Eltecon Data Science Course by Emarsys

EDA and Data Visualization

András Bérczi

September 15, 2021

About me

- Background in Economics
- Works as Data Scientist @ Emarsys

Today's topics

- Coding guidelines
- Exploratory Data Analysis (EDA)
- Data visualization with ggplot

Section 1

Coding guidelines

Why writing good code is important?

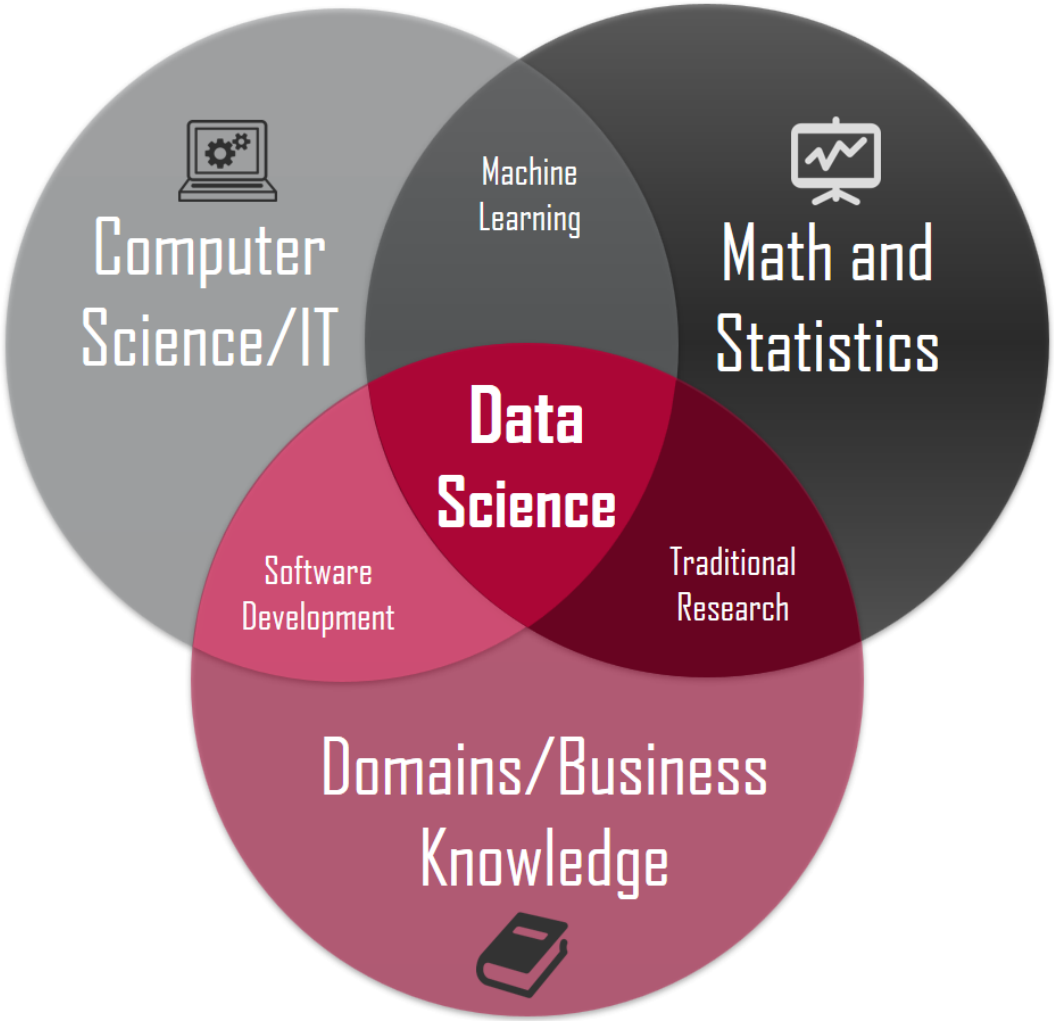


Figure 1: https://www.inf.elte.hu/en/content/data-science-in-computer-science-msc.t.1732?m=361

Try to make your code 'clean'!

- **D**(on't) **R**(epeat) **Y**(ourself)!
- Make your code readable!
- Use style guides!

Keep it DRY!

```
average_of_A <- sum(data['A_column']) / nrow(data['A_column'])
variance_of_A <- sum((data['A_column'] - average_of_A) ^ 2) /
    (nrow(data['A_column']) - 1)

... some line of codes ...

average_of_B <- sum(data['B_column']) / nrow(data['B_column'])
variance_of_B <- sum((data['B_column'] - average_of_B) ^ 2) /
    (nrow(data['B_column']) - 1)

... some line of codes ...

average_of_C <- sum(data['C_column']) / nrow(data['C_column'])
variance_of_C <- sum((data['C_column'] - average_of_C) ^ 2) /
    (nrow(data['C_column']) - 1)
```

Move repeating code into functions!

```
# Define a function
variance <- function(values) {
    N <- length(values)
    average <- sum(values) / N

    return ((values - N) ^ 2) / N
}
```

Use the function instead!

```
variance_of_A <- variance(data['A_column'])

... some line of codes ...

variance_of_B <- variance(data['B_column'])

... some line of codes ...

variance_of_C <- variance(data['C_column'])
```

Watch out for messy code!

```
dataset <- "customers"
if (isDatasetUpToDate(dataset, max_hour_delay = 14 * 24, filepath = "emsconnectr_data/customers.csv")) {
    customers <- getListOfAllCustomers()
} else {
    customers <- getListOfAllCustomers(refetch = TRUE)
    saveRefreshTimeOfDataset(dataset)
}

effect_dt <- sto_performance %>%
    .[, versions := segment] %>%
    .[, segment := NULL] %>%
    calculateEffect("open_rate", by_cols = c("customer_id", "customer_name")) %>%

if (metric == "first_2_hour_open_rate") {
    metric <- "open_rate"
}

num_col_name <- paste0("num_", gsub("_rate", "", "open_rate"))

effect_dt %>%
    copy() %>%
    .[, num_delivered := as.numeric(num_delivered_control) + as.numeric(num_delivered_treatment)] %>%
    .[, total_num := get(str_c(num_col_name, "_control")) + get(str_c(num_col_name, "_treatment"))] %>%
    setnames(paste0("open_rate", c("_control", "_treatment")), c("control_rate", "treatment_rate")) %>%
    setnames("open_rate_uplift", "treatment_uplift") %>%
    .[, sto_uplift := round(treatment_uplift, 4)] %>%
    .[, .(customer_id, sto_uplift)]

adoption_metrics_dt <- merge(customers, sto_uplift, by = "customer_id", all.x = TRUE)

id_col <- ifelse("customer_id" %in% names(adoption_metrics_dt), "customer_id", "customer_name")
real_customers <- getFreshListOfAllCustomers(exclude_test = TRUE)[, .SD, .SDcols = id_col]

lifecycle_for_relevant_customers <- merge(adoption_metrics_dt, real_customers, by = id_col) %>%
    .[, lifecycle := dplyr::case_when(
        is.na(num_sto_campaign) | num_sto_campaign == 0 ~ "Never used (product is on, but never used)",
        num_sto_campaign <= 20 | is.na(sto_uplift) ~ "Testing",
        days_since_last_sto_campaign > 15 ~ "Abandoned use",
        sto_uplift > 0 ~ "Seeing value",
        TRUE ~ "Using"
    )] %>%
    .[, .(customer_id, customer_name, lifecycle)] %>%
    .[order(customer_name)]

```

Make it easy(er) to read!

```r
customers <- getFreshListOfAllCustomers(exclude_test = FALSE)

sto_uplift <- sto_performance %>%
    .[, versions := segment] %>%
    .[, segment := NULL] %>%
    calculateEffect("open_rate", by_cols = c("customer_id", "customer_name")) %>%
    transformColsForSimplePlot("open_rate") %>%
    .[, sto_uplift := round(treatment_uplift, 4)] %>%
    .[, .(customer_id, sto_uplift)]

adoption_metrics_dt <- merge(customers, sto_uplift, by = "customer_id", all.x = TRUE)

lifecycle_for_relevant_customers <- adoption_metrics_dt %>%
    filterInternalAccounts() %>%
    calculateLifecycleForAdoptionMetrics_()
```

Quick recap

Structure of functions:

- name the function
- define input parameters / arguments
  - set default values if needed
- write down the logic
- provide a return value

How to write functions

```
name_of_the_function <- function(argument_1, argument_2, ...)
    Body of the function

    last line is the return value
}
```

Style guide

- tidyverse
- Google (- Your own within your company)

Why use style guides?

1. Provides consistency
2. Easier to read (for you and others - for us to grade your homework :))
3. Easier to write good - you need to make fewer decisions

What's a style guide about?

- **naming** of files, functions, objects, etc.
- **structure** of files, functions
- **syntax** - spacing, brackets, inlining etc.

Packages supporting style guides

- lintr
- styler

Your turn!

1. Write a function that calculates the average of mpg from the mtcars dataset and prints: "The average of mpg is X"!
2. Use lintr and styler to check your code!

Section 2

Exploratory Data Analysis

What's EDA about?

- Put it in a usable format
- Clean it
- Understand your data/features so it can help answer your question

This usually takes much more time than creating models for prediction

Data cleaning

- Check if features are in the correct format
- Are there missing values? Is there anything we can do with them?
- Are there 'suspicious' values? (eg.: birthday = 1900-01-01)

Understanding your data

- Understand distribution of features
- Do they make sense?
- Do they help answering your question?

Example on Manhattan house price

You can check the code later at `week2/EDA.R`

Section 3

Data visualization with ggplot2 (and plotly)

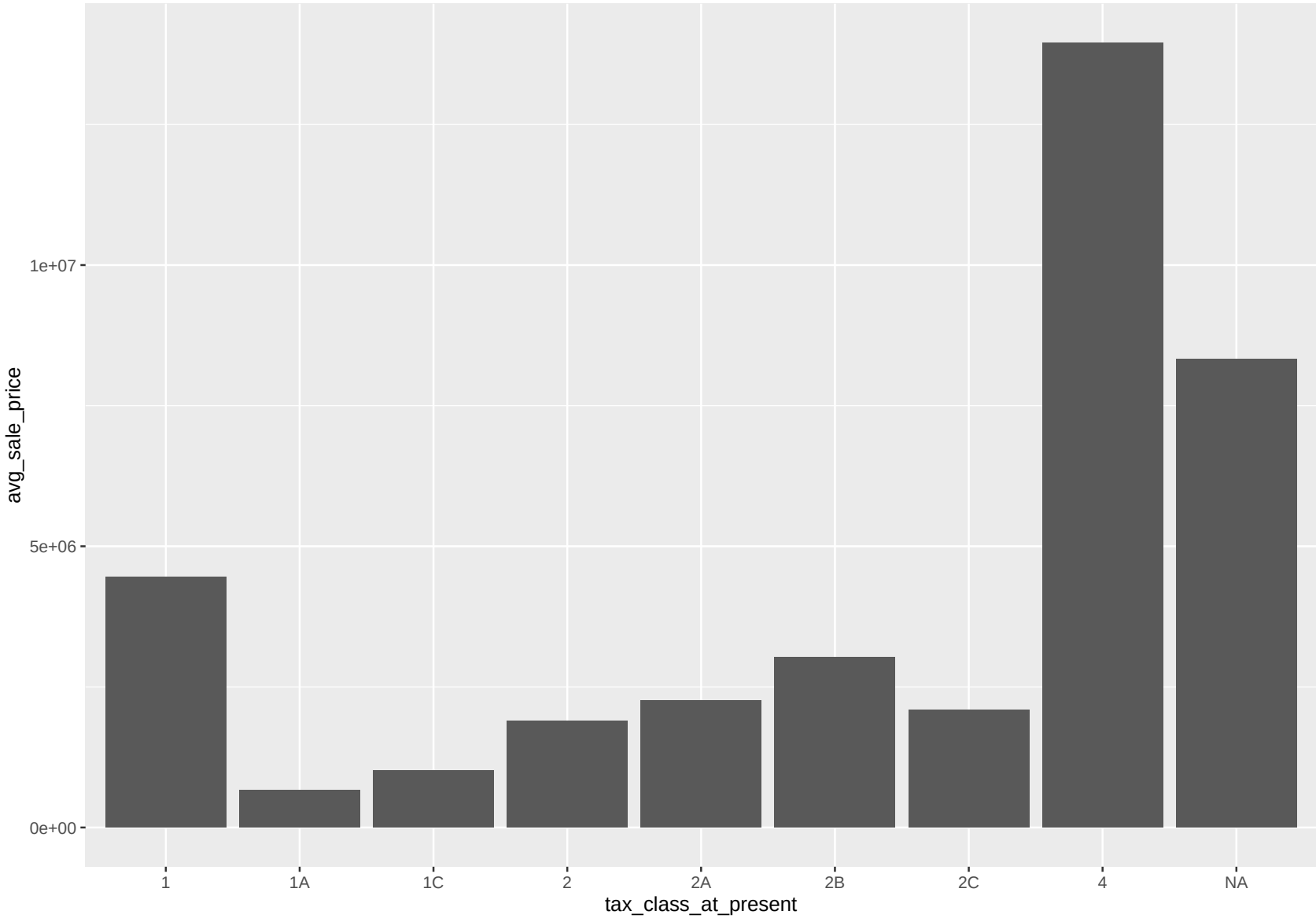
“Warning”

- I’m not a ‘data visualization expert’
- But I did make a lot of visualizations for different kinds of audiences

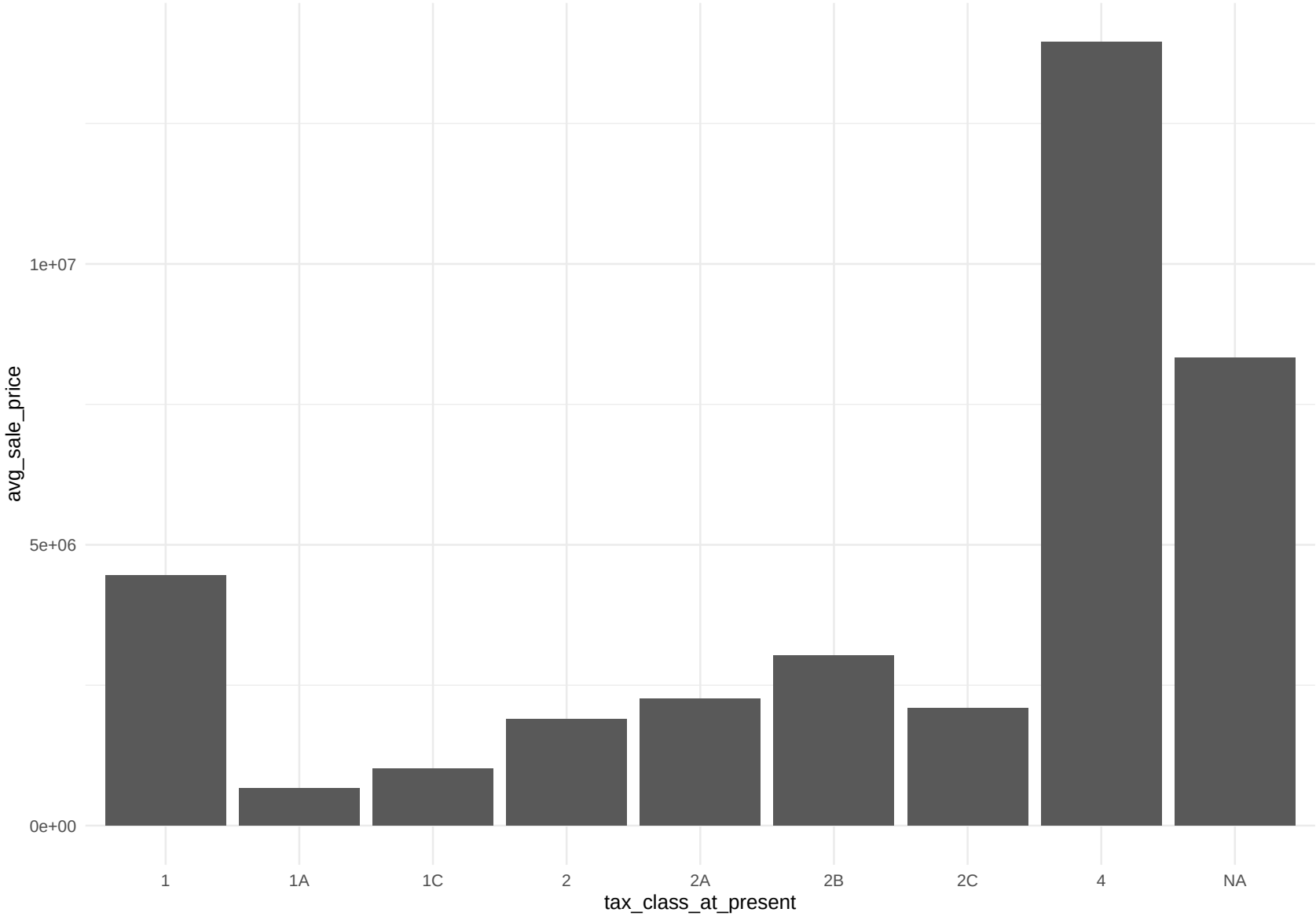
What to look out for when creating visualizations?

- Should be easy to read - easy to understand the message
- Know your audience
- Keep it simple
- Add only relevant information

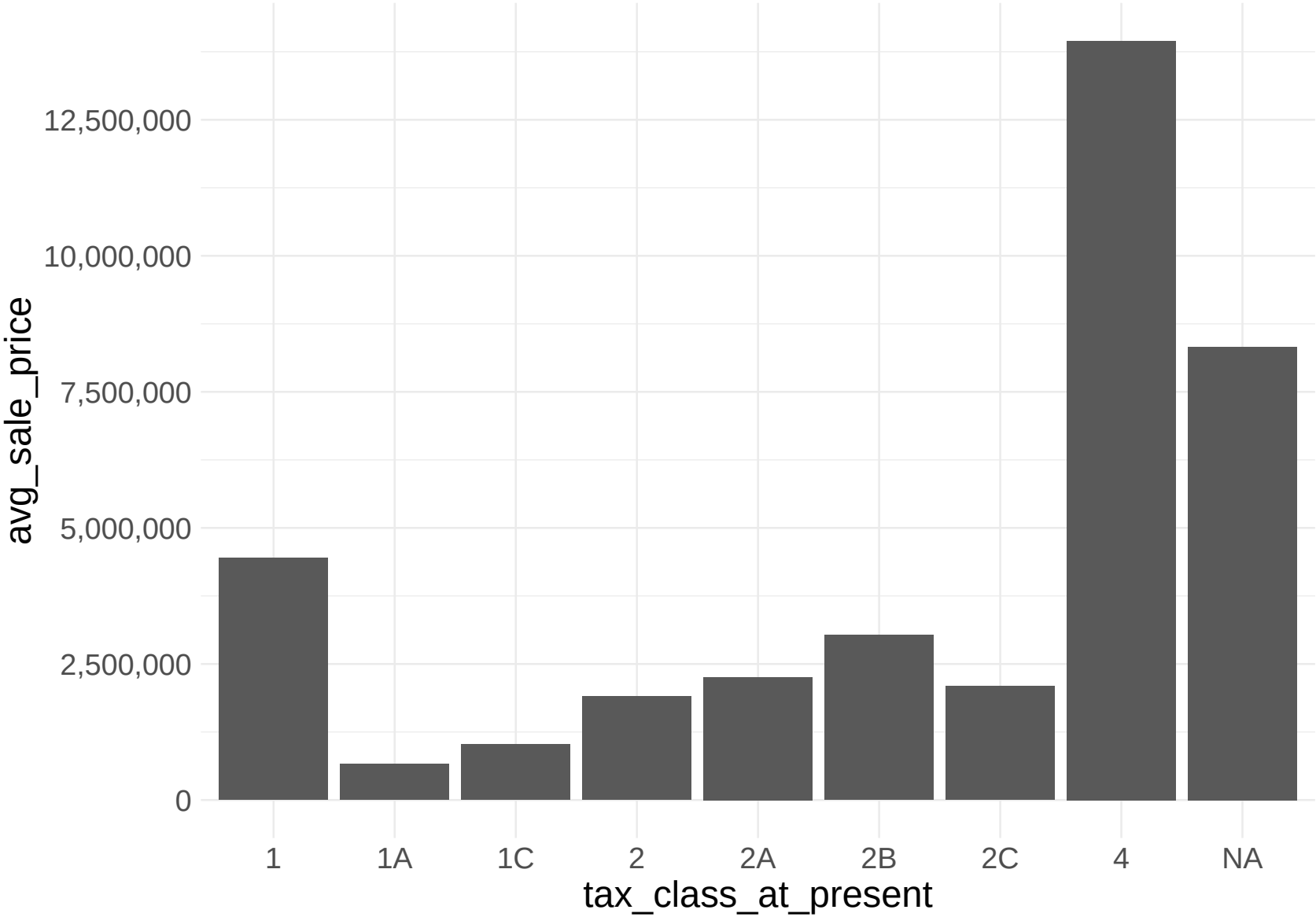
Visualization examples



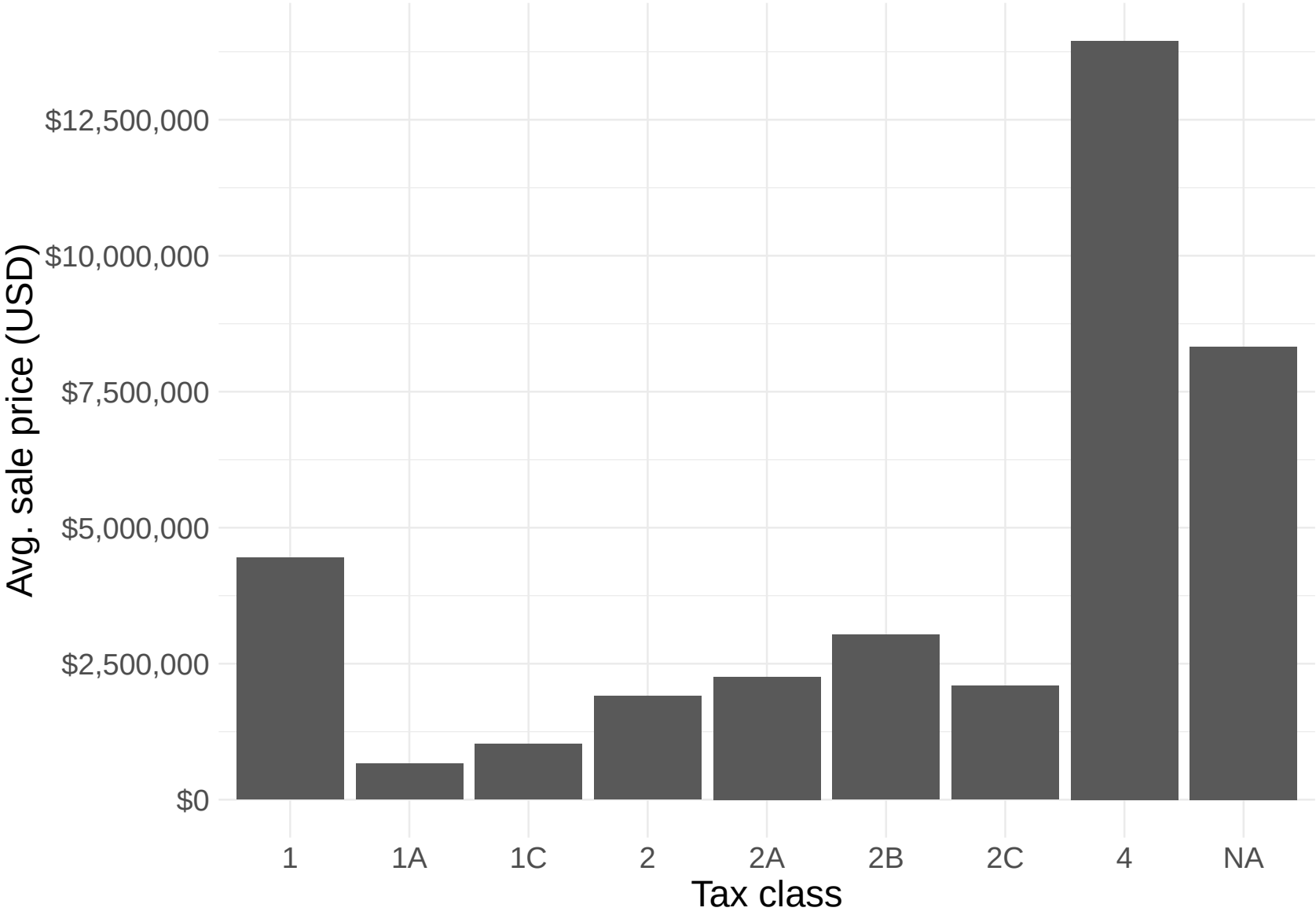
Change theme



Axis labels should be easy to read and understand



Change axis titles



Add title to your plot

Visualizing with ggplot

- 'gg' = grammar of graphics
- we map data frame columns to different parts (aestethics) of the plot (color, shape, size, etc.)
- you can add layers to improve your plot

How ggplot looks like

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

source: https://r4ds.had.co.nz/data-visualisation.html#the-layered-grammar-of-graphics

How it works in the background

1. **`geom_bar()`** begins with the **`diamonds`** data set

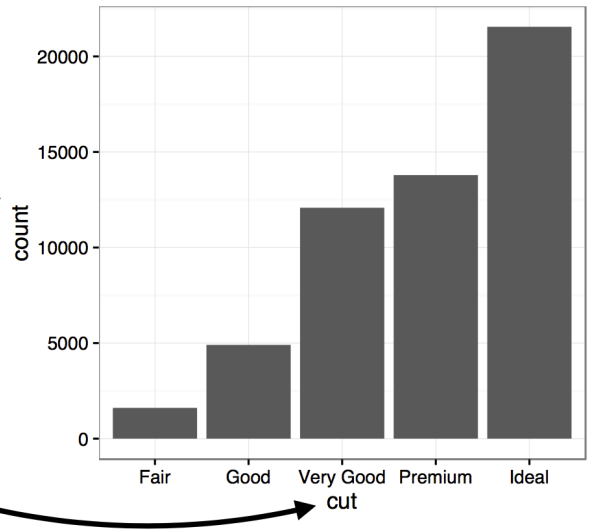
2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

stat_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1



Why plot things?

- Understand how features are distributed
- Helps to spot outliers / mistakes in data
- Easier to present results (than in tabular format)

Let's check our your data by visualizing it!

You can check the code later at `week2/ggplot.R`

Homework for next week

Create 2 plots from a dataset of your choice! One should include a categorical and the other should have a continuous variable.

Points will be given based on how understandable the plot is and also based on how 'nice' it is.