Exploring Data with R

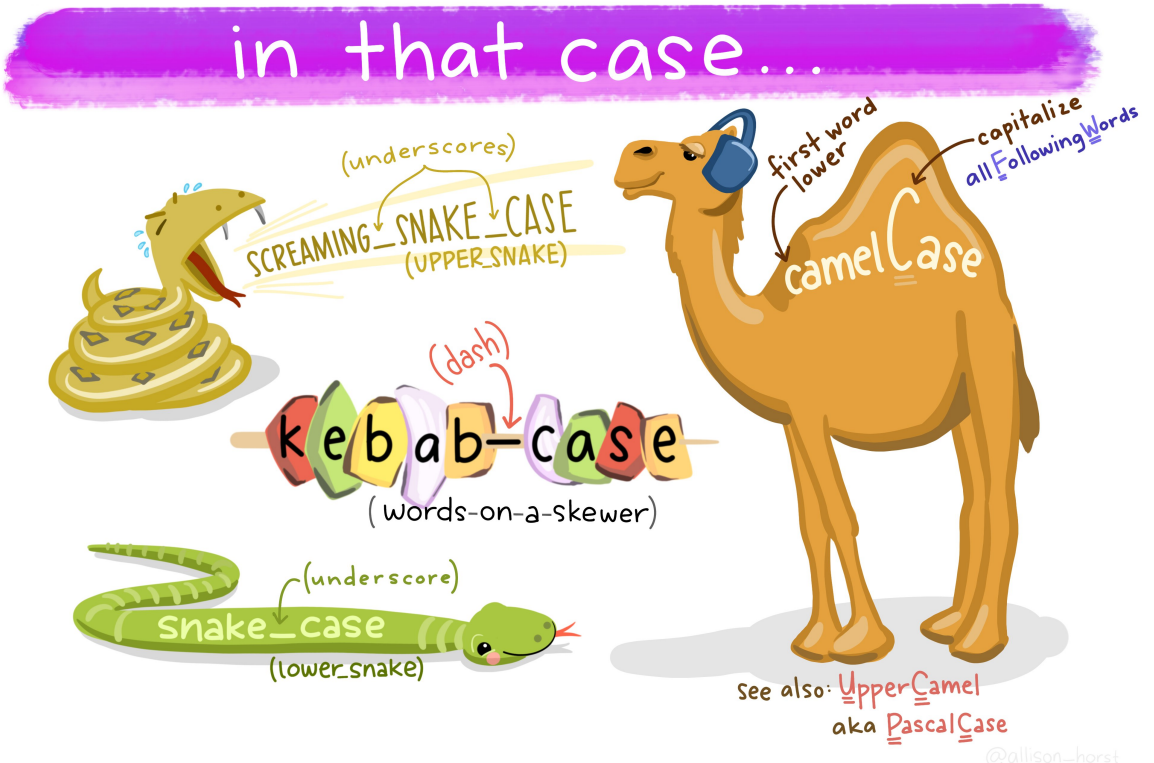
Data Analysis Using R

Dr. Arun Mitra

AMCHSS, SCTIMST

Before I forget..

R is Case Sensitive.



Which is better?

- Matter of personal choice... But...

Outline of the talk

- What is EDA?
- Generate questions about your data.
- Search for answers by visualising, transforming your data.
- Live Demo Session

What is EDA?

Exploratory Data Analysis

- **EDA** is the critical first step.
- **EDA** is a state of mind.
- **EDA** is exploring your ideas.
- **EDA** has no strict rules.
- **EDA** helps understand your data.
- **EDA** is an iterative cycle.
- **EDA** is a creative process.

What is EDA?

It is mostly a **philosophy** of data analysis where the researcher examines the data without any pre-conceived ideas in order to discover what the data can tell him or her about the phenomena being studied.

"detective work – numerical detective work – or counting detective work – or graphical detective work"

— **Tukey, 1977** *Page 1, Exploratory Data Analysis*

Questions

The easiest way to do **EDA** is to use questions as tools to guide your investigation. **EDA** is an important part of any data analysis, even if the questions are known already.

"There are no routine statistical questions, only questionable statistical routines."

— Sir David Cox

"Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise."

— John Tukey

Asking the right questions

Key to asking ***quality*** questions is to generate a large ***quantity*** of questions.

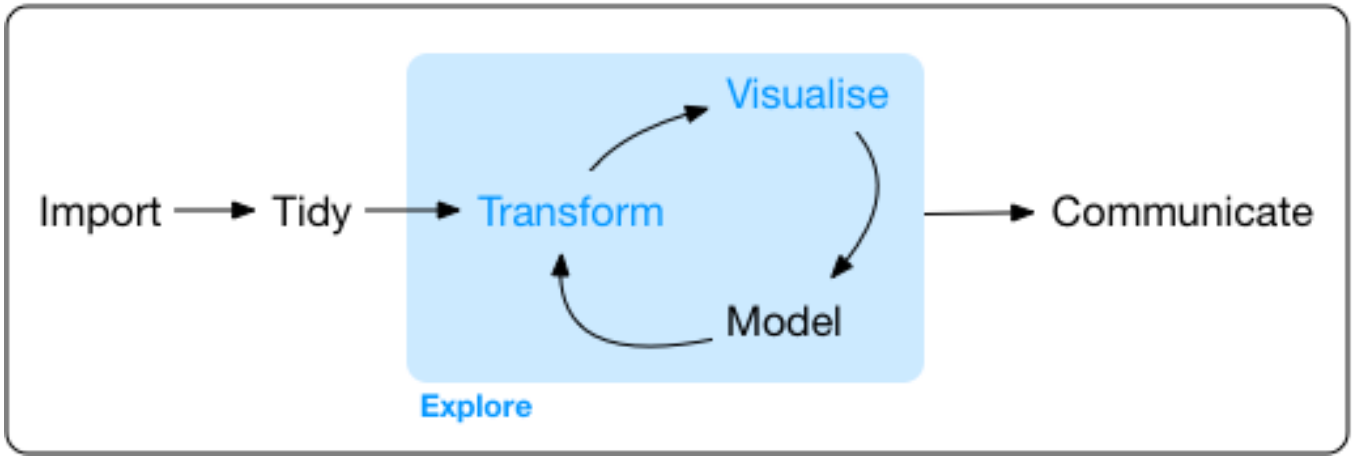
It is difficult to ask revealing questions at the start of the analysis.

But, each new question will expose a new aspect and increase your chance of making a discovery.

Questions to ask:

- What type of variation occurs within your variables?
- What type of covariation occurs between your variables?
- Whether your data meets your expectations or not.
- Whether the quality of your data is robust or not.

The process of EDA



1. Import
2. Tidy
3. Explore
    - Transform
    - Visualize
    - Transform
    - Visualize
    - Transform
    - Visualise ...
    - ...

What did we cover yesterday?

Concepts

- Health Data Science
- Reproducible Research
- Tidy Data

How to Visualize Data

- `ggplot2`

How to Import Data

- `readr`
- `readxl`
- `here::here()`

What will be covered today?

Preparing Tidy Data

- Data Cleaning
- Data Wrangling

Data Exploration

- Data Transformation
- Data Visualization

Statistical Analysis

Summary Tables

Before we get started

To recap what we learnt in the previous sessions..

We now know to work within the R Project environment.

`here::here()` makes it easy for us to manage file paths.

You can quickly have a look at your data using the `View()` and `glimpse()` functions.

Most of the tidy data is read as `tibble` which is a workhorse of `tidyverse`.

Getting Started with the Data Exploration Pipeline

Step Zero: Get your documents and data in place.

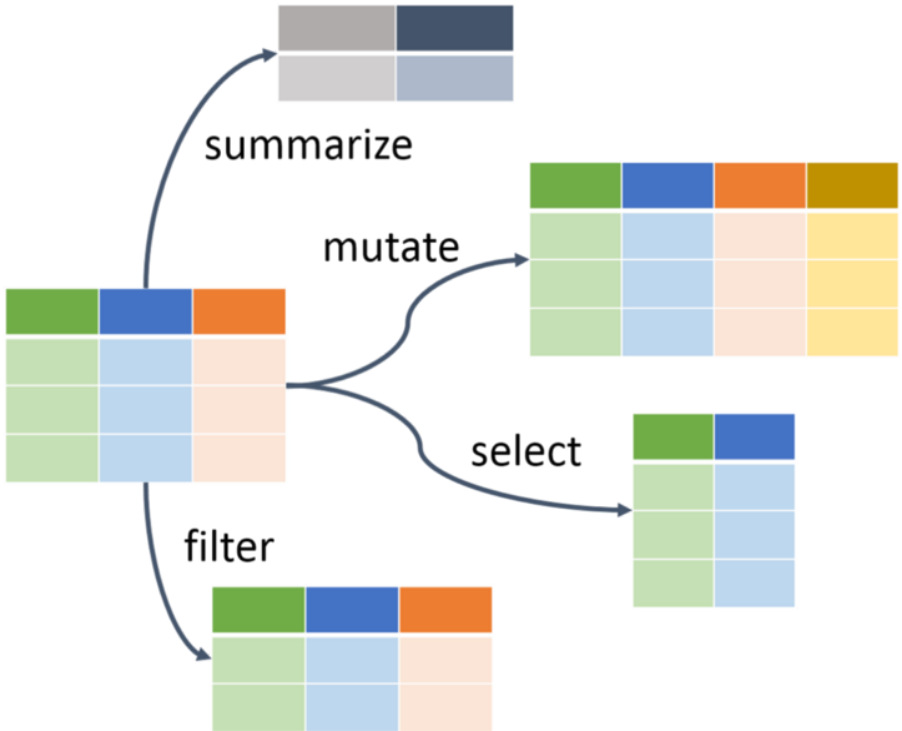
- Today we will be working in a RMarkdown Document (`.Rmd`).
- RMarkdown interweaves prose with code.
- Please download the `.Rmd` file provided to you via email and place it in the working directory.
- We will work from within yesterday's project.
- Yes, `here::here()`!

Today, we will be introducing you to three new packages:

1. `dplyr`
2. `skimr`
3. `DataExplorer`

`dplyr` Package

The `dplyr` is a powerful R-package to manipulate, clean and summarize unstructured data. In short, it makes data exploration and data manipulation easy and fast in R.

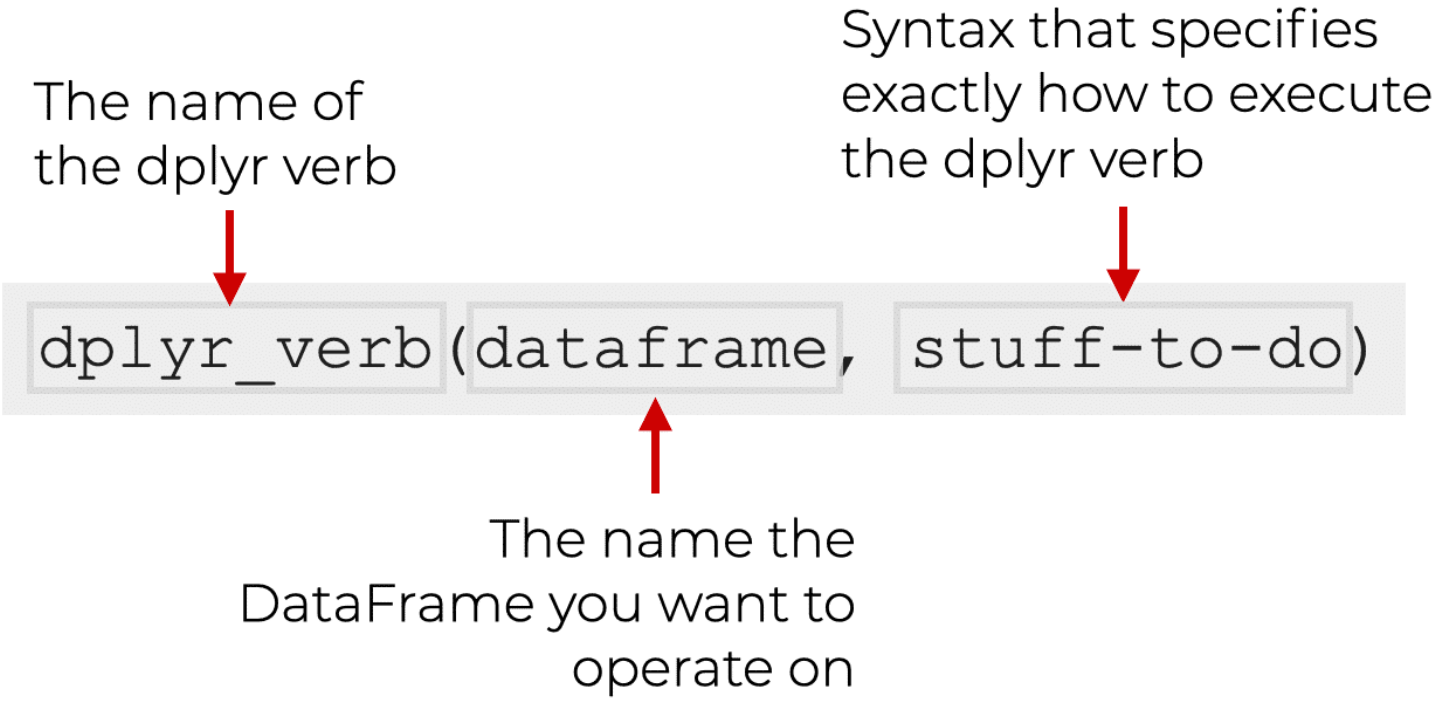


Verbs of the `dplyr`

There are many verbs in `dplyr` that are useful.

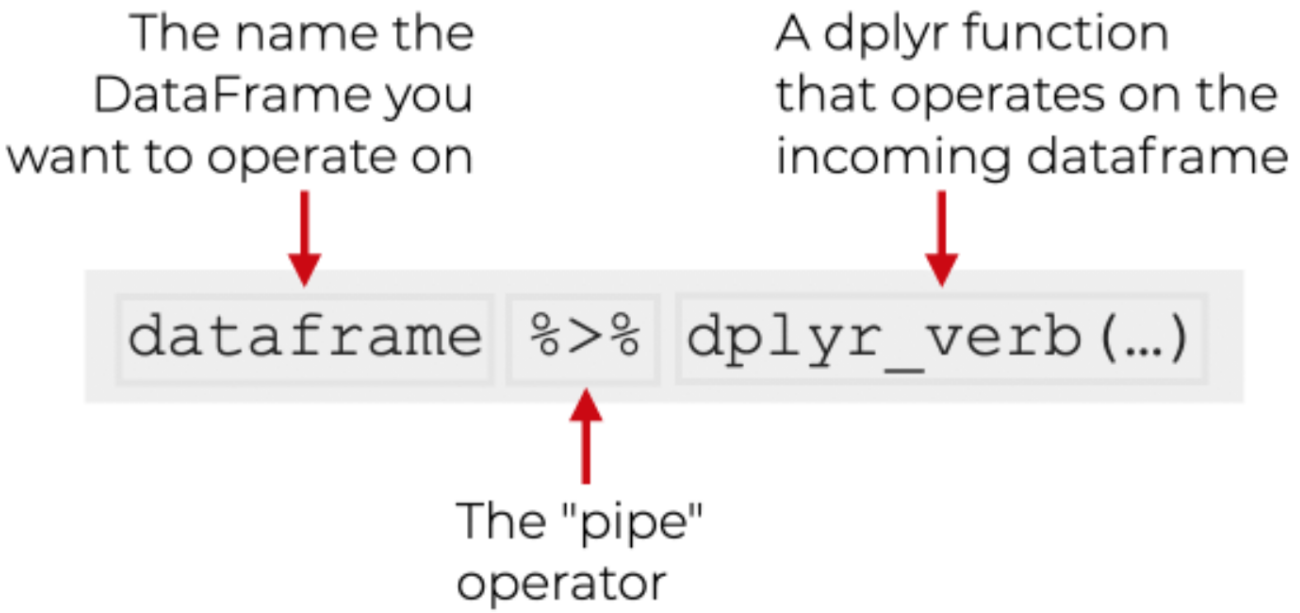
dplyr Function	Description
select()	Selecting columns (variables)
filter()	Filter (subset) rows.
group_by()	Group the data
summarise()	Summarise (or aggregate) data
arrange()	Sort the data
join()	Joining data frames (tables)
mutate()	Creating New Variables

Syntax of the `dplyr` verbs



Getting used to the pipe %>%

The pipe operator in `dplyr`



The pipe %>% means **THEN**...

The pipe is an operator in R that allows you to chain together functions in `dplyr`.

Lets load some sample data

```
tb <- read_csv(here("who_tubercolosis_data.csv"))

glimpse(tb)
```

```
## Rows: 3,850
## Columns: 18
## $ country                <chr> "Afghanistan", "Afghanistan", "Afghanistan"
## $ who_region             <chr> "EMR", "EMR", "EMR", "EMR", "EMR", "EMR", "
## $ year                   <dbl> 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2
## $ pop                    <dbl> 20093756, 20966463, 21979923, 23064851, 241
## $ incidence_100k         <dbl> 190, 189, 189, 189, 189, 189, 189, 189, 189
## $ incidence_number       <dbl> 38000, 40000, 42000, 44000, 46000, 47000, 4
## $ hiv_percent            <dbl> 0.36, 0.30, 0.26, 0.23, 0.22, 0.22, 0.22, 0
## $ hiv_incidence_100k     <dbl> 0.68, 0.57, 0.49, 0.44, 0.41, 0.42, 0.42, 0
## $ hiv_number             <dbl> 140, 120, 110, 100, 100, 100, 110, 120, 120
## $ mort_nohiv_100k        <dbl> 67.00, 62.00, 56.00, 57.00, 51.00, 46.00, 4
## $ mort_nohiv_number      <dbl> 14000, 13000, 12000, 13000, 12000, 12000, 1
## $ mort_hiv_100k          <dbl> 0.15, 0.17, 0.27, 0.25, 0.21, 0.19, 0.18, 0
## $ mort_hiv_number        <dbl> 31, 35, 60, 57, 50, 48, 46, 45, 48, 55, 56,
## $ mort_100k              <dbl> 67.00, 62.00, 56.00, 57.00, 51.00, 46.00, 4
## $ mort_number            <dbl> 14000, 13000, 12000, 13000, 12000, 12000, 1
## $ case_fatality_ratio    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
## $ new_incidence_100k     <dbl> 35, 48, 63, 60, 76, 87, 98, 108, 104, 93, 9
```

Let's look at the `head()`

```
head(tb)
```

```
## # A tibble: 6 × 18
##   country   who_r…¹  year    pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##   <chr>     <chr>    <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Afghanis… EMR       2000 2.01e7     190   38000    0.36    0.68     140
## 2 Afghanis… EMR       2001 2.10e7     189   40000    0.3     0.57     120
## 3 Afghanis… EMR       2002 2.20e7     189   42000    0.26    0.49     110
## 4 Afghanis… EMR       2003 2.31e7     189   44000    0.23    0.44     100
## 5 Afghanis… EMR       2004 2.41e7     189   46000    0.22    0.41     100
## 6 Afghanis… EMR       2005 2.51e7     189   47000    0.22    0.42     100
## # … with 8 more variables: mort_nohiv_number <dbl>, mort_hiv_100k <dbl>,
## #   mort_hiv_number <dbl>, mort_100k <dbl>, mort_number <dbl>,
## #   case_fatality_ratio <dbl>, new_incidence_100k <dbl>,
## #   case_detection_percent <dbl>, and abbreviated variable names ¹who_regi
## #   ²incidence_100k, ³incidence_number, ⁴hiv_percent, ⁵hiv_incidence_100k,
## #   ⁶hiv_number, ⁷mort_nohiv_100k
```

Let's look at the `names()`

```
dim(tb)


names(tb)
```

```
## [1] 3850   18
```

```
##  [1] "country"                "who_region"             "year"
##  [4] "pop"                    "incidence_100k"         "incidence_number"
##  [7] "hiv_percent"            "hiv_incidence_100k"     "hiv_number"
## [10] "mort_nohiv_100k"        "mort_nohiv_number"      "mort_hiv_100k"
## [13] "mort_hiv_number"        "mort_100k"              "mort_number"
## [16] "case_fatality_ratio"    "new_incidence_100k"     "case_detection_per
```

Let's look at the different countries

Find the unique countries in the bottom 50 rows of tb.

```
# without the pipe
unique(tail(tb, n = 50)$country)

# with the pipe
tb %>%
  tail(50) %>%
  distinct(country)
```

```
## [1] "Yemen"    "Zambia"   "Zimbabwe"
```

```
## # A tibble: 3 × 1
##   country
##   <chr>
## 1 Yemen
## 2 Zambia
## 3 Zimbabwe
```

Find the unique countries in the bottom 50 rows of `tb`

You will notice that we used different functions to complete our task.

The code without the pipe uses functions from base R while the code with the pipe uses a mixture (`tail()` from base R and `distinct()` from `dplyr`).

Not all functions work with the pipe, but we will usually opt for those that do when we have a choice.

`distinct()` and `count()`

The `distinct()` function will return the distinct values of a column, while `count()` provides both the distinct values of a column and then number of times each value shows up.

```
tb %>%
  distinct(who_region)
```

```
## # A tibble: 6 × 1
##   who_region
##   <chr>
## 1 EMR
## 2 EUR
## 3 AFR
## 4 WPR
## 5 AMR
## 6 SEA
```

```
tb %>%
  count(who_region)
```

```
## # A tibble: 6 × 2
##   who_region     n
##   <chr>      <int>
## 1 AFR          835
## 2 AMR          808
## 3 EMR          396
## 4 EUR          967
## 5 SEA          196
## 6 WPR          648
```

Notice that there is a new column produced by the count function called `n`.

arrange()

The `arrange()` function does what it sounds like. It takes a data frame or tbl and arranges (or sorts) by column(s) of interest.

The first argument is the data, and subsequent arguments are columns to sort on.

Use the `desc()` function to arrange by descending.

```
tb %>%
  count(who_region) %>%
  arrange(n)
```

```
## # A tibble: 6 × 2
##   who_region     n
##   <chr>      <int>
## 1 SEA          196
## 2 EMR          396
## 3 WPR          648
## 4 AMR          808
## 5 AFR          835
## 6 EUR          967
```

```
tb %>%
  count(who_region) %>%
  arrange(desc(n)) # use can als
```

```
## # A tibble: 6 × 2
##   who_region     n
##   <chr>      <int>
## 1 EUR          967
## 2 AFR          835
## 3 AMR          808
## 4 WPR          648
## 5 EMR          396
## 6 SEA          196
```

Logical Operators

Here are the logical criteria in R:

- ==: *Equal to*
- !=: *Not equal to*
- >: *Greater than*
- >=: *Greater than or equal to*
- <: *Less than*
- <=: *Less than or equal to*

If you want to satisfy *all* of multiple conditions, you can use the "and" operator, &.

The "or" operator | (the vertical pipe character, shift-backslash) will return a subset that meet *any* of the conditions.

Filter 2015 and above

```
tb %>%
  filter(year >= 2015)
```

```
## # A tibble: 648 × 18
##    country  who_r…¹  year    pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##    <chr>    <chr>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Afghani… EMR      2015 3.37e7   189     64000     0.3     0.57     190
##  2 Afghani… EMR      2016 3.47e7   189     65000     0.31    0.58     200
##  3 Afghani… EMR      2017 3.55e7   189     67000     0.31    0.59     210
##  4 Albania  EUR      2015 2.92e6    16       480     0.82    0.21       6
##  5 Albania  EUR      2016 2.93e6    16       480     0.83    0.21       6
##  6 Albania  EUR      2017 2.93e6    20       580     0.85    0.17       5
##  7 Algeria  AFR      2015 3.99e7    74     30000     0.58    0.43     170
##  8 Algeria  AFR      2016 4.06e7    70     29000     0.6     0.42     170
##  9 Algeria  AFR      2017 4.13e7    70     29000     0.62    0.43     180
## 10 America… WPR      2015 5.55e4     8.3       5     0.1     0.01       0
## # … with 638 more rows, 8 more variables: mort_nohiv_number <dbl>,
## #   mort_hiv_100k <dbl>, mort_hiv_number <dbl>, mort_100k <dbl>,
## #   mort_number <dbl>, case_fatality_ratio <dbl>, new_incidence_100k <dbl>
## #   case_detection_percent <dbl>, and abbreviated variable names ¹who_regi
## #   ²incidence_100k, ³incidence_number, ⁴hiv_percent, ⁵hiv_incidence_100k,
## #   ⁶hiv_number, ⁷mort_nohiv_100k
```

Filter India

```
tb %>%
  filter(country == "India")
```

```
## # A tibble: 18 × 18
##    country who_re…¹  year     pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##    <chr>   <chr>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 India   SEA       2000 1.05e9     289 3040000       7      20  213000
##  2 India   SEA       2001 1.07e9     288 3090000       7      20  216000
##  3 India   SEA       2002 1.09e9     287 3130000       7      20  219000
##  4 India   SEA       2003 1.11e9     285 3160000       7      20  221000
##  5 India   SEA       2004 1.13e9     282 3180000       7      20  223000
##  6 India   SEA       2005 1.14e9     279 3190000       7      20  223000
##  7 India   SEA       2006 1.16e9     274 3180000       7      19  223000
##  8 India   SEA       2007 1.18e9     268 3160000       7      19  221000
##  9 India   SEA       2008 1.20e9     261 3130000       6.8    18  214000
## 10 India   SEA       2009 1.21e9     254 3090000       6.6    17  204000
## 11 India   SEA       2010 1.23e9     247 3050000       6.3    16  191000
## 12 India   SEA       2011 1.25e9     241 3000000       5.9    14  176000
## 13 India   SEA       2012 1.26e9     234 2960000       5.4    13  160000
## 14 India   SEA       2013 1.28e9     228 2920000       4.9    11  142000
## 15 India   SEA       2014 1.29e9     223 2880000       4.3     9.5 123000
## 16 India   SEA       2015 1.31e9     217 2840000       4       8.6 113000
```

Both India and 2015 or more recent

```
tb %>%
  filter(year >= 2015 & country == "India")
```

```
## # A tibble: 3 × 18
##   country who_reg…¹  year     pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##   <chr>   <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 India   SEA        2015 1.31e9     217 2840000     4         8.6  113000
## 2 India   SEA        2016 1.32e9     211 2790000     3.3       7     92000
## 3 India   SEA        2017 1.34e9     204 2740000     3.1       6.4   86000
## # … with 8 more variables: mort_nohiv_number <dbl>, mort_hiv_100k <dbl>,
## #   mort_hiv_number <dbl>, mort_100k <dbl>, mort_number <dbl>,
## #   case_fatality_ratio <dbl>, new_incidence_100k <dbl>,
## #   case_detection_percent <dbl>, and abbreviated variable names ¹who_regi
## #   ²incidence_100k, ³incidence_number, ⁴hiv_percent, ⁵hiv_incidence_100k,
## #   ⁶hiv_number, ⁷mort_nohiv_100k
```

`%in%` function

To `filter()` a categorical variable for only certain levels, we can use the `%in%` operator.

Let's see data from India, Nepal, Pakistan and Bangladesh First we will have to figure out how those are spelled in this dataset.

Open the spreadsheet viewer and find out.

We'll see a way to find them in code later on in the course.

Indian Subcontinent

```
indian_subcont <- c("India",
           "Nepal",
           "Pakistan",
           "Bangladesh")

tb %>% filter(country %in% indian_subcont)
```

```
## # A tibble: 72 × 18
##    country  who_r…¹  year    pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##    <chr>    <chr>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Banglad… SEA      2000 1.32e8     221  291000    0.07    0.14     190
##  2 Banglad… SEA      2001 1.34e8     221  297000    0.07    0.16     210
##  3 Banglad… SEA      2002 1.37e8     221  302000    0.08    0.18     240
##  4 Banglad… SEA      2003 1.39e8     221  307000    0.09    0.2      280
##  5 Banglad… SEA      2004 1.41e8     221  313000    0.1     0.22     320
##  6 Banglad… SEA      2005 1.43e8     221  317000    0.12    0.26     370
##  7 Banglad… SEA      2006 1.45e8     221  322000    0.13    0.28     410
##  8 Banglad… SEA      2007 1.47e8     221  325000    0.14    0.3      450
##  9 Banglad… SEA      2008 1.49e8     221  329000    0.14    0.32     470
## 10 Banglad… SEA      2009 1.50e8     221  333000    0.14    0.32     480
## # … with 62 more rows, 8 more variables: mort_nohiv_number <dbl>,
## #   mort_hiv_100k <dbl>, mort_hiv_number <dbl>, mort_100k <dbl>,
## #   mort_number <dbl>, case_fatality_ratio <dbl>, new_incidence_100k <dbl>
```

Other Useful Functions

`drop_na()`

The `drop_na()` function is extremely useful for when we need to subset a variable to remove missing values.

`select()`

Whereas the `filter()` function allows you to return only certain *rows* matching a condition, the `select()` function returns only certain *columns*. The first argument is the data, and subsequent arguments are the columns you want.

summarize()

The `summarize()` function summarizes multiple values to a single value.

On its own the `summarize()` function doesn't seem to be all that useful.

The dplyr package provides a few convenience functions called `n()` and `n_distinct()` that tell you the number of observations or the number of distinct values of a particular variable.

`summarize()` is the same as `summarise()`

```
tb %>%
  summarize(hiv_percent = mean(hiv_percent, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   hiv_percent
##         <dbl>
## 1        12.1
```

group_by()

We saw that `summarize()` isn't that useful on its own. Neither is `group_by()`.

All this does is takes an existing data frame and converts it into a grouped data frame where operations are performed by group.

```
tb %>%
  group_by(year)
```

```
## # A tibble: 3,850 × 18
## # Groups:   year [18]
##    country  who_r…¹  year    pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##    <chr>    <chr>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Afghani… EMR      2000 2.01e7     190   38000    0.36    0.68     140
##  2 Afghani… EMR      2001 2.10e7     189   40000    0.3     0.57     120
##  3 Afghani… EMR      2002 2.20e7     189   42000    0.26    0.49     110
##  4 Afghani… EMR      2003 2.31e7     189   44000    0.23    0.44     100
##  5 Afghani… EMR      2004 2.41e7     189   46000    0.22    0.41     100
##  6 Afghani… EMR      2005 2.51e7     189   47000    0.22    0.42     100
##  7 Afghani… EMR      2006 2.59e7     189   49000    0.22    0.42     110
##  8 Afghani… EMR      2007 2.66e7     189   50000    0.23    0.43     120
##  9 Afghani… EMR      2008 2.73e7     189   52000    0.23    0.44     120
## 10 Afghani… EMR      2009 2.80e7     189   53000    0.24    0.45     130
```

Two variable `group_by()`

```
tb %>%
  group_by(year, who_region)
```

```
## # A tibble: 3,850 × 18
## # Groups:   year, who_region [108]
##    country  who_r…¹  year    pop incid…² incid…³ hiv_p…⁴ hiv_i…⁵ hiv_n…⁶ mo
##    <chr>    <chr>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Afghani… EMR      2000 2.01e7     190   38000    0.36    0.68     140
##  2 Afghani… EMR      2001 2.10e7     189   40000    0.3     0.57     120
##  3 Afghani… EMR      2002 2.20e7     189   42000    0.26    0.49     110
##  4 Afghani… EMR      2003 2.31e7     189   44000    0.23    0.44     100
##  5 Afghani… EMR      2004 2.41e7     189   46000    0.22    0.41     100
##  6 Afghani… EMR      2005 2.51e7     189   47000    0.22    0.42     100
##  7 Afghani… EMR      2006 2.59e7     189   49000    0.22    0.42     110
##  8 Afghani… EMR      2007 2.66e7     189   50000    0.23    0.43     120
##  9 Afghani… EMR      2008 2.73e7     189   52000    0.23    0.44     120
## 10 Afghani… EMR      2009 2.80e7     189   53000    0.24    0.45     130
## # … with 3,840 more rows, 8 more variables: mort_nohiv_number <dbl>,
## #   mort_hiv_100k <dbl>, mort_hiv_number <dbl>, mort_100k <dbl>,
## #   mort_number <dbl>, case_fatality_ratio <dbl>, new_incidence_100k <dbl>
## #   case_detection_percent <dbl>, and abbreviated variable names ¹who_regi
## #   ²incidence_100k, ³incidence_number, ⁴hiv_percent, ⁵hiv_incidence_100k,
```

`group_by()` and `summarize()` together

The real power comes in where `group_by()` and `summarize()` are used together. First, write the `group_by()` statement. Then pipe the result to a call to `summarize()`.

```
tb %>%
  group_by(year) %>%
  summarize(mean_inc = mean(incidence_100k, na.rm = TRUE))
```

```
## # A tibble: 18 × 2
##     year mean_inc
##    <dbl>    <dbl>
##  1  2000     138.
##  2  2001     138.
##  3  2002     140.
##  4  2003     141.
##  5  2004     139.
##  6  2005     137.
##  7  2006     135.
##  8  2007     132.
##  9  2008     129.
```

mutate()

Mutate creates a new variable or modifies an existing one.

ifelse()

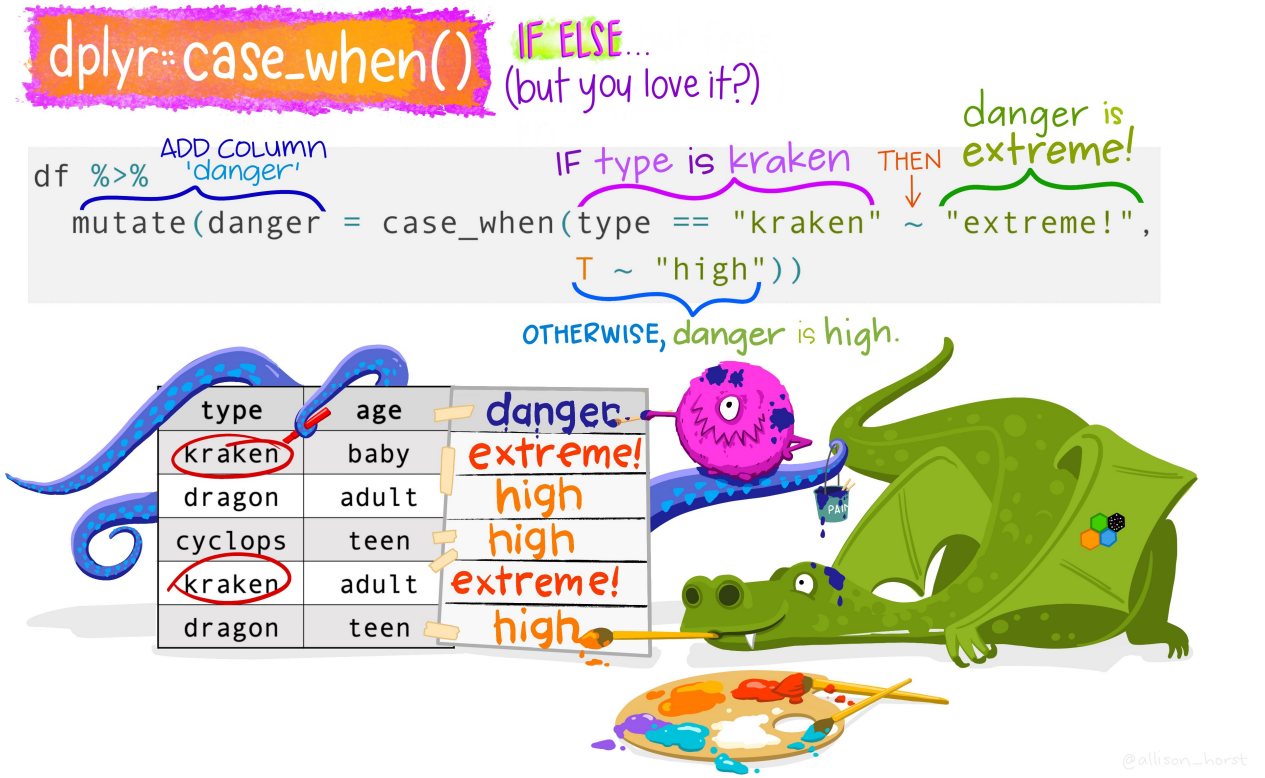
Lets create a column called `ind_sub` if the country is in the Indian Subcontinent.

```
tb %>%
  mutate(indian_sub1 = ifelse(country %in% indian_subcont,
                              "Indian Subcontinent", "Others"))
```

```
## # A tibble: 3,850 × 19
##    country  who_r…[1]  year    pop incid…[2] incid…[3] hiv_p…[4] hiv_i…[5] hiv_n…[6] mo
##    <chr>    <chr>     <dbl>  <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 Afghani… EMR        2000 2.01e7      190    38000     0.36     0.68      140
##  2 Afghani… EMR        2001 2.10e7      189    40000     0.3      0.57      120
##  3 Afghani… EMR        2002 2.20e7      189    42000     0.26     0.49      110
##  4 Afghani… EMR        2003 2.31e7      189    44000     0.23     0.44      100
##  5 Afghani… EMR        2004 2.41e7      189    46000     0.22     0.41      100
##  6 Afghani… EMR        2005 2.51e7      189    47000     0.22     0.42      100
##  7 Afghani… EMR        2006 2.59e7      189    49000     0.22     0.42      110
##  8 Afghani… EMR        2007 2.66e7      189    50000     0.23     0.43      120
##  9 Afghani… EMR        2008 2.73e7      189    52000     0.23     0.44      120
## 10 Afghani… EMR        2009 2.80e7      189    53000     0.24     0.45      130
## # … with 3,840 more rows, 9 more variables: mort_nohiv_number <dbl>,
## #   mort_hiv_100k <dbl>, mort_hiv_number <dbl>, mort_100k <dbl>,
```

case_when()

Alternative of `ifelse()`



`ifelse()` vs `case_when()`

Note that the `if_else()` function may result in slightly shorter code if you only need to code for 2 options.

For more options, nested `if_else()` statements become hard to read and could result in mismatched parentheses so `case_when()` will be a more elegant solution.

More Resources for `dplyr`

- Check out the Data Wrangling Cheatsheet that covers dplyr and tidyr functions.
- Review the Tibbles chapter of the excellent, free ***R for Data Science* book**.
- Check out the Transformations chapter to learn more about the `dplyr` package. Note that this chapter also uses the graphing package `ggplot2` which we have covered yesterday.

`skimr` Package

`skimr` is designed to provide summary statistics about variables in data frames, tibbles, data tables and vectors.

The core function of `skimr` is the `skim()` function, which is designed to work with (grouped) data frames, and will try coerce other objects to data frames if possible.

`DataExplorer` Package

The `DataExplorer` package aims to automate most of data handling and visualization, so that users could focus on studying the data and extracting insights.^[DataExplorer Package]

The single most important function from the `DataExplorer` package is `create_report()`

Lets Begin!