

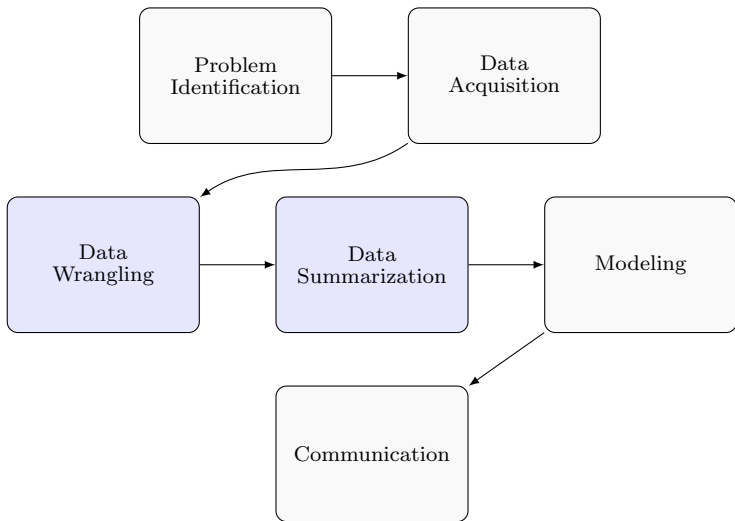
# An Introduction to R with the Tidyverse

ART Forum 2018

# Preamble

- This is *an* introduction to R.
- My goal is to get you doing as much in R as quickly as possible.
- This is the first two weeks of my Marketing Analytics class.
- I assume you know nothing about R.
- We'll focus on data wrangling and summarization.
- The slides (and cheatsheet) are designed to serve as a reference.
- Slides and data at [github.com/marcdotson/introduction-to-r](https://github.com/marcdotson/introduction-to-r).

# Data Analysis Process



R and Data  
Manipulation

Data Description  
and Visualization

Cleaning Data  
and Summarization

# R

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

R

[cran.r-project.org](https://cran.r-project.org)

RStudio

[www.rstudio.com](https://www.rstudio.com)

Open An Introduction to R with the Tidyverse.Rproj.

# R/RStudio Orientation

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

```
# Use comments to explain your code.
```

```
2 * (18 - 7)
```

- Environment: A snapshot of what you have loaded.
- Help: Look up documentation.

```
?library
```

## RStudio Preferences

- General > Save workspace to .RData on exit: Never.
- Code > Editing > Execution > Ctrl + Enter executes: Multiple consecutive R lines.
- Code > Display > General > Highlight selected line.

# R Packages

- An **R package** is a collection of functions, documentation, and sometimes data.
- There are a number of packages that are part of the base installation of R (look at Packages tab).
- You can download and load other packages from CRAN.
- Not all packages are created equal.



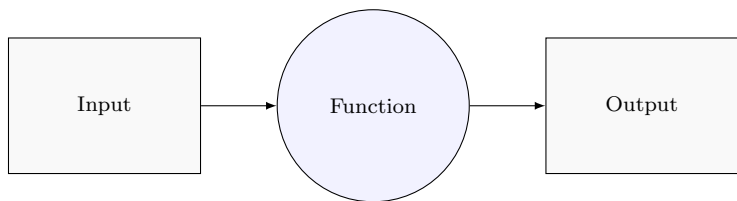
# The Tidyverse

- The **tidyverse** is a collection of R packages that share common philosophies and are designed to work together.
  - readr – importing data
  - dplyr – manipulating data
  - tidyr – cleaning data
  - ggplot2 – visualizing data
- 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.

```
install.packages("tidyverse")  
library(tidyverse)
```

- For more detail about the tidyverse, visit the [website](#) and watch [Hadley Wickham explain](#).

# 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.

# Importing Data

- To import data, R needs to know where to look.
- This folder is called the **working directory**.

```
getwd()
```

- Using an **R Project** is the best way to set your working directory, share your code, and open up multiple instances of R.
- Let's import `store_data.csv`.

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

- We can always use `?read_csv` if we have questions about the arguments.
- Note that `store_data` now appears in our Environment.
- The Environment lists any **objects** (i.e., data or functions) we've assigned a name in our script.
- We can view objects in their own tab.

# Data Frames

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

`store_data`

- How many **observations** and **variables**?
- Each column has a name. What is consistent about how the columns are named?
- Each column is also known as a **vector** (a data structure all its own) where each vector can be a different **data type**. What types do you see here?

# Data Manipulation

- The heart of data wrangling is **data manipulation**.
  - Filter observations.
  - Arrange observations.
  - Select variables.
  - 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 Operator

- One of the common philosophies of 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.

# Filtering 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))
```



## Selecting Variables

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

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

## Recoding/Creating New 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.

## Joining 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`.

## 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 (as part of an **exploratory analysis**) is all about discovery: What is your data saying?
  - Describe data with numerical summaries (i.e., **statistics**).
  - Visualize 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 = drv**.
- 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`.

```
ggplot(mpg, 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`.
- 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 and Spread 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*).
- 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

- You can now (hopefully) do quite a bit with R!
- We've covered the basics of using R with the tidyverse.
- Practiced some essential data manipulation functions with dplyr.
- Explored the flexibility of grouped summaries for describing data.
- Practice the basics of data visualization with ggplot2.
- Discussed the philosophy of tidy data.
- Used four functions for cleaning data with tidyr.

# What About Base R?!

- Base R refers (primarily) to the packages that come with R.
- In my experience, it's easier to start with the tidyverse.
  - Data frames (i.e., tibbles).
  - Functions across packages work together.
  - The `%>%` adds clarity to writing and reading code.
  - It helps make some difficult analyses trivial (e.g., text analysis).
- The tidyverse doesn't (and never will) cover everything you want to do with R, it's simply a great place to start.



## What's Next?

- *R for Data Science*
- *R for Marketing Research and Analytics*
- DataCamp
- Access detailed cheatsheets under Help.
- For modeling, follow-up with the Stan tutorial.
- For text analysis, stick around for my second tutorial.