
```

In [ ]:

In [ ]:
```
del df,df_final,df_valid , df_test

# del df_final

gc.collect()
```

Out[32]:
```
55
```

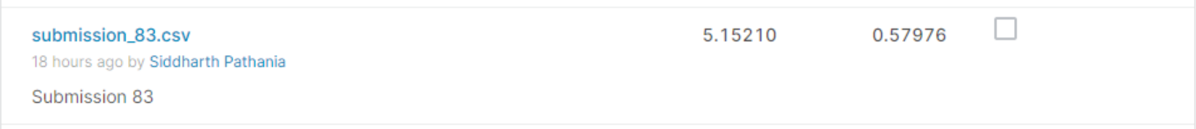
In [ ]:
```
# df_final_train.info()
```

In [ ]:
```
# df_final_train = downcast(df_final_train)

```

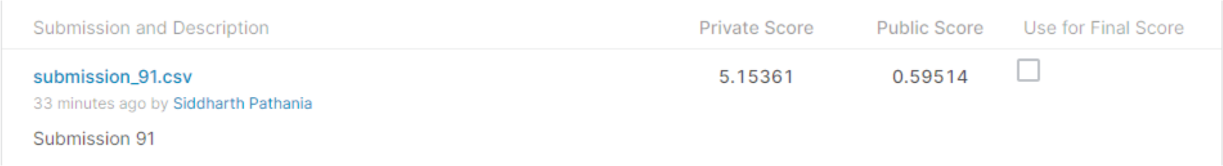
In [ ]:

```
# category_cols = ['item_id','dept_id','cat_id','store_id','state_id','event_name

category_cols = ['item_id','dept_id','cat_id','store_id','state_id','event_name_1


for i in categories :

    print("category :-" , i)
    df_cat = df_final_train[df_final_train['cat_id'] == i]

    test = X_test[X_test['cat_id'] == i]


    df_train = df_cat[df_cat['d']<1914]
    df_valid = df_cat[(df_cat['d']>=1914) & (df_cat['d']<1942)]


    train = lgb.Dataset(df_train.drop('sold' , axis =1 ), df_train['sold'], categ

    valid = lgb.Dataset(df_valid.drop('sold' , axis =1 ), df_valid['sold'], categ

    params = {
        'boosting_type': 'gbdt',
        'metric': 'rmse',
        'objective': 'poisson',
        'bagging_fraction': 0.6, # bootstrap sampling
        'bagging_freq' : 1,
        'colsample_bytree': 0.6, # feature sampling
        'max_depth': 200, # max depth of decision trees
        'num_leaves': 100,  #  number of leaves
         'learning_rate' : 0.05
          }

    model = lgb.train( params  , train_set = train , early_stopping_rounds = 50,
                          valid_sets = valid, verbose_eval = 100, num_boost_round


    lgb.plot_importance(model, importance_type = 'gain', precision = 0,
                            figsize = (6, 10),
                            title = 'feature importance')


    # model.fit(X_train, y_train, eval_set=[(X_train,y_train),(X_valid,y_valid)]

    y_valid_pred[df_valid.index] = model.predict(df_valid.drop('sold' , axis =1 )

    y_test[test.index] = model.predict(test)



    del model ,df_train, df_cat , train,df_valid,valid , test
    gc.collect()
  # y_train_pred = model.predict(X_train)

  # y_valid_pred = model.predict(X_valid)
```

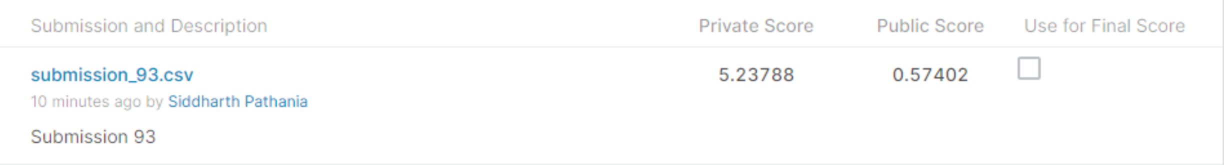
```
category :- 1
Training until validation scores don't improve for 50 rounds.
[100]	valid_0's rmse: 1.70011
[200]	valid_0's rmse: 1.64265
[300]	valid_0's rmse: 1.63331
[400]	valid_0's rmse: 1.63085
[500]	valid_0's rmse: 1.63001
[600]	valid_0's rmse: 1.62943
[700]	valid_0's rmse: 1.62926
[800]	valid_0's rmse: 1.62918
[900]	valid_0's rmse: 1.62904
[1000]	valid_0's rmse: 1.62892
[1100]	valid_0's rmse: 1.6288
[1200]	valid_0's rmse: 1.62866
Early stopping, best iteration is:
[1162]	valid_0's rmse: 1.62863
category :- 2
Training until validation scores don't improve for 50 rounds.
[100]	valid_0's rmse: 1.53022
```



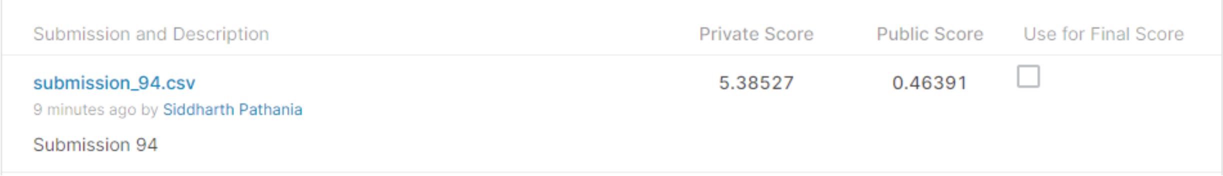
| | | | |
|---|---|---|---|
| submission_83.csv<br>18 hours ago by Siddharth Pathania<br>Submission 83 | 5.15210 | 0.57976 | □ |



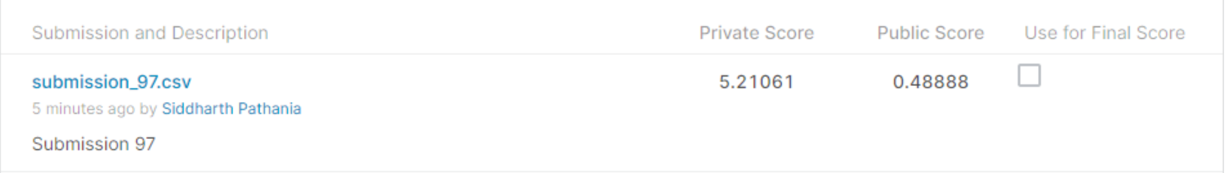
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_91.csv<br>33 minutes ago by Siddharth Pathania<br>Submission 91 | 5.15361 | 0.59514 | □ |



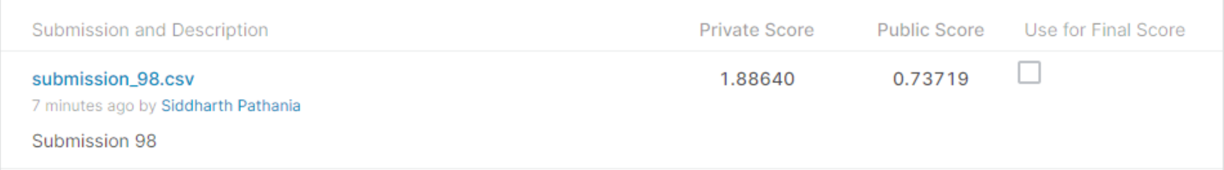
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_93.csv<br>10 minutes ago by Siddharth Pathania<br>Submission 93 | 5.23788 | 0.57402 | □ |



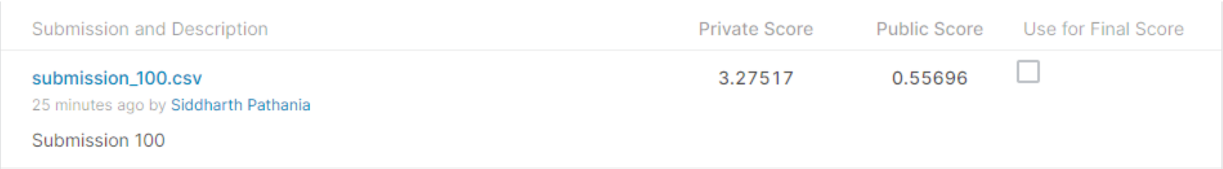
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_94.csv<br>9 minutes ago by Siddharth Pathania<br>Submission 94 | 5.38527 | 0.46391 | □ |



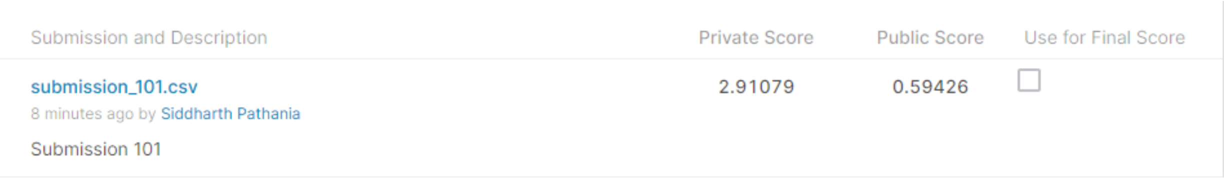
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_97.csv<br>5 minutes ago by Siddharth Pathania<br>Submission 97 | 5.21061 | 0.48888 | □ |



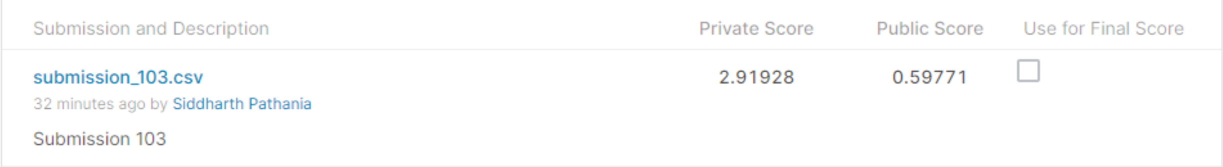
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_98.csv<br>7 minutes ago by Siddharth Pathania<br>Submission 98 | 1.88640 | 0.73719 | □ |



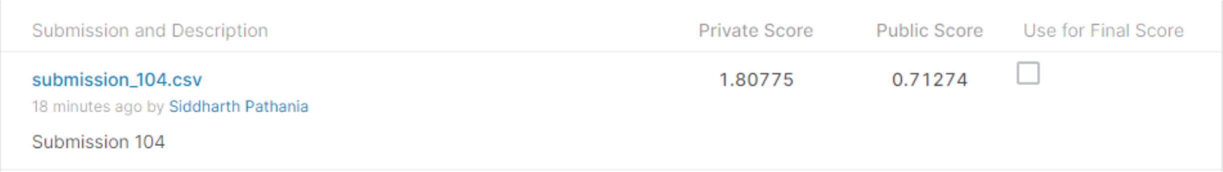
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_100.csv<br>25 minutes ago by Siddharth Pathania<br>Submission 100 | 3.27517 | 0.55696 | □ |



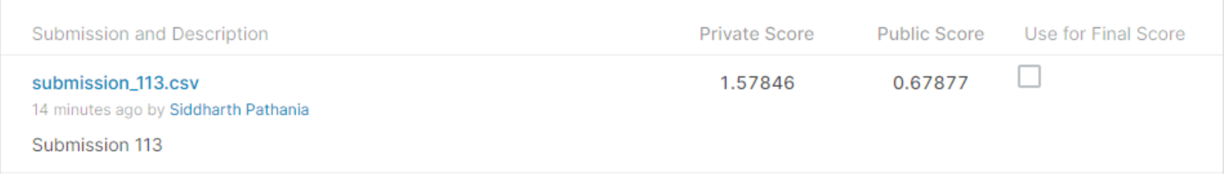
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_101.csv<br>8 minutes ago by Siddharth Pathania<br>Submission 101 | 2.91079 | 0.59426 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_103.csv<br>32 minutes ago by Siddharth Pathania<br>Submission 103 | 2.91928 | 0.59771 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_104.csv<br>18 minutes ago by Siddharth Pathania<br>Submission 104 | 1.80775 | 0.71274 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_113.csv<br>14 minutes ago by Siddharth Pathania<br>Submission 113 | 1.57846 | 0.67877 | □ |

```
In [ ]:
```

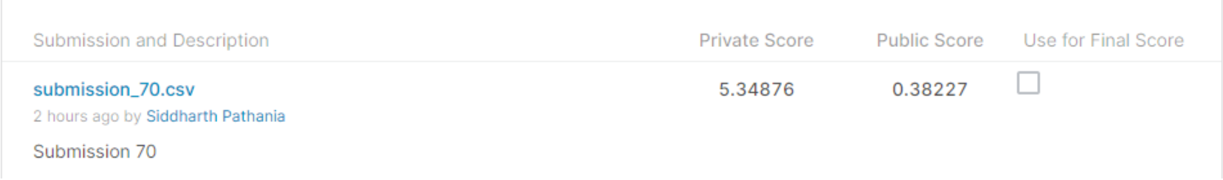
## LGBM on different stores

In [ ]:

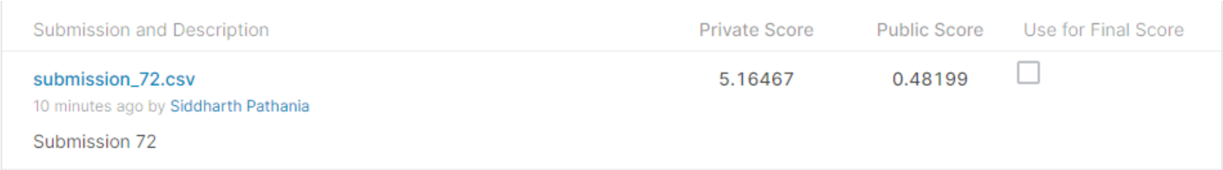
```
category_cols = ['item_id','dept_id','cat_id','store_id','state_id','event_name_1


for i in stores :

    print("store :-" , i)
    df_store = df_final_train[df_final_train['store_id'] == i]

    test = X_test[X_test['store_id'] == i]


    df_train = df_store[df_store['d']<1914]
    df_valid = df_store[(df_store['d']>=1914) & (df_store['d']<1942)]


    train = lgb.Dataset(df_train.drop('sold' , axis =1 ), df_train['sold'], categ

    valid = lgb.Dataset(df_valid.drop('sold' , axis =1 ), df_valid['sold'], categ

    params = {
        'boosting_type': 'gbdt',
        'metric': 'rmse',
        'objective': 'poisson',
        'bagging_fraction': 0.6, # bootstrap sampling
        'bagging_freq' : 1,
        'colsample_bytree': 0.6, # feature sampling
        'max_depth': 200, # max depth of decision trees
        'num_leaves': 100,  #  number of leaves
         'learning_rate' : 0.05 }

    model = lgb.train( params  , train_set = train , early_stopping_rounds = 50,
                          valid_sets = valid, verbose_eval = 100, num_boost_round


    lgb.plot_importance(model, importance_type = 'gain', precision = 0,
                            figsize = (6, 10),
                            title = f'feature importance for store - {i}')


    y_valid_pred[df_valid.index] = model.predict(df_valid.drop('sold' , axis =1 )

    y_test[test.index] = model.predict(test)

    model.save_model(f'model{i}.lgb')

    del model ,df_train, df_store , train,df_valid,valid , test
    gc.collect()
```

```
store :- 0
Training until validation scores don't improve for 50 rounds.
[100]   valid_0's rmse: 2.18808
[200]   valid_0's rmse: 2.15993
```

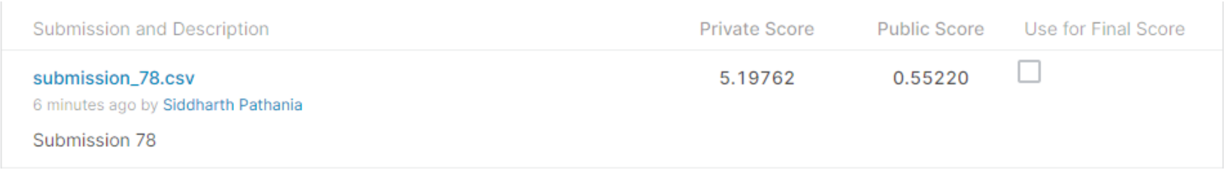
```
[300]   valid_0's rmse: 2.15312
[400]   valid_0's rmse: 2.14899
[500]   valid_0's rmse: 2.14696
Early stopping, best iteration is:
[505]   valid_0's rmse: 2.14679
store :- 1
Training until validation scores don't improve for 50 rounds.
[100]   valid_0's rmse: 2.03823
[200]   valid_0's rmse: 1.9991
Early stopping, best iteration is:
[246]   valid_0's rmse: 1.99791
store :- 2
Training until validation scores don't improve for 50 rounds.
[100]   valid_0's rmse: 2.65634
[200]   valid_0's rmse: 2.64983
```



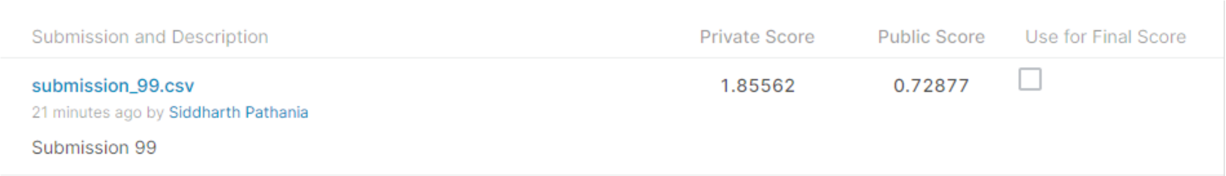
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_70.csv<br>2 hours ago by Siddharth Pathania<br>Submission 70 | 5.34876 | 0.38227 | □ |



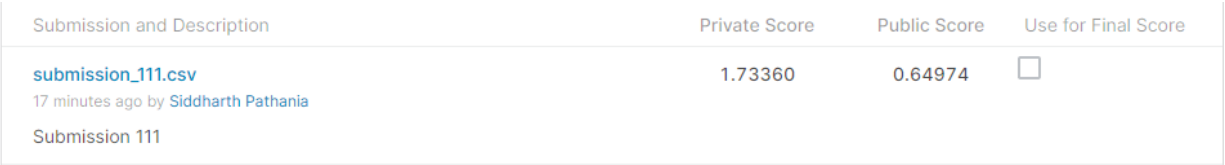
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_72.csv<br>10 minutes ago by Siddharth Pathania<br>Submission 72 | 5.16467 | 0.48199 | □ |



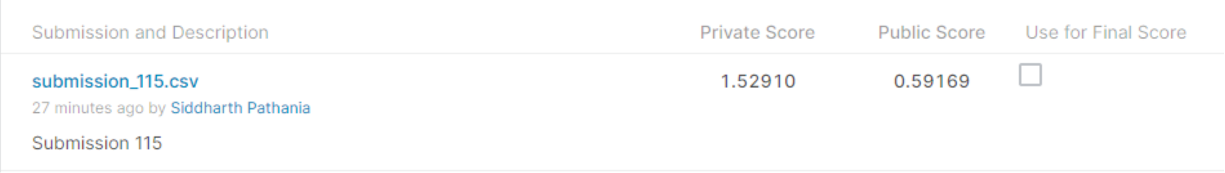
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_78.csv<br>6 minutes ago by Siddharth Pathania<br>Submission 78 | 5.19762 | 0.55220 | □ |



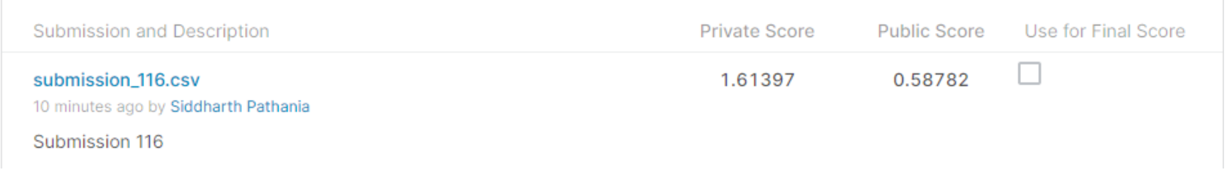
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_99.csv<br>21 minutes ago by Siddharth Pathania<br>Submission 99 | 1.85562 | 0.72877 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_111.csv<br>17 minutes ago by Siddharth Pathania<br>Submission 111 | 1.73360 | 0.64974 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_115.csv<br>27 minutes ago by Siddharth Pathania<br>Submission 115 | 1.52910 | 0.59169 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_116.csv<br>10 minutes ago by Siddharth Pathania<br>Submission 116 | 1.61397 | 0.58782 | □ |

```
In [ ]: gc.collect()

Out[28]: 94335
```

```
In [ ]: # df_final_train
```

```
In [ ]:
```

## LGBM Regressor with CV fold

```
In [ ]:
```

```
In [ ]: 
        df_train = df_final[ (df_final['d'] > 600) & (df_final['d']<1942)]

        df_train = downcast(df_train)

        x_train , y_train = df_train.drop('sold',axis=1) , df_train['sold']
```

```
In [ ]: # To clear the ram

        del df,df_final,df_valid , df_test ,df_train

        gc.collect()

Out[28]: 97
```

```
In [ ]:
```

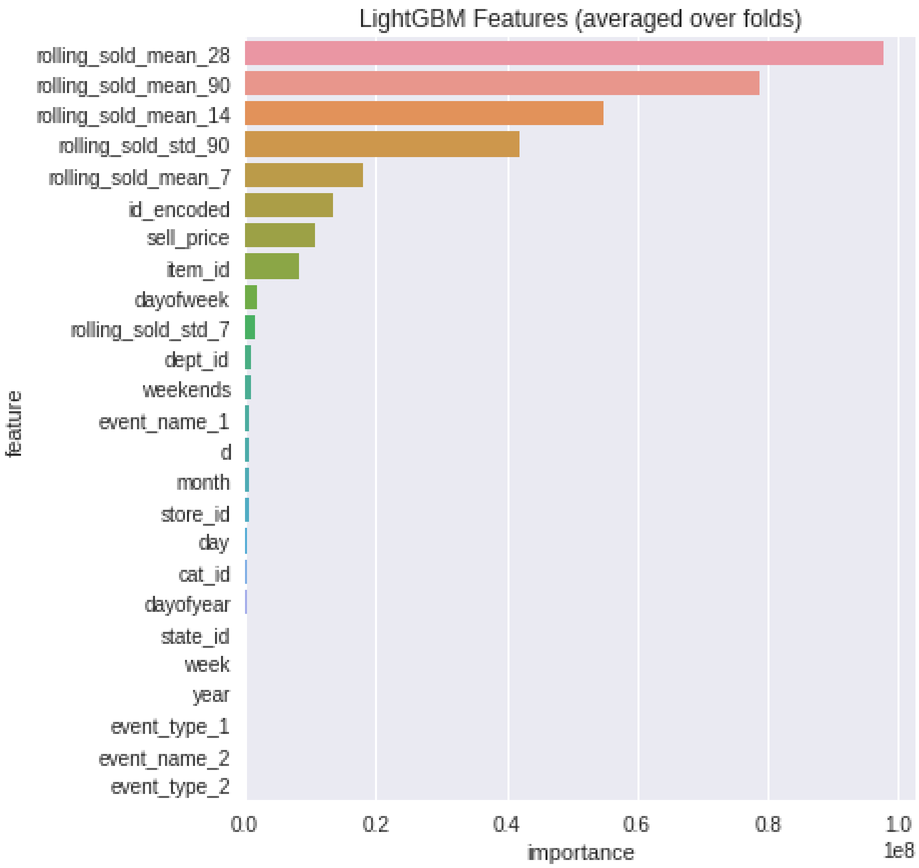
In [ ]:

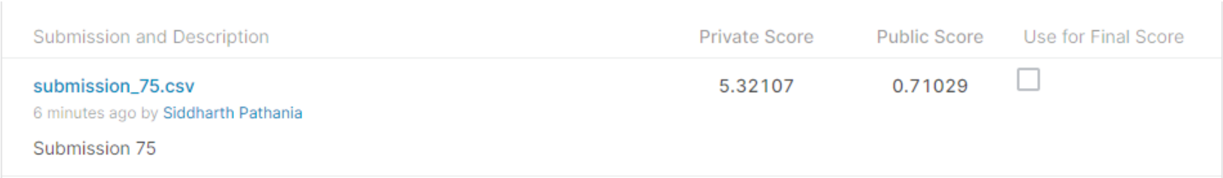
```
## Ref. link :- https://www.kaggle.com/ratan123/m5-forecasting-lightgbm-with-time

## Ref. link :- https://www.kaggle.com/rikdifos/timeseriessplit-cv-poisson


category_cols = ['id_encoded','item_id','dept_id','cat_id','store_id','state_id',



# To avoid crashing of memory keep max depth low and number of leaves less
params = {
        'boosting_type': 'gbdt',
        'metric': 'rmse',
        'objective': 'poisson',
        'max_depth': 5, # max depth of decision trees
        'num_leaves': 64,  #  number of leaves

        'bagging_fraction': 0.6, # bootstrap sampling
        'bagging_freq' : 1,
        'colsample_bytree': 0.6, # feature sampling
        'learning_rate' : 0.05

        }

# n_fold = 3
n_fold = 5
# n_fold = 6
# n_fold = 8

folds = TimeSeriesSplit(n_splits=n_fold)
splits = folds.split(x_train , y_train)
feature_importance_df = pd.DataFrame()


for fold_n, (train_index, valid_index) in enumerate(splits):

    print('Fold:',fold_n+1)

    # training and validation sets for model training

    train_set = lgb.Dataset(x_train.iloc[train_index] , y_train.iloc[train_index]

    val_set = lgb.Dataset(x_train.iloc[valid_index] , y_train.iloc[valid_index],

    model = lgb.train(params, train_set, valid_sets = [val_set] , early_stopping_


    lgb.plot_importance(model, importance_type = 'gain', precision = 0,
                            height = 0.5, figsize = (6, 10),
                            title = f'fold {fold_n+1} feature importance', ignore


    fold_importance_df = pd.DataFrame()
    fold_importance_df['feature'] = x_train.columns
    fold_importance_df['importance'] = model.feature_importance(importance_type =
    fold_importance_df['fold'] = fold_n + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df]


    y_valid_pred[X_valid.index] = model.predict(X_valid)

    y_test[X_test.index] += model.predict(X_test) /n_fold

    # Saving the models
    model.save_model(f'model{fold_n+1}.lgb')

    del  train_set,val_set

    gc.collect()

model.save_model('model.lgb')
```

```
Fold: 1
Training until validation scores don't improve for 50 rounds.
[50]	valid_0's rmse: 3.07784
[100]	valid_0's rmse: 2.96086
[150]	valid_0's rmse: 2.94903
[200]	valid_0's rmse: 2.94763
[250]	valid_0's rmse: 2.94296
[300]	valid_0's rmse: 2.94106
[350]	valid_0's rmse: 2.93843
[400]	valid_0's rmse: 2.93686
[450]	valid_0's rmse: 2.93559
[500]	valid_0's rmse: 2.93194
[550]	valid_0's rmse: 2.93005
[600]	valid_0's rmse: 2.92874
[650]	valid_0's rmse: 2.92833
Early stopping, best iteration is:
[620]	valid_0's rmse: 2.92793
Fold: 2
Training until validation scores don't improve for 50 rounds.
[50]	valid_0's rmse: 2.76958
```

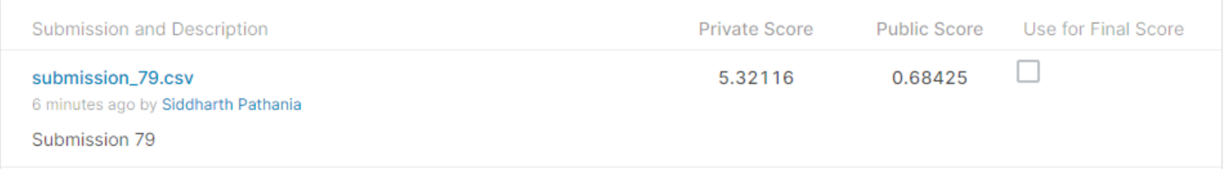
```
In [ ]: feature_importance = (feature_importance_df[['feature', 'importance']]
                .groupby('feature')
                .mean()
                .sort_values(by='importance', ascending=False))
        feature_importance['feature'] = feature_importance.index
        plt.figure(figsize=(6,7))
        sns.barplot(x='importance', y='feature', data=feature_importance[:40])
        plt.title('LightGBM Features (averaged over folds)')
```

```
Out[30]: Text(0.5, 1.0, 'LightGBM Features (averaged over folds)')
```

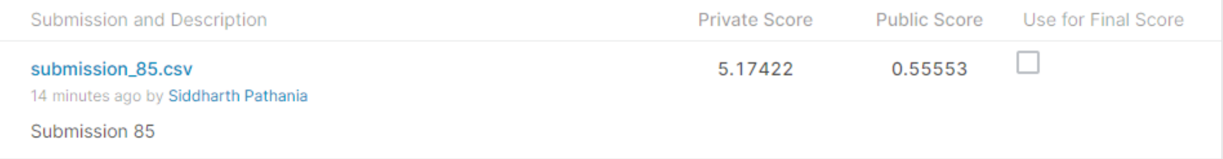




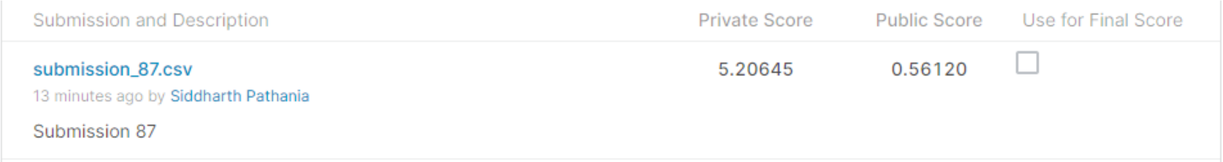
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_75.csv<br>6 minutes ago by Siddharth Pathania<br>Submission 75 | 5.32107 | 0.71029 | □ |



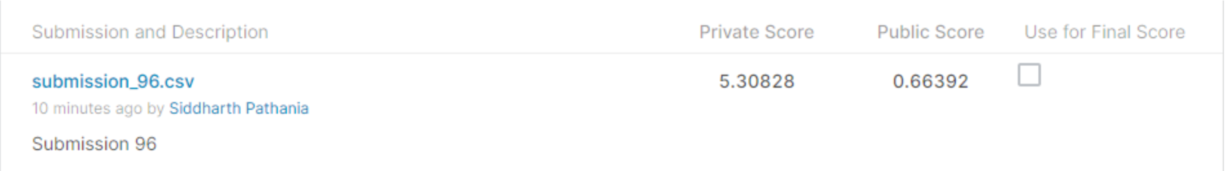
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_79.csv<br>6 minutes ago by Siddharth Pathania<br>Submission 79 | 5.32116 | 0.68425 | □ |



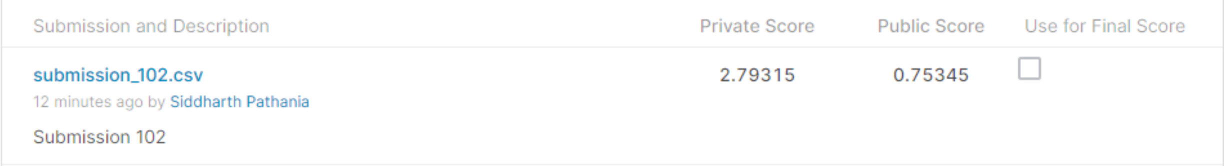
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_85.csv<br>14 minutes ago by Siddharth Pathania<br>Submission 85 | 5.17422 | 0.55553 | □ |



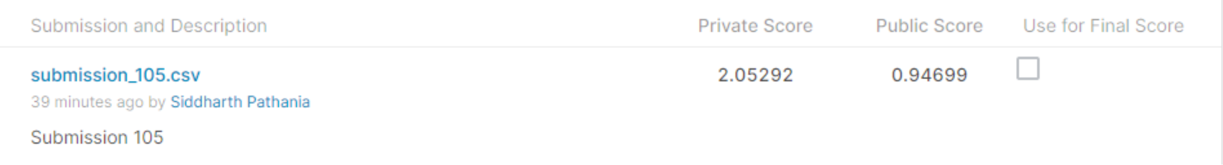
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_87.csv<br>13 minutes ago by Siddharth Pathania<br>Submission 87 | 5.20645 | 0.56120 | □ |



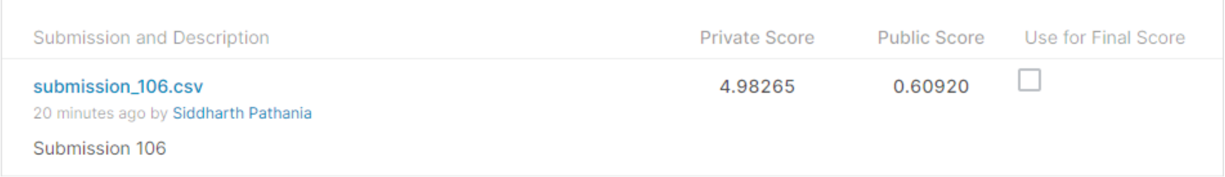
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_96.csv<br>10 minutes ago by Siddharth Pathania<br>Submission 96 | 5.30828 | 0.66392 | □ |



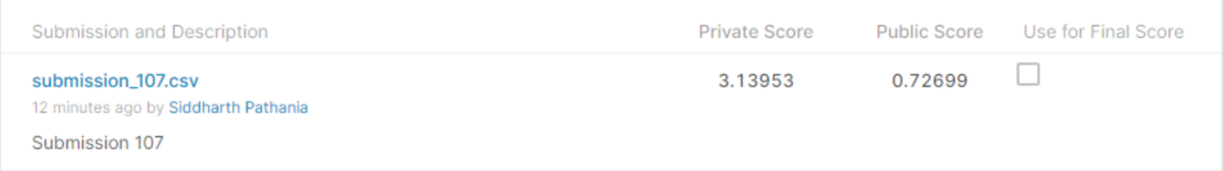
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_102.csv<br>12 minutes ago by Siddharth Pathania<br>Submission 102 | 2.79315 | 0.75345 | □ |



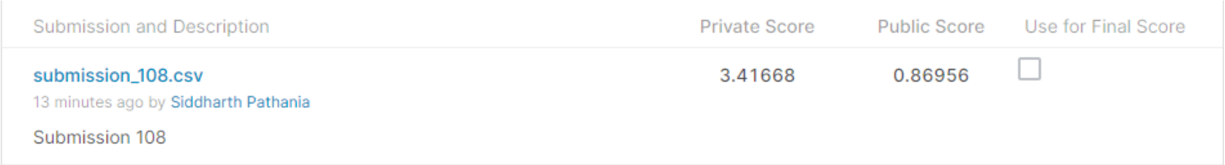
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_105.csv<br>39 minutes ago by Siddharth Pathania<br>Submission 105 | 2.05292 | 0.94699 | □ |



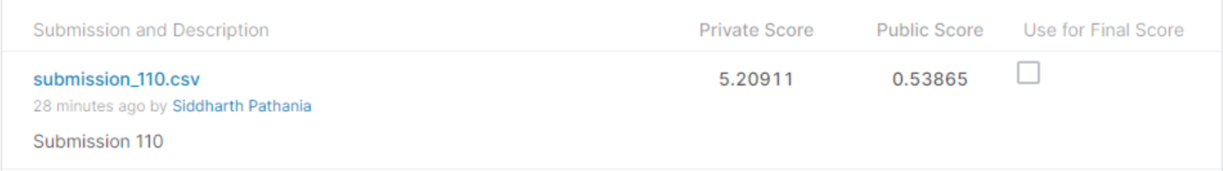
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_106.csv<br>20 minutes ago by Siddharth Pathania<br>Submission 106 | 4.98265 | 0.60920 | □ |



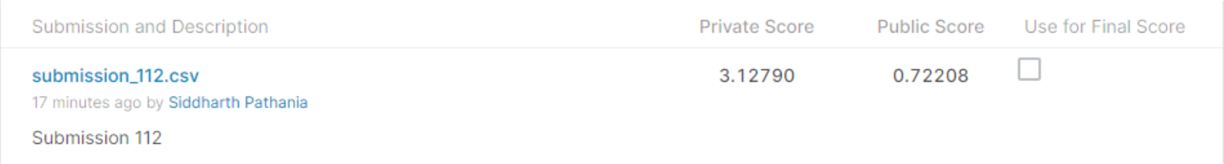
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_107.csv<br>12 minutes ago by Siddharth Pathania<br>Submission 107 | 3.13953 | 0.72699 | □ |



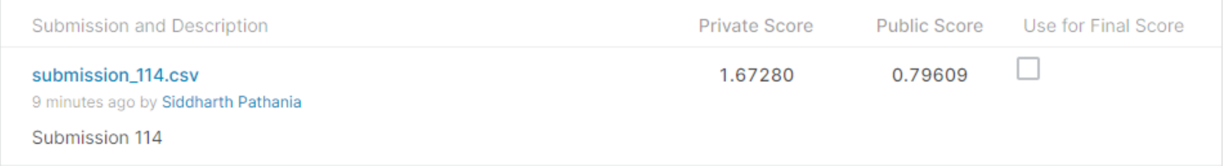
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_108.csv<br>13 minutes ago by Siddharth Pathania<br>Submission 108 | 3.41668 | 0.86956 | □ |



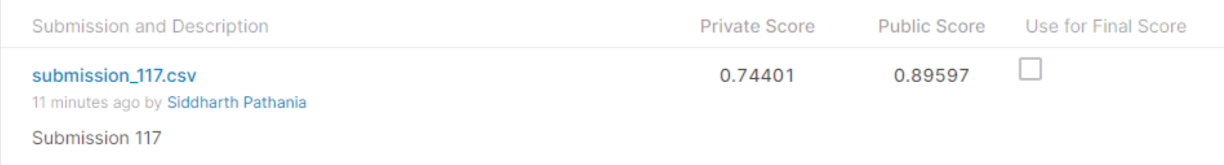
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_110.csv<br>28 minutes ago by Siddharth Pathania<br>Submission 110 | 5.20911 | 0.53865 | □ |



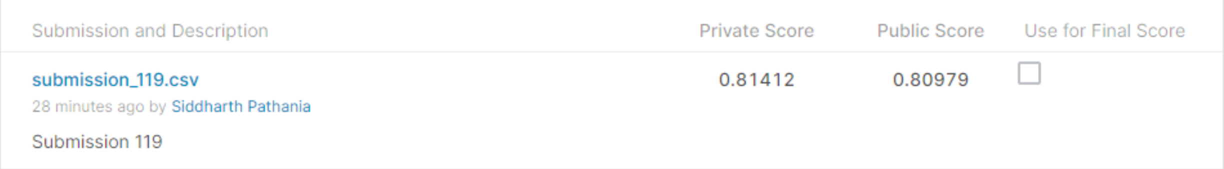
| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_112.csv<br>17 minutes ago by Siddharth Pathania<br>Submission 112 | 3.12790 | 0.72208 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_114.csv<br>9 minutes ago by Siddharth Pathania<br>Submission 114 | 1.67280 | 0.79609 | □ |

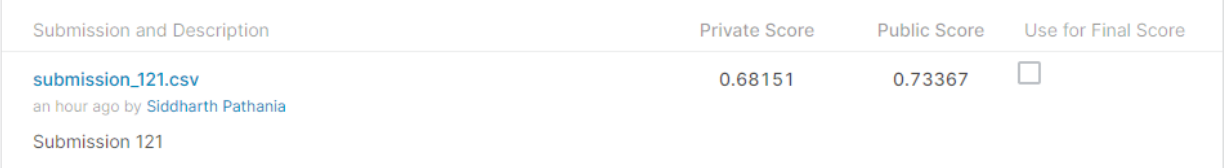


| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_117.csv<br>11 minutes ago by Siddharth Pathania<br>Submission 117 | 0.74401 | 0.89597 | □ |



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_119.csv<br>28 minutes ago by Siddharth Pathania<br>Submission 119 | 0.81412 | 0.80979 | □ |

### *Best Result*



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_121.csv<br>an hour ago by Siddharth Pathania<br>Submission 121 | 0.68151 | 0.73367 | □ |

In [ ]:

## XGBoost Regressor

In [ ]:

In [ ]:

```
import xgboost as xgb
from xgboost import XGBRegressor
```

In [ ]:

In [ ]:

```
df_final_train = downcast(df_final)
```

In [ ]:

```
category_cols = [ "item_id", "dept_id", "store_id", "cat_id", "state_id" , "event


for i in category_cols:

  # Training data
  df_final_train[i] = pd.to_numeric(df_final_train[i])

  # Valid for submission
  X_valid[i] = pd.to_numeric(X_valid[i])

  # Test data for submission
  X_test[i] = pd.to_numeric(X_test[i])
```

In [ ]:

```
# df_final_train
df_final_train = downcast(df_final_train)
```

In [ ]:

```
df_final_train = df_final_train[ df_final_train['d']>= 1200 ]

df_final_train = downcast(df_final_train)
```

In [ ]:

```
del df,df_final,df_valid , df_test

gc.collect()
```

Out[23]:

```
11
```

In [ ]:

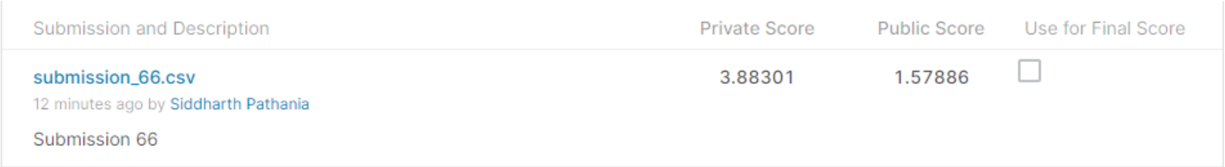
```
for i in stores :

    print("store :-" , i)
    df_cat = df_final_train[df_final_train['store_id'] == i]

    test = X_test[X_test['store_id'] == i]


    df_train = df_cat[  df_cat['d']<1914 ] # to avoid ram crashing
    df_valid = df_cat[(df_cat['d']>=1914) & (df_cat['d']<1942)]


    train = xgb.DMatrix(df_train.drop('sold' , axis =1 ), df_train['sold'])

    valid = xgb.DMatrix(df_valid.drop('sold' , axis =1 ), df_valid['sold'])

    params = {
        'boosting_type': 'gbdt',
        'metric': 'rmse',
        'obj': 'poisson',
        'max_depth': 5, # max depth of decision trees
        'num_leaves': 32,   #  number of leaves
         'learning_rate' : 0.02   }


    watchlist = [(valid, 'test'), (train, 'train')]


    model = xgb.train( params , train , num_boost_round = 50 , early_stopping_rou


    xgb.plot_importance(model, title = f'feature importance for store :- {i}'   ,

    y_valid_pred[df_valid.index] = model.predict(xgb.DMatrix(df_valid.drop('sold'

    y_test[test.index] = model.predict(xgb.DMatrix(test))



    del model ,df_train, df_cat , train,df_valid,valid , test
    gc.collect()
```

```
store :- 0
[0]	test-rmse:3.70154	train-rmse:3.80611
Multiple eval metrics have been passed: 'train-rmse' will be used for early s
topping.

Will train until train-rmse hasn't improved in 5 rounds.
[10]	test-rmse:3.25889	train-rmse:3.35842
```

```
[20]	test-rmse:2.92061	train-rmse:3.01386
[30]	test-rmse:2.66772	train-rmse:2.75457
[40]	test-rmse:2.48068	train-rmse:2.56021
[49]	test-rmse:2.35663	train-rmse:2.42815
store :- 1
[0]	test-rmse:3.28405	train-rmse:2.76585
Multiple eval metrics have been passed: 'train-rmse' will be used for early s
topping.

Will train until train-rmse hasn't improved in 5 rounds.
[10]	test-rmse:2.91283	train-rmse:2.47191
[20]	test-rmse:2.63821	train-rmse:2.24071
```

In [ ]:
```
gc.collect()
```

Out[25]:
```
52864
```



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_66.csv<br>12 minutes ago by Siddharth Pathania<br>Submission 66 | 3.88301 | 1.57886 | □ |

In [ ]:

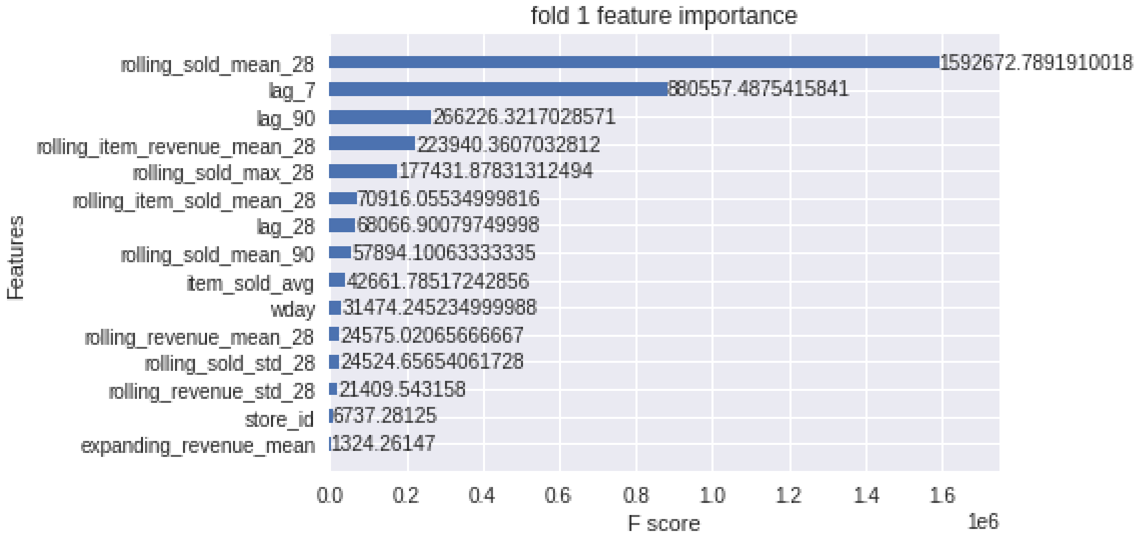
In [ ]:

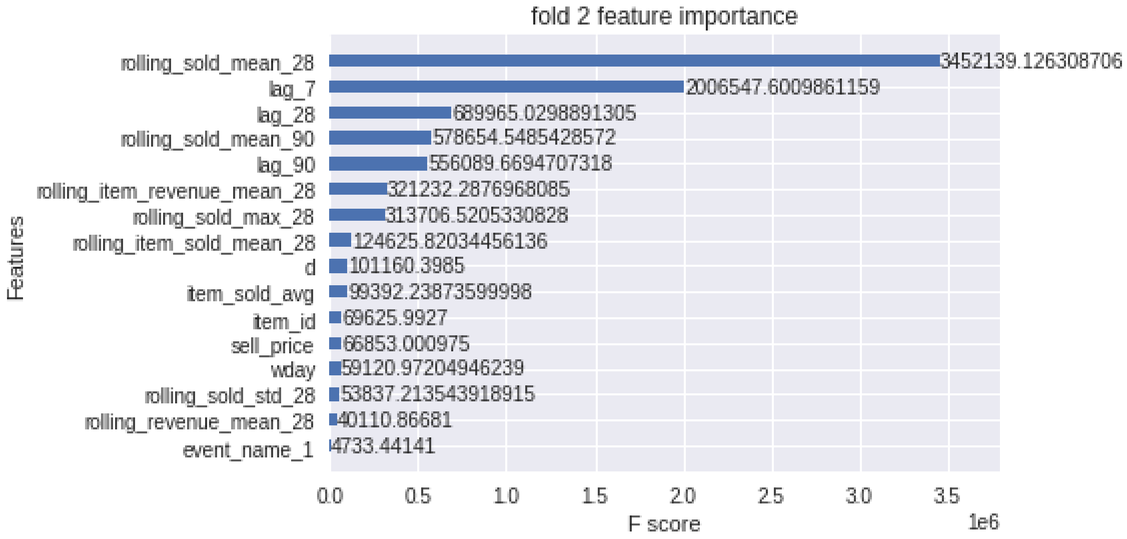
## XGBoost Regressor with CV fold

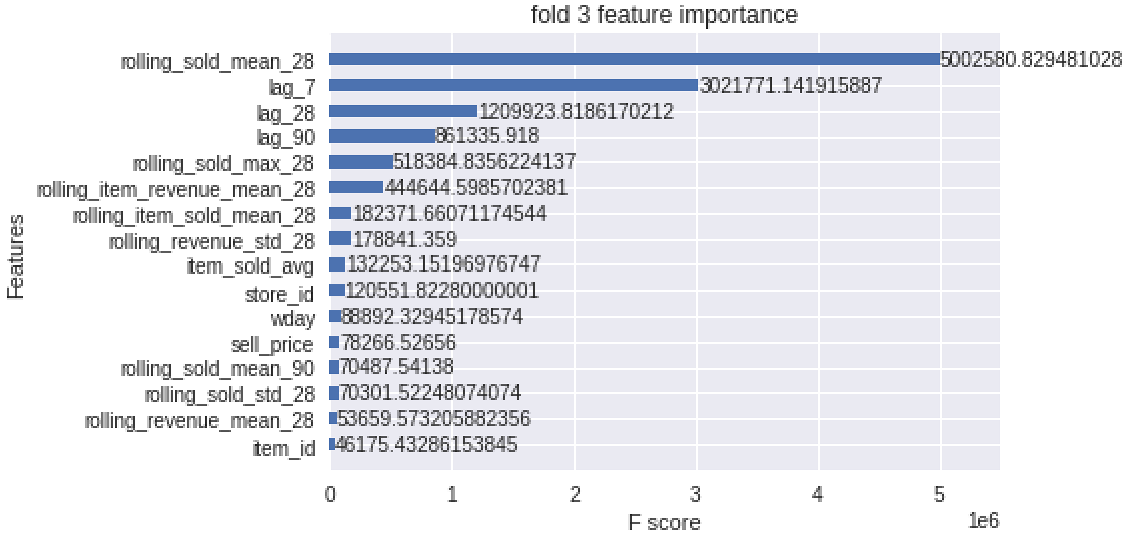
```
In [ ]: ## Ref. link :- https://www.kaggle.com/ratan123/m5-forecasting-lightgbm-with-time


        df_final_train = df_final_train[ df_final_train['d']>= 1400 ]


        # To avoid crashing of memory keep max depth low and number of leaves less
        params = {
                'boosting_type': 'gbdt',
                'metric': 'rmse',
                'obj': 'poisson',
                'max_depth': 5, # max depth of decision trees
                'num_leaves': 64,  #  number of leaves
                 'learning_rate' : 0.02

                }

        n_fold = 3
        # n_fold = 5
        # n_fold = 8

        x_train , y_train = df_final_train.drop('sold' , axis = 1)  , df_final_train['sol

        folds = TimeSeriesSplit(n_splits=n_fold)
        splits = folds.split(x_train , y_train)
        feature_importance_df = pd.DataFrame()


        for fold_n, (train_index, valid_index) in enumerate(splits):

            print('Fold:',fold_n+1)

            # training and validation sets for model training

            train = xgb.DMatrix(x_train.iloc[train_index] , y_train.iloc[train_index] )

            valid = xgb.DMatrix(x_train.iloc[valid_index] , y_train.iloc[valid_index] )


            watchlist = [(valid, 'test'), (train, 'train')]


            model = xgb.train( params , train , num_boost_round = 50 , early_stopping_rou


            xgb.plot_importance(model, importance_type = 'gain',height = 0.5,
                                    title = f'fold {fold_n+1} feature importance'   )


            y_valid_pred[X_valid.index] += model.predict(xgb.DMatrix(X_valid)) / n_fold

            y_test[X_test.index] += model.predict(xgb.DMatrix(X_test))  / n_fold

            # save model to file
            pickle.dump(model, open(f'model{fold_n+1}.pkl' , "wb"))
```

```
    del  train ,valid

    gc.collect()

pickle.dump( model , open('model.pkl' , "wb"))
```

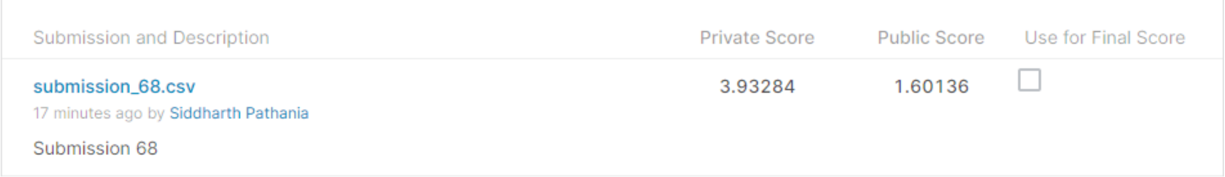
```
Fold: 1
[12:54:55] WARNING: /workspace/src/learner.cc:686: Tree method is automatically
selected to be 'approx' for faster speed. To use old behavior (exact greedy alg
orithm on single machine), set tree_method to 'exact'.
[0]	test-rmse:3.70692	train-rmse:3.38599
Multiple eval metrics have been passed: 'train-rmse' will be used for early sto
pping.

Will train until train-rmse hasn't improved in 5 rounds.
[10]	test-rmse:3.29457	train-rmse:3.02138
[20]	test-rmse:2.98271	train-rmse:2.74455
[30]	test-rmse:2.74994	train-rmse:2.53963
[40]	test-rmse:2.5766	train-rmse:2.3875
[49]	test-rmse:2.46109	train-rmse:2.28629
Fold: 2
[13:09:35] WARNING: /workspace/src/learner.cc:686: Tree method is automatically
selected to be 'approx' for faster speed. To use old behavior (exact greedy alg
orithm on single machine), set tree_method to 'exact'.
[0]	test-rmse:3.50797	train-rmse:3.54244
Multiple eval metrics have been passed: 'train-rmse' will be used for early sto
pping.

Will train until train-rmse hasn't improved in 5 rounds.
[10]	test-rmse:3.10681	train-rmse:3.14599
[20]	test-rmse:2.80101	train-rmse:2.84626
[30]	test-rmse:2.57726	train-rmse:2.6232
[40]	test-rmse:2.41418	train-rmse:2.45922
[49]	test-rmse:2.30705	train-rmse:2.34979
Fold: 3
[13:38:27] WARNING: /workspace/src/learner.cc:686: Tree method is automatically
selected to be 'approx' for faster speed. To use old behavior (exact greedy alg
orithm on single machine), set tree_method to 'exact'.
[0]	test-rmse:3.32356	train-rmse:3.53102
Multiple eval metrics have been passed: 'train-rmse' will be used for early sto
pping.

Will train until train-rmse hasn't improved in 5 rounds.
[10]	test-rmse:2.94007	train-rmse:3.13151
[20]	test-rmse:2.64691	train-rmse:2.83247
[30]	test-rmse:2.42931	train-rmse:2.61242
[40]	test-rmse:2.26786	train-rmse:2.45076
[49]	test-rmse:2.16016	train-rmse:2.34294
```


fold 1 feature importance
rolling_sold_mean_28
lag_7
lag_90
rolling_item_revenue_mean_28
rolling_sold_max_28
rolling_item_sold_mean_28
lag_28
rolling_sold_mean_90
item_sold_avg
wday
rolling_revenue_mean_28
rolling_sold_std_28
rolling_revenue_std_28
store_id
expanding_revenue_mean
1592672.7891910018
880557.4875415841
266226.3217028571
223940.3607032812
177431.87831312494
70916.05534999816
68066.90079749998
57894.10063333335
42661.78517242856
31474.245234999988
24575.02065666667
24524.65654061728
21409.543158
6737.28125
1324.26147
Features
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
F score
1e6


fold 2 feature importance
rolling_sold_mean_28
lag_7
lag_28
rolling_sold_mean_90
lag_90
rolling_item_revenue_mean_28
rolling_sold_max_28
rolling_item_sold_mean_28
d
item_sold_avg
item_id
sell_price
wday
rolling_sold_std_28
rolling_revenue_mean_28
event_name_1
3452139.126308706
2006547.6009861159
689965.0298891305
578654.5485428572
556089.6694707318
321232.2876968085
313706.5205330828
124625.82034456136
101160.3985
99392.23873599998
69625.9927
66853.000975
59120.97204946239
53837.213543918915
40110.86681
4733.44141
Features
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
F score
1e6


fold 3 feature importance
rolling_sold_mean_28
lag_7
lag_28
lag_90
rolling_sold_max_28
rolling_item_revenue_mean_28
rolling_item_sold_mean_28
rolling_revenue_std_28
item_sold_avg
store_id
wday
sell_price
rolling_sold_mean_90
rolling_sold_std_28
rolling_revenue_mean_28
item_id
5002580.829481028
3021771.141915887
1209923.8186170212
861335.918
518384.8356224137
444644.5985702381
182371.66071174544
178841.359
132253.15196976747
120551.82280000001
88892.32945178574
78266.52656
70487.54138
70301.52248074074
53659.573205882356
46175.43286153845
Features
0
1
2
3
4
5
F score
1e6

```
In [ ]:
```



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission_68.csv<br>17 minutes ago by Siddharth Pathania<br>Submission 68 | 3.93284 | 1.60136 | □ |

In [ ]:

# Get submission file

In [ ]:

In [ ]:

```
# X_test['d'].unique()
filename = 'features_1.pkl'

df = joblib.load(filename)
```

In [ ]:

```
# Get final dataframe with id.
df_final = df.drop(columns = ['date','weekday'])

df_valid = df_final[(df_final['d']>=1914) & (df_final['d']<1942)]
df_test = df_final[df_final['d']>=1942]


X_valid , y_valid = df_valid.drop('sold',axis=1), df_valid['sold']

X_test = df_test.drop('sold',axis=1)
```

In [ ]:
```
X_test
```

Out[36]:

| | id | item_id | dept_id | cat_id | store_id | state_id | d | wm_y |
|---|---|---|---|---|---|---|---|---|
| **59181090** | HOBBIES_1_001_CA_1_evaluation | 1437 | 3 | 1 | 0 | 0 | 1942 | |
| **59181091** | HOBBIES_1_002_CA_1_evaluation | 1438 | 3 | 1 | 0 | 0 | 1942 | |
| **59181092** | HOBBIES_1_003_CA_1_evaluation | 1439 | 3 | 1 | 0 | 0 | 1942 | |
| **59181093** | HOBBIES_1_004_CA_1_evaluation | 1440 | 3 | 1 | 0 | 0 | 1942 | |
| **59181094** | HOBBIES_1_005_CA_1_evaluation | 1441 | 3 | 1 | 0 | 0 | 1942 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **60034805** | FOODS_3_823_WI_3_evaluation | 1432 | 2 | 0 | 9 | 2 | 1969 | |
| **60034806** | FOODS_3_824_WI_3_evaluation | 1433 | 2 | 0 | 9 | 2 | 1969 | |
| **60034807** | FOODS_3_825_WI_3_evaluation | 1434 | 2 | 0 | 9 | 2 | 1969 | |
| **60034808** | FOODS_3_826_WI_3_evaluation | 1435 | 2 | 0 | 9 | 2 | 1969 | |
| **60034809** | FOODS_3_827_WI_3_evaluation | 1436 | 2 | 0 | 9 | 2 | 1969 | |

853720 rows × 64 columns

In [ ]:

In [ ]:

In [ ]:
```
# Making submission for validation dataset

X_valid['sold'] = y_valid_pred

submission_1 = X_valid[['id','d' ,'sold']]

submission_1 = pd.pivot(submission_1, index='id', columns='d', values='sold').res

# Ref. link :- https://stackoverflow.com/questions/28986489/how-to-replace-text-i
submission_1['id'] = submission_1['id'].str.replace('_evaluation','_validation')

submission_1.columns=['id'] + ['F' + str(i + 1) for i in range(28)]
```

In [ ]:

In [ ]:

In [ ]:

```
submission_1
```

Out[39]:

```
                                id        F1        F2        F3        F4        F5
    0       FOODS_1_001_CA_1_validation  1.090049  0.691359  0.757798  0.467963  0.880220  0.822
    1       FOODS_1_001_CA_2_validation  1.081022  1.058795  1.050121  1.032109  0.509392  0.877
    2       FOODS_1_001_CA_3_validation  0.954654  0.607795  0.762436  0.677451  0.907921  1.452
    3       FOODS_1_001_CA_4_validation  0.462395  0.240622  0.364826  0.240954  0.381104  0.426
    4       FOODS_1_001_TX_1_validation  0.444126  0.446315  0.476044  0.359166  0.475025  0.395
  ...                               ...       ...       ...       ...       ...       ...
30485  HOUSEHOLD_2_516_TX_2_validation  0.242864  0.237017  0.268691  0.264291  0.302106  0.262
30486  HOUSEHOLD_2_516_TX_3_validation  0.118108  0.212519  0.214481  0.115412  0.127480  0.192
30487  HOUSEHOLD_2_516_WI_1_validation  0.070076  0.070515  0.068988  0.069328  0.080053  0.105
30488  HOUSEHOLD_2_516_WI_2_validation  0.043630  0.043630  0.043630  0.064170  0.045180  0.046
30489  HOUSEHOLD_2_516_WI_3_validation  0.102776  0.091501  0.089739  0.091949  0.116414  0.135

30490 rows × 29 columns
```

In [ ]:

```
# Making submission for test dataset

X_test['sold'] = y_test

submission_2 = X_test[['id','d' ,'sold']]

submission_2 = pd.pivot(submission_2, index='id', columns='d', values='sold').res
submission_2.columns=['id'] + ['F' + str(i + 1) for i in range(28)]
```

In [ ]:

In [ ]:

In [ ]:

```
# Making final submission
submission = [submission_1 , submission_2]

final_submission_1 = pd.concat(submission)

final_submission_1.to_csv('submission_121.csv', index = False)
```

In [ ]: