ETW2001 Foundations of Data Analysis

Foo Kai Yan 33085625

kfoo0012@student.monash.edu

Section A

First, the required libraries are loaded after setting the correct working directory. The environment is cleaned to eliminate potential conflicts that might occur in the code and to ensure that only all required files are loaded and processed correctly into the environment without errors.

```
R 4.3.3 · C:/Monash/ETW2001/
> ################################################################################
> # Set Working Directory
> setwd("C:/Monash/ETW2001")
> # Load required libraries
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.0     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
✓ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
i Use the conflicted package to force all conflicts to become errors
> library(tidyr)
> library(dplyr)
> ################################################################################
> #                                  SECTION A                                   #
> ################################################################################
> # Remove/Clean the environment
> rm(list=ls())
```

1. After importing the 3 datasets into the environment, the dimensions and the columns of each dataset is printed out to give a brief understanding of the 3 datasets. The 'Date' column of the sales datasets is found to be 'character' datatype and not numerical nor datatime format. Hence, extra work done in code is to convert the whole 'Date' column into datetime format so year 2020 can be filter out with higher accuracy. In conclusion, In year 2020, there is a total of 50032 sales transactions.

```
R 4.3.3 · C:/Monash/ETW2001/
> ## Question 1
> # Load the provided datasets
> sales_dataset = read.csv("sales.csv", header = TRUE)
> products_dataset = read.csv("products.csv", header = TRUE)
> inventory_dataset = read.csv("inventory.csv", header = TRUE)
> # Look into the dimensions of each datasets
> dim(sales_dataset) # There is 200000 rows and 6 columns
[1] 200000      6
> dim(products_dataset) # There is 1001 rows and 4 columns
[1] 1001    4
> dim(inventory_dataset) # There is 1000 rows and 6 columns
[1] 1000    6
> # Column names of each datasets
> names(sales_dataset)
[1] "SalesId"   "StoreId"   "ProductId" "Date"      "UnitPrice" "Quantity"
> names(products_dataset)
[1] "ProductId"   "ProductName" "Supplier"    "ProductCost"
> names(inventory_dataset)
[1] "ProductId"         "StoreId"           "StoreName"         "Address"           "neighborhood"
[6] "QuantityAvailable"
> # Filter to include transactions occurred in year 2020
> typeof(sales_dataset$Date)
[1] "character"
> # From here we know that data in Date column is 'character', not a Datetime format
> # So here, we convert the data in Date column to Datetime format
> sales_dataset$Date <- as.Date(sales_dataset$Date, format="%Y-%m-%d")
> year2020_transactions <- sales_dataset %>% filter(format(Date, "%Y") == "2020")
> head(year2020_transactions)
  SalesId StoreId ProductId       Date UnitPrice Quantity
1   11624   71053       883 2020-07-13    7.3675       33
2   29702   22623       632 2020-07-20    0.6475        8
3   82434   22748       922 2020-04-29    0.7875       68
4   59436   22745       441 2020-04-17    1.6800       82
5   35313   22914       356 2020-12-25    3.9900       92
6   21997   22632       648 2020-04-03    0.4025       96
> # 'count' was used to count the total number of sales transactions happening in 2020
> count(year2020_transactions)
      n
1 50032
> 
```

2. To calculate the total revenue for each product in the sales data, the sales dataset is first grouped by the product ID as there might be repeated transactions present for each product within the sales dataset. The total revenue is then calculated by summing up the Unit Price of each unique product ID and its sales quantity. The value is stored under a new column named 'TotalRevenue' within the sales dataset. To get the highest total revenue product ID, the 'TotalRevenue' column is arranged with decending order to get the highest total revenue product ID as the first product ID within the list presented. The highest total revenue product ID is 248. The product dataset is filtered to get the 'Product Name' of the product with the 'ProductID' of 248 which is found to be Sultanas. Inventory dataset is then filtered to get the 'StoreID' of the store that sold Sultanas, the highest total revenue product which is found to be 21724, which with further filtering of the Inventory dataset, we can find that the store with 'StoreID' 21724 is Kmart.

```
R 4.3.3 · C:/Monash/ETW2001/
> ## Question 2
> # First, we group sales data by their Product Id because there might be repeated transactions for each products
> product_sales <- group_by(sales_dataset, ProductId)
> product_sales
# A tibble: 200,000 × 6
# Groups:   ProductId [1,000]
   SalesId StoreId ProductId Date       UnitPrice Quantity
     <int>   <int>     <int> <date>         <dbl>    <int>
 1   82319   22726       590 2019-12-02    0.0525       93
 2   15022   21754       390 2017-11-19    5.11         28
 3   11624   71053       883 2020-07-13    7.37         33
 4   63101   22914       658 2019-05-12    2.08         76
 5   29702   22623       632 2020-07-20    0.648         8
 6   35660   22749       170 2019-09-30    5.67         93
 7   69913   22633       444 2017-11-03    0.438        98
 8   47278   84969       184 2018-04-17    9.20         17
 9   46126   48187       316 2017-10-13    3.40         47
10   69718   22726       972 2018-09-12    1.59         43
# ℹ 199,990 more rows
# ℹ Use `print(n = ...)` to see more rows
> # Then calculate the total revenue for each product after grouping
> product_revenue <- summarize(product_sales, TotalRevenue = sum(UnitPrice * Quantity))
> product_revenue
# A tibble: 1,000 × 2
   ProductId TotalRevenue
       <int>        <dbl>
 1         1       20195
 2         2       51572.
 3         3        3042.
 4         4       94313.
 5         5       12548.
 6         6        6543.
 7         7       25213.
 8         8       24836.
 9         9       45558.
10        10        1905.
# ℹ 990 more rows
# ℹ Use `print(n = ...)` to see more rows
> # Arrange the products by total revenue in descending order so highest sales is at the top
> desc_total_revenue <- arrange(product_revenue, desc(TotalRevenue))
> desc_total_revenue
# A tibble: 1,000 × 2
   ProductId TotalRevenue
       <int>        <dbl>
 1       248      223612.
 2       117      197093.
 3       885      195622.
 4        89      182576.
 5       845      167054.
 6        99      160996.
 7        88      156044.
 8       367      153489.
 9       805      152316.
10       149      145419.
# ℹ 990 more rows
# ℹ Use `print(n = ...)` to see more rows
> # Get the Product Id of the product with the highest total revenue
> top_revenue_product_id <- desc_total_revenue[1, ]$ProductId
> top_revenue_product_id
[1] 248
```

```
> # Filter to find the Product Name of that Product Id from products data
> top_revenue_product_name <- products_dataset %>%
+   filter(ProductId == top_revenue_product_id) %>%
+   select(ProductName)
> top_revenue_product_name
  ProductName
1    Sultanas
> # Filter to find the the Store Id of the product with the highest total revenue
> top_revenue_product_store_id <- inventory_dataset %>%
+   filter(ProductId == top_revenue_product_id) %>%
+   select(StoreId)
> top_revenue_product_store_id
  StoreId
1   21724
> # Filter to find the Store name of the product with the highest total revenue
> top_revenue_store_name <- inventory_dataset %>%
+   filter(StoreId == top_revenue_product_store_id[1, ]) %>%
+   select(StoreName)
> unique(top_revenue_store_name)
  StoreName
1     Kmart
>
```

3. To get the average Quantity Available across all products, the inventory dataset is first grouped by 'StoreID' as there might be repeated 'StoreID' present for each store within the inventory dataset. Then, the average quantity across all product is calculated by getting the mean of the 'QuantityAvailable' and during the calculation of the mean, na.rm = TRUE is used to remove missing values from the calculation. The 'StoreID' by 'AverageQuantity' is arranged in ascending order to find the lowest average Quantity Available which is found to be 22914.

```
R 4.3.3 · C:/Monash/ETW2001/
> ## Question 3
> # Group inventory data by Store Id
> inventory_by_store_id <- group_by(inventory_dataset, StoreId)
> inventory_by_store_id
# A tibble: 1,000 × 6
# Groups:   StoreId [34]
   ProductId StoreId StoreName              Address                neighborhood            QuantityAvailable
       <int>   <int> <chr>                  <chr>                  <chr>                               <int>
 1         1   85123 National Stores        9 Springview Point     Bolton Hill                            11
 2         2   71053 Family Dollar          5434 Daystar Circle    Ashburton                               1
 3         3   84406 BJ's Wholesale Club    3 Darwin Drive         Morrell Park                           11
 4         4   84029 Ocean State Job Lot    684 Bunting Lane       Fells Point                             1
 5         5   37444 Ollie's Bargain Outlet 50162 John Wall Drive  Charles Village                         3
 6         6   84406 BJ's Wholesale Club    8 Fordem Pass          Morrell Park                            9
 7         7   84406 BJ's Wholesale Club    89058 Monument Hill    Morrell Park                            4
 8         8   22752 Costco                 1 Clarendon Way        Baltimore Highlands                     3
 9         9   21730 Fred's                 629 Crescent Oaks Center Pulaski Industrial Area               8
10        10   22633 Big Lots               175 Kim Place          Brooklyn                                7
# ℹ 990 more rows
# ℹ Use `print(n = ...)` to see more rows
```

```
> # Summarize the average Quantity Available for each Store Id whilst removing any NA or empty data
> average_quantity_available <- summarize(inventory_by_store_id, AverageQuantity = mean(QuantityAvailable, na.rm
 = TRUE))
> average_quantity_available
# A tibble: 34 × 2
   StoreId AverageQuantity
     <int>           <dbl>
 1   10002            6.89
 2   21035            6.32
 3   21724            6.36
 4   21730            7.03
 5   21754            6.04
 6   21755            6.31
 7   21756            6.76
 8   21777            5.31
 9   21791            6.62
10   21883            6.39
# ℹ 24 more rows
# ℹ Use `print(n = ...)` to see more rows
> # Arrange the Store Id by average Quantity Available in ascending order to find the lowest average Quantity Ava
ilable
> sorted_average_quantity <- arrange(average_quantity_available, AverageQuantity)
> sorted_average_quantity
# A tibble: 34 × 2
   StoreId AverageQuantity
     <int>           <dbl>
 1   22914            5
 2   22623            5.2
 3   21777            5.31
 4   22622            5.41
 5   37444            5.52
 6   71053            5.55
 7   84029            5.67
 8   22728            5.73
 9   22726            5.77
10   22633            6
# ℹ 24 more rows
# ℹ Use `print(n = ...)` to see more rows
> # Get the Store Id with the lowest average quantity available
> lowest_average_quantity_store_id <- sorted_average_quantity[1, ]$StoreId
> lowest_average_quantity_store_id
[1] 22914
>
```

4. The new column in the sales dataset that categorizes sales is named ‘SaleStatus’. Initially, all rows under ‘SaleStatus’ is temporarily placed with NA as it serves as a temporary placeholder until the sales can be accurately classified according to the determined categories of “High”, “Medium” or “Low”. Once sales can be accurately classified, the ‘SaleStatus’ is mutated to insert the sales categories with case_when. The number of sales categorised as “High” is 101742.

```
> ## Question 4
> # Insert new column named 'SaleStatus' and temporarily put NA for all the rows
> sales_dataset$SaleStatus <- NA
> names(sales_dataset)
[1] "SalesId"    "StoreId"    "ProductId"  "Date"       "UnitPrice"  "Quantity"   "SaleStatus"
> # Mutate to add data into the new column
> sales_dataset <- sales_dataset %>%
+   mutate(SaleStatus = case_when(
+     Quantity >= 50 ~ 'High',
+     Quantity >= 20 & Quantity < 50 ~ 'Medium',
+     TRUE ~ 'Low'
+   ))
> # Count of 'High' category sales
> high_count <- sum(sales_dataset$SaleStatus == 'High')
> high_count
[1] 101742
>
```

5. In tidyr, separate function is used with the indication of the separator as ' - ' to to separate the 'ProductName' into two columns which are 'Product' and 'Brand'. With that, we can see that the 3rd most expensive brand is Rye that sold Flour.

```
> ## Question 5
> # Arrange in descending order of Product Cost
> products_dataset <- products_dataset %>% arrange(desc(ProductCost))
> head(products_dataset)
  ProductId              ProductName            Supplier ProductCost
1       885             Broom - Corn Harbor Freight Tools       12.09
2       248                 Sultanas               Kmart       12.05
3       117              Flour - Rye         Ben Franklin       11.39
4        89     Pop Shoppe Cream Soda          Ross Stores        9.62
5       367          Salmon - Fillets  Ocean State Job Lot        8.88
6        99 Cookies - Englishbay Wht               Kmart        8.84
> # Separate Product Name into Product - Brand
> products_dataset <- products_dataset %>% separate(ProductName, into = c("Product", "Brand"), sep = " - ")
Warning messages:
1: Expected 2 pieces. Additional pieces discarded in 26 rows [19, 55, 80, 134, 173, 314, 354, 395, 438, 443,
482, 534, 547, 563, 667, 672, 687, 721, 732, 761, ...].
2: Expected 2 pieces. Missing pieces filled with `NA` in 242 rows [2, 4, 7, 11, 20, 22, 30, 34, 47, 49, 53, 56,
59, 62, 67, 70, 74, 75, 78, 86, ...].
> head(products_dataset)
  ProductId               Product           Brand            Supplier ProductCost
1       885                 Broom            Corn Harbor Freight Tools       12.09
2       248              Sultanas            <NA>               Kmart       12.05
3       117                 Flour             Rye         Ben Franklin       11.39
4        89 Pop Shoppe Cream Soda            <NA>          Ross Stores        9.62
5       367                Salmon         Fillets  Ocean State Job Lot        8.88
6        99               Cookies Englishbay Wht               Kmart        8.84
> # List with 3rd index to find the 3rd most expensive brand
> third_most_expensive_brand <- products_dataset$Brand[3]
> third_most_expensive_brand
[1] "Rye"
> 
```

6. First, the average price and quantity sold for each product present in the dataset is calculated by having 'AvergePrice' as the mean of each product's 'UnitPrice' and 'AverageQuantity' as the mean of each products' 'Quantity'. Then, the percentage change is calculated before calculating the price elasticity of demand for each product. Finally, the product with the least and most sensitive to price changes is taken from the output where the product with the most sensitive to price changes is product with the product ID 247 with price elasticity of 9.47 and the product with the least sensitive to price changes is product with the product ID 900 with price elasticity of 0.0000292. Eventhough the product with the most sensitive to price changes is product with the product ID 973 but due to price elasticity of Inf, product with product ID of 973 is not included. And finally, the percentage changes between consecutive time periods are already implicitly calculated within the code.

   Products most sensitive to price changes are typically those with many substitutes available in the market, where even a minor price change can significantly affect demand as most consumers would prefer products that are cheaper than other products even though both products present the same functionalities and usage. Hence, the products sales competition is based on price rather than differentiated features as most often the features of these products is the same or similar enough. From the output, it is noted that product ID 247 is Veal supplied by Leg Five Below. Although it is unknown on what is Veal but from the price elasticity, we know that Veal could be easily replaced in the market.

On the other hand, products least sensitive to price changes are usually unique and don't have a valid subsitute like essential goods or premium products with strong brand loyalty. For these products least sensitive to price changes, customers may be willing to pay more, regardless of minor price fluctuations as these products could not be easily replicated in terms of functionality and usage. From the output, it is noted that product ID 900 is Chives supplied by Fresh Five Below. From the price elasticity, we know that Chives should be unique and the only one in the market for the consumers to purchase.

```
R 4.3.3 · C:/Monash/ETW2001/
> ## Question 6
> # Calculate the average price and quantity sold for each product present in the dataset
> head(sales_dataset)
  SalesId StoreId ProductId       Date UnitPrice Quantity
1   82319   22726       590 2019-12-02    0.0525       93
2   15022   21754       390 2017-11-19    5.1100       28
3   11624   71053       883 2020-07-13    7.3675       33
4   63101   22914       658 2019-05-12    2.0825       76
5   29702   22623       632 2020-07-20    0.6475        8
6   35660   22749       170 2019-09-30    5.6700       93
> average_price_quantity <- sales_dataset %>%
+   group_by(ProductId) %>%
+   summarise(AveragePrice = mean(UnitPrice),
+          AverageQuantity = mean(Quantity))
> head(average_price_quantity)
# A tibble: 6 × 3
  ProductId AveragePrice AverageQuantity
      <int>        <dbl>           <dbl>
1         1        2.19             52.8
2         2        5.23             46.7
3         3        0.298            50.6
4         4        9.24             49.5
> # Calculate the percentage change in quantity sold and unit price
> # Then, the percentage change within each Product Id group is calculated
> average_price_quantity <- average_price_quantity %>%
+   mutate(PercentChangeQuantity = (AverageQuantity - lag(AverageQuantity)) / lag(AverageQuantity) * 100,
+          PercentChangePrice = (AveragePrice - lag(AveragePrice)) / lag(AveragePrice) * 100)
> head(average_price_quantity)
# A tibble: 6 × 5
  ProductId AveragePrice AverageQuantity PercentChangeQuantity PercentChangePrice
      <int>        <dbl>           <dbl>                 <dbl>              <dbl>
1         1        2.19             52.8                 NA                 NA
2         2        5.23             46.7                -11.5              139.
3         3        0.298            50.6                  8.36             -94.3
4         4        9.24             49.5                 -2.10            3006.
5         5        1.36             51.4                  3.65             -85.2
6         6        0.648            47.7                 -7.19             -52.6
> # Calculate the price elasticity of demand
> # Now, we can summarize the percentage changes to calculate elasticity
> elasticity <- average_price_quantity %>%
+   group_by(ProductId) %>%
+   mutate(PriceElasticity = abs(PercentChangeQuantity / PercentChangePrice))
> elasticity
# A tibble: 1,000 × 6
# Groups:   ProductId [1,000]
   ProductId AveragePrice AverageQuantity PercentChangeQuantity PercentChangePrice PriceElasticity
       <int>        <dbl>           <dbl>                 <dbl>              <dbl>           <dbl>
 1         1        2.19             52.8                 NA                 NA          NA
 2         2        5.23             46.7                -11.5              139.          0.0823
 3         3        0.298            50.6                  8.36             -94.3         0.0886
 4         4        9.24             49.5                 -2.10            3006.          0.000700
 5         5        1.36             51.4                  3.65             -85.2         0.0428
 6         6        0.648            47.7                 -7.19             -52.6         0.137
 7         7        2.71             48.9                  2.63             319.          0.00826
 8         8        2.61             51.5                  5.24              -3.87        1.35
 9         9        4.08             50.8                 -1.36              56.4         0.0241
10        10        0.175            53.6                  5.56             -95.7         0.0581
# ℹ 990 more rows
# ℹ Use `print(n = ...)` to see more rows
```

```
> # Product most sensitive to price changes
> most_sensitive <- elasticity[order(-elasticity$PriceElasticity), ]
> most_sensitive
# A tibble: 1,000 × 6
# Groups:   ProductId [1,000]
   ProductId AveragePrice AverageQuantity PercentChangeQuantity PercentChangePrice PriceElasticity
       <int>        <dbl>           <dbl>                 <dbl>              <dbl>           <dbl>
 1       973         1.59            54.3                  3.33              0                 Inf
 2       247         3.92            54.2                  8.38             -0.885            9.47
 3       801         4.66            50.5                 -3.33             -0.375            8.88
 4       830         3.90            49.8                 -2.55             -0.446            5.71
 5       364         3.32            53.7                 12.2               2.15             5.69
 6       180         3.74            48.1                 -5.67              1.42             3.99
 7        35         3.88            47.5                 -5.68             -2.20             2.58
 8        60         3.17            52.3                  6.89              2.84             2.43
 9       173         3.59            46.9                -13.8              -8.07             1.71
10       561         2.70            53.9                  7.86              4.76             1.65
# ℹ 990 more rows
# ℹ Use `print(n = ...)` to see more rows
> most_sensitive_product <- products_dataset %>%
+   filter(ProductId == most_sensitive$ProductId[2])
> most_sensitive_product
  ProductId ProductName   Supplier ProductCost
1       247  Veal - Leg Five Below        2.24
> # Product least sensitive to price changes
> least_sensitive <- elasticity[order(elasticity$PriceElasticity), ]
> least_sensitive
# A tibble: 1,000 × 6
# Groups:   ProductId [1,000]
   ProductId AveragePrice AverageQuantity PercentChangeQuantity PercentChangePrice PriceElasticity
       <int>        <dbl>           <dbl>                 <dbl>              <dbl>           <dbl>
 1       900         5.06            52.6               0.0107              366.         0.0000292
 2       472         8.35            48.6              -1.46              23750          0.0000614
 3       563        10.4             53.4              -0.0780              962.         0.0000810
 4       246         3.96            50.0               1.94              22500          0.0000864
 5       559         1.59            49.8               0.00693             -72.1        0.0000961
 6       866         2.98            50.2               0.0637              467.         0.000137
 7       479         1.05            52.2              -0.156              1100          0.000142
 8       347         0.49            48.3              -0.0153              -91.4        0.000167
 9       896         3.18            51.4               1.87               9000          0.000208
10       342         4.13            50.0              -0.206               973.         0.000212
# ℹ 990 more rows
# ℹ Use `print(n = ...)` to see more rows
> least_sensitive_product <- products_dataset %>%
+   filter(ProductId == least_sensitive$ProductId[1])
> least_sensitive_product
  ProductId    ProductName   Supplier ProductCost
1       900 Chives - Fresh Five Below        2.89
> 
```

Section B

1. From the video, it is noted that number of stores Starbucks have worldwide is 30,000 and there is a total of 100 million transactions processed by Starbucks per week. So, X is 30,000 and Y should be 100 but since the question categorised the transaction level with how many thousands transaction per week per store so the value of Y is 100,000,000. Average transaction per store is calculated by dividing 100,000,000 by 30,000. 100,000,000 divided by 30,000 is approximately 3333.33 which is under the “Medium” category.

```
R 4.3.3 · ~/
> ## Question 1
> # Video timestamps:
> # 0:22 --> over 100million transactions per week
> # 1:20 --> 30,000 stores worldwide & process almost 100million transactions per week
> x_number_of_stores <- 30000
> y_total_transactions <- 100000000
> average_transactions_per_store <- y_total_transactions / x_number_of_stores
>
> average_transaction_category <- ifelse(average_transactions_per_store > 3500, "High",
+                                 ifelse(average_transactions_per_store >= 2500 & average_transactions_per_store
= 3500, "Medium", "Low"))
> paste("The average Starbucks stores transaction category level is", average_transaction_category)
[1] "The average Starbucks stores transaction category level is Medium"
>
```

2. From the video, it is noted that the 5 potential variables to predict the locations for opening a new Starbucks store is population, income level, traffic, competitor presence and proximity to other Starbucks locations. The value of each of these 5 potential variables is assigned randomly and hence might not be logical and valid but these values would be used for presentation purposes. The classification that indicates the suitability of the location for opening a new Starbucks store with the use of these 5 potential variables is done with if-else function. With population of 77,777 with their income level of 27,000 and area traffic of 'Medium' level with only 7 competitors present in the area and only a distance of 7km away from the nearest Starbucks store, the location is classified to be an average location for opening a new Starbucks store.

```
> population <- 77777      # Total population in the area
> income_level <- 27000    # Average income level of the population
> traffic <- 'Medium'         # Average daily traffic in the area
> competitor_presence <- 7 # Number of competitors in the area
> proximity_to_other_starbucks_locations <- 17 # Distance in km to the nearest Starbucks
>
> location_category_level <- ifelse(population > 58333 & income_level > 20250 & traffic == 'Very High' & competit
or_presence < 2 & proximity_to_other_starbucks_locations > 12, "Ideal location",
+                              ifelse(population > 38888 & income_level > 13500 & traffic == 'High' & competitor_p
resence < 7 & proximity_to_other_starbucks_locations > 7, "Good location",
+                                     ifelse(population > 19444 & income_level > 6750 & traffic == 'Medium' & comp
etitor_presence < 9 & proximity_to_other_starbucks_locations > 4, "Average location",
+                                            "Poor location")))
> paste("This is a", location_category_level, "for opening a new Starbucks store")
[1] "This is a Average location for opening a new Starbucks store"
>
```

3. From the video, it is noted that in mid-2018, Starbucks used local factors weather and time of the day to promote specific products. The promotions are categorized with if-else function as shown in the code below. In addition to logical functions if conditions like on a rainy or cold weather then the Starbucks store should promote hot beverages to warm up the consumers on such cold or cool weathers. The example given is when time of the day is mid-night which is when most Starbucks stores is close unless the Starbucks is opened for 24 hours but in this scenario, it is assumed that Starbucks is not opened 24 hours so there is an extra condition to express that is the time of the day is mid-night then Starbucks is not promoting any beverages or meals as Starbucks is closed.

```
> ## Question 3
> # Video timestamps:
> # 5:33 --> collect data based on weather patterns and their relationship with customer order pattern
> #      --> provide personalised experiences & promotions
> #      --> eg. delivering cold drinks on hot weathers
> # 6:55 --> promote specific products based on local factors such as weather or time of the day
> weather <- "rainy"
> time_of_day <- "mid-night"
>
> if (time_of_day == "mid-night") {
+   promotion <- "No promotion"
+ } else if (weather == "cold" || weather == "rainy") {
+   promotion <- "Promote hot beverages"
+ } else if (weather == "hot" && time_of_day == "afternoon") {
+   promotion <- "Promote cold beverages"
+ } else if (time_of_day == "morning" || time_of_day == "evening") {
+   promotion <- "Promote pastries"
+ } else {
+   promotion <- "No promotion"
+ }
> paste("Today's promotion is", promotion)
[1] "Today's promotion is No promotion"
```