

# ISA 401: Business Intelligence & Data Visualization

## 10: Towards Technically Correct and Consistent Data

Fadel M. Megahed, PhD

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

Twitter: [FadelMegahed](#)

GitHub: [fmegahed](#)

Email: [fmegahed@miamioh.edu](mailto:fmegahed@miamioh.edu)

Office Hours: [Automated Scheduler for Virtual Office Hours](#)

Spring 2022

# Quick Refresher from Last Class

- ✓ Define tidy data
- ✓ Perform pivot and rectangling operations in 

# Learning Objectives for Today's Class

- Examine the different column types and their summaries
- Recode factors and convert dates
- Manipulate characters (string normalization and approximate string matching)



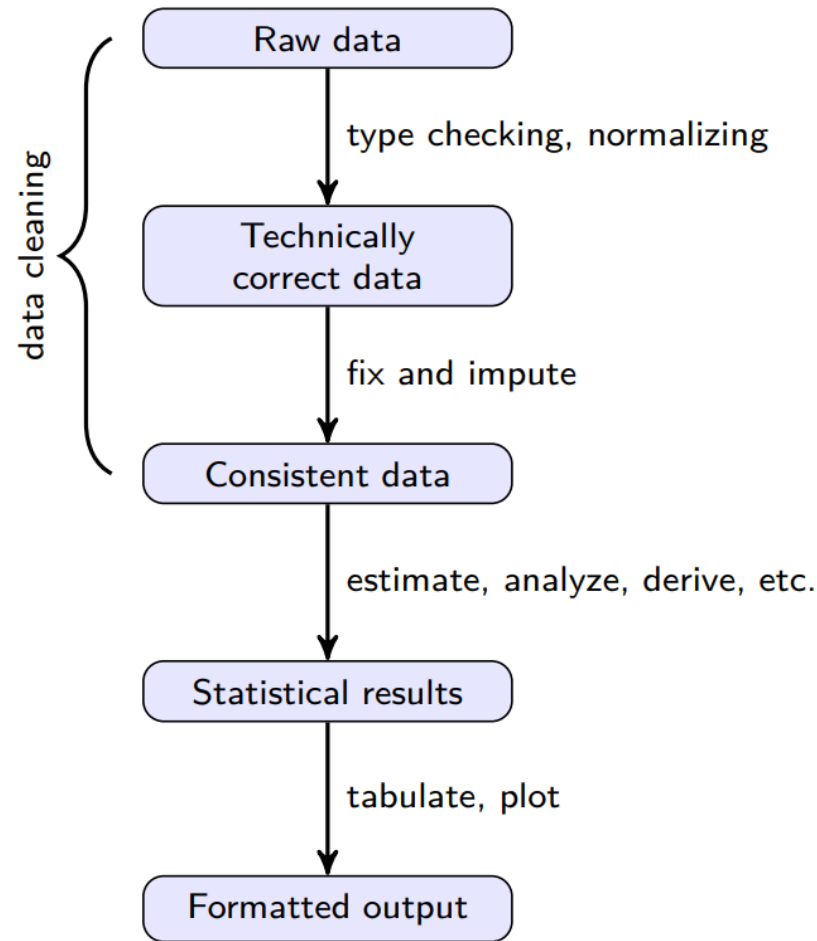
# An Alternative Pipeline for Statistical Data Analysis

# 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



# 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
<b>cleaning names</b>	janitor	clean_names()	Resulting names are unique, consisting only of the _ character, numbers, and letters
<b>renaming</b>	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'
<b>renaming</b>	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

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

```
## # A tibble: 150 x 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
## # ... with 140 more rows
```

# An Example for Renaming Columns: Code

```
iris_tbl = 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 = 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"
```

```
# renaming with rename and two-way pipe  
iris_tbl %<>% rename(c(sepal_length = sepal_l, sepal_width = sepal_w))  
names(iris_tbl)
```

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

# Functions for Examining Column Classes

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

# Non-Graded Class Activity: Exploring the Previous Functions

Activity	Your Solution
----------	---------------

*Over the next 5 minutes, please examine the following:*

- Use `pacman::p_load()` to install/load the previous packages. Notes:
  - `base` R packages are loaded with each R session.
  - `purrr` and `dplyr` are part of the `tidyverse` ecosystem.
- Run the following code: `iris_tbl = tibble(iris) %>% clean_names()` and examine the 7 approaches highlighted in the previous slide.

# Non-Graded Class Activity: Exploring the Previous Functions

Activity	Your Solution
----------	---------------



- Please write a script file in RStudio to examine the different approaches.
- The purpose of this activity is two-fold:
  - Making sure that you can make these different functions work.
  - Allowing you to develop a preference for quickly examining different datasets.
- As promised, I will **never** force you to follow a specific approach.

# Functions for Changing Column Classes

- **Convert to Factor:**

- To convert a numeric vector to factors, we will typically use the chain of `as_character() %>% as_factor()`.
- 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	<code>levels(df\$fct_column)= c()</code>	Passing a character vector to recode the levels of a factor column. <b>Order matters here so you need to check the order of both the levels and the inputs to c()</b>
forcats	<code>df\$fct_column = fct_recode(df\$fct_column, fruit = 'apple')</code>	If the factor column contained a level called apple, it will be renamed to fruit. Multiple levels can be changed per <code>?fct_recode</code>



# 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

[2022-02-22|14:39:52]

TIBBLE

bike\_tbl

WARN

0.01

STOP

—

NOTIFY

0.01

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

# Changing the Column Types that Need "Fixing"

Please refer to our in-class code.

# Examining the Consistency of the Data

## Pointblank Validation

[2022-02-22|14:48:54]

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 1.00	0 0.00		—		—



# Changing the Column Values that Need "Fixing"

Please refer to our in-class code.

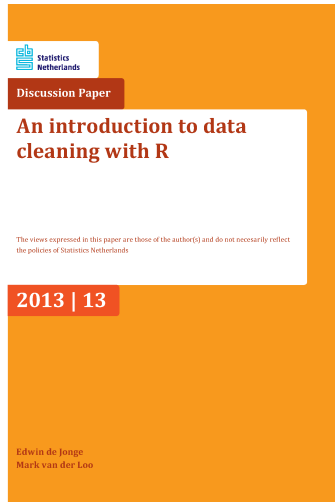
# Recap

# Summary of Main Points

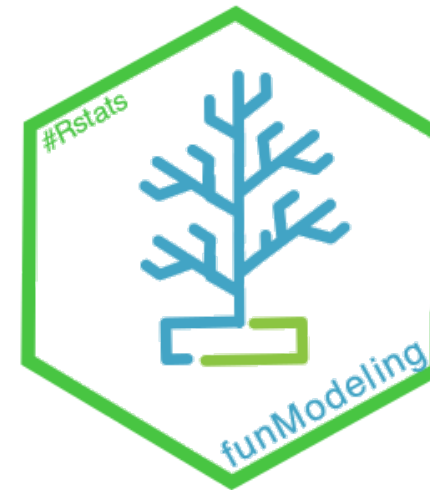
By now, you should be able to do the following:

- Examine the different column types and their summaries
- Recode factors and convert dates
- Manipulate characters (string normalization and approximate string matching)

# Supplementary Reading



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



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