

Week-4: Manipulating data

NM2207: Computational Media Literacy

Narayani Vedam, Ph.D.

Department of Communications and New Media



NUS

National University
of Singapore

Faculty of Arts
& Social Sciences



This week

Table of contents

I. Tidy data ([click here](#))

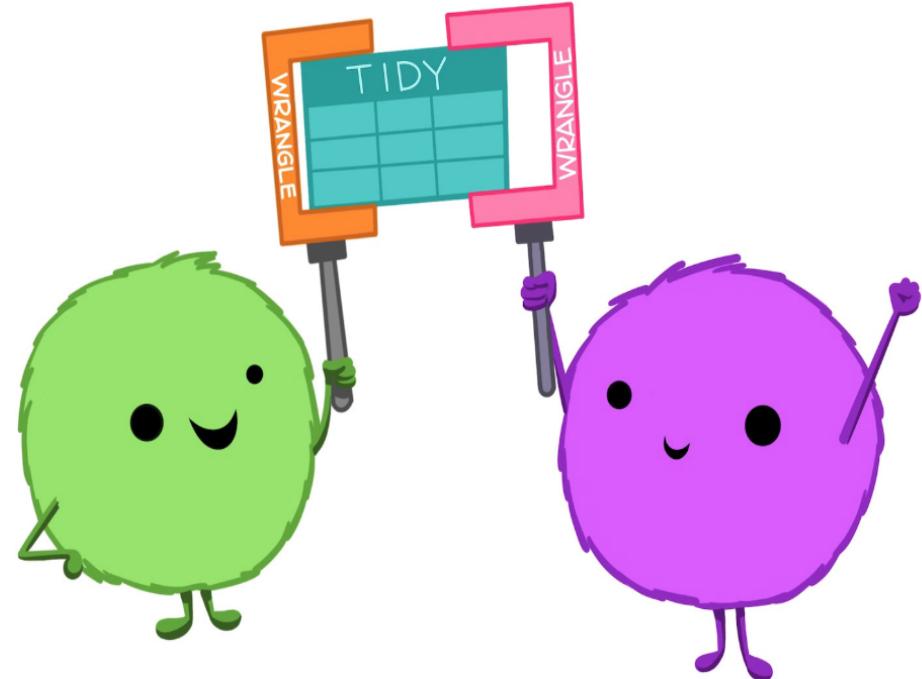
II. Basics of data handling ([click here](#))

III. Selecting rows/columns ([click here](#))

IV. Combining commands ([click here](#))

V. Creating columns ([click here](#))

VI. More operations ([click here](#))



Source: Allison Horst (<https://allisonhorst.com/other-r-fun>)

I. Tidy data

Tidy data: characteristics

- Every column is a variable
- Every row is an observation
- Each type of observational unit forms a table

“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”
—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable

id	name	color
1	floop	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

each row an observation

Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

Source: Allison Horst (<https://allisonhorst.com/other-r-fun>)

Tidy data: example

country	year	cases	population
	<int>	<int>	<int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	Observation	1999	37737
Brazil		2000	172006362
China		1999	80488
China		2000	1272915272
6 rows	Variable	Table	

Table 1: Illustration of Tidy Data

Tidy data: what makes data untidy?

Look closely! the table has the same **variables** as before, but in a different *format*.

country <chr>	year <int>	type <chr>	count <int>
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272

1-10 of 12 rows Previous **1** [2](#) [Next](#)

Table 2: Illustration of data with non-tidy structure

Tidy data: what makes data untidy?

Multiple variables (year, months) in columns!

Airplanes on Hand in the AAF, By Major Type: Jul 1939 to Aug 1945											
End of Month	Total	Very Heavy Bombers	Heavy Bombers	Medium Bombers	Light Bombers	Fighters	Reconnaissance	Transports	Trainers	Communications	
1939											
Jul	2,402	-	16	400	276	494	356	118	735	7	
Aug	2,440	-	18	414	276	492	359	129	745	7	
[Germany invades Poland, 1 Sep 1939]											
Sep	2,473	-	22	428	278	489	359	136	754	7	
Oct	2,507	-	27	446	277	490	365	137	758	7	
Nov	2,536	-	32	458	275	498	375	136	755	7	
Dec	2,546	-	39	464	274	492	378	131	761	7	
1940											
Jan	2,588	-	45	466	271	464	409	128	798	7	
Feb	2,658	-	49	470	271	458	415	128	860	7	
Mar	2,709	-	54	468	267	453	415	125	920	7	
Apr	2,806	-	54	468	263	451	416	125	1,022	7	
May	2,906	-	54	470	259	459	410	124	1,123	7	
Jun	2,966	-	54	478	166	477	414	127	1,243	7	
[France surrenders to Germany, 25 Jun 1940] [Battle of Britain begins, 10 July 1940]											
Jul	3,102	-	56	483	161	500	410	128	1,357	7	
Aug	3,295	-	65	485	158	539	407	128	1,506	7	

Table 3: Example of non-tidy dataframe (source: Army Air Forces Statistical Digest, WW II)

Tidy data: what makes data untidy?

Observations are in columns!

	A	AA	AB	AC	AD	AE	AF	AG	AH
1	Estimated HIV Prevalence% - (Ages 15-49)	2004	2005	2006	2007	2008	2009	2010	2011
2	Abkhazia						0.06	0.06	0.06
3	Afghanistan								
4	Akrotiri and Dhekelia								
5	Albania								
6	Algeria	0.1	0.1	0.1	0.1	0.1			
7	American Samoa								
8	Andorra								
9	Angola	1.9	1.9	1.9	1.9	2.1	2.1	2.1	
10	Anguilla								
11	Antigua and Barbuda								
12	Argentina	0.4	0.4	0.4	0.4	0.5	0.4	0.4	0.4
13	Armenia	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2
14	Aruba								
15	Australia	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2
16	Austria	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4
17	Azerbaijan	0.06	0.06	0.06	0.1	0.1	0.1	0.1	0.1
18	Bahamas	3	3	3	3.1	3.1	2.9	2.8	2.8

Table 4: Example of non-tidy dataframe (source: Gapminder, Estimated HIV prevalence among 15-49 year olds)

Tidy data: what makes data untidy?

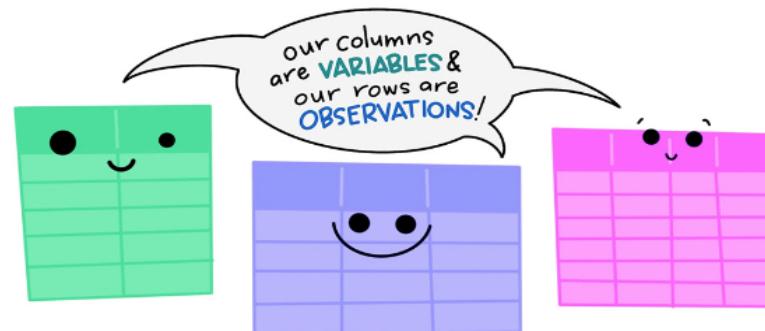
Multiple variables (year, events) in columns!

Subject	United States			
	Estimate	Margin of Error	Percent	Percent Margin of Error
EMPLOYMENT STATUS				
Population 16 years and over	255,797,692	+/-17,051	255,797,692	(X)
In labor force	162,184,325	+/-135,158	63.4%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	63.0%	+/-0.1
Employed	150,599,165	+/-138,066	58.9%	+/-0.1
Unemployed	10,560,305	+/-27,385	4.1%	+/-0.1
Armed Forces	1,024,855	+/-10,363	0.4%	+/-0.1
Not in labor force	93,613,367	+/-126,007	36.6%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	161,159,470	(X)
Unemployment Rate	(X)	(X)	6.6%	+/-0.1
Females 16 years and over				
Population 16 years and over	131,092,196	+/-11,187	131,092,196	(X)
In labor force	76,493,327	+/-75,824	58.4%	+/-0.1
Civilian labor force	76,350,498	+/-75,238	58.2%	+/-0.1
Employed	71,451,559	+/-79,007	54.5%	+/-0.1
Own children of the householder under 6 years	22,939,897	+/-14,240	22,939,897	(X)
All parents in family in labor force	14,957,537	+/-36,506	65.2%	+/-0.1
Own children of the householder 6 to 17 years	47,007,147	+/-19,644	47,007,147	(X)
All parents in family in labor force	33,238,793	+/-49,036	70.7%	+/-0.1

Figure: Example of non-tidy dataframe (Source: GUS Census Fact Finder, General Economic Characteristics, ACS 2017)

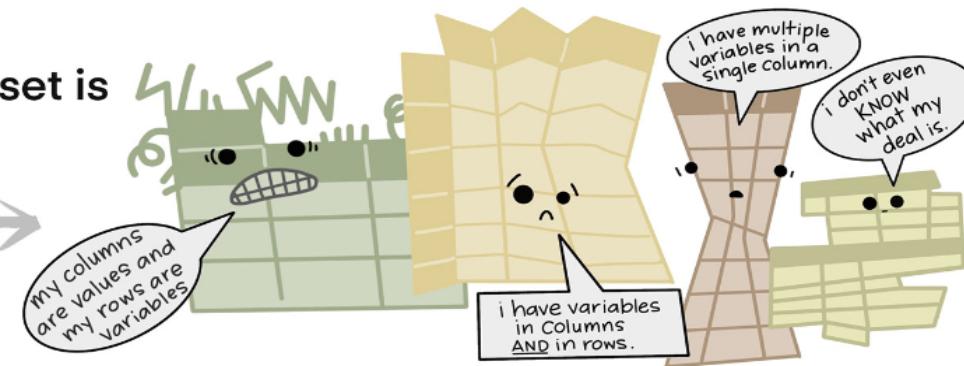
Tidy data: what makes data untidy?

The standard structure of tidy data means that
“tidy datasets are all alike...”



“...but every messy dataset is
messy in its own way.”

—HADLEY WICKHAM



Source: Allison Horst (<https://allisonhorst.com/other-r-fun>)

Ease of manipulation: Tidy vs Non-tidy data

Let us compute the mean number of cases;

- For data in **Tidy** format the code would be,

```
# Tidy data in table1
mean(table1$cases)
```

- For data in **Non-Tidy** format the code would be,

```
# Non-tidy data in table2
mean(table2$count[seq(from=1,to=nrows(table2))]
```

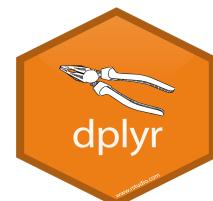
- *Which line of code is easier to write?*
- *Which line could you write if you've only looked at the first row of the data?*

II. Data wrangling

Basics of data wrangling

Once you load data in , you would want to explore it

- To do so, `dplyr` package is used. It is contained in `tidyverse`
- Some basic functions are,
 - `select()` to pick columns by name
 - `arrange()` to order data
 - `mutate()` to create new columns
 - `filter()` to pick rows matching the criteria
- The first argument (*what goes inside the brackets*) to the functions is always a data-set
- The subsequent arguments indicate operations on the data-set
- The output of the functions is also a data-set
- To store the output, it has to be assigned to a variable



Loading data from files

- **Download** the file with data
- **Save** the file in the same directory as your `.R` file or `.Rmd` file
- We will be using `read_csv()` function to **read** a `.csv` file
 - `.csv` files are text files that can be opened in a text editor
 - They are structured like dataframes or excel sheets with commas separating the values (hence `.csv`)

Data: hotel booking

- Data of two hotels - a city hotel and a resort
- Each row corresponds to a booking
- It was collected with the aim of developing prediction models to classify a hotel booking's likelihood to be cancelled
 - See: [Antonio et al. 2019](#)

```
# Load the package
library(tidyverse)
# Read data from the csv file
hotels <- read_csv("hotels.csv")
```

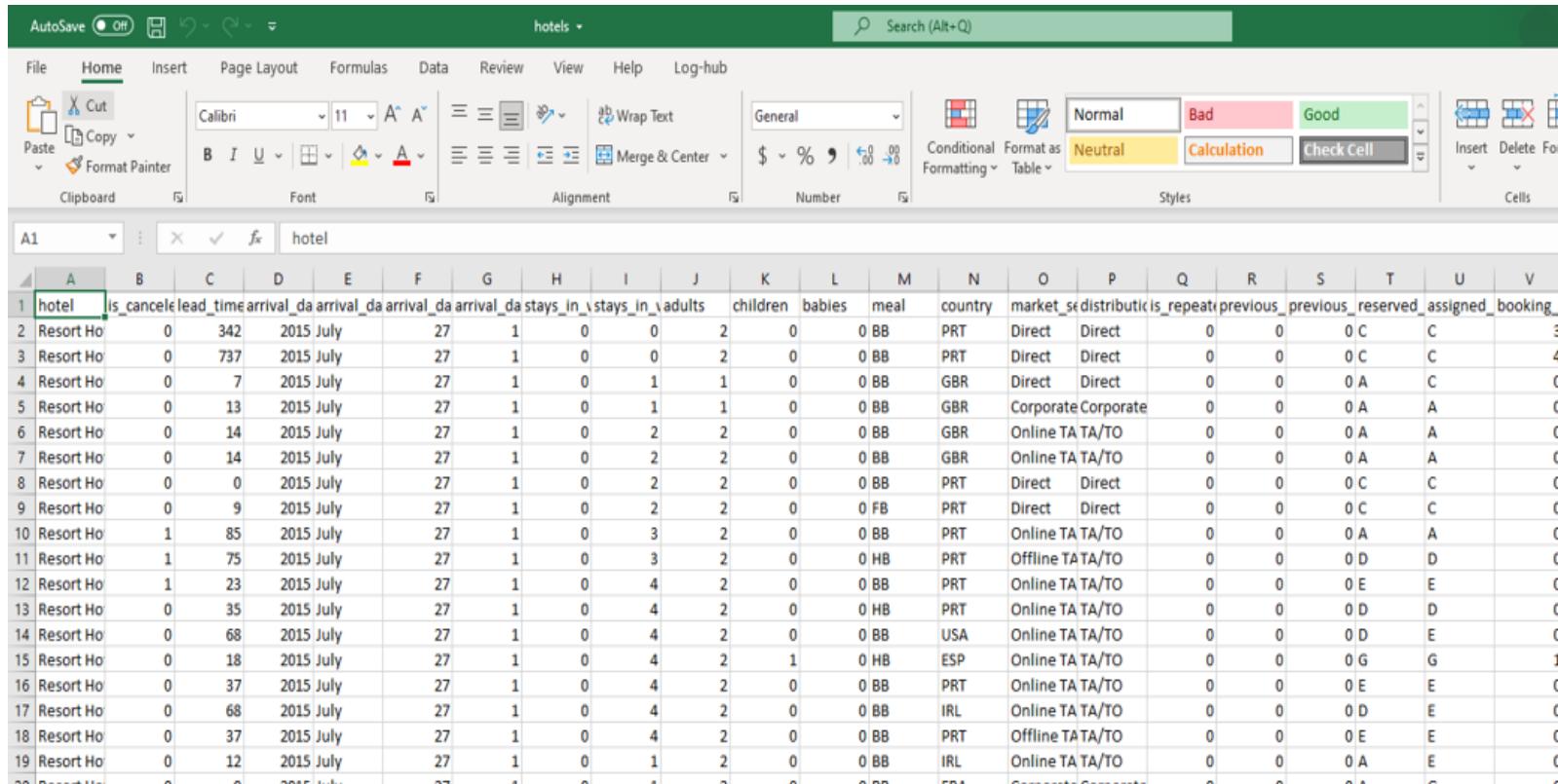
Data: hotel booking

The raw file looks like this (using Notepad on a Windows PC orTextEdit on Mac)

Figure: Hotel booking data

Data: hotel booking

You can open the file in excel too



hotel	is_cancelled	lead_time	arrival_date	arrival_date	arrival_date	stays_in_weekend_nights	stays_in_week_nights	children	babies	meal	country	market_segment	distribution渠	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	assigned_booking_commission	booking_status
Resort Hotel	0	342	2015-07-27	1	0	0	2	0	0 BB	PRT	Direct	Direct	0	0	0 C	C	3	
Resort Hotel	0	737	2015-07-27	1	0	0	2	0	0 BB	PRT	Direct	Direct	0	0	0 C	C	4	
Resort Hotel	0	7	2015-07-27	1	0	1	1	0	0 BB	GBR	Direct	Direct	0	0	0 A	C	0	
Resort Hotel	0	13	2015-07-27	1	0	1	1	0	0 BB	GBR	Corporate	Corporate	0	0	0 A	A	0	
Resort Hotel	0	14	2015-07-27	1	0	2	2	0	0 BB	GBR	Online TA	TA/TO	0	0	0 A	A	0	
Resort Hotel	0	14	2015-07-27	1	0	2	2	0	0 BB	GBR	Online TA	TA/TO	0	0	0 A	A	0	
Resort Hotel	0	0	2015-07-27	1	0	2	2	0	0 BB	PRT	Direct	Direct	0	0	0 C	C	0	
Resort Hotel	0	9	2015-07-27	1	0	2	2	0	0 FB	PRT	Direct	Direct	0	0	0 C	C	0	
Resort Hotel	1	85	2015-07-27	1	0	3	2	0	0 BB	PRT	Online TA	TA/TO	0	0	0 A	A	0	
Resort Hotel	1	75	2015-07-27	1	0	3	2	0	0 HB	PRT	Offline T	T/TO	0	0	0 D	D	0	
Resort Hotel	1	23	2015-07-27	1	0	4	2	0	0 BB	PRT	Online TA	TA/TO	0	0	0 E	E	0	
Resort Hotel	0	35	2015-07-27	1	0	4	2	0	0 HB	PRT	Online TA	TA/TO	0	0	0 D	D	0	
Resort Hotel	0	68	2015-07-27	1	0	4	2	0	0 BB	USA	Online TA	TA/TO	0	0	0 D	E	0	
Resort Hotel	0	18	2015-07-27	1	0	4	2	1	0 HB	ESP	Online TA	TA/TO	0	0	0 G	G	1	
Resort Hotel	0	37	2015-07-27	1	0	4	2	0	0 BB	PRT	Online TA	TA/TO	0	0	0 E	E	0	
Resort Hotel	0	68	2015-07-27	1	0	4	2	0	0 BB	IRL	Online TA	TA/TO	0	0	0 D	E	0	
Resort Hotel	0	37	2015-07-27	1	0	4	2	0	0 BB	PRT	Offline T	T/TO	0	0	0 E	E	0	
Resort Hotel	0	12	2015-07-27	1	0	1	2	0	0 BB	IRL	Online TA	TA/TO	0	0	0 A	E	0	

Figure: Hotel booking data

Hotel booking: look at the variables

- `names()` is a function
- `hotels` is the argument

```
names(hotels)
```

```
## [1] "hotel"
## [3] "lead_time"
## [5] "arrival_date_month"
## [7] "arrival_date_day_of_month"
## [9] "stays_in_week_nights"
## [11] "children"
## [13] "meal"
## [15] "market_segment"
## [17] "is_repeated_guest"
## [19] "previous_bookings_not_canceled"
## [21] "assigned_room_type"
## [23] "deposit_type"
## [25] "company"
## [27] "customer_type"
## [29] "required_car_parking_spaces"
## [31] "reservation_status"
## [33] "is_canceled"
## [35] "arrival_date_year"
## [37] "arrival_date_week_number"
## [39] "stays_in_weekend_nights"
## [41] "adults"
## [43] "babies"
## [45] "country"
## [47] "distribution_channel"
## [49] "previous_cancellations"
## [51] "reserved_room_type"
## [53] "booking_changes"
## [55] "agent"
## [57] "days_in_waiting_list"
## [59] "adr"
## [61] "total_of_special_requests"
## [63] "reservation_status_date"
```

Hotel booking: data overview

```
glimpse(hotels)
```

```
## Rows: 119,390
## Columns: 32
## $ hotel
## $ is_canceled
## $ lead_time
## $ arrival_date_year
## $ arrival_date_month
## $ arrival_date_week_number
## $ arrival_date_day_of_month
## $ stays_in_weekend_nights
## $ stays_in_week_nights
## $ adults
## $ children
## $ babies
## $ meal
## $ country
## $ market_segment
## $ distribution_channel
## $ is_repeated_guest
## $ previous_cancellations
## $ previous_bookings_not_canceled
```

```
<chr> "Resort Hotel", "Resort Hotel", "Resort...
<dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, ...
<dbl> 342, 737, 7, 13, 14, 14, 0, 9, 85, 75, ...
<dbl> 2015, 2015, 2015, 2015, 2015, 2015, 201...
<chr> "July", "July", "July", "July", "July", ...
<dbl> 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, ...
<dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
<dbl> 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 4, 4, 4, ...
<dbl> 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, ...
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
<chr> "BB", "BB", "BB", "BB", "BB", "BB", "BB...
<chr> "PRT", "PRT", "GBR", "GBR", "GBR", "GBR...
<chr> "Direct", "Direct", "Direct", "Corporat...
<chr> "Direct", "Direct", "Direct", "Corporat...
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
<dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
```

III. Choosing rows or columns

Selecting columns

| `select()` let's us select specific columns from a data-set

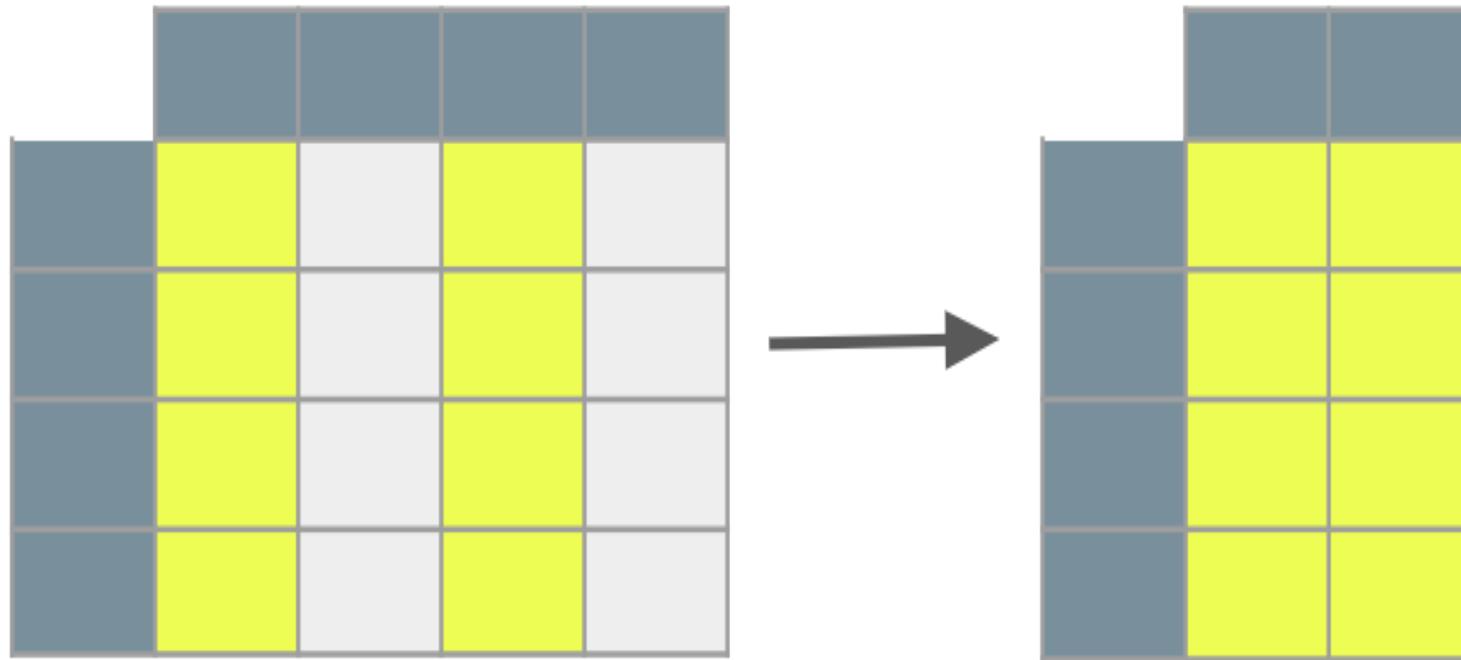


Figure: Selecting columns

Hotel booking: select a single column

- Start with the verb

```
select(           # <-- verb
         hotels,
         lead_time
     )
```

- First argument is the data-set

```
select(
         hotels,      # <-- data-set
         lead_time
     )
```

- Second argument is the column we want to select

```
select(
         hotels,
         lead_time    # <-- chosen column/variable
     )
```

Hotel booking: select a single column

- View only `lead_time` (number of days between booking and arrival)

```
select(hotels, lead_time)
```

```
## # A tibble: 119,390 × 1
##   lead_time
##       <dbl>
## 1     342
## 2     737
## 3      7
## 4     13
## 5     14
## 6     14
## 7      0
## 8      9
## 9     85
## 10    75
## # i 119,380 more rows
```

- **Remember:** `dplyr` functions always expect a dataframe and the output is also a dataframe

Hotel booking: selecting multiple columns

- View `lead_time`, `agent` and `market_segment`

```
select(hotels, lead_time, agent, market_segment)
```

```
## # A tibble: 119,390 × 3
##   lead_time agent market_segment
##       <dbl> <chr>  <chr>
## 1      342  NULL   Direct
## 2      737  NULL   Direct
## 3        7  NULL   Direct
## 4      13  304   Corporate
## 5      14  240   Online TA
## 6      14  240   Online TA
## 7        0  NULL   Direct
## 8        9  303   Direct
## 9      85  240   Online TA
## 10     75  15    Offline TA/TO
## # i 119,380 more rows
```

- **Note:** you can use `-c("column_names")` to remove specific columns

Arranging columns

`arrange()` let's us reorder data

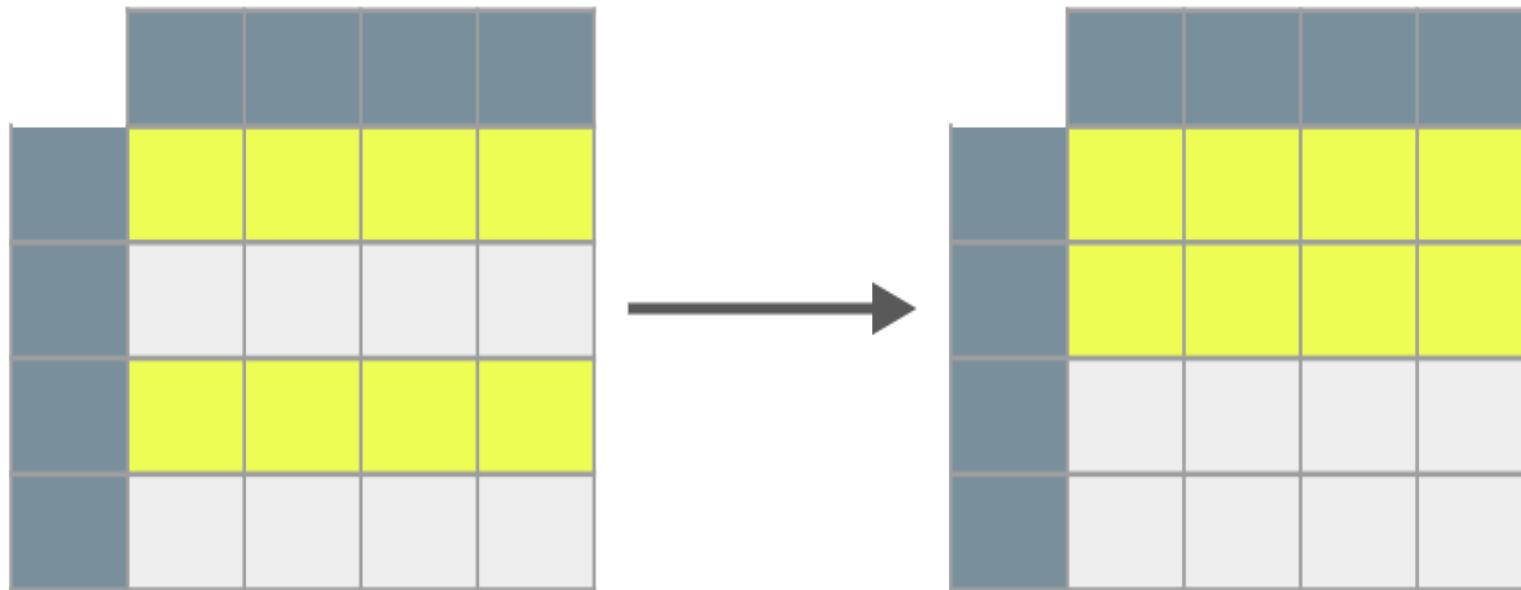


Figure: Arranging columns

Hotel booking: arrange

Let us try reordering entries of the column `lead_time`

- Start with the verb

```
arrange(          # <-- verb
          hotels,
          lead_time
        )
```

- First argument is the data-set

```
arrange(
          hotels,      # <-- data-set
          lead_time
        )
```

- Second argument is the column whose entries we want to reorder

```
arrange(
          hotels,
          lead_time    # <-- chosen column/variable
        )
```

Hotel booking: arrange

- View the rearranged entries of `lead_time`

```
arrange(hotels, lead_time)
```

```
## # A tibble: 119,390 × 32
##   hotel      is_canceled lead_time arrival_date_year arrival_date_month
##   <chr>          <dbl>     <dbl>            <dbl>        <chr>
## 1 Resort Hotel       0         0            2015 July
## 2 Resort Hotel       0         0            2015 July
## 3 Resort Hotel       0         0            2015 July
## 4 Resort Hotel       0         0            2015 July
## 5 Resort Hotel       0         0            2015 July
## 6 Resort Hotel       0         0            2015 July
## 7 Resort Hotel       0         0            2015 July
## 8 Resort Hotel       0         0            2015 July
## 9 Resort Hotel       0         0            2015 July
## 10 Resort Hotel      0         0            2015 July
## # i 119,380 more rows
## # i 27 more variables: arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,
## #   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,
## #   meal <chr>, country <chr>, market_segment <chr>,
## #   ...
```

Hotel booking: arrange

- Original order of the entries of `lead_time`

hotels

```
## # A tibble: 119,390 × 32
##   hotel      is_canceled lead_time arrival_date_year arrival_date_month
##   <chr>          <dbl>     <dbl>             <dbl>        <chr>
## 1 Resort Hotel       0       342             2015 July
## 2 Resort Hotel       0       737             2015 July
## 3 Resort Hotel       0        7             2015 July
## 4 Resort Hotel       0       13             2015 July
## 5 Resort Hotel       0       14             2015 July
## 6 Resort Hotel       0       14             2015 July
## 7 Resort Hotel       0        0             2015 July
## 8 Resort Hotel       0        9             2015 July
## 9 Resort Hotel       1       85             2015 July
## 10 Resort Hotel      1       75             2015 July
## # i 119,380 more rows
## # i 27 more variables: arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,
## #   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,
## #   meal <chr>, country <chr>, market_segment <chr>,
## #   ...
```

Hotel booking: arrange

- By default, `arrange()` arranges the entries of the chosen column in the **increasing** order
- Re-arranging applies to the entire row and not just the entries of the column
- In other words, the observation (rows) get rearranged so that the entries of `lead_time` are in the ascending order
- To rearrange entries in the descending order, use `desc()`

```
arrange(hotels, desc(lead_time))
```

Hotel booking: select + arrange

- After picking the column/variable `lead_time`, let us say we want to rearrange its entries in the decreasing order
- Step-1 is to pick the desired column, `select(hotels, lead_time)`

```
select(hotels, lead_time)
```

- Step-2 is to rearrange the output of `select(hotels, lead_time)`

```
arrange(      # <-- start with the verb
  select(hotels, lead_time), # <-- first argument is the dataframe *
  desc(lead_time) # <--- second argument is the how you want arrange
)                  # i.e. decreasing order of lead_time
```

* in this case, it is the output of the `select` operation.

Hotel booking: select + arrange

```
select(hotels, lead_time)
```

```
## # A tibble: 119,390 × 1
##   lead_time
##       <dbl>
## 1     342
## 2     737
## 3      7
## 4     13
## 5     14
## 6     14
## 7      0
## 8      9
## 9     85
## 10    75
## # i 119,380 more rows
```

```
arrange(
  select(hotels, lead_time),
  desc(lead_time)
)
```

```
## # A tibble: 119,390 × 1
##   lead_time
##       <dbl>
## 1     737
## 2     709
## 3     629
## 4     629
## 5     629
## 6     629
## 7     629
## 8     629
## 9     629
## 10    629
## # i 119,380 more rows
```

Hotel booking: select + arrange

- Did you notice how many functions we used to arrive at a rearranged column?
- A function(s) within another function(s) is called a *nested function*
- Now, imagine having to do more than ten operations on a data-frame
- How confusing could it get to have functions inside another??!
- You think not? Read on.

```
arrange(                      # <-- Function 1
         select(hotels, lead_time), # <-- Function 2
         desc(lead_time) # <-- Function 3
      )
```

IV. Combining two or more operations

Nesting operations

- Think of the activities that you might be doing every morning
 - find keys, unlock car, start car, drive to work, park
- As  functions, this would hypothetically look like:

```
park(drive(start_car(find("keys"))), to = "work"))
```

- This is very confusing to read, given that the first thing you write, is the last thing you do,
 - `park` then `drive` then `start_car` then `find(keys)`, and then the `to` argument again refers to `drive`, not what is right next to it (i.e. `find`)
- Is there an easier way to tackle this?? **Pipes**

Pipes

- We have already used this operator earlier in our course
- They are a part of `magrittr` package that is automatically loaded when you load `tidyverse`
- `%>%` operator takes the left-hand side and passes it as an input to the right-hand side
- It makes the code easier to read and write
- Let us apply it to the hypothetical scenario from earlier

```
"keys" %>%
  find() %>%
  start_car() %>%
  drive(to = "work") %>%
  park()
```

- *Isn't it easier to read and follow?*

Hotel booking: select + arrange using pipes

```
arrange(
  select(hotels, lead_time),
  desc(lead_time)
)
```

```
## # A tibble: 119,390 × 1
##   lead_time
##       <dbl>
## 1     737
## 2     709
## 3     629
## 4     629
## 5     629
## 6     629
## 7     629
## 8     629
## 9     629
## 10    629
## # i 119,380 more rows
```

```
hotels %>%
  select(lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 × 1
##   lead_time
##       <dbl>
## 1     737
## 2     709
## 3     629
## 4     629
## 5     629
## 6     629
## 7     629
## 8     629
## 9     629
## 10    629
## # i 119,380 more rows
```

Piping versus Layering

- Let us recall visualization using `ggplot2`
 - we created the plot in layers, separated by `+`
- Layering works differently from pipelines of `dplyr`
 - As noted in earlier slides, the output of the previous line of code serves as the input to the next

Piping in dplyr

This is incorrect

```
hotels +  
  select(hotel,lead_time)
```

```
## Error in select(hotel, lead_time): object '1'
```

This is correct

```
hotels %>%  
  select(hotel,lead_time)
```

```
## # A tibble: 119,390 × 2  
##       hotel      lead_time  
##       <chr>        <dbl>  
## 1 Resort Hotel     342  
## 2 Resort Hotel     737  
## 3 Resort Hotel      7  
## 4 Resort Hotel     13  
## 5 Resort Hotel     14  
## 6 Resort Hotel     14  
## 7 Resort Hotel      0  
## 8 Resort Hotel      9  
## 9 Resort Hotel     85  
## 10 Resort Hotel    75  
## # i 119,380 more rows
```

Layering in ggplot2

- This is incorrect

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) %>%  
  geom_bar()
```

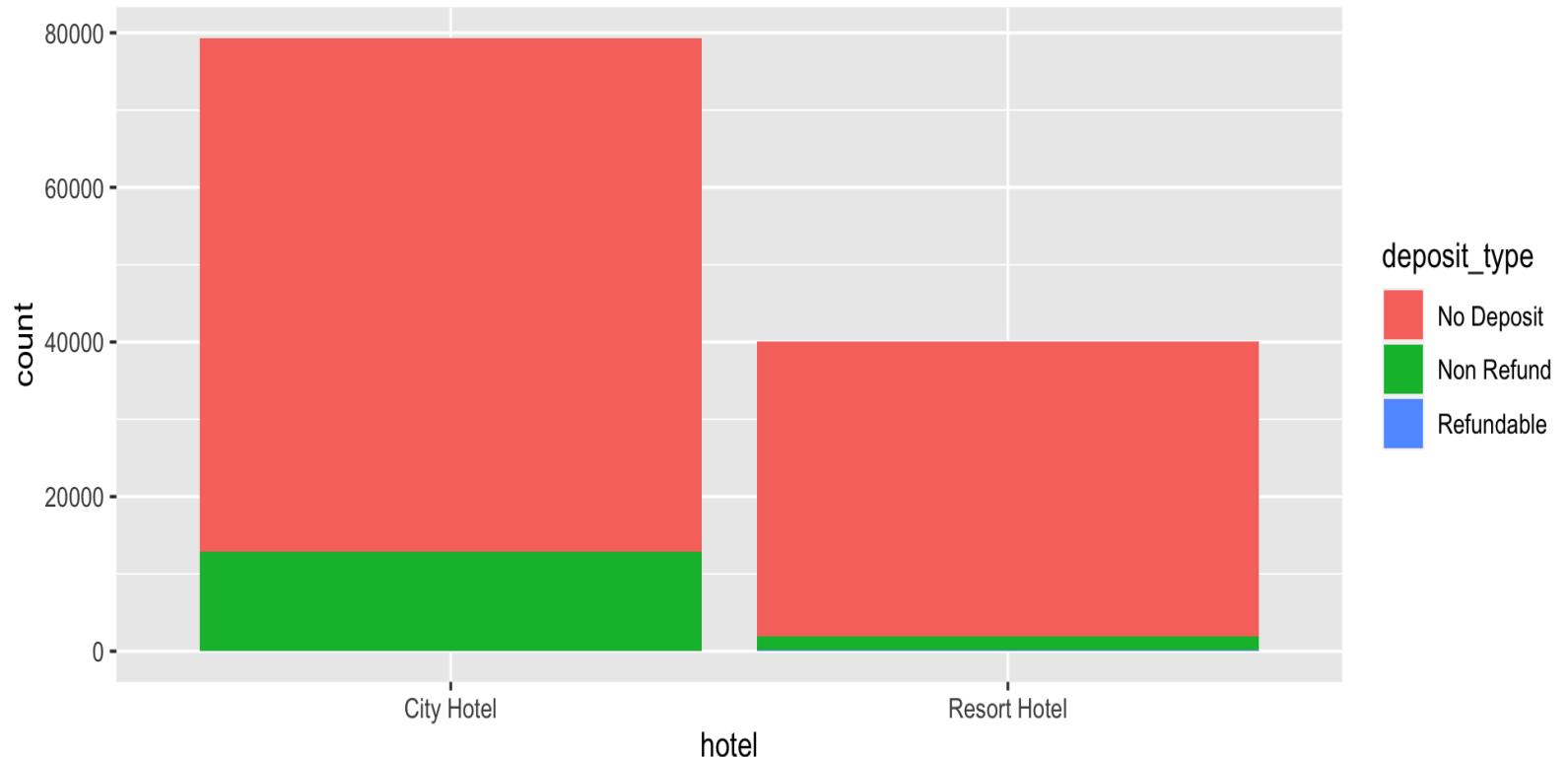
```
## Error in `geom_bar()`:  
## ! `mapping` must be created by `aes()`  
## i Did you use `%>%` or `|>` instead of `+`?
```

- This is correct

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) +  
  geom_bar()
```

Layering in ggplot2

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) +  
  geom_bar()
```



Pick rows that match a condition, `filter()`

- Use `filter()` to pick rows matching a condition

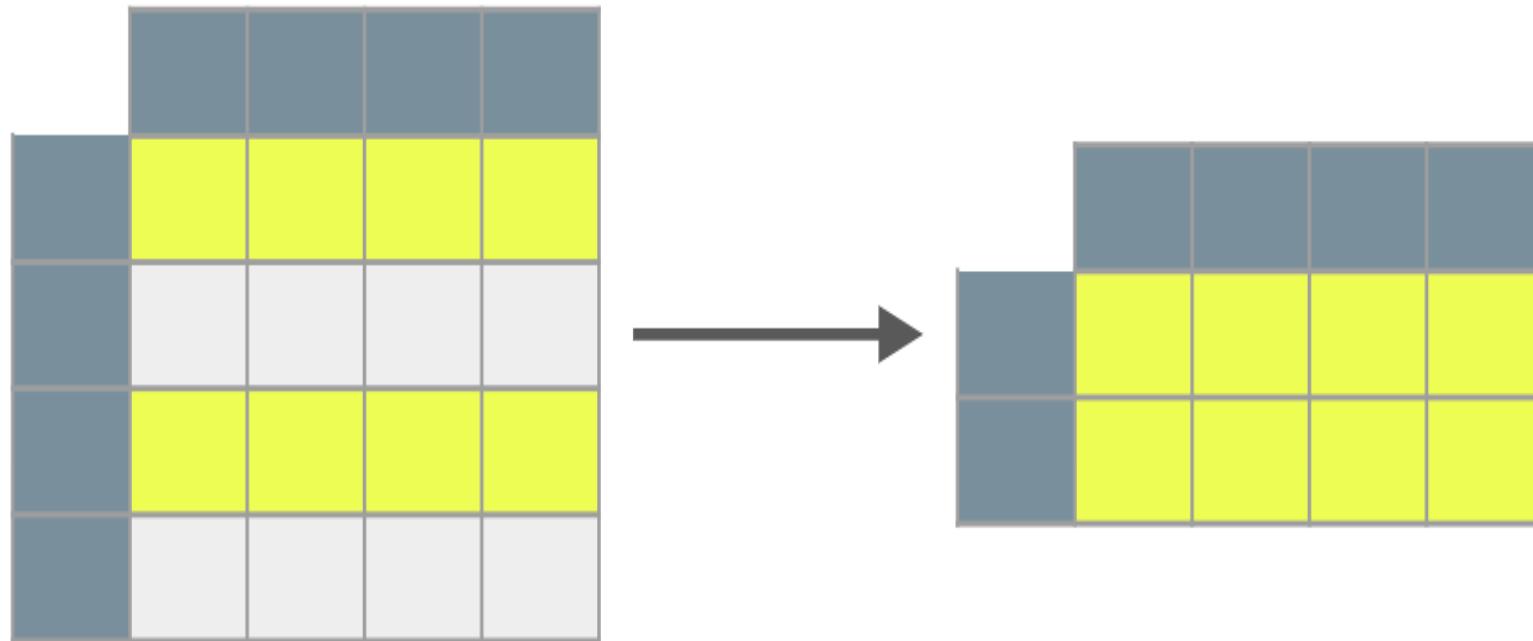


Figure: Selecting rows

Hotel Booking: Pick rows matching a condition using `filter()`

- Start with the verb

```
hotels %>%
  filter(                               # <-- verb 1
    children >= 1
  ) %>%
    select(hotel, children) # <-- verb 2
```

- Pass the condition, the column/variable on the left of the conditional operator

```
hotels %>%
  filter(
    children >= 1                      # <-- condition
  ) %>%
    select(hotel, children)
```

Hotel Booking: Pick rows matching a condition using `filter()`

```
hotels %>%
  filter(children >= 1) %>%
  select(hotel, children)
```

```
## # A tibble: 8,590 × 2
##   hotel      children
##   <chr>        <dbl>
## 1 Resort Hotel     1
## 2 Resort Hotel     2
## 3 Resort Hotel     2
## 4 Resort Hotel     2
## 5 Resort Hotel     1
## 6 Resort Hotel     1
## 7 Resort Hotel     2
## 8 Resort Hotel     2
## 9 Resort Hotel     1
## 10 Resort Hotel    2
## # i 8,580 more rows
```

Hotel Booking: Pick rows matching more than one condition using `filter()`

- Start with the verb

```
hotels %>%
  filter(
    children >= 1
  ) %>%
  select(hotel, children) # <-- verb 2
```

- Pass the conditions, separated by `,`

```
hotels %>%
  filter(
    children >= 1,           # <-- condition 1
    hotel == "City Hotel"   # <-- condition 2
  ) %>%
  select(hotel, children)
```

Hotel Booking: Pick rows matching a condition using `filter()`

```
hotels %>%
  filter(children >= 1, hotel == "City Hotel") %>%
  select(hotel, children)
```

```
## # A tibble: 5,106 × 2
##   hotel      children
##   <chr>        <dbl>
## 1 City Hotel     1
## 2 City Hotel     2
## 3 City Hotel     1
## 4 City Hotel     1
## 5 City Hotel     1
## 6 City Hotel     1
## 7 City Hotel     1
## 8 City Hotel     1
## 9 City Hotel     1
## 10 City Hotel    1
## # i 5,096 more rows
```

Pick rows using `slice()`: No conditional selection

- || Choose certain rows/observations without conditions using `slice()`
- Quick recap
 - Choose certain variables/columns using `select()`
 - Conditionally pick rows or observations using `filter()`

Pick rows using `slice()`: No conditional selection

- Start with the verb

```
hotels %>%
  slice(      #<-- verb
    1:5
  )
```

- Pass the indices or the range of indices of the rows to be picked

```
hotels %>%
  slice(
    1:5    #<-- indices
  )
```

Pick rows using `slice()`: No conditional selection

- Using a sequence of indices

```
hotels %>% slice(1:5)
```

```
## # A tibble: 5 × 32
##   hotel      is_canceled lead_time arrival_date_year arrival_date_month
##   <chr>          <dbl>     <dbl>             <dbl> <chr>
## 1 Resort Hotel       0        342            2015 July
## 2 Resort Hotel       0        737            2015 July
## 3 Resort Hotel       0         7            2015 July
## 4 Resort Hotel       0        13            2015 July
## 5 Resort Hotel       0        14            2015 July
## # i 27 more variables: arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,
## #   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,
## #   meal <chr>, country <chr>, market_segment <chr>,
## #   distribution_channel <chr>, is_repeated_guest <dbl>,
## #   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,
## #   reserved_room_type <chr>, assigned_room_type <chr>, ...
```

Pick rows using `slice()`: No conditional selection

- Using specific indices

```
hotels %>%  
  slice(1,3,5)
```

```
## # A tibble: 3 × 32  
##   hotel      is_canceled lead_time arrival_date_year arrival_date_month  
##   <chr>          <dbl>     <dbl>           <dbl> <chr>  
## 1 Resort Hotel        0       342            2015 July  
## 2 Resort Hotel        0        7            2015 July  
## 3 Resort Hotel        0       14            2015 July  
## # i 27 more variables: arrival_date_week_number <dbl>,  
## #   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,  
## #   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,  
## #   meal <chr>, country <chr>, market_segment <chr>,  
## #   distribution_channel <chr>, is_repeated_guest <dbl>,  
## #   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,  
## #   reserved_room_type <chr>, assigned_room_type <chr>, ...
```

Pick unique rows using `distinct()`

- Start with the verb

```
hotels %>%
  distinct(          #<-- verb,
            hotel)
```

- Pass the dataframe

```
hotels %>%           #<-- dataset
  distinct(
            hotel)
```

- Pass the column or variable whose unique entries are to be picked

```
hotels %>%
  distinct(
            hotel) #<-- column/variable
```

Pick unique rows using `distinct()`: output

- The output is a dataframe with only those rows with the unique entries of `hotel`

```
hotels %>% distinct(hotel)
```

```
## # A tibble: 2 × 1
##   hotel
##   <chr>
## 1 Resort Hotel
## 2 City Hotel
```

V. Creating new columns

Creating columns with `mutate()`

Use `mutate()` to create new columns using existing columns

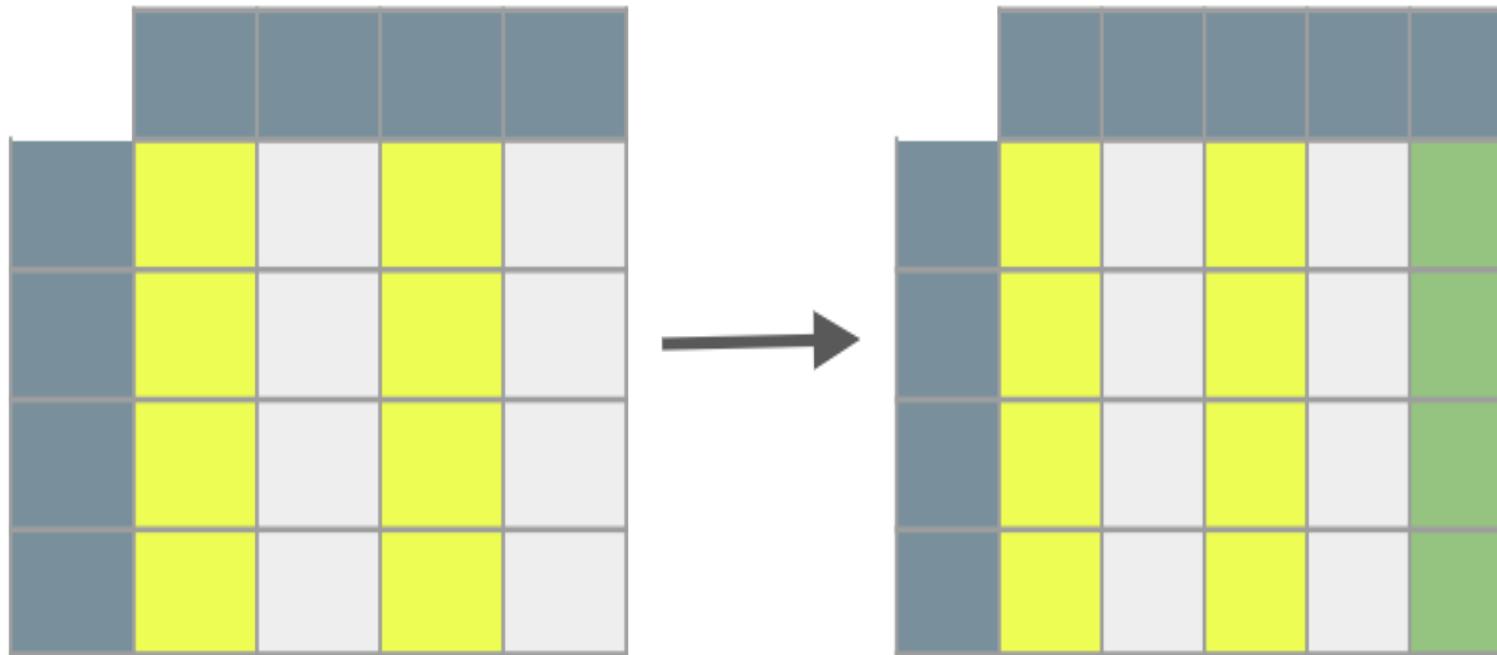


Figure: Create new columns

Creating a single column with `mutate()`

- To create a new column, on the left side of `=` is the name of the new column
- On the right of `=` are the operations on existing columns

```
hotels %>%  
  mutate(little_ones = children + babies) %>% #<-- Look here!  
  select(hotel, little_ones, children, babies)
```

Creating a single column with `mutate()`

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  select(hotel, little_ones, children, babies)
```

```
## # A tibble: 119,390 × 4
##   hotel      little_ones children babies
##   <chr>        <dbl>     <dbl>    <dbl>
## 1 Resort Hotel       0         0        0
## 2 Resort Hotel       0         0        0
## 3 Resort Hotel       0         0        0
## 4 Resort Hotel       0         0        0
## 5 Resort Hotel       0         0        0
## 6 Resort Hotel       0         0        0
## 7 Resort Hotel       0         0        0
## 8 Resort Hotel       0         0        0
## 9 Resort Hotel       0         0        0
## 10 Resort Hotel      0         0        0
## # i 119,380 more rows
```

Creating multiple columns with `mutate()`

- To create a new column, on the left side of `=` is the name of the new column
- On the right of `=` are the operations on existing columns
- The columns are separated by `,`
- The newer columns can even use a newly created column(s)

```
hotels %>%  
  mutate(little_ones = children + babies,           #<-- New column 1  
        average_little_ones = mean(little_ones)) %>% #<-- New column 2  
  select(hotel, little_ones,children,babies)
```

Creating multiple columns with `mutate()`

```
hotels %>%
  mutate(little_ones = children + babies,
        average_little_ones = mean(little_ones)) %>%
  select(hotel, little_ones, children, babies, average_little_ones)
```

```
## # A tibble: 119,390 × 5
##   hotel      little_ones children babies average_little_ones
##   <chr>       <dbl>     <dbl>   <dbl>             <dbl>
## 1 Resort Hotel       0        0      0               NA
## 2 Resort Hotel       0        0      0               NA
## 3 Resort Hotel       0        0      0               NA
## 4 Resort Hotel       0        0      0               NA
## 5 Resort Hotel       0        0      0               NA
## 6 Resort Hotel       0        0      0               NA
## 7 Resort Hotel       0        0      0               NA
## 8 Resort Hotel       0        0      0               NA
## 9 Resort Hotel       0        0      0               NA
## 10 Resort Hotel      0        0      0               NA
## # i 119,380 more rows
```

VI. More operations with examples

count() to get frequencies (counts)

```
hotels %>%  
  count(market_segment)
```

```
## # A tibble: 8 × 2  
##   market_segment     n  
##   <chr>             <int>  
## 1 Aviation           237  
## 2 Complementary      743  
## 3 Corporate          5295  
## 4 Direct              12606  
## 5 Groups              19811  
## 6 Offline TA/TO       24219  
## 7 Online TA            56477  
## 8 Undefined            2
```

- **Note** that count automatically arranges in alphabetical order

count() to get frequencies (counts)

```
hotels %>%  
  count(market_segment, sort = TRUE) # <-- decreasing order of counts
```

```
## # A tibble: 8 × 2  
##   market_segment     n  
##   <chr>             <int>  
## 1 Online TA          56477  
## 2 Offline TA/TO      24219  
## 3 Groups             19811  
## 4 Direct              12606  
## 5 Corporate           5295  
## 6 Complementary       743  
## 7 Aviation             237  
## 8 Undefined            2
```

count() multiple variables

```
hotels %>%  
  count(hotel, market_segment)
```

```
## # A tibble: 14 × 3  
##   hotel      market_segment     n  
##   <chr>      <chr>          <int>  
## 1 City Hotel Aviation        237  
## 2 City Hotel Complementary  542  
## 3 City Hotel Corporate     2986  
## 4 City Hotel Direct        6093  
## 5 City Hotel Groups        13975  
## 6 City Hotel Offline TA/TO 16747  
## 7 City Hotel Online TA    38748  
## 8 City Hotel Undefined     2  
## 9 Resort Hotel Complementary 201  
## 10 Resort Hotel Corporate   2309  
## 11 Resort Hotel Direct     6513  
## 12 Resort Hotel Groups     5836  
## 13 Resort Hotel Offline TA/TO 7472  
## 14 Resort Hotel Online TA  17729
```

summarise() for summary statistics

summarise() collapses the dataframe down to a single summary statistic

```
# mean average daily rate for all bookings
hotels %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 1 × 1
##   mean_adr
##       <dbl>
## 1     102.
```



summarise() by variable using group_by()

```
# mean average daily rate for all booking at city and resort hotels
hotels %>%
  group_by(hotel) %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 2 × 2
##   hotel      mean_adr
##   <chr>       <dbl>
## 1 City Hotel    105.
## 2 Resort Hotel   95.0
```



summarise() by variable using group_by()

```
# Count of bookings at city and resort hotels
hotels %>%
  group_by(hotel) %>%
  summarise(count = n())
```

```
## # A tibble: 2 × 2
##   hotel      count
##   <chr>     <int>
## 1 City Hotel  79330
## 2 Resort Hotel 40060
```

Calculating frequencies

- These two give the same results

```
hotels %>%
  group_by(hotel) %>%
  summarise(count = n())
```

```
## # A tibble: 2 × 2
##   hotel      count
##   <chr>     <int>
## 1 City Hotel  79330
## 2 Resort Hotel 40060
```

```
hotels %>%
  count(hotel)
```

```
## # A tibble: 2 × 2
##   hotel      n
##   <chr>     <int>
## 1 City Hotel  79330
## 2 Resort Hotel 40060
```

summarise() for multiple summary statistics

```
hotels %>%  
  summarise(  
    min_adr = min(adr),  
    mean_adr = mean(adr),  
    median_adr = median(adr),  
    max_adr = max(adr)  
)
```

```
## # A tibble: 1 × 4  
##   min_adr mean_adr median_adr max_adr  
##     <dbl>     <dbl>      <dbl>     <dbl>  
## 1     -6.38     102.       94.6     5400
```

select(), slice(), and arrange()

```
hotels %>%
  select(hotel, lead_time) %>%
  slice(1:5) %>%
  arrange(lead_time)
```

```
## # A tibble: 5 × 2
##   hotel      lead_time
##   <chr>        <dbl>
## 1 Resort Hotel      7
## 2 Resort Hotel     13
## 3 Resort Hotel     14
## 4 Resort Hotel    342
## 5 Resort Hotel   737
```

select(), arrange(), and slice()

- How is that different from?

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(lead_time) %>%
  slice(1:5)
```

select(), arrange(), and slice()

- How is that different from?

```
hotels %>%  
  select(hotel, lead_time) %>%  
  arrange(lead_time) %>%  
  slice(1:5)
```

```
## # A tibble: 5 × 2  
##   hotel      lead_time  
##   <chr>        <dbl>  
## 1 Resort Hotel      0  
## 2 Resort Hotel      0  
## 3 Resort Hotel      0  
## 4 Resort Hotel      0  
## 5 Resort Hotel      0
```

- the **order of the piping** matters!

as ignored code or comments

```
hotels %>%  
  # slice the first five rows # this line is a comment  
  #select(hotel) %>%        # this one doesn't run  
  slice(1:5)                 # this line runs
```

```
## # A tibble: 5 × 32  
##   hotel      is_canceled lead_time arrival_date_year arrival_date_month  
##   <chr>          <dbl>     <dbl>           <dbl> <chr>  
## 1 Resort Hotel       0        342            2015 July  
## 2 Resort Hotel       0        737            2015 July  
## 3 Resort Hotel       0         7             2015 July  
## 4 Resort Hotel       0        13            2015 July  
## 5 Resort Hotel       0        14            2015 July  
## # i 27 more variables: arrival_date_week_number <dbl>,  
## #   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,  
## #   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,  
## #   meal <chr>, country <chr>, market_segment <chr>,  
## #   distribution_channel <chr>, is_repeated_guest <dbl>,  
## #   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,  
## #   reserved_room_type <chr>, assigned_room_type <chr>, ...
```

filter() to select rows based on conditions

- get all rows where `hotel` is "City Hotel" (note the use of `==` as opposed to `=`)

```
# bookings in City Hotels
hotels %>%
  filter(hotel == "City Hotel")
```

```
## # A tibble: 79,330 × 32
##   hotel      is_canceled lead_time arrival_date_year arrival_date_month
##   <chr>          <dbl>       <dbl>           <dbl>        <chr>
## 1 City Hotel      0            6            2015 July
## 2 City Hotel      1           88            2015 July
## 3 City Hotel      1           65            2015 July
## 4 City Hotel      1           92            2015 July
## 5 City Hotel      1          100            2015 July
## 6 City Hotel      1           79            2015 July
## 7 City Hotel      0            3            2015 July
## 8 City Hotel      1           63            2015 July
## 9 City Hotel      1           62            2015 July
## 10 City Hotel     1           62            2015 July
## # i 79,320 more rows
## # i 27 more variables: arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,
## #   ...
```

filter() to select rows based on conditions

- get all rows where adults is 0 and children is greater than or equal to 1

```
hotels %>%
  filter(
    adults == 0,
    children >= 1
  ) %>%
  select(adults, babies, children)
```

```
## # A tibble: 223 × 3
##   adults babies children
##   <dbl>   <dbl>     <dbl>
## 1 0       0       3
## 2 0       0       2
## 3 0       0       2
## 4 0       0       2
## 5 0       0       2
## 6 0       0       3
## 7 0       1       2
## 8 0       0       2
## 9 0       0       2
## 10 0      0       2
## # ... with 213 more rows
```

filter() to select rows based on (complex) conditions

- get all rows where adults is 1 and children is greater than or equal to 1 or babies is greater than or equal to 1

```
hotels %>%
  filter( adults == 1,
         children >= 1 | babies >=1) %>%    # | means OR
  select(adults, babies, children)
```

```
## # A tibble: 450 × 3
##   adults babies children
##     <dbl>   <dbl>     <dbl>
## 1     1       1        2
## 2     1       1        2
## 3     1       1        1
## 4     1       1        0
## 5     1       0        1
## 6     1       0        1
## 7     1       0        2
## 8     1       0        2
## 9     1       0        1
## 10    1      NA        1
```

Logical operators in R

Operator	Definition
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
x & y	x AND y
x y	x OR y
is.na(x)	test if x is NA
!is.na(x)	test if x is not NA
x %in% y	test if x is in y
!(x %in% y)	test if x is not in y
!x	not x

count() and arrange() instead

```
hotels %>%  
  count(market_segment) %>%  
  arrange(desc(n)) # <-- decreasing order of counts
```

```
## # A tibble: 8 × 2  
##   market_segment     n  
##   <chr>              <int>  
## 1 Online TA          56477  
## 2 Offline TA/TO      24219  
## 3 Groups             19811  
## 4 Direct              12606  
## 5 Corporate           5295  
## 6 Complementary        743  
## 7 Aviation             237  
## 8 Undefined            2
```

mutate() to add new variables

```
hotels %>%  
  mutate(little_ones = children + babies) %>% # <----  
  select(children, babies, little_ones) %>%  
  arrange(desc(little_ones))
```

```
## # A tibble: 119,390 × 3  
##   children babies little_ones  
##       <dbl>   <dbl>      <dbl>  
## 1       10     0        10  
## 2       0     10        10  
## 3       0      9         9  
## 4       2      1         3  
## 5       2      1         3  
## 6       2      1         3  
## 7       3      0         3  
## 8       2      1         3  
## 9       2      1         3  
## 10      3      0         3  
## # i 119,380 more rows
```

mutate() and filter() using new columns

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  filter(
    little_ones >= 1,
    hotel == "Resort Hotel"
  ) %>%
  select(hotel, little_ones)
```

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  filter(
    little_ones >= 1,
    hotel == "City Hotel"
  ) %>%
  select(hotel, little_ones)
```

mutate() and filter() using new columns

```
hotels %>%  
  mutate(little_ones = children + babies) %>%  
  filter(  
    little_ones >= 1,  
    hotel == "Resort Hotel"  
  ) %>%  
  select(hotel, little_ones)
```

```
hotels %>%  
  mutate(little_ones = children + babies) %>%  
  filter(  
    little_ones >= 1,  
    hotel == "City Hotel"  
  ) %>%  
  select(hotel, little_ones)
```

```
## # A tibble: 3,929 × 2  
##   hotel      little_ones  
##   <chr>        <dbl>  
## 1 Resort Hotel     1  
## 2 Resort Hotel     2  
## 3 Resort Hotel     2  
## 4 Resort Hotel     2  
## 5 Resort Hotel     1  
## 6 Resort Hotel     1  
## 7 Resort Hotel     2  
## 8 Resort Hotel     2  
## 9 Resort Hotel     1  
## 10 Resort Hotel    1  
## # ... 3,919 more rows
```

```
## # A tibble: 5,403 × 2  
##   hotel      little_ones  
##   <chr>        <dbl>  
## 1 City Hotel     1  
## 2 City Hotel     1  
## 3 City Hotel     2  
## 4 City Hotel     1  
## 5 City Hotel     1  
## 6 City Hotel     1  
## 7 City Hotel     1  
## 8 City Hotel     1  
## 9 City Hotel     1  
## 10 City Hotel    1  
## # ... 5,393 more rows
```

Thanks!

Slides created via the R packages:

xaringan
gadenbuie/xaringanthemer.



Faculty of Arts
& Social Sciences