Northeastern University

Module 11: Assignment
XN Project: Final Draft – Group submission

Divya Upadhyay - 002139508

Shlok Gadiya - 002103806

Varun Lokesha – 001521139

Utsav Desai – 002103753

Prabh Arjun Singh - 002989231

ALY – 6080: Integrated Experiential Learning

Sec 08 Spring 2022, CPS

Prof. Shanu Sushmita

June 28, 2022

Index

Abstract

During the past several weeks, we researched our client Awan Tunai's industry, their products, competitors, the future scope of their business, and how data can help them to achieve their business goals. In this project, we applied various techniques and learned about different methods and tools that can help us to solve the business questions that we had and we also tried to propose different projects that can be useful to the company. In this final project, we will present our final analysis of the dataset and also present a model that could predict or present basket recommendations to the merchants. In this project, we feel that data preparation was the most intense part since we had to decide which features should be used for each business question and the final model, and then convert those features into a usable format. Since we had limited data, the most challenging part was to draw meaningful and accurate inferences. We will first present the analysis and solution of each business question, then we will introduce our model and present our conclusions.

Visualization and key insights

Part 1: Cleaning the data

First, we have imported the data into R:

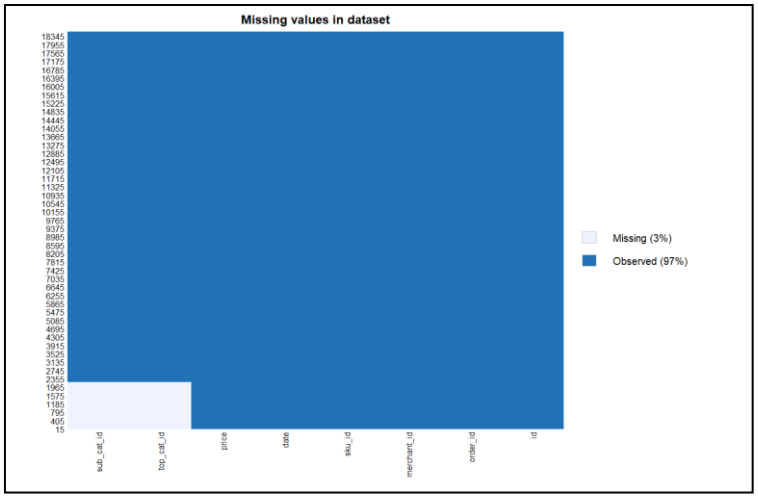
```
> Awan <- read.csv(file.choose())
> head(Awan, n=5)
      id order_id merchant_id sku_id       date top_cat_id sub_cat_id  price
1 654713    82186        1754    981 2021-06-29          7          44 877000
2 654712    82186        1754  23060 2021-06-29          7          44 877000
3 654711    82186        1754   1548 2021-06-29          7          44 877000
4 654714    82186        1754   1969 2021-06-29          7          44 877000
5 654687    82186        1754    343 2021-06-29          7          44 312000
```

Let us look at the structure of the data:

```
> str(Awan)
'data.frame':   18555 obs. of  8 variables:
 $ id         : int  654713 654712 654711 654714 654687 1058235 1866936 1866935 1343282 9
71220 ...
 $ order_id   : int  82186 82186 82186 82186 82186 129785 231405 231405 164702 119929 ...
 $ merchant_id: int  1754 1754 1754 1754 1754 1760 1626 1626 1440 1523 ...
 $ sku_id     : int  981 23060 1548 1969 343 1548 2127 343 927 981 ...
 $ date       : chr  "2021-06-29" "2021-06-29" "2021-06-29" "2021-06-29" ...
 $ top_cat_id : num  7 7 7 7 7 7 7 7 7 7 ...
 $ sub_cat_id : num  44 44 44 44 44 44 44 44 44 44 ...
 $ price      : num  877000 877000 877000 877000 312000 3750 1850 306000 18500 3750 ...
```

We have 18,555 rows of data with 8 variables, 4 are integers, 3 are numbers and 1 is a character variable. We will later change the character and the number variable into factors wherever needed for visualization and analysis.

Now we want to see if we have any NA values that could affect our analysis:



```
> colSums(is.na(Awan))
         id    order_id merchant_id      sku_id        date  top_cat_id  sub_cat_id
          0           0           0           0           0        2208        2208
      price
          0
```

As we can see top category id and sub-category id have 2208 NA values, we can deal with NA values in different ways like replacing them by mean or mode but for our project, the best solution will be to remove these values since replacing the values with mode can affect our final prescriptive analysis but removing these values which are relatively smaller in proportion will not have much effect on the analysis:

```
       id              order_id       merchant_id       sku_id          date
Min.   : 513219   Min.   : 65439   Min.   : 875   Min.   :    0   Length:16347
1st Qu.:1058947   1st Qu.:129522   1st Qu.:1477   1st Qu.:  412   Class :character
Median :1280819   Median :156847   Median :1754   Median : 2037   Mode  :character
Mean   :1291425   Mean   :158543   Mean   :1813   Mean   : 6174
3rd Qu.:1550515   3rd Qu.:191249   3rd Qu.:2139   3rd Qu.:12426
Max.   :1958354   Max.   :242390   Max.   :3114   Max.   :23756
  top_cat_id       sub_cat_id         price                year          month
Min.   : 0.000   Min.   : 0.00   Min.   :       0   Min.   :2021   Min.   : 5.00
1st Qu.: 0.000   1st Qu.:12.00   1st Qu.:   41500   1st Qu.:2021   1st Qu.:10.00
Median : 5.000   Median :18.00   Median :  102500   Median :2021   Median :10.00
Mean   : 4.988   Mean   :24.74   Mean   :  139603   Mean   :2021   Mean   :10.23
3rd Qu.: 7.000   3rd Qu.:28.00   3rd Qu.:  180000   3rd Qu.:2021   3rd Qu.:11.00
Max.   :29.000   Max.   :82.00   Max.   :58750000   Max.   :2021   Max.   :12.00
     day
Min.   : 1.00
1st Qu.: 8.00
Median :17.00
Mean   :16.39
3rd Qu.:25.00
Max.   :31.00
```

As we can see from the summary, there are no more NA values in the data set. After removing NA values, we decided to create a few more handful variables from the date variable, such as a month, week number within the month, and year quarter. Below graphic shows some observations after we created new variables:

```
> head(AwanTunai_data, n = 10)
       id order_id merchant_id sku_id       date top_cat_id sub_cat_id  price month month_week_no quarter
1   654713    82186        1754    981 2021-06-29          7         44 877000     6             5      2nd
2   654712    82186        1754  23060 2021-06-29          7         44 877000     6             5      2nd
3   654711    82186        1754   1548 2021-06-29          7         44 877000     6             5      2nd
4   654714    82186        1754   1969 2021-06-29          7         44 877000     6             5      2nd
5   654687    82186        1754    343 2021-06-29          7         44 312000     6             5      2nd
6  1058235   129785        1760   1548 2021-10-04          7         44   3750    10             2      4th
7  1866936   231405        1626   2127 2021-12-24          7         44   1850    12             5      4th
8  1866935   231405        1626    343 2021-12-24          7         44 306000    12             5      4th
9  1343282   164702        1440    927 2021-11-05          7         44  18500    11             2      4th
10  971220   119929        1523    981 2021-09-24          7         44   3750     9             5      3rd
```

Let's also look at the summary of the dataset:

```
> summary(Awan)
       id              order_id       merchant_id       sku_id          date
 Min.   : 440787   Min.   : 55780   Min.   : 875   Min.   :    0   Length:18555
 1st Qu.:1061537   1st Qu.:129518   1st Qu.:1477   1st Qu.:  415   Class :character
 Median :1301516   Median :158683   Median :1758   Median : 2037   Mode  :character
 Mean   :1305701   Mean   :160114   Mean   :1815   Mean   : 6176
 3rd Qu.:1597224   3rd Qu.:196004   3rd Qu.:2139   3rd Qu.:12426
 Max.   :1958354   Max.   :242390   Max.   :3114   Max.   :23756

   top_cat_id       sub_cat_id         price
 Min.   : 0.000   Min.   : 0.00   Min.   :       0
 1st Qu.: 0.000   1st Qu.:12.00   1st Qu.:   43000
 Median : 5.000   Median :18.00   Median :  102000
 Mean   : 4.988   Mean   :24.74   Mean   :  138881
 3rd Qu.: 7.000   3rd Qu.:28.00   3rd Qu.:  179000
 Max.   :29.000   Max.   :82.00   Max.   :58750000
 NA's   :2208     NA's   :2208
>
```

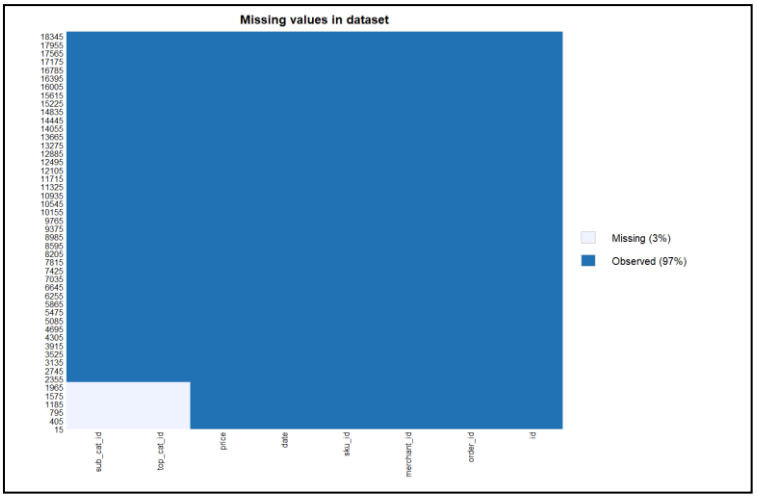
The summary doesn't give us any proper insight into the dataset as most of the columns are ids, we only have an idea that on an average the total order value is 1,38,881 Rupiah.

Let us also take a look at the unique sku_ids and merchant ids that we have after removing the NA values:

```
> length(unique(new_Awan$sku_id))
[1] 714
> length(unique(new_Awan$merchant_id))
[1] 495
>
```

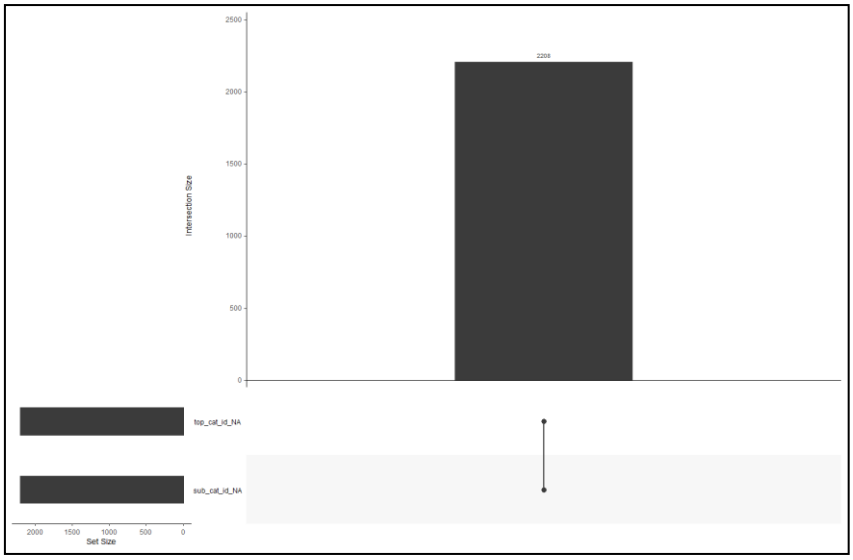
Part 2: EDAs for each of the questions.

Q. 1: Do merchants adjust their SKU purchases over time and if so, what are the different ways they adjust their purchases?

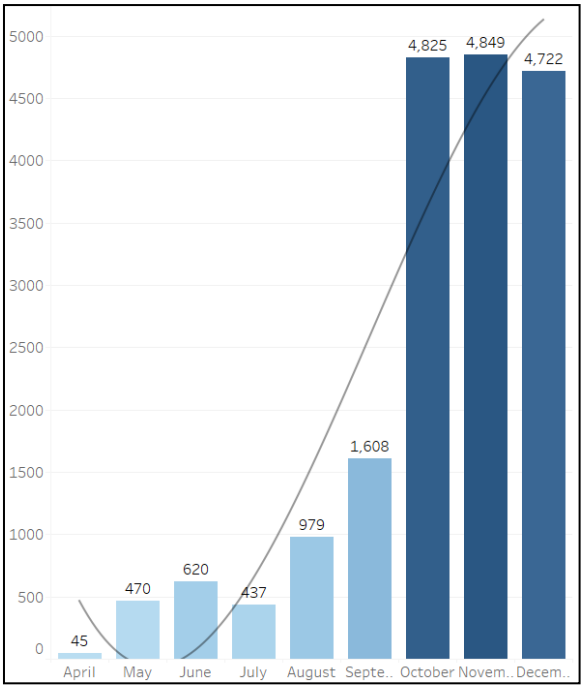


First, I count the number of missing data to make coding easier later on in this report. Then, using the missmap() function from the Amelia package, I create a map to identify all of the accessible missing values, which we will use to code later on. After that, I perform a double check by computing the total number of missing values using the sum(is.na(x)) function of sapply(). As illustrated in the preceding image, there are around 3% missing values in the data set under consideration. Also, as shown in the following graphic, there are around 2208 N/A values in the columns: Top Category ID (top_cat_id) and Sub Category ID (sub_cat_id).

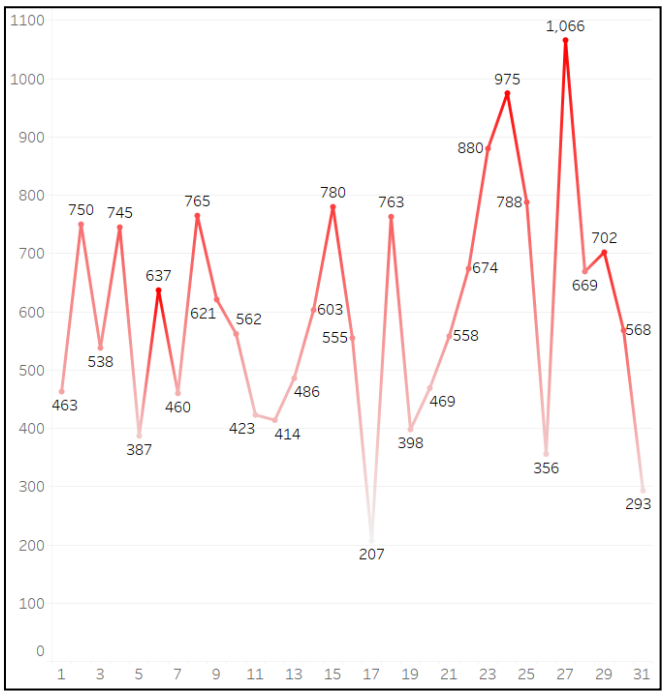
```
> sapply(awantunai,function(x) sum(is.na(x)))
        id   order_id merchant_id     sku_id       date top_cat_id sub_cat_id      price
         0          0           0          0          0       2208       2208          0
```



It is a fundamental idea for any data scientist and analyst. The datasets and the values they contain are always our major concern before moving on to any other steps. As you can see, we made two graphs for the missing values, but they are extremely different. So, while addressing missing values, I discovered that aside from NA values, certain datasets had blank values. That's when I understood that only read.csv will read the CSV file and display its contents. So, the issue was at the data gathering source. To solve this problem, I discovered that the read.csv function has two parameters: header and as.strings. The na.strings option matches strings within the file that should be replaced with NA. In this case, the empty string should match the missing character string, which is white space.

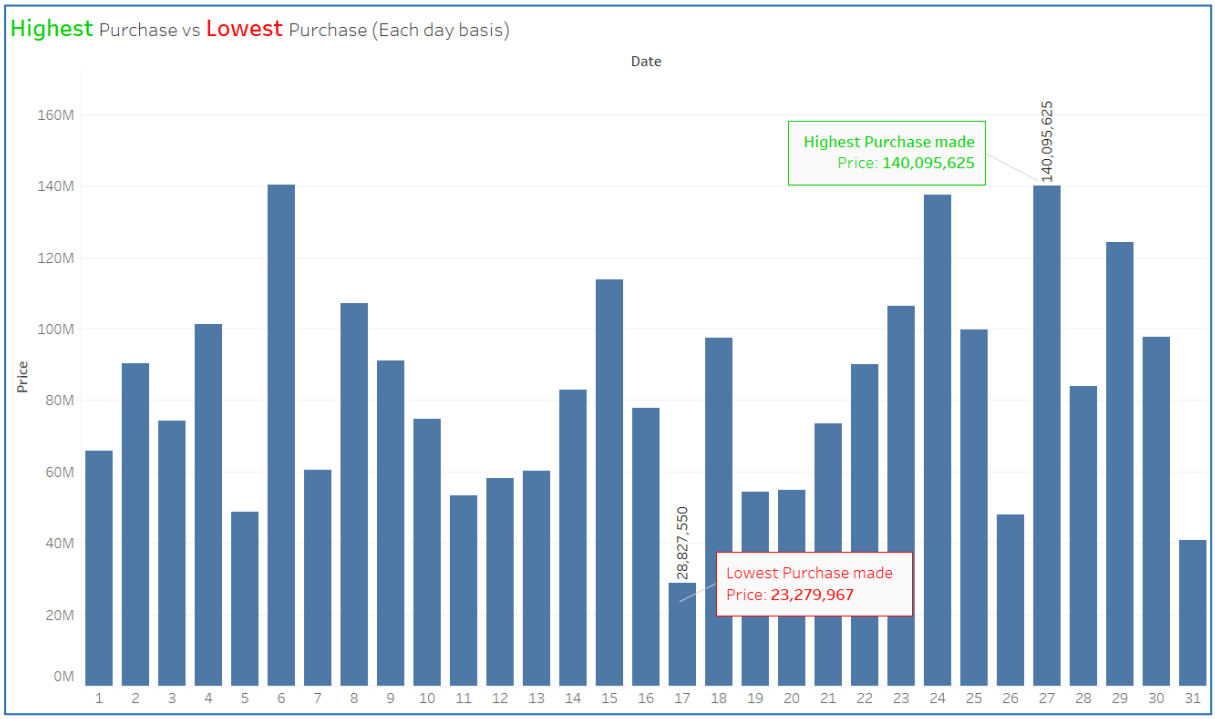


The above figure is a bar chart showing the SKUs that were purchased by the merchant for each month of the year. In order to better observe the pattern, I've added a trend line with the degree of the polynomial, which is three. It has been steadily increasing month after month.

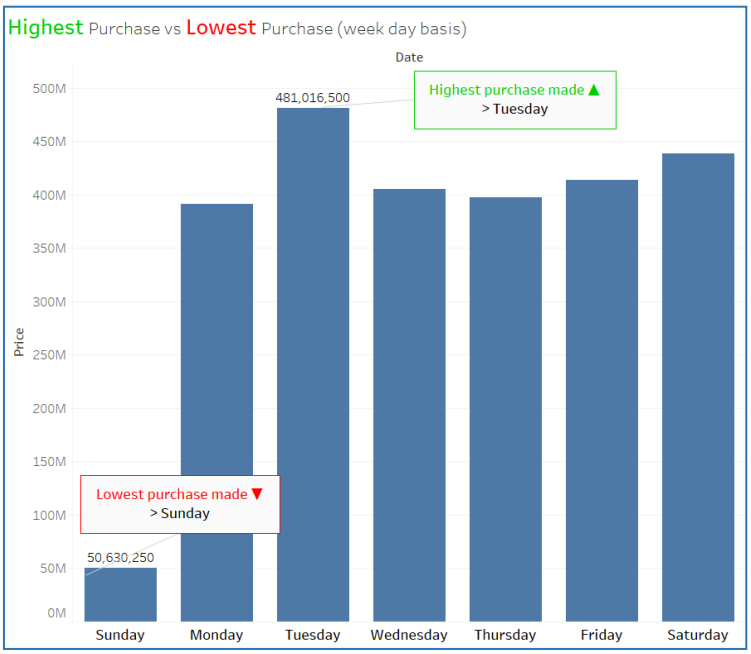


The line chart in the above picture depicts the total number of SKUs purchased by the merchant on each day of a given month. If you follow the pattern, you'll see that more purchases were made near the end of the month than at the beginning. When it comes to monthly purchases, it was found that the biggest number of purchases were made on the 27th of every month and that the lowest number of purchases were made on the 17th of every month.

Now to support the above statement: “When it comes to monthly purchases, it was found that the biggest number of purchases were made on the 27th of every month and that the lowest number of purchases were made on the 17th of every month”, I have made this below graph which supports my statement.



As you can see from the above figure, the lowest purchase made from the merchants were on the 17th of each month and the highest purchase made were on the 27th of each month.



If we now examine the figure next to this text box, we can see that merchants purchased a larger number of SKUs on Tuesdays and Saturdays. Sunday was the most discounted day for the purchase.

Question-2: /Do merchants tend to repeat their purchase patterns?

The main objective of this problem was to find out repetitive trends or behaviors of the merchants over time and for which particular products merchants repeated the purchase. Moreover, how do these patterns differ from merchant to merchant?

```
[11] AwanTunai.nunique()

id             18555
order_id        2794
merchant_id      505
sku_id           735
date             187
top_cat_id        29
sub_cat_id        83
price            685
dtype: int64
```

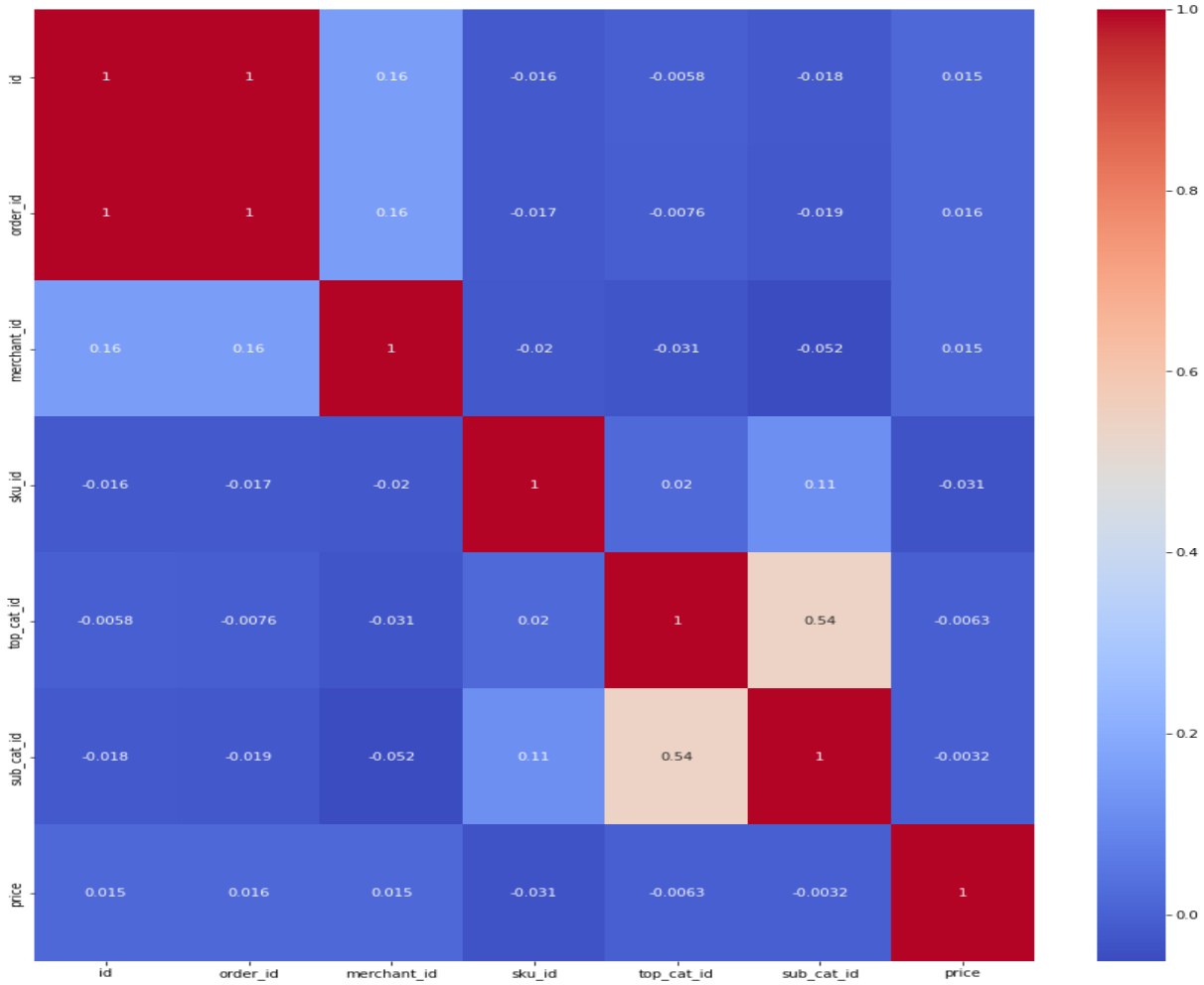
Unique values.

Correlation between the attributes:

	id	order_id	merchant_id	sku_id	top_cat_id	sub_cat_id	price
id	1.000000	0.997886	0.155179	-0.016225	-0.005791	-0.018447	0.015410
order_id	0.997886	1.000000	0.155718	-0.017221	-0.007604	-0.018911	0.015534
merchant_id	0.155179	0.155718	1.000000	-0.019864	-0.030743	-0.051571	0.014967
sku_id	-0.016225	-0.017221	-0.019864	1.000000	0.020028	0.112809	-0.031485
top_cat_id	-0.005791	-0.007604	-0.030743	0.020028	1.000000	0.540630	-0.006254
sub_cat_id	-0.018447	-0.018911	-0.051571	0.112809	0.540630	1.000000	-0.003238
price	0.015410	0.015534	0.014967	-0.031485	-0.006254	-0.003238	1.000000

Correlation Table.

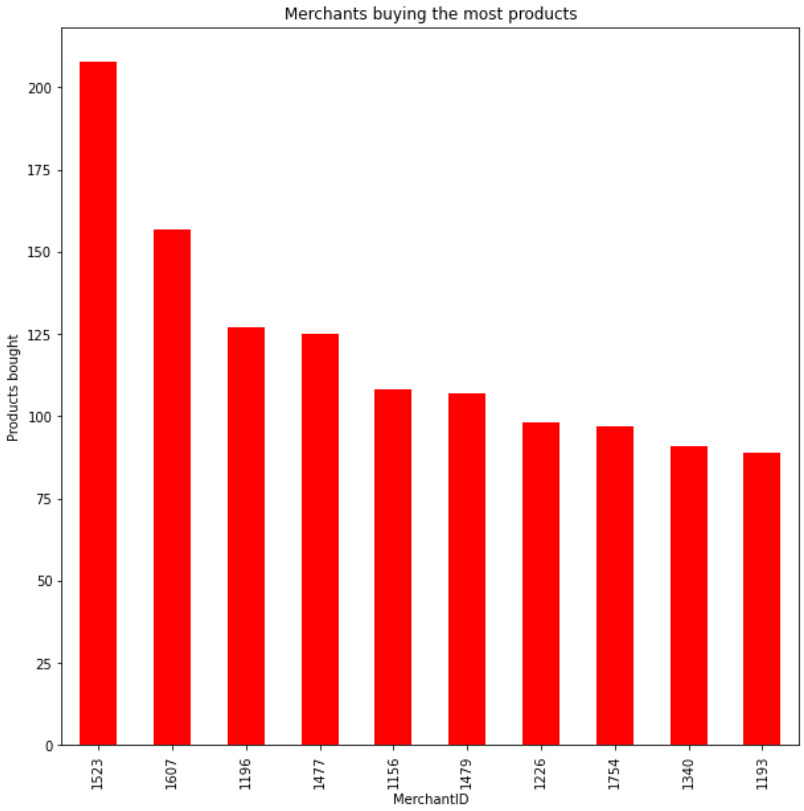
We can see that the attributes with the highest positive correlation are ‘order_id’ and ‘id’ and the attributes with the highest negative correlation are ‘sub_cat_id’ and ‘merchant_id’. This can also be visualized as follows:



Correlation Plot.

Most Frequent buyers:

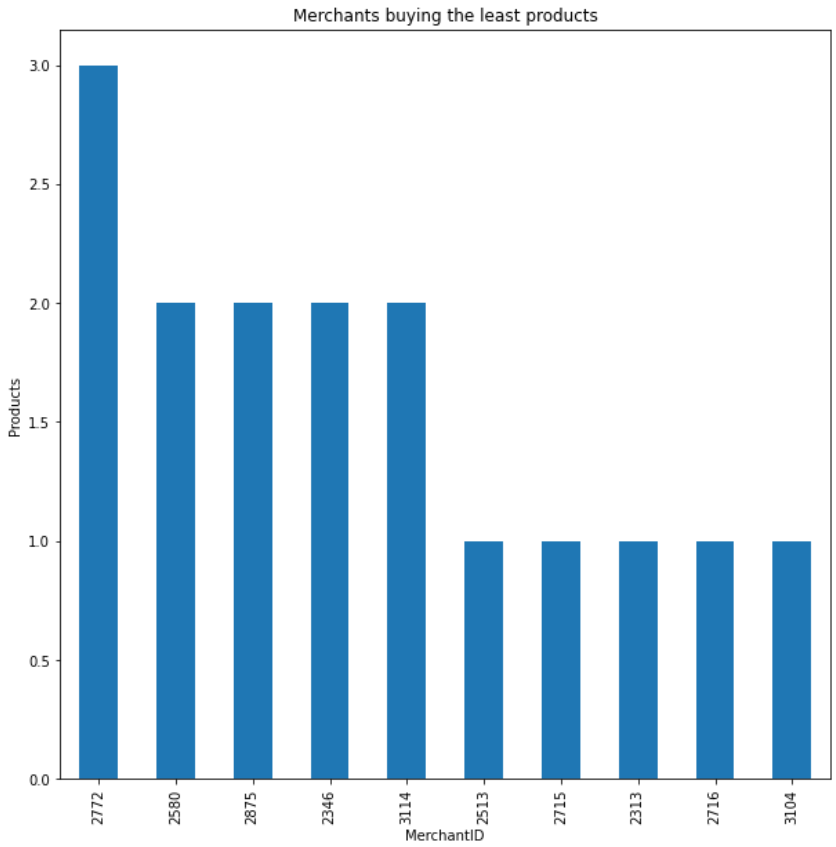
To get the patterns and compare the purchase trends of the merchants we need to know the merchants who are purchasing the most and least products.



Top buyers

We can see that merchant with merchant_id 1523 and 1607 purchased the most products whereas merchant 1523 has bought products over 200 times and merchant 1607 has bought products nearly 160 times.

Least Frequent Buyers:



Least Frequent buyers.

We can see that merchant id number 2513, 2715, 2313, 2716, and 3104 has bought the products just once.

Merchants' monthly buying trends:

			id	order_id	date	top_cat_id	sub_cat_id	price	year
			count	count	count	count	count	count	count
merchant_id	sku_id	month							
875	4	11	1	1	1	1	1	1	1
	8	11	1	1	1	1	1	1	1
	60	12	1	1	1	1	1	1	1
	105	11	1	1	1	1	1	1	1
	232	11	2	2	2	2	2	2	2
...	...	...	...	...	...	...	...	...	...
3104	2936	12	1	1	1	1	1	1	1
3114	1204	10	1	1	1	1	1	1	1
		11	2	2	2	2	2	2	2
	2740	10	1	1	1	1	1	1	1
		11	2	2	2	2	2	2	2

13853 rows × 7 columns

The above table is depicting the merchants with their corresponding sku_id and the months they were bought in. We can also see the frequency of the products being bought.

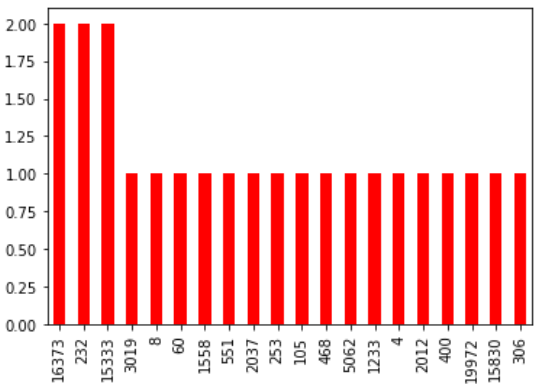
Looking at individual merchant trends:

Since the merchant with merchant_id = 1523 is the most frequent buyer we can compare his/her purchasing trends with the merchant with merchant_id = 875 and 3114.

Purchase Trends of merchant_id = 875:

			count						
			date	id	merchant_id	price	sub_cat_id	top_cat_id	year
sku_id	month	order_id							
4	11	189135	1	1	1	1	1	1	1
8	11	179014	1	1	1	1	1	1	1
60	12	210448	1	1	1	1	1	1	1
105	11	193034	1	1	1	1	1	1	1
232	11	178936	1	1	1	1	1	1	1
		192722	1	1	1	1	1	1	1
253	11	192722	1	1	1	1	1	1	1
306	11	178936	1	1	1	1	1	1	1
400	11	178936	1	1	1	1	1	1	1
468	11	193034	1	1	1	1	1	1	1
551	12	210448	1	1	1	1	1	1	1
643	11	161145	1	1	1	1	1	1	1
688	11	161145	1	1	1	1	1	1	1
1233	11	189135	1	1	1	1	1	1	1

Pivot table for purchase history of merchant 875.



Top 20 purchases of merchant_id = 875.

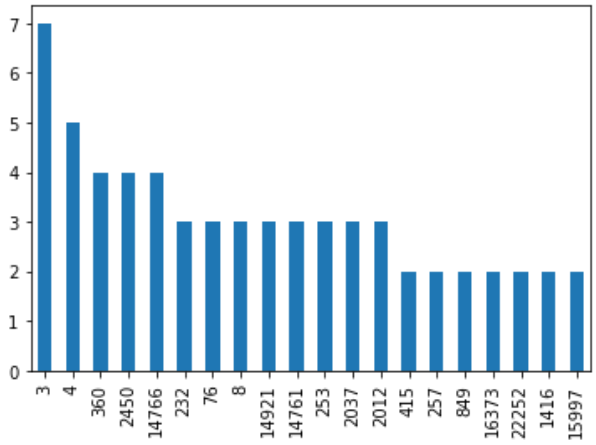
From the above graph, we can see that product with sku_id = 16373, 232, and 15333 is bought twice which is the most by this merchant.

Purchase Trends for merchant_id = 1523:

			count						
			date	id	merchant_id	price	sub_cat_id	top_cat_id	year
sku_id	month	order_id							
2	10	150035	2	2	2	2	2	2	2
3	9	119929	4	4	4	4	4	4	4
	10	150035	3	3	3	3	3	3	3
4	9	119929	3	3	3	3	3	3	3
	10	150035	2	2	2	2	2	2	2
...	...	...	...	...	...	...	...	...	...
23281	9	119929	1	1	1	1	1	1	1
23456	9	119929	1	1	1	1	1	1	1
23747	9	119929	1	1	1	1	1	1	1
	10	150035	1	1	1	1	1	1	1
23749	9	119929	1	1	1	1	1	1	1

277 rows × 7 columns

Pivot table showing trends for merchant_id =1523



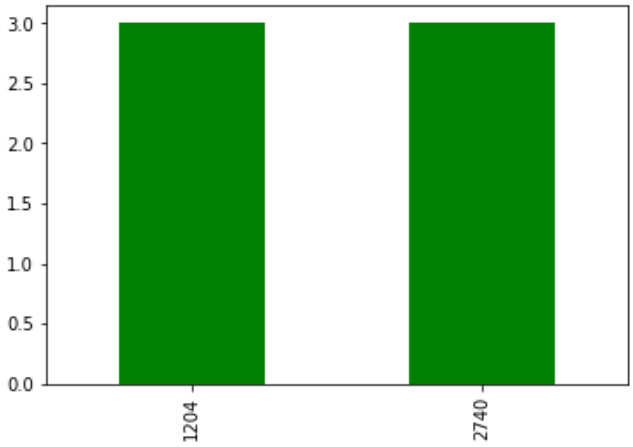
Top 20 purchases of merchant_id =1523

From the above plot, we can say that product with sku_id = 3 was bought seven times which is the most by this merchant, and products with sku_id =4 were bought five times which is the second-highest frequency of purchase by this merchant.

Purchase trends of merchant_id =3114:

			count						
			date	id	merchant_id	price	sub_cat_id	top_cat_id	year
sku_id	month	order_id							
1204	10	138832	1	1	1	1	1	1	1
	11	160446	1	1	1	1	1	1	1
		187745	1	1	1	1	1	1	1
2740	10	138832	1	1	1	1	1	1	1
	11	160446	1	1	1	1	1	1	1
		187745	1	1	1	1	1	1	1

Pivot table for merchant_id =3114



Top purchases of merchant_id =3114.

From the above table and plot, we can see that only two products were bought by this merchant. The products are sku_id = 1204 and 2740. Both were bought thrice once in October and twice in November.

Comparing Sku_id:

Comparing the most products bought by the month:

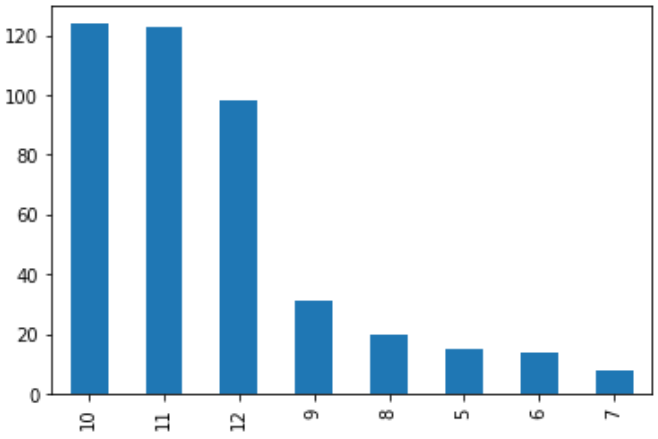
```
df['sku_id'].value_counts()[:5]
```

```
2037    433
360     404
8       397
468     354
1558    291
Name: sku_id, dtype: int64
```

Top 5 products bought.

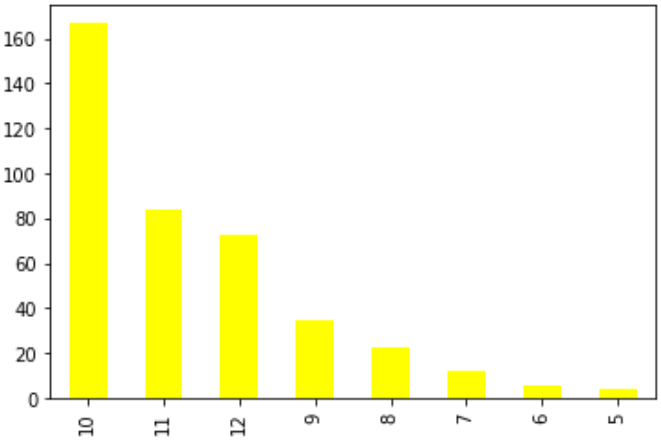
The two topmost products that are most frequently bought are sku_id = 2037 and 360.

Comparing their purchase histories categorizing by month.



Purchase patterns of sku_id = 2037

This product was bought the most i.e., over 120 times in the 10th month (October) and nearly the same times in November.



Purchase patterns of sku_id = 360

This product was bought the most times over 160 in October and around 80 times (second highest) in November.

Q. 3 Do sku_ids move at different speeds and if so, what are the fast-moving and slow items/categories?

For this question we have tried to break the dates into 3 separate columns so that we can use them for further exploration:

```
> head(new_Awan_1,5)
      id order_id merchant_id sku_id       date top_cat_id sub_cat_id  price year
1 654713    82186        1754    981 2021-06-29          7         44 877000 2021
2 654712    82186        1754  23060 2021-06-29          7         44 877000 2021
3 654711    82186        1754   1548 2021-06-29          7         44 877000 2021
4 654714    82186        1754   1969 2021-06-29          7         44 877000 2021
5 654687    82186        1754    343 2021-06-29          7         44 312000 2021
  month day
1     6  29
2     6  29
3     6  29
4     6  29
5     6  29
```

As we can see, we have 3 new columns in the data frame- year, month, and day, however, we have kept the date column as it is and not removed it from the data frame.

I will now try to see the frequency of purchase for all the sku_ids, this will give us a base later on to compare and verify our results for the fast-moving and the slow-moving items. I will use the group by() function to group the data according to sku_ids and then use summarize() function to generate a column for the total number of times the sku_ids have been purchased:

```
> head(by_sku_id,5)
# A tibble: 5 x 2
  sku_id total
   <int> <int>
1      0    25
2      1     4
3      2    78
4      3   157
5      4   180
> tail(by_sku_id,5)
# A tibble: 5 x 2
  sku_id total
   <int> <int>
1  23746   106
2  23747    17
3  23748    13
4  23749    54
5  23756     3
>
```

Let’s arrange them in ascending and descending order :

```
> by_sku_id[rev(order(by_sku_id$total)),]
# A tibble: 714 x 2
   sku_id total
    <int> <int>
 1   2037   433
 2    360   404
 3      8   397
 4    468   354
 5   1558   291
 6  14766   265
 7     33   252
 8   2740   230
 9   1767   222
10   1204   222
# ... with 704 more rows
```

The above table shows the most frequently purchased sku_ids.

```
> by_sku_id[order(by_sku_id$total),]
# A tibble: 714 x 2
   sku_id total
    <int> <int>
 1     20     1
 2     41     1
 3     42     1
 4     80     1
 5    154     1
 6    179     1
 7    189     1
 8    228     1
 9    260     1
10    335     1
# ... with 704 more rows
```

This table shows the least frequently purchased sku_ids. However, we should note that these sku_ids have appeared in the dataset only once.

Now, for knowing the fast-moving and the slow-moving items, I will create a table by grouping the data according to sku_ids and date, the next column will use the summarize () function to represent the total number of times the sku_ids have appeared on that date and the last column will be the difference between the first date and the second date. Since we do not have the in-dates for the sku_ids, we are trying to find out the difference between the selling

dates of sku_ids to see how many days does it take to sell the second item for the same sku_ids, it will give us the number of days in between each sale for every sku_id:

```
# Groups:   sku_id [1]
  sku_id date       total  diff
   <int> <chr>      <int> <dbl>
1      0 2021-08-24     2     0
2      0 2021-09-18     1    25
3      0 2021-09-25     1     7
4      0 2021-10-15     1    20
5      0 2021-10-16     2     1
> tail(by_date_id,5)
# A tibble: 5 x 4
# Groups:   sku_id [2]
  sku_id date       total  diff
   <int> <chr>      <int> <dbl>
1  23749 2021-12-27     1     3
2  23749 2021-12-31     1     4
3  23756 2021-09-06     1     0
4  23756 2021-10-26     1    50
5  23756 2021-11-04     1     9
>
```

To convert the date into proper format I have also use as.date() function so that when I will do the average of the day difference, it will not throw me an error:

```
by_date_id$diff <- unlist(tapply(as.Date(test$date), INDEX = test$sku_id,
                                 FUN = function(x) c(0, `units<-`(diff(x), "days"))))
```

```
> head(by_date_id)
# A tibble: 6 x 4
# Groups:   sku_id [1]
  sku_id date       total  diff
   <int> <chr>      <int> <dbl>
1      0 2021-08-24     2     0
2      0 2021-09-18     1    25
3      0 2021-09-25     1     7
4      0 2021-10-15     1    20
5      0 2021-10-16     2     1
6      0 2021-10-18     1     2
>
```

Now I will take an average of day difference, which means that for each sku_ids I will add all the difference in days and divide it by the total number of instances which gives me the following table:

```
> by_date_id_group
# A tibble: 714 x 2
   sku_id Avg_diff
    <int>    <dbl>
 1      0     7.81
 2      1     4.75
 3      2     4.07
 4      3     3.05
 5      4     3.17
 6      8     2.17
 7      9    11
 8     15    13
 9     18     6.53
10     20     0
# ... with 704 more rows
>
```

Here we can see each sku_id and on average how many days does it take for them to make the next sales. For e.g., for sku_id 0, it takes around 8 days to sell the next item after the first sale has been made. To see the maximum average day difference and the minimum average day difference for these sku_ids:

```
> #checking the maximum value
>  max(by_date_id_group$Avg_diff)
[1] 64
> min(by_date_id_group$Avg_diff)
[1] 0
>
```

We can see the max average day difference is 64 days and the minimum average day difference is 0 days. We will investigate further whether the items whose average day difference is 0 are actually the fastest moving items or if we just have 1 entry for those items. I will now arrange the data in descending order to see the items with the highest average day difference, those items will be our slowest moving items:

```
> by_date_id_group %>%
+   arrange(desc(Avg_diff))
# A tibble: 714 x 2
   sku_id Avg_diff
    <int>    <dbl>
 1  21716     64
 2   6420     56.3
 3    404     55.5
 4  23166     54.3
 5    143     53
 6  23060     52.7
 7   2895     50
 8  23174     50
 9   1027     49.3
10   5959     49
# ... with 704 more rows
> |
```

These are out top 10 slowest moving items.

Now let's see our fastest moving items:

```
> by_date_id_group %>%
+   arrange(Avg_diff)
# A tibble: 714 x 2
   sku_id Avg_diff
    <int>    <dbl>
 1     20        0
 2     41        0
 3     42        0
 4     80        0
 5    154        0
 6    179        0
 7    189        0
 8    228        0
 9    260        0
10    271        0
# ... with 704 more rows
> |
```

Now when I have checked, there are total of 134 items whose average day difference is 0 but all these sku_ids, except 3(which have appeared twice on the same date), have appeared only once in the dataset. And as I had mentioned earlier that our fast moving and slow moving items are based on the sales dates, hence we cannot take these sku_ids into consideration as

they have appeared just once in 8 months. Therefore, I am going to ignore these sku_ids and consider those whose average day difference is more than or equal to 1:

```
> fast_sku_id %>%
+ arrange(Avg_diff)
# A tibble: 580 x 2
   sku_id Avg_diff
    <int>    <dbl>
 1  13343     1
 2  12426     1.26
 3  12048     1.4
 4   7536     1.5
 5   8881     1.5
 6    360     1.81
 7   2037     1.85
 8    468     1.86
 9  14766     1.98
10    951     2
# ... with 570 more rows
>
```

Hence these are our fastest moving sku_ids.

Now to cross-check our results I will compare the fast moving and the slow moving items with the most frequently and the least frequently purchased items:

```
# A tibble: 714 x 2
   sku_id total
    <int> <int>
 1   2037   433
 2    360   404
 3      8   397
 4    468   354
 5   1558   291
 6  14766   265
 7     33   252
 8   2740   230
 9   1767   222
10   1204   222
# ... with 704 more rows
>
```

```
# A tibble: 580 x 2
   sku_id Avg_diff
    <int>    <dbl>
 1  13343     1
 2  12426     1.26
 3  12048     1.4
 4   7536     1.5
 5   8881     1.5
 6    360     1.81
 7   2037     1.85
 8    468     1.86
 9  14766     1.98
10    951     2
# ... with 570 more rows
>
```

```
# A tibble: 571 x 2
   sku_id Avg_diff
    <int>    <dbl>
 1    951     2
 2   5627     2
 3   1558     2.11
 4   5068     2.17
 5      8     2.17
 6  14761     2.31
 7   5062     2.33
 8   6694     2.33
 9   1204     2.37
10   2740     2.38
# ... with 561 more rows
>
```

As we can see that the most frequently purchased items (table 1) are in fact the fastest moving items (tables 2 and 3).

However, if we compare the least frequently purchased items with the slowest moving items we can see that not all items match:

	sku_id	total
	<int>	<int>
1	62	2
2	143	2
3	271	2
4	404	2
5	437	2
6	600	2
7	699	2
8	887	2
9	951	2
10	1235	2

... with 573 more rows

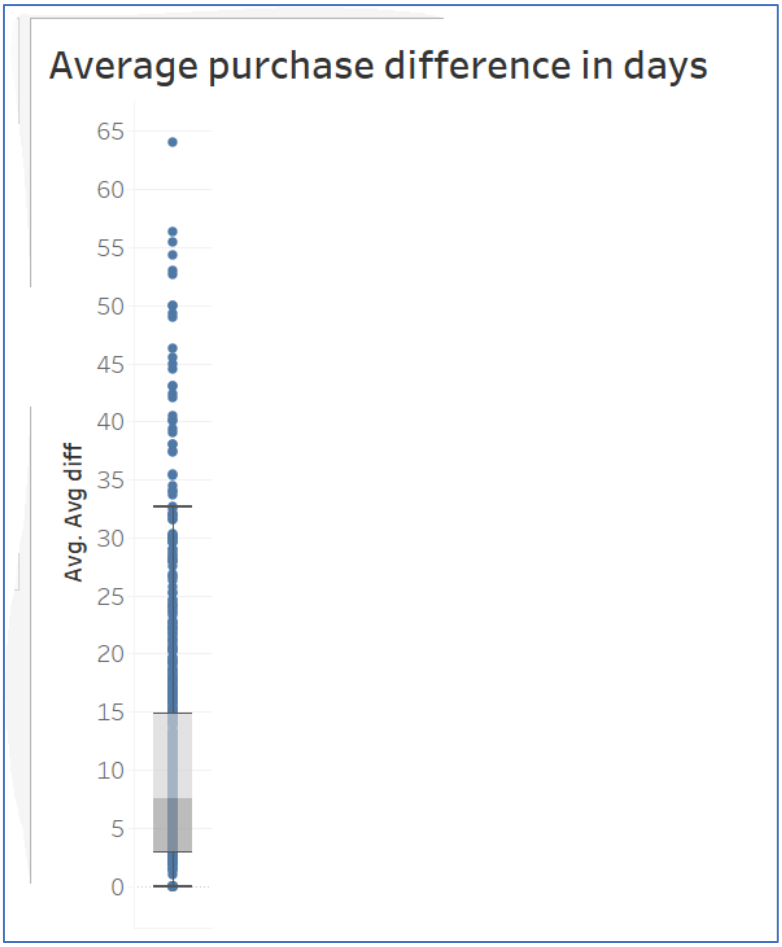
A tibble: 714 x 2

	sku_id	Avg_diff
	<int>	<dbl>
1	21716	64
2	6420	56.3
3	404	55.5
4	23166	54.3
5	143	53
6	23060	52.7
7	2895	50
8	23174	50
9	1027	49.3
10	5959	49

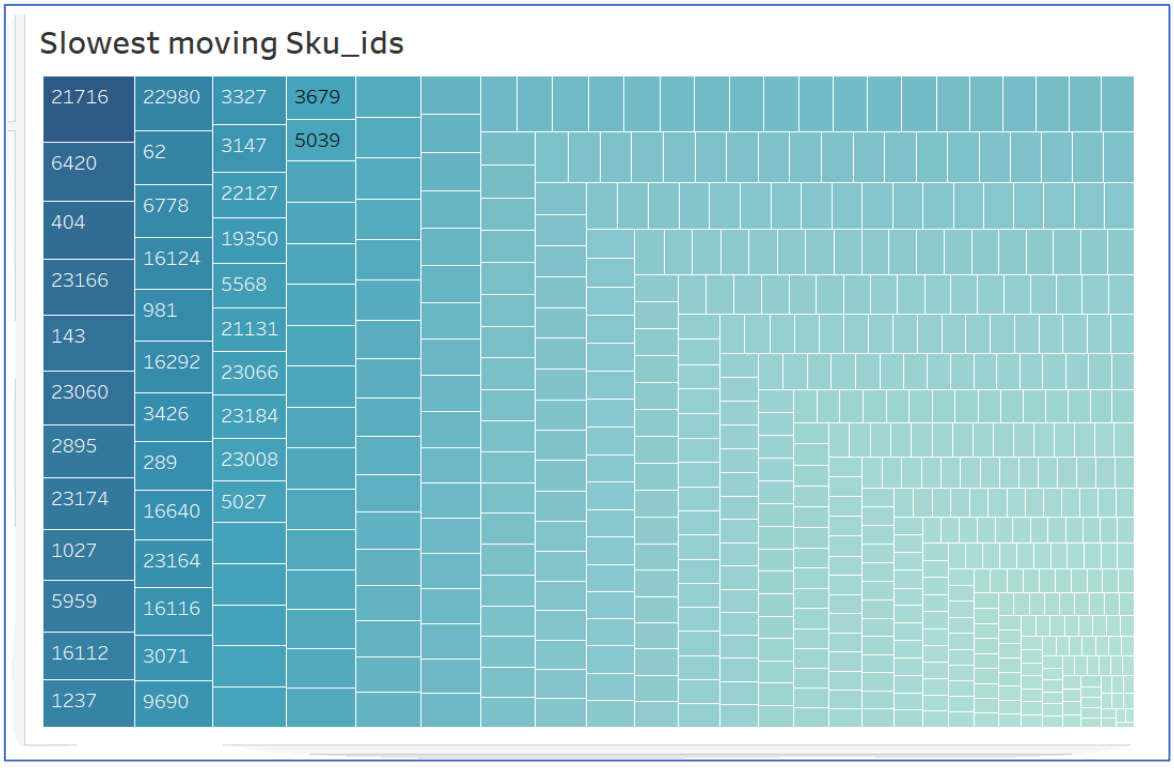
... with 704 more rows

This confirms our conclusion and we can say that we have successfully found the fast moving and the slow moving items. I will now convert my table into csv and make some visualizations in Tableau.

The first boxplot shows that on an average, the items have a day difference of 3 to 15 days between their sale dates. We can also see that our slow moving and fast moving items will be considered outliers.



The next chart is the heat map which shows the slowest moving items:



Question 4 - Do merchant transaction histories form any natural clusters that have similar purchase baskets and buying patterns, and if so, what are some clusters and patterns?

Clustering analysis is the process of grouping objects so that objects in the same cluster are closer to one another or similar than to the objects in another cluster. Clustering helps us to classify data into structures that can be easily understood and manipulated. Clustering analysis requires adequate data preparation to understand and group data points into clusters so that they can, later on, be manipulated to draw insights.

Clustering analysis also requires adequate data preparation so that insightful clustering analysis can be carried out.

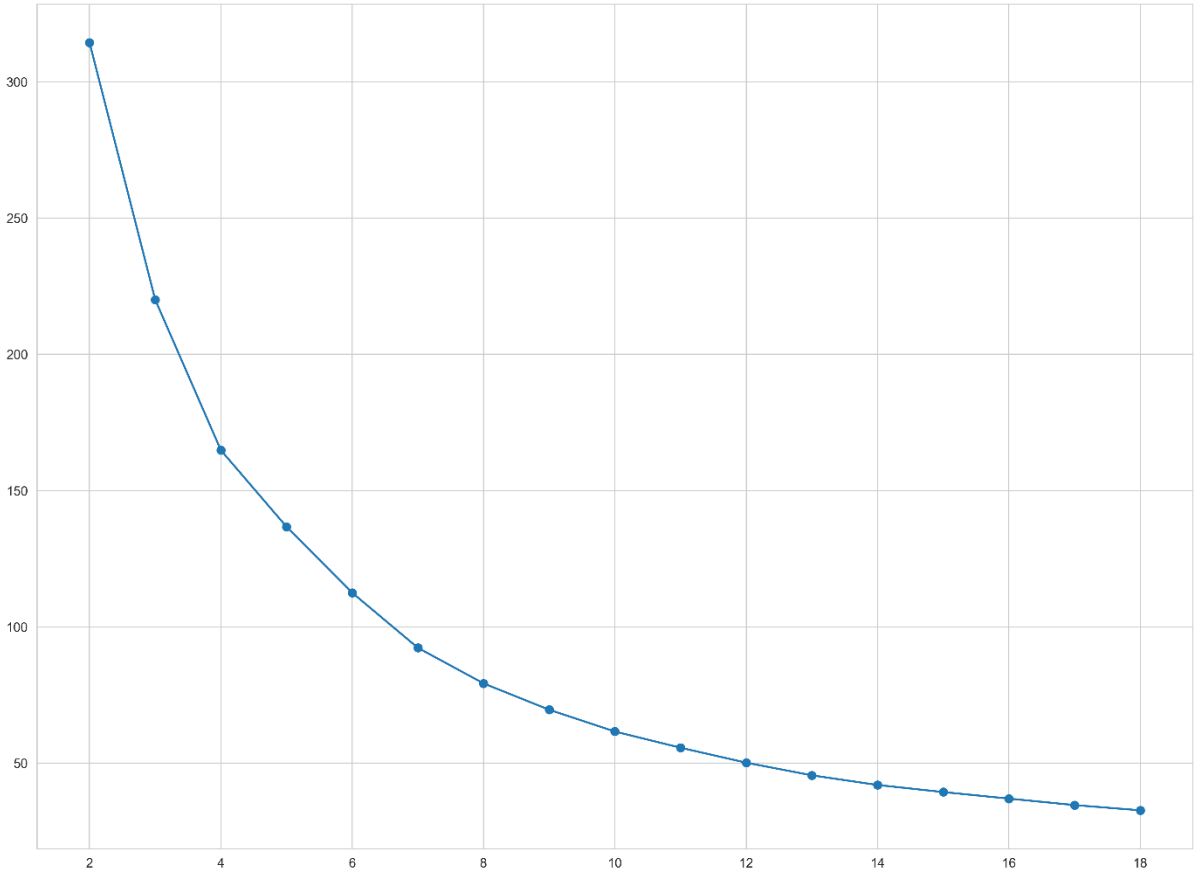
For model 1, clustering analysis is performed on the number of total orders placed by each merchant during the fiscal year. This would help us categorize merchants into different clusters based on their order count. Later on, using, the same data, we can further analyse differences between merchants and suggest improvements.

	merchant_id	order_count
0	875	32
1	1016	5
2	1041	8
3	1103	48
4	1156	249
..	...	...
490	2954	11
491	2973	8
492	3075	4
493	3104	1

The above picture shows the data values of how many orders each merchant has placed.

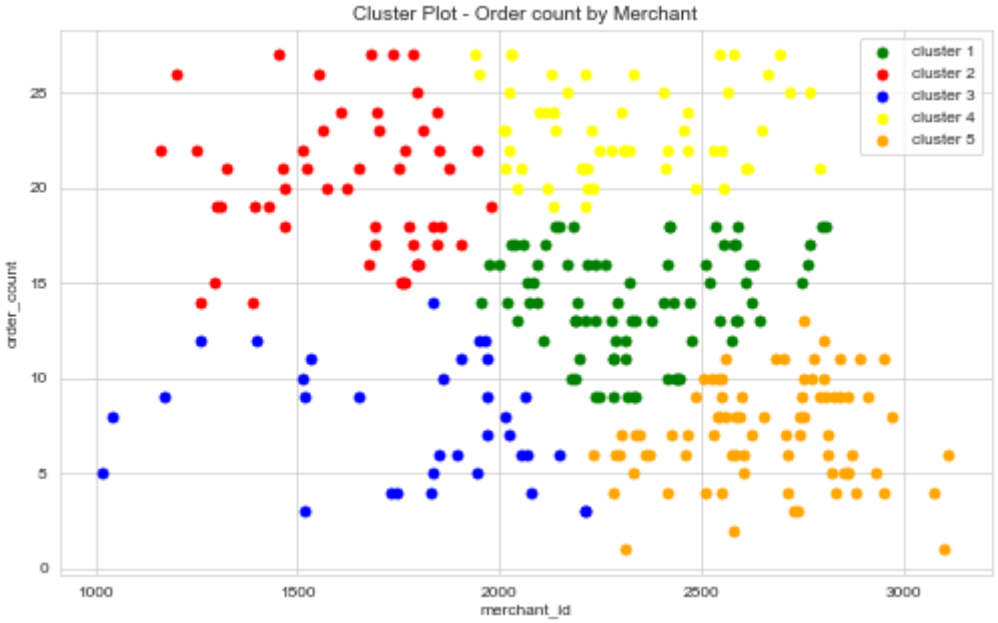
As a part of data pre-processing, the outliers among the dataset are filtered and the data is standardized to reduce the impact of highly varying values.

Then, the elbow method is used to determine the optimum number of clusters for the data.



The elbow method suggests that having 5 numbers clusters is optimum. Hence k means clustering is carried out using the value of 5. The new dataset with cluster values I shown below.

merchant_id	order_count	cluster
2313	1	4
3104	1	4
2580	2	4
2213	3	2
1519	3	2
2730	3	4
2738	3	4



The above cluster plot depicts the total count of orders placed by merchants. Each of the clusters is represented by different colours and the centroid of each cluster is indicated by a purple star.

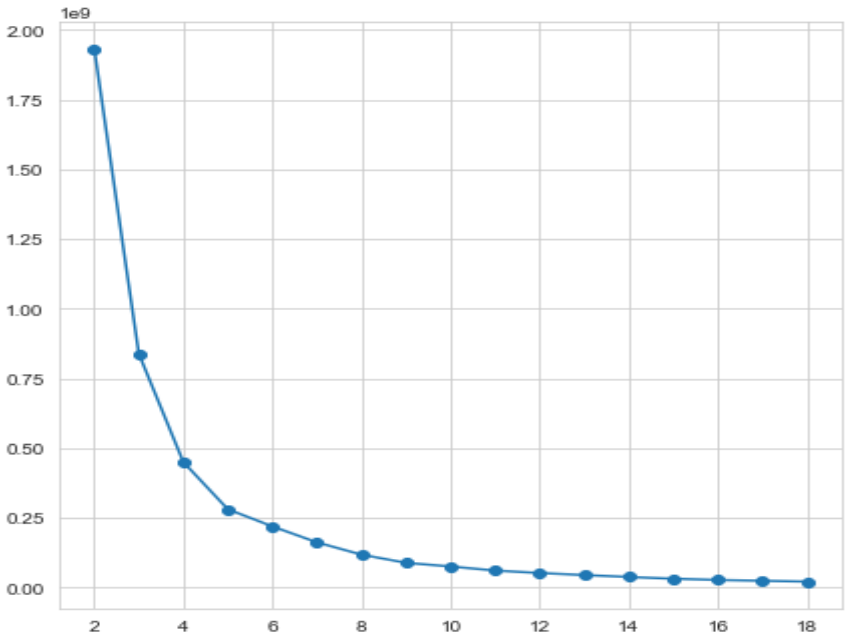
Merchants in cluster 4 have the highest number of orders followed by merchants in cluster 1. Cluster 2 has the least count of orders of all the clusters.

For model 2, clustering analysis is performed to study how merchants are clustered based on the average purchase price.

	merchant_id	mean_price
351	2335	0.000000
483	2864	0.000000
378	2445	13350.000000
421	2580	18250.000000
481	2861	21600.000000
295	2213	27916.666667
358	2362	30233.333333
374	2433	30520.238095

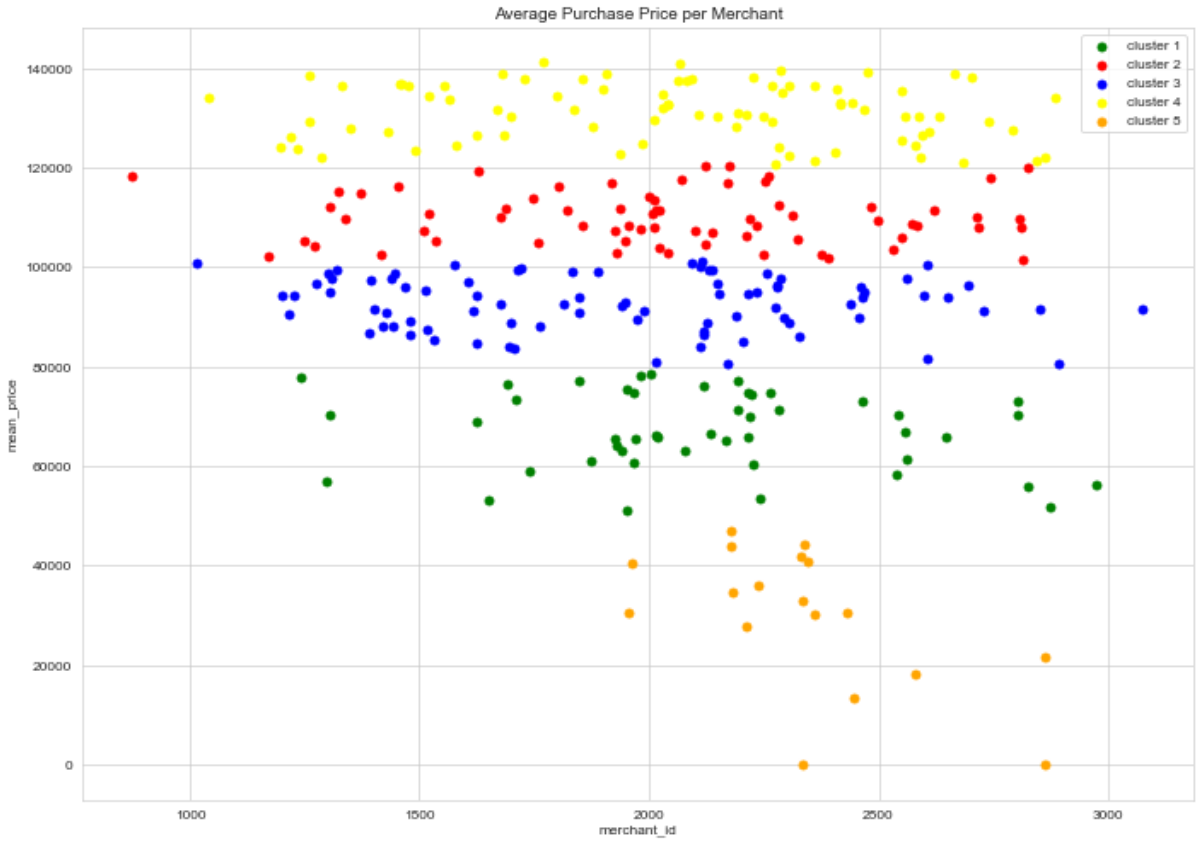
The data is prepared and we can observe that few merchants have not bought anything and few merchants have purchased a lot of items and have a very high-value price.

The data is pre-processed to remove outliers and scaled to remove the effect of high magnitude values. The elbow method suggests having 5 clusters in optimum.



Using the optimum number of clusters, the k means algorithm is used to study the data. The clustered data is shown below.

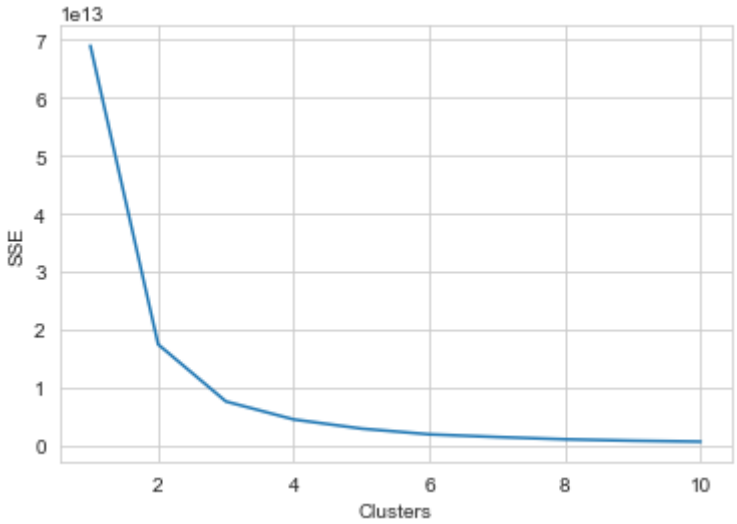
merchant_id	mean_price	cluster
2335	0.000000	2
2864	0.000000	2
2445	13350.000000	2
2580	18250.000000	2
2861	21600.000000	2
2213	27916.666667	2
2362	30233.333333	2



We can observe that the merchants who have the highest average purchase price are in cluster 4, the merchants who have the lowest average purchase price are in cluster 5. It's interesting to note that each cluster has a varying number of merchants as well.

For model 3, clustering analysis is performed to study the total purchase value by merchants segmented into quarters.

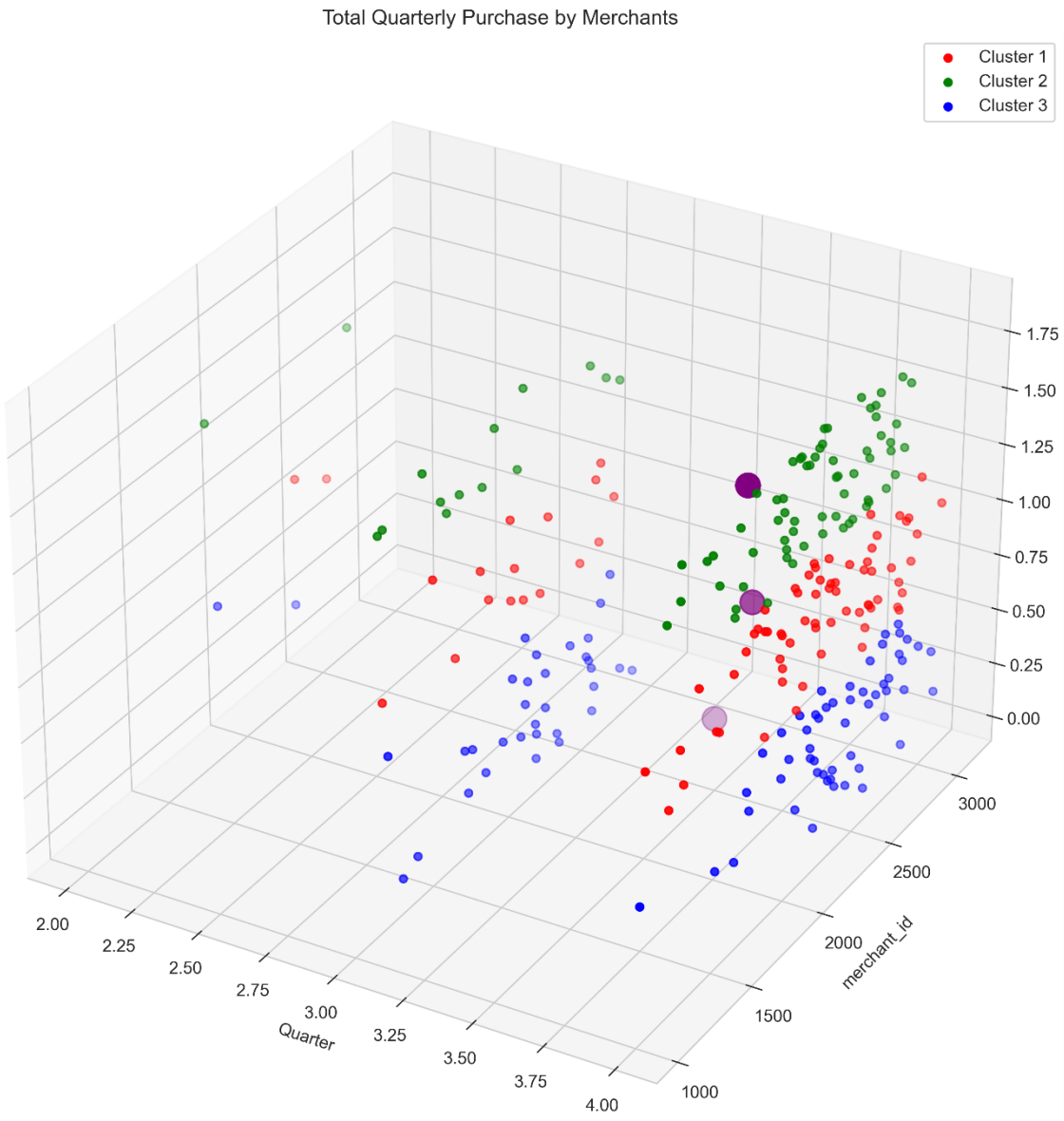
Once, the data is prepared, the elbow method suggests that the optimum number of clusters is 3.



By, using the k means algorithm, and using 3 as the optimum number of clusters, the model is developed.

The new data frame with clustered values is shown below.

Quarter	merchant_id	price	cluster
3	2346	19000.0	2
3	2574	26500.0	2
4	2580	36500.0	2
3	2187	41500.0	2
3	2861	43750.0	2
3	1306	46000.0	2



We can observe that there was no sale in the 1st quarter and the quarterly purchase price value increases over each quarter with more merchants buying more. The most purchase was done in the last quarter of the year.

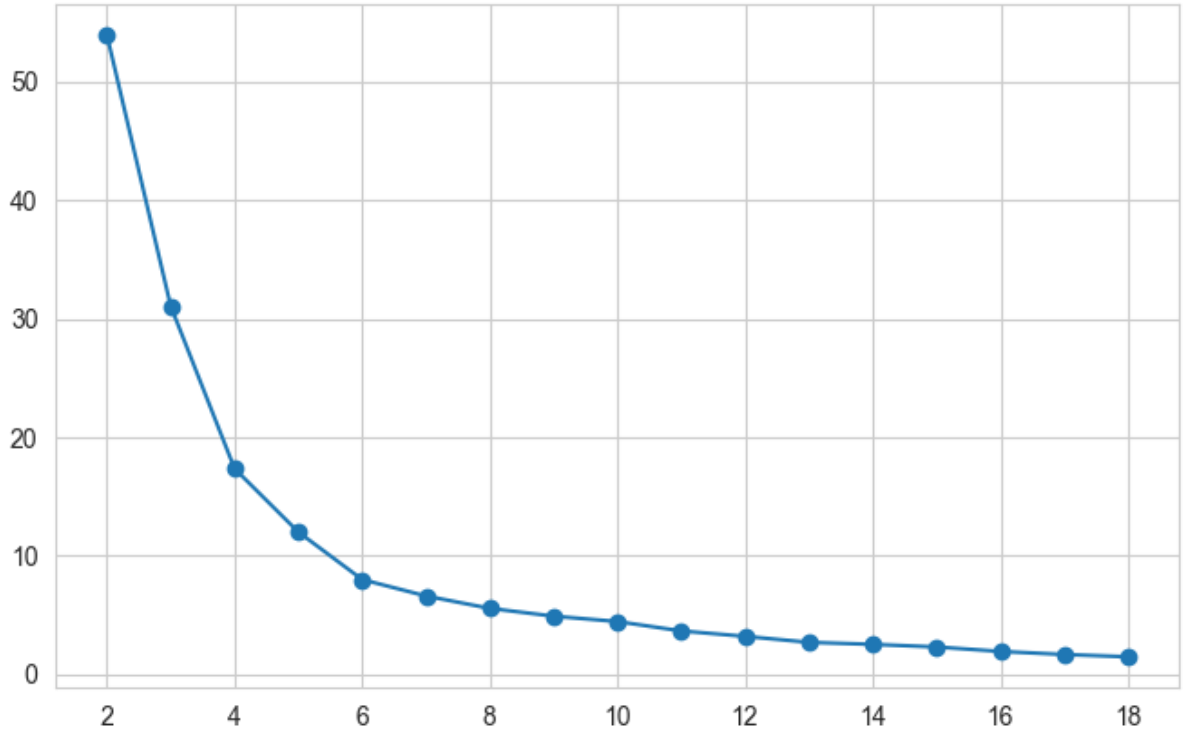
The most consistent pattern in terms of buying is that trend of purchase value increase over each quarter starting from 2nd quarter. Also, merchants who were top spenders in quarter 2 are seen consistently repeating their purchase pattern in the later quarters.

For model 4, clustering analysis is performed on the SKU movement speeds. The data is shown below.

	sku_id	Avg_diff
0	0	7.812500
1	1	4.750000
2	2	4.068966
3	3	3.051282
4	4	3.169014
..	...	...
709	23746	2.777778
710	23747	8.000000
711	23748	7.307692
712	23749	4.512195

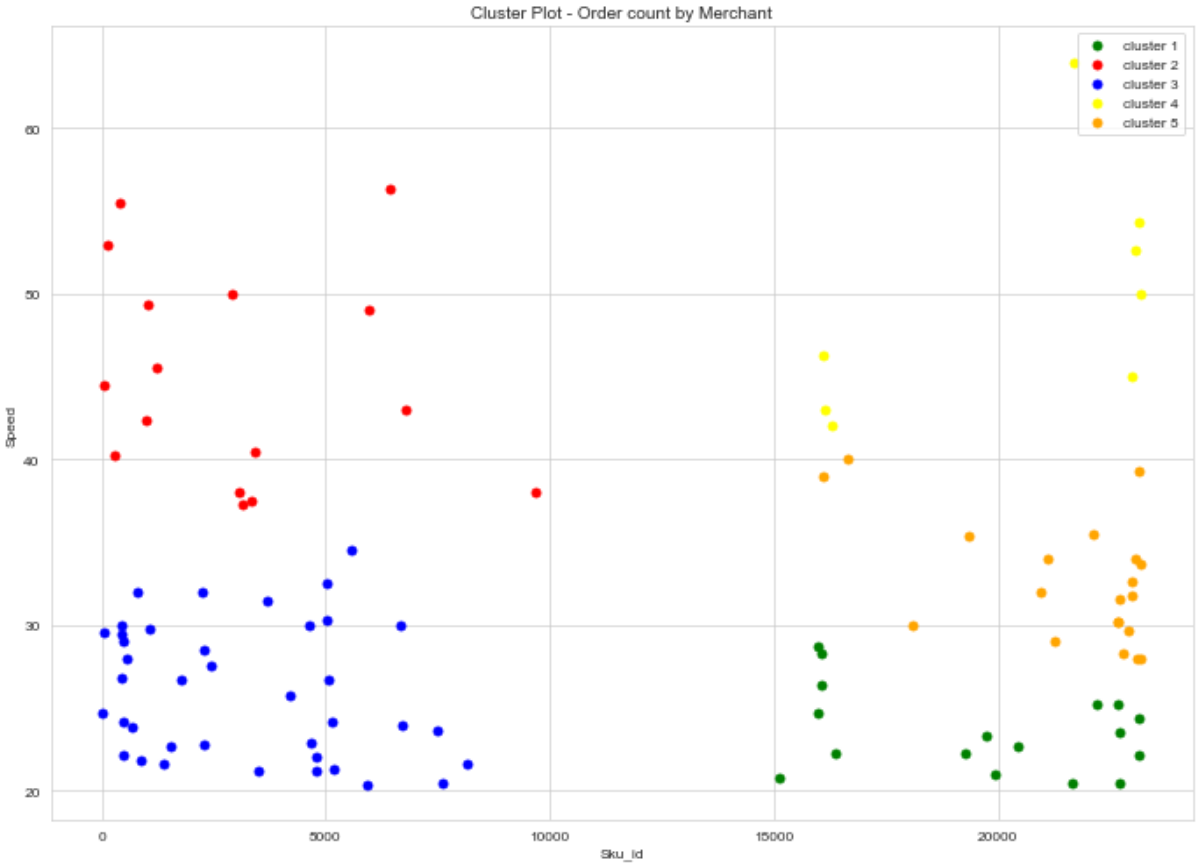
An SKU with a high value of avg_diff is classified as a slow-moving item and a fast-moving SKU has a lower value of avg_diff.

Scaling is performed as a part of data pre-processing and an elbow is used to determine the optimum number of clusters. The optimum number of clusters is calculated to be 5.



Then k means clustering analysis is performed and the cluster for each data point is calculated. The clustered data is shown below.

sku_id	Avg_diff	cluster
22923	29.666667	0
1088	29.800000	2
448	30.000000	2
18107	30.000000	0
6686	30.000000	2
4621	30.000000	2



From the above scatter cluster plot, we can see that SKUs within clusters 1 and 3 are the fast-moving items whereas SKUs within clusters 4 and 2 are the slow-moving items.

672	23060	52.666667
39	143	53.000000
686	23166	54.333333
81	404	55.500000
403	6420	56.333333
608	21716	64.000000

Some of the fats moving SKU's are 21716, 6420, 404 as shown above.

Q 5 - **Are there any associations among the product SKUs/categories that exist in the overall transaction data, and if so, what are some meaningful associations?**

We decided to identify association rules for sku after cleaning the dataset and reviewing the unique sku ids, category ids, and subcategory ids present in the dataset, which are 714, 29, and 83 respectively. We further examined the number of times each sku occurred inside the dataset. The list of sku ids and where they appear in the dataset are shown in the images below along with a bar graph.

	sku_id	count
0	2037	433
1	360	404
2	8	397
3	468	354
4	1558	291
5	14766	265
6	33	252
7	2740	230
8	1767	222
9	1204	222

Fig.1: sku count list

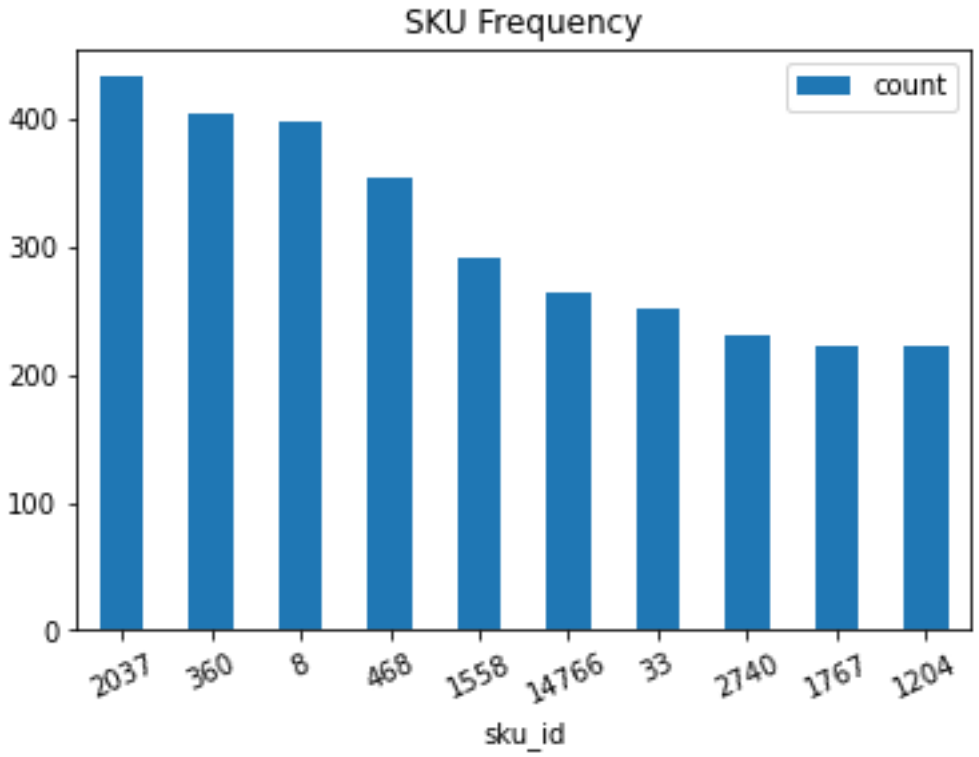


Fig.2: bar plot of sku count

After identifying the count of each sku in the dataset, we decided to examine the frequency of each sku when grouped into groups of two, three, and four according to the order ids we have and individually. Figures below display the item sets and their frequencies.

	itemsets	support
69	frozenset({2037})	16.504097
28	frozenset({360})	14.826375
36	frozenset({468})	13.811939
3	frozenset({8})	11.939134
60	frozenset({1558})	11.275849
111	frozenset({14766})	10.222396
77	frozenset({2740})	8.973859
57	frozenset({1204})	8.661725
112	frozenset({14767})	7.452204
22	frozenset({232})	7.101053

Fig.3: Individual sku frequency

	itemsets	support
350	frozenset({468, 2037})	5.852517
402	frozenset({2037, 1558})	5.579399
345	frozenset({468, 1558})	5.267265

Fin.4: Group of two sku frequency

	itemsets	support
650	frozenset({468, 2037, 1558})	3.316426
668	frozenset({1204, 2037, 1558})	2.184940
682	frozenset({2037, 1558, 14767})	1.989856

Fig.5: Group of three sku frequency

	itemsets	support
720	frozenset({468, 2037, 1558, 14767})	1.443621
708	frozenset({360, 14761, 2037, 415})	1.248537
709	frozenset({360, 468, 2037, 1558})	1.248537

Fig.6: Group of four sku frequency

It is evident from the above-mentioned observations of the frequency of the item sets that as an additional sku is added, the frequency of the item set decreases, guiding us to maintain low support while constructing association rules. The figures below display association rules that were produced with a minimum level of 0.01 support and sorted by high lift and high confidence, respectively.

	antecedents	consequents	support	confidence	lift
1955	frozenset({22251, 22252})	frozenset({360, 16373})	0.011705	0.500000	34.635135
1954	frozenset({360, 16373})	frozenset({22251, 22252})	0.011705	0.810811	34.635135
1952	frozenset({360, 22251})	frozenset({22252, 16373})	0.011705	0.666667	34.173333
1957	frozenset({22252, 16373})	frozenset({360, 22251})	0.011705	0.600000	34.173333
1828	frozenset({360, 16373})	frozenset({22252, 4})	0.010144	0.702703	33.352352

Fig.7: High Lift association rules

	antecedents	consequents	support	confidence	lift
1949	frozenset({360, 22251, 16373})	frozenset({22252})	0.011705	1.000000	25.376238
1824	frozenset({360, 4, 16373})	frozenset({22252})	0.010144	1.000000	25.376238
1349	frozenset({360, 16373})	frozenset({22252})	0.014046	0.972973	24.690393
1721	frozenset({22251, 1558})	frozenset({22252})	0.010535	0.964286	24.469943
1817	frozenset({22251, 16373})	frozenset({22252})	0.014826	0.950000	24.107426

Fig.8: High Confidence association rules

According to the above mentioned lists, lift decreases to an average level when high confidence association rules are taken into account as opposed to what was observed in the high lift association list. As a result, we made the decision to analyse the created association rules to determine how lift and confidence are sprayed. To do this, we limited the antecedents to a single item and looked at the association rules. The list of association rules with just one antecedent is shown in the figure below.

	antecedents	consequents	support	confidence	lift
1718	frozenset({16373})	frozenset({22252, 1558})	0.011705	0.483871	31.004032
1961	frozenset({16373})	frozenset({360, 22251, 22252})	0.011705	0.483871	30.247836
855	frozenset({16373})	frozenset({22251, 4})	0.010144	0.419355	29.855735
1835	frozenset({16373})	frozenset({360, 22252, 4})	0.010144	0.419355	29.048823
1347	frozenset({16373})	frozenset({360, 22251})	0.011705	0.483871	27.559140
1821	frozenset({16373})	frozenset({22251, 22252})	0.014826	0.612903	26.181183
1585	frozenset({22251})	frozenset({22252, 862})	0.010144	0.313253	25.898951
1959	frozenset({22251})	frozenset({360, 22252, 16373})	0.011705	0.361446	25.732932
1960	frozenset({22252})	frozenset({360, 22251, 16373})	0.011705	0.297030	25.376238
1833	frozenset({22252})	frozenset({360, 4, 16373})	0.010144	0.257426	25.376238

Fig.9: Association rules with only 1 antecedent

Last but not least, we chose to associate rules with greater than or equal to 25 and .50 lift and confidence, respectively, after witnessing supports of various item sets, high lift, high confidence, and confidence and lift with only one antecedent in the association rules. The top 10 association rules are shown in the image below, where it is obvious that even if they are sorted with a high lift, we are still receiving sprayed confidence levels, indicating that these rules might be valuable in the future.

	antecedents	consequents	support	confidence	lift
60	frozenset({22251, 22252})	frozenset({360, 16373})	0.011705	0.500000	34.635135
59	frozenset({360, 16373})	frozenset({22251, 22252})	0.011705	0.810811	34.635135
57	frozenset({360, 22251})	frozenset({22252, 16373})	0.011705	0.666667	34.173333
62	frozenset({22252, 16373})	frozenset({360, 22251})	0.011705	0.600000	34.173333
48	frozenset({360, 16373})	frozenset({22252, 4})	0.010144	0.702703	33.352352
51	frozenset({4, 16373})	frozenset({360, 22252})	0.010144	0.764706	33.219342
61	frozenset({22251, 16373})	frozenset({360, 22252})	0.011705	0.750000	32.580508
58	frozenset({360, 22252})	frozenset({22251, 16373})	0.011705	0.508475	32.580508
30	frozenset({22252, 1558})	frozenset({16373})	0.011705	0.750000	31.004032
54	frozenset({360, 22251, 22252})	frozenset({16373})	0.011705	0.731707	30.247836

Fig.10: Final list of association rules

Part 3 - Phase 2 – Implementing a recommender system

Following the first phase's evaluation of 5 questions, we decided to construct a recommender system employing the KNN algorithm and fast/slow-moving items. Since EDA on fast-moving/slow-moving items was previously done in question 3, all that is left to do is map our findings to the original dataset. The figure below depicts the steps we took.

```
AwanTunai_data["sku_movement"] = 0
```

```
AwanTunai_data.sku_movement.update(AwanTunai_data.sku_id.map(AwanTunai_sku_movement.set_index('sku_id').Avg_diff))
```

```
AwanTunai_data = AwanTunai_data[AwanTunai_data['sku_movement'] > 0]
```

```
AwanTunai_data['sku_movement'] = AwanTunai_data['sku_movement'].round(decimals = 2)
```

```
AwanTunai_data.head()
```

	id	order_id	merchant_id	sku_id	date	top_cat_id	sub_cat_id	price	sku_movement
0	654713	82186	1754	981	2021-06-29	7.0	44.0	877000.0	42.33
1	654712	82186	1754	23060	2021-06-29	7.0	44.0	877000.0	52.67
2	654711	82186	1754	1548	2021-06-29	7.0	44.0	877000.0	22.67
4	654687	82186	1754	343	2021-06-29	7.0	44.0	312000.0	17.80
5	1058235	129785	1760	1548	2021-10-04	7.0	44.0	3750.0	22.67

Fig.1: Mapping SKU movement with the original dataset

There were 134 sku_ids whose average day difference was 0. Technically, objects that move quickly have a difference of 0 days, but when we looked at how frequently those SKUs appeared, we discovered that, except for a few SKUs, the majority of SKUs only appeared once on a given date, so there was not enough data to prove their movements. Hence, we deleted those SKUs from our data and so in our case, the fastest moving items have an average difference of 1 day. Next, we checked the number of SKUs in various day bins after mapping this data, as indicated in the picture below.

```
bins = [1, 15, 30, 45, (AwanTunai_sku_movement["Avg_diff"]).max()]
df = pd.cut(AwanTunai_sku_movement["Avg_diff"], bins)
df.value_counts()
```

```
(1.0, 15.0]     409
(15.0, 30.0]    126
(30.0, 45.0]     32
(45.0, 64.0]     12
Name: Avg_diff, dtype: int64
```

Fig.2: SKUs in different movement bins

We would like to mention again that the difference in the days means how many days it takes to sell the next item for the same SKU, which was calculated by taking the difference between the sale dates and then averaging them out.

We can see that 409 items fall under the 1 to 15 days bin, 126 items fall under the 15 to 30 days bin, 32 items fall in the 30 to 45 days bin and lastly, 12 items have a day difference between 45 to 64 days. This gives another important insight into the dataset that most of the items have smaller day differences between their sale dates hence they might be fast-moving items, hence our dataset is skewed. This pattern can also be seen in the boxplot for Q.3.

For using the KNN algorithm, first, we combined our SKU movement data with the original dataset after which we needed to convert this data into a matrix format, hence additional data preparation was necessary before continuing with the model generation. The matrix we constructed and the indexing we offer to each SKU is shown in the figure below.

```
df_movement_day_diff_features = df_merchant_sku_N_movement.pivot(
    index = 'sku_id',
    columns = 'merchant_id',
    values = 'sku_movement'
).fillna(0)
```

```
df_movement_day_diff_features
```

merchant_id	875	1016	1041	1103	1156	1160	1161	1170	1171	1172	...	2885	2891	2913	2932	2953	2954	2973	3075	3104	3114
sku_id																					
0	0.00	0.0	0.0	0.0	0.00	0.00	0.0	7.81	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.00	0.0	0.0	0.0	4.75	0.00	0.0	0.00	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.00	0.0	0.0	0.0	4.07	4.07	0.0	0.00	0.0	4.07	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.00	0.0	0.0	0.0	3.05	3.05	0.0	0.00	0.0	3.05	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	3.17	0.0	0.0	0.0	0.00	0.00	0.0	0.00	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23746	0.00	0.0	0.0	0.0	0.00	0.00	0.0	2.78	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23747	0.00	0.0	0.0	0.0	0.00	0.00	0.0	8.00	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23748	0.00	0.0	0.0	0.0	0.00	0.00	0.0	0.00	0.0	7.31	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23749	0.00	0.0	0.0	0.0	0.00	0.00	0.0	4.51	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23756	0.00	0.0	0.0	0.0	0.00	0.00	0.0	0.00	0.0	0.00	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

580 rows × 495 columns

Fig.3: Data matrix for the algorithm

```
sku_list = pd.DataFrame(df_merchant_sku_N_movement.sku_id.unique())
sku_list.columns = ['sku_id']
sku_list = sku_list.sort_values(by='sku_id', ascending=True).reset_index()
sku_list.insert(loc=0, column='row_num', value=np.arange(len(sku_list)))
sku_list = sku_list[['row_num', 'sku_id']]
sku_list.head()
```

	row_num	sku_id
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4

```
sku_movement_to_idx = sku_list.to_dict()
sku_movement_to_idx = sku_movement_to_idx['sku_id']
```

```
sku_movement_to_idx
```

```
{0: 0,
 1: 1,
 2: 2,
 3: 3,
 4: 4,
 5: 8,
 6: 9,
 7: 15,
 8: 18,
 9: 21,
 10: 22,
 11: 25,
 12: 27,
 13: 33,
 14: 34,
 15: 38,
 16: 40,
 17: 45,
 18: 56,
```

Fig.4: SKU indexing

After preparing the data for the algorithm, we created functions for two distinct proposals: one function finds the index of the chosen SKU and examines its movement, while the other finds the data in the model and also suggests SKUs that are connected to the chosen SKU. Both functions are depicted in the diagram below.

```
def fuzzy_matching(mapper, selected_sku, verbose=True):
    match_tuple = []
    # get match
    for idx, sku_id in mapper.items():
        ratio = fuzz.ratio(str(sku_id), str(selected_sku))
        if ratio == 100:
            match_tuple.append((sku_id, idx, ratio))
    # sort
    match_tuple = sorted(match_tuple, key = lambda x: x[2])[::-1]
    if not match_tuple:
        print('Oops! No match is found')
        return
    if verbose:
        sku_movement = AwanTunai_sku_movement[AwanTunai_sku_movement["sku_id"] == 20315]["Avg_diff"]\
        .values[0].round(decimals = 2)
        is_fast = "Fast Moving"
        if sku_movement > 10:
            is_fast = "Slow Moving"
        print('Found possible matches in our database: {0} [{1}]'.format([x[0] for x in match_tuple], is_fast))
        print('{0} moving with a speed of {1} days on an average.\n'.format(selected_sku, sku_movement))
    return match_tuple[0][1]
```

Fig.5: Matching SKU functions

```
def make_recommendation(model_knn, data, mapper, selected_sku, n_recommendations):
    # fit
    model_knn.fit(data)
    # get input movie index
    print('You have input SKU:', selected_sku)
    idx = fuzzy_matching(mapper, selected_sku, verbose = True)
    if(idx is not None):
        # inference
        print('Recommendation system start to make inference')
        print('......\n')
        distances, indices = model_knn.kneighbors(data[idx], n_neighbors = n_recommendations)
        # get list of raw idx of recommendations
        raw_recommends = \
            sorted(list(zip(indices.squeeze().tolist(), distances.squeeze().tolist())), key=lambda x: x[1])[:0:-1]
        # print recommendations
        print('Recommendations for {}:'.format(selected_sku))
        for i, (idx, dist) in enumerate(raw_recommends):
            try:
                print('{0}: {1}, with distance of {2}'.format(i+1, mapper[idx], dist))
            except:
                print("{0}: Exception thrown. {1} SKU does not exist.".format(i+1, idx))
```

Fing.6: Recommendation functions

After creating the functions, we build a KNN model and used random SKUs to test the model's predictions. The model generation code and suggestions for the chosen SKU are shown in the image below.

```
# define model
model_knn = NearestNeighbors(metric='cosine',\
                             algorithm='brute',\
                             n_neighbors = 3,\
                             n_jobs = -1)
```

```
selected_sku = '20315'

make_recommendation(
    model_knn = model_knn,
    selected_sku = selected_sku,
    data = matrix_movement_day_diff_features,
    mapper = sku_movement_to_idx,
    n_recommendations = 11)
```

```
You have input SKU: 20315
Found possible matches in our database: [20315] [Fast Moving]
20315 moving with a speed of 6.07 days on an average

Recommendation system start to make inference
......

Recommendations for 20315:
1: 306, with distance of 0.7962152135151943
2: 20889, with distance of 0.7817821097640076
3: 16280, with distance of 0.7776252050016697
4: 7209, with distance of 0.7609542781331213
5: 19477, with distance of 0.758253110792386
6: 20313, with distance of 0.746453723581445
7: 23185, with distance of 0.746453723581445
8: 19281, with distance of 0.7327387580875757
9: 16640, with distance of 0.6913933000758162
10: 1865, with distance of 0.6913933000758162
```

Fig.7: Model and Recommended SKUs

As seen in the above-mentioned image, we created a nearest neighbour's model by taking into account the three closest neighbours and utilizing the cosine metric to calculate distance. The idea behind employing this matrix is to simply enhance the model's performance and accuracy according to our expectations. We can see from the image above that we have fed sku_id 20315 into the model and the model has recommended 10 similar items. Hence, whenever any sku_id is fed into the model, the model will give recommendations of the similar items that a merchant can buy based on the similar speed with which the items sell. There are a few advantages of this model-

1. We can use the names of the items instead of sku_ids to search for similar items, for this you can change the similarity ratio from 100 to 80 and you can get an 80% match from the product name.
2. We can feed any sku_id into the model and we would know whether the item is slow-moving or fast-moving. For e.g., in the above image, you can see the model has identified sku_is 20315 as the fast-moving item.
3. Since it is vice to stock variety, the merchants would get recommendations for the fast-moving as well as slow-moving items, the advantage of which has been discussed in the conclusion.

Conclusion

When we decided to build a model, we had shortlisted 2 options, the Net basket recommender system(NBR) and KNN, our professor was kind enough to share a lot of research paper on this matter and when we read it, we came to knew that KNN performs better than NBR and gives many accurate results. So, we decided to go with the KNN model, we again researched similar industries where such models are used and we came across a very interesting finding that most of the online sites use KNN for making movie recommendations. KNN works on **Collaborative filtering** systems which use the actions of users to recommend other movies. In general, they can either be user-based or item-based. The user-based approach is often harder to scale because of the dynamic nature of users, whereas items usually don't change much and often can be computed offline and served without constantly re-training. Our aim was also similar, i.e., to use an item-based approach for recommendations, hence KNN was a perfect go-to model and also a very good baseline for recommender system development.

Our idea was to implement our fast-moving and slow-moving dataset to make recommendations, in that scenario we would be able to recommend the merchants similar items that he is purchasing based on similar movements. In other words, we will be able to make recommendations that if a merchant is buying 10 fast-moving items, then which other similar items he can buy.

Some people can argue that our model will suggest only fast-moving items to the merchants who are already buying fast-moving items and vice versa but in a real-world scenario, a retailer will always buy mixed items. Our idea is based on the fact that even the slow-moving items are essential and, in some cases, dependent or related to the sale of another item. Hence even if the merchant is buying slow-moving items and the recommender system suggesting him similar items it could be a vice for him to stock such items in less quantity and stock fast-moving items in more quantity. There is a popular fact in business that, high-frequency items tend to be relatively competitive items with little profit for the merchant. On the other hand, the lower frequency items have larger profit margins. and also, as we discussed earlier, in the case of small retailers, every item can be important, in fact carrying items that no one in the neighbourhood has, strengthens the image of the shop and its customer base

Coming back to our inspiration for this model, a movie recommender system shows the list of movies which is not only popular during the same era but also share very similar genres and topics. Sometimes they are also recommended based on similar ratings. However, Recommender systems in the case of movies have 2 shortcomings –

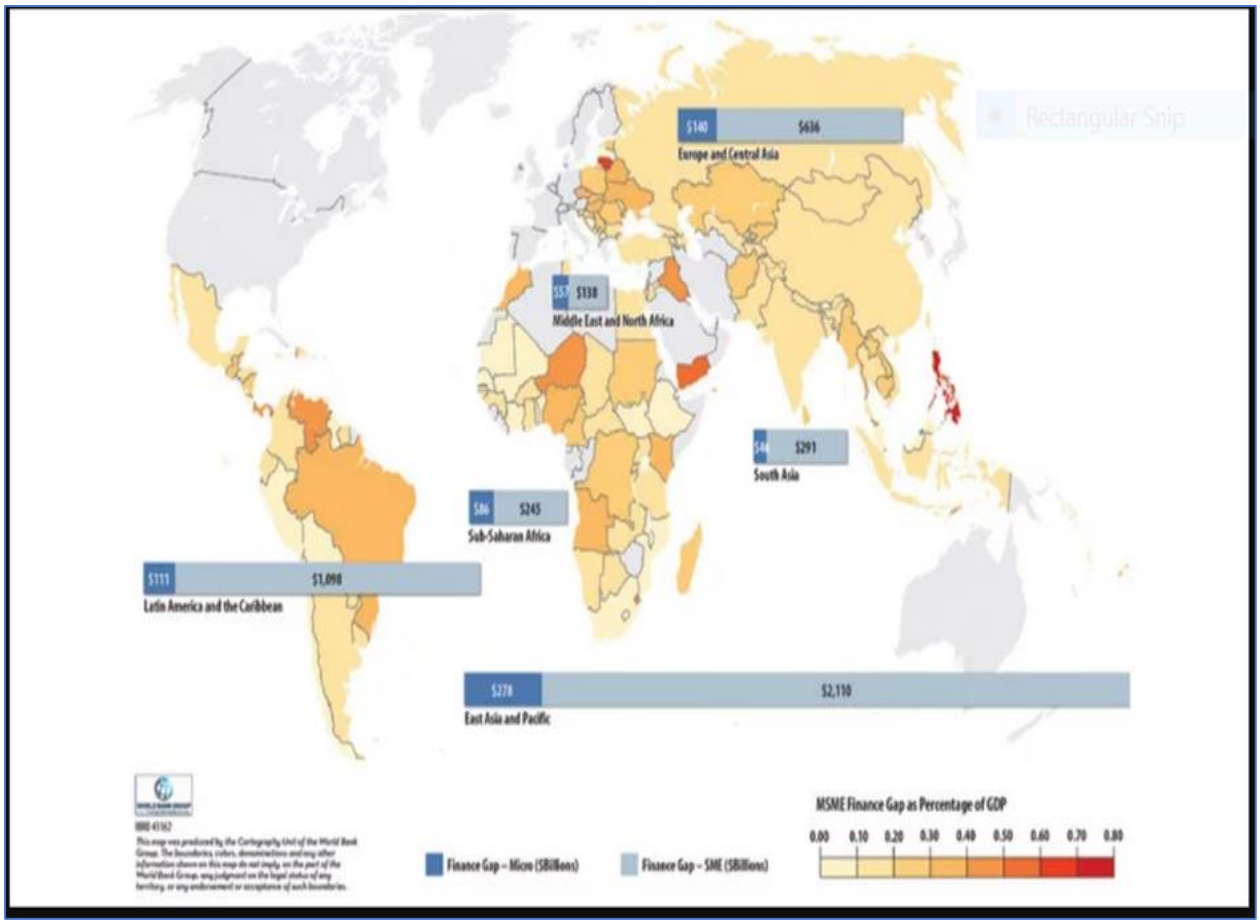
1. Popularity bias: recommender is prone to recommend popular items.

2. Item cold-start problem: recommender fails to recommend new or less-known items because items have either none or very few interactions.

But in our case, this shortcoming will not have any effect since our system will recommend the items based on their speed and not frequency.

However, as a future project, we would like to recommend that using alternating least square matrix factorization in collaborative filtering may lead to even stronger models.

We also went deeper to understand the Indonesian market so that we could understand what our models could mean and achieve.

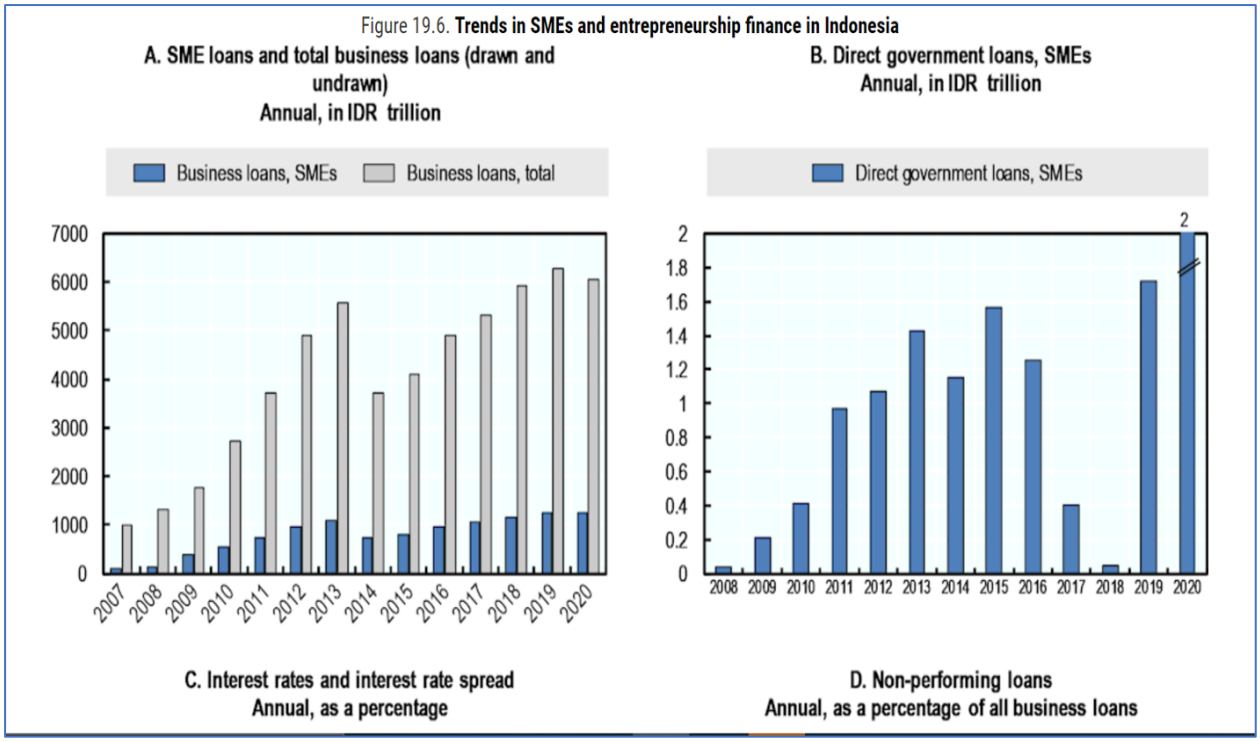


As we can see from the above map published by the world bank, there is a 20 to 30 % gap in MSME finance as a percentage of GDP in Southeast Asia. There are some other facts published by the **Ministry of Cooperatives and SMEs of the Republic of Indonesia**:

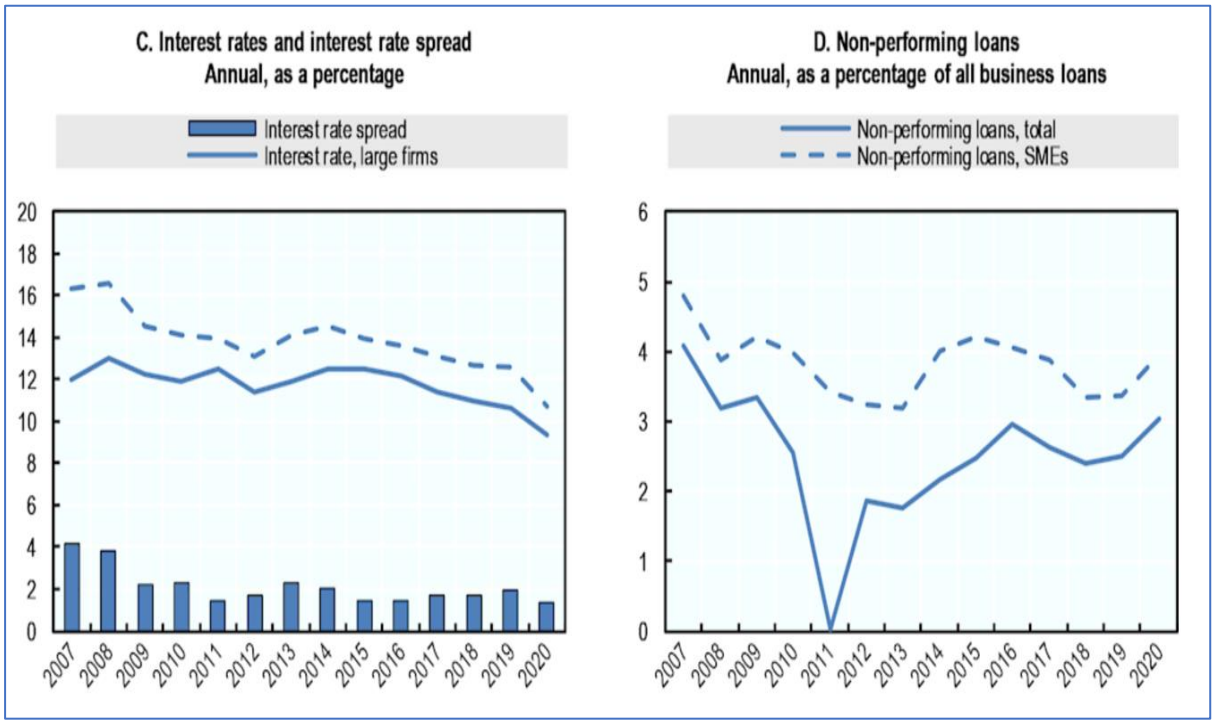
- There were 64 194 056 SMEs in 2019, which made up 99.99% of the total business population and employed 96.9% of the total workforce.
- Outstanding loans to all businesses declined by 3.2% year on year (y-o-y) in 2020 (IDR 6,069.39 trillion), with only 20.43% of this amount (IDR 1 240.23 trillion) allocated to SMEs. Despite the decrease in 2020, outstanding loans in the past ten years (2011-20) still experienced growth with an average yearly growth rate of 13.38%.
- In the period from 2011-2020, interest rates on loans declined for all businesses, by 3.8% for SMEs (from 14.53% to 10.69%) and 2.92% for large companies (from 12.28% to 9.36%). Although interest rates are declining in Indonesia, they are still very high compared to the average in other countries.

All these facts show the importance of FinTech for the Indonesian economy as they have the potential to provide low-cost and easier financing to these SMEs.

If we also see some of the graphs below taken from the site of the Ministry of cooperatives and SMEs, we can see that SME loans are very low as compared to the larger businesses and direct govt lending is also hugely fluctuating.



From the below graph we can see that the interest rates for the SMEs are always higher and even loan failures are quite high in the SME sector. All these facts show that this sector needs a lot of help, especially considering the amount of employment it generates in the country.



Awan Tunai can help these SMEs not only in securing low-cost loans but also by improving their profitability and turnovers, and we hope that the models made by the students during this project can be of some value to the company.

References

- *Datacamp*. (n.d.). Retrieved 28 June 2022, from https://app.datacamp.com/workspace/w/2999da42-509f-4d2f-bf9c-c3fe1ed47abd/edit
- *How to create a drop-down list with the symbol in Excel?* (n.d.). ExtendOffice. Retrieved 28 June 2022, from https://www.extendoffice.com/documents/excel/3594-excel-drop-down-list-with-symbol.html
- *How to find the mode for an R data frame column?* (n.d.). Retrieved 28 June 2022, from https://www.tutorialspoint.com/how-to-find-mode-for-an-r-data-frame-column
- Kabacoff, R. (n.d.). *Data visualization with r*. Retrieved 28 June 2022, from https://rkabacoff.github.io/datavis/
- Marsja, E. (2020, August 27). *How to extract the year from the date in r with examples*. Erik Marija. https://www.marsja.se/how-to-extract-year-from-date-in-r-with-examples/
- Liao K., Nov 17, 2018, Towards data science, Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering, https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1
- OECD iLibrary, Financing SMEs and Entrepreneurs 2022: An OECD Scoreboard, https://www.oecd-ilibrary.org/sites/13753156-en/index.html?itemId=/content/component/13753156-en

Appendix

Q.1

#1 Import libraries.

```
library(FSA)
library(FSAdata)
library(plyr)
library(dplyr)
library(tidyr)
library(tidyverse)
library(magrittr)
library(ggplot2)
library(corrplot)
library(scatterPlotMatrix)
library(Amelia)
library(readxl)
library(car)
library(leaps)
library(gginference)
library(UpSetR)
library(naniar)
library(lubridate)
library(Hmisc)
```

#2 Import the dataset and describe it.

```
awantunai<-read.csv("20220413_Northeastern_AwanTunai_Capstone_Data.csv")
dim(awantunai)
str(awantunai)
summary(awantunai)
```

#3 Find any duplicate values and discard them.

```
awantunai_duplicate <- awantunai[!duplicated(awantunai), ]

#4 Find the NA values and discard them.

missmap(awantunai, main = "Missing values in dataset")
sapply(awantunai,function(x) sum(is.na(x)))
gg_miss_upset(awantunai)
awantunai_clean <- awantunai[complete.cases(awantunai), ]

#5 Break the date column into the day, month, and year.

awantunai_updated <- awantunai_clean %>%
  mutate(date = as.Date(date, format = "%Y-%m-%d"),
         day = as.integer(strftime(date, format = "%d")),
         month = as.integer(strftime(date, format = "%m")),
         year = as.integer(strftime(date, format = "%Y")))
head(awantunai_updated,5)
summary(awantunai_updated)

#6 Now we will sort the data in descending order from
the price column.

awantunai_sorted <- awantunai_updated[order(-awantunai_updated$price),]

#7 Here we will create the mode function to calculate the mode for the column month.

mode<-function(x){which.max(tabulate(x))}
mode(awantunai_sorted$month)

#8 Visualization

hist.data.frame(awantunai_sorted)
hist(awantunai_sorted$day)
```

#Q2

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


#Reading the dataset
AwanTunai = pd.read_csv('20220413_Northeastern_AwanTunai_Capstone_Data.csv')
AwanTunai.head()
AwanTunai.tail()
AwanTunai.shape


AwanTunai.info()


AwanTunai.columns


AwanTunai.describe()


AwanTunai.nunique()


AwanTunai.isnull().sum()


#Removing these null values
AwanTunai.dropna(inplace=True)


#Grouping by 'merchant_id'
gk = AwanTunai.groupby('merchant_id').agg({'price': ['mean', 'min', 'max']})
print("Mean, min, and max values of Purchase Amount grouped by Merchant id")
print(gk)
```

```
#Creating data frame
df = pd.DataFrame(AwanTunai)

#mean price spent by the top 5 frequent merchants
mean_values = df.groupby('merchant_id', as_index=False)['price'].mean()
mean_1607 = mean_values.loc[mean_values['merchant_id'] == 1607]
print(mean_1607)

mean_values.loc[mean_values['merchant_id'] == 1477]

mean_values.loc[mean_values['merchant_id'] == 1196]

df['order_id'].value_counts()[:5]

df['order_id'].value_counts()[:5].hist(bins=30)

#Correaltion b/w attributes
AwanTunai.corr()

#Correlation plot
plt.figure(figsize=(15,17))
sns.heatmap(AwanTunai.corr(),annot=True, cmap='coolwarm')

#Creating new column
df[['date']]=df[['date']].apply(pd.to_datetime)
df['month'] = df['date'].dt.month
df.head()

merchant_sku = df.groupby(['merchant_id','sku_id'], as_index = True).agg({'count'}).reset_in
dex()
```

```
merchant_sku


frequency_df = df.groupby(
   by=['merchant_id'], as_index=False)['date'].count()
frequency_df.columns = ['merchant_id', 'Frequency']
frequency_df.head()


most_frq_buyers = merchant_sku.merchant_id.value_counts().sort_values(ascending = False)


#Top 10 most frequent buyers
most_frq_buyers.head(10).plot(kind = 'bar', figsize = (10,10), title = "Merchants buying the
most products", xlabel = "MerchantID", ylabel = "Products bought", color = "red")


#Least 10 buyers
most_frq_buyers.tail(10).plot(kind = 'bar', figsize = (10,10), title = 'Merchants buying the leas
t products', xlabel = "MerchantID", ylabel = "Products")


#Merchants repeating purchases
merchant_repeating = merchant_sku.id.value_counts()
merchant_repeating


merchant_sku = df.groupby(['merchant_id', 'sku_id', 'month'], as_index= True).agg({'count'})
merchant_sku


merchant_sku = df.groupby(['merchant_id', 'sku_id'], as_index= True).agg({'count'})
merchant_sku


df = df.sort_values('date', ascending = True)
df = df.dropna(how='any')
df['year'] = df['date'].dt.year
```

```
df.head()
```

Comparing merchant id's

```
df.groupby(['merchant_id', 'sku_id'], as_index= True).count()
```

```
df875 = df[df['merchant_id']==875]
df875 = pd.pivot_table(df875, index = ['sku_id','month','order_id'], aggfunc = ['count'])
df875
```

```
df[df['merchant_id']==875].sku_id.value_counts()[:20].plot(kind='bar', color = 'Red')
```

```
df1523 = df[df['merchant_id']==1523]
df1523 = pd.pivot_table(df1523, index = ['sku_id','month','order_id'], aggfunc = ['count'])
df1523
```

```
df[df['merchant_id']==1523].sku_id.value_counts()[:20].plot(kind='bar')
```

```
df[df['merchant_id']==3114]
df3114 = df[df['merchant_id']==3114]
df3114 = pd.pivot_table(df3114, index = ['sku_id','month','order_id'], aggfunc = ['count'])
df3114
```

```
df[df['merchant_id']==3114].sku_id.value_counts().head(20).plot(kind='bar', color = 'green')
```

****Comparing sku_id****

```
df['sku_id'].value_counts()[:5]
df[df['sku_id']==2037].month.value_counts().head(20).plot(kind='bar')
```

```
df[df['sku_id']==360].month.value_counts().head(20).plot(kind='bar', color = 'yellow')
```

Q.3

```
# load the libraries
library(dplyr)
library(lubridate)
library(broom)
library(tidyr)
# upload the file
Awan <- read.csv(file.choose())
head(Awan, n=5)
str(Awan)
summary(Awan)
length(unique(Awan$sku_id))
length(unique(Awan$merchant_id))

is.na(Awan)
colSums(is.na(Awan))
which(colSums(is.na(Awan))>0)
names(which(colSums(is.na(Awan))>0))

# remove NAs
new_Awan <- Awan[complete.cases(Awan), ]        # Apply complete.cases function
head(new_Awan, 5)
summary(new_Awan)
length(unique(new_Awan$sku_id))
```

```
length(unique(new_Awan$merchant_id))


# group by sku_id
by_sku_id <- new_Awan_1 %>%
  group_by(sku_id) %>%
  summarize(total=n())
head(by_sku_id,5)
tail(by_sku_id,5)
max(by_sku_id$total, na.rm=TRUE)


# sorting lowest and highest frequency items
by_sku_id[rev(order(by_sku_id$total)),]
by_sku_id[order(by_sku_id$total),]


lowSales_sku_id <- by_sku_id %>%
  filter(total > 1)
head(lowSales_sku_id)
lowSales_sku_id[order(lowSales_sku_id$total),]


# group by date and sku_ids
by_date_id <- new_Awan_1 %>%
  group_by(sku_id,date)%>%
  summarise(total=n())
head(by_date_id, 5)
tail(by_date_id,5)
```

```
by_date_id %>%
  arrange(desc(total))


by_date_id %>%
  arrange(total)
# taking the difference in dates
by_date_id$diff <- unlist(tapply(as.Date(test$date), INDEX = test$sku_id,
                                 FUN = function(x) c(0, `units<-`(diff(x), "days"))))
head(by_date_id)
#taking the average of the dates per sku_id (total difference/number of times)
by_date_id_group <- by_date_id %>%
  group_by(sku_id) %>%
  summarise(Avg_diff = mean(diff))
      diff_days = as.numeric(diff, units = 'days')
head(by_date_id_group)
by_date_id_group


#checking the maximum value
 max(by_date_id_group$Avg_diff)
which.max(by_date_id_group$Avg_diff)
min(by_date_id_group$Avg_diff)


# Slowest moving item = highest average difference
by_date_id_group %>%
  arrange(desc(Avg_diff))
```

```
by_date_id_group %>%
  arrange(Avg_diff)
#fastest moving sku_ids with avg day diff. of >= 1
fast_sku_id <- by_date_id_group %>%
  filter(Avg_diff >= 1)

fast_sku_id %>%
arrange(Avg_diff)
#converting the data into csv file for visualization in tableau
write.csv(by_date_id_group,"D:\\ALY6080\\Module 4\\sku_movement.csv", row.names =
FALSE)
```

Q 4.

```
#ALY 6080 - Varun Clustering
# Packages Section
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#%%
# Reading Input file
my_data = pd.read_csv("D:/MS DATA ANALYTICS/Quarter 3/ALY
6080/20220413_Northeastern_AwanTunai_Capstone_Data (1).csv")
print(my_data)
#%%
# Dataset summary
my_data.info()
# Data Cleaning - Removing NA values
my_data = my_data.dropna()
print(my_data)
#%%
# EDA of the dataset
# Correlation Plot
corrs = my_data.corr()
sns.heatmap(corrs, annot=True)
#%%
# model 1 - number of orders placed by each merchant

model_1 = my_data.groupby('merchant_id', as_index=False).agg({"order_id":
"count"})
```

```
model_1.rename(columns={'order_id': 'order_count'}, inplace=True)
print(model_1)
#%%
model_1 = model_1.sort_values('order_count')
model_1 = model_1.head(300)
plt.scatter(model_1['merchant_id'],model_1['order_count'])
#%%
# Standardizing the values
#before standardizing
fig, (ax1,ax2) = plt.subplots(ncols=2,figsize=(6, 5))

ax1.set_title('Before Scaling')
ax2.set_title('After Scaling')
sns.kdeplot(model_1['merchant_id'], ax=ax1)
sns.kdeplot(model_1['order_count'], ax=ax2)
#%%
# Scaling - Transformation

col_names = model_1.columns
features = model_1[col_names]

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
scaled = pd.DataFrame(features, columns = col_names)
scaled.head()
#%%
#after transforming
fig, (ax1,ax2) = plt.subplots(ncols=2,figsize=(6, 5))
ax1.set_title('After Scaling')
sns.kdeplot(scaled['merchant_id'], ax=ax1)
sns.kdeplot(scaled['order_count'], ax=ax2)
#%%
# Clustering with K means
from sklearn.cluster import KMeans

# Using Elbow method to test for best clusters
clusters_range=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]
inertias=[]

for c in clusters_range:
    kmeans=KMeans(n_clusters=c, random_state=0).fit(scaled)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(16, 12), dpi=360)
plt.plot(clusters_range,inertias, marker='o')
#%%
# k means clustering
km = KMeans(n_clusters=5)
y_predicted = km.fit_predict(scaled[['merchant_id','order_count']])
y_predicted
#%%
model_1['cluster'] = y_predicted
model_1.head(10)
#%%
import matplotlib

scaled1 = model_1[model_1.cluster==0]
scaled2 = model_1[model_1.cluster==1]
scaled3 = model_1[model_1.cluster==2]
```

```
scaled4 = model_1[model_1.cluster==3]
scaled5 = model_1[model_1.cluster==4]
#scaled6 = scaled[scaled.cluster==5]
# scatter plot
plt.figure(figsize=(10, 6), dpi=60)
plt.scatter(scaled1.merchant_id,scaled1['order_count'],color='green',label=
'cluster 1')
plt.scatter(scaled2.merchant_id,scaled2['order_count'],color='red',label='c
luster 2')
plt.scatter(scaled3.merchant_id,scaled3['order_count'],color='blue',label='
cluster 3')
plt.scatter(scaled4.merchant_id,scaled4['order_count'],color='yellow',label
='cluster 4')
plt.scatter(scaled5.merchant_id,scaled5['order_count'],color='orange',label
='cluster 5')
#plt.scatter(scaled6.merchant_id,scaled6['order_count'],color='pink',label=
'cluster 6')
plt.legend()
#plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purpl
e',marker='*',label='centroid')
plt.title('Cluster Plot - Order count by Merchant')
plt.xlabel('merchant_id')
plt.ylabel('order_count')
plt.show()

#%%
#Interpretation
for i in range(5):
    average_order_count = model_1[model_1.cluster == i].order_count.count()
    print(" for cluster {} average order count is {}. ".format(i+1,
average_order_count))
#%%
# Model 2 - average price by merchant

model_2 = my_data.groupby('merchant_id', as_index=False).agg({"price":
"mean"})
model_2.rename(columns={'price': 'mean_price'}, inplace=True)
model_2 = model_2.sort_values('mean_price')
model_2 = model_2.head(300)
model_2.head(25)
#plt.scatter(model_2['merchant_id'],model_2['mean_price'])
#%%
# Standardizing the values
#before standardizing
fig, (ax1,ax2) = plt.subplots(ncols=2,figsize=(6, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(model_2['merchant_id'], ax=ax1)
sns.kdeplot(model_2['mean_price'], ax=ax2)
#%%
# Scaling - Transformation

col_names = model_2.columns
features2 = model_2[col_names]

from sklearn.preprocessing import StandardScaler

scaler2 = StandardScaler().fit(features2.values)
features2 = scaler.transform(features2.values)
scaled2 = pd.DataFrame(features2, columns = col_names)
scaled2.head()
```

```
#%%
#after transforming
fig, (ax1,ax2) = plt.subplots(ncols=2,figsize=(6, 5))
ax1.set_title('After Scaling')
sns.kdeplot(scaled2['merchant_id'], ax=ax1)
sns.kdeplot(scaled2['mean_price'], ax=ax2)
#%%
# Clustering with K means
from sklearn.cluster import KMeans

# Using Elbow method to test for best clusters
clusters_range=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]
inertias=[]

for c in clusters_range:
    kmeans=KMeans(n_clusters=c, random_state=0).fit(scaled2)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(7,7))
plt.plot(clusters_range,inertias, marker='o')
#%%
# k means clustering
km = KMeans(n_clusters=5)
y_predicted2 = km.fit_predict(scaled2[['merchant_id','mean_price']])
y_predicted2
model_2['cluster'] = y_predicted2
model_2.head(9)
#%%
# cluster plot
scaled21 = model_2[model_2.cluster==0]
scaled22 = model_2[model_2.cluster==1]
scaled23 = model_2[model_2.cluster==2]
scaled24 = model_2[model_2.cluster==3]
scaled25 = model_2[model_2.cluster==4]

# scatter plot
plt.figure(figsize=(14, 10), dpi=60)
plt.scatter(scaled21.merchant_id,scaled21['mean_price'],color='green',label
='cluster 1')
plt.scatter(scaled22.merchant_id,scaled22['mean_price'],color='red',label='
cluster 2')
plt.scatter(scaled23.merchant_id,scaled23['mean_price'],color='blue',label=
'cluster 3')
plt.scatter(scaled24.merchant_id,scaled24['mean_price'],color='yellow',labe
l='cluster 4')
plt.scatter(scaled25.merchant_id,scaled25['mean_price'],color='orange',labe
l='cluster 5')
#plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purpl
e',marker='*',label='centroid')
plt.xlabel('merchant_id')
plt.ylabel('mean_price')
plt.legend()
plt.title('Average Purchase Price per Merchant')
#%%
# model 3 - quarterly order count

df = my_data
df['date'] = pd.to_datetime(df['date'])
df["Quarter"] = df.date.dt.quarter

df1 = df[df['Quarter'] == 1]
```

```
df2 = df[df['Quarter'] == 2]
df3 = df[df['Quarter'] == 3]
df4 = df[df['Quarter'] == 4]
#%%
df_1 = df1.groupby('merchant_id', as_index=False).agg({"order_id":
"count"})
df_1.rename(columns={'order_id': 'order_count'}, inplace=True)
df_2 = df2.groupby('merchant_id', as_index=False).agg({"order_id":
"count"})
df_2.rename(columns={'order_id': 'order_count'}, inplace=True)
df_3 = df3.groupby('merchant_id', as_index=False).agg({"order_id":
"count"})
df_3.rename(columns={'order_id': 'order_count'}, inplace=True)
df_4 = df4.groupby('merchant_id', as_index=False).agg({"order_id":
"count"})
df_4.rename(columns={'order_id': 'order_count'}, inplace=True)
df_4
#%%
plt.scatter(df_1['merchant_id'],df_1['order_count'])
plt.scatter(df_2['merchant_id'],df_2['order_count'])
plt.scatter(df_3['merchant_id'],df_3['order_count'])
plt.scatter(df_4['merchant_id'],df_4['order_count'])
#%%
# Model X - Total purchase by merchant on quarterly basis
df
model_x = df.groupby(['Quarter','merchant_id'],
as_index=False).agg({"price": "sum"})

model_x = model_x.sort_values('price')
model_x = model_x.head(300)
model_x = model_x[model_x['price'] != 0]
#%%
from matplotlib.pyplot import figure


# Fixing random state for reproducibility
np.random.seed(123)


fig = plt.figure(figsize=(12, 8), dpi=80)
ax = fig.add_subplot(projection='3d')
xs = model_x['Quarter']
ys = model_x['merchant_id']
zs = model_x['price']
ax.scatter(xs, ys, zs)
ax.set_xlabel('Quarter')
ax.set_ylabel('merchant_id')
ax.set_zlabel('Total Price')

plt.show()
#%%
#elbow method
wcss = []
for i in range(1,11):
    k_means = KMeans(n_clusters=i,init='k-means++', random_state=42)
    k_means.fit(model_x)
    wcss.append(k_means.inertia_)
#plot elbow curve
plt.plot(np.arange(1,11),wcss)
plt.xlabel('Clusters')
plt.ylabel('SSE')
```

```
plt.show()
#%%
k_means_optimum = KMeans(n_clusters = 3, init = 'k-
means++',  random_state=42)
y = k_means_optimum.fit_predict(model_x)
print(y)
model_x['cluster'] = y
model_x.head(6)
#%%
data1 = model_x[model_x.cluster==0]
data2 = model_x[model_x.cluster==1]
data3 = model_x[model_x.cluster==2]


kplot = plt.figure(figsize=(16, 12), dpi=360)
kplot = plt.axes(projection='3d')
kplot.scatter3D(data1.Quarter, data1.merchant_id, data1.price, c='red',
label = 'Cluster 1')
kplot.scatter3D(data2.Quarter, data2.merchant_id, data2.price, c='green',
label = 'Cluster 2')
kplot.scatter3D(data3.Quarter, data3.merchant_id, data3.price, c='blue',
label = 'Cluster 3')
#kplot.scatter(k_means_optimum.cluster_centers_[:,0],
k_means_optimum.cluster_centers_[:,1],
k_means_optimum.cluster_centers_[:,2], color = 'purple', s = 200)
kplot.legend()
plt.title("Total Quarterly Purchase by Merchants ")
plt.xlabel('Quarter')
plt.ylabel('merchant_id')
plt.zlabel('Total Price')
plt.show()


#%%
# Reading Input file
model_1x = pd.read_csv("D:/MS DATA ANALYTICS/Quarter 3/ALY 6080/Week
6/datafiles/sku_movement.csv")
print(model_1x)

# model - sales by sku speed
model_1x = model_1x.sort_values('Avg_diff')
model_1x = model_1x.tail(100)
plt.scatter(model_1x['sku_id'],model_1x['Avg_diff'])
#%%
# Standardizing the values
#before standardizing
fig, (ax1,ax2) = plt.subplots(ncols=2,figsize=(6, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(model_1x['sku_id'], ax=ax1)
sns.kdeplot(model_1x['Avg_diff'], ax=ax2)
#%%
# Scaling - Transformation

col_names = model_1x.columns
features = model_1x[col_names]

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
```

```
scaled = pd.DataFrame(features, columns=col_names)
scaled.head()
#%%
#after transforming
fig, (ax1,ax2) = plt.subplots(ncols=2,figsize=(6, 5))
ax1.set_title('After Scaling')
sns.kdeplot(scaled['sku_id'], ax=ax1)
sns.kdeplot(scaled['Avg_diff'], ax=ax2)
#%%
# Clustering with K means
from sklearn.cluster import KMeans

# Using Elbow method to test for best clusters
clusters_range=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]
inertias=[]

for c in clusters_range:
    kmeans=KMeans(n_clusters=c, random_state=0).fit(scaled)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(8, 5), dpi=100)
plt.plot(clusters_range,inertias, marker='o')
#%%
# k means clustering
km = KMeans(n_clusters=5)
y_predicted = km.fit_predict(scaled[['sku_id','Avg_diff']])
y_predicted
#%%
model_1x['cluster'] = y_predicted
model_1x.head(6)
model_1x = model_1x.sort_values('Avg_diff')
model_1x
#%%
import matplotlib

scaled1 = model_1x[model_1x.cluster==0]
scaled2 = model_1x[model_1x.cluster==1]
scaled3 = model_1x[model_1x.cluster==2]
scaled4 = model_1x[model_1x.cluster==3]
scaled5 = model_1x[model_1x.cluster==4]

# scatter plot
plt.figure(figsize=(14, 10), dpi=60)
plt.scatter(scaled1.sku_id,scaled1['Avg_diff'],color='green',label='cluster
1')
plt.scatter(scaled2.sku_id,scaled2['Avg_diff'],color='red',label='cluster
2')
plt.scatter(scaled3.sku_id,scaled3['Avg_diff'],color='blue',label='cluster
3')
plt.scatter(scaled4.sku_id,scaled4['Avg_diff'],color='yellow',label='cluste
r 4')
plt.scatter(scaled5.sku_id,scaled5['Avg_diff'],color='orange',label='cluste
r 5')

plt.legend()
#plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purpl
e',marker='*',label='centroid')
plt.title('Cluster Plot - Order count by Merchant')
plt.xlabel('Sku_id')
plt.ylabel('Speed')
plt.show()
```

Q. 5

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.preprocessing import TransactionEncoder
AwanTunai_data =
pd.read_excel('20220413_Northeastern_AwanTunai_Capstone_Data.xlsx',
keep_default_na = 'TRUE')
AwanTunai_data.head()
AwanTunai_data.isna().sum()
na_AwanTunai_data = AwanTunai_data[AwanTunai_data["top_cat_id"].isna()]
AwanTunai_data = AwanTunai_data.dropna(subset=["top_cat_id"])
AwanTunai_data.isna().sum()
print(AwanTunai_data.dtypes)
AwanTunai_data["order_id"] = pd.Categorical(AwanTunai_data.order_id)
AwanTunai_data["sku_id"] = pd.Categorical(AwanTunai_data.sku_id)
AwanTunai_data["count"] = 1
Order_SKU = AwanTunai_data[['sku_id', 'order_id', 'count']]
print(Order_SKU.dtypes)
print(Order_SKU.nunique())
# Transactions done in
baskets =
Order_SKU.groupby(['order_id'])['sku_id'].apply(list).values.tolist()
baskets
Order_SKU_table = Order_SKU.groupby("sku_id").sum().sort_values("count",
ascending=False).reset_index()
Order_SKU_table.head(10).style.background_gradient(subset='count')
Order_SKU_table.head(10).plot(x="sku_id", y="count", kind="bar", title="SKU
Frequency")
plt.xticks(rotation = 25)
plt.show()
te = TransactionEncoder()
te_array = te.fit(baskets).transform(baskets)
baskets_df = pd.DataFrame(te_array, columns=te.columns_)
baskets_df
from mlxtend.frequent_patterns import apriori

frequent_itemsets_ap = apriori(baskets_df, min_support=0.01,
use_colnames=True)
frequent_itemsets_ap['length'] = frequent_itemsets_ap['itemsets'].apply(lambda
x: len(x))
frequent_itemsets_ap['support'] = frequent_itemsets_ap['support'].apply(lambda
x: x*100)
frequent_itemsets_ap[(frequent_itemsets_ap['length'] ==
1)].sort_values(by='support', ascending=False).head(10)\
[['itemsets', 'support']].style.background_gradient(axis=0,
gmap=frequent_itemsets_ap['support'])
frequent_itemsets_ap[(frequent_itemsets_ap['length'] ==
2)].sort_values(by='support', ascending=False).head(3)\
[['itemsets', 'support']].style.background_gradient(axis=0,
gmap=frequent_itemsets_ap['support'])
frequent_itemsets_ap[(frequent_itemsets_ap['length'] ==
3)].sort_values(by='support', ascending=False).head(3)\
[['itemsets', 'support']].style.background_gradient(axis=0,
gmap=frequent_itemsets_ap['support'])
```

```
frequent_itemsets_ap[(frequent_itemsets_ap['length'] ==
4)].sort_values(by='support', ascending=False).head(3)\
[['itemsets', 'support']].style.background_gradient(axis=0,
gmap=frequent_itemsets_ap['support'])
frequent_itemsets_ap[(frequent_itemsets_ap['length'] ==
4)].sort_values(by='support', ascending=False).head(3)\
[['itemsets', 'support']].style.background_gradient(axis=0,
gmap=frequent_itemsets_ap['support'])
frequent_itemsets_ap['support'] = frequent_itemsets_ap['support'].apply(lambda
x: x/100)
from mlxtend.frequent_patterns import association_rules

rules_ap = association_rules(frequent_itemsets_ap, metric="lift",
min_threshold=20)
rules_ap["antecedents_length"] = rules_ap["antecedents"].apply(lambda x:
len(x))
rules_ap["consequents_length"] = rules_ap["consequents"].apply(lambda x:
len(x))
len(rules_ap)
rules_ap.sort_values(by='confidence', ascending=False).head(5)\
[['antecedents', 'consequents', 'support', 'confidence', 'lift']]\
.style.background_gradient(axis=0, gmap=rules_ap['confidence'])
rules_ap.sort_values(by='lift', ascending=False).head(5)\
[['antecedents', 'consequents', 'support', 'confidence', 'lift']]\
.style.background_gradient(axis=0, gmap=rules_ap['lift'])
rules_ap[rules_ap["antecedents_length"] == 1].sort_values(by='lift',
ascending=False).head(10)\
[['antecedents', 'consequents', 'support', 'confidence', 'lift']]\
.style.background_gradient(axis=0, gmap=rules_ap['lift'])
rules = rules_ap.loc[np.where((rules_ap['lift'] >= 25) &
(rules_ap['confidence'] >= 0.50))]
rules.sort_values(by='lift', ascending=False).head(10)\
[['antecedents', 'consequents', 'support', 'confidence', 'lift']]\
.style.background_gradient(axis=0, gmap=rules_ap['lift'])
```

Model codes:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.preprocessing import TransactionEncoder
AwanTunai_data =
pd.read_excel('20220413_Northeastern_AwanTunai_Capstone_Data.xlsx',
keep_default_na = 'TRUE')
AwanTunai_data.head()
AwanTunai_data.isna().sum()
na_AwanTunai_data = AwanTunai_data[AwanTunai_data["top_cat_id"].isna()]
AwanTunai_data = AwanTunai_data.dropna(subset=["top_cat_id"])
AwanTunai_data.isna().sum()
print(AwanTunai_data.dtypes)
AwanTunai_sku_movement = pd.read_csv('sku_movement.csv')
```

```
AwanTunai_sku_movement.head()
AwanTunai_data["sku_movement"] = 0
AwanTunai_data.sku_movement.update(AwanTunai_data.sku_id.map(AwanTunai_sku_mov
ement.set_index('sku_id').Avg_diff))
AwanTunai_data = AwanTunai_data[AwanTunai_data['sku_movement'] > 0]
AwanTunai_data['sku_movement'] = AwanTunai_data['sku_movement'].round(decimals
= 2)
AwanTunai_data.head()
bins = [1, 15, 30, 45, (AwanTunai_sku_movement["Avg_diff"]).max()]
df = pd.cut(AwanTunai_sku_movement["Avg_diff"], bins)
df.value_counts()
df_merchant_sku_N_movement = AwanTunai_data[['merchant_id', 'sku_movement',
'sku_id']]
df_merchant_sku_N_movement = df_merchant_sku_N_movement.drop_duplicates(subset
= ['merchant_id', 'sku_id'])
df_merchant_sku_N_movement.head()
from scipy.sparse import csr_matrix
from fuzzywuzzy import fuzz
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
df_movement_day_diff_features = df_merchant_sku_N_movement.pivot(
    index = 'sku_id',
    columns = 'merchant_id',
    values = 'sku_movement'
).fillna(0)
sku_list = pd.DataFrame(df_merchant_sku_N_movement.sku_id.unique())
sku_list.columns = ['sku_id']
sku_list = sku_list.sort_values(by='sku_id', ascending=True).reset_index()
sku_list.insert(loc=0, column='row_num', value=np.arange(len(sku_list)))
sku_list = sku_list[['row_num', 'sku_id']]
sku_list.head()
sku_movement_to_idx = sku_list.to_dict()
sku_movement_to_idx = sku_movement_to_idx['sku_id']
sku_movement_to_idx
matrix_movement_day_diff_features =
csr_matrix(df_movement_day_diff_features.values)
def fuzzy_matching(mapper, selected_sku, verbose=True):
    match_tuple = []
    # get match
    for idx, sku_id in mapper.items():
        ratio = fuzz.ratio(str(sku_id), str(selected_sku))
        if ratio == 100:
            match_tuple.append((sku_id, idx, ratio))
    # sort
    match_tuple = sorted(match_tuple, key = lambda x: x[2])[::-1]
    if not match_tuple:
        print('Oops! No match is found')
```

```
        return
    if verbose:
        sku_movement = AwanTunai_sku_movement[AwanTunai_sku_movement["sku_id"]
== 20315]["Avg_diff"]\
        .values[0].round(decimals = 2)
        is_fast = "Fast Moving"
        if sku_movement > 10:
            is_fast = "Slow Moving"
        print('Found possible matches in our database: {0} [{1}]'.format([x[0]
for x in match_tuple], is_fast))
        print('{0} moving with a speed of {1} days on an
average.\n'.format(selected_sku, sku_movement))
    return match_tuple[0][1]
def make_recommendation(model_knn, data, mapper, selected_sku,
n_recommendations):
    # fit
    model_knn.fit(data)
    # get input movie index
    print('You have input SKU:', selected_sku)
    idx = fuzzy_matching(mapper, selected_sku, verbose = True)
    if(idx is not None):
        # inference
        print('Recommendation system start to make inference')
        print('......\n')
        distances, indices = model_knn.kneighbors(data[idx], n_neighbors =
n_recommendations)
        # get list of raw idx of recommendations
        raw_recommends = \
            sorted(list(zip(indices.squeeze().tolist(),
distances.squeeze().tolist())), key=lambda x: x[1])[:0:-1]
        # print recommendations
        print('Recommendations for {}:'.format(selected_sku))
        for i, (idx, dist) in enumerate(raw_recommends):
            try:
                print('{0}: {1}, with distance of {2}'.format(i+1,
mapper[idx], dist))
            except:
                print("{0}: Exception thrown. {1} SKU does not
exist.".format(i+1, idx))
# define model
model_knn = NearestNeighbors(metric='cosine',\
                             algorithm='brute',\
                             n_neighbors = 3,\
                             n_jobs = -1)
selected_sku = '20315'

make_recommendation(
    model_knn = model_knn,
```

```
    selected_sku = selected_sku,
    data = matrix_movement_day_diff_features,
    mapper = sku_movement_to_idx,
    n_recommendations = 11)
```