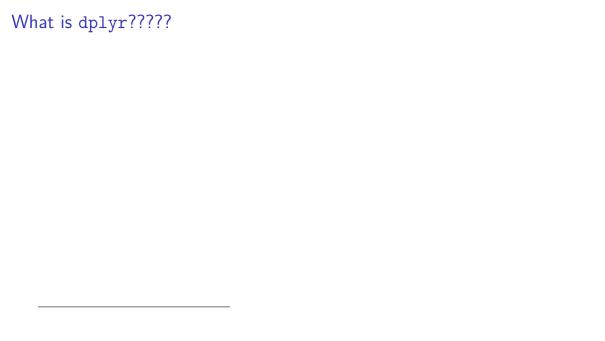
R Workshop Featuring dplyr

Jorge Loría

October 20th, 2020

Where can we find this presentation?

https://github.com/Jels95/Dplyr-Workshop



What is	dplyr?????
---------	------------

Dplyr is a package that permits a nice interface in R to work with data.frames.

What is dplyr?????

Dplyr is a package that permits a *nice* interface in R to work with data.frames.

Ok, nice, but what is a data.frame?

¹categorical

²https://vita.had.co.nz/papers/tidy-data.pdf

What is dplyr?????

Dplyr is a package that permits a *nice* interface in R to work with data.frames.

Ok, nice, but what is a data.frame?

A data frame is like a matrix of dimensions $n \times p$, where we have several different types of data. Each column corresponds to a single variable, and each variable has a specific type (numeric, string, logical, factor¹). Each row should correspond to a single observation².

¹categorical

²https://vita.had.co.nz/papers/tidy-data.pdf

What dataset are we going to use?

We are going to use a dataset from tidytuesday, about Himalayan Climbers

What dataset are we going to use?

We are going to use a dataset from tidytuesday, about Himalayan Climbers



Figure 1: Himalaya? It must be easy to survive there

How to access it?

```
install.packages('tidytuesdayR')
library(tidytuesdayR)
himalaya <- tidytuesdayR::tt_load('2020-09-22')
members <- himalaya$members</pre>
```

Let's start with questions

What questions can we ask this dataset that you think would be interesting to know? (Please write them in the chat!)

Let's start with questions

What questions can we ask this dataset that you think would be interesting to know? (Please write them in the chat!)

The idea is to be able to solve most, if not all of those questions by the end of the workshop!

Let's start with questions

What questions can we ask this dataset that you think would be interesting to know? (Please write them in the chat!)

The idea is to be able to solve most, if not all of those questions by the end of the workshop!

We will study some tools, and you should be able to answer them by the end of the workshop. If I can see that's not possible, I'll tell you what libraries are good to answer them. Ok?

Let's take a look:

```
members %>%
head()
```

%>%

This is a function that doesn't do much, but does a lot. It allows to compose functions (as the math people do) but in a way that permits an easy reading of the functions, and what is happening.

%>%

This is a function that doesn't do much, but does a lot. It allows to compose functions (as the math people do) but in a way that permits an easy reading of the functions, and what is happening. So, instead of writing: f(g(h(i(x))))

%>%

This is a function that doesn't do much, but does a lot. It allows to compose functions (as the math people do) but in a way that permits an easy reading of the functions, and what is happening. So, instead of writing: f(g(h(i(x)))), we write:

```
x %>%
i() %>%
h() %>%
g() %>%
f()
```

Which is arguably easier to read than the previous expression. Specially if some of those functions had extra arguments.

There are 6 main verbs in dplyr that we will study:

function	action
filter arrange	keeps rows that satisfy a condition sorts the rows following the order
select mutate	keeps/eliminates the columns by name creates new variables from existing variables
summarise	summarises the data
groub_by*	groups under specific conditions

All the verbs work very similarly (in dplyr, in English verbs are more confusing):

▶ first argument is a dataframe,

- first argument is a dataframe,
- arguments describe what to do with the dataframe, using the existing variables (without quotations).

- first argument is a dataframe,
- arguments describe what to do with the dataframe, using the existing variables (without quotations).
- output is a dataframe

- first argument is a dataframe,
- arguments describe what to do with the dataframe, using the existing variables (without quotations).
- output is a dataframe

All the verbs work very similarly (in dplyr, in English verbs are more confusing):

- first argument is a dataframe,
- arguments describe what to do with the dataframe, using the existing variables (without quotations).
- output is a dataframe

This structure allows us to concatenate (%>%) simple operations to obtain complex results.



Figure 2: Another type of filters

This functions removes rows that don't satisfy a (or several) condition that we specify. The arguments it receives are logical, and will use it to do that removal:

```
library(dplyr) ## load the library
members %>%
filter(oxygen_used)
```

We can use several columns to filter, and can even modify them. Let's see what people older than 75 years **didn't** need to use oxygen

```
members %>%
filter(!oxygen_used,age > 75)
```

We can also use several columns at once to do a filter. Let's see what climber(s?) died a little bit after getting injured:

```
members %>%
filter(death_height_metres > injury_height_metres)
```

▶ From those who where not injured, how many died in 1979?

- From those who where not injured, how many died in 1979?
- ► How many of those that succeeded, were doing it solo and got injured above 1000 meters?

- From those who where not injured, how many died in 1979?
- ► How many of those that succeeded, were doing it solo and got injured above 1000 meters?
- ▶ How many people died on their climb below 1000 meters?

- From those who where not injured, how many died in 1979?
- ► How many of those that succeeded, were doing it solo and got injured above 1000 meters?
- ▶ How many people died on their climb below 1000 meters?

- From those who where not injured, how many died in 1979?
- ► How many of those that succeeded, were doing it solo and got injured above 1000 meters?
- ▶ How many people died on their climb below 1000 meters?

filter saving

Let's now save the dataframe of the members that went on a expedition in 1905:

```
members1905 <- members %>%
filter(year == 1905)
```



arrange



Figure 3: Another type of arrangement



arrange

This verb sorts the data frame with the column(s) that we tell it to use

```
members %>%
arrange(year)
```

arrange on characters

Let's go back to the 1905 dataset, and check how it orders when we use a string, instead of a number:

```
members1905 %>%
arrange(member_id)
```

arrange uses the lexicographic order, to sort when it encounters letters/characters, so: $a < A < b < B \cdots < z < Z$. If you're not sure if a number or a symbol is smaller than another one, you can try it out on the 'Console' in R for example type '1'<'a'. Is this true? Or false?

arrange uses the lexicographic order, to sort when it encounters letters/characters, so: $a < A < b < B \cdots < z < Z$. If you're not sure if a number or a symbol is smaller than another one, you can try it out on the 'Console' in R for example type '1'<'a'. Is this true? Or false? TRUE!

arrange uses the lexicographic order, to sort when it encounters letters/characters, so: $a < A < b < B \cdots < z < Z$. If you're not sure if a number or a symbol is smaller than another one, you can try it out on the 'Console' in R for example type '1'<'a'. Is this true? Or false? TRUE!How about '&' > '2'?

arrange uses the lexicographic order, to sort when it encounters letters/characters, so: $a < A < b < B \cdots < z < Z$. If you're not sure if a number or a symbol is smaller than another one, you can try it out on the 'Console' in R for example type '1'<'a'. Is this true? Or false? TRUE!How about '&' > '2'?FALSE!

arrange uses the lexicographic order, to sort when it encounters letters/characters, so: $a < A < b < B \cdots < z < Z$. If you're not sure if a number or a symbol is smaller than another one, you can try it out on the 'Console' in R for example type '1'<'a'. Is this true? Or false? TRUE!How about '&' > '2'?FALSE! Is ' ' < '1' (space less than one)?

arrange uses the lexicographic order, to sort when it encounters letters/characters, so: $a < A < b < B \cdots < z < Z$. If you're not sure if a number or a symbol is smaller than another one, you can try it out on the 'Console' in R for example type '1'<'a'. Is this true? Or false? TRUE!How about '&' > '2'?FALSE! Is ' ' < '1' (space less than one)? TRUE!

arrange on several columns

When we include several arguments, it first sorts the first one, then the second one *within* the first order, then the third one *within* the second one, and so on...

```
members %>%
arrange(desc(year),citizenship,hired,peak_name)
```

arrange on transformations

First, let's only take those that got injured, and then see who got furthest away with respect to their injury. Is there anything weird going on? Is my code correct? Is the data correct?

```
members %>%
  filter(injured) %>%
  arrange(highpoint_metres - injury_height_metres)
```



Exercises for arrange:

► From those that died, sort by those who got the highest before dying w.r.t. where they died.

Exercises for arrange:

- ► From those that died, sort by those who got the highest before dying w.r.t. where they died.
- ➤ Sort by year and then season. Does this make sense? If it doesn't, how could we fix it?

Exercises for arrange:

- ► From those that died, sort by those who got the highest before dying w.r.t. where they died.
- ➤ Sort by year and then season. Does this make sense? If it doesn't, how could we fix it?
- ▶ If you arrange by died, can you tell how arrange interprets the logicals?



select

This verbs selects the columns that we want to keep. Sometimes, we only need a couple of variables, and it's good to get rid of the rest:

```
members %>%
select(year,sex,citizenship)
```

select ranges

Sometimes we want to select all columns between two other columns, so we can use the colon (:, not the organ) to do this:

select ranges

Sometimes we want to select all columns between two other columns, so we can use the colon (:, not the organ) to do this:

```
members %>%
select(age:success)
```

Eliminating with select

Other times we want to get rid of specific variables. For this, we can use the – (minus) symbol.

```
members %>%
select(-expedition_id,-member_id,-peak_id)
```

More of select

We can also use the number of the column to indicate which columns to select, and combine it with the names.

```
members %>%
select(4,6:10,highpoint_metres)
```

There's a family of functions designed to work with select, so we can work more easily. Among them we have:

starts_with('some') that looks for all columns that begin with 'some'

- starts_with('some') that looks for all columns that begin with 'some'
- ends_with('thing') that looks for all the columns that start with 'thing'

- starts_with('some') that looks for all columns that begin with 'some'
- ends_with('thing') that looks for all the columns that start with 'thing'
- contains('else') that looks for all the columns that contain 'else' in their name

- starts_with('some') that looks for all columns that begin with 'some'
- ends_with('thing') that looks for all the columns that start with 'thing'
- contains('else') that looks for all the columns that contain 'else' in their name
- matches(...) that looks for all the columns that match with a regular expression (see ?regexp) that we indicate.

- starts_with('some') that looks for all columns that begin with 'some'
- ends_with('thing') that looks for all the columns that start with 'thing'
- contains('else') that looks for all the columns that contain 'else' in their name
- matches(...) that looks for all the columns that match with a regular expression (see ?regexp) that we indicate.
- ▶ num_range('x',1:15) that looks for all columns that are like x1, x2,...,x15

- starts_with('some') that looks for all columns that begin with 'some'
- ends_with('thing') that looks for all the columns that start with 'thing'
- contains('else') that looks for all the columns that contain 'else' in their name
- matches(...) that looks for all the columns that match with a regular expression (see ?regexp) that we indicate.
- ▶ num_range('x',1:15) that looks for all columns that are like x1, x2,...,x15

There's a family of functions designed to work with select, so we can work more easily. Among them we have:

- ▶ starts_with('some') that looks for all columns that begin with 'some'
- ends_with('thing') that looks for all the columns that start with 'thing'
- contains('else') that looks for all the columns that contain 'else' in their name
- ▶ matches(...) that looks for all the columns that match with a regular expression (see ?regexp) that we indicate.
- ▶ num_range('x',1:15) that looks for all columns that are like x1, x2,...,x15

Select all the columns that end with 'ed', and the column solo. Is there something that stands out?

There's a family of functions designed to work with select, so we can work more easily. Among them we have:

- ▶ starts with('some') that looks for all columns that begin with 'some'
- ▶ ends_with('thing') that looks for all the columns that start with 'thing'
- contains('else') that looks for all the columns that contain 'else' in their name
- ▶ matches(...) that looks for all the columns that match with a regular expression (see ?regexp) that we indicate.
- ▶ num_range('x',1:15) that looks for all columns that are like x1, x2,...,x15

Select all the columns that end with 'ed', and the column solo. Is there something that stands out? Define a dataframe named members_n that only has members that died, and no columns end with 'ed' or 'id'.

Renaming with select

We can use select to rename the columns we are choosing:

```
members %>%
select(gender = sex,used_oxygen = oxygen_used,role = expedition_role)
```

rename

What did you notice from the previous slide?

rename

What did you notice from the previous slide? We lost all of the columns we didn't mentioned. We can modify this by including everything() next to the last column. Or we can use the function rename.

```
members %>%
  rename(gender = sex,used_oxygen = oxygen_used,role = expedition_role)
```



mutate



Figure 4: Another type of mutate

mutate

Let's concatenate the role and the citizenship, using the function paste, which binds together two strings:

mutate

Let's concatenate the role and the citizenship, using the function paste, which binds together two strings:

```
members_n %>%
mutate(Role_citizenship = paste(expedition_role,citizenship))
```

mutate

We can also do numeric operations, for example getting differences explicitly:

```
members_n %>%
  mutate(Difference_mts_died = death_height_metres - highpoint_metres)
```

mutate to create brand new columns

We can also add and create our own columns, using our own values or from other places. But we have to be very careful they are in the appropriate order, otherwise we risk making a very dangerous mistake. How can it be dangerous?

```
members1905 %>% # this has 9 rows!
mutate(my_row = (1:9)^2 + log(15)*(9-row_number()))
```

mutate + if_else

R has a very neat function called if_else that is just like an if, and checks wethere a condition is true, then do something, if it's not, then do something else:

mutate + other functions

We can also combine mutate with other functions to obtain new columns that depend on all the values from specific columns of the dataframe:



Exercises mutate

▶ What's the percentage of people that died out of those that got injured?

Exercises mutate

- ▶ What's the percentage of people that died out of those that got injured?
- Out of those that died, how many got injured before?

Exercises mutate

- ▶ What's the percentage of people that died out of those that got injured?
- Out of those that died, how many got injured before?
- Create a new column that is the concatenation of the member id and it's citizenship.

summarise

summarise

It's just like mutate, but the output has to be of only one row, and it eliminates (in the output) everything else that is not mentioned*:

summarise

It's just like mutate, but the output has to be of only one row, and it eliminates (in the output) everything else that is not mentioned*:

We are only getting one row, and losing everything else. Using, filter, mutate and summarise, indicate how far the average of the highpoint metres for the women from France differs from the average from all the table (use na.rm=TRUE).





Figure 5: Another type of group

```
members %>%
group_by(died) %>%
summarise(Percentage_injured = mean(injured))
```

What do you think group_by does?

On it's own, group_by doesn't do much, it really shines when we combine it with the other 5 verbs that we have studied.

On it's own, group_by doesn't do much, it really shines when we combine it with the other 5 verbs that we have studied.

All it does is group by the variables we tell it to, and the following modifications that happen on the data.frame are done on each of the groups we defined, as if each group was a dataframe.

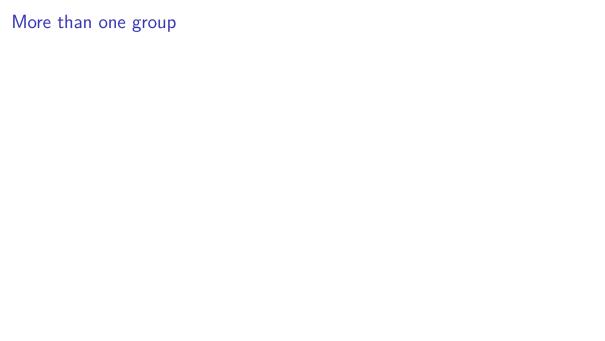
group_by + mutate

When we combine it with mutate, we get a new observation, that is only different for each of the grouped variables.

```
members_n %>% select(-(4:14)) %>%
  group_by(season) %>%
  mutate(Obs_per_season = n())
```

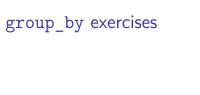
group_by + mutate

Repeat what we did above, but grouping by year, instead of season.



More than one group

If you want to group with more than one variable, you can simply add the column in the argument:



group_by exercises

▶ What year had the most dead people? (Hint: arrange)

group_by exercises

- ► What year had the most dead people? (Hint: arrange)
- Using only those with a role of climbers, compute the mean and standard deviation of the yearly number of climbers for each season. (Hint: use season as the first grouping variable)

group_by exercises

- ► What year had the most dead people? (Hint: arrange)
- Using only those with a role of climbers, compute the mean and standard deviation of the yearly number of climbers for each season. (Hint: use season as the first grouping variable)

```
group_by + summarize
```

Answer to the second question above:

```
group_by + filter
```

If you want to eliminated the groups that don't have enough, or that have too many observations, you can do it by combining filter and group_by directly:

```
(surviving_members <- members %>%
  group_by(expedition_id) %>%
  filter(n() > 10))
```

rowwise

This function, as it name tells us, is like doing a group_by, but operates on each row. This one is particular useful when you are creating your own functions and they have weird interactions with vectors, like using sum, mean, and such... But we would probably get different results on each row.

Thank you :)

Any questions?

Further references

- Check the help page and the vignettes of dplyr! (type ?dplyr, or: vignette('dplyr') on the console)
- R for Data Science, by Hadley Wickham
- Advanced R, by Hadley Wickham
- ► The R Inferno, by Patrick Burns