Exploratory Data Analysis with R

Data Manipulation Using Dplyr

Xuemao Zhang
East Stroudsburg University

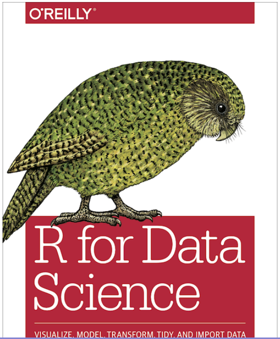
September 30, 2022

Outline

- Overview
- Filter observations (rows)
  - Removing missing values
- Select/drop variables (columns)
- The %>% operator
- Add/remove variables
- Rename variables
- Sort data
- Remove duplicate rows

Overview

- `dplyr`, part of Tidyverse is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges.
- The dplyr package is one of the most powerful and popular package in R.
- The author, Hadley Wickham, is also the author of the package ‘ggplot2.
- R for Data Science (O’Reilly 2017) by Hadley Wickham
- Free Online: https://r4ds.had.co.nz/



Overview

- Data Exploration = Data manipulation + Data visualization.
- **Data wrangling** (a term used in data science) is the process of transforming/mapping data from raw format into ready-to-analyze format.
- Besides `ggplot2()` for data visualization, Hadley Wickham created a series of R packages for data wrangling, including
  - `tidyr` for reshaping your data for plotting and use by different R functions
  - `tibble` for better ways to create, print and subset data frames
  - `dplyr` for data manipulation will be covered in this course
- The `dplyr` package also interfaces well with tibbles.
- `dplyr` functions process **faster** than base R functions. It is because `dplyr` functions were written in a computationally efficient manner. They are also more stable in the syntax and better supports data frames than vectors.

Overview

- There are 8 fundamental data manipulation verbs in `dplyr` that you will use to do most of your data manipulations.
  - `mutate()` and `transmutate()`: Add/create new variables.
  - `select()`: Select columns (variables) by their names.
  - `filter()`: Pick rows (observations/samples) based on their values.
  - `distinct()`: Remove duplicate rows.
  - `arrange()`: Reorder the rows.
  - `rename()`: Rename columns.
  - `summarise()`: Compute statistical summaries (e.g., computing the mean or the sum) and thus reduce multiple values down to a single summary. It is similar to `R::base::aggregate`. - `group_by()` to group variables for summarise

Recall. The double colon `::` is used to specify the package where a function is from: `packagename::functionname()`.

Overview

- All these functions work similarly as follow:
  - The first argument is a data frame
  - The subsequent arguments are comma separated list of unquoted variable names and the specification of what you want to do
  - The result is a new data frame
- A special feature in `dplyr` is that you can chain your data manipulation operations using the pipe operator (%>%).

Note. `dplyr package allows to use the forward-pipe chaining operator (%>%) for combining multiple operations. For example, x %>% f is equivalent to f(x). Using the pipe (%>%), the output of each operation is passed to the next operation. This makes R programming easy:`

```
# input      +--------+      +--------+      +--------+      result
# data  %>% |  verb  | %>% |  verb  | %>% |  verb  | ->   data
# frame      +--------+      +--------+      +--------+      frame
```

Creating a `data.frame` to work with

Again, here we use one of the datasets that comes with R called `mtcars` create a toy data.frame named `df`

```
data(mtcars);
dfm = mtcars; # to save original
```

No rownames in tibbles!

In the “tidy” data format, all information of interest is a variable (not a name). **as of tibble 2.0, rownames are removed**. For example, mtcars has each car name as a row name:

```
head(dfm, 2)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4       21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag   21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
library(tibble)
head(as_tibble(dfm), 2)
```

```
## # A tibble: 2 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear    ca
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
## 1    21     6   160   110   3.9  2.62  16.5     0     1     4
## 2    21     6   160   110   3.9  2.88  17.0     0     1     4
```

No rownames in tibbles!

If you run into this, use `rownames_to_column` to add it before turning it into a `tibble` to keep them:

```
dfm = rownames_to_column(dfm, var = "car");
dfm = as_tibble(dfm);
dfm;
```

```
## # A tibble: 32 x 12
##    car              mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <c
##  1 Mazda RX4       21       6  160    110  3.9   2.62  16.5     0
##  2 Mazda RX4 ~     21       6  160    110  3.9   2.88  17.0     0
##  3 Datsun 710      22.8     4  108     93  3.85  2.32  18.6     1
##  4 Hornet 4 D~     21.4     6  258    110  3.08  3.22  19.4     1
##  5 Hornet Spo~     18.7     8  360    175  3.15  3.44  17.0     0
##  6 Valiant         18.1     6  225    105  2.76  3.46  20.2     1
##  7 Duster 360      14.3     8  360    245  3.21  3.57  15.8     0
##  8 Merc 240D       24.4     4  147.    62  3.69  3.19  20       1
##  9 Merc 230        22.8     4  141.    95  3.92  3.15  22.9     1
## 10 Merc 280        19.2     6  168.   123  3.92  3.44  18.3     1
## # ... with 22 more rows
```

No rownames in tibbles!

- The function dplyr::glimpse is a little like str() applied to a data frame but it tries to show you as much data as possible.

```
dplyr::glimpse(dfm);
```

```
## Rows: 32
## Columns: 12
## $ car  <chr> "Mazda RX4", "Mazda RX4 Wag", "Datsun 710", "Hornet
## $ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8
## $ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4
## $ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146
## $ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123,
## $ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92
## $ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.1
## $ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.
## $ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1
## $ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
## $ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4
## $ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1
```

Filter observations

- The function `filter()` is used to subset data with matching logical conditions.
- It is Similar to base::`which()` or `subset()`

```
str(dfm);
```

```
## tibble [32 x 12] (S3: tbl_df/tbl/data.frame)
##  $ car : chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hor
##  $ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2
##  $ cyl : num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num [1:32] 160 160 108 258 360 ...
##  $ hp  : num [1:32] 110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num [1:32] 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.
##  $ wt  : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num [1:32] 16.5 17 18.6 19.4 17 ...
##  $ vs  : num [1:32] 0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num [1:32] 4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
```

Filter observations

- The data set is data frame with 32 observations on 11 (numeric) variables.
  - [, 1] mpg Miles/(US) gallon
  - [, 2] cyl Number of cylinders
  - [, 3] disp Displacement (cu.in.)
  - [, 4] hp Gross horsepower
  - [, 5] drat Rear axle ratio
  - [, 6] wt Weight (1000 lbs)
  - [, 7] qsec 1/4 mile time
  - [, 8] vs Engine (0 = V-shaped, 1 = straight)
  - [, 9] am Transmission (0 = automatic, 1 = manual)
  - [,10] gear Number of forward gears
  - [,11] carb Number of carburetors

Filter observations

- base::unique() returns unique values of a variable/data frame.

```
unique(dfm$cyl);
```

```
## [1] 6 4 8
```

```
unique(dfm$disp);
```

```
##  [1] 160.0 108.0 258.0 360.0 225.0 146.7 140.8 167.6 275.8 472.0
## [13]  78.7  75.7  71.1 120.1 318.0 304.0 350.0 400.0  79.0 120.3
## [25] 145.0 301.0 121.0
```

```
unique(dfm$gear);
```

```
## [1] 4 3 5
```

Filter observations

- The command in `dplyr` for subsetting rows is `filter`. Try `?filter`
- Note, no $ or subsetting is necessary. R "knows" `mpg` refers to a column of `dfm`.

```
library(dplyr);
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
filter(dfm, mpg > 20);
```

```
## # A tibble: 14 x 12
##    car              mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <
```

Filter observations

- Multiple arguments can be provided to `filter()`.

```
filter(dfm, mpg > 20, cyl == 4);
```

```
## # A tibble: 11 x 12
##    car              mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <d
##  1 Datsun 710      22.8     4 108      93  3.85  2.32  18.6     1
##  2 Merc 240D       24.4     4 147.     62  3.69  3.19  20       1
##  3 Merc 230        22.8     4 141.     95  3.92  3.15  22.9     1
##  4 Fiat 128        32.4     4  78.7    66  4.08  2.2   19.5     1
##  5 Honda Civic     30.4     4  75.7    52  4.93  1.62  18.5     1
##  6 Toyota Cor~     33.9     4  71.1    65  4.22  1.84  19.9     1
##  7 Toyota Cor~     21.5     4 120.     97  3.7   2.46  20.0     1
##  8 Fiat X1-9       27.3     4  79      66  4.08  1.94  18.9     1
##  9 Porsche 91~     26       4 120.     91  4.43  2.14  16.7     0
## 10 Lotus Euro~     30.4     4  95.1   113  3.77  1.51  16.9     1
## 11 Volvo 142E      21.4     4 121     109  4.11  2.78  18.6     1
```

Filter observations

You can have multiple logical conditions using the following:

- & : AND
- | : OR

By default, you can separate conditions by commas, and `filter` assumes these statements are joined by &:

```
filter(dfm, mpg > 20 & cyl == 4);
```

```
## # A tibble: 11 x 12
##    car            mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <c
##  1 Datsun 710    22.8     4 108      93  3.85  2.32  18.6     1
##  2 Merc 240D     24.4     4 147.     62  3.69  3.19  20       1
##  3 Merc 230      22.8     4 141.     95  3.92  3.15  22.9     1
##  4 Fiat 128      32.4     4  78.7    66  4.08  2.2   19.5     1
##  5 Honda Civic   30.4     4  75.7    52  4.93  1.62  18.5     1
##  6 Toyota Cor~   33.9     4  71.1    65  4.22  1.84  19.9     1
##  7 Toyota Cor~   21.5     4 120.     97  3.7   2.46  20.0     1
##  8 Fiat X1-9     27.3     4  79      66  4.08  1.94  18.9     1
##  9 Porsche 91~   26       4 120.     91  4.43  2.14  16.7     0
```

Filter observations

If you want OR statements, you need to use | explicitly:

```
filter(dfm, mpg > 20 | cyl == 4);
```

```
## # A tibble: 14 x 12
##    car            mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <c
##  1 Mazda RX4     21       6 160     110  3.9   2.62  16.5     0
##  2 Mazda RX4 ~   21       6 160     110  3.9   2.88  17.0     0
##  3 Datsun 710    22.8     4 108      93  3.85  2.32  18.6     1
##  4 Hornet 4 D~   21.4     6 258     110  3.08  3.22  19.4     1
##  5 Merc 240D     24.4     4 147.     62  3.69  3.19  20       1
##  6 Merc 230      22.8     4 141.     95  3.92  3.15  22.9     1
##  7 Fiat 128      32.4     4  78.7    66  4.08  2.2   19.5     1
##  8 Honda Civic   30.4     4  75.7    52  4.93  1.62  18.5     1
##  9 Toyota Cor~   33.9     4  71.1    65  4.22  1.84  19.9     1
## 10 Toyota Cor~   21.5     4 120.     97  3.7   2.46  20.0     1
## 11 Fiat X1-9     27.3     4  79      66  4.08  1.94  18.9     1
## 12 Porsche 91~   26       4 120.     91  4.43  2.14  16.7     0
## 13 Lotus Euro~   30.4     4  95.1   113  3.77  1.51  16.9     1
## 14 Volvo 142E    21.4     4 121     109  4.11  2.78  18.6     1
```

Filter observations

- `filter()` is similar to `subset()`.
- `subset()` can select both variables (columns) and observations(rows).
- `filter()` works exclusively for observations(rows).
- Recall: Two useful comparison operators `%in%` and `is.na()`:
  - `x %in% c("a","b","c")` is used to check if x is one of the values in the right hand side.
  - `is.na()` is used to check if a value is missing.

Filter observations - Removing missing values

- like the function `na.omit()` and `questionr::na.rm()`, the function `tidyr::drop_na()` drop rows containing missing values.

```
exam =read.csv("../data/exam.csv",header=TRUE, sep=",");
any(is.na(exam));
```

```
## [1] TRUE
```

```
which(is.na(exam),arr.ind=TRUE);
```

```
##      row col
## [1,]   5   4
```

```
library(tidyr);
exam%>%drop_na();
```

```
##   ID   T1   T2   T3
## 1  1 21.5 16.5 23.5
## 2  2 21.0 16.0 20.0
## 3  3 30.5 31.0 30.0
## 4  4 23.0 18.5 25.5
## 5  6 27.5 27.0 35.5
```

Select/drop variables

Recall that we can grab the carb column using the $ operator.

```
dfm$carb;
```

```
##  [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6
```

Select/drop variables

- `select()` allows you to select specific **columns**(variables) from your data.

```
select(dfm, car, mpg);
```

```
## # A tibble: 32 x 2
##    car                 mpg
##    <chr>             <dbl>
##  1 Mazda RX4          21
##  2 Mazda RX4 Wag      21
##  3 Datsun 710         22.8
##  4 Hornet 4 Drive     21.4
##  5 Hornet Sportabout  18.7
##  6 Valiant            18.1
##  7 Duster 360         14.3
##  8 Merc 240D          24.4
##  9 Merc 230           22.8
## 10 Merc 280           19.2
## # ... with 22 more rows
```

Select/drop variables

The select command from dplyr allows you to subset columns of

```
select(dfm, car, mpg, cyl);
```

```
## # A tibble: 32 x 3
##    car                   mpg   cyl
##    <chr>               <dbl> <dbl>
##  1 Mazda RX4            21       6
##  2 Mazda RX4 Wag        21       6
##  3 Datsun 710           22.8     4
##  4 Hornet 4 Drive       21.4     6
##  5 Hornet Sportabout    18.7     8
##  6 Valiant              18.1     6
##  7 Duster 360           14.3     8
##  8 Merc 240D            24.4     4
##  9 Merc 230             22.8     4
## 10 Merc 280             19.2     6
## # ... with 22 more rows
```

```
select(dfm, starts_with("c"));
```

```
## # A tibble: 32 x 3
##    car                   cyl  carb
```

See the Select "helpers"

Run the command:

```
library(tidyselect);
?tidyselect::select_helpers
```

Here are a few:

```
any_of()
all_of()
last_col()
starts_with()
ends_with()
contains() # like searching
matches() # Matches a regular expression - cover later
```

Select/drop variables

- select() allows you to **drop** specific columns from your data.

```
select(dfm, -c(vs, am, gear, carb));
```

```
## # A tibble: 32 x 8
##    car                 mpg   cyl  disp    hp  drat    wt  qsec
##    <chr>             <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Mazda RX4          21       6  160    110  3.9   2.62  16.5
##  2 Mazda RX4 Wag      21       6  160    110  3.9   2.88  17.0
##  3 Datsun 710         22.8     4  108     93  3.85  2.32  18.6
##  4 Hornet 4 Drive     21.4     6  258    110  3.08  3.22  19.4
##  5 Hornet Sportabout  18.7     8  360    175  3.15  3.44  17.0
##  6 Valiant            18.1     6  225    105  2.76  3.46  20.2
##  7 Duster 360         14.3     8  360    245  3.21  3.57  15.8
##  8 Merc 240D          24.4     4  147.    62  3.69  3.19  20
##  9 Merc 230           22.8     4  141.    95  3.92  3.15  22.9
## 10 Merc 280           19.2     6  168.   123  3.92  3.44  18.3
## # ... with 22 more rows
```

Combining `filter` and `select`

You can combine `filter` and `select` to subset the rows and columns, respectively, of a `data.frame`:

```
select(filter(dfm, mpg > 20, cyl == 4), car,cyl, hp);
```

```
## # A tibble: 11 x 3
##    car              cyl    hp
##    <chr>          <dbl> <dbl>
##  1 Datsun 710         4    93
##  2 Merc 240D          4    62
##  3 Merc 230           4    95
##  4 Fiat 128           4    66
##  5 Honda Civic        4    52
##  6 Toyota Corolla     4    65
##  7 Toyota Corona      4    97
##  8 Fiat X1-9          4    66
##  9 Porsche 914-2      4    91
## 10 Lotus Europa       4   113
## 11 Volvo 142E         4   109
```

In `R`, the common way to perform multiple operations is to wrap functions around each other in a nested way such as above.

The %>% operator

Recently, the pipe %>% makes things such as this much more readable. It reads left side "pipes" into right side. RStudio `CMD/Ctrl + Shift + M` shortcut. Pipe `dfm` into `filter`, then pipe that into `select`:

```
dfm %>% filter(mpg > 20, cyl == 4) %>% select(car,cyl, hp);
```

```
## # A tibble: 11 x 3
##    car               cyl    hp
##    <chr>           <dbl> <dbl>
##  1 Datsun 710          4    93
##  2 Merc 240D           4    62
##  3 Merc 230            4    95
##  4 Fiat 128            4    66
##  5 Honda Civic         4    52
##  6 Toyota Corolla      4    65
##  7 Toyota Corona       4    97
##  8 Fiat X1-9           4    66
##  9 Porsche 914-2       4    91
## 10 Lotus Europa        4   113
## 11 Volvo 142E          4   109
```

The %>% operator

- Powerful trick for coding a sequence of operations
- Output of old operation as the first argument of new operation
- Especially useful in combined with package `ggplot2`

```
# input     +--------+      +--------+      +--------+     result
# data  %>% |  verb  |  %>% |  verb  |  %>% |  verb  | ->  data
# frame     +--------+      +--------+      +--------+     frame
```

Add/remove variables

- Adding new columns to a `data.frame`: base R

You can add a new column, called `newcol` to `dfm`, using the $ operator:

```
dfm$newcol = dfm$wt/2;
head(dfm,3)
```

```
## # A tibble: 3 x 13
##   car      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  ge
##   <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
## 1 Mazd~   21       6   160   110  3.9   2.62  16.5     0     1
## 2 Mazd~   21       6   160   110  3.9   2.88  17.0     0     1
## 3 Dats~   22.8     4   108    93  3.85  2.32  18.6     1     1
```

Add/remove variables

- The $ method is very common.
- The mutate function in dplyr allows you to add variables to your dataset.

```
dfm = mutate(dfm, newcol = wt/2);
```

Creating conditional variables

- One frequently-used tool is creating variables with conditions.
- A general function for creating new variables based on existing variables is the `ifelse()` function, which "returns a value with the same shape as test which is filled with elements selected from either yes or no depending on whether the element of test is `TRUE` or `FALSE`." It is much faster than combining "if(){} else{}" and "for" loop.

```
ifelse(test, yes, no)

# test: an object which can be coerced
    to logical mode.
# yes: return values for true elements of test.
# no: return values for false elements of test.
```

Add/remove variables

Combined with `ifelse(condition, TRUE, FALSE)`, it can give you:

```
dfm = mutate(dfm,
            disp_cat = ifelse(
              disp <= 200,
              "Low",
              ifelse(disp <= 400,
                     "Medium",
                     "High")
                         )
           );
head(dfm$disp_cat);
```

```
## [1] "Low"    "Low"    "Low"    "Medium" "Medium" "Medium"
```

Add/remove variables

Alternatively, `dplyr::case_when` provides a clean syntax as well.

```
dfm = mutate(dfm,
            disp_cat2 = case_when(
              disp <= 200 ~ "Low",
              disp > 200 & disp <= 400 ~ "Medium",
              disp > 400 ~ "High")
            );
head(dfm$disp_cat2);
```

```
## [1] "Low"    "Low"    "Low"    "Medium" "Medium" "Medium"
```

Transmutation

The transmute function in dplyr combines both the mutate and select functions. One can create new columns and keep the only the columns wanted:

```
transmute(dfm, newcol2 = wt/2, mpg, hp);
```

```
## # A tibble: 32 x 3
##    newcol2   mpg    hp
##      <dbl> <dbl> <dbl>
##  1    1.31  21     110
##  2    1.44  21     110
##  3    1.16  22.8    93
##  4    1.61  21.4   110
##  5    1.72  18.7   175
##  6    1.73  18.1   105
##  7    1.78  14.3   245
##  8    1.60  24.4    62
##  9    1.58  22.8    95
## 10    1.72  19.2   123
## # ... with 22 more rows
```

Add/remove variables

- Recall that we can remove a column by assigning to NULL:

```
dfm$newcol = NULL;
```

- The NULL method is still very common.
- Or we can use the select function a minus (-)

```
select(dfm, -newcol);
```

```
## # A tibble: 32 x 14
##    car           mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <c
##  1 Mazda RX4    21       6  160    110  3.9   2.62  16.5     0
##  2 Mazda RX4 ~  21       6  160    110  3.9   2.88  17.0     0
##  3 Datsun 710   22.8     4  108     93  3.85  2.32  18.6     1
##  4 Hornet 4 D~  21.4     6  258    110  3.08  3.22  19.4     1
##  5 Hornet Spo~  18.7     8  360    175  3.15  3.44  17.0     0
##  6 Valiant      18.1     6  225    105  2.76  3.46  20.2     1
##  7 Duster 360   14.3     8  360    245  3.21  3.57  15.8     0
##  8 Merc 240D    24.4     4  147.    62  3.69  3.19  20       1
##  9 Merc 230     22.8     4  141.    95  3.92  3.15  22.9     1
```

Rename variables

We can use the `colnames` function to directly reassign column names of `df`:

```
colnames(dfm)[2:4] = c("MPG", "CYL", "DISP");
head(dfm);
```

```
## # A tibble: 6 x 15
##   car     MPG   CYL  DISP    hp  drat    wt  qsec    vs    am ge
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
## 1 Mazd~  21       6   160   110  3.9   2.62  16.5     0     1
## 2 Mazd~  21       6   160   110  3.9   2.88  17.0     0     1
## 3 Dats~  22.8     4   108    93  3.85  2.32  18.6     1     1
## 4 Horn~  21.4     6   258   110  3.08  3.22  19.4     1     0
## 5 Horn~  18.7     8   360   175  3.15  3.44  17.0     0     0
## 6 Vali~  18.1     6   225   105  2.76  3.46  20.2     1     0
## # ... with 2 more variables: disp_cat <chr>, disp_cat2 <chr>
```

```
colnames(dfm)[2:4] = c("mpg", "cyl", "disp"); #reset
```

Rename variables

- rename( ) function can be used to change variable name.

```
rename(data, new_name = old_name)
```

```
dfm=rename(dfm, WT= wt);
head(dfm);
```

```
## # A tibble: 6 x 15
##   car     mpg   cyl  disp    hp  drat    WT  qsec    vs    am  ge
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
## 1 Mazd~  21       6   160   110  3.9   2.62  16.5     0     1
## 2 Mazd~  21       6   160   110  3.9   2.88  17.0     0     1
## 3 Dats~  22.8     4   108    93  3.85  2.32  18.6     1     1
## 4 Horn~  21.4     6   258   110  3.08  3.22  19.4     1     0
## 5 Horn~  18.7     8   360   175  3.15  3.44  17.0     0     0
## 6 Vali~  18.1     6   225   105  2.76  3.46  20.2     1     0
## # ... with 2 more variables: disp_cat <chr>, disp_cat2 <chr>
```

```
dfm=rename(dfm, wt= WT);  #reset
```

Sort data

- The `select` function can reorder columns. Put `newcol` first, then select the rest of columns.

```
select(dfm, cyl, everything());
```

```
## # A tibble: 32 x 15
##      cyl car            mpg  disp    hp  drat    wt  qsec    vs
##    <dbl> <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <c
##  1     6 Mazda RX4     21    160    110  3.9   2.62  16.5     0
##  2     6 Mazda RX4 ~   21    160    110  3.9   2.88  17.0     0
##  3     4 Datsun 710    22.8  108     93  3.85  2.32  18.6     1
##  4     6 Hornet 4 D~   21.4  258    110  3.08  3.22  19.4     1
##  5     8 Hornet Spo~   18.7  360    175  3.15  3.44  17.0     0
##  6     6 Valiant       18.1  225    105  2.76  3.46  20.2     1
##  7     8 Duster 360    14.3  360    245  3.21  3.57  15.8     0
##  8     4 Merc 240D     24.4  147.    62  3.69  3.19  20       1
##  9     4 Merc 230      22.8  141.    95  3.92  3.15  22.9     1
## 10     6 Merc 280      19.2  168.   123  3.92  3.44  18.3     1
## # ... with 22 more rows, and 3 more variables: newcol <dbl>, disp
## #   disp_cat2 <chr>
```

Sort data

- arrange() sorts your data (by rows). In base R, this is commonly done with order().
- By default, arrange orders in ascending order.
- Use the desc to arrange the rows in descending order:

```
data=data.frame(x = c(2,3,5,1,4), y=c(10, 9, 7, 8, 6));
arrange(data,x,y);
```

```
##   x  y
## 1 1  8
## 2 2 10
## 3 3  9
## 4 4  6
## 5 5  7
```

```
arrange(data, desc(x));
```

```
##   x  y
## 1 5  7
## 2 4  6
## 3 3  9
```

Sort data

- It is a bit more straightforward to mix increasing and decreasing orderings

```
arrange(dfm, mpg, desc(hp));
```

```
## # A tibble: 32 x 15
##    car              mpg   cyl  disp    hp  drat    wt  qsec    vs
##    <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <c
##  1 Lincoln Co~     10.4     8  460    215  3     5.42  17.8     0
##  2 Cadillac F~     10.4     8  472    205  2.93  5.25  18.0     0
##  3 Camaro Z28      13.3     8  350    245  3.73  3.84  15.4     0
##  4 Duster 360      14.3     8  360    245  3.21  3.57  15.8     0
##  5 Chrysler I~     14.7     8  440    230  3.23  5.34  17.4     0
##  6 Maserati B~     15       8  301    335  3.54  3.57  14.6     0
##  7 Merc 450SLC     15.2     8  276.   180  3.07  3.78  18       0
##  8 AMC Javelin     15.2     8  304    150  3.15  3.44  17.3     0
##  9 Dodge Chal~     15.5     8  318    150  2.76  3.52  16.9     0
## 10 Ford Pante~     15.8     8  351    264  4.22  3.17  14.5     0
## # ... with 22 more rows, and 3 more variables: newcol <dbl>, disp
## #   disp_cat2 <chr>
```

Remove duplicate rows

- The `distinct()` function is used to eliminate duplicates.
- Remove duplicate rows based on all variables

```
dim(dfm);
```

```
## [1] 32 15
```

```
dfm1 = distinct(dfm);
dim(dfm1);
```

```
## [1] 32 15
```

- Remove duplicate rows **based on some variables**
  - `.keep_all` option is used to retain all other variables in the output data frame. It is FALSE be default

```
dfm2 = distinct(dfm, mpg,cyl, .keep_all= TRUE)
dim(dfm2)
```

```
## [1] 27 15
```

License