SESSION 3.1

Doin' stuff with stuff in R

WELCOME BACK!

- In the last session we covered basic data types and structures in R
  - numerics, characters, vectors, matrices, data frames, lists
- But you also wrote code in the R terminal for the first time!
  - You installed packages (quarto and rmarkdown)
- And you also previewed your first quarto document!
  - You wrote some markdown and R code
  - You learned about code chunks and inline code

TODAY'S QUIZ

- You know the drill by now
- It's up on brighstpace
- It will cover the material from the last session
- Ask eachother questions if you're unsure about something

THIS SESSION

- We’re going to cover more general programming concepts in R
  - loops, conditionals, functions
  - You’ve seen these in python, so it should be familiar
- We’re also going to look quickly at some other packages in R
  - tidyverse, janitor, psych
- And as always, you’ll be writing your own `.qmd` with explanations that make sense to you
- Let’s get started with loops

LOOPS

- You will remember from python that a loop is a way of repeating code some amount of times
- Unlike a `function` which is a way of packaging code together so you can use it when you need it
- Like if we wanted to print the numbers 1 to 10, we could write a loop

```
#looping

for i in range(1, 11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

LOOPS IN R

- In R, we have a few different ways of writing loops
- But just like python, the most common way is with a `for` loop
- Here’s how we would write a `for` loop in R

LOOPS IN R

```r
# making a vector of names

names <- c("Alice", "Bobby", "Charlie", "Dave", "Eve")
#looping
for (name in names) {
  print(paste("name = ", name))
}
```

```
[1] "name =  Alice"
[1] "name =  Bobby"
[1] "name =  Charlie"
[1] "name =  Dave"
[1] "name =  Eve"
```

LOOPS IN R

- The `for` loop in R is a little different from python
- And this difference is indicative of a larger difference between the two languages
- Where python has a lot of ‘whitespace’ (which just referese to things like spaces, tabs, and newlines)
- R uses a lot more puntuation, like curly braces `{}` and parentheses `()`

whitespace VS punctuation

- You'll remember that in python we said 'indentation is a whole thing'
- This is because python uses whitespace to determine the structure of the code
- This means that in python, if one line is indented more than another, it means that the indented line is **inside** the other line
- So indentation is what let's python know that a line is inside a loop or a conditional

whitespace VS punctuation

~~~
```{python}

for i in range(1, 4):# this line starts the loop, creating the placeholder i
    print(f"{i} times around the loop") # this line is inside the loop, so it will be repeated using

print("I'm outside the loop") # this line is outside the loop, so it will only be run once
```
~~~

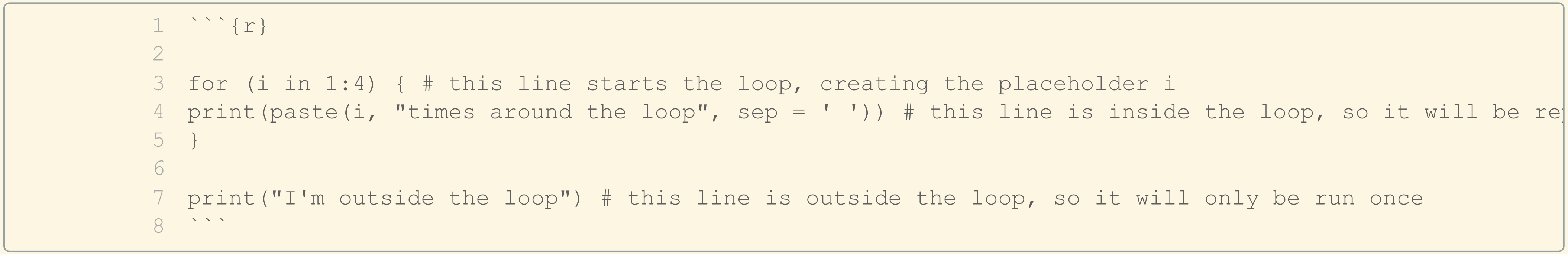
```
1 times around the loop
2 times around the loop
3 times around the loop

I'm outside the loop
```

`whitespace` VS `punctuation`

- In R, we use punctuation to determine the structure of the code
- This means that we use curly braces `{}` to indicate that a line is inside a loop, function, or conditional
- So in R, the same loop would look like this

whitespace VS punctuation



~~~
```{r}

for (i in 1:4) { # this line starts the loop, creating the placeholder i
print(paste(i, "times around the loop", sep = ' ')) # this line is inside the loop, so it will be re
}

print("I'm outside the loop") # this line is outside the loop, so it will only be run once
```
~~~

```
[1] "1 times around the loop"
[1] "2 times around the loop"
[1] "3 times around the loop"
[1] "4 times around the loop"

[1] "I'm outside the loop"
```

whitespace VS punctuation

- We **could** use indentation to make the code more readable

```r
for (i in 1:4) { # this line starts the loop, creating the placeholder i
  print(paste(i, "times around the loop", sep = ' ')) # this line is inside the loop, so it will be
}
```

```
[1] "1 times around the loop"
[1] "2 times around the loop"
[1] "3 times around the loop"
[1] "4 times around the loop"
```

- And that’s really good practice
- But in R it’s not neccessary the way it **really is** in python

`whitespace` VS `punctuation`

- You’ll also notice that we use `()` in our loop to set the bounds of the loop
- `for (thing in collection_of_things) {…}
- This is also different from python where whitespace is used to set the bounds of the loop
- `for thing in collection_of_things: ...`

whitespace VS punctuation

- Neither of these practices is better or worse than the other
- They’re just different, and it’s a matter of preference and training
- When I started learning to code, it was in R and I really hated the punctuation
- It felt like it was getting in the way of the code, and I found it really hard to read
- But now that I’ve been coding in python for a while….
- I still hate R’s punctuation, but I can read it just fine
- I know other coders, who are much better than I am, who really dislike python’s whitespace
- I suggest you learn to love both, because they’re both excellent languages

PRACTICE TIME

- You’re going to make a new `.qmd` file called `stuff_in_R.qmd`
- Make a short yaml header that shows the title and author

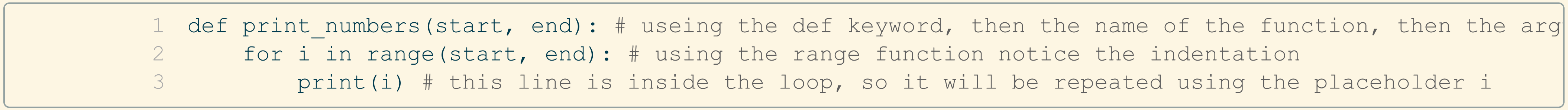
~~~
```{yaml}
---
title: "stuff in R"
author: "your name"
---
```
~~~

- Then make a level one heading that says ‘Loops in R’
- Then make a code chunk that shows a `for` loop that prints the numbers 1 to 10
- If you get stuck, try looking up “for loop in R” the examples will help
- When you’re done, preview the document, this will run the code and show you the output

Functions IN R

- You’ll remember from python that a function is a way of packaging code together so you can use it when you need it
- And they allow you to change what the code does by passing in arguments
- So, say we wanted to be able to control the range of numbers we print in our loop, we could write a function
- In python it would look like this

Functions RECAP

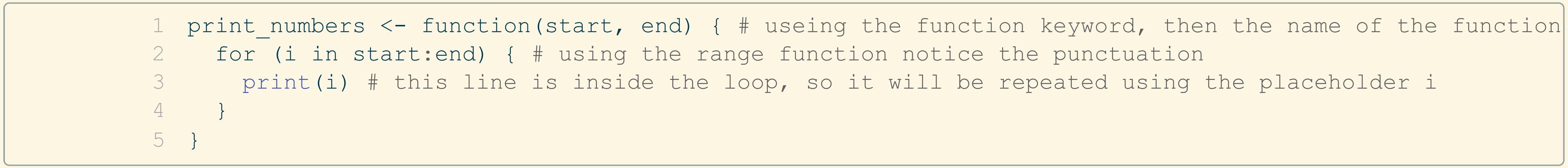


```python
def print_numbers(start, end): # useing the def keyword, then the name of the function, then the arg
    for i in range(start, end): # using the range function notice the indentation
        print(i) # this line is inside the loop, so it will be repeated using the placeholder i
```

`Functions` IN R

- In R, we use the `function` keyword to define a function
- And we use `()` to set the arguments of the function
- And we use `{}` to set the body of the function
- So the same function in R would look like this

Functions IN R



```r
print_numbers <- function(start, end) { # useing the function keyword, then the name of the function
  for (i in start:end) { # using the range function notice the punctuation
    print(i) # this line is inside the loop, so it will be repeated using the placeholder i
  }
}
```

Functions IN R

- You'll notice that the function in R is very similar to the function in python
- The main difference again, is the use of punctuation rather than whitespace in R
- And the use of the assignment operator `<-` to assign the function to a name
- In python we use the `def` keyword to define a function
- In R we use the `function` keyword

FUNCTIONS AS OBJECTS

- In R, functions are objects just like any other object
- So in R we say
  - `thing <- function() {...}`
- As opposed to python where we say
  - `def thing(): ...`

CALLING A FUNCTION

- In python, we call a function by using the name of the function and passing in the arguments
- So to call the function we just wrote, we would write

```
print_numbers(1, 5) # this will print the numbers 1 to 4
```

```
1
2
3
4
```

- In R, we call a function the same way

```
print_numbers(1, 4) # this will print the numbers 1 to 4
```

```
[1] 1
[1] 2
[1] 3
[1] 4
```

PRACTICE TIME

- In the file called `stuff_in_R.qmd`
- Make a level one heading that says ‘Functions in R’
- Then make a code chunk that shows a function that prints the numbers between two arguments
- Try using the `paste` function to make the output more readable

CONDITIONALS

- You’ll remember from python that a conditional is a way of running code only if a certain condition is met
- In python we focused on the `if-elif-else` structure where we could run different code depending on the value of a variable
- We use the boolean operators `==`, `!=`, `>`, `<`, `>=`, `<=` to compare values
- if the condition is `True` the code runs, if the condition is `False` the code doesn’t run

```python
x = 5

if x < 5:
    print("x is less than 5")
elif x < 5:
    print("x is greater than 5")
else:
    print("x is equal to 5")
```

```
x is equal to 5
```

CONDITIONALS IN R

- In R, we use the `if-else` structure to run code depending on the value of a variable
- And we use the boolean operators `==`, `!=`, `>`, `<`, `>=`, `<=` to compare values
- So conditionals in R are really similar…
- But with the same punctuation vs whitespace difference that we've been talking about

CONDITIONALS IN R

```r
x <- 5

if (x < 5) {
  print("x is less than 5")
} else if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is equal to 5")
}
```

```
[1] "x is equal to 5"
```

CONDITIONALS IN R

- You'll notice that the `if-else` structure in R is very similar to the `if-elif-else` structure in python
- except we use the {} to contains the code that runs if the condition is `True`
- And instead of `elif` we use `else if` in R
- Which again, might be more typing, but it might be a little more readable
- I don't love it because I like the clear distinction between 'else' and 'elif' in python
- But again, it's a matter of preference

PRACTICE TIME

- In the file called `stuff_in_R.qmd`
- Make a level one heading that says ‘Conditionals in R’
- Look up how to count the number of symbols in a `character` in R
- Look up how to use the modulo operator `%` in R (this is also an operator in python)
- Write a conditional that prints a message depending on the number of symbols in a `character` is odd or even

PRACTICE RECAP

- You’ve now written a loop, a function, and a conditional in R
- You’ve seen how R uses punctuation to structure code
- And you’ve seen how R uses the `function` keyword to define a function
- And you’ve seen how R uses the `if-else` structure to run code depending on the value of a variable
- You’ve also seen how to look up how to do things in R
- And you’ve seen how to write a `.qmd` file with explanations that make sense to you
- You’re doing great! Keep up the good work!

PRACTICE RECAP

- Before we move onto the next section, let’s put all this together
- Make a level one heading that says ‘Putting it all together’
- Then write a function that takes in a `vector` of `characters` (maybe names)
- The function should iterate over the `vector` and print each name
- And the function should print a message depending on the number of symbols in the name (nchar(name) %% 2 == 0)
- Then call the function with a `vector` of your choice

PRACTICE

- Take your time and work through the problem
- Try making the loop first and then the conditional
- Then see how you might wrap all that into a function.

PUTTING IT ALL TOGETHER

```r
names <- c("Alice", "Bobbie", "Charlie", "Dave", "Eve")

print_items <- function(x) {
  for (item in x) {
    if (nchar(item) %% 2 == 0) {
      print(paste(item, "has an even number of symbols", sep = ' '))
    } else {
      print(paste(item, "has an odd number of symbols", sep = ' '))
    }
  }
}

print_items(names)
```

```
[1] "Alice has an odd number of symbols"
[1] "Bobbie has an even number of symbols"
[1] "Charlie has an odd number of symbols"
[1] "Dave has an even number of symbols"
[1] "Eve has an odd number of symbols"
```

OTHER PACKAGES

- In the earlier session you learned how to `install.packages’ using the R terminal
- This is part of getting your project ready to rock
- In the rest of this session we’re going to take a quick look at some other packages in R
- And so you need to install them

INSTALLING MULTIPLE PACKAGES

- In vscode you can have multiple terminals open at once so you don't have to stop the quarto preview
- Go to the menu and select `Terminal` -> `New Terminal`
- This will open a new terminal window, where you can type R to open the R terminal
- You'll remember that installing a package in R is as simple as typing `install.packages("package_name")`
- You can also pass a vector of package names to install multiple packages at once
  - `install.packages(c("tidyverse", "janitor", "psych"))`
- Once they're installed you can `quit()` the R terminal which will bring you back to the main terminal

IMPORTING PACKAGES

- You’ll remember that in python, when we wanted to bring a package into our code we used the `import` keyword
- This gives us access to the functions, classes, and objects in the package

```python
import pandas as pd
import numpy as np
```

IMPORTING PACKAGES

- In R, packages/modules are more commonly called `libraries`
- And we use the `library` function to bring a library into our quarto document

```r
library(tidyverse)
library(janitor)
library(psych)
```

- In your `stuff_in_R.qmd` file, make a level one heading that says ‘Importing packages’
- Then write a code chunk that imports the `tidyverse` and `psych` libraries

TIDYVERSE

- The `tidyverse` is a **collection** of packages that are designed to work together
- It’s a little like the `pandas` library in python
- It’s designed to make data manipulation and visualization easier
- It allows you to work with data frames in a way that is more intuitive than base R
- And it has a lot of functions that make data manipulation easier
- It’s a really commonly used package in R, and it’s a great place to start

TIDYVERSE - IMPORTING DATA

- One of the most common things you'll do with the `tidyverse` is import data
- The `read_csv` function is a really useful function that reads a csv file into a data frame
- You can use the `read_csv` function by calling it on a file path
- Which is exactly the same as the `pd.read_csv` function in python

TIDYVERSE - IMPORTING DATA

```r
data <- read_csv("../../data/movies_df_2_2.csv")# you'll use `\\` instead of `/` in windows
head(data)
```

```
# A tibble: 6 × 8
  Director                `Movie Title`     Genre `Year of Release` `ImdB Score`
  <chr>                   <chr>             <chr>             <dbl>        <dbl>
1 John Carpenter          The Thing         Horr…              1982           82
2 <NA>                    Blade Runner 2049 Sci-…              2017           80
3 Nicolas Winding Refn    Drive             Acti…              2011           78
4 Matthijs van Heijningen The Thing         Horr…              2011           62
5 Damien Chazelle         Whiplash          Drama              2014           NA
6 Dennis Villanueve       Arrival           Sci-…              2016           79
# ℹ 3 more variables: `Rotten Tomatoes Score` <dbl>,
#   `Rotten Tomatoes Fan Score` <dbl>, `Gender of Lead` <chr>
```

TIDYVERSE - IMPORTING DATA

```
# A tibble: 6 × 8
  Director                `Movie Title`     Genre `Year of Release` `ImdB Score`
  <chr>                   <chr>             <chr>             <dbl>        <dbl>
1 John Carpenter          The Thing         Horr…              1982           82
2 <NA>                    Blade Runner 2049 Sci-…              2017           80
3 Nicolas Winding Refn    Drive             Acti…              2011           78
4 Matthijs van Heijningen The Thing         Horr…              2011           62
5 Damien Chazelle         Whiplash          Drama              2014           NA
6 Dennis Villanueve       Arrival           Sci-…              2016           79
# ℹ 3 more variables: `Rotten Tomatoes Score` <dbl>,
#   `Rotten Tomatoes Fan Score` <dbl>, `Gender of Lead` <chr>
```

- So in the code chunk above, we’re reading in a csv file called `movies_df_2_2.csv` and storing it in a data frame called `data`
- Then we use the `head` function to show the first few rows of the data frame
- Notice that in R `head` is a function, whereas in python it’s a method of the data frame
- Look at the output above and see what eles you can spot

TIDYVERSE TIBBLES

- In the earlier session we talked about matrices and data frames
- These are multidimensional data structures that are really useful for storing data
- Tidyverse has a similar data structure called a `tibble`
- A `tibble` is a data frame that is designed to be more user friendly
- And tidyverse functions give us a lot of power to manipulate `tibbles` in a way that is more intuitive than base R

TIDYVERSE TIBBLES

- You can convert a data frame to a `tibble` using the `as_tibble` function
- And you can convert a `tibble` to a data frame using the `as.data.frame` function
- You can also create a `tibble` from scratch using the `tibble` function
- But you can also do lots of stuff with tibbles that you can do with a pandas dataframe

CLEANING UP COLUMNS - JANITOR

- The `janitor` package is a package that is designed to make data cleaning easier
- It has a lot of functions that make it easier to clean up column names, remove duplicates, and other data cleaning tasks
- One of the most useful functions in the `janitor` package is the `clean_names` function
- This function takes a data frame and makes the column names lowercase and snake_case
- This is really useful because it makes the column names easier to work with

CLEANING UP COLUMNS - JANITOR

```r
data <- clean_names(data)

head(data)
```

```
# A tibble: 6 × 8
  director   movie_title genre year_of_release imd_b_score rotten_tomatoes_score
  <chr>      <chr>       <chr>           <dbl>       <dbl>                 <dbl>
1 John Carp… The Thing   Horr…            1982          82                    82
2 <NA>       Blade Runn… Sci-…            2017          80                    88
3 Nicolas W… Drive       Acti…            2011          78                    93
4 Matthijs … The Thing   Horr…            2011          62                    34
5 Damien Ch… Whiplash    Drama            2014          NA                    NA
6 Dennis Vi… Arrival     Sci-…            2016          79                    94
# ℹ 2 more variables: rotten_tomatoes_fan_score <dbl>, gender_of_lead <chr>
```

CLEANING UP COLUMNS

- Look at what the code above does
- It takes the `data` data frame and passes it to the `clean_names` function
- The `clean_names` function then makes the column names lowercase and snake_case
- This takes a lot of the code we had to write in python yesterday and compresses it into a really simple function
- This is a common thing with R, whily python is good at lits of stuff, R is **really** good at data manipulation
- And the `tidyverse` and `janitor` packages are a big part of that

SELECTING COLUMNS

- In the `tidyverse` package, the `select` function is used to select columns from a data frame
- You can use the `select` function to select columns by name
- You can also use the `select` function to select columns by index
- And you can use the `select` function to select columns by a range of indexes
- The `select` function is really useful for selecting the columns you need for your analysis

SELECTING COLUMNS

```r
#print the columns of the data frame
print(colnames(data))
```

```
[1] "director"                 "movie_title"
[3] "genre"                    "year_of_release"
[5] "imd_b_score"              "rotten_tomatoes_score"
[7] "rotten_tomatoes_fan_score" "gender_of_lead"
```

```r
selected_data <- select(data, c(movie_title, year_of_release, genre, imd_b_score))

head(selected_data)
```

```
# A tibble: 6 × 4
  movie_title       year_of_release genre  imd_b_score
  <chr>                       <dbl> <chr>        <dbl>
1 The Thing                    1982 Horror          82
2 Blade Runner 2049            2017 Sci-Fi          80
3 Drive                        2011 Action          78
4 The Thing                    2011 Horror          62
5 Whiplash                     2014 Drama           NA
6 Arrival                      2016 Sci-Fi          79
```

SELECTING COLUMNS

- Look at what the code above does
- It uses the `colnames` function to print the column names of the `data` data frame
- Then it uses the `select` function to select the columns `movie_title`, `year_of_release`, `genre`, and `imd_b_score`
- Notice that the first argument of the `select` function is the name of the data frame
- And the second argument is a vector of the column names you want to select

FILTERING ROWS

- The `filter` function is used to filter rows from a data frame
- You can use the `filter` function to filter rows based on a condition
- You can also use the `filter` function to filter rows based on multiple conditions
- The `filter` function is really useful for selecting the rows you need for your analysis

FILTERING ROWS

- Let’s just filter rows where the word ‘comedy’ is in the genre column

```r
filtered_data <- filter(data, str_detect(genre, "Comedy"))

head(filtered_data)
```

```
# A tibble: 3 × 8
  director   movie_title genre year_of_release imd_b_score rotten_tomatoes_score
  <chr>      <chr>       <chr>           <dbl>       <dbl>                 <dbl>
1 Kelly Asb… Shrek 2     Come…            2004          73                    89
2 Edgar Wri… Hot Fuzz    Come…            2007          78                    91
3 Coen Brot… Fargo       Dark…            1996          81                    94
# ℹ 2 more variables: rotten_tomatoes_fan_score <dbl>, gender_of_lead <chr>
```

FILTERING ROWS

- Look at what the code above does
- It uses the `filter` function to filter rows where the word ‘comedy’ is in the `genre` column
- The `str_detect` function is used to check if the word ‘Comedy’ is in the `genre` column
- The `filter` function then filters the rows where the `str_detect` function returns `True`
- Notice that the first argument of the `filter` function is the name of the data frame
- And the second argument is the condition you want to filter on

GROUPING AND SUMMARIZING

- The `group_by` function is used to group rows in a data frame
- You can use the `group_by` function to group rows by a column
- You can also use the `summarize` function to summarize the grouped data
- The `summarize` function is really useful for summarizing the grouped data

GROUPING AND SUMMARIZING

```r
grouped_data <- group_by(data, genre)

summarized_data <- summarize(grouped_data, mean(imd_b_score))

head(summarized_data)
```

```
# A tibble: 6 × 2
  genre       `mean(imd_b_score)`
  <chr>                     <dbl>
1 Action                     78  
2 Comedy                     75.5
3 Dark Comedy                81  
4 Drama                      NA  
5 Horror                     72  
6 Sci-Fi                     79.5
```

GROUPING AND SUMMARIZING

- Look at what the code above does
- It uses the `group_by` function to group rows by the `genre` column
- The `summarize` function is then used to summarize the grouped data
- The `mean` function is used to calculate the mean of the `imd_b_score` column
- The `summarize` function then calculates the mean of the `imd_b_score` column for each group
- Notice that the first argument of the `group_by` function is the name of the data frame
- And the second argument is the column you want to group by
- The first argument of the `summarize` function is the name of the grouped data
- And the second argument is the function you want to use to summarize the data

COMMON PATTERNS

- You'll notice that the `tidyverse` package has a lot of functions that follow a similar pattern
- You use the function to manipulate the data frame in some way
- And you pass the name of data frame to the function as the first argument
- And you pass the arguments you need to the function as the other arguments
- This makes the `tidyverse` package really easy to use
- It also makes working with tidyverse functions really intuitive

PRACTICE TIME

- In the file called `stuff_in_R.qmd`
- Make a level one heading that says ‘Tidyverse’
- Write a code chunk that imports the `tidyverse` and `janitor` libraries
- Write a code chunk that reads in the `movies_df_2_2.csv` file
- Write a code chunk that cleans the column names of the data frame
- Write a code chunk that selects the columns `movie_title`, `year_of_release`, `genre`, and `imd_b_score`
- Write a code chunk that filters the rows where the word ‘comedy’ is in the `genre` column
- Write a code chunk that groups the data by the `genre` column and calculates the mean of the `imd_b_score` column

THE %>% OPERATOR

- The `%>%` (pipe) operator is used to chain functions together
- You can use the `%>%` operator to pass the output of one function to the next function
- This makes it really easy to chain functions together
- And it makes the code more readable
- The `%>%` operator is really useful for working with the `tidyverse` package

THE %>% OPERATOR

- For exampe, we could read in the data, clean the column names, select the columns, filter the rows, and group the data all in one line

```r
summary <- read_csv("../../data/movies_df_2_2.csv") %>%
  clean_names() %>%
  select(movie_title, year_of_release, genre, imd_b_score) %>%
  filter(str_detect(genre, "Comedy")) %>%
  group_by(genre) %>%
  summarize(mean(imd_b_score))

head(summary)
```

```
# A tibble: 2 × 2
  genre       `mean(imd_b_score)`
  <chr>                     <dbl>
1 Comedy                     75.5
2 Dark Comedy                81
```

THE %>% OPERATOR

- Look at what the code above does
- It uses the `%>%` operator to chain the `read_csv`, `clean_names`, `select`, `filter`, `group_by`, and `summarize` functions together
- The `%>%` operator passes the output of one function to the next function
- This makes the code more readable and easier to understand
- The `%>%` operator is really useful for working with the `tidyverse` package

PRACTICE TIME

- In the file called `stuff_in_R.qmd`
- Make a level one heading that says ‘The %>% operator’
- Write a code chunk that reads in the `movies_df_2_2.csv` file, cleans the column names, selects the columns `movie_title`, `year_of_release`, `genre`, and `imd_b_score`, filters the rows where the word ‘comedy’ is in the `genre` column, groups the data by the `genre` column, and calculates the mean of the `imd_b_score` column
- Use the `%>%` operator to chain the functions together

PSYCH

- The `psych` package is a package that is designed to make it easier to work with psychological data
- It has a lot of functions that make it easier to analyze psychological data
- One of the most useful functions in the `psych` package is the `describe` function
- This function gives you a summary of the data in a data frame
- It tells you the mean, median, mode, standard deviation, and other statistics for each column in the data frame

PSYCH - DESCRIBE

- Let’s use the `describe` function to get a summary of the `data` data frame

```
describe(data)
```

```
                          vars  n    mean    sd median trimmed  mad  min  max
director*                    1  9    4.11  2.57    4.0    4.11 2.97    1    8
movie_title*                 2 10    5.30  2.75    5.5    5.38 3.71    1    9
genre*                       3 10    3.80  1.75    4.0    3.88 2.22    1    6
year_of_release              4 10 2006.50 10.62 2009.0 2008.25 7.41 1982 2017
imd_b_score                  5  9   77.22  6.34   79.0   77.22 2.97   62   82
rotten_tomatoes_score        6  9   84.22 19.22   91.0   84.22 4.45   34   94
rotten_tomatoes_fan_score    7 10   80.50 15.62   84.0   83.62 9.64   42   94
gender_of_lead*              8 10    3.90  1.60    4.5    4.00 1.48    1    6
                          range  skew kurtosis   se
director*                     7  0.13    -1.68 0.86
movie_title*                  8 -0.17    -1.61 0.87
genre*                        5 -0.17    -1.58 0.55
year_of_release              35 -1.11     0.12 3.36
imd_b_score                  20 -1.44     0.81 2.11
rotten_tomatoes_score        60 -1.92     2.21 6.41
rotten_tomatoes_fan_score    52 -1.36     0.86 4.94
gender_of_lead*               5 -0.45    -1.30 0.50
```

PSYCH - DESCRIBE

- Look at what the code above does
- It uses the `describe` function to get a summary of the `data` data frame
- The `describe` function gives you a summary of the data in the data frame
- It tells you the mean, median, mode, standard deviation, and other statistics for each column in the data frame
- The `describe` function is really useful for getting a quick summary of the data in a data frame

PRACTICE TIME

- In the file called `stuff_in_R.qmd`
- Make a level one heading that says ‘Psych’
- Write a code chunk that imports the `psych` library
- Write a code chunk that uses the `describe` function to get a summary of the `data` data frame

OTHER ANALYSIS

- The `psych` package has a lot of other functions that make it easier to analyze psychological data
- For example, the `alpha` function is used to calculate Cronbach's alpha
- The `fa` function is used to do factor analysis
- The `principal` function is used to do principal components analysis
- We won't go into these functions in detail, but they're really useful for analyzing psychological data

SUMMARY

- In this session we covered more general programming concepts in R
  - loops, conditionals, functions
- We also looked quickly at some other packages in R
  - tidyverse, janitor, psych
- And as always, you wrote your own `.qmd` with explanations that make sense to you
- You’re doing great! Keep up the good work!

SUMMARY

- This is very much a whirlwind tour of R
- And there’s so much more, but this coveres the basics
- Youtube channels like `R for Data Science` and `Statists of Doom` are great resources for more specific stuff
- And the `R` community is really active and helpful

THE REST OF THE PROGRAMME

- For the rest of this summer school we're going to have you work in groups on an actual project
- You'll be working with a dataset and using the skills you've learned to analyze the data
- You'll be writing a report that explains your analysis and your findings
- So go take a break, while we set up the details for what you need on brightspace