

Predicting Units of Product Sold (EDA)

Business Understanding

Understanding the demand of products is one of the most difficult problems for all businesses that produce physical goods. Even understanding the demand of a single product marketed to a single consumer base can be extremely challenging work. This challenge is compounded when dealing with the variety of products in the variety of markets that Wal-Mart is involved in. Developing a generalizable model that can be used to understand demand could give Wal-Mart a significant competitive edge over competition. An important factors that allowed Wal-Mart to keep its prices competitive and consolidate so many markets was its efficient supply chain. [Profitworks \(https://profitworks.ca/small-business-sales-and-marketing-resources/blog/marketing-strategy/579-sam-walton-wal-mart-what-is-wal-mart-s-secret-to-success.html\)](https://profitworks.ca/small-business-sales-and-marketing-resources/blog/marketing-strategy/579-sam-walton-wal-mart-what-is-wal-mart-s-secret-to-success.html), a marketing agency, credits one of the largest reasons for Wal-Mart's national success as its extremely efficient supply chain. Being able to accurately forecast demand is imperative to efficiently running a supply chain whose supply can consistently meet demand without having an excessive amount of inventory. Keeping all products available and properly managing inventory can be difficult, and it becomes exponentially more difficult when storage bins are cluttered and/or full. Managing this balance, especially in stores with less space for inventory such as Wal-Mart Neighborhood Markets, can demand high levels of precision.

Data Understanding

The data used in this analysis can be found from the M5 Forecasting - Accuracy competition on Kaggle. It contains a 5 files: calendar.csv, sales_train_validation.csv, sample_submission.csv, sell_prices.csv, and sales_train_evaluation.csv. The calendar.csv file contains the date for every day, as well as the weekday, month, and Wal-Mart week. It also includes information of the days holiday status, and the days when snap is available. The sales_train_validation.csv contains information on the amount of units sold every day for 3019 different products across 10 different Wal-Mart stores in three different states. The sales_train_evaluation data contains the same information as the validation data, along with the correct values for the 28 day forecast. The sample_submission.csv contains an example csv for the format needed to submit the 28 day forecasts. The sell_prices.csv contains the prices at which the items were sold for every Wal-Mart week.

```
In [1]: # imports for notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, MultiLabelBinarizer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
```

Dates Data

```
In [2]: # read in the dates and their recorded unique characteristics
dates = pd.read_csv('C:/Users/TWood/Downloads/m5-forecasting-accuracy/calenda
```

```
In [3]: # take a look at the dates df
dates.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1969 entries, 0 to 1968
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   date                  1969 non-null  datetime64[ns]
1   wm_yr_wk              1969 non-null  int64
2   weekday               1969 non-null  object
3   wday                  1969 non-null  int64
4   month                 1969 non-null  int64
5   year                  1969 non-null  int64
6   d                     1969 non-null  object
7   event_name_1          162 non-null   object
8   event_type_1          162 non-null   object
9   event_name_2          5 non-null     object
10  event_type_2          5 non-null     object
11  snap_CA               1969 non-null  int64
12  snap_TX               1969 non-null  int64
13  snap_WI               1969 non-null  int64
dtypes: datetime64[ns](1), int64(7), object(6)
```

```
In [4]: # most days have no event, replace NaN with "None"
dates.replace(np.NaN, 'None', inplace=True)
```

Multi-Label Binarizer

Some days have multiple events. One hot encoding will not be able to represent columns with multiple events, so the information will need to be in a format that the MultiLabelBinarizer can use. I'll make a new column that contains a list of all events on a given day.

```
In [5]: # remove spaces from all the events names
dates['event_name_1'] = dates['event_name_1'].str.replace(' ', '')
dates['event_name_2'] = dates['event_name_2'].str.replace(' ', '')

In [6]: # create event column that contains a string of both events with a space between
dates['event'] = dates['event_name_1'] + ' ' + dates['event_name_2']

In [7]: # split will turn the string into a list of both events
dates['event'] = dates['event'].str.split()

In [8]: # removes the second element from the list when it is None
dates['event'] = dates['event'].apply(lambda x: [x[0]] if x[1] == 'None' else x)

In [9]: # instantiate the MultiLabelBinarizer and fit it to the event column
mlb = MultiLabelBinarizer()
mlb.fit(dates['event'])
values = pd.DataFrame(mlb.transform(dates['event']), columns=mlb.classes_)

In [10]: # adds the encoded columns to the dates dataframe
dates = pd.concat([dates, values], axis=1)

In [11]: # drops the redundant event columns from dates
dates.drop(columns=['event_name_1', 'event_type_1', 'event_name_2', 'event_type_2'], inplace=True)

In [12]: # instantiate the OneHotEncoder and fit it to weekday and month columns
ohe = OneHotEncoder(sparse=False)
ohe.fit(dates[['weekday', 'month']])
values = ohe.transform(dates[['weekday', 'month']])
values = pd.DataFrame(values, columns=ohe.get_feature_names())

In [13]: # add the one hot encoded columns to the dates dataframe
dates = pd.concat([dates, values], axis=1)

In [14]: # drop the unnecessary columns from the dates dataframe
dates.drop(columns=['wday', 'year', 'month', 'weekday'], inplace=True)
```

Units Sold Data

The data containing units sold every day takes up a large amount of RAM, and becomes even larger when melted into long format. Then the dataframe needs to be merged with the other two dataframes, taking up even more memory. Using the entirety of the data would not fit on most computers with 8GB of RAM. I decided to limit my analysis and modeling only to the CA_1 store to deal with the large amount of memory needed.

```
In [15]: ▶ # read in the data for units sold
val = pd.read_csv('C:/Users/TWood/Downloads/m5-forecasting-accuracy/sales_tra

In [16]: ▶ # select data only from store CA_1
CA1 = val[val['store_id'] == 'CA_1']

In [17]: ▶ # reducing the unnecessary columns to make the melt faster
CA1.drop(columns=['item_id', 'dept_id', 'cat_id', 'store_id', 'state_id'], in

C:\Users\TWood\anaconda3\envs\learn-env\lib\site-packages\pandas\core\fram
e.py:4163: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
    return super().drop(

In [18]: ▶ # make an iterable array of all the unique ids
items = CA1['id'].unique()

In [19]: ▶ # convert the dataframe into long format
CA1_ts = CA1.melt(id_vars=['id'], var_name='d', value_name='sales')
```

Prices Data

```
In [20]: ▶ # read in the data on the prices for the products
prices = pd.read_csv('C:/Users/TWood/Downloads/m5-forecasting-accuracy/sell_p

In [21]: ▶ # make a new id column that matches the format of the id column in CA1_ts
prices['id'] = prices['item_id'] + '_' + prices['store_id'] + '_validation'

In [22]: ▶ # 30490 total items
len(val)

Out[22]: 30490

In [23]: ▶ # the sell prices are present for all 30490 items for the final week, matches
(prices['wm_yr_wk'] == 11621).sum()

Out[23]: 30490
```

```
In [24]: # create a dataframe with the information from all dataframes
CA1_price = CA1_ts.merge(dates.drop(columns=['snap_TX', 'snap_WI']), on='d').
```

Exploratory Data Analysis

```
In [25]: # Looking at head of df
CA1_price.head()
```

Out[25]:

		id	d	sales	date	wm_yr_wk	snap_CA	ChanukahEnd	Ch
0	HOBBIES_1_001_CA_1_validation	d_1		0	2011-01-29	11101	0	0	
1	HOBBIES_1_002_CA_1_validation	d_1		0	2011-01-29	11101	0	0	
2	HOBBIES_1_003_CA_1_validation	d_1		0	2011-01-29	11101	0	0	
3	HOBBIES_1_004_CA_1_validation	d_1		0	2011-01-29	11101	0	0	
4	HOBBIES_1_005_CA_1_validation	d_1		0	2011-01-29	11101	0	0	

5 rows × 57 columns

```
In [26]: # summary statistics of the numerical variables
CA1_price.describe()
```

Out[26]:

	sales	wm_yr_wk	snap_CA	ChanukahEnd	Christmas	CincoDeMayo
count	5.832737e+06	5.832737e+06	5.832737e+06	5.832737e+06	5.832737e+06	5.832737e+06
mean	1.319829e+00	1.133919e+04	3.293257e-01	2.613696e-03	2.613696e-03	2.613696e-03
std	4.058652e+00	1.503742e+02	4.699684e-01	5.105747e-02	5.105747e-02	5.105747e-02
min	0.000000e+00	1.110100e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	1.121700e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	0.000000e+00	1.133300e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	1.000000e+00	1.144800e+04	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
max	6.480000e+02	1.161300e+04	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

8 rows × 54 columns

```
In [27]: # summary statistics for non numerical values
CA1_price.describe(include='object')
```

Out[27]:

	id	d
count	5832737	5832737
unique	3049	1913
top	HOBBIES_2_061_CA_1_validation	d_979
freq	1913	3049

Missing Prices

There are several missing prices in the dataframe, which could cause a big problem when modeling. However, upon further inspection, there were always zero sales of a product when its price was missing. This suggests that the price was not entered for these weeks because it was not on the shelves for those weeks. The weeks where the price is missing can easily be dropped when fitting the model as they will not be representative of the actual demand of the product.

```
In [28]: # there are several nulls after left joining with prices
CA1_price['sell_price'].isna().sum()
```

Out[28]: 1129842

```
In [29]: # there are never any sales of a product when its price is missing for the week
((CA1_price['sales'] == 0) & (CA1_price['sell_price'].isna())).sum()
```

Out[29]: 1129842

```
In [30]: CA1_price.groupby('date')['sales'].sum()
```

Out[30]:

date	
2011-01-29	4337
2011-01-30	4155
2011-01-31	2816
2011-02-01	3051
2011-02-02	2630
...	
2016-04-20	3722
2016-04-21	3709
2016-04-22	4387
2016-04-23	5577
2016-04-24	6113

Name: sales, Length: 1913, dtype: int64

Data Challenges

One of the most challenging parts of finding a model that can accurately predict the number of units sold is the extremely noisy nature of the data. The total number of items sold every day varies hugely. The standard deviation of total items sold is 992 items per day, where the mean number of items sold is 4024 items per day. Looking at the Total Items Sold per Day graph below clearly illustrates this. This variation is after aggregating together all the items sold in the entire store. This high variance becomes an even greater issue when looking at the number of units sold for a singular item. For example, the item 43 from the Hobbies 1 department sells, on average, 3.29 units per day. However, the standard deviation is even larger than the mean number of units sold, at 5.13 units a day. Looking at the Total Units Sold per Day graph below illustrates this huge variance. It also depicts another issue with predicting units sold; a huge proportion of the items sell 0 units a majority of the days. In fact, 55% of the entries show zero units sold for the day. This high daily volatility can make it extremely challenging to give reasonable numbers at a daily frequency. It is very likely that large proportion of this noise is irreducible, so expectations for the model will need to take this into account.

```
In [31]: CA1_price.groupby('date')['sales'].sum().mean()
```

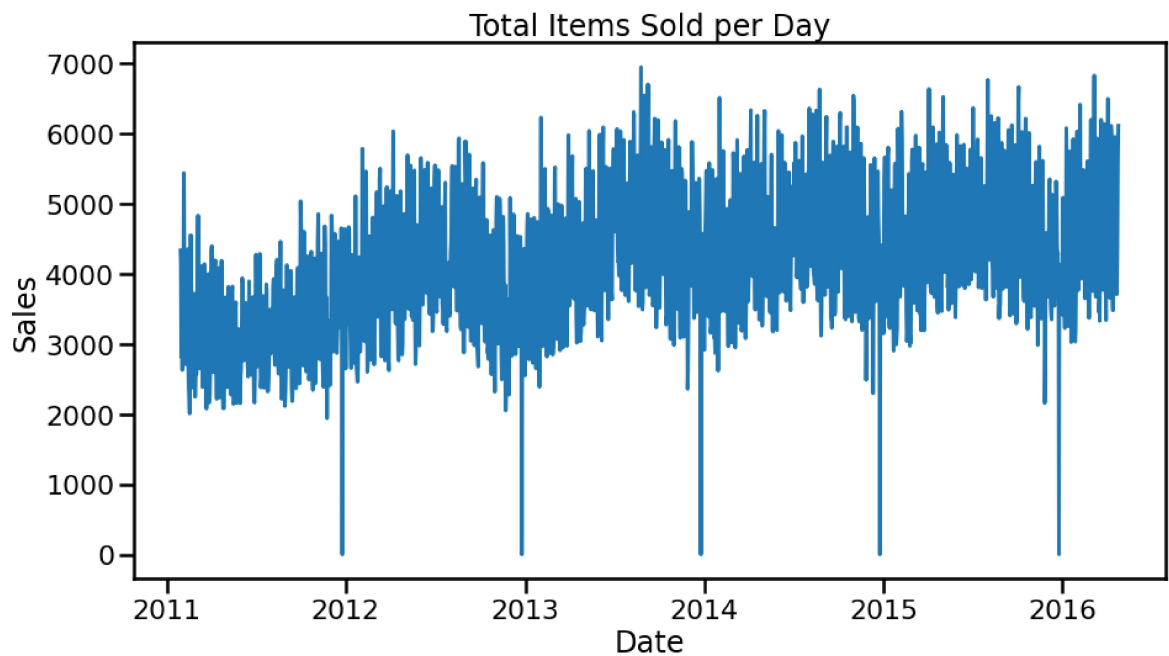
```
Out[31]: 4024.1589127025613
```

```
In [32]: CA1_price.groupby('date')['sales'].sum().std()
```

```
Out[32]: 991.8472057455526
```

```
In [33]: sns.set_context('poster')
```

```
In [34]: fig, ax = plt.subplots(figsize=(15,8))
sns.lineplot(data=CA1_price.groupby('date')['sales'].sum(), ax=ax)
ax.set_title('Total Items Sold per Day')
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.savefig('./images/total_items_sold.jpeg', facecolor='white');
```



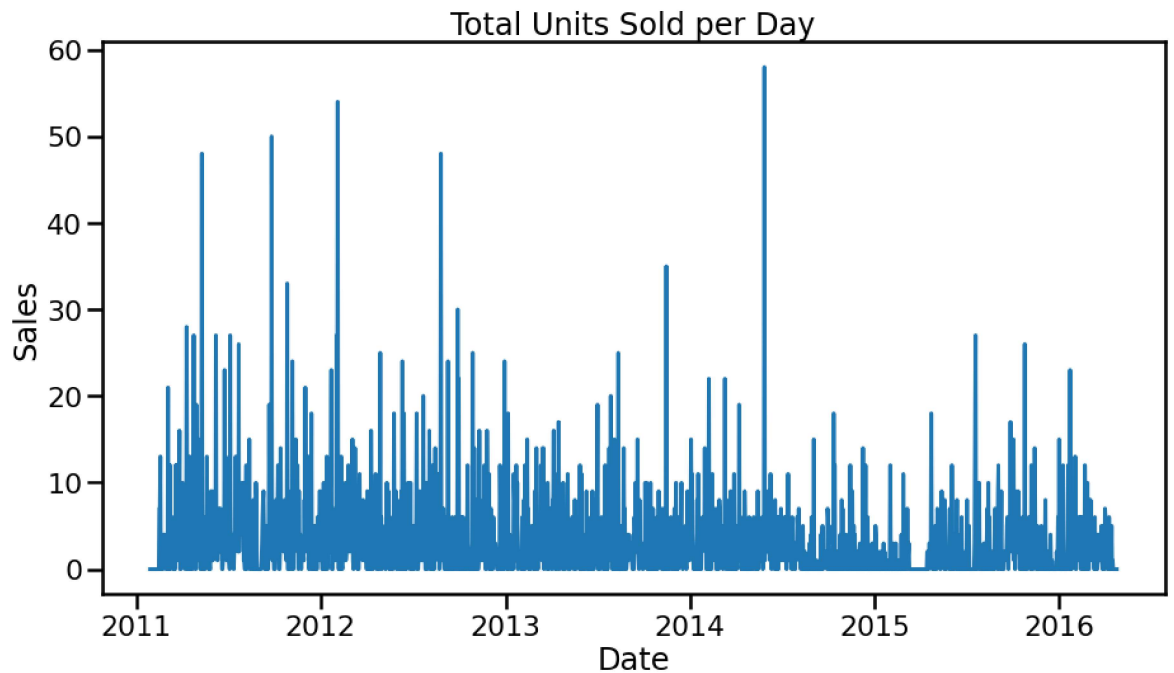
```
In [35]: CA1_price[CA1_price['id'] == items[42]]['sales'].std()
```

Out[35]: 5.133077157392545

```
In [36]: CA1_price[CA1_price['id'] == items[42]]['sales'].mean()
```

Out[36]: 3.288029273392577


```
In [37]: fig, ax = plt.subplots(figsize=(15,8))
sns.lineplot(data=(CA1_price[CA1_price['id'] == items[42]].set_index('date')[
ax.set_title('Total Units Sold per Day')
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.savefig('./images/total_units_sold', facecolor='white');
```



```
In [38]: # over 2 million observations have no sales
(CA1_price.dropna()['sales'] == 0).sum()
```

Out[38]: 2600038

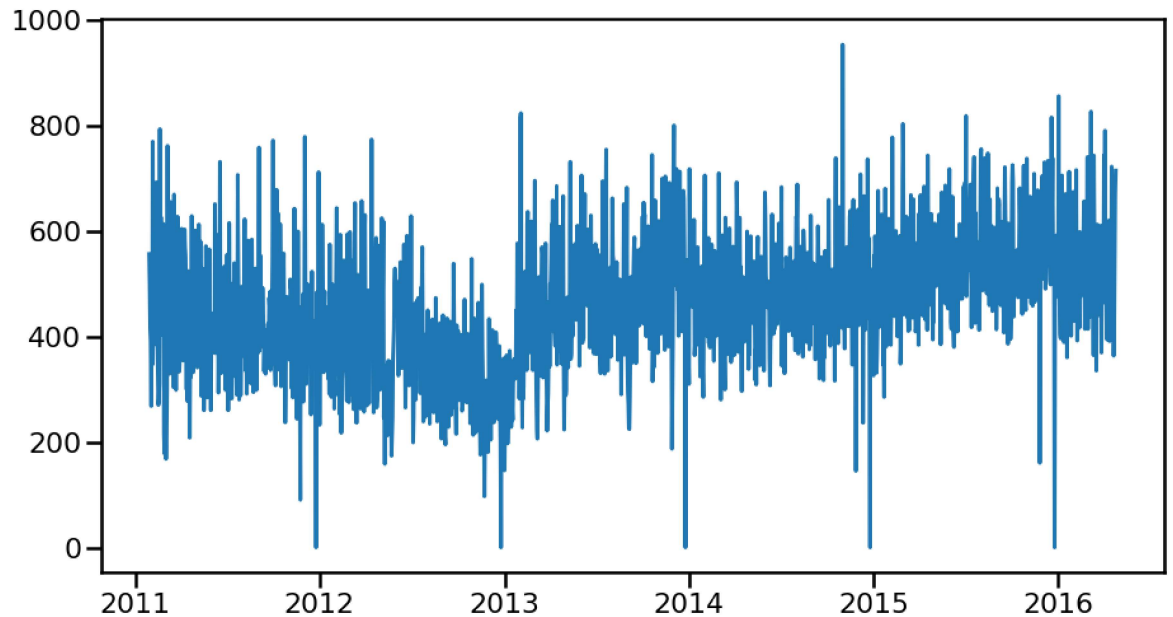
```
In [39]: ▶ # this is over half of the observations  
(CA1_price.dropna()['sales'] == 0).sum() / len(CA1_price.dropna())
```

Out[39]: 0.552859036827316

Sales by Department

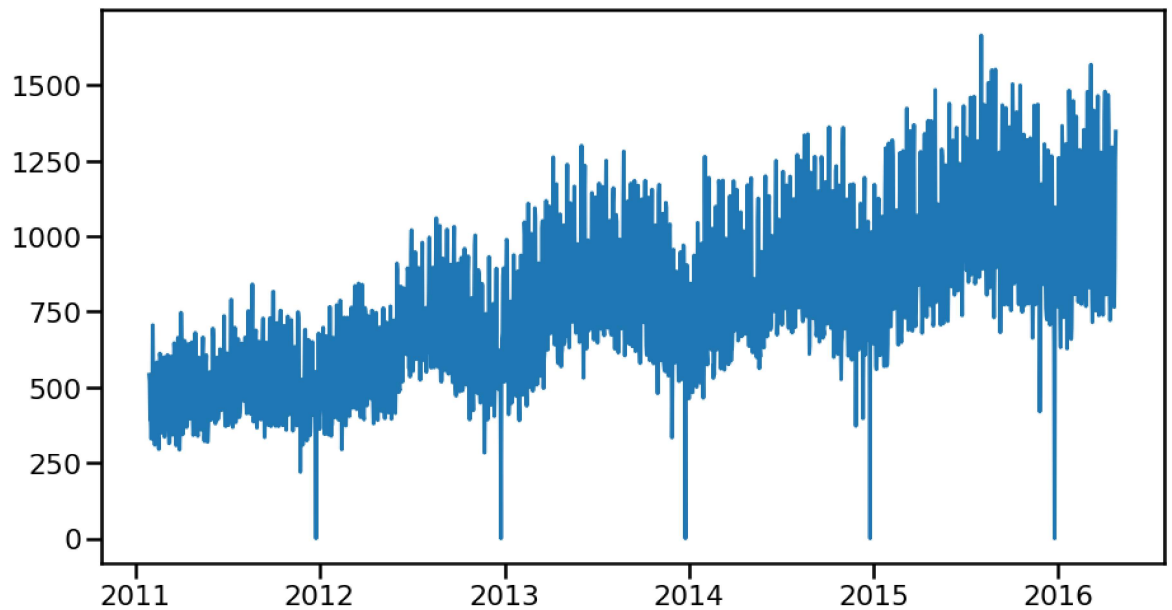
```
In [40]: ▶ # Sales in HOBBIES departments  
fig, ax = plt.subplots(figsize=(15,8))  
ax.plot(CA1_price[CA1_price['id'].str.contains('HOB')].groupby('date')['sales'])
```

Out[40]: [<matplotlib.lines.Line2D at 0x2ab61486460>]



```
In [41]: # sales in HOUSE departments
fig, ax = plt.subplots(figsize=(15,8))
ax.plot(CA1_price[CA1_price['id'].str.contains('HOU')].groupby('date')['sales
```

Out[41]: [



```
In [42]: # Sales in FOODS departments
fig, ax = plt.subplots(figsize=(15,8))
ax.plot(CA1_price[CA1_price['id'].str.contains('FOO')].groupby('date')['sales
```

Out[42]: [



EDA Conclusion

We see that there is a lot of volatility in this data, and it is uncertain that we have the features that will be able to make sense of the high variance in daily sales. Furthermore, there is an extremely

high proportion of zeroes in the target variable, which will make this data particularly challenging to model. Working with a dataset this large puts a large strain on my computers resources, so I'll use a further subset of the data to select the algorithm and parameters for my model. To see my modelling process, go onto the Modeling_Notebook.