Jamie Yachera

Rossman Store Sales Project Paper – DC-DAT-9

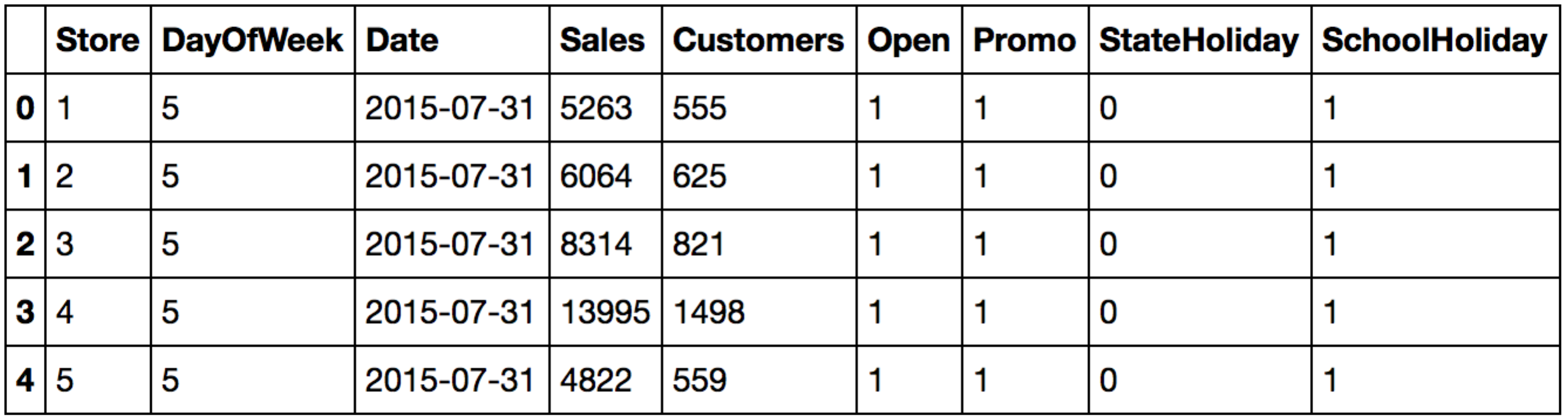
12/7

**Model Overview**

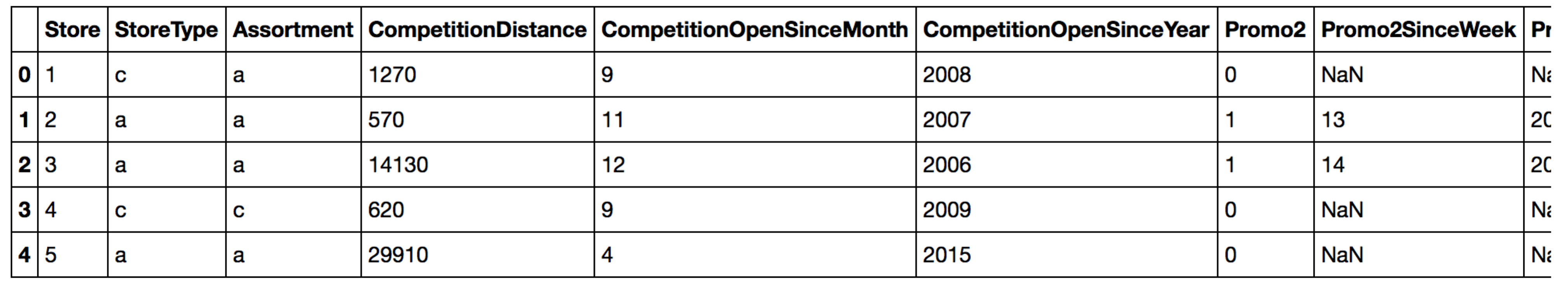
Rossman Stores, a European drug store chain, provided store and sale information for 1,115 of their German stores as part of a 2015 Kaggle competition. Rossman seeks to eliminate the variability in their manual sales forecasting process by requesting participants develop a model that can accurately predict the next six weeks of daily sales. Kaggle posted two csv files of anonymized sale and store data for use in this exercise:

* Historical sales data from 1/1/13-7/31/15 (1k+ rows), including the following features:
  + Store ID
  + Day of Week
  + Date
  + Sales ($)
  + Customers (#)
  + Store open flag (0,1)
  + Store promo flag (0,1)
  + State holiday flag (category)
  + School holiday flag
* Store-level metadata (1115 rows), including the following features:
  + Store ID
  + Store Type (categorical)
  + Assortment (categorical)
  + Competition distance
  + Competition open month
  + Competition open year
  + Flag as to whether the store participates in “Promotion 2”
  + Week Promotion 2 started
  + Year Promotion 2 started
  + Consecutive intervals in which the promotion starts new

*Figure 1: Snapshot of sales dataframe ‘Train.csv’*



*Figure 2: Snapshot of the store dataframe from ‘Store.csv’*

****

Kaggle also provided a third ‘Test.csv’ file to be used to generate the Kaggle submission. This file included all variables of the ‘Train.csv’ set with the exception of ‘Sales’ and ‘Customers’. Each Kaggle submission was scored using the “RMSPE” (root mean squared percentage error) metric.

**Exploratory Data Analysis**

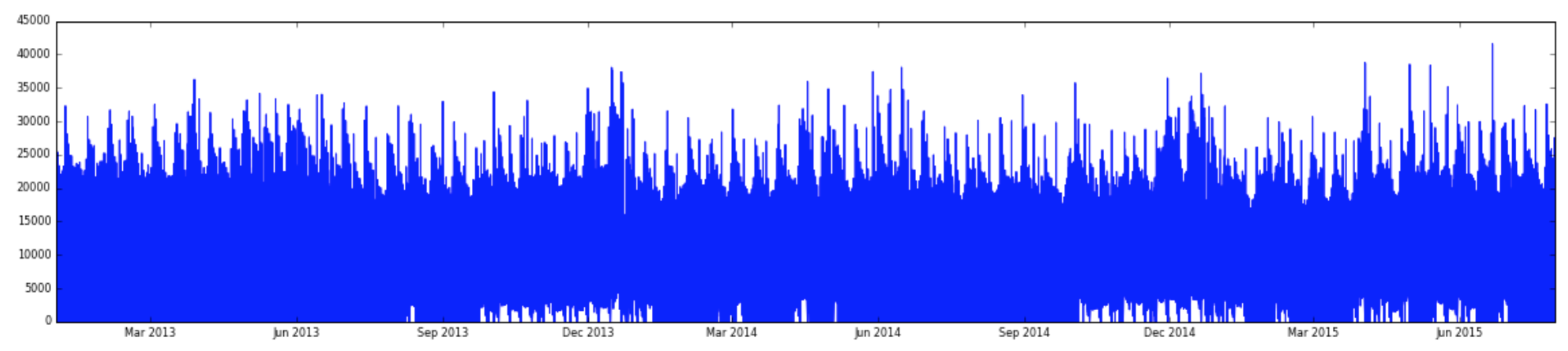
Each file was first evaluated for completeness. The isnull() function indicated that out of the 1115 stores, 3 were missing the Competition Distance, 354 were missing the Competition Open Month/ Year, and for the 544 stores that did not participate in the special promotion, they were missing the Week/ Year that the promotion started.

Two data sets were created to accommodate for the missing information. The first data set removed the three stores that were missing competition details and the 354 stores that were missing the Competition Open Month/ Year. The Promotion Week/Year was replaced with 52/1900 to allow for later date difference calculations. I initially took this approach because I was not intending to submit results to the Kaggle Leaderboard. Removing the competition information yielded a sufficient sample size of 693861 rows. A second “Data Set 2” was created and eventually used in the final analysis. Null competition distances were replaced with the average competition distance from the other 1112 stores and each row containing a missing Competition Open Since Month/ Year was replaced was replaced with 1/1900. The Promotion Week/Year was again replaced with 52/1900 to allow for later date difference calculations. Both data sets were evaluated and used throughout the model build and showed only slight differences.

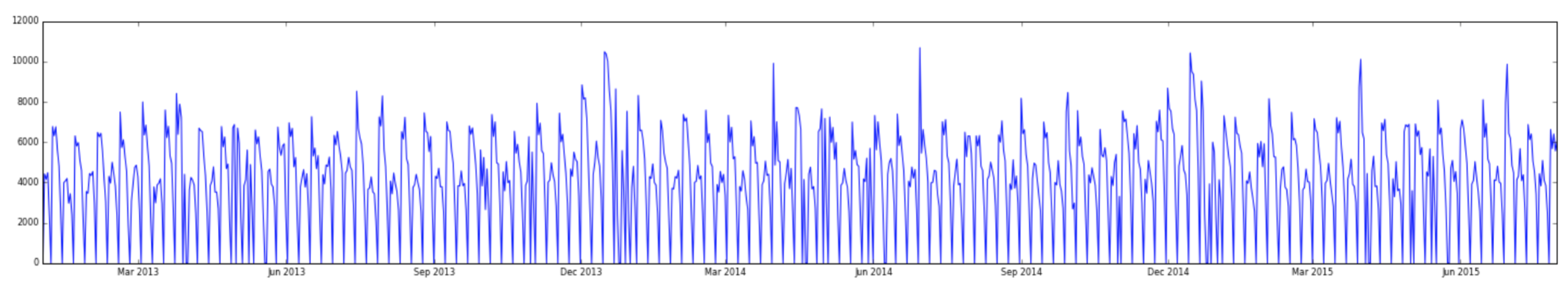
Upon further inspection, 180 stores had missing entire rows of sales data from June 2014 until the end of the year. No adjustments were made to this data as the entire row was missing vs. a subset of information.

The groupby().Sales.mean(), .decribe, plot(kind=’hist’) plot(kind=’scatter’) functions were used on the majority of features to determine whether or not these would be indicative of sale price for the final model. The daily sales data across all stores was not linear and immediately suggested that using a linear regression with store sales as the dependent variable would not yield favorable results. Closer examination of individual stores indicated that stores part of the promotion (Promotion = 1) had no sales on Saturday or Sunday. In general, stores with promotions had higher overall average sales and minimal sales were made on Sunday.

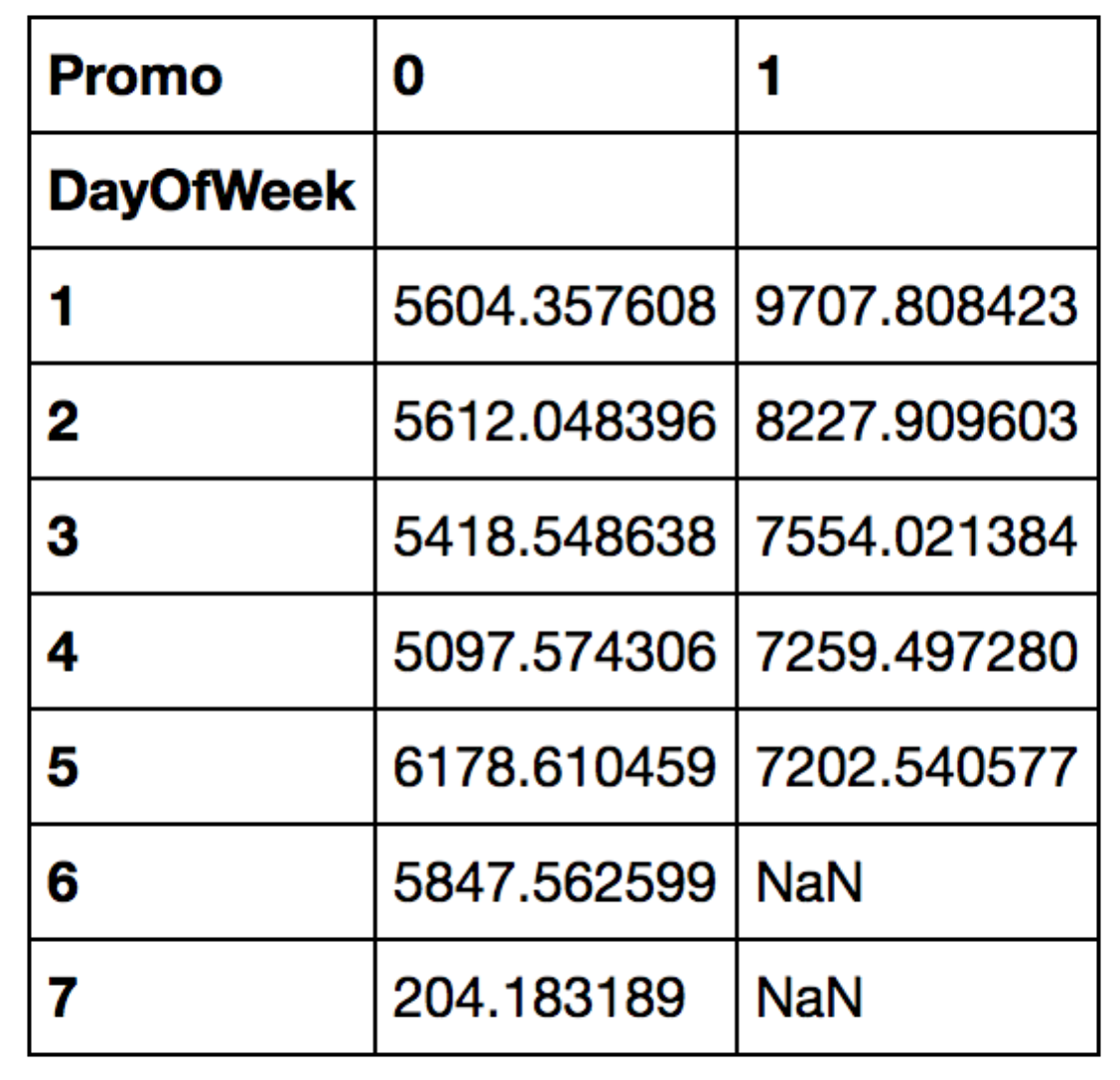
*Figure 3: Stores sales by date (all stores)*

**

*Figure 4: Stores sales by date (Store 2, selected as an example).*

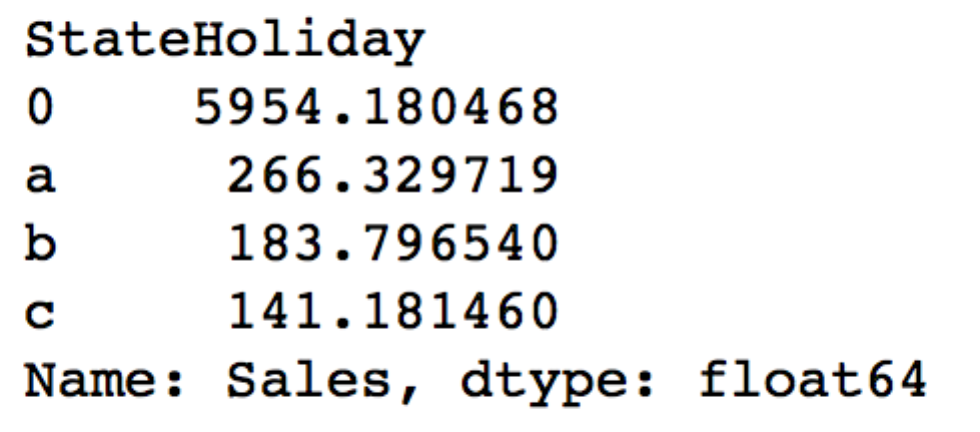
**

*Figure 5: Mean of store sales by Promotion Type and Day Of Week*

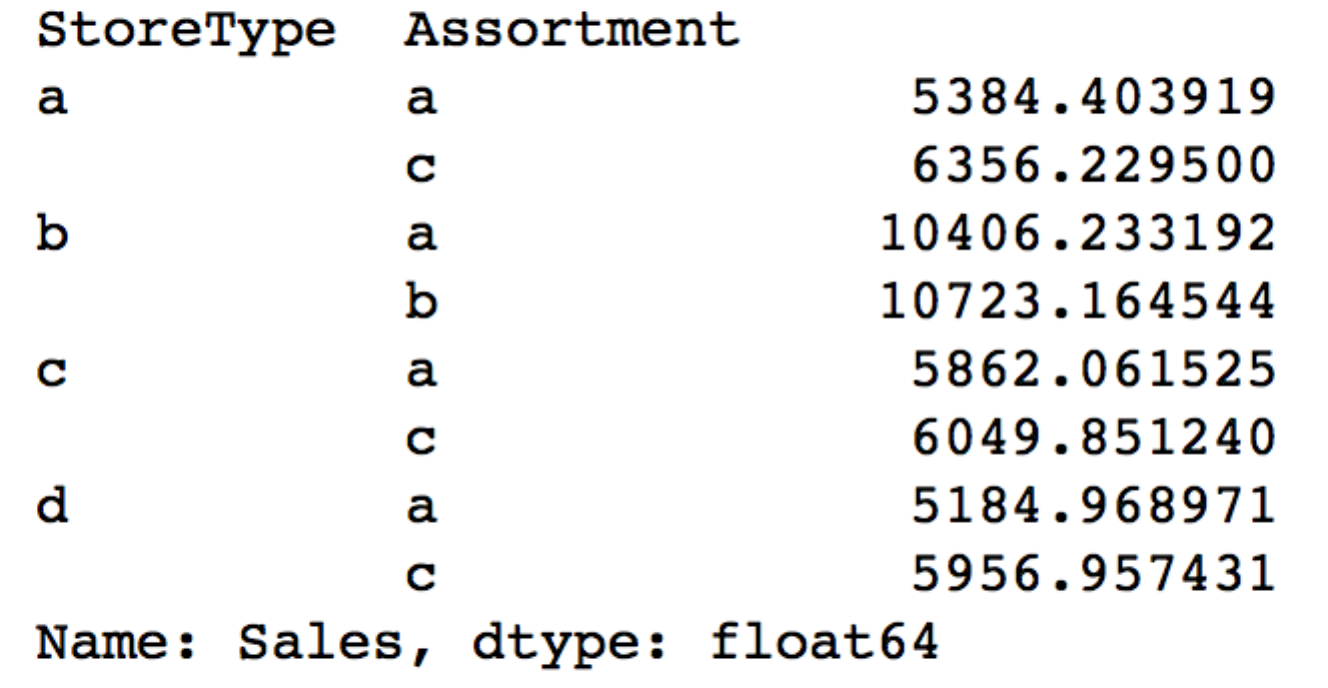
**

The two datasets were merged on Store ID and further evaluated. Overall, the data was not that interesting and the majority of variables showed only slight differences. This suggested that all of the variables should be tested for inclusion in the final model. Outcomes worth noting included: less sales occurred on state holidays and store type ‘b’ had higher sales on average than a,c, or d. Promotion Interval Type was one of the few features that did not appear to have an impact on sales. The most interesting finding was on the Competition Distance variable. The four stores with the top sales all had low Competition Distances. Intuitively I would have expected that stores with the least competition would have the highest sales, however stores with high sales may be in areas of more dense populations (e.g. large cities and metropolitan areas) which would also have close competition.

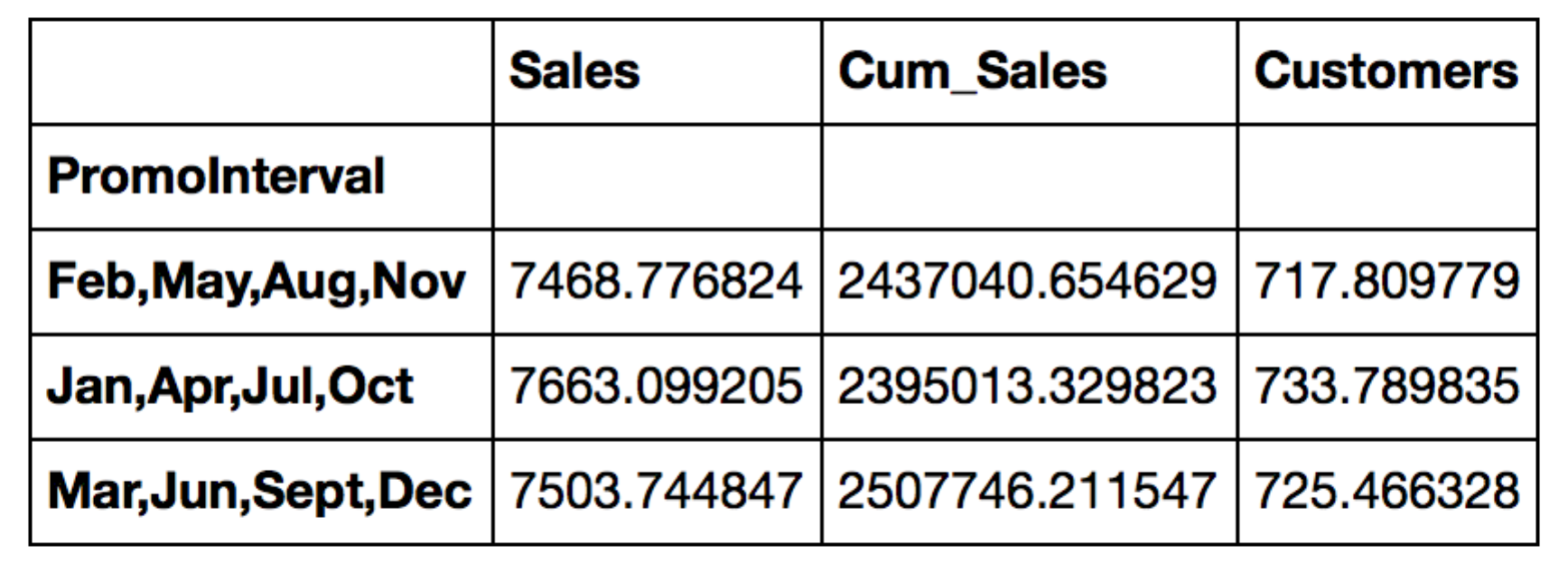
*Figure 6: Mean Sales by State Holiday*

**

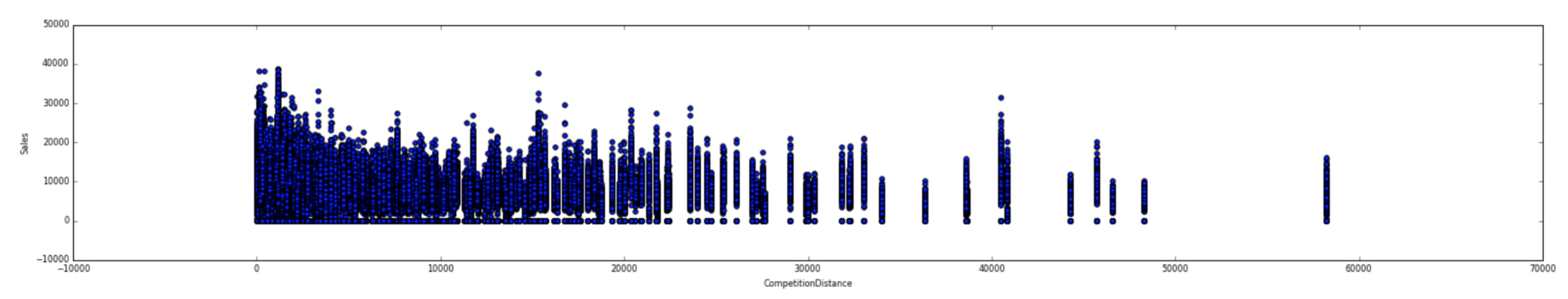
*Figure 7: Mean Sales by Store Type and Assortment*



*Figure 8: Mean Sales, Mean Cumulative Sales, and Mean Customers by Promotion Interval Type*

**

*Figure 9: Sales by Competition Distance*

**

**Feature Creation**

Numerous features were initially created to enhance the sales data, store data and merged data sets. Initial enhancements included:

* Sales ‘Date’ was converted to an actual date using to\_datetime()
* ‘Sale\_Month’ and ‘Sale\_Week’ were extracted from ‘Date’ using the .dt.month and .dt.weekofyear function
* Cumulative sales (‘Cum\_Sales’) and the cumulative number of customers (‘Cum\_Cust’) for each store based on date progression
* Average sales per customer (‘Avg\_Cust\_Sales’) based on newly created cumulative sales and cumulative customers feature
* Promotion start date (‘Promo\_ Date’) using ‘Promo\_Week’ and ‘Promo\_Month’. By default the first day of the week was used in this calculation, based on the isoweek standard. For example if the Promo Week = 5 and Promo Year = 2013, the resulting date would equal 1/27/2013.
* Sale Since Promo Started (‘SSPS’) calculated by the date difference between the current sale date and the date the promotion started. These date differences were then classified as integers. All negative numbers (i.e. the promotions started after the sale date) were reclassified as 0.
* Number of competitors opened during the same month (‘Comp\_Count’)
* Weekend flag (‘Weekend’) for when the Day of the Week equaled Saturday or Sunday
* School Holidays were augmented to factor in State Holidays. Rossman’s data dictionary indicated that all schools were closed on state holidays
* Dummy features for ‘StoreType’, ‘Assortment’, ‘PromoIntervals’ and ‘State Holiday’
* Response variable ‘Sales’ was created into bins (‘Sale\_Bins’) at intervals of $500 and $1000 using the pd.cut function. ‘Sale\_Bins’ would be used to run classification models and the bin cuts sizes were chosen after reviewing descriptive sales statistics indicating the mean of $5780 and standard deviation of $3784

Additional features were created for Date Set 2 and used later in the model whose output was submitted to Kaggle:

* Competition start date (‘Comp\_ Date’) using ‘CompetitionOpenSinceMonth’ and ‘CompetitionOpenSinceYear’. By default the first day of the month was used in this calculation.
* Sale Since Competition Opened (‘SSC0’) calculated by the date difference between the current sale date and the date the competition opened (‘Comp\_Date’). These date differences were then classified as integers. All negative numbers (i.e. the promotions started after the sale date) were reclassified as 0.
* Median sales by store/day\_of\_week/promotion flag (‘Sales\_New’).
* Maximum of cumulative sales by store/day\_of\_week/promotion flag (‘Cum\_Sales\_New’). This was used to enhance the final model. The ‘Cum\_Sales’ feature could not be included as a feature because the test set did not include the ‘Sales’ feature required for this calculation.
* Median number of customers by store/day\_of\_week/promotion flag (‘Customers\_New’). This was used to enhance the final model. The ‘Cum\_Cust’ feature could not be included as a feature because the test set did not include the ‘Customer’ feature required for this calculation.
* Median average sales per customer by store/day\_of\_week/promotion flag (‘Avg\_Cust\_Sale\_New’). This was used to enhance the final model. The ‘Avg\_Cust\_Sale’ feature could not be included as a feature because the test set did not include the ‘Customer’ or ‘Sales’ features required for this calculation.

**Feature Selection and Model Evaluation - Classification**

The Rossman sales data is affected by seasonality and is not linear. Without significant fine-tuning and adding in a time series component, a linear regression model using Sales data ‘as-is’ will poorly predict six weeks of future sales. This problem was first approached using classification models. A KNN classification model was first chosen because there may be high correlation among the attributes in the data set.

Over 20 combinations of features were evaluated to determine the best model prediction at various values of k{10,12,15,20,30,60}. Based on the binning exercise, null accuracy = .17. Using Data Set 1, features were added and subtracted based on intuition from the data exploratory phase. The KNN model with the highest train-test-split accuracy (.379 using the $500 sale bins, k=10, .506 using the $1000 sale bins, k=10) included the following features: Store, Day of Week, Store Type (via dummy variables), Store’s Assortment (via dummy variables), Sale Month, School Holiday, State Holiday, and Promotion. However, when the model was re-run using Cross Validation, the accuracy was significantly lower at .168, less than the null hypothesis. This result was perplexing, considering all else was equal and the random state was checked to ensure the train-test-split model was not inflated. Knowing that the KNN features were not normalized, I deemed that this model was not stable as-is. Similar attempts were tried using Logistic Regression and Random Forests. Each iteration took hours to run or crashed my operating system, encouraging me to investigate other model options.

**Feature Selection and Model Evaluation – XGBoost**

XGBoost is an extreme gradient boosting package and a top choice for Kaggle competitors. Many posters on the Rossman Store’s Kaggle forum achieved superior results using this model package. Additionally, XGBoost performs much faster than SciKit-Learn due to its distributed processing feature. A summary of the gradient boosting methodology can be described as:

“Gradient boosting is a machine learning technique for regression problems, which produces a prediction model in the form of an ensemble of weak prediction models. Gradient Boosting Decision Trees use decision tree as the weak prediction model in gradient boosting, and it is one of the most widely used learning algorithms in machine learning today. Its high accuracy makes that almost half of the machine learning contests are won by GBDT models.” (Source: <http://zhanpengfang.github.io/418home.html>).

After additional research on XGBoost and its application, I decided to change my methodology and use the XGBoost package and submit my model’s results to Kaggle. Since the Sales data was not linear, I used the ln of Sales as the dependent variable and later converted it back to ‘Sales’ when calculating the accuracy score.

The XGBoost package allows the user to vary numerous parameters including (Source: http://xgboost.readthedocs.org/en/latest/parameter.html):

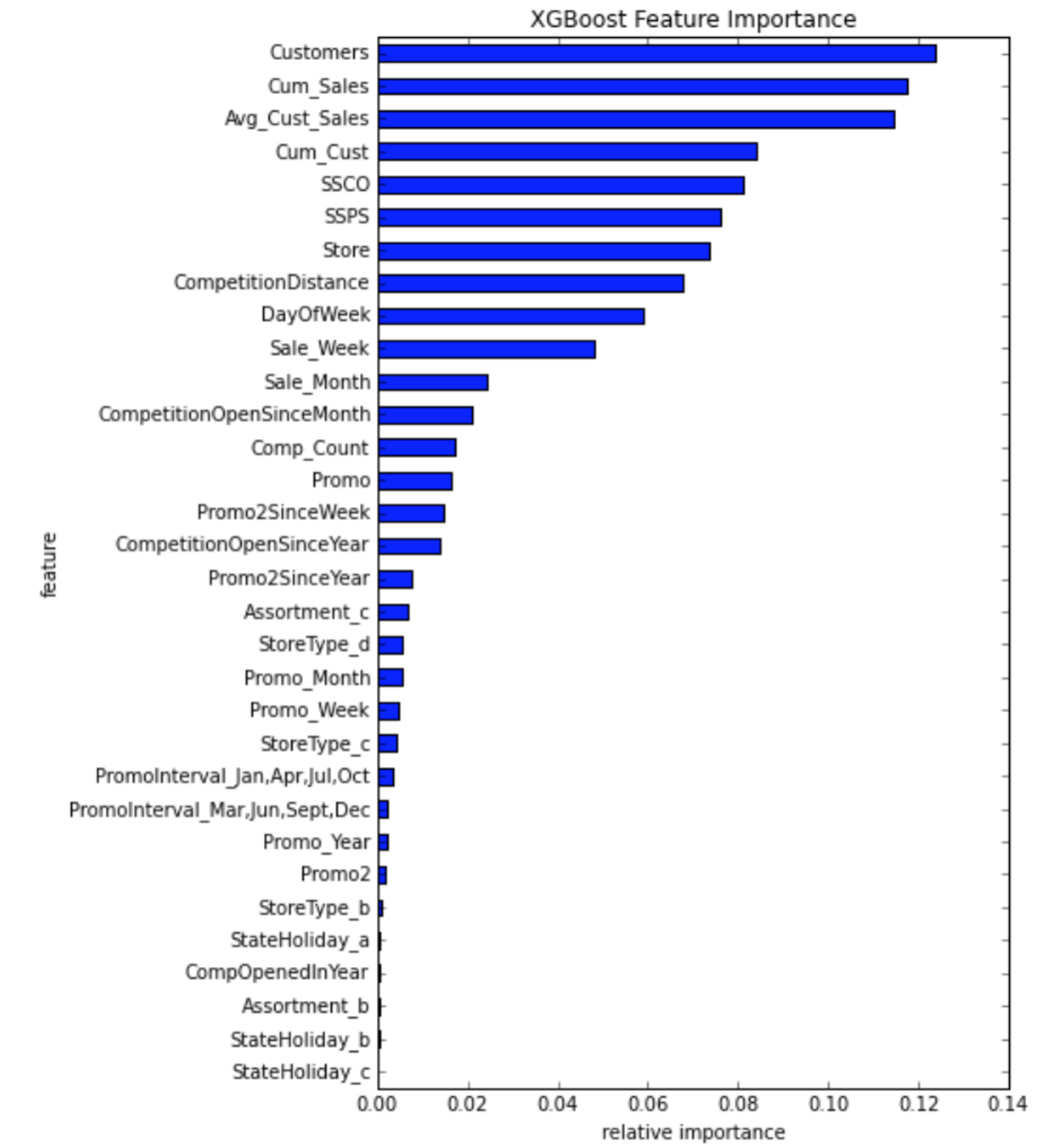
* objective: The learning task of the model. I chose the default: ‘reg:linear’, specifying linear regression given sales was continuous and not categorical
* booster: Linear model or tree boosting option. I chose the default: ‘gbtree’
* eta: shrinks the boosting weights after each boosting step to be more conservative. I chose the default: ‘0.3’
* max\_depth: the max depth of the tree, similar to the max\_depth in a random forest. I chose ‘10’ over the default ‘6’
* subsample: subsample ratio of training instance to test. I varied between the default (‘1.0’) and the alternate ‘0.9’
* colsample\_bytree: substample ratio of columns when constructing trees. I varied between the default (‘1.0’) and a smaller value of (‘0.7’)

Additional options included:

* number of boost rounds: the number of booster iterations. I initially chose 300 and then tested different feature sets using 100 rounds in order to reduce processing time. The final submission to Kaggle used 300 rounds. 400 rounds was also tried but produced a higher error indicating that less is not always more.
* stopping rounds: indicates the maximum number of rounds that you will allow the model to run and have no change in the evaluation metric. This prevents the model running for long periods of time and not improving. I chose 100 for the majority of models. Models using 50 performed worse.
* feval: the evaluation metric. I used the root-mean-squared-percentage-error function posted on the Kaggle forum and embedded it within my model

One last modification is that I removed all rows that had 0$ sales before I began testing. Using the full feature set created during the KNN phase, I achieved a .051975 RMSPE, more accurate than the top Kaggle submitter. On closer evaluation, the XGBoost Feature Importance graph indicated that ‘Customers’, ‘Cum\_Sales’, ‘Avg\_Cust\_Sales’ and ‘Cum\_Cust’ were the most significant:

*Figure 10: XGBoost Feature Importance on the full feature set*



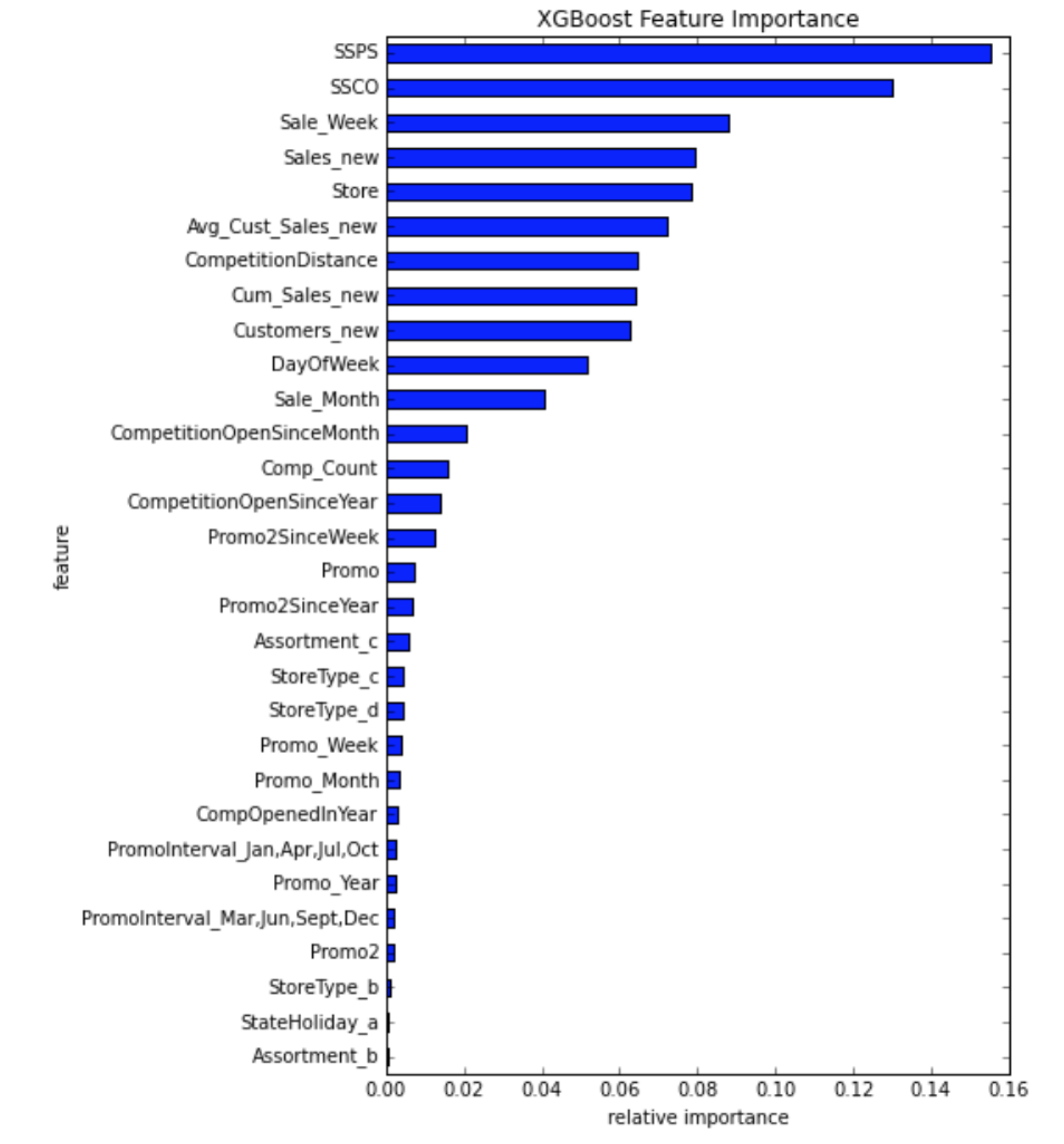
It was after numerous iterations trying to refine the full feature set that I realized the most important features in my model could not be replicated on the test set for Kaggle submission. Recall that the test set did not include ‘Sales’ (what we are trying to predict) and the number of Customers. It was at this time that I created the new features described above for Data Set 2.

Three RMSPE values were created: that on the train set, based on the test (labeled as Eval), and the overall accuracy of RMSPE calculated using y^hat and the true values of y. In all instances the Eval RMSPE was significantly higher than the train RMSPE. Many other Kaggle competitors had a similar issue and indicated that XGBoost can be volatile. The following table shows a sampling of the different models run with different features/ parameters.

*Figure 11: Table of different XGBoost models and the resulting RMSPE (example)****.*** *‘E’ is the model used on the test set for submission to Kaggle.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** |
| Parameters | Eta = .03, Num Boosters = 100  Stopping round = 50  (Ended up stopping at 62, best iteration 12) | Eta = .03, Num Boosters = 100  Stopping round = 100 | Eta = .03, Num Boosters = 100  Stopping round = 100 | Eta = .03, Num Boosters = 100  Stopping round = 100 | Eta = .03, Num Boosters = 300,  Stopping round = 100 |
| RMSPE – Train | 0.203276 | 0.093468 | 0.097273 | 0.132051 | .073730 |
| RMSPE – Eval | 0.231554 | 0.198595 | 0.201777 | 0.235762 | 0.189150 |
| RMSPE - Overall | 0.24268 | 0.198595 | 0.201777 | 235762 | 0.189150 |
| Features | Sale\_Month, Store, Sales\_new, DayOfWeek, Promo | 29 features. Top 5: SSPS, SSCO, Sale\_Week, Avg\_Cust\_Sale\_new, Store | 9 features using the top 9 from prior model. Top 5 of importance were same as prior model | 15 features at random. Top 5 of importance: Store, Competition Distance, Promo, Promo2, Sale\_Month | 29 features. Note this is the same as “B” but uses a 30- vs. 100 boost. |

*Figure 12: Model (‘E’) feature importance – Submitted to Kaggle*

**

Prior to submitting my submission file to Kaggle, I modified the Sales amount to be ‘0’ for any day that the Store was closed. This final submission landed me on the LeaderBoard in the 2125 spot with a score of (.14242). Note that this score was between my train-RMSPE score (.073730) and my Eval RMSPE score (.189150). I also tried the same features in a linear regression SciKit-Learn model (using the ln of Sales) and had a higher error at .23263. All else held equal, the XGBoost model performs more favorably than the standard linear regression model.