Data Wrangling Assessment Task 2: Creating and pre-processing synthetic data

Timm Rahrt [redacted]

Setup

Please download and insert all libraries:

```
# Please run the necessary libraries, un-commend to install
#install.packages("tidyverse")
#install.packages("ds4psy")
#install.packages("randomNames")
library(tidyverse) #covers ggplot2, dplyr,tidyr,readr,readxl, lubridate, stringr, e
tc
library(ds4psy)
library(randomNames)
library(magrittr)
library(Hmisc)
set.seed(123)
```

1.0 Data Description

Creation of multiple synthetic datasets

This section covers the creation of five synthetic datasets, all representing *fictive* business insight data for multiple Subway branches located in metro Sydney. After every dataset creation, we add outliers and missing data `NA` to create more realistic datasets. Some of the variables below will contain normal distributed data, which has been modified to be slightly right-skewed to add imperfections. To simplify the creation of right-skewed-data, I created a function `right_skewed_data` :

```
# Function to create slightly right-skewed data with 5 outlier
right_skewed_data <- function(n,mean,sd,outlier_min,outlier_max){
  rnorm(n, mean=mean, sd=sd) %>%
  { . * (1+ runif(n, min=0, max=0.3)) } %>%
  c(., runif(5, min=outlier_min, max=outlier_max))}

example_data_test_result <- right_skewed_data(95, 65, 5,10,100)
summary(example_data_test_result)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   14.92   68.42   73.88   73.25   79.97   94.85
```

The above function takes five inputs. `n` represents how many values the function has to create, `mean` represents the mean, `sd` the standard deviation (measures the spread of the data around the mean), `outlier_min` the minimum outlier and `outlier_max` the maximum outlier. `example_data_test_result` gives an impression on how the function works, we have 100 datapoints with a mean of 65, where 95 values are normal distributed with a slight right-skewness of the factor 1.3. The vector contains 5 outlier which sit around the `outlier_min` of 10 and `outlier_max` of 100.

1.1 First synthetic dataset `onl_orders`

The below code chunk will create our first synthetic dataset `onl_orders` , representing variables for online orders our Subway branches in Sydney metro received during a period of seven days, from the 1st of April 2024 to the 7th of April 2024. The variables are as follows:

- order_num: Unique order identifier
- username_id: Unique user identifier
- date: Date of order
- amount: Amount of order
- transaction_id: Unique transaction identifier
- item_number: Unique item number
- collected: Whether order was collected or not
- store_id: Unique store identifier
- weekend: Whether order was placed on the weekend or not, displays "Yes" when order was placed on the 6th of 7th of April 2024 using `ifelse`

```
# Create a rows between 50 and 100
rows1 <- sample(50:100,1)
# Synthetic dataframe for all online orders
onl_orders <- data.frame(order_num = sample(as.character(10500:10600), rows1, repla
ce=FALSE),
                         username_id = sample(as.character(1:10600), rows1, replace
=TRUE),
                         date = sample_date(from = as.Date('2024-04-01'), to = as.D
ate('2024-04-07'), size = rows1, replace = TRUE),
                         amount = right_skewed_data((rows1-5),30,7.5,0.15,500),
                         transaction_id = sample(as.character(21000:(21000+rows1)),
rows1, replace = FALSE),
                         item_number = sample(as.factor(1:42), rows1, replace = TRU
E),
                         collected = sample(c("TRUE","FALSE"),rows1,replace = TRUE,
prob = c(0.9,0.1)),
                         store_id = paste("AU",sample(c(50:150),rows1,replace = FAL
SE), sep = ""))
```

```
onl_orders$amount <- round(onl_orders$amount,2)
# Add another variable `weekend` - integration standalone as the variable date need
s to be created
onl_orders$weekend <- ifelse(onl_orders$date %in% as.Date(c('2024-04-06','2024-04-0
7')),"YES","NO")
# Create three missing values in random rows of onl_orders$amount
onl_orders[sample(1:80,3), "amount"] <- NA
head(onl_orders,2)
```

	order_n...	username_id	date	amo...	transaction_id	item_number	collected	st
	<chr>	<chr>	<date>	<dbl>	<chr>	<fct>	<chr>	<c
1	10600	10264	2024-04-07	48.67	21032	33	TRUE	AU
2	10522	6083	2024-04-02	NA	21029	3	TRUE	AU

2 rows

1.2 Second synthethic dataframe `items`

Next, we create the second dataframe `items` containing all information about the products Subway sells. It contains variables containing information about the product ID, product description, product price and product specific customer rating:

- item_number: Unique item identifier
- item_description: Description of the item
- item_price: Price of the item
- rubric: Category of item
- customer_rating: Customer rating of item

```
# Synthetic dataframe for all Subway items
# Define all products Subway sells
sandwiches <- c("Black Forest Ham","Buffalo Chicken","B.L.T","Cold Cut Combo","Gril
led Chicken","Italien B.M.T","Meatball Marinara","Oven-Roasted Turkey","Oven-Roaste
d Turkey & Ham","Pizza Sub","Roast Beef","Spicy Italian","Steak & Cheese","Subway C
lub","Subway Melt","Sweet Onion Chicken Teriyaki","Tuna","Turkey Breast","Veggie De
lite")
sides <- c("Fries","Sweet Potato Fried","Caramel Cookie","Chocolate Chip Cookie","O
atmeal Raisin Cookie","White Chip Macadamia Nut Cookie","Dark Chocolate Cherry Cook
ie","Broccoli Soup","Chicken Noodle Soup","Broccoli Cheddar Soup","Cheese Flatizz
a","Pepperoni Flatizza","Spicy Italian Flatizza","Veggie Flatizza")
drinks <- c("Coffee","Fountain Drinks","Dasani Water","X2 All Natural Energy","Gato
rade","Honest Kids","Bootle Beverage","Low Fat Milk","Juice")
```

```
# Define all prices
sandwich_price <- c(7,8,8.3,8.55,8.75,9.75)
side_price <- c(3.25,3.45,4,4.35,4.5,4.95)
drinks_price <- c(2,2.25,2.5,3,3.5,3.75)
item_prices <- c(sample(sandwich_price,length(sandwiches),replace=TRUE),
                 sample(side_price, length(sides), replace=TRUE),
                 sample(drinks, length(drinks), replace=TRUE))
# Define item description
all_items <- c(sandwiches, sides, drinks)

# Defining the necessary vectors
items <- data.frame(item_number = as.factor(1:length(all_items)),
                    item_description = all_items,
                    item_price = item_prices,
                    rubric = rep(c("Sandwich","Side","Drink"), times = c(length(san
dwiches),length(sides),length(drinks))),
                    customer_rating = rnorm(length(all_items), mean = 4, sd = 0.5))
items$customer_rating <- round(items$customer_rating ,2)


# Outliers and missing values
items$customer_rating[sample(1:nrow(items), 5)] <- rnorm(5, mean = 1, sd=0.5)
items[sample(1:length(all_items),3),"customer_rating"] <- NA
head(items,2)
```

item_number <fct>	item_description <chr>	item_price <chr>	rubric <chr>	customer_rating <dbl>
1 1	Black Forest Ham	9.75	Sandwich	4.97
2 2	Buffalo Chicken	8.75	Sandwich	4.40

2 rows

1.3 Third synthethic dataframe `user`

Our third dataframe `user` contains information about the customer. We add missing data as before and purposely format `b_day` as `POSIXct` instead of `Date` for the type conversion.

- username_id: Unique user identifier
- first_name: Customers First name
- last_name: Customers Last name
- b_day: Customers Birthday
- email: Customers Email
- postal_code: Customers Postal Code
- member: Membership status
- total_orders: Total orders placed
- last_activity: Customers last activity

```
# Synthetic dataframe for all user data
# Get all unique usernames you find in onl_orders
username <- unique(onl_orders$username_id)
# Generate random first and last name, email provider and postal code of SYD
first_name <- randomNames(length(username),which.names = "first")
last_name <- randomNames(length(username), which.names = "last")
email_provider = c("gmail.com","yahoo.com","gmx.com","icloud.com","me.com","outloo
k.com","hotmail.com","zoho.com","live.com","fastmail.com","hushmail.com","tutanota.
com","usa.com","safe-mail.net","excite.com","bigstring.com","inbox.com","mail.ru","
runbox.com","iname.com")
postal_code <- c(2000,2001,2006,2010,2011,2015,2020,2021,2022,2026,2031,2037,2042,2
043,2044,2050,2060,2061,2065,2067,2068,2070,2071,2074,2085,2086,2090,2093,2095,210
0,2110,2111,2112,2127,2129,2130,2134,2140,2150,2166)

user <- data.frame(username_id = username,
                    first_name = first_name,
                   last_name = last_name,
                   b_day = sample(seq(as.POSIXct("1970/01/01"), as.POSIXct("2008/0
7/25"), by="day"), length(username), replace = TRUE),
                   email = paste(last_name, sample(email_provider, length(usernam
e), replace = TRUE), sep = "@"),
                   postal_code = sample(postal_code, length(username), replace = TR
UE ),
                   member = sample(c(TRUE, FALSE), length(username), replace = TRU
E),
                   total_orders = round(rnorm(length(username),mean = 15, sd = 4)),
                   last_activity = sample(seq(as.Date("2024/04/01"), as.Date("2024/
04/07"), by = "day"), length(username), replace= TRUE))

# Outliers in total_orders and add missing data NA
user$total_orders[sample(1:nrow(user),5)] <- round(rnorm(5, mean=90, sd=8))
user[sample(1:length(all_items),6),"email"] <- NA
head(user,2)
```

	username_id	first_name	last_name	b_day	email	posta
	<chr>	<chr>	<chr>	<dttm>	<chr>	
1	10264	Isabella	Johnson-Scott	1980-03-31	Johnson-Scott@gmail.com	
2	6083	Sameera	Ramai	1970-12-18	Ramai@safe-mail.net	

2 rows | 1-8 of 10 columns

1.4 Fourth synthethic dataframe `stores`

Next, we create a dataframe containing the following information about the store. It will be crucial for analysing store performance and efficiency:

- store_id: Unique store identifier
- location: Store location

- revenue: Revenue generated by the store
- employees: Number employees
- manager_name: Name of store manager
- store_type: Type of Store

```
# Synthetic dataframe for all Subway_stores
# Extract all unique Store IDs
storeid <- sort(unique(onl_orders$store_id))
# Create the locations
location <- c("Sydney Central", "Redfern","University of Sydney","Darlinghurst","Po
tts Point","Alexandria","Sydney Domestic Airport","Sydney International Airport","P
addington","Bondi Junction","Bondi Beach","Randwick","Glebe","Newtown","Erskinevill
e","St Peters","Camperdown","North Sydney","Kirribilli","Crows Nest","Chatswood","W
illoughby","Lindfield","Killara","Turramurra","Belrose","Frenchs Forest","Mosman","
Manly","Fairlight","Brooksvale","Hunters Hill","Gladsville","Ryde","Sydney Olympic
Park","Sydney Markets","Summer Hill","Burwood","Homebush","Parramatta","Cabramatt
a")
# Create the revenue
revenue <- round(right_skewed_data(length(storeid)-5,mean = 200000, sd=20000,outlie
r_min = 15000, outlier_max = 400000), digits = 2)


# Create the synthetic dataframe
stores <- data.frame(store_id = storeid,
                     location = sample(location, length(storeid), replace = TRUE),
                     revenue = round(revenue, digits = 2),
                     employees = sample(4:20, length(storeid), replace = TRUE),
                     manager_name = randomNames(length(storeid), which.names = "las
t"),
                     store_type = as.factor(sample(c("express","dine-in","delivery-
only"), length(storeid),replace = TRUE)))

# Outliers and missing data
stores$revenue[sample(1:nrow(stores), 5)] <- round(rnorm(5, mean=395000, sd= 10), d
igits = 2)
stores$employees[sample(1:nrow(stores), 3)] <- NA
head(stores,2)
```

	store_id <chr>	location <chr>	revenue <dbl>	employees <int>	manager_name <chr>	store_type <fct>
1	AU103	Redfern	254057.1	11	al-Shahin	express
2	AU105	North Sydney	243706.1	4	Secord	dine-in

2 rows

1.5 Fifth synthetic dataframe `transactions`

The last dataframe contains information about the transactions the user did. Also, 40% of all users applied a discount which returns a 20% discount when, `ifelse` a discount exists and the amount is greater than 50 and 10% when the amount is below 50.(Rdocumentation.org, 2024)

- transaction_id: Unique transaction identifier
- payment_type: Type of payment used
- order_date: Transaction date
- amounts: Transaction amount, generated in amounts variable with above explained function
- discount_ref: Discount reference and amount applied
- cancelled: Shows if transaction was cancelled

```
# Create a synthetic dataset aboout the transactions
# Get transaction id from onl_orders
transactionid <- sort(unique(onl_orders$transaction_id))
payment_type <- c("creditcard","cash","applepay","googleplay","paypal","afterpay","
debitcard")
amounts <- round(right_skewed_data(length(transactionid)-5, 30, 7.5, 0.15,500), dig
its = 2)
# Create the variable transactions
transactions <- data.frame(transaction_id = transactionid,
                           payment_type = as.factor(sample(payment_type, length(tra
nsactionid), replace = TRUE)),
                           order_date = sample(seq(ymd_hm('2024-04-01 00:00'), ymd_
hm('2024-04-07 23:59'), by = "min"), length(transactionid)),
                           amounts = amounts)
# Add the discount_ref variable , 40% of the user apply one
transactions$discount_ref <- ifelse(
  runif(nrow(transactions)) <= 0.4,
  sapply(transactions$amounts, function(x) {
  discount_amount <- ifelse(x >= 50, round(x * 0.20, 2), round(x * 0.10, 2))
  paste0("ref-", formatC(sample(1000:9999,1), width = 4, flag = "0"), "$", formatC(
discount_amount, width = 5, flag = "0"))
  }),
  "ref-not_applied")
# Add the `cancelled` variable
transactions$cancelled <- sample(c(TRUE,FALSE), length(transactionid), replace = TR
UE, prob = c(0.05,0.95))

# Missing data since outliers are already in the amounts data
transactions$payment_type[sample(1:nrow(transactions),4)] <- NA
head(transactions,2)
```

	transaction_id <chr>	payment_type <fct>	order_date <dttm>	amou... <dbl>	discount_ref <chr>	canc
1	21000	paypal	2024-04-02 23:38:00	24.00	ref-4619$002.4	FA
2	21001	afterpay	2024-04-06 18:14:00	52.99	ref-not_applied	FA

2 rows

We used the `runif` function to create a random number between 0 and 1, if its 0.4 or below a discount will be applied as explained above.(Rdocumentation, 2024) `paste0` concatenates the strings together, `formatC` generates a random number *string* between 1000 and 9999, with a width of 4 digits. The variable `cancelled` has a 95% chance to display `FALSE` and 5% `TRUE` , the function `prob` controls the probability.(Rdocumentation, 2024)

Every synthetic dataset contains at least 3 missing values and 5 outliers in one of the numeric variable. Lastly, we introduce a *correlation* between some of the variables by adjusting the variables using `mutate` . The `log` function applies the natural logarithm on the numeric `item_price` variable. It reduces skewness and eliminates large price variations. We use it here to adjust the ratings based on the item price. Lastly, we convert `weekend` to a numeric variable with `ifelse` to check the correlation between amount and ordered on a weekend.

```
# Create a correlation between revenue and employees and customer_rating and item_p
rice
stores <- stores %>%
  mutate(revenue = revenue + (employees * 10000))
items <- items %>%
  mutate(customer_rating = customer_rating + log(as.numeric(item_price)))
onl_orders$weekend_numeric <- ifelse(onl_orders$weekend == "YES",1,0)
# Displaying the correlation
par(mfrow = c(1,2))
plot(stores$employees, stores$revenue,
     main = "Corr. Revenue and Employees",
     xlab = "Employees",
     ylab = "Revenue")
plot(items$item_price, items$customer_rating,
     main = "Corr. Customer_rat and Item_pr",
     xlab = "Item Price",
     ylab = "Customer Rating")
```

It looks like there is a moderate Correlation between Revenue and Employees and also a positive linear relationship of Customer Rating and Item Price. We call the `cor` correlation function to proof this Hypothesis and remove the variable from `subway_data` .

```
cor(stores$revenue, stores$employees, use = "complete.obs")
```

```
## [1] 0.4873733
```

```
cor(items$customer_rating, as.numeric(items$item_price), use = "complete.obs")
```

```
## [1] 0.4746862
```

```
cor(onl_orders$amount, onl_orders$weekend_numeric, use = "complete.obs")
```

```
## [1] -0.1688293
```

As expected, the correlation of Revenue and Employees and the correlation between Customer Rating and Item Price is moderate positive. We also see, that there is no correlation between the `amount` spent on the `weekend` .

```
onl_orders$weekend_numeric <- NULL
```

2.0 Merge - Creating of merged dataframe `subway_data`

The below code chunk will merge the synthetic datasets to create a comprehensive one that includes multiple data types and variables. We can use the newly created dataset to perform advanced analytics and check correlations between variables.

2.1 Merging `onl_order` and `user`

First, we merge `onl_orders` and `user` using a `left_join` on the variable `username_id` , returning all rows and variables of the left table `onl_orders` and the variables of the right table `user` matching with `username_id` . We save the output in a new variable `subway_data` and display three rows to see if we were successful.

```
# First we merge onl_orders and users
subway_data <- left_join(onl_orders, user, by = "username_id")
head(subway_data,3)
```

	order_n... <chr>	username_id <chr>	date <date>	amo... <dbl>	transaction_id <chr>	item_number <fct>	collected <chr>	st <c
1	10600	10264	2024-04-07	48.67	21032	33	TRUE	AU
2	10522	6083	2024-04-02	NA	21029	3	TRUE	AU
3	10513	7634	2024-04-01	37.30	21033	40	TRUE	AU

3 rows | 1-10 of 18 columns

The code outputs a dataframe of 17 columns.

2.2 Merging `subway_data` and `items`

Next, we use the same `left_join` to join the newly created dataframe with `items` . We display three rows again. Another four variables where added!(Tidyverse.org, 2024)

```
# Then we merge the dataset item_number
subway_data <- left_join(subway_data , items, by = "item_number")
head(subway_data,3)
```

	order_n... <chr>	username_id <chr>	date <date>	amo... <dbl>	transaction_id <chr>	item_number <fct>	collected <chr>	st <c
1	10600	10264	2024-04-07	48.67	21032	33	TRUE	AU

2	10522	6083	2024-04-02	NA	21029	3	TRUE	AU
3	10513	7634	2024-04-01	37.30	21033	40	TRUE	AU

3 rows | 1-10 of 22 columns

2.3 Merging `subway_data` and `stores`

We continue and `left_join` `subway_data` with `stores`. Again, we display three rows.

```
# Merge stores
subway_data %<>% left_join(stores %>% dplyr::select(store_id, location, revenue, em
ployees, manager_name, store_type), by = "store_id")
# Check result
head(subway_data,3)
```

	order_n... <chr>	username_id <chr>	date <date>	amo... <dbl>	transaction_id <chr>	item_number <fct>	collected <chr>	st <c
1	10600	10264	2024-04-07	48.67	21032	33	TRUE	AU
2	10522	6083	2024-04-02	NA	21029	3	TRUE	AU
3	10513	7634	2024-04-01	37.30	21033	40	TRUE	AU

3 rows | 1-10 of 27 columns

2.4 Merging `subway_data` and `transaction`

Almost done, one more time with the `transactions` dataframe and display the `tail` with the last three rows.

```
# Merge transactions
subway_data <- left_join(subway_data, transactions, by = "transaction_id")
tail(subway_data,3)
```

	order_n... <chr>	username_id <chr>	date <date>	amo... <dbl>	transaction_id <chr>	item_number <fct>	collected <chr>	s <
56	10501	9181	2024-04-04	127.02	21041	35	TRUE	A
57	10545	4052	2024-04-03	20.03	21004	33	TRUE	A
58	10512	10153	2024-04-01	317.46	21038	39	TRUE	A

3 rows | 1-10 of 32 columns

Finally, we display an unique dataframe specific variable to see if all of our mergings were successful.

```
# Check unique dataframe specific variables, everything works so far!
subway_data %>%
  dplyr::group_by(date) %>%
  dplyr::select(customer_rating, email, employees, cancelled) %>%
  head(3)
```

```
## Adding missing grouping variables: `date`
```

date <date>	customer_rating <dbl>	email <chr>	employees <int>	cancelled <lgl>
2024-04-07	5.740176	Johnson-Scott@gmail.com	8	FALSE
2024-04-02	6.857267	Ramai@safe-mail.net	8	FALSE
2024-04-01	NA	Rivera@hushmail.com	12	FALSE

3 rows

Since we piped the `select` function through `subway_data` and received the dataframe specific variable, we know that our merging was successful.(Rdocumentation, 2024)

3.0 Understand

This section inspects the complete dataframe `subway_data`. We use `str` to check all variables, their types, all rows and numbers and the first few observations. It's also a great way to see that the data tidy `Rule 1. and 2.` are fulfilled - every variable is a column and every observations a row.(Wickham & Grolemund, 2016)

```
# display the structure
str(subway_data)
```

```
## 'data.frame':    58 obs. of  31 variables:
##  $ order_num       : chr  "10600" "10522" "10513" "10505" ...
##  $ username_id     : chr  "10264" "6083" "7634" "2213" ...
##  $ date            : Date, format: "2024-04-07" "2024-04-02" ...
##  $ amount          : num  48.7 NA 37.3 20.6 33.7 ...
##  $ transaction_id  : chr  "21032" "21029" "21033" "21001" ...
##  $ item_number     : Factor w/ 42 levels "1","2","3","4",..: 33 3 40 2 1 9 7 13
29 20 ...
##  $ collected       : chr  "TRUE" "TRUE" "TRUE" "TRUE" ...
##  $ store_id        : chr  "AU81" "AU138" "AU60" "AU123" ...
##  $ weekend         : chr  "YES" "NO" "NO" "YES" ...
##  $ first_name      : chr  "Isabella" "Sameera" "Gabrielle" "Belicia" ...
##  $ last_name       : chr  "Johnson-Scott" "Ramai" "Rivera" "Heimann" ...
##  $ b_day           : POSIXct, format: "1980-03-31 00:00:00" "1970-12-18 00:00:0
0" ...
##  $ email           : chr  "Johnson-Scott@gmail.com" "Ramai@safe-mail.net" "River
a@hushmail.com" "Heimann@hushmail.com" ...
##  $ postal_code     : num  2110 2015 2110 2129 2022 ...
##  $ member          : logi  TRUE FALSE TRUE TRUE FALSE TRUE ...
##  $ total_orders    : num  12 18 22 18 11 18 20 12 15 91 ...
##  $ last_activity   : Date, format: "2024-04-03" "2024-04-06" ...
##  $ item_description: chr  "Veggie Flatizza" "B.L.T" "Bootle Beverage" "Buffalo C
hicken" ...
##  $ item_price      : chr  "4.35" "9.75" "Honest Kids" "8.75" ...
##  $ rubric          : chr  "Side" "Sandwich" "Drink" "Sandwich" ...
##  $ customer_rating : num  5.74 6.86 NA 6.57 7.25 ...
##  $ location        : chr  "Sydney Olympic Park" "Cabramatta" "Summer Hill" "Padd
ington" ...
##  $ revenue         : num  352039 475012 515007 424133 277578 ...
##  $ employees       : int  8 8 12 17 5 NA 19 5 11 15 ...
##  $ manager_name    : chr  "Carter" "Zuther" "Barlow" "Hathaway" ...
##  $ store_type      : Factor w/ 3 levels "delivery-only",..: 3 2 2 3 2 3 2 1 2 2
...
##  $ payment_type    : Factor w/ 7 levels "afterpay","applepay",..: 4 5 1 1 2 4 NA
1 3 2 ...
##  $ order_date      : POSIXct, format: "2024-04-07 19:41:00" "2024-04-01 05:15:0
0" ...
##  $ amounts         : num  35.9 19.3 45.7 53 23.9 ...
##  $ discount_ref    : chr  "ref-not_applied" "ref-not_applied" "ref-6214$04.57" "
ref-not_applied" ...
##  $ cancelled       : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

3.1 Converting data into their correct variable types

Since `str` displays all classes, we can see that we might want to change some of the data types in our merged dataset. We will convert the variable `collected` and `weekend` to factors with different labels later, but the above output shows that we should convert `postal_code` to a character variable, as we are not performing any quantitative analytics on this variable. `item_price` will be converted to a numeric class below and `b_day` to a date. We select all changed variables and display their type.

```
# Convert the variable types
subway_data$postal_code <- as.character(subway_data$postal_code)
subway_data$item_price <- as.numeric(subway_data$item_price)
subway_data$b_day <- as.Date(subway_data$b_day)
# Get a more concise list of variable types
subway_data %>%
  select(postal_code,item_number,rubric,collected,weekend) %>%
  sapply(typeof)
```

```
## postal_code item_number      rubric   collected     weekend
## "character"   "integer" "character" "character" "character"
```

3.2 Creating Factor variables and changing their levels

Since `subway_data` contains several variables that needs to be changed into a `factor`, we will also adjust their labeling with `levels` for a better understanding.(GeeksforGeeks, 2021) We `mutate` the variables to convert them into a `factor` with clearer `levels` and `labels`, as it helps for future data analysis of categorical data. Next, we display the amendments using an `anonymous function` within `sapply`, which iterates over each factor variables and siplays their name and levels. `invisible` is used to hide the output of the sapply, ensuring only factor variable names and levels are returned. (Rdocumentation, 2024)

```
# Label factor variables
subway_data %<>%
  mutate(payment_type = factor(payment_type, levels = c("creditcard","cash","applep
ay","googlepay","afterpay","debitcard")),
         store_type = factor(store_type, levels = c("express","dine-in","delivery-o
nly")),
         collected = factor(collected, levels = c("TRUE","FALSE"), labels = c("Coll
ected","Outstanding")),
         member = factor(member, levels = c(TRUE,FALSE),labels = c("Active","Inacti
ve")),
         weekend = factor(weekend, levels = c("NO","YES"), labels = c("Weekday","We
ekend")))

factors <- c("payment_type", "store_type", "collected", "member", "weekend")

factors <- c("payment_type", "store_type", "collected", "member", "weekend")

invisible(sapply(factors, function(var) {
  cat("\nFactor Variable:", var, "\nLevels:", levels(subway_data[[var]]), "\n")
}))
```

```
##
## Factor Variable: payment_type
## Levels: creditcard cash applepay googlepay afterpay debitcard
##
## Factor Variable: store_type
## Levels: express dine-in delivery-only
##
## Factor Variable: collected
## Levels: Collected Outstanding
##
## Factor Variable: member
## Levels: Active Inactive
##
## Factor Variable: weekend
## Levels: Weekday Weekend
```

We pipe `select_if(is.factor)` through `subway_data` , which will only return factor types in a subsetted dataframe. We only return the last three rows using `tail` .(Zach, 2022)

```
subway_data %>%
  select_if(is.factor) %>%
  tail(3)
```

	item_number <fct>	collected <fct>	weekend <fct>	member <fct>	store_type <fct>	payment_type <fct>
56	35	Collected	Weekday	Active	delivery-only	applepay
57	33	Collected	Weekday	Inactive	express	debitcard
58	39	Collected	Weekday	Active	delivery-only	NA

3 rows

4.0 Manipulate Data

Mutate `order_total`, `revenue_per_head` and `age` to `subway_data`

The below will mutate three new variables to our dataframe `subway_data` . First, we mutate `order_total` to the dataframe which displays the product of `amount` of an order and his `total_orders` . The variable displays his total he spent at subway, if all previous order had the same amount as this one. Next, we mutate `revenue_per_head` , which takes the `revenue` per branch and divides it by their `employees` . The output is a new variables displaying the efficiency of each branch. Lastly, we mutate `age` to `subway_data` , which displays the age of the customer. `format()` extracts the year `"%Y"` from dates as a numeric value and subtracts it from the year of `b_day` .(Rdocumentation, 2024) The result is saved as a numeric value in `age` . We pipe a `summarise` function through `subway_data` to show that our mutation was successful, as R is able to find all three new variables in our dataframe and performs a bit of summary statistics on them. `[var[var != 0], na.rm=TRUE]` ensures no `NULL` or `NA` values are used when summarising the minimum age, median revenue per head, `round()` by two digits after `.` , and maximum total of all-time orders done at Subway.(Zach, 2021)

```
# Add the total amount spent at all subways
subway_data %<>%
  mutate(order_total = amount * total_orders)
# Add the revenue per head count per subway branch
subway_data %<>%
  mutate(revenue_per_head = round(revenue / employees, digits = 2))
# Current age of the customer
subway_data %<>%
  mutate(age = as.numeric(format(Sys.Date(), "%Y")) - as.numeric(format(b_day, "%
Y")))

subway_data %>%
  summarise(minimum_age = min(age[age != 0], na.rm = TRUE),
            median_rev_per_head = round(median(revenue_per_head, na.rm = TRUE), dig
its = 2),
            max__order_total = round(max(order_total, na.rm = TRUE),digits=0))
```

minimum_age <dbl>	median_rev_per_head <dbl>	max__order_total <dbl>
16	28738	4444

1 row

The below filters through the newly created variables in `subway_data` , displaying all rows where age is greater than 50 `or` revenue per head greater than 55000 `or` the total order greater than 100. `select` displays all successfully mutated variables with their last name and branch location.

```
subway_data %>%
  filter(age > 80 |
         revenue_per_head > 55000 |
         total_orders > 100) %>%
  select(last_name, age, revenue_per_head, location, total_orders)
```

last_name <chr>	a... <dbl>	revenue_per_head <dbl>	location <chr>	total_orders <dbl>
Ramai	54	59376.48	Cabramatta	18
Baldwin	43	55515.60	Killara	11
Bridwell	33	57802.91	Potts Point	12
el-Abdelrahman	33	70926.52	North Sydney	79

Rocco	48	58104.61	Randwick	15
Anderson	27	59078.37	Camperdown	14

6 rows

5.1 Scan I

5.1.1 Locating and displaying the missing values `NA`

This chapter will fix all missing values of our dataset `subway_data` . First, we display if all values are as per the `3. Tidy Rule` standalone values and not e.g. lists.(Grolemund & Wickham, 2016) `vapply` will search through all variables of our dataframe and return `TRUE` , if lists are given and `FALSE` if not. After that, we display all variables where the function returned `TRUE` , the output is `0` names logical variables, our dataset is tidy!(Rpubs.com, 2024)

```
# Scan if values are standalone values
exist_lists <- vapply(subway_data, is.list, logical(1))
exist_lists[exist_lists == TRUE] # set to FALSE, displays all standalone variables
```

```
## named logical(0)
```

The below code will save all variables with `NA` in the variable `missing_values` . The next line will display just the variables with missing data (where missing data > 0).

```
# Scan for missing values in the variables
missing_values <- colSums(is.na(subway_data))
missing_values[missing_values > 0]
```

```
##           amount             email        item_price  customer_rating
##                3                 6                11               15
##          revenue         employees      payment_type      order_total
##                3                 3                24                3
## revenue_per_head
##                3
```

We can even check their position in the dataframe by calling the `which` function and its attribute `arr.ind = TRUE` , which returns a dataframe of two variables displaying the exact row and column of the `NA` values (Ethz.ch, 2024). We display the dataframe and proof this by indexing the exact position of the first row and column, the output should be `NA` .

```
subset <- subway_data %>% select(amount,email,item_price,customer_rating,revenue,em
ployees, payment_type, order_total, revenue_per_head)

na_locations <- which(is.na(subway_data), arr.ind = TRUE)
head(na_locations,3)
```

```
##      row col
## [1,]   2   4
## [2,]   6   4
## [3,]  17   4
```

```
subway_data[17,4]
```

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

5.1.2 Cleaning our dataset from missing values `NA`

Since we located all of our missing values, it's now time to clean or replace them. First, we apply the `sapply` function on our dataset and filter out all numeric variables with `is.numeric`, we save the result in `var_numeric`. Next, we call the `lapply` function to iterate through all numeric variables and replace the `NA` of the variable with the variables *mean* by using an *anonymous function* and `fun = mean` (Had.co.nz, 2024). We replace the missing values of our character variable `email` with `Unknown`, as we don't know the customers email address, but make use of the `mode` of the next character variable `payment_type` as we can assume there is a big chance the customers used the most used payment method for their order. We do this, by calling the same `impute` function but replacing *mean* with *mode* in `fun`. Lastly, we pipe `count` through our dataset and count all missing values in the affected variables.

```
# Fixing missing values with the `impute()` function
var_numeric <- sapply(subway_data, is.numeric)
subway_data[var_numeric] <- lapply(subway_data[var_numeric], function(x) impute(x,
fun = mean))

# Change all missing emails to `Unknown`
subway_data$email[is.na(subway_data$email)] <- "Unknown"
# Change payment_type to its mode
subway_data$payment_type <- impute(subway_data$payment_type, fun = mode)
# Check if it worked
which(is.na(subway_data), arr.ind = TRUE)
```

```
##      row col
```

As the output of `which(is.na(subway_data))` is empty rows and columns, no more missing values exist.

5.2 Scan II

After we fixed all missing data, we need to handle numeric outliers to ensure the dataset is free from anomalies that could influence the analysis.

5.2.1 Calculating the `z score` of `amount` and manually detecting outliers

We start by displaying the `z score` of our variable `amount` , which returns the distribution of the data by displaying the variables minimum, 1st quartile, median, mean, third quartile and maximum. `scores` from the package `outliers` used with `type="z"` computes the z-scores - how many standard deviations each value is from the mean (GeeksforGeeks, 2021).

```
library(MVN)
library(outliers)
# Detect univariate outliers
z.scores <- subway_data$amount %>%
  outliers::scores(type = "z")
# Summary
summary(z.scores)
```

```
##
##  3 values imputed to -1.497645e-16
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.61099 -0.33055 -0.20735  0.00000 -0.03283  5.69034
```

To manually calculate outliers, we save the 1st and third quartile in a variable, calculate `iqr <- q3-q1` , the lower and upper fence and see if the `min` and `max` are beyond those fences.

```
# Manually searching for outliers
q1 <- -0.317898
q3 <- -0.04303
iqr <- q3-q1
lower_fence <- q1 - 1.5 * iqr
upper_fence <- q3 + 1.5 * iqr
cat("Lower Border for Outlier - Upper Border for Outlier\n")
```

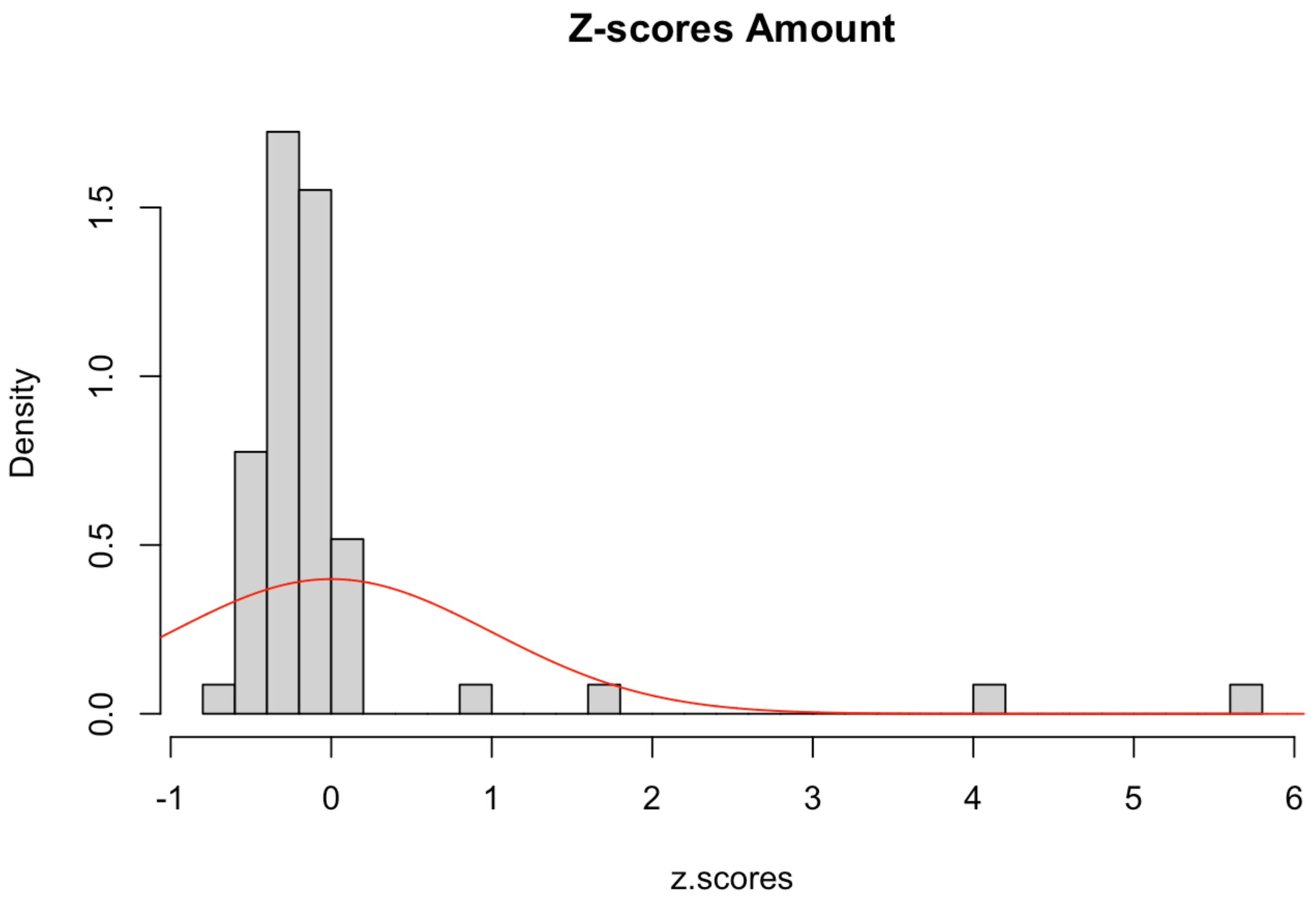
```
## Lower Border for Outlier - Upper Border for Outlier
```

```
cat(lower_fence,"------------------",upper_fence)
```

```
## -0.7302 ------------------ 0.369272
```

We can see that the variable contains upper outliers, but no lower, as the `Max` is above the upper fence of `amount` . We display the result in a histogram “Z-score Amount” showing the density by setting `freq = FALSE` and 30 bins (`breaks=30`) and create a red line indicating the normal distribution using `lines` (Rdocumentation, 2024).

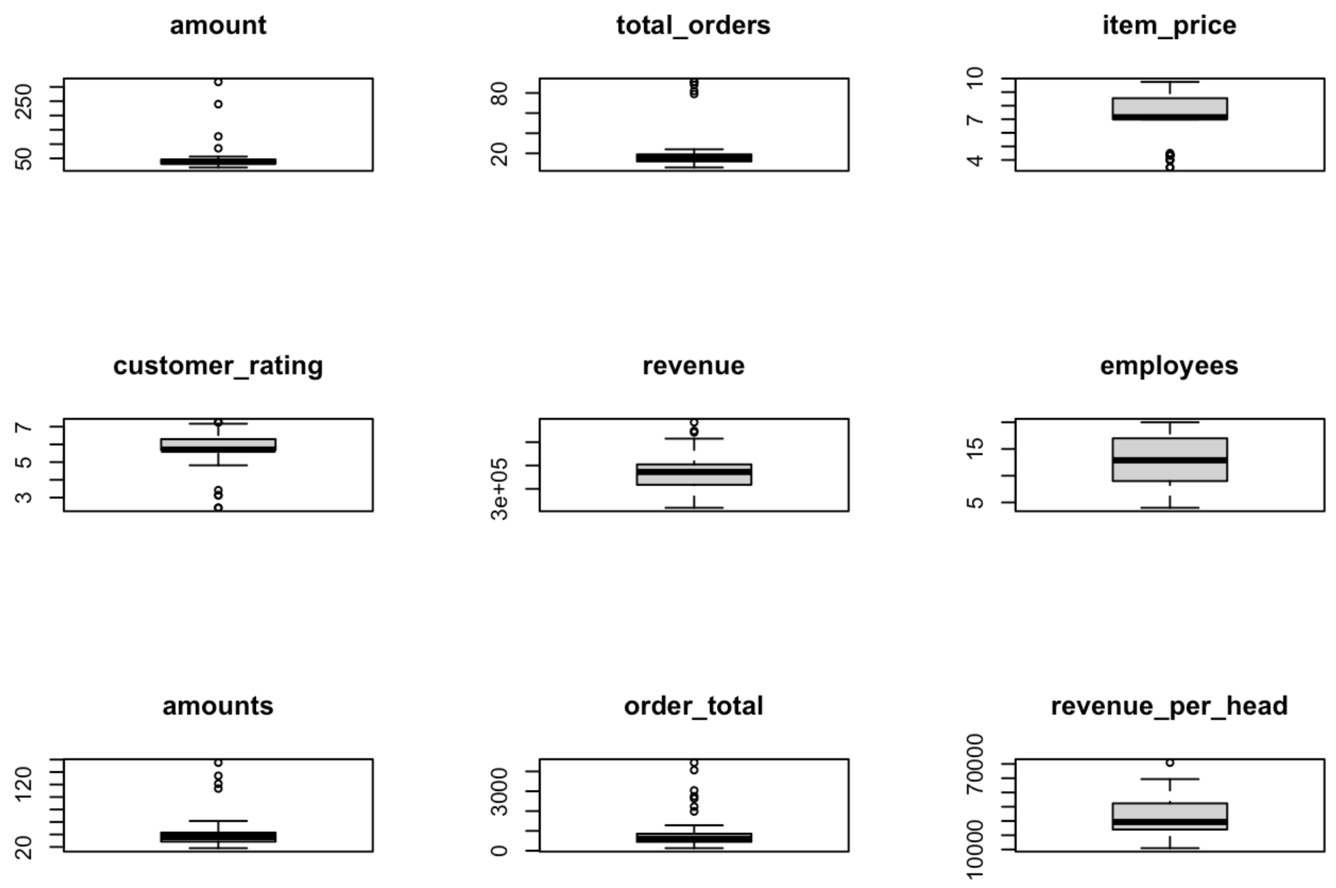
```
# Histogram
hist(z.scores, freq = FALSE, breaks = 30, main = "Z-scores Amount")
x <- seq(-15,15, by = 0.001)
lines(x = x, y = dnorm(x), col = 'red')
```



5.2.2 Visualising all outlier of all numeric variables

This section will check, if the other numeric variables are free from outliers. First, we exclude the variable `age` from our population, as we created this variable ourself. Next, we store all variables `is.numeric` variables from `exclude_age` in a new variable `numeric_var`. We display a 3x3 boxplot grid using `par` and continue with a `for loop` to iterate through the numeric variables `[numeric_var]` and display them as a boxplot. `main` sets the title of each boxplot to the name of the variable.

```
# Boxplot for all numeric variables
exclude_age <- subway_data %>%  select(-age)
numeric_var <- sapply(exclude_age, is.numeric)
par(mfrow = c(3,3))
for (col in names(exclude_age)[numeric_var]){
  boxplot(exclude_age[[col]], main = col)}
```

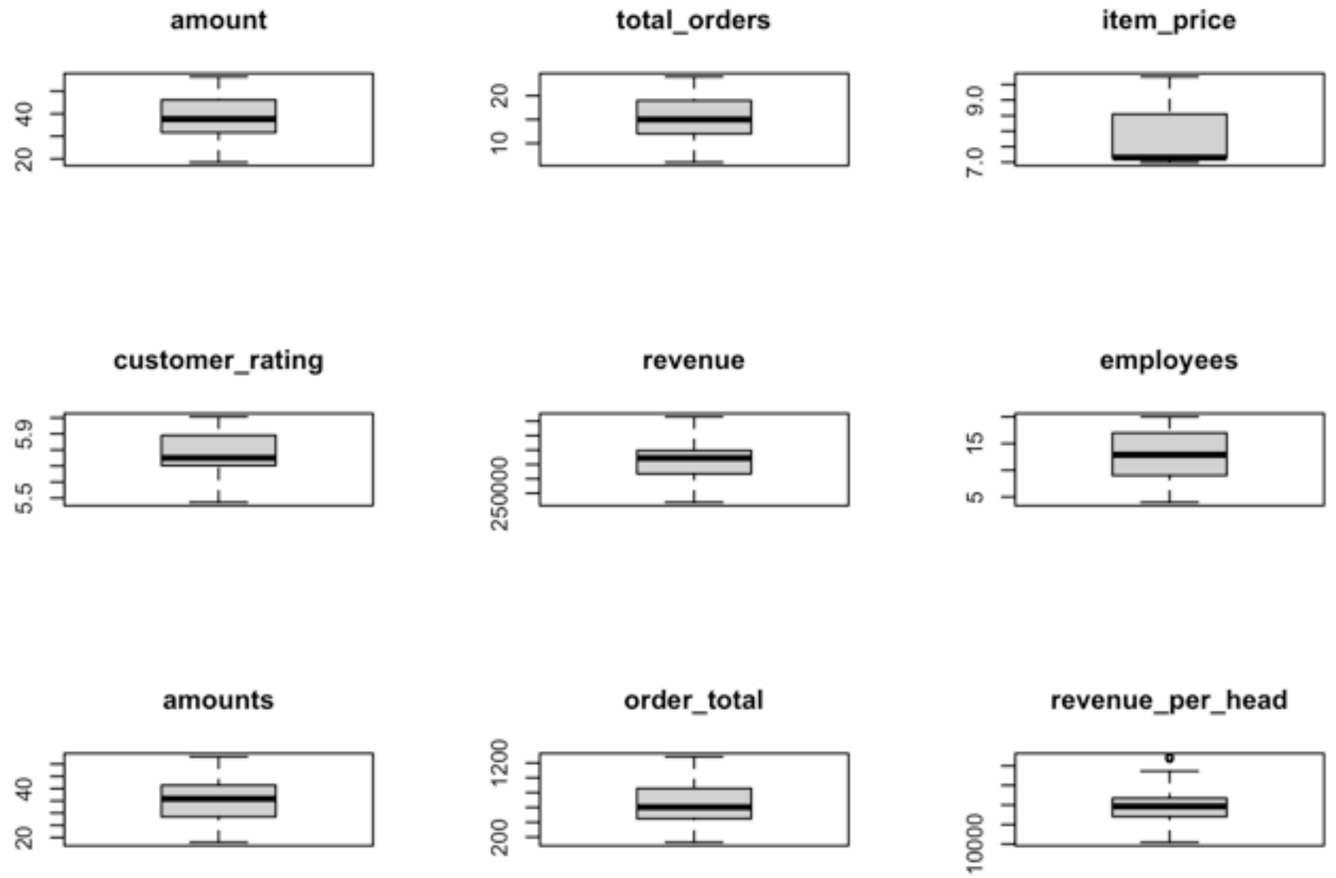


We can see that `amount` , `total_order` , `customer_rating` , `revenue` , `amounts` , `order_total` and `revenue_per_head` all contain outliers. We create a function `impute_outliers` to replace them with the *mean* value of each variable by calculating the lower and upper fences (with IQR). Is the caluclated value below the `lower_fence` or above the `upper_fence` it will be replaced by the *mean*. We then apply this function to all variables using `mutate_if` . The last part will display the same 3x3 boxplot grid as above to check if any more outliers are existing.

```
# Apply mean replacement for lower_fence and upper_fence outliers
impute_outliers <- function(x){
  q1 <- quantile(x, 0.25, na.rm = TRUE)
  q3 <- quantile(x, 0.75, na.rm = TRUE)
  iqr <- q3 - q1
  lower_fence <- q1 - 1.5 * iqr
  upper_fence <- q3 + 1.5 * iqr
  mean_var <- mean(x, na.rm=TRUE)

  x <- ifelse(x < lower_fence | x > upper_fence, mean_var , x)
  return(x)
}
# Apply the imputation to numeric variables three times to ensure it worked
for (i in 1:3){
  subway_data <- subway_data %>% mutate_if(is.numeric, impute_outliers)}
```

```
# Check if we were successful
exclude_amount <- subway_data %>%  select(-age)
numeric_var <- sapply(exclude_amount, is.numeric)
par(mfrow = c(3,3))
for (col in names(exclude_amount)[numeric_var]){
  boxplot(exclude_amount[[col]], main = col)}
```



Our function was successful.

6.0 Transform

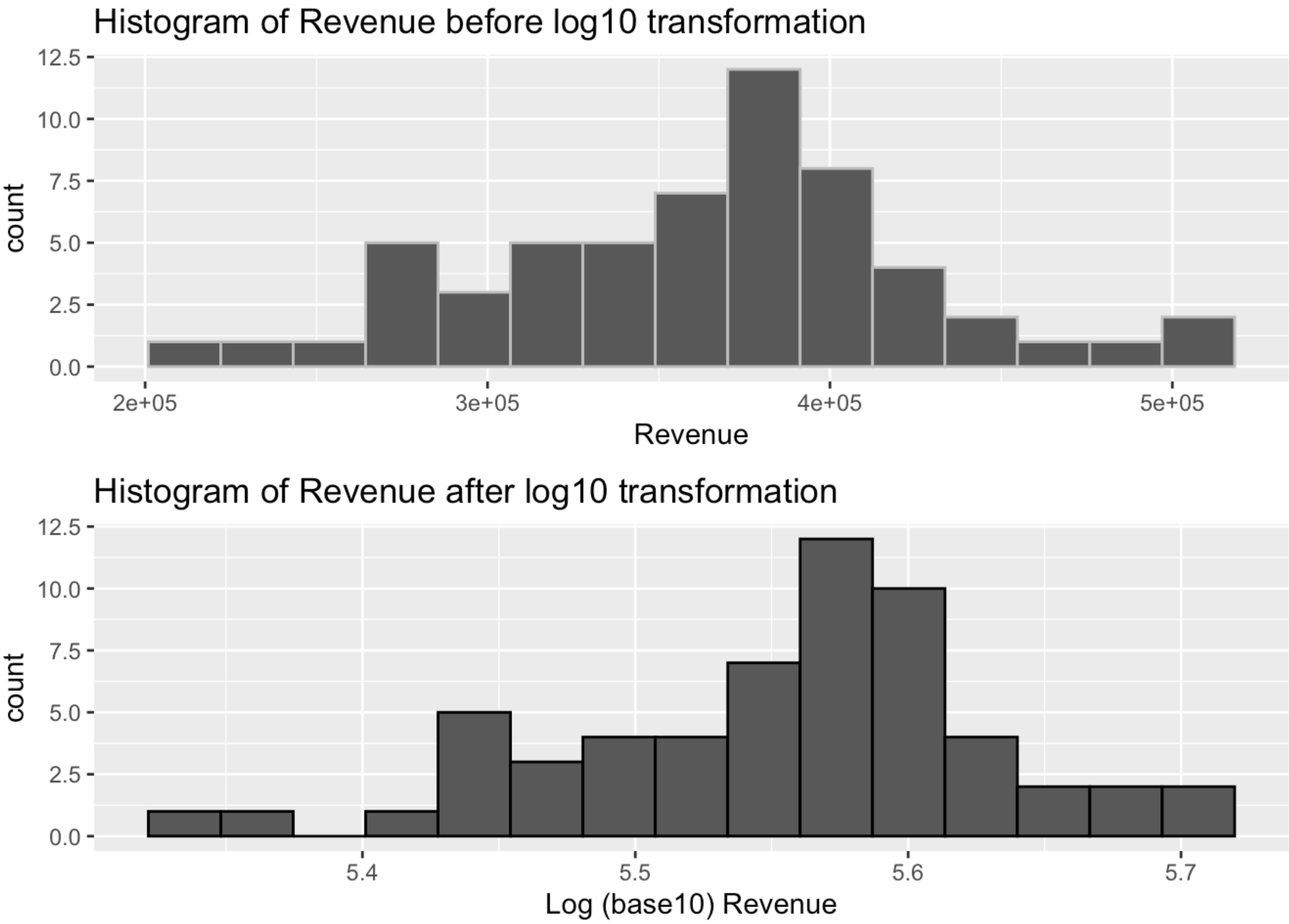
6.1 Logarithmic Transformation

Since all of our data is free from `NA` values and outliers, we proceed with data transformation to our `subway_data` to improve their distributions for better understanding and visualisation. The goal is, to transform complex non-linear relationships into linear ones and to achieve a symmetric distribution. Before we apply the logarithmic transformation on the variable `revenue` , we create a variable with the current distribution and *display the graphs a bit later for easier interpretation*.

```
# This is a chunk where you apply an appropriate transformation to at least one of
the variables.
p1 <- ggplot(subway_data, aes(x=revenue)) +
  geom_histogram(bins = 15, color = 'grey')+
  labs(title = "Histogram of Revenue before log10 transformation",
       x = "Revenue")
```

Next, we create the `log10` transformed data distribution and save it into `p2`. We display this graph using the `ggplot2` package with `gridExtra::grid.arrange` to plot it next to our original distribution `p1` for an easier comparison (Package 'gridExtra', 2017).

```
subway_data <- subway_data %>% mutate(log10_revenue = log10(revenue))
# Visualise with ggplot2
p2 <- ggplot(subway_data, aes(x=log10_revenue)) + geom_histogram(bins = 15, color =
'black') + labs(title = "Histogram of Revenue after log10 transformation",
       x = "Log (base10) Revenue")
# Display before and after
gridExtra::grid.arrange(p1 ,p2, nrow = 2)
```



We can see, that the logarithmic transformation *very slightly* improved the distribution, resulting in a more symmetric but still left-skewed distribution.

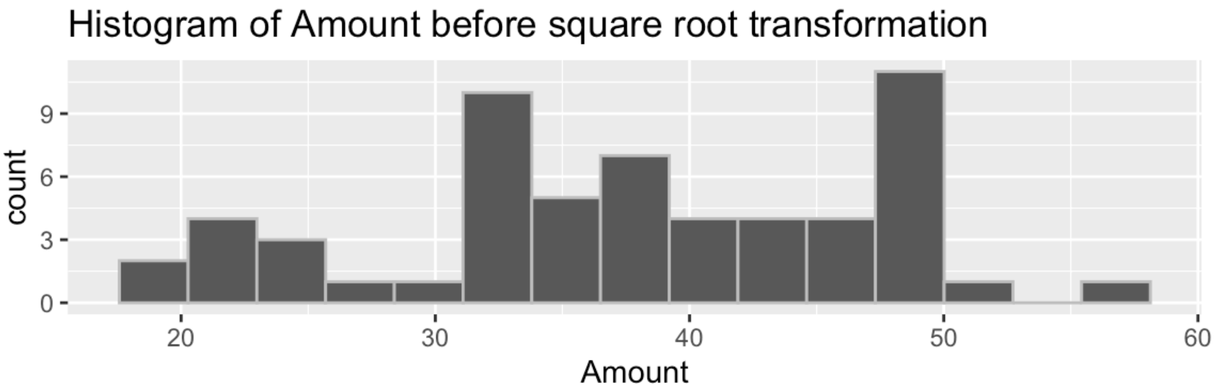
6.2 Square Root Transformation

Since the logarithmic transformation wasn't too successful, let's try a different transformation method `square root`. We structure the code similar to before, creating two `ggplot()` visualisations of the variable `amount`, the first one `o1` displaying the distribution before any transformation has been applied, the **second graph** `o2` displaying the result of the transformation (Educative, 2024).

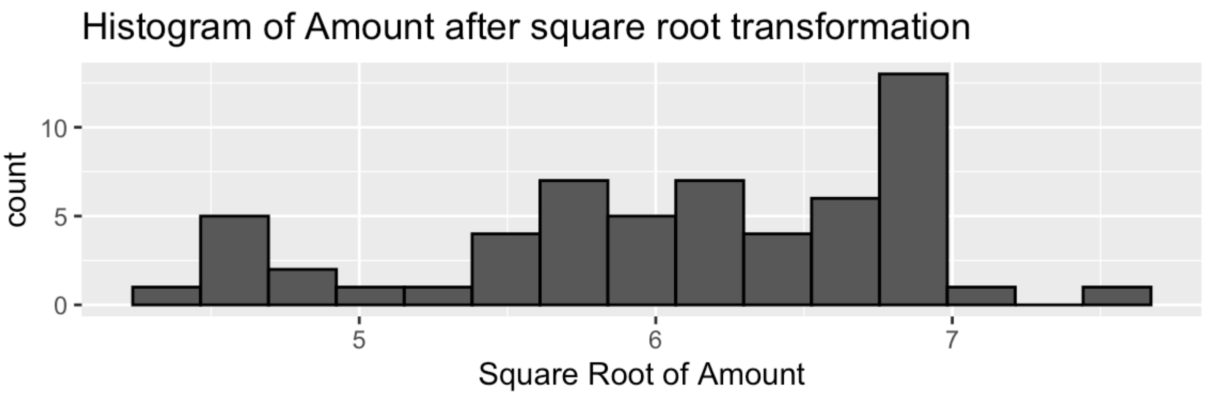
```
subway_data <- subway_data %>% mutate(sqrt_amount = sqrt(amount))
# Create the visualisations of amount before and after square root transformation
o1 <- ggplot(subway_data, aes(x=amount)) + geom_histogram(bins = 15, color='grey')
+
  labs(title = "Histogram of Amount before square root transformation",
       x = "Amount")

o2 <- ggplot(subway_data, aes(x=sqrt_amount)) +
  geom_histogram(bins = 15, color='black')+
  labs(title = "Histogram of Amount after square root transformation",
  x = "Square Root of Amount")
```

```
gridExtra::grid.arrange(o1 ,o2, nrow = 2) # Display both graphs
```



The



visualisations proof, that the `square root` transformation *improved* the symmetry of the amount distribution.

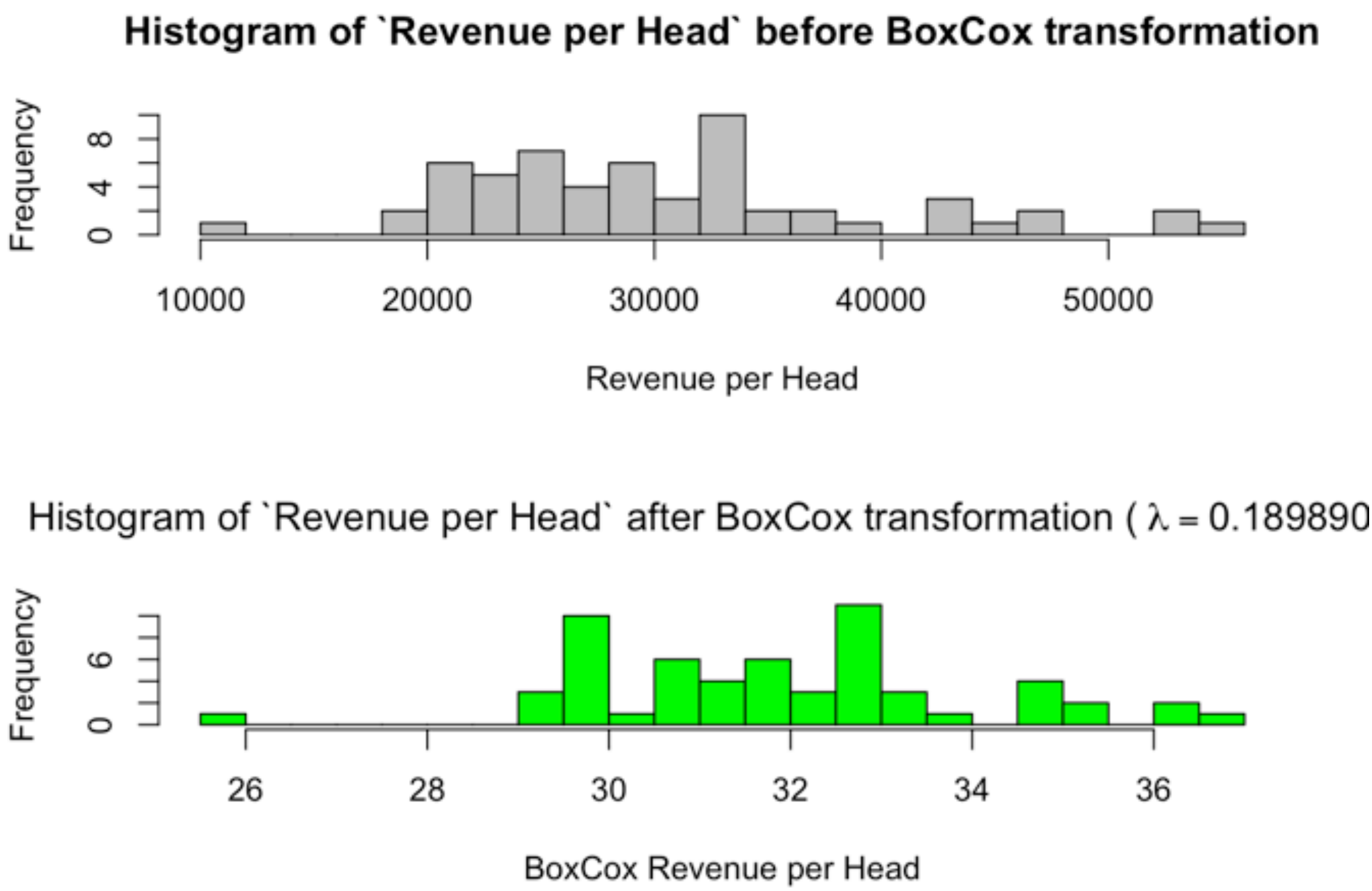
6.3 Box Cox Transformation

The last transformation I would like to apply, is the `BoxCox` transformation on the variable `total_orders`. `BoxCox` is typically used to achieve normality (Rdocumentation, 2024).

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
# Use Box Cox transformation on subway_data$revenue_per_head
boxcox_revenue_per_head <- BoxCox(subway_data$revenue_per_head, lambda = "auto")
lambda <- attr(boxcox_revenue_per_head, which = "lambda")
# Create the before and after transformation visualisation
i1 <- hist(subway_data$revenue_per_head, breaks = 25, plot = FALSE)
i2 <- hist(boxcox_revenue_per_head, breaks = 25, plot = FALSE)
# Plot both visualisations next to each other
par(mfrow = c(2,1))
plot(i1, main = "Histogram of `Revenue per Head` before BoxCox transformation", xla
b = "Revenue per Head", col = "grey")
plot(i2, main = bquote("Histogram of `Revenue per Head` after BoxCox transformation
(" ~ lambda == .(lambda) ~ ")"), xlab = "BoxCox Revenue per Head", col = "green")
```



As shown above, the BoxCox transformation of `subway_data$revenue_per_head` has a significant impact on the distribution of the data, reducing skewness. One outlier on the left corner is very noticeable though.

7.0 Summary statistics

7.1 Summary Statistics grouped by `location`

The below calculates several summary statistics of `subway_data` grouped by `location` variable. We calculate the *mean* of `amount` , *median* of `item_price` , *min* of employees, *max* of `revenue` and standard deviation *sd* of `amount` . All results will be rounded by two digits after zero. We display the first three rows as the `group_by` causes multiple outputs of `summarise` .(Tidyverse.org, 2016)

```
# Just an overview so we drop the created variable mean_amount asap after display
subway_data %>%
  group_by(location) %>%
  summarise(mean_amount = round(mean(amount),digits = 2),
            median_item_price = round(median(as.numeric(item_price)),digits = 2),
            min_employ = min(employees),
            max_rev = round(max(revenue),digits = 2), .groups = "drop") %>%
            head(3)
```

location <chr>	mean_amount <dbl>	median_item_price <dbl>	min_employ <dbl>	max_rev <dbl>
Alexandria	56.35	8.30	16	423734.3
Belrose	38.32	8.55	19	421357.2
Bondi Beach	25.42	7.15	6	395931.2

3 rows

7.2 Summary Statistics ungrouped

Lastly, we perform another summary statistics returning a dataframe with a single row, as intended by `summarise` . The below will display return the *sum* of `total_orders` , *mean* of `customer_rating` , *median* of `discount_ref` *excluding* rows without a voucher, *max* and *min* of age and the *mean* of `revenue_per_head` .

```
subway_data %>%
  summarise(total_orders = round(sum(total_orders),digits = 0),
            avg_rating = round(mean(customer_rating), digits = 2),
            median_discount = median(amounts[discount_ref != "ref-not_applied"]),
            max_age = max(age),
            min_age = min(age),
            mean_revenue_per_h = round(mean(revenue_per_head), digits = 2),
            .groups = "drop")
```

total_orders <dbl>	avg_rating <dbl>	median_discount <dbl>	max_... <dbl>	min_a... <dbl>	mean_revenue_per_h <dbl>
880	5.78	33.97402	54	16	30618.88

1 row

`.groups = "drop"` ensures that the resulting dataframes will no longer be grouping variables after the result has been displayed (Tidyverse, 2016).

Sources

- Rdocumentation.org. (2024). ifelse function | R Documentation. [online] Available at: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/ifelse (https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/ifelse) [Accessed 22 Jul. 2024].
- Rdocumentation.org. (2024). Uniform function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/Uniform (https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/Uniform) [Accessed 22 Jul. 2024].
- Rdocumentation.org. (2024). paste function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/paste (https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/paste) [Accessed 22 Jul. 2024].
- Tidyverse.org. (2024). Join data tables — left_join.dtplyr_step. [online] Available at: https://dtplyr.tidyverse.org/reference/left_join.dtplyr_step.html (https://dtplyr.tidyverse.org/reference/left_join.dtplyr_step.html) [Accessed 24 Jul. 2024].
- Tidyverse.org. (2024). Keep or drop columns using their names and types — select. [online] Available at: https://dplyr.tidyverse.org/reference/select.html (https://dplyr.tidyverse.org/reference/select.html) [Accessed 24 Jul. 2024].
- Wickham, H. and Grolemund, G. (2016). R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. 1st Edition. O'Reilly Media [Accessed 26 Jul. 2024].
- GeeksforGeeks. (2021). Group Factor Levels in R. [online] Available at: https://www.geeksforgeeks.org/group-factor-levels-in-r/ (https://www.geeksforgeeks.org/group-factor-levels-in-r/) [Accessed 26 Jul. 2024].
- Rdocumentation.org. (2019). invisible function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/invisible

(https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/invisible) [Accessed 1 Aug. 2024].

- Zach (2022). How to Use select_if with Multiple Conditions in dplyr. [online] Statology. Available at: https://www.statology.org/dplyr-select-if-multiple-conditions/[Accessed (https://www.statology.org/dplyr-select-if-multiple-conditions/%5BAccessed) 28 Jul. 2024].
- Rdocumentation.org. (2024). format function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/format (https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/format) [Accessed 28 Jul. 2024].
- Zach (2021). How to Round Numbers in R (5 Examples). [online] Statology. Available at: https://www.statology.org/round-in-r/ (https://www.statology.org/round-in-r/) [Accessed 29 Jul. 2024].
- Rpubs.com. (2024). RPubs - lapply, sapply, and vapply. [online] Available at: https://rpubs.com/GilbertTeklevchiev/1113536 (https://rpubs.com/GilbertTeklevchiev/1113536) [Accessed 29 Jul. 2024].
- Ethz.ch. (2024). R: Which indices are TRUE? [online] Available at: https://stat.ethz.ch/R-manual/R-devel/library/base/html/which.html (https://stat.ethz.ch/R-manual/R-devel/library/base/html/which.html) [Accessed 1 Aug. 2024].
- Had.co.nz. (2024). Functional programming · Advanced R. [online] Available at: http://adv-r.had.co.nz/Functional-programming.html (http://adv-r.had.co.nz/Functional-programming.html) [Accessed 1 Aug. 2024]. *GeeksforGeeks. (2021). Plot Z-Score in R. [online] Available at: https://www.geeksforgeeks.org/plot-z-score-in-r/ (https://www.geeksforgeeks.org/plot-z-score-in-r/) [Accessed 30 Jul. 2024].
- Rdocumentation.org. (2024). lines function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/lines[Accessed (https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/lines%5BAccessed) 30 Jul. 2024].
- Package ‘gridExtra’. (2017). Available at: https://cran.r-project.org/web/packages/gridExtra/gridExtra.pdf (https://cran.r-project.org/web/packages/gridExtra/gridExtra.pdf) [Accessed 30 Jul. 2024].
- Educative. (2024). Educative Answers - Trusted Answers to Developer Questions. [online] Available at: https://www.educative.io/answers/how-to-calculate-the-square-root-of-a-number-in-r (https://www.educative.io/answers/how-to-calculate-the-square-root-of-a-number-in-r) [Accessed 30 Jul. 2024].
- Rdocumentation.org. (2023). boxcox function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/EnvStats/versions/2.8.1/topics/boxcox (https://www.rdocumentation.org/packages/EnvStats/versions/2.8.1/topics/boxcox) [Accessed 2 Aug. 2024].
- Tidyverse.org. (2019). Group by one or more variables — group_by. [online] Available at: https://dplyr.tidyverse.org/reference/group_by.html (https://dplyr.tidyverse.org/reference/group_by.html) [Accessed 2 Aug. 2024].
- Tidyverse.org. (2024). Summarise each group to fewer rows — summarise. [online] Available at: https://dplyr.tidyverse.org/reference/summarise.html (https://dplyr.tidyverse.org/reference/summarise.html) [Accessed 2 Aug. 2024].