STA5092Z EDA Lecture 3-4

Table of contents

Managing Data Frames with `dplyr` Package

```
#install.packages("tidyverse")
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.4.3

Warning: package 'tibble' was built under R version 4.4.3

Warning: package 'purrr' was built under R version 4.4.3

Warning: package 'lubridate' was built under R version 4.4.2

-- 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   3.5.2     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.1.0
-- 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 becor
```

Load tidyverse ecosystem

---

dplyr Verbs

`select()` function

We rarely need all variables at once. Early exploration is about reducing noise. For example, you received a hospital dataset with 120 variables, but to understand length of stay, you only want:

- age
- diagnosis category
- admission type

- length of stay

You select just those columns so that you can start reasoning without distraction. We use the `select()` function for this purpose.

> **Note**
>
> EDA begins by asking: Which variables matter for this question?

`filter()` function

You may want to look at a meaningful subset of the data. We often explore specific populations rather than the whole dataset. For example, we may want to examine only weekend crashes to see if severity differs. Or we may want to look only at students who passed prerequisites before analysing progression.

It is also useful to clear mistakes in your dataset. For example, some variables may not be negative or zero and therefore we may want to remove clearly impossible measurements (negative income, age = 200).

We use the `filter()` function for this purpose.

> **Note**
>
> EDA is iterative zooming — like using a microscope.

`arrange()` function

Arranging/sorting the data can reveal structure. Some patterns often appear only after sorting. For example, to see the inequality gradients, we should order the municipalities by unemployment rate. Or if we have a time series data, arranging observations by time helps us visually detect trends or regime shifts.

Moreover, sorting can reveal extreme cases. For example, if we had blood pressure data on patients, we may want to sort these to inspect the extreme cases.

We use the `arrange()` function for this purpose.

> **Note**
>
> Ordering is one of the oldest statistical tools — Tukey did this constantly.

`rename()` function

We often would like combine different datasets into a single dataframe. We can incorporate external datasets. To do this, we need to have consistent variable names per different sources. One way to ensure consistency, is to rename the variables with the names you understand the best. Real-world datasets are full of cryptic labels. If you don't understand the variable name, you don't understand the variable.

We use the `rename()` function for this purpose.

`mutate()` function

Raw variables are rarely the ones we need. We construct meaningful measures using the variables we have. Some examples:

- We only have data for height and weight and we would like to model BMI. Then we need to compute BMI from height and weight.
- We have number of crashes at intersections. But we may need to adjust this with the traffic flow at the intersection.
- We may want to convert timestamps into hour-of-day to detect behavioural cycles. Before and after traffic… etc.
- If we are working with countries, we should be calculating income per capita instead of total income.

We use the `mutate()` function for this purpose.

> **Note**
>
> EDA is not passive — we actively engineer understanding.

Pipeline operator %>% or |>

This is one of my favourite operators, it helps us follow sequentially what refinements are done to the data. The pipeline represents a narrative of thought. For example, we may want to do our analysis on the 2023 crash counts, focusing on pedestrian crashes. We may want to see the severity rate across districts.

We use the |> operator for this purpose to articulate our thinking process and so that anyone can follow our steps. Helps immensely with reproducability.

> **Note**
>
> EDA is not passive — we actively engineer understanding.

`group_by()` function

EDA is fundamentally about detecting heterogeneity. How does behaviour change across different sub-populations rather than assuming one global pattern? Most insights come from structured comparison, comparing like with like. Without grouping, everything is averaged into uselessness. For example, you may want to:

- Compare average exam marks by school.
- Examine mortality rates by age group.
- Look at sales patterns by month.
- Analyse crash counts by road type.

Let's say you compute the overall crash severity rate for Cape Town. But that mixes high-density CBD with informal settlements and high-speed arterials and residential suburbs. Grouping by road type or district may reveal arterials with fewer crashes but much more severe. We can reveal that CBD has many crashes but low severity. On the other hand informal areas have different time-of-day pattern entirely.

Without grouping, you would conclude something misleading like: "Crash severity is moderate."

This information can lead to the decision of mixture modelling.

> **Tip**
>
> Think about Simpson's Paradox

Family of join operations

We often would like to use some external datasets such as environmental, macroeconomic datasets. Some examples:

- We may want to link crash data to weather records to see if rainy/foggy days have an impact on the crash counts.
- Add census demographics to geographic regions.
- Merge student marks with attendance records.
- Strontium isotope fingerprinting is used for tackling cycad poaching. To provide a strontium predictive map, you may need to attach bedrock type to spatial measurements from a masters' thesis.

In summary

Function	Question it Answers
select()	What variables matter?
filter()	Which cases should we study?
arrange()	What happens if we order the data?
rename()	Do we understand what things mean?
mutate()	What new quantities reveal structure?
|>	What is the logical workflow?
group_by()	Where do groups differ?
joins	What additional context do we need?

EDA is not about running these functions. It is about interrogating reality — these verbs are simply the language we use to ask questions

R Examples

Create data frame 1

```
names = c("A", "B", "C", "D", "E", "F")
rcomputing = c(45, 40, 90, 94, 80, 65)
eda = c(63, 66, 75, 83, 80, 59)
sl = c(59, 56, 91, 92, 86, 67)
marks1 = data.frame(names, rcomputing, eda, sl)
marks1
```

```
  names rcomputing eda sl
1     A         45  63 59
2     B         40  66 56
3     C         90  75 91
4     D         94  83 92
5     E         80  80 86
6     F         65  59 67
```

select

```
subset = marks1 %>%
  select(-1) %>%

subset
```

filter

```
subset = marks1 %>%
  filter(rcomputing > 50 & eda > 60)

subset
```

```
  names rcomputing eda sl
1     C         90  75 91
2     D         94  83 92
3     E         80  80 86
```

arrange

```
subset = marks1 %>%
  arrange(eda)
subset
```

```
  names rcomputing eda sl
1     F         65  59 67
2     A         45  63 59
3     B         40  66 56
4     C         90  75 91
5     E         80  80 86
6     D         94  83 92
```

rename

```
marks1 = marks1 %>%
  rename(supervisedl = sl)
marks1
```

```
  names rcomputing eda supervisedl
1     A         45  63          59
2     B         40  66          56
3     C         90  75          91
4     D         94  83          92
5     E         80  80          86
6     F         65  59          67
```

mutate

```
marks1 = marks1 %>%
  mutate(st.eda = (eda - mean(eda)) / sd(eda))
marks1
```

```
  names rcomputing eda supervisedl    st.eda
1     A         45  63          59 -0.821648
2     B         40  66          56 -0.513530
```

```
3     C         90  75          91  0.410824
4     D         94  83          92  1.232472
5     E         80  80          86  0.924354
6     F         65  59          67 -1.232472
```

```
marks1 = marks1 %>%
  mutate(pass.rcomp = ifelse(rcomputing < 50, "fail", "pass"))

marks1
```

```
  names rcomputing eda supervisedl    st.eda pass.rcomp
1     A         45  63          59 -0.821648       fail
2     B         40  66          56 -0.513530       fail
3     C         90  75          91  0.410824       pass
4     D         94  83          92  1.232472       pass
5     E         80  80          86  0.924354       pass
6     F         65  59          67 -1.232472       pass
```

group by

```
means = marks1 %>%
  group_by(pass.rcomp) %>%
  summarise(mean(supervisedl))

means
```

```
# A tibble: 2 x 2
  pass.rcomp `mean(supervisedl)`
  <chr>                    <dbl>
1 fail                      57.5
2 pass                      84
```

```
count = marks1 %>%
  count(pass.rcomp)

count
```

```
  pass.rcomp n
1       fail 2
2       pass 4
```

distinct

```
distinctcat = marks1 %>%
  distinct(pass.rcomp)

distinctcat
```

```
  pass.rcomp
1       fail
2       pass
```

Create data frame 2

```
name = c("A", "B", "C", "E", "F", "G", "H")
rcomputing = c(45, 40, 90, 80, 65, 50, 51)
dsi = c(63, 66, 75, 80, 59, 65, 0)
marks2 = data.frame(name, rcomputing, dsi)
marks2
```

```
  name rcomputing dsi
1    A         45  63
2    B         40  66
3    C         90  75
4    E         80  80
5    F         65  59
6    G         50  65
7    H         51   0
```

inner_join

```
marks1 = marks1 %>%
  rename(name = names)

inner_join(marks1, marks2, by = c("name", "rcomputing"))
```

```
  name rcomputing eda supervisedl    st.eda pass.rcomp dsi
1    A         45  63          59 -0.821648       fail  63
2    B         40  66          56 -0.513530       fail  66
3    C         90  75          91  0.410824       pass  75
4    E         80  80          86  0.924354       pass  80
5    F         65  59          67 -1.232472       pass  59
```

left_join

```
newdata = full_join(marks1, marks2, by = c("name", "rcomputing"))

newdata
```

```
  name rcomputing eda supervisedl    st.eda pass.rcomp dsi
1    A         45  63          59 -0.821648       fail  63
2    B         40  66          56 -0.513530       fail  66
3    C         90  75          91  0.410824       pass  75
4    D         94  83          92  1.232472       pass  NA
5    E         80  80          86  0.924354       pass  80
6    F         65  59          67 -1.232472       pass  59
7    G         50  NA          NA        NA       <NA>  65
8    H         51  NA          NA        NA       <NA>   0
```

Class exercise

Please navigate to the class exercise to see the description of the data and the questioned to be answered. This needs to be submitted by 20th of Feb 2026 before 4pm.

Steps:

- Join the google classroom with this link
- Open a github account with the email account you used for the google classroom.
- This will add you to the Github classroom where you will submit your codes.