Rohlik Orders Challenge

Yuanhang(Charles) Zhang

2024-08-11

Introduction

Rohlik Group is a leading European e-grocery provider. It operates across 11 warehouses in Czech Republic, Germany, Austria, Hungary, and Romania. To better allocate the resources, it is important for Rohlik to predict the order number in advance. Here in this project, we will attempt to build a prediction model to give guidance for the operation in the coming months.

In the beginning, we need to load the packages used in the following code:

```
# Download and install the packages if not directly available
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(broom)) install.packages("broom", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: broom
```

```
if(!require(readxl)) install.packages("rlang", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: readxl
```

```r
# Load the packages
library(tidyverse)
library(dplyr)
library(caret)
library(broom)
library(readxl)
```

Data

Importing Data

The data used in this project can be downloaded from https://www.kaggle.com/competitions/rohlik-orders-forecasting-challenge/data . In this project, all the files downloaded from Kaggle is stored in `./data`.

It can also be downloaded using this terminal code (API):

`kaggle competitions download -c rohlik-orders-forecasting-challenge`

Here is the format and the information of these sources:

No.	File Name	File Path	Description
1	train	./data/train.csv	training set with historical data
2	test	./data/test.csv	test set with all indicators lacking order numbers and some extra indicators
3	train_calendar	./data/train_calendar .csv	calendar for the training set
4	test_calendar	./data/test_calendar.csv	calendar for the test set
5	solution_example	./data/ solution_example .csv	an example of how to submit results to Kaggle

The original format of the data is `.csv` file. So we first need to read these data:

```r
# Read the data from Kaggle

dat <- read_csv('./data/train.csv')
final_holdout_test <- read_csv('./data/test.csv')
```

Here, we named the data in the `./data/train.csv` as `dat`, marking it as the data we have. And the data in `./data/test.csv` as `final_holdout_test` to show it is the final test set we need to predict.

Exploratory Analysis and Pre-processing

Size and indicators

We can start by seeing the size of the `dat` and `final_houldout_set`.

```r
# Explore the size of data

tibble("# Rows" = nrow(dat), "# Columns" = ncol(dat))
```

```
## # A tibble: 1 x 2
##   `# Rows` `# Columns`
##      <int>       <int>
## 1     7340          18
```

The `dat` set has 7340 rows with 18 columns. The column names are:

```r
names(dat) # Print out the column names
```

```
##  [1] "warehouse"              "date"                   "orders"
##  [4] "holiday_name"           "holiday"                "shutdown"
##  [7] "mini_shutdown"          "shops_closed"           "winter_school_holidays"
## [10] "school_holidays"        "blackout"               "mov_change"
## [13] "frankfurt_shutdown"     "precipitation"          "snow"
## [16] "user_activity_1"        "user_activity_2"        "id"
```

The `orders` column is the order number, the key output result.

And here is the `final_holdout_test` set:

```r
# Explore the size of final_holdout_test

tibble("# Rows" = nrow(final_holdout_test),
       "# Columns" = ncol(final_holdout_test))
```

```
## # A tibble: 1 x 2
##   `# Rows` `# Columns`
##      <int>       <int>
## 1      397           8
```

The number of entries we need to predict is 397. It is worth noting that **there are only 8 columns in the final test set.** It means there are extra columns in the `dat` that we cannot directly use! These extra columns are:

```r
# List all the columns that in dat but not in final_holdout_test

tibble('Col_names' = names(dat)) %>%
  filter(!Col_names %in% names(final_holdout_test))
```

```
## # A tibble: 10 x 1
##    Col_names
##    <chr>
##  1 orders
##  2 shutdown
##  3 mini_shutdown
##  4 blackout
##  5 mov_change
##  6 frankfurt_shutdown
##  7 precipitation
##  8 snow
##  9 user_activity_1
## 10 user_activity_2
```

Among them, `orders` is reasonably missing because it is the key output result. For other indicators, we will later check their status and then decide whether we simply drop these parameters or find a way to generate similar indicators to attach to the `final_holdout_test`.

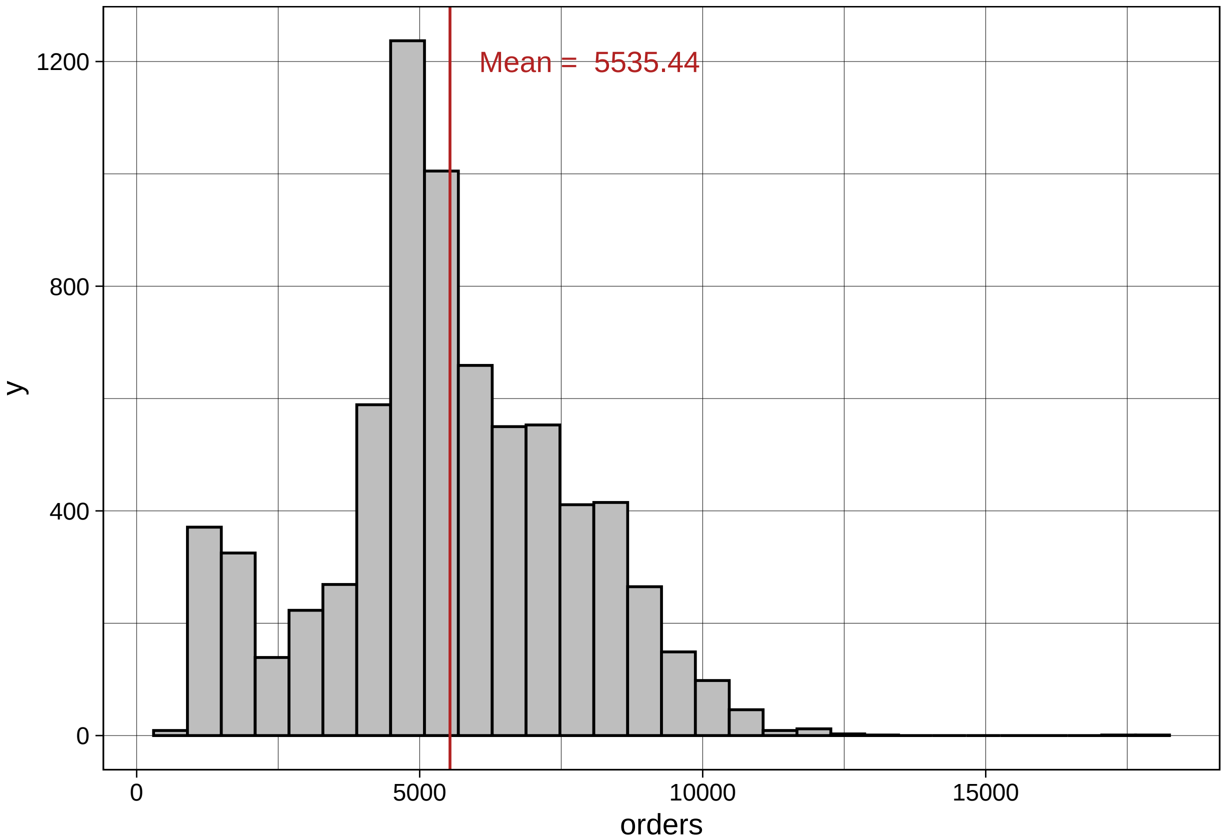
Orders

The `orders` is column is the one we are going to predict. Here we derive a histogram to first see its distributions:

```r
# Derive a plot of order number distribution

```

```r
dat %>% ggplot(aes(x = orders)) +
  geom_histogram(fill = "grey", col = "black") +
  geom_vline(aes(xintercept = mean(orders)), col = "firebrick") + # Add a line of average orders
  annotate(
    geom = "text", x = 8000, y = 1200,
    label = paste("Mean = ", round(mean(dat$orders), 2)), # Add text annotations
    col = "firebrick") +
  theme_linedraw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



From the graph, we know the graph is somehow unevenly distributed, with some deviation from normal distribution and extreme values.

Here is more information on the characteristics of the order amount:

```r
summary(dat$orders) # Summarize the characteristics
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     790    4434    5370    5535    7009   18139
```

Indicators in both sets

Here are the columns that appear in both sets:

```r
# List all the columns that in both sets

tibble('Col_names' = names(dat)) %>%
  filter(Col_names %in% names(final_holdout_test))
```

```
## # A tibble: 8 x 1
##   Col_names
##   <chr>
## 1 warehouse
## 2 date
## 3 holiday_name
## 4 holiday
## 5 shops_closed
## 6 winter_school_holidays
## 7 school_holidays
## 8 id
```

On this list, the `id` column consists of warehouse name and date, used to mark each entry distinctly. The other statistically significant indicators will be discussed in this section:

Warehouses Rohlik would deliver orders from different warehouses. From the table below, we see the warehouses and their orders situation:

```r
# Derive the average orders for every warehouse

warehouse_tab <- dat %>% group_by(warehouse) %>%
  summarise(total_orders = sum(orders), average_orders = mean(orders), entries = n()) %>%
  arrange(desc(total_orders))

warehouse_tab
```

```
## # A tibble: 7 x 4
##   warehouse   total_orders average_orders entries
##   <chr>              <dbl>          <dbl>   <int>
## 1 Prague_1        10182657          8535.    1193
## 2 Brno_1           8678517          7275.    1193
## 3 Budapest_1       6411468          5556.    1154
## 4 Prague_2         6134517          5142.    1193
## 5 Prague_3         5614152          4706.    1193
## 6 Munich_1         2665933          3396.     785
## 7 Frankfurt_1       942914          1499.     629
```

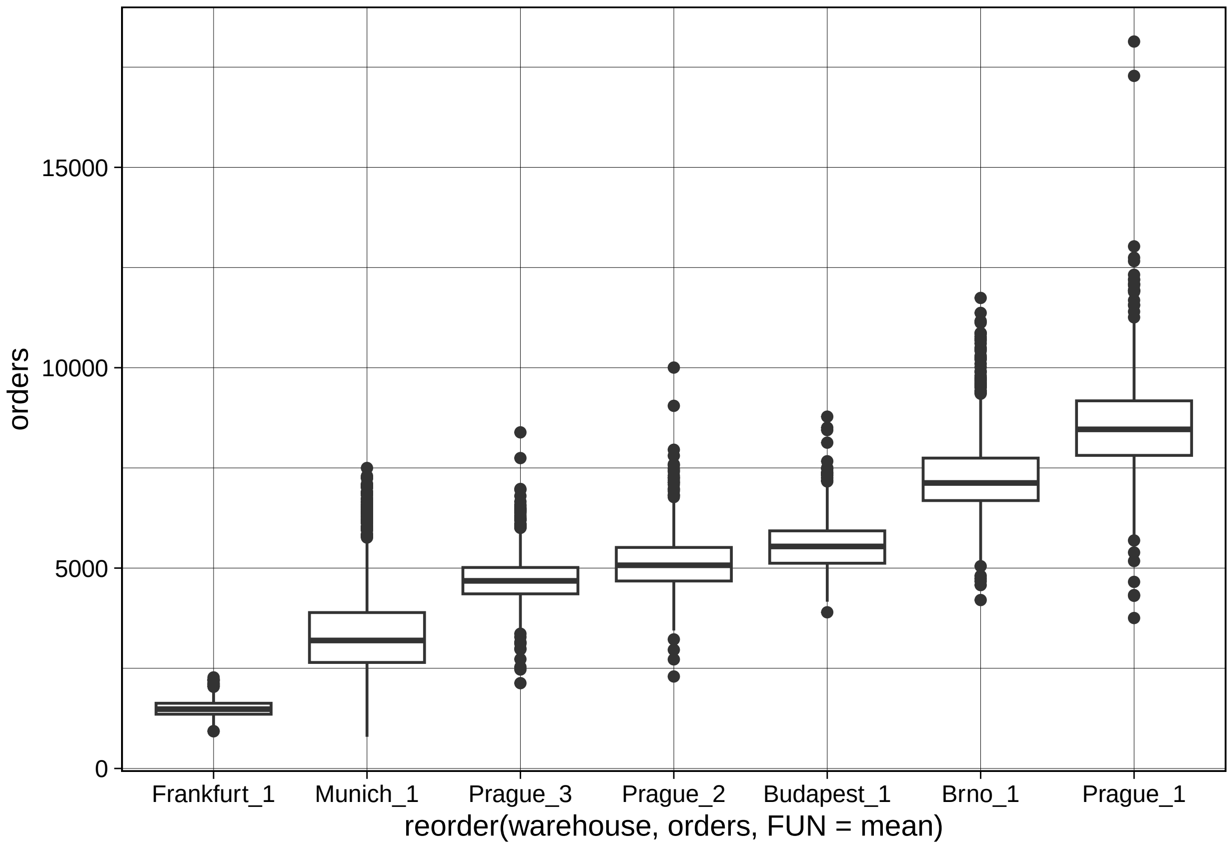
And by quick check we can know that `final_holdout_test` shares the same warehouses with `dat`.

```r
# Check if the name of warehouses in both sets are the same

sum(!unique(final_holdout_test$warehouse) %in% warehouse_tab$warehouse)
```

```
## [1] 0
```

The plot here shows that the `orders` distributes differently in warehouses. Besides, there are two obvious outliers in the warehouse 'Prague_1' with order number over 15000. Later we will deal with these abnormalites to see if it's necessary to eliminate their effect.

```r
# Derive a plot to depict distributions in different warehouses

dat %>% ggplot(aes(
  x = reorder(warehouse, orders, FUN = mean),
  y = orders)) +
  geom_boxplot() +
  theme_linedraw()
```



The chunk above (line 2) is composed with the help of this StackOverflow Post.

Date `date` column is included in the data sets. And this column is checked to have been the tidy format of date:

```
class(dat$date)
```

```
## [1] "Date"
```

Now we need to confirm the time range in the `dat` and `final_holdout_test`.

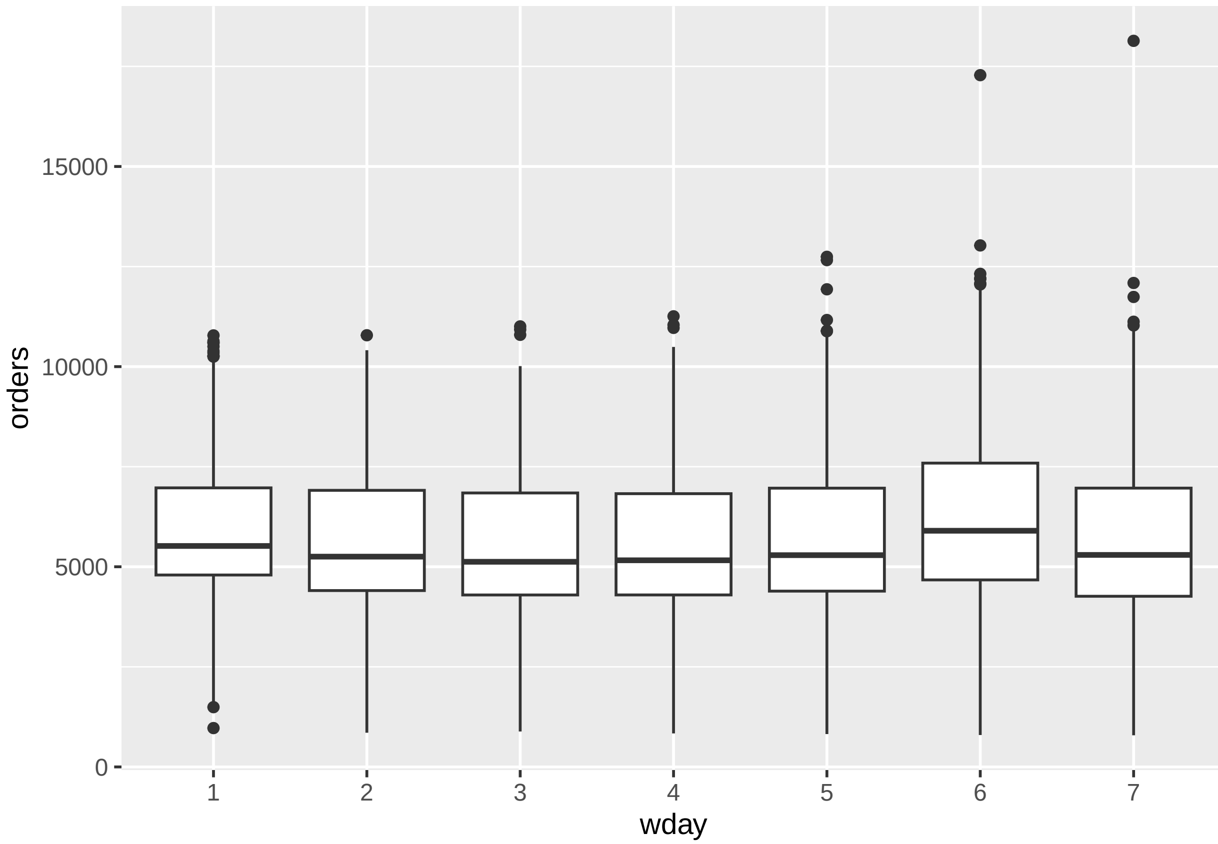
```
# Compare the time range of dat and final_holdout_test

tibble("data_set" = c("dat", "final_holdout_test"),
       "Min_date" = c(min(dat$date), min(final_holdout_test$date)),
       "Max_date" = c(max(dat$date), max(final_holdout_test$date))
       )
```

```
## # A tibble: 2 x 3
##   data_set           Min_date   Max_date
##   <chr>              <date>     <date>
## 1 dat                2020-12-05 2024-03-15
## 2 final_holdout_test 2024-03-16 2024-05-15
```

From above, we now know that the date range of `dat` and `final_holdout_test` does not overlap. Therefore, we cannot directly use `date` column to put in our model. However, there may be time effect of Month or Day in the data. Now we will explore them.

We start by checking the week day effect (Monday, Tuesday, …).

```r
# Derive a plot to depict distributions in different week days

dat %>% mutate(wday = factor(wday(date))) %>%
  ggplot(aes(x = wday, y = orders, group = wday)) +
  geom_boxplot()
```



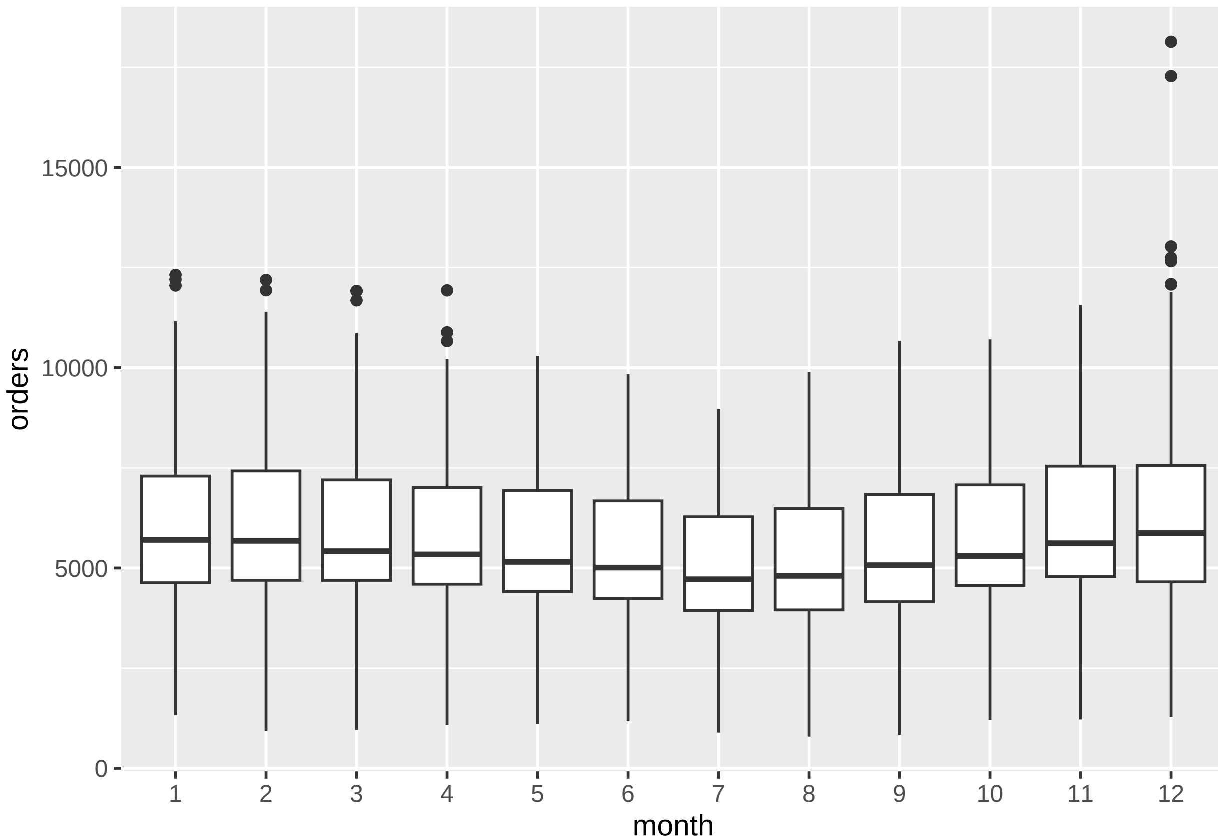
The boxplot reveals slightly differences in orders on workdays, with a higher average on Saturday and Monday. It is necessary to include this effect in the analysis since on the business side it is significant and intepretable, since people normally would order differently on weekdays and weekends.

So, we will create two data sets called `dat_n` and `final_holdout_test_n` to include the wday column (and other columns we want to add later):

```r
# Create a factorized week day parameter for both sets

dat_n <- dat %>% mutate(wday = factor(wday(date)))
final_holdout_test_n <- final_holdout_test %>%
  mutate(wday = factor(wday(date)))
```

Also, the month may effect the order number. This plot confirms this effect:

```r
# Derive a plot to depict distributions in different months

dat %>% mutate(month = factor(month(date))) %>%
  ggplot(aes(x = month, y = orders, group = month)) +
  geom_boxplot()
```

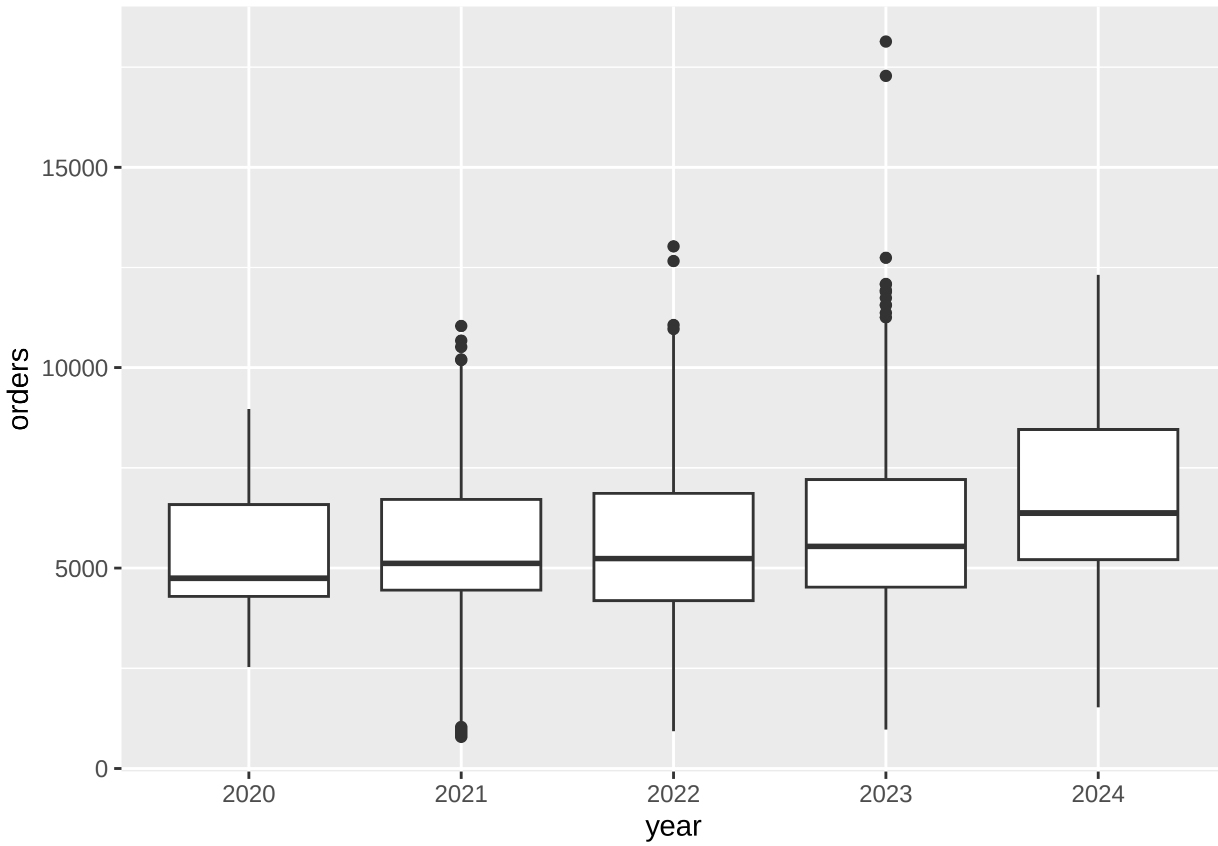


Generally, we can see the orders decrease in the month of 6,7,8, which is maybe due to the hotter whether causing the rotting of groceries delivered or users' stronger intention to go out and shopping by themselves. We will also add month into our new data sets:

```
# Create a factorized month parameter for both sets

dat_n <- dat_n %>% mutate(month = factor(month(date)))
final_holdout_test_n <- final_holdout_test_n %>%
  mutate(month = factor(month(date)))
```

Finally, it is necessary to include `year` in the model. That's because the `year` variable marks the growth of the platform:

```
# Derive a plot to depict distributions in different years

dat %>% mutate(year = factor(year(date))) %>%
  ggplot(aes(x = year, y = orders, group = year)) +
  geom_boxplot()
```

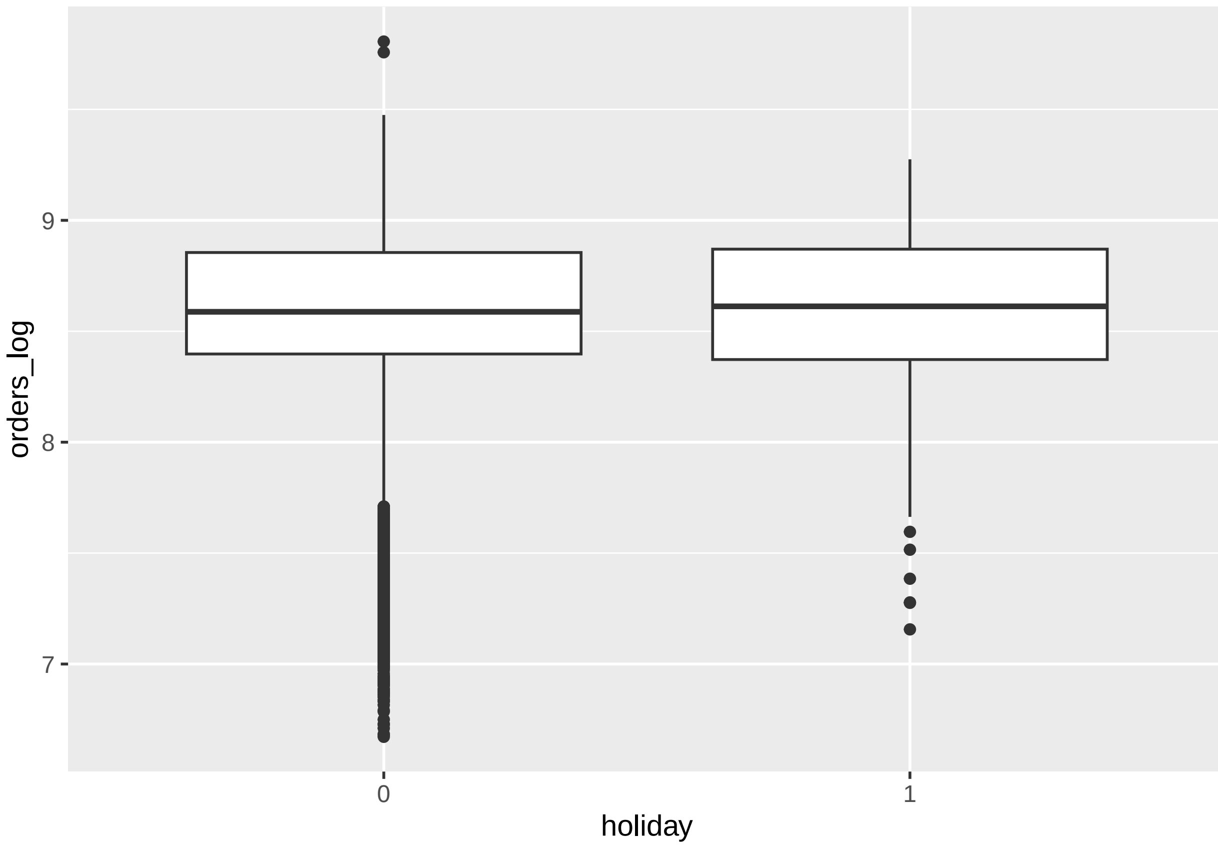


So, we will include `year` as a numeric value in the model (because the number value can help the model identify the annual growth:

```
# Create a year parameter for both sets

dat_n <- dat_n %>% mutate(year = year(date))
final_holdout_test_n <- final_holdout_test_n %>%
  mutate(year = year(date))
```

Holidays The data also contains holiday-related information. `holiday` is to identify whether the date is holiday, and `holiday_name` provides the specific name for the holiday. We first use this box plot to identify the different performance in holidays and common days. Note that is this plot, we log-transformed the order number to mitigate the effect of extreme values:

```
# Derive a plot to depict distributions in normal days and holidays

dat %>% mutate(holiday = factor(holiday),
               orders_log = log(orders)) %>%
  ggplot(aes(x = holiday, y = orders_log, group = holiday)) +
  geom_boxplot()
```



From this graph it is hard to tell if there is obvious difference between the distributions in normal days and holidays. Maybe the effect from `holiday` is mixed. To test our hypothesis, we next explore the order numbers in different specific holidays. There are totally 24 different kinds of holidays:

```
# List all existent holiday names

unique(dat$holiday_name)
```

```
##  [1] NA
##  [2] "Christmas Eve"
##  [3] "2nd Christmas Day"
##  [4] "New Years Day"
##  [5] "International womens day"
##  [6] "Good Friday"
##  [7] "Easter Monday"
##  [8] "Labour Day"
##  [9] "Den osvobozeni"
## [10] "Cyrila a Metodej"
## [11] "Jan Hus"
## [12] "Den ceske statnosti"
## [13] "Den vzniku samostatneho ceskoslovenskeho statu"
## [14] "Den boje za svobodu a demokracii"
## [15] "Peace Festival in Augsburg"
## [16] "Reformation Day"
## [17] "Memorial Day of the Republic"
## [18] "Memorial Day for the Victims of the Communist Dictatorships"
## [19] "Memorial Day for the Victims of the Holocaust"
## [20] "National Defense Day"
```

```
## [21] "Day of National Unity"
## [22] "Independent Hungary Day"
## [23] "Memorial Day for the Martyrs of Arad"
## [24] "1848 Revolution Memorial Day (Extra holiday)"
## [25] "All Saints' Day Holiday"
```

However, after checking, we soon find that `holiday` and `holiday_names` are not entirely correspondent.

There are some `holiday_names` with `holiday = 0` (nominal "holidays"):

```r
# List all named holidays without actual days off

dat %>% filter(holiday == 0 & !is.na(holiday_name)) %>%
  distinct(holiday_name)
```

```
## # A tibble: 8 x 1
##   holiday_name
##   <chr>
## 1 International womens day
## 2 Memorial Day of the Republic
## 3 Memorial Day for the Victims of the Communist Dictatorships
## 4 Memorial Day for the Victims of the Holocaust
## 5 National Defense Day
## 6 Day of National Unity
## 7 Independent Hungary Day
## 8 Memorial Day for the Martyrs of Arad
```

There are also rows when `holiday == 1` but without a `holiday_name` (unnamed holidays):

```r
# List all holidays without names

dat %>% filter(holiday == 1 & is.na(holiday_name)) %>%
  distinct(date)
```

```
## # A tibble: 6 x 1
##   date
##   <date>
## 1 2021-04-03
## 2 2021-04-04
## 3 2022-04-16
## 4 2022-04-17
## 5 2023-04-08
## 6 2023-04-09
```

For the **nominal holidays,** we can apply ANOVA method to determine whether it is significant. ANOVA (analysis of variance) is a degenerate form of linear regression, which can help determine whether a categorical variable has important effect to a numeric variable. Here, we first generate a dummy variable to mark all the days on nominal holidays:

```r
# Create an 'nominal holiday' identifier

dat_nom <- dat %>% filter(holiday == 0) %>%
  mutate(nominal_holiday = factor(ifelse(is.na(holiday_name), 1, 0)))
```

Then, we can apply ANOVA using the existent function `aov()`:

```r
# Use ANOVA to test the relevance of 'nominal holiday'

anova_result <- aov(dat_nom$orders ~ dat_nom$nominal_holiday)
summary(anova_result)
```

```
##                           Df    Sum Sq Mean Sq F value Pr(>F)
## dat_nom$nominal_holiday    1 1.869e+06 1868858   0.391  0.532
## Residuals               7138 3.410e+10 4777005
```

The ANOVA analysis gives the p-value of 0.532, which is not significant, meaning `nominal_holiday` will not be much useful if included in the model. Thus, we will drop the analysis of nominal holidays.

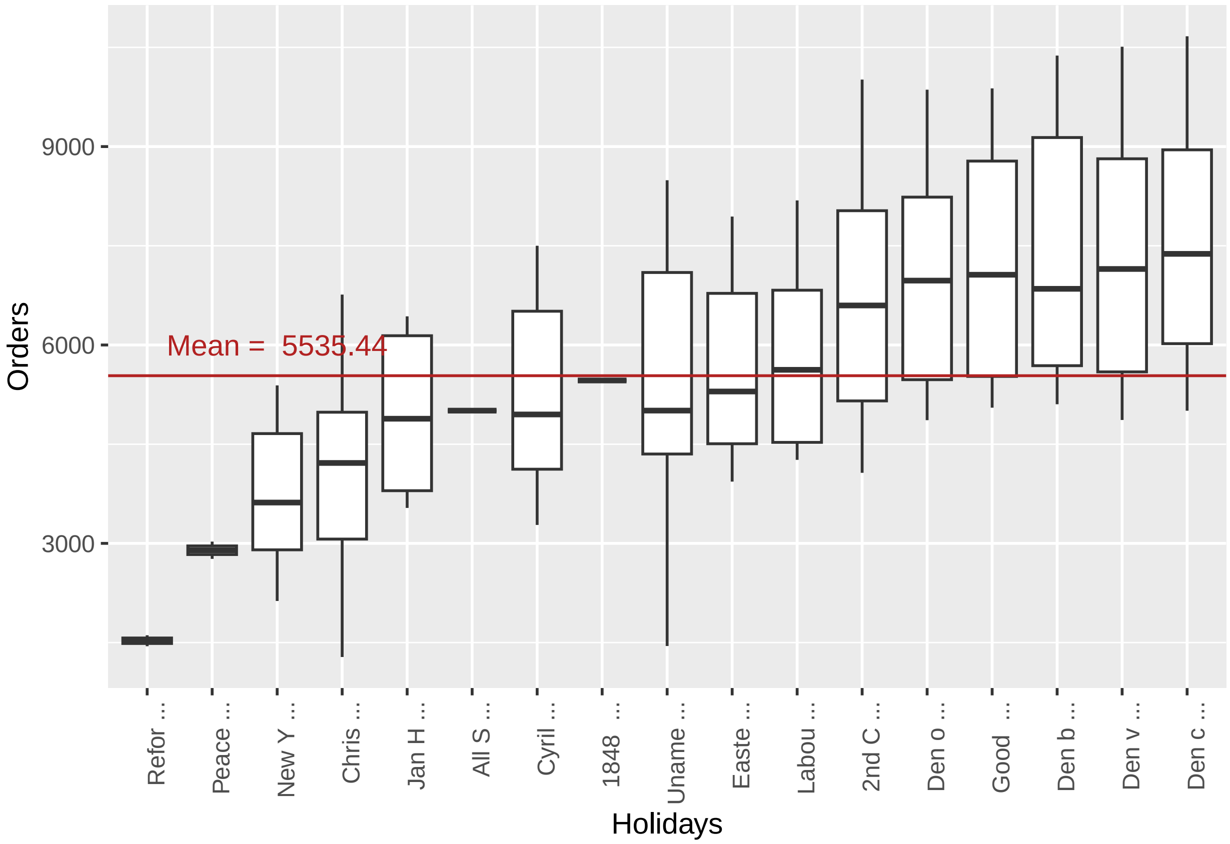
For the **unnamed holidays,** we note that these holidays almost all land on April. We can then hypothesize that these unnamed holidays are homogeneous. So here we will name these left ones `Unamed holiday`.

```r
# Name the 'unamed holidays'

dat_n <- dat_n %>% mutate(holiday_name =
                            ifelse(holiday == 1 & is.na(holiday_name),
                                   "Unamed holiday",
                                   holiday_name
                                   )
                            )

final_holdout_test_n <- final_holdout_test_n %>% mutate(holiday_name =
                            ifelse(holiday == 1 & is.na(holiday_name),
                                   "Unamed holiday",
                                   holiday_name
                                   )
                            )
```

Now, for all the holidays, we derive a plot to observe its distribution:

```r
# first extract the list of nominal holidays
nominal_holidays <- dat %>% filter(holiday == 0 &
                                     !is.na(holiday_name)) %>%
  distinct(holiday_name) %>%
  .$holiday_name

# Also calculate the average order number
avg_orders <- mean(dat$orders)

# # Derive a plot to depict distributions in different holidays
dat_n %>% filter(!is.na(holiday_name) &
                 !holiday_name %in% nominal_holidays) %>% # filter out nominal holidays
  mutate(holiday_abb = paste(str_sub(holiday_name, 1, 5), '...')) %>%
  # Create abbreviations for holidays for better labeling
  ggplot() +
  geom_boxplot(aes(x = reorder(holiday_abb, orders, FUN = mean), # reorder the holidays
                   y = orders)) +
  geom_hline(aes(yintercept = avg_orders), col = "firebrick") +
  # add a line to highlight the average orders
  annotate(
```

```
    geom = "text", x = 3, y = 6000,
    label = paste("Mean = ", round(mean(avg_orders), 2)),
    col = "firebrick") + # give text annotations
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(
    x = "Holidays",
    y = "Orders"
  )
```



From this boxplot, we can see that festivals have different impact on the order number. Some holidays lead to excessive order volume, while some do the opposite. That's why the aggregated effect of holidays is mixed. Here, we tag the top five holiday periods with higher average as `holiday_high`, and the last 4 holiday periods with lower average as `holiday_low`.

```
# Create a table to show the average in different holidays

holiday_avg <- dat_n %>% filter(!is.na(holiday_name) &
                  !holiday_name %in% nominal_holidays) %>%
  group_by(holiday_name) %>%
  summarise(avg_orders = mean(orders)) %>%
  arrange(desc(avg_orders))
```

Here is the holidays of high demand:

```
holiday_high_list <- holiday_avg$holiday_name[1:5] # Extract the list of high demand holidays
print(holiday_high_list)
```

```
## [1] "Den ceske statnosti"
## [2] "Den vzniku samostatneho ceskoslovenskeho statu"
```

```
## [3] "Den boje za svobodu a demokracii"
## [4] "Good Friday"
## [5] "Den osvobozeni"
```

And here is the holidays with low demand:

```r
l <- length(holiday_avg$holiday_name)
holiday_low_list <- holiday_avg$holiday_name[(l-3):l] # Extract the list of low demand holidays
print(holiday_low_list)
```

```
## [1] "Christmas Eve"                "New Years Day"
## [3] "Peace Festival in Augsburg" "Reformation Day"
```

Now, we will add 2 dummy variables to mark high(low) demand holidays:

```r
# Create high(low) demand holiday identifier for both sets

dat_n <- dat_n %>% mutate(
  holiday_high = factor(ifelse(
    holiday_name %in% holiday_high_list, 1, 0
  )),
  holiday_low = factor(ifelse(
    holiday_name %in% holiday_low_list, 1, 0
  ))
)

final_holdout_test_n <- final_holdout_test_n %>% mutate(
  holiday_high = factor(ifelse(
    holiday_name %in% holiday_high_list, 1, 0
  )),
  holiday_low = factor(ifelse(
    holiday_name %in% holiday_low_list, 1, 0
  ))
)
```

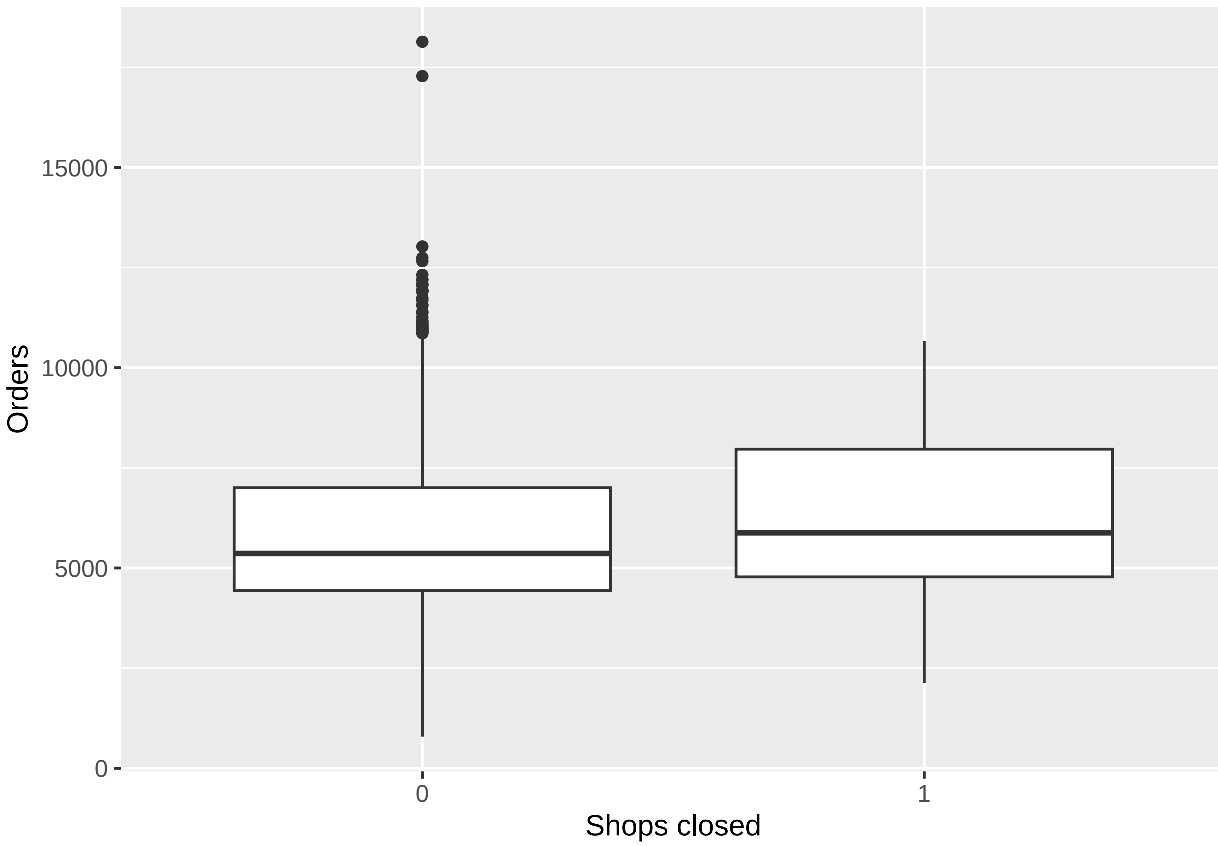
Shops Closed From Kaggle, we know that the it is a variable to show whether the date is a public holiday when large part of shops close. And a quick check tells us it is a dummy variable:

```r
unique(dat$shops_closed) # see the unique values of shops_closed
```

```
## [1] 0 1
```

Besides, this box plot tells us the difference in these two categories:

```r
# Derive a plot to depict distributions in shops_closed days or not

dat %>% mutate(shops_closed = factor(shops_closed)) %>%
  ggplot(aes(x = shops_closed, y = orders)) +
  geom_boxplot() +
  labs(
    x = "Shops closed",
    y = "Orders"
  )
```



From the box plot, it is hard to tell the difference. However, applying the ANOVA method, the relationship between `orders` and `shops_closed` is significant:

```
# Use ANOVA to test the relevance of shops_closed

anova_result <- aov(dat$orders ~ factor(dat$shops_closed))
summary(anova_result)
```

```
##                            Df    Sum Sq  Mean Sq F value  Pr(>F)
## factor(dat$shops_closed)    1 4.240e+07 42401274    8.91 0.00285 **
## Residuals                7338 3.492e+10  4758964
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So we will include the factorized `shops_closed` in our model:

```
# Factorize shops_closed in both sets

dat_n <- dat_n %>% mutate(shops_closed = factor(shops_closed))
final_holdout_test_n <- final_holdout_test_n %>%
  mutate(shops_closed = factor(shops_closed))
```

School holidays For the remaining `winter_school_holidays` and `school_holidays`, we first confirm that they are also dummy variables:

```
# List all the unique values of (winter_)school_holidays

print(unique(dat$winter_school_holidays))
```

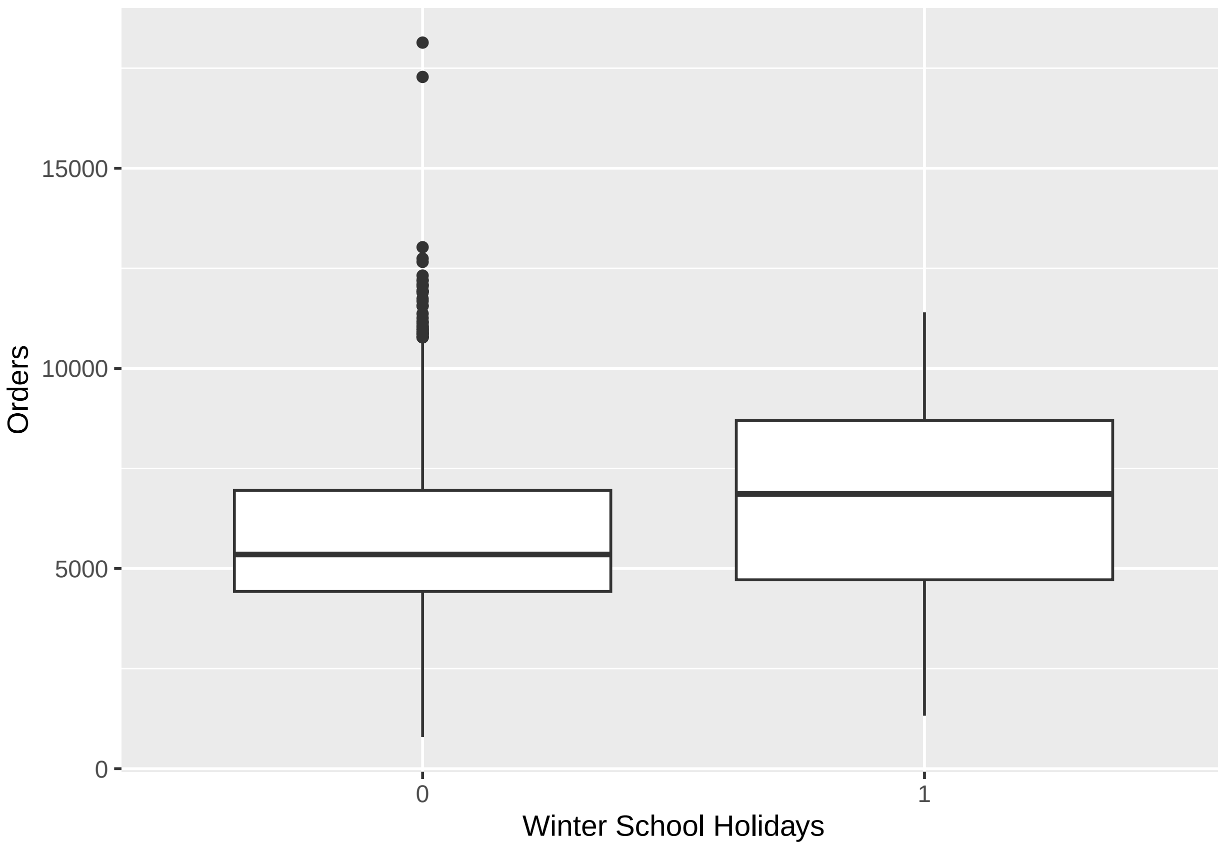
```
## [1] 0 1
```

```r
print(unique(dat$school_holidays))
```

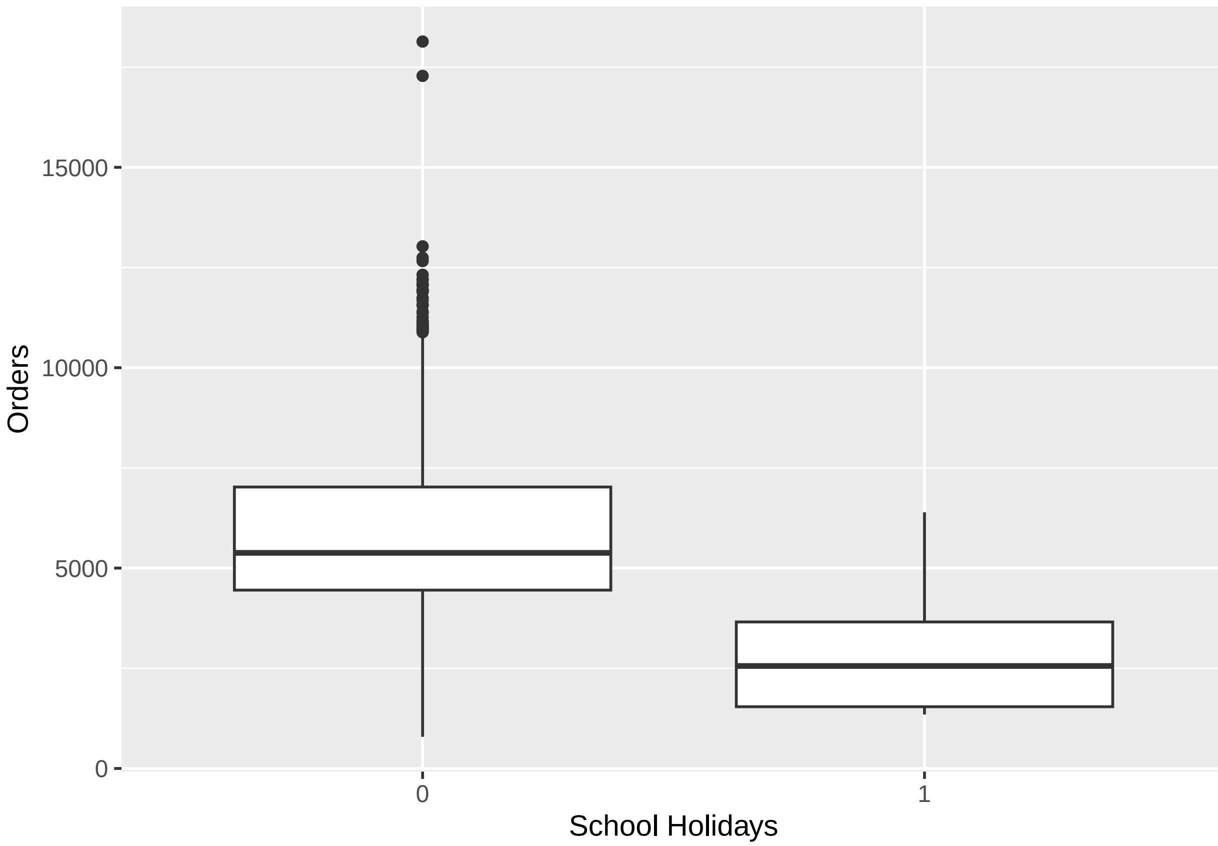
```
## [1] 0 1
```

We can derive box plots to check their relevance to the order number:

```r
# Derive a plot to depict distributions in (winter_)school_holidays or not

dat %>% mutate(winter_school_holidays = factor(winter_school_holidays)) %>%
  ggplot(aes(x = winter_school_holidays, y = orders)) +
  geom_boxplot() +
  labs(
    x = "Winter School Holidays",
    y = "Orders"
  )
```




```r
dat %>% mutate(school_holidays = factor(school_holidays)) %>%
  ggplot(aes(x = school_holidays, y = orders)) +
  geom_boxplot() +
  labs(
    x = "School Holidays",
    y = "Orders"
  )
```



From the plots, it can be identified that the `school_holidays` has negative effect on the orders, while the `winter_school_holidays` has slight positive effect. Thus, we will also include these dummy variables in our model after factorization:

```
# Factorize (winter_)school_holidays in both sets

dat_n <- dat_n %>% mutate(school_holidays = factor(school_holidays))
final_holdout_test_n <- final_holdout_test_n %>%
  mutate(school_holidays = factor(school_holidays))

dat_n <- dat_n %>% mutate(winter_school_holidays = factor(winter_school_holidays))
final_holdout_test_n <- final_holdout_test_n %>%
  mutate(winter_school_holidays = factor(winter_school_holidays))
```

Indicators only in `dat`

As discussed above, there are many indicators from `dat` missing in the `final_holdout_test`. They are listed here:

```
# List all the columns that in dat but not in final_holdout_test

tibble('Col_names' = names(dat)) %>%
  filter(!Col_names %in% names(final_holdout_test))
```

```
## # A tibble: 10 x 1
##    Col_names
##    <chr>
##  1 orders
```

```
##   2 shutdown
##   3 mini_shutdown
##   4 blackout
##   5 mov_change
##   6 frankfurt_shutdown
##   7 precipitation
##   8 snow
##   9 user_activity_1
## 10 user_activity_2
```

Here we briefly discuss how to process these variables:

`shutdown` marks the incident of warehouse shutdown due to operation problems. In the `dat` set, only one entry has `shutdown=1`:

```
# Explore the occurance of specific variables

dat %>% filter(shutdown == 1) %>% nrow()
```

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

So it lacks universality in our data, meaning it is acceptable to ignore them in the model. The same applies to `mini_shutdown`, `blackout`, and `frankfurt_shutdown` :

```
dat %>% filter(mini_shutdown == 1) %>% nrow()
```

```
## [1] 4
```

```
dat %>% filter(blackout == 1) %>% nrow()
```

```
## [1] 7
```

```
dat %>% filter(frankfurt_shutdown == 1) %>% nrow()
```

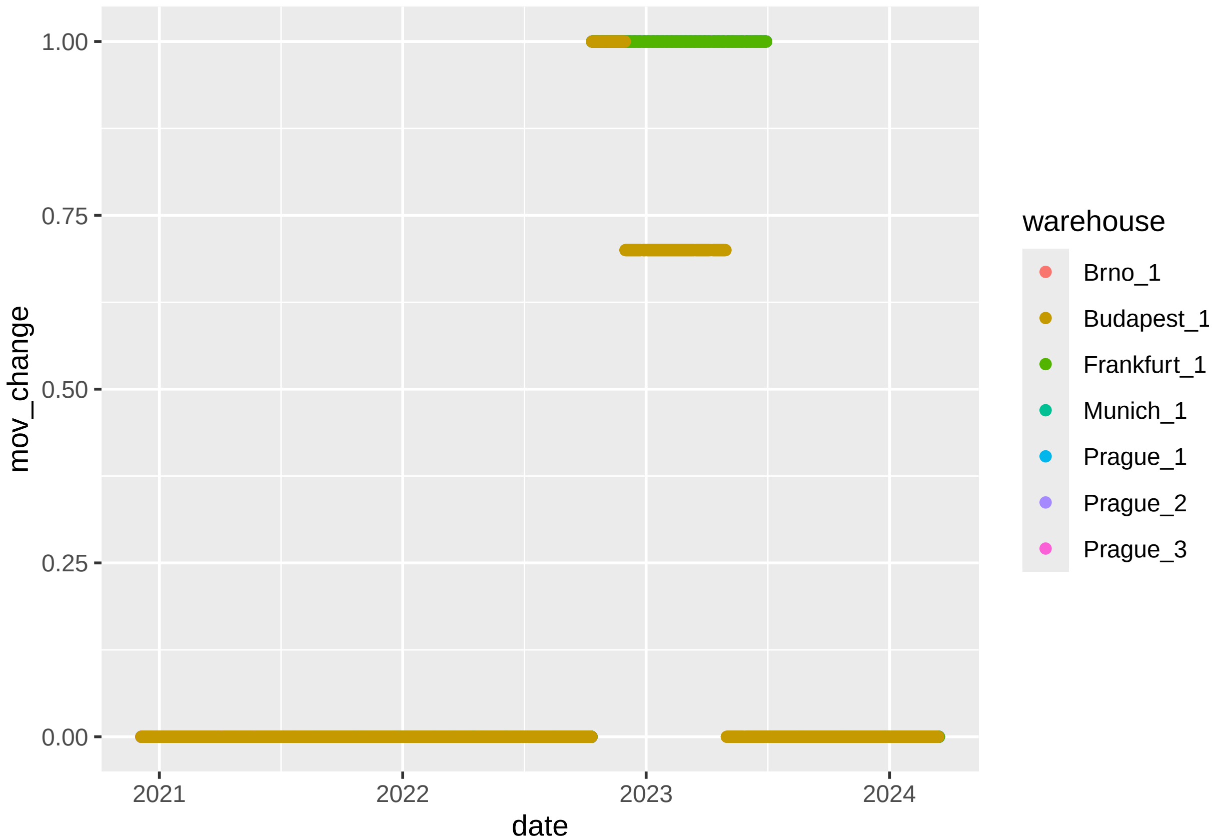
```
## [1] 2
```

Then come two weather-related indicators: `snow` and `precipitation`. The issue with these variables is that it is hard to predict them at the granularity of days without further information. Thus they will also be abandoned in `final_holdout_test`.

What's left are three indicators relating to the users' activity: `mov_change`, `user_activity_1`, and `user_activity_2`.

First, let's check the distribution of the `mov_change`. This indicator identifies a change in the minimum order value, which often indicates potential change in customer behavior. The following plot draws its distribution:

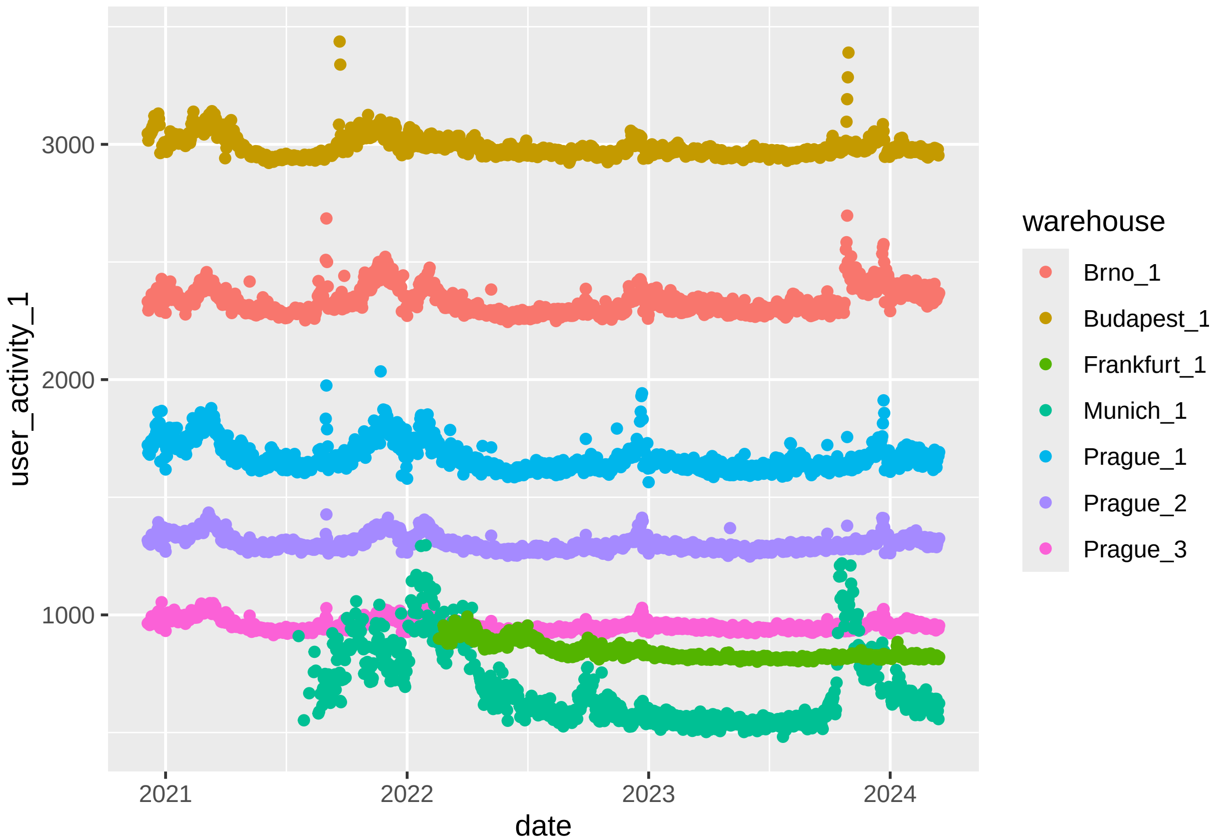
```
# Distribution of mov_change

dat %>% ggplot(aes(x = date, y = mov_change, colour = warehouse)) +
  geom_point()
```



As shown, the period when `mov_change != 1` concentrates on the end of 2022 and start of 2023, without regularity. Since the period of `final_holdout_test` is from March 2024 to May 2024, we can not ensure the value of `mov_change` at that time. Thus, it will not be included in the model.

Then, `user_activity_1` and `user_activity_2` are numeric values to describe the 'user activity on the web site'. Let's first explore the distribution of `user_activity_1`:

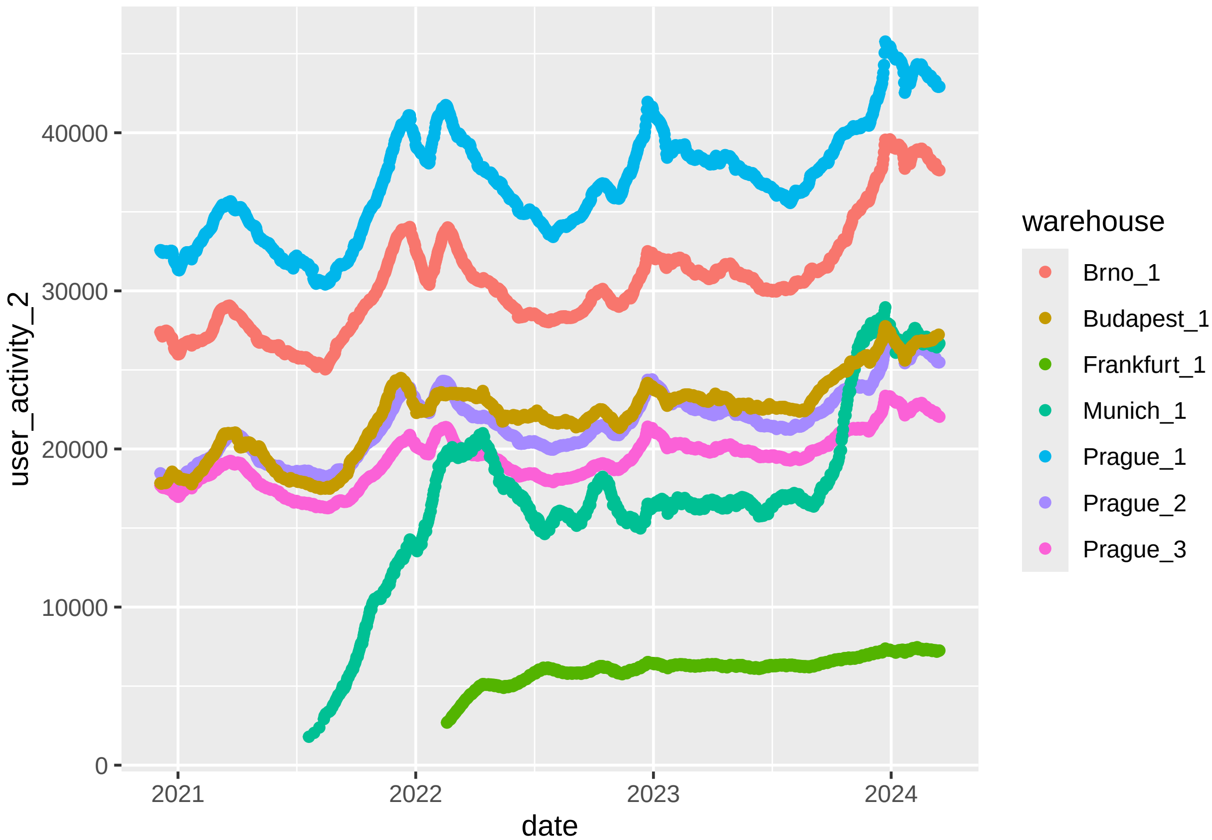
```
# Distribution of user_activity_1

dat %>% ggplot(aes(x = date, y = user_activity_1, colour = warehouse)) +
  geom_point()
```



The plot above shows that the `user_activity_1` is highly periodical, basically mainly depends on the `year`, `month`, and `warehouse`. And since we have already included `year`, `month`, `warehouse` in our model, it is unnecessary to add this variable whose characteristics has been captured by existing indicators.

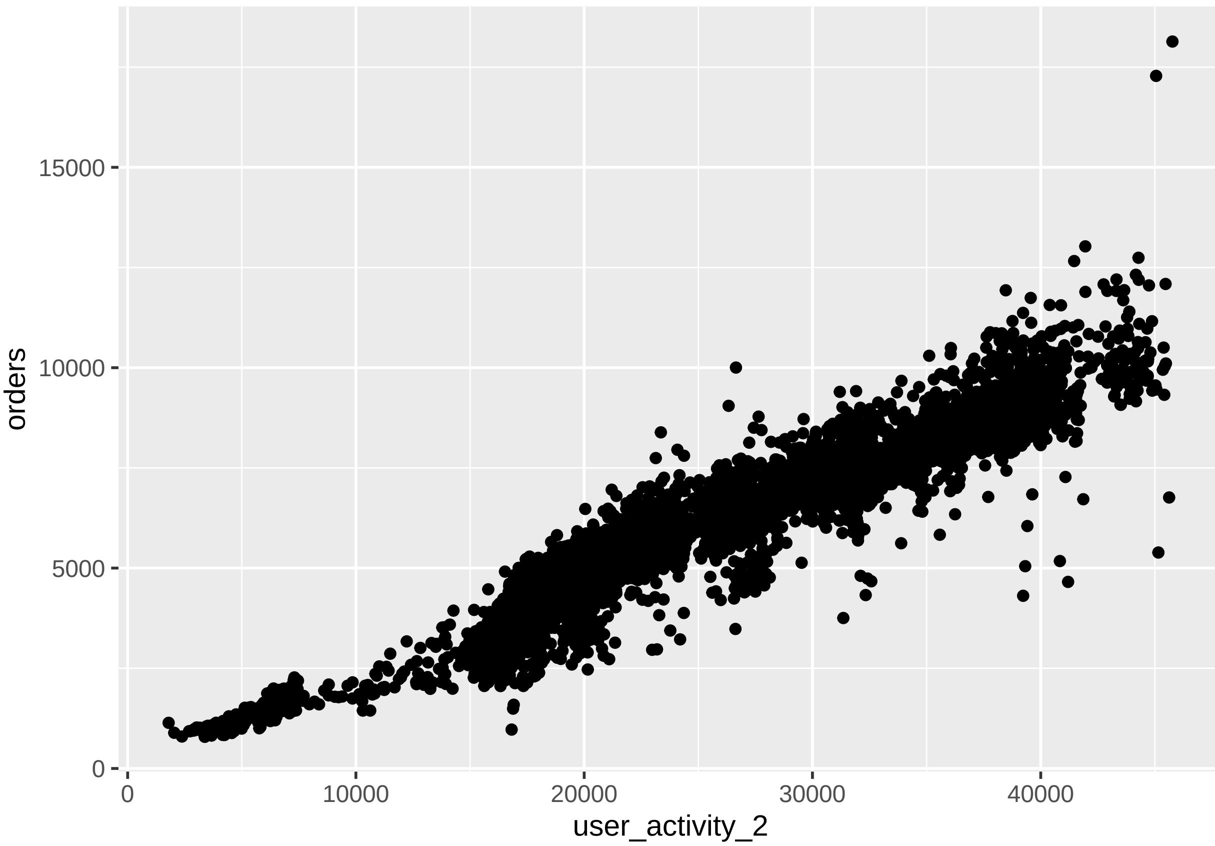
Moving on, the `user_activity_2` is distributed in the way shown below:

```r
# Distribution of user_activity_2

dat %>% ggplot(aes(x = date, y = user_activity_2, colour = warehouse)) +
  geom_point()
```



Further exploratory analysis shows it is very highly correlates with `orders`:

```
# Derive a plot to depict relevance between user_activity_2 and orders

dat %>% ggplot(aes(x = user_activity_2, y = orders)) +
  geom_point()
```



Thus, we will try to predict the `user_activity_2` using the the parameters that include the characteristics of the dates: `year`, `month`, `wday`, `holiday`; and `warehouse`. However, we will not execute the prediction in this part. Since we are not sure the stability and performance of our prediction on `user_activity_2`. We will compare the models with and without the predicted `user_activity_2` in our later analysis.

Abnormality

During our discussion of the `warehouse` parameter, we notice 2 abnormalities in the record of `Prague_1`:

```
# Extract the abnormal entries

dat_n %>% filter(warehouse == 'Prague_1') %>%
  arrange(desc(orders)) %>%
  slice(1:2)
```

```
## # A tibble: 2 x 23
##   warehouse date       orders holiday_name holiday shutdown mini_shutdown
##   <chr>     <date>      <dbl> <chr>          <dbl>    <dbl>         <dbl>
## 1 Prague_1  2023-12-23  18139 <NA>               0        0             0
## 2 Prague_1  2023-12-22  17282 <NA>               0        0             0
## # i 16 more variables: shops_closed <fct>, winter_school_holidays <fct>,
## #   school_holidays <fct>, blackout <dbl>, mov_change <dbl>,
## #   frankfurt_shutdown <dbl>, precipitation <dbl>, snow <dbl>,
## #   user_activity_1 <dbl>, user_activity_2 <dbl>, id <chr>, wday <fct>,
## #   month <fct>, year <dbl>, holiday_high <fct>, holiday_low <fct>
```

These 2 entries happened on the weekend before the Christmas. However, the data period we will predict is from March 2024 to May 2024. So here it is ok to remove these records to improve the stability our model.

```r
# Drop the abnormal entries

dat_n <- dat_n %>% filter(id != "Prague_1_2023-12-23" &
                          id != "Prague_1_2023-12-22")
```

Summary

After analysis, here are the list of the indicators we will finally include in our model:

1. `warehouse`
2. `wday`
3. `month`
4. `year`
5. `holiday_high`
6. `holiday_low`
7. `shops_closed`
8. `school_holidays`
9. `winter_school_holidays`
10. `user_activity_2` (tentative)

So we only keep these relevant variables in the `dat_f` and `final_holdout_test_f` test (along with `orders`):

```r
# Keep the indicators we need in the final set

dat_f <- dat_n %>% select(warehouse, wday, month, year, holiday_high, holiday_low,
                          shops_closed, school_holidays, winter_school_holidays, orders)

final_holdout_test_f <- final_holdout_test_n %>% select(warehouse, wday, month, year,
      holiday_high, holiday_low, shops_closed, school_holidays, winter_school_holidays)
```

Also, we will remove the lines with NA:

```r
# remove NA lines

dat_f <- na.omit(dat_f)
```

Creating Training Set and Test Set

To choose the best model, we need to further partition our know data set, `dat` into the training set `train` and test set `test`. Later, we will train our models on `train` and compare their performance on `test`. To ensure the reproducibility, `set.seed` is used.

```r
# Randomly create training set and test set

set.seed(1) # To ensure the code and results are reproducible
test_indices <- createDataPartition(y = dat_f$orders, times = 1, p = 0.1, list = FALSE)

test <- dat_f[test_indices,]
train <- dat_f[-test_indices,]
```

Methods

In this part, we plan to apply kNN and Random Forest algorithm to build our predictions:

kNN

We start by using kNN model. In kNN, we set k as the parameter to tune, with cross-validation:

```
# Model 1: kNN without predicted user_activity_2

#set cross validation methods for all ML algorithms
control <- trainControl(method = "cv", number = 10, p = .9)
# train kNN algorithm
train_knn <- train(orders ~ ., method = "knn",
                      data = train,
                      tuneGrid = data.frame(k = seq(5, 20)),
                      trControl = control)
train_knn
```

```
## k-Nearest Neighbors
##
## 6602 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5941, 5943, 5941, 5942, 5941, 5942, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5  1002.2242  0.8476638  742.8309
##    6  1002.5988  0.8486607  745.2908
##    7  1003.5383  0.8489374  747.5167
##    8  1001.0211  0.8501249  747.5594
##    9   989.8230  0.8549248  743.6689
##   10   985.9210  0.8566062  742.7436
##   11   980.5089  0.8591733  741.0229
##   12   976.8893  0.8617406  741.0175
##   13   980.5933  0.8620999  743.9498
##   14   983.1032  0.8626629  746.1906
##   15   986.4808  0.8629406  749.2622
##   16   992.5508  0.8633003  754.3621
##   17  1002.0141  0.8634889  760.4541
##   18  1006.3432  0.8649443  763.1324
##   19  1012.9329  0.8666646  767.2032
##   20  1016.4167  0.8687526  769.2154
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 12.
```

The algorithm chose $k = 12$ as the best parameter. And the model is stored in `train_knn`.

Then, we can predicted the order numbers in `test`:

```
y_hat_knn <- predict(train_knn, test) # predict order numbers in test set
```

Given the challenge evaluates the submission result by **Mean Absolute Percentage Error (MAPE),** we create a function to calculate it. The function of MAPE is:

$$\text{MAPE} = 100\frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

```
# define a function to calculate MAPE

MAPE <- function(forecast, actual){
  100 * sum(abs((actual - forecast) / actual)) / length(forecast)
}
```

Then, we can get the MAPE of our first kNN model as **23.7045.**

```
# calculate MAPE of Model 1

mape_knn <- MAPE(y_hat_knn, test$orders)
mape_knn
```

```
## [1] 23.7045
```

With predicted `user_activity_2`

As discussed above, we are not sure if it will be better to include the predicted user_activity_2 in our model. Here we will try to build it to see its performance:

We first need to build new data sets to include `user_activity_2`:

```
# Create data sets if we want to predict user_activity_2 first

dat_f_2 <- dat_n %>% select(warehouse, wday, month, year, holiday_high, holiday_low,
                            shops_closed, school_holidays, winter_school_holidays, orders, user_activity

dat_f_2 <- na.omit(dat_f_2)

set.seed(1) # To ensure the code and results are reproducible
test_indices_2 <- createDataPartition(y = dat_f_2$orders, times = 1, p = 0.1, list = FALSE)

test_2 <- dat_f_2[test_indices,]
train_2 <- dat_f_2[-test_indices,]
```

Then, we the kNN model would give predictions on `user_activity_2`:

```
# Model 2: kNN with predicted user_activity_2

# First predict user_activity_2
train_usract_knn <- train(user_activity_2 ~ year + month + wday + holiday_high +
                            holiday_low + warehouse,
                      method = "knn",
                      data = train_2,
                      tuneGrid = data.frame(k = seq(5, 20)),
                      trControl = control)
train_usract_knn
```

```
## k-Nearest Neighbors
##
## 6602 samples
##    6 predictor
##
## No pre-processing
```

```
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5942, 5942, 5941, 5942, 5941, 5942, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3804.249  0.8800639  2812.018
##    6  3803.852  0.8801623  2813.502
##    7  3791.718  0.8813431  2811.835
##    8  3754.289  0.8846046  2797.163
##    9  3732.216  0.8865915  2790.667
##   10  3706.680  0.8887306  2780.016
##   11  3648.653  0.8938502  2756.171
##   12  3640.458  0.8954075  2758.421
##   13  3649.735  0.8957950  2767.443
##   14  3680.600  0.8949821  2789.600
##   15  3699.946  0.8947678  2800.776
##   16  3732.219  0.8944004  2822.497
##   17  3759.339  0.8951092  2846.210
##   18  3798.141  0.8970625  2881.420
##   19  3820.304  0.8994671  2902.468
##   20  3851.416  0.9027636  2930.723
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 12.
```

```
# put the predicted user_activity in the test set
user_activity_2_hat_knn <- predict(train_usract_knn, test)
```

Next, we include the predicted column into the `test` set:

```
test_2_knn <- test %>% mutate(user_activity_2 = user_activity_2_hat_knn)
```

Now that the data sets are ready, we repeat the steps in the kNN without predicted user activity to generate the model:

```
# Then build a model to predict order number
train_knn_2 <- train(orders ~ ., method = "knn",
                     data = train_2,
                     tuneGrid = data.frame(k = seq(60, 70)),
                    # the range of the tune grid is changed according to results from multiple trials
                     trControl = control)
train_knn_2
```

```
## k-Nearest Neighbors
##
## 6602 samples
##   10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5942, 5942, 5942, 5942, 5943, 5941, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##   60  572.9165  0.9305156  394.2323
```

```
##   61  572.8276  0.9305352  394.2202
##   62  572.9635  0.9305079  394.2530
##   63  572.7435  0.9305623  393.9779
##   64  572.6089  0.9305963  393.8251
##   65  572.4968  0.9306202  393.8491
##   66  572.4131  0.9306417  393.7969
##   67  572.2333  0.9306809  393.6390
##   68  572.1742  0.9306984  393.6569
##   69  572.1606  0.9307000  393.5962
##   70  572.2700  0.9306753  393.6112
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 69.
```

And this model gives the MAPE of 22.79, indicating a better performance than the first model:

```r
# calculate MAPE of Model 2

y_hat_knn_2 <- predict(train_knn_2, test_2_knn)

mape_knn_2 <- MAPE(y_hat_knn_2, test_2_knn$orders)
mape_knn_2
```

```
## [1] 22.79027
```

Random Forest

Now, we try to use Random Forest algorithm to do our prediction. `predFixed` and `minNode` are parameters:

```r
# Model 3: Random Forest without predicted user_activity_2

train_rf <- train(orders ~ ., method = "Rborist",
                  data = train,
                  tuneGrid = data.frame(predFixed = 2, minNode = seq(1:10)),
                  trControl = control)
train_rf
```

```
## Random Forest
##
## 6602 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5942, 5942, 5942, 5942, 5942, 5942, ...
## Resampling results across tuning parameters:
##
##   minNode  RMSE      Rsquared     MAE
##    1       2170.276  0.009373258  1693.146
##    2       2170.371  0.009447870  1693.257
##    3       2170.332  0.009380202  1693.186
##    4       2170.245  0.009544924  1693.074
##    5       2170.305  0.009616111  1693.210
##    6       2170.272  0.009484323  1693.114
##    7       2170.248  0.009378306  1693.072
##    8       2170.253  0.009502994  1693.074
```

```
##    9       2170.135  0.009481046  1692.988
##   10       2170.190  0.009447135  1693.076
##
## Tuning parameter 'predFixed' was held constant at a value of 2
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were predFixed = 2 and minNode = 9.
```

We use our model to predict the orders in the test set:

```
# calculate MAPE of Model 3

y_hat_rf <- predict(train_rf, test)

mape_rf <- MAPE(y_hat_rf, test$orders)
mape_rf
```

```
## [1] 51.49416
```

Random Forest gives MAPE of 51.49.

With predicted `user_activity_2`

Would it better if we first predict `user_activity_2` first? Here we follow the similar strategy as kNN:

```
# Model 4: Random Forest with predicted user_activity_2

# First predict user_activity_2
train_usract_rf <- train(user_activity_2 ~ year + month + wday + holiday_high + holiday_low + warehouse
                      method = "Rborist",
                      data = train_2,
                      tuneGrid = data.frame(predFixed = 2, minNode = seq(1:10)),
                      trControl = control)
train_usract_rf
```

```
## Random Forest
##
## 6602 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5942, 5941, 5942, 5942, 5942, 5942, ...
## Resampling results across tuning parameters:
##
##   minNode  RMSE      Rsquared    MAE
##    1       8918.575  0.01055367  6959.737
##    2       8918.321  0.01058561  6959.276
##    3       8918.256  0.01055264  6959.752
##    4       8918.616  0.01058630  6960.630
##    5       8918.477  0.01060764  6960.292
##    6       8918.927  0.01060637  6960.365
##    7       8918.124  0.01059506  6959.904
##    8       8918.801  0.01055741  6960.222
##    9       8918.120  0.01055514  6959.330
##   10       8918.679  0.01056019  6960.571
##
## Tuning parameter 'predFixed' was held constant at a value of 2
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were predFixed = 2 and minNode = 9.
```

```r
# put the predicted user_activity in the test set
user_activity_2_hat_rf <- predict(train_usract_rf, test)
test_2_rf <- test %>% mutate(user_activity_2 = user_activity_2_hat_rf)
```

Then, we execute random forest again to predict `orders`:

```r
# Then build a model to predict order number

train_rf_2 <- train(orders ~ ., method = "Rborist",
                      data = train_2,
                      tuneGrid = data.frame(predFixed = 2, minNode = seq(1:10)),
                     # the range of the tune grid is changed according to results from multiple trials
                      trControl = control)
train_rf_2
```

```
## Random Forest
##
## 6602 samples
##   10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5942, 5942, 5941, 5943, 5941, 5942, ...
## Resampling results across tuning parameters:
##
##   minNode  RMSE      Rsquared   MAE
##    1       2077.543  0.6191732  1601.481
##    2       2074.732  0.6274956  1599.226
##    3       2083.524  0.6257992  1607.892
##    4       2077.174  0.6258638  1601.652
##    5       2075.888  0.6187831  1599.883
##    6       2078.831  0.6251112  1603.037
##    7       2082.199  0.6284335  1606.648
##    8       2084.650  0.6206854  1608.592
##    9       2069.352  0.6320332  1594.197
##   10       2078.180  0.6266219  1602.528
##
## Tuning parameter 'predFixed' was held constant at a value of 2
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were predFixed = 2 and minNode = 9.
```

```r
# calculate MAPE of Model 4

y_hat_rf_2 <- predict(train_rf_2, test_2_rf)

mape_rf_2 <- MAPE(y_hat_rf_2, test_2_rf$orders)
mape_rf_2
```

```
## [1] 50.68889
```

The final result gives the MAPE of 50.87.

Results

After the modelling process, we compare the performance of the models we have:

```
# List all the models and their MAPEs

tibble(
  "Model" = c("kNN", "kNN (with predicted user activity)",
              "Random forest", "Random forest (with predicted user activity)"),
  "MAPE" = c(mape_knn, mape_knn_2, mape_rf, mape_rf_2)
)
```

```
## # A tibble: 4 x 2
##   Model                                        MAPE
##   <chr>                                       <dbl>
## 1 kNN                                          23.7
## 2 kNN (with predicted user activity)           22.8
## 3 Random forest                                51.5
## 4 Random forest (with predicted user activity)  50.7
```

In the above table, the kNN model with predicted `user_activity_2` gives the least MAPE. So we will apply that method in the `final_holdout_test`.

```
# Finally train the kNN model on dat

# First predict user_activity_2
train_usract_knn <- train(user_activity_2 ~ year + month + wday + holiday_high + holiday_low + warehous
                      method = "knn",
                      data = dat_f_2,
                      tuneGrid = data.frame(k = seq(5, 20)),
                      trControl = control)
train_usract_knn
```

```
## k-Nearest Neighbors
##
## 7338 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 6604, 6603, 6604, 6605, 6605, 6605, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3788.991  0.8841959  2804.220
##    6  3794.020  0.8838076  2807.059
##    7  3797.126  0.8834868  2810.031
##    8  3787.686  0.8844715  2808.685
##    9  3766.957  0.8863561  2801.402
##   10  3732.711  0.8892318  2788.348
##   11  3693.994  0.8921887  2772.200
##   12  3662.906  0.8950399  2761.220
##   13  3643.161  0.8968459  2756.516
##   14  3648.034  0.8970614  2761.452
##   15  3661.850  0.8967785  2770.888
##   16  3687.747  0.8955813  2787.954
```

```
##   17  3708.708  0.8948884  2801.888
##   18  3733.953  0.8953377  2827.028
##   19  3758.106  0.8964832  2850.088
##   20  3785.926  0.8978243  2872.209
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

Then, we join the predicted `user_activity_2` in to the test set:

```
# put the predicted user_activity in the final_holdout_test
user_activity_2_hat_knn <- predict(train_usract_knn, final_holdout_test_f)
```

```
final_holdout_test_f <- final_holdout_test_f %>% mutate(user_activity_2 = user_activity_2_hat_knn)
```

Finally, we train the model and join the predicted `orders`:

```
# Then build a model to predict order number

train_knn_f <- train(orders ~ ., method = "knn",
                      data = dat_f_2,
                      tuneGrid = data.frame(k = seq(80, 90)),
                     # the range of the tune grid is changed according to results after multiple trials
                      trControl = control)
train_knn_f
```

```
## k-Nearest Neighbors
##
## 7338 samples
##   10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 6603, 6605, 6605, 6605, 6604, 6604, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##   80  567.0331  0.9319871  389.8571
##   81  566.9907  0.9319954  389.8536
##   82  566.9615  0.9320015  389.8737
##   83  566.9104  0.9320162  389.8418
##   84  566.7225  0.9320570  389.6765
##   85  566.5823  0.9320913  389.6121
##   86  566.6761  0.9320748  389.6748
##   87  566.7448  0.9320529  389.7386
##   88  566.7371  0.9320577  389.6395
##   89  566.9109  0.9320175  389.6803
##   90  566.9249  0.9320153  389.6485
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 85.
```

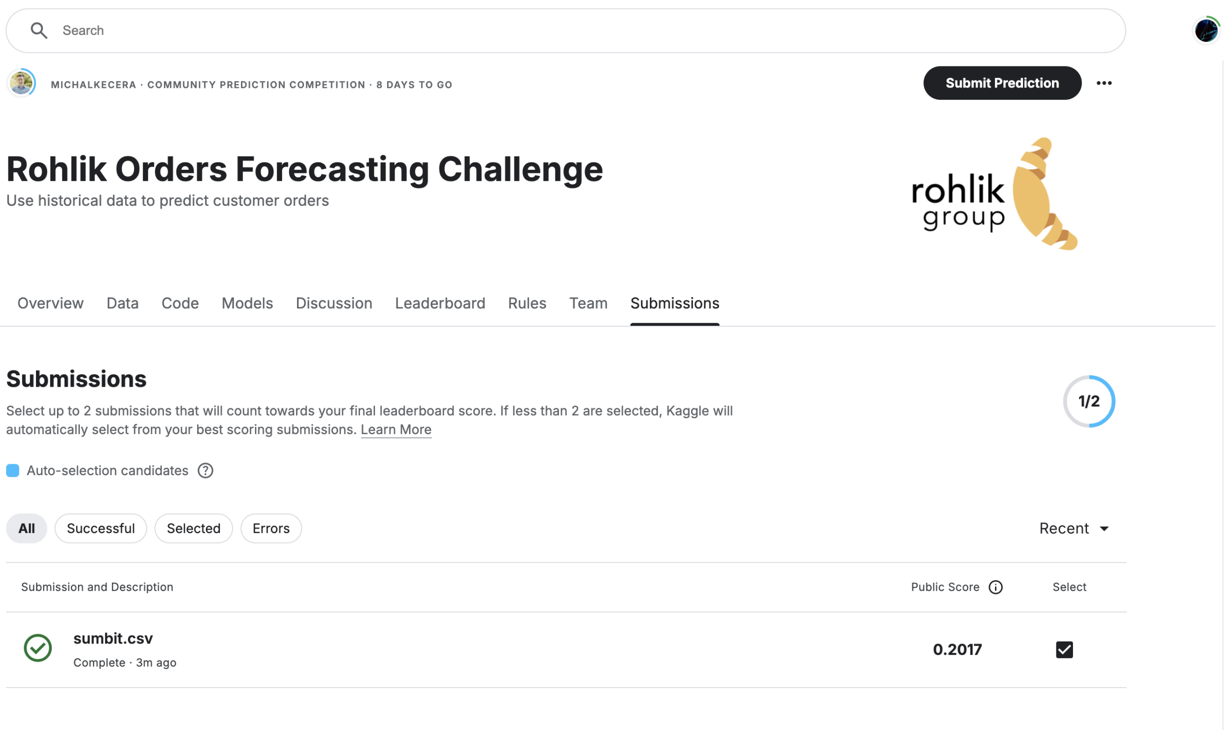
```
y_hat_knn_f <- predict(train_knn_f, final_holdout_test_f)
```

After we join the predicted result to the `final_holdout_test` set, we are ready to submit our prediction to Kaggle:

```r
# Create a file to submit following the guidance from Kaggle

submit_file <- final_holdout_test %>% mutate(orders = y_hat_knn_f) %>%
  select(id, orders)
```

```r
# Write the data frame to a CSV file
write.csv(submit_file, file = "./sumbit.csv", row.names = FALSE)
```

After the submission, the platform gives out the publice score of **0.2017** (MAPE of 20.17).



Conclusion

This report aims to predict future order number for an online grocery platform. From the history order record, we first explored data characteristics and transformed our data, then designed kNN and Random Forest models to forecast order outcome. The final MAPE of our model is 20.17.

The result is still not accurate enough. In the future, more measures to increase the model performance are needed, including creating more parameters to make full use of the data we have, and applying more advanced algorithms etc.

References

MichalKecera. (2024). Rohlik Orders Forecasting Challenge. Kaggle. https://kaggle.com/competitions/rohlik-orders-forecasting-challenge

Wang Zhiqiang. (2020). How to reorder x-axis based on y-axis values in R ggplot2. Stack Overflow. https://stackoverflow.com/questions/63165943/how-to-reorder-x-axis-based-on-y-axis-values-in-r-ggplot2