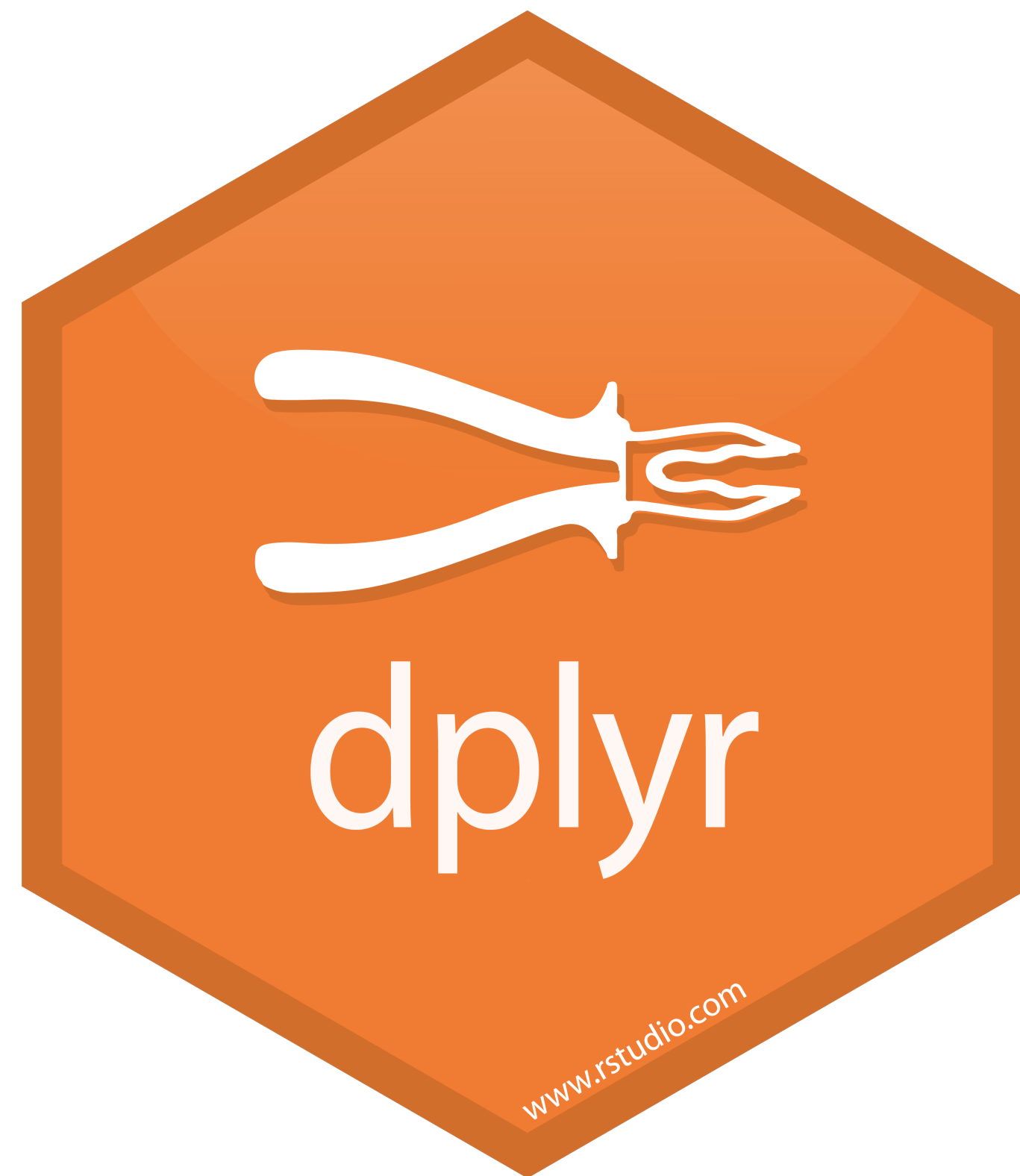


Transform Data with



gapminder



A subset of Gapminder data: population, GDP per capita and life expectancy, for countries over time.

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



Your Turn 0

04-Transform-data.Rmd

Run the setup chunk

```
install.packages("gapminder")  
library(gapminder)
```

04:00

gapminder

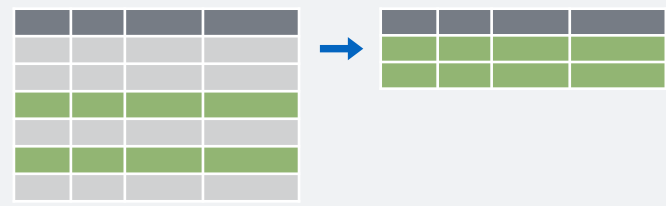
country <fctr>	continent <fctr>	year <int>	lifeExp <dbl>	pop <int>	gdpPercap <dbl>
Afghanistan	Asia	1952	28.80100	8425333	779.4453
Afghanistan	Asia	1957	30.33200	9240934	820.8530
Afghanistan	Asia	1962	31.99700	10267083	853.1007
Afghanistan	Asia	1967	34.02000	11537966	836.1971
Afghanistan	Asia	1972	36.08800	13079460	739.9811
Afghanistan	Asia	1977	38.43800	14880372	786.1134
Afghanistan	Asia	1982	39.85400	12881816	978.0114
Afghanistan	Asia	1987	40.82200	13867957	852.3959
Afghanistan	Asia	1992	41.67400	16317921	649.3414
Afghanistan	Asia	1997	41.76300	22227415	635.3414

1-10 of 1,704 rows

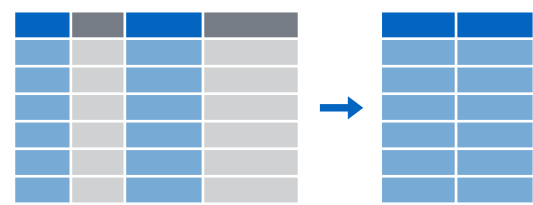
Previous 1 2 3 4 5 6 ... 100 Next



dplyr: Data manipulation verbs



Extract cases with **filter()**



Extract variables with **select()**



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



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



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

along with **group_by()**



filter()

filter()

Extract rows that meet logical criteria.

```
filter(.data, ... )
```

**data frame to
transform**

one or more logical tests
(filter returns each row for
which the test is TRUE)



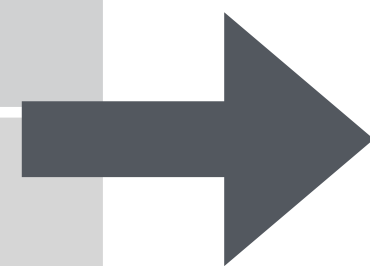
filter()

Extract rows that meet logical criteria.

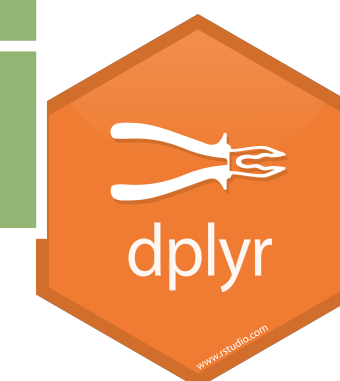
```
filter(gapminder, country == "New Zealand")
```

gapminder

country	continent	year	...
Afghanistan	Asia	1952	
Afghanistan	Asia	1957	
...	
Netherlands	Europe	2007	
New Zealand	Oceania	1952	
New Zealand	Oceania	1957	



country	continent	year	...
New Zealand	Oceania	1952	
New Zealand	Oceania	1957	
New Zealand	Oceania	1962	
New Zealand	Oceania	1967	
...



filter()

Extract rows that meet logical criteria.

```
filter(gapminder, country == "New Zealand")
```

gapminder

country	continent	year	...
Afghanistan	Asia	1952	
Afghanistan	Asia	1957	
...	
Netherlands	Europe	2007	
New Zealand	Oceania	1952	
New Zealand	Oceania	1957	

= sets
(returns nothing)

== tests if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

<code>x < y</code>	Less than
<code>x > y</code>	Greater than
<code>x == y</code>	Equal to
<code>x <= y</code>	Less than or equal to
<code>x >= y</code>	Greater than or equal to
<code>x != y</code>	Not equal to
<code>x %in% y</code>	Group membership
<code>is.na(x)</code>	Is NA
<code>!is.na(x)</code>	Is not NA



Your Turn 1

04-Transform-data.Rmd

See if you can use the logical operators to manipulate our code below to show:

1. The data for Canada
2. All data for countries in Oceania
3. Rows where the life expectancy is greater than 82

04:00

```
filter(gapminder, country == "Canada")
```

```
filter(gapminder, continent == "Oceania")
```

```
filter(gapminder, lifeExp > 82)
```

Two common mistakes

1. Using `=` instead of `==`

```
filter(gapminder, continent = "Oceania")  
filter(gapminder, continent == "Oceania")
```

2. Forgetting quotes

```
filter(gapminder, continent == Oceania)  
filter(gapminder, continent == "Oceania")
```



filter()

Extract rows that meet *every* logical criteria.

```
filter(gapminder, country == "New Zealand", year > 2000)
```

gapminder								
country	continent	year	...		country	continent	year	...
...			New Zealand	Oceania	2002	
New Zealand	Oceania	1952			New Zealand	Oceania	2007	
...						
New Zealand	Oceania	2002						
New Zealand	Oceania	2007						

Boolean operators

?base::Logic

<code>a & b</code>	and
<code>a b</code>	or
<code>!a</code>	not



filter()

Extract rows that meet *every* logical criteria.

```
filter(gapminder, country == "New Zealand" & year > 2000)
```

gapminder

country	continent	year	...
...	
New Zealand	Oceania	1952	
...	
New Zealand	Oceania	2002	
New Zealand	Oceania	2007	

country	continent	year	...
New Zealand	Oceania	2002	
New Zealand	Oceania	2007	

Your Turn 2

Use Boolean operators to alter the code below to return only the rows that contain:

1. Canada before 1970
2. Countries where life expectancy in 2007 is below 50
3. Countries where life expectancy in 2007 is below 50, and are not in Africa.

A digital timer with a black border and a white background, displaying the time 04:00 in a large, black, digital font.

```
filter(gapminder, country == "Canada", year < 1970)
```

```
filter(gapminder, year == 2007, lifeExp < 50)
```

```
filter(gapminder, year == 2007, lifeExp < 50, !(continent == "Africa"))
```

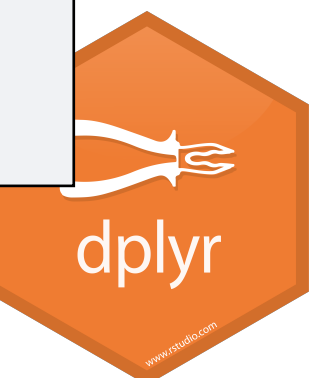

Two more common mistakes

3. Collapsing multiple tests into one

```
filter(gapminder, 1960 < year < 1980)  
filter(gapminder, 1960 < year, year < 1980)
```

4. Stringing together many tests (when you could use %in%)

```
filter(gapminder, country == "New Zealand" |  
  country == "Canada" | country == "United States")  
filter(gapminder,  
  country %in% c("New Zealand", "Canada", "United States"))
```



common syntax

Each function takes a data frame / tibble as its first argument and returns a data frame / tibble.

```
filter(.data, ...)
```

dplyr function

**data frame to
transform**

**function specific
arguments**



mutate()

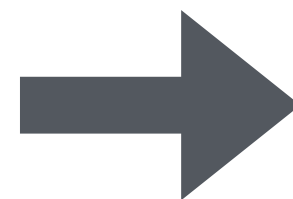
mutate()

Create new columns.

```
mutate(gapminder, gdp = gdpPercap * pop)
```

gapminder

country	continent	year	...
Afghanistan	Asia	1952	
Afghanistan	Asia	1957	
Afghanistan	Asia	1962	
Afghanistan	Asia	1967	
Afghanistan	Asia	1972	
Afghanistan	Asia	1977	



country	continent	year	...	gdp
Afghanistan	Asia	1952		6567086330
Afghanistan	Asia	1957		7585448670
Afghanistan	Asia	1962		8758855797
Afghanistan	Asia	1967		9648014150
Afghanistan	Asia	1972		9678553274
Afghanistan	Asia	1977		11697659231



mutate()

Create new columns.

```
mutate(gapminder, gpd = gdpPercap * pop)
```

dplyr function

data frame to
transform

function specific
arguments

gapminder				...	gdp
Afghanistan	Asia	1957			6567086330
Afghanistan	Asia	1962			7585448670
Afghanistan	Asia	1967			8758855797
Afghanistan	Asia	1972			9648014150
Afghanistan	Asia	1977			9678553274
Afghanistan	Asia	1977			11697659231



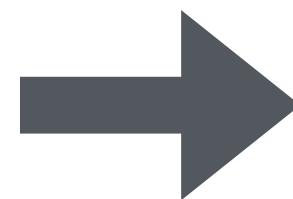
mutate()

Create new columns.

```
mutate(gapminder, gdp = gdpPercap * pop,  
        pop_mill = round(pop/1000000))
```

gapminder

country	continent	year	...
Afghanistan	Asia	1952	
Afghanistan	Asia	1957	
Afghanistan	Asia	1962	
Afghanistan	Asia	1967	
Afghanistan	Asia	1972	
Afghanistan	Asia	1977	



country	continent	year	...	gdp	pop_mill
Afghanistan	Asia	1952		6567086330	8
Afghanistan	Asia	1957		7585448670	9
Afghanistan	Asia	1962		8758855797	10
Afghanistan	Asia	1967		9648014150	12
Afghanistan	Asia	1972		9678553274	13
Afghanistan	Asia	1977		11697659231	15

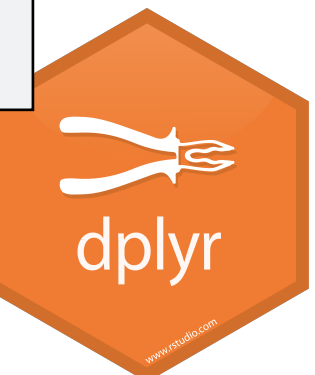
round()

Round a number to a specified number of decimal digits (0 by default)

```
x <- c(1.2, 1/3, 10.01)
round(x)
[1] 1 0 10
round(x, digits = 2)
[1] 1.20 0.33 10.01
```

When used in `mutate()`
the argument will be a
column name

```
mutate(gapminder, gpd = gdpPercap * pop,
       pop_mill = round(pop/1000000))
```



Vectorized functions

Take a vector as input.

Return a vector of the same length as output.

Vectorized 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() - $x \geq \text{left} \ \& \ x \leq \text{right}$
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

Vectorized 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() - $x \geq \text{left} \ \& \ x \leq \text{right}$
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() - n 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() - mean 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(fis, var = "r")`

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

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).

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.

Most useful:

- Math
- Misc



min_rank()

A goto 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 3

Add an `africa` column, which contains TRUE is the country is on the Africa continent.

Add a `rank_pop` column to rank each row in `gapminder` from largest `pop` to smallest `pop`.

02:00


```
mutate(gapminder,  
  africa = continent == "Africa",  
  rank_pop = min_rank(desc(pop)))
```

```
## # A tibble: 1,704 x 8
```

```
##      country continent  year lifeExp      pop gdpPercap africa rank_pop  
##      <fctr>      <fctr> <int>   <dbl>    <int>    <dbl>   <lgl>    <int>  
## 1 Afghanistan      Asia  1952  28.801  8425333  779.4453 FALSE     762  
## 2 Afghanistan      Asia  1957  30.332  9240934  820.8530 FALSE     706  
## 3 Afghanistan      Asia  1962  31.997 10267083  853.1007 FALSE     638  
## 4 Afghanistan      Asia  1967  34.020 11537966  836.1971 FALSE     576  
## 5 Afghanistan      Asia  1972  36.088 13079460  739.9811 FALSE     536
```

$0\% > 0\%$

Multistep Operations

Consider the following:

Add a new column for rank, **then**

Extract rows with rank lower than 10

Use intermediate variables:

```
gapminder_ranked <- mutate(gapminder,  
                             rank = min_rank(desc(pop)))  
filter(gapminder_ranked, rank < 10)
```

Multistep Operations

Consider the following:

Add a new column for rank, **then**

Extract rows with rank lower than 10

Do it all in one line:

```
filter(mutate(gapminder,  
              rank = min_rank(desc(pop))),  
       rank < 10)
```

Multistep Operations

Consider the following:

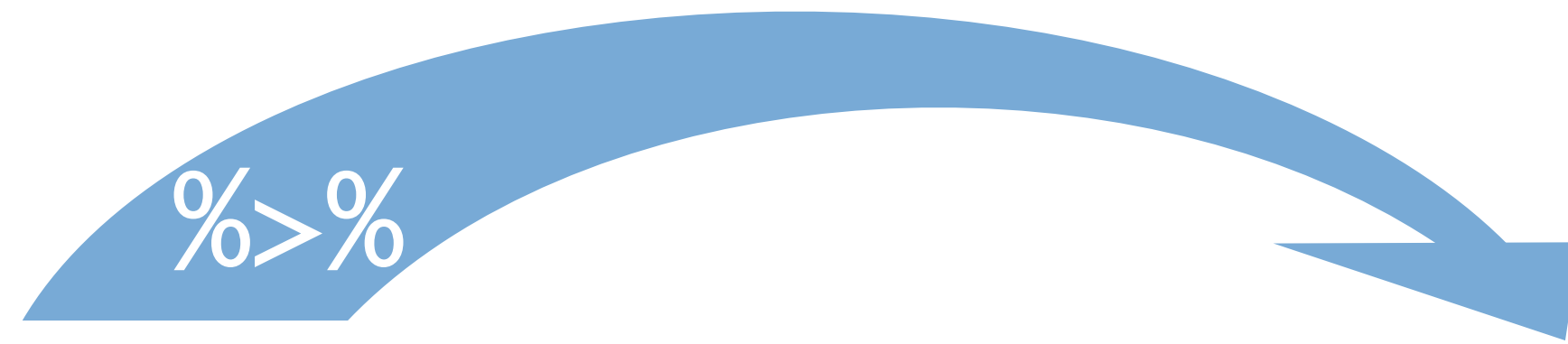
Add a new column for rank, **then**

Extract rows with rank lower than 10

Do it all in one line:

```
filter(mutate(gapminder, rank = min_rank(desc(pop))),  
rank < 10)
```


The pipe operator %>%



gapminder filter(_____, country == "Canada")

Passes result on left into first argument of function on right.
So, for example, these do the same thing. Try it.

```
filter(gapminder, country == "Canada")  
gapminder %>% filter(country == "Canada")
```



Pipes

```
gapminder_ranked <- mutate(gapminder,  
                           rank = min_rank(desc(pop)))  
filter(gapminder_ranked, rank < 10)
```

```
gapminder %>%
```

```
  mutate(rank = min_rank(desc(prop))) %>%
```

```
  filter(rank < 10)
```

Consider the following:

Add a new column for rank, **then**

Extract rows with rank lower than 10

Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)



summarise()

summarise()

Compute table of summaries.

```
gapminder %>% summarise(mean_life = mean(lifeExp))  
summarise(gapminder, mean_life = mean(lifeExp))
```

gapminder

country	continent	year	lifeExp	...
Afghanistan	Asia	1952	28.801	
Afghanistan	Asia	1957	30.332	
Afghanistan	Asia	1962	31.997	
Afghanistan	Asia	1967	34.020	
Afghanistan	Asia	1972	36.088	



mean_life
59.47444



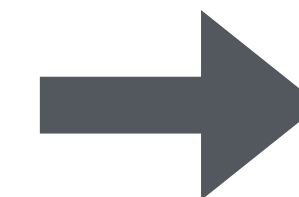
summarise()

Compute table of summaries.

```
gapminder %>% summarise(mean_life = mean(lifeExp),  
                        min_life = min(lifeExp))
```

gapminder

country	continent	year	lifeExp	...
Afghanistan	Asia	1952	28.801	
Afghanistan	Asia	1957	30.332	
Afghanistan	Asia	1962	31.997	
Afghanistan	Asia	1967	34.020	
Afghanistan	Asia	1972	36.088	



mean_life	min_life
59.47444	23.599



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() - mean absolute deviation
sd() - standard deviation
var() - variance

Vectorized 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::cumc_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() - $x \geq \text{left} \ \& \ x \leq \text{right}$
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 un-ques
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() - mean absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which stores 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(fis, var = "r")`

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

Also has **rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

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"))`

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

bind_rows() returns tables placed below each other as they are.

Use **bind_rows(..., id = NULL)** to paste tables 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, ...)
Flows that appear in both x and z.

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

union(x, y, ...)
Rows that appear in x or z. (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. © 2017 RStudio, Inc. All rights reserved. 844-645-1712 | r-studio.com | Learn more with interactive guides (package = "tidyverse", lib = "tidyverse") | dplyr: 0.5.0 | slide 120 | Updated: 2017-01



Your Turn 4

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 number of countries represented in the data (Hint: use cheatsheet)

03:00

```
gapminder %>%  
  summarise(first = min(year),  
            last = max(year),  
            n_countries = n_distinct(country))
```

```
# A tibble: 1 x 3  
#   first last n_countries  
#   <dbl> <dbl>     <int>  
# 1  1952  2007         142
```



Your Turn 5

Extract the rows where **continent == "Africa"** and **year == 2007**.

Then use `summarise()` and summary functions to find:

1. The number of unique countries
2. The median life expectancy

03:00


```
gapminder %>%  
  filter(continent == "Africa", year == 2007) %>%  
  summarise(n_countries = n_distinct(country), med_le = median(lifeExp))  
  
## A tibble: 1 x 2  
#   n_countries med_life_exp  
#       <int>       <dbl>  
#1         52       52.9265
```



Grouping cases

group_by()

Groups cases by common values of one or more columns.

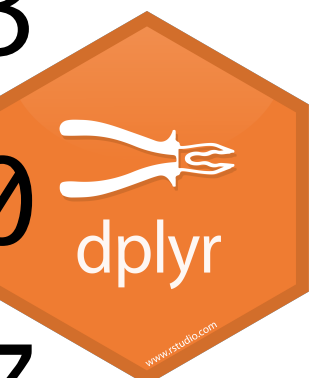
```
gapminder %>%  
  group_by(continent)
```

In console

```
# A tibble: 1,704 x 6
```

```
# Groups:   continent [5]
```

	country	continent	year	lifeExp	pop	gdpPercap
	<fctr>	<fctr>	<int>	<dbl>	<int>	<dbl>
1	Afghanistan	Asia	1952	28.801	8425333	779.4453
2	Afghanistan	Asia	1957	30.332	9240934	820.8530
3	Afghanistan	Asia	1962	31.997	10267083	853.1007



group_by()

Groups cases by common values, then summarise acts by group

```
gapminder %>%  
  group_by(continent) %>%  
  summarise(n_countries = n_distinct(country))
```

continent	n_countries
Africa	52
Americas	25
Asia	33
Europe	30
Oceania	2



Manipulate Data Notebook

```
01-manipulate-data.Rmd
1 |---
2 |title: "Manipulate Data"
3 |output: html_document
4 |---
5 |
6 |```{r setup}
7 |library(tidyverse)
8 |library(babynames)
9 |
10 |# Toy dataset to use
11 |pollution <- tribble(
12 |   ~city, ~size, ~amount,
13 |   "New York", "large", 23,
14 |   "New York", "small", 14,
15 |   "London", "large", 22,
16 |   "London", "small", 16,
17 |   "Beijing", "large", 121,
18 |   "Beijing", "small", 56
19 | )
20 |
21 |
22 |## babynames
23 |
24 |```{r}
25 |babynames
```

```
pollution <- tribble(
  ~city, ~size, ~amount,
  "New York", "large", 23,
  "New York", "small", 14,
  "London", "large", 22,
  "London", "small", 16,
  "Beijing", "large", 121,
  "Beijing", "small", 56
)
```

Toy data set to
practice with




```

pollution <- tribble(
  ~city, ~size, ~amount,
  "New York", "large", 23,
  "New York", "small", 14,
  "London", "large", 22,
  "London", "small", 16,
  "Beijing", "large", 121,
  "Beijing", "small", 56
)

```

pollution

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



city	particle size	amount ($\mu\text{g}/\text{m}^3$)
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 ($\mu\text{g}/\text{m}^3$)
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 ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14



mean	sum	n
18.5	37	2

London	large	22
London	small	16



19.0	38	2
------	----	---

Beijing	large	121
Beijing	small	56

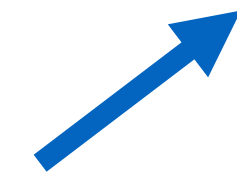


88.5	177	2
------	-----	---

`group_by() + summarise()`

group_by()

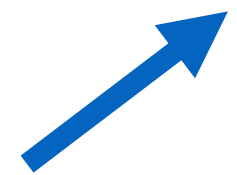
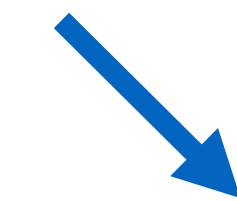
city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



city	particle size	amount ($\mu\text{g}/\text{m}^3$)
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())
```


Your Turn 6

Find the median life expectancy by continent

05:00

```
gapminder %>%  
  group_by(continent) %>%  
  summarise(med_life_exp = median(LifeExp))
```

```
# # A tibble: 5 x 2  
#   continent med_life_exp  
#   <fctr>      <dbl>  
# 1 Africa      47.7920  
# 2 Americas    67.0480  
# 3 Asia        61.7915  
# 4 Europe      72.2410  
# 5 Oceania     73.6650
```

Final task

I want to find the country with **biggest jump** in life expectancy (between any two consecutive records) for each continent.

lag()

Gives previous (based on order in data) value

```
lag(c(1, 2, 3))  
# [1] NA  1  2
```


Your Turn 7

Brainstorm with your neighbour

What sequence of operations would you need to do?

05:00

Your Turn 8

Putting it all together

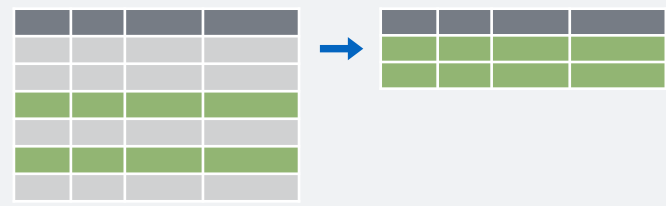
Find the country with biggest jump in life expectancy (between any two consecutive records) for each continent.

05:00

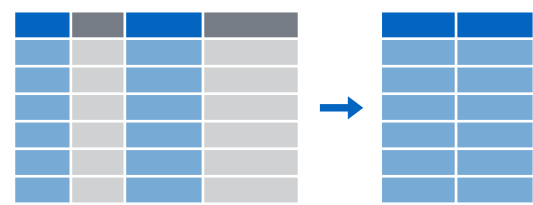

```
# One of many solutions
gapminder %>%
  group_by(country) %>%
  mutate(jump = lifeExp - lag(lifeExp)) %>%
  group_by(continent) %>%
  mutate(rank = min_rank(desc(jump))) %>%
  filter(rank == 1)
```



dplyr: Data manipulation verbs



Extract cases with **filter()**



Extract variables with **select()**



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



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



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

along with **group_by()**

