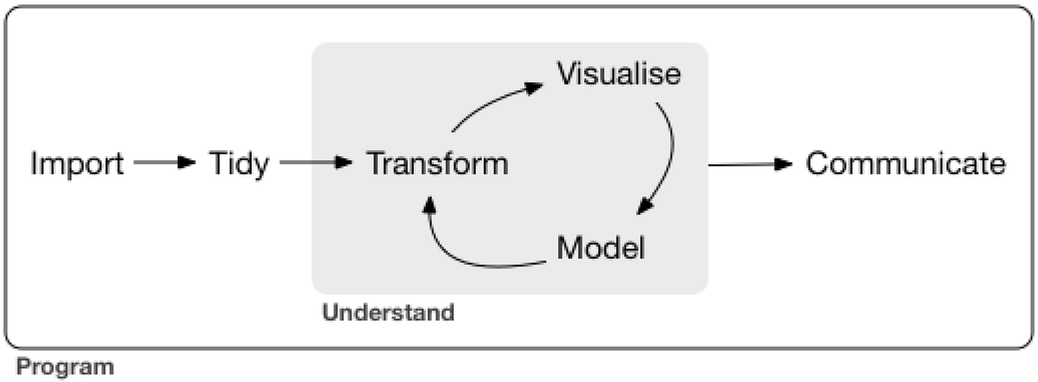
Exploring data

Into the tidyverse

June 6th, 2023

Data Science workflow

According to Hadley Wickham in R for Data Science:



First two weeks: data wrangling and visualization

Aspects of data **wrangling**:

- **import**: reading in data (e.g. `read_csv()`)
- **tidy**: rows = observations, columns = variables (i.e. **tabular** data)
- **transform**: filter observations, create new variables, summarize, etc.

What is Exploratory Data Analysis (EDA)?

(broadly speaking) EDA = questions about data + wrangling + visualization

R for Data Science: *"EDA is a state of mind"*, an iterative cycle:

- generate questions
- answer via transformations and visualizations

Example of questions?

- What type of **variation** do the variables display?
- What type of **relationships** exist between variables?

EDA is **NOT** a replacement for statistical inference and learning

EDA is an **important** and **necessary** step to build intuition

Now for an example...

Exploring MLB batting statistics

Import `Batting` table of historical MLB statistics from the `Lahman` package, explore using the `tidyverse`

```
library(tidyverse) # Load the tidyverse suite of packages
library(Lahman) # Load the Lahman package to access its datasets
Batting <- as_tibble(Batting) # Initialize the Batting dataset
```

Basic info about the `Batting` dataset:

```
dim(Batting) # displays same info as c(nrow(Batting), ncol(Batting))
```

```
## [1] 112184     22
```

```
class(Batting)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

`tbl` (pronounced `tibble`) is the `tidyverse` way of storing tabular data, like a spreadsheet or `data.frame`

Always look at your data: view the first 6 (by default) rows with `head()`

```
head(Batting) # Try just typing Batting into your console, what happens
```

```
## # A tibble: 6 × 22
##   playerID  yearID stint teamID lgID      G    AB     R     H   X2B   X3B
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <
## 1 abercda01   1871     1 TRO    NA        1     4     0     0     0     0
## 2 addybo01    1871     1 RC1    NA       25   118    30    32     6     0
## 3 allisar01   1871     1 CL1    NA       29   137    28    40     4     5
## 4 allisdo01   1871     1 WS3    NA       27   133    28    44    10     2
## 5 ansonca01   1871     1 RC1    NA       25   120    29    39    11     3
## 6 armstbo01   1871     1 FW1    NA       12    49     9    11     2     1
## # … with 10 more variables: RBI <int>, SB <int>, CS <int>, BB <int>, SO <in
## #   IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Is our `Batting` dataset **tidy**?

- Each row = a player's season stint with a team (i.e. players can play for multiple teams in year)

- Each column = different measurement or recording about the player-team-season observation (can print out column names directly with `colnames(Batting)` or `names(Batting)`)

Can we explore how baseball has changed over time with `Batting`?

Let the data wrangling begin...

Summarize *continuous* (e.g. `yearID`, `AB`) and *categorical* (e.g. `teamID`, `lgID`) variables in different ways

Compute **summary statistics** for *continuous* variables with the `summary()` function:

```
summary(Batting$yearID)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    1871    1938    1978    1969    2003    2022 
```

Compute **counts** of *categorical* variables with `table()` function:

```
table("Leagues" = Batting$lgID) # be careful it ignores NA values!
```

```
## Leagues
##    AA    AL    FL    NA    NL    PL    UA 
##  1893 51799   472   737 56800   149   334 
```

How do we remove the other leagues?

`dplyr` is a package within the `tidyverse` with functions for data wrangling

"Grammar of data manipulation": `dplyr` functions are **verbs**, datasets are **nouns**

- **We can `filter()` our dataset to choose observations meeting conditions**

```
mlb_batting <- filter(Batting, lgID %in% c("AL", "NL"))
nrow(Batting) - nrow(mlb_batting) # Difference in rows
```

```
## [1] 3585
```

- **We can `select()` variables of interest**

```
sel_batting <- select(Batting, yearID, lgID, G, AB, R, H, HR, BB, SO)
head(sel_batting, n = 3)
```

```
## # A tibble: 3 × 9
##   yearID lgID      G    AB     R     H    HR    BB    SO
##    <int> <fct> <int> <int> <int> <int> <int> <int> <int>
## 1   1871 NA        1     4     0     0     0     0     0
## 2   1871 NA       25   118    30    32     0     4     0
## 3   1871 NA       29   137    28    40     0     2     5
```

- **We can `arrange()` our dataset to sort observations by variables**

```
hr_batting <- arrange(Batting, desc(HR)) # use desc() for descending orc
head(hr_batting, n = 3)
```

```
## # A tibble: 3 × 22
##   playerID  yearID stint teamID lgID      G    AB     R     H   X2B   X3B
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <
## 1 bondsba01   2001     1 SFN    NL      153   476   129   156    32     2
## 2 mcgwima01   1998     1 SLN    NL      155   509   130   152    21     0
## 3 sosasa01    1998     1 CHN    NL      159   643   134   198    20     0
## # … with 10 more variables: RBI <int>, SB <int>, CS <int>, BB <int>, SO <in
## #   IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>
```

- **We can `summarize()` our dataset to one row based on functions of variables**

```
summarize(Batting, max(stint), median(AB))
```

```
## # A tibble: 1 × 2
##   `max(stint)` `median(AB)`
##          <int>        <dbl>
## 1            5           45
```

- **We can `mutate()` our dataset to create new variables** (mutate is a weird name...)

```
new_batting <- mutate(Batting, batting_avg = H / AB, so_to_bb = SO / BB)
head(new_batting, n = 1)
```

```
## # A tibble: 1 × 24
##   playerID  yearID stint teamID lgID      G    AB     R     H   X2B   X3B
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <
## 1 abercda01   1871     1 TRO    NA        1     4     0     0     0     0
## # … with 12 more variables: RBI <int>, SB <int>, CS <int>, BB <int>, SO <in
## #   IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>, batting_avg <dbl>
## #   so_to_bb <dbl>
```

How do we perform several of these actions?

```
head(arrange(select(mutate(Batting, BA = H / AB), playerID, BA), desc(BA
```

```
## # A tibble: 1 × 2
##   playerID     BA
##   <chr>     <dbl>
## 1 snowch01      1
```

That's awfully annoying to do, and also difficult to read...

Enter the pipeline

The `%>%` (*pipe*) operator is used in the `tidyverse` (from `magrittr`) to chain commands together

`%>%` directs the **data analyis pipeline**: output of one function pipes into input of the next function

```
Batting %>%
  filter(lgID %in% c("AL", "NL"),
         AB > 300) %>%
  mutate(batting_avg = H / AB) %>%
  arrange(desc(batting_avg)) %>%
  select(playerID, yearID, batting_avg) %>%
  head(n = 5)
```

```
## # A tibble: 5 × 3
##   playerID  yearID batting_avg
##   <chr>      <int>       <dbl>
## 1 duffyhu01   1894       0.440
## 2 barnero01   1876       0.429
## 3 lajoina01   1901       0.426
## 4 keelewi01   1897       0.424
## 5 hornsro01   1924       0.424
```

More pipeline actions!

Instead of `head()`, **we can `slice()` our dataset to choose the observations based on the position**

```r
Batting %>%
  filter(lgID %in% c("AL", "NL"),
         AB > 300) %>%
  mutate(so_to_bb = SO / BB) %>%
  arrange(so_to_bb) %>%
  select(playerID, yearID, so_to_bb) %>%
  slice(c(1, 2, 10, 100))
```

```
## # A tibble: 4 × 3
##   playerID  yearID so_to_bb
##   <chr>      <int>    <dbl>
## 1 roweja01    1882   0
## 2 seweljo01   1932   0.0536
## 3 holloch01   1922   0.0862
## 4 collied01   1918   0.178
```

Grouped operations

We group_by() to split our dataset into groups based on a variable's values

```
Batting %>%
  filter(lgID %in% c("AL", "NL")) %>%
  group_by(yearID) %>%
  summarize(hr = sum(HR), so = sum(SO), bb = sum(BB)) %>%
  arrange(desc(hr)) %>%
  slice(1:5)
```

```
## # A tibble: 5 × 4
##   yearID    hr    so    bb
##    <int> <int> <int> <int>
## 1   2019  6776 42823 15895
## 2   2017  6105 40104 15829
## 3   2021  5944 42145 15794
## 4   2000  5693 31356 18237
## 5   2016  5610 38982 15088
```

`group_by()` is only useful in a pipeline (e.g. with `summarize()`), and pay attention to its behavior

`ungroup()` can solve your problems afterwards

Putting it all together...

We'll create a **tidy** dataset where each row = a year with the following variables:

- total HRs (homeruns), SOs (strikeouts), and BBs (walks)
- year's BA = total H / total AB
- only want AL and NL leagues

```r
year_batting_summary <- Batting %>%
  filter(lgID %in% c("AL", "NL")) %>%
  group_by(yearID) %>%
  summarize(total_hits = sum(H, na.rm = TRUE),
            total_hrs = sum(HR, na.rm = TRUE),
            total_sos = sum(SO, na.rm = TRUE),
            total_walks = sum(BB, na.rm = TRUE),
            total_atbats = sum(AB, na.rm = TRUE)) %>%
  mutate(batting_avg = total_hits / total_atbats)
head(year_batting_summary, n = 2)
```

```
## # A tibble: 2 × 7
##   yearID total_hits total_hrs total_sos total_walks total_atbats batting_av
##    <int>      <int>     <int>     <int>       <int>        <int>       <dbl
## 1   1876       5338        40       589         336        20121       0.26
## 2   1877       3705        24       726         345        13667       0.27
```

Top three years with the most HRs?

```
year_batting_summary %>%
  arrange(desc(total_hrs)) %>%
  slice(1:3)
```

```
## # A tibble: 3 × 7
##   yearID total_hits total_hrs total_sos total_walks total_atbats batting_av
##    <int>      <int>     <int>     <int>       <int>        <int>      <dbl
## 1   2019      42039      6776     42823       15895       166651      0.25
## 2   2017      42215      6105     40104       15829       165567      0.25
## 3   2021      39484      5944     42145       15794       161941      0.24
```

Top three years with highest batting average?

```
year_batting_summary %>%
  arrange(desc(batting_avg)) %>%
  slice(1:3)
```

```
## # A tibble: 3 × 7
##   yearID total_hits total_hrs total_sos total_walks total_atbats batting_av
##    <int>      <int>     <int>     <int>       <int>        <int>      <dbl
## 1   1894      17809       629      3333        5870        57577      0.30
## 2   1895      16827       488      3621        5120        56788      0.29
## 3   1930      25597      1565      7934        7654        86571      0.29
```

Best and worst strikeout to walk ratios?

```
year_batting_summary %>%
  mutate(so_to_bb = total_sos / total_walks) %>%
  arrange(so_to_bb) %>%
  slice(c(1, n()))
```

```
## # A tibble: 2 × 8
##   yearID total_hits total_hrs total_sos total_walks total_atbats batti…¹ so_
##    <int>      <int>     <int>     <int>       <int>        <int>   <dbl>
## 1   1893      15913       460      3341        6143        56898   0.280   (
## 2   1879       6171        58      1843         508        24155   0.255   3
## # … with abbreviated variable names ¹batting_avg, ²so_to_bb
```

We can make better looking tables... **rename() variables in our dataset**

```
year_batting_summary %>%
  select(yearID, batting_avg) %>%
  rename(Year = yearID, `Batting AVG` = batting_avg) %>%
  slice(c(1, n()))
```

```
## # A tibble: 2 × 2
##    Year `Batting AVG`
##   <int>         <dbl>
## 1  1876         0.265
## 2  2022         0.243
```

Grammar of tables preview

We can go one step further - **and use the new gt package** to create a nice-looking table for presentation

```r
library(gt)
year_batting_summary %>%
  select(yearID, batting_avg) %>%
  rename(Year = yearID,
         `Batting AVG` = batting_
  arrange(desc(`Batting AVG`)) %>%
  slice(c(1:3, (n()-2):n())) %>%
  gt() %>%
  tab_header(
    title = "Best / worst MLB Sea
    subtitle = "Top / bottom thre
  )
```

```
##
## Attaching package: 'gt'
## The following object is masked from '
##
##     google_font
```

Best / worst MLB Seasons by AVG

Top / bottom three are presented

Year	Batting AVG
1894	0.3093075
1895	0.2963126
1930	0.2956764
1908	0.2389593
1888	0.2387601

Data visualization

"The simple graph has brought more information to the data analyst's mind than any other device." — Tukey

- **TOMORROW**: the **grammar of graphics**
- Use `ggplot2` to visually explore our data
- More intuitive than base `R` plotting!
- Will walkthrough different types of visualizations for 1D, 2D, continuous, categorical, facetting, etc.
- `tidyverse` verbs and `%>%` leads to natural pipeline for EDA