DATA SOCIETY:

Data Wrangling in R - Data Wrangling - 3

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

Warm-up: review activity

- Match each term to the correct definition below, and share responses in the chat (ex: “1,E” to match the first function to the definition labelled “E”)

1. `filter`
2. `arrange`
3. `select`
4. `mutate`
5. `summarize`
6. `group_by`

A. Pick variables by their names

B. Collapse many values down to a single summary

C. Reorder the rows

D. Allows the above functions to operate on a dataset group by group

E. Pick observations by their value

F. Create new variables with functions of existing variables

Warm-up: Review activity answers

- The answers are: **1,E; 2,C; 3,A; 4,F; 5,B; 6,D**

Function	Use Case
`filter`	Pick observations by their value
`arrange`	Reorder the rows
`select`	Pick variables by their names
`mutate`	Create new variables with functions of existing variables
`summarize`	Collapse many values down to a single summary
`group_by`	Allows the above functions to operate on a dataset group by group

- Any questions?

Module completion checklist

Objective	Complete
Apply the filter function to subset data	
Rank data using the arrange function	

Getting started with dplyr

- Now that we have dplyr installed and the nycflights13 dataset loaded, we can start transforming our dataset
- Our goal is to get exposure to the core dplyr syntax and practice using some of the major function verbs
- For now, we will focus on just two:
  - `filter()`, to subset the observations in the data
  - `arrange()`, to reorder the observations in the data

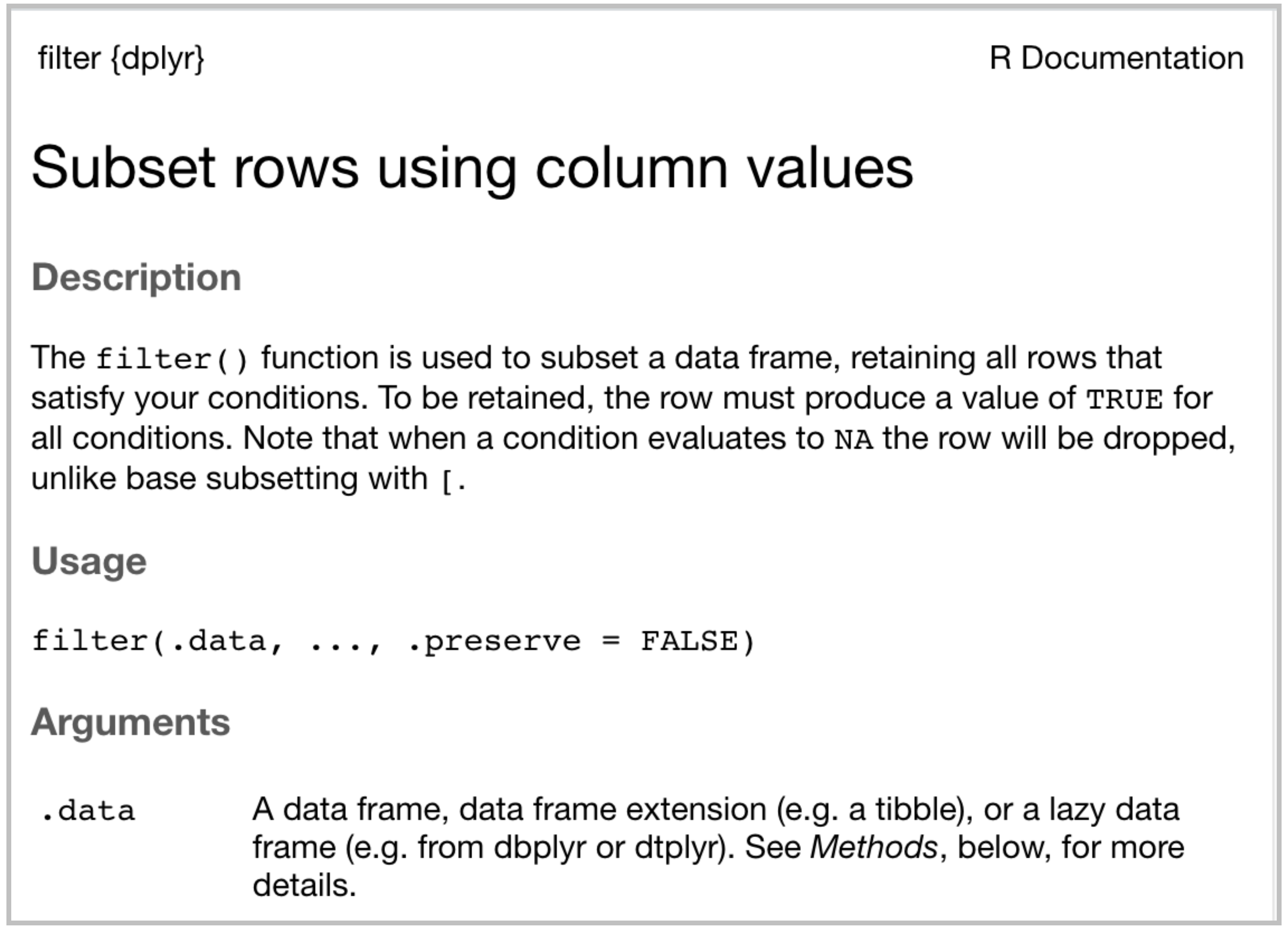
Filter

- `filter` allows you to subset observations based on their values
- Basic use cases for `filter` function include:
  - Subsetting the data to include flights from January 2013
  - Subsetting the data that contain missing values

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

# Use cases for `filter` function

filter(df,            #<- dataframe
       filter_cond1,  #<- subsetting rule(s)
       ...)           #<- other arguments
```



filter {dplyr}   R Documentation

Subset rows using column values

Description

The `filter()` function is used to subset a data frame, retaining all rows that satisfy your conditions. To be retained, the row must produce a value of `TRUE` for all conditions. Note that when a condition evaluates to `NA` the row will be dropped, unlike base subsetting with `[`.

Usage

```
filter(.data, ..., .preserve = FALSE)
```

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.

Filter

- Let's say you would like to see all flights from January 2013
- Using `filter`, pass the original dataframe followed by filtering criteria

```
# Load the flights dataset into the environment.
data(flights)

# Filter `flights` data frame to display all records from January (month == 1) of 2013 (year ==
2013).
filter(flights,       #<- set data
       month == 1,    #<- filter by month
       year == 2013)  #<- filter by year
```

```
# A tibble: 27,004 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# … with 26,994 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Filter

- If you want to build on top of the filtered dataset, you will need to save your new subset to a new variable and perform further operations on this new subset

```
# You will have to make sure to save the subset. To do this, use `=`.
filter_flights = filter(flights, month == 1, year == 2013)

# View your output.
filter_flights
```

```
# A tibble: 27,004 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# … with 26,994 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Filter options

- You can use the standard filtering operations when working with integer data types:

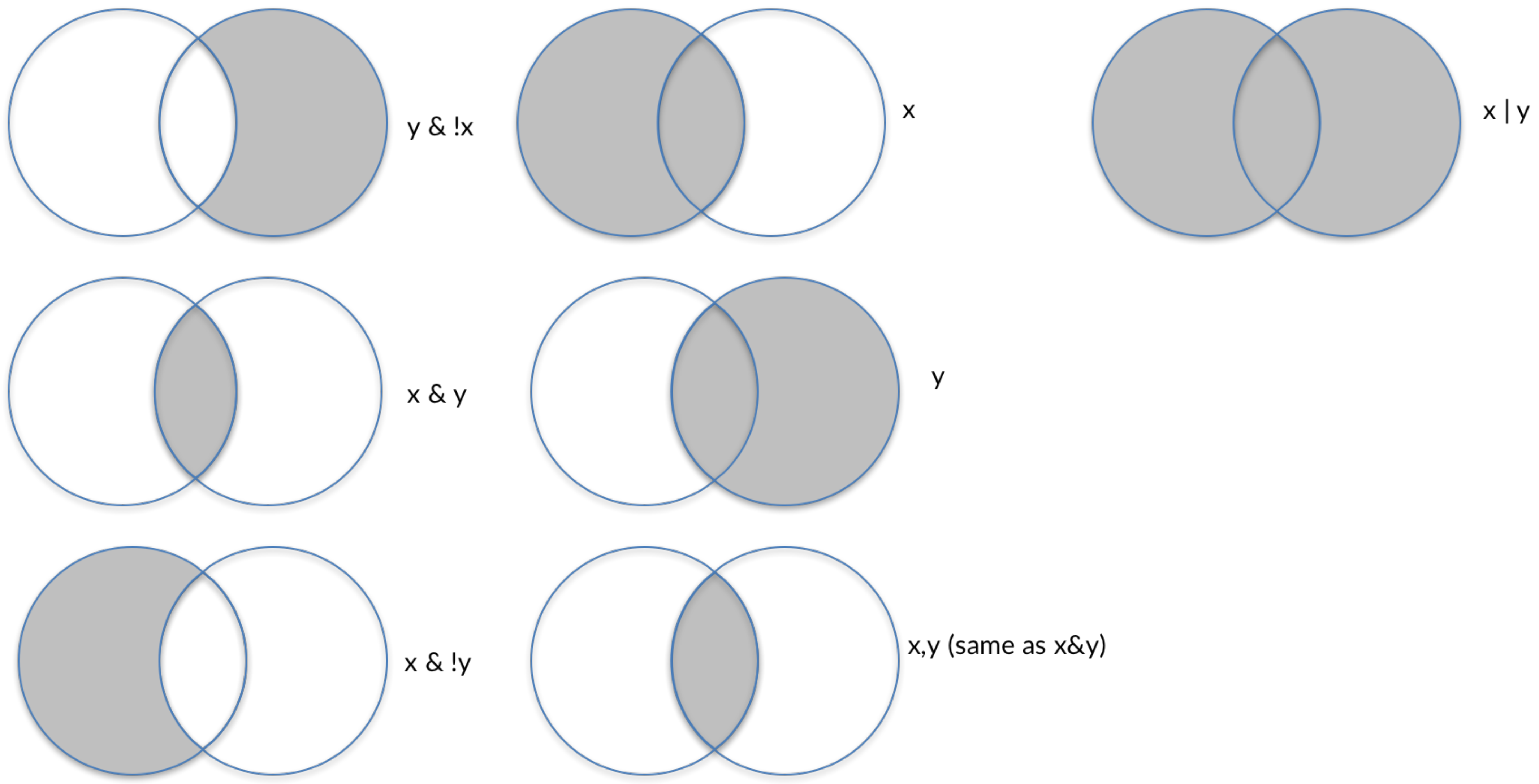
Operation	Use Case	Example
>	Greater than	6 > 4
>=	Greater than or equal to	4 >= 4
<	Less than	4 < 6
<=	Less than or equal to	4 <= 4
! =	Not equal to	4 != 6
==	Equal to	4 == 4

Filter options (cont'd)

- And more general operators:

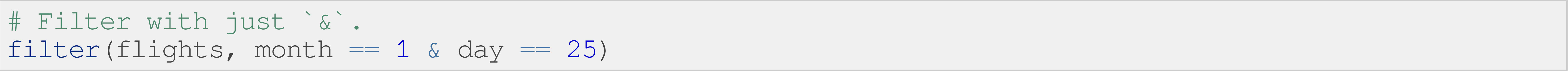
Operation	Use Case	Example
|	either can be true to satisfy	x == 4 | x == 12, x==2 | x==13
&	and, both need to be true	x == 4 & y == 2
!	Not true, inverse selection	x != 4
%in%	value in the following list of values	x %in% c(4,16,32)

Filter - logical operators

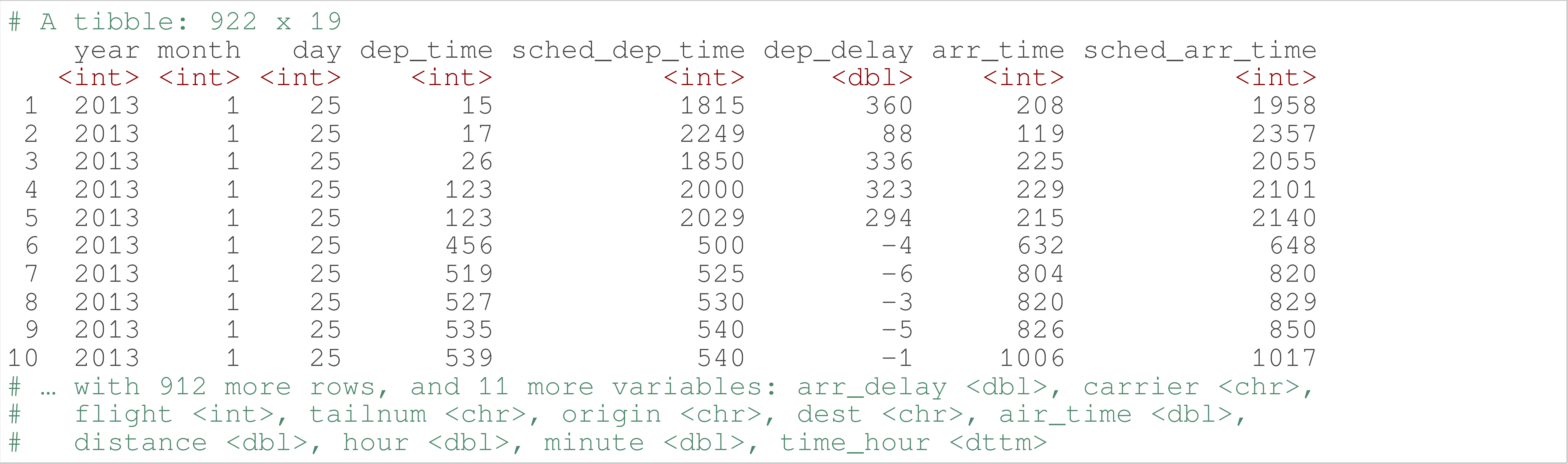


Filter - examples of logical operators

- What if we want to see all flights from January **and** on the 25th?



```
# Filter with just `&`.
filter(flights, month == 1 & day == 25)
```



```
# A tibble: 922 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1    25       15           1815       360      208           1958
 2  2013     1    25       17           2249        88      119           2357
 3  2013     1    25       26           1850       336      225           2055
 4  2013     1    25      123           2000       323      229           2101
 5  2013     1    25      123           2029       294      215           2140
 6  2013     1    25      456            500        -4      632            648
 7  2013     1    25      519            525        -6      804            820
 8  2013     1    25      527            530        -3      820            829
 9  2013     1    25      535            540        -5      826            850
10  2013     1    25      539            540        -1     1006           1017
# … with 912 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

- ***Note:*** *After running each example, we will record the number of rows. This will help illustrate each operator and how different a simple change of one boolean operator can have on the dataset. Total number of rows should be 922*

Filter - examples of logical operators (cont’d)

- What if we want to see all flights, but **exclude** those from January and those on the 25th?

```
# Filter with `!`.
filter(flights, month != 1 & day != 25)
```

```
# A tibble: 299,597 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013    10     1      447            500       -13      614            648
 2  2013    10     1      522            517         5      735            757
 3  2013    10     1      536            545        -9      809            855
 4  2013    10     1      539            545        -6      801            827
 5  2013    10     1      539            545        -6      917            933
 6  2013    10     1      544            550        -6      912            932
 7  2013    10     1      549            600       -11      653            716
 8  2013    10     1      550            600       -10      648            700
 9  2013    10     1      550            600       -10      649            659
10  2013    10     1      551            600        -9      727            730
# … with 299,587 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

- *Here we are looking for all flights that are not in January and not on the 25th; total number of rows should be 299,597*

Filter - examples of logical operators (cont’d)

```
# Filter with `%in%`.
filter(flights, month %in% c(1, 2) & day == 25)
```

```
# A tibble: 1,883 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1    25       15           1815       360      208           1958
 2  2013     1    25       17           2249        88      119           2357
 3  2013     1    25       26           1850       336      225           2055
 4  2013     1    25      123           2000       323      229           2101
 5  2013     1    25      123           2029       294      215           2140
 6  2013     1    25      456            500        -4      632            648
 7  2013     1    25      519            525        -6      804            820
 8  2013     1    25      527            530        -3      820            829
 9  2013     1    25      535            540        -5      826            850
10  2013     1    25      539            540        -1     1006           1017
# … with 1,873 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

- *This is a combination of `&` and `%in%` subsetting all flights from January and February that are on the 25th; number of rows should be 1,883*

Using filter with NA values

- `filter` only includes rows where the condition is TRUE; it **excludes** both FALSE and NA values
- If you want to preserve missing values, ask for them explicitly

```
# Create a data frame with 2 columns.
NA_df = data.frame(x = c(1, NA, 2),   #<- column x with 3 entries with 1 NA
                   y = c(1, 2, 3))    #<- column y with 3 entries

# Filter without specifying anything regarding NAs.
filter(NA_df, x >= 1)
```

```
  x y
1 1 1
2 2 3
```

```
# Filter with specifying to keep rows if there is an NA.
filter(NA_df, is.na(x) | x >= 1)
```

```
   x y
1  1 1
2 NA 2
3  2 3
```

Module completion checklist

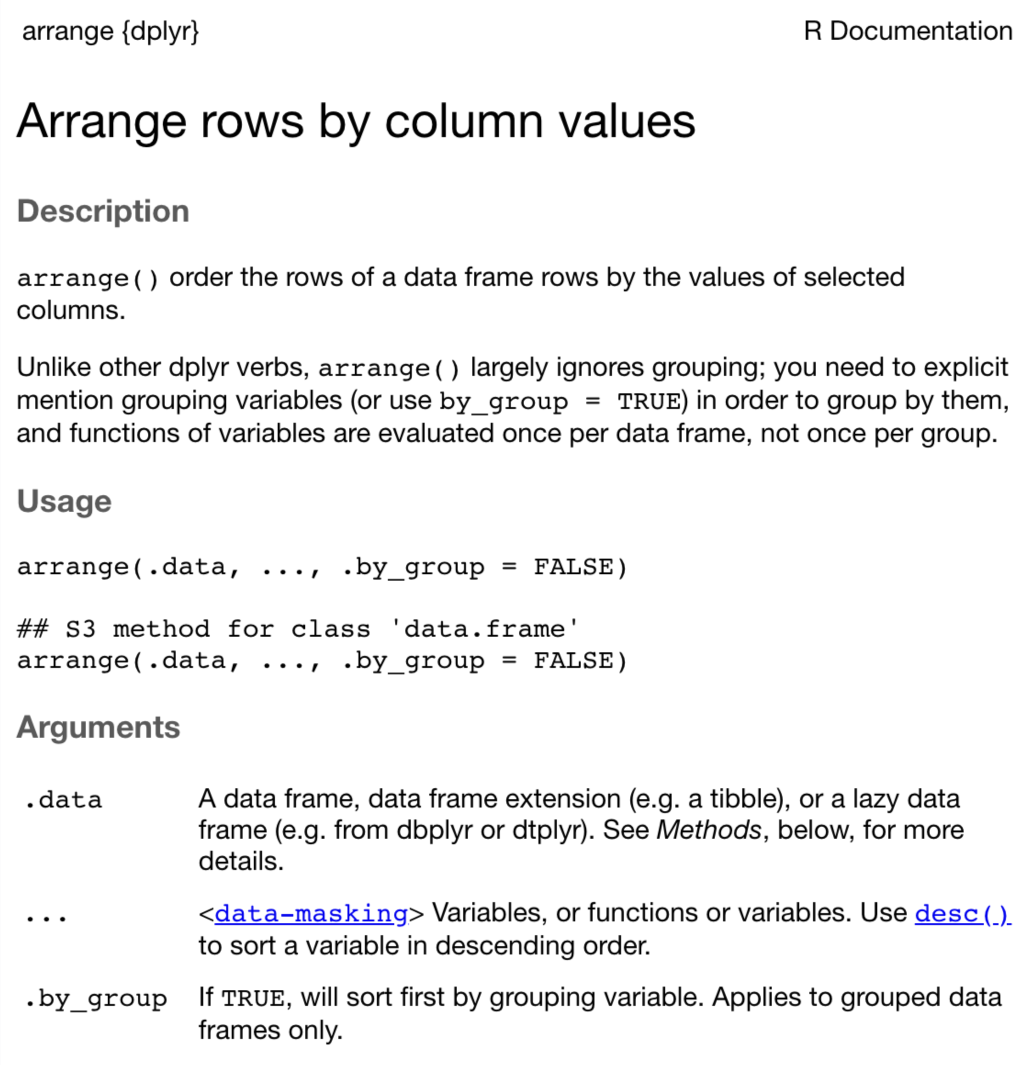
Objective	Complete
Apply the filter function to subset data	✔
Rank data using the arrange function	

Arrange

- `arrange` is used to reorder the observations within the specified column(s)
- It is the equivalent of `sort` in SAS or `order by` in SQL

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

# Use cases for `arrange` function
arrange(df,             #<- data frame
        arrange_cond1,  #<- column by which
                        #   to arrange
        ...)            #<- other args.
```



arrange {dplyr} R Documentation

Arrange rows by column values

Description

`arrange()` order the rows of a data frame rows by the values of selected columns.

Unlike other dplyr verbs, `arrange()` largely ignores grouping; you need to explicit mention grouping variables (or use `by_group = TRUE`) in order to group by them, and functions of variables are evaluated once per data frame, not once per group.

Usage

```
arrange(.data, ..., .by_group = FALSE)

## S3 method for class 'data.frame'
arrange(.data, ..., .by_group = FALSE)
```

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.
`...`	<`data-masking`> Variables, or functions or variables. Use `desc()` to sort a variable in descending order.
`.by_group`	If `TRUE`, will sort first by grouping variable. Applies to grouped data frames only.

Arrange example

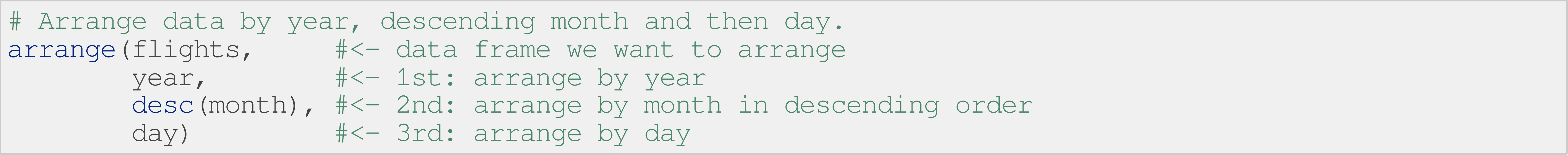
- When using multiple columns with `arrange`, the additional columns will be used to break ties in the values of preceding columns

```
# Arrange data by year, then month, and then day.
arrange(flights, #<- data frame we want to arrange
        year,    #<- 1st: arrange by year
        month,   #<- 2nd: arrange by month
        day)     #<- 3rd: arrange by day
```

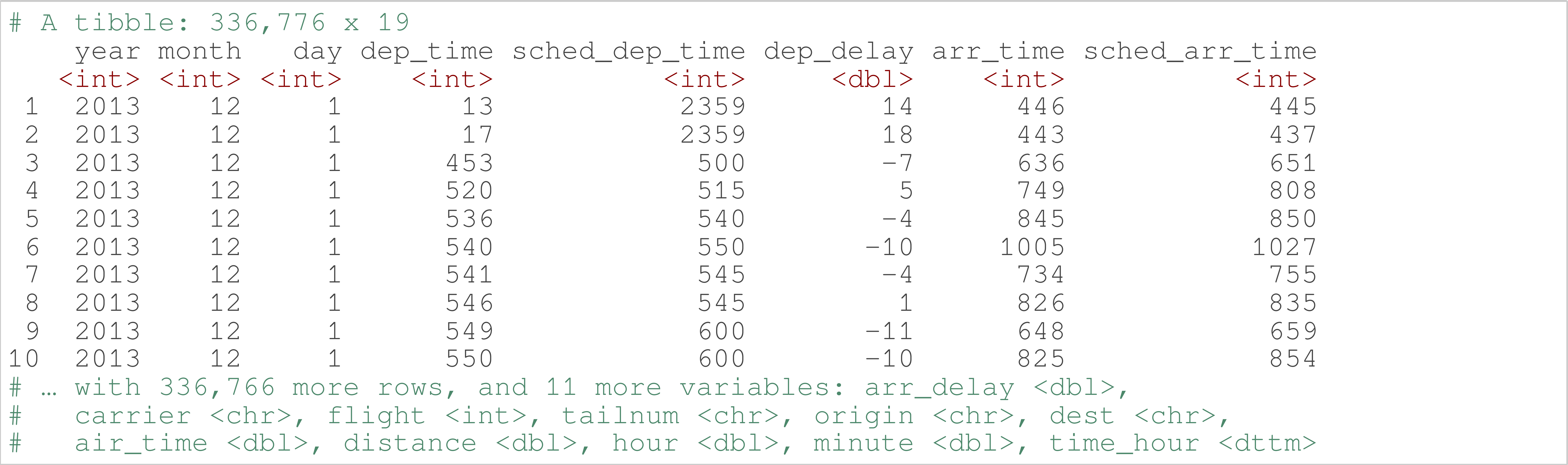
```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Arrange options

- `arrange` by default sorts everything in ascending order; to arrange in descending, use **`desc`**
- You can now see that the month at the top of the dataset is **December** (i.e., 12th month)



```
# Arrange data by year, descending month and then day.
arrange(flights,      #<- data frame we want to arrange
        year,         #<- 1st: arrange by year
        desc(month),  #<- 2nd: arrange by month in descending order
        day)          #<- 3rd: arrange by day
```



```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013    12     1       13           2359        14      446            445
 2  2013    12     1       17           2359        18      443            437
 3  2013    12     1      453            500        -7      636            651
 4  2013    12     1      520            515         5      749            808
 5  2013    12     1      536            540        -4      845            850
 6  2013    12     1      540            550       -10     1005           1027
 7  2013    12     1      541            545        -4      734            755
 8  2013    12     1      546            545         1      826            835
 9  2013    12     1      549            600       -11      648            659
10  2013    12     1      550            600       -10      825            854
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Arrange with NA values

- Missing values are **always** sorted at the end

```
# Arrange data with missing values.
arrange(NA_df, x)
```

```
   x y
1  1 1
2  2 3
3 NA 2
```

```
# Even when we use `desc` the `NA` is taken to the last row.
arrange(NA_df, desc(x))
```

```
   x y
1  2 3
2  1 1
3 NA 2
```

Knowledge check

Module completion checklist

Objective	Complete
Apply the filter function to subset data	✔
Rank data using the arrange function	✔

Congratulations on completing this module!

You are now ready to try Tasks 6-12 in the Exercise for this topic