



## Data Wrangling in R - Data Wrangling - 4

*One should look for what is and not what he thinks should be. (Albert Einstein)*

# Module completion checklist

Objective	Complete
Select specific variables using the select command	
Derive new variables from the existing variables using the mutate and transmute commands	

# Select

- `select()` helps you select specific columns within your dataframe

```
# Check for detailed documentation
?dplyr::select

# Use cases for `select` function
select(df,                #<- dataframe
       select_cond1,    #<- selection rule(s)
       ...)
```

- We often use this function with pipes(`%>%`) which we will cover later in the course
- The selection criteria can be written in multiple ways, as shown in the next couple of slides

## Usage

```
select(.data, ...)
```

## Arguments

- `.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.
- `...` [<tidy-select>](#) One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like `x:y` can be used to select a range of variables.

# Select a subset

- Simply specify the column name(s)

```
# Select columns from `flights` data frame.
select(flights, #<- specify the data frame
        year,   #<- specify the 1st column
        month,  #<- specify the 2nd column
        day)    #<- specify the 3rd column
```

```
# A tibble: 336,776 x 3
   year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
4  2013     1     1
5  2013     1     1
6  2013     1     1
7  2013     1     1
8  2013     1     1
9  2013     1     1
10 2013     1     1
# ... with 336,766 more rows
```

- You can also specify a range of columns with the range operator (i.e. :)

```
# Select columns from `flights` data frame
select(flights, #<- specify the data frame
        year:day) #<- specify the range of columns
```

```
# A tibble: 336,776 x 3
   year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
4  2013     1     1
5  2013     1     1
6  2013     1     1
7  2013     1     1
8  2013     1     1
9  2013     1     1
10 2013     1     1
# ... with 336,766 more rows
```

# Select by excluding

- Finally, you can select by excluding certain columns using the exclusion operator (i.e. `-`)

```
# Select multiple columns from `flights` data frame by providing which columns to exclude in selection
select(flights,          #<- specify the data frame
       -(year:day))      #<- specify the range of columns to exclude
```

```
# A tibble: 336,776 x 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
   <int>      <int>      <dbl>    <int>      <int>      <dbl>    <chr>
1     517         515         2      830         819        11    UA
2     533         529         4      850         830        20    UA
3     542         540         2      923         850        33    AA
4     544         545        -1     1004        1022       -18    B6
5     554         600        -6      812         837       -25    DL
6     554         558        -4      740         728        12    UA
7     555         600        -5      913         854        19    B6
8     557         600        -3      709         723       -14    EV
9     557         600        -3      838         846        -8    B6
10    558         600        -2      753         745         8    AA
# ... with 336,766 more rows, and 9 more variables: flight <int>, tailnum <chr>,
# origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
# minute <dbl>, time_hour <dtm>
```

# Select - helper functions

- Helpers are multiple functions you can use to select variables based on their names
- They act like regular expressions, but in a more simplified manner
- Here are some of the more commonly used helper functions:

Helper Function	Use Case
<code>starts_with("abc")</code>	matches names that begin with "abc"
<code>ends_with("xyz")</code>	matches names that end with "xyz"
<code>contains("ijk")</code>	matches names that contain "ijk"
<code>num_range("x", 1:3)</code>	matches "x1", "x2" and "x3"

# Select: Helper functions example

- To select columns whose names start with 'arr':

```
select(flights, starts_with("arr"))
```

```
# A tibble: 336,776 x 2
  arr_time arr_delay
  <int>      <dbl>
1     830         11
2     850         20
3     923         33
4    1004        -18
5     812        -25
6     740         12
7     913         19
8     709        -14
9     838         -8
10    753          8
# ... with 336,766 more rows
```

# Module completion checklist

Objective	Complete
Select specific variables, sometimes using specific rules, using the select command	✓
Derive new variables from the existing variables using the mutate and transmute commands	



# Mutate

- `mutate()` is an essential function of `dplyr`
- It allows us to **create** new variables using the current data and **append** these variables to the existing dataframe

```
?dplyr::mutate
```

```
mutate(df,           # <- dataframe  
       new_col1,    # <- rule(s) for the new column  
       ...)
```

- **Note:** `mutate()` always adds columns to the end of the dataset, so make sure you are able to see the last columns

mutate {dplyr}

R Documentation

## Create, modify, and delete columns

### Description

`mutate()` adds new variables and preserves existing ones; `transmute()` adds new variables and drops existing ones. New variables overwrite existing variables of the same name. Variables can be removed by setting their value to `NULL`.

### Usage

```
mutate(.data, ...)
```

```
## S3 method for class 'data.frame'
```

```
mutate(  
  .data,  
  ...,  
  .keep = c("all", "used", "unused", "none"),  
  .before = NULL,  
  .after = NULL  
)
```

```
transmute(.data, ...)
```

### Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code> ). See <i>Methods</i> , below, for more details.
--------------------	---

# First - create a new dataset

- Create the dataset using `select()`

```
# Let's select columns of `flights` data frame and save them as `flights_sml`.
flights_sml = select(flights, #<- specify data frame
  year:day, #<- specify range of columns to include
  ends_with("delay"), #<- find all columns that end with `delay`
  distance, #<- select `distance` column
  air_time) #<- select `air_time` column

flights_sml
```

```
# A tibble: 336,776 x 7
   year month   day dep_delay arr_delay distance air_time
  <int> <int> <int>    <dbl>    <dbl>    <dbl>    <dbl>
1  2013     1     1         2        11     1400      227
2  2013     1     1         4        20     1416      227
3  2013     1     1         2        33     1089      160
4  2013     1     1        -1       -18     1576      183
5  2013     1     1        -6       -25      762      116
6  2013     1     1        -4        12      719      150
7  2013     1     1        -5        19     1065      158
8  2013     1     1        -3       -14      229       53
9  2013     1     1        -3        -8      944      140
10 2013     1     1        -2         8      733      138
# ... with 336,766 more rows
```

# Mutate - arguments

- The first argument is the dataframe
- The following arguments are the columns we would like to add to the data frame

```
# Add two columns `gain` and `speed` to `flights_sml`.
mutate(flights_sml,                                #<- specify the data frame
       gain = arr_delay - dep_delay,               #<- create `gain` column by subtracting departure delay
                                              # from arrival delay
       speed = distance / air_time * 60)          #<- create `speed` from distance and air time columns
```

```
# A tibble: 336,776 x 9
   year month   day dep_delay arr_delay distance air_time   gain speed
  <int> <int> <int>   <dbl>   <dbl>   <dbl>   <dbl> <dbl> <dbl>
1  2013     1     1         2        11    1400    227     9   370.
2  2013     1     1         4        20    1416    227    16   374.
3  2013     1     1         2        33    1089    160    31   408.
4  2013     1     1        -1       -18    1576    183   -17   517.
5  2013     1     1        -6       -25     762    116   -19   394.
6  2013     1     1        -4        12     719    150    16   288.
7  2013     1     1        -5        19    1065    158    24   404.
8  2013     1     1        -3       -14     229     53   -11   259.
9  2013     1     1        -3        -8     944    140    -5   405.
10 2013     1     1        -2         8     733    138    10   319.
# ... with 336,766 more rows
```

# Transmute

- `transmute()` is a function that does the same thing as `mutate()` **except it will only keep the new columns**

```
transmute(df,           # <- dataframe
          new_col1,     # <- rule(s) for new column
          ...)
```

- The first argument is the dataframe
- The following arguments are the columns that will be included in your new data frame

mutate {dplyr}

R Documentation

## Create, modify, and delete columns

### Description

`mutate()` adds new variables and preserves existing ones; `transmute()` adds new variables and drops existing ones. New variables overwrite existing variables of the same name. Variables can be removed by setting their value to `NULL`.

### Usage

```
mutate(.data, ...)
```

```
## S3 method for class 'data.frame'
```

```
mutate(
  .data,
  ...,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)
```

```
transmute(.data, ...)
```

### Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code> ). See <i>Methods</i> , below, for more details.
--------------------	---

# Transmute example

- With the same arguments as in the `mutate()` example, we can see `transmute()` function only returns new columns

```
# Add two columns `gain` and `speed` to `flights_sml`.
transmute(flights_sml,                                #<- specify the data frame
          gain = arr_delay - dep_delay,                #<- create `gain` column by subtracting departure delay
                                                    #   from arrival delay
          speed = distance / air_time * 60)           #<- create `speed` from distance and air time columns
```

```
# A tibble: 336,776 x 2
   gain speed
  <dbl> <dbl>
1      9  370.
2     16  374.
3     31  408.
4    -17  517.
5    -19  394.
6     16  288.
7     24  404.
8    -11  259.
9     -5  405.
10     10  319.
# ... with 336,766 more rows
```

# Mutate and transmute - useful functions

- When creating new variables with `mutate/transmute`, there are many helpful widgets and functions that can assist in creating interesting features:

Useful Functions	Explanation
<code>+, -, *, /, ^</code>	all mathematical operators can be used on variables
<code>log, log2, log10</code>	logarithmic functions for variable transformation can be used
<code>%/%</code> and <code>%%</code>	modulus and remainder are useful when converting time
<code>lag(x)</code> and <code>lead(x)</code>	lag and lead allow reference to leading or lagging values - useful for detecting changes in values.
<code>cumsum(x), cummean(x), cummax(x), cumprod(x)</code>	cumulative, running functions, mins, max, prod, mean, etc.



# Mutate and transmute - useful functions (cont'd)

- **Ranking functions** are very helpful in data manipulation
- There are several within the `dplyr` package such as `row_number()`, `ntile()` and `dense_rank()`

```
# Check for detailed documentation
rank_function(x) # <- one of rank functions with
                # a vector of values to rank
```

ranking {dplyr}

R Documentation

## Windowed rank functions.

### Description

Six variations on ranking functions, mimicking the ranking functions described in SQL2003. They are currently implemented using the built in `rank` function, and are provided mainly as a convenience when converting between R and SQL. All ranking functions map smallest inputs to smallest outputs. Use [desc\(.\)](#) to reverse the direction.

### Usage

`row_number(x)`

`ntile(x = row_number(), n)`

`min_rank(x)`

`dense_rank(x)`

`percent_rank(x)`

`cume_dist(x)`

### Arguments

`x` a vector of values to rank. Missing values are left as is. If you want to treat them as the smallest or largest values, replace with `Inf` or `-Inf` before ranking.

`n` number of groups to split up into.

# Knowledge check





# Exercise

You are now ready to try Tasks 13-18 in the Exercise for this topic



# Module completion checklist

Objective	Complete
Select specific variables, sometimes using specific rules, using the select command	✓
Derive new variables from the existing variables using the mutate and transmute commands	✓

# Data Wrangling: Topic summary

In this part of the course, we have covered:

- Data wrangling basics
- Use dplyr for data wrangling

# Congratulations on completing this module!

