These materials adapted by Amelia McNamara from the RStudio CC BY-SA materials Introduction to R (2014) and Master the Tidyverse (2017).

Introduction to R & RStudio:

deck 07: Transforming data

Amelia McNamara
Visiting Assistant Professor of Statistical and Data Sciences
Smith College

January 2018

dplyr methods for isolating data

select() - extract **variables**
filter() - extract **cases**
arrange() - reorder **cases**

dplyr methods for deriving data

summarise() - summarise **variables**

group_by() - group **cases**

mutate() - create new **variables**

Back to `babynames`

```
library(babynames)
data(babynames)
babynames %>% skim()
```

summarise()

/

British and American spellings recognized! Apologies for my (inevitable) inconsistency

summarize()

summarise()

Compute table of summaries.

```
babynames %>% summarise(total = sum(n), max = max(n))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

total	max
127538	99680

Your Turn

Use summarise() to compute three statistics *about the data*:

1. The first (minimum) year in the dataset
2. The last (maximum) year in the dataset
3. The total number of children represented in the data

```
babynames %>%
  summarise(first = min(year),
            last = max(year),
            total = sum(n))
#   first  last     total
# 1  1880  2015 340851912
```

Your Turn

Extract the rows where **name == "Khaleesi"**. Then use summarise() and a summary functions to find:

1. The total number of children named Khaleesi
2. The first **year** Khaleesi appeared in the data

```r
babynames %>%
  filter(name == "Khaleesi") %>%
  summarise(total = sum(n), first = min(year))
#   total first
# 1  1127  2011
```

Summary functions

Take a vector as input.

Return a single value as output.

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, -, *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

x + y =

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

x
+ y

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

x + y =

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

dplyr

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

n()

The number of rows in a dataset/group

```
babynames %>% summarise(n = n())
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

n
1858689

n_distinct()

The number of distinct values in a variable

```
babynames %>% summarise(n = n(), nname = n_distinct(name))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

n	nname
1858689	95025

Grouping cases

Your Turn

Read in the toy dataset pollution.csv

Your Turn

Read in the toy dataset pollution.csv

pollution

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

```
pollution %>%
 summarise(mean = mean(amount), sum = sum(amount), n = n())
```

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
18.5	37	2
19.0	38	2
88.5	177	2

group_by() + summarise()

group_by()

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	mean	sum	n
New York	18.5	37	2
London	19.0	38	2
Beijing	88.5	177	2

```
pollution %>%
  group_by(city) %>%
  summarise(mean = mean(amount), sum = sum(amount), n = n())
```

group_by()

Groups cases by common values of one or more columns.

```
babynames %>%
  group_by(sex)
```

```
Source: local data frame [1,825,433 x 5]
Groups: sex [2]

     year    sex       name      n        prop
    <dbl> <chr>      <chr> <int>       <dbl>
1   1880      F       Mary  7065 0.07238359
```

group_by()

Groups cases by common values.

```
babynames %>%
  group_by(sex) %>%
  summarise(total = sum(n))
```

sex	total
F	167070477
M	170064949

Your Turn

Use **group_by()**, **summarise()**, and **arrange()** to display the ten most popular baby names. Compute popularity as the total number of children of a single gender given a name.

05:00

```r
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total))
#       name   sex   total
#  1   James     M 5120990
#  2    John     M 5095674
#  3  Robert     M 4803068
#  4 Michael     M 4323928
#  5    Mary     F 4118058
#  6 William     M 4071645
#  7   David     M 3589754
#  8  Joseph     M 2581785
#  9 Richard     M 2558165
# 10 Charles     M 2371621
# ... with 105,376 more rows
```

Your Turn

Use grouping to calculate and then plot the number of children born each year over time.

05:00

```
babynames %>%
  group_by(year) %>%
  summarise(n_children = sum(n)) %>%
  ggplot() +
    geom_line(mapping = aes(x = year, y = n_children))
```

How does this affect our measure of popularity?

ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total))
#       name   sex   total
#  1   James     M 5120990
#  2    John     M 5095674
#  3  Robert     M 4803068
#  4 Michael     M 4323928
#  5    Mary     F 4118058
```

ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%
  group_by(name, sex) %>%
  ungroup() %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total))
#       total
# 1 340851912
```

mutate()

mutate()

Create new columns.

```
babynames %>%
  mutate(percent = round(prop*100, 2))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop	percent
1880	M	John	9655	0.0815	8.15
1880	M	William	9532	0.0805	8.05
1880	M	James	5927	0.0501	5.01
1880	M	Charles	5348	0.0451	4.51
1880	M	Garrett	13	0.0001	0.01
1881	M	John	8769	0.081	8.1

mutate()

Create new columns.

```
babynames %>%
  mutate(percent = round(prop*100, 2), nper = round(percent))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop	percent	nper
1880	M	John	9655	0.0815	8.15	8
1880	M	William	9532	0.0805	8.05	8
1880	M	James	5927	0.0501	5.01	5
1880	M	Charles	5348	0.0451	4.51	5
1880	M	Garrett	13	0.0001	0.01	0
1881	M	John	8769	0.081	8.1	8

Vector functions

Take a vector as input.

Return a vector of the same length as output.

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

x y

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

x
+ y

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

x y

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

dplyr

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

min_rank()

A go to ranking function (ties share the lowest rank)

```
min_rank(c(50, 100, 1000))
# [1] 1 2 3
```

```
min_rank(desc(c(50, 100, 1000)))
# [1] 3 2 1
```

Your Turn

Use **min_rank()** and **mutate()** to rank each row in babynames from largest **n** to lowest **n**.

```
babynames %>%
  mutate(rank = min_rank(desc(prop)))
#     year   sex      name     n       prop  rank
#  1  1880     F      Mary  7065 0.07238433    14
#  2  1880     F      Anna  2604 0.02667923   709
#  3  1880     F      Emma  2003 0.02052170  1131
#  4  1880     F Elizabeth  1939 0.01986599  1192
#  5  1880     F    Minnie  1746 0.01788861  1427
#  6  1880     F  Margaret  1578 0.01616737  1683
#  7  1880     F       Ida  1472 0.01508135  1897
#  8  1880     F     Alice  1414 0.01448711  2040
#  9  1880     F    Bertha  1320 0.01352404  2279
# 10  1880     F     Sarah  1288 0.01319618  2387
# ... with 1,858,679 more rows
```

Your Turn

Compute each name's rank *within its year and sex*.

Then compute the median rank for each combination of *name and sex*, and arrange the results from highest median rank to lowest.

```r
babynames %>%
  group_by(year, sex) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  group_by(name, sex) %>%
  summarise(score = median(rank)) %>%
  arrange(score)
#        name   sex score
#  1      Mary     F     1
#  2     James     M     3
#  3      John     M     3
#  4   William     M     4
#  5    Robert     M     6
# ... with 105,381 more rows
```

Recap: Single table verbs

Extract variables with **select()**

Extract cases with **filter()**

Arrange cases, with **arrange()**.

Make tables of summaries with **summarise()**.

Make new variables, with **mutate()**.

Joining Datasets

nycflights13

Data about every flight that departed La Guardia, JFK, or Newark airports in 2013

```
# install.packages("nycflights13")
library(nycflights13)
```

Flights

```
View(flights)
```

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
2013	1	1	517	515	2	830	819	11	UA	1545	N14228	EWR	IAH	227	1400	5	15	2013-01-01 05:00:00
2013	1	1	533	529	4	850	830	20	UA	1714	N24211	LGA	IAH	227	1416	5	29	2013-01-01 05:00:00
2013	1	1	542	540	2	923	850	33	AA	1141	N619AA	JFK	MIA	160	1089	5	40	2013-01-01 05:00:00
2013	1	1	544	545	-1	1004	1022	-18	B6	725	N804JB	JFK	BQN	183	1576	5	45	2013-01-01 05:00:00
2013	1	1	554	600	-6	812	837	-25	DL	461	N668DN	LGA	ATL	116	762	6	0	2013-01-01 06:00:00
2013	1	1	554	558	-4	740	728	12	UA	1696	N39463	EWR	ORD	150	719	5	58	2013-01-01 05:00:00
2013	1	1	555	600	-5	913	854	19	B6	507	N516JB	EWR	FLL	158	1065	6	0	2013-01-01 06:00:00
2013	1	1	557	600	-3	709	723	-14	EV	5708	N829AS	LGA	IAD	53	229	6	0	2013-01-01 06:00:00
2013	1	1	557	600	-3	838	846	-8	B6	79	N593JB	JFK	MCO	140	944	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	753	745	8	AA	301	N3ALAA	LGA	ORD	138	733	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	849	851	-2	B6	49	N793JB	JFK	PBI	149	1028	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	853	856	-3	B6	71	N657JB	JFK	TPA	158	1005	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	924	917	7	UA	194	N29129	JFK	LAX	345	2475	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	923	937	-14	UA	1124	N53441	EWR	SFO	361	2565	6	0	2013-01-01 06:00:00
2013	1	1	559	600	-1	941	910	31	AA	707	N3DUAA	LGA	DFW	257	1389	6	0	2013-01-01 06:00:00
2013	1	1	559	559	0	702	706	-4	B6	1806	N708JB	JFK	BOS	44	187	5	59	2013-01-01 05:00:00
2013	1	1	559	600	-1	854	902	-8	UA	1187	N76515	EWR	LAS	337	2227	6	0	2013-01-01 06:00:00
2013	1	1	600	600	0	851	858	-7	B6	371	N595JB	LGA	FLL	152	1076	6	0	2013-01-01 06:00:00

nycflights13

nycflights13

What airline had the longest delays?

Airline names

```
View(flights["carrier"])
```

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6

```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.

types of joins

types of joins

Mutating joins use information from one data set **to add variables** to another data set (like **mutate()**)

Filtering joins use information from one data set **to extract cases** from another data set (like **filter()**)

We're only going to talk about these

common syntax

Each join function returns a data frame / tibble.

```
left_join(x, y, by = NULL, … )
```

join function

data frames to join

names of columns to join on

Your Turn

Read in the toy datasets `band` and `instrument`

Toy data

```r
band <- data.frame(
    name = c("Mick", "John", "Paul"),
    band = c("Stones", "Beatles", "Beatles"))
```

```r
band <- tribble(
    ~name,     ~band,
   "Mick",  "Stones",
   "John", "Beatles",
   "Paul", "Beatles"
)
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

```r
instrument <- data.frame(
    name = c("John", "Paul", "Keith"),
     plays = c("guitar", "bass", "guitar"))
```

```r
instrument <- tribble(
    ~name,   ~plays,
   "John", "guitar",
   "Paul",   "bass",
  "Keith", "guitar"
)
```

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

Toy data

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

left

```
band %>% left_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

right

```
band %>% right_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

full

```
band %>% full_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

inner

```
band %>% inner_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

Airline names

```
View(flights["carrier"])
```

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6

```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.

Your Turn

Which airlines had the largest arrival delays? Work in groups to complete the code below.

```
flights %>%
  drop_na(arr_delay) %>%
  ______________________ %>%
  group_by(_________) %>%
  ______________________ %>%
  arrange(_______)
```

1. Join airlines to flights

2. Compute and order the average arrival delays by airline. Display full names, no codes.

```r
flights %>%
  drop_na(arr_delay) %>%
  left_join(airlines, by = "carrier") %>%
  group_by(name) %>%
  summarise(delay = mean(arr_delay)) %>%
  arrange(delay)
```

```
## # A tibble: 16 × 2
##                          name      delay
##                         <chr>      <dbl>
## 1         Alaska Airlines Inc. -9.9308886
## 2       Hawaiian Airlines Inc. -6.9152047
## 3       American Airlines Inc.  0.3642909
## 4         Delta Air Lines Inc.  1.6443409
## 5               Virgin America  1.7644644
```

Your Turn

Read in the toy dataset `instrument2`

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument2

artist	plays
John	guitar
Paul	bass
Keith	guitar

What if the names do not match?

Use a named vector to match on variables with different names.

```
band %>% left_join(instrument2, by = c("name" = "artist"))
```

A named vector

The name of the element = the column name in the first data set

The value of the element = the column name in the second data set

common syntax - matching names

```
band %>% left_join(instrument2, by = c("name" = "artist"))
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument2

artist	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

Airport names

```
View(flights["dest"])
```

	dest
1	IAH
2	IAH
3	MIA
4	BQN
5	ATL
6	ORD

```
View(airports)
```

faa	name
04G	Lansdowne Airport
06A	Moton Field Municipal Airport
06C	Schaumburg Regional
06N	Randall Airport
09J	Jekyll Island Airport
0A9	Elizabethton Municipal Airport

Airport names

```
airports %>% select(1:3)
```

faa <chr>	name <chr>
04G	Lansdowne Airport
06A	Moton Field Municipal Airport
06C	Schaumburg Regional
06N	Randall Airport
09J	Jekyll Island Airport
0A9	Elizabethton Municipal Airport
0G6	Williams County Airport
0G7	Finger Lakes Regional Airport

```
flights %>% select(14:15)
```

dest <chr>	air_time <dbl>
IAH	227
IAH	227
MIA	160
BQN	183
ATL	116
ORD	150
FLL	158
IAD	53

common syntax - matching names

```
airports %>% left_join(flights, by = c("faa" = "dest"))
```

faa <chr>	name <chr>
04G	Lansdowne Airport
06A	Moton Field Municipal Airport
06C	Schaumburg Regional
06N	Randall Airport
09J	Jekyll Island Airport
0A9	Elizabethton Municipal Airport
0G6	Williams County Airport
0G7	Finger Lakes Regional Airport

dest <chr>	air_time <dbl>
IAH	227
IAH	227
MIA	160
BQN	183
ATL	116
ORD	150
FLL	158
IAD	53

Recap: Two table verbs

left_join() retains all cases in **left** data set

right_join() retains all cases in **right** data set

full_join() retains all cases in **either** data set

inner_join() retains only cases in **both** data sets

semi_join() extracts cases that **have a match**

anti_join() extracts cases that **do not have a match**

Two table verbs

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , * , / , ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.