1. Introduction

1.1 Objective

- Apply multiple *Exploratory Data Analysis* methods on Grocery dataset.
- Apply data mining algorithm like *Market Basket Analysis* on the same dataset to find out hidden association between products, and to find consumer buying behaviours.
- Apply deep learning methodology using *word2vec* model to predict the highly corelated products within a vector space.
- Demonstrate how such statistical methods can help small/medium businesses to grow their sales and sustainability.

1.2 Problem Definition

- Retail industry is highly complex in nature because of the multiple diversification in the type of products, and also because of very high quantity of products within the market.
- When consumers buy products from retail stores, there are hidden association between products that are most of the time neglected.
- Using statistical methods, such hidden insights can be derived which could result in the increase of sales of associated items.
- Multiple new statistical and machine learning methods are being researched on a regular basis to find better results and accuracy
- Aim is that using such algorithms, retailers can grow their businesses by storing and utilizing transaction data of consumers.

1.3 Scope

- *Exploratory Data Analysis employs a variety of techniques to:*
  - Maximize insight into datasets
  - Uncover underlying structure of datasets
  - Extract important variables
  - Detect outliers and anomalies
- *Market Basket Analysis can be used to analyse:*
  - Association of grocery transactions
  - Credit card purchases
  - Telephone calling patterns
  - Fraudulent medical insurance claims
- *word2vec algorithm can be used to predict:*
  - Sentiment Analysis
  - Speech recognition
  - Information retrieval
  - Question answering

1.4 Hardware and Software Specifications

Hardware :

- RAM of more than 4GB
- GPU of more than 2GB (optional)

Software:

- Windows/MacOS/Linux operating system
- R
- Rstudio
- Python
- Internet Connectivity

2. Exploratory Data Analysis

2.1 Taking a Look at the Data:

Before deep diving into the data with any kind of predictive analysis, let's take a look at the Data provided by Instacart. There are total 6 preprocessed CSV files provided :

```
> head(orders)
   order_id user_id eval_set order_number order_dow order_hour_of_day days_since_prior_order
1:  2539329       1    prior            1         2                 8                    NA
2:  2398795       1    prior            2         3                 7                    15
3:   473747       1    prior            3         3                12                    21
4:  2254736       1    prior            4         4                 7                    29
5:   431534       1    prior            5         4                15                    28
6:  3367565       1    prior            6         2                 7                    19
>
```

orders (3.4m rows, 206k users):

- order_id: order identifier
- user_id: customer identifier
- eval_set: which evaluation set this order belongs in (see SET described below)
- order_number: the order sequence number for this user (1 = first, n = nth)
- order_dow: the day of the week the order was placed on
- order_hour_of_day: the hour of the day the order was placed on
- days_since_prior: days since the last order, capped at 30 (with NAs for order_number = 1)

```
> head(products)
   product_id                                                product_name aisle_id department_id
1:          1                                  Chocolate Sandwich Cookies       61            19
2:          2                                           All-Seasons Salt      104            13
3:          3                       Robust Golden Unsweetened Oolong Tea       94             7
4:          4 Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce       38             1
5:          5                                 Green Chile Anytime Sauce        5            13
6:          6                                            Dry Nose Oil       11            11
>
```

products (50k rows):

- product_id: product identifier
- product_name: name of the product
- aisle_id: foreign key
- department_id: foreign key

```
> head(aisles)
   aisle_id                         aisle
1:        1      prepared soups salads
2:        2          specialty cheeses
3:        3        energy granola bars
4:        4                instant foods
5:        5 marinades meat preparation
6:        6                         other
>
```

aisles (134 rows):

- aisle_id: aisle identifier
- aisle: the name of the aisle

```
> head(departments)
   department_id    department
1:             1        frozen
2:             2         other
3:             3        bakery
4:             4       produce
5:             5       alcohol
6:             6 international
>
```

departments (21 rows):

- department_id: department identifier
- department: the name of the department

```
> head(order_products_train)
   order_id product_id add_to_cart_order reordered
1:        1      49302                 1         1
2:        1      11109                 2         1
3:        1      10246                 3         0
4:        1      49683                 4         0
5:        1      43633                 5         1
6:        1      13176                 6         0
> head(order_products_prior)
   order_id product_id add_to_cart_order reordered
1:        2      33120                 1         1
2:        2      28985                 2         1
3:        2       9327                 3         0
4:        2      45918                 4         1
5:        2      30035                 5         0
6:        2      17794                 6         1
>
```

order_products__SET (30m+ rows):

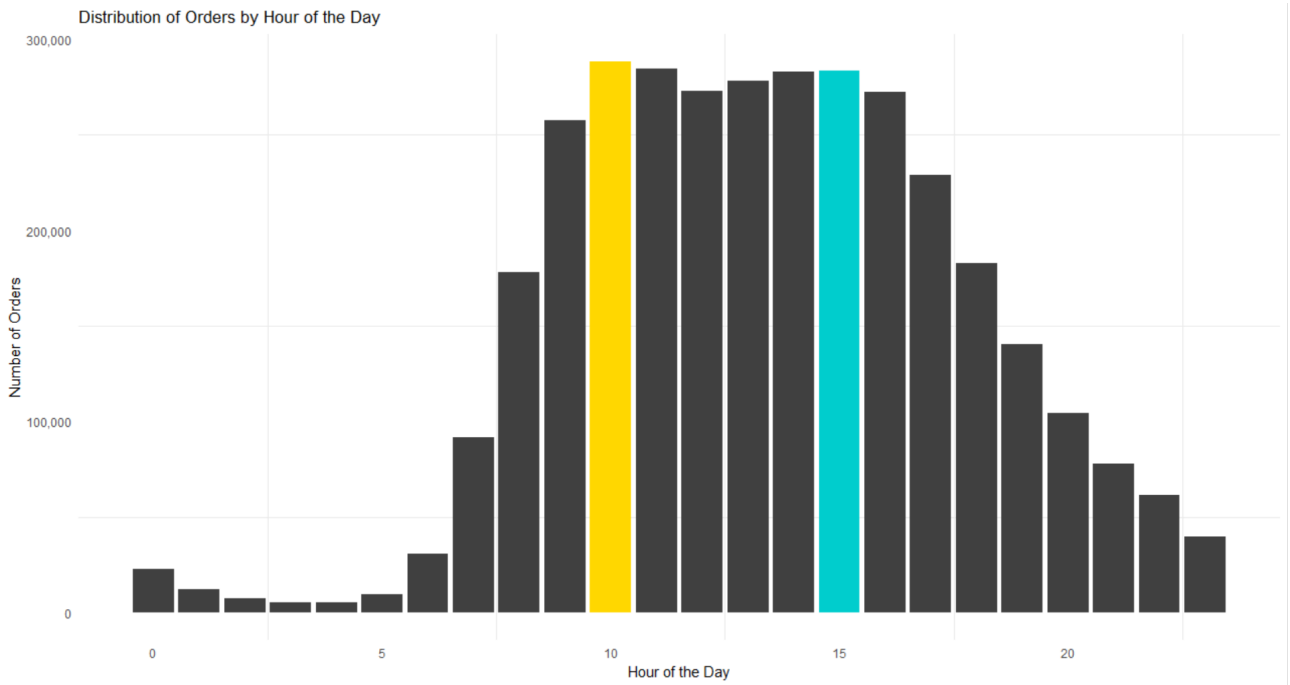
- order_id: foreign key
- product_id: foreign key
- add_to_cart_order: order in which each product was added to cart
- reordered: 1 if this product has been ordered by this user in the past, 0 otherwise

where SET is one of the four following evaluation sets (eval_set in orders):

- "prior": orders prior to that users most recent order (~3.2m orders)
- "train": training data supplied to participants (~131k orders)

2.2. Data Analysis & Visualization:

1. How many Orders by the Hour of the Day?



Output:

Most orders are made between the hours 09 and 16. But does this pattern exist over all days of the week?

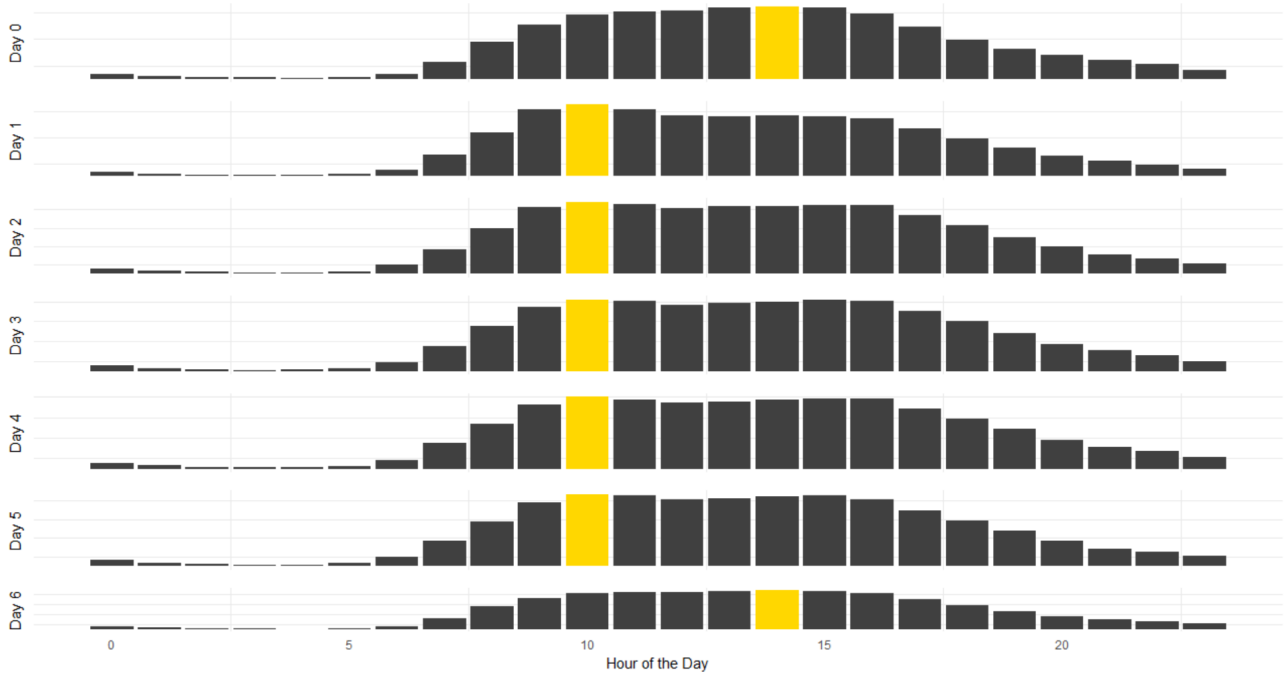
Process:

To generate this visualization, the dataset *Orders* is used and the column named *order_hour_of_the_day* is taken on the x-axis and generated a bar plot showing total *Number of Orders* on the y-axis with respect to the values in x-axis. To generate such visualization, the library *ggplot2* is used.

Code:

```
ggplot(orders, aes(x = order_hour_of_day)) +
  geom_bar(fill = c(rep("grey25", 10), "gold", rep("grey25", 4),
                    "cyan3",rep("grey25", 8))) +
  theme_minimal() +
  theme(axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "none",
        panel.grid.major = element_blank()) +
  labs(x = "Hour of the Day",
       y = "Number of Orders",
       title = "Distribution of Orders by Hour of the Day") +
  scale_y_continuous(labels = comma)
```

2. How many Orders by Hour and Day of the Week?



Output :

Here, the pattern in *(1)* is no longer observed. Instead, Day 1 to Day 5 has hour 10 as the most popular hour for orders, while days 0 and 6 have hour 14 as the most popular. It suggests that Days 1 - 5 are weekdays, while Days 0 and 6 are weekends.

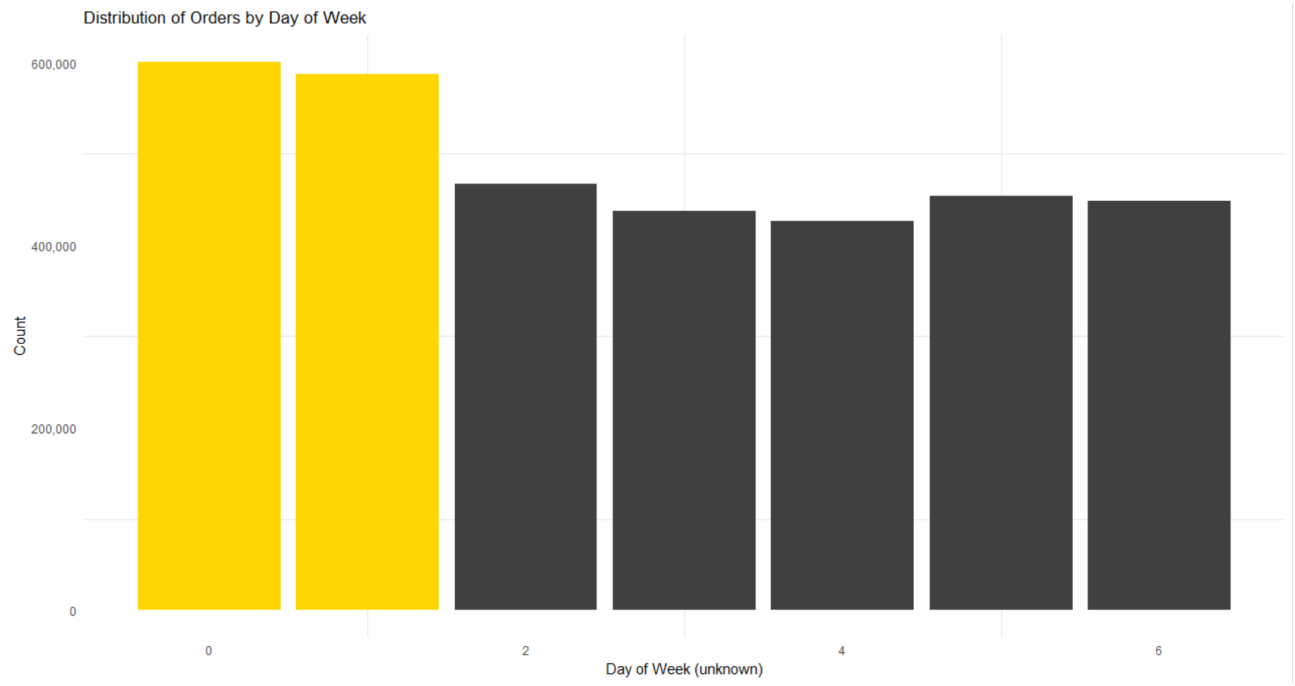
Process :

Here, the dataset *Orders* is used. The column *order_hour_of_day* is used in the x-axis along with total *Number of Orders* is on the y-axis. To generate plot for multiple days, 7 variables are created where each stores the plot of each day. And then joined all of them together with *grid.arrange* command.

Code :

```
p0 <- ggplot(orders[orders$order_dow == 0, ],
                          aes(x = order_hour_of_day)) + ...
p1 <- ...
...
p6 <- ggplot(orders[orders$order_dow == 6, ],
                          aes(x = order_hour_of_day)) + ...
grid.arrange(p0, p1, p2, p3, p4, p5, p6, ncol = 1)
```

3. How many Orders by Day of the Week?



Output :

There are a lot of orders on each day of the week. However, there are significantly more orders on days 0 and 1. This may represent the weekend.

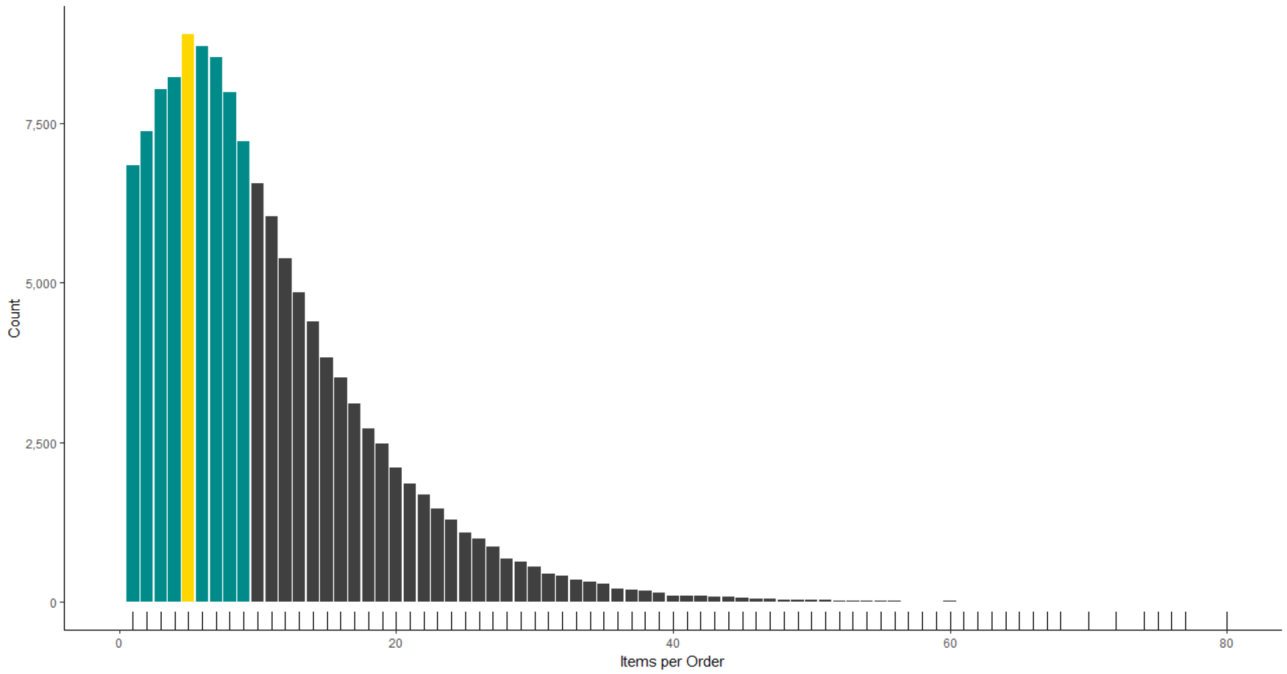
Process :

To generate this visualization, the dataset *Orders* is used and the column named *order_dow* is taken on the x-axis and generated a bar plot showing total *Number of Orders* on the y-axis with respect to the values in x-axis.

Code:

```
ggplot(orders, aes(x = order_dow)) +
  geom_bar(fill = c(rep("gold", 2), rep("grey25", 5))) +
  theme_minimal() +
  theme(axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "none",
        panel.grid.major = element_blank()) +
  labs(x = "Day of Week (unknown)",
       y = "Count",
       title = "Distribution of Orders by Day of Week") +
  scale_y_continuous(labels = comma)
```

4. How many items do People buy per Order?



Output :

Majorly, number of items per Order is between 1-10 items, with 5 items per Order being the highest among all the orders.

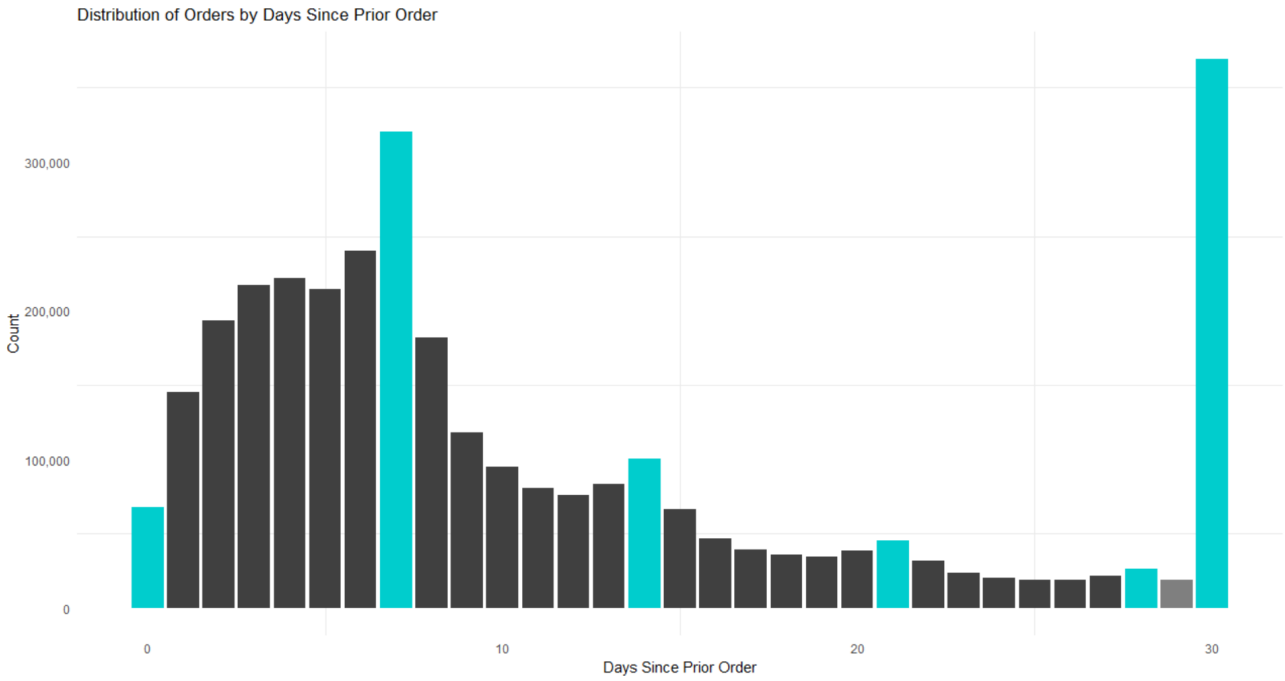
Process :

To generate this visualization, the dataset *order_products_train* is used and grouped the dataset according to the column named *order_id* using *group_by* command of R. Then extracted the last values of *add_to_cart_order* column to get the highest number of items that each Order has, using the *summarize* command provided in *dplyr* library of R. And then plotted the last values of *add_to_cart_order* column on the x-axis and it's count on the y-axis.

Code:

```
order_products_train %>%
  group_by(order_id) %>%
  summarize(n_items = last(add_to_cart_order)) %>%
  ggplot(aes(x = n_items)) +
  geom_bar(fill = c(rep("cyan4",4),"gold",rep("cyan4",4),
                    rep("grey25",66)))+
  geom_rug()+
  coord_cartesian(xlim = c(0,80))+
  scale_x_continuous(name = "Items per Order",labels = comma)+
  scale_y_continuous(name = "Count", labels = comma)
```

5. How many Days between Current & Prior Orders?



Output:

From the above graph, it is observed that around 50,000 consumers make orders on the same day since last order, around 300,000 consumers make orders on a weekly basis, and so on.

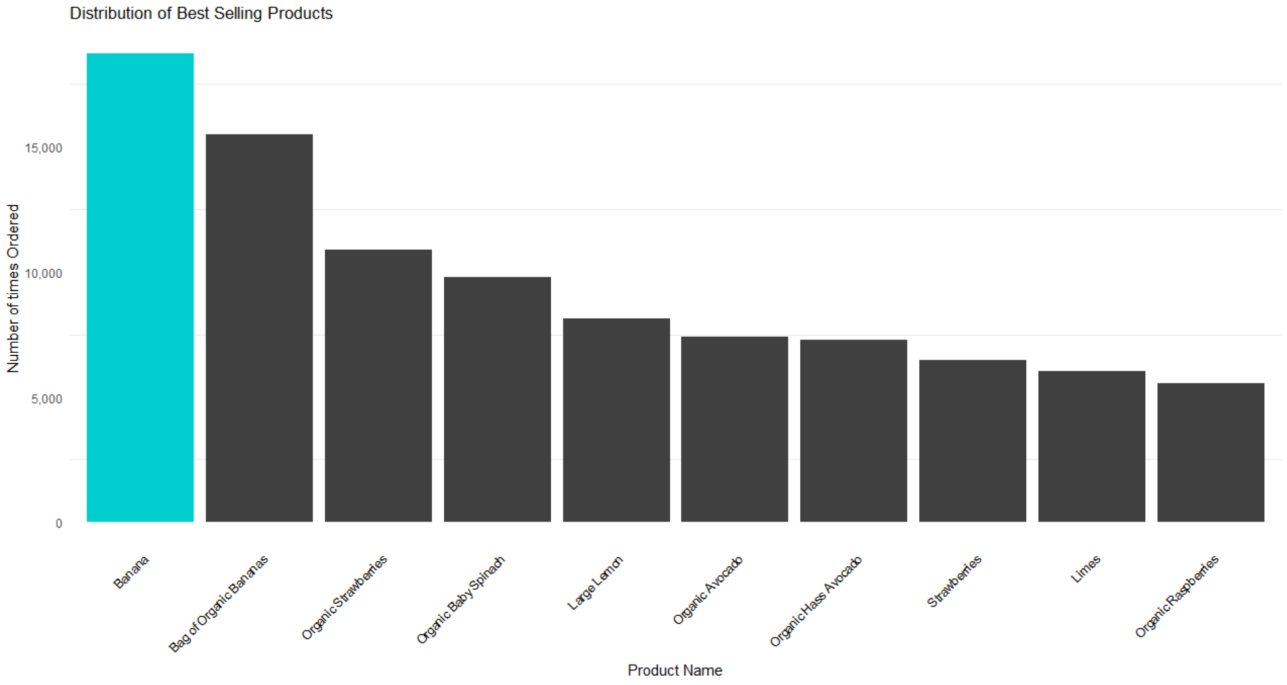
Process:

To generate this visualization, the dataset *Orders* is used and the column named *days_since_prior_orders* is taken on the x-axis and generated a bar plot showing total *Count* on the y-axis with respect to the values in x-axis.

Code:

```
ggplot(orders, aes(x = days_since_prior_order)) +
  geom_bar(fill = c("cyan3", rep("grey25", 6),
                    "cyan3", rep("grey25", 6),
                    "cyan3", rep("grey25", 6),
                    "cyan3", rep("grey25", 6),
                    "cyan3", "grey50", "cyan3")) +
  theme_minimal() +
  labs(x = "Days Since Prior Order",
       y = "Count",
       title = "Distribution of Orders by Days Since Prior Order")
+scale_y_continuous(labels = comma)
```

6. What are the best selling products?



Output :

Highest Selling product is Banana, which is bought by consumers 18,726 times. Other best selling products includes *Bag of Organic Bananas, Organic Raspberries, etc.*

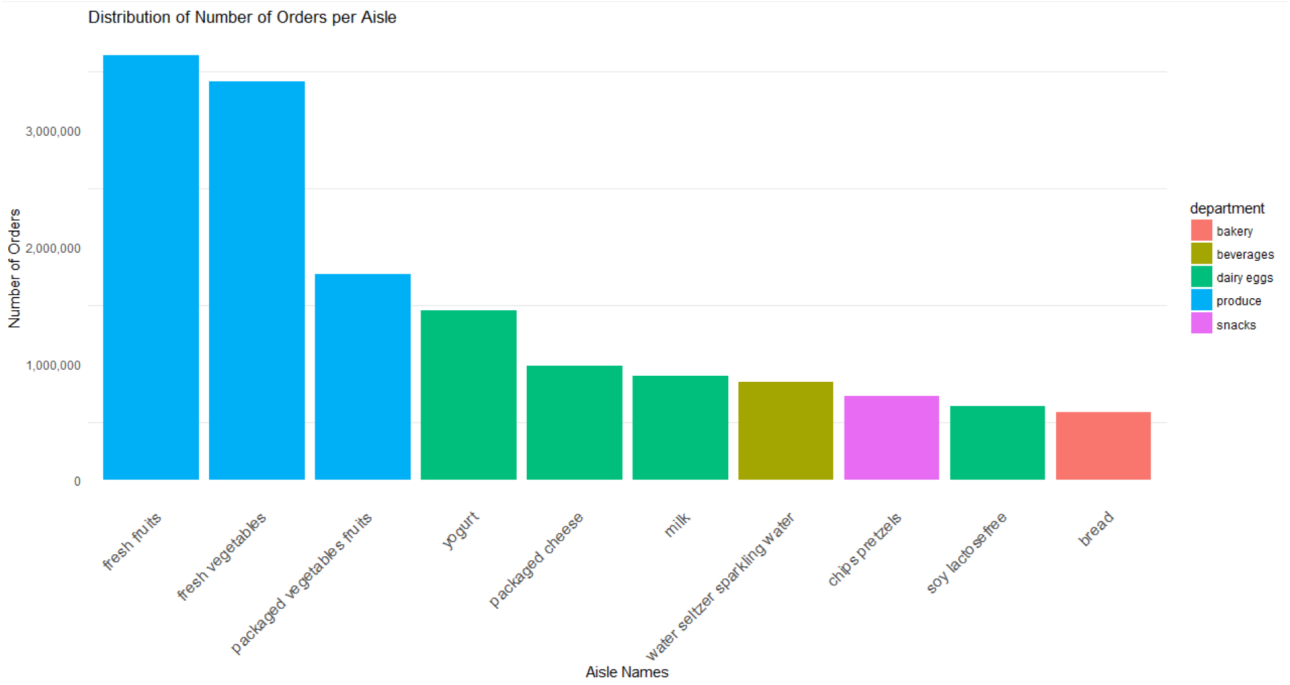
Process :

To generate this visualization, the dataset *order_products_train* is used and grouped the dataset according to the column named *product_id* using *group_by* command of R. Then extracted the number of observations in the current group using the command *n()* (*dplyr*) and stored it's values in a variable called *count*, using the *summarize* command (*dplyr*). Then select the top 10 number of observations from the *count* variable using the *top_n* command (*dplyr*). Then, to know the best-selling product's names, the *left_join* command (*dplyr*) is used on the *order_products_train* by selecting the *product_name* from the *products* dataset. Then arranged the *count* variable in descending order using the *arrange & desc* commands (*dplyr*). And then plotted the reordered *product_name* column on the x-axis according to it's *count* and placing the total count of each such product on the y-axis.

Code:

```
order_products_train %>%
group_by(product_id) %>%
summarize(count = n()) %>%
top_n(10, wt = count) %>%
left_join(select(products,product_id,product_name),by="product_id"
) %>%
arrange(desc(count)) %>%
ggplot(aes(x=reorder(product_name,-count), y=count)) +  ...
```

7. What are the Top Selling Aisles?



Output :

The top 10 aisles given above represent the 45.3% of the Total Sales. Out of the 10 top selling aisles, the aisles *Fresh Fruits & Fresh Vegetables* generated a maximum number of sales of more than 600,000 orders.

Process :

To generate this visualization, the dataset *order_products_train* is used and attached the datasets *products, aisles & departments* into the existing dataset by using the command *left_join* (*dplyr*) over all 3 given datasets. Then grouped the dataset *order_products_train* according to the *aisle & department* column.

The command *tally* (*dplyr*) is used to get the number of observations in the current group, and then sorted the output in descending order. Then added a new column called *perc* to find out the percentage of individual *aisle & department* making the sell using the command *mutate* (*dplyr*).

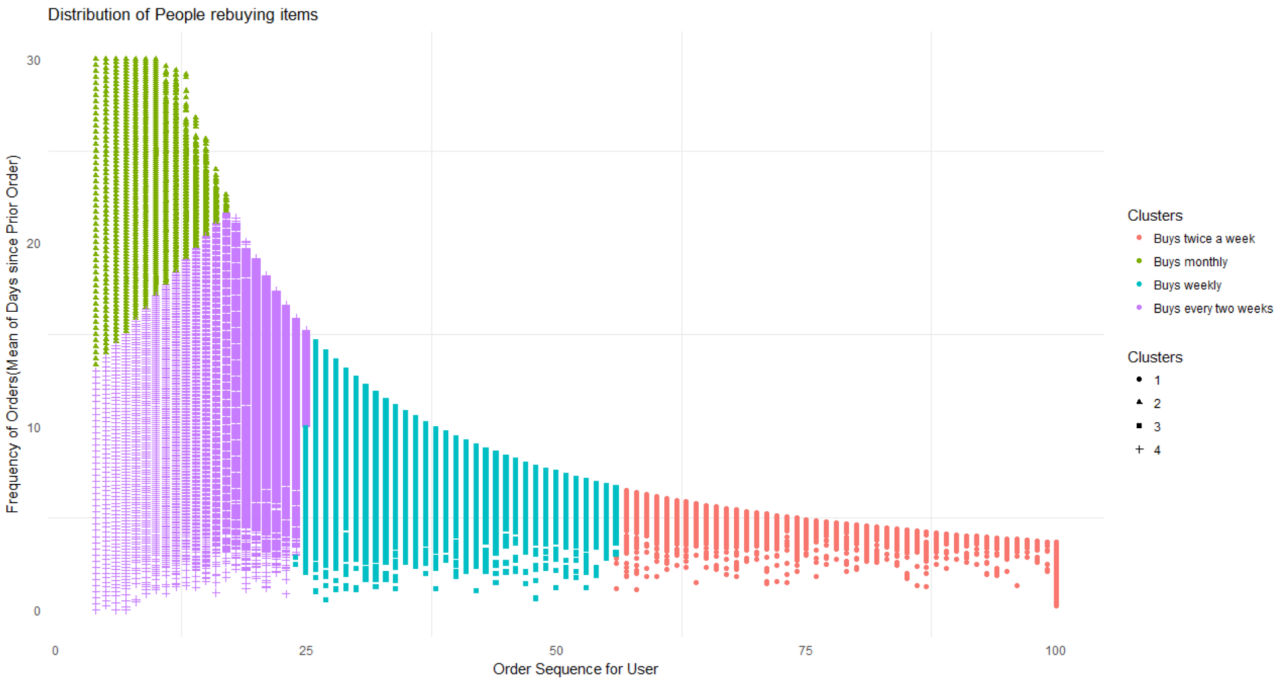
Removed the grouping by *ungroup* command (*dplyr*) and selected the top 10 observations using the *top_n* command (*dplyr*). Then extracted the number of observations in the current group using the command *n()* (*dplyr*) and stored it's values in a variable called *count*, using the *summarize* command (*dplyr*).

Then selected the top 10 number of oservations from the *count* variable using the *top_n* command (*dplyr*). And then plotted the reordered *aisle* column on the x-axis according to it's *count(n)* and placing the total count (*n*) of each such aisle on the y-axis.

Code:

```
order_products_train %>%
  left_join(products,by="product_id") %>%
  left_join(aisles,by="aisle_id") %>%
  left_join(departments,by="department_id")%>%
  group_by(aisle,department)%>%
  tally(sort=TRUE)%>%
  mutate(perc = round(100*n/nrow(order_products_train),2))%>%
  ungroup() %>%
  top_n(10,n) %>%
  ggplot(aes(x=reorder(aisle, -n), y=n, fill=department)) +
  geom_bar(stat="identity") +
  theme_minimal() +
  theme(axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x=element_text(angle=45, hjust=1, size = 12),
        panel.grid.major = element_blank()) +
  labs(x = "Aisle Names",
       y = "Number of Orders",
       title = "Distribution of Number of Orders per Aisle") +
  scale_y_continuous(labels = comma)
```

8. How often People rebuy Items?



Output :

From the above visualization, 4 clusters of consumers are observed. The ones who bought less than 25 items on Instacart, are having average freuqncy of buying monthly or buying every two weeks. The ones with total orders between 25 to 55, are having buying frequency of 7 days on average. And the ones with total orders more than 55, are having buying frequency of 4 days on average, i.e, they buy almost twice a week.

Process :

First of all, group the data according to the users. Creat a new dataset called *user_freq* from the dataset *orders. S*elect the columns *user_id, order_number & days_since_prior_order* from the dataset *orders* and after grouping them according to *user_id* using the *group_by* command (*dplyr*) creat two columns of *user_freq,* called *total* and *frequency* where *total* will store the maximum of *order_number* and *frequency* will store the mean of *days_since_prior_order.* Now observe that *total* will tell the maximum number of orders done by each consumer and *frequency* will tell the average days between each consumer's orders.
Once the dataset *user_freq* is created, create a *K-Means* clustering of the dataset, using the columns *total* and *frequency* and create a total of 4 clusters.

Then plot the graph with *total o*n the x-axis and *frequency* on the y-axis, and give them multiple colors according to the clusters created by *Kmeans.*

Code :

```
user_freq <- orders %>%
  select(user_id, order_number, days_since_prior_order)%>%
  group_by(user_id) %>%
  summarise(total = max(order_number),
            frequency = mean(days_since_prior_order, na.rm =
TRUE))

set.seed(42)
clus <- kmeans(user_freq[,2:3], 4)
clus$cluster <- as.factor(clus$cluster)
Clusters <- clus$cluster
ggplot(user_freq,
       aes(total, frequency, shape = Clusters, color = Clusters))
+
  geom_point()+
  labs(x = "Order Sequence for User",
       y = "Frequency of Orders(Mean of Days since Prior Order)",
       title = "Distribution of People rebuying items")+
  scale_colour_discrete(name="Clusters",
                        labels = c("Buys twice a week",
                                   "Buys monthly",
                                   "Buys weekly",
                                   "Buys every two weeks"))+
  theme_minimal() +
  theme(axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        panel.grid.major = element_blank())
```

3. Market Basket Analysis

(*also known as **Affinity Analysis** or **Association Analysis***)

Market Basket Analysis typically encompasses several modeling techniques based upon the simple principle that while shopping if you buy a certain group of items (also known as an itemset in machine learning lingo), you are likely to buy an other specific item or items along with that itemset. We analyze human shopping patterns and apply statistical techniques to generate frequent itemsets. These itemsets contain combination of items that people are most likely to buy together, based on past shopping history.

A simple example of an itemset would be people buying beer and diapers frequently at the market. The itemset can be depicted as { beer, diapers }.

Build item association rules on top of these itemsets by analyzing shopping purchases. An example association rule can be denoted by using itemsets using the notation, { beer, diapers } -> { milk } which would indicate that if someone is buying beer and diapers together, that person is most likely to also purchase milk along with that!

3.1 Core Concepts & Definitions

- **Transactional datasets** indicate databases or datasets where the customer's shopping transactions are recorded daily/weekly and consist of various items bought together by the customers. In this case, use the Datasets provided by Instacart.

- **Itemsets** are defined as sets or groups of items which were bought together in any shopping transaction. Hence, these items are said to co-occur based on the transactions. Denote itemsets as ISn where *n* denotes the n-th itemset number. The itemset values will be enclosed in curly braces.

- **Association Rules** or just *rules* are statements which have a **left-hand side** (**LHS**) and a **right-hand side** (**RHS**), and indicate that if we have the items on the LHS for purchase, we are likely to be interested in purchasing the RHS items too. This signifies that the itemsets are associated with each other. They are denoted as ISx → ISy, which means that if someone has itemset x in his shopping cart, he will also be interested in purchasing itemset y along with it. An example rule can be

  `{ beer } → { diapers }`

  which indicates that if someone has beer in his cart, there is a chance he will buy diapers too!

- The **frequency** of an itemset is basically the number of times a particular itemset occurs in the list of all transactions. It is denoted as f(ISn), where ISn is a particular itemset and function f( ) gives the frequency of that itemset in the whole transactional based dataset.

- The **support** of an itemset is defined as the fraction of transactions in our transactional dataset which consists of that particular itemset. Basically, it means the number of times that itemset was purchased divided by the total number of transactions in the dataset. It can be denoted as :

$$S\left(IS_n\right) = \frac{f\left(IS_n\right)}{count\left(\sum_{i=1}^{n} IS_i\right)}$$

where S( ) denotes the support of the itemset Isn.
The support for an association rule is similar and can be depicted as :

$$S\left(IS_x \rightarrow IS_y\right) = \frac{f\left(IS_x \cup IS_y\right)}{count\left(\sum_{i=1}^{n} IS_i\right)}$$

where the union operator is used to see the frequency of both the itemsets occurring together in the transactional dataset.

When evaluating results from association rules or frequent itemsets, the higher the support, the better it is. Support is more about measuring the quality of rules detecting what has already happened from the past transactions.

- The **confidence** of an association rule is defined as the probability or likelihood that, for a new transaction containing itemset in the LHS of the rule, the transaction also contains the itemset on the RHS of the rule. The confidence for a rule can be depicted as :

$$C\left(IS_x \rightarrow IS_y\right) = \frac{f\left(IS_x \cup IS_y\right)}{f\left(IS_x\right)}$$

where C( ) denotes the confidence of the rule. Remember that confidence is more about detecting the quality of rules predicting what can happen in the future based on the past transactional data.

- The **lift** of an association rule is defined as the ratio of the support of the combination of two itemsets on the LHS and RHS together divided by the product of the support of each of the itemsets. The lift for a rule can be depicted as :

$$L\left(IS_x \rightarrow IS_y\right) = \frac{S\left(IS_x \cup IS_y\right)}{S\left(IS_x\right) \times S\left(IS_y\right)}$$

where L( ) denotes the lift of the rule.

The lift of a rule in general is another metric to evaluate the quality of the rule. If the lift is > 1 then it indicates that the presence of the itemset in the LHS is responsible for the increase in probability that the customer is also going to buy the itemset on the RHS. This is another very important way to determine itemset associations and which items influence people to buy other items, because if the lift has a value = 1, it means that the itemsets on the LHS and RHS are independent and buying one itemset will not affect the customer to buy the other itemset. If the lift is < 1, it indicates that if the customer has an itemset on the LHS then the probability of buying the itemset on the RHS is relatively low.

3.2 Implementation on R

For this kind of analysis, the required dataset is *order_products_prior* which contains orders prior to that users most recent order, and it consists of approximately 3.2 million orders.

But, the dataset *order_products_prior* gives information about following: *order_id, product_id, add_to_cart_order* and *reordered*, where each columns are self explanatory. But it is necessary to know which products were bought in each order with their names for further analysis, and not just the *product_id*.

To do so, the following command is given:

```
order_baskets <- order_products_prior %>%
                 inner_join(products,by="product_id") %>%
                 group_by(order_id) %>%
                 summarise(basket=as.vector(list(product_name)))
```

What this basically does is, it creates multiple new columns in the dataset *order_products_prior* according to the dataset *products* and then it groups the whole dataset *order_products_prior* acording to the column called *order_id*. And finally, it creates a new column called *baskets* , which will contain all the *product_name*(s) and it removes all the columns except *order_id.*

Now, take a look at the dataset *order_baskets* :

```
> head(order_baskets)
# A tibble: 6 x 2
  order_id     basket
     <int>     <list>
1        2  <chr [9]>
2        3  <chr [8]>
3        4 <chr [13]>
4        5 <chr [26]>
5        6  <chr [3]>
6        7  <chr [2]>
> head(order_baskets$basket[1])
[[1]]
[1] "Organic Egg Whites"                    "Michigan Organic Kale"
[3] "Garlic Powder"                         "Coconut Butter"
[5] "Natural Sweetener"                     "Carrots"
[7] "Original Unflavored Gelatine Mix"      "All Natural No Stir Creamy Almond Butter"
[9] "Classic Blend Cole Slaw"

>
```

When executing the 2nd command in the picture above, observe that the *order_id 2* has a total 9 items in its cart, along with their names.

But the variable *order_baskets* is a data frame. Any kind of Assiciation analysis with such type of datasets is not possible. Next step would be to convert the variable *order_baskets* into a transaction data type, where the aim would be fulfilled.

The command to do so would be :

```
orderTransactions <- as(order_baskets$basket,"transactions")
```

Now, take a look at the variable *orderTransactions* which is of type *transactions*, as expected. Look at first two transactions of the dataset *orderTransactions*:

```
> inspect(orderTransactions[1:2,])
    items
[1] {All Natural No Stir Creamy Almond Butter,
     Carrots,
     Classic Blend Cole Slaw,
     Coconut Butter,
     Garlic Powder,
     Michigan Organic Kale,
     Natural Sweetener,
     Organic Egg Whites,
     Original Unflavored Gelatine Mix}
[2] {Air Chilled Organic Boneless Skinless Chicken Breasts,
     Lemons,
     Organic Baby Spinach,
     Organic Ezekiel 49 Bread Cinnamon Raisin,
     Organic Ginger Root,
     Total 2% with Strawberry Lowfat Greek Strained Yogurt,
     Unsweetened Almondmilk,
     Unsweetened Chocolate Almond Breeze Almond Milk}
>
```

Now that with our Grocery transactions are ready, find out more information about the data. One important step store all the basket sizes in a variable, which will be used later on.
The command to give is:

```
basketSizes<-size(orderTransactions)
```

And to get a summary of all the baskets (approximately 3.2 millions transactions) , then:

```
summary(basketSizes)
```

We can see the distribution of basket sizes as follows:

```
> summary(basketSizes)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    5.00    8.00   10.09   14.00  145.00
>
```

In some order, someone actually ordered a total of 145 items!

3.2.1 Generate Rules for Apriori

Now the interesting thing begins!

Generate rules using Apriori algorithm for the most frequent items in given transactional dataset *orderTransactions.* Using the library *arules*, there exists a built in function called *apriori* which will generate rules, for given support/confidence/lift and length of itemsets.

Dgiven ataset is a very high dimensional, that is, there are a lot of transactions and each transactions has a lot of items in it. So, in high dimensional data, almost every event is rare! So when generating our rules, support will be kept quite low.

So first attempt, keep *support* to be 0.01, that is, 1 in every 100 transactions.
And then, generate the rules, using the *apriori algorithm* with the following command:

```
support<- 0.01
itemSets <- apriori(orderTransactions,
                    parameter = list(target="frequent itemsets",
                                     supp=support,minlen=2),
                                     control = list(verbose=FALSE))
```

Here, the variable *itemSets* will store all the output generated by the *apriori* command. Take a summary of the variable *itemSets* :

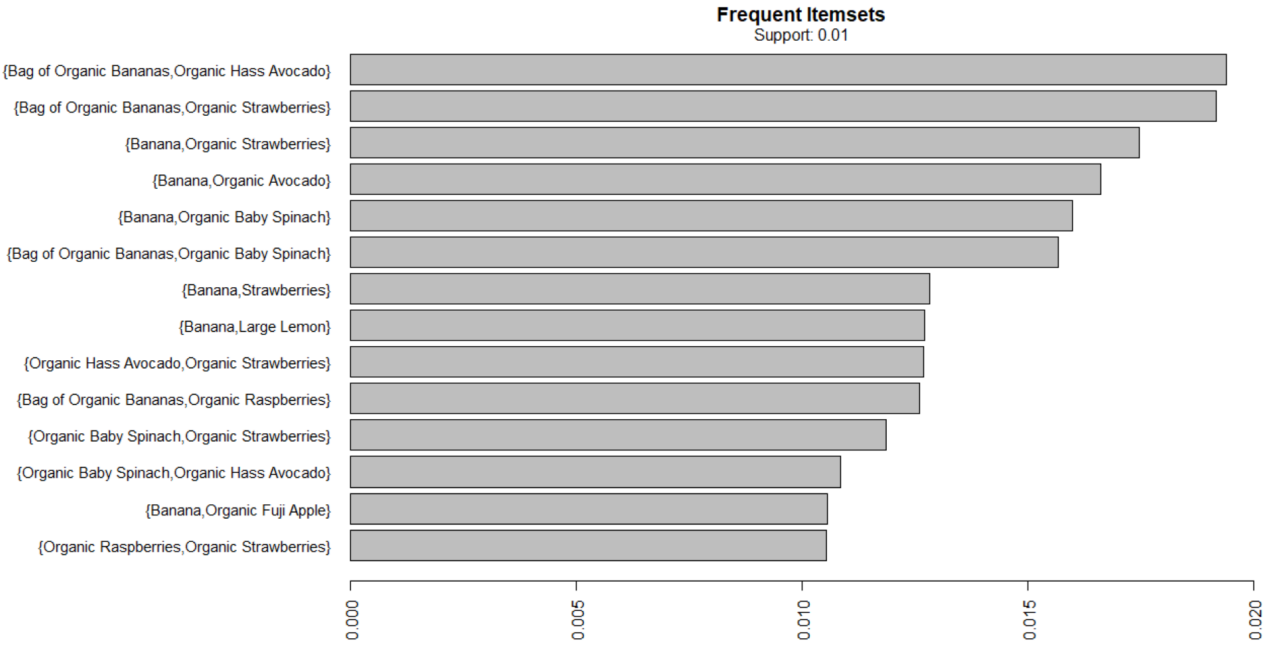
```
> summary(itemSets)
set of 14 itemsets

most frequent items:
                Banana   Organic Strawberries Bag of Organic Bananas   Organic Baby Spinach
                     6                      5                      4                      4
  Organic Hass Avocado                 (Other)
                     3                      6

element (itemset/transaction) length distribution:sizes
 2
14

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       2       2       2       2

summary of quality measures:
    support           count
 Min.   :0.01053   Min.   :33863
 1st Qu.:0.01205   1st Qu.:38726
 Median :0.01277   Median :41056
 Mean   :0.01421   Mean   :45681
 3rd Qu.:0.01645   3rd Qu.:52895
 Max.   :0.01939   Max.   :62341

includes transaction ID lists: FALSE

mining info:
             data ntransactions support confidence
 orderTransactions       3214874    0.01          1
>
```

Now let's see what are the most frequent itemsets bought in a total of ~3.2 million transactions :

```
> inspect(itemSets)
     items                                              support    count
[1]  {Banana,Organic Fuji Apple}                        0.01055811 33943
[2]  {Banana,Strawberries}                              0.01282539 41232
[3]  {Organic Raspberries,Organic Strawberries}         0.01053323 33863
[4]  {Bag of Organic Bananas,Organic Raspberries}       0.01259863 40503
[5]  {Banana,Large Lemon}                               0.01271589 40880
[6]  {Banana,Organic Avocado}                           0.01660874 53395
[7]  {Organic Baby Spinach,Organic Hass Avocado}        0.01085610 34901
[8]  {Organic Hass Avocado,Organic Strawberries}        0.01268914 40794
[9]  {Bag of Organic Bananas,Organic Hass Avocado}      0.01939143 62341
[10] {Organic Baby Spinach,Organic Strawberries}        0.01186174 38134
[11] {Bag of Organic Bananas,Organic Baby Spinach}      0.01566842 50372
[12] {Banana,Organic Baby Spinach}                      0.01598663 51395
[13] {Bag of Organic Bananas,Organic Strawberries}      0.01916965 61628
[14] {Banana,Organic Strawberries}                      0.01746756 56156
>
```

Itemset number 9 i.e., *{ Bag of Organic Bananas, Organic Hass Avocado}* is the winner out of all. It has been bought by consumers total 62,341 times, and thus it has the highest support values of 0.01939143!

Create a visualization of the most frequent itemsets :



Things are observed much more clearly because of the visualization. Here, x-axis represents the support of each itemsets.

In the above example, try to find out the frequent itemsets using the *apriori* algorithm. But find out *Association rules* of multiple items on each other.

3.2.2 Generating New Association Rules

In order to find products that occur together in each basket, it's hard any direct use of people who haven't yet shown interest in multiple products. So restrict the dataset to customers who have expressed interest in at least two products.

```
product_baskets <- orderTransactions[basketSizes > 1 ]
```

Try restricting the itemsets that are supported by at least 1500 baskets. This leads to a minimum support of: 0.0005 (1.5k divided by 3million) and take a minimum confidence of 0.01, that is, rule is correct 1% of the time.

Now generate new rules with the following command :

```
rules <- apriori(product_baskets,
                 parameter =
     list(supp=0.0005,conf=0.01,minlen=2),
                 control = list(verbose = FALSE))
```

See a summary of the newly generated rules :

```
> summary(rules)
set of 15616 rules

rule length distribution (lhs + rhs):sizes
    2     3     4
10904  4584   128

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    2.00    2.00    2.31    3.00    4.00

summary of quality measures:
    support           confidence          lift             count
 Min.   :0.0005003   Min.   :0.01001   Min.   :  0.4625   Min.   : 1530
 1st Qu.:0.0005925   1st Qu.:0.03698   1st Qu.:  1.6953   1st Qu.: 1812
 Median :0.0007478   Median :0.07852   Median :  2.2363   Median : 2287
 Mean   :0.0010686   Mean   :0.11372   Mean   :  4.1877   Mean   : 3268
 3rd Qu.:0.0011088   3rd Qu.:0.17143   3rd Qu.:  3.0696   3rd Qu.: 3391
 Max.   :0.0203854   Max.   :0.69027   Max.   :408.4814   Max.   :62341

mining info:
            data ntransactions support confidence
 product_baskets       3058126   5e-04       0.01
> inspect(rules[1:4])
    lhs                          rhs                          support      confidence lift      count
[1] {Organic Fuji Apples}  => {Bag of Organic Bananas} 0.0005748619 0.3282913   2.665304 1758
[2] {Zero Calorie Cola}    => {Soda}                   0.0012981807 0.4994339  46.021990 3970
[3] {Soda}                 => {Zero Calorie Cola}      0.0012981807 0.1196252  46.021990 3970
[4] {Organic Low Fat Milk} => {Bag of Organic Bananas} 0.0005679949 0.2178330   1.768524 1737
> |
```

The newly generated rules are a set of 15,616 rules in total, where 10904 rules are of length 2,4584 are of length 3 and 128 rules are of length 4. Also the range of variations of multiple parameters like support, confidence, lift and count. Using the *inspect* command, first 4 rules that is being generated.

Just to understand what these rules are trying to say, take the example of rule number 1. *{Organic Fuji Apples}* → *{Bag of Organic Bananas}* gives us a lift of ~2.66, telling us that if a consumer buys *Organic Fuji Apples*, then it is more than two times likely that the consumer will buy *Bag of Organic Bananas* along with it.

Now as there are a total of 15,616 rules, it is very likely that many of them are subsets of each other. So it is necessary to find out the unique rules only. We will find out such rules using the following command:

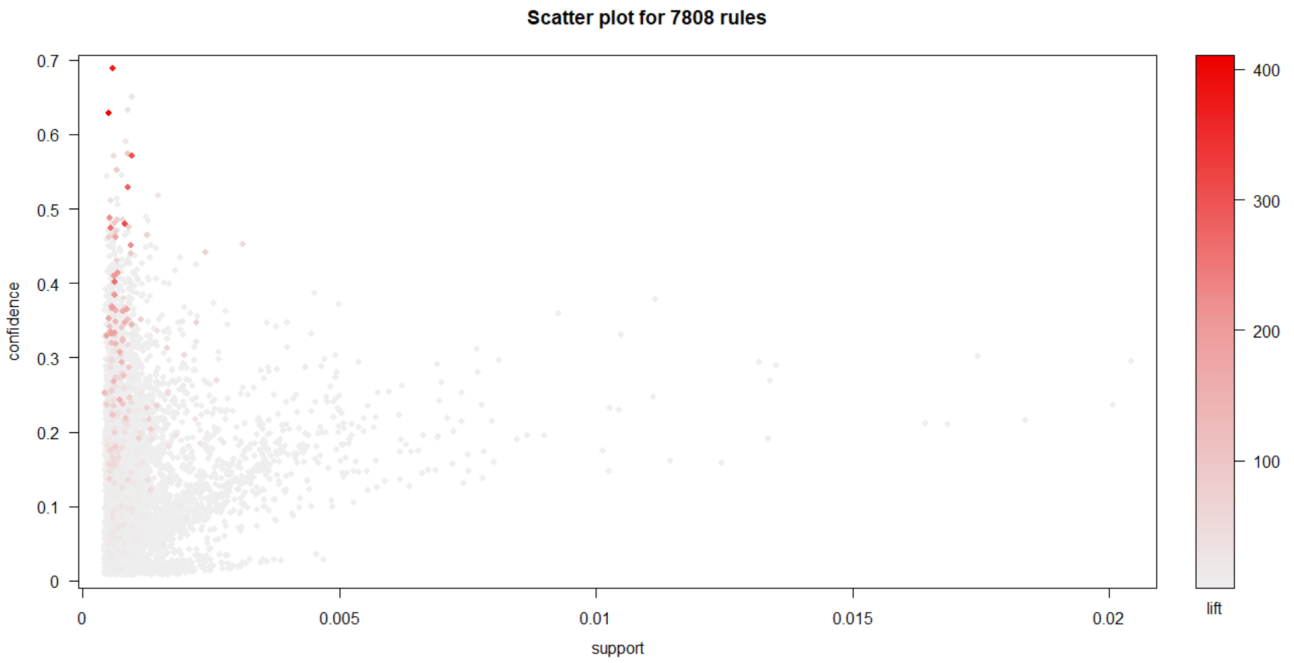
```
nonRedundantRules <- rules[unique(as.vector(is.subset(rules,
                                rules)))]
```

Once redundant rules are removed, only 7,808 rules are left which are truly unique.

3.2.3 Scatter Plot

Create a visualization of all the unique rules in a scatter plot keep *support* on the x-axis, *confidence* on the y-axis and *lift* according to the colour indicated alongside.

From the scatter plot below, it is observed that maximum *lift* is having *support* less than 0.005 but having a high *confidence* at ~0.7. And the rule with maximum *support* has confidence ~0.3 and has very low *lift* also.



3.2.4 Matrix Shading Plot

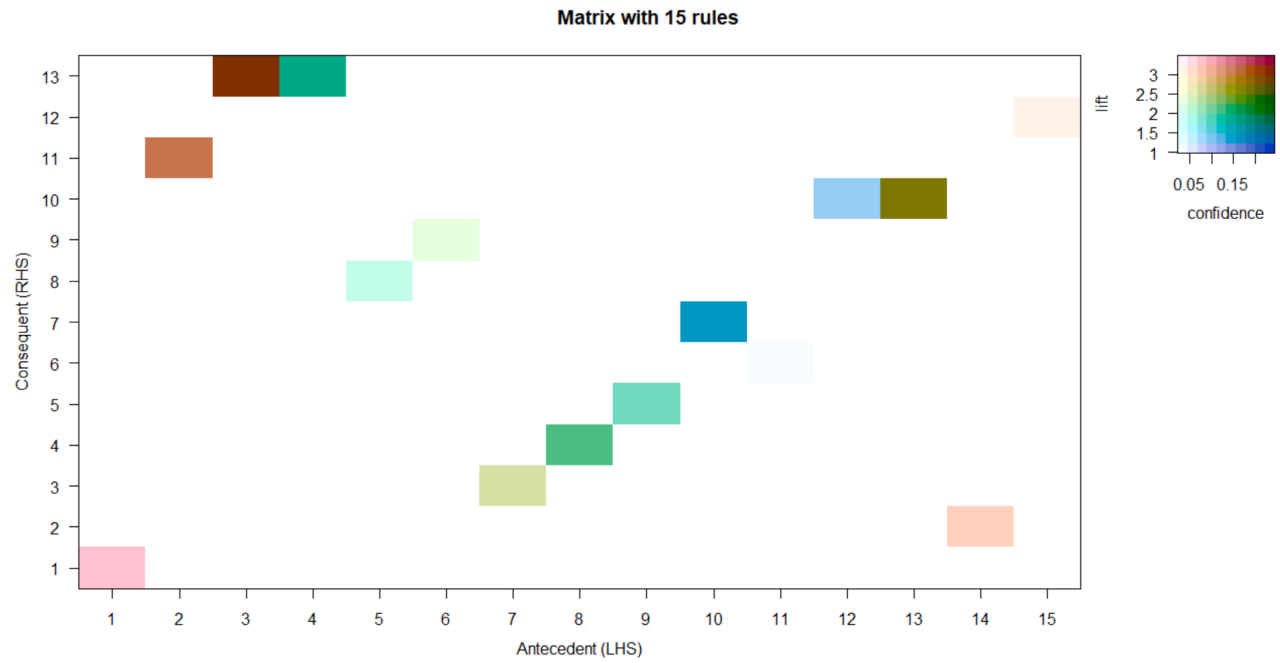
Another type of visualization done on association rules is called *Matrix Shading* where on the x-axis, keep the LHS Rules (also called *Antecedent*) and on the y-axis keep the RHS Rules (also called *Consequent*). And observe the matrix graph with respect to two measures of *lift* and *confidence.* But this works best on smaller number of rules. So first of all take a sample of 15 rules to perform matrix shading by the following command :

```
subrules <- sample(nonRedundantRules, 15)
```

Then generate the matrix shading with the following command :

```
plot(subrules, method="matrix", shading=c("lift", "confidence"))
```

This is the following output :



```
> plot(subrules, method="matrix", shading=c("lift", "confidence"))
Itemsets in Antecedent (LHS)
 [1] "{Bag of Organic Bananas,Organic Blueberries}" "{Organic Baby Spinach,Organic Cilantro}"
 [3] "{Fresh Cauliflower,Organic Zucchini}"         "{Organic Cauliflower}"
 [5] "{Organic Garnet Sweet Potato (Yam)}"          "{Organic Grape Tomatoes}"
 [7] "{Banana,Organic Avocado}"                     "{Organic White Onions}"
 [9] "{Organic Raw Kombucha Gingerade}"             "{Sinfully Sweet Campari Tomatoes}"
[11] "{Organic Half & Half}"                        "{Fresh Ginger Root}"
[13] "{Organic Jalapeno Pepper}"                    "{Organic Baby Spinach,Organic Hass Avocado}"
[15] "{Fresh Cauliflower}"
Itemsets in Consequent (RHS)
 [1] "{Organic Blackberries}"             "{Organic Granny Smith Apple}"       "{Organic Baby Arugula}"
 [4] "{Organic Avocado}"                  "{Organic Raspberries}"              "{Carrots}"
 [7] "{Bag of Organic Bananas}"           "{Organic Peeled Whole Baby Carrots}" "{Organic Broccoli Florets}"
[10] "{Organic Hass Avocado}"             "{Large Lemon}"                      "{Brussels Sprouts}"
[13] "{Organic Baby Spinach}"
> |
```

Here, the LHS 3 and RHS 13 are having very high confidence and lift and thus indicating that if a consumer buys LHS item 3 it is very likely that the consumer will buy RHS 13 also, where LHS 3 is *{Fresh Cauliflower, Organic Zucchini}* and RHS 13 is *{Organic Baby Spinach}*.
Similarly LHS 2 and RHS 11 are also highly co-related where LHS 2 is *{Organic Baby Spinach, Organic Cilantro}* and RHS 11 is *{Large Lemon}.*

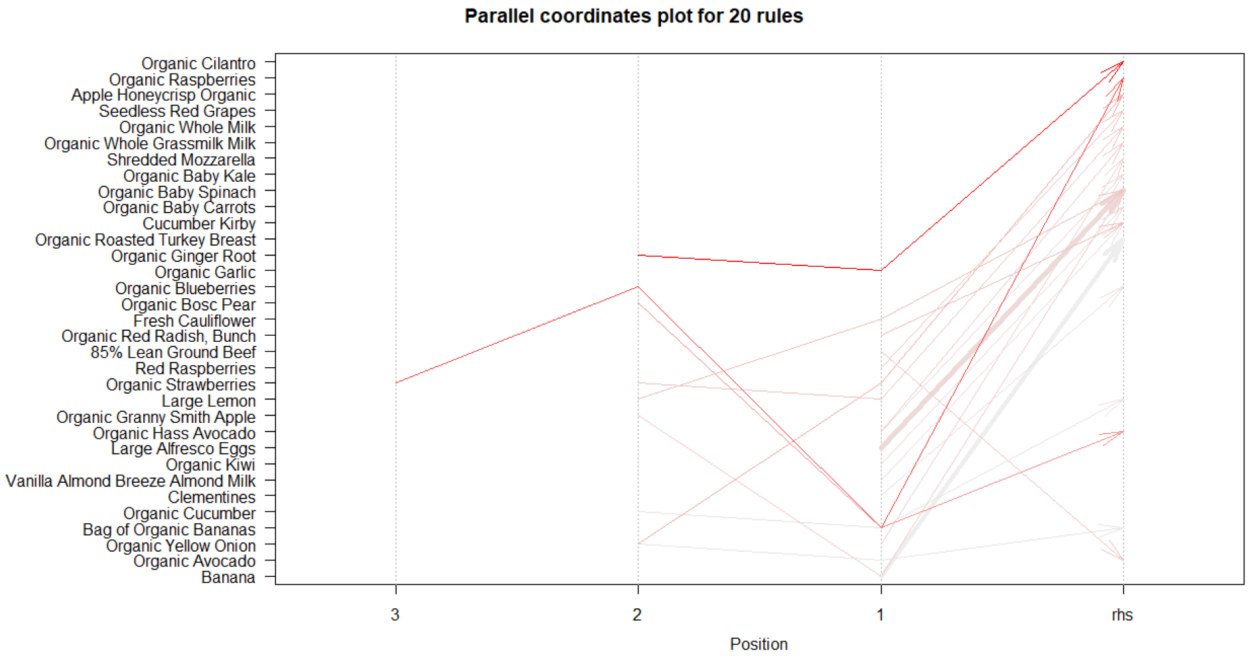
Note that in this case, a *Matrix Shading* is generated using parameters *confidence* and *lift*. But different parameters like *support* or *count* are also used.

3.2.5 Parallel Coordinates Plot

Another type of visualization that can be done on association rules is known as *Parallel coordinates plot.* In this type of plot, a small sample of rules are taken and parallel coordinates are drawn indicating the sequence of items bought by consumers and the likely RHS item, where the density of colour of the paracords gives the *lift.* To generate a paracord, the following commandiis given:

```
plot(subrules, method="paracoord")
```

It generates the following plot:



Here, it is observed that the rule *{Organic Garlic, Organic Ginger Root} => {Organic Cilantro}* are having the highest *lift* of 7.12 indicating that if a consumer buys *{Organic Garlic, Organic Ginger Root}* together, it is 7 times likely that the consumer will buy *{Organic Cilantro}* also.

3.2.6 Rules leading to specific item(s)

The apriori algorithm can also be used to find out the items that lead to buy specific item(s). To extract such kind of rules, give specific item names in the parameter of *rhs* keeping *default* parameter as *lhs. Banana* being the highest selling item, take the specific item to be *Banana* (can take other items also). The command to generate items that lead to buying *Bananas* is:

```
rules<-apriori(orderTransactions,
          parameter=list(supp=0.0005,conf = 0.05),
          appearance = list(default="lhs",rhs="Banana"),
               control = list(verbose=F))
```

As mentioned earlier, such rules are always redundant. Next step would be to extract only unique rules from the set of all the rules! Once that is done, sort all the rules by the value of it's *confidence.*

Take a look at first 10 rules that is being generated.

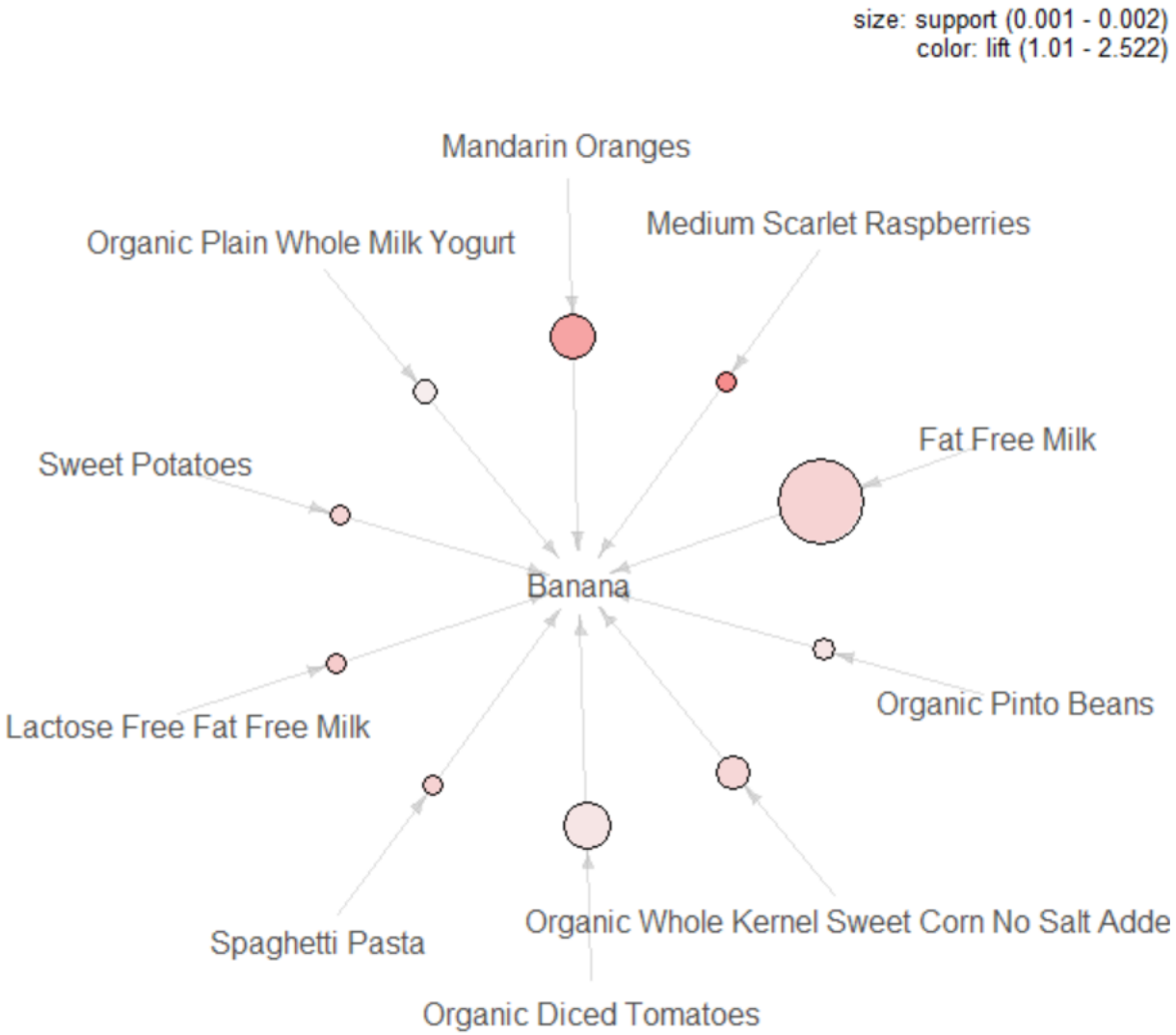
```
> inspect(bananaRules[1:10])
     lhs                                                rhs      support      confidence lift     count
[1]  {Bartlett Pears,Organic Fuji Apple}             => {Banana} 0.0007148025 0.5442918  3.702834 2298
[2]  {Bartlett Pears,Large Lemon}                    => {Banana} 0.0005919361 0.4924948  3.350457 1903
[3]  {Broccoli Crown,Strawberries}                   => {Banana} 0.0005340800 0.4698960  3.196717 1717
[4]  {Cucumber Kirby,Organic Fuji Apple}             => {Banana} 0.0010249235 0.4625860  3.146986 3295
[5]  {Organic Avocado,Organic Fuji Apple}            => {Banana} 0.0013222913 0.4535851  3.085753 4251
[6]  {Cucumber Kirby,Organic Peeled Whole Baby Carrots} => {Banana} 0.0005555428 0.4472827  3.042878 1786
[7]  {Cucumber Kirby,Strawberries}                   => {Banana} 0.0012314635 0.4451315  3.028243 3959
[8]  {Honeycrisp Apple,Seedless Red Grapes}          => {Banana} 0.0008522885 0.4355428  2.963011 2740
[9]  {Organic Fuji Apple,Original Hummus}            => {Banana} 0.0005129283 0.4273128  2.907022 1649
[10] {Cucumber Kirby,Organic Avocado}                => {Banana} 0.0020510913 0.4235067  2.881129 6594
>
```

Predict the first rule as: If a consumer buys itemsets *{Barlett Pears, Organic Fuji Apple}* then it is 4 times likely that the consumer will buy *{Banana}* too, which is backed by a confidence of 0.544 that is, 54.4% of time that happened.

A Plot of graph-based visualization is also possible with such kind of rules. But a small sample needs to be taken out of all the rules. So after taking 10 samples from the set of all the rules, plot the graph based visualization with the following command:

```
       subrules2 <- sample(sort(bananaRules, by="lift"), 10)
plot(subrules2, method="graph",control=list(type="items",main=""))
```

See the visualization below to understand what is being generated:



3.2.7 Rules based on specific item(s)

The apriori algorithm can also be used to find out the items that consumers are likely to purchase after specific item(s). To extract such kind of rules, give specific item names in the parameter of *lhs* keeping *default* parameter as *rhs*. *Banana* being the highest selling item, take the specific item to be *Banana* (can take other items also). The command to generate items that are likely bought after *Bananas* is:

```
rulesAfterBanana <- apriori (orderTransactions,
                parameter=list (supp=0.001,conf = 0.05,minlen=2),
                appearance = list(default="rhs",lhs="Banana"),
                control = list (verbose=F))
```

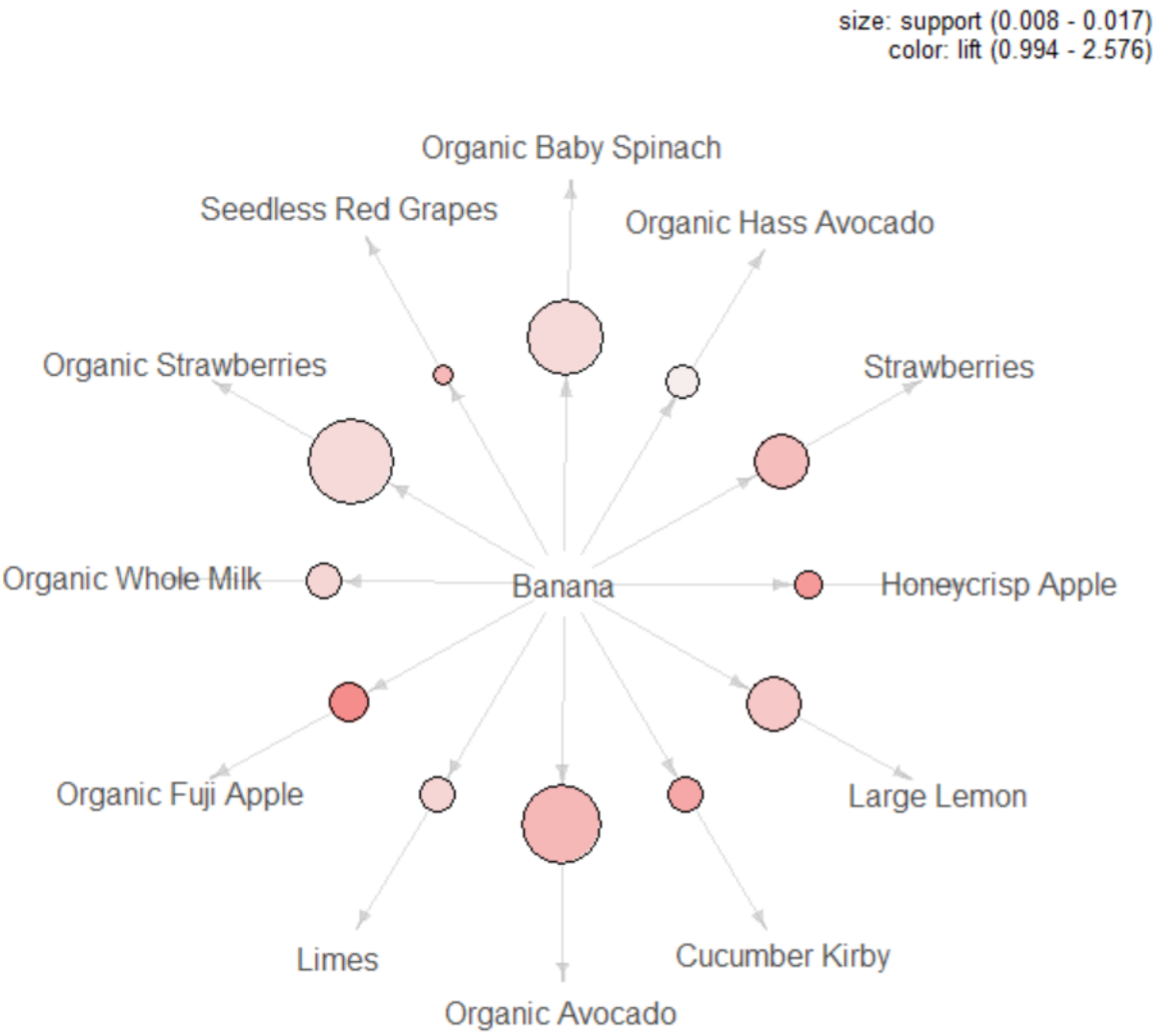
Take a look at first 5 rules that is being generated.

```
> inspect(rulesAfterBanana[1:5])
    lhs          rhs                     support     confidence lift     count
[1] {Banana} => {Honeycrisp Apple}    0.008836427 0.06011448 2.422752 28408
[2] {Banana} => {Seedless Red Grapes} 0.007650067 0.05204363 2.023410 24594
[3] {Banana} => {Organic Fuji Apple}  0.010558112 0.07182716 2.576259 33943
[4] {Banana} => {Cucumber Kirby}      0.009983906 0.06792082 2.243815 32097
[5] {Banana} => {Organic Whole Milk}  0.009842065 0.06695587 1.560891 31641
>
```

Take a plot of the rules generated, to understand better with the following command:

```
plot(rulesAfterBanana,
method="graph",control=list(type="items",main=""))
```

Output:



4.Deep Learning with word2vec

In the past few years the field of Natural Language Processing has been rigged with new research ideas and new breakthroughs. State of the art scores have been broken on many tasks including Machine Translation, sentiment analysis, dependency parsing and many other tasks. These breakthroughs are due to many factors among which an algorithm called Word2Vec.

4.1 What is word2vec algorithm

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

It is an *unsupervised* algorithm that learns vector representation of words from a huge corpus of text. The way it works is by benefiting from the context data of words, for every word in a given context the algorithm tries to learn vector representation (think of these as features) of each word that would group (cluster) these words together so in a nutshell it would try to group similar words in a vector space.

4.2 What is word2vec useful for

The possibilities of Word2Vec are endless, from transforming any word into a meaningful vector representation instead of the extremely sparse BOW representation this would overcome the curse of dimensionality in NLP models. Word2Vec can be used for tasks like analogies. Word vectors are also the input to any modern NLP model, in Neural Machine Translation an encoder RNN is fed word vectors of input words from a source language and the decoder RNN is responsible for translating the input from the source language to the target language.

Word2Vec can also be used to query nearest neighbours to a certain word, this would usually find words that are similar in meaning to a certain word, we can exploit this in domains other than NLP.

4.3 Implementation in Python

All the 6 datasets that are given by *Instacart* have already been visualized in the *Exploratory Data Analysis* part. The most interesting dataset out of all for this model is the *orders* dataset. The first task of the implementation in python would be to call all the external libraries and the datasets. The external libraries used for this implementation of model is called *gensim* which internally consist of a function called *word2vec* to train a model based on current algorithm. Skipping the part where the data is loaded.

The first step would be to transform the coloumn *product_id* into a string instead of an integer for the datasets *order_products__train & order_products__prior* because this model demands words/sentences instead of integers. The following command is given to achieve this:

```
train_orders["product_id"] =train_orders["product_id"].astype(str)
prior_orders["product_id"] =prior_orders["product_id"].astype(str)
```

where *train_orders* is *order_products__train* and *prior_orders* is *order_products__prior.*

Now, grouping *orders* dataset according to *order_id* into a list of products by using the column *product_id.* This will be done by the following command :

```
train_products = train_orders.groupby("order_id").apply(lambda
order: order['product_id'].tolist())
prior_products = prior_orders.groupby("order_id").apply(lambda
order: order['product_id'].tolist())
```

Each order is now a list of products and each product is represented by its ID string. See the first five orders with products as list using the *head()* function:

```
... print(train_products.head())
order_id
1     [49302, 11109, 10246, 49683, 43633, 13176, 472...
36    [39612, 19660, 49235, 43086, 46620, 34497, 486...
38    [11913, 18159, 4461, 21616, 23622, 32433, 2884...
96    [20574, 30391, 40706, 25610, 27966, 24489, 39275]
98    [8859, 19731, 43654, 13176, 4357, 37664, 34065...
dtype: object
>>>
```

The next step would be to merge the two dataframes (*prior_products & train_products*) together into a new variable called *sentences* and then find the longest order using the *max()* function from external library *numpy*.

```
>>> longest = np.max(sentences.apply(len))
>>> print(longest)
145
>>>
```

Transforming the variable *sentences* into a *numpy* array for further processing by using the function *array()*. The code to implement is:

```
sentences = sentences.values
```

Finally, train a *gensim* implementation of *word2vec* model with the following command:

```
model = gensim.models.Word2Vec(sentences, size=100,
window=longest, min_count=2, workers=4)
```

where

size is the dimensionality of the feature vectors.

window is the maximum distance between the current and predicted word within a sentence.

min_count = ignore all words with total frequency lower than this.

workers = use this many worker threads to train the model (=faster training with multicore machines).

Notice the usage of window = longest in the training of the model. Since there is no sequence characteristics of the products in an order -Because each product in an order is independent on the orders that were chosen before it in the cart- we should have a training window huge enough to accommodate all the products together or else we imply that products that are far apart even though they're in the same cart are dissimilar which is not true.

Next step would be to organize the data for visualization. It is done by the following command:

```
vocab = list(model.wv.vocab.keys())
```

The model has now learnt vector representations of each product (Except for those below min_count) so let's see what has it learnt. First of all we need to project our vectors onto 2 dimensions so we can visualize them. We do the projection by using PCA. This will be done by the following command:

```
pca = PCA(n_components=2)
pca.fit(model.wv.syn0)
```

Principal component analysis (**PCA**) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation). In this case we are interested in finding only two principal component, thus the varialbe *n_components* has a value of 2.

Next, we need two helper functions out of which one will select the random sample from all the predicted output of the model and the other function will generate a plot for better visualization. They are as follows:

```
def get_batch(vocab, model, n_batches)
def plot_with_labels(low_dim_embs, labels, filename)
```

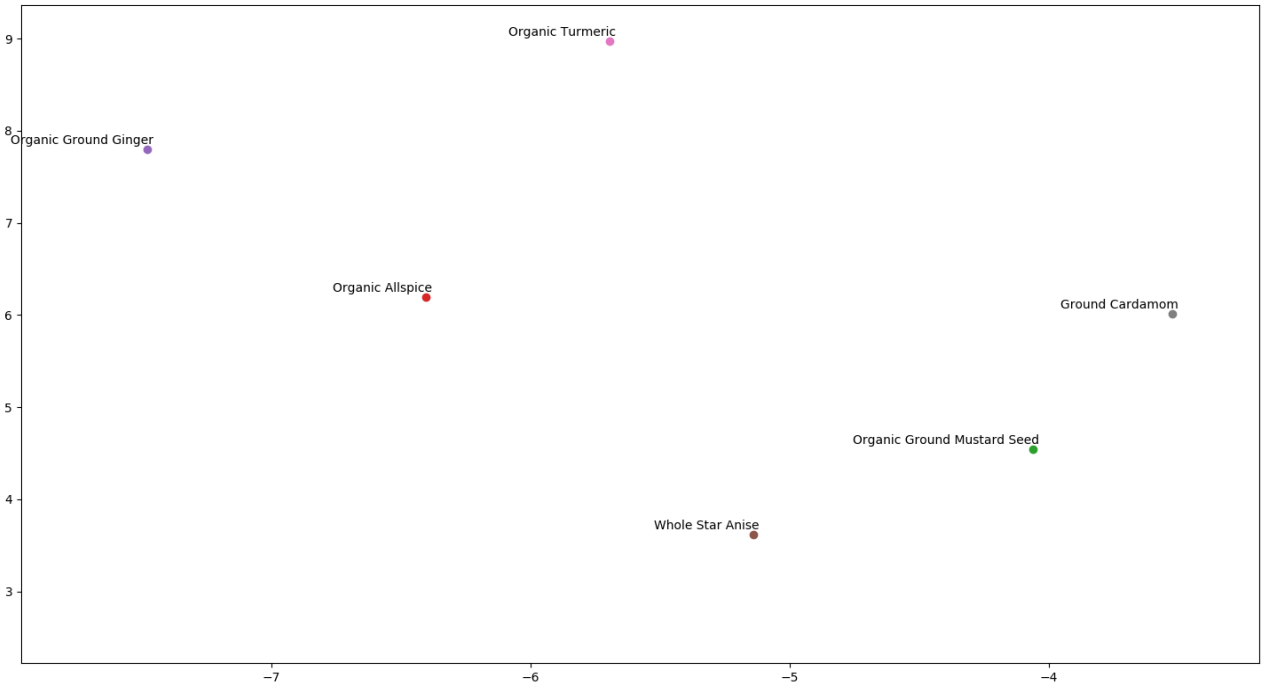
Finally extract random sample of products and visualize these products and their neighbours. This is the job of the *get_batch* function as it will extract the nearest 5 products for each product *n_batches* products it will sample. Let's call these functions and visualize their output using *matplotlib*.

Let's visualize the sample with the following code:

```
embeds = []
```

```
labels = []
for item in get_batch(vocab, model, n_batches=3):
    embeds.append(model[item])
    labels.append(products.loc[int(item)]['product_name'])

embeds = np.array(embeds)
embeds = pca.fit_transform(embeds)
plot_with_labels(embeds, labels)
```

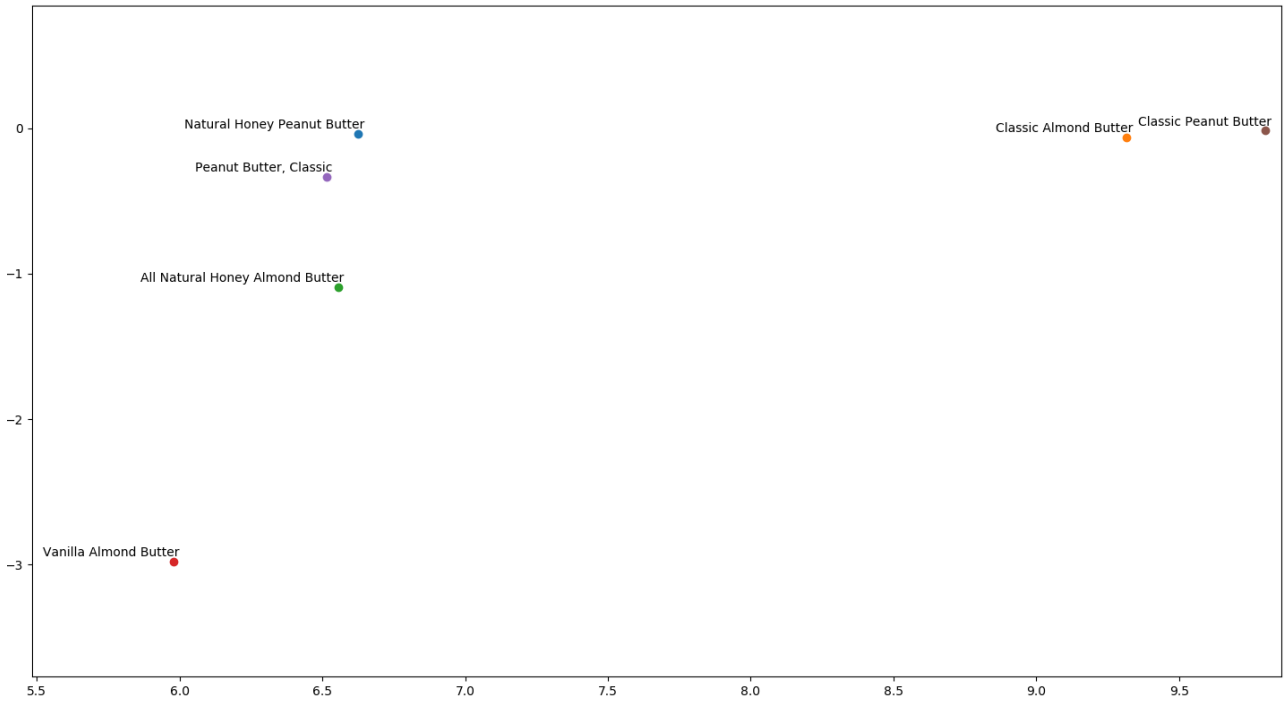
Visualizations with multiple random samples are as follows:



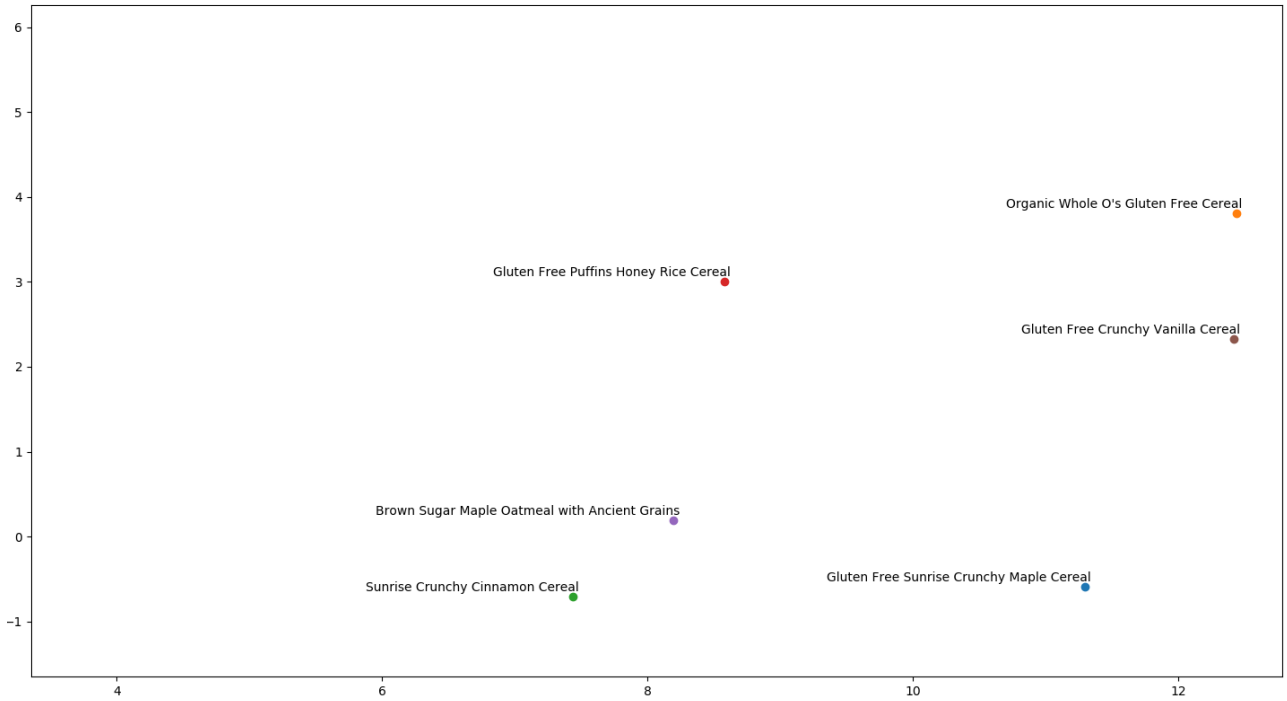
A Cluster of Organic Foods

The original cluster created was very wide, and after much zooming we came to the above graph. As we can very well see that the model predicted that the items shown in the above graph are very highly corelated, and thus are forming clusters. So if a consumer buys *Turmeric, Ginger or Star Anise* it is very likely that the consumer will buy *Mustard Seed, Cardamom or Allspice.*

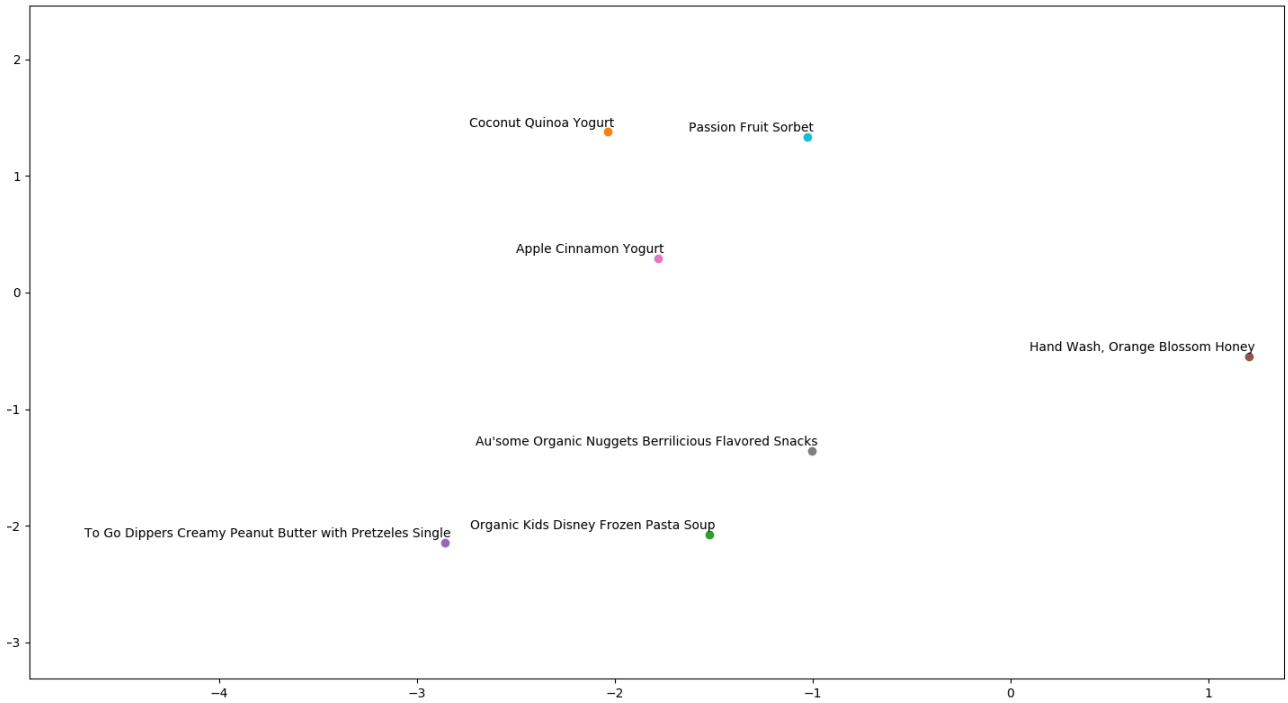
Similarly, we have seen multiple random samples generated from the model. They are below:



Multiple types of Butter cluster



A Cluster of Morning cereals



A Cluster of Kids foods

4.4 Conslusions

Word2Vec is a powerful and very quick algorithm for learning a meaningful representation of words from context information. Furthermore this is not limited to words as you can expand it to many other domains. We applied it to a shopping based problem and it showed very useful insights from the data and can supply useful features for models that can be built to solve a certain task.

5. Limitations of the Project

- As the dataset provided by *Instacart* were having more than 3.2 million transactions, much more complex *Exploratory Data Analysis* and *visualizations* were unable to be produced due to computational limitations.
- Due to the enormous size and dimensions of datasets provided, *differential Market Basket Analysis* would have been a much better choice. But was not included because of the lack of source and knowledge.
- After conversion of *orders* into sentences and *products* into words, when *word2vec* algorithm is applied, due to the enormous corpus of words, unknown or out-of-vocabulary(OOV) words were unhandled by the algorithm.
- This project lacks a system where once an *order* from *order_products_prior* dataset is fed into an algorithm, predicts the most likely *products* that particular consumer is going to buy again.

6. Future Enhancements

- Implementation of *Differential Market Basket Analysis* for better rules and predictions.
- Improving the *word2vec* model by adding more hidden layers of the deep neural network, and training it for a longer period of time.
- Instead of word embeddings, using original dataset to train a *Deep Neural Network* and predict the output.
- Using *Extreme Gradient Boosting* to train a model and predict the output.
- Using a system to check the output accuracy of multiple predictive algorithms, and to understand which one works the best!

6. Bibliography

Exploratory Data Analysis with R : The ggplot2 Plotting System (https://bookdown.org/rdpeng/exdata/the-ggplot2-plotting-system-part-1.html)

In this article, a comprehensive description of how exploratory data analysis is done using the library *ggplot2* is given. Examples and description are given to make sure the reader understands the core concepts.

Raghav Bali, Dipanjan Sarkar. *"R Machine Learning by Example"*. Packt Publishing

In the chapter 3 from this book, named *Predicting Customer Shopping Trends with Market Basket Analysis*, the author explains all the tools and techniques used to predict outcomes from a sample grocery transaction. Association analysis is also described in this chapter.

Jason Browniee. *How to Develop Word Embeddings in Python with Gensim*.

(https://machinelearningmastery.com/develop-word-embeddings-python-gensim/)

In this post, the author explains how the word2vec algorithm works, it's implementation in Python and also, how to visualize the vector space of word2ve algorithm.