Ivey BA R Workshop

Introduction to the Tidyverse

Hayden MacDonald

Ivey

Goals

- Learn a different way to manipulate, explore, and visualize data in R
- Apply these new methods to familiar data in a workshop analysis
- Achieve greater productivity and confidence in your R skills
- Leave the workshop with resources for your continued learning

Rules of Engagement

- This is an open workshop - Ask questions!
- Ask me to slow down or clarify when needed.
- Check in with your neighbours to help each other.

Why R when I can use Excel?

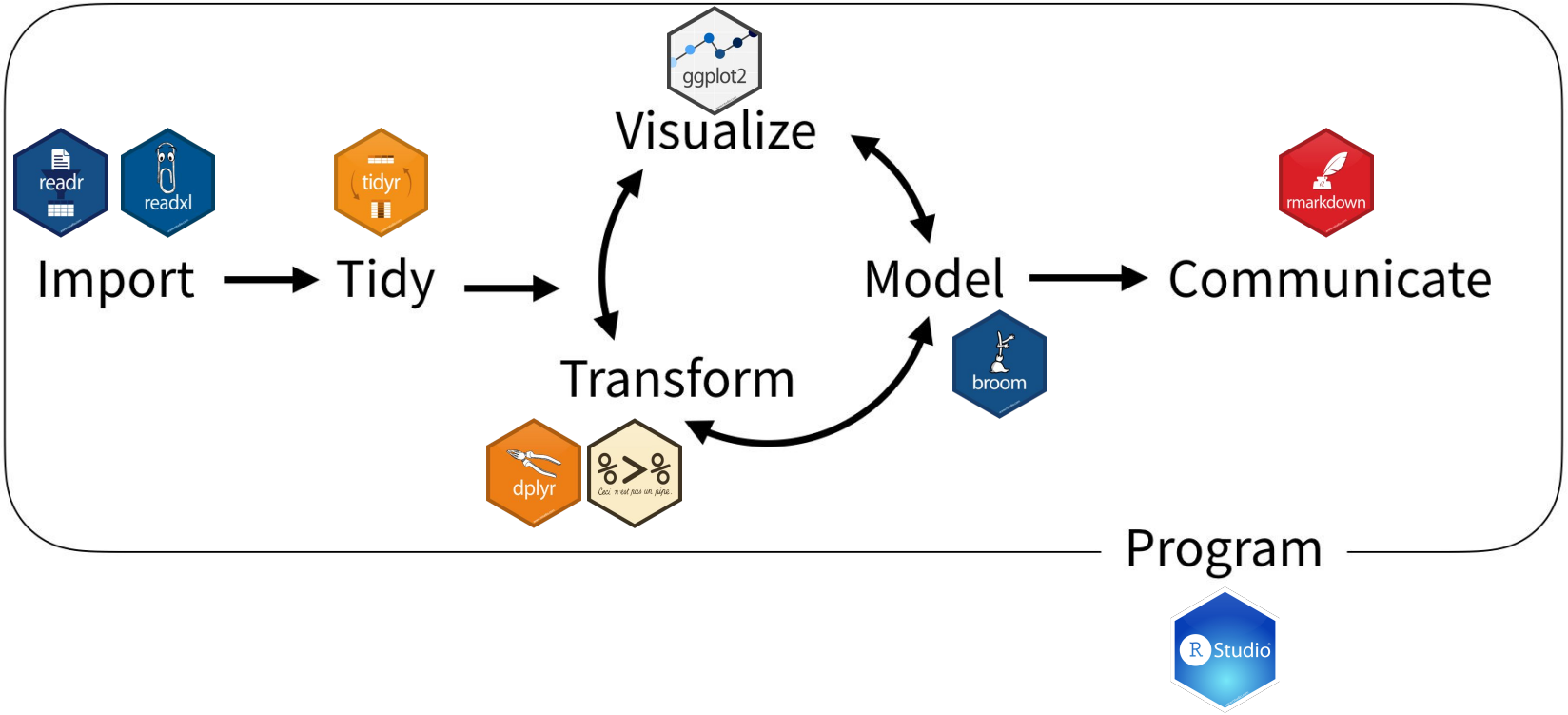
- Reproducible analyses
- More powerful data manipulation capabilities
- It reads any type of data
- It supports larger data sets
- Faster computation
- Supports advanced statistical methods
- Pretty graphs (ggplot2)
- It's free!!!!!!

Why the Tidyverse?

- Highly integrated set of packages
- Designed for analytics and data science
- Emphasis on code readability
- Facilitates an intuitive data analysis workflow

Learning dplyr helps you learn SQL!!

Data Analysis Workflow



Quick power example of the Tidyverse

Tibble

- A special class of data frame for the tidyverse
- Works with base R and tidyverse functions

Tibble

Advantages over data.frame:

- Does not automatically change character vectors into factors
    - Never worry about `stringsAsFactors = FALSE` again
- It never changes your variable names (unlike base R)
- Prints a helpful preview of your data
- Facilitates sequential data analysis operations

Tibbles & Tidy Data

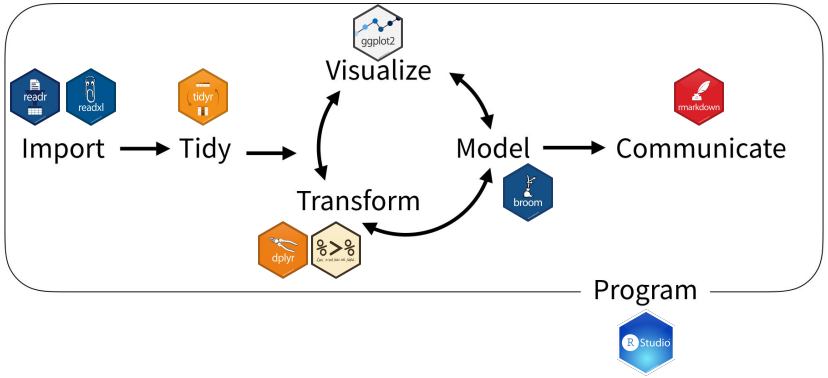
Variable

Observation

```
pain

## # A tibble: 72 x 17
##    STOR     EARN     K  SIZE EMPL  total   P15   P25   P35   P45   P55
##    <chr>   <dbl> <dbl> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 1        28.3   861.   129 14     8580   980  1280   560  1000  3100
##  2 2        -1.46  630.    91 12     8460  1290   720  1200  1490  3100
##  3 3        68.9  1074.   140 13    19250  2940  2490  3710  4030  5270
##  4 4       202.    882.   184 7     20920  3570  4930  4420  4300  2960
##  5 5       116.    931.   144 14    11660  1700  1140  2200  2140  2630
##  6 6       222.   1185.   160 11    25780  4640  3150  5720  5330  5920
##  7 7       293.    907.    94 5     19000  3600  2330  4750  4970  3030
##  8 8       134.    764.   100 8     18500  3450  2560  3630  3520  4800
##  9 9        37.4   643.    85 14    14210  1930  4280  1740  2060  2960
## 10 10      181.    666.    92 6     17440  3520  1780  4350  4020  3470
## # ... with 62 more rows, and 6 more variables: INC <dbl>, COMP <dbl>,
## #   NCOMP <dbl>, NREST <dbl>, PRICE <dbl>, CLI <dbl>
```

Value

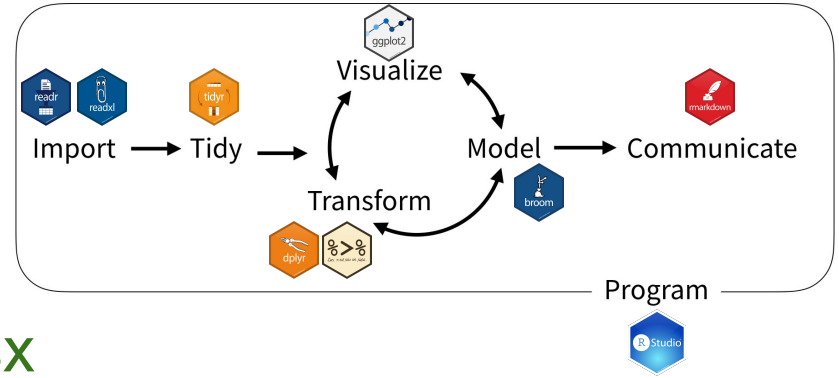
Workshop Analysis




```
install.packages("tidyverse")

library(tidyverse)


install.packages("readxl")

library(readxl)
```

Access CroqPainData_Feb14.xlsx

Workshop Analysis



Access CroqPainData_Feb14.xlsx

Goal

- Predict whether a restaurant will hit the target performance ratio of 0.26
- Logistic regression of “Target”

Import

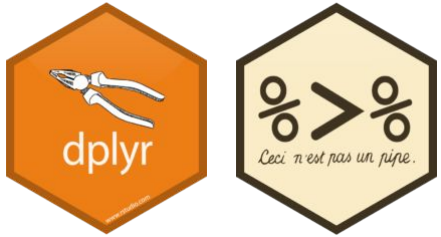

```
pain <- read_xlsx("CroqPainData_Feb14.xlsx",
                  sheet = 1,
                  range = "A1:Q73")
```

Import


```
pain
```

```
## # A tibble: 72 x 17
##    STOR     EARN     K  SIZE EMPL  total   P15   P25   P35   P45   P55
##    <chr>   <dbl> <dbl> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 1        28.3   861.   129 14     8580   980  1280   560  1000  3100
##  2 2        -1.46  630.    91 12     8460  1290   720  1200  1490  3100
##  3 3        68.9  1074.   140 13    19250  2940  2490  3710  4030  5270
##  4 4       202.    882.   184 7     20920  3570  4930  4420  4300  2960
##  5 5       116.    931.   144 14    11660  1700  1140  2200  2140  2630
##  6 6       222.   1185.   160 11    25780  4640  3150  5720  5330  5920
##  7 7       293.    907.    94 5     19000  3600  2330  4750  4970  3030
##  8 8       134.    764.   100 8     18500  3450  2560  3630  3520  4800
##  9 9        37.4   643.    85 14    14210  1930  4280  1740  2060  2960
## 10 10      181.    666.    92 6     17440  3520  1780  4350  4020  3470
## #  ... with 62 more rows, and 6 more variables: INC <dbl>, COMP <dbl>,
## #   NCOMP <dbl>, NREST <dbl>, PRICE <dbl>, CLI <dbl>
```

Clean the data

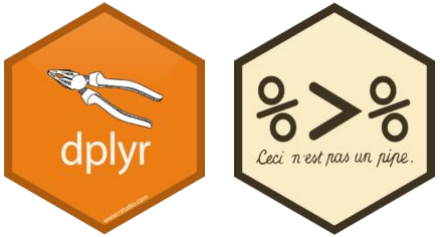


- Remove extra rows
- Change character variables to numeric variables
- Change names in STOR to numeric values
- Impute missing values
- Create new variables: PR and Target

```
pain
```

```
## # A tibble: 72 x 17
##    STOR     EARN     K  SIZE EMPL  total   P15   P25   P35   P45   P55
##    <chr>   <dbl> <dbl> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 1        28.3   861.   129 14     8580   980  1280   560  1000  3100
##  2 2        -1.46  630.    91 12     8460  1290   720  1200  1490  3100
##  3 3        68.9  1074.   140 13    19250  2940  2490  3710  4030  5270
##  4 4       202.    882.   184 7     20920  3570  4930  4420  4300  2960
##  5 5       116.    931.   144 14    11660  1700  1140  2200  2140  2630
##  6 6       222.   1185.   160 11    25780  4640  3150  5720  5330  5920
##  7 7       293.    907.    94 5     19000  3600  2330  4750  4970  3030
##  8 8       134.    764.   100 8     18500  3450  2560  3630  3520  4800
##  9 9        37.4   643.    85 14    14210  1930  4280  1740  2060  2960
## 10 10      181.    666.    92 6     17440  3520  1780  4350  4020  3470
## # ... with 62 more rows, and 6 more variables: INC <dbl>, COMP <dbl>,
## #   NCOMP <dbl>, NREST <dbl>, PRICE <dbl>, CLI <dbl>
```

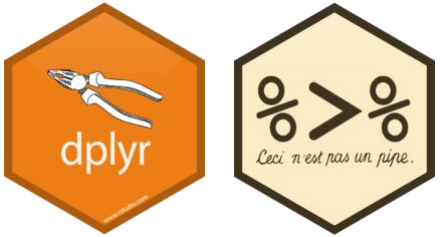
Transform



Pipe Operator %>% - “Take whatever is on the left
and make it the first argument on the right”

`f(x)` is the same as `x %>% f()`

Transform



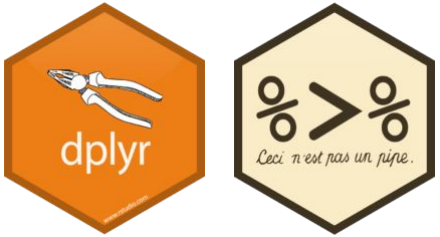
Use the pipe to calculate the mean of a vector

```
data %>%
  mean(na.rm = TRUE)
```

You can read the above code as:

```
Take data, then
  calculate the mean without NAs
```

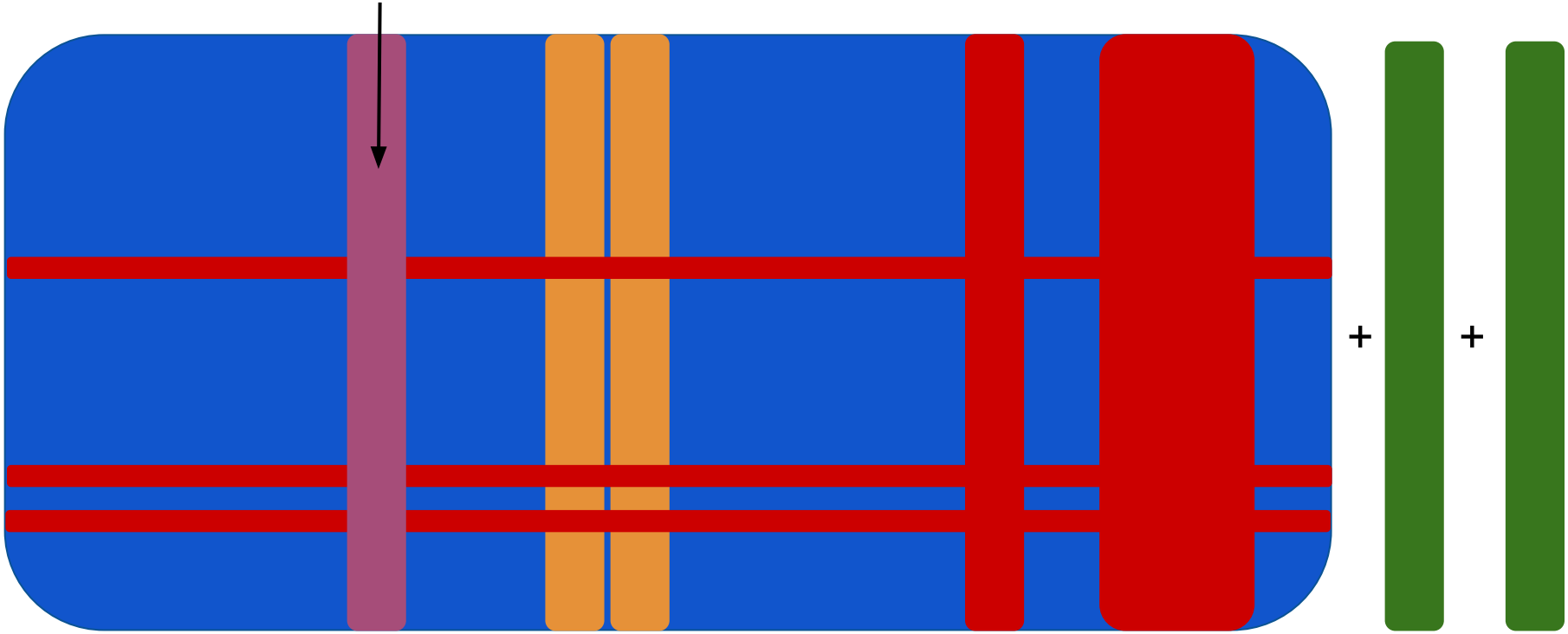
Transform



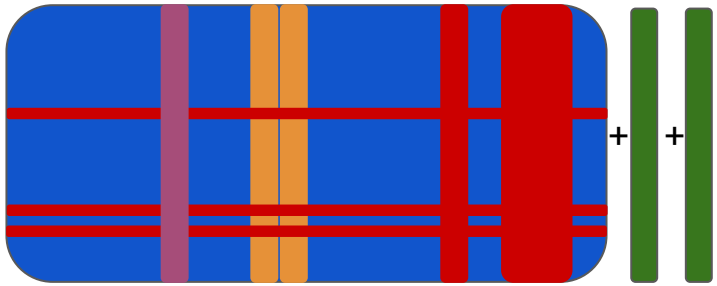
The pipe operator does several awesome things:

- Avoids nested functions; improves readability
- Chains sequences of data operations
- Prints an output of your data after those operations are executed
- Minimizes the number of versions of your data

Transform



Transform

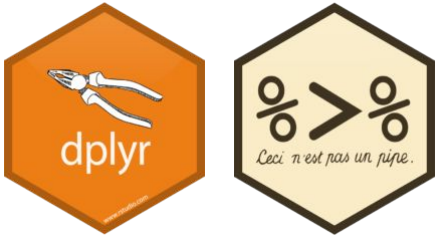


- Change existing variables
- Remove unwanted variables
- Remove specific rows
- Create new variables
- Rearrange the data frame

```
data %>%
    mutate(...) %>%
    select(...) %>%
    filter(...) %>%
    mutate(...) %>%
    arrange(...)
```

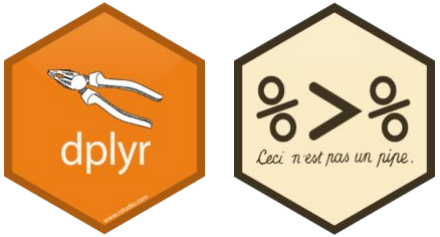
```
data2 <- data %>%
            mutate(...) %>%
            select(...) %>%
            filter(...) %>%
            mutate(...) %>%
            arrange(...)
```

Back to Croq'Pain - Transform



- Remove extra rows #51 & #62
- Change character variables to numeric variables
- Change names in STOR to numeric values
- Impute missing values
- Create new variables: Performance Ratio (PR) and Target

Back to Croq'Pain - Transform

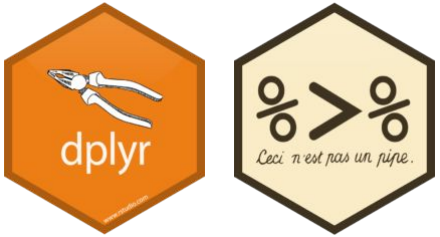


- Remove extra rows #51 & #62

```
pain %>%
  filter(rownames(pain) != c(51,62))
```

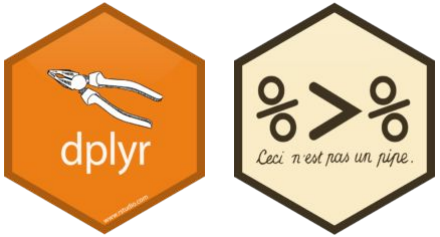
```
Take data, then
  filter rows where row names do NOT equal 51 or 62
```

Transform



- Remove extra rows #51 & #62
- Change character variables to numeric variables
- Change names in STOR to numeric values
- Impute missing values
- Create new variables: Performance Ratio (PR) and Target

Transform

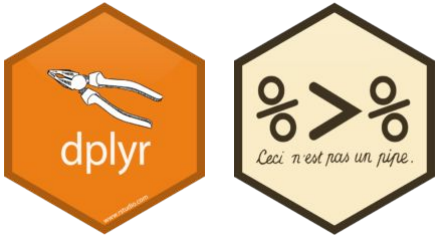


- Change character variables to numeric variables

```
pain %>%
  filter(rownames(pain) != c(51,62)) %>%
  mutate(STOR = as.numeric(STOR),
         EMPL = as.numeric(EMPL))
```

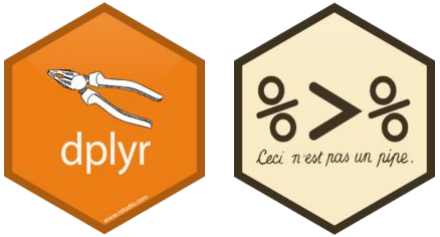
Take data, then
  filter rows where rownames do NOT equal 51 or 62, then
  change STOR and EMPL to numeric variables

Transform



- Remove extra rows #51 & #62
- Change character variables to numeric variables
- Change names in STOR to numeric values
- Impute missing values
- Create new variables: Performance Ratio (PR) and Target

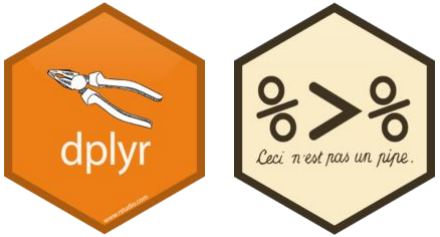
Transform



- Change names in STOR to numeric values

```
pain <- pain %>%
  filter(rownames(pain) != c(51,62)) %>%
  mutate(STOR = as.numeric(STOR),
         EMPL = as.numeric(EMPL)) %>%
  mutate(STOR = seq(1, 70, by = 1))
```

Transform



- Remove extra rows #51 & #62
- Change character variables to numeric variables
- Change names in STOR to numeric values
- Impute missing values
- Create new variables: Performance Ratio (PR) and Target

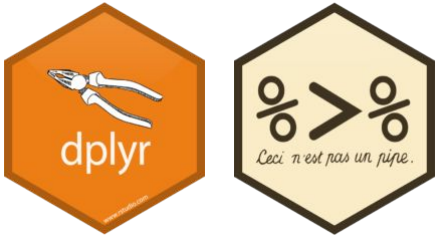
Transform

- Impute missing values

```
install.packages("naniar")

library(naniar)
```

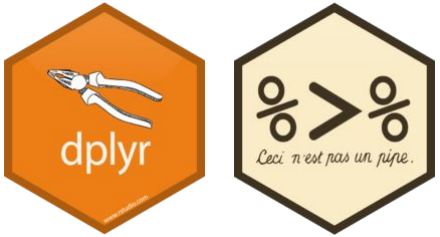
```
pain <- pain %>%
          impute_mean_all()
```

Transform



- Remove extra rows #51 & #62
- Change character variables to numeric variables
- Change names in STOR to numeric values
- Impute missing values
- Create new variables: Performance Ratio (PR) and Target

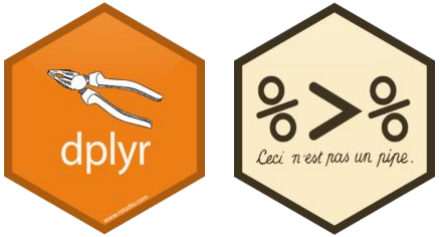
Transform



- Create new variables: Performance Ratio (PR) and Target

```
pain <- pain %>%
  mutate(PR = EARN / K,
         Target = case_when(PR >= 0.26 ~ 1, PR < 0.26 ~ 0))
```

Transform

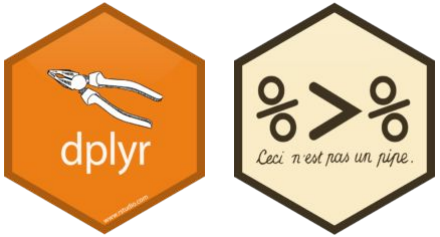


- Remove extra rows #51 & #62 ✔
- Change character variables to numeric variables ✔
- Change names in STOR to numeric values ✔
- Impute missing values ✔
- Create new variables: Performance Ratio (PR) and Target ✔

Visualize correlations


```
install.packages("GGally")

library(GGally)

ggcorr(pain, label = TRUE, hjust = 1)
```

Model




```
pain_train <- pain %>%
                filter(STOR <= 60)

pain_test <- pain %>%
                filter(STOR > 60)
```

Model

Quick trick with stringr!

```
str_c(names(pain_train), collapse = " + ")
```

Collects all variable names with a plus sign in between each.

Saves a lot of typing when you have many variables in your model.

Model


```
mod <- glm(Target ~ ..., family = binomial, data = pain_train)
```

```
summary(mod)
```

`summary(mod)` is gross because it has too much unorganized information.

If only we could make this information… tidy...

Model


```
mod <- glm(Target ~ ..., family = binomial, data = pain_train)

tidy(mod) %>%
  arrange(desc(p.value))
```

```
## # A tibble: 11 x 5
##    term         estimate std.error statistic p.value
##    <chr>           <dbl>     <dbl>     <dbl>   <dbl>
##  1 COMP         -0.0395     0.242      -0.163  0.870
##  2 SIZE         -0.00388    0.0115     -0.337  0.736
##  3 P25           0.000192   0.000444    0.431  0.666
##  4 (Intercept) -6.14        8.52       -0.721  0.471
##  5 NCOMP         0.129      0.169       0.764  0.445
##  6 P55          -0.000449   0.000574   -0.782  0.434
##  7 CLI          -0.0614     0.0670     -0.915  0.360
##  8 EMPL         -0.283      0.185      -1.53   0.126
##  9 INC           0.425      0.229       1.85   0.0640
## 10 PRICE        -0.344      0.165      -2.08   0.0379
## 11 P35           0.00200    0.000812    2.46   0.0139
```

Remove the first variable every time!

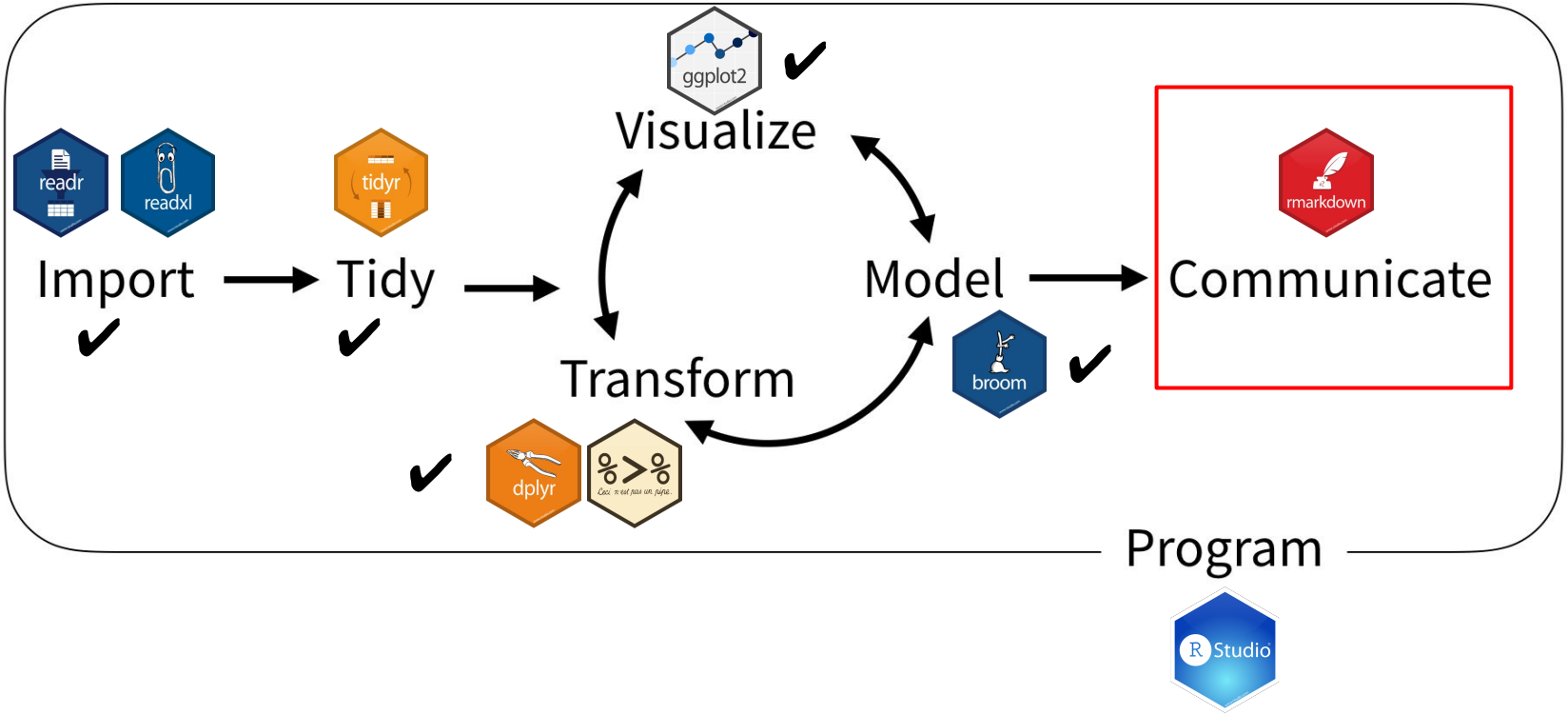
Model


```
mod <- glm(Target ~ ..., family = binomial, data = pain_train)
```

```
glance(mod)
```

```
## # A tibble: 1 x 7
##   null.deviance df.null logLik   AIC   BIC deviance df.residual
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int>
## 1          65.2      59  -18.3  42.6  48.9     36.6          57
```

Data Analysis Workflow



Communicate

- R Markdown puts code, graphs, and text all in one document
- Render .Rmd files into HTML, Word, or PDF documents
- Use Markdown formatting for regular text
- Use code chunks to insert data, statistics, and graphs

Communicate


```
install.packages("tinytex")

library(tinytex)

install_tinytex()
```

Let’s render BAWorkshop.Rmd into a PDF

Shameless plug

- My package: **`hmdrmd`**
- A set of rmarkdown templates
  - Tidy analysis
  - Case study analysis

hmdrmd

GitHub repository

```
install.packages("devtools")

devtools::install_github("HaydenMacDonald/hmdrmd")
```

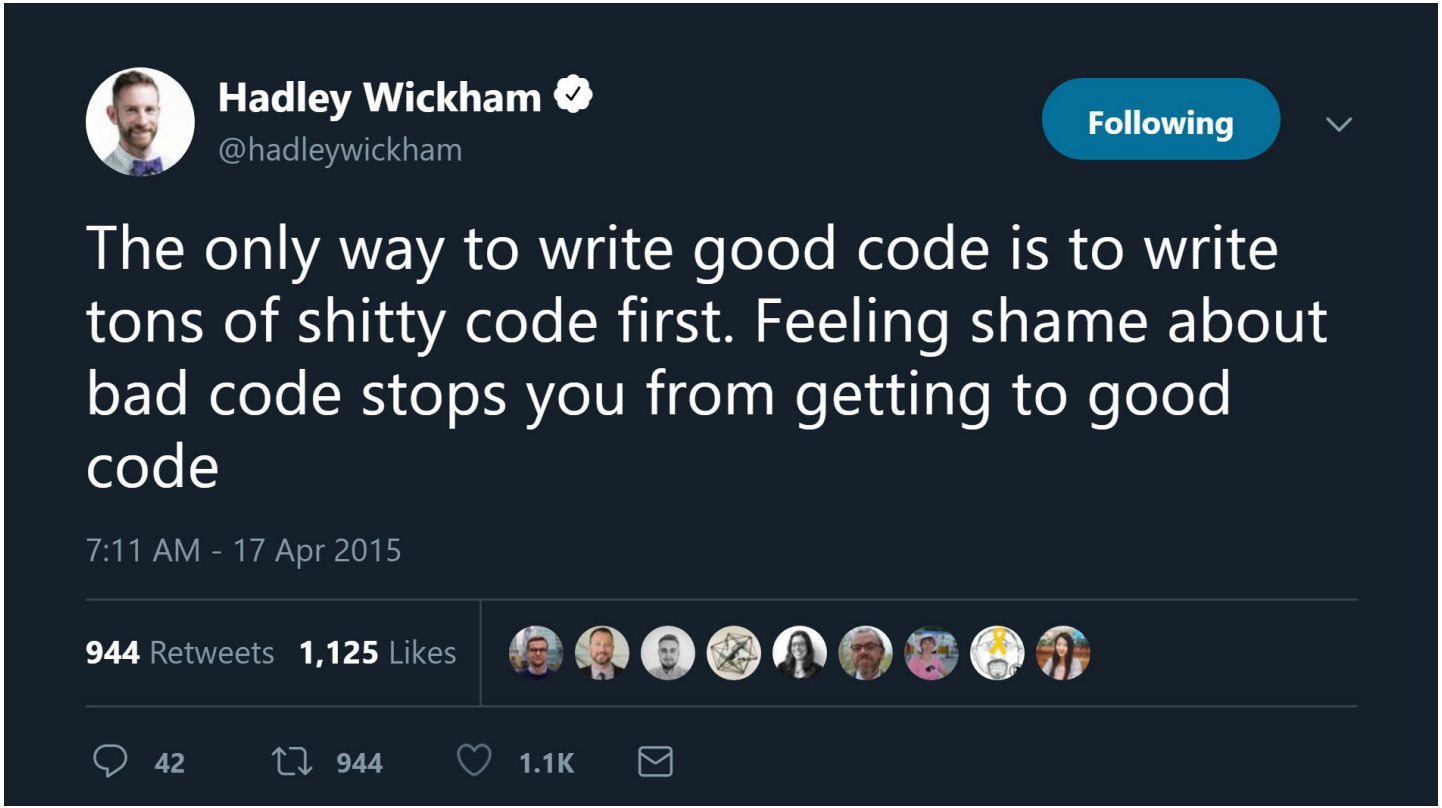
Resources

- RStudio Cheatsheets
    - https://github.com/rstudio/cheatsheets/tree/54f418c245ba22ee2f65ff2b760c77650c08888e
- Markdown tutorial
    - https://www.markdowntutorial.com/

Advice

- Learn to Google every R question / error you have
- Give yourself time.
    - Rest in between coding sessions
- Try breaking your code to understand how and why it works

Advice


Hadley Wickham
@hadleywickham
Following
The only way to write good code is to write tons of shitty code first. Feeling shame about bad code stops you from getting to good code
7:11 AM - 17 Apr 2015
944 Retweets 1,125 Likes
42
944
1.1K

Extras

- dplyr syntax into ggplot2
- Standardized residual plots using broom and ggplot2