Day 2 - 04 MAY 2021

Slide 2:

This are useful tidyverse help links that you can explore after the session today

I want to point out that frustration is natural when you start programming in R. But, it happens to everyone, and the only way to get over it is to keep trying.

Slide 3:

This is just a reminder of the format for how to install and load package in R.

You only need to install a package once, but you need to reload it every time you start a new session.

**Slide 12:**

These are the 3 most common packages for reading data into R:

Readr – reads plan text file e.g .csv are comma separated values text files which allows data to be saved in a table structured format.

**Slide 13:**

Let us try and load the birth weight dataset one of the dataset that will be used during this training

We will load the birthweight data and assign it an object called bw\_df

Let try and do this now…..hint we will use read\_csv after loading the package tidyverse

Readxl – reads excel files

Package haven & foreign reads data from other softwares e.g STATA, SPSS or SAS

**Slide 14:**

Here is the solution, you should have this on your script

**Slide 15:**

In the next few slides, I will take you through how to transform your data using the dplyr package which is embedded in tidyverse

**Slide 16:**

To demonstrate how to transform data, we will use the gapminder dataset which is a short extract on life expectancy, GDP per capita, and population by country and it is dataset that you can load using gapminder package

So the data frame has 1704 rows/observation and 6 columns/variables

The variables includes:

Country which is a factor variable with 142 levels

Continent which is also a factor variable with 5 levels

Year ranges from 1952 to 2007 in increments of 5 years

Life expectancy at birth and this is in years

Population size

The GDP per capita in US dollars

We will install and load the gapminder package later when get to your first task and within the package you will find the gapminder dataset

**Slide 17:`**

I have run these two commands to allow us to get a glimpse of the data.

So that remember the command library loads the package and

the command glimpse makes it possible to see every variable/column in a data frame.

It tries to show you as much data as possible and it always shows the underlying variables which I have just described

**Slide 18:**

Often you’ll need to create some new variables or summaries, or maybe you just want to rename variables or reorder the observations in order to make the data a little easier to work with.

In this session, you will learn the 5 most common dplyr verbs for manipulating data which are:

1 Filter which extracts row

2 We have select for extracting columns

3 arrange for sorting rows

4 mutate to create new columns

5 and summarize in combination with the group by verb to makes group summaries, for example, if you want to know the mean life expectancy in the Africa continent you would you use these verbs.

**Slide 19:**

So I will walk you through on how the select verb works and how to use it in R

**Slide 20:**

It’s not uncommon to get datasets with many variables. The first challenge is often narrowing in on the variables you’re actually interested in.

If for example you wanted the blue and yellow column only, then select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

**Slide 21:**

In our gapminder dataset, let’s look at the first few rows in this data by running head(gapminder) – head is the function

We see that variable country contains Afghanistan, the continent is Asia, the years do vary we see 1952, 1957 etc, we have the life expectancy at birth which is quite low in Afghanistan at 28 years in some of the years and the population size is also shown with the gdp per capita also included and finally we have year category which shows before 1980.

One thing you will also notice is that the table includes the data types i.e factor variables, integer, double – which is precision floating number and character

**Slide 22:**

If you want to subset country, life expectancy and year variables only from the dataset, this is how you do it

The function is select and the data is gapminder and what you want to extra is country, year, lifeexp

select(.data = gapminder, c(country,year , lifeExp))

The first argument is the name of the data frame. The subsequent argument is the expressions of the specific variables you want to select from the data frame.

The output below shows you the data after applying the select function and now the data only has the same number of observations i.e. 1704 but now instead of 7 variables, we have 3 variables

**Slide 23:**

**Task 1:**

Please note dplyr functions never modify their inputs, so if you want to save the result, you’ll need to use the assignment operator, <- to create the new dataset

**Slide 24**

**Solution**

**Slide 25: Filter**

Let’s look at the filter verb

**Slide 26:**

So Filter extracts rows that meet some criteria that is filter allows us to subset observations based on some values.

Again, the first argument is the name of the data frame in the filter function. The second and subsequent arguments are the expressions that extras rows in the dataset.

**Slide 27:**

For example, if you want to extra only row in blue, then you would use the filter verb to do this

If you wanted for example to extra countries that are in the Africa continent only, then you would use the filter function

**Slide 28:**

Again using the gapminder dataset, let’s look at the first rows in our data by running head(gapminder) just as a reminder of what the data looks like

**Slide 29:**

Let us subset our dataset to only show Tanzania data

I should mention at this point that when you are testing for equality, we use a double equal sign (==) and single equal (=) is used when you are setting an argument.

So if you want to subset the dataset to show Tanzania only, this is how you would do it

filter(.data = gapminder, country == “Tanzania”)

Let look at the output, so this is a snapshot of the data once you’ve been able to subset the data

So the country now reads Tanzania, continent is Africa, years again vary e.g 1952,1957 etc, life expectancy was also low at 41 year but a bit higher than Afghanistan, and the population size is also shown with the gdp per capita also included and finally we have year category now shows before 1980, 1980-2000, After 200 must have been a typo should read After 2000.

**Slide 30: Logical operators**

To use filtering effectively, you have to know how to select the observations that you want using some logical operators. R provides a suite of these operator which include :less than denoted by that <, >, ==, <=, >=, !=.

A useful short-hand operator is this group membership operator x %in% y which selects every row where x is one of the values in y. I will show you an example later

If you want to determine if a value is missing, you can either use is.na and the logical operator of non-missing values we use !is.na. where not is denoted by the exclamation mark

**Slide 31:**

**Task 2**

**I want us to use filter and the logical test to show**

1. **Data for Kenya**

**Slide 32-34:**

**Solution**

**Slide 35:**

Common mistakes that you encounter include using a single equal sign instead of a double equal sign.

So when you use = instead == you’ll get an informative error and you should amend accordingly

Another common mistake is forgetting quotes for character or factor variables and this will also return an error

Let’s run this in R studio to see what these errors are

**Slide 36:**

You can also filter data based on multiple conditions.

So if you wanted to extract data for Tanzania before the year 2000, this is how you would do this.

This is a snapshot of what that data will look like after applying the filter

So the country still reads Tanzania, continent is Africa, years again vary e.g 1952,1957 etc, life expectancy was also low at 41 year but a bit higher than Afghanistan, and the population size is also shown with the gdp per capita also included

The year category should now only include before 1980, 1980-2000 and shouldn’t include After 200 since we want to subset the dataset to only include observations before the year 2000.

**Slide 37:**

So we can also use Boolean operators to filter multiple arguments instead of commas

filter() can be combined with “and”: where a & b must be true in order for a row to be included in the output.

Other types of combinations, you’ll will use are: a | b “or”, and ! “not”.

An example when it’s useful to you not is for example if you wanted to subset your data to include all continent except Africa then the not operator would be useful in this case

**Slide 38:**

So we can write the first command which was

filter(.data = gapminder, country == "Tanzania" , year<2000)

As

filter(.data = gapminder, country == "Tanzania" & year<2000)

The commands are the same expect that we have used the and operator in place of the comma

**Slide 39:**

**Task 3**

**So let’s use the filter function and Boolean logical to show**

1. **Kenya after 2000**

**Slide 40-42**

**Solution**

**Slide 43**

Again here are some common mistakes

Collapsing multiple tests into one and what I mean by that is that if you want to subset the data based on the range in year in this case between the year 1960 and 1980 then the first command will return an error

This is how you should specify the command since you using multiple condition

When you are using multiple condition within the same variable, then it is easier to use the group membership x %in% y operator

Let’s run this in R studio to see what these errors are

We are going to take a 15 mins health break

**Slide 44:**

The last verb we will look at today is mutate

**Slide 45**

Again, the first argument is the name of the data frame. The second and subsequent arguments are the expressions of the new columns you want to add to the dataset.

**Slide 46**

For example, if you want to add a new column to the data, then the data set would transform from what we see on the left to a data frame that includes a new red column and you would use the mutate verb to achieve this

**Slide 47:**

As an example, let’s create a new variable called gdp which the product of gdppercap and the population size.

This is how we would do this

This is how the data would look like, so we have the same number of observation but have 8 columns instead of the original 7 columns. There is an addition gdp variable included at the end

**Slide 48:**

You can also create more than one variable within the same expression, and this is how you do this

Besides the gdp variable we have just created, we can also create a new population variable which is the population estimated per 1,000,000 and this is how you specify that

Once you do this,, the data will now contain 9 columns instead of the original 7 columns.

Let’s run this in R studio

**Slide 49:**

You can also use the ifelse condition to create/generate new variable and I will take you through that

**Slide 50:**

The first argument in the ifelse statement is the logical test that you want to perform, and the subsequent arguments are the value it should return if the condition is true and the value it should return if the condition is false.

**Slide 51:**

If you wanted to create a new variable which show before and after 2000 using the gapminder data this is how you would do it

So using the mutate verb we first specify the name of the data frame and then you specify the name of the new variable thereafter you now apply the ifelse argument as follow

There are two logical tests here I will deal with the last command first and move then move to this one

The logical test is whether year > 2000, and if true the argument should returns the value “After 2000” and if it’s not that is the year <= 2000 then the value “Before 2000” is returned

This is a snapshot of the result of this mutation, the observations are the same, but we now have a new variable called after\_2000

Let’s run this in R studio

The other logical test here is whether year > 1960, the argument return a True value if year > 1960 otherwise it would return false if the year is <= 1960

Let’s run this in R studio

**Slide 52:**

**Task 4**

**Slide 53-56**

**Solutions**

**Slide 57:**

If you have multiple condition, what do you do?

If you wanted to make a dataset with just the year 2002 and you also wanted to calculate the log gdp per capita then you would first use the filter verb to extract rows with the year 2002 then create a new column with the log of gdp per capita as follows:

Solution 1

Let’s run this in R studio

Solution 2

However, the is another way to go about this within the same argument using the pipe operator

The pipe operator denoted by this sign allow you to take objects on the left and then pass it as the first argument of the function on the right,

For example, you can use the pipe operator to extra row where the country is Kenya as follows:

**Slide 58:**

So these two commands are essentially doing the same thing, the second command however, utilizes the pipe operator

The original command we used was this

The second command now uses pipe operator

Let’s run these 2 commands in R studio so that you see the output of each

**Slide 59:**

This is how you use the pipe operator for multiple condition

you will first need to use the assignment operator, <- to create the new data frame called gapminder\_2002\_log and then specify the name of the data frame and then using the piping operator we use filter to extract the year 2002 and then we use mutate to create the new variable log\_gdppercap which contains the log of gdp per cap as shown in this command

Notice that we also use the pipe operator between each statement that would have otherwise been written as a stand alone command

Let’s run this in R studio

**Slide 60:**

So using the pipe operator, you can read a series of imperative statements e.g select, filter or mutate within the same command

You can use the pipe to rewrite multiple operations in a way that you can read from left-to-right, and from top-to-bottom.

Pipes are a powerful tool for clearly expressing a sequence of multiple operations

To see why the pipe operator is so useful, we’re going to explore this further in later session.

**This for example are multiple**

**END**

Day 5 – 07 May 2021

* Finally, in [exploratory data analysis](https://r4ds.had.co.nz/exploratory-data-analysis.html#exploratory-data-analysis), you’ll combine visualisation and transformation with your curiosity and scepticism to ask and answer interesting questions about data.