Name: Yuen Zhikun Student id: 20505288 ITSC email: zyuen@connect.ust.hk
Name: Cheng Wai Kit Student id: 20520707 ITSC email: wkchengae@connect.ust.hk

Our Project: M5 Forecasting - Accuracy

## Description of the dataset:

There are 4 CSV file for the data.

`calendar.csv` : This csv file mainly stores Walmart's discount events in each day and the event types as well as the snap for the 3 states (CA, TX and WI) from 2011-01-29 to 2016-06-19.

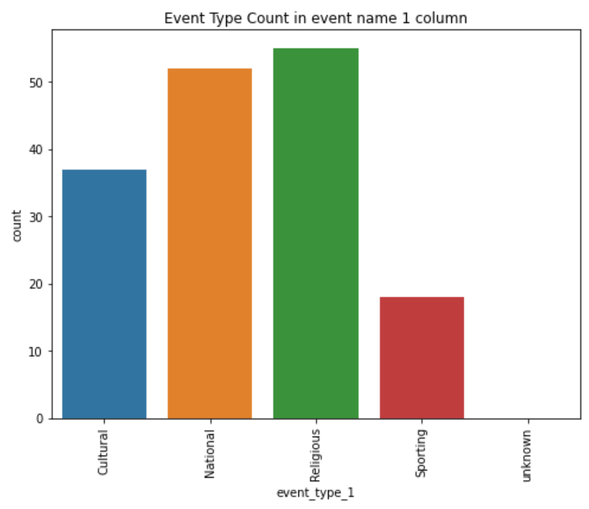
SNAP provides nutrition benefits to supplement the food budget of needy families so they can purchase healthy food and move towards self-sufficiency.

`sales_train_validation.csv`: This file stores all the sales for 30490 items from 2011-01-29 to 2016-06-19, but not all items are available on the start day 2011-01-29.

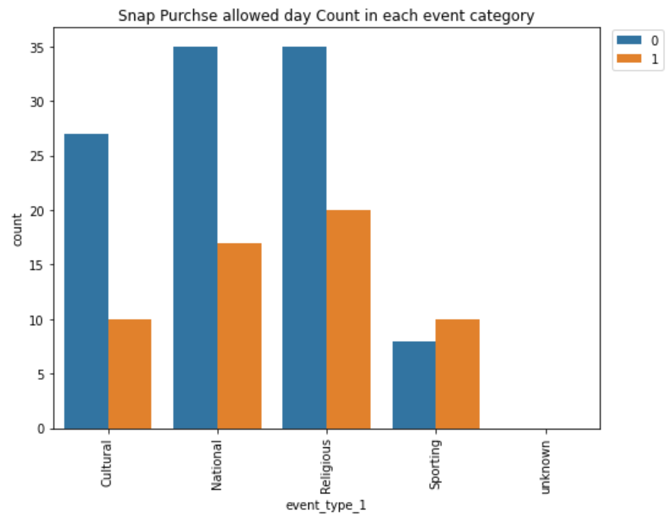
`sample_submission.csv`: This is just the submission csv. From day 1914 to day 1941, total 28 days.

`sell_prices.csv`: The sell prices of all 30490 items but not all of them are available from 2011-01-29 to 2016-06-19. There are 1048575 rows in this file, but we have 30490 items and 1913 days. Every item only has 34 days' record on average.

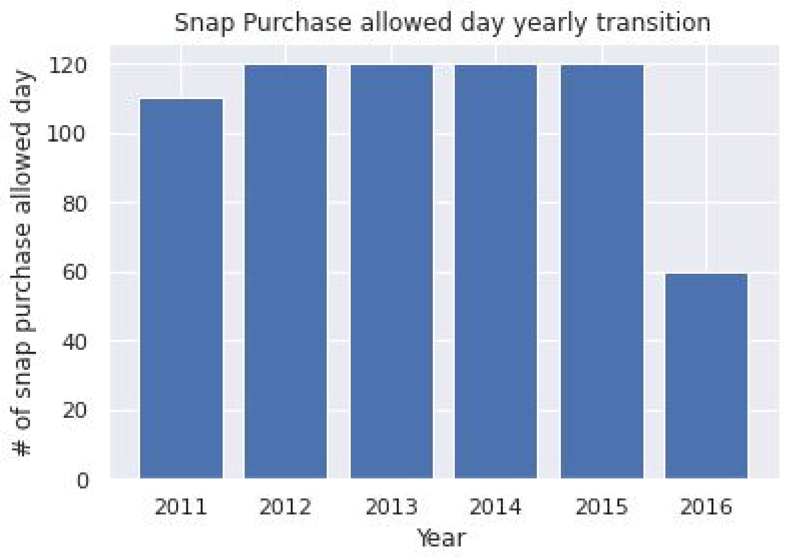
## Exploratary Data Analysis



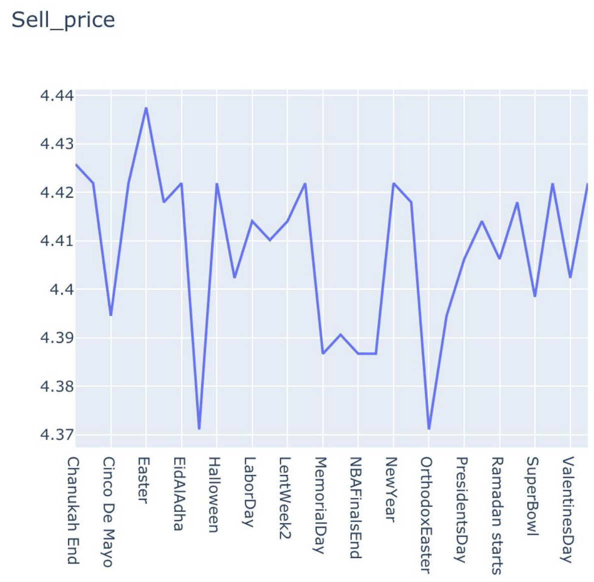
There are 4 event types of event 1 column.

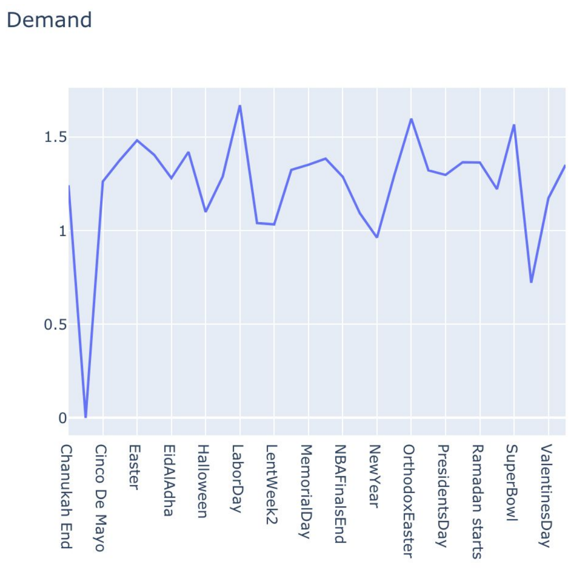


The relationship between snap and the event type 1.

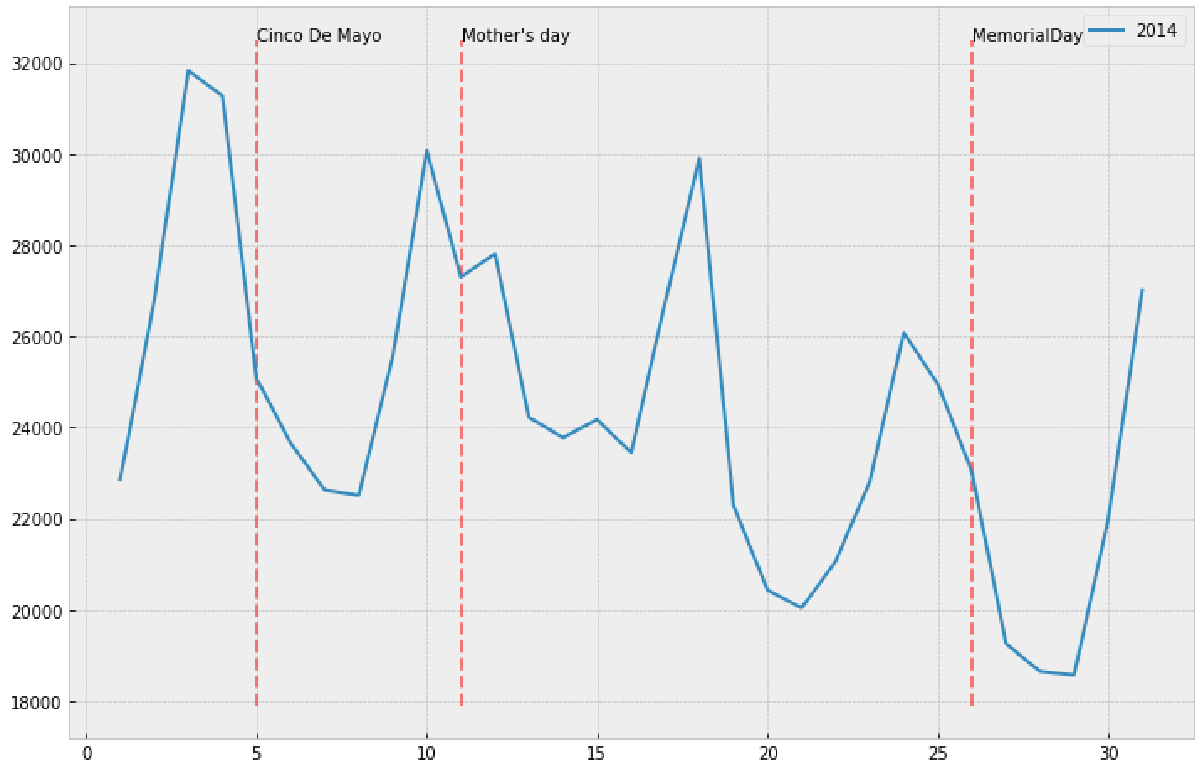


Since the data of 2016 was not covered the whole year (only to 19-06-2016), it only had the half of the other years.

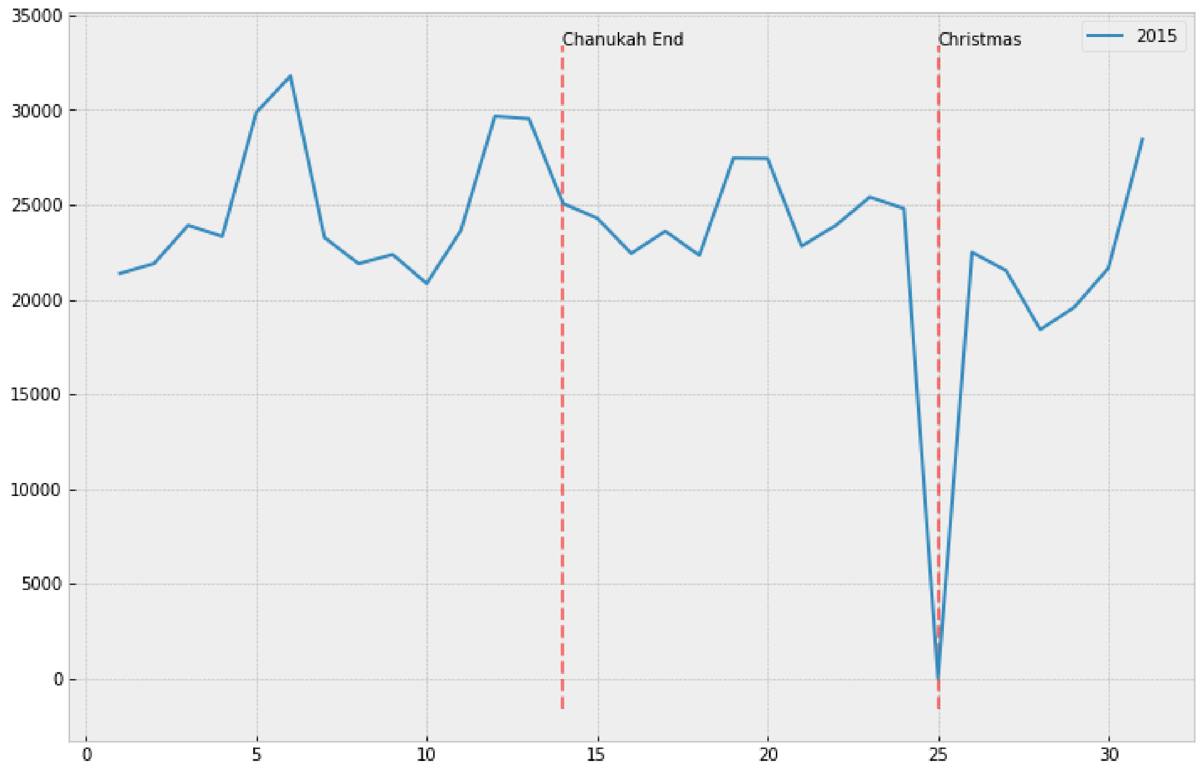




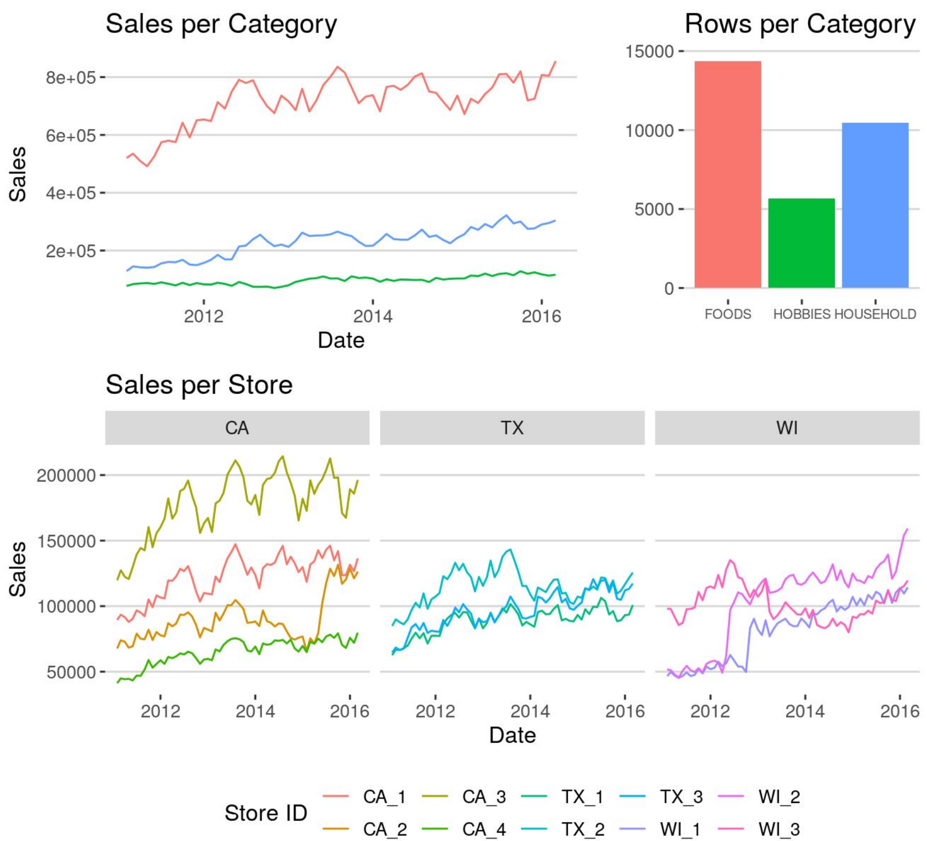
From these two graphs, we observe that the demand on some event was 0 and the sell price of EidAIAdha and OrthodoxEaster were lower that others.



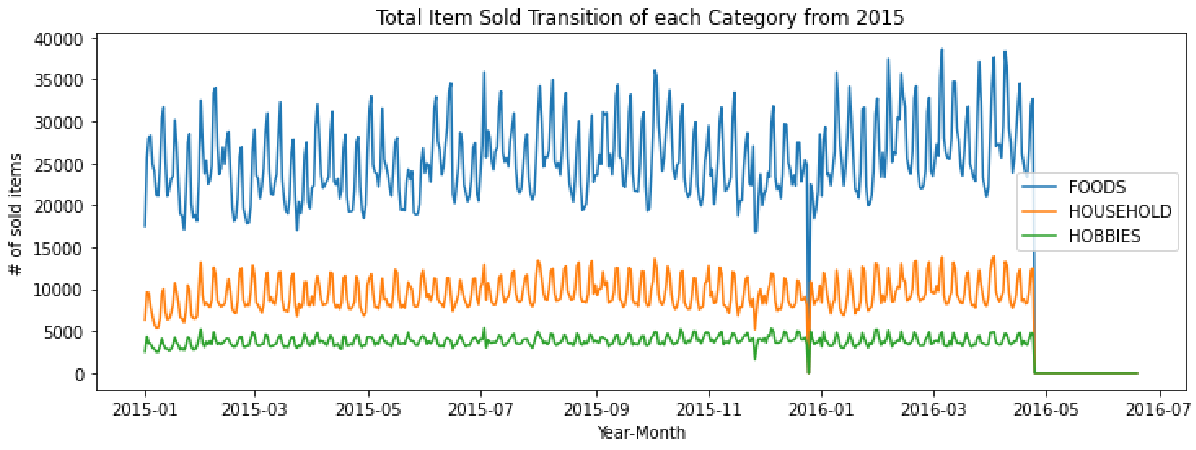
We can observe that there is a peak before the events and a dip for a few day after, so we need to add features to let the machine knows the events.



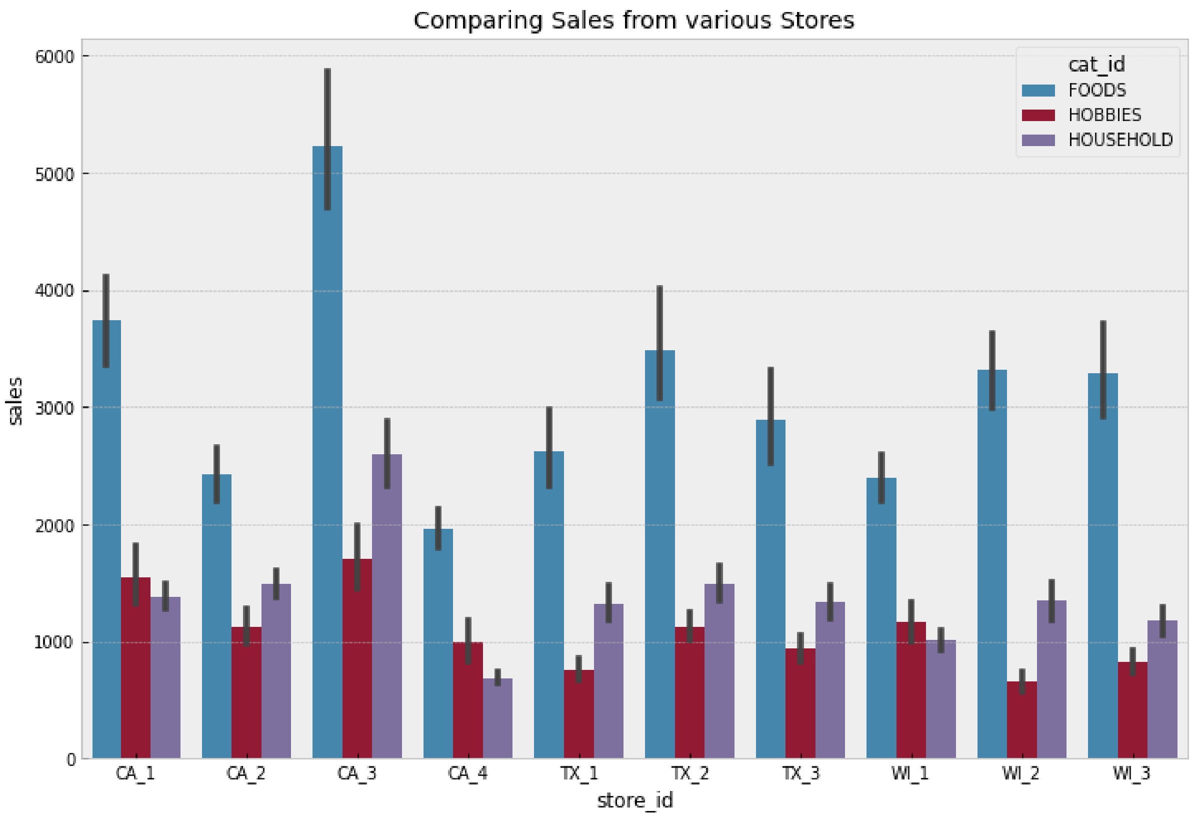
The sales on Christmas is zero which is quite reasonable.



There are 10 stores in 3 states and 3 product types (foods, hobbies, household). The total sales of food type is much higher than the other two. The total sales of CA_3 is much higher than the other 9 stores and tendency of the sales of all store were increasing with in the period.

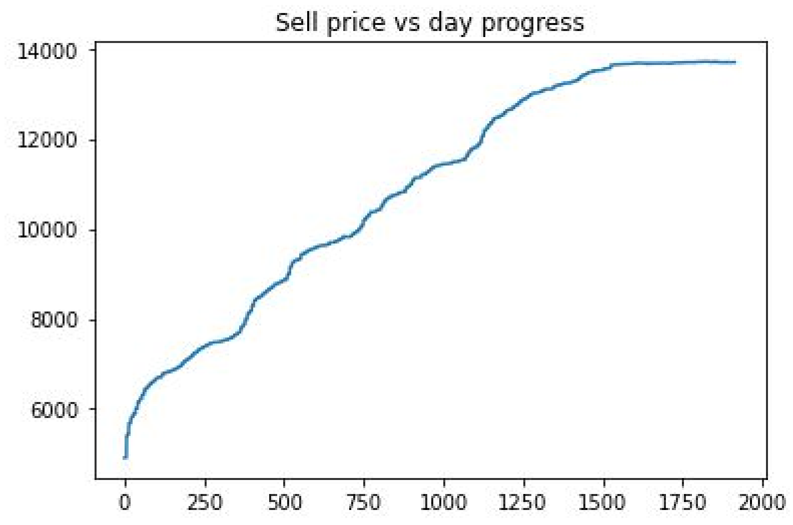


There was one day that the sales of three categories decreased to 0 as shops did not open at Christmas.

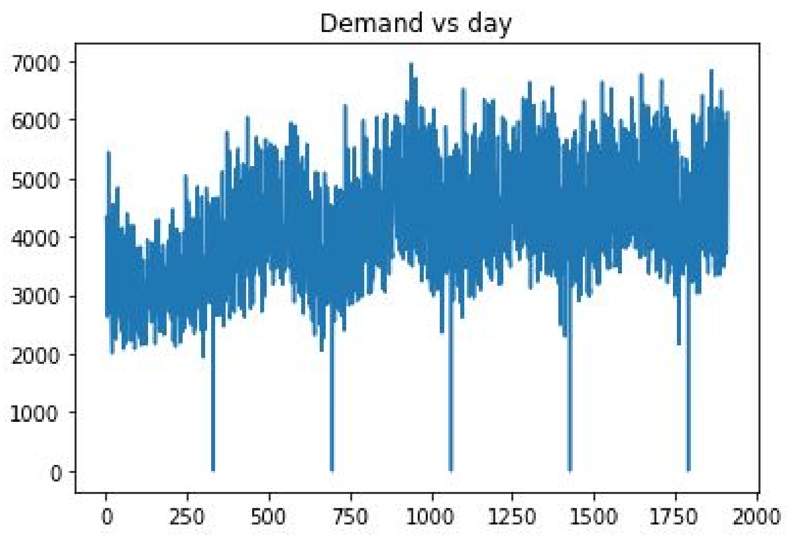


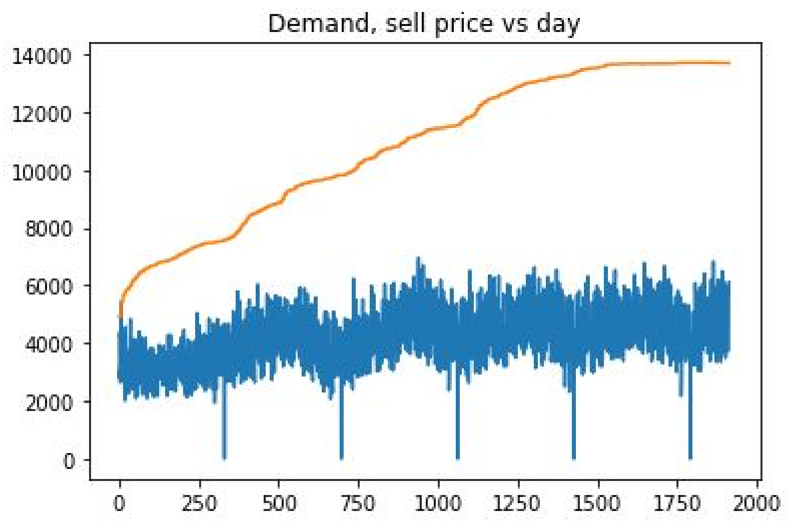
We can see that the total sales of FOODS in all stores are much higher than the other two types.

Sum of sell price vs day.

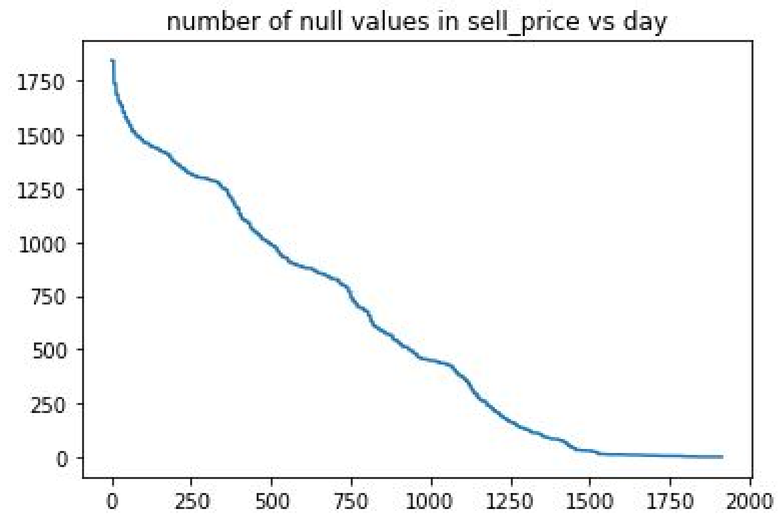


We see that the sell price increase over time generally, that could imply inflation.





We have also plotted the total sales vs day and put them into one picture to capture the trend. We noted that the total sales(demand) also increase in general.There could be some correlation between the sell_price and the total sales(unexpectedly, a positive relation). And the demand tend to be higher at the end of each year.



We also noted that there are many null values in sell_price column. That is because not all the products is in the supermarket at day 1. But with day progress, most of the products has price.

We also tried to investigate the relations of total sales between having an event on the day with no event. We also investigate the total sales in different event day.

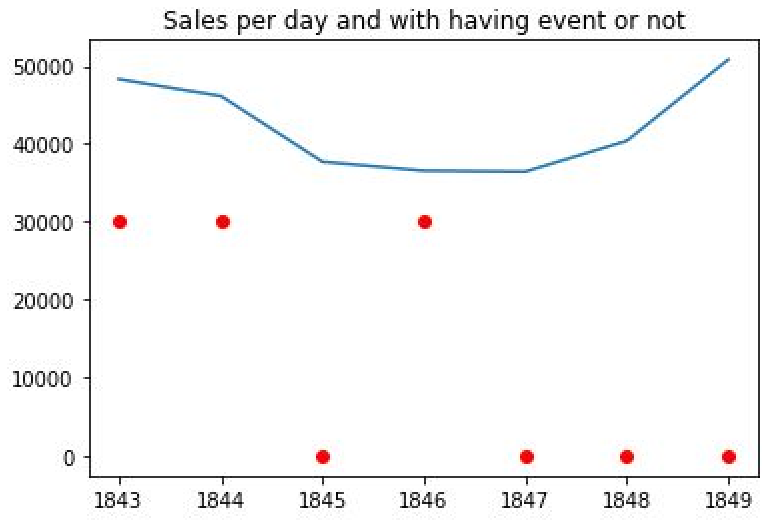
```
event_name_1
Chanukah End            33426.800000
Christmas                  15.600000
Cinco De Mayo           32823.500000
ColumbusDay             35164.800000
Easter                  39517.600000
Eid al-Fitr             35267.600000
EidAlAdha               34504.000000
Father's day            36562.000000
Halloween               30059.400000
IndependenceDay         33710.800000
LaborDay                42154.600000
LentStart               30727.000000
LentWeek2               29747.000000
MartinLutherKingDay     33378.600000
MemorialDay             32699.600000
Mother's day            34211.400000
NBAFinalsEnd            33564.200000
NBAFinalsStart          31874.400000
NewYear                 25768.400000
OrthodoxChristmas       33344.200000
OrthodoxEaster          38277.000000
Pesach End              33450.200000
PresidentsDay           34446.333333
Purim End               35957.000000
Ramadan starts          34779.600000
StPatricksDay           33459.666667
SuperBowl               40924.000000
Thanksgiving            21130.000000
ValentinesDay           33495.666667
VeteransDay             35118.800000
Name: demand, dtype: float64
```

Mean and SD of having event vs no event:

```
has_event                         has_event
0     34489.207504                0     7133.701801
1     32655.149351                1     9300.905832
Name: demand, dtype: float64      Name: demand, dtype: float64
```

Picking some days to visualise the total sales vs having event:

(red dot lies on y=30000 means having event on that day)

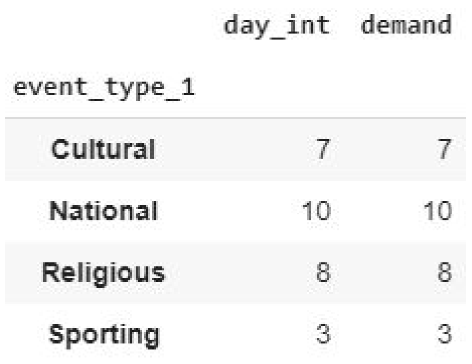


We also investigated the relation between event type and total sales. Moreover, we also try to see the distribution of number of days of each event type and the total sales on those day.

Mean of total sales in each event type compared with having event in the last year:

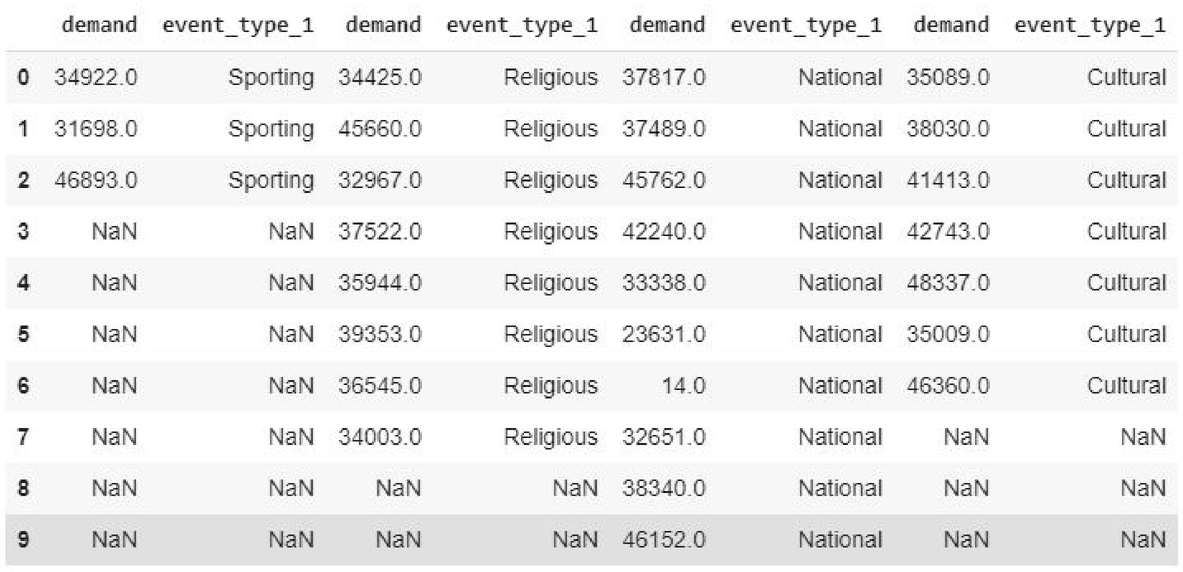
```
event_type_1
Cultural      40997.285714
National      33743.400000      has_event
Religious     37052.375000      0     39628.436202
Sporting      37837.666667      1     36940.964286
Name: demand, dtype: float64    Name: demand, dtype: float64
```

```
Number of days in each event type in the last year:
```



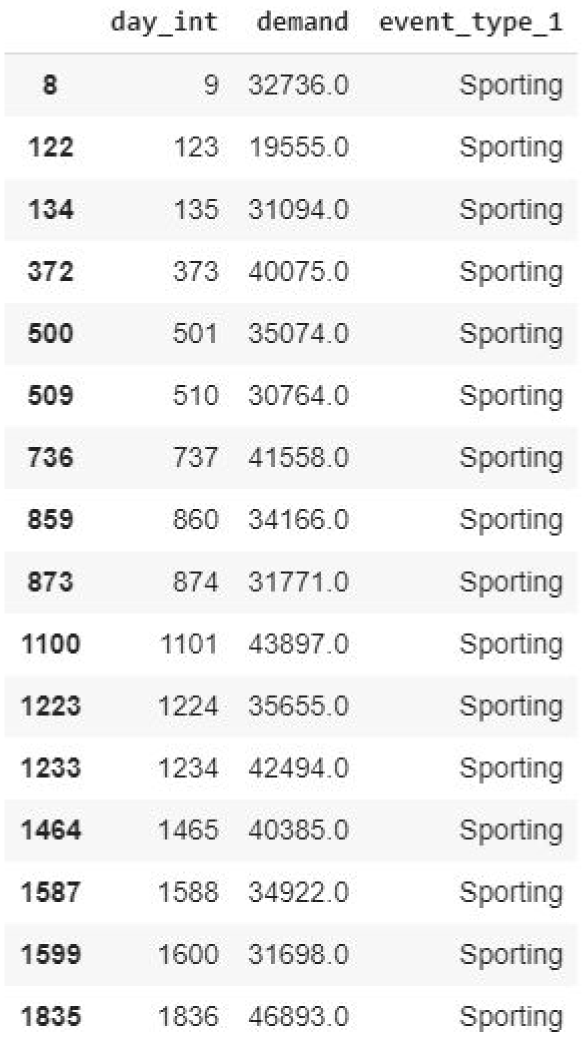
| | day_int | demand |
|---|---|---|
| event_type_1 | | |
| Cultural | 7 | 7 |
| National | 10 | 10 |
| Religious | 8 | 8 |
| Sporting | 3 | 3 |

```
Total demand of each day in each event type in the last year:
```



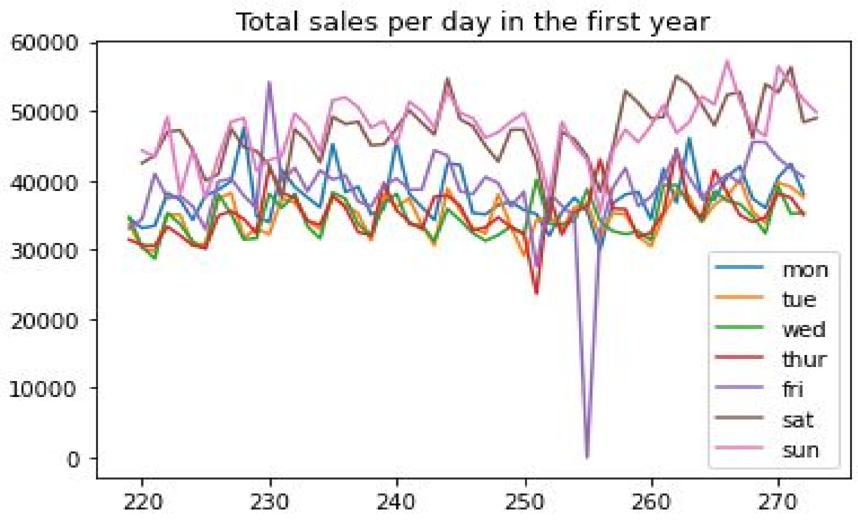
| | demand | event_type_1 | demand | event_type_1 | demand | event_type_1 | demand | event_type_1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 34922.0 | Sporting | 34425.0 | Religious | 37817.0 | National | 35089.0 | Cultural |
| 1 | 31698.0 | Sporting | 45660.0 | Religious | 37489.0 | National | 38030.0 | Cultural |
| 2 | 46893.0 | Sporting | 32967.0 | Religious | 45762.0 | National | 41413.0 | Cultural |
| 3 | NaN | NaN | 37522.0 | Religious | 42240.0 | National | 42743.0 | Cultural |
| 4 | NaN | NaN | 35944.0 | Religious | 33338.0 | National | 48337.0 | Cultural |
| 5 | NaN | NaN | 39353.0 | Religious | 23631.0 | National | 35009.0 | Cultural |
| 6 | NaN | NaN | 36545.0 | Religious | 14.0 | National | 46360.0 | Cultural |
| 7 | NaN | NaN | 34003.0 | Religious | 32651.0 | National | NaN | NaN |
| 8 | NaN | NaN | NaN | NaN | 38340.0 | National | NaN | NaN |
| 9 | NaN | NaN | NaN | NaN | 46152.0 | National | NaN | NaN |

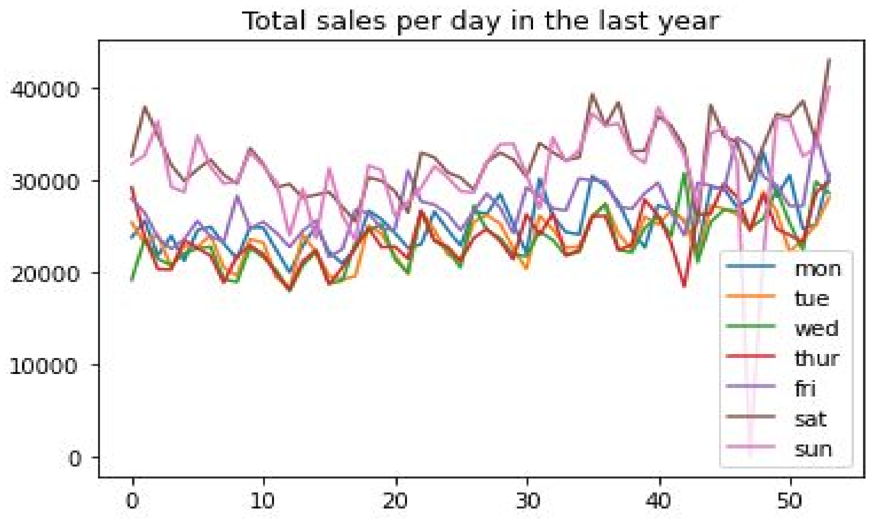
Specifically looked for the demand of the days with event sporting type in all 5 year:



| | day_int | demand | event_type_1 |
|---|---|---|---|
| **8** | 9 | 32736.0 | Sporting |
| **122** | 123 | 19555.0 | Sporting |
| **134** | 135 | 31094.0 | Sporting |
| **372** | 373 | 40075.0 | Sporting |
| **500** | 501 | 35074.0 | Sporting |
| **509** | 510 | 30764.0 | Sporting |
| **736** | 737 | 41558.0 | Sporting |
| **859** | 860 | 34166.0 | Sporting |
| **873** | 874 | 31771.0 | Sporting |
| **1100** | 1101 | 43897.0 | Sporting |
| **1223** | 1224 | 35655.0 | Sporting |
| **1233** | 1234 | 42494.0 | Sporting |
| **1464** | 1465 | 40385.0 | Sporting |
| **1587** | 1588 | 34922.0 | Sporting |
| **1599** | 1600 | 31698.0 | Sporting |
| **1835** | 1836 | 46893.0 | Sporting |

We also want to see how weekdays/weekend influence the total sales. We investigated the data from the first year and the last year.

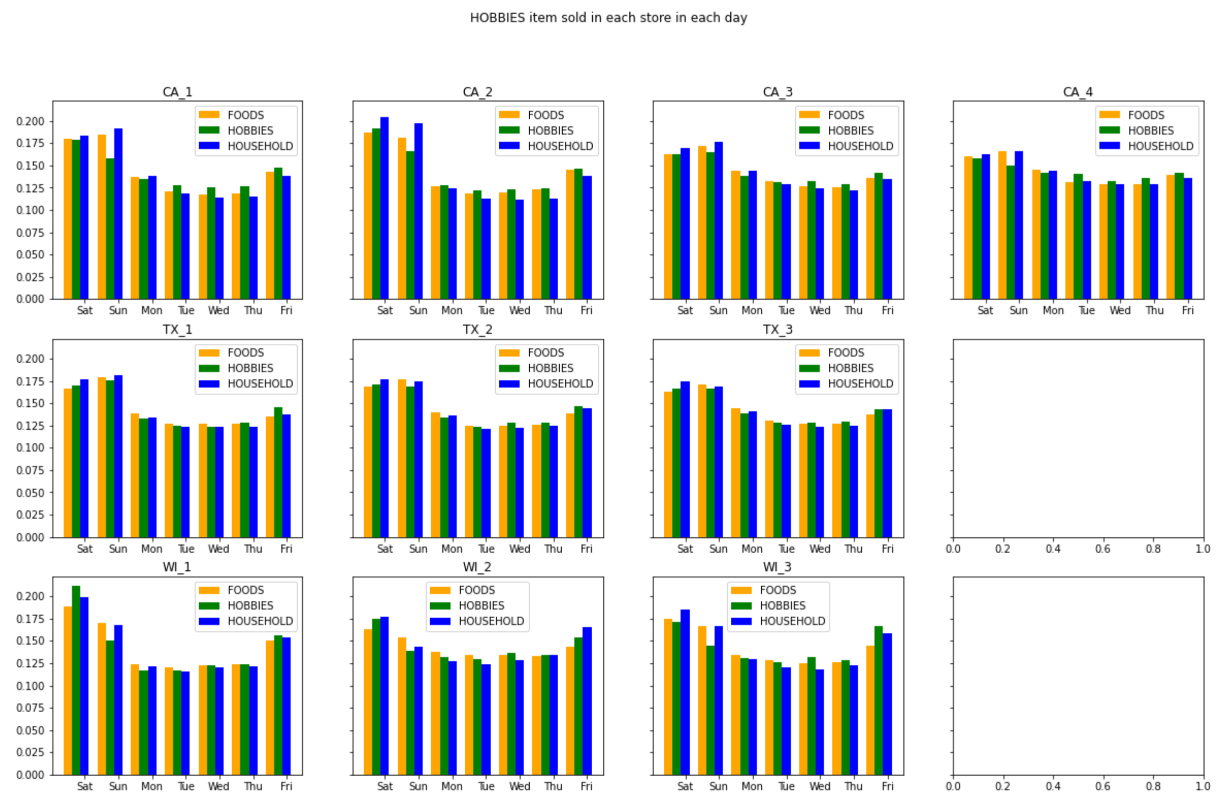


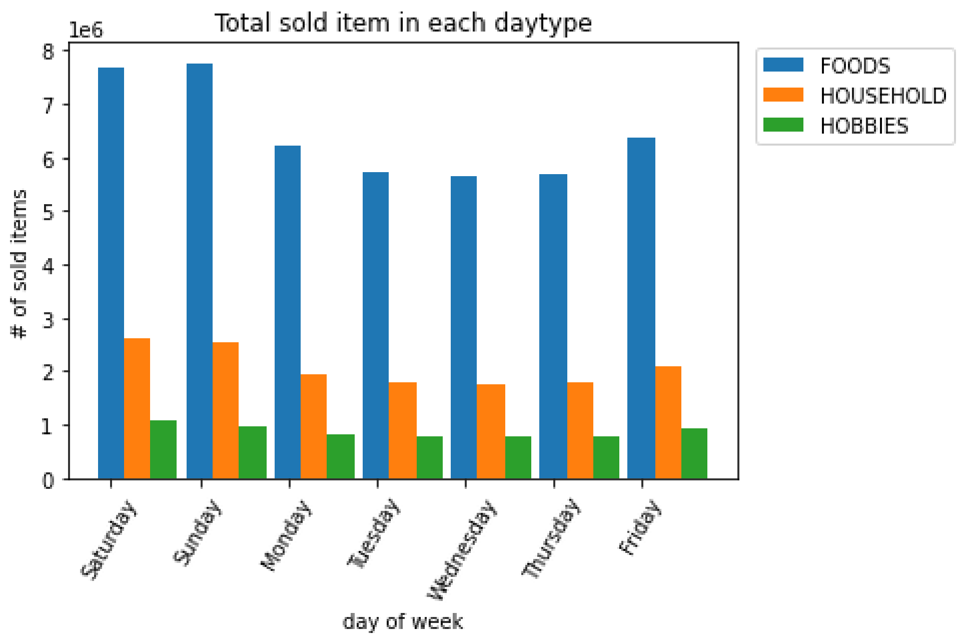


People tends to buy more in the weekend significantly.

Mean and SD of the total sales in each week days:

```
weekday                       weekday
Friday        1.122531        Friday        3.882051
Monday        1.077500        Monday        3.656730
Saturday      1.362640        Saturday      4.593460
Sunday        1.348968        Sunday        4.508957
Thursday      0.990653        Thursday      3.429051
Tuesday       0.996024        Tuesday       3.408766
Wednesday     0.984258        Wednesday     3.398607
Name: demand, dtype: float64  Name: demand, dtype: float64
```



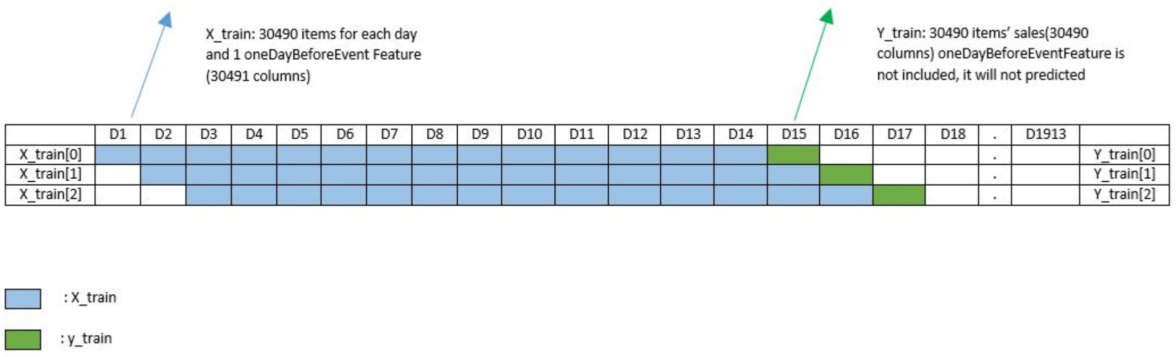


We can observe that in each store, the sales of the 3 types of product in weekend were higher than weekdays, so we need a feature to let the machine that the day is weekend or not.

**What we need to predict:**
We have the training data from 2011-01-29 to 2016-06-19 (total 1913 days). We need to predict the sales of Walmart's 30490 items from day 1914 to day 1941. Walmart is an American multinational retail corporation that operates a chain of hypermarkets, discount department stores, and grocery stores, headquartered in Bentonville, Arkansas.

**Data preprocessing:**
There are two different hypothesises to consider those data. One the time sequence (depends most on the past sales record) and one is non-time series (tree base, depends on weekend or weekday, what events, sell price). We separate them into two parts with two different ways to do data processing.

**Part 1 (time sequence):**


X_train: 30490 items for each day and 1 oneDayBeforeEvent Feature (30491 columns)
Y_train: 30490 items' sales(30490 columns) oneDayBeforeEventFeature is not included, it will not predicted
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 . D1913
X_train[0]
X_train[1]
X_train[2]
Y_train[0]
Y_train[1]
Y_train[2]
: X_train
: y_train

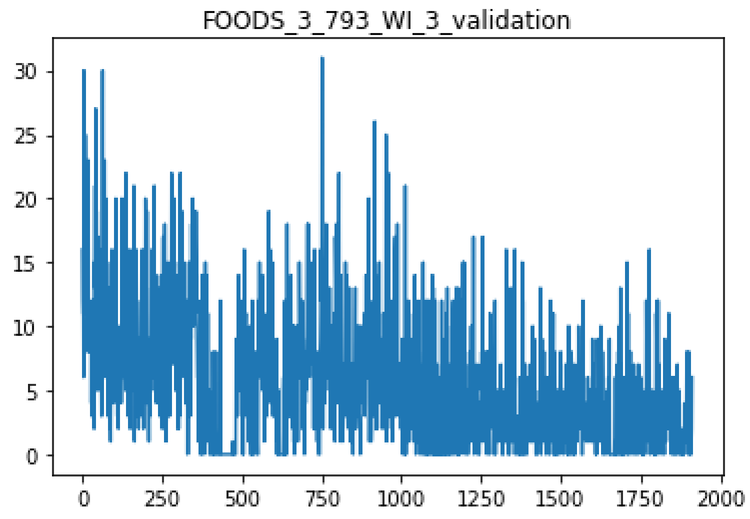
"X_train" and "y_train" data is created. For each X_train item, 14 past days' sales and some other time feature, like weekend or not, event on that day... are included. So one element of X_train's shape is (14, 30498). For y_train we are predicting one day sales of 30490 items therefore one element of y_train's shape is (1, 30490)

- x_train[0]:day 1 to 28, such that each day has 30490 sales and 1 weekend feature, 4 event_type feature (what type event on that day) and snap feature (snap in three states).
- x_train[1]:day 2 to 29, such that each day has 30490 sales and 1 weekend feature, 4 event_type feature (what type event on that day) and snap feature (snap in three states).
- x_train[2]:day 3 to 30, such that each day has 30490 sales and 1 weekend feature, 4 event_type feature (what type event on that day) and snap feature (snap in three states).
- ……
- x_train[1827]: day 1828 to 1855, such that each day has 30490 sales and 1 weekend feature, 4 event_type feature (what type event on that day) and snap feature (snap in three states).

We use 97% for training data and 3% (last 30 days for validation).

```
The shape of features of train is: torch.Size([1827, 28, 30498])
The shape of labels of train is: torch.Size([1827, 30490])
The shape of features of train is: torch.Size([30, 28, 30498])
The shape of labels of train is: torch.Size([30, 30490])
```

Apart from the shape of the training data. We need to use `MinMaxScaler(feature_range = (0, 1))` normalize some features like sales of all 30490 items as the sales of some products are high and some are lower and other time features are one-hot encoding (weekend, cultural, national, religious,sporting, snap_CA, snap_TX, snap_WI). The MinMaxScaler would considers each columns of the 30490 items independently to do normalize.



Pick an item randomly, the highest sales is 30 and the lowest is 0.

Since the data set is too large, we need to use data loader with a simple customized dataset to load the data with a batch size for each iteration. Otherwise, Colab would crash, even Colab Pro has 25 GB RAM.

We only prepare other two data set for the best model in the above data set to predice. The first one is only use the last 500 days for training, not all data from 2011-01-29 to 2016-06-19. The second one is only focus on 1 item rather than 30490 items.

**Part 2 (non-time sequence):**

The data preprocessing used for tree-based model is simpler than the time sequence one because as no context window should be taken into consideration.

feature selection:

```
features = ['item_id', 'dept_id', 'cat_id', 'store_id', 'state_id',
'wday', 'month',
      'event_name_1', 'event_type_1', 'event_name_2', 'event_type_2',
      'snap_CA', 'snap_TX', 'snap_WI', 'sell_price', 'demand',
'day_int']
```

Obviously, item_id, depth_id, cat_id, store_id, state_id are predictive features as they combined to specify item. Apart from the item itself, the 'time' is also an important feature for predicting the sales.

For example, the following questions are important:

Is it a holiday? (related to people willingness and time to consumption)
Which month is it? (related to the sales of seasonal product i.e. ice-cream tends to sell better in July than in January)
Is there any important event happening? (potentailly stimulate people shopping intention, i.e. Halloween)
Is there a SNAP? (subsidising people to shop?)
What is the selling price? (by the law of demand, price lower means quantity demand higher)

To answer these question, the remaining features is required.

'day_int' converted 'day' to 'int', i.e. d_1 -> 1 # the first day, d_1900 -> 1900 # the 1900th day which give us a good idea about the demand of which specific date we are talking about.

Melting DataFrame:

The form of the default data frame is not very convenience for us to analyse, we must put the feature and demand ('sales' that we are predicting) in a row such that it is convenience to feed into a tree instead of putting extra dimension of timestamps into the model. Thus, we melt the data frame first, then select the features.

At the end, we noted that there are some features that is important to the prediction of the sales of the items that is not included in the dataset. For example, the expand of Walmart supermarket (e.g. increase in market share, people as a result tend to go to Walmart more often and hence a general increase in the sales of the item), the economy of the US (is people getting richer?), the inflation rate (the demand tends to drop further if the price rise is not due to inflation). By the fact that we cannot take these features into our model, we should make an assumption for these features to remain constant during our training and testing phrase. Therefore, we only used the data after 2015 since we are predicting for the sales of 2016, it is reasonable for us to assume those economic environment factors remain unchanged or at least not changing drastically during the time 2015 - 2016.

```
train_melt_2015 = train_melt[train_melt['date'] >
pd.to_datetime('2015-05-19')]
```

Converting Categorical features:

We make use of the technique of label encoder to convert the categorical data.

```
X['item_id_int'] = X['item_id'].apply(lambda x: x[-3:]).astype(int)
X['dept_id'] = X['dept_id'].apply(lambda x: x[-1]).astype(int)
X['store_id'] = X['store_id'].apply(lambda x: x[-1]).astype(int)
X['event_name_1'] = le.fit_transform(X['event_name_1'].astype(str)) + 1
X['event_type_1'] = le.fit_transform(X['event_type_1'].astype(str)) + 1
X['event_name_2'] = le.fit_transform(X['event_name_2'].astype(str)) + 1
X['event_type_2'] = le.fit_transform(X['event_type_2'].astype(str)) + 1
X['cat_id'] = le.fit_transform(X['cat_id'].astype(str)) + 1
X['state_id'] = le.fit_transform(X['state_id'].astype(str)) + 1
```

item_id, dept_id, store_id did not make use of the python label encoder, but the techniques involved is indeed label encoding.

We know that the categorical data we are dealing with is not ordinal, normally one-hot encoding is more preferred. However, in this case, we found that label encoding is more reasonable. Because:

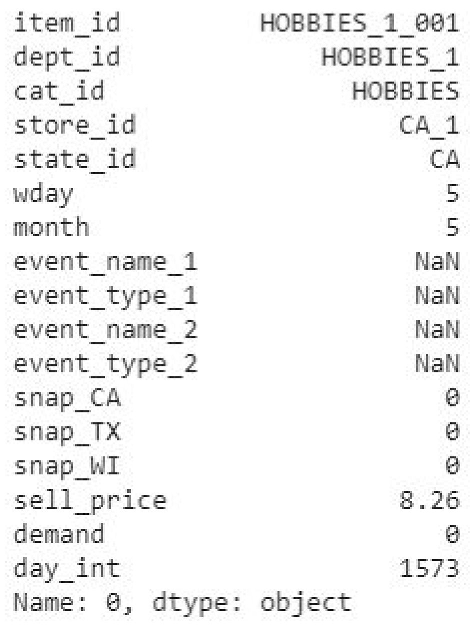
1) it does not harm too much to tree-based model
2) it is a huge dataset and there are many categories for 1 categorical feature, one-hot encoding is too computational expensive.

Tackling null value in “sell_price”

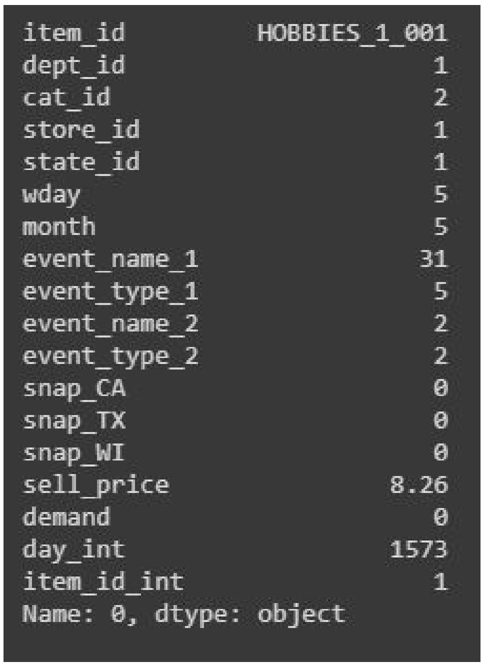
We have found that many entries is null in the column “sell_price”, we put 0 to replace the null value as a result no sell_price other than null values should take 0. Thus giving the tree-based model a hint to know whether the sell_price is 0.
e.g. a node may ask (is sell_price < 0.001) yes: do subsequent analysis based on the fact that sell_price is a null value.

The final features used as training data :

Before vs After



```
item_id          HOBBIES_1_001
dept_id              HOBBIES_1
cat_id                 HOBBIES
store_id                  CA_1
state_id                    CA
wday                         5
month                        5
event_name_1               NaN
event_type_1               NaN
event_name_2               NaN
event_type_2               NaN
snap_CA                      0
snap_TX                      0
snap_WI                      0
sell_price                8.26
demand                       0
day_int                   1573
Name: 0, dtype: object
```

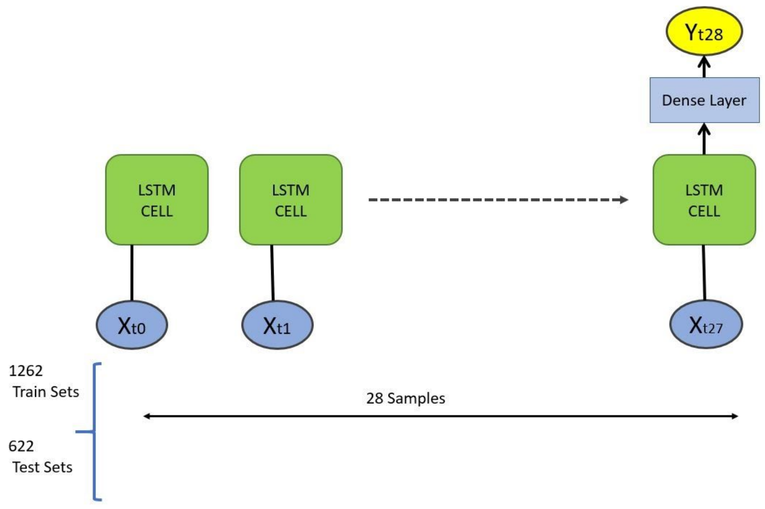


```
item_id          HOBBIES_1_001
dept_id                      1
cat_id                       2
store_id                     1
state_id                     1
wday                         5
month                        5
event_name_1                31
event_type_1                 5
event_name_2                 2
event_type_2                 2
snap_CA                      0
snap_TX                      0
snap_WI                      0
sell_price                8.26
demand                       0
day_int                   1573
item_id_int                  1
Name: 0, dtype: object
```

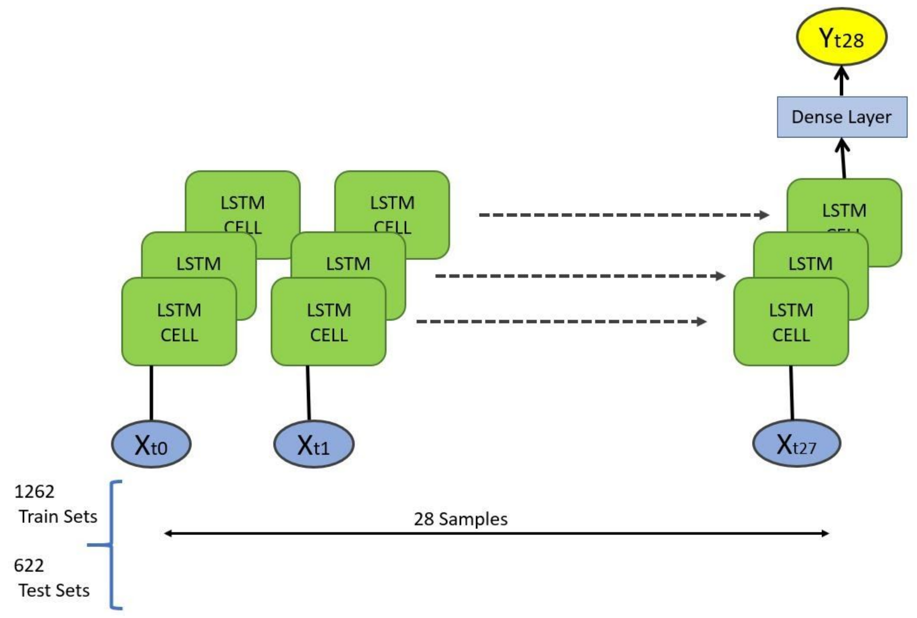
## Machine learning methods used for solving the task

**Part 1(time sequence):**
Hardware: T4 or P100 GPU
Software: Colab Pro
We use several time series models like 3 layer RNN, 1 layer LSTM, 3 layer LSTM, 3 layer bidirectional LSTM and 3 layer GRU on the first part. Since the data is time series, use RNN or its variants like LSTM/GRU would be one of the possible ways. The **window size** is 28 days and we use the previous 28 days' data to predict the day 29.

1 layer LSTM:



3 layers LSTM:



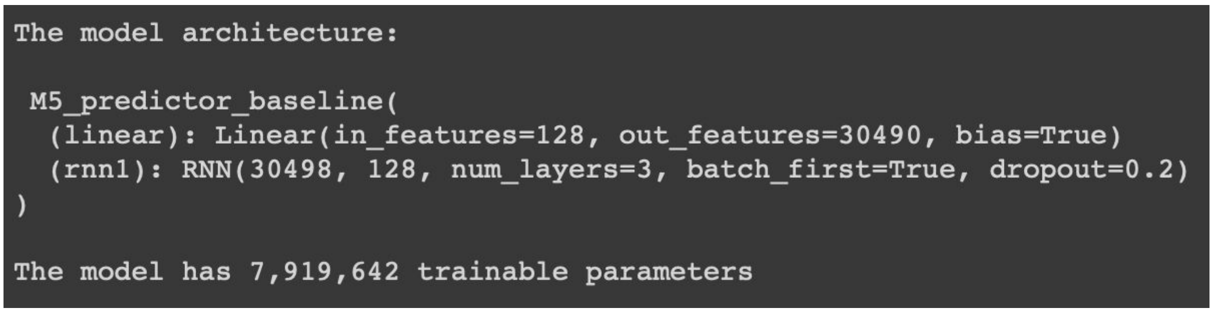
Parameters of different models:

| Model | No. layer | bidirectional | Batch size | dropout | learning rate | hidden units | Optimizer | Loss function: |
|---|---|---|---|---|---|---|---|---|
| RNN | 3 | No | 56 | - | 0.001 | 128 | Adam | MSE |
| LSTM1 | 1 | No | 56 | 0.2 | 0.001 | 128 | Adam | MSE |
| LSTM2 | 3 | No | 56 | 0.2 | 0.001 | 128 | Adam | MSE |
| LSTM3 | 3 | Yes | 56 | 0.2 | 0.001 | 128 | Adam | MSE |
| GRU | 3 | No | 56 | 0.2 | 0.001 | 128 | Adam | MSE |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| GRU | 3 | No | 28 | 0.2 | 0.01 | 128 | Adam | MSE |

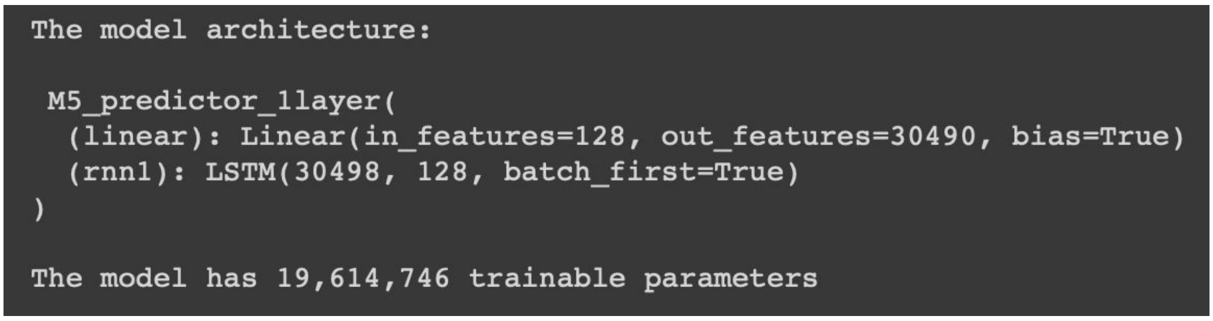
Architecture:

RNN Model:



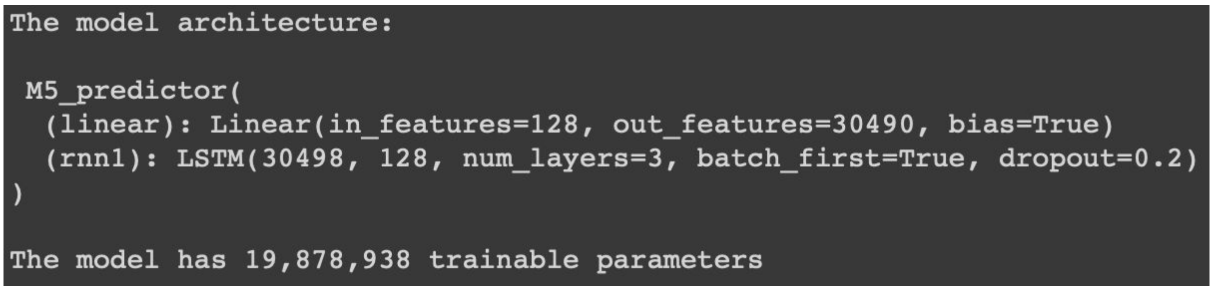
```
The model architecture:

 M5_predictor_baseline(
  (linear): Linear(in_features=128, out_features=30490, bias=True)
  (rnn1): RNN(30498, 128, num_layers=3, batch_first=True, dropout=0.2)
)

The model has 7,919,642 trainable parameters
```

LSTM 1 (1 layer):



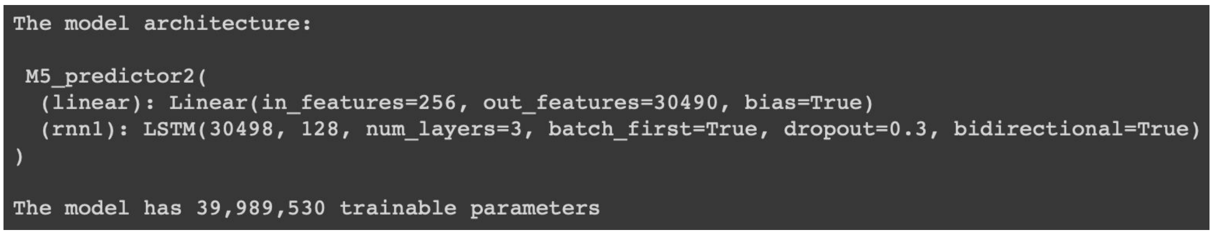
```
 The model architecture:

  M5_predictor_1layer(
   (linear): Linear(in_features=128, out_features=30490, bias=True)
   (rnn1): LSTM(30498, 128, batch_first=True)
 )

 The model has 19,614,746 trainable parameters
```

LSTM 2 (3 layers):



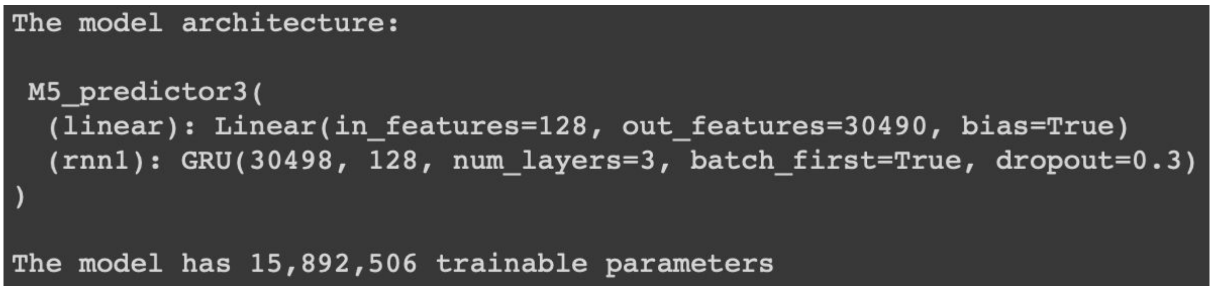
```
The model architecture:

 M5_predictor(
  (linear): Linear(in_features=128, out_features=30490, bias=True)
  (rnn1): LSTM(30498, 128, num_layers=3, batch_first=True, dropout=0.2)
)

The model has 19,878,938 trainable parameters
```

LSTM 3(3 layers with bidirectional):



```
The model architecture:

 M5_predictor2(
  (linear): Linear(in_features=256, out_features=30490, bias=True)
  (rnn1): LSTM(30498, 128, num_layers=3, batch_first=True, dropout=0.3, bidirectional=True)
)

The model has 39,989,530 trainable parameters
```

GRU(3 layers):



```
The model architecture:

 M5_predictor3(
  (linear): Linear(in_features=128, out_features=30490, bias=True)
  (rnn1): GRU(30498, 128, num_layers=3, batch_first=True, dropout=0.3)
)

The model has 15,892,506 trainable parameters
```

The bidirectional LSTM has the most parameters and the GRU has less parameters that LSTM.

**Part 2(non-time sequence):**

In this part, we will mainly explore tree-based model.
Random Forest is a versatile choice, and we will also explore XGBoost and Lightgbm 2 kinds of boosting tree which often win a kaggle competition.

The selection of tree-based model is due to the observation that the data is probably not normal distributed, and also inspired by the natural way of thinking if we would like to estimate the sale of an item on a particular date. For example, if we were asked to give an estimate of the sales of an item on 20/5, we would naturally ask what is the item (so we are asking what is the item_id)? Is 20/5 a holiday (wday)? is there any event happening on 20/5 such that stimulate the sale (event_name, event_type)?

The way of how we determine what estimate we will give is like a decision tree, instead of try to link different variable and forming functions.

```
rf = RandomForestRegressor(n_estimators = 100,
                          n_jobs = -1,
                          random_state = 4211,
                           max_depth = 20,
                           verbose = 1)
```
The above is the parameter setting of the random forest.

```
xgb = XGBRegressor(max_depth = 10,
                  n_jobs = -1,
                  random_state = 4211,
                  n_estimator = 50,
                  verbose = 1
                  )
```
The above is the parameter setting of the XGoost.

```
bst_params = {
   "boosting_type": "gbdt",
   "metric": "rmse",
   "objective": "regression",
   "n_jobs": -1,
   "seed": 4211,
   "learning_rate": 0.1,
   "bagging_fraction": 0.75,
   "bagging_freq": 10,
   "colsample_bytree": 0.75,
}

fit_params = {
   "num_boost_round": 100_000,
   "early_stopping_rounds": 200,
   "verbose_eval": 100,
}
lgb_regressor = lgb.train(bst_params, lgb_train, … )
```

The above is the parameter setting of the Lightgbm.

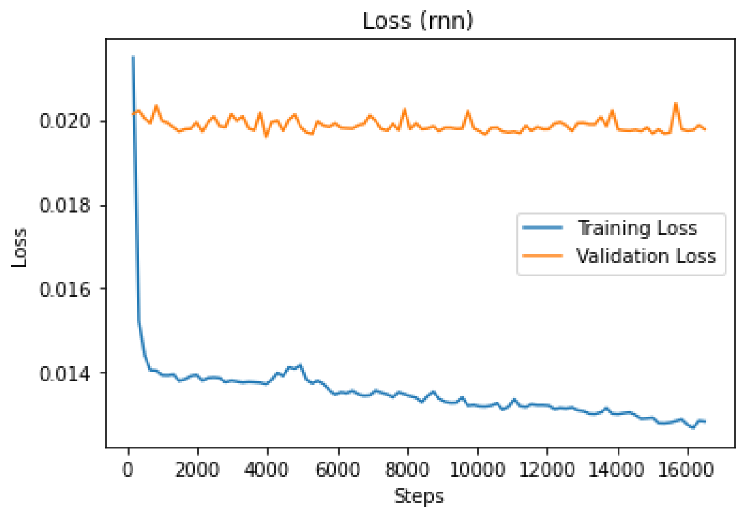
## Description of the machine learning task performed on the dataset

## and Experiments and results:

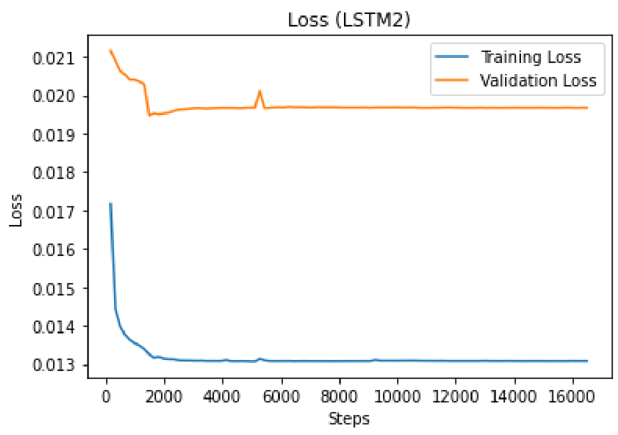
### Part 1(time series):

We do 500 epochs on each model and use the best training model to predict the data on the validation set.

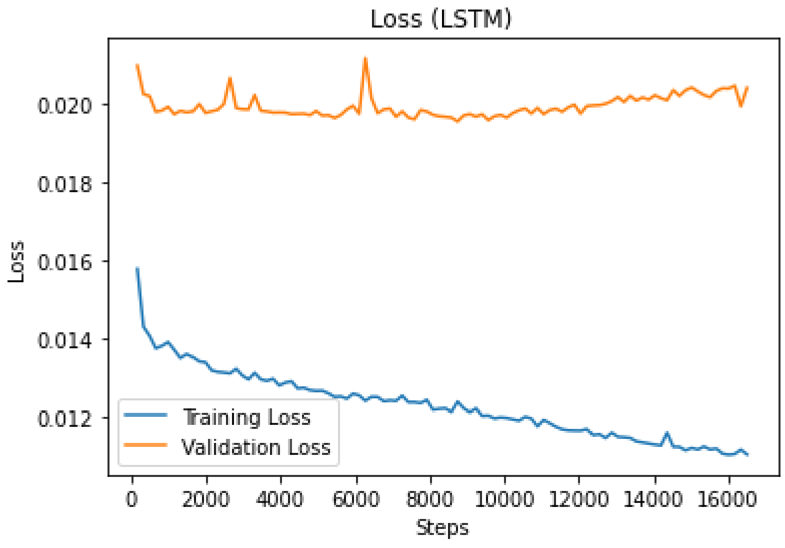
Loss of Rnn:



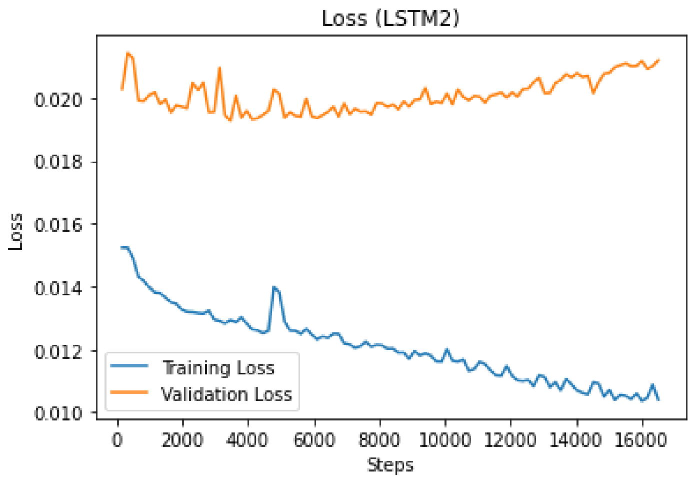
Loss of LSTM1 (1 layer):



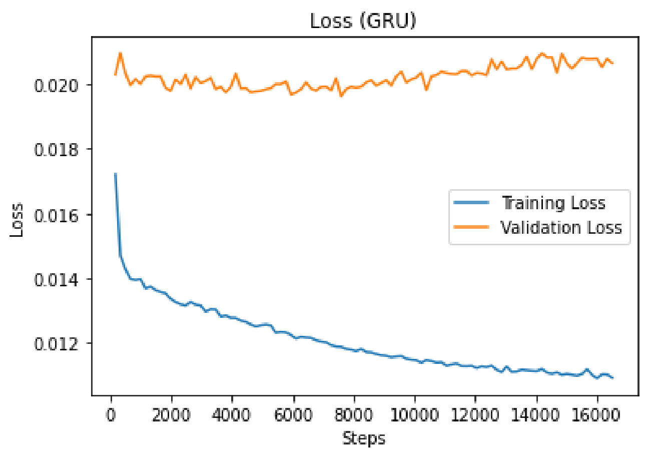
Loss of LSTM2 (3 layers):



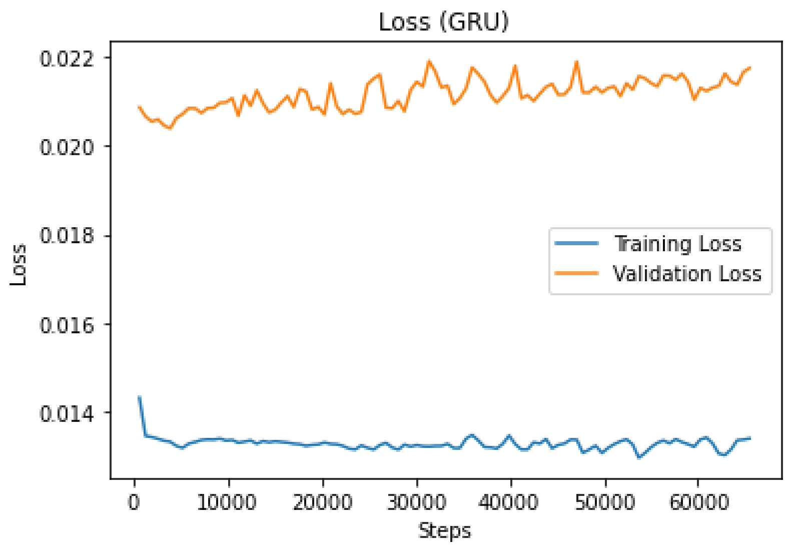
Loss of LSTM3 (3 layers with bidirectional):



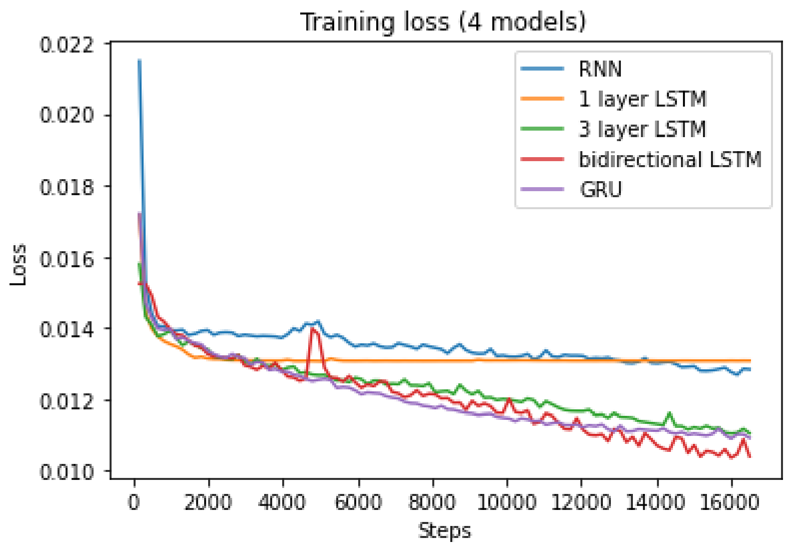
Loss of GRU:



Loss of GRU(batch size = 14):

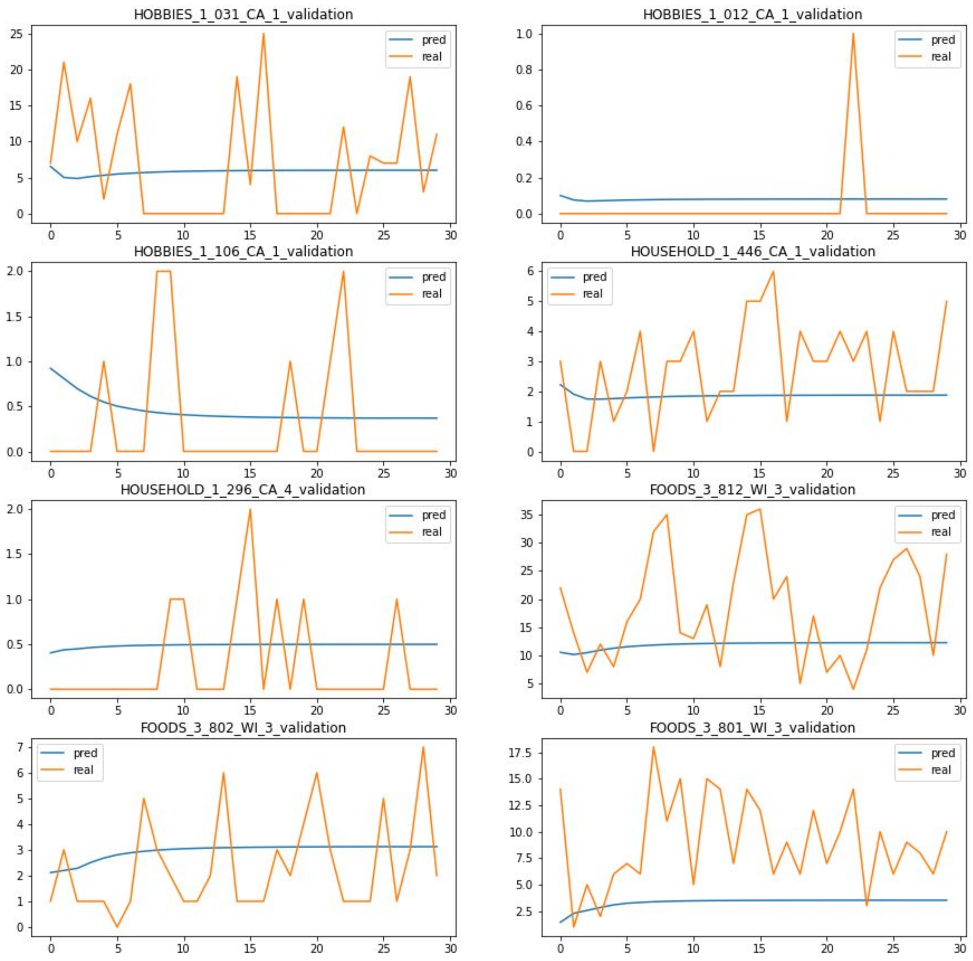


The training loss of the 5 models:



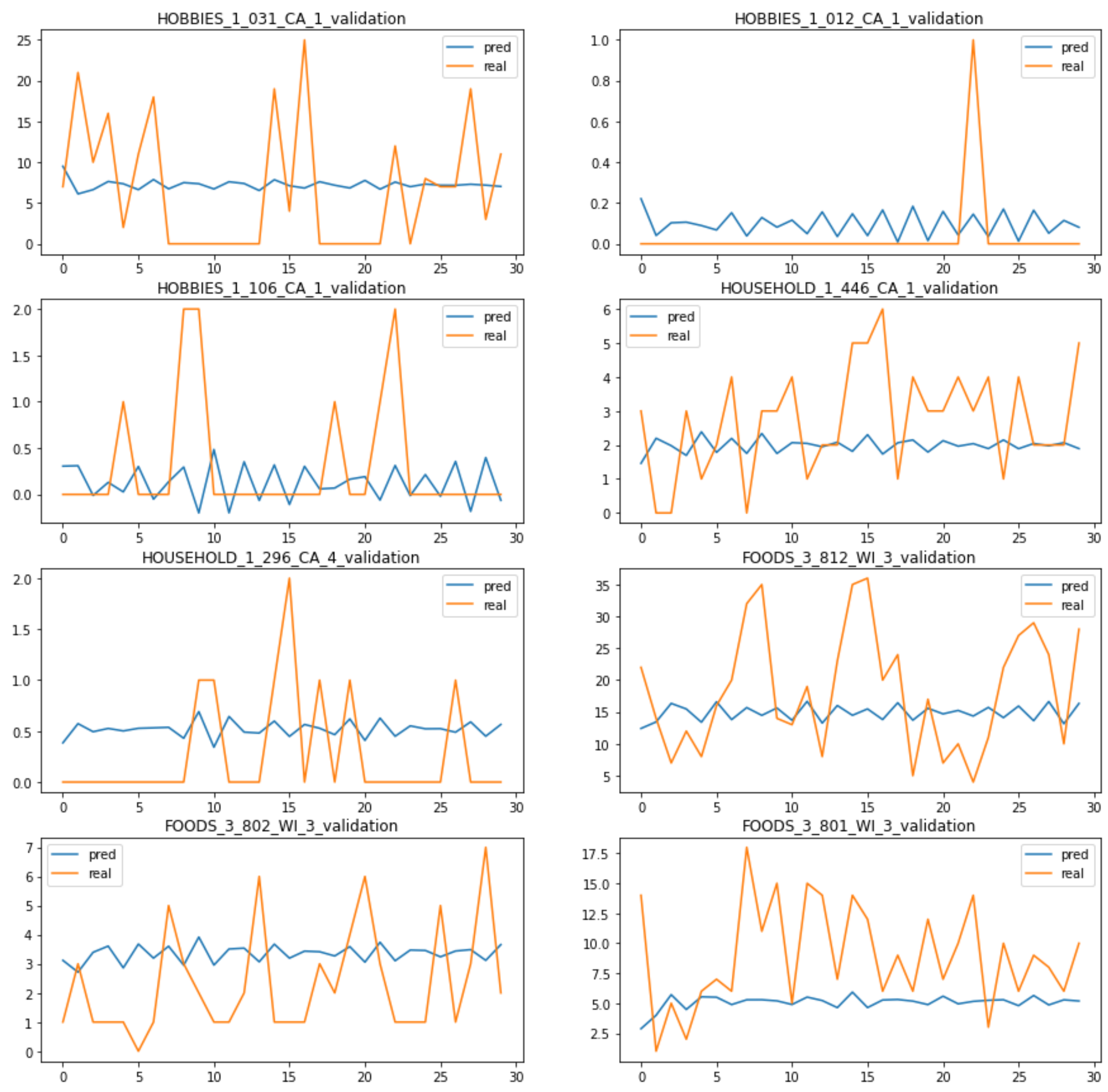
From this graph, it seems that bidirectional LSTM is the best model as its lowest training loss, but if we compare the validation loss, we can observe that the validation loss do not decrease too much, there may be some overfitting on the training set. However, if we choose the LSTM 2 model (3 layers) with the lowest validation loss, you can see the model only predict the sales with a straight line and the validation is the lowest. However, we want the prediction more vibration on the prediction. With the smaller batch size = 14, the loss converges much faster than the GRU with batch size = 56. It converges before 5000 steps but the GRU with size = 56 converges after 12000 steps.

The following graph is the prediction with the lowest validation loss of the LSTM with 3 layer



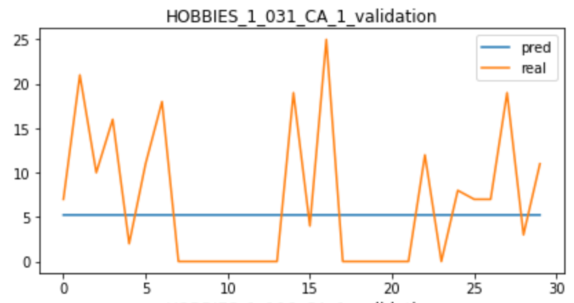
Then, we change to use the model with the lowest training loss and randomly pick 8 items from 30490 items to do visualization.

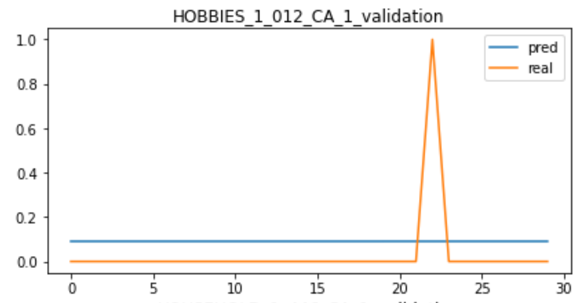
RNN:

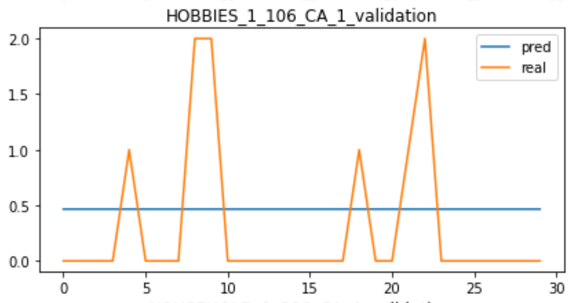


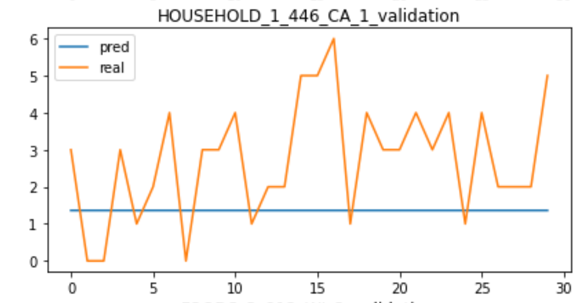
The prediction performance of the lowest training loss of RNN is slightly better than the 3 layers LSTM model with loss validation loss. This model would not predict a straight line (same sales on each day) however the tendency is not too different from the straight one. However, the tendency does not match the real one.

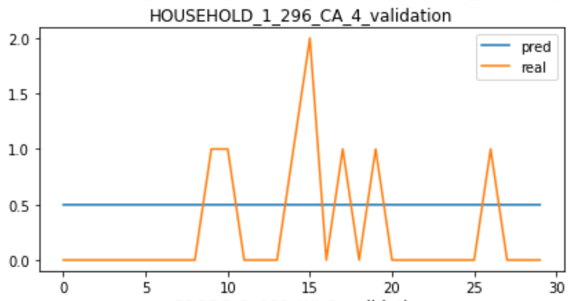
LSTM1 (1 layer):

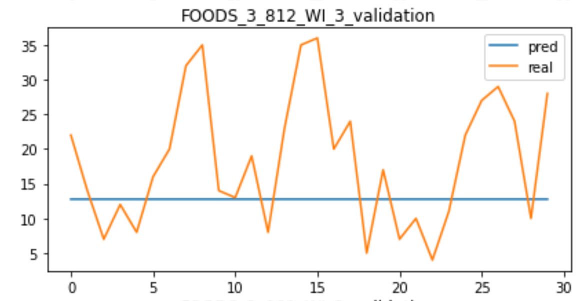


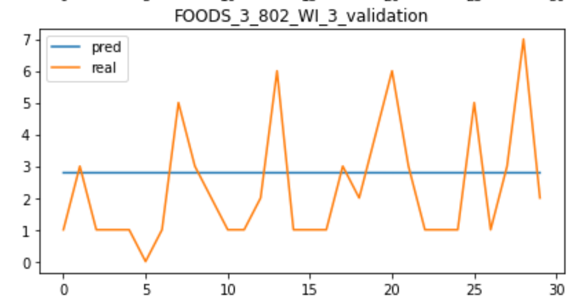


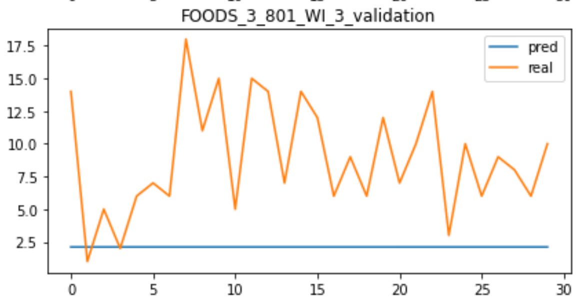






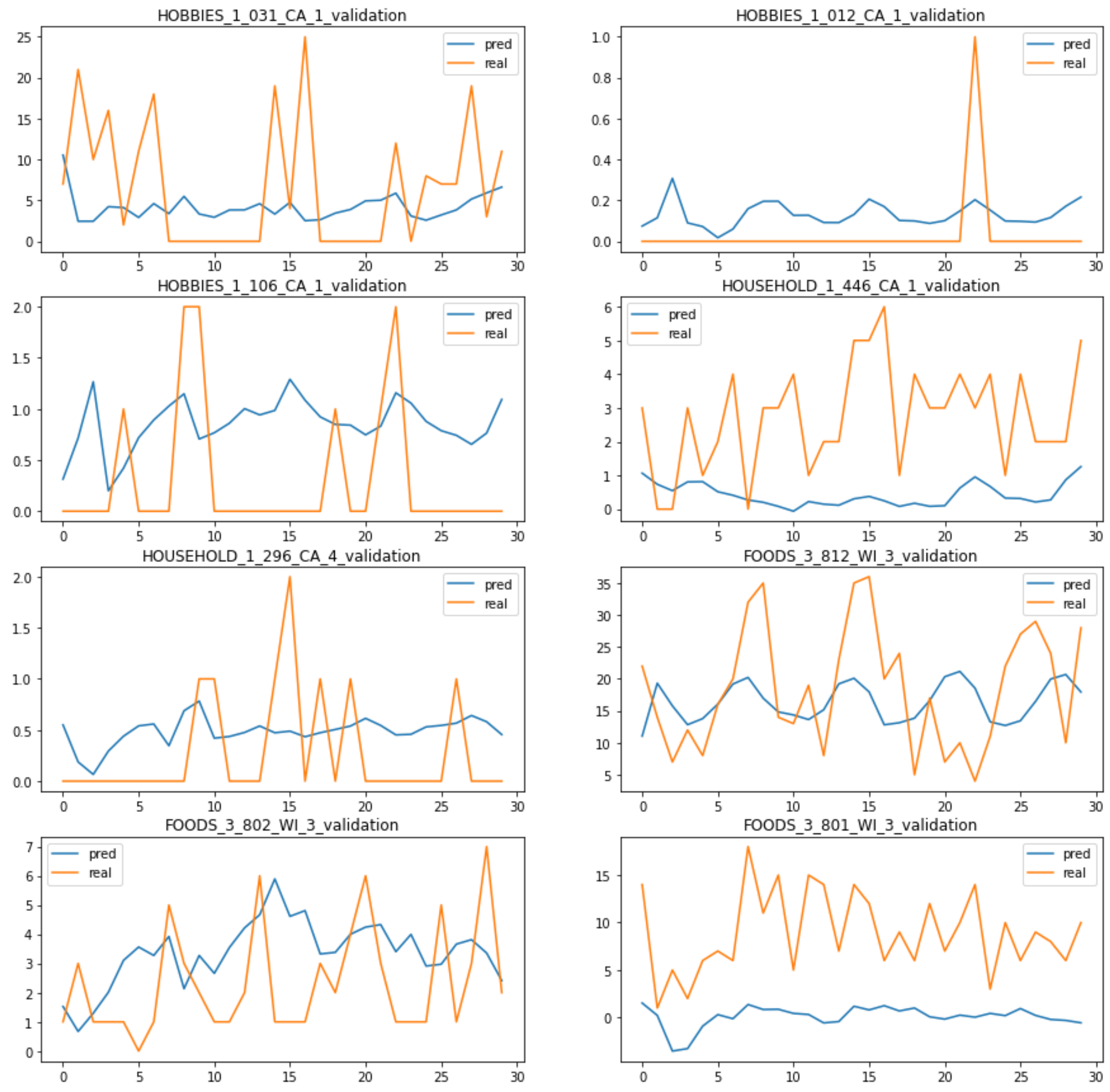






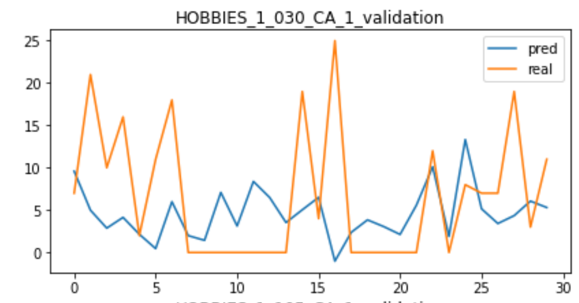
The 1 layer model is quite poor and the same as the 3 layers LSTM model with loss validation loss.

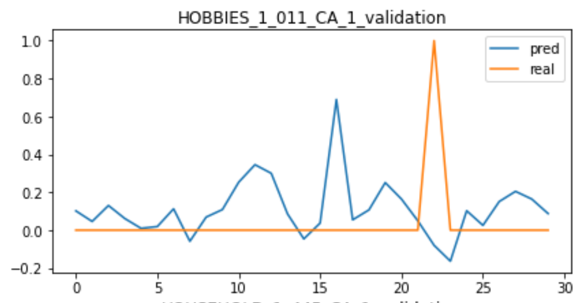
LSTM2 (3 layer):

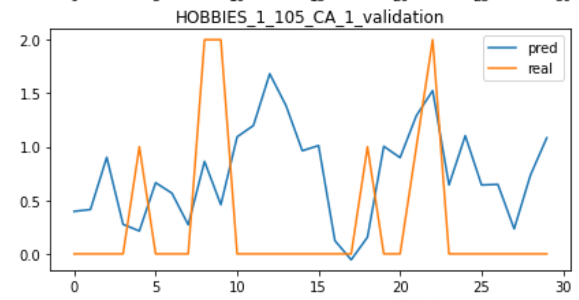


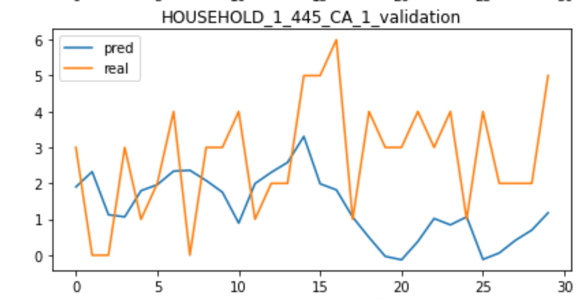
This model is better than RNN and 1 layer LSTM and we can observe that it can slight predict some tendency of the validation data, but still not very good like the last item. The predicted sales is much lower than the real one.

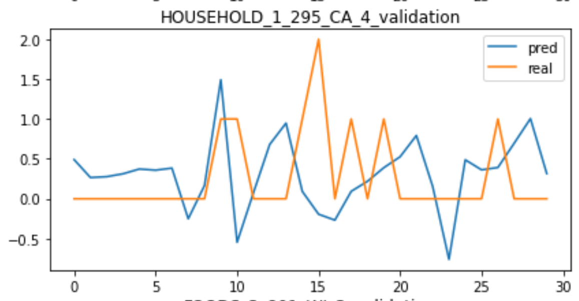
LSTM 3( 3 layers with bidirectional):

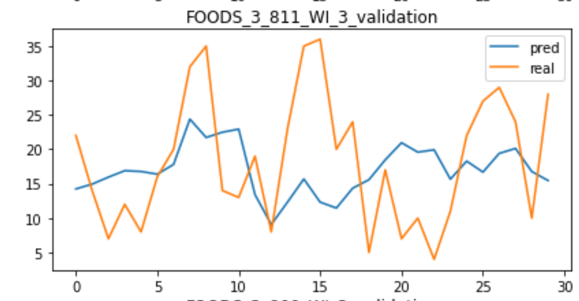


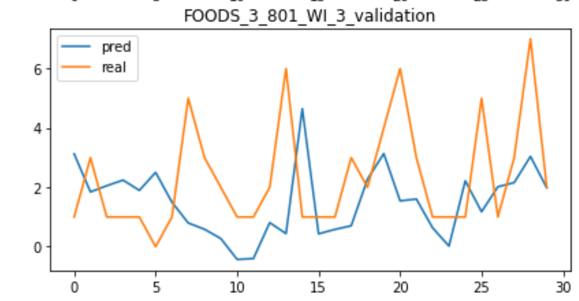


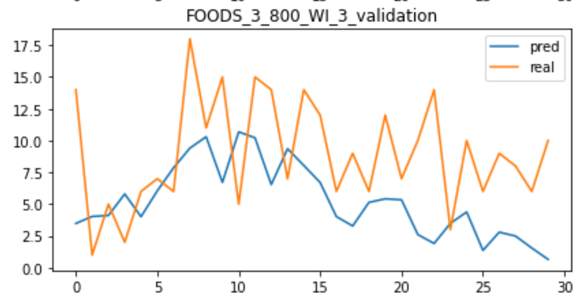






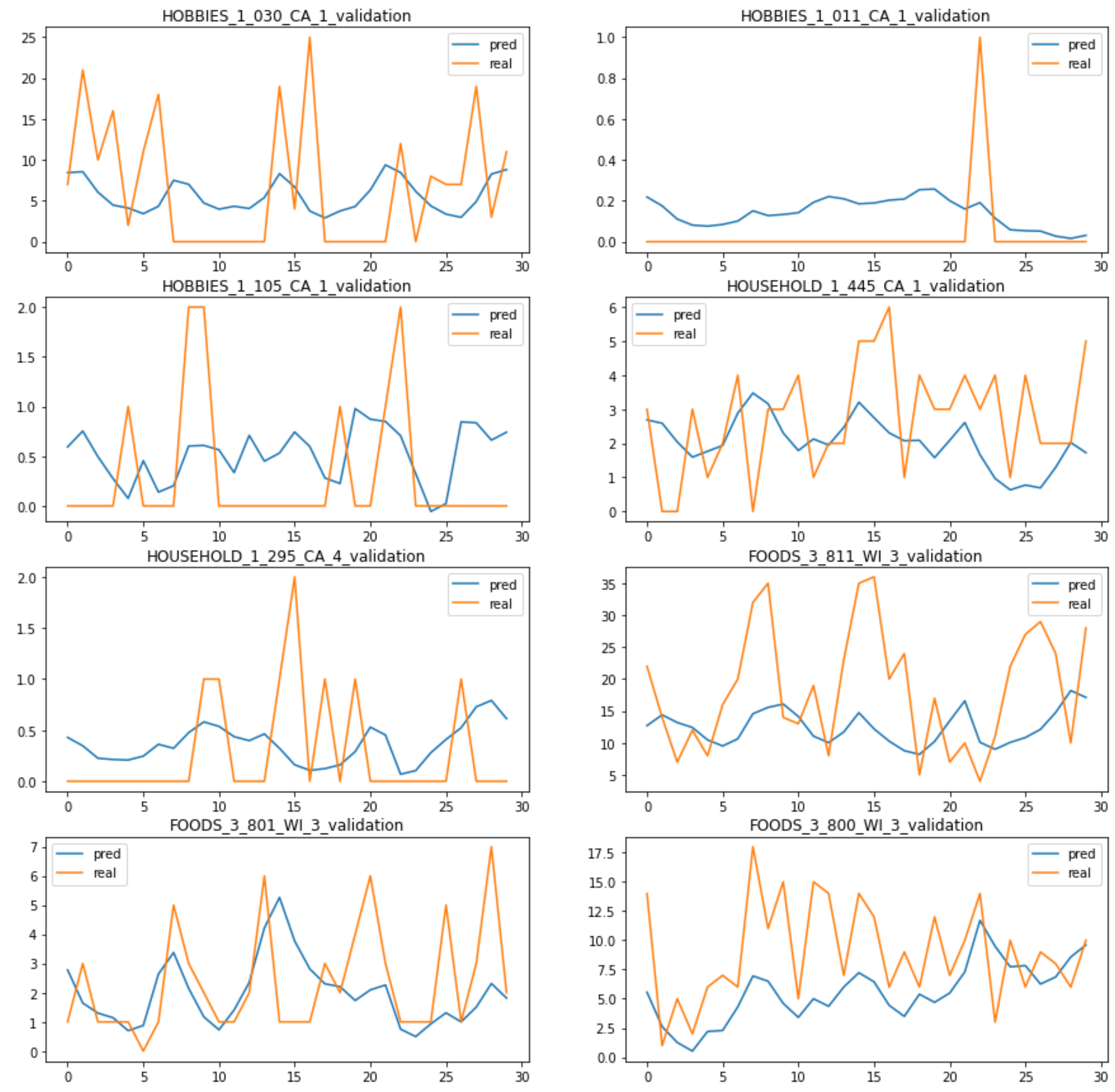






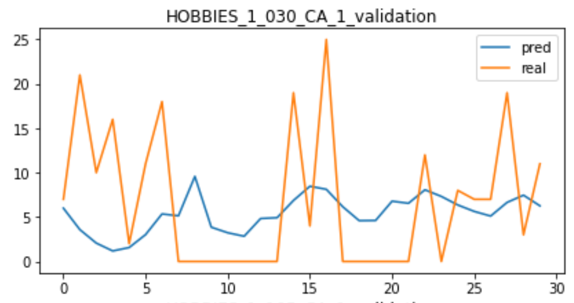
This one has more vibration than the 3 layer LSTM, and some of the prediction is quite match the real sales like the last two item, FOODS_3_801_WI_3 and FOOD_3_800_WI_3.

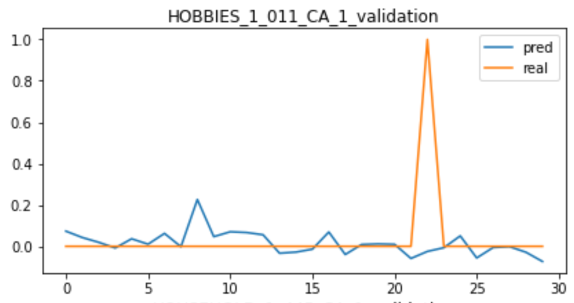
GRU:

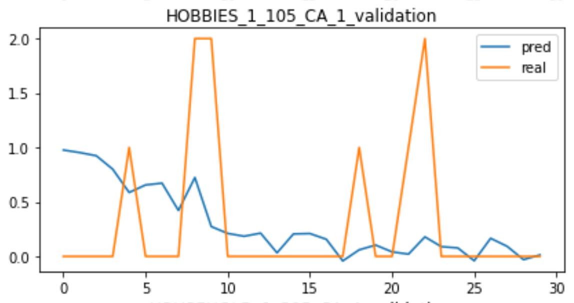


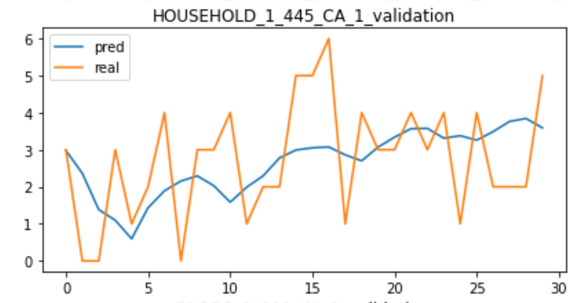
The performance of this one is similar to the bidirectional one, especially on the last two items and the second one, HOBBIES_1_011_CA_1. The predicted tendency is similar to the real one. However, some items are very difficult to predict like the second one, it only had sales 1 within 30 days. However, the model predicts all around 0, the model predicts this item quite good and make sense and minimize the loss.

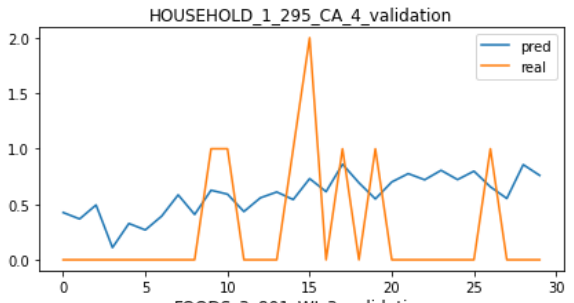
GRU (batch size = 14):

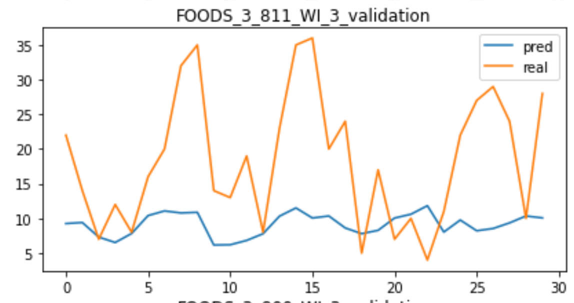


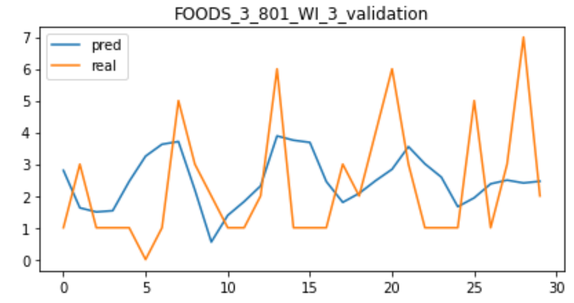


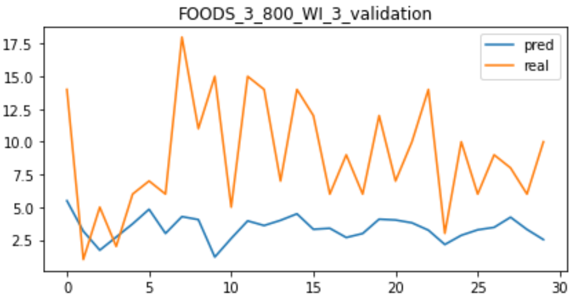












The performance is poor than the batch size = 56, like the last item, it predict the lower sales than the previous one and the third one, the prediction is almost zero but the real sales are not but others predictions are similar. Thus, the batch size would affect the model's performance, more batch size the model can capture the longer pattern from the longer time series as the data load does not load the data with shuffle.

**RMSE of different models on the validation set**:
RNN: 2.3944933185875734
1 layer LSTM: 2.4179607522618194
3 layer LSTM (with the lowest validation loss): 2.384870365114386
3 layer LSTM :2.601654975165163
Bidirectional LSTM: 2.6002844936111074
GRU: 2.4264079810078143

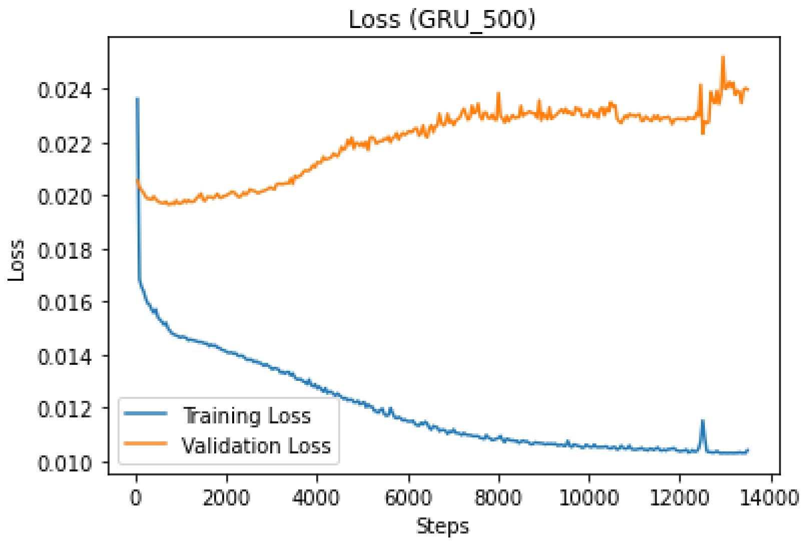
GRU (28 batch size): 2.4074239856789355

Summary on time series method:
We would say the best model is GRU with the less parameters and running time than bidirectional LSTM. The model is only good for some items but not all since some data are too uncertain and without a time sequence. The models has some good result on some items but not all. And even the model has a lowest RMSE loss, which does not imply that the model has a good result and good performance. Just take the RNN and the 3 layer LSTM (with the lowest MSE validation loss) as examples, they have the lower RMSE loss than other but you can see their prediction performance is not that good, they only predict a straight line with the mean since this method can prevent the wrong prediction and minimize the loss. However, this is not what we need. On the GRU and bidirectional model, although their RMSE of all 30490 items are higher than others, but they predict some item quite well. For the training loss and validation loss, the training loss keeps on decreasing but the validation converge much faster than the training loss, however, there are 30490 items, the models may perform better on some items. The model with lowest validation loss just predicted a straight with mean for all items.

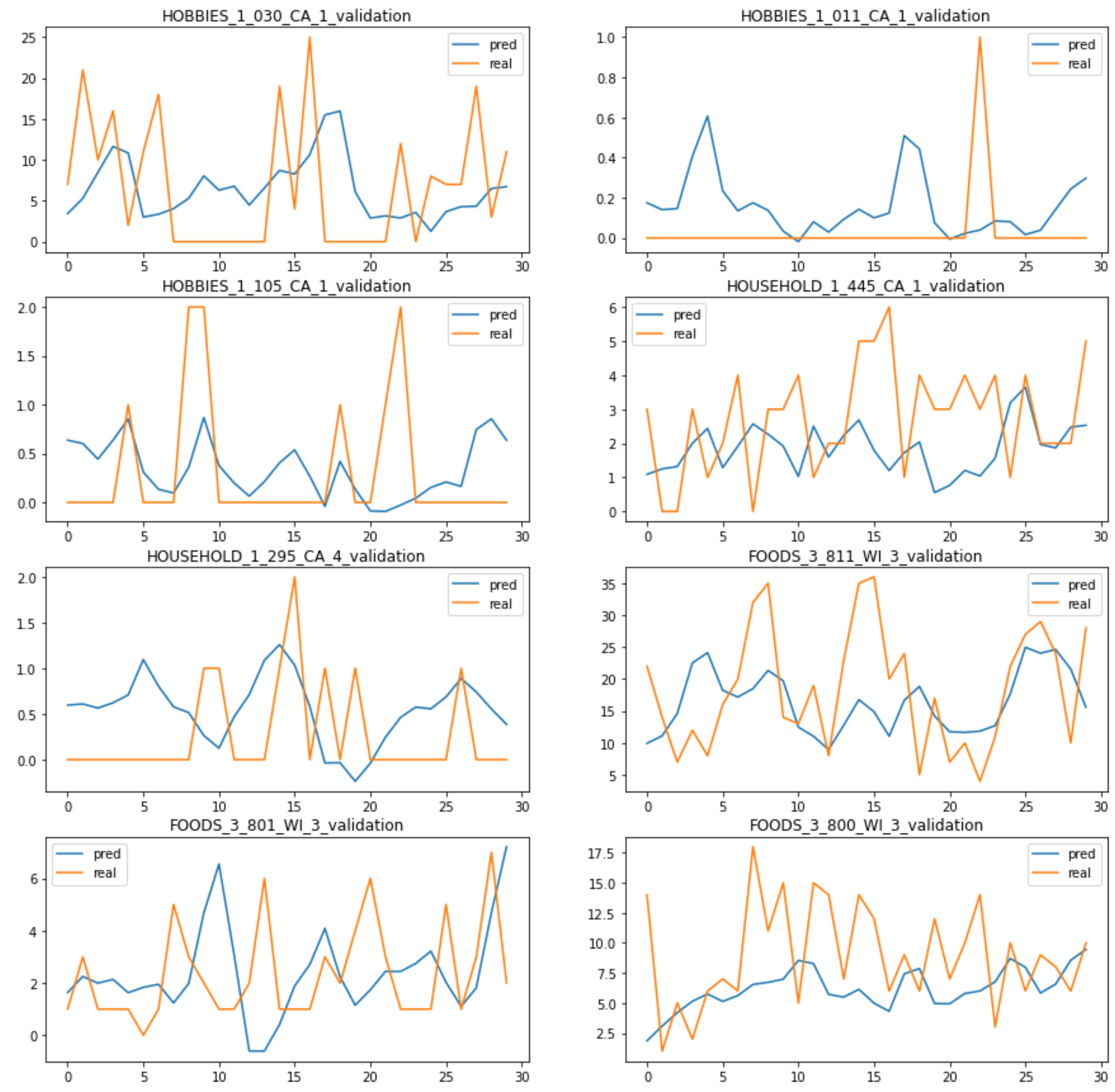
Also, we observe that using data loader randomly pick the data or not would affect the loss. If we randomly pick the data, the loss would not converge too much, even the training loss. Thus, this imply that the loss would be affected by the time series.

We also try some **other two different training datasets**. **The first one is only consider the last 500 days instead of the whole period** and use them to train the GRU model (all hyperparameters, features and labels are the same as before).

```
The shape of features of train is: torch.Size([500, 28, 30498])
The shape of labels of train is: torch.Size([500, 30490])
```

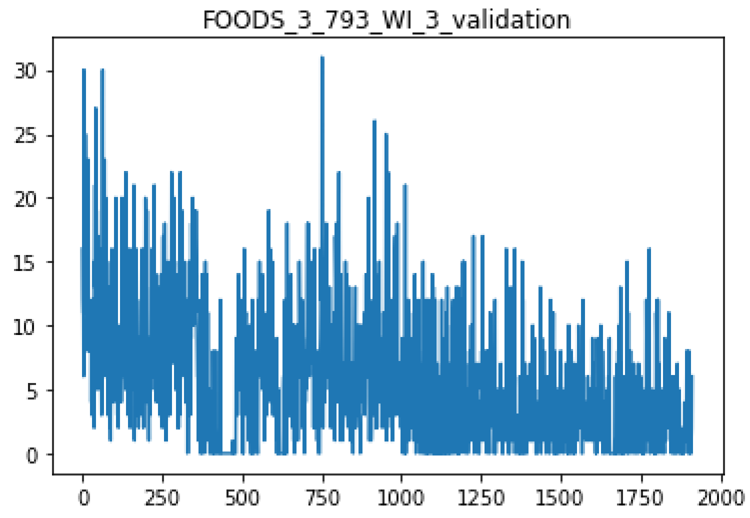


Since the data set is much smaller than before so I can train the model with more epochs to see the prediction. However, the problem of overfitting in here is much serious than the above models. After 4000 steps, the overfitting occurs and the validation loss keeps on increasing. Its RMSE is 5.173446517778725 which is much higher than all the above models. The following is its prediction:



You can see its prediction is not good as the previous data set with same GRU model. Especially the second one, its do not predict all closer to zero but predict wrongly and most of predicted sales are lower than the real labels. Then, we believe that there is some sequence with a long time sequence, for example, one year has one cycle. Thus, 500 days are not enough to let the machine know the pattern.

Our another data set is only train the data of one items and predict its sales. We randomly pick one item from 30490 items.

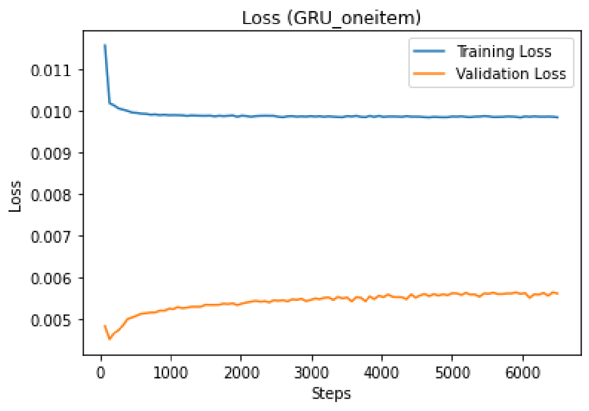


This is its sales from 2011-01-29 to 2016-06-19. And its sales price is the same throughout the whole period.

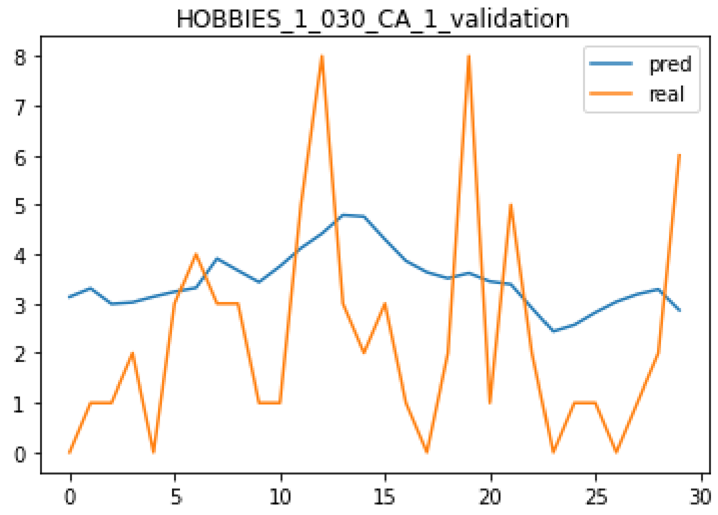
Since the sales in the previous 1000 days are not similar to the sales after day 1000. We only consider the sales of the last 700 day. Also, we still consider the weekend or not, 4 event type on each day, snap in 3 states with one-hot encoding. Training this data set with GRU model.

```
The model architecture:

 M5_predictor3(
  (linear): Linear(in_features=128, out_features=1, bias=True)
  (rnn1): GRU(9, 128, num_layers=2, batch_first=True, dropout=0.3)
)

The model has 152,577 trainable parameters
```

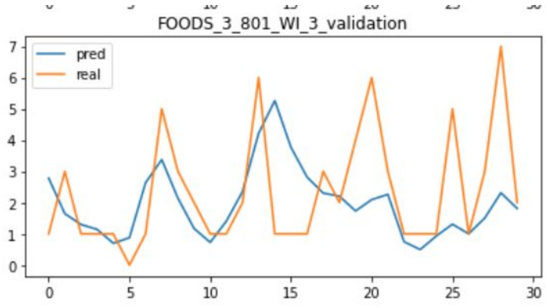
Since we do not consider all 30490 items, the parameters are much lesser than before.
The loss:

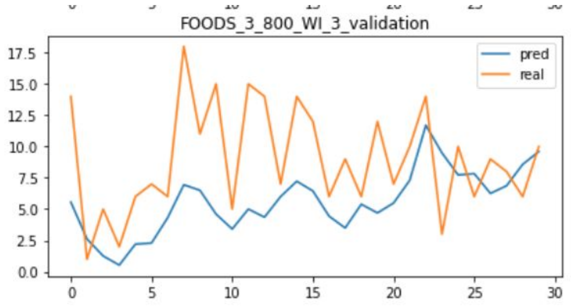


The validation loss and train loss do not decrease too much. And the RMSE is 2.316734156463791 which is slightly lower than other datasets.



The prediction is not that good as the GRU model that consider all 30490 items, as come items in the previous dataset are predicted very well. The following predictions are two good prediction of the previous dataset with the GRU model.





Thus, we believe that the sales between different items have some relationship. Some items may have similar tendency of sales. More item can let the model predicts the items more precisely.

However, we can see that all models and different dataset, their model predicts a lower sales than the real one. Since many items have many days with 0 sales, so that the model predicts lower sales can minimize the loss. We also think that time sequence is not the best way to predict the sales because sometimes how many items sold today would not affect the sales of tomorrow. The sales of each items in each day is independent so tree base model would be another suitable way to try to deal with this problem. Also, tree base model can consider some feature that the time sequence cannot cover like the item type, which store sold the item, the sell price of the items (for time series model, if consider all items with its sell prices, the input features would be extremely large) etc.

**Part 2(tree-based model)**

The training set consist of all data from 2015-05-19 to 2016-05-19.
The testing(validation) set consist of all data from 2016-05-19 to 2016-06-19.
The submission.csv is to predict all data from 2016-06-19 to 2016-07-19.

We have tried different tree-based model, including decision tree, random forest, XGBoost and LightGBM.

<u>Decision tree</u>

```
params = {"max_depth": [20, 30, 40]}
from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor()

dtr = GridSearchCV(dt, params)
```

```
dtr.best_score_
0.4471290009260098
```

```
dt = dtr.best_estimator_
dt_pred_train = dt.predict(train_X)
dt_pred_test = dt.predict(test_X)

print(mean_squared_error(train_y, dt_pred_train))
print(mean_squared_error(test_y, dt_pred_test))

3.457522588977114
6.767866385758176
```

```
dt.get_depth()
20
```

As the dataset is huge, we would better to determine some key tree parameter before training other tree. We have found that the max_depth parameter is very critical and have an important import to the performance of tree-based model. After quit an effort, we have found that max_depth = 20 gives the best result generally for all the tree model, and the intuition comes from this grid search of max_depth in simple decision tree. We have also seen that, the performance of a simple decision tree is already quite good, compared to highly complicated deep LSTM, RNN.

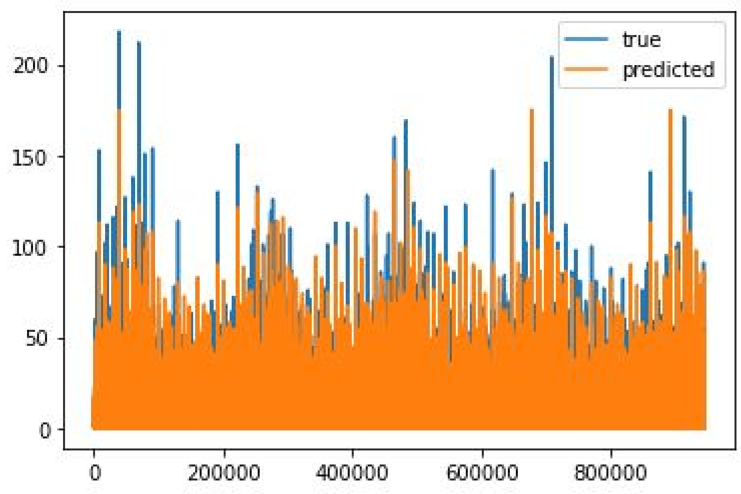
<u>Random Forest</u>

```
print(mean_squared_error(train_y, rf_pred_train))
print(mean_squared_error(test_y, rf_pred_test))
3.047500818538384
5.0893634196554585
```

Not surprisingly, the random forest has a significant improvement on single decision tree, attaining a validation RMSE of ~2.256.

```
rf.feature_importances_
array([0.02775385, 0.02038977, 0.10151449, 0.06810297, 0.03761863,
       0.02461519, 0.0049768 , 0.00329328, 0.        , 0.        ,
       0.00540635, 0.00557357, 0.00795484, 0.19753638, 0.11909089,
       0.37617298])
```

The random forest suggested that the item_id, sell_price, store_id and day are the most important features, which make senses.



XGBoost

```
xgb.fit(train_X, train_y)
print(mean_squared_error(train_y, train_pred_y))
print(mean_squared_error(test_y, test_pred_y))
11.389151697758551
11.785555539603152
```

The performance of XGBoost is disappointing. With double the validation loss and even tripled training loss compared to random forest.

Also, we noted that it takes the longest time to train.

Lightgbm

```
lgb_regressor = lgb.train(bst_params, lgb_train, valid_sets=[lgb_train,
lgb_test], valid_names=["train", "valid"], **fit_params)
```

The best performance of Lightgbm:

```
print(mean_squared_error(train_y, lg_pred_train))
print(mean_squared_error(test_y, lg_pred_test))
```

```
4.079638344945075
5.179346136576904
```

Lightgbm also takes a long time to train. However, it is able to attain a low loss. However, it still doesn’t beat the random forest.

I used the same fit_params through out the training phrase:

```
fit_params = {
    "num_boost_round": 100_000,
    "early_stopping_rounds": 200,
    "verbose_eval": 100,
}
```

I have tried different training params.

The performance of the below bst_params is better, converge faster and is able to attain a slightly lower loss (which is the best loss in the above).

```
bst_params = {
    "boosting_type": "gbdt",
    "metric": "rmse",
    "objective": "regression",
    "n_jobs": -1,
    "seed": 4211,
    "learning_rate": 0.1,
    "bagging_fraction": 0.75,
    "bagging_freq": 10,
    "colsample_bytree": 0.75,
}
```

The performance of the below bst_params is slightly worse, it takes longer time to train(lower learning rate) and stop at a higher loss.

```
bst_params = {
    'boosting_type': 'gbdt',
    'objective': 'tweedie',
    'tweedie_variance_power': 1.1,
    'metric': 'rmse',
    'subsample': 0.5,
    'subsample_freq': 1,
    'learning_rate': 0.03,
    'num_leaves': 2**11-1,
    'min_data_in_leaf': 2**12-1,
    'feature_fraction': 0.5,
    'max_bin': 100,
    'boost_from_average': False,
    'seed': 4211,
    'verbose': -1,
}
```

Validation loss:

```
print(mean_squared_error(train_y, lg_pred_train))
print(mean_squared_error(test_y, lg_pred_test))
```

```
3.987569590546374
5.378605057469887
```

<u>Summary on tree-based model:</u>

They are fast to train but slower to predict (not a problem, it still predict within several minutes). They are also great and are able to achieve low loss.

<u>Possible improvement:</u>

1) Add rolling mean and sd of sales of each item in the past month to the data

If there were extra rams and computing power (as we melted and merged a huge dataset, the dataset is even bigger, we were also not able to handle the huge dataset, which is another reason of why we only use data from 1 year, apart from we want to assume there is no inflation and the market of Walmart remains constant), we might be able to add extra attribute to the tree. One common way is putting rolling mean and rolling SD of the sales in past 1 month of each item, which overcome the factors of inflation and give us a good reference of sales of the item recently. And we will also be able to use all 5 years of data in this way.

2) Train different tree for different categories of data

We see that there are huge gap between the sales across different store or different state. It could be useful that we train different model for each categories of data. It could be each tree for each state, or each stores, or each categories of items. Using ensemble method in this way, the tree models should be able to attain a lower loss as each tree is only trained on a subset of the data and should fit better.

## Compare the time series models and the tree base models:

In general, the tree-based model is better. They converge faster and gives slightly better result. However, the final submission of data could be clumsy as we have to pivot the whole table again.

The implementation of tree-based model is also easier. Scikit-learn, XGBoost and LightGBM provide easy-to-use user interface. However, the data processing phrase in tree-based model should be more careful as the dataset is huge.

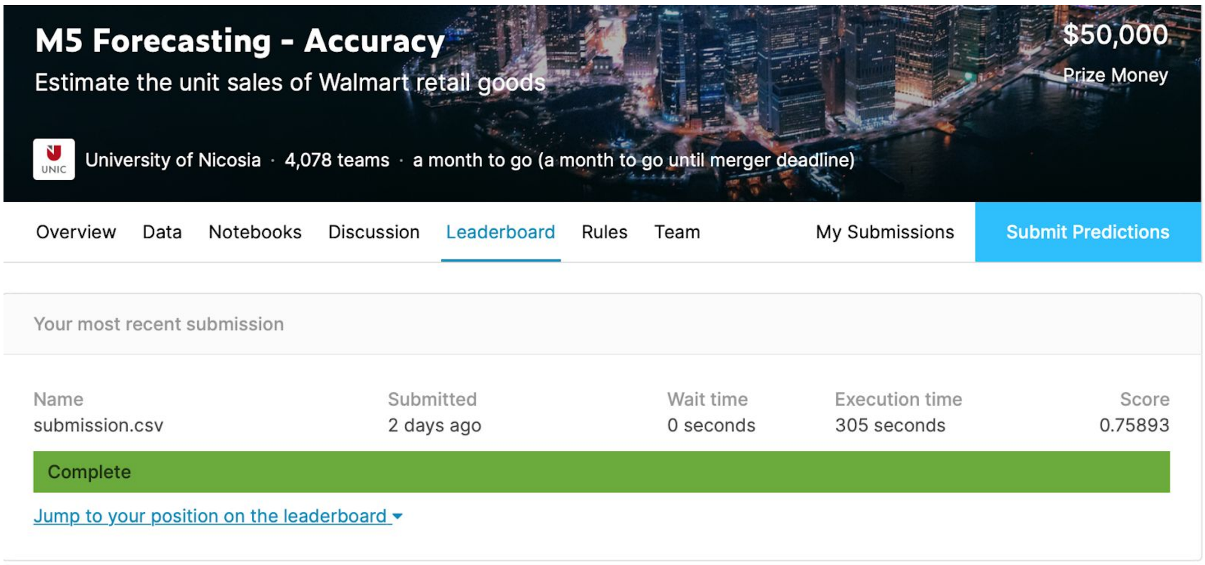
We would say in the first part (time-series models), we focused on searching the best parameters; in the second part (tree-based models), we focused more on hypothesis and explainability of the data and model.

## Division of labor:

**Yuen Zhikun** is in charge of part of the data visualization, part of the design and all of the implementation of all the time sequence part: the data preprocessing of the data with time sequence and other 5 models (RNN, 1 layer LSTM, 3 layers LSTM, 3 layers bidirectional LSTM, 3 layers GRU). Other two training data, last 500 days and the dataset which only consider 1 item (55%).

**Cheng Wai Kit** is incharge of exploratory data analysis, part of the design of experiment of the time sequence part, design and implementation of all of the experiment with tree-based model, submission of the final prediction using tree-based model (45%).

## Screenshot of our competition's result:



M5 Forecasting - Accuracy

Estimate the unit sales of Walmart retail goods

$50,000 Prize Money

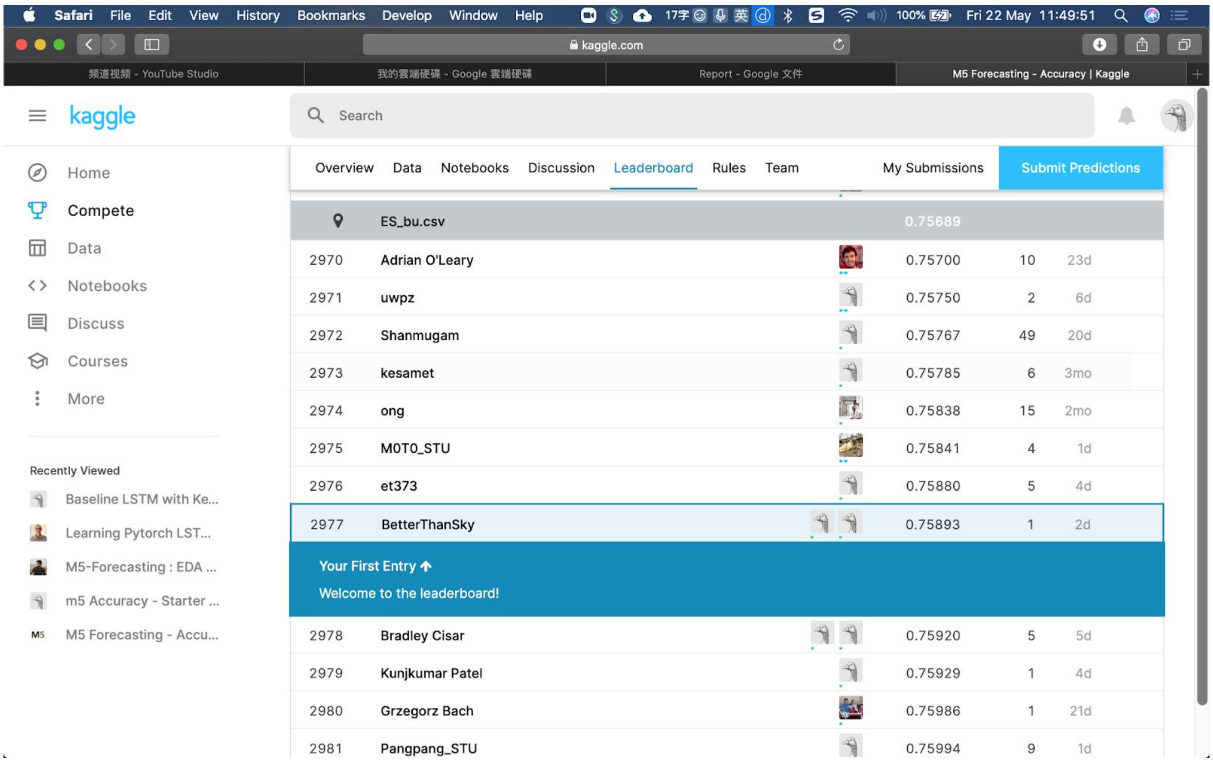
University of Nicosia · 4,078 teams · a month to go (a month to go until merger deadline)

Your most recent submission

| Name | Submitted | Wait time | Execution time | Score |
|---|---|---|---|---|
| submission.csv | 2 days ago | 0 seconds | 305 seconds | 0.75893 |

Complete

Jump to your position on the leaderboard



| | | | | |
|---|---|---|---|---|
| | ES_bu.csv | 0.75689 | | |
| 2970 | Adrian O'Leary | 0.75700 | 10 | 23d |
| 2971 | uwpz | 0.75750 | 2 | 6d |
| 2972 | Shanmugam | 0.75767 | 49 | 20d |
| 2973 | kesamet | 0.75785 | 6 | 3mo |
| 2974 | ong | 0.75838 | 15 | 2mo |
| 2975 | M0T0_STU | 0.75841 | 4 | 1d |
| 2976 | et373 | 0.75880 | 5 | 4d |
| 2977 | BetterThanSky | 0.75893 | 1 | 2d |
| 2978 | Bradley Cisar | 0.75920 | 5 | 5d |
| 2979 | Kunjkumar Patel | 0.75929 | 1 | 4d |
| 2980 | Grzegorz Bach | 0.75986 | 1 | 21d |
| 2981 | Pangpang_STU | 0.75994 | 9 | 1d |

Your First Entry

Welcome to the leaderboard!

## Hyperlink to YouTube video:

https://youtu.be/zAf1OwHDEOA