EDA and Descriptive Statistics using RStudio

Course Taught at SUAD

Tanujit Chakraborty & Madhurima Panja (TA)

MDA Course @ Sorbonne

Code Link: https://github.com/tanujit123/MATH-260

What we quest to achieve through the sessions

- Data Pre-processing
- Playing around Email dataset
- Exploratory Data Analysis

Data Pre-processing

DATA PREPROCESSING

1. Missing value replenishment
2. Transformation or normalization
3. Random Sampling

Missing Value Handling

Example: Suppose a telecom company wants to analyze the performance of its circles based on the following parameters

1. Current Month’s Usage
2. Last 3 Month’s Usage
3. Average Recharge
4. Projected Growth

The data set is given in next slide. (Missing_Values_Telecom Data)

Missing Value Handling

Example: Circle wise Data

SL No.	Current Month's Usage	Last 3 Month's Usage	Average Recharge	Projected Growth	Circle
1	5.1	3.5	99.4	99.2	A
2	4.9	3	98.6	99.2	A
3		3.2		99.2	A
4	4.6	3.1	98.5	9..2	A
5	5		98.4	99.2	A
6	5.4	3.9	98.3	99.4	A
7	7	3.2	95.3	98.4.	B
8	6.4	3.2	95.5	98.5	B
9	6.9	3.1	95.1	98.5	B
10		2.3	96	98.3	B
11	6.5	2.8	95.4	98.5	B
12	5.7		95.5	98.3	B
13	6.3	3.3		98.6	B
14	6.7	3.3	94.3	97.5	C
15	6.7	3	94.8	97.3	C
16	6.3	2.5	95	98.9	C
17		3	94.8	98	C
18	6.2	3.4	94.6	97.3	C
19	5.9	3	94.9	98.8	C

Missing Value Handling

Example: Read data and variables to R

```
> mydata = Missing_Values_Telecom
> cmusage = mydata[,2]
> l3musage = mydata[,3]
> avrecharge = mydata[,4]
```

Missing Value Handling

Option 1: Discard all records with missing values

```
>newdata = na.omit(mydata)

>write.csv(newdata,"E:/SUAD/newdata.csv")
```

SL.No.	Current.Month.s.Usage	Last.3.Month.s.Usage	Average.Recharge	Projected.Growth	Circle
1	5.1	3.5	99.4	99.2	A
2	4.9	3	98.6	99.2	A
4	4.6	3.1	98.5	9..2	A
6	5.4	3.9	98.3	99.4	A
7	7	3.2	95.3	98.4.	B
8	6.4	3.2	95.5	98.5	B
9	6.9	3.1	95.1	98.5	B
11	6.5	2.8	95.4	98.5	B
14	6.7	3.3	94.3	97.5	C
15	6.7	3	94.8	97.3	C
16	6.3	2.5	95	98.9	C
18	6.2	3.4	94.6	97.3	C
19	5.9	3	94.9	98.8	C

Missing Value Handling

Option 2: Replace the missing values with variable mean, median, etc

Replacing the missing values with mean

Compute the means excluding the missing values

```
>cmusage_mean = mean(cmusage, na.rm = TRUE)
>l3musage_mean = mean(l3musage, na.rm = TRUE)
> avrecharge_mean = mean(avrecharge, na.rm = TRUE)
```

Replace the missing values with mean

```
> cmusage[is.na(cmusage)]=cmusage_mean
> l3musage[is.na(l3musage)]= l3musage_mean
>avrecharge[is.na(avrecharge)]=avrecharge_mean
```

Missing Value Handling

Option 2: Replace the missing values with variable mean, median, etc

Replacing the missing values with mean

Replace the missing values with mean

```
> cmusage[is.na(cmusage)]=cmusage_mean
> l3musage[is.na(l3musage)]= l3musage_mean
>avrecharge[is.na(avrecharge)]=avrecharge_mean
```

Making the new file

```
> mynewdata = cbind(cmusage, l3musage, avrecharge, mydata[,5],mydata[,6])
> write.csv(mynewdata, "E:/SUAD/mynewdata.csv")
```

Missing Value Handling

Option 2: Replace the missing values with variable mean, median, etc

Replacing the missing values with men

SL No	cmusage	l3musage	avrecharge	Proj Growth	Circle
1	5.1	3.5	99.4	11	1
2	4.9	3	98.6	11	1
3	5.975	3.2	96.14117647	11	1
4	4.6	3.1	98.5	1	1
5	5	3.105882353	98.4	11	1
6	5.4	3.9	98.3	12	1
7	7	3.2	95.3	6	2
8	6.4	3.2	95.5	7	2
9	6.9	3.1	95.1	7	2
10	5.975	2.3	96	5	2
11	6.5	2.8	95.4	7	2
12	5.7	3.105882353	95.5	5	2
13	6.3	3.3	96.14117647	8	2
14	6.7	3.3	94.3	3	3
15	6.7	3	94.8	2	3
16	6.3	2.5	95	10	3
17	5.975	3	94.8	4	3
18	6.2	3.4	94.6	2	3
19	5.9	3	94.9	9	3

TRANSFORMATION / NORMALIZATION

z transform:

Transformed data = (Data – Mean) / SD

Exercise : Normalize the variables in the Supply_Chain.csv ?

Read the files

```
>mydata  = Supply_Chain
> mydata = mydata[,2:7]
```

Normalize or standardize the variable

```
>mystddata = scale(mydata)
```

RANDOM SAMPLING

Example: Take a sample of size 60 (10%) randomly from the data given in the file bank data.csv and save it as a new csv file?

Read the files

```
>mydata  = bank data
> mysample = mydata[sample(1:nrow(mydata), 60, replace = FALSE),]
>write.csv(mysample,"E:/SUAD/mysample.csv")
```

Example: Split randomly the data given in the file bank data.csv into sets namely training (75%) and test (25%) ?

Read the files

```
>mydata  = bank data
>sample = sample(2, nrow(mydata), replace = TRUE, prob = c(0.75, 0.25))
> sample1 = mydata[sample ==1, ]
> sample2 = mydata[sample ==2,]
```

PLAY WITH YOUR DATA

WHAT WE WILL LEARN IN THIS?

- Tools for data exploration and transformation
- Intuitive to write and easy to read
- Super-fast on data frames

INSTALLATION OF PACKAGES

We will use the ‘dplyr’ package in R. Since it is a part of ‘tidyverse’ package installing ‘dplyr’ separately is not mandatory.

```
library(tidyverse)
```

LOAD AN EXAMPLE DATASET

```
suppressMessages(library(dplyr))

install.packages("hflights")

library(hflights)
data(hflights)          # hflights is flights departing from two Houston airports in 2011
```

EXPLORE THE DATA

head(hflights)

```
     Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum TailNum ActualElapsedTime
5424 2011     1          1         6    1400    1500            AA       428  N576AA                60
5425 2011     1          2         7    1401    1501            AA       428  N557AA                60
5426 2011     1          3         1    1352    1502            AA       428  N541AA                70
5427 2011     1          4         2    1403    1513            AA       428  N403AA                70
5428 2011     1          5         3    1405    1507            AA       428  N492AA                62
5429 2011     1          6         4    1359    1503            AA       428  N262AA                64
     AirTime ArrDelay DepDelay Origin Dest Distance TaxiIn TaxiOut Cancelled CancellationCode Diverted
5424      40      -10        0    IAH  DFW      224      7      13         0                         0
5425      45       -9        1    IAH  DFW      224      6       9         0                         0
5426      48       -8       -8    IAH  DFW      224      5      17         0                         0
5427      39        3        3    IAH  DFW      224      9      22         0                         0
5428      44       -3        5    IAH  DFW      224      9       9         0                         0
5429      45       -7       -1    IAH  DFW      224      6      13         0                         0
```

as_tibble creates a "local data frame", which is just a wrapper for a data frame that prints nicely.

 # convert to local data frame
flights <- as_tibble(hflights)

EXPLORE THE DATA

flights

```
## A tibble: 227,496 × 21
##     Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum TailNum
##    <int> <int> <int>          <int>   <int>   <int>         <chr>     <int>   <chr>
##  1  2011     1     1              6    1400    1500            AA       428  N576AA
##  2  2011     1     2              7    1401    1501            AA       428  N557AA
##  3  2011     1     3              1    1352    1502            AA       428  N541AA
##  4  2011     1     4              2    1403    1513            AA       428  N403AA
##  5  2011     1     5              3    1405    1507            AA       428  N492AA
##  6  2011     1     6              4    1359    1503            AA       428  N262AA
##  7  2011     1     7              5    1359    1509            AA       428  N493AA
##  8  2011     1     8              6    1355    1454            AA       428  N477AA
##  9  2011     1     9              7    1443    1554            AA       428  N476AA
## 10  2011     1    10              1    1443    1553            AA       428  N504AA
## ... with 227,486 more rows, and 12 more variables: ActualElapsedTime <int>,
##   AirTime <int>, ArrDelay <int>, DepDelay <int>, Origin <chr>, Dest <chr>,
##   Distance <int>, TaxiIn <int>, TaxiOut <int>, Cancelled <int>,
##   CancellationCode <chr>, Diverted <int>
```

This prints the data nicely with 10 rows and as many columns that fits the screen.

EXPLORE THE DATA

 # Revert to the original data frame to view all the columns

head(data.frame(flights))

```
  Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum TailNum
1 2011     1          1         6    1400    1500            AA       428  N576AA
2 2011     1          2         7    1401    1501            AA       428  N557AA
3 2011     1          3         1    1352    1502            AA       428  N541AA
4 2011     1          4         2    1403    1513            AA       428  N403AA
5 2011     1          5         3    1405    1507            AA       428  N492AA
6 2011     1          6         4    1359    1503            AA       428  N262AA
  ActualElapsedTime AirTime ArrDelay DepDelay Origin Dest Distance TaxiIn TaxiOut
1                60      40      -10        0    IAH  DFW      224      7      13
2                60      45       -9        1    IAH  DFW      224      6       9
3                70      48       -8       -8    IAH  DFW      224      5      17
4                70      39        3        3    IAH  DFW      224      9      22
5                62      44       -3        5    IAH  DFW      224      9       9
6                64      45       -7       -1    IAH  DFW      224      6      13
  Cancelled CancellationCode Diverted
1         0                         0
2         0                         0
3         0                         0
4         0                         0
5         0                         0
6         0                         0
```

EXPLORE THE DATA

Functions in dplyr:

- Pick observations by their values: *filter()*
- Reorder the rows: *arrange()*.
- Pick variables by their names: *select()*.
- Create new variables with functions of existing variables: *mutate()*
- Collapse many values down to a single summary: *summarise()*
- These can all be used in conjunction with *group_by()* which changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

These six functions provide the verbs for a language of data manipulation.

EXPLORE THE DATA

filter: Keep rows matching criteria

- Base R approach to filtering forces you to repeat the data frame's name
- dplyr approach is simpler to write and read
- Command structure (for all dplyr verbs):
  - first argument is a data frame
  - return value is a data frame
  - nothing is modified in place

Qn: Extract details of flights operating on January 01.

```
# base R approach to view all flights on January 1
flights[flights$Month==1 & flights$DayofMonth==1, ]

# try the filter function of dplyr for the same question
# dplyr approach
# note: you can use comma or ampersand to represent AND condition
filter(flights, Month==1, DayofMonth==1)
```

EXPLORE THE DATA

Qn: Extract details of flights operated by American Airlines or United Airlines.

```
# use pipe for OR condition
filter(flights, UniqueCarrier=="AA" | UniqueCarrier=="UA")

# you can also use %in% operator
filter(flights, UniqueCarrier %in% c("AA", "UA"))
# Check the two answers
```

EXPLORE THE DATA

select: Pick columns by name

Qn: Show flight details with Dep/Arr times along with flight numbers only.

 # Show flight details with Dep/Arr times alongwith flight numbers only.
 # base R approach to select DepTime, ArrTime, and FlightNum columns
flights[, c("DepTime", "ArrTime", "FlightNum")]

 # dplyr approach
select(flights, DepTime, ArrTime, FlightNum)
 # Check the two answers

 # Use colon to select multiple contiguous columns, and use contains to match columns by name
select(flights, Year:DayofMonth, contains("Taxi"), contains("Delay"))

 # starts_with, ends_with, and matches (for regular expressions) can also be used to match columns by name
select(flights, Year:DayofMonth, starts_with("Taxi"), ends_with("Delay"))
 # Check the two answers

EXPLORE THE DATA

"Chaining" or "Pipelining“

Usual way to perform multiple operations in one line is by nestingCan write commands in a natural order by using the %>% infix operator (pipes, which can be pronounced as "then")

```
# Show unique carrier details for flights having departure delays of more than 1 hour.
# nesting method
filter(select(flights, UniqueCarrier, DepDelay), DepDelay > 60)

# chaining method
flights %>% select(UniqueCarrier, DepDelay) %>% filter(DepDelay > 60)
```

- Chaining increases readability significantly when there are many commands
- Operator is automatically imported from the magrittr package
- Can be used to replace nesting in R commands outside of dplyr

```
# create two vectors and calculate Euclidian distance between them
x1 <- 1:5; x2 <- 2:6
sqrt(sum((x1-x2)^2))

# chaining method
(x1-x2)^2 %>% sum() %>% sqrt()
## [1] 2.236068
```

EXPLORE THE DATA

arrange: Reorder rows

Qn: Extract flight carrier details and show departure delays only sorted by delay lengths.

```r
# base R approach
flights[order(flights$DepDelay), c("UniqueCarrier", "DepDelay")]

# dplyr approach
flights %>% select(UniqueCarrier, DepDelay) %>% arrange(DepDelay)
# arrange(desc(DepDelay)) for descending order
```

EXPLORE THE DATA

mutate: Add new variables

Qn. Find mean speed of flights.

```
# base R approach
flights$Speed <- flights$Distance / flights$AirTime*60
flights[, c("Distance", "AirTime", "Speed")]

# dplyr approach (prints the new variable but does not store it)
flights %>%
  select(Distance, AirTime) %>%
  mutate(Speed = Distance/AirTime*60)

# storing the speed variable
flights <- flights %>% mutate(Speed = Distance/AirTime*60)
flights
```

EXPLORE THE DATA

summarise: Reduce variables to values

- Primarily useful with data that has been grouped by one or more variables
- group_by creates the groups that will be operated on
- summarise uses the provided aggregation function to summarise each group

Qn. Find average delay to each destination.

 # dplyr approach: create a table grouped by Dest, and then summarise each group by taking the mean of ArrDelay

```
flights %>% group_by(Dest) %>% summarise(avg_delay = mean(ArrDelay, na.rm=TRUE))
```

EXPLORE THE DATA

group_by function

Qn. For each day of the year, count the total number of flights and sort in descending order.

```
flights %>% group_by(Month, DayofMonth) %>% summarise(flight_count = n()) %>%
  arrange(desc(flight_count))
```

```
## `summarise()` has grouped output by 'Month'. You can override using the
## `.groups` argument.
```

n_distinct (vector): counts the number of unique items in the vector

Qn. For each destination, count the total number of flights and the number of distinct planes that flew there.

```
flights %>% group_by(Dest) %>%  summarise(flight_count = n(), plane_count =
n_distinct(TailNum))
```

EXPLORE THE DATA

Window Functions

- Aggregation function (like mean) takes n inputs and returns 1 value
- Window function takes n inputs and returns n values
- Includes ranking and ordering functions (like min_rank), offset functions (lead and lag), and cumulative aggregates (like cummean).

Qn. For each carrier, calculate which two days of the year they had their longest departure delays.

```
# Note: smallest (not largest) value is ranked as 1, so you have to use `desc` to rank by largest value
flights %>%   group_by(UniqueCarrier) %>%  select(Month, DayofMonth, DepDelay) %>%
  filter(min_rank(desc(DepDelay)) <= 2) %>%   arrange(UniqueCarrier, desc(DepDelay))

# rewrite more simply with the `top_n` function
flights %>%  group_by(UniqueCarrier) %>%  select(Month, DayofMonth, DepDelay) %>%
  top_n(2) %>%  arrange(UniqueCarrier, desc(DepDelay))
```

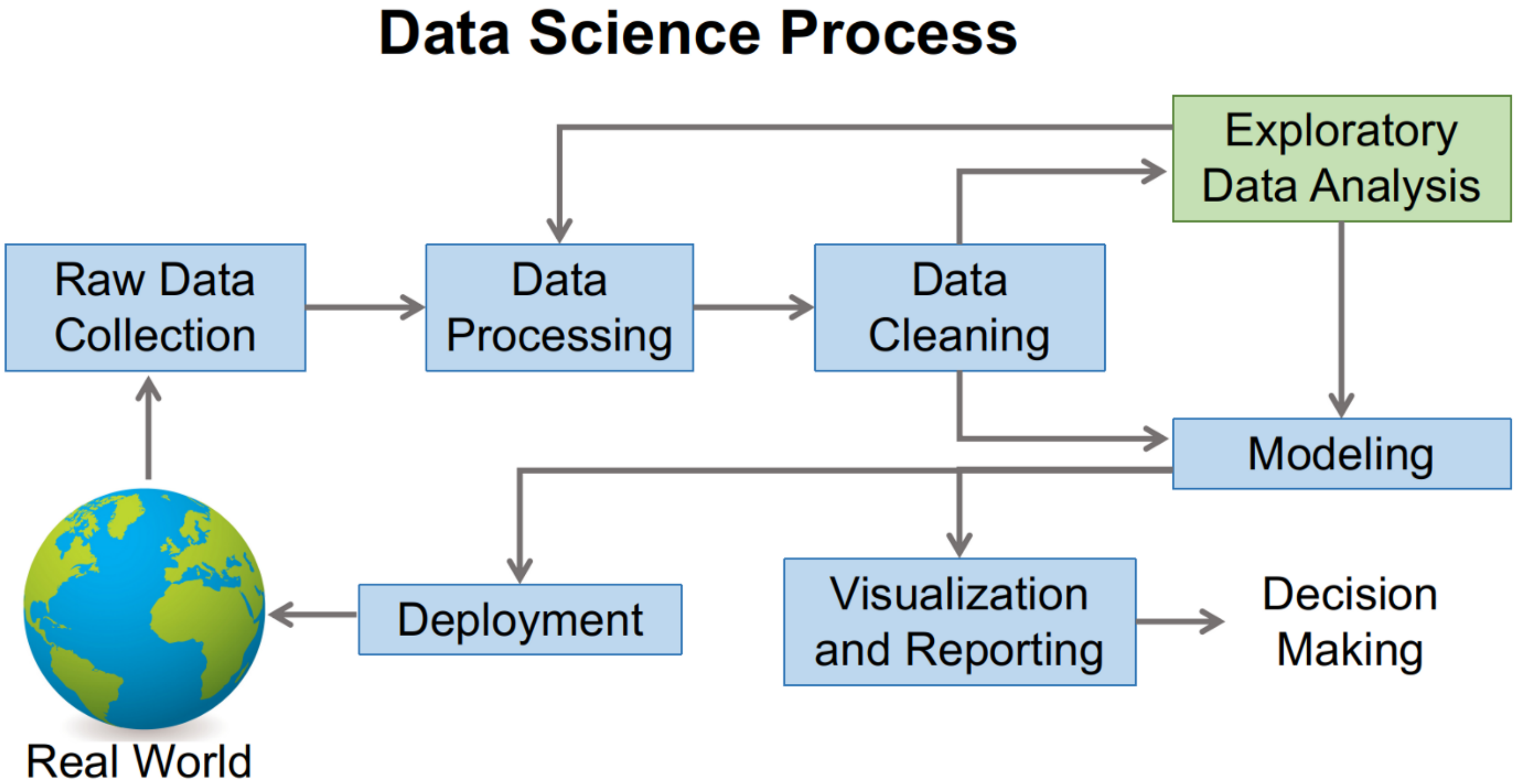
EXPLORE THE DATA

Useful Convenience Functions

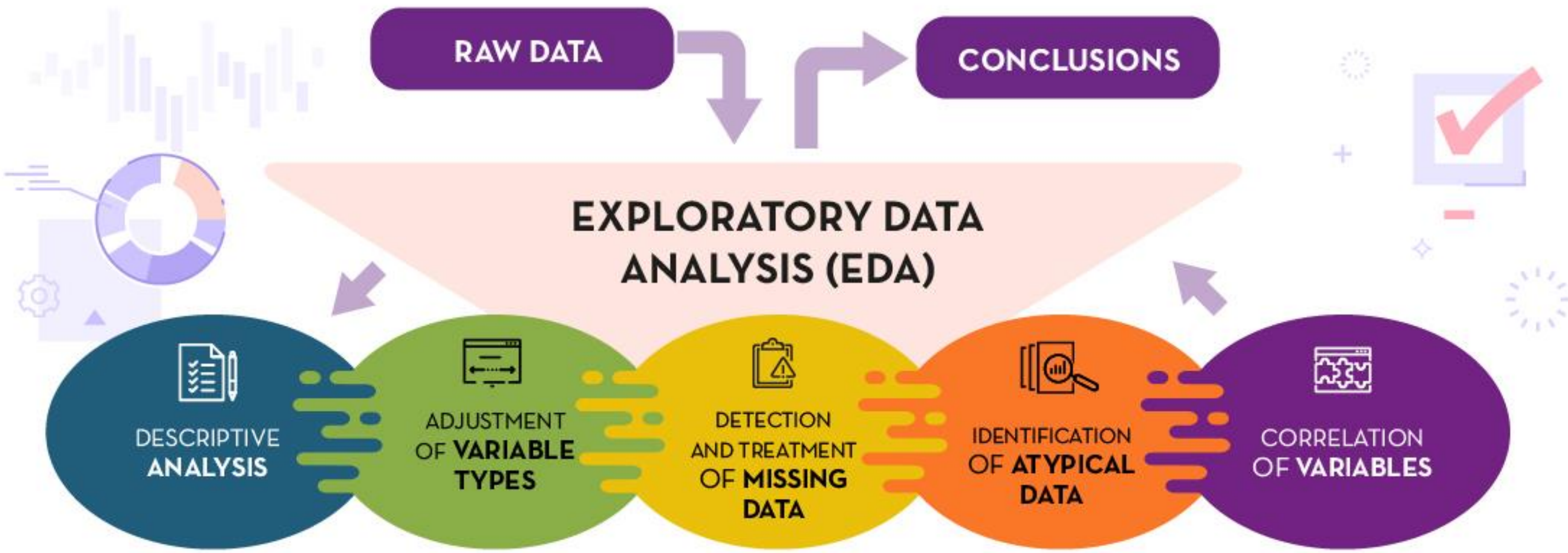
```
# randomly sample a fixed number of rows, without replacement
flights %>% sample_n(5)

# randomly sample a fraction of rows, with replacement
flights %>% sample_frac(0.25, replace=TRUE)

# base R approach to view the structure of an object
str(flights)

# dplyr approach: better formatting, and adapts to your screen width
glimpse(flights)
```

EXPLORATORY DATA ANALYSIS (EDA)

WHAT IS EDA IN DATA SCIENCE?



WHAT IS EDA?


RAW DATA
CONCLUSIONS
EXPLORATORY DATA ANALYSIS (EDA)
DESCRIPTIVE ANALYSIS
ADJUSTMENT OF VARIABLE TYPES
DETECTION AND TREATMENT OF MISSING DATA
IDENTIFICATION OF ATYPICAL DATA
CORRELATION OF VARIABLES

WHAT IS EDA?

- Exploratory Data Analysis, or EDA in short, use visualization and transformation to explore data in a systematic way. It is an iterative cycle where one
  - Generates questions about the data.
  - Search for answers by visualizing, transforming, and modelling the data.
  - Use what we learn to refine our questions and/or generate new questions.

- EDA is not a formal process with a strict set of rules. During the initial phases of EDA one should feel free to investigate every idea that occurs.

- Some of these ideas will pan out, and some will be dead ends. As our exploration continues, we will home in on a few particularly productive areas that you'll eventually write up and communicate to others.

- EDA is an important part of any data analysis, even if the questions are handed to you on a platter, because you always need to investigate the quality of your data.

- Data cleaning is just one application of EDA: we ask questions about whether our data meets our expectations or not.

- To do data cleaning, we'll need to deploy all the tools of EDA: visualisation, transformation, and modelling.

EDA Case Study

Let's begin with the ***email*** dataset

- These data represent incoming emails for the first three months of 2012 for an email account.
- The data frame has 3921 observations on 21 variables.
- Some of the variables are:

  - ***spam***: Indicator for whether the email was spam.
  - ***to multiple***: Indicator for whether the email was addressed to more than one recipient.
  - ***from***: Whether the message was listed as from anyone (this is usually set by default for regular outgoing email).
  - ***dollar***: The number of times a dollar sign or the word "dollar" appeared in the email.
  - ***winner*** : Indicates whether "winner" appeared in the email.
  - ***inherit***: The number of times "inherit" (or an extension, such as "inheritance") appeared in the email.
  - ***num_char***: The number of characters in the email, in thousands.
  - ***urgent_subj*** : Whether the word "urgent" was in the email subject.

EDA Case Study

Loading the dataset

```
library(tidyverse)
library(openintro) #for loading our case study dataset
data(email)
email
```

Can you tell the format of the dataset?

EDA Case Study

Possible Questions for EDA:

- How is the length of an email related to an email being a spam one?
- Experience tells us that presence of excessive exclamation marks in an email makes it more likely to be a spam. Explore that relationship for this dataset.
- How is the number of images attached with the email relate to spam emails?
- Returning to the length of an email, how do non-spam emails characterize wrt the number of recipients?

EDA

- Data Analysis is not always about providing answers to the specific questions asked.
- More than often, asking the right questions are more than important.
- Many industries/institutions generate huge volume of data, but have no clue what to do with them.

EDA Case Study

Measuring variability

- Variation is the tendency of the values of a variable to change from measurement to measurement.
- You can see variation easily in real life; if you measure any continuous variable twice, you will get two different results. This is true even if you measure quantities that are constant, like the speed of light.
- Each of your measurements will include a small amount of error that varies from measurement to measurement.
- Categorical variables can also vary if you measure across different subjects (e.g. the eye colors of different people), or different times (e.g. fluctuation [up/down] of a stock over different time periods).
- Every variable has its own pattern of variation, which can reveal interesting information.
- The best way to understand that pattern is to visualize the distribution of the variable's values.

EDA Case Study

Exploring number of characters in spam vs non-spam emails

- Let us first explore how the number of characters vary from spam to non-spam emails.
- In this quest, we can first look into summaries of central tendencies and variability for the ***num_char*** variable grouped by the ***spam*** status.
- A good way to visualize and compare the location and spread is to create side-by-side boxplots.
- Note that spam is of numeric type. We have to change it to factor type.
- We also rename 0s as "Not_spam" and 1s as "Spam".

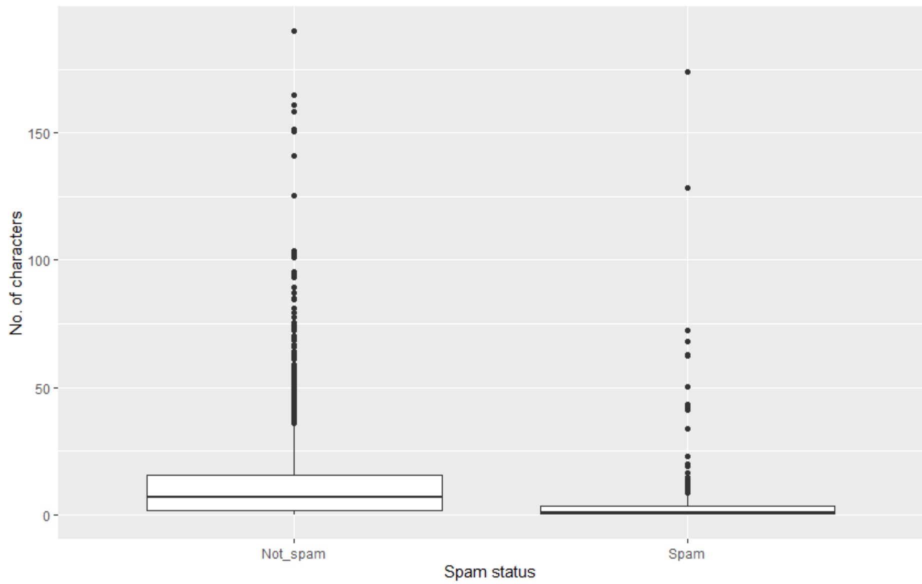
```
# Exploring number of characters in spam vs non-spam emails
email <- email %>%
  mutate(spam = as.factor(spam)) %>%
  mutate(spam = recode(spam, "0" = "Not_spam", "1" = "Spam"))

email
```

What change do you notice?

EDA Case Study

```
# Compute summary statistics
email %>%
  group_by(spam) %>% summarize(median(num_char), IQR(num_char))

# A tibble: 2 × 3
##    spam     `median(num_char)` `IQR(num_char)`
##    <fct>          <dbl>               <dbl>
## 1 Not_spam        6.83                13.6
## 2   Spam          1.05                2.82
```

```
# Visualize
email %>%
  ggplot(aes(x = spam, y = num_char)) +
  geom_boxplot() +
  labs(x = "Spam status",
       y = "No. of characters")
```

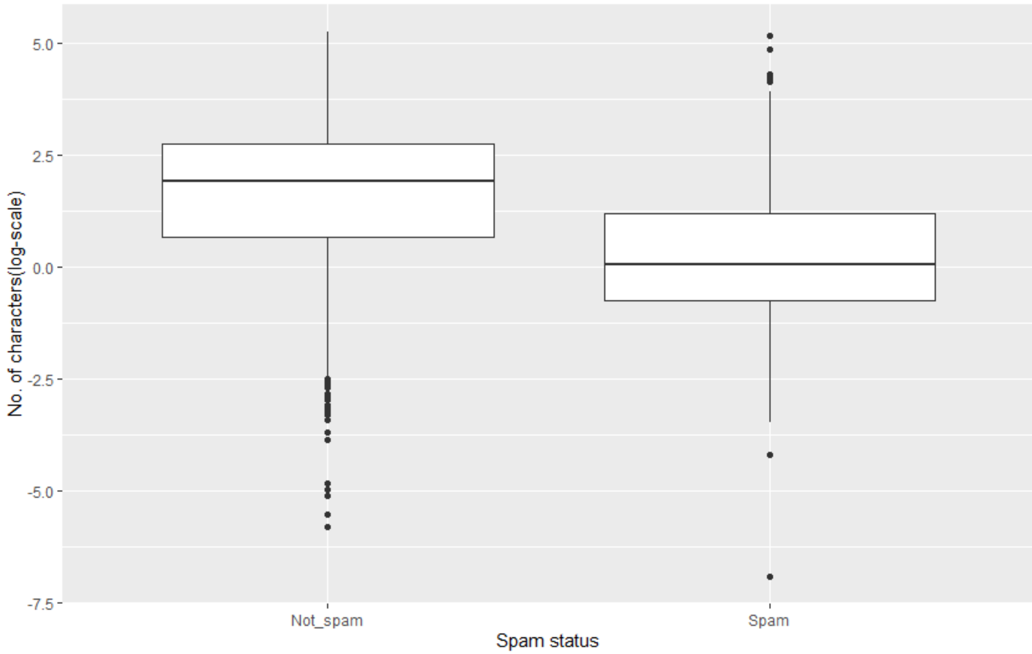


What can you infer from the plot?
Any suggestions on how to improve the plot?

EDA Case Study

It might be a good idea to change the scale of the y-axis.

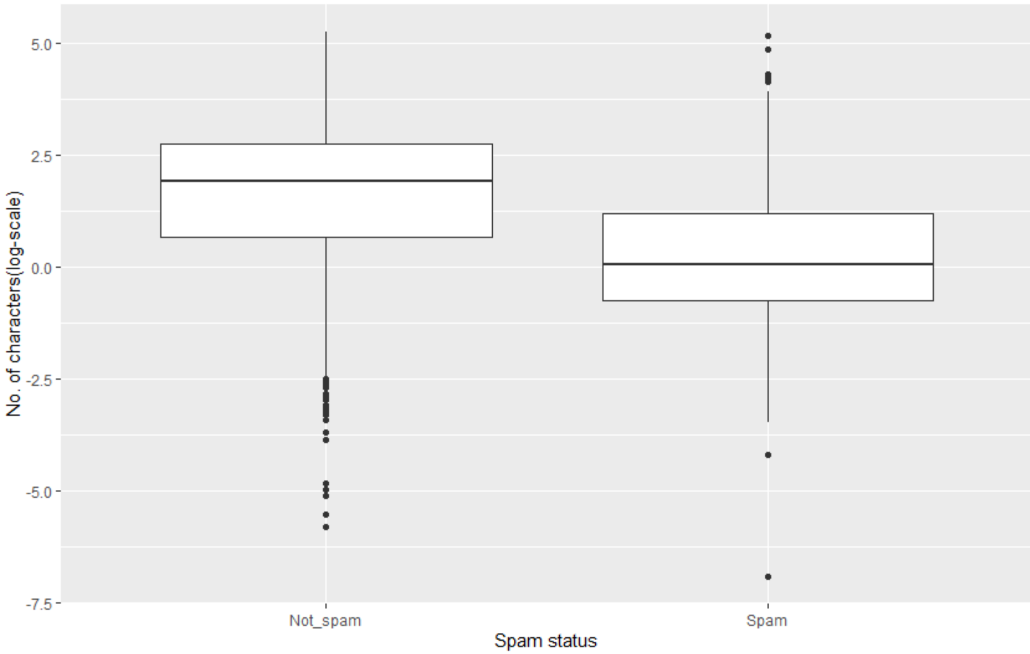
```
email %>%
  mutate(log_num_char = log(num_char)) %>%
  ggplot(aes(x = spam, y = log_num_char)) +
  geom_boxplot()+
  labs(x = "Spam status", y = "No. of characters(log-scale)")
```



EDA Case Study

 # It might be a good idea to change the scale of the y-axis.

```
email %>%
  mutate(log_num_char = log(num_char)) %>%
  ggplot(aes(x = spam, y = log_num_char)) +
  geom_boxplot()+
  labs(x = "Spam status", y = "No. of characters(log-scale)")
```



EDA Case Study

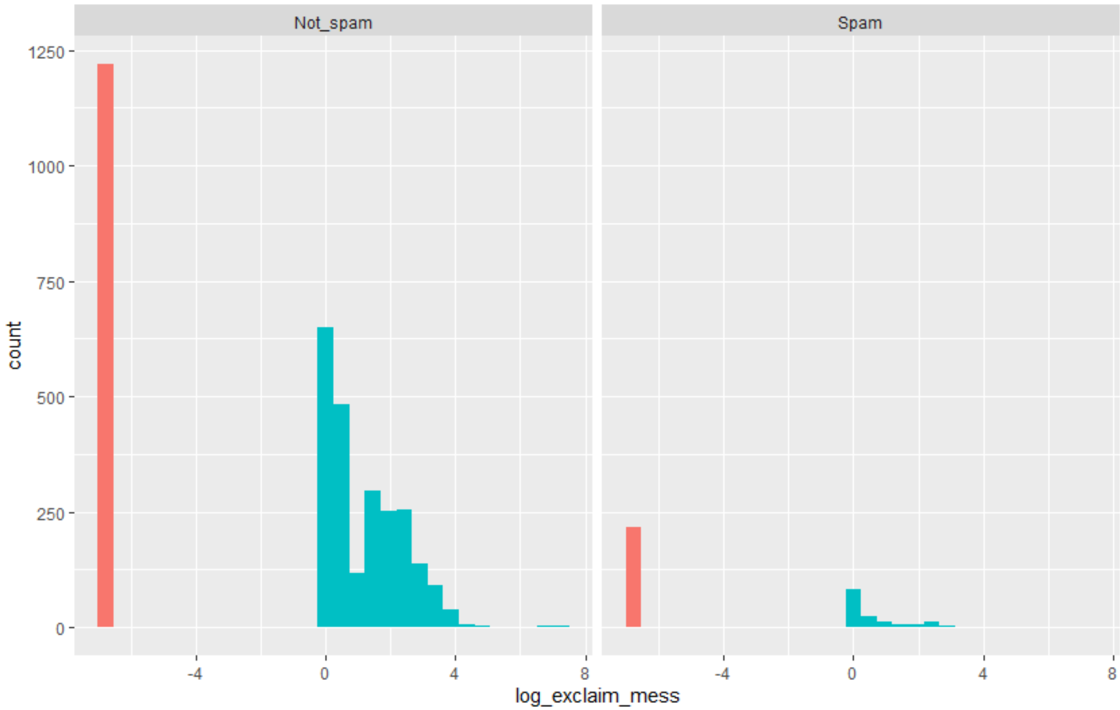
exclaim_mess vs spam

- We now explore another variable ***exclaim_mess***, the number of exclamation marks in the emails.
- We deploy summary statistics and visualizations to explore the differences.
- Note that many of the emails do not have any exclamation mark, hence log-transformation of the scale will return $-Inf$, corresponding to log(0).
- A way-around is to add a small quantity, say 0.001 to the values to prevent returning $-Inf$ $(-\infty)$.

```
email %>%
  group_by(spam) %>%
  summarise(MED = median(exclaim_mess),
            IQR = IQR(exclaim_mess), SD = sd(exclaim_mess))

##  A tibble: 2 × 4
##    spam       MED   IQR    SD
##    <fct>     <dbl> <dbl> <dbl>
## 1 Not_spam     1     5    47.6
## 2  Spam        0     1    79.9
```

EDA Case Study

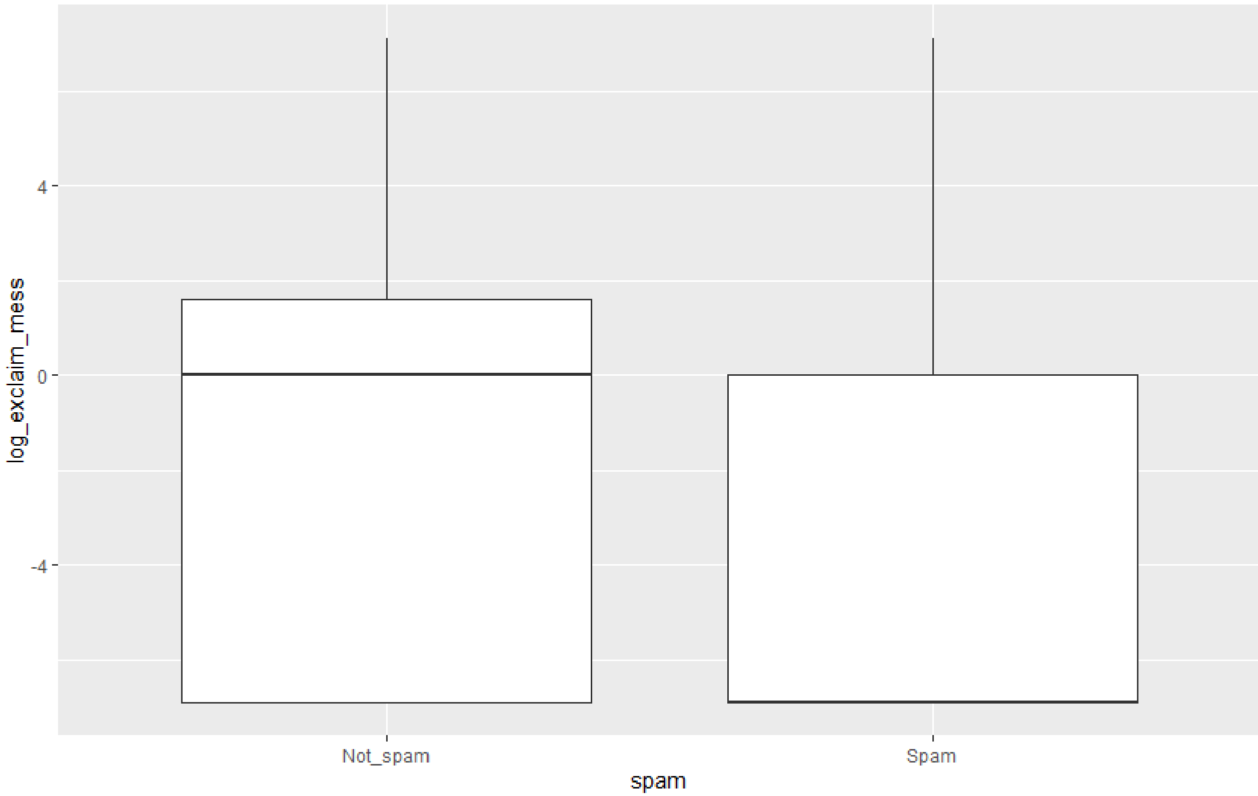
```
# Visualize
email %>%
  mutate(log_exclaim_mess = log(exclaim_mess + 0.001)) %>%
  mutate(custom_fill = as.factor(exclaim_mess > 0)) %>%
  ggplot(aes(x = log_exclaim_mess, fill = custom_fill)) +
  geom_histogram() + theme(legend.position = "none") + facet_wrap(~ spam)
```



EDA Case Study

Visualize: Box-plots

```
email %>%
  mutate(log_exclaim_mess = log(exclaim_mess + 0.001)) %>%
  ggplot(aes(x = spam, y = log_exclaim_mess)) +
  geom_boxplot()
```



EDA Case Study

Tackling inflated Zeroes

- We can see that both the non-spam and spam groups have the maximum counts for zeroes (no exclamation marks). This 'inflation' of zeroes affects the summary of data.

- Zero Inflation is common in many data applications.

- Suppose you are monitoring the sales of a new chocolate recently launched in the market.

- Data is collected from every customer entering a store, and number of the new chocolate bought by that customer is stored in a variable.

- A zero might not mean that a customer is choosing an alternate product; it may also mean that the customer is not interested in buying chocolates.

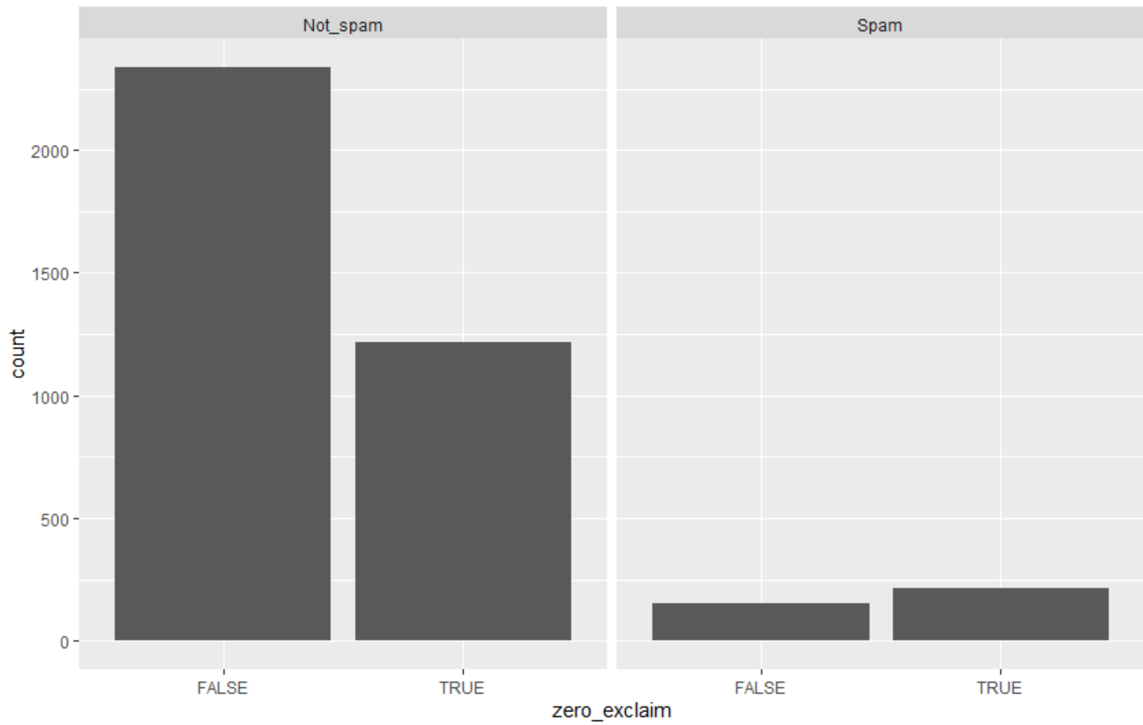
- One strategy is to categorize the variable into levels; for example, we can have two groups, one with zero exclamation marks, and the other with positive number of exclamation marks.

- Next, we can summarize and visualize the data for the two groups separately.

EDA Case Study

Tackling inflated Zeroes

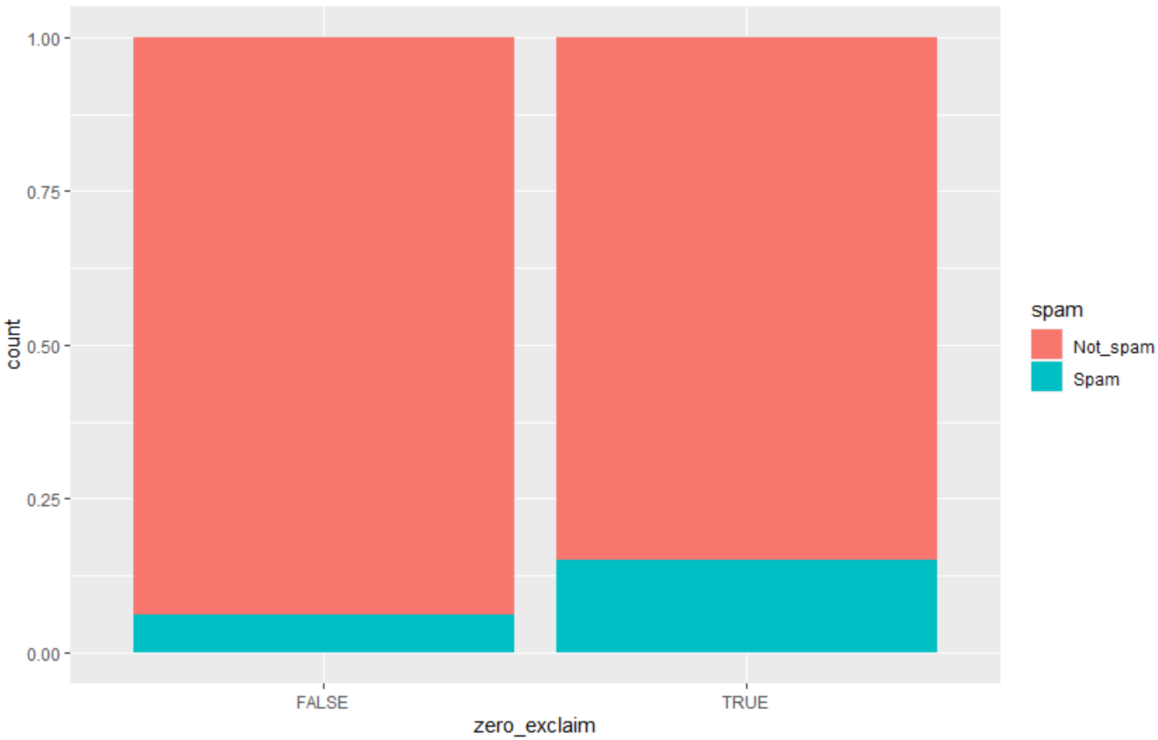
```
email %>%
mutate(zero_exclaim = (exclaim_mess == 0)) %>%
ggplot(aes(x = zero_exclaim)) +
geom_bar() +
facet_wrap(~spam)
```



EDA Case Study

Tackling inflated Zeroes

```
email %>%
mutate(zero_exclaim = (exclaim_mess == 0)) %>%
ggplot(aes(x = zero_exclaim, fill = spam)) +
geom_bar(position = "fill")
```



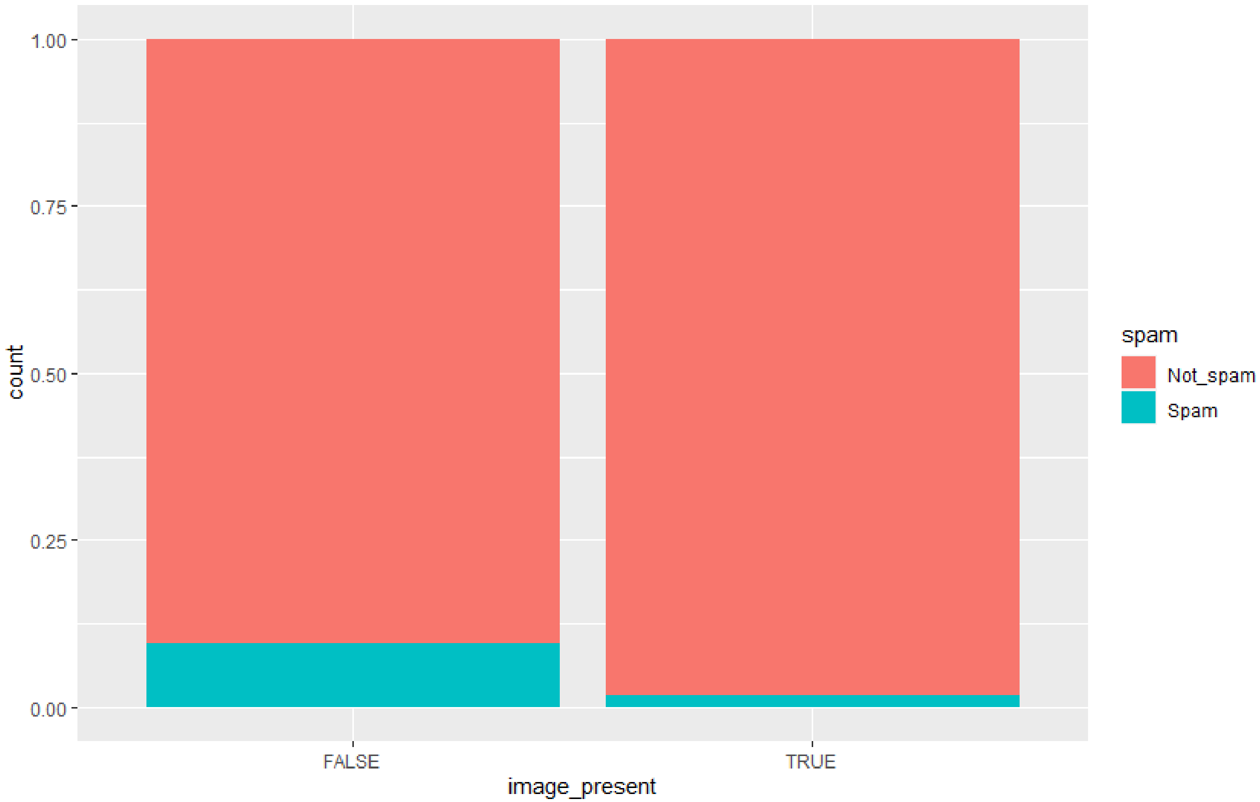
EDA Case Study

```
# Tackling inflated zeroes: example with the image variable
email %>%
  count(image)

## A tibble: 8 × 2
##   image    n
##   <dbl>  <int>
## 1    0    3811
## 2    1     76
## 3    2     17
## 4    3     11
## 5    4      2
## 6    5      2
## 7    9      1
## 8   20      1
```

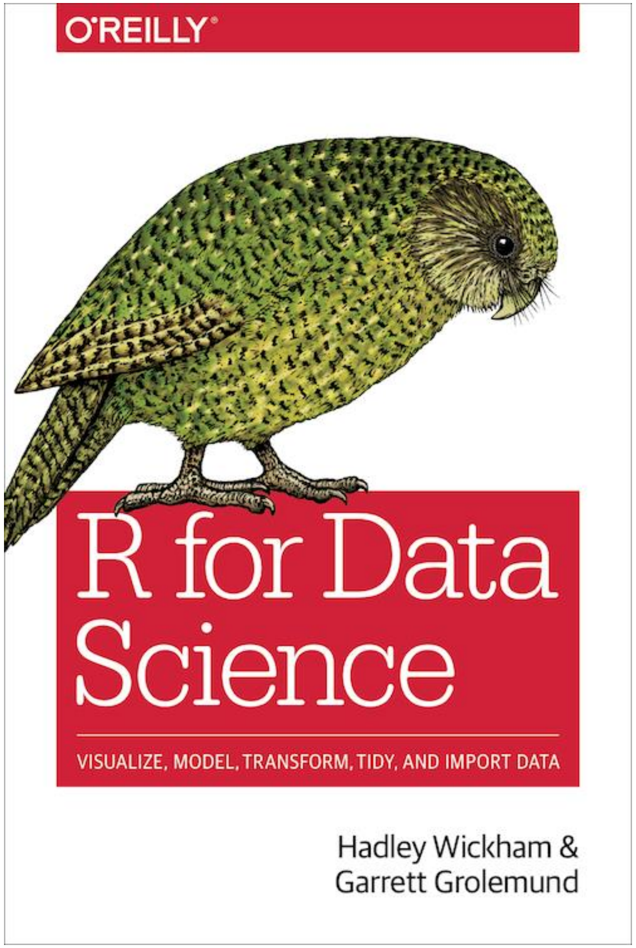
- There are 3811 images with no images attached.
- This poses a zero inflation problem.

EDA Case Study

```
email %>%
  mutate(image_present = (image > 0)) %>%
  ggplot(aes(x = image_present, fill = spam)) +
  geom_bar(position = "fill")
```



Reference



https://r4ds.had.co.nz/