ISA 401: Business Intelligence & Data Visualization

09: Towards Technically Correct and Consistent Data

Fadel M. Megahed, PhD

Professor of Information Systems and Business Analytics
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
Automated Scheduler for Office Hours

Fall 2024

Quick Refresher from Last Class

- ✅ Define tidy data
- ✅ Perform pivot and rectangling operations in R

Learning Objectives for Today's Class

- Explain the concept of "technically correct" data
- Examine the different column types and their summaries
- Recode factors and convert dates
- Manipulate characters
- Explain the concept of "consistent" data

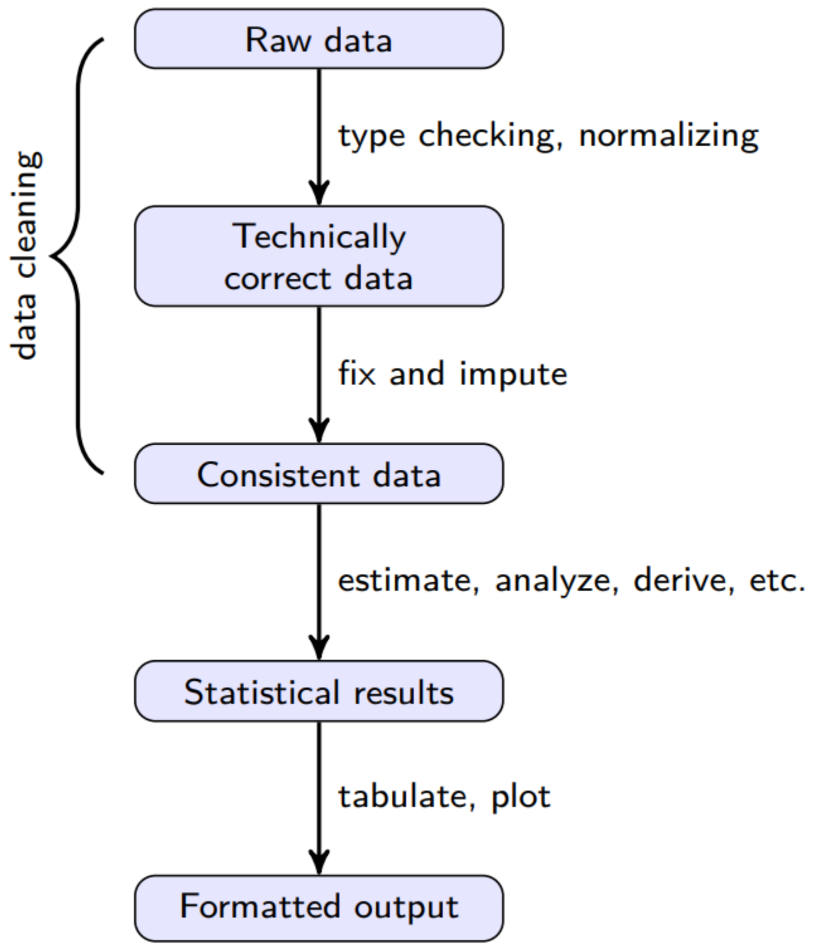
🔍 An Alternative Pipeline for Statistical Data Analysis

Data Analysis: A Crowd-Sourced Definition

Wikipedia's data analysis article defines it to be the **process** of:

> inspecting, cleansing, transforming and modeling data with the goal of discovering useful information, informing conclusion and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, and is used in different business, science, and social science domains.

Post Tidy Data: The Data Analysis Value Chain



Source: De Jonge and Van Der Loo (2013), An introduction to data cleaning with R. Please click on the link and read p. 7-8.

Technically Correct Data

> Raw data files may lack headers, contain wrong data types (e.g. numbers stored as strings), wrong category labels, unknown or unexpected character encoding and so on.
>
> **Technically correct data** is the state in which data can be read into an R data.frame, with correct **name**, **types** and **labels**, without further trouble. However, that **does not mean that the values are error-free or complete**. --- De Jonge and Van Der Loo (2013)

Technically Correct Data

Functions for Cleaning/Renaming Variables

type	package	function()	description
cleaning names	janitor	clean_names()	Resulting names are unique, consisting only of the _ character, numbers, and letters
renaming	base R	names()[colNum(s)]	Will rename a column by its number, e.g., names(df)[1] = ‘new_name’ will rename the first column in df to ‘new_name’
renaming	dplyr	rename(df, new_name = old_name)	Will rename a the column titled ‘old_name’ to ‘new_name’ You can pass a vector of names to rename multiple cols, e.g., rename(df, c(new_name1 = old_name1, new_name2 = old_name2))

An Example for Renaming Columns: The Data

```r
iris_tbl = tibble::tibble(iris) # convert to tibble
print(iris_tbl, # printing it
      width= 80) # make it wider
```

```
## # A tibble: 150 × 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ℹ 140 more rows
```

An Example for Renaming Columns: Code

```
iris_tbl = janitor::clean_names(iris_tbl) #<< # overwrite w/ clean_names
names(iris_tbl) # seeing new names
```

```
## [1] "sepal_length" "sepal_width"  "petal_length" "petal_width"  "species"
```

```
names(iris_tbl)[names(iris_tbl)=='species'] = 'type'
names(iris_tbl)
```

```
## [1] "sepal_length" "sepal_width"  "petal_length" "petal_width"  "type"
```

```
# renaming with rename
iris_tbl = dplyr::rename(
  iris_tbl, c(sepal_l = sepal_length, sepal_w = sepal_width)
  )
names(iris_tbl)
```

```
## [1] "sepal_l"      "sepal_w"      "petal_length" "petal_width"  "type"
```

Functions for Examining Column Classes

package	function()	description
base R	class(iris_tbl$sepal_length)	Will return the class of the column titled 'sepal_length'
base R	sapply(iris_tbl, class)	Will apply the class() function to all columns in the iris_tbl
base R	str(iris_tbl)	Will return the internal structure of the iris_tbl, which includes the dimensions of the df/tibble, column names, column types and first few observation values
purrr	map_chr(iris_tbl, class)	The tidyverse equivalent to sapply()
dplyr	glimpse(iris_tbl)	glimpse() is like a transposed version of print(): columns run down the page, and data runs across. It's a little like str() but shows you as much data as possible.
skimr	skim(iris_tbl)	returns variable types, names, statistical summaries & histograms of numeric variables.
DataExplorer	plot_str(iris_tbl)	while more useful for lists, returns a plot of the internal structure of your data.

Functions for Changing Column Classes

- **Convert to Factor:**
  - To convert a numeric vector to factors, we will typically use the chain of `as_factor( as_character() )`.
  - To convert a character vector to factors, we will simply use the `as_factor()`.
- **Convert to Date:**
  - To convert a character vector to date, you should use an appropriate function or chain of functions from the `lubridate` 📦.
- To change multiple columns at once, we will resort to the `mutate_at()` or `mutate_if` functions from the `dplyr` 📦.
- **For any of the above operations, we will need to overwrite the original column's data.**

Functions for Labeling Factors

package	function()	description
base R	levels(df$fct_column)= c()	Passing a character vector to recode the levels of a factor column. **Order matters here so you need to check the order of both the levels and the inputs to c()**
forcats	df $fct_column = fct_recode(df$ fct_column, fruit = 'apple')	If the factor colum contained a level called apple, it will be renamed to fruit. Multiple levels can be changed per `?fct_recode`

Towards Consistent Data

Consistent Data

> **Consistent data** is the stage where data is ready for statistical inference. It is the data that most statistical theories use as a starting point. Ideally, such theories can still be applied without taking previous data cleaning steps into account. In practice however, data cleaning methods like imputation of missing values will influence statistical results and so must be accounted for in the following analyses or interpretation thereof. --- De Jonge and Van Der Loo (2013)

Features of Consistent Data

- [Usually] no missing data
- Values within and across columns meet your expected "rules" for the data
- It is what we will colloquially refer to as "clean" data

Considerations to Achieving Consistent Data

- Do you have **missing values** in any of the variables?
- Are the **values for each variable** reasonable? e.g., do you have a negative age?
- Are the values **across the columns for a given observation reasonable**? e.g., does the count of registered and casual riders add up to the total number of riders?

Some Useful Packages in R

- The pointblank 📦 for data validation.
- The editrules and deducorrect 📦 for localizing errors and performing some basic imputations of data.
- Note that any efforts to achieve consistent data **will have to be:**
  - Dataset dependent; and
  - Research question dependent

A Demo Using a Popular Bikesharing Dataset

Tasks

In this in-class, demo we will:

- Read the bike_sharing_data.csv data. The description of all fields (with the exception of the `sources` column) can be found at Kaggle dataset description.
- Check whether the dataset is **technically correct**, **generate a report on how each variable meets/contradicts its expected data type/class**, and **fix any observed issue(s)**.
- We will create a set of **consistency rules** for variables and examine the instances (rows) when any of the rules are violated.
- We will generate an additional report on the status of our **data's consistency**.

Data Validation Pipeline: `pointblank`

Pointblank Validation

[2024-09-30|10:16:12]

TIBBLE	bike_tbl	WARN	0.01	STOP	—	NOTIFY	0.01

	STEP	COLUMNS	VALUES	TBL	EVAL	UNITS	PASS	FAIL	W	S	N	EXT
1	D `col_is_date()`	datetime	—	○→	✓	1	0 0.00	1 1.00	●	—	●	—
2	f `col_is_factor()`	season	—	○→	✓	1	0 0.00	1 1.00	●	—	●	—
3	f `col_is_factor()`	holiday	—	○→	✓	1	0 0.00	1 1.00	●	—	●	—
4	f `col_is_factor()`	workingday	—	○→	✓	1	0 0.00	1 1.00	●	—	●	—
5	f `col_is_factor()`	weather	—	○→	✓	1	0 0.00	1 1.00	●	—	●	—
6	d `col_is_numeric()`	temp	—	○→	✓	1	1	0	○	—	○	—

Changing the Column Types that Need "Fixing"

Please refer to our in-class code.

Examining the Consistency of the Data

```
library(pointblank)

act <- action_levels(warn_at = 0.01, notify_at = 0.01, stop_at = NULL)

agent <-
  create_agent(bike_tbl, actions = act) |>
  col_vals_between(columns = vars(temp, atemp, humidity, windspeed), 0, 100) |>
  col_vals_gte(columns = vars(casual, registered), 0) |>
  col_vals_gt(columns = vars(count), 0) |>
  col_is_factor(columns = vars(season, holiday, workingday, weather)) |>
  col_vals_in_set(columns = vars(hour), set = seq(0, 23, by = 1)) |>
  col_vals_not_null(columns = names(bike_tbl)) |>
  col_vals_expr(expr(count == casual + registered) ) |>
  col_vals_expr(expr = expr(sources %in% c('AD campaign', 'ad campaign'))) |>
  col_vals_expr(expr = ~ str_detect(sources, pattern = 'google'),
                label = 'non google sources')

res <- interrogate(agent, sample_limit = nrow(bike_tbl))
res |> export_report(filename = 'report_consistency.html')
```

Examining the Consistency of the Data

Pointblank Validation

[2024-09-30|10:16:15]

TIBBLE	bike_tbl	WARN	0.01	STOP	—	NOTIFY	0.01

STEP		COLUMNS	VALUES	TBL	EVAL	UNITS	PASS	FAIL	W	S	N	EXT
1	`col_vals_between()`	temp	[0, 100]	○→	✓	17K	17K 1.00	0 0.00	○	—	○	—
2	`col_vals_between()`	atemp	[0, 100]	○→	✓	17K	17K 1.00	0 0.00	○	—	○	—
3	`col_vals_between()`	humidity	[0, 100]	○→	✓	17K	17K 0.99	1 0.01	○	—	○	CSV
4	`col_vals_between()`	windspeed	[0, 100]	○→	✓	17K	17K 1.00	0 0.00	○	—	○	—
5	`col_vals_gte()`	casual	0	○→	✓	17K	17K 1.00	0 0.00	○	—	○	—
6	`col_vals_gte()`	registered	0	○→	✓	17K	17K	0	○	—	○	—

Changing the Column Values that Need "Fixing"

Please refer to our in-class code.

Recap

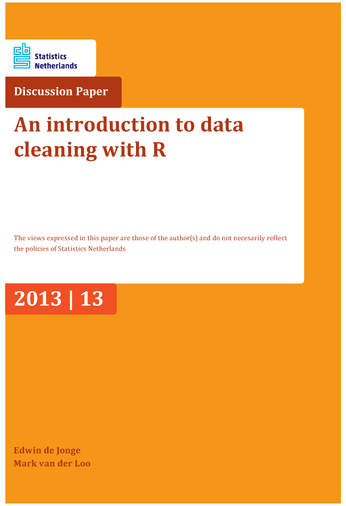
Summary of Main Points

By now, you should be able to do the following:

- Explain the concept of "technically correct" data
- Examine the different column types and their summaries
- Recode factors and convert dates
- Manipulate characters
- Explain the concept of "consistent" data

Things to Do Prior to Next Class

Please go through the following two supplementary readings and complete assignment 07.


Statistics Netherlands

Discussion Paper

An introduction to data cleaning with R

The views expressed in this paper are those of the author(s) and do not necesarily reflect the policies of Statistics Netherlands

2013 | 13

Edwin de Jonge
Mark van der Loo

- From raw data to technically correct data
- From technically correct to consistent data

- An introduction to the funmodeling package
- Data preperation with funmodeling