Samuel Gonzalez & Luke Rouquette

Spring 2018 Project

Due: Apr 22,2018

ISTM601

Grocery Spending Data Analysis using Python

# Background & Objective

Understanding the dynamics of a shopper’s buying patterns is crucial for a range of reasons. From grocery marketing to resource allocation, retailers rely on these patterns to make business decisions. To be able to understand the dynamics of the shoppers a predictive model is needed to understand purchasing patterns associated with larger spend, and to understand the statistically significant contributors.

Weekly data from a grocer’s transaction database was retrieved and utilized for this project. Throughout the project, the data is manipulated into a shape and form that can be used to build a predictive model for ‘Basket Total’, which is the total amount spent by a customer on a given transaction.

Some of the parameters which were available were related to Transactions, Customer, Product Department, Store, and Spending:

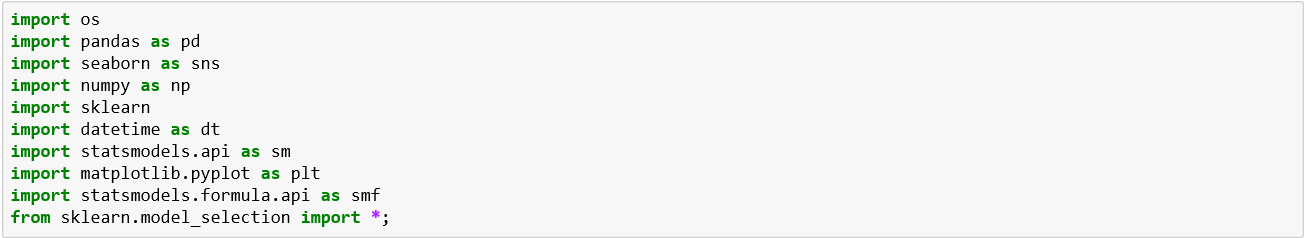
* Transaction:
  + Shop Week, Date, Hour
* Customer:
  + Life Stage
  + Price Sensitivity
  + Basket Type
  + Basket Dominant Mission
* Departments
  + Department
  + Sub-Department
  + Class
* Store
  + Format
  + Region
  + Retailer (Franchise ID)

# Approach

We used the traditional data science approach, where about 80% of the effort was spent understanding and exploring data, and 20% of the effort was spent on building the predictive model, including validation of the model.

# Tools and Libraries Used

Several Python modules were utilized in the development of the analysis. The modules are listed below. The **Code** section below describes the usage of each tool in detail



# Python Code

Python code written to carry out each of the steps in the **Approach** section. The code contains a main function and supplemental functions. Each function along with a description of usage are laid out below.

## Supplemental Functions

**importData()**

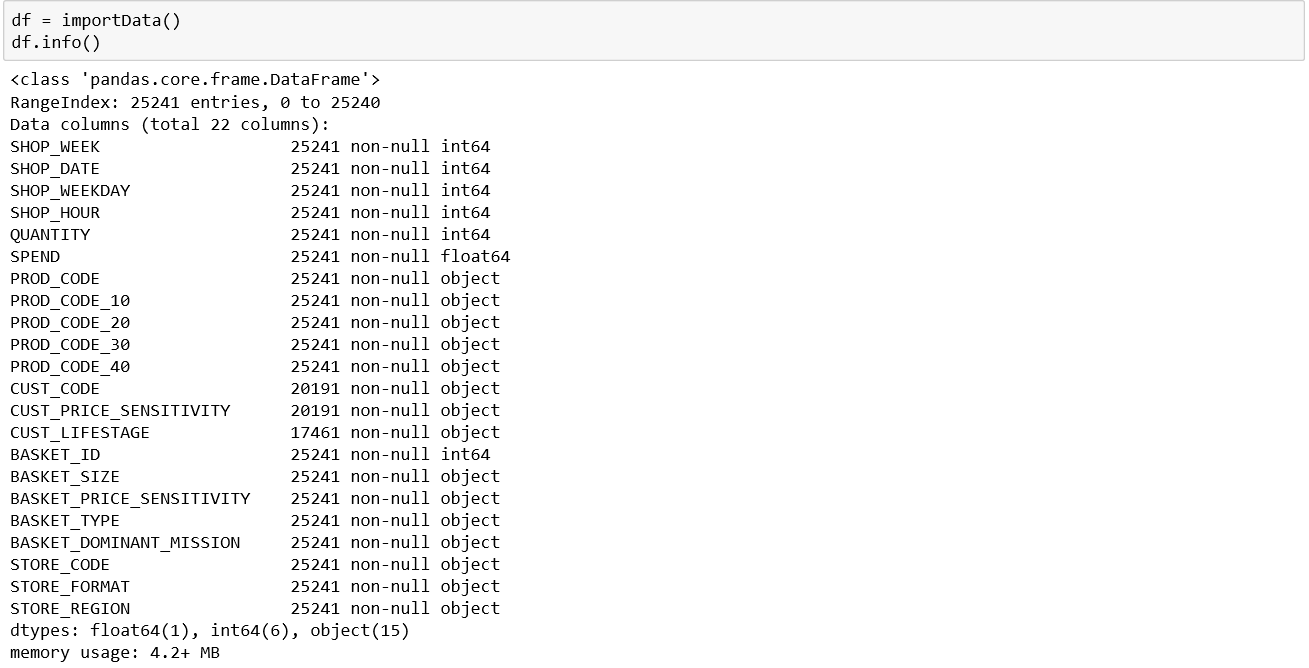
**This function uses the *pd.read\_csv* function within the *pandas* model in order to import a .csv file used for the analysis. It requests user input for the name of the file to analyze. A given file can contain data for a certain week, so this gives the user the ability to select which week to analyze. The *pd.read\_csv* function is wrapped around a *try/catch* python operator along with a *while* statement to validate the name of the file give. The continuous loop displays an error message until a .csv file name is given which exists in the file directory.**



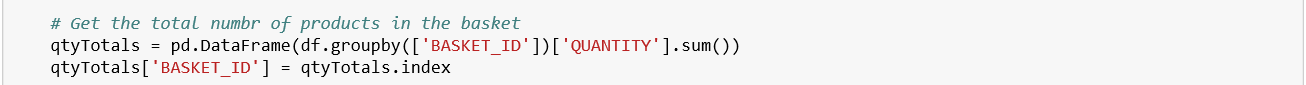
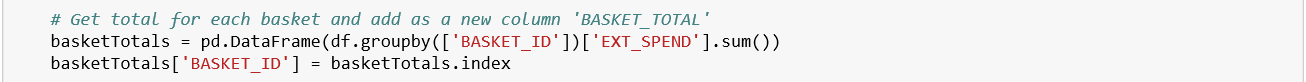
**manipData()**

This function addresses aggregation and pivoting of data. It relies on the *pandas* module and base python to conduct the tasks. The original dataset was ≈25k rows representing a single item each for any given shop. The shop might have one or many line items depending on how many different items were in the shopper’s basket. The code takes in the raw dataset and converts it into the proper format to predict ‘BASKET\_TOTAL’ spend.

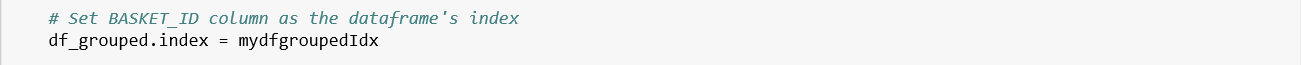
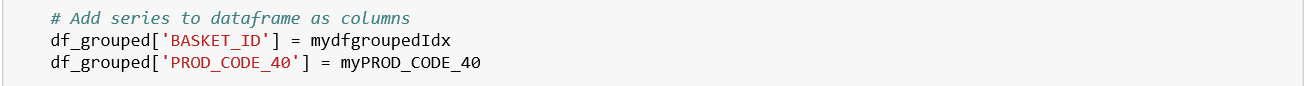
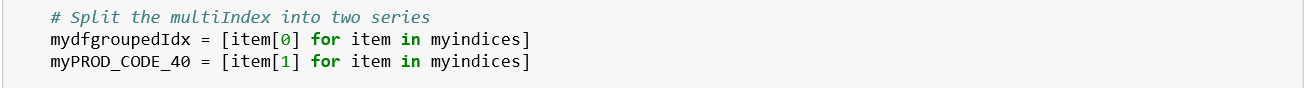
Original dataset:



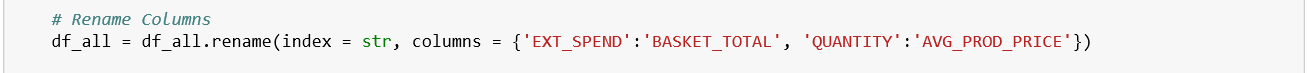
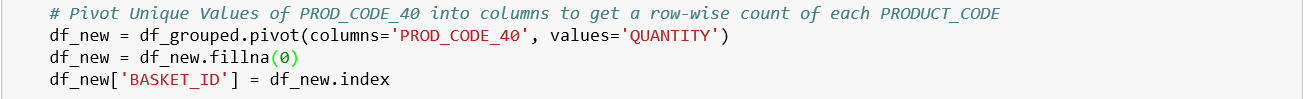
Conduct aggregation & manipulation because each line item could represent multiple of the same item. To get the actual total spend per line we needed to calculate this value . This was critical to convert the multiple rows of items into a single row representing the full shop.



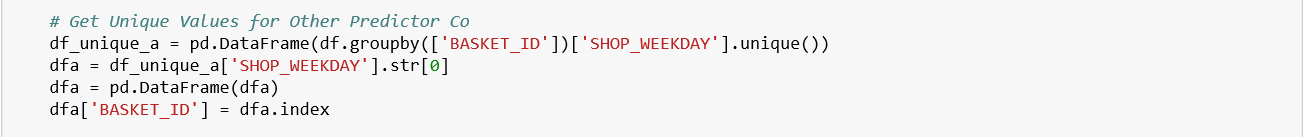
Splitting the multi-index from the aggregate by grouping of two different variables (two steps ago) represented somewhat of a challenge. The multi-index was converted to two different lists and added to the df\_grouped data frame to be able pivot by the PROD\_CODE\_40 values as columns.



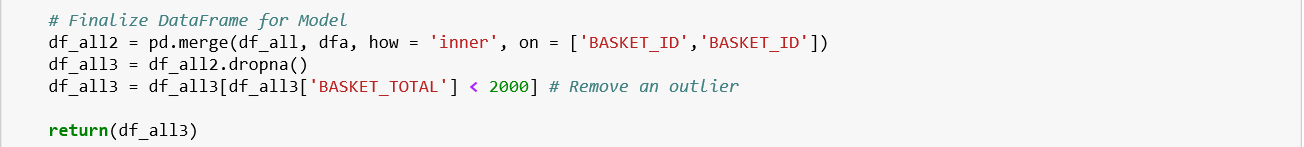
The *df.pivot* function was then used to convert each unique value of PROD\_CODE\_40 into a column, where each rows in the new dataframe is a single entry of ‘BASKET\_ID’ with the sum of the quantities of products in each PROD\_CODE\_40 category.

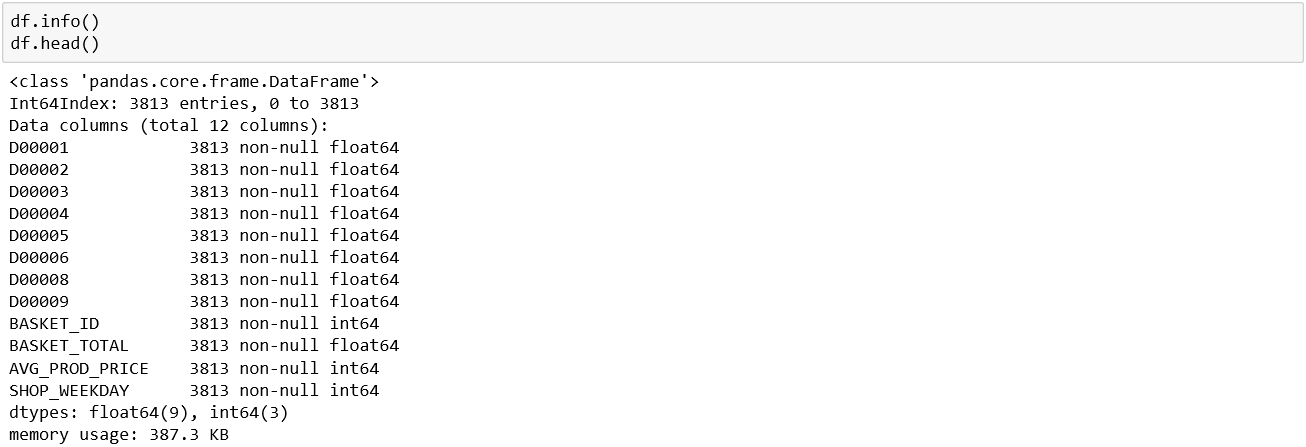


The last piece of data we choose to show in our dataset was the day of the week that the given shop happened on.

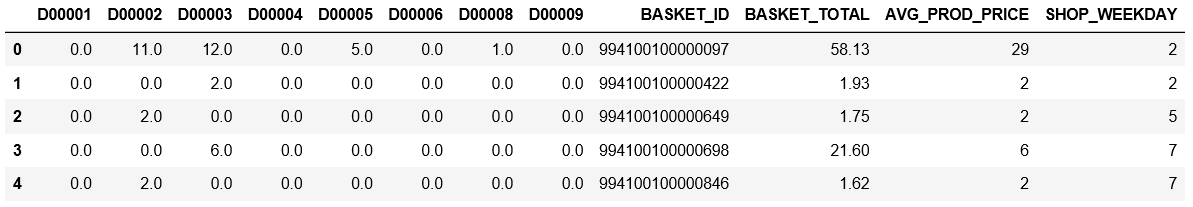


An outlier from a shopper going on a spending spree was removed.



The final dataset now represents a full shop record per line.

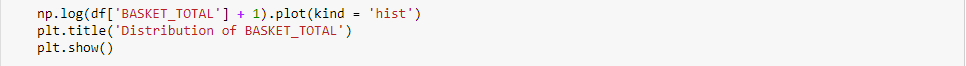
Final Dataset Structure



There were several items from the original dataset that we left out as they proved to not be predictive. The final dataset shown above is what we used to build our models from.

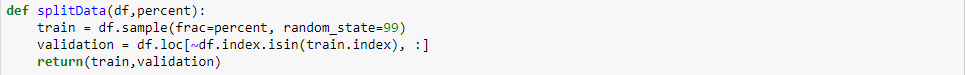
**plotData()**

**The plot data function create visualizations to aid in the feature selection process and the data transformation processes. It relies on the *matplotlib* and *seaborn (sns)* modules to create histograms and boxplots of our data. A few examples are shown below:**



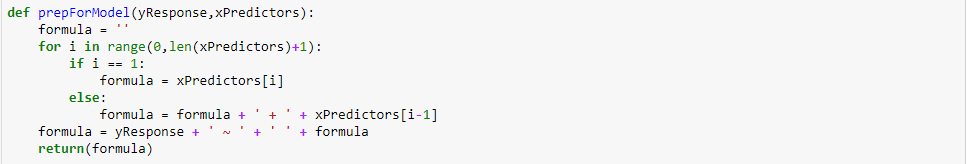
**splitData()**

**This function creates training and validation splits. It relies on the *sklearn.model\_selection* module which provided stratified data splits with a specified percent split. The results are stored against training and validation dataframes which go into building the model**



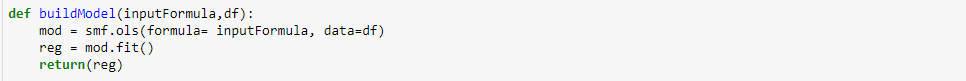
**prepForModel()**

**The prepForModel() function constructs a string based on a single string for the yResponse variable, and a list of strings for the xPredictors. It relies on base python code which implements the creation of a range, and the iteration through list elements to concatenate the string into the desired formula. The output can be passed to the buildModel() function.**



**buildModel()**

**One of the primary functions of the code is to build the model. This function allows the fitting of Ordinary Least Square regression to a given data frame where the model specification is provided in the form of a string expression, which is constructed by the prepForModel() function above. This function relies on the *statsmodels.formula.api* module to build and fit the model. It takes in the inputFormula and df parameters and returns a regressed object.**



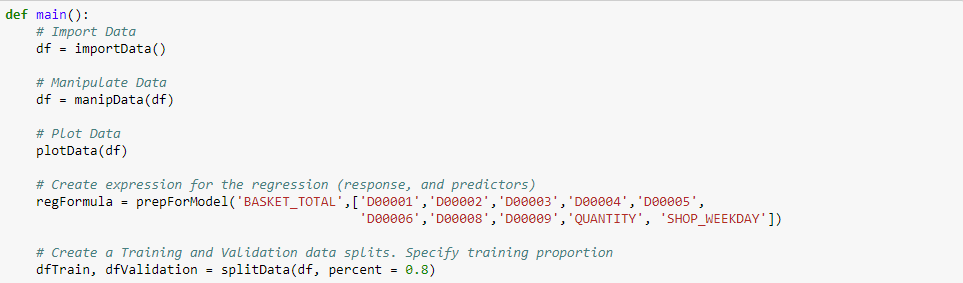
**outputPredictionFile()**

**This function uses the pandas methods *pd.DataFrame* and *df.to\_csv* to create a DataFrame from the array of predicted values coming from the series obtained from the output object of the buildModel() function. The function receives an output file name string, and the series of predicted value. It does not return an object but it places the new file in the working directory with the specified name.**



**main()**

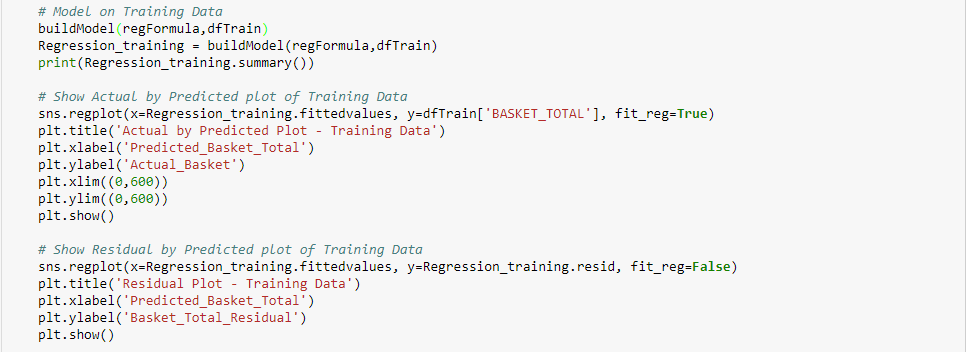
The main function is used to conduct the analysis. An advantage of having a main function calling the supplemental functions is that different iterations of the analysis can be run by changing a few input parameters rather than modifying the source code in each iteration. As an example, the analysis can be done with different dataset, using a different training/validation split, or using a different set of predictors simply by calling the supplemental functions with different arguments. To dissect the main() function, the first half of the function is described below:



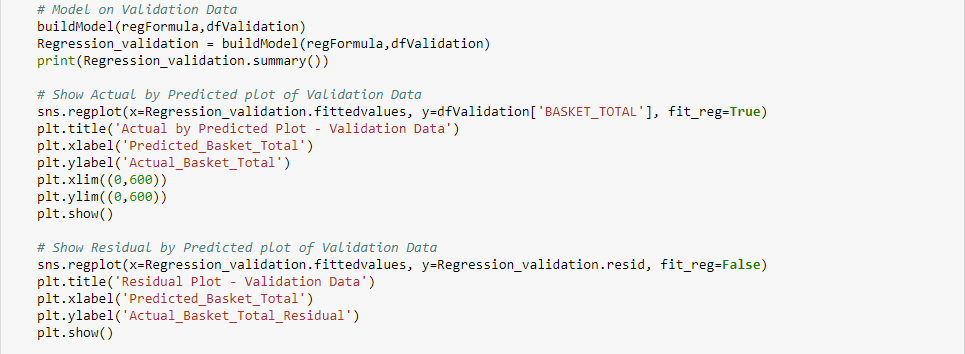
* The first part of the main function calls the **importData()** function to pull the desired data. The user must input a filename to process. This is the first argument that could be iterated on for the purposes of conducting different analysis.
* Then the data is manipulated by calling the **manipData()** function. The only argument to pass is the data frame. The manipulation logic is self-contained in the function and cannot be altered by any input argument to the function.
* The call to the **plotData()** function allows the user to visualize a certain set of parameters. The only argument to pass is the data frame. The manipulation logic is self-contained in the function and cannot be altered by any input argument to the function.
* The call to the **prepforModel()** function takes in the desired yResponse variable and a list xPredictors. The return string is the formula to be passed to the **buildModel()** function
* The **splitData()** function was

The second half of the main function is shown below

Training Data Model



Validation Data Model



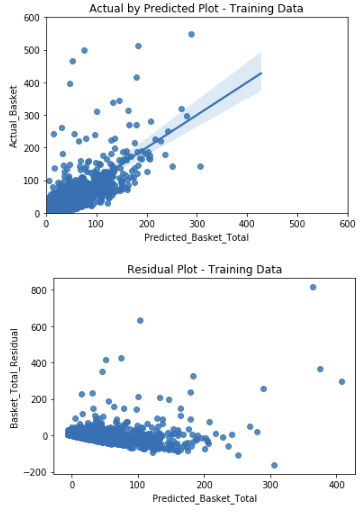
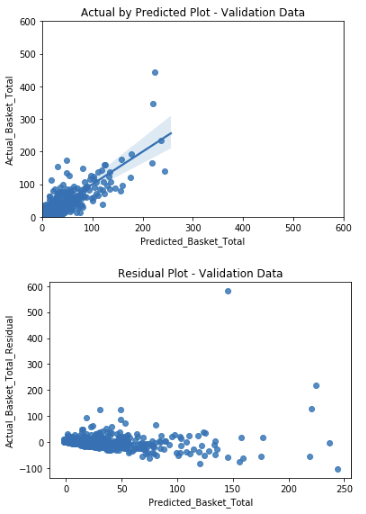
* The same series of function calls are conducted in the second half of the main() function. However, the first is done on the training set and the second is done on the validation set. The objective of conducing two separate models is to compare predictor p-values from both the training and validation dataset, as well as residuals to ensure that the model did not overfit
* A call to the **buildModel()** function is done first. It creates a fitted model object. Then, a summary of a parameter estimates is printed for the user to view statistical significance and coefficient estimates
* Plots of Actual by Predicted as well as Residual Plots are done by using seaborn plotting functions *sns.regplot* and these are not built into a function so as to give the user the liberty to plot predicted, actuals, and residuals as desired for the analysis.
* Plot formatting statements are utilized using *matplotlib.plt* functions



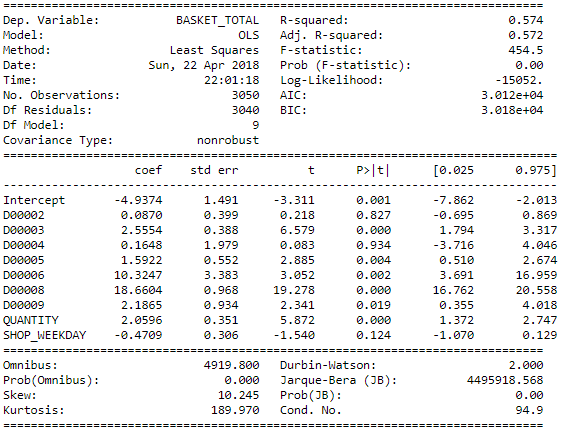
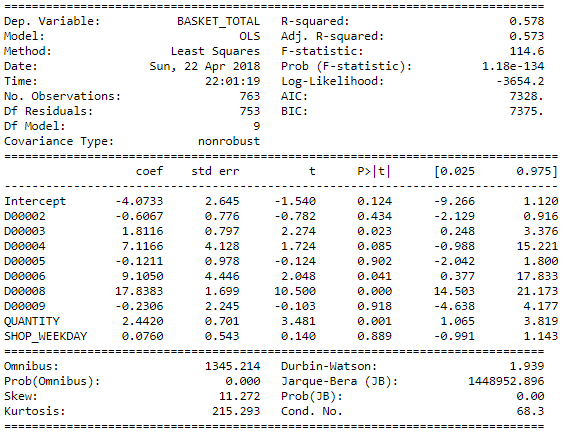
* Finally, the predicted output is saved to a file in the working directory by calling the outputPredictionFile() function and providing the ‘Basket\_Predictions.csv’ file, which is submitted along with this report.

# Statistical Output

Plots of the model output are show below for the training (left) and validation (right) sets. It is worth noting that the residual plots show non-constant variance as well as a mean which is slightly not constant. This does show some downfalls in the model constructed. It is also apparent that there are some outliers that could be accounted for with the use of dummy (binary) variables. One of the biggest challenges in this analysis was the lack of continuous variables. Just as the PROD\_CODE\_40 categorical variable was converted into continuous variables other categorical variables could have been transformed to help explain some of the variance. Additionally, the use of average price of product for each basket could have been a continouos variable that was helpful in the prediction of the BASKET\_TOTAL.

The parameter estimates and p-values for both the training (left) and validation (right) sets are shown below for comparison. Most of the parameters have on the training and validation sets show to be significant. However, there are some discrepancies in the p-values of both sets and coefficients of some of the predictors in both sets. This could be revealing of some outliers which fell into one set but not the other, and their leverage power was significant enough to sway the estimates. Further model refinement could be conducted but was deemed out of the scope of this project.

# Business Conclusions

Though the model built may not be used in prediction of BASKET\_TOTAL, it can be concluded that DEPT0006 and DEPT0008 have a heavy influence on BASKET\_TOTAL. The retailers could be advised to put emphasis on the purchase of products from these departments to boost store revenues.