MCDA5580 Assignment 5

Team Member

- Hemalatha Srinivasan A00452621
- Ajay Jain A00455849
- Kin Wa, Chan A00467755

Table of Content

Executive Summary 3

Introduction 4

Objectives 4

Data Analysis 5

- Data Extraction 5
- Data cleaning 7

Design/ Methodology/ Approach 11

- Overview 11
  - Models Used in Analysis 11
  - Training/ testing data sets 11
  - Sampling Method 11
  - Mean Absolute Percentage Error (MAPE) 11
- Linear Regression Analysis 12
- Neural Network Analysis 15
- Support Vector Machine (SVM) Analysis 24
- Holt Winters Analysis 35
- ARIMA Analysis 36
- Random Forest Analysis 38

Conclusion 41

- Comparison of all models used 41
- Recommendation and next step 43

Appendix 44

- SQL Script for Data Extraction 44
- R Script for Data Analysis 44
  - Linear Regression Analysis 44
  - Neural Network Analysis 52

SVM Analysis 66
HoltWinters, ARIMA and Random Forest Analysis 78
Reference/ Citation 86

Executive Summary

This report examines recent product sales data obtained from a retail store to identify potential strategies for improving sales performance. The study focuses on analyzing the sales quantity of the top consumed products and predict the next day sales with previous 7 days through using time series prediction methods. It is implemented by extracting daily sales from database and transposed to dataset with the each record contains the sales of 8 continuous days.

The main objective of this study was to find future trends from past data, forecasting future values, identifying patterns and trends, optimizing product stocking, and improving decision-making. The use of these predictive models can help businesses and organizations with inventory planning for the future and make informed decisions to optimize their operations and maximize profits.

The tested methods in this study include Linear Regression Analysis, Neural Network Analysis, Support Vector Machine (SVM) Analysis, HoltWinters Analysis, Autoregressive Integrated Moving Average (ARIMA) Analysis, and Random Forest Analysis. The best model was found to be the averaged neural network and ARIMA models with an error rate of ~30%.

The report recommends using the averaged neural network and ARIMA models for future prediction of sales quantity and maintaining inventory levels at the store appropriately. The report also recommends keeping track of any deviation in the model prediction vs actuals to adjust appropriately and update the model using the latest data.

Introduction

The purpose of this data analysis report is to examine recent product sales data obtained from a retail store and identify potential strategies for improving sales performance. Specifically, this study will focus on analyzing the sales quantity of the top consumed products. To accomplish this, we will employ time series prediction methods in forecasting, using the quantity sold in the past seven days to predict the next day's quantity. Our analysis will utilize statistical, neural network, and support vector regression techniques, and we will compare the performance of these models.

Objectives

The main objective of time series prediction is to find the future trends from the past data. The goal is to find a model which can accurately predict future value. The prediction will help in satisfying below purpose:

1. Forecasting future values: This is the major goal in using Time series predictions. Knowing the future results can increase business efficiency in decision making.
2. Identifying patterns and trends: We can find the patterns and trends using the past value which can be used for predicting the future trends and patterns.
3. Optimizing product stocking: This can help in predicting the best performing product. Using the obtained results, we can predict inventory levels for various products.
4. Improving decision-making: This will help in allocating the budget for the products involved in the analysis.

This can help businesses and organizations with inventory planning for the future and make informed decisions to optimize their operations and maximize their profits.

Data Analysis

Data Extraction

The sales data with the following fields between 2/1/2015 and 14/9/2015 are gathered for analysis (total 1328 records)

Field Name	Data Type	Description
date	Date	Sales Date
item_sk	String	Item No
quantity	Integer	Total no of sales the date

The quantities are aggregated by product and top 6 product are ranked out:

item_sk	total_quantity
11740941	90239
11740923	61482
11741274	28261
11636550	23839
11629829	22338
11743201	20716

Top 1st, 3rd, 5th (11740941, 11741274, 11629829) are selected for study.

For each item, the records are transposed for further analysis. Example:

Original record

date	item_sk	quantity
2/1/2015	11740941	430
3/1/2015	11740941	433
4/1/2015	11740941	305
5/1/2015	11740941	469
6/1/2015	11740941	533
7/1/2015	11740941	273
8/1/2015	11740941	279
9/1/2015	11740941	292
10/1/2015	11740941	399
11/1/2015	11740941	362

12/1/2015	11740941	315
13/1/2015	11740941	579
14/1/2015	11740941	364
15/1/2015	11740941	388
16/1/2015	11740941	383
17/1/2015	11740941	393

Transposed record

V1	V2	V3	V4	V5	V6	V7	V8
430	433	305	469	533	273	279	292
399	362	315	579	364	388	383	393

The study is going to use the quantities in previous 7 days (V1 – V7) to predict the quantity in the next day (V8).

Total 31 sets of transposed records are obtained for each product.

Data cleaning

Box plot analysis is used to figure out the outliners. Extreme outliners are removed:

- For 1st top item, removed the item with quantity = 3
- For 3rd top item, removed the item with quantity = 1
- For 1Sth top item, removed the item with quantity = 603

Box plot for 1st top item

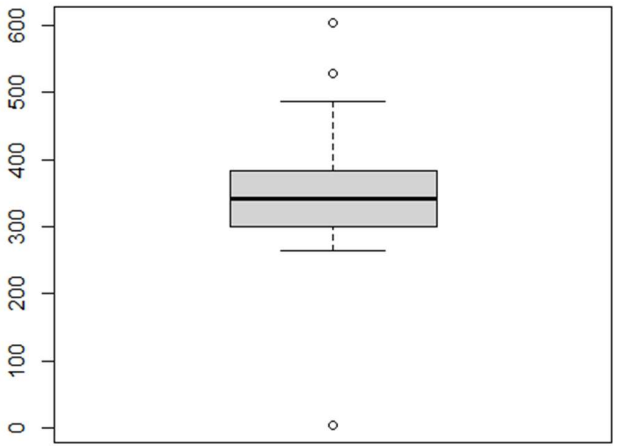

Figure 1

Box plot for 3rd top item

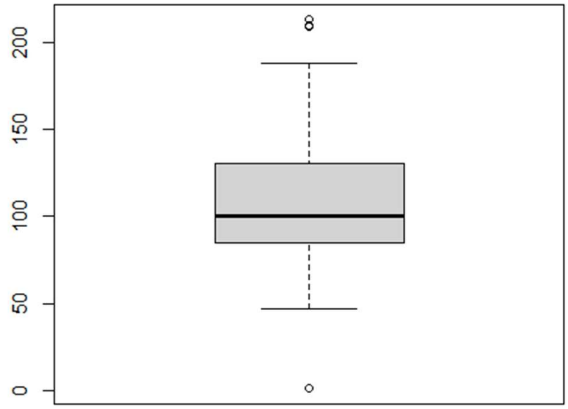

Figure 2

Box plot for 5th top item

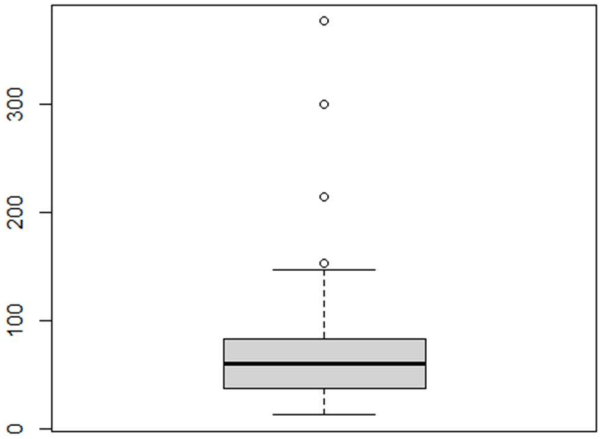

Figure 3

Plot as weekly time series (for three products respectively)

We can observe there are seasonal patterns, however the overall trends are not obvious.

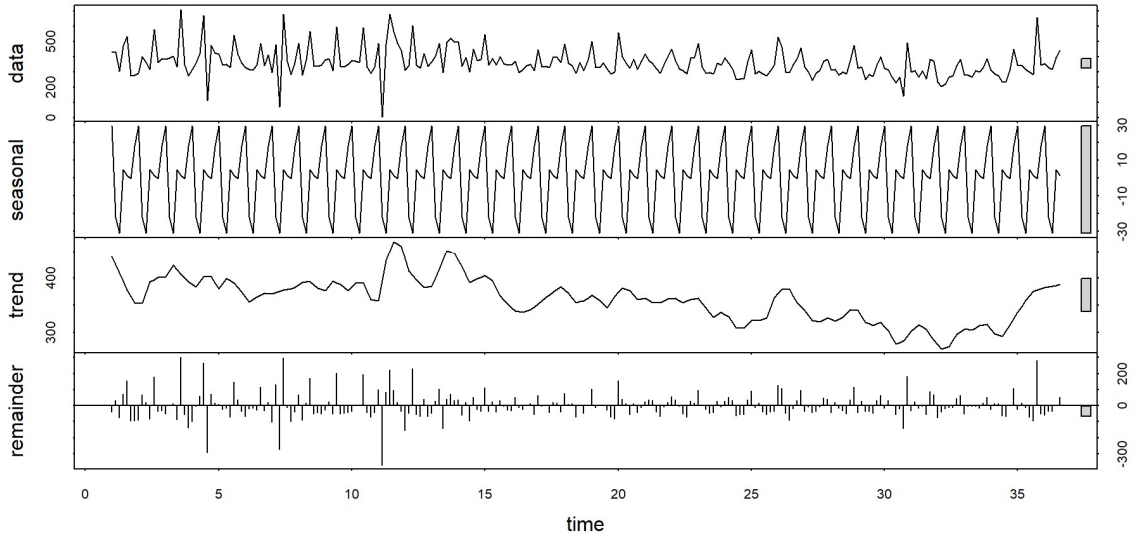

Figure 4

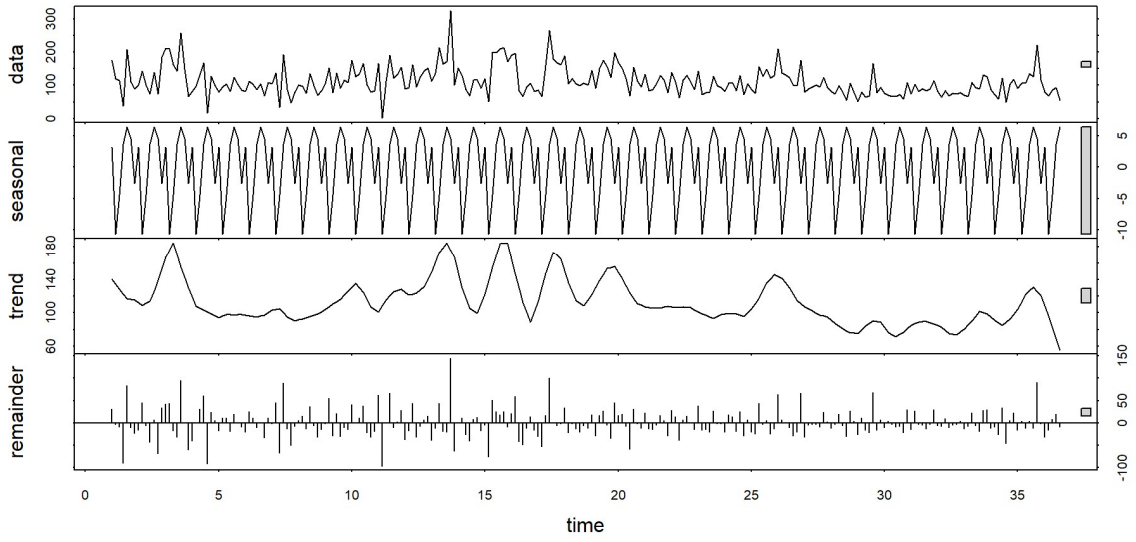

Figure 5

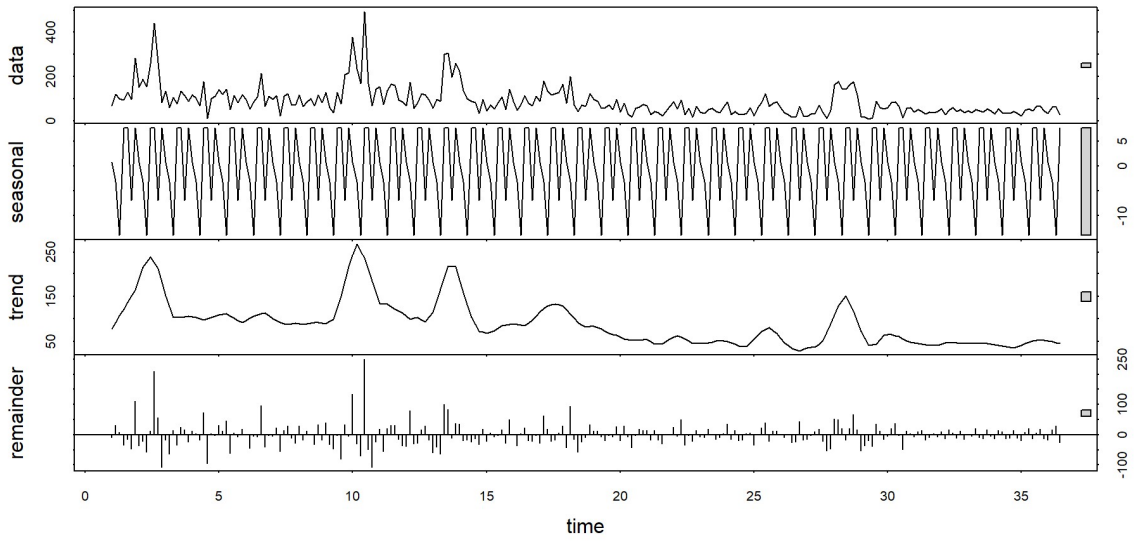
data
seasonal
trend
remainder
0
200
400
50
150
250
-10
0
5
-100
0
100
250
0
5
10
15
20
25
30
35
time

Figure 6

Design/ Methodology/ Approach

Overview

Models Used in Analysis

The following methods are tested in this study:

- Linear Regression Analysis
- Neural Network Analysis
- Support Vector Machine (SVM) Analysis
- HoltWinters Analysis
- Autoregressive Integrated Moving Average (ARIMA) Analysis
- Random Forest Analysis

Each model has its strengths and weaknesses and speciality on specific problems. Through the experiments with these models, we could better understand the nature of the data. In general speaking, Linear regression is simple and interpretable, neural networks and SVMs can handle complex relationships between variables, Holt-Winters and ARIMA are effective for time-series data, and Random Forest can handle both categorical and continuous data.

Caret Package (short for Classification And REgression Training) in R is used to implement these models. The package contains tools for data splitting, pre-processing, feature selection, model tuning using resampling, and variable importance estimation, all helps to streamline the process of creating predictive models.

Training/ testing data sets

To evaluate the performance of models, the studies split the data into training and testing datasets in an 80%/20% ratio.

Sampling Method

Repeated cross-validation (repeatedCV) is used to avoid overfitting and improve the generalizability of the model. Cross-validation is a technique used to evaluate the performance of a model by splitting the data into training and validation sets. Repeated cross-validation takes this a step further by repeating the process multiple times with different random splits of the data.

Mean Absolute Percentage Error (MAPE)

MAPE is used to evaluate the performance of predictions. It is calculated as the average absolute percent error for each time-period minus actual values divided by actual values. The lower value of MAPE means the model has a better accuracy in prediction.

Linear Regression Analysis

Linear regression is a statistical method that aims to discover linear relationship between predictors and dependent variable, it is used to make predictions with continuous outcomes. It is easy to implement and interpret. However, it is not capable of dealing with nonlinear relationship between variables.

Result summary

1st Item

```
> summary(glmFitTime)

Call:
NULL

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-164.620   -42.292    -0.005    45.095   131.736

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  351.667     15.027  23.402  1.8e-15 ***
V1             3.385     19.099   0.177    0.861
V2             3.888     17.015   0.228    0.822
V3            14.645     16.626   0.881    0.389
V4            -5.533     17.299  -0.320    0.753
V5           -10.380     17.526  -0.592    0.561
V6            10.828     17.289   0.626    0.539
V7            27.416     19.023   1.441    0.166
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 6097.067)

    Null deviance: 160218  on 26  degrees of freedom
Residual deviance: 115844  on 19  degrees of freedom
AIC: 320.46
```

3rd Top item

```
> summary(glmFitTime)

Call:
NULL

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-49.443  -19.106   -9.702   15.997   75.577

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  109.704      6.642  16.518 9.99e-13 ***
V1             7.858      7.633   1.029    0.316
V2             5.752      8.669   0.663    0.515
V3            -4.781      8.060  -0.593    0.560
V4           -14.374     10.381  -1.385    0.182
V5            -4.475      7.778  -0.575    0.572
V6             7.554      9.430   0.801    0.433
V7            26.064      9.741   2.676    0.015 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 1191.014)

    Null deviance: 38446  on 26  degrees of freedom
Residual deviance: 22629  on 19  degrees of freedom
AIC: 276.36

Number of Fisher Scoring iterations: 2
```

5th Top item

```
> summary(glmFitTime)

Call:
NULL

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-58.661  -22.065  -13.948    3.842  189.033

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   78.074     11.435   6.828 1.62e-06 ***
V1            -1.602     21.331  -0.075    0.941
V2            26.200     20.896   1.254    0.225
V3           -20.331     22.031  -0.923    0.368
V4            -7.051     29.779  -0.237    0.815
V5            -3.294     27.560  -0.120    0.906
V6            24.181     19.625   1.232    0.233
V7            12.770     22.087   0.578    0.570
---
Signif. codes:  0  '***'  0.001  '**'  0.01  '*'  0.05  '.'  0.1  ' '  1

(Dispersion parameter for gaussian family taken to be 3530.286)

    Null deviance: 101130  on 26  degrees of freedom
Residual deviance:  67075  on 19  degrees of freedom
AIC: 305.7

Number of Fisher Scoring iterations: 2
```

We can observe the P value of the predictors are not statistically significant (>0.05) except the V7 predictor in 3rd top item. It shows linear regression is not an appropriate model for the data set.

MAPE Summary

MAPE trends to be high especially for the training set. It may be another evidence that shows linear regression is not a suitable model.

Item	Training	Testing
1st Top Item	16.91%	23.78%
3rd Top Item	23.43%	28.53%
5th Top Item	44.80%	172.90%
Average	**28.38%**	**75.07%**

Neural Network Analysis

Neural networks are a type of machine learning model that is inspired by the structure and function of the human brain. They consist of interconnected nodes or neurons that work together to learn patterns in the input data and make predictions or decisions based on that learning.

The advantages include being able to model nonlinear relationships, high accuracy, and high fault tolerance to noisy data. However, it had the drawback of black box nature (relationship learned is difficult to explain), high computational power requirement, and require large amount of data for training.

There are two main parameters control capacity of Neural networks model:

1. Number of nodes: Increasing the number of nodes in a single layer can improve the model's ability to capture complex relationships between the input and output.
2. Number of layers: Increasing the number of layers not only increases the capacity of the model, but also enables it to capture nonlinear relationship.

Increasing these parameters could improve the ability to recognize features but it causes overfitting problem.

Neural networks with different configurations are tested in this analysis:

1. Neural network with single layer, number of nodes = 15
2. Neural network with 3 layers, 5 nodes in each layer
3. Averaged Neural Network – It trains multiple neural network models and then averaging their predictions to produce the final output. The main advantage is to overcome the overfitting problem of normal neural network but resulting more computationally expensive.

Result summary

Neural network with single layer, number of nodes = 15

1st Item

Neural network Plot

It demonstrates the single layer neural network architecture and its parameters

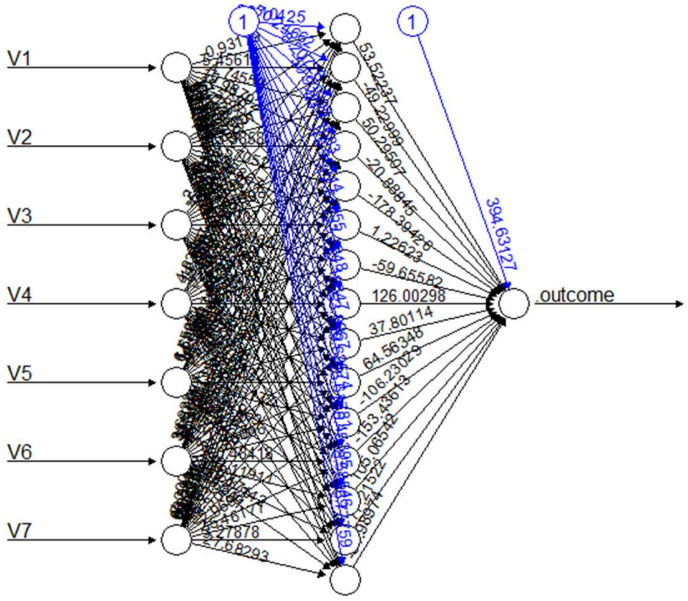

Figure 7

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
                  Length Class      Mode
call                7    -none-     call
response           27    -none-     numeric
covariate         189    -none-     numeric
model.list          2    -none-     list
err.fct             1    -none-     function
act.fct             1    -none-     function
linear.output       1    -none-     logical
data                8    data.frame list
exclude             0    -none-     NULL
net.result          1    -none-     list
weights             1    -none-     list
generalized.weights 1    -none-     list
startweights        1    -none-     list
result.matrix     139    -none-     numeric
xNames              7    -none-     character
problemType         1    -none-     character
tuneValue           3    data.frame list
obsLevels           1    -none-     logical
param               3    -none-     list
```

3rd Top item

Neural network Plot

It demonstrates the single layer neural network architecture and its parameters.

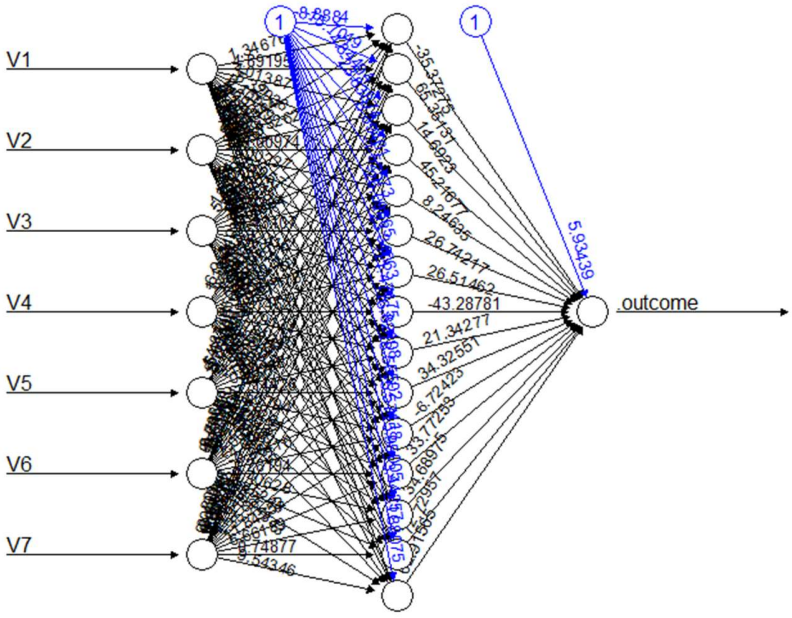

Figure 8

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
                  Length Class      Mode
call                 8   -none-     call
response            27   -none-     numeric
covariate          189   -none-     numeric
model.list           2   -none-     list
err.fct              1   -none-     function
act.fct              1   -none-     function
linear.output        1   -none-     logical
data                 8   data.frame list
exclude              0   -none-     NULL
net.result           1   -none-     list
weights              1   -none-     list
generalized.weights  1   -none-     list
startweights         1   -none-     list
result.matrix      139   -none-     numeric
xNames               7   -none-     character
problemType          1   -none-     character
tuneValue            3   data.frame list
obsLevels            1   -none-     logical
param                4   -none-     list
```

5th Top item

Neural network Plot

It demonstrates the single layer neural network architecture and its parameters

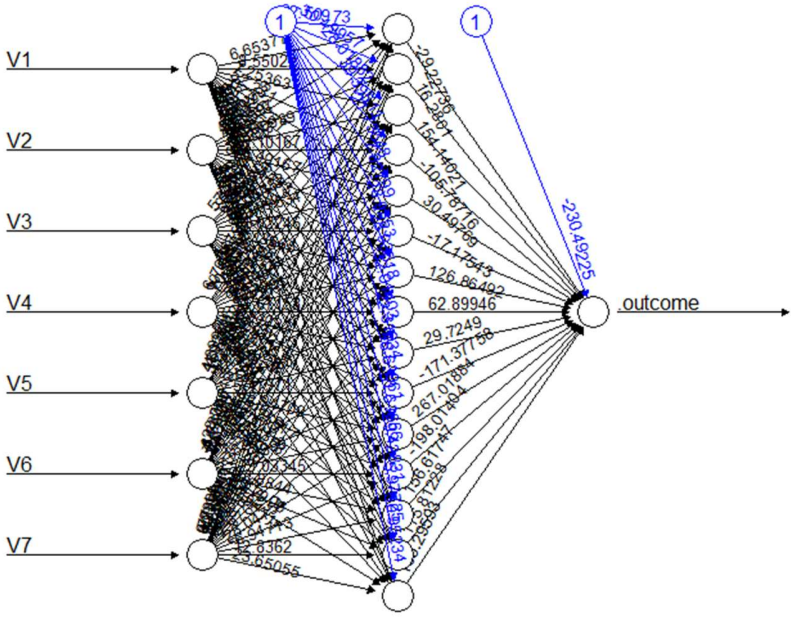

Figure 9

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
                  Length Class      Mode
call                7    -none-     call
response           27    -none-     numeric
covariate         189    -none-     numeric
model.list          2    -none-     list
err.fct             1    -none-     function
act.fct             1    -none-     function
linear.output       1    -none-     logical
data                8    data.frame list
exclude             0    -none-     NULL
net.result          1    -none-     list
weights             1    -none-     list
generalized.weights 1    -none-     list
startweights        1    -none-     list
result.matrix     139    -none-     numeric
xNames              7    -none-     character
problemType         1    -none-     character
tuneValue           3    data.frame list
obsLevels           1    -none-     logical
param               3    -none-     list
```

Neural network with 3 layers, 5 nodes in each layer

1st Item

Neural network Plot

It demonstrates the 3 layers neural network architecture and their parameters.

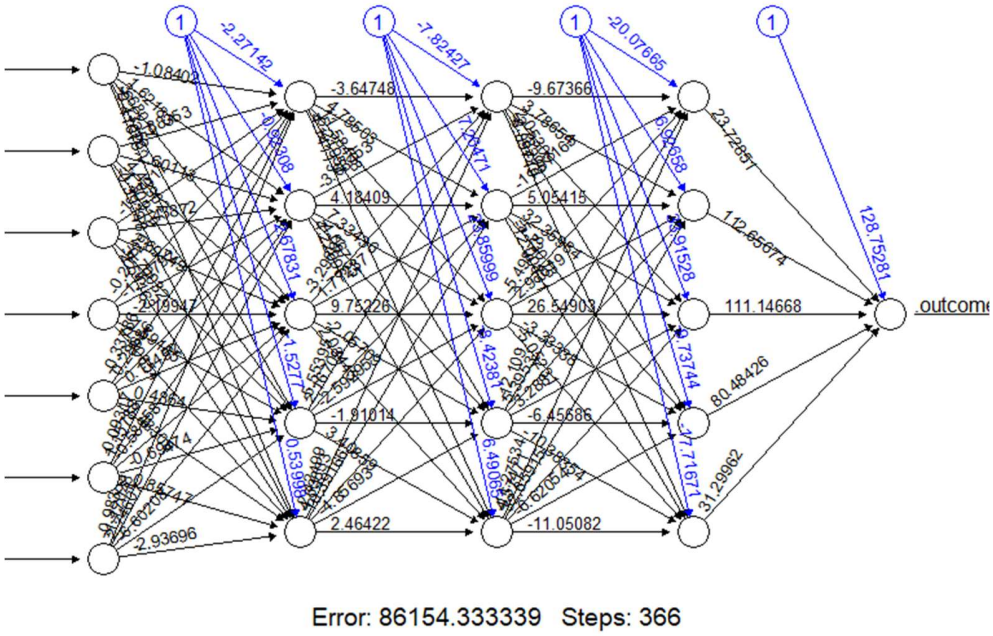

Figure 10

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
                  Length Class      Mode
call                7    -none-     call
response           27    -none-     numeric
covariate         189    -none-     numeric
model.list          2    -none-     list
err.fct             1    -none-     function
act.fct             1    -none-     function
linear.output       1    -none-     logical
data                8    data.frame list
exclude             0    -none-     NULL
net.result          1    -none-     list
weights             1    -none-     list
generalized.weights 1    -none-     list
startweights        1    -none-     list
result.matrix     109    -none-     numeric
xNames              7    -none-     character
problemType         1    -none-     character
tuneValue           3    data.frame list
obsLevels           1    -none-     logical
param               3    -none-     list
```

3rd Top item

Neural network Plot

It demonstrates the 3 layers neural network architecture and their parameters.

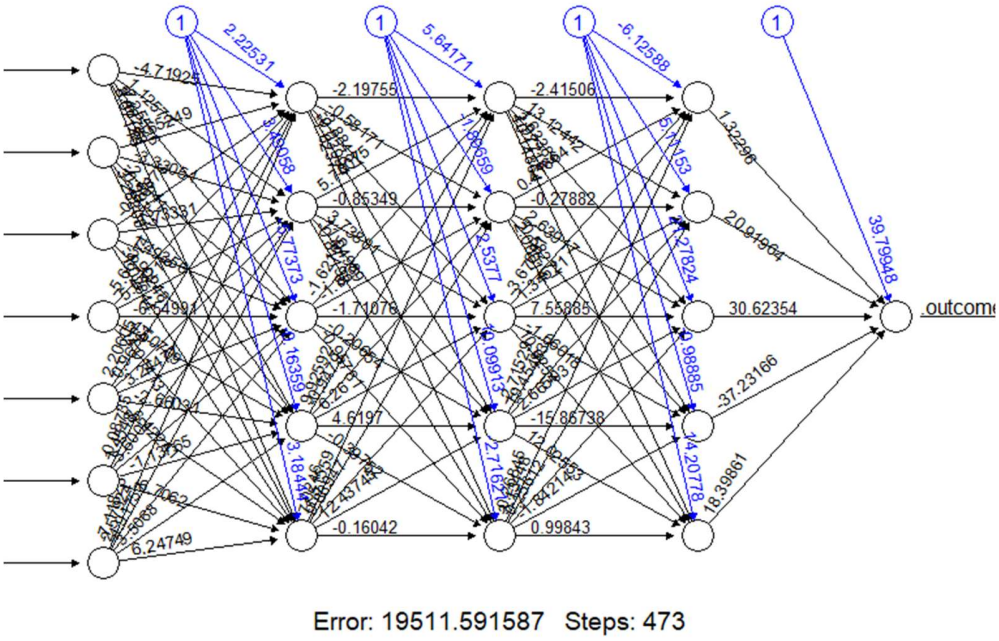

Figure 11

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
                  Length Class      Mode
call                7    -none-     call
response           27    -none-     numeric
covariate         189    -none-     numeric
model.list          2    -none-     list
err.fct             1    -none-     function
act.fct             1    -none-     function
linear.output       1    -none-     logical
data                8    data.frame list
exclude             0    -none-     NULL
net.result          1    -none-     list
weights             1    -none-     list
generalized.weights 1    -none-     list
startweights        1    -none-     list
result.matrix     109    -none-     numeric
xNames              7    -none-     character
problemType         1    -none-     character
tuneValue           3    data.frame list
obsLevels           1    -none-     logical
param               3    -none-     list
```

5th Top item

Neural network Plot

It demonstrates the 3 layers neural network architecture and their parameters.

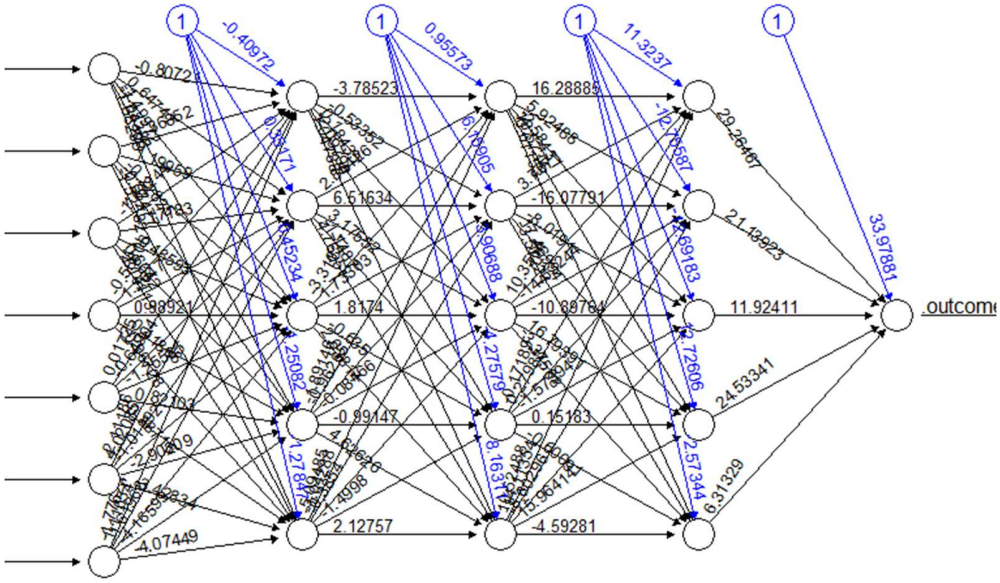

Figure 12

Model summary

```
> summary(nnFitTime)
                  Length Class      Mode
call                7    -none-     call
response           27    -none-     numeric
covariate         189    -none-     numeric
model.list          2    -none-     list
err.fct             1    -none-     function
act.fct             1    -none-     function
linear.output       1    -none-     logical
data                8    data.frame list
exclude             0    -none-     NULL
net.result          1    -none-     list
weights             1    -none-     list
generalized.weights 1    -none-     list
startweights        1    -none-     list
result.matrix     109    -none-     numeric
xNames              7    -none-     character
problemType         1    -none-     character
tuneValue           3    data.frame list
obsLevels           1    -none-     logical
param               3    -none-     list
```

Averaged Neural Network

1st Item

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
            Length Class      Mode
model       5      -none-     list
repeats     1      -none-     numeric
bag         1      -none-     logical
seeds       5      -none-     numeric
names       7      -none-     character
terms       3      terms      call
coefnames   7      -none-     character
xlevels     0      -none-     list
xNames      7      -none-     character
problemType 1      -none-     character
tuneValue   3      data.frame list
obsLevels   1      -none-     logical
param       4      -none-     list
```

3rd Top item

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
            Length Class      Mode
model       5      -none-     list
repeats     1      -none-     numeric
bag         1      -none-     logical
seeds       5      -none-     numeric
names       7      -none-     character
terms       3      terms      call
coefnames   7      -none-     character
xlevels     0      -none-     list
xNames      7      -none-     character
problemType 1      -none-     character
tuneValue   3      data.frame list
obsLevels   1      -none-     logical
param       4      -none-     list
```

5th Top item

Model summary

It shows the model configuration used.

```
> summary(nnFitTime)
           Length Class      Mode
model      5      -none-     list
repeats    1      -none-     numeric
bag        1      -none-     logical
seeds      5      -none-     numeric
names      7      -none-     character
terms      3      terms      call
coefnames  7      -none-     character
xlevels    0      -none-     list
xNames     7      -none-     character
problemType 1     -none-     character
tuneValue  3      data.frame list
obsLevels  1      -none-     logical
param      4      -none-     list
```

MAPE Summary

We can see the 1 layer neural had a very lowest MAPE value in training set, but it is unstable that also shows the highest value among all configurations in its testing set result. The 3 layers neural network and averaged neural network had similar performance between their training and testing sets. However, the averaged neural network shows lower MAPEs compare with those of 3 layers neural network, therefore it is the best performed model.

Neural network with single layer, number of nodes = 15

Item	Training	Testing
1st Top Item	10.35%	27.76%
3rd Top Item	9.35%	68.94%
5th Top Item	17.18%	97.36%
Average	12.29%	64.69%

Neural network with 3 layers, 5 nodes in each layer

Item	Training	Testing
1st Top Item	19.91%	16.16%
3rd Top Item	21.31%	58.21%
5th Top Item	93.81%	68.23%
Average	45.01%	47.53%

Averaged Neural Network

Configuration	Item	Training	Testing
size = 3 (Autotune)	1st Top Item	13.90%	24.74%
size = 1 (Autotune)	3rd Top Item	22.40%	53.19%
size = 1 (Autotune)	5th Top Item	55.48%	25.39%
	Average	30.59%	34.44%

Support Vector Machine (SVM) Analysis

Support Vector Machine (SVM), a form of supervised learning algorithm, can perform classification or regression tasks. The central concept is to identify a hyperplane that effectively separates the distinct classes present in the training data. SVM shared some similarity with neural network. They could be able to handle high dimensional data and the result models are difficult to explain. However, it requires lesser computation power, lesser sensitive overfitting, more suitable for smaller and less complex data sets.

It allows use of different kernels in modelling. The kernel is a function measuring the similarity of two data points. Changing kernel function allows us to cate different data patterns.

SVM with different kernels will be tested in the analysis:

1. Radial kernel – It is the most used kernel in SVM which is a flexible method that can capture complex nonlinear relationships between the predictor variables and the response variable.
2. Polynomial kernel – It could also model nonlinear relationships but for less complex data only.
3. Linear kernel – It is suitable when the relationship between the predictor variables and the response variable is linear.
4. Sigmoid kernel - It is like a two-layer perceptron model of the neural network.

Result Summary

Radial kernel

1st Top item

Model summary

It shows the model configuration used and some performance measurements.

```
Support Vector Machines with Radial Basis Function Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 24, 25, 24, 25, 24, 24, ...
Resampling results across tuning parameters:

  C       RMSE      Rsquared   MAE
    0.25  64.08148  0.7975307  53.88688
    0.50  62.42463  0.7610670  51.97124
    1.00  65.37114  0.7222749  55.19914
    2.00  71.44978  0.6975062  60.53483
    4.00  78.35225  0.6853026  66.24954
    8.00  82.13271  0.6843033  69.74372
   16.00  82.05953  0.6852691  69.68440
   32.00  82.05953  0.6852691  69.68440
   64.00  82.05953  0.6852691  69.68440
  128.00  82.05953  0.6852691  69.68440

Tuning parameter 'sigma' was held constant at a value of 0.2024405
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.2024405 and C = 0.5.
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

3rd Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Radial Basis Function Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 25, 24, 24, 24, 24, 25, ...
Resampling results across tuning parameters:

  C       RMSE      Rsquared   MAE
    0.25  32.63416  0.7574235  27.93539
    0.50  31.11401  0.7345328  26.30532
    1.00  31.81240  0.7166902  26.97604
    2.00  31.50752  0.7167079  27.08650
    4.00  33.19105  0.7237418  28.28989
    8.00  37.28125  0.7124632  31.47990
   16.00  39.46992  0.7136572  33.63473
   32.00  39.71876  0.7236006  34.02679
   64.00  38.92351  0.6666740  33.16580
  128.00  38.60918  0.6671279  32.86900

Tuning parameter 'sigma' was held constant at a value of 0.07200923
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.07200923 and C = 0.5.
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

5th Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Radial Basis Function Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 24, 25, 24, 25, 24, 24, ...
Resampling results across tuning parameters:

  C       RMSE      Rsquared   MAE
    0.25  48.82893  0.7129794  38.67265
    0.50  48.75766  0.7402870  39.05445
    1.00  49.78422  0.7329748  40.60900
    2.00  52.34226  0.7338731  44.19451
    4.00  53.97358  0.7348476  46.50983
    8.00  53.97358  0.7348476  46.50983
   16.00  53.97358  0.7348476  46.50983
   32.00  53.97358  0.7348476  46.50983
   64.00  53.97358  0.7348476  46.50983
  128.00  53.97358  0.7348476  46.50983

Tuning parameter 'sigma' was held constant at a value of 1.025851
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 1.025851 and C = 0.5.
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

Linear kernel

1st Top item

Model summary

It shows the model configuration used and some performance measurements.

```
Support Vector Machines with Linear Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 24, 24, 25, 24, 24, 25, ...
Resampling results:

  RMSE      Rsquared   MAE
  82.45617  0.6387506  69.15861

Tuning parameter 'C' was held constant at a value of 1
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

3rd Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Linear Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 25, 24, 24, 24, 24, 24, ...
Resampling results:

  RMSE      Rsquared   MAE
  40.66145  0.6177963  34.83248

Tuning parameter 'C' was held constant at a value of 1
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

5th Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Linear Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 24, 24, 25, 24, 24, 25, ...
Resampling results:

  RMSE     Rsquared   MAE
  54.3962  0.7154952  43.76852

Tuning parameter 'C' was held constant at a value of 1
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

Polynomial kernel

1st Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Polynomial Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 24, 25, 25, 24, 24, 24, ...
Resampling results across tuning parameters:

  C      degree  RMSE       Rsquared   MAE
  1e-01  1        73.55824  0.6147493   63.76561
  1e-01  2        73.46478  0.6151560   63.64232
  1e-01  3        73.37252  0.6140579   63.53394
  1e-01  4        73.27987  0.6127780   63.42548
  1e-01  5        73.18732  0.6107181   63.31717
  1e-01  6        73.09354  0.6120832   63.20859
  1e+00  1        72.76204  0.6201522   62.89869
  1e+00  2        72.10869  0.6332848   62.44691
  1e+00  3        71.71738  0.6441611   62.18239
  1e+00  4        71.69037  0.6254789   62.27158
  1e+00  5        71.62639  0.6219910   62.33192
  1e+00  6        71.40476  0.6234699   62.23827
  1e+01  1        70.96112  0.6215037   61.90183
  1e+01  2        71.99497  0.6187099   62.71736
  1e+01  3        73.43456  0.6061303   63.89202
  1e+01  4        73.97552  0.6194540   64.33366
  1e+01  5        74.38728  0.6210145   64.64864
  1e+01  6        74.82973  0.6211369   64.88486
  1e+02  1        77.55029  0.5959197   66.56543
  1e+02  2        80.61461  0.6062223   67.83912
  1e+02  3        82.36920  0.5928625   69.45635
  1e+02  4        83.16996  0.5970504   69.97326
  1e+02  5        82.74938  0.5727291   69.31515
  1e+02  6        81.47318  0.5659279   67.93159
  1e+03  1        88.06281  0.5616421   74.49083
  1e+03  2        85.39427  0.5585207   71.58462
  1e+03  3        75.79175  0.5980777   63.40067
  1e+03  4        69.63759  0.6264520   59.42871
  1e+03  5        70.17160  0.6510630   60.92572
  1e+03  6        73.53432  0.6728755   64.71354
  1e+04  1        89.28412  0.5642811   75.48799
  1e+04  2        70.33507  0.6487463   61.02457
  1e+04  3        85.68248  0.6594849   77.84804
  1e+04  4       104.13878  0.6251877   95.07928
  1e+04  5       126.59597  0.5931050  113.57099
  1e+04  6       151.53478  0.5943769  133.57521

Tuning parameter 'scale' was held constant at a value of 0.001
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were degree = 4, scale = 0.001 and C = 1000.
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

3rd Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Polynomial Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 24, 24, 25, 24, 24, 24, ...
Resampling results across tuning parameters:

  C      degree  RMSE      Rsquared   MAE
  1e-01  1       35.19608  0.7208977  30.23723
  1e-01  2       35.15046  0.7209920  30.18742
  1e-01  3       35.10822  0.7210888  30.14117
  1e-01  4       35.07399  0.7188189  30.10235
  1e-01  5       35.04627  0.7183507  30.06829
  1e-01  6       35.01966  0.7222145  30.03816
  1e+00  1       34.94991  0.7126496  29.96176
  1e+00  2       34.65707  0.7050013  29.73213
  1e+00  3       34.24467  0.7152903  29.46053
  1e+00  4       33.86468  0.7202941  29.21025
  1e+00  5       33.69649  0.7178549  29.13168
  1e+00  6       33.66267  0.7139571  29.14174
  1e+01  1       33.45751  0.7229232  29.04526
  1e+01  2       31.12152  0.7282326  26.71747
  1e+01  3       29.81660  0.7387301  25.12709
  1e+01  4       29.25251  0.7536692  24.47846
  1e+01  5       29.05237  0.7603414  24.18877
  1e+01  6       29.21704  0.7561434  24.27790
  1e+02  1       30.41701  0.7306747  25.35022
  1e+02  2       33.41891  0.7006489  28.00263
  1e+02  3       35.00559  0.6856153  29.43440
  1e+02  4       35.93058  0.6812804  30.30483
  1e+02  5       36.40875  0.6800142  30.81734
  1e+02  6       36.61398  0.6815856  31.12266
  1e+03  1       38.37440  0.6649887  32.66915
  1e+03  2       39.10605  0.6646026  33.48628
  1e+03  3       37.35864  0.6854311  31.91964
  1e+03  4       35.09394  0.6951529  29.45974
  1e+03  5       34.54099  0.6844530  28.87964
  1e+03  6       36.71960  0.6968385  30.39085
  1e+04  1       41.01972  0.6568932  34.92762
  1e+04  2       34.70197  0.6851100  29.01608
  1e+04  3       44.66433  0.7105190  35.50840
  1e+04  4       55.19882  0.7152357  42.51253
  1e+04  5       62.22965  0.6932210  48.18695
  1e+04  6       64.53151  0.6920106  50.47946

Tuning parameter 'scale' was held constant at a value of 0.001
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were degree = 5, scale = 0.001 and C = 10.
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

5th Top item

Model summary

It shows the model configuration used and some performance measurements.

```
> svmFitTime
Support Vector Machines with Polynomial Kernel

27 samples
 7 predictor

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 25, 24, 25, 24, 25, 24, ...
Resampling results across tuning parameters:

  C      degree  RMSE      Rsquared   MAE
  1e-01  1       53.51092  0.6711496  40.43352
  1e-01  2       53.30223  0.6710852  40.24886
  1e-01  3       53.10234  0.6701733  40.06634
  1e-01  4       52.88030  0.6702767  39.86685
  1e-01  5       52.64639  0.6702039  39.67303
  1e-01  6       52.42760  0.6696172  39.48958
  1e+00  1       51.88695  0.6712816  38.88837
  1e+00  2       51.04189  0.6765926  38.05875
  1e+00  3       50.50220  0.6812487  37.55065
  1e+00  4       50.30587  0.6805641  37.47978
  1e+00  5       50.09255  0.6800060  37.42031
  1e+00  6       49.83100  0.6812503  37.18393
  1e+01  1       48.54773  0.6865618  35.89431
  1e+01  2       47.73407  0.6873564  35.43876
  1e+01  3       46.71474  0.6811025  34.75657
  1e+01  4       46.64791  0.6766222  35.00931
  1e+01  5       46.75303  0.6968584  35.27321
  1e+01  6       47.02294  0.7044783  35.78149
  1e+02  1       47.40532  0.7019156  36.59658
  1e+02  2       47.90805  0.6795584  37.98176
  1e+02  3       48.66916  0.6748717  39.02141
  1e+02  4       49.23215  0.6606780  39.55231
  1e+02  5       50.12300  0.6523057  40.33334
  1e+02  6       50.92408  0.6497880  41.05963
  1e+03  1       51.76276  0.6587156  41.55344
  1e+03  2       54.82264  0.6531382  44.27845
  1e+03  3       55.84916  0.6523686  44.88988
  1e+03  4       56.09791  0.6626589  44.69503
  1e+03  5       56.99596  0.6450441  45.48808
  1e+03  6       56.38439  0.6377818  45.41998
  1e+04  1       56.07393  0.6564929  45.19943
  1e+04  2       58.79531  0.6417996  46.99251
  1e+04  3       56.25816  0.6462018  45.44232
  1e+04  4       59.89173  0.6276660  48.44658
  1e+04  5       70.90026  0.5944395  55.64933
  1e+04  6       74.57747  0.5924811  57.92365

Tuning parameter 'scale' was held constant at a value of 0.001
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were degree = 4, scale = 0.001 and C = 10.
> summary(svmFitTime)
Length  Class   Mode
     1   ksvm     S4
```

Sigmoid kernel

1st Top item

Model summary

It shows the model configuration used.

```
> svmFitTime

Call:
svm(formula = V8 ~ ., data = train, kernek = "sigmoid")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1428571
    epsilon:  0.1


Number of Support Vectors:  27
```

3rd Top item

Model summary

It shows the model configuration used.

```
> svmFitTime

Call:
svm(formula = V8 ~ ., data = train, kernek = "sigmoid")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1428571
    epsilon:  0.1


Number of Support Vectors:  23
```

5th Top item

Model summary

It shows the model configuration used.

```
> svmFitTime

Call:
svm(formula = V8 ~ ., data = train, kernek = "sigmoid")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1428571
    epsilon:  0.1


Number of Support Vectors:  21
```

MAPE Summary

In general, the average MAPE of training data sets is much lower than the testing data sets where the ratio for Radial and Sigmoid models is roughly 1:4 and those for Linear and Polynomial are roughly 1:2. It implies Linear and Polynomial models have more stable performance in recognizing the features in data (They have better performance even average MAPE from training and testing sets). It is an interesting finding because they are relatively simple models.

Radial kernel

Item	Training	Testing
1st Top Item	11.72%	18.27%
3rd Top Item	18.06%	40.82%
5th Top Item	23.20%	145.20%
Average	17.66%	68.10%

Linear kernel

Item	Training	Testing
1st Top Item	14.89%	20.88%
3rd Top Item	19.41%	51.38%
5th Top Item	40.52%	66.68%
Average	24.94%	46.31%

Polynomial kernel

Configuration	Item	Training	Testing
method = svmPoly , degree = 4 (Autotune)	1st Top Item	10.73%	18.60%
method = svmPoly , degree = 5 (Autotune)	3rd Top Item	19.46%	45.95%
method = svmPoly , degree = 4 (Autotune)	5th Top Item	39.54%	78.42%
	Average	23.24%	47.66%

Sigmoid kernel

Item	Training	Testing
1st Top Item	10.36%	18.26%
3rd Top Item	13.86%	42.10%
5th Top Item	25.25%	164.32%
Average	16.49%	74.90%

Holt Winters Analysis

This method applies triple exponential smoothing for level, trend, and seasonal components of time series. It is a predictive forecasting model where the future is predicted by historical data and its trend. Its components can be controlled by parameters. Alpha represents level smoothing; beta specifies trend smoothing and gamma represents seasonality. Seasonality can be classified as additive and multiplicative. In additive seasonality where change is by constant number and multiplicative seasonality changes by a constant factor.

We used tuning grid to optimize the model and select the parameters using metric of MAPE. Also, used ets() function to build the holt winters model.

MAPE Summary

The averaged MAPE is moderate (34.96%) compared to other models. However, the variations between items are quite high (18.1% to 55.09%). It implies the model may not serve as a generalized method to analyze different products.

Item	Training
1st Top Item	18.10
3rd Top Item	31.69
5th Top Item	55.09
Average	34.96

ARIMA Analysis

ARIMA stands for AutoRegressive Integrated Moving Average. It is a statistical model for analyzing and forecasting time series data. Below are the key components of this model:

AR (Auto Regression) – It describes the relationship between an observation and some number of past observations.

I (Integrated) – It uses difference between current and one previous time step in order to make it stationary.

MA (Moving Average) – It uses the relationship between an observation and its residual error with moving averages of lagged observations.

These components can be controlled in model by parameters p, d and q respectively. We used a tuning grid to tune these parameters in our modeling.

Arima and Holt winters time series models had similar performance with arima being better with MAPE of 33% while holt winters had 34%. We see from the plots that some outliers have impacted the performance of these models which is expected in these.

Arima & Holt-Winters Charts

Below are plots of actual data vs prediction from Arima and Holt Winters for three products respectively

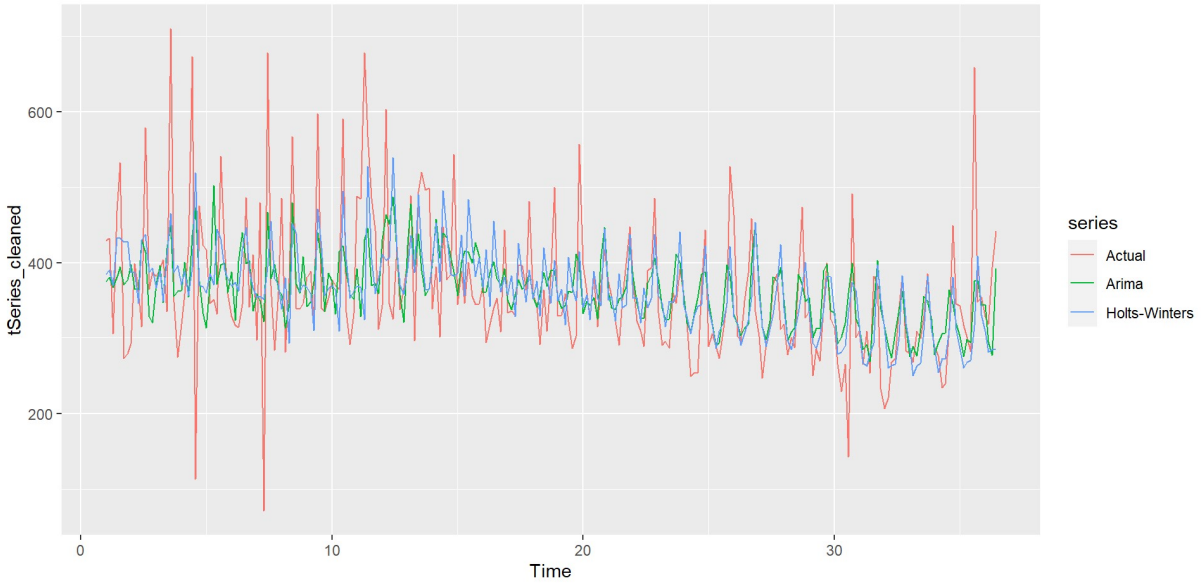

Figure 13

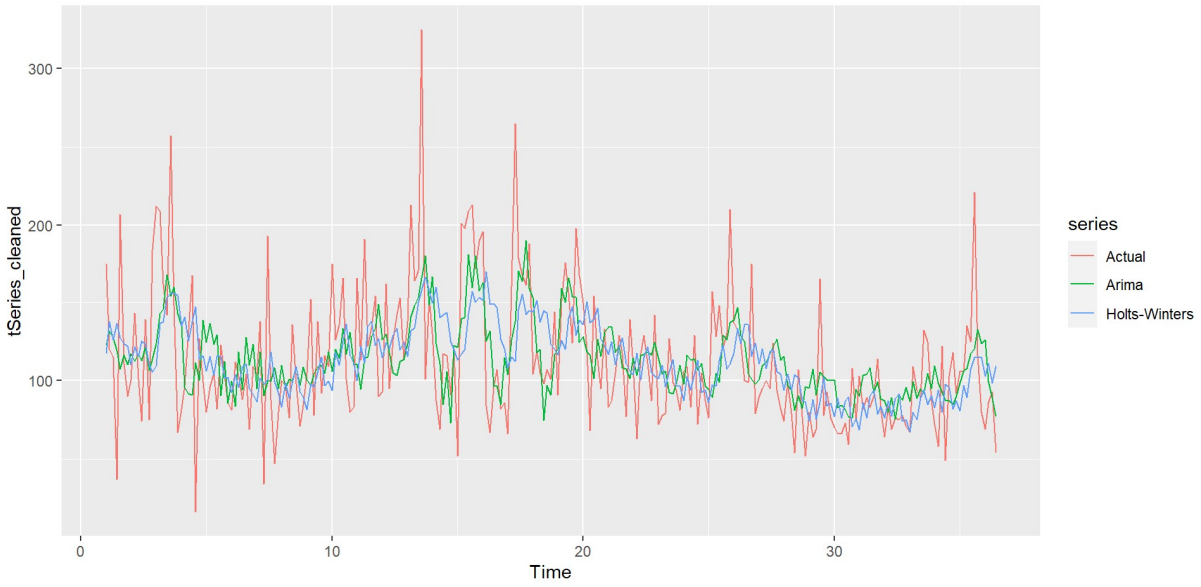

Figure 14

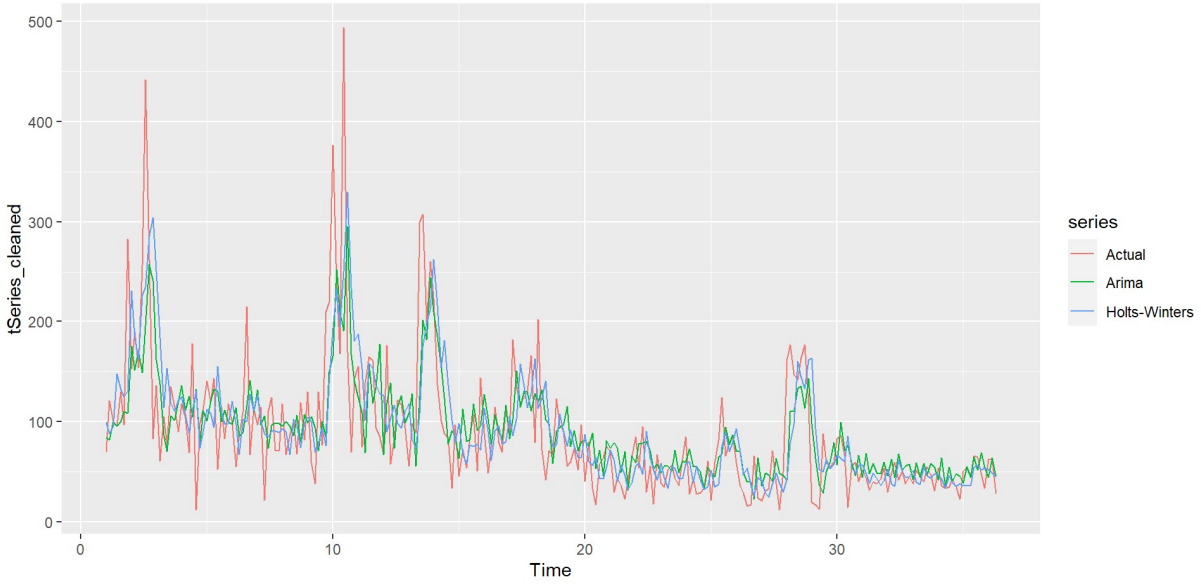

Figure 15

MAPE Summary

The characteristics is similar to the figures in Holt Winters where it also has moderate level (33.02%) of average MAPE and high variations among products (17.53% to 52.31%). It means it also not suitable to work a generalized solution in prediction.

Item	Training
1st Top Item	17.53%
3rd Top Item	29.21%
5th Top Item	52.31%
Average	33.02%

Random Forest Analysis

Random forest is a predictive model for classification and regression. It can also be used for time series forecasting after transforming time series into supervised learning problem. For this purpose, we transformed the time series data into 8 columns with 7 independent and one dependent variables. This was just to see how this model works on the time series for assignment purpose only.

Below are plots of actual (red line) vs predicted (blue line) for three products respectively. We see similar kinds of charts in all three products but there are large differences in peaks and valleys which results in large errors.

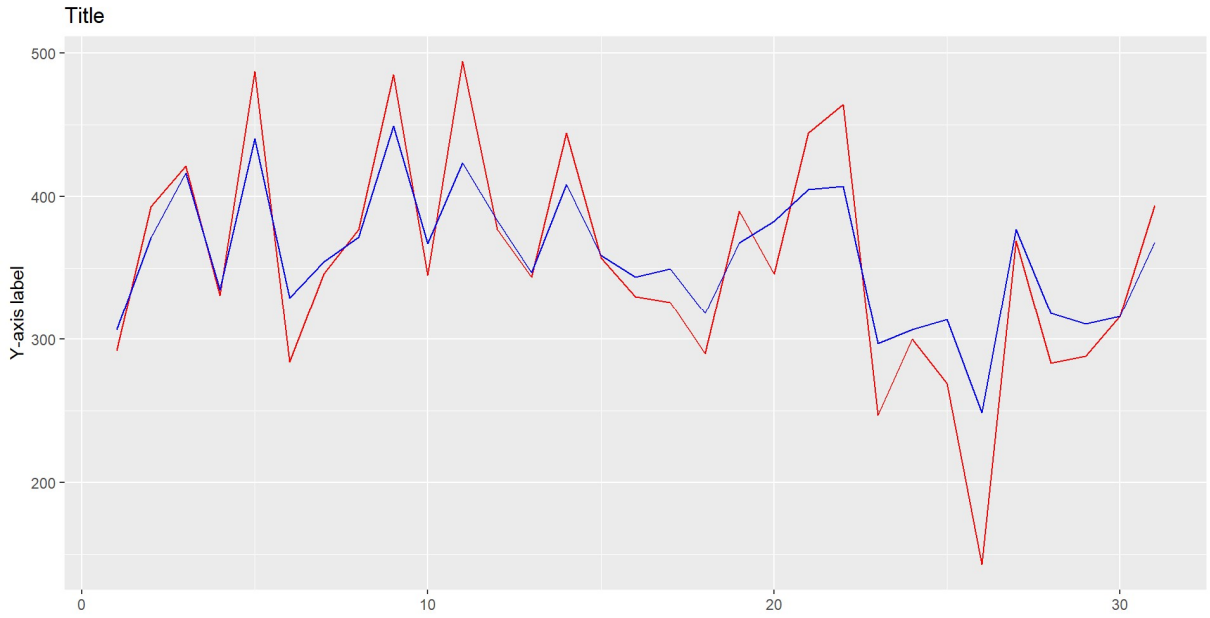

Figure 16

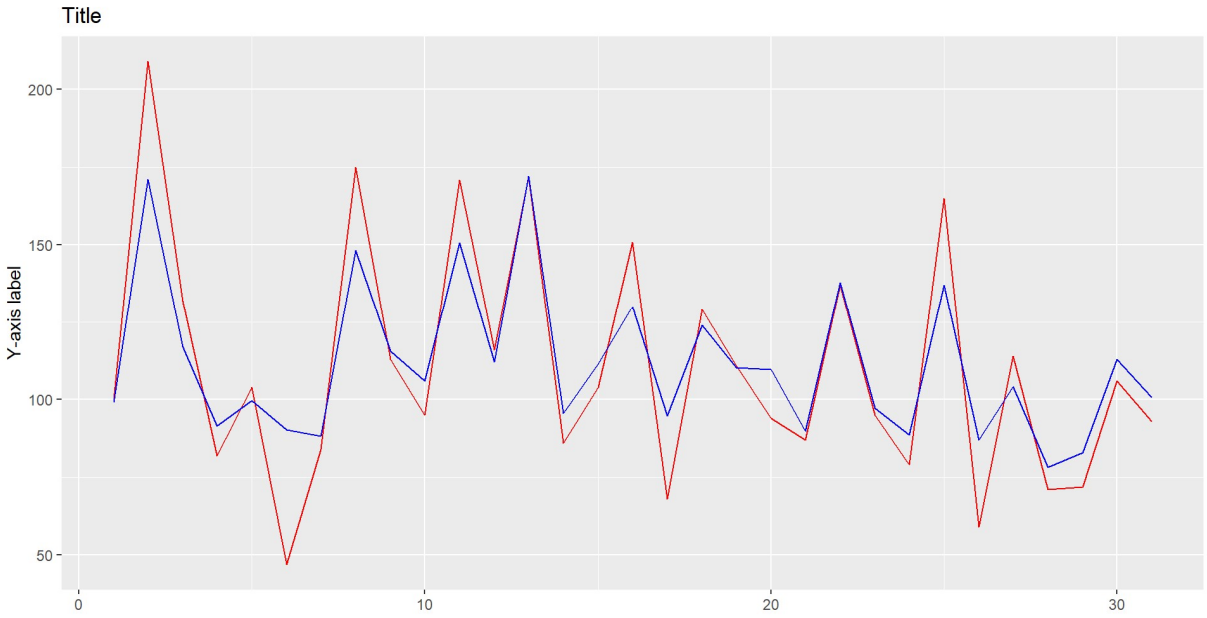

Figure 17

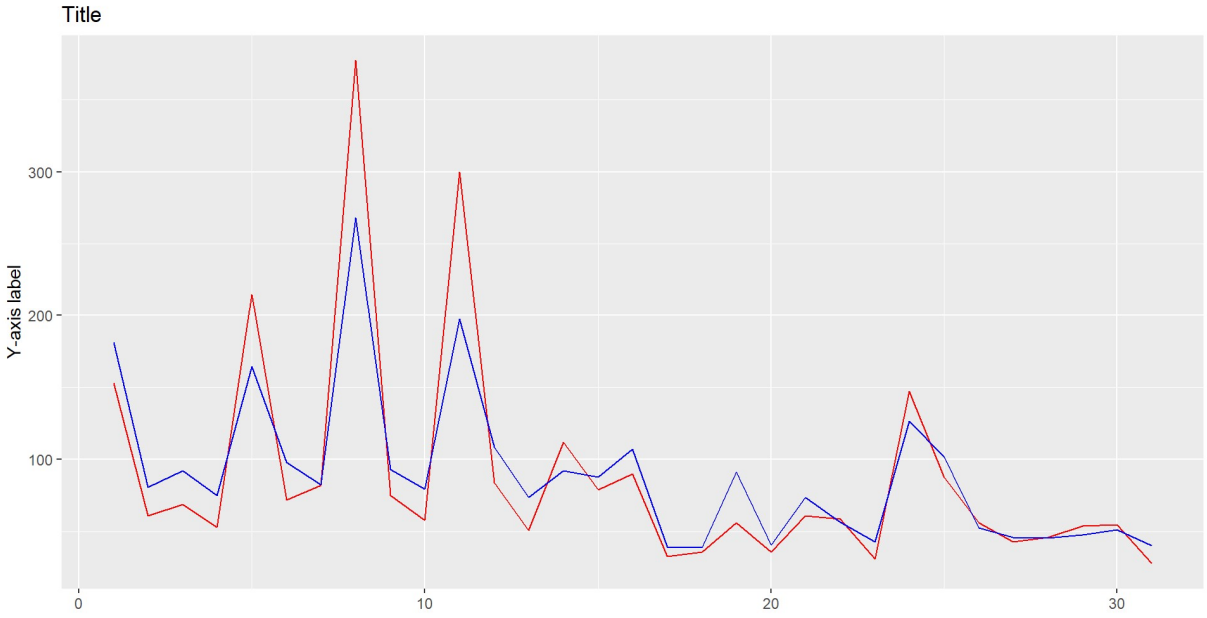

Figure 18

MAPE Summary

The MAPE are very high (roughly 100 -200 %) it shows Random Forest may not a suitable tools in this prediction.

Item	Training
1st Top Item	93.00 %
3rd Top Item	133.00%
5th Top Item	233.00%
Average	153.00

Conclusion

Comparison of all models used

Overall, all the models did poorly with the best model being averaged neural networks with MAPE of ~ 30% for training and test set. One hidden layer network produces minimum error ~ 12% in training but in testing it increases to 60% which indicates overfitting.

SVM produced decent models in training, but error increased in testing set. This was true for all the kernel methods. Sigmoid function produced minimum error in training of ~16%.

In time series forecasting models, we found similar performance in both ARIMA and holt winters. Both had an error rate of ~30%.

Linear regression had an error of ~30% in training but in testing set it shot up to 75%. This should be expected since relationships in time series is not linear. Random forest was the worst model with over 150% error.

			MAPE (%)	
Analysis Type	Configuration	Item	Training	Testing
Linear Regression	n/a	1st Top Item	16.91	23.78
		3rd Top Item	23.43	28.53
		5th Top Item	44.80	172.90
		Average	**28.38**	**75.07**
Averaged Neural Network	size = 3 (Autotune)	1st Top Item	13.90	24.74
	size = 1 (Autotune)	3rd Top Item	22.40	53.19
	size = 1 (Autotune)	5th Top Item	55.48	25.39
		Average	**30.59**	**34.44**
Neural Network	1 hidden layer (15)	1st Top Item	10.35	27.76
		3rd Top Item	9.35	68.94
		5th Top Item	17.18	97.36
		Average	**12.29**	**64.69**
	3 hidden layer (5, 5, 5)	1st Top Item	19.91	16.16
		3rd Top Item	21.31	58.21
		5th Top Item	93.81	68.23
		Average	**45.01**	**47.53**
SVM	method = svmRadial	1st Top Item	11.72	18.27
		3rd Top Item	18.06	40.82
		5th Top Item	23.20	145.20
		Average	**17.66**	**68.10**
	method = svmLinear	1st Top Item	14.89	20.88
		3rd Top Item	19.41	51.38
		5th Top Item	40.52	66.68
		Average	**24.94**	**46.31**
	method = svmPoly , degree = 4 (Autotune)	1st Top Item	10.73	18.60
	method = svmPoly , degree = 5 (Autotune)	3rd Top Item	19.46	45.95
	method = svmPoly , degree = 4 (Autotune)	5th Top Item	39.54	78.42
		Average	**23.24**	**47.66**
	kernal = "sigmoid"	1st Top Item	10.36	18.26
		3rd Top Item	13.86	42.10
		5th Top Item	25.25	164.32
		Average	**16.49**	**74.90**
HoltWinters	tuning grid search	1st Top Item	18.10	N/A
	tuning grid search	3rd Top Item	31.69	N/A
	tuning grid search	5th Top Item	55.09	N/A
		Average	**34.96**	N/A
ARIMA	tuning grid search	1st Top Item	17.53	N/A
	tuning grid search	3rd Top Item	29.21	N/A
	tuning grid search	5th Top Item	52.31	N/A
		Average	**33.02**	N/A
Random Forest	autotune	1st Top Item	93.00	N/A
	autotune	3rd Top Item	133.00	N/A
	autotune	5th Top Item	233.00	N/A
		Average	**153.00**	N/A

Recommendation and next step

We recommend using Averaged neural network and arima models for future prediction of sales quantity. Use this prediction to maintain inventory levels at the store appropriately. Also, keep track of any deviation in the model prediction vs actuals. This divergence signals two things, firstly performance of the product sales is not consistent with past. If it is negative direction then collect more data like feedback from customers and adjust appropriately or offer discounts to boost the sales. If divergence is in a positive direction then continue any sales promotion which are in play. Divergence also means that the model needs to be updated using latest data.

Appendix

SQL Script for Data Extraction

```
select date, item_sk, sum(item_qty) as quantity, sum(item_weight) as weight

from sales219 where item_sk in ('11740941', '11740923', '11680016', '11610106', '11686823',
'11741143')

group by item_sk, date order by date;
```

R Script for Data Analysis

Linear Regression Analysis

```
library(tidyverse)

library(caret)


setwd("D:/#Spring 2023/5580 - Text Mining/Assignment5")


# setup functions

mape <- function(actual,pred) {

  mape <- mean(abs((actual-pred)/actual))*100

  return (mape)

}


# Read the data file which has info about top 6 products from sales219

data <- read.csv('data.csv')


# Get top 6 product ids in list

top_products <- data %>%

  group_by(item_sk) %>%

  summarize(total_quantity = sum(quantity)) %>%

  select(item_sk, total_quantity) %>%
```

```
  arrange(desc(total_quantity))

#-------------------------------------------------------------------------
# selected product = Top 1st
#11740941
#11741274
#11629829
product <- top_products$item_sk[1]
product

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 3) %>%
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))
```

```
trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]


# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                        number=10,
                        repeats = 5)

############################

# Linear Regression
# Using pre-sliced data
# Linear regression
glmFitTime <- train(V8 ~ .,
                data = train,
                method = "glm",
                preProc = c("center", "scale"),
                tuneLength = 10,
```

```
                trControl = myCvControl)
glmFitTime
summary(glmFitTime)

# MAPE Training
y_hat = predict(glmFitTime, newdata = X)
mape(y,y_hat)
# MAPE Testing
y_hat2 = predict(glmFitTime, newdata = X_test)
mape(y_test,y_hat2)



#------------------------------------------------------------------------
# selected product = Top 3th
#11740941
#11741274
#11629829
product <- top_products$item_sk[3]
product

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 1) %>%
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)
```

```
# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))



trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]



# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5)
```

```
############################

# Linear Regression
# Using pre-sliced data
# Linear regression
glmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "glm",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
glmFitTime
summary(glmFitTime)

# MAPE Training
y_hat = predict(glmFitTime, newdata = X)
mape(y,y_hat)
# MAPE Testing
y_hat2 = predict(glmFitTime, newdata = X_test)
mape(y_test,y_hat2)

#-------------------------------------------------------------------------
# selected product = Top 5th
#11740941
#11741274
#11629829
product <- top_products$item_sk[5]
product
```

```
# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 603) %>%
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))


trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]


# Set X and y
X <- train[,1:7]
```

```
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                    number=10,
                    repeats = 5)

############################

# Linear Regression
# Using pre-sliced data
# Linear regression
glmFitTime <- train(V8 ~ .,
                data = train,
                method = "glm",
                preProc = c("center", "scale"),
                tuneLength = 10,
                trControl = myCvControl)
glmFitTime
summary(glmFitTime)

# MAPE Training
y_hat = predict(glmFitTime, newdata = X)
mape(y,y_hat)
# MAPE Testing
y_hat2 = predict(glmFitTime, newdata = X_test)
```

```
mape(y_test,y_hat2)
```

Neural Network Analysis

```
library(tidyverse)
library(caret)

#setwd("C:/")
setwd("D:/#Spring 2023/5580 - Text Mining/Assignment5/draft")



# Read the data file which has info about top 6 products from sales219
data <- read.csv('data.csv')

# Get top 6 product ids in list
top_products <- data %>%
  group_by(item_sk) %>%
  summarize(total_quantity = sum(quantity)) %>%
  select(item_sk, total_quantity) %>%
  arrange(desc(total_quantity))

# Get the first product and analyze



#-----------------------------------------------------------------
# selected product = 1st Top
#11740941
#11741274
#11629829
```

```
product <- top_products$item_sk[1]
product

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 3) %>%  # filter product & outliner
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))


trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]
```

```
# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5)

############################

# Neural Network
# Averaged Neural Network

nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "avNNet",
                   preProc = c("center", "scale"),
                   trControl = myCvControl,
                   tuneLength = 10,
                   linout = T,
                   trace = F,
                   MaxNWts = 10 * (ncol(train) + 1) + 10 + 1,
                   maxit = 500)
nnFitTime
```

```
summary(nnFitTime)
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)



# Neural Network 1 hidden layer (15)
grid <-  expand.grid(layer1 = 15,
                     layer2 = 0,
                     layer3 = 0)

nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "neuralnet",
                   algorithm="backprop",
                   tuneGrid = grid,
                   learningrate=0.01,
                   preProc = c("center", "scale"),
                   threshold = 0.05,
                   trControl = myCvControl
)

nnFitTime
summary(nnFitTime)
# MAPE Training
```

```
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
plot(nnFitTime$finalModel)



# Neural Network 3 hidden layer (5, 5, 5)
grid <-  expand.grid(layer1 = 5,
                     layer2 = 5,
                     layer3 = 5)



nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "neuralnet",
                   algorithm="backprop",
                   learningrate=0.01,
                   tuneGrid = grid,
                   threshold = 0.05,
                   preProc = c("center", "scale", "nzv"),
                   trControl = myCvControl
)



summary(nnFitTime)
```

```
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
plot(nnFitTime$finalModel)

#-----------------------------------------------------------------
# selected product = 3rd Top
#11740941
#11741274
#11629829

product <- top_products$item_sk[3]
product

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 1) %>%  # filter product & outliner
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL
```

```
# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))



trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]



# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5)

############################

# Neural Network
```

```
# Averaged Neural Network

nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "avNNet",
                   preProc = c("center", "scale"),
                   trControl = myCvControl,
                   tuneLength = 10,
                   linout = T,
                   trace = F,
                   MaxNWts = 10 * (ncol(train) + 1) + 10 + 1,
                   maxit = 500)
nnFitTime
summary(nnFitTime)
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)


# Neural Network 1 hidden layer (15)
grid <-  expand.grid(layer1 = 15,
                     layer2 = 0,
                     layer3 = 0)

nnFitTime <- train(V8 ~ .,
```

```
                data = train,
                method = "neuralnet",
                algorithm="backprop",
                tuneGrid = grid,
                learningrate=0.01,
                preProc = c("center", "scale"),
                threshold = 0.05,
                trControl = myCvControl,
                linear.output = T
)

nnFitTime

summary(nnFitTime)
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
plot(nnFitTime$finalModel)

# Neural Network 3 hidden layer (5, 5, 5)
grid <-  expand.grid(layer1 = 5,
                     layer2 = 5,
                     layer3 = 5)
```

```
nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "neuralnet",
                   algorithm="backprop",
                   learningrate=0.01,
                   tuneGrid = grid,
                   threshold = 0.05,
                   preProc = c("center", "scale", "nzv"),
                   trControl = myCvControl
)

nnFitTime
summary(nnFitTime)
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
plot(nnFitTime$finalModel)



#------------------------------------------------------------------
# selected product = 5th Top
#11740941
#11741274
#11629829
```

```
product <- top_products$item_sk[5]
product

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 603) %>%  # filter product & outliner
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))


trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]
```

```
# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                    number=10,
                    repeats = 5)

############################

# Neural Network
# Averaged Neural Network

nnFitTime <- train(V8 ~ .,
              data = train,
              method = "avNNet",
              preProc = c("center", "scale"),
              trControl = myCvControl,
              tuneLength = 10,
              linout = T,
              trace = F,
              MaxNWts = 10 * (ncol(train) + 1) + 10 + 1,
              maxit = 500)
nnFitTime
summary(nnFitTime)
```

```
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)



# Neural Network 1 hidden layer (15)
grid <-  expand.grid(layer1 = 15,
                     layer2 = 0,
                     layer3 = 0)

nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "neuralnet",
                   algorithm="backprop",
                   tuneGrid = grid,
                   learningrate=0.01,
                   preProc = c("center", "scale"),
                   threshold = 0.1,
                   trControl = myCvControl
)

nnFitTime
summary(nnFitTime)
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
```

```
mean(100*abs(y_hat-y)/y)
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
plot(nnFitTime$finalModel)

# Neural Network 3 hidden layer (5, 5, 5)
grid <-  expand.grid(layer1 = 5,
                     layer2 = 5,
                     layer3 = 5)



nnFitTime <- train(V8 ~ .,
                   data = train,
                   method = "neuralnet",
                   algorithm="backprop",
                   learningrate=0.01,
                   tuneGrid = grid,
                   threshold = 0.1,
                   preProc = c("center", "scale", "nzv"),
                   trControl = myCvControl
)

nnFitTime
summary(nnFitTime)
# MAPE Training
y_hat = predict(nnFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
```

```
# MAPE Testing
y_hat2 = predict(nnFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
plot(nnFitTime$finalModel)
```

SVM Analysis

```
library(tidyverse)
library(caret)
library(e1071)

setwd("D:/#Spring 2023/5580 - Text Mining/Assignment5")

set.seed(1)

# Read the data file which has info about top 6 products from sales219
data <- read.csv('data.csv')

# Get top 6 product ids in list
top_products <- data %>%
  group_by(item_sk) %>%
  summarize(total_quantity = sum(quantity)) %>%
  select(item_sk, total_quantity) %>%
  arrange(desc(total_quantity))

#------------------------------------------------------------------------
# selected product = Top 1st
#11740941
#11741274
```

```
#11629829
product <- top_products$item_sk[1]
product

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 3) %>%
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))


trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]
```

```
# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5)



# Support Vector Regression
# method = svmRadial
cat(product , "svmRadial")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmRadial",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
```

```
# method = svmLinear
cat(product , "svmLinear")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmLinear",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)


# method = svmPoly
cat(product , "svmPoly")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmPoly",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl,
                    tuneGrid = expand.grid(
                      C = c(1e-1, 1e0, 1e1, 1e2, 1e3, 1e4),
                      degree = c(1, 2, 3, 4, 5, 6),
```

```
                  scale = 0.001)
                )
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

# Kernel = “sigmoid”
cat(product , "sigmoid")

svmFitTime=svm(V8 ~ .,
            data = train,kernek="sigmoid")
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

#------------------------------------------------------------------------
# selected product =  3 rd Top
#11740941
#11741274
#11629829
product <- top_products$item_sk[3]
product
```

```
# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product, quantity != 1) %>%
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))


trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]


# Set X and y
X <- train[,1:7]
```

```
y <- train[,8]

X_test <- test[,1:7]
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5)



# Support Vector Regression
# method = svmRadial
cat(product , "svmRadial")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmRadial",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

# method = svmLinear
cat(product , "svmLinear")
```

```
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmLinear",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)



# method = svmPoly
cat(product , "svmPoly")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmPoly",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl,
                    tuneGrid = expand.grid(
                      C = c(1e-1, 1e0, 1e1, 1e2, 1e3, 1e4),
                      degree = c(1, 2, 3, 4, 5, 6),
                      scale = 0.001)
)
svmFitTime
```

```
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

# Kernel = “sigmoid”
cat(product , "sigmoid")

svmFitTime=svm(V8 ~ .,
            data = train,kernek="sigmoid")
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

#------------------------------------------------------------------------
# selected product = Top 5 th
#11740941
#11741274
#11629829
product <- top_products$item_sk[5]
product

# Filter the data for selected product only
xy <- data %>%
```

```
  filter(item_sk == product, quantity != 603) %>%
  select(-item_sk)

# Check for any missing dates in the data

is_continuous <- all(diff(xy$Date) == 1)

# Remove the data as well
xy$date <- NULL

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))


trainIndex<-createDataPartition(reshaped_xy$V8, p=0.8, list=FALSE)
train<-reshaped_xy[trainIndex,]
test<-reshaped_xy[-trainIndex,]


# Set X and y
X <- train[,1:7]
y <- train[,8]

X_test <- test[,1:7]
```

```
y_test <- test[,8]

myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5)


# Support Vector Regression
# method = svmRadial
cat(product , "svmRadial")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmRadial",
                    preProc = c("center", "scale"),
                    tuneLength = 10,
                    trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

# method = svmLinear
cat(product , "svmLinear")
svmFitTime <- train(V8 ~ .,
                    data = train,
                    method = "svmLinear",
```

```
                preProc = c("center", "scale"),
                tuneLength = 10,
                trControl = myCvControl)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)



# method = svmPoly
cat(product , "svmPoly")
svmFitTime <- train(V8 ~ .,
                data = train,
                method = "svmPoly",
                preProc = c("center", "scale"),
                tuneLength = 10,
                trControl = myCvControl,
                tuneGrid = expand.grid(
                  C = c(1e-1, 1e0, 1e1, 1e2, 1e3, 1e4),
                  degree = c(1, 2, 3, 4, 5, 6),
                  scale = 0.001)
)
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
```

```
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)

# Kernel = “sigmoid”
cat(product , "sigmoid")

svmFitTime=svm(V8 ~ .,
        data = train,kernek="sigmoid")
svmFitTime
summary(svmFitTime)
y_hat = predict(svmFitTime, newdata = X)
mean(100*abs(y_hat-y)/y)
y_hat2 = predict(svmFitTime, newdata = X_test)
mean(100*abs(y_hat2-y_test)/y_test)
```

HoltWinters, ARIMA and Random Forest Analysis

```
library(tidyverse)
library(tseries)
library(forecast)
library(caret)
library(rpart)

# Clear and setup run environment

setwd('C:/Users/ajay2/OneDrive/OneDrive - Saint Marys University/Desktop/MCDA5580 Mining/Assignmen4')

# Clear environment
rm(list = ls())
```

```
# Clear plots
while (dev.cur() != 1) dev.off()

# Clear console
cat("\014")

# custom summary function for MAPE
custom_mape <- function(data, lev = NULL, model = NULL)
{
  MAPE <- mean(100 * abs(data$obs - data$pred)/data$obs)
  names(MAPE) <- c('custom_mape')
  return(MAPE)
}



product_fit_function <- function(prod_index)

{

# Get the product data and analyze

product <- top_products$item_sk[prod_index]

# Filter the data for selected product only
xy <- data %>%
  filter(item_sk == product) %>%
  select(-item_sk)
```

```
# Remove the date as well
xy$date <- NULL

# create box plot
ggplot(data = xy, aes(y = quantity)) +
  geom_boxplot(color = 'black', fill = 'white') +
  ggtitle('Box Plot of x')

# Time series

tSeries = ts(xy, start = 1, freq = 7)
plot(stl(tSeries[,1],s.window='periodic'))

# Remove minimum value which is extremely low

if (min(xy) < 10)
  {
  print('Removing minimum value from data which is less than 10')
  xy_cleaned <- data.frame(xy$quantity[-which.min(xy$quantity)])
  xy <- xy_cleaned
  }

tSeries_cleaned <- ts(xy, start = 1, freq = 7)

# Holt Winters Model with grid search

parameter_grid <- expand.grid(error = c('A', 'M'),
```

```
                    trend = c('A', 'M'),
                    seasonal = c('A', 'M'))

# Initialize variables for grid search
best_hw <- NULL
best_mape <- Inf

# Loop through each model combination
for (i in 1:nrow(parameter_grid))
 {
 # Extract the model parameters from the grid
 error <- parameter_grid$error[i]
 trend <- parameter_grid$trend[i]
 seasonal <- parameter_grid$seasonal[i]

 model_type = paste0(error, trend, seasonal)

 if (model_type == 'AMA' | model_type == 'MMA' |
    model_type == 'AAM' | model_type == 'AMM')
 {
  next
 }

 # Fit the ETS model
 fit <- ets(tSeries_cleaned, model = model_type)

 # Compute the MAPE
 mape <- mean(100*abs(fitted(fit) - tSeries_cleaned)/tSeries_cleaned)
```

```
  # Check if this is the best model so far
  if (mape < best_mape)
    {
    best_hw <- fit
    best_mape <- mape
    }
}

# Print the best model and its MAPE value
print(best_hw)
print(paste0('MAPE for holt winters: ', best_mape))

# Use auto tuning arima method and print its coefficients
arima_auto_model <- auto.arima(tSeries_cleaned)
summary(arima_auto_model)

# Plain arima with intuition of weekly repeatation of data
arima_plain <- Arima(tSeries_cleaned, order = c(7, 0, 7))
summary(arima_plain)

# Plain arima looks reasonable and simple
print(paste0('MAPE for arima = ',
  mean(100*abs(fitted(arima_plain) - tSeries_cleaned)/tSeries_cleaned)))

print(autoplot(tSeries_cleaned, series = 'Actual') +
  autolayer(fitted(arima_plain), series = 'Arima') +
  autolayer(fitted(best_hw), series = 'Holts-Winters'))
```

```
# Random Forest

# Reshape the dataframe to have 7 independent and 1 dependent (8 in total cols)

total_rows = nrow(xy)
rows_to_reshape = total_rows - (total_rows %% 8)
reshaped_xy <-
  as.data.frame(matrix(xy[1:rows_to_reshape, 1], ncol = 8, byrow = TRUE))

# Set X and y

X <- reshaped_xy[,1:7]
y <- reshaped_xy[,8]

# Training Control
myCvControl <- trainControl(method = "repeatedCV",
                            number=10,
                            repeats = 5,
                            summaryFunction = custom_mape)

# Random Forest
random_forest <- train(V8 ~ .,
                       data = reshaped_xy,
                       method = "rf",
                       preProc = c("center", "scale"),
                       tuneLength = 10,
                       metric = 'custom_mape',
```

```
                    trControl = myCvControl)

y_pred <- predict(random_forest, newdata = reshaped_xy)

print(paste0('MAPE for random forest = ',
     round(mean(100 * abs(y_pred - y)/y )), 3))

df <- data.frame(y_actual = y, y_predicted = y_pred)

plot <- ggplot(df) +
  geom_line(aes(x = seq_along(y_actual), y = y_actual), col = "red") +
  geom_line(aes(x = seq_along(y_predicted), y = y_predicted), col = "blue") +
  labs(x = NULL, y = "Y-axis label", title = "Title")

print(plot)
}

################################################


# Main script starts here



# Read the data file which has info about top 6 products from sales219
data <- read.csv('data.csv')

# Get top 6 product ids in list
```

```
top_products <- data %>%
  group_by(item_sk) %>%
  summarize(total_quantity = sum(quantity)) %>%
  select(item_sk, total_quantity) %>%
  arrange(desc(total_quantity))

# Call the function to analyze and fit various models

for (i in seq(1, length(top_products$item_sk), by = 2))
  {
  product_fit_function(i)
  }
```

Reference/ Citation

Linear Regression

https://towardsdatascience.com/understanding-linear-regression-output-in-r-7a9cbda948b3

Neural Network

https://www.rdocumentation.org/packages/caret/versions/6.0-80/topics/avNNet

https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

https://machinelearningmastery.com/how-to-control-neural-network-model-capacity-with-nodes-and-layers/

https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev

SVM

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

https://bookdown.org/mpfoley1973/supervised-ml/support-vector-machines.html

https://dataaspirant.com/svm-kernels/

https://koalatea.io/r-svm-regression/

https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/

Caret Package

https://towardsdatascience.com/a-guide-to-using-caret-in-r-71dec0bda208

https://cran.r-project.org/web/packages/caret/vignettes/caret.html

Holt Winters

Holt-Winters - RapidMiner Documentation

ARIMA

https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp

How to Create an ARIMA Model for Time Series Forecasting in Python - MachineLearningMastery.com

Random Forest for time series forecasting

Random Forest for Time Series Forecasting - MachineLearningMastery.com