Store Sales Prediction

Predicting daily sales up to six weeks in advance across 3,000 locations in Europe

Final Report

Paul Marten
July 2021

1. Introduction

1.1 Context

Rossmann operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

1.2 Problem

How can we best predict the Rossmann stores' daily sales up to six weeks in advance, through means of using data about each location and past sales to formulate a regression model that will minimize percentage error?

1.3 Data

The data is obtained from the Kaggle competition titled "Rossmann Store Sales." The data can be found here: https://www.kaggle.com/c/rossmann-store-sales/data

- **train.csv** - historical data including sales.
- **store.csv** - supplemental information about the stores.

2. Data Wrangling

2.1 Loading

Being as that both files were already supplied and in csv (comma separated values) format, they were both read into dataframes. The train.csv file had a date column, so I read this file in with the index being the parsed dates, and named the dataframe 'train'. The store.csv file was read in using a standard index, and was named 'store' accordingly.

2.2 Cleaning

- Checking out the initial raw dataframes, ‘train’ on top and ‘store’ on the bottom:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1017209 entries, 2015-07-31 to 2013-01-01
Data columns (total 8 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   Store          1017209 non-null  int64
 1   DayOfWeek      1017209 non-null  int64
 2   Sales          1017209 non-null  int64
 3   Customers      1017209 non-null  int64
 4   Open           1017209 non-null  int64
 5   Promo          1017209 non-null  int64
 6   StateHoliday   1017209 non-null  object
 7   SchoolHoliday  1017209 non-null  int64
dtypes: int64(7), object(1)
memory usage: 69.8+ MB
----------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1112 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

2.2.1 ‘train’

Features:

- **Store**: The location (unique id for each store).
- **DayOfWeek**: Starting on Monday as 1, and ending on Sunday as 7.
- **Sales**: The turnover, or gross revenue, on a given day (target variable).
- **Customers**: The amount of customers on a given day.
- **Open**: Indicator of whether the store was open or closed (0 = closed, 1 = open).

    - **Promo**: Indicates if the store was running a promotion that day (0 = no, 1 = yes).
    - **StateHoliday**: Indicates if it was a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends (a = public holiday, b = Easter holiday, c = Christmas, 0 = None).
    - **SchoolHoliday**: Indicates if the (Store, Date) was affected by the closure of public schools.

- There were no missing values in any feature, which is always great to see. This dataframe's version of missing data is wherever there are 0 sales. Days with no sales will be ignored in model evaluation, given the nature of the root mean squared percentage error metric being used, but that will be explained further in the modeling section of this report.
- Doing some investigation, about 17% of the records accounted for days when a store was closed and sales were 0, and about 0.005% of records indicated when a store was open but had no sales. These rows were dropped from the data.
- For further analysis later in the notebook, I used the datetime nature of the index to create new features by extracting the year, month, day, and week of year.
- The final shape of 'train' was (844338, 12).

2.2.2 'store'

Features:

    - **Store**: The location (unique id for each store).
    - **StoreType**: Differentiates between 4 different store models: a, b, c, d.
    - **Assortment**: Describes an assortment level: a = basic, b = extra, c = extended.
    - **CompetitionDistance**: Distance in meters to the nearest competitor store.
    - **CompetitionOpenSince[Month/Year]**: Gives the approximate year and month of the time the nearest competitor was opened.
    - **Promo2**: A continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating.
    - **Promo2Since[Year/Week]**: Describes the year and calendar week when the store started participating in Promo2.
    - **PromoInterval**: Describes the consecutive intervals Promo2 is started, naming the months the promotion is started. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store.

- Checking for missing values, there were quite a bit, 3 rows having almost half of their entries missing. After further investigation, it turned out to be very clear why.

```
# Check missing values

store.isnull().sum()
```

```
Store                        0
StoreType                    0
Assortment                   0
CompetitionDistance          3
CompetitionOpenSinceMonth  354
CompetitionOpenSinceYear   354
Promo2                       0
Promo2SinceWeek            544
Promo2SinceYear            544
PromoInterval              544
dtype: int64
```

- For the 3 rows missing value for **CompetitionDistance**, there was really no way to infer the values from the data, and being as that it was only 3 of 1115 rows, I imputed these missing values with the mean of the column.
- For the two features **CompetitionOpenSinceMonth** & **CompetitionOpenSinceYear**, there were a significant amount amount of NaN values, there was one thought that came to mind to impute these values. I could possibly observe the sales for the stores that had missing values for these, and see where the sales had some variance that could indicate new competition. The issue with this is that I had not yet explored the data well enough to know if this was even true, so to save the headache and possible error, I just imputed these with 0s so as to not lose a nice chunk of the data dropping these rows.
- Looking at the next three features that were missing a lot of values, **Promo2SinceWeek**, **Promo2SinceYear**, & **PromoInterval**, I was able to see that these were only missing values for records where **Promo2** was equal to 0. This made total sense, and with a quick check, I concluded that this was the case and also imputed these NaNs with 0s.

2.2.3 'train_store'

- Now that both dataframes were cleaned up, it was time to merge them into one. I used an inner join, and joined the two on the 'Store' column. The resulting dataframe had 21 features and 844,338 records.

```
# Join train and store on the 'Store' column, using an inner merge

train_store = pd.merge(train, store, on='Store', how='inner', sort=True)
print(train_store.shape)
train_store.head(10)
```

```
(844338, 21)
```

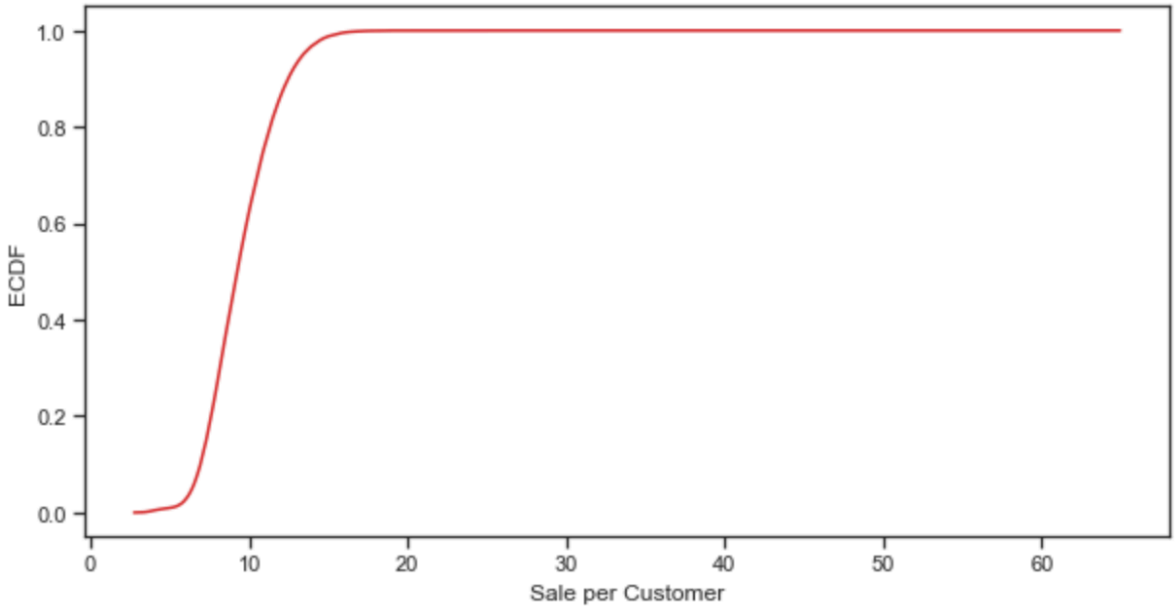
3. Exploratory Data Analysis

- The first thing I did in the EDA portion of the notebook was create a new feature, **SalePerCustomer** in order to aid in the exploration of data. Calling the describe() method on the new feature, I was able to see that the gross revenue per customer per day

```
count    844338.000000
mean          9.493641
std           2.197448
min           2.749075
25%           7.895571
50%           9.250000
75%          10.899729
max          64.957854
Name: SalePerCustomer, dtype: float64
```

was about $9.50. The upper limit was approximately $65, with the lowest being about $2.75.

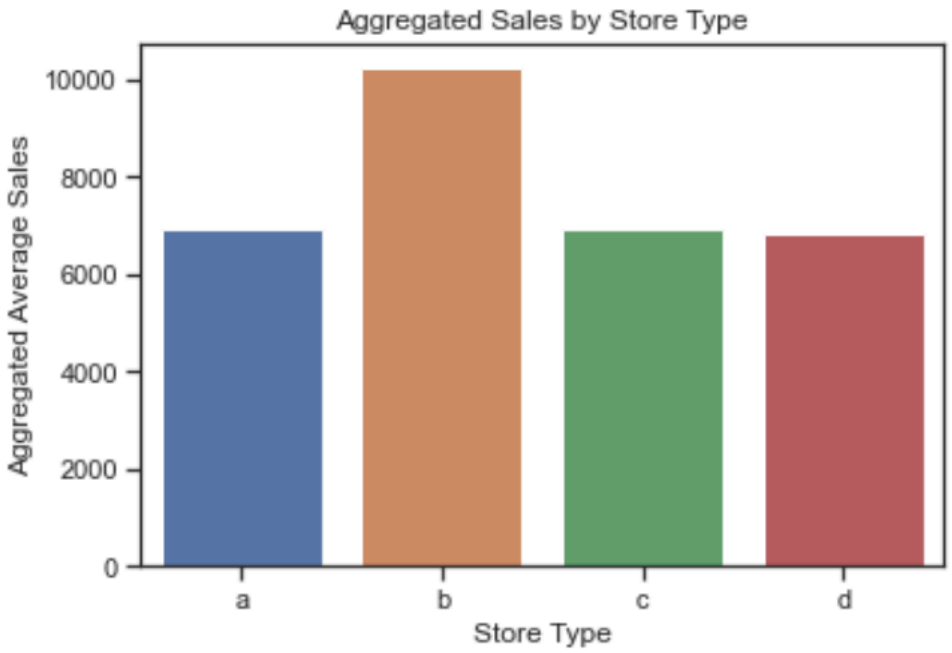
- Following this, I plotted the empirical cumulative distribution function (ECDF) of the the new variable, **SalePerCustomer**. The purpose of this was to have a visual method for understanding the distribution of **SalePerCustomer**. Its value at any specified value of the measured variable is the fraction of observations of the measured variable that are less than or equal to the specified value.

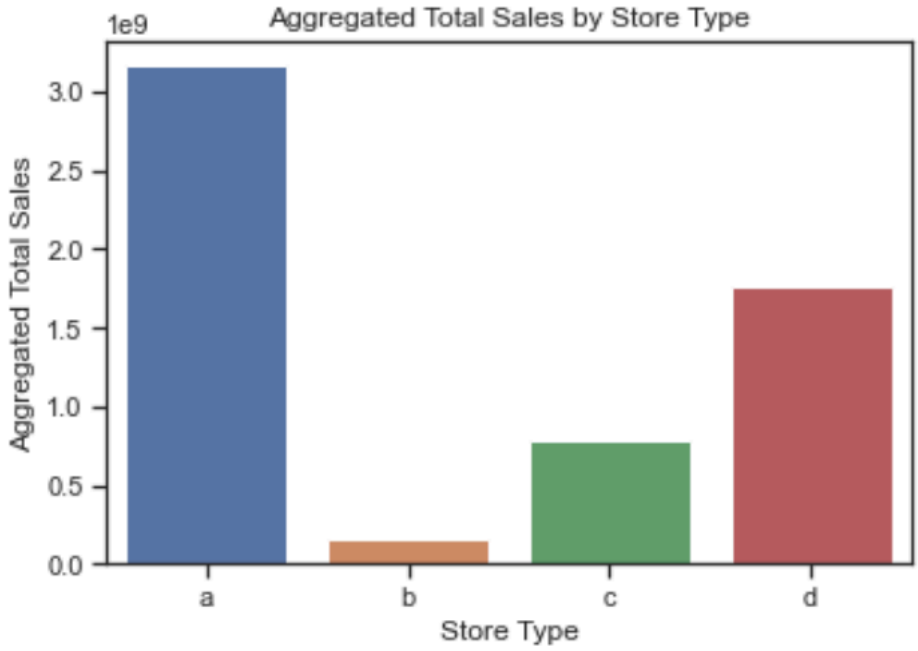


So although the maximum sale per customer was $65, it is very rare for a customer to spend that much, indicated by the fact that almost 100% of customers spend less than that. As a matter of fact, almost 100% of customers spend just $15 or less. This is indicated by the flattening of the curve being around the point (15, 1.0).

3.1 Store Type Analysis

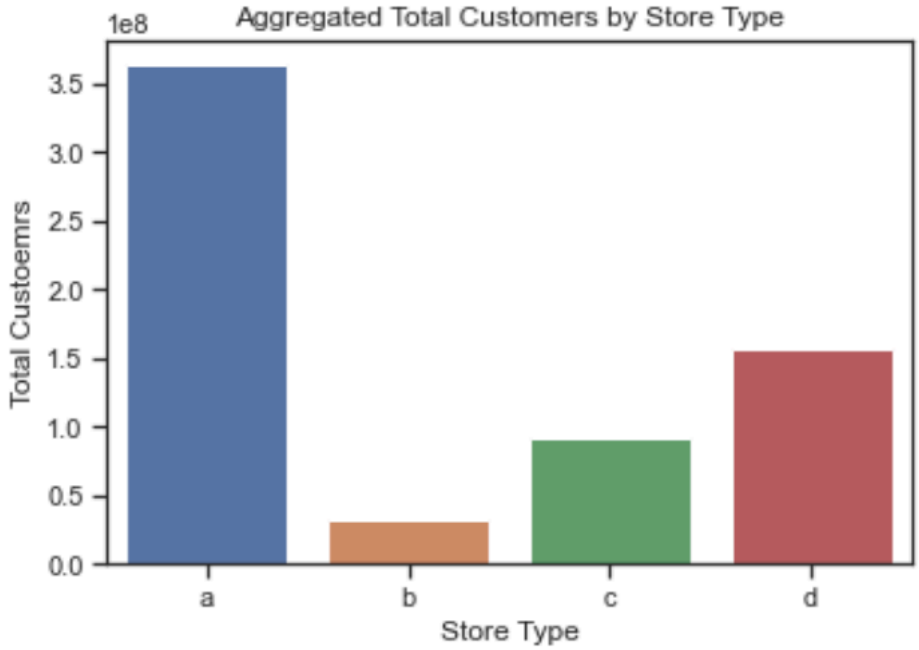
- Being as that there are about 3,000 stores, aggregated on each store would be quite messy. Luckily, **StoreType** only has four distinct values, a-d, and groups every location into one of these categories. I did a few different types of aggregations on these types, which resulted in the following bar charts:





- Looking at these two graphs, we see that store type b has the highest average sale value, but the least total sales. This could mean that it is geared towards selling higher end, more expensive products, resulting in fewer sales but more expensive ones at that.

- Looking at the average sale per customer by store type, its clear that store type d has the highest average sales per customer. Store type b also gets some redemption, not lagging behind the other store types as hard in this regard, but still ranking the lowest.



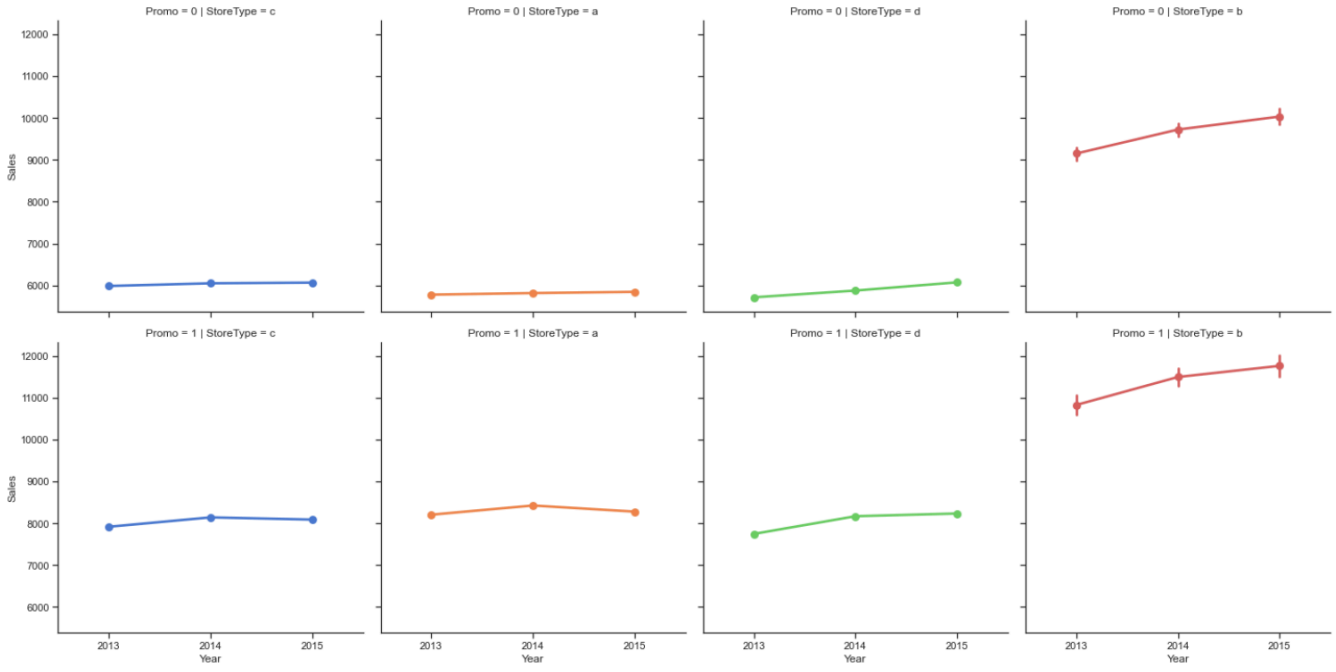
- By grouping the data by **StoreType** on the **Customers** columns and computing the sum, we see the above chart. Store type a has the highest number of customers, which makes sense given it has the highest number of sales

Conclusions**:**

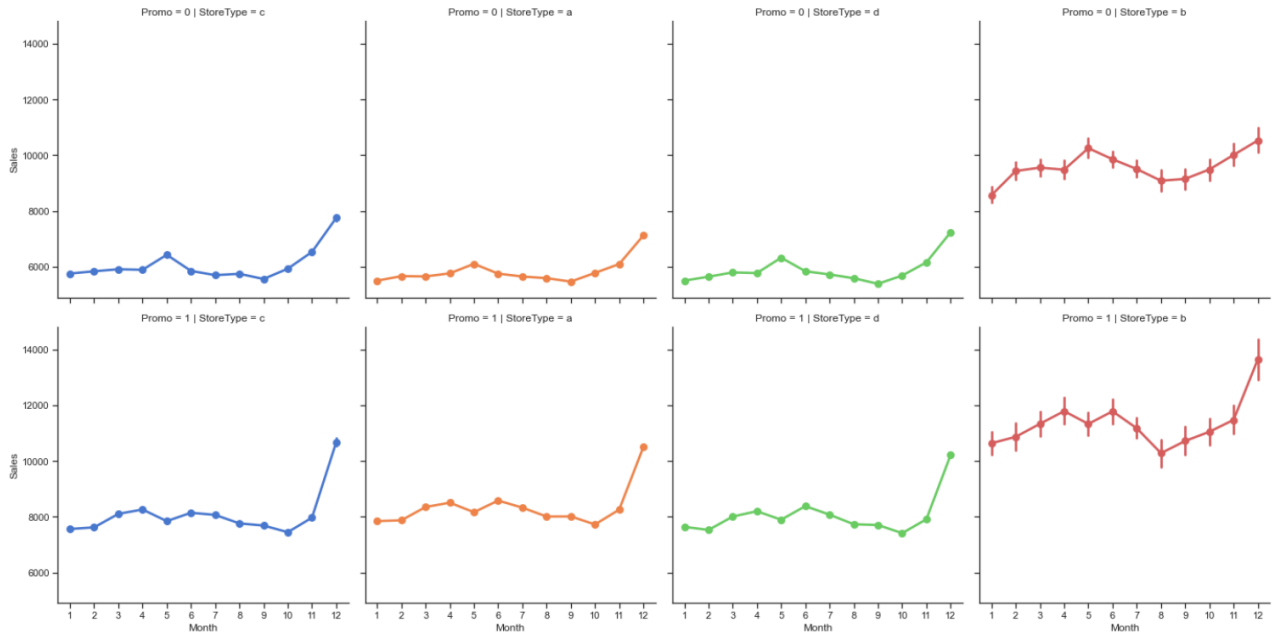
- Type **a**: First in total number of customers and sales, third in average sales value, second in average sales per customer.
- Type **b**: Last in total customers and sales, first in average sales value, last in average sales per customer.
- Type **c**: Third in total sales and customers, second in average sales value, third in average sales per customer.
- Type **d**: Second in total number of customers and sales, last in average sales value, first in average sales per customer.

- These are quite general summary statistics, and although useful, they don't paint the whole picture. We still have promos and holidays to worry about. Let's see how the sales trend when promos are considered. I fdisaplayed this information using seaborn's factorplot, which is very helpful when plotting with categorical features (**StoreType** & **Promo**).

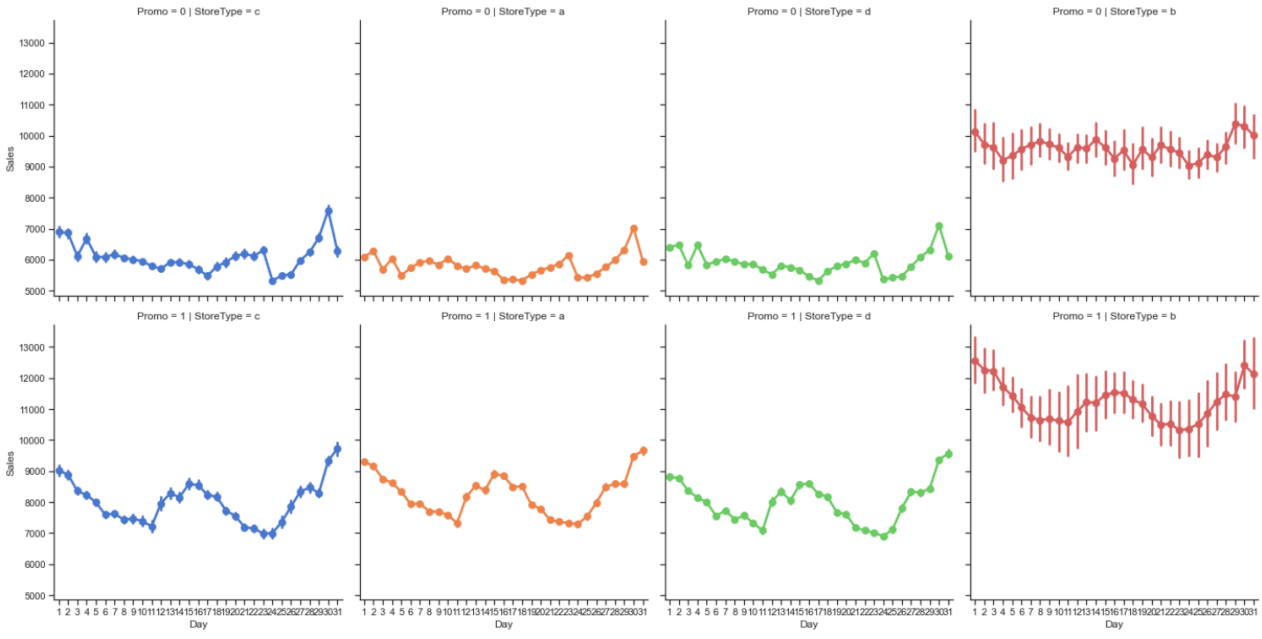
By Year



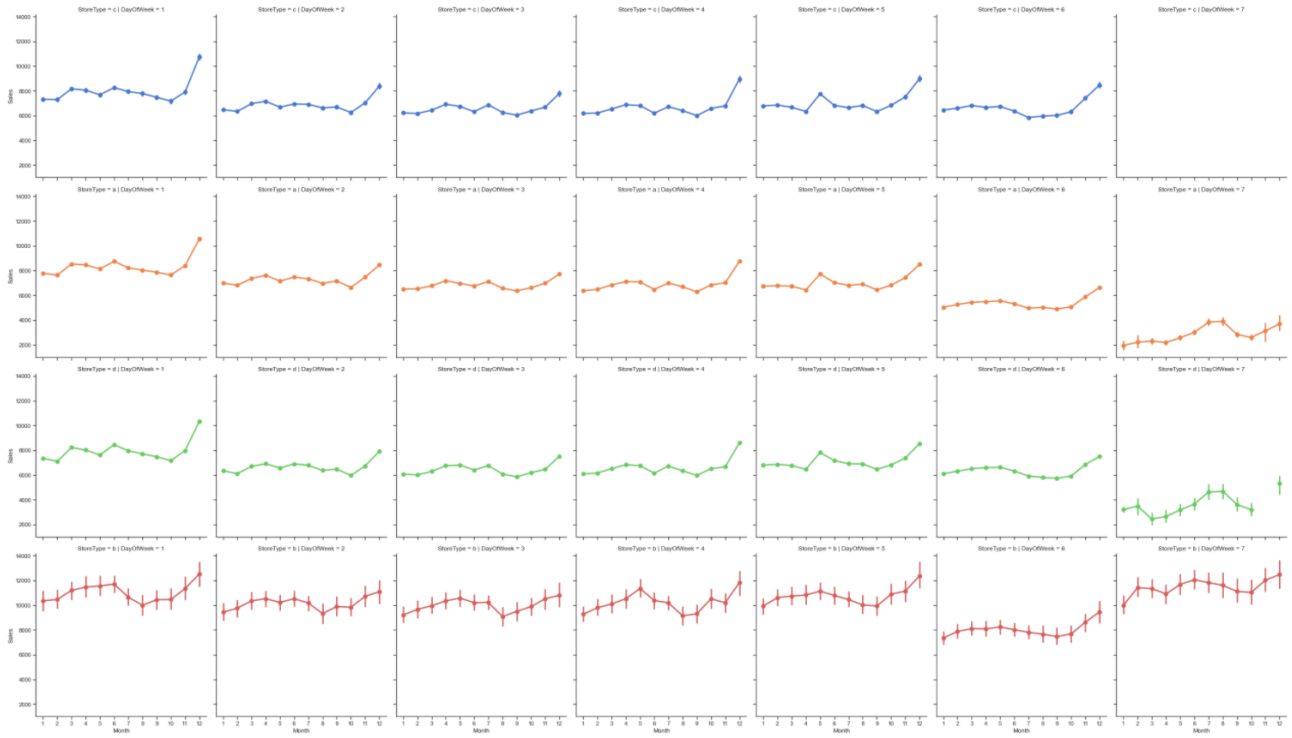
By Month



By Day



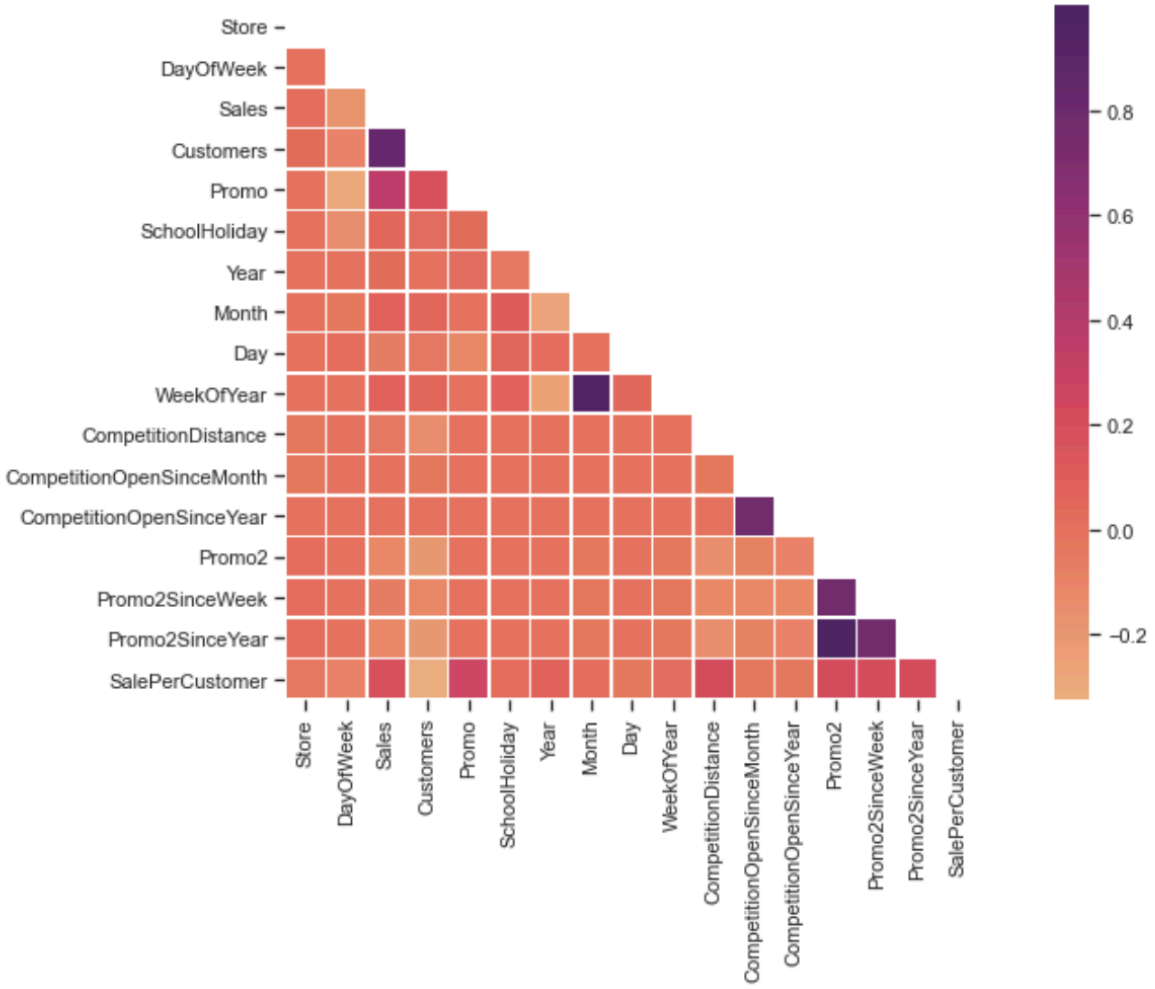
By DayOfWeek



- Sales seem pretty consistent from day to day, with some obvious trends upwards towards the end of the year and minor dips around August/September in most cases. Apparently store type c is closed on Sundays, and some stores of type d are closed on Sundays in October, November, and December. After further investigation, I found that approximately 3% of stores are closed on Sundays.

3.2 Feature Correlations

- The next step to understanding the relationships within the data was understanding how the features were correlated with the target variable, **Sales**. To display this, I plotted a heat map, which uses different shades to identify how correlated features are with one another. The darker they are, the more correlated, and the lighter, the less correlated.



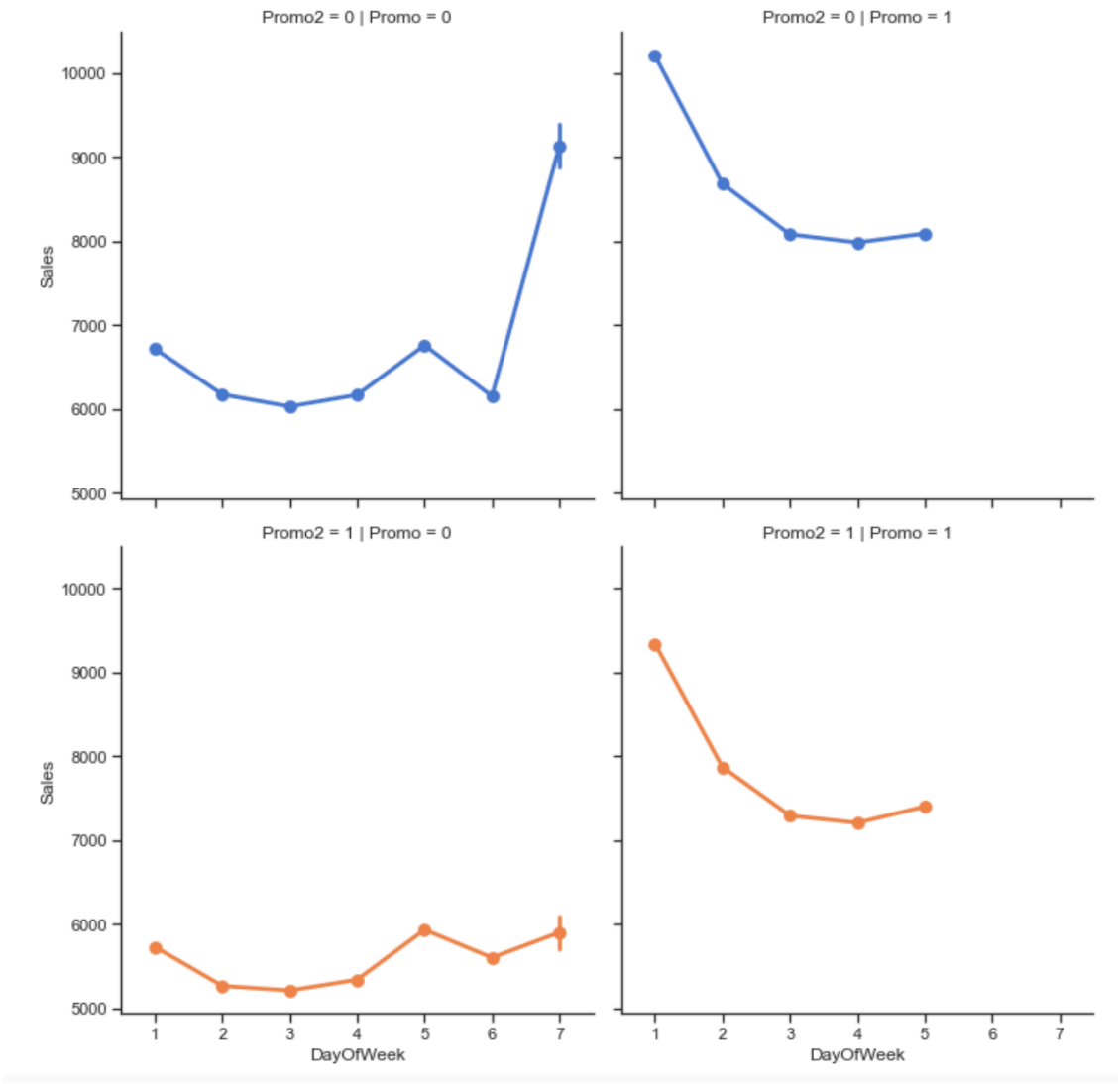
- By observing this chart, I noticed that the features highly correlated with **Sales** were not very abundant, although few had very negative correlations. In order to better understand this, I displayed some of the actual correlations numerically, basing it off of which features seemed darkest at first glance.

```
                    Sales  Customers     Promo    Promo2  Promo2SinceWeek  \
Sales            1.000000   0.823552  0.368199 -0.127556        -0.058493
Customers        0.823552   1.000000  0.182859 -0.202207        -0.130864
Promo            0.368199   0.182859  1.000000 -0.000316        -0.000795
Promo2          -0.127556  -0.202207 -0.000316  1.000000         0.759536
Promo2SinceWeek -0.058493  -0.130864 -0.000795  0.759536         1.000000
SalePerCustomer  0.186563  -0.323926  0.280027  0.215883         0.198835

                 SalePerCustomer
Sales                   0.186563
Customers              -0.323926
Promo                   0.280027
Promo2                  0.215883
Promo2SinceWeek         0.198835
SalePerCustomer         1.000000
```

- In order of correlation strength with **Sales**, these can be ranked **Customers**, **Promo**, then **SalePerCustomer**. **Customers** has a decently strong correlation with **PromoOpen**, meaning that its likely that running a promo draws more customers.

Interestingly, **Promo2** and related variables seem to have a negative correlation with sales. To further understand this, I plotted the following factorplot:



- From this, I could see that when there are no promotions at all (top left plot), the sales slowly rise through the week, dip slightly on Saturday, and shoot back up on Sunday. When both promotions are happening (bottom right), sales are highest on Mondays, but this trend does not differ from when promo is running alone without promo2. Sales for only promo2 are fairly low, with no major changes from day to day.

3.3 EDA Conclusions

- **StoreType a** has the most sales and customers in total.

- **StoreType** b has the lowest sales per customer on average, but highest average sales quantity across stores. This would mean that they most likely sell a lot of cheap things.
- **StoreType** d has the highest sales per customer, but is still beat by a by about twice as much in terms of total sales and customers. This eludes to them selling fewer, more expensive things.
- Customers seem to spend the most money on Sunday when there are no promotions, and the most on Mondays when there are promotions.
- **Promo2** does not seem to help sales in any clear way, while promo definitely does.

4. Feature Engineering and Preprocessing

4.1 Feature Engineering

- Being as that I preliminarily created a few new features, the main goal of this portion was to create dummy variables for the categorical features.

- I removed the **Store** column, for it was no longer necessary and would become an issue with modeling.

- Not to go too into the details here, but I basically renamed the data frame 'df' for simplicities sake, and created new features for each categorical column. For example, the **DayOfWeek** column had values 1-7 denoting each day of the week, so I used one-hot encoding to turn this into 7 boolean columns for each day, with the value 0 denoting it was not that particular day, and 1 denoting it was.

- The final dataframe was as follows:

- I did a quick check to make sure there no missing entries, which there weren't, and the final dataframe had 37 features in total.

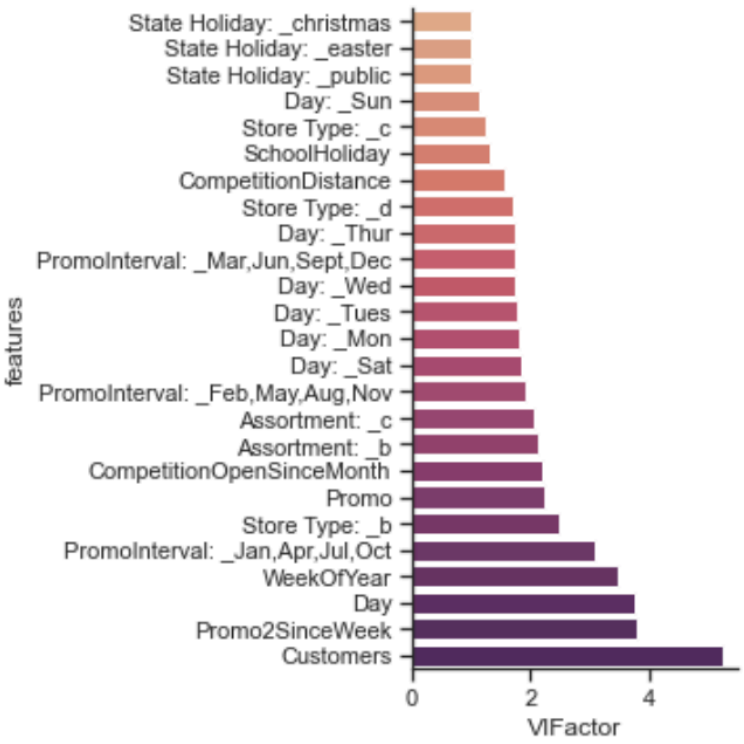
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 844337
Data columns (total 37 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   Sales                           844338 non-null  int64
 1   Customers                       844338 non-null  int64
 2   Open                            844338 non-null  int64
 3   Promo                           844338 non-null  int64
 4   SchoolHoliday                   844338 non-null  int64
 5   Year                            844338 non-null  int64
 6   Month                           844338 non-null  int64
 7   Day                             844338 non-null  int64
 8   WeekOfYear                      844338 non-null  int64
 9   CompetitionDistance             844338 non-null  float64
 10  CompetitionOpenSinceMonth       844338 non-null  float64
 11  CompetitionOpenSinceYear        844338 non-null  float64
 12  Promo2                          844338 non-null  int64
 13  Promo2SinceWeek                 844338 non-null  float64
 14  Promo2SinceYear                 844338 non-null  float64
 15  SalePerCustomer                 844338 non-null  float64
 16  Day: _Fri                       844338 non-null  uint8
 17  Day: _Mon                       844338 non-null  uint8
 18  Day: _Sat                       844338 non-null  uint8
 19  Day: _Sun                       844338 non-null  uint8
 20  Day: _Thur                      844338 non-null  uint8
 21  Day: _Tues                      844338 non-null  uint8
 22  Day: _Wed                       844338 non-null  uint8
 23  State Holiday: _christmas       844338 non-null  uint8
 24  State Holiday: _easter          844338 non-null  uint8
 25  State Holiday: _public          844338 non-null  uint8
 26  Store Type: _a                  844338 non-null  uint8
 27  Store Type: _b                  844338 non-null  uint8
 28  Store Type: _c                  844338 non-null  uint8
 29  Store Type: _d                  844338 non-null  uint8
 30  Assortment: _a                  844338 non-null  uint8
 31  Assortment: _b                  844338 non-null  uint8
 32  Assortment: _c                  844338 non-null  uint8
 33  PromoInterval: _0               844338 non-null  uint8
 34  PromoInterval: _Feb,May,Aug,Nov 844338 non-null  uint8
 35  PromoInterval: _Jan,Apr,Jul,Oct 844338 non-null  uint8
 36  PromoInterval: _Mar,Jun,Sept,Dec 844338 non-null  uint8
dtypes: float64(6), int64(10), uint8(21)
memory usage: 166.4 MB
```

3.2 Preprocessing

- I split my data into a train/test set, using a test size of 25%.

- My next task was to use a variance inflation factor (VIF) function to remove multicollinearity from the data. The purpose of this is to determine which predictor variables are correlated with one another enough to hinder the regression model. The VIF estimates how much of the variance of the model is inflated due to this multicollinearity. Using the following function I was able to weed out the important features I would be using for modeling. Typically, anything over a VIF threshold of 5 is removed due to high correlation, but being as that my main predictor **Customers** had a VIF of 5.67, I adjusted this threshold to 5.7. Luckily, **Customers** was the only feature that was between 5 and 5.7, so I didn't end up with a slew of additional variables in making this adjustment.

- This function iterated 12 times, and left me with the following features and their corresponding VIFactor:



	VIFactor	features
0	5.227738	Customers
7	3.775175	Promo2SinceWeek
3	3.740556	Day
4	3.467383	WeekOfYear
23	3.095265	PromoInterval: _Jan,Apr,Jul,Oct
17	2.474988	Store Type: _b
1	2.224932	Promo
6	2.189495	CompetitionOpenSinceMonth
20	2.116724	Assortment: _b
21	2.069636	Assortment: _c
22	1.914283	PromoInterval: _Feb,May,Aug,Nov
9	1.855767	Day: _Sat
8	1.806585	Day: _Mon
12	1.786106	Day: _Tues
13	1.750986	Day: _Wed
24	1.728423	PromoInterval: _Mar,Jun,Sept,Dec
11	1.727826	Day: _Thur
19	1.702966	Store Type: _d
5	1.555779	CompetitionDistance
2	1.298553	SchoolHoliday
18	1.253352	Store Type: _c
10	1.124919	Day: _Sun
16	1.010495	State Holiday: _public
15	1.006602	State Holiday: _easter
14	1.005391	State Holiday: _christmas

- I redefined the train and test set to only include these features, and it was time to begin modeling.

5. Modeling

- I used 4 models here, and tested their predictive power to see which model performed the best. These models are linear regression, lasso regression, random forrest regression, and xg boost regression. The metric of choice for comparison is the RMSPE or **R**oot **M**ean **S**quared **P**ercentage **E**rror, and is defined as

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2},$$

where y_i denotes the sales of a single store on a single day and $\hat{y}_i$ denotes the corresponding prediction. Any day and store with 0 sales is ignored in scoring. I defined a function that would compute this metric for each model.

- Fitting and predicting with all four models, I found their RMSPE scores and added them to a dataframe for easy comparison:

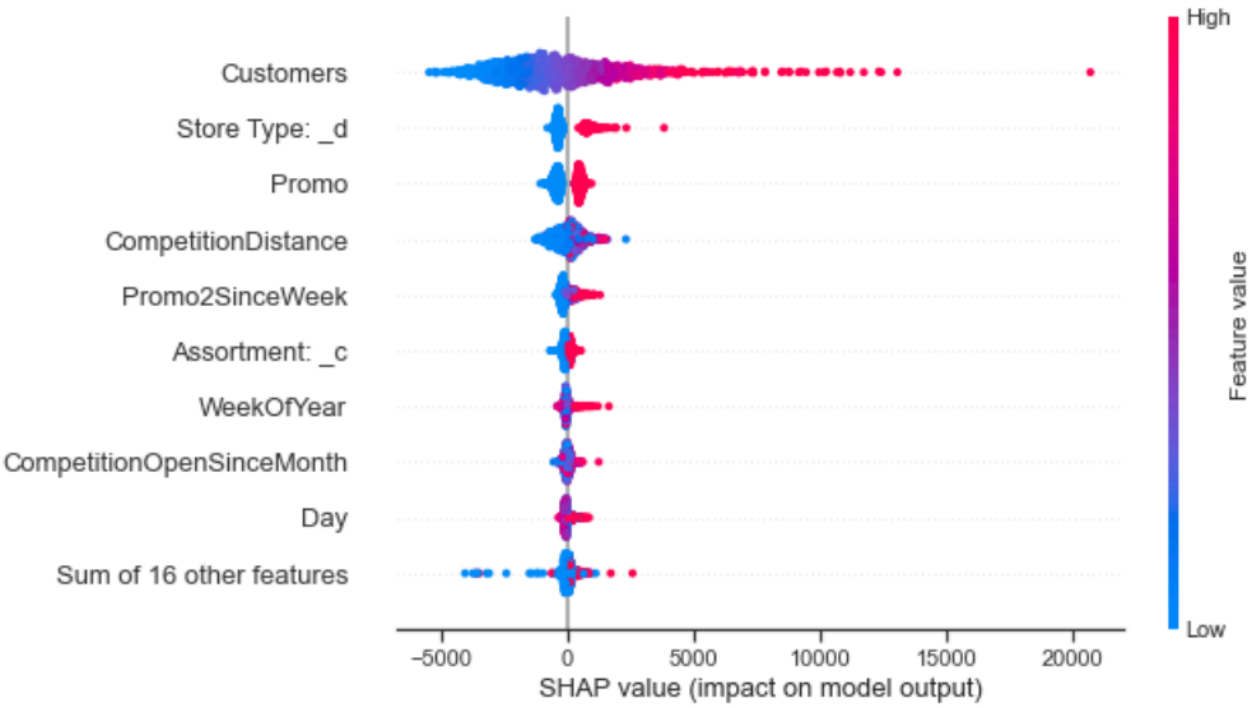
	Linear Regression	**Lasso Regression**	**Random Forest Regression**	**XGBoost Regression**
Metrics/ Model				
RMSPE (Test data)	0.194	0.364	0.143	0.084

- The XG Boost regression model clearly performed the best, almost twice as good as the next best model. (Further hyperparameter tuning via cross validation with random search will be implemented at a later time).

6. SHAP Analysis

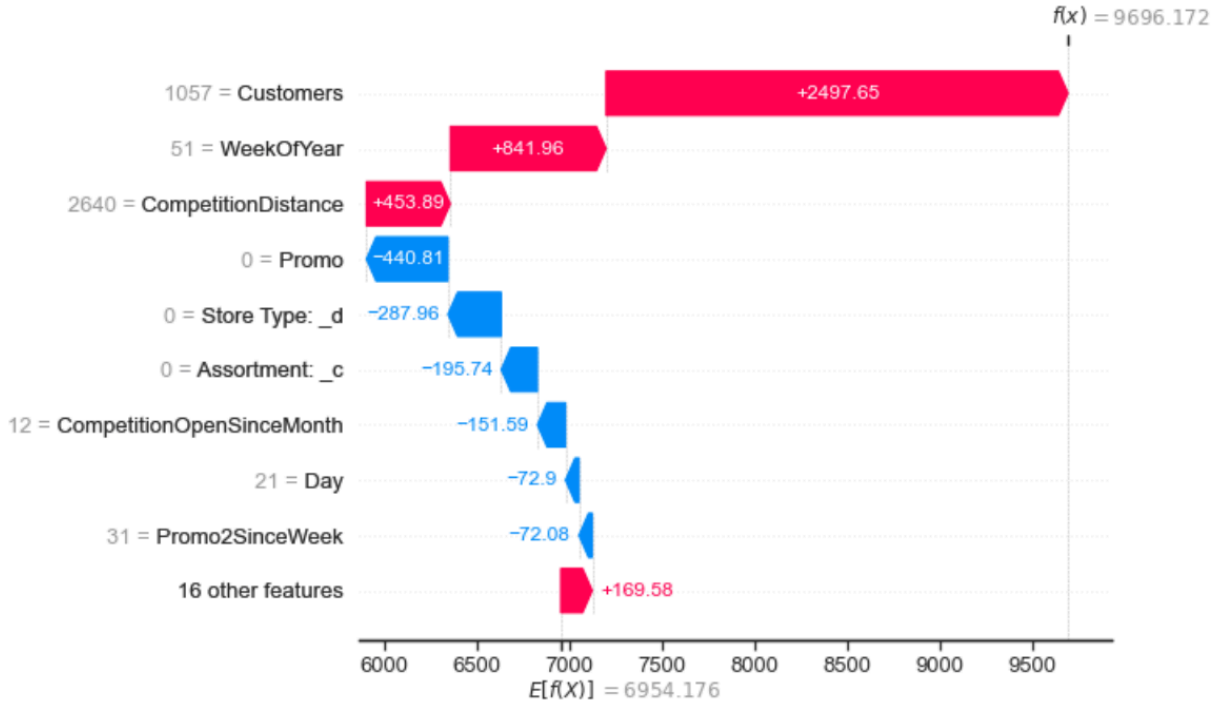
- SHAP stands for **SH**apley **A**dditive ex**P**planations, and is used to interpret the impact features have on the predictability of the model.

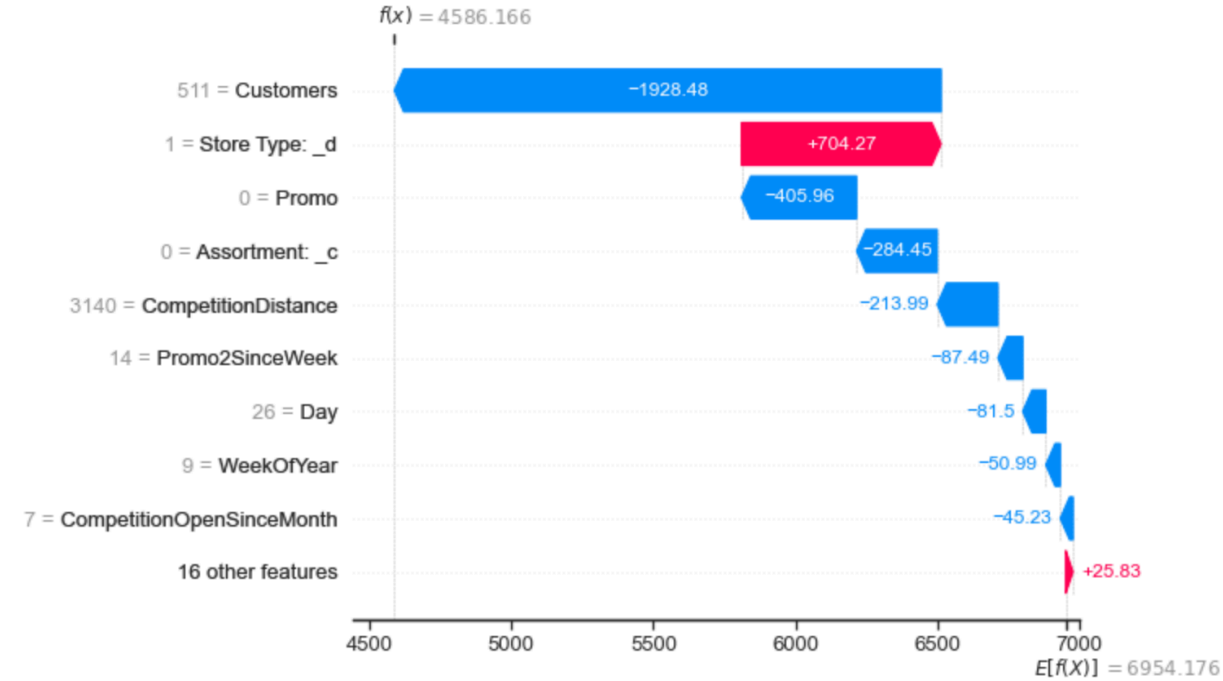
Summary Plot

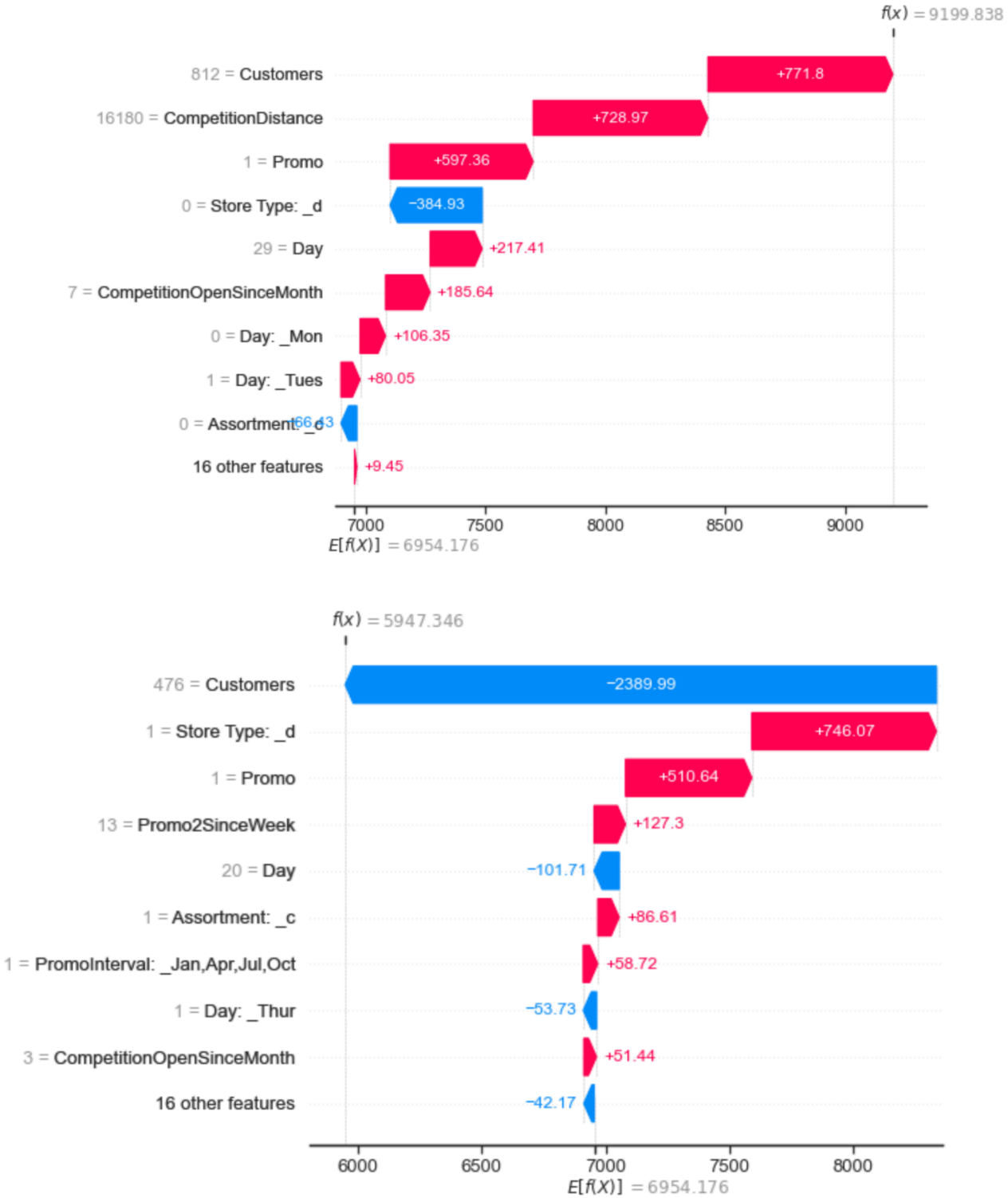


- **Feature importance**: From top to bottom, features are ranked In order of importance.
- **Impact**: The horizontal position shows whether the effect of the value is associated with a higher or lower prediction.
- **Original value**: Color shows whether that variable is high (in red) or low (in blue) for that observation.
- **Correlation**: A high level of 'Customers' has a high positive impact on the amount of sales predicted. The “high” comes from the red color, and the “positive” impact is shown on the x-axis. Similarly, we can say that just about all of the other features have positive impacts, whether indicated by low negative impacts, or high positive ones.

Waterfall Plots







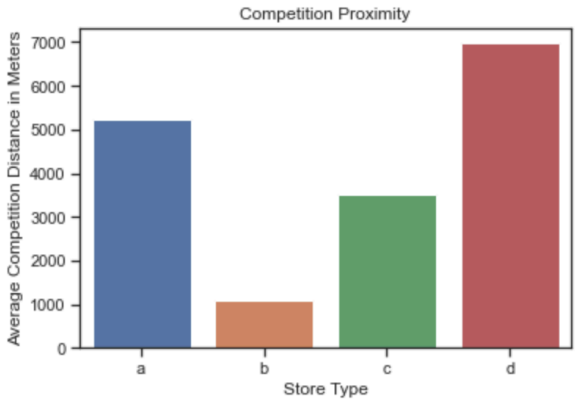
Waterfall plots are used to display explanations for individual predictions, so they expect a single row of an explanation object as input. The bottom of a waterfall plot starts as the expected value of the model output, and then each row as you go up shows how the positive (red) or negative (blue) contribution of each feature moves the value from the expected model output over the background dataset to the model output for this prediction

7. Conclusion

- Of the 4 models tested, the XG Boost regressor performed the best.

- The top 4 features that drove sales were **Customers**, **StoreType: _d**, **Promo**, and **CompetitionDistance.**

- By observing the 4 waterfall plots with values selected at random, conclusions can be made about the nature of these top features:
  - **Customers:** Somewhere between 511 and 812, the amount of customers changes from negative to positive in terms of contribution to sales. Higher numbers of customers indicate significantly more sales.
  - **StoreType: _d:** Looking back on the exploratory analysis, store type d had the highest average sale per customer. From the above charts, we see that if store is of type d, they are more likely to see higher sales. This checks out with the aforementioned statement.
  - **Promo:** As suspected, if a promotion is being held, it will have a positive impact on sales, while no promotion results in a negative impact. This also checks out with the exploratory analysis, for **Promo** had the second highest correlation with **Sales**, second to **Customers**.
  - **CompetitionDistance:** Here it appears that the farther a competitor store is located, the more positive of an effect this can have on sales. Plotting the average distance to nearby competition, store type d has the highest, which can be another reason as to why they do well in sales.



  - There are several other features represented in these waterfall plots, each having meanings that relate back to the exploratory analysis of the data.

- For future improvements, it would be very useful if there were more information on the customers, how the store types vary, what types of promotions there are, how competition operates, etc. Having additional information about the most important features for modeling can really fine tune how well sales can be predicted. Only about a third of all stores (~1,000 of ~3,000) were accounted for in the data as well, so having the data on as many more locations as possible wouldn't hurt.

- For further improvements, doing a time series analysis would be very helpful. In doing so, I would be able to understand how the sales change with time as well.