

Tidy data & dplyr

Lecture 06

Dr. Colin Rundel



Tidy data

country	year	cases	population
Afghanistan	1999	745	15467071
Afghanistan	2000	2666	20495360
Brazil	1999	37737	17206362
Brazil	2000	80488	17404898
China	1999	212258	1272015272
China	2000	216766	128042583

variables

country	year	cases	population
Afghanistan	1999	745	15467071
Afghanistan	2000	2666	20495360
Brazil	1999	37737	17206362
Brazil	2000	80488	17404898
China	1999	212258	1272015272
China	2000	216766	128042583

observations

country	year	cases	population
Afghanistan	1999	745	15467071
Afghanistan	2000	2666	20495360
Brazil	1999	37737	17206362
Brazil	2000	80488	17404898
China	1999	212258	1272015272
China	2000	216766	128042583

values

From R4DS - tidy data

3

Tidy vs Untidy

Happy families are all alike; every unhappy family is unhappy in its own way

— Leo Tolstoy, Anna Karenina

A tibble: 317 × 7

	artist	track	date.entered	wk1	wk2	wk3	wk4
	<chr>	<chr>	<date>	<dbl>	<dbl>	<dbl>	<dbl>
1	2 Pac	Baby Don't Cry (Kee...	2000-02-26	87	82	72	77
2	2Ge+her	The Hardest Part Of...	2000-09-02	91	87	92	NA
3	3 Doors Down	Kryptonite	2000-04-08	81	70	68	67
4	3 Doors Down	Loser	2000-10-21	76	76	72	69
5	504 Boyz	Wobble Wobble	2000-04-15	57	34	25	17
6	98^0	Give Me Just One Ni...	2000-08-19	51	39	34	26
7	A*Teens	Dancing Queen	2000-07-08	97	97	96	95
8	Aaliyah	I Don't Wanna	2000-01-29	84	62	51	41
9	Aaliyah	Try Again	2000-03-18	59	53	38	28
10	Adams Yolanda	Open My Heart	2000-08-26	76	76	71	60

Is the above data set tidy?

More tidy vs untidy

Is the following data tidy?

List of 3

```
$ :List of 8
..$ name      : chr "Luke Skywalker"
..$ height    : chr "172"
..$ mass      : chr "77"
..$ hair_color: chr "blond"
..$ skin_color: chr "fair"
..$ eye_color : chr "blue"
..$ birth_year: chr "19BBY"
..$ gender    : chr "male"
$ :List of 8
..$ name      : chr "C-3P0"
..$ height    : chr "167"
```

List of 3

```
$ :List of 8
..$ name      : chr "Darth Vader"
..$ height    : chr "202"
..$ mass      : chr "136"
..$ hair_color: chr "none"
..$ skin_color: chr "white"
..$ eye_color : chr "yellow"
..$ birth_year: chr "41.9BBY"
..$ gender    : chr "male"
$ :List of 8
..$ name      : chr "Leia Organa"
..$ height    : chr "150"
```



Modern data frames

The tidyverse includes the tibble package that extends data frames to be a bit more modern. The core features of tibbles is to have a nicer printing method as well as being “surly” and “lazy”.

```
1 library(tibble)
```

```
1 iris
```

	Sepal.Length	Sepal.Width	Petal.Length
1	5.1	3.5	1.4
2	4.9	3.0	1.4
3	4.7	3.2	1.3
4	4.6	3.1	1.5
5	5.0	3.6	1.4
6	5.4	3.9	1.7
7	4.6	3.4	1.4
8	5.0	3.4	1.5
9	4.4	2.9	1.4
10	4.9	3.1	1.5
11	5.4	3.7	1.5
12	4.8	3.4	1.6
13	4.8	3.0	1.4
14	4.3	3.0	1.1
15	5.8	4.0	1.2
16	5.7	4.4	1.5

```
1 (tbl_iris = as_tibble(iris))
```

```
# A tibble: 150 × 5
  Sepal.Length Sepal.Width Petal.Length
      <dbl>      <dbl>      <dbl>
1         5.1         3.5         1.4
2         4.9         3.0         1.4
3         4.7         3.2         1.3
4         4.6         3.1         1.5
5          5.0         3.6         1.4
6         5.4         3.9         1.7
7         4.6         3.4         1.4
8          5.0         3.4         1.5
9         4.4         2.9         1.4
10        4.9         3.1         1.5
# i 140 more rows
# i 2 more variables: Petal.Width <dbl>,
#   Species <fct>
```

Tibbles are lazy - preserving type

By default, subsetting tibbles always results in another tibble (`$` or `[[` can still be used to subset for a specific column). i.e. tibble subsets are always preserving and therefore type consistent.

```
1 tbl_iris[1,]
```

```
# A tibble: 1 × 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>         <dbl>         <dbl>         <dbl> <fct>
1      5.1         3.5         1.4         0.2 setosa
```

```
1 tbl_iris[,1]
```

```
# A tibble: 150 × 1
  Sepal.Length
    <dbl>
1      5.1
2      4.9
3      4.7
4      4.6
5      5
6      5.4
7      4.6
8      5
9      4.4
10     4.9
# i 140 more rows
```

```
1 head(tbl_iris[[1]])
```

```
[1] 5.1 4.9 4.7 4.6 5.0 5.4
```

```
1 head(tbl_iris$Species)
```

```
[1] setosa setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica
```


Tibbles are lazy - No partial matching

Tibbles do not use partial matching when the `$` operator is used.

```
1 head( iris$Species )
```

```
[1] setosa setosa setosa setosa setosa  
setosa  
Levels: setosa versicolor virginica
```

```
1 head( iris$Sp )
```

```
[1] setosa setosa setosa setosa setosa  
setosa  
Levels: setosa versicolor virginica
```

```
1 head( tbl_iris$Species )
```

```
[1] setosa setosa setosa setosa setosa  
setosa  
Levels: setosa versicolor virginica
```

```
1 head( tbl_iris$Sp )
```

```
Warning: Unknown or uninitialised  
column: `Sp`.  
NULL
```

Tibbles are lazy - length coercion

Only vectors with length 1 will undergo length coercion / recycling - anything else throws an error.

```
1 data.frame(x = 1:4, y = 1)
```

```
  x y
1 1 1
2 2 1
3 3 1
4 4 1
```

```
1 data.frame(x = 1:4, y = 1:2)
```

```
  x y
1 1 1
2 2 2
3 3 1
4 4 2
```

```
1 tibble(x = 1:4, y = 1)
```

```
# A tibble: 4 × 2
      x     y
  <int> <dbl>
1     1     1
2     2     1
3     3     1
4     4     1
```

```
1 tibble(x = 1:4, y = 1:2)
```

```
Error in `tibble()`:
! Tibble columns must have compatible
sizes.
• Size 4: Existing data.
• Size 2: Column `y`.
i Only values of size one are recycled.
```

Tibbles and S3

```
1 t = tibble(  
2   x = 1:3,  
3   y = c("A","B","C")  
4 )  
5  
6 class(t)
```

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

```
1 d = data.frame(  
2   x = 1:3,  
3   y = c("A","B","C")  
4 )  
5  
6 class(d)
```

```
[1] "data.frame"
```

```
1 methods(class="tbl_df")
```

```
[1] [  
[6] $<-      as.data.frame coerce      initialize  names<-  
[11] Ops      row.names<- show          slotsFromS3 str  
[16] tbl_sum  
see '?methods' for accessing help and source code
```

```
1 methods(class="tbl")
```

```
[1] [[<-      [<-      $<-      coerce      format  
[6] glimpse    initialize Ops      print      show  
[11] slotsFromS3 tbl_sum  
see '?methods' for accessing help and source code
```

Tibble support?

Tibbles are just specialized data frames, and will fall back to the base data frame methods when needed.

```
1 d = tibble(  
2   x = rnorm(100),  
3   y = 3 + x + rnorm(100, sd = 0.1)  
4 )
```

```
1 lm(y~x, data = d)
```

Call:

```
lm(formula = y ~ x, data = d)
```

Coefficients:

(Intercept)	x
3.0044	0.9942

What did this model?

12



magrittr

What is a pipe

In software engineering, a pipeline consists of a chain of processing elements (processes, threads, coroutines, functions, etc.), arranged so that the output of each element is the input of the next;

[Wikipedia - Pipeline \(software\)](#)

Magrittr's pipe is an infix operator that allows us to link two functions together in a way that is readable from left to right.

The two code examples below are equivalent,

```
1 f(g(x=1, y=2), n=2)
```

```
1 g(x=1, y=2) %>% f(n=2)
```

Readability

Consider the following sequence of actions that describe the process of getting to campus in the morning:

I need to find my key, then unlock my car, then start my car, then drive to school, then park.

Expressed as a set of nested functions in R pseudocode this would look like:

```
1 park(drive(start_car(find("keys")), to="campus"))
```

Writing it out using pipes give it a more natural (and easier to read) structure:

```
1 find("keys") %>%  
2   start_car() %>%  
3   drive(to="campus") %>%  
4   park()
```

Approaches

All of the following are fine, it comes down to personal preference:

Nested:

```
1 h( g( f(x), y=1), z=1 )
```

Piped:

```
1 f(x) %>%  
2   g(y=1) %>%  
3   h(z=1)
```

Intermediate:

```
1 res = f(x)  
2 res = g(res, y=1)  
3 res = h(res, z=1)
```

What about other arguments?

Sometimes we want to send our results to a function argument other than first one or we want to use the previous result for multiple arguments. In these cases we can refer to the previous result using `..`

```
1 data.frame(a = 1:3, b = 3:1) %>% lm(a~b, data=.)
```

Call:

```
lm(formula = a ~ b, data = .)
```

Coefficients:

(Intercept)	b
4	-1

```
1 data.frame(a = 1:3, b = 3:1) %>% .[[1]]
```

```
[1] 1 2 3
```

```
1 data.frame(a = 1:3, b = 3:1) %>% .[[length(.)]]
```

```
[1] 3 2 1
```

17

The base R pipe

As of R v4.1.0 a native pipe operator was added to the base language in R, it is implemented as `|>`.

```
1 1:10 |> cumsum()
```

```
[1] 1 3 6 10 15 21 28 36 45 55
```

```
1 1:10 |> cumsum() |> mean()
```

```
[1] 22
```

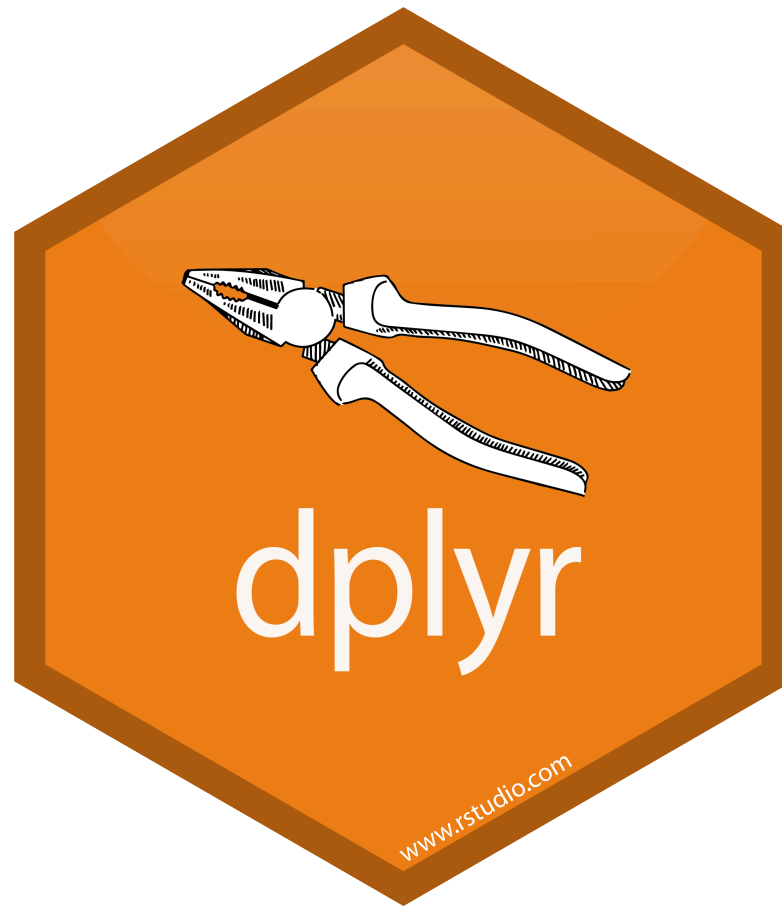
The current version of RStudio on the departmental servers is v4.4.1 so you are welcome to use it.

Base R pipe considerations:

- Depending on an R version ≥ 4.1 is a harder dependency than depending on the `magrittr` package
- `|>` has less overhead than `%>%` but the difference is unlikely to matter in practice most of the time
- `|>` supports an equivalent to `.` using `_` as of R v4.2 (but only for named arguments)

```
1 data.frame(a = 1:3, b = 3:1) |>  
2   lm(a~b, data=_)
```

Generally we will prefer the base pipe in this class, but using either is fine.



A Grammar of Data Manipulation

dplyr is based on the concepts of functions as verbs that manipulate data frames.

Core single data frame functions / verbs:

- `filter()` / `slice()` - pick rows based on criteria
- `select()` / `rename()` - select columns by name
- `pull()` - grab a column as a vector
- `arrange()` - reorder rows
- `mutate()` / `transmute()` - create or modify columns
- `distinct()` - filter for unique rows
- `summarise()` / `count()` - reduce variables to values
- `group_by()` / `ungroup()` - modify other verbs to act on subsets
- `relocate()` - change column order
- ... (many more)

dplyr rules

1. First argument is *always* a data frame
2. Subsequent arguments say what to do with the data frame
3. *Always* return a data frame
4. Don't modify in place
5. Magic via non-standard evaluation + lazy evaluation and S3

Example Data

We will demonstrate dplyr's functionality using the nycflights13 data.

```
1 library(dplyr)
2 library(nycflights13)
```

```
1 flights
```

```
# A tibble: 336,776 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812
6	2013	1	1	554	558	-4	740
7	2013	1	1	555	600	-5	913
8	2013	1	1	557	600	-3	709
9	2013	1	1	557	600	-3	838
10	2013	1	1	558	600	-2	753

```
# i 336,766 more rows
```

filter() - March flights

```
1 flights |> filter(month == 3)
```

```
# A tibble: 28,834 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	3	1	4	2159	125	318
2	2013	3	1	50	2358	52	526
3	2013	3	1	117	2245	152	223
4	2013	3	1	454	500	-6	633
5	2013	3	1	505	515	-10	746
6	2013	3	1	521	530	-9	813
7	2013	3	1	537	540	-3	856
8	2013	3	1	541	545	-4	1014
9	2013	3	1	549	600	-11	639
10	2013	3	1	550	600	-10	717

filter() - Flights in the first 7 days of March

```
1 flights |> filter(month == 3, day <= 7)
```

```
# A tibble: 6,530 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	3	1	4	2159	125	318
2	2013	3	1	50	2358	52	526
3	2013	3	1	117	2245	152	223
4	2013	3	1	454	500	-6	633
5	2013	3	1	505	515	-10	746
6	2013	3	1	521	530	-9	813
7	2013	3	1	537	540	-3	856
8	2013	3	1	541	545	-4	1014
9	2013	3	1	549	600	-11	639
10	2013	3	1	550	600	-10	717

filter() - Flights to LAX or JFK in March

```
1 flights |> filter(dest == "LAX" | dest == "JFK", month==3)
```

```
# A tibble: 1,178 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	3	1	607	610	-3	832
2	2013	3	1	629	632	-3	844
3	2013	3	1	657	700	-3	953
4	2013	3	1	714	715	-1	939
5	2013	3	1	716	710	6	958
6	2013	3	1	727	730	-3	1007
7	2013	3	1	836	840	-4	1111
8	2013	3	1	857	900	-3	1202
9	2013	3	1	903	900	3	1157
10	2013	3	1	004	021	22	1150

slice() - First 10 flights

```
1 flights |> slice(1:10)
```

```
# A tibble: 10 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812
6	2013	1	1	554	558	-4	740
7	2013	1	1	555	600	-5	913
8	2013	1	1	557	600	-3	709
9	2013	1	1	557	600	-3	838
10	2013	1	1	558	600	-2	752

slice() - Last 5 flights

```
1 flights |> slice((n()-4):n())
```

```
# A tibble: 5 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	9	30	NA	1455	NA	NA
2	2013	9	30	NA	2200	NA	NA
3	2013	9	30	NA	1210	NA	NA
4	2013	9	30	NA	1159	NA	NA
5	2013	9	30	NA	840	NA	NA

```
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,  
#   time_hour <dtm>
```

slice_tail() - Last 5 flights

```
1 flights |> slice_tail(n = 5)
```

```
# A tibble: 5 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	9	30	NA	1455	NA	NA
2	2013	9	30	NA	2200	NA	NA
3	2013	9	30	NA	1210	NA	NA
4	2013	9	30	NA	1159	NA	NA
5	2013	9	30	NA	840	NA	NA

```
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,  
#   time_hour <dtm>
```


select() - Individual Columns

```
1 flights |> select(year, month, day)
```

```
# A tibble: 336,776 × 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

select() - Exclude Columns

```
1 flights |> select(-year, -month, -day)
```

```
# A tibble: 336,776 × 16
```

	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	517	515	2	830	819	11
2	533	529	4	850	830	20
3	542	540	2	923	850	33
4	544	545	-1	1004	1022	-18
5	554	600	-6	812	837	-25
6	554	558	-4	740	728	12
7	555	600	-5	913	854	19
8	557	600	-3	709	723	-14
9	557	600	-3	838	846	-8
10	558	600	-2	753	745	8

select() - Ranges

```
1 flights |> select(year:day)
```

```
# A tibble: 336,776 × 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

select() - Exclusion Ranges

```
1 flights |> select(-(year:day))
```

A tibble: 336,776 × 16

	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	517	515	2	830	819	11
2	533	529	4	850	830	20
3	542	540	2	923	850	33
4	544	545	-1	1004	1022	-18
5	554	600	-6	812	837	-25
6	554	558	-4	740	728	12
7	555	600	-5	913	854	19
8	557	600	-3	709	723	-14
9	557	600	-3	838	846	-8
10	558	600	-2	753	745	8

select() - Matching contains()

```
1 flights |> select(contains("dep"), contains("arr"))
```

A tibble: 336,776 × 7

	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	517	515	2	830	819	11
2	533	529	4	850	830	20
3	542	540	2	923	850	33
4	544	545	-1	1004	1022	-18
5	554	600	-6	812	837	-25
6	554	558	-4	740	728	12
7	555	600	-5	913	854	19
8	557	600	-3	709	723	-14
9	557	600	-3	838	846	-8
10	558	600	-2	753	745	8

select() - Matching starts_with()

```
1 flights |> select(starts_with("dep"), starts_with("arr"))
```

A tibble: 336,776 × 4

	dep_time	dep_delay	arr_time	arr_delay
	<int>	<dbl>	<int>	<dbl>
1	517	2	830	11
2	533	4	850	20
3	542	2	923	33
4	544	-1	1004	-18
5	554	-6	812	-25
6	554	-4	740	12
7	555	-5	913	19
8	557	-3	709	-14
9	557	-3	838	-8
10	558	-2	753	8

Other helpers provide by tidyselect:

`starts_with`, `ends_with`, `everything`, `matches`, `num_range`, `one_of`, `everything`, `last_col`.

35

select() + where() - Get numeric columns

```
1 flights |> select(where(is.numeric))
```

```
# A tibble: 336,776 × 14
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812
6	2013	1	1	554	558	-4	740
7	2013	1	1	555	600	-5	913
8	2013	1	1	557	600	-3	709
9	2013	1	1	557	600	-3	838
10	2013	1	1	558	600	-2	753

```
# i 336,766 more rows
```

select() + where() - Get non-numeric columns

```
1 flights |> select(where(function(x) !is.numeric(x)))
```

```
# A tibble: 336,776 × 5
  carrier tailnum origin dest time_hour
  <chr>    <chr>    <chr> <chr> <dtm>
1 UA      N14228  EWR   IAH   2013-01-01 05:00:00
2 UA      N24211  LGA   IAH   2013-01-01 05:00:00
3 AA      N619AA   JFK   MIA   2013-01-01 05:00:00
4 B6      N804JB   JFK   BQN   2013-01-01 05:00:00
5 DL      N668DN   LGA   ATL   2013-01-01 06:00:00
6 UA      N39463   EWR   ORD   2013-01-01 05:00:00
7 B6      N516JB   EWR   FLL   2013-01-01 06:00:00
8 EV      N829AS   LGA   IAD   2013-01-01 06:00:00
9 B6      N593JB   JFK   MCO   2013-01-01 06:00:00
10 AA     N3ALAA   LGA   ORD   2013-01-01 06:00:00
# i 336,766 more rows
```


relocate - to the front

```
1 flights |> relocate(carrier, origin, dest)
```

```
# A tibble: 336,776 × 19
```

	carrier	origin	dest	year	month	day	dep_time	sched_dep_time	dep_delay
	<chr>	<chr>	<chr>	<int>	<int>	<int>	<int>	<int>	<dbl>
1	UA	EWB	IAH	2013	1	1	517	515	2
2	UA	LGA	IAH	2013	1	1	533	529	4
3	AA	JFK	MIA	2013	1	1	542	540	2
4	B6	JFK	BQN	2013	1	1	544	545	-1
5	DL	LGA	ATL	2013	1	1	554	600	-6
6	UA	EWB	ORD	2013	1	1	554	558	-4
7	B6	EWB	FLL	2013	1	1	555	600	-5
8	EV	LGA	IAD	2013	1	1	557	600	-3
9	B6	JFK	MCO	2013	1	1	557	600	-3
10	AA	LGA	ORD	2013	1	1	558	600	-2

relocate - to the end

```
1 flights |> relocate(year, month, day, .after = last_col())
```

```
# A tibble: 336,776 × 19
```

	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	517	515	2	830	819	11
2	533	529	4	850	830	20
3	542	540	2	923	850	33
4	544	545	-1	1004	1022	-18
5	554	600	-6	812	837	-25
6	554	558	-4	740	728	12
7	555	600	-5	913	854	19
8	557	600	-3	709	723	-14
9	557	600	-3	838	846	-8
10	558	600	-2	753	745	8

rename() - Change column names

```
1 flights |> rename(tail_number = tailnum)
```

```
# A tibble: 336,776 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812
6	2013	1	1	554	558	-4	740
7	2013	1	1	555	600	-5	913
8	2013	1	1	557	600	-3	709
9	2013	1	1	557	600	-3	838
10	2013	1	1	558	600	-2	752

select() vs. rename()

```
1 flights |> select(tail_number = tailnum)
```

```
# A tibble: 336,776 × 1
  tail_number
  <chr>
1 N14228
2 N24211
3 N619AA
4 N804JB
5 N668DN
6 N39463
7 N516JB
8 N829AS
9 N593JB
10 N3ALAA
# i 336,766 more rows
```

```
1 flights |> rename(tail_number = tailnum)
```

```
# A tibble: 336,776 × 19
  year month   day dep_time sched_dep_time
  <int> <int> <int>   <int>         <int>
1  2013     1     1     517             515
2  2013     1     1     533             529
3  2013     1     1     542             540
4  2013     1     1     544             545
5  2013     1     1     554             600
6  2013     1     1     554             558
7  2013     1     1     555             600
8  2013     1     1     557             600
9  2013     1     1     557             600
10 2013     1     1     558             600
# i 336,766 more rows
# i 14 more variables: dep_delay <dbl>,
#   arr_time <int>, sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>.
```

pull()

```
1 names(flights)
```

```
[1] "year"          "month"         "day"           "dep_time"  
[5] "sched_dep_time" "dep_delay"     "arr_time"      "sched_arr_time"  
[9] "arr_delay"     "carrier"       "flight"        "tailnum"  
[13] "origin"        "dest"          "air_time"      "distance"  
[17] "hour"          "minute"        "time_hour"
```

```
1 flights |> pull("year") |> head()
```

```
[1] 2013 2013 2013 2013 2013 2013
```

```
1 flights |> pull(1) |> head()
```

```
[1] 2013 2013 2013 2013 2013 2013
```

```
1 flights |> pull(-1) |> head()
```

```
[1] "2013-01-01 05:00:00 EST" "2013-01-01 05:00:00 EST"  
[3] "2013-01-01 05:00:00 EST" "2013-01-01 05:00:00 EST"  
[5] "2013-01-01 06:00:00 EST" "2013-01-01 05:00:00 EST"
```

arrange() - Sort data

```
1 flights |> filter(month==3,day==2) |> arrange(origin, dest)
```

```
# A tibble: 765 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	3	2	1336	1329	7	1426
2	2013	3	2	628	629	-1	837
3	2013	3	2	637	640	-3	903
4	2013	3	2	743	745	-2	945
5	2013	3	2	857	900	-3	1117
6	2013	3	2	1027	1030	-3	1234
7	2013	3	2	1134	1145	-11	1332
8	2013	3	2	1412	1415	-3	1636
9	2013	3	2	1633	1636	-3	1848
10	2013	3	2	1655	1700	-5	1857

arrange() w/ desc() - descending order

```
1 flights |>
2   filter(month==3, day==2) |>
3   arrange(desc(origin), dest) |>
4   select(origin, dest, tailnum)
```

```
# A tibble: 765 × 3
  origin dest  tailnum
  <chr>  <chr> <chr>
1 LGA    ATL   N928AT
2 LGA    ATL   N623DL
3 LGA    ATL   N680DA
4 LGA    ATL   N996AT
5 LGA    ATL   N510MQ
6 LGA    ATL   N663DN
7 LGA    ATL   N942DL
8 LGA    ATL   N511MQ
9 LGA    ATL   N910DE
10 LGA    ATL   N000DE
```

distinct() - Find unique rows

```
1 flights |>
2   select(origin, dest) |>
3   distinct() |>
4   arrange(origin,dest)
```

```
# A tibble: 224 × 2
```

	origin	dest
	<chr>	<chr>
1	EWB	ALB
2	EWB	ANC
3	EWB	ATL
4	EWB	AUS
5	EWB	AVL
6	EWB	BDL
7	EWB	BNA
8	EWB	BOS
9	EWB	BQN
10	EWB	RTV

mutate() - Modify / create columns

```
1 flights |>
2   select(year:day) |>
3   mutate(date = paste(year, month, day, sep="/"))
```

```
# A tibble: 336,776 × 4
   year month   day date
  <int> <int> <int> <chr>
1  2013     1     1 2013/1/1
2  2013     1     1 2013/1/1
3  2013     1     1 2013/1/1
4  2013     1     1 2013/1/1
5  2013     1     1 2013/1/1
6  2013     1     1 2013/1/1
7  2013     1     1 2013/1/1
8  2013     1     1 2013/1/1
9  2013     1     1 2013/1/1
10 2013     1     1 2013/1/1
```

summarise() - Aggregate rows

```
1 flights |>
2   summarize(n(), min(dep_delay), max(dep_delay))
```

```
# A tibble: 1 × 3
  `n()` `min(dep_delay)` `max(dep_delay)`
  <int>      <dbl>      <dbl>
1 336776      NA        NA
```

```
1 flights |>
2   summarize(
3     n = n(),
4     min_dep_delay = min(dep_delay, na.rm = TRUE),
5     max_dep_delay = max(dep_delay, na.rm = TRUE)
6   )
```

```
# A tibble: 1 × 3
  n min_dep_delay max_dep_delay
  <int>      <dbl>      <dbl>
1 336776      -43       1301
```

group_by()

```
1 flights |> group_by(origin)
```

```
# A tibble: 336,776 × 19
```

```
# Groups:   origin [3]
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812
6	2013	1	1	554	558	-4	740
7	2013	1	1	555	600	-5	913
8	2013	1	1	557	600	-3	709
9	2013	1	1	557	600	-3	838

summarise() with group_by()

```
1 flights |>
2   group_by(origin) |>
3   summarize(
4     n = n(),
5     min_dep_delay = min(dep_delay, na.rm = TRUE),
6     max_dep_delay = max(dep_delay, na.rm = TRUE)
7   )
```

A tibble: 3 × 4

	origin	n	min_dep_delay	max_dep_delay
	<chr>	<int>	<dbl>	<dbl>
1	EWR	120835	-25	1126
2	JFK	111279	-43	1301
3	LGA	104662	-33	911

Groups after summarise

```
1 flights |>
2   group_by(origin, month) |>
3   summarize(
4     n = n(),
5     min_dep_delay = min(dep_delay, na.rm=TRUE),
6     max_dep_delay = max(dep_delay, na.rm=TRUE)
7   )
```

`summarise()` has grouped output by 'origin'. You can override using the
`.groups` argument.

A tibble: 36 × 5

Groups: origin [3]

	origin	month	n	min_dep_delay	max_dep_delay
	<chr>	<int>	<int>	<dbl>	<dbl>
1	EWR	1	9893	-21	1126
2	EWR	2	9107	-21	786
3	EWR	3	10420	-22	443
4	EWR	4	10531	-21	545
5	EWR	5	10592	-20	878
6	EWR	6	10175	-19	502
7	EWR	7	10475	-18	653
8	EWR	8	10359	-17	424
9	EWR	9	9550	-23	486
10	EWR	10	10104	-25	702

Avoid the message

```
1 flights |>
2   group_by(origin, month) |>
3   summarize(
4     n = n(),
5     min_dep_delay = min(dep_delay, na.rm=TRUE),
6     max_dep_delay = max(dep_delay, na.rm=TRUE),
7     .groups = "drop"
8   )
```

```
# A tibble: 36 × 5
  origin month      n min_dep_delay max_dep_delay
  <chr>   <int> <int>         <dbl>         <dbl>
1 EWR         1  9893          -21          1126
2 EWR         2  9107          -21           786
3 EWR         3 10420          -22           443
4 EWR         4 10531          -21           545
5 EWR         5 10592          -20           878
6 EWR         6 10175          -19           502
7 EWR         7 10475          -18           653
8 EWR         8 10359          -17           424
9 EWR         9  9550          -23           486
10 EWR        10 10104          -25           702
# i 26 more rows
```

```
1 flights |>
2   group_by(origin, month) |>
3   summarize(
4     n = n(),
5     min_dep_delay = min(dep_delay, na.rm=TRUE),
6     max_dep_delay = max(dep_delay, na.rm=TRUE),
7     .groups = "keep"
8   )
```

```
# A tibble: 36 × 5
# Groups:   origin, month [36]
  origin month      n min_dep_delay max_dep_delay
  <chr>   <int> <int>         <dbl>         <dbl>
1 EWR         1  9893          -21          1126
2 EWR         2  9107          -21           786
3 EWR         3 10420          -22           443
4 EWR         4 10531          -21           545
5 EWR         5 10592          -20           878
6 EWR         6 10175          -19           502
7 EWR         7 10475          -18           653
8 EWR         8 10359          -17           424
9 EWR         9  9550          -23           486
10 EWR        10 10104          -25           702
# i 26 more rows
```

The .by argument

The `.by` (and `by`) arguments are used for per operation grouping while `group_by()` is intended for persistent grouping. See [?dplyr_by](#) for more details and examples.

```
1 flights |>
2   summarize(
3     n = n(),
4     min_dep_delay = min(dep_delay, na.rm=TRUE),
5     max_dep_delay = max(dep_delay, na.rm=TRUE),
6     .by = origin
7   )
```

A tibble: 3 × 4

	origin	n	min_dep_delay	max_dep_delay
	<chr>	<int>	<dbl>	<dbl>
1	EWB	120835	-25	1126
2	LGA	104662	-33	911
3	JFK	111279	-43	1301

count()

```
1 flights |>
2   summarize(
3     n = n(),
4     .by = c(origin, carrier)
5   )
```

```
# A tibble: 35 × 3
  origin carrier      n
  <chr>   <chr>   <int>
1 EWR     UA     46087
2 LGA     UA      8044
3 JFK     AA     13783
4 JFK     B6     42076
5 LGA     DL     23067
6 EWR     B6      6557
7 LGA     EV      8826
8 LGA     AA     15459
9 JFK     UA      4534
10 LGA     B6      6007
```

```
1 flights |>
2   count(origin, carrier)
```

```
# A tibble: 35 × 3
  origin carrier      n
  <chr>   <chr>   <int>
1 EWR     9E      1268
2 EWR     AA     3487
3 EWR     AS       714
4 EWR     B6     6557
5 EWR     DL     4342
6 EWR     EV    43939
7 EWR     MQ     2276
8 EWR     00         6
9 EWR     UA     46087
10 EWR     11C     1105
```

mutate() with .by

```
1 flights |>
2   mutate(n = n(), .by = origin) |>
3   select(origin, n)
```

A tibble: 336,776 × 2

	origin	n
	<chr>	<int>
1	EWB	120835
2	LGA	104662
3	JFK	111279
4	JFK	111279
5	LGA	104662
6	EWB	120835
7	EWB	120835
8	LGA	104662
9	JFK	111279
10	LGA	104662

Exercises / Examples

1. How many flights to Los Angeles (LAX) did each of the legacy carriers (AA, UA, DL or US) have in May from JFK, and what was their average duration?
2. What was the shortest flight out of each airport in terms of distance? In terms of duration?
3. Which plane (check the tail number) flew out of each New York airport the most?
4. Which date should you fly on if you want to have the lowest possible average departure delay? What about arrival delay?