

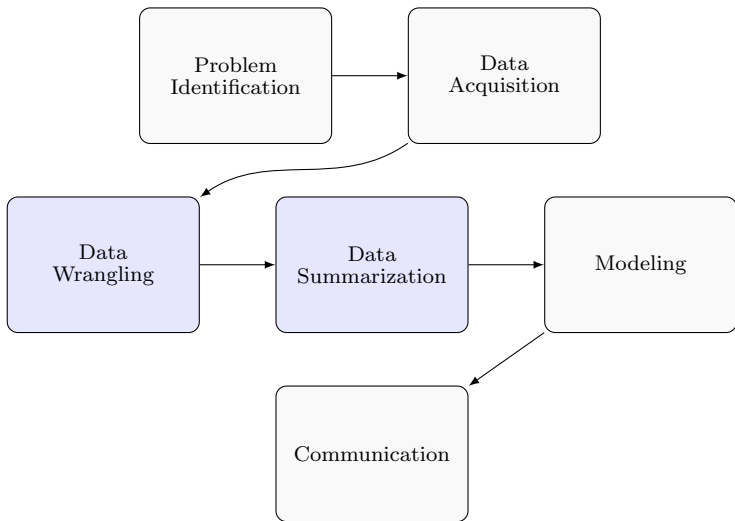
An Introduction to R with the Tidyverse

ART Forum 2019

Preamble

- This is *an* introduction to R.
- My goal is to get you doing as much in R as quickly as possible.
- Materials at github.com/marcdotson/introduction-to-r.

Data Analysis Process



R and Data
Manipulation

Data Description
and Visualization

Cleaning Data
and Summarization

R/RStudio

R is an open-source programming language for statistical computing, data analysis, and data science.

cran.r-project.org

RStudio is an integrated development environment (IDE) that makes it easier to use R.

rstudio.com

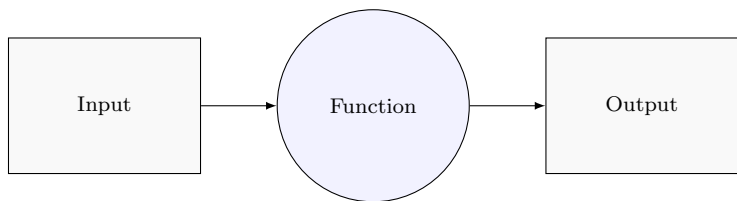
Go to our shared RStudio Cloud project bit.ly/2V1QoGf.

Orientation

- *Console*: Where you run code.
- *Source*: Create and save R scripts and send code to the Console.

```
# Use comments to explain (the why of) your code.  
2 * (18 - 7)
```
- *Environment*: A snapshot of what you have loaded.
- *Help*: Look up documentation.

Functions



- Functions are composed of **arguments** that tell the function how to operate.
- Using a function is referred to as a “call” or a “function call.”
- Don’t forget you have *Help*.

Packages

- A **package** is a collection of functions, documentation, and sometimes data.
- There are a number of packages that are part of base R.
- You can install other packages from CRAN.
- Not all packages are created equal.

The Tidyverse

“The **tidyverse** is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.”

readr – importing data

dplyr – manipulating data

tidyr – cleaning data

ggplot2 – visualizing data

```
# Load the (already installed) tidyverse.  
library(tidyverse)
```

Importing Data

- Let's import `store_data.csv`.

```
store_data <- read_csv("store_data.csv")
```

- Note that `store_data` now appears in our Environment. The Environment lists **objects** we've assigned a name in our script.
- You should always use an **R Project** to organize your work and set your **working directory** (where R looks for data to import).

Data Frames

- Data frames are the most common data structure in R.

`store_data`

- What can we learn by printing `store_data`?

Data Manipulation

- The heart of data wrangling is **data manipulation**.
 - Filter observations.
 - Arrange observations.
 - Select variables.
 - Mutate variables (i.e., recode or create new variables).
 - Join data frames.
- One of the most-used packages, dplyr provides a consistent **grammar of data manipulation** with functions (a.k.a., **verbs**) that mirror SQL.

The Pipe

- Part of the common philosophy for the tidyverse is that:
 1. Each function should do one specific thing well.
 2. Each function should have a data frame as an input and a data frame as an output.
- This allows us to to `%>%` together functions in consecutive lines of code so that it is easy for humans to read.

Filter Observations

- We often want to filter our data by **keeping certain observations**.

```
store_data %>%  
  filter(gender == "Female")
```

```
store_data %>%  
  filter(store_spend > 100)
```

- How would we filter by `gender == "Female"` and `store_spend > 100`?
- Why are we putting quotes around "Female" but not `gender`?

Arrange Observations

- Arrange observations to reveal helpful information and check data.

```
store_data %>%  
  arrange(store_trans)
```

```
store_data %>%  
  arrange(desc(store_trans))
```

Select Variables

- Sometimes we only care about **keeping certain variables**, especially when working with a large dataset.

```
store_data %>%  
  select(store_spend, age, gender)
```


Mutate Variables

- We can also **recode existing variables** or **create new variables**.

```
store_data %>%  
  mutate(store_spend = store_spend / 100)
```

- Note how we can **overwrite** variables in a data frame as well as objects if we use the same name.

Join Data Frames

- In the simplest case, a common variable (like an ID) allows us to join two data frames.

```
sat_data <- read_csv("sat_data.csv")
```

```
crm_data <- store_data %>%  
  left_join(sat_data, by = "id")
```

- Print `crm_data`.
- Other common joins include:
 - `inner_join` to keep everything that has a matching common variable in both the left and right data frames.
 - `anti_join` to keep everything that *doesn't* have a matching common variable in both the left and right data frames.

Exercise

In *consecutive lines of code*, do the following.

1. Join `store_data`, `sat_data`, and `online_data` by `= "id"`.
2. Filter the data to keep only observations in the "US".
3. Select `id`, `store_spend`, `online_spend`, and `gender`.
4. Create `total_spend = store_spend + online_spend`.
5. Find who has the highest `total_spend`.
6. Practice reading this code (read `%>%` as “then”).

R and Data
Manipulation

Data Description
and Visualization

Cleaning Data
and Summarization

Data Summarization

- Data summarization includes the following.
 - Describing data with numerical summaries (i.e., **statistics**).
 - Visualizing data with graphical summaries.
- How we summarize depends on the whether the data is **discrete** or **continuous**.
 - Discrete is also called qualitative or categorical.
 - Continuous is also called quantitative or numerical.
- We will use both dplyr and ggplot2 to summarize data.

Describing Discrete Data

- The simplest numeric summary for a discrete variable is a **count**.

```
crm_data %>%  
  count(gender)
```

- Now get a count by both gender and country to produce a “tidy” **cross-tab**.

Visualizing Discrete Data

- Perhaps the most popular package, ggplot2 uses a consistent **grammar of graphics** built with **layers**.
 1. Data – Data to visualize.
 2. Aesthetics – Mapping graphical elements to data.
 3. Geometries – Or “geom,” the graphic representing the data.
- Let's plot our previous summary (note how + is different from %>%).

```
ggplot(crm_data, aes(x = gender)) +  
  geom_bar()
```

- Let's visualize a second variable. Add the aesthetic `fill = country`.
- The geom `position` argument is set to "stack" by default. Try "fill" instead.

Describing Continuous Data

- The simplest numeric summary for a continuous variable is a **mean**.

```
crm_data %>%  
  summarize(avg_store_spend = mean(store_spend))
```

- Note that `summarize()` is a more general version of `count()`.
- Compute the mean of both `store_spend` and `sat_overall`.
- We can also compute the **mode**, **median**, **variance**, **standard deviation**, **minimum**, **maximum**, etc.

Visualizing Continuous Data

- Let's plot the **distribution** of `store_spend`.

```
ggplot(crm_data, aes(x = store_spend)) +  
  geom_histogram()
```

- Change the geom `bins` argument to 10.
- Visualize the relationship between `store_spend` and `sat_overall`.

```
crm_data %>%  
  ggplot(aes(x = store_spend, y = sat_overall)) +  
  geom_point()
```

- As part of the aesthetic, `log(store_spend + 1)`.
- Play with the `size` and `alpha` geom arguments.
- Add a `geom_smooth()` layer.

Describing Discrete and Continuous Data

- **Grouped summaries** provide a powerful solution for describing a combination of discrete and continuous data.

```
crm_data %>%  
  group_by(gender, country) %>%  
  summarize(  
    n = n(),  
    avg_store_spend = mean(store_spend),  
    avg_sat_overall = mean(sat_overall)  
  ) %>%  
  arrange(desc(avg_store_spend))
```

- Grouping by a discrete variable is equivalent to filtering the data separately for each value of the group variable.
- `count()` is a **wrapper** around a grouped summary with `n()`.

Visualizing Discrete and Continuous Data

- There are geoms specific to visualizing the relationship between discrete and continuous data.
- Use `geom_boxplot()` and `geom_density()` to visualize the relationship between `gender` and `sat_overall`.

Adding Layers

- There are numerous options to quickly create impressive visuals.

```
crm_data %>%  
  ggplot(  
    aes(  
      x = log(store_spend + 1),  
      y = sat_overall,  
      color = gender  
    )  
  ) +  
  geom_jitter(size = 2, alpha = 0.5) +  
  geom_smooth(method = "lm", se = FALSE) +  
  facet_wrap(~ country) +  
  ggtitle("Store Spend by Overall Satisfaction")
```

Exercise

In *consecutive lines of code*, do the following.

1. Join `store_data`, `sat_data`, and `online_data` by `= "id"`.
2. Filter the data to keep only observations in the "CA".
3. Create `total_spend = store_spend + online_spend`.
4. Assign this final data frame to `ca_crm_data`.

Use `ca_crm_data` to explore the following questions.

1. How many men and women are there?
2. Is there a relationship between `credit_score` and `sat_overall`?
3. What is the average `total_spend` by gender?
4. Is the relationships between `total_spend` and `sat_overall` different for men and women?

R and Data
Manipulation

Data Description
and Visualization

Cleaning Data
and Summarization

Tidy Data

- **Tidy data** is defined as follows:
 1. Each observation has its own row.
 2. Each variable has its own column.
 3. Each value has its own cell.
- Fourth (sort of), each table has one type of observational unit.
- This may seem obvious or simple, but this common philosophy is at the heart of the *tidyverse*.
- It also means we will often prefer **long** datasets to **wide** datasets.

Gather Columns

- The most common problem with **messy data** is when columns are really values. Where is this problem in `online_data`?
- Use `tidyr` to `gather()` the columns into key-value variable pairs.

```
online_data <- online_data %>%  
  gather(  
    key = week,  
    value = visits,  
    week1_visit:week4_visit  
  )
```

- Note how this makes a wide dataset long (or at least *longer*) by iteratively transposing and stacking data.

Spread Columns

- If the data has the opposite problem (i.e., values that should really be columns), `spread()` key-value pairs into columns.

```
online_data %>%  
  spread(key = week, value = visits)
```

Separate and Unite Columns

- If two (or more) values are in one column, `separate()` the values into two (or more) columns.

```
online_data <- online_data %>%  
  separate(year_mo, c("year", "month"))
```

- When two (or more) values should be in one column, `unite()` the values into one column.

```
online_data %>%  
  unite(year_mo, year, month)
```

Wrangling \leftrightarrow Summarization

- Tidy data changes how we can summarize.

```
online_data %>%  
  group_by(id) %>%  
  summarize(total_visits = sum(visits)) %>%  
  left_join(sat_data, by = "id")
```

- Continue the consecutive lines of code to visualize the relationship between `total_visits` and `sat_overall` by `country`.
- What's the difference between assigning `color` as an aesthetic at the `ggplot` vs. the `geom` layer?

Inspecting Data

- What is the data structure for `online_data`?
- What about the data structure for `faithful`?
- How do we inspect data (whether or not its a tibble)?

```
str(faithful)
summary(faithful)
class(faithful)
dim(faithful)
names(faithful)
head(faithful)
tail(faithful)
```

- We could also **coerce** data into a tibble: `as_tibble(faithful)`.

Data Types and Coercion

- We can also coerce data types.
 - Numeric (i.e., double) with `as.numeric()`
 - Integer with `as.integer()`
 - Character with `as.character()`
 - **Factor** with `as.factor()`
- Why would we want to coerce a data type?

```
online_data %>%  
  filter(visits > 700) %>%  
  ggplot(aes(x = id, y = visits)) +  
  geom_col()
```

- The `x` in a bar/column plot needs to be discrete. Coerce `id` to be a factor and try the plot again.

Missing Values

- **Missing values** are (and should be) encoded as NA.
- NA is treated in a particular way.

```
online_data %>%  
  group_by(year) %>%  
  summarize(avg_visits = mean(visits))
```

- Add the argument `na.rm = TRUE` to the `mean()` function.
- Because NA is a special character, it can be referenced.

```
online_data2 <- online_data %>%  
  filter(visits != is.na(visits))
```

```
unique(online_data2$visits)
```

Exercise

In *consecutive lines of code*, do the following.

1. Using the latest `online_data`, separate `week` into two variables.
2. Use `select()` to get rid of the variable with “visit” repeated.
3. Group by `id` and compute the `total_visits`.
4. Join this `online_data` with `store_data` and `sat_data`.
5. Overwrite `crm_data` with this final data frame.

Use the new `crm_data` to explore the following questions.

1. Visualize the average `total_visits` by `country` and `gender`.
Which group has the highest average `total_visits`?
2. What's the relationship between `total_visits` and `store_spend`? Is this different for each `country`?

Epilogue

- The tidyverse represents the state of the art for data wrangling and summarization in R and serves as the foundation for a growing number of additional packages (e.g., tidytext).
- *R for Data Science* r4ds.had.co.nz
- DataCamp datacamp.com
- Cheatsheets rstudio.com/resources/cheatsheets
- The tidyverse tidyverse.org
- Watch [Hadley Wickham explain](#)

Advanced Topics

- Iteration r4ds.had.co.nz/iteration.html
- List Columns adv-r.had.co.nz
- Database Queries tidyverse.org/articles
- Tidymodels tidyverse.org/articles
- *Advanced R* adv-r.had.co.nz