MSBA Capstone

Maverik Team 1: Data Analysis Report

Time-Series Forecasting Candy Bar Sales
April 25, 2021

Sam Erickson | Tim Gain | Kelsey Metivier | Willem van der Schans

GitHub:
https://github.com/Kydoimos97/CapstoneMSBA2021
Dropbox:
https://www.dropbox.com/sh/i0n97zm0yexbg4f/AABNZB4SAILaxHla_i7kdLpDa?dl=0

Table of Contents

Executive Summary 6
- Overview 6
- Challenges 6
- Solutions 6
- Highlights 6

Introduction 8
- Project overview 8
- Business Problem 8
- Primary Objective 8
- Summary 9
- Outline 9

Data Preparation 10
- Overview 10
- Outliers 10
- Missing Data 11

Restrictions 12
- SQL Packages 12
- Computing Limitations 12

Implementation 13
- ARIMA 13
- Inputs 13
- Seasonality 13

Application 15
- Overview 15
- Fail-Safes 15

Results 17
- Inputs 17
- Global performance 17
- Extremity performance 17

Conclusion 19

Appendix 20

Appendix A 20
Appendix B 20
Appendix C 21
Appendix D 21

Executive Summary

Overview

Maverik would like to improve the sales forecasting model of candy bars at their convenience stores. The purpose of this Capstone data analysis is to create a time-series forecasting model for Maverik to accurately predict candy bar sales at each of its convenience store locations. As a result, company management can better optimize resources and increase profitability by predicting item-level candy bar sales at each of its convenience stores, projecting ten days into the future.

Challenges

Maverik provided almost three years of candy bar sales data containing over 1.3 million rows of data. The amount of data presented a few challenges:

1. Requiring granularity on a large matrix size caused computational limitations
2. Maverik SQL databases are only compatible with specific packages in R and Python
3. Data contained large outliers
4. Pricing data likely contains errors ([redacted])
5. [redacted] NA's in the minimum and maximum temperature fields
6. Seasonality and lag

Solutions

To overcome the issues above, we utilized the following solutions:

1. Sub-sampling loops provide granularity on data with more manageable sizes
2. Utilized Python for SQL compatibility and efficiency
3. Four standard deviations identified 12 extreme outliers, which were removed
4. Corrected pricing data
5. Interpolation to impute missing data points

Highlights

Our team created an application that allows Maverik management to generate a predictive model based on given inputs. The application's output produces the best out-of-sample RMSE error metric to show how accurate the model is and saves the result in CSV files for future reference. The application allows company management to customize forecasting depending on specific needs or queries. By accurately predicting specific candy bar sales at specific store locations, the company can optimize resources by ordering the optimal number of candy bars for each of its sites, thus improving company profitability.

Introduction

Project overview

Maverik is one of the most well-known names in gas stations and accompanying convenience stores throughout the intermountain region. Their website indicates they are located in "more than 350 locations across 11 western states, making it the largest independent fuel marketer in the Intermountain West"[1]. Although Maverik has been very successful with both its fuel and convenience store sales, the company would like to improve the sales forecasting model of candy bars sold in its convenience stores. This project seeks to provide a reliable time-series forecast that the company can use to predict the sales of specific candy bars within each of its stores.

Business Problem

Maverik requires reliable forecasting models for the inventory carried in its convenience store locations to place restocking orders and generate financial forecasts. Numerous factors affect individual stores' or groups of stores' calculations of these forecasting models, including location, size of the store, day of the week, holidays, weather, and many more variables. Maverik also has the challenge of determining whether they fulfill customers' needs by carrying the items that Maverik customers want. Accurate forecasting models allow the company to better allocate resources by ordering and maintaining the optimal quantity of each candy bar to increase company profitability.

Primary Objective

The primary objective of this analysis was to build a time-series forecasting model to predict candy bar sales at the item_id and site_id levels. Maverik provided over 1.3 million rows of historical sales data spanning from November 2017 to September 2020. The end goal of this project is to enable better management of candy bar inventory, pricing, budgeting, and forecasting.

[1] https://www.maverik.com/about-maverik/

Summary

While the initial business problem and objective seemed straightforward, the team encountered several obstacles illuminating the project's complexity. The size of the data set proved to be the main roadblock during the project's lifetime. Without access to the computing power necessary to perform a time-series model on the entire data set and the requirement to maintain granularity, we opted to use dynamically-generated data subsamples to feed to an ARIMA model. ARIMA is a time-series specific model where the algorithm's assumptions consider the unique attributes of time-series data, often producing more robust results than a simpler linear regression model would with less assumption violation. To make our model user-friendly, we built a stand-alone program with a GUI in an executable file that allows the end-user to perform predictions without having to interface with the code or install Python. The program focuses on transparency and saves all output in CSV for record-keeping and further analysis.

Outline

The report will first explore our data preparation, providing an overview and deep-dive into outliers and missing data. Next, the restrictions and limitations of the project will be discussed, followed by the implementation of our time-series model. Finally, the report will detail the application and our final product.

Data Preparation

Overview

The data set contained historical data points from a subset of Maverik's convenience stores from 11/08/2017 to 09/05/2020. There were 17 variables and 1.3 million rows of data. The data is separated by day per candy bar sold.

Outliers

[redacted]

The methodology used in determining and removing outliers in this data set was based on comparing the actual sales of each candy to the mean sales of that same candy bar. Any sales entries that were four or more standard deviations from the mean sales were considered an outlier and, therefore, removed from the data. In utilizing this approach, only the 12 most extreme items (0.0009% of the data) were removed from the analysis. We chose to use four standard deviations to account for possible discounts or price surges that allow more considerable differences between the mean price and actual daily sales of a specific Item and Site combination.

Additionally, an item-specific outlier was spotted; namely, Item_ID [redacted] $.10) appears as though it should cost $0.10 each based on the name of the product. However, there are several instances where the price exceeds $5.00 and even goes as high as $51.00, thus skewing the sales data for that item. To solve this issue, the price value of $0.10 was imputed for item_id -12066 to match the named description, making the data more accurate and reliable.

A comparison between the originally supplied data and the cleaned data is shown in figure 1. Note that the before visualization already included the price fix for item_id -12066

Missing Data

The 'maxtemp' and 'mintemp' variables were each missing [redacted] values. We could not pinpoint a single factor for why this temperature data was missing as the NA's were randomly scattered across dates and site ids. Early on in our analysis, we considered removing the temperature variables altogether; however, after consultation with Maverik, we learned the company plans to utilize forecasted weather data. The interpolate function from the Scipy package was used to impute the NA's. Interpolate uses a linear function and the nearest data points to impute the missing data points.

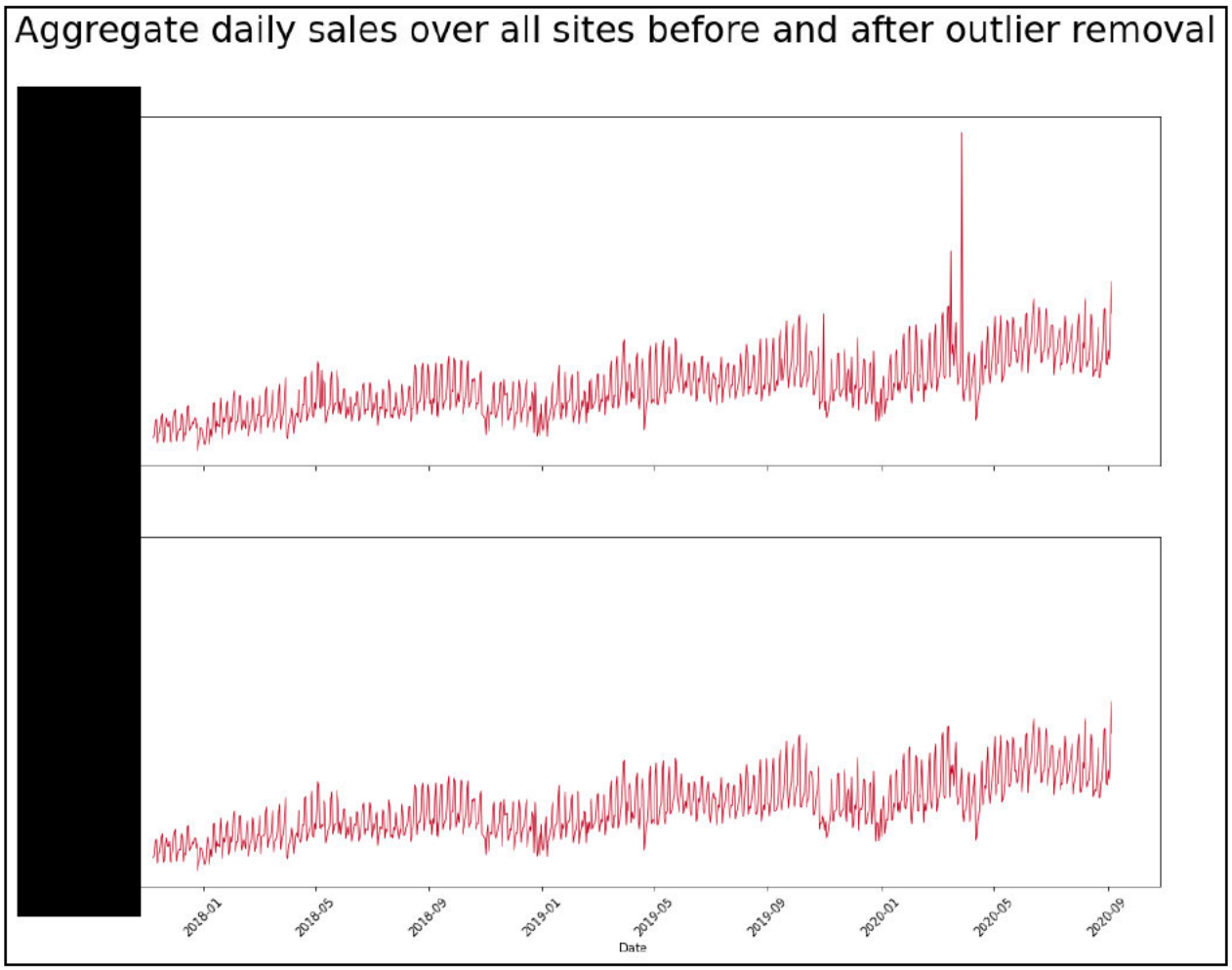


Figure 1: Original Data vs. Cleaned Data Comparison

Restrictions

SQL Packages

Maverik limited the use of packages in R or Python to those compatible with their SQL database. The team decided to use Python as there were more compatible packages available that allowed for time-series analysis. Using Python provided an additional challenge as collectively, the team was more experienced with R. Python was used to perform all data operations - cleaning, initial analysis, and time series analysis. Two groups of packages were used the essential packages that ensure the functionality of the code and the optional packages that merely provided quality of life improvements to the developed application. Application-specific uses can be found in the main code deliverable. The used packages are outlined below:

SQL Compatible (Essential)

- Pandas
- Numpy
- Statsmodels
- Scipy
- Sk-Learn
- Matplotlib
- Math
- Tkinter
- sys

Unknown Compatibility (Optional)

- Subprocess
- Threading

Computing Limitations

Computational power problems occurred throughout the project due to the size of the data matrix. As required by the project's scope, maintaining granularity in the data was crucial for producing predictions at the item level and eliminated the option of dimension reduction or aggregation of the data into a more manageable size. Subsampling loops were created to perform time-series predictions in a manner our computers were able to handle. Performing a time-series model on the entire data set without access to the necessary computing power would have taken several days, rendering this option infeasible.

Implementation

ARIMA

Our time-series analysis uses the ARIMA (autoregressive integrated moving average) model. ARIMA was explicitly built for time-series modeling, and the assumptions in the algorithm apply to the unique attributes of time-series data. The algorithm is versatile and capable of handling both univariate and multivariate data. ARIMA was critical for this project, as the computational power required is input-based, allowing for flexibility to adjust the input variables based on the computing power available and enabled the team to run models with limited RAM. However, it should be noted, that while computational power required is based on inputs, so is the performance of the model.

Inputs

ARIMA requires three inputs: p, d, and q.

p defines the number of autoregressive terms, which are the number of immediately preceding values in the series. These initial values are used to predict the value at present.

One stipulation of using ARIMA is that the data needs to be stationary, meaning the data should not show a trend of growth or decline, d is the number of nonseasonal differences required for the data to be stationary. With d being able to be grid searched, no adjustments have to be made to ensure stationarity within provided data to the ARIMA model allowing for simplification of feature engineering.

Finally, q defines what is taken into account for the Moving Average and is the number of lagged forecast errors in the prediction equation.

Our final application allows for grid searching of the ARIMA model, which selects the best p, d, and q for any given data set by iteratively analyzing RMSE based on a p,d, and q configuration grid provide the best results.

Seasonality

ARIMA assumes no or very little seasonality to be present in the provided data. Figure two shows a subsample of 4 item and site combinations. As shown, little seasonality occurs. Very few items, however, like the tootsie rolls, do show seasonality. SARIMAX and extension of ARIMA can deal with seasonality, but we did not find time to implement this approach yet.

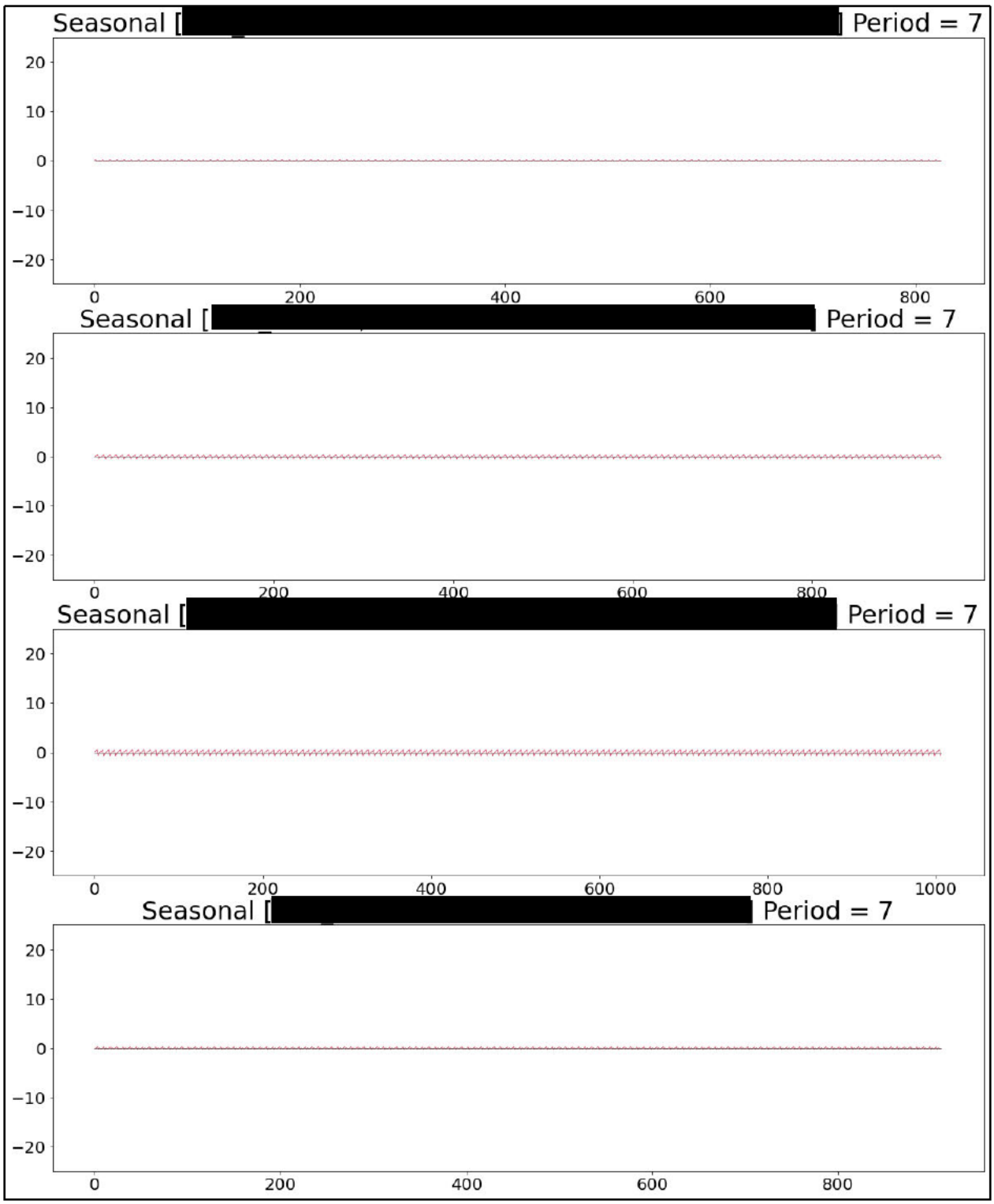


Figure 2: Seasonality of Sub Samples

Application

Overview

To solve Maverik's need to predict item-level candy bar sales at each of its convenience stores, our team's analysis resulted in creating an application which company management can use when placing new orders for candy bars and maintaining existing inventory. The application is customizable, allowing management to predict individual candy bar sales at each of its locations based on historical sales and other relevant time-series variables. The user may choose various factors, including sales forecasts for certain specific candy bar item numbers or forecasting the top N number of most-sold items. Depending on the need, the user may predict sales for all locations or drill down to a specific store number. The individual p, d, and q parameters of ARIMA models may be selected based on need. The user may also decide on how many days of data should be reviewed when building future predictions. As with many predictive models, using inputs that consider more data will often lead to a more accurate forecast, but the model will require a lot more time and machine processing power to run. Once all the input parameters have been entered, the option is given to save the data in a CSV file which may be referenced later if needed. The application's output will also display the forecast's Root Mean Square Error (RMSE) on the test set within the data to show how accurate the model predicts unknown future values. The GUI can be seen in figure three. Further information can be found in the documentation, which will be provided at a later date.

Fail-Safes

The application also has a few "fail-safes" built into it. Error message boxes are used to help direct the user to the correct usage of the application. For example, if the user inputs a site id that is not present in the dataset, the application will throw an error stating that "Model failed for all Item_ID and Site_ID combinations." Fail safes are essential to make sure the application does not need to be babysat while running. Stress testing the application has uncovered many potential breaking points, which are all accounted for. All errors are coerced to a CSV file regardless of the output option being selected as yes. CSV examples can be seen in Appendix B and C. Error message examples can be seen in Appendix D.

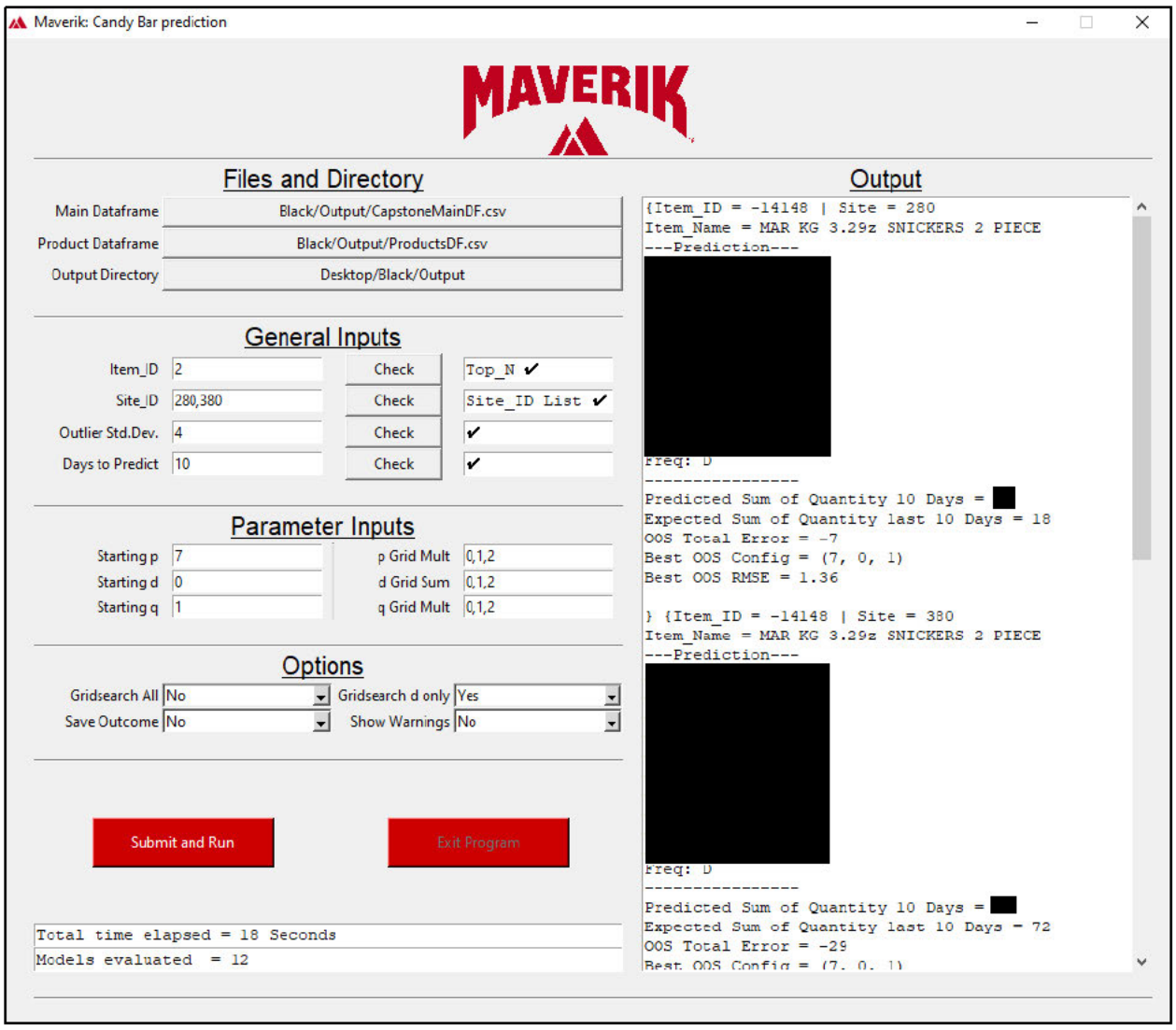


Figure 3: Application GUI after A completed prediction run

Results

Inputs

A stress test was completed with a list of fifty items overall sites included in the data set. The items were picked based on their sales data. Two critical points of sales were identified in which the program would behave differently; minimal sales and maximal sales. This prompted us to select the ten least and most sold items and add ten items surrounding the 1st quartile of sales, mean sales, and the 3rd quartile of sales. This list of fifty items can be found in appendix A.

The parameter inputs are set to default and were chosen to limit total runtime and ensure a prediction that was reflective of potential model performance. The input options can be found in appendix figure 4. As shown, grid searching is allowed, and the outcome is saved. Warnings are turned off, so the program doesn't require any user input while running.

Global performance

The program performed well without running into any unforeseen errors. 13500 Models were considered, and of those models, 10420 ran. One hundred sixteen combinations ran into errors causing not all considered models to be completed. Of these 116 errors, 94% were invalid combinations of site_id and item_id, 2% were lu decomposition errors, which is a bug with the stats models package, and 4% were historical data errors where not enough historical data was present to make a prediction. The total runtime of the program was 2 hours and 36 minutes with an average ram usage of 805Mb and 30% CPU utilization (6 core 3.70 GHz). The mean hold-out RMSE is 1.02 overall items, and with 0 RMSE predictions removed 2.13. The program predicted a total quantity of [redacted] to be sold in the next ten days over [redacted] in the last ten days. Note again that RMSE can be significantly improved by increasing the range of p and q grid searching; we didn't do this due to time restrictions on runtime.

Extremity performance

Both the least and most sold items are extremities that we want to further outline in this paper. First, the most sold items. This list of 10 considered 2700 models, and the program completed all of them, meaning no errors occurred. The mean hold-out RMSE is 3.22 per item. The total predicted quantity is [redacted] over [redacted] in the last ten days. The total runtime for this list was 41 minutes. The least sold item list also considered 2700 models, but only 216 were completed. The program predicted 0 sales for all items included and ran into 92 errors. All errors were handled correctly and are presented in an error CSV. The runtime for this section was only 3 minutes, showing the impact of efficiently coercing errors and moving on to the next item.

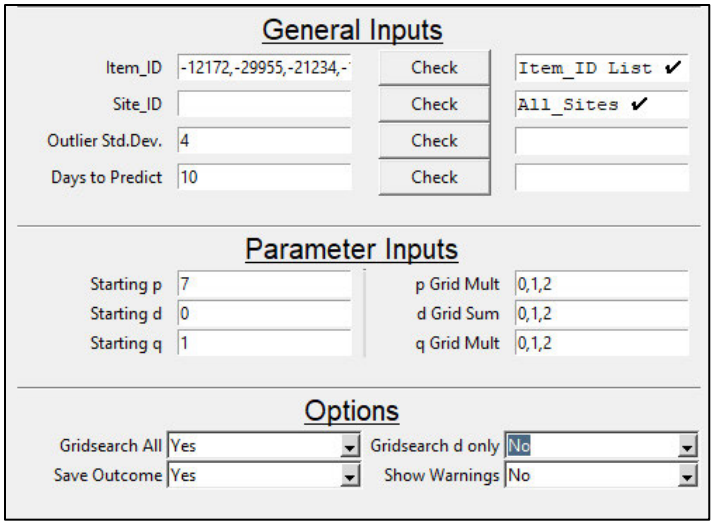


Figure 4: Stress Test Inputs

Conclusion

Maverik's initial need to forecast item-level candy bar sales can be satisfied by utilizing the predictive application created. Although potentially helpful, it was not practical to run predictions for all candy bar item numbers due to the amount of computing power required. There is a trade-off between utilizing enough historical data to produce an accurate forecast and the bottleneck of the machine's RAM to run those calculations.

Given the constraints of the packages compatible with Maverik's database and the size and scope of the data provided, we are confident the application created with the ARIMA algorithm will assist company management in determining the optimal item-level quantity of candy bars to stock in each store to maximize company profit. While some predictive models may be more general in nature, this application allows the user to customize the ARIMA model based on specific targets, allowing the degree of granularity and flexibility needed to address the initial business problem of forecasting specific item-level candy bar sales at each Maverik convenience store.

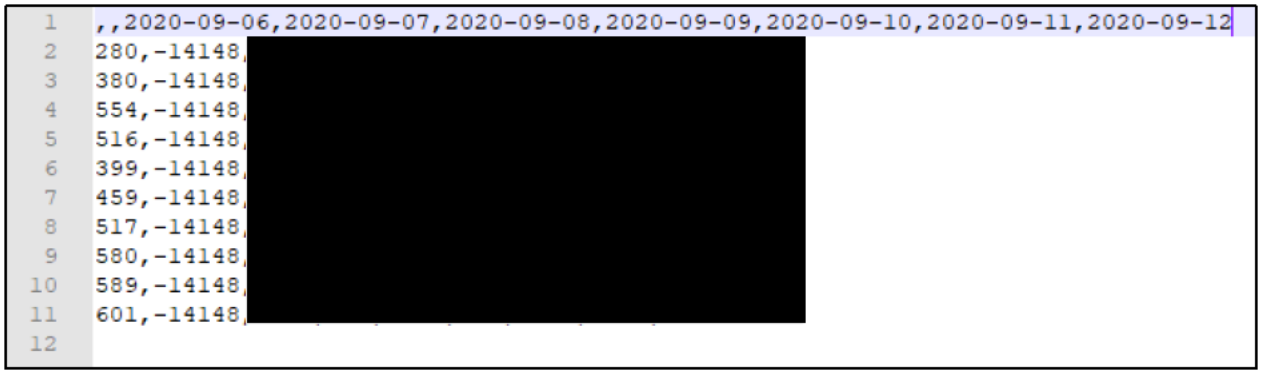
Appendix

Appendix A

Item Stress test list

Appendix B

sample sales prediction output for item_ID -14148 (MAR KG 3.29z SNICKERS 2 PIECE) for each store generated into CSV format for future reference



```
,,2020-09-06,2020-09-07,2020-09-08,2020-09-09,2020-09-10,2020-09-11,2020-09-12
280,-14148,
380,-14148,
554,-14148,
516,-14148,
399,-14148,
459,-14148,
517,-14148,
580,-14148,
589,-14148,
601,-14148,
```

Appendix C

sample error output log generated into CSV format for future reference

```
,0
"(-44763, 280)",Combination Does not Exist
"(-30755, 280)",Combination Does not Exist
"(-40507, 280)",Combination Does not Exist
"(-48395, 280)",Combination Does not Exist
"(-40508, 280)",Combination Does not Exist
"(-34071, 280)",Combination Does not Exist
"(-48453, 280)",Combination Does not Exist

```

Appendix D

Example Error Messages