NPD IMRAD Draft

Frank Kovacs, Ning Gao, Pragya Jain, Wonil Lee

April 2021

Abstract

The main scope of the project was to develop two anomaly detecting algorithms to identify and report data collection errors for the 3 key variables, *Receipt_count*, *Sum_total_paid*, and *Item_total*, provided by the NPD group. In particular, the Moving First Differences Filter calculated the desired quantile of the slopes in the selected window size to check if the new slope lied between the calculated quantile. The Trimmed Moving Weighted Average Filter calculated the robust mean and variance in the selected window size to check if the value of new data point lies within the weighted average ± 3(standard deviation). The final output for the project would be a series of visualization plots and a record of all anomalies detected, outputted using a Python package interface.

1 Introduction

The client 'NPD Group' is a market research company. Their mission includes creating reliable raw data assets for their end clients. These datasets are useful for their clients to strategize on increasing their brand's market share, identifying up/down trends in sales, and understanding customer behavior. Ensuring quality of data resources is critical for the NPD team. Being a leader in market research, NPD deals with more than 8 billion B2B transactions per year. This report aims to assist them in automatic detection of anomalies in incoming data. The detected anomalies will then be assessed more closely by their data team, to ensure provision of error-free data to their end clients.

The project aims to develop data-driven filtering methods to help identify anomalies in data and facilitate further data analyst review. The report presents an automated data error flagging process which can be run weekly to flag new anomalies in the incoming data series. The report proposes solutions for the following research questions:

1. Detection of unexpected changes in incoming data series for 3 key variables: *Receipt_count*, *Sum_total_paid*, and *Item_total*. Proposed methods should detect anomalies in addition to errors already flagged by the client.

2. Detection of both rapid drops and gradual dips spanned over larger time periods.

3. Detection of peaks with significant amplitude in addition to detection of dips.

2 Data

2.1 Data sets

The data used for the project were provided by our client, the NPD group. There were 6 response variables that we were interested in using, which were *Receipt_count*, *Sum_total_paid*, *Sum_items_distinct*, *Sum_item_spend*, *item_total*, and *Panelists* (Table 1).

If you're not going to capitalize later, don't capitalize here.

There were overall three data sets given: Source Data, Retail Data, and Issue Data. The source data set consisted of 516 rows and 8 columns. The data set provided weekly values of the key variables listed above by 4 different data source types which were iPhone, Android, Sift, and Receipt pal on device. The second data set, retail data, consisted of 983,953 rows and 11 columns and provided weekly values of the key variables per each merchant as well as information about the merchant name/ID and data acquire type/ID. Finally, the issue data consisted of 31 rows and 5 columns and it provided information about when (the Acquired date) and where (merchant name and source type) the data collecting error had occurred.

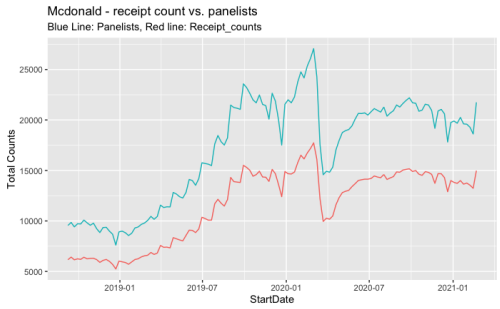
Variable Name	Values	Description
receipt_count	Integer	Weekly value of total number of receipt
Sum_total_paid	Integer	Weekly value of total amount paid in USD
sum_items_distinct	Integer	Weekly value of total number of distinct items
sum_item_spend	Integer	Weekly value of total number of items spent
panelists	Integer	Weekly value of total participants
item total	Integer	Weekly value of total items purchased
MerchantName	Text Labels	Name of the merchant where the transaction occurred
MerchantID	Integer	Unique identifier for each merchant
AcquireTypeDesc	Text Labels	Source where the data were acquired *(iPhone, Android, Sift, Receipt pal on Device)*
AcquireTypeID	Integer	Unique identifier for each data source

Table 1: Key Variables Explained

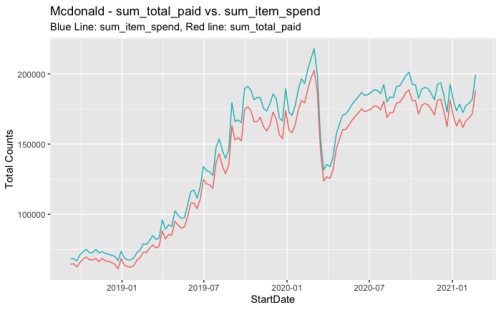
2.2 Variable Selection

After taking a closer look at the data description provided from NPD, our team explored whether there were correlations between some of the key response variables. To elaborate, *Receipt_count*, which was the total number of transaction in given week might be closely associated with *Panelist* which was the total number of participant in a given week. Similarly, *Sum_total_paid*, which was the dollar amount paid in a given week might be closely associated with *Sum_item_spend* which was the total number of items purchased. And the same logic went to *Item_total* and *Sum_items_distinct*.

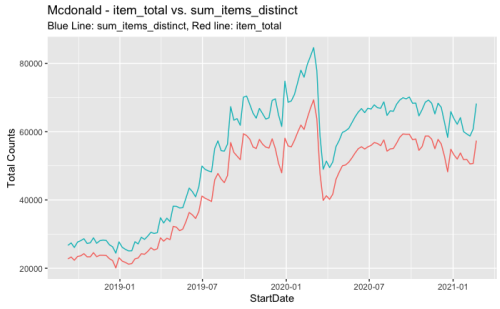
In order to test our hypothesis, we first plotted the time series of the variables we wanted to compare to check if the general trend of the variables matched with one another. As shown in Figure 1 below, the three pairs of response variables all showed highly similar trends.



(a) Receipt count vs Panelist



(b) Sum total paid vs Sum item spend



(c) Item total vs Sum items

Figure 1: Example: MacDonald Case

Once checking that the trends for each pair of variables were similar, our team calculated the average correlations between each pair for all merchants. The calculated correlations in Table 2 strongly supported our hypothesis since all three correlations had a score over 0.99. With both visual and mathematical backups, our team decided to use only the three variable, *Receipt_count*, *Sum_total_paid*, and *Item_total*, as our key response variables for the project.

Pair of Columns	Correlation
Receipt_count & Panelists	**0.9937866**
Sum_total_paid & Sum_item_spend	**0.9990581**
Item_total & Sum_items_distinct	**0.9991476**

Table 2: Correlations Calculated

2.3 Exploratory Data Analysis

Due to the fact that the client provided 31 issues logged by them in the past 2 years, our team decided to use it as a guide to understand the types of errors their algorithm could detect and also discover hidden errors that might worth detecting. Exploratory Data Analysis was conducted with the goal of assessing possible error types existing in the data set and to compare possible errors with errors flagged by the client. Un-flagged errors were then studied and discussed with the client to discover client's preferences in detecting errors.

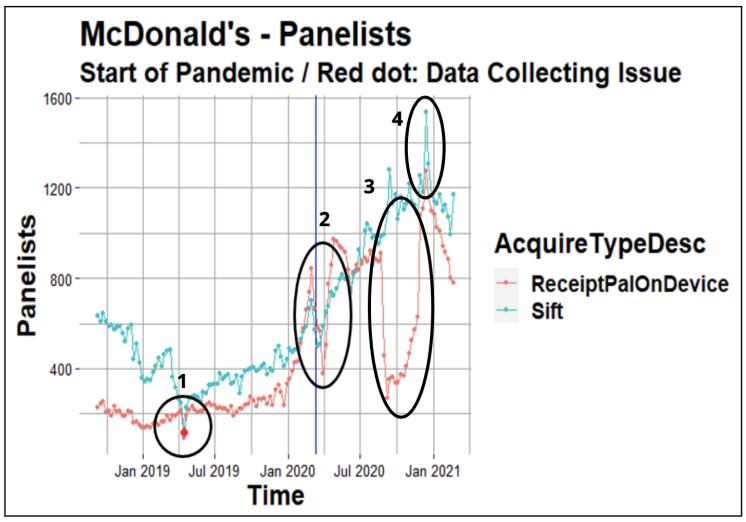


Figure 2: EDA Plot for Merchant: McDonald's

Figure 2 illustrates one compact example for error analysis on merchant 'McDonald's'. The red dot at Circle 1 indicated a flagged error, which was a rapid drop marked by NPD. When looking at Circle 1 and 3 at the same time, 3 had a larger rapid drop compared to 1, yet not detected. The observation at Circle 2 happened around the outbreak of COVID-19 pandemic on March 11th, 2020, which was marked using a blue vertical line. This drop happened across more than a month, instead of over a short period of time like 1, but the dropping trend was not detected. Cirlce 4 contained a sudden peak. Even though the amplitude change of this jump was definitely greater than Circle 1, it was not detected, which brought the question of whether NPD had a preference of marking dips over peaks. This compact example was only one out of many EDA plots generated, please refer to our EDA section in Technical Appendix 6.1 for detailed error visualizations.

3 Methods

3.1 First Differences Filter

The first method our team implemented to detect anomalies in the time series was the First Differences Filtering method. The differences being calculated were essentially the slopes of changes from time t to $t+1$. Once the difference was know, the large values, which represented the large sudden change, could be spotted. Ultimately, the First Differences Filtering method allowed the user to detect anomalies based on the past trend of slopes. The logic of this method mainly consisted of two concepts: shifting windows and 95% quantile.

what are you using in the window from t-n to t, to predict at time t+1?

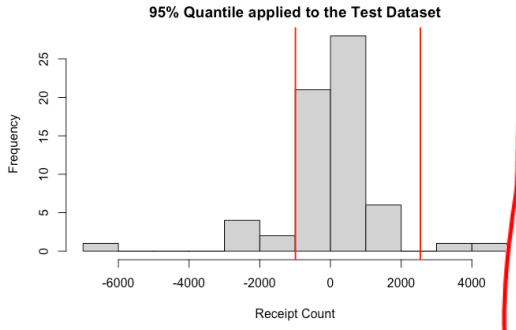
To begin with, the shifting window was a concept of selecting proper training data set size at time t, to detect anomaly at time $t+1$. The window was not fixed but set as sliding to ensure the training data set represented the most recent trend of changes in slope. For each point in the future, the algorithm would perform anomaly detection using the most up-to-date window, or filter of size n.

Once the window size was selected by users, the next step was to calculate the appropriate percent quantile of differences within the window. The default quantile bounds was set at 5% and 95% for each window. Once the quantile was calculated, the algorithm would spot the value at time $t+1$ as an anomaly if it lied outside of the calculated quantile.

As a sanity check, the McDonald's Receipt Count data set collected from iPhone was used as an example for illustration purpose. We divided the data set into 1^{st} half and 2^{nd} half. The 95% quantile from the 1^{st} half of the data set was calculated (Figure 3) and applied on the 2^{nd} half of the data set. In Figure 4, any data point that lied outside of the two red quantile boundaries would be marked as anomalies.

I don't understand the relationship between "window" and "training data" here, nor how this is connected to the reference to "training data" and "test data" in figure 3 below.

can you illustrate the first difference method with a plot showing tangent or secant lines to the original time series data? Or a pair of time series plots, one the original data and the other the instantaneous slopes at time t? Or ...?



these look like results for the results section of the paper

What is zero here? just constant slope?

What is actualy being plotted in each plot?

(a) 1st Half - Training Dataset

(b) 2nd Half - Test Dataset

Figure 3: Sanity Check on McDonald's Data - 1/2

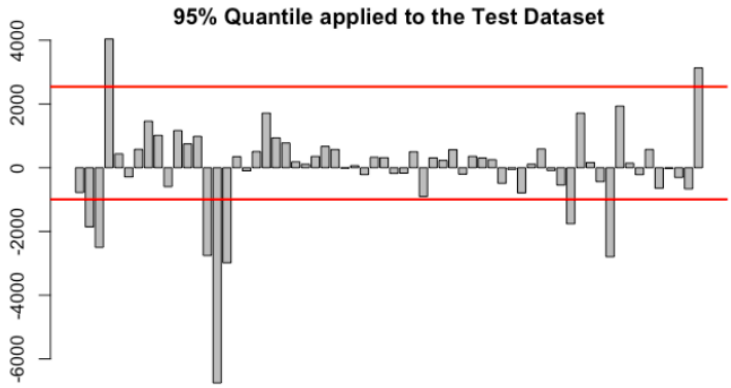


Figure 4: Sanity Check on McDonald's Data - 2/2

3.2 Trimmed Moving Weighted Average Filter

The second method our team implemented for anomaly detection was the trimmed moving weighted average method, which calculated the robust estimates of weighted average and standard deviation in a chosen time window.

3.2.1 Robust Estimate for Weighted Average

The first step to ensure the weighted average in any given window was robust was to remove a reasonable amount of outliers from that window. The algorithm could remove up to 20% of data in a set window. To define such outliers, the median of the data in the window was computed and used as a reference to calculate distances of all points from it. Depending on the maximum number of outliers user wanted to remove, the points with largest absolute distances from the median would be dropped. For example, if a user wanted to drop 2 out of 10 points (20%) in a window of size 10, the algorithm would drop the two points that were farthest away from the median. The remaining 8 points, in this case, were then used to calculate the robust weighted average.

Our algorithm can

Is the removal symmetric about the median, so the highest and lowest point would be dropped, or can the 2 points be dropped from the same side of the median?

The second step was to assign weight to each remaining points in a given window and compute the weighted average for all rolling windows. Given that the time series data were not stationary, the algorithm assigned higher weights to more recent points in order to capture more relevant trend. For example, if the window spanned from $t-5$ to t, with t being the current time, the weight assigned to point at $t-5$ would be the lowest, and assigned to t would be the highest. Specifically, We utilized the reversed half normal distribution to compute such weights for points in a given window. The half normal pdf was:

$$w_i = \sqrt{\frac{2}{\pi\sigma^2}} exp(-\frac{i^2}{2\sigma^2}) \tag{1}$$

In equation (1), the variance σ^2 was represented in terms of window size. In particular, the standard deviation was taken to be a third of the window size. Let n represent the size of a window, equation (1) then becomes:

can you say why, or perhaps what the consequence of a different choice would be?

$$w_i = \sqrt{\frac{2}{\pi(\frac{n}{3})^2}} exp(-\frac{i^2}{2(\frac{n}{3})^2}) \tag{2}$$

Finally, to calculate the weighted average WA, the weight w_i for each point in the window would be mapped to the response column value y_i:

$$WA = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i} \tag{3}$$

3.2.2 Robust Estimate for Standard Deviation

In the trimmed moving weighted average filter, standard deviation in a window served as the boundaries when testing anomalous behaviour for a future observation. Since standard deviation could be blown up easily by outliers in a data set, a more stable, robust estimate was needed.

the

standard deviation of what in the window?

To achieve such goal, the first step was to take the first differences of points in a given window. For example, if at time t, the response column value was Y_t, then the difference D_t would be:

$$D_t = Y_t - Y_{t-1} \tag{4}$$

The second step was to calculate the variance of the first difference D_t. Since,

$$var[D_t] = var[Y_t] + var[Y_{t-1}] - 2cov[Y_t, Y_{t-1}] \tag{5}$$

the variance of each Y_t,

Let τ^2 represent the variance for D_t, and σ^2 represent the sample variance, if Y_t and Y_{t-1} are independent then assuming $cov[Y_t, Y_{t-1}]$ to be 0, τ^2 became:

you don't need to say the same thing twice.

$$\tau^2 = 2\sigma^2 \tag{6}$$

Thus, sample variance σ^2 was estimated from τ^2 i.e. the variance of first differences in a window. However, since variance was sensitive to outliers, the median absolute deviation (MAD) was applied on τ^2 to ease the influence of outliers. The sample variance was then estimated using:

what does this mean? what actual calculation was done here?

$$\sigma^2 = \frac{\tau^2_{MAD-estimate}}{2} \tag{7}$$

Together, the robust moving weighted average and robust standard deviation granted acceptable ranges for a future point to land on. This range was defined in terms of 3 standard deviations up and below the weighted average. If an new observation had a value outside this range, it was then marked as an anomaly.

yielded a

in which

would be accepted as non-anomalous.

you haven't mentioned 1 and 2 sigma intervals, which are also shown in your plots in results section. please do.

3.2.3 Sequential Filters

To facilitate the detection of slow-trending anomalies, our Furthermore, for anomaly detection at a future point, the algorithm applied 4 sequential filters. All filters had the same window size but covered different points in the past. For example, if the window size was n, for a point in the future $t+1$, 4 filters would perform anomaly detection of that point simultaneously. Filter 1 would cover points from $t-n$ to t. Filter 2 would cover points from $t-n-1$ to $t-1$ and so on.

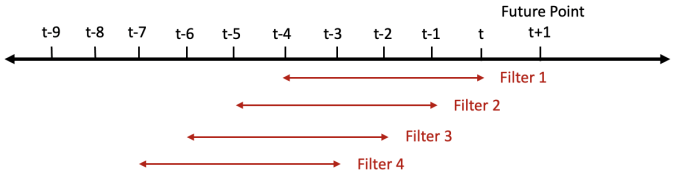


Figure 5: Illustration on Four Sequential Filters

The use of sequential filters facilitated the detection of slow-trending anomalies. For example, in the figure above, Filter 1 might miss anomalies embedded in a slow moving trend, since it ended right before the future point. As a result, the weighted average value would follow the anomalous trend. This could impact the ability of Filter 1 in detecting anomalies. However, Filter 4 would be helpful in such scenarios, since there was a gap between its ending point and the future point. In a case where there was a steady, decreasing trend, the boundary of Filter 4 would reflect on the normal data before the steady drop, and signify such slow drop by marking the new point as an anomaly. Thus, ensuring some distance between filtering window and anomalous point would serve in the interest of picking up slow trends in the time series.

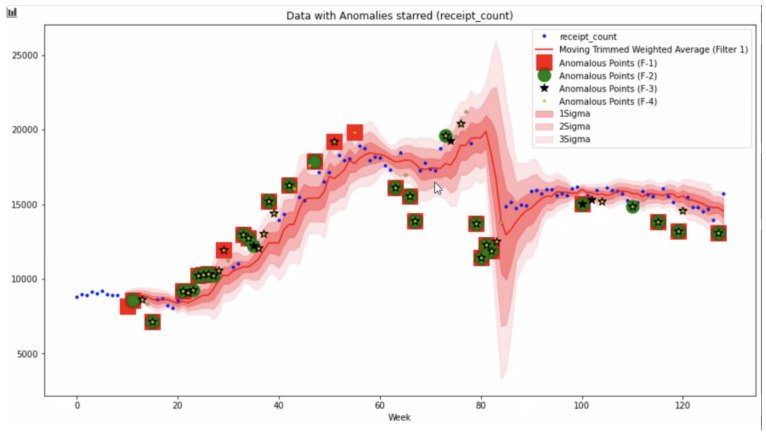
4 Results

good start. you should present results in the same order as in the methods section, in the final paper.

4.1 Types of anomalies we want to detect

4.1.1 Trimmed Moving Weighted Average Filter

Below are some resultant visualizations using the Trimmed Moving Weighted Average Filter.



(1) please enlarge figures to make them easier to read and understand.

(2) remove extraneous lines at the edges of the figures

Figure 6: Visualization: Merchant=McDonald's, Acquire Type=Android, Window=10, Outliers Removed=1, Response Column = Receipt Count

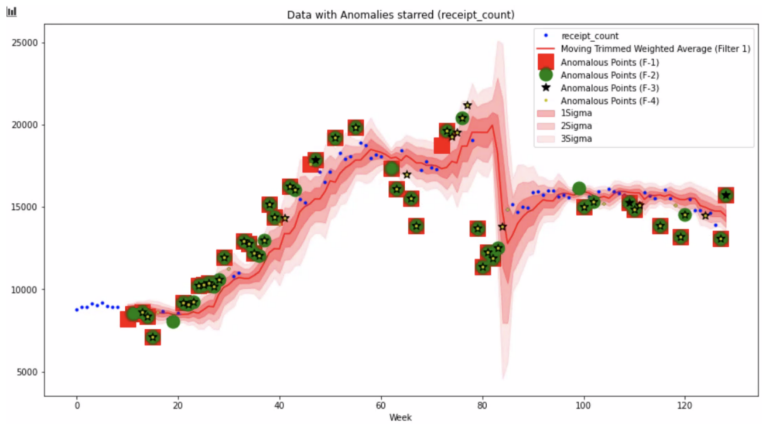


Figure 7: Visualization: Merchant=McDonald's, Acquire Type=Android, Window=10, Outliers Removed= 2, Response Column = Receipt Count

5 Discussion

I'll look for full discussion in final paper.

The approach of using variance of first differences was analogous to using variance of residuals. However, if there was a trend in the series, $E[d_t]$ would not necessarily be 0, which was unlike expectation of residuals after fitting a regression model. This could result in a slightly inflated sample variance, but could be still reasonably employed for our purposes of bounds in the filter.

ok

Please add a description and discussion of the python dashboard you have created, to the results and discussion sections.

No references?

* Add a reference section

* include papers you have read or referred to for this project

* include documentation for R and python libraries you have used

* cite each reference at least once in the main body of the paper

6 Technical Appendices

6.1 EDA and Variable Selection

Good start.

The "narration" of technical material in sections 6.3 and 6.4 is much better than in 6.1 and 6.2.

Please remember to refer reader to specific appendix pages for details, in the methods and results sections of your final paper.

Data Preparation and Wrangling

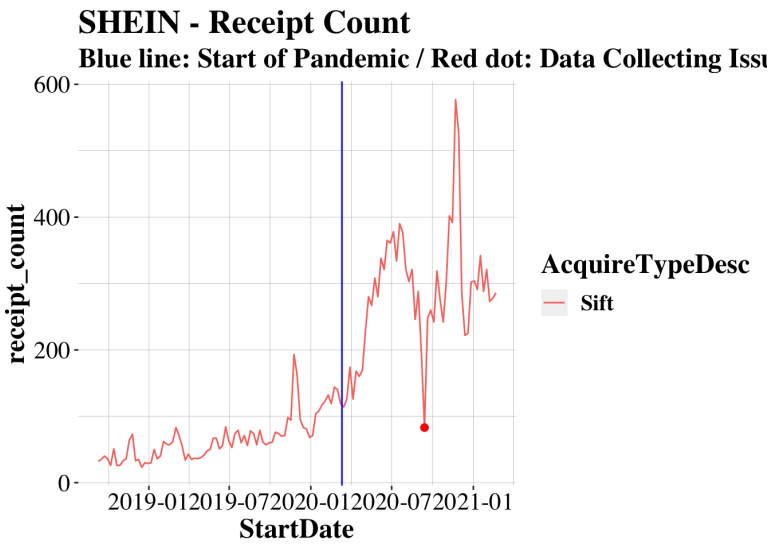
```
## make the date
retailer_data$StartDate <- as.Date(retailer_data$StartDate)
source_data$StartDate <- as.Date(source_data$StartDate)
issue_data$StartDate <- as.Date(issue_data$StartDate,format = "%m/%d/%Y")



issue_data$problem_detected <- rep(1, nrow(issue_data))

## look at the characteristics of retailer_data and the dates
## retailer data has the startDate of the week, we need to check which interval it is in
## i.e. if we have 9/16 and 9/23 with an issue on 9/20, we would have the it matched to 9/16
## for the jersey mikes we have issues in receiptPalOnline but no data for it

## next retailer is 13540 --> U.S. general

merged_data <- subset(retailer_data, MerchantName %in% unique(issue_data$MerchantName) &
                        MerchantName != "Jerseymikes") %>%
  left_join(x = ., y = issue_data,
            by = c("MerchantID", "MerchantName","AcquireTypeID", "AcquireTypeDesc",
                   "StartDate"))
merged_data$problem_detected[which(is.na(merged_data$problem_detected))] <- 0
## add in the jersey mikes data
jerseymikes_retailer <- subset(retailer_data, MerchantName == "Jerseymikes")
jerseymikes_retailer$problem_detected <- rep(0, nrow(jerseymikes_retailer))
## add in the rows
jm_add_df <- subset(issue_data, MerchantName == "Jerseymikes")
jm_add_df$problem_detected <- NULL
jm_add_df$receipt_count <- rep(0, nrow(jm_add_df))
jm_add_df$sum_total_paid <- rep(0, nrow(jm_add_df))
jm_add_df$item_total <- rep(0, nrow(jm_add_df))
jm_add_df$sum_items_distinct <- rep(0, nrow(jm_add_df))
jm_add_df$sum_item_spend <- rep(0, nrow(jm_add_df))
jm_add_df$panelists <- rep(0, nrow(jm_add_df))
jm_add_df$problem_detected <- rep(1, nrow(jm_add_df))
jerseymikes_retailer <- rbind(jerseymikes_retailer, jm_add_df)

merged_data <- rbind(merged_data, jerseymikes_retailer)
unique(merged_data$MerchantName)
```

```
##  [1] "McDonald's"        "Walmart"           "SHELL"
##  [4] "Petco"             "Firehouse Subs"    "Dominos.com"
##  [7] "U.S. Cellular"     "Moe's"             "Popeyes"
## [10] "Delivery.com"      "Jimmyjohns.com"    "Dollargeneral.com"
## [13] "Shopjustice.com"   "Customer Center"   "SHEIN"
## [16] "Jerseymikes"
```

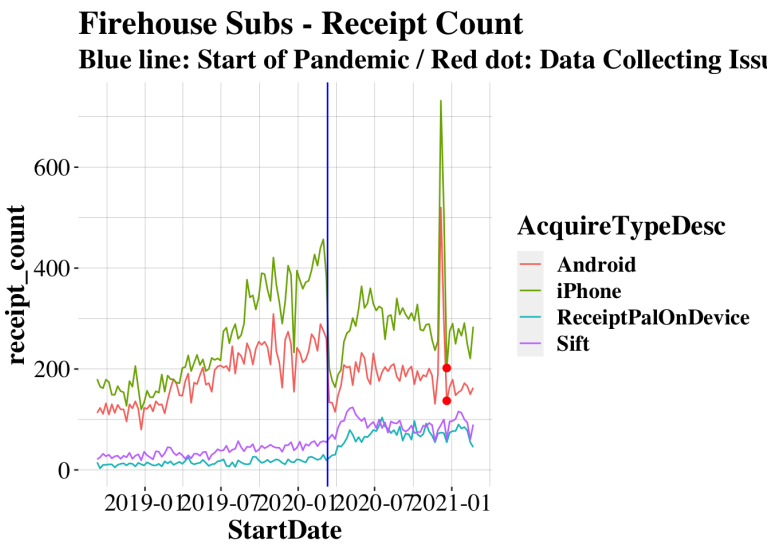
EDA

SHEIN, Firehouse Subs, JimmyJohns - Receipt Count

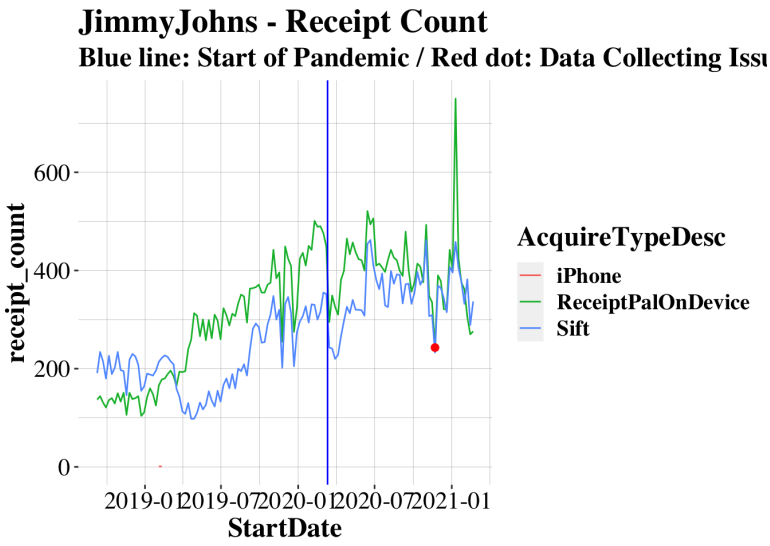
```
shein_data <- subset(merged_data, MerchantName == "SHEIN")

ggplot(data = shein_data,
       aes(x = StartDate, y = receipt_count, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=shein_data[which(shein_data$problem_detected==1), ],aes(x=StartDate, y=receipt_count), col
our="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'SHEIN - Receipt Count', subtitle = "Blue line: Start of Pandemic / Red dot: Data Collecting
Issue") + NPD_theme
```




```
Firehouse_data <- subset(merged_data, MerchantName == "Firehouse Subs")

ggplot(data = Firehouse_data,
       aes(x = StartDate, y = receipt_count, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=Firehouse_data[which(Firehouse_data$problem_detected==1), ],aes(x=StartDate, y=receipt_cou
nt), colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'Firehouse Subs - Receipt Count', subtitle = "Blue line: Start of Pandemic / Red dot: Data Co
llecting Issue") + NPD_theme
```

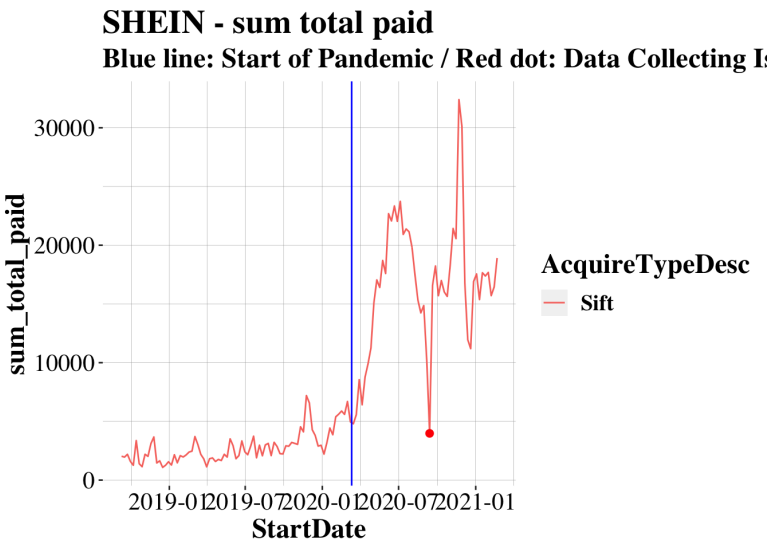


```
JimmyJohns_data <- subset(merged_data, MerchantName == "Jimmyjohns.com")

ggplot(data = JimmyJohns_data,
       aes(x = StartDate, y = receipt_count, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=JimmyJohns_data[which(JimmyJohns_data$problem_detected==1), ],aes(x=StartDate, y=receipt_c
ount), colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'JimmyJohns - Receipt Count', subtitle = "Blue line: Start of Pandemic / Red dot: Data Collec
ting Issue") + NPD_theme
```

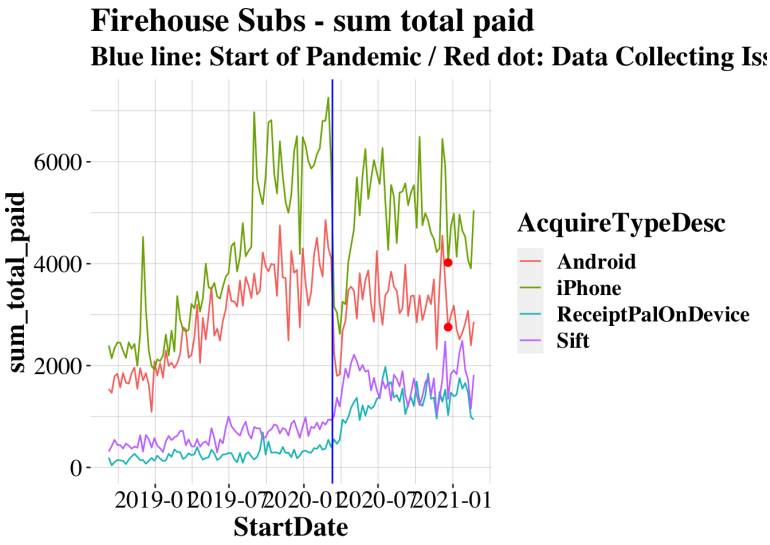


SHEIN, Firehouse Subs, JimmyJohns - Sum Total

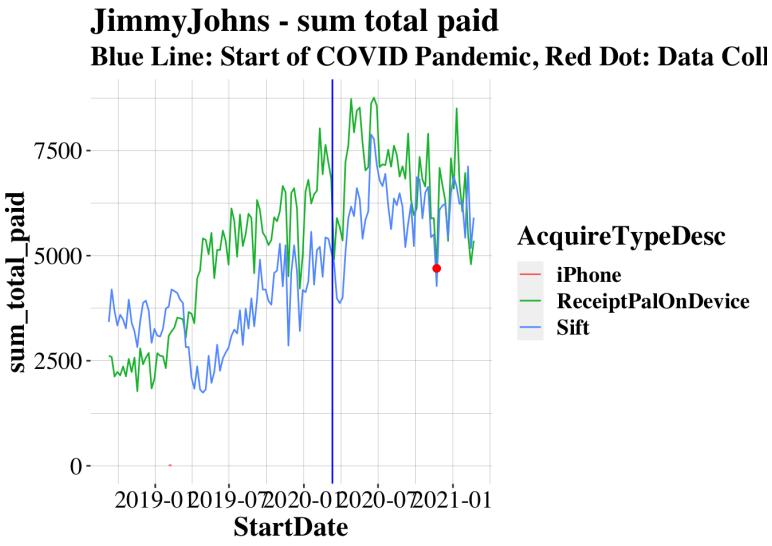
```
ggplot(data = shein_data,
       aes(x = StartDate, y = sum_total_paid, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=shein_data[which(shein_data$problem_detected==1), ],aes(x=StartDate, y=sum_total_paid), co
lour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'SHEIN - sum total paid', subtitle = "Blue line: Start of Pandemic / Red dot: Data Collecting
Issue")+ NPD_theme
```




```
ggplot(data = Firehouse_data,
       aes(x = StartDate, y = sum_total_paid, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=Firehouse_data[which(Firehouse_data$problem_detected==1), ],aes(x=StartDate, y=sum_total_p
aid), colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'Firehouse Subs - sum total paid', subtitle = "Blue line: Start of Pandemic / Red dot: Data C
ollecting Issue")+ NPD_theme
```

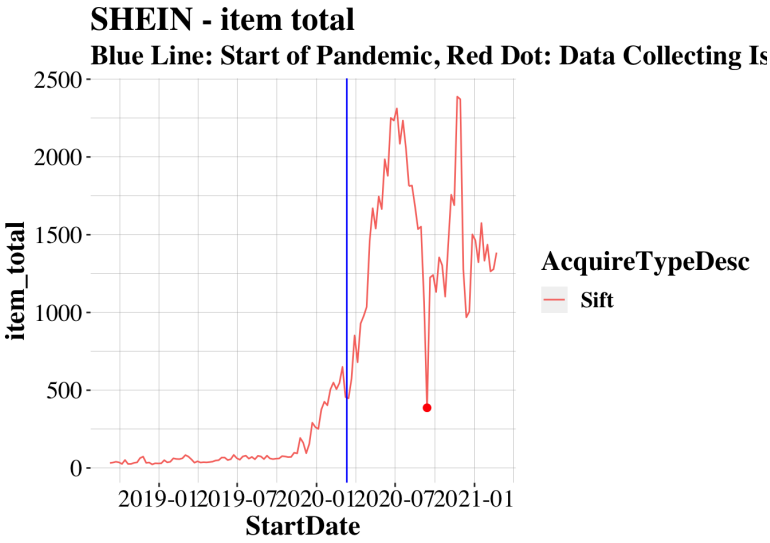


```
ggplot(data = JimmyJohns_data,
       aes(x = StartDate, y = sum_total_paid, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=JimmyJohns_data[which(JimmyJohns_data$problem_detected==1), ],aes(x=StartDate, y=sum_total
_paid), colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'JimmyJohns - sum total paid', subtitle = "Blue Line: Start of COVID Pandemic, Red Dot: Data
Collecting Issue")+ NPD_theme
```

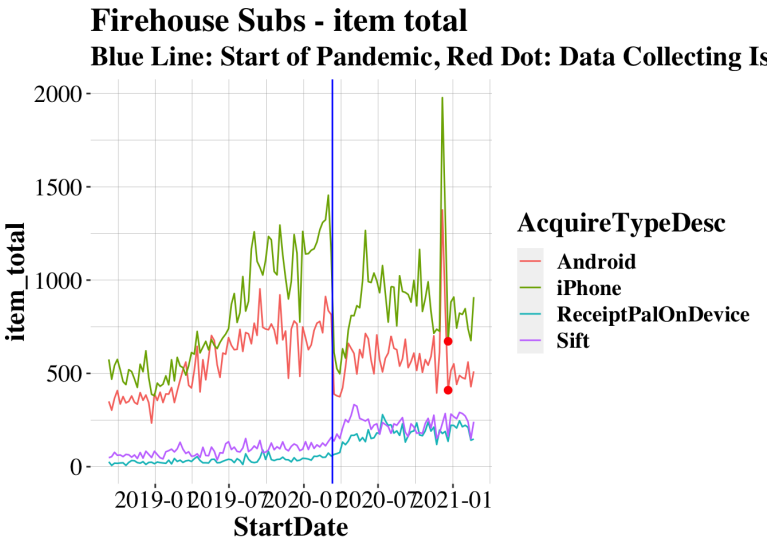


SHEIN, Firehouse Subs, JimmyJohns - item total

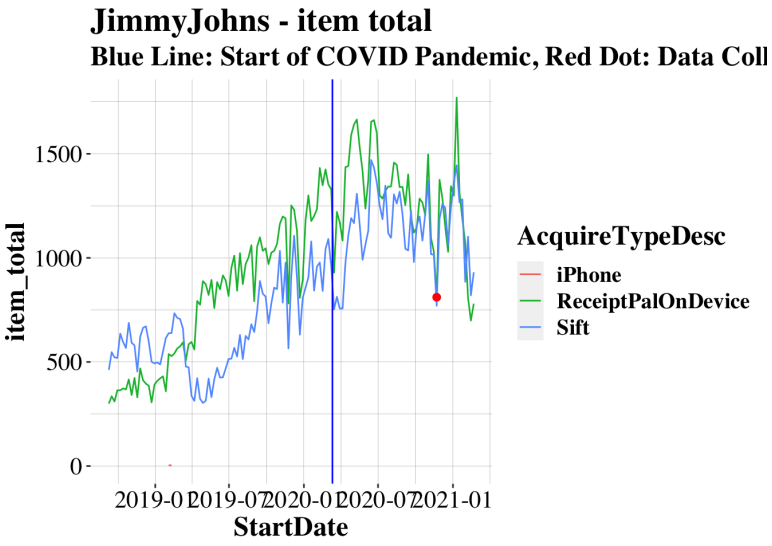
```
ggplot(data = shein_data,
       aes(x = StartDate, y = item_total, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=shein_data[which(shein_data$problem_detected==1), ],aes(x=StartDate, y=item_total), colour
="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'SHEIN - item total', subtitle = "Blue Line: Start of Pandemic, Red Dot: Data Collecting Issu
e")+ NPD_theme
```




```
ggplot(data = Firehouse_data,
       aes(x = StartDate, y = item_total, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=Firehouse_data[which(Firehouse_data$problem_detected==1), ],aes(x=StartDate, y=item_total)
, colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'Firehouse Subs - item total', subtitle = "Blue Line: Start of Pandemic, Red Dot: Data Collec
ting Issue")+ NPD_theme
```

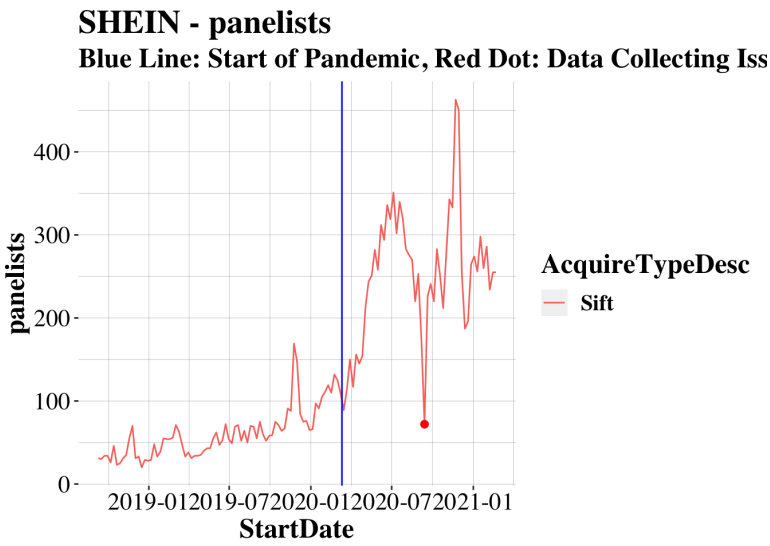


```
ggplot(data = JimmyJohns_data,
       aes(x = StartDate, y = item_total, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=JimmyJohns_data[which(JimmyJohns_data$problem_detected==1), ],aes(x=StartDate, y=item_tota
l), colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'JimmyJohns - item total', subtitle = "Blue Line: Start of COVID Pandemic, Red Dot: Data Coll
ecting Issue")+ NPD_theme
```

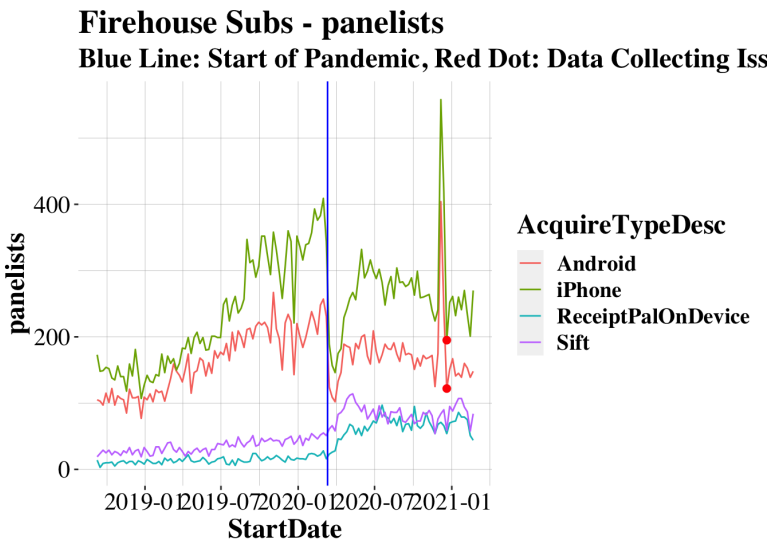


SHEIN, Firehouse Subs, JimmyJohns - panelist

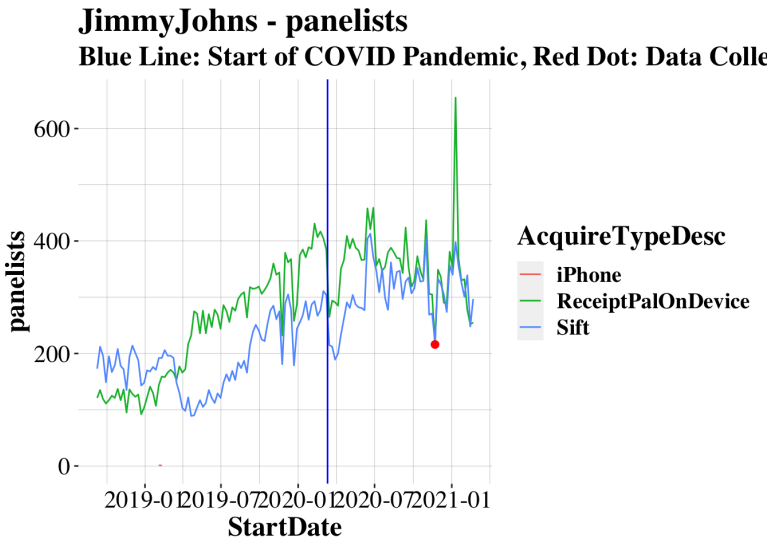
```
ggplot(data = shein_data,
       aes(x = StartDate, y = panelists, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=shein_data[which(shein_data$problem_detected==1), ],aes(x=StartDate, y=panelists), colour=
"red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'SHEIN - panelists', subtitle = "Blue Line: Start of Pandemic, Red Dot: Data Collecting Issue
")+ NPD_theme
```




```
ggplot(data = Firehouse_data,
       aes(x = StartDate, y = panelists, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=Firehouse_data[which(Firehouse_data$problem_detected==1), ],aes(x=StartDate, y=panelists),
colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'Firehouse Subs - panelists', subtitle = "Blue Line: Start of Pandemic, Red Dot: Data Collect
ing Issue")+ NPD_theme
```



```
ggplot(data = JimmyJohns_data,
       aes(x = StartDate, y = panelists, col = AcquireTypeDesc)) +
  geom_line() +
  geom_point(data=JimmyJohns_data[which(JimmyJohns_data$problem_detected==1), ],aes(x=StartDate, y=panelists
), colour="red", size=2) +
  geom_vline(xintercept = as.Date("2020-03-11"), col = "blue") +
  labs(title = 'JimmyJohns - panelists', subtitle = "Blue Line: Start of COVID Pandemic, Red Dot: Data Colle
cting Issue")+ NPD_theme
```

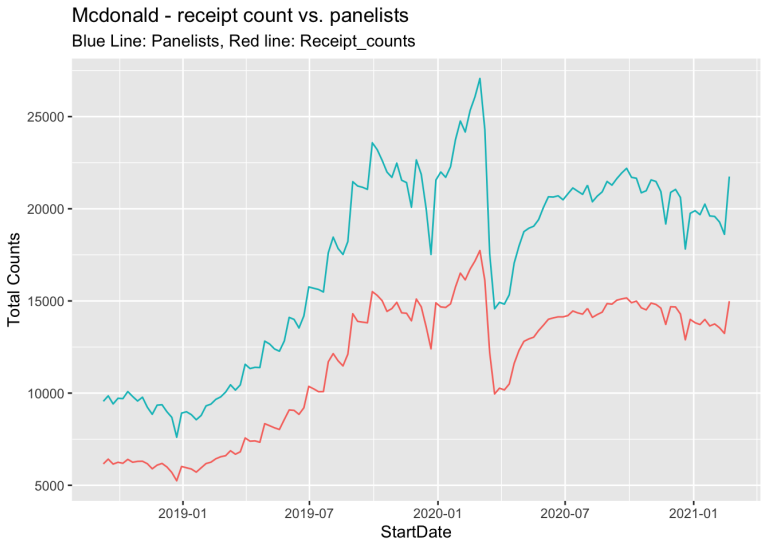


Sample Dataset creation

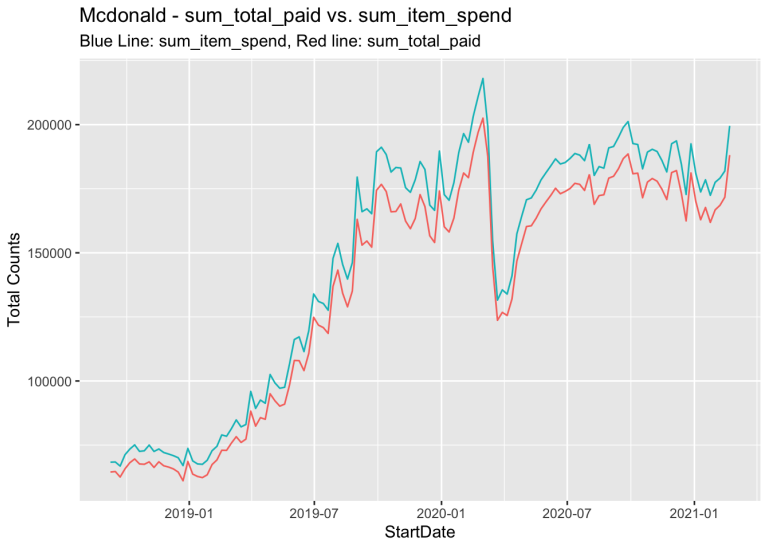
```
# Creating a Sample Data of McDonald's
sample_data <- subset(merged_data, MerchantName == "McDonald's")

# Dividing the Sample Data by the data sources
sample_data1 <- subset(sample_data, AcquireTypeID == 3)
sample_data2 <- subset(sample_data, AcquireTypeID == 4)
sample_data3 <- subset(sample_data, AcquireTypeID == 7)
sample_data4 <- subset(sample_data, AcquireTypeID == 8)

# Making datasets of the first differences for the 4 datasets created above
mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data1$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data1$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data1$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data1$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data1$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data1$panelists)
mcdonald_iphone <- as.data.frame(mcdonald_diff)

mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data2$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data2$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data2$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data2$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data2$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data2$panelists)
mcdonald_android <- as.data.frame(mcdonald_diff)

mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data3$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data3$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data3$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data3$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data3$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data3$panelists)
mcdonald_sift <- as.data.frame(mcdonald_diff)

mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data4$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data4$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data4$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data4$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data4$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data4$panelists)
mcdonald_mobile <- as.data.frame(mcdonald_diff)

mcdonald_iphone$StartDate <- sample_data1$StartDate[-1]
mcdonald_android$StartDate <- sample_data2$StartDate[-1]
mcdonald_sift$StartDate <- sample_data3$StartDate[-1]
mcdonald_mobile$StartDate <- sample_data4$StartDate[-1]
```

Variable Selection

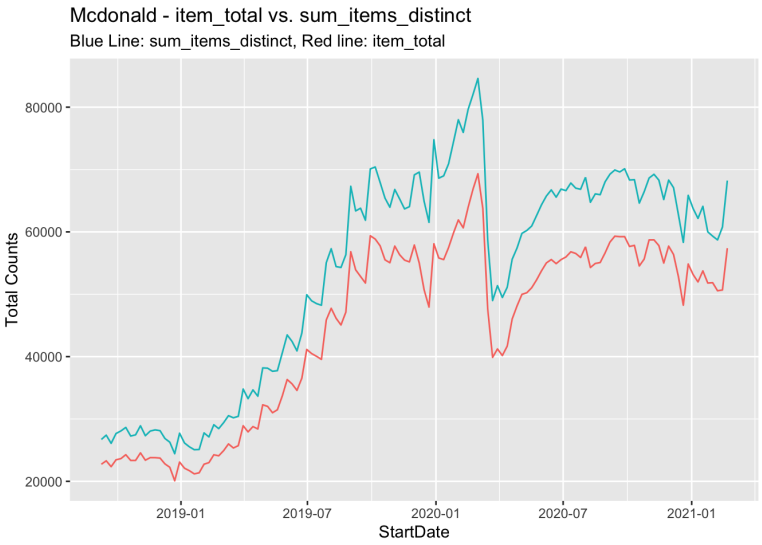
```
# Comparing time series for receipt_count & panelists
ggplot(sample_data1, aes(x=StartDate)) +
  geom_line(aes(y=receipt_count, color="red")) +
  geom_line(aes(y=panelists, color="blue")) +
  labs(title = 'Mcdonald - receipt count vs. panelists', subtitle = "Blue Line: Panelists, Red line: Receipt
_counts", y = "Total Counts") +
  theme(legend.position = "none")
```




```
# Comparing time series for sum_total_paid & sum_item_spend
ggplot(sample_data1, aes(x=StartDate)) +
  geom_line(aes(y=sum_total_paid, color="red")) +
  geom_line(aes(y=sum_item_spend, color="blue")) +
  labs(title = 'Mcdonald - sum_total_paid vs. sum_item_spend', subtitle = "Blue Line: sum_item_spend, Red li
ne: sum_total_paid", y = "Total Counts") +
  theme(legend.position = "none")
```




```
# Comparing time series for item_total & sum_items_distinct
ggplot(sample_data1, aes(x=StartDate)) +
  geom_line(aes(y=item_total, color="red")) +
  geom_line(aes(y=sum_items_distinct, color="blue")) +
  labs(title = 'Mcdonald - item_total vs. sum_items_distinct', subtitle = "Blue Line: sum_items_distinct, Re
d line: item_total", y = "Total Counts") +
  theme(legend.position = "none")
```




```
# Calculating the Average Correlations for the Key Variables
cor(merged_data$receipt_count, merged_data$panelists)
```

```
## [1] 0.9937866
```

```
cor(merged_data$sum_total_paid, merged_data$sum_item_spend)
```

```
## [1] 0.9990581
```

```
cor(merged_data$item_total, merged_data$sum_items_distinct)
```

```
## [1] 0.9991476
```

```
# Selecting only the Key Variables used for the Project (receipt_count, sum_total_paid, item_total)
mcdonald_iphone <- mcdonald_iphone[c(1:3,7)]
mcdonald_android <- mcdonald_android[c(1:3,7)]
mcdonald_sift <- mcdonald_sift[c(1:3,7)]
mcdonald_mobile <- mcdonald_mobile[c(1:3,7)]
```

6.2 First Difference Filter

First Differences Calculation (slope)

```
# Creating a Sample Data of McDonald's
sample_data <- subset(merged_data, MerchantName == "McDonald's")

# Dividing the Sample Data by the data sources
sample_data1 <- subset(sample_data, AcquireTypeID == 3)
sample_data2 <- subset(sample_data, AcquireTypeID == 4)
sample_data3 <- subset(sample_data, AcquireTypeID == 7)
sample_data4 <- subset(sample_data, AcquireTypeID == 8)

# Making datasets of the first differences for the 4 datasets created above
mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data1$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data1$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data1$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data1$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data1$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data1$panelists)
mcdonald_iphone <- as.data.frame(mcdonald_diff)

mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data2$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data2$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data2$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data2$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data2$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data2$panelists)
mcdonald_android <- as.data.frame(mcdonald_diff)

mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data3$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data3$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data3$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data3$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data3$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data3$panelists)
mcdonald_sift <- as.data.frame(mcdonald_diff)

mcdonald_diff <- list()
mcdonald_diff$receipt_count <- diff(sample_data4$receipt_count)
mcdonald_diff$sum_total_paid <- diff(sample_data4$sum_total_paid)
mcdonald_diff$item_total <- diff(sample_data4$item_total)
mcdonald_diff$sum_items_distinct <- diff(sample_data4$sum_items_distinct)
mcdonald_diff$sum_item_spend <- diff(sample_data4$sum_item_spend)
mcdonald_diff$panelists <- diff(sample_data4$panelists)
mcdonald_mobile <- as.data.frame(mcdonald_diff)

mcdonald_iphone$StartDate <- sample_data1$StartDate[-1]
mcdonald_android$StartDate <- sample_data2$StartDate[-1]
mcdonald_sift$StartDate <- sample_data3$StartDate[-1]
mcdonald_mobile$StartDate <- sample_data4$StartDate[-1]
```

First Differences Filter

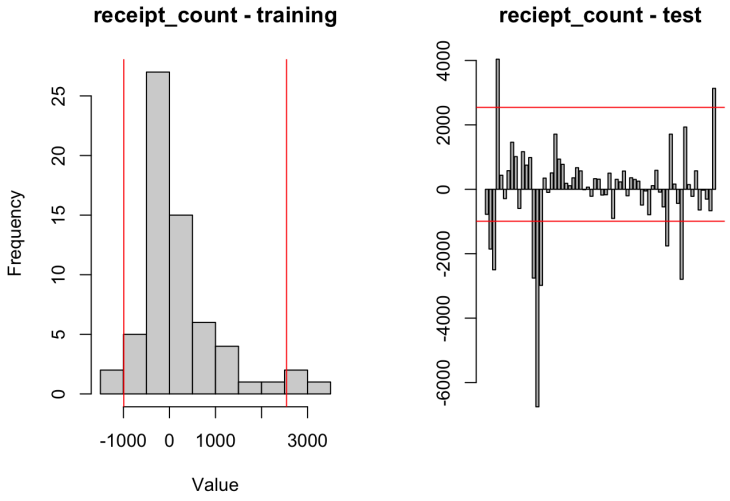
Mcdonald's - iPhone

```
library(dplyr)
library(zoo)
```

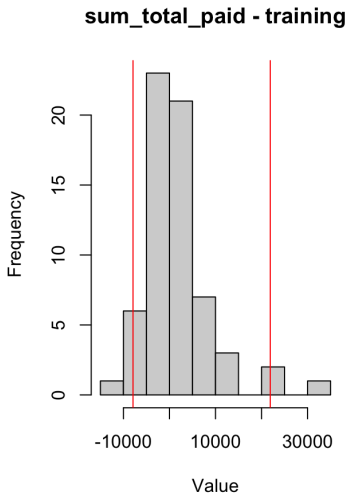
```
##
## Attaching package: 'zoo'
```

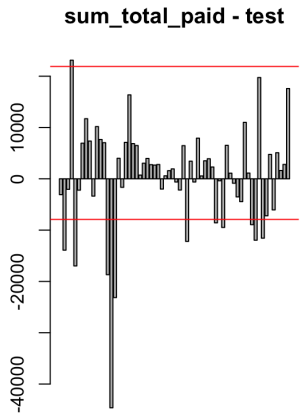
```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
# Dividing the Dataset into 1st and 2nd half where 1st half is training set and 2nd half is test set
mcdonald_iphone_1st <- mcdonald_iphone[1:64,]
mcdonald_iphone_2nd <- mcdonald_iphone[65:128,]

#Calculating the 95% Quantiles for the 3 variables in the training set
qts_rc <- quantile(mcdonald_iphone_1st$receipt_count, probs = c(0.025,0.975))
qts_stp <- quantile(mcdonald_iphone_1st$sum_total_paid, probs = c(0.025,0.975))
qts_it <- quantile(mcdonald_iphone_1st$item_total, probs = c(0.025,0.975))

#receipt count
par(mfrow = c(1,2))
hist(mcdonald_iphone_1st$receipt_count, nclass = 10, main = "receipt_count - training", xlab = "Value")
abline(v=qts_rc[1], col = 'red')
abline(v=qts_rc[2], col = 'red')

barplot(mcdonald_iphone_2nd$receipt_count, main = "reciept_count - test")
abline(h=qts_rc[1], col = 'red')
abline(h=qts_rc[2], col = 'red')
```

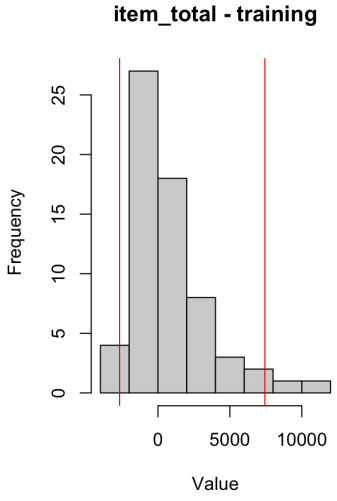


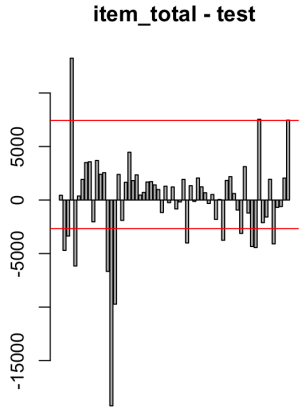

```
#sum total paid
par(mfrow = c(1,2))
hist(mcdonald_iphone_1st$sum_total_paid, nclass = 10, main = "sum_total_paid - training", xlab = "Value")
abline(v=qts_stp[1], col = 'red')
abline(v=qts_stp[2], col = 'red')

barplot(mcdonald_iphone_2nd$sum_total_paid, main = "sum_total_paid - test")
abline(h=qts_stp[1], col = 'red')
abline(h=qts_stp[2], col = 'red')
```






```
#items total
par(mfrow = c(1,2))
hist(mcdonald_iphone_1st$item_total, nclass = 10, main = "item_total - training", xlab = "Value")
abline(v=qts_it[1], col = 'red')
abline(v=qts_it[2], col = 'red')

barplot(mcdonald_iphone_2nd$item_total, main = "item_total - test")
abline(h=qts_it[1], col = 'red')
abline(h=qts_it[2], col = 'red')
```

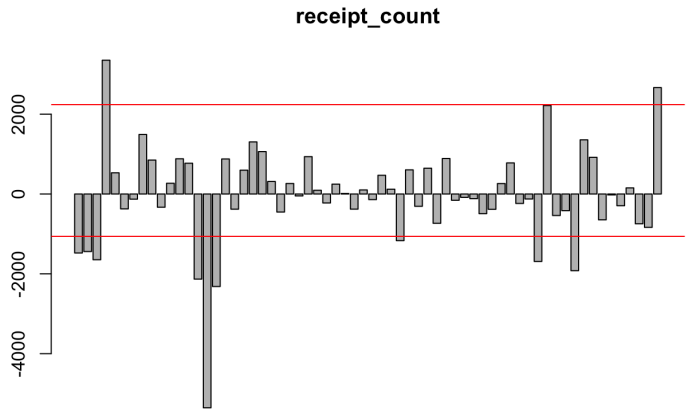




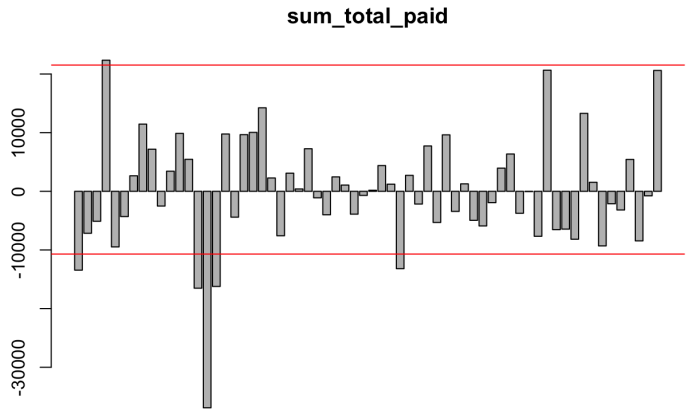
```
# Labeling the dates for when the anomalies occurred
mcdonald_iphone_2nd$rc_anom <- (mcdonald_iphone_2nd %>% mutate(i = receipt_count > qts_rc[2] | item_total <
qts_rc[1]))$i
mcdonald_iphone_2nd$stp_anom <- (mcdonald_iphone_2nd %>% mutate(i = sum_total_paid > qts_stp[2] | item_total
< qts_stp[1]))$i
mcdonald_iphone_2nd$it_anom <- (mcdonald_iphone_2nd %>% mutate(i = item_total > qts_it[2] | item_total < qts
_it[1]))$i
```

Mcdonald's - Android

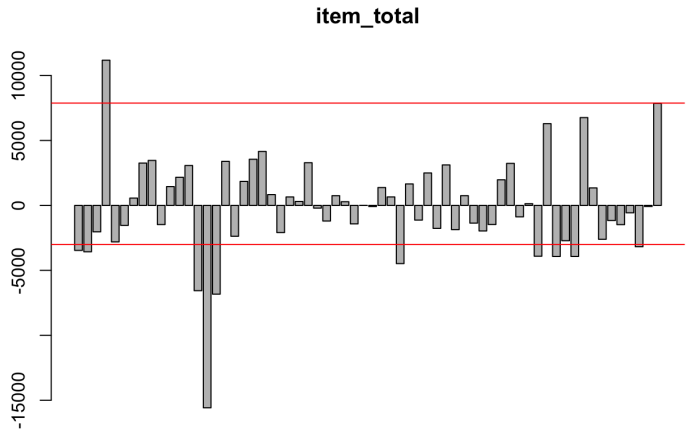
```
# Dividing the Dataset into 1st and 2nd half where 1st half is training set and 2nd half is test set
mcdonald_android_1st <- mcdonald_android[1:64,]
mcdonald_android_2nd <- mcdonald_android[65:128,]

#receipt count
qts_rc <- quantile(mcdonald_android_1st$receipt_count, probs = c(0.025,0.975))
barplot(mcdonald_android_2nd$receipt_count, main = "receipt_count")
abline(h=qts_rc[1], col = 'red')
abline(h=qts_rc[2], col = 'red')
```




```
#sum total paid
qts_stp <- quantile(mcdonald_android_1st$sum_total_paid, probs = c(0.025,0.975))
barplot(mcdonald_android_2nd$sum_total_paid, main = "sum_total_paid")
abline(h=qts_stp[1], col = 'red')
abline(h=qts_stp[2], col = 'red')
```



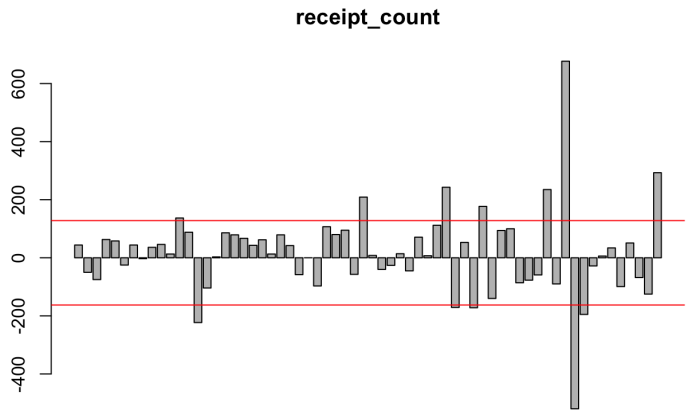

```
#items total
qts_it <- quantile(mcdonald_android_1st$item_total, probs = c(0.025,0.975))
barplot(mcdonald_android_2nd$item_total, main = "item_total")
abline(h=qts_it[1], col = 'red')
abline(h=qts_it[2], col = 'red')
```



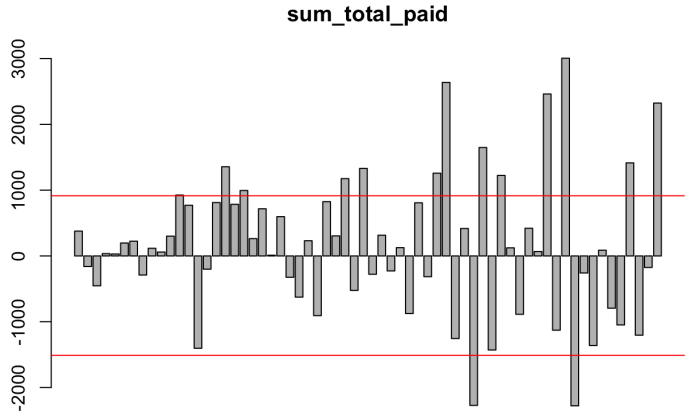

```
# Labeling the dates for when the anomalies occurred
mcdonald_android_2nd$rc_anom <- (mcdonald_android_2nd %>% mutate(i = receipt_count > qts_rc[2] | item_total
< qts_rc[1]))$i
mcdonald_android_2nd$stp_anom <- (mcdonald_android_2nd %>% mutate(i = sum_total_paid > qts_stp[2] | item_tot
al < qts_stp[1]))$i
mcdonald_android_2nd$it_anom <- (mcdonald_android_2nd %>% mutate(i = item_total > qts_it[2] | item_total < q
ts_it[1]))$i
```

Mcdonald's - Sift

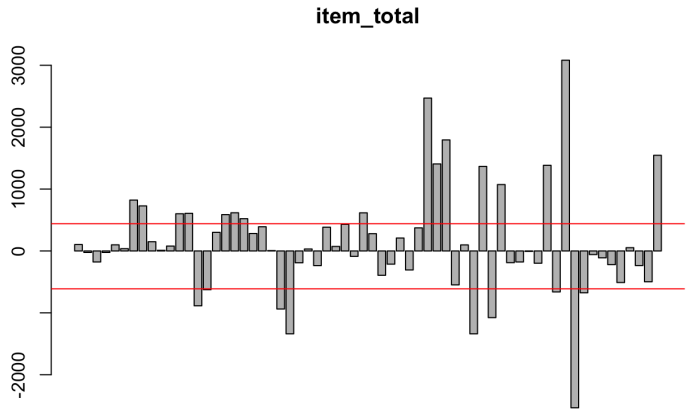
```r
# Dividing the Dataset into 1st and 2nd half where 1st half is training set and 2nd half is test set
mcdonald_sift_1st <- mcdonald_sift[1:64,]
mcdonald_sift_2nd <- mcdonald_sift[65:128,]

#receipt count
qts_rc <- quantile(mcdonald_sift_1st$receipt_count, probs = c(0.025,0.975))
barplot(mcdonald_sift_2nd$receipt_count, main = "receipt_count")
abline(h=qts_rc[1], col = 'red')
abline(h=qts_rc[2], col = 'red')
```




```r
#sum total paid
qts_stp <- quantile(mcdonald_sift_1st$sum_total_paid, probs = c(0.025,0.975))
barplot(mcdonald_sift_2nd$sum_total_paid, main = "sum_total_paid")
abline(h=qts_stp[1], col = 'red')
abline(h=qts_stp[2], col = 'red')
```



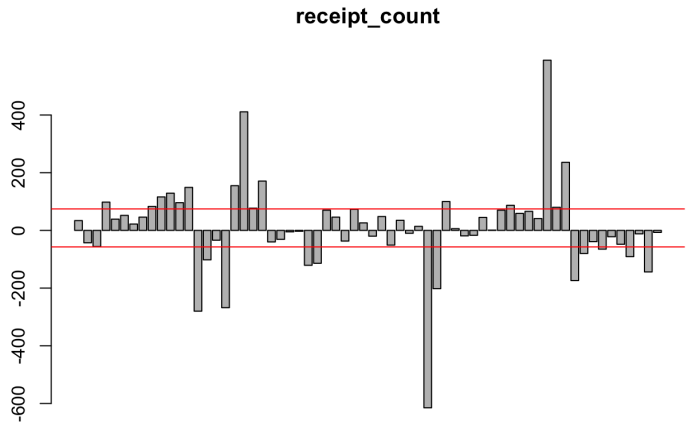
```
#items total
qts_it <- quantile(mcdonald_sift_1st$item_total, probs = c(0.025,0.975))
barplot(mcdonald_sift_2nd$item_total, main = "item_total")
abline(h=qts_it[1], col = 'red')
abline(h=qts_it[2], col = 'red')
```



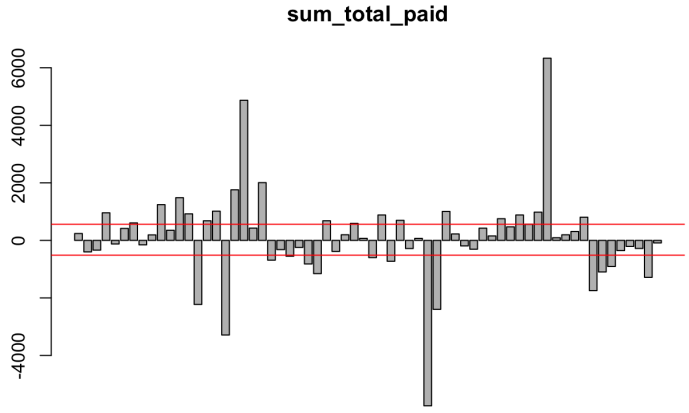

```
# Labeling the dates for when the anomalies occurred
mcdonald_sift_2nd$rc_anom <- (mcdonald_sift_2nd %>% mutate(i = receipt_count > qts_rc[2] | item_total < qts_
rc[1]))$i
mcdonald_sift_2nd$stp_anom <- (mcdonald_sift_2nd %>% mutate(i = sum_total_paid > qts_stp[2] | item_total < q
ts_stp[1]))$i
mcdonald_sift_2nd$it_anom <- (mcdonald_sift_2nd %>% mutate(i = item_total > qts_it[2] | item_total < qts_it[
1]))$i
```

Mcdonald's - recipepaidpal

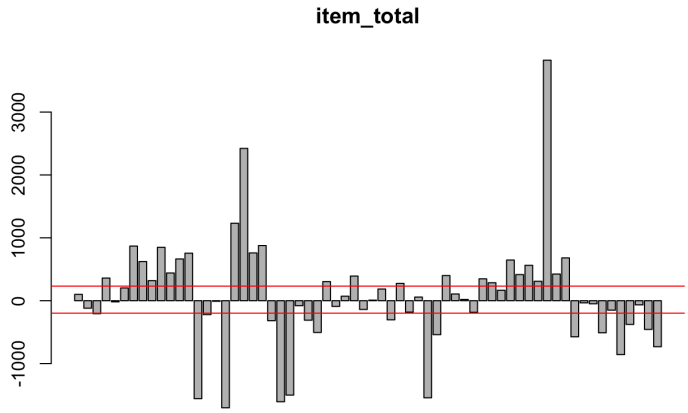
```
# Dividing the Dataset into 1st and 2nd half where 1st half is training set and 2nd half is test set
mcdonald_mobile_1st <- mcdonald_mobile[1:64,]
mcdonald_mobile_2nd <- mcdonald_mobile[65:128,]

#receipt count
qts_rc <- quantile(mcdonald_mobile_1st$receipt_count, probs = c(0.025,0.975))
barplot(mcdonald_mobile_2nd$receipt_count, main = "receipt_count")
abline(h=qts_rc[1], col = 'red')
abline(h=qts_rc[2], col = 'red')
```




```
#sum total paid
qts_stp <- quantile(mcdonald_mobile_lst$sum_total_paid, probs = c(0.025,0.975))
barplot(mcdonald_mobile_2nd$sum_total_paid, main = "sum_total_paid")
abline(h=qts_stp[1], col = 'red')
abline(h=qts_stp[2], col = 'red')
```




```
#items total
qts_it <- quantile(mcdonald_mobile_lst$item_total, probs = c(0.025,0.975))
barplot(mcdonald_mobile_2nd$item_total, main = "item_total")
abline(h=qts_it[1], col = 'red')
abline(h=qts_it[2], col = 'red')
```




```
# Labeling the dates for when the anomalies occurred
mcdonald_mobile_2nd$rc_anom <- (mcdonald_mobile_2nd %>% mutate(i = receipt_count > qts_rc[2] | item_total <
qts_rc[1]))$i
mcdonald_mobile_2nd$stp_anom <- (mcdonald_mobile_2nd %>% mutate(i = sum_total_paid > qts_stp[2] | item_total
< qts_stp[1]))$i
mcdonald_mobile_2nd$it_anom <- (mcdonald_mobile_2nd %>% mutate(i = item_total > qts_it[2] | item_total < qts
_it[1]))$i
```

6.3 Trimmed Moving Weighted Average Filter

6.3.1 Input Parameters Sanity Check

This section of code was implemented to ensure that the users only input parameters that were acceptable. If not, an error message would pop up to guide users to re-choose a parameter in the correct form.

```
df_clean = df.copy(deep = True)

res_arr = df_clean[response_column]
window = self.window
outlier = self.max_outliers
x = list(range(window-outlier-1))

if df_clean.shape[1] < 2:
    raise IndexError\
        ('Expected at least 2 dimentional dataframe, %d dimensions given.' % (df_clean.shape[1]))
if (scaling==True) and (df_clean.shape[1] < 3):
    raise IndexError\
        ('Expected at least 3 dimentional dataframe, %d dimensions given.' % (df_clean.shape[1]))
if (scaling==True) and (scaling_column==None):
    raise IndexError\
        ('Expected at least 3 dimentional dataframe, %d dimensions given.' % (df_clean.shape[1]))
if (outlier > np.around(window*0.2)) or (outlier < 0):
    raise IndexError\
        ('Please input an acceptable outlier range. You can input a minimum of 0 outlier,
        and a maximum of %d outliers.' % (np.around(window*0.2)))
```

6.3.2 Deletion of Outliers in a Given Window

In order to remove outliers in rolling windows, total_list was used to store list of all possible rolling windows with size n. For each rolling window, the median was firstly computed. The absolute distance between each point and the median was then computed and ranked. Based on user's max_outlier (x) preference, the top x points which had the biggest distance would be dropped from the list.

```
as_strided = np.lib.stride_tricks.as_strided
total_lst = as_strided(res_arr, (len(res_arr) - (window - 1), window), (res_arr.values.strides * 2)).tolist()
median_lst = list(res_arr.rolling(window).median())
del median_lst[:window-1]

absolute = []
for i in range(len(total_lst)):
    dists = []
    for j in range(window):
        dist = abs(total_lst[i][j]-median_lst[i])
        dists.append(dist)
    absolute.append(dists)

remove_idxs = []
for abss in absolute:
    sorted_abss = sorted(((v, i) for i, v in enumerate(abss)), reverse=True)

    remove_idx = []
    for i, (value, index) in enumerate(sorted_abss):
        if i <= outlier:
            remove_idx.append(index)
    remove_idxs.append(remove_idx)

for i in range(len(total_lst)):
    for j in range(window):
        if j in remove_idxs[i]:
            total_lst[i][j] = 0
    total_lst[i] = list(filter(lambda num:num != 0, total_lst[i]))
```

6.3.3 Half Normal PDF Weights Assignment

Now having the modified total_list after dropping outliers, the next step was to assign weight to each points remaining in the rolling windows. The weights were calculated and mapped to each point using the reversed half-normal distribution. Then the weighted averages were calculated accordingly.

```
# https://arxiv.org/pdf/1610.00541.pdf
pdf_lst = [math.sqrt(2/math.pi)/((window-outlier)/3)*math.exp((-i**2)/(2*((window-outlier)/3)**2)) for i in x]
# reverse list to assign more value to near-current points
pdf_reverse = list(reversed(pdf_lst))

w_avg = []
for sub_lst in total_lst:
    value = sum([a*b for a,b in zip(sub_lst,pdf_reverse)])
    w_avg.append(value/sum(pdf_reverse))
```

6.3.4 Weighted Average Columns

Once all the weighted averages were computed, the values were then added to the actual data frame 4 times. In particular, a shifting effect was applied to each w_avg column to ensure that the weighted averages were matched up with the actual period of time in the data set, with w_avg_1 started at (initial time + window size), w_avg_1 started at (initial time + window size + 1) and so on.

```
w_avg = pd.DataFrame({'w_avg_1': np.around(w_avg, 2)})

df_w_avg = pd.concat([df_clean,w_avg], axis=1)

df_w_avg['w_avg_1'] = df_w_avg.w_avg_1.shift(int(window))
df_w_avg['w_avg_2'] = df_w_avg.w_avg_1.shift(1)
df_w_avg['w_avg_3'] = df_w_avg.w_avg_2.shift(1)
df_w_avg['w_avg_4'] = df_w_avg.w_avg_3.shift(1)
```

6.3.5 Standard Deviation Columns

need to add more code for standard deviation computation. Similar to weighted average, once all the standard deviations were computed, the values were then added to the actual data frame 4 times. In particular, a shifting effect was applied to each w_std column to ensure that the standard deviations were matched up with the actual period of time in the data set, with w_std_1 started at (initial time + window size), w_std_1 started at (initial time + window size + 1) and so on.

```
w_std = pd.DataFrame({'w_std_1': np.around(w_std, 2)})

df_w_avg_std = pd.concat([df_w_avg,w_std], axis=1)
df_w_avg_std['w_std_1'] = df_w_avg_std.w_std_1.shift(int(window))
df_w_avg_std['w_std_2'] = df_w_avg_std.w_std_1.shift(1)
df_w_avg_std['w_std_3'] = df_w_avg_std.w_std_2.shift(1)
df_w_avg_std['w_std_4'] = df_w_avg_std.w_std_3.shift(1)
```

6.3.6 Defining Bounds for Anomaly Detection

Now having both weighted average values and standard deviation values ready, the 3 sigma boundaries were calculated and appended to the same data frame for anomaly filtering purpose.

```
# filter 1
df_w_avg_std['pos_std_w_1'] = df_w_avg_std.w_avg_1 + df_w_avg_std.w_std_1
df_w_avg_std['neg_std_w_1'] = df_w_avg_std.w_avg_1 - df_w_avg_std.w_std_1
df_w_avg_std['pos_std_2_w_1'] = df_w_avg_std.w_avg_1 + 2*df_w_avg_std.w_std_1
df_w_avg_std['neg_std_2_w_1'] = df_w_avg_std.w_avg_1 - 2*df_w_avg_std.w_std_1
df_w_avg_std['pos_std_3_w_1'] = df_w_avg_std.w_avg_1 + 3*df_w_avg_std.w_std_1
df_w_avg_std['neg_std_3_w_1'] = df_w_avg_std.w_avg_1 - 3*df_w_avg_std.w_std_1
df_w_avg_std['pos_std_4_w_1'] = df_w_avg_std.w_avg_1 + 4*df_w_avg_std.w_std_1
df_w_avg_std['neg_std_4_w_1'] = df_w_avg_std.w_avg_1 - 4*df_w_avg_std.w_std_1
# filter 2
df_w_avg_std['pos_std_w_2'] = df_w_avg_std.w_avg_2 + df_w_avg_std.w_std_2
df_w_avg_std['neg_std_w_2'] = df_w_avg_std.w_avg_2 - df_w_avg_std.w_std_2
df_w_avg_std['pos_std_2_w_2'] = df_w_avg_std.w_avg_2 + 2*df_w_avg_std.w_std_2
df_w_avg_std['neg_std_2_w_2'] = df_w_avg_std.w_avg_2 - 2*df_w_avg_std.w_std_2
df_w_avg_std['pos_std_3_w_2'] = df_w_avg_std.w_avg_2 + 3*df_w_avg_std.w_std_2
df_w_avg_std['neg_std_3_w_2'] = df_w_avg_std.w_avg_2 - 3*df_w_avg_std.w_std_2
df_w_avg_std['pos_std_4_w_2'] = df_w_avg_std.w_avg_2 + 4*df_w_avg_std.w_std_2
df_w_avg_std['neg_std_4_w_2'] = df_w_avg_std.w_avg_2 - 4*df_w_avg_std.w_std_2
# filter 3
df_w_avg_std['pos_std_w_3'] = df_w_avg_std.w_avg_3 + df_w_avg_std.w_std_3
df_w_avg_std['neg_std_w_3'] = df_w_avg_std.w_avg_3 - df_w_avg_std.w_std_3
df_w_avg_std['pos_std_2_w_3'] = df_w_avg_std.w_avg_3 + 2*df_w_avg_std.w_std_3
df_w_avg_std['neg_std_2_w_3'] = df_w_avg_std.w_avg_3 - 2*df_w_avg_std.w_std_3
df_w_avg_std['pos_std_3_w_3'] = df_w_avg_std.w_avg_3 + 3*df_w_avg_std.w_std_3
df_w_avg_std['neg_std_3_w_3'] = df_w_avg_std.w_avg_3 - 3*df_w_avg_std.w_std_3
df_w_avg_std['pos_std_4_w_3'] = df_w_avg_std.w_avg_3 + 4*df_w_avg_std.w_std_3
df_w_avg_std['neg_std_4_w_3'] = df_w_avg_std.w_avg_3 - 4*df_w_avg_std.w_std_3
# filter 4
df_w_avg_std['pos_std_w_4'] = df_w_avg_std.w_avg_4 + df_w_avg_std.w_std_4
df_w_avg_std['neg_std_w_4'] = df_w_avg_std.w_avg_4 - df_w_avg_std.w_std_4
df_w_avg_std['pos_std_2_w_4'] = df_w_avg_std.w_avg_4 + 2*df_w_avg_std.w_std_4
df_w_avg_std['neg_std_2_w_4'] = df_w_avg_std.w_avg_4 - 2*df_w_avg_std.w_std_4
df_w_avg_std['pos_std_3_w_4'] = df_w_avg_std.w_avg_4 + 3*df_w_avg_std.w_std_4
df_w_avg_std['neg_std_3_w_4'] = df_w_avg_std.w_avg_4 - 3*df_w_avg_std.w_std_4
df_w_avg_std['pos_std_4_w_4'] = df_w_avg_std.w_avg_4 + 4*df_w_avg_std.w_std_4
df_w_avg_std['neg_std_4_w_4'] = df_w_avg_std.w_avg_4 - 4*df_w_avg_std.w_std_4
```

6.3.7 Detection of Outliers based on 4 filters

In details, a new point was categorized as an anomaly when it landed outside of 3 standard deviations in any of the 4 filters defined. The outliers were then recorded in binary forms, with 1 as an indicator of anomaly.

```
for i in range(len(df_w_avg_std)):
    if (df_w_avg_std.loc[i, response_column] > df_w_avg_std.loc[i, 'pos_std_4_w_1']) or
    (df_w_avg_std.loc[i, response_column] < df_w_avg_std.loc[i, 'neg_std_4_w_1']):
        df_w_avg_std.loc[i, 'outlier_1'] = 1
    else:
        df_w_avg_std.loc[i, 'outlier_1'] = 0

    if (df_w_avg_std.loc[i, response_column] > df_w_avg_std.loc[i, 'pos_std_4_w_2']) or
    (df_w_avg_std.loc[i, response_column] < df_w_avg_std.loc[i, 'neg_std_4_w_2']):
        df_w_avg_std.loc[i, 'outlier_2'] = 1
    else:
        df_w_avg_std.loc[i, 'outlier_2'] = 0

    if (df_w_avg_std.loc[i, response_column] > df_w_avg_std.loc[i, 'pos_std_4_w_3']) or
    (df_w_avg_std.loc[i, response_column] < df_w_avg_std.loc[i, 'neg_std_4_w_3']):
        df_w_avg_std.loc[i, 'outlier_3'] = 1
    else:
        df_w_avg_std.loc[i, 'outlier_3'] = 0

    if (df_w_avg_std.loc[i, response_column] > df_w_avg_std.loc[i, 'pos_std_4_w_4']) or
    (df_w_avg_std.loc[i, response_column] < df_w_avg_std.loc[i, 'neg_std_4_w_4']):
        df_w_avg_std.loc[i, 'outlier_4'] = 1
    else:
        df_w_avg_std.loc[i, 'outlier_4'] = 0


self.df_w_avg_std = df_w_avg_std

return df_w_avg_std
```

6.3.8 Anomaly Detection Method

This function served as an initializer for the trimmed moving weighted average filter to run and output anomalies in a data frame as the result. The data frame would then be called by the plot function to output anomaly detection visualizations for any given time series data.

```
df = pd.DataFrame(df)

if self.method == 'weighted_average':
    self.difference(df, response_column, anom_detect=True)
    df_out = self.moving_weighted_average(df, response_column, time_column, scaling, scaling_column)
    self.results = df_out

if self.method == 'median':
    self.moving_median(df, response_column, time_column, scaling, scaling_column)
    df_out = self.deviation_stats_median(response_column, time_column, scaling, scaling_column)
    self.results = df_out

# create subsets of outliers based on filters
if anom_detect:
    anom_points_1 = df_out.loc[df_out['outlier_1'] > 0]
    anom_points_2 = df_out.loc[df_out['outlier_2'] > 0]
    anom_points_3 = df_out.loc[df_out['outlier_3'] > 0]
    anom_points_4 = df_out.loc[df_out['outlier_4'] > 0]

    ## TODO: MAKE IT BETTER
    self.anom_points_1 = anom_points_1
    self.anom_points_2 = anom_points_2
    self.anom_points_3 = anom_points_3
    self.anom_points_4 = anom_points_4

    return anom_points_1, anom_points_2, anom_points_3, anom_points_4
```

6.3.9 Anomaly Detection Visualization

Once all outliers were identified, matplotlib was utilized to visualize anomalies detected using all filters. Four different shapes and colors were used on anomalies to distinguish which filter detected them.

```
df = self.results

anom_points_1 = self.anom_points_1
anom_points_2 = self.anom_points_2
anom_points_3 = self.anom_points_3
anom_points_4 = self.anom_points_4

fig, ax1 = plt.subplots(1, 1,figsize=(15, 8))
ax1.plot(list(df.index),df[response_column],'b.',label=str(response_column))
ax1.plot(list(df.index),df.w_avg_1,'r',label='Moving Trimmed Weighted Average (Filter 1)')
ax1.fill_between(df.index,df.pos_std_w_1,df.neg_std_w_1,color='red',alpha=0.3,label='1Sigma')
ax1.fill_between(df.index,df.pos_std_2_w_1,df.neg_std_2_w_1,color='red',alpha=0.2,label='2Sigma')
ax1.fill_between(df.index,df.pos_std_3_w_1,df.neg_std_3_w_1,color='red',alpha=0.1,label='3Sigma')

ax1.plot(list(anom_points_1.index),anom_points_1[response_column],'rs', markersize=18, label='Anomalous Points (F-1)')
ax1.plot(list(anom_points_2.index),anom_points_2[response_column],'go', markersize=15, label='Anomalous Points (F-2)')
ax1.plot(list(anom_points_3.index),anom_points_3[response_column],'k*', markersize=10, label='Anomalous Points (F-3)')
ax1.plot(list(anom_points_4.index),anom_points_4[response_column],'y.', markersize=5, label='Anomalous Points (F-4)')

ax1.set_xlabel('Week')
ax1.set_ylabel(data_label)
ax1.set_title('Data with Anomalies starred'+' ('+str(response_column)+')')
ax1.set_xlim(left=left,right=right)
ax1.set_ylim(bottom=bottom,top=top)
ax1.legend()
```

6.4 Python Interface

As per NPD's request we packaged our anomaly detection functionality in a python package PyAnomalyDetect. The package was written with the principles of single responsibility in that class, method and function performs one specific action and tracability. To reflect these principles we divided the anomaly detection process into two distinct steps; data preparation and analysis.

6.4.1 Data Preparation

What user does: 1. Subset by merchant and acquire type

What pacakge does: 2. everything else

To ensure that our package was broadly applicable to the data analyzed by NPD. The data provided was defined primarily by the merchants, data acquisition methodology and a collection of six variables. To provide optimal functionality we designed a data structure, AnomalyDf, to capture the relevant information needed to prepare the data for analysis. The data structure can be defined by the following parameters within the __init__ function:

```
def __init__(self, data: pd.DataFrame,
             response_column: str,
             time_column: str, scale_column: str = ""):
```

- data: A pandas.DataFrame with $p \geq 2$ columns and $n \geq 3$ rows. When provided it is assumed that the data will be reflective of a unique merchant-acquisition combination. We will refer to the data parameter as "the raw data" henceforth and will distinguish this from "processed data" produced in the constructor function.
- response_column: A python string indicating the name of the column to be considered as the response variable. As this parameter will be used to subset the provided data the constraint response_column $\in$ columns(data).
- time_column: A python string indicating the name of the column of data containing the time variable. As this parameter will be used to subset the provided data the constraint time_column $\in$ columns(data).
- scale_column: A python string indicating the name of the column used to scale the measures of spread. Defaults to None.

Provided the above parameters the AnomalyDf constructor will create an instance of the AnomalyDf class. In imagining and implementing the class we sought to balance processing work done by the user while standardizing the interface. Our initial exploration and calculations saw repeated tracking of the columns to be examined and therefore implemented the relevant columns as parameters. The `get_data(column)` method is used almost exclusively to access the data to reduce the repetition of the column names. The interface furthermore removes the burden from the user to format the data in a specific way.

- `df`: A `pandas.DataFrame` with n rows and $p \in [2,\ 3]$ columns. The columns present are the columns corresponding to the time column, response variable and the scaling column, if applicable. Note that this variable is public to allow for easy examination of the data but should be accessed with the `get_data(variable)` function.
- `sd`: A `numpy.array` with dimension `(n,)`. This field is initialized to an array of zeroes (produced by `self.sd = numpy.zeros(self.df.shape[0])`. We designated the calculation of the spread measures to the `update_sd` method both to adhere to the principle of single responsibility and to ensure that the user can change the standard deviation calculation without needing to recreate the instance.
- `diff_taken`: A boolean indicating if the first difference of the processed data (`df`) was computed. If `True` then the first differences of the response column may be found in the `diff` field.

The following methods are needed to modify, access and ultimately utilize the data in a standardized manner. The seemingly trivial methods (such as `nrows()` and `ncols`) were designed to reduce the user's interacions with the underlying data and discourage modification of the data.

def nrows(self):

This method takes no arguments and returns the number of rows in the processed data (`self.df`).

def ncols(self):

This method takes no arguments and returns the number of columns in the processed data (`self.df`). Note that this method will always return either two or three.

def get_copy(self):

This method returns a deep copy of the processed data. Users are strongly encouraged to use this function when accessing or modifying the data to prevent aliasing issues that can be abundant in pandas.

def get_data(self, column: **str**):

This method takes a string `column` argument and returns an $n \times 1$ `pandas.DataFrame` with data of the indicated `column` type. `column` must be one of `"response"`, `"time"` or `"scale"` indicating the response, timing and scaling columns respectively.

def update_sd(self, sd_method: **str**, window: **int**):

This method is a non-constructive method that modifies the `sd` and $\texttt{sd}_i initialized functions to represent the measure of sprea$

Technical Appendix: A: EDA variable selection - Wonil B: First difference - Frank C: Weighted Average - Ning Pragya D: Interface - Frank

- weighted average - rolling window (sort in lists of list) - compute median, drop max outliers - assign weights (based on half normal) to points in window [equation] - compute the weighted average [equation]

- Standard deviation - rolling window (sort in lists of list) - compute first difference [equation] - compute variance tau [equation] - apply MAD on variance [equation]