DDO week 6 test

Fontys ICT

10/11/2019

Before you start

Make sure that you have copied the StudentNoteBook to your Exam_HandIn folder, as well as your 2 .CSV files. Test this by running the following chunk

```
getwd()
```

Make sure it returns your Exam_HandIn folder. If it does not return, save your StudentNoteBook in the Exam_HandIn folder. If you know that you are saving in the Exam_HandIn folder you can continue.

Now add your name:

Name:

Studentnumber:

Class: PC___

Introduction text

In this exam we will go over some of the assignments you made earlier. Based on the "introduction to the tidyverse", the "working with data in the tidyverse", and the "data visualisation with ggplot2" courses. In three sections we will focus on some of the questions and theory presented in these courses, you might actually recognize the dataset and some of the questions.

Starting with a recap of the introduction of the Tidyverse we will look at some exercises of the Gapminder we worked with. As we will be using different libraries you need to enable these:

Let's refresh our minds by looking at the gapminder dataset.

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Exam questions

Part 1

Let's try for the first assignment something straight forward:

1) Which 2 filters would you need to get below mentioned table?

```
gapminder %>% filter( ___ , ___ )
```

```
## # A tibble: 1 x 6
##   country continent  year lifeExp        pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>      <int>     <dbl>
## 1 China   Asia       2002    72.0 1280400000     3119.
```

2) After filter we learned about arrange for the Gapminder. If we want to complete this code to have the filter for 1957 and the dataset arranged descending for population.

```
# Filter for the year 1957,
#  then arrange in descending order of population

gapminder %>%
filter( ___ ) %>%
arrange( ___ )
```

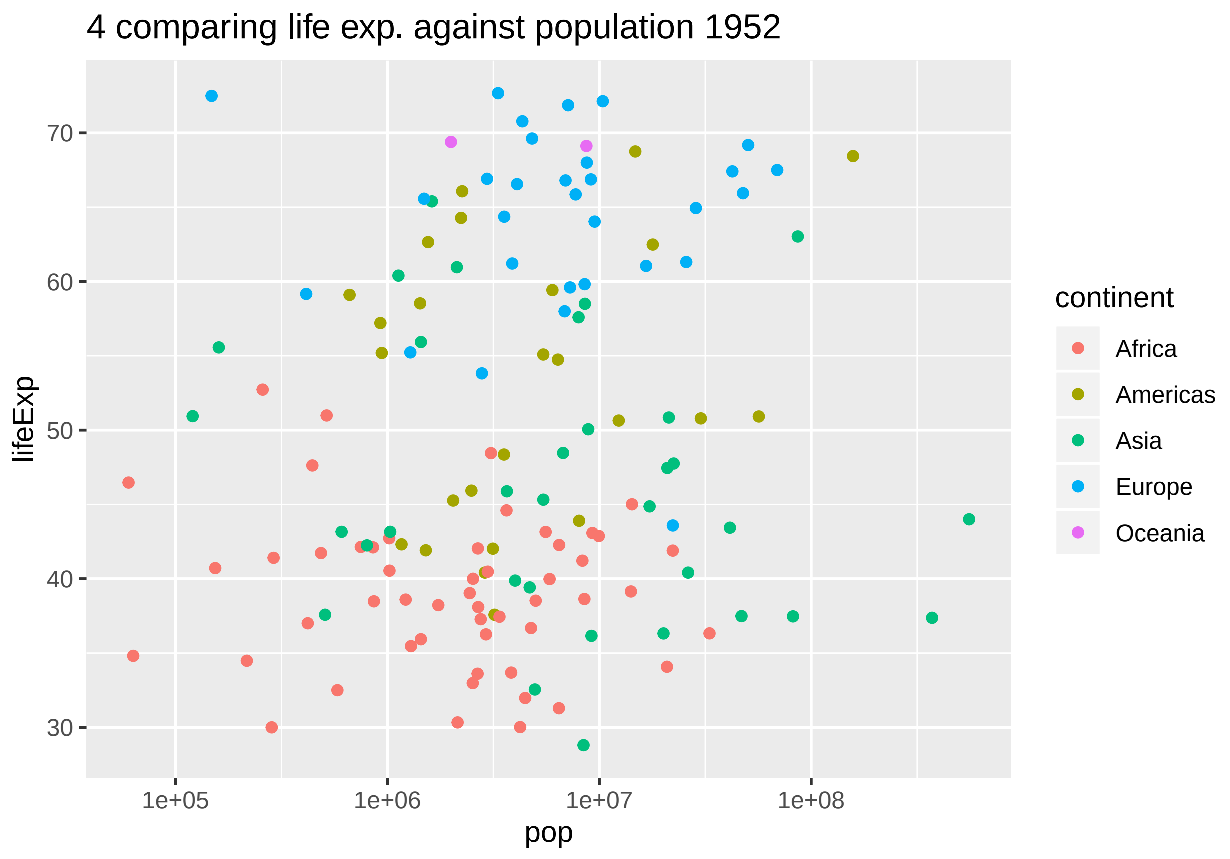
3) finally we can combine this with a mutate, In this exercise, you'll combine all three of the verbs you've learned in this chapter, to find the countries from high to low "life expectancy in months", for the year 2002. Complete the code below:

```
# Filter for the year 2002
# mutate the life expectancy to represent months instead of years
# and arrange the gapminder dataset with life expectancy in months from high to low

gapminder %>%
filter( ___ ) %>%
mutate(lifeExpMonths = 12 * ___ ) %>%
arrange( ___ )
```

Now that we have gained some heat in our coding we will pick up our pace. Firstly we want to look at the data in 1952, to filter for 1952 we use the following code:

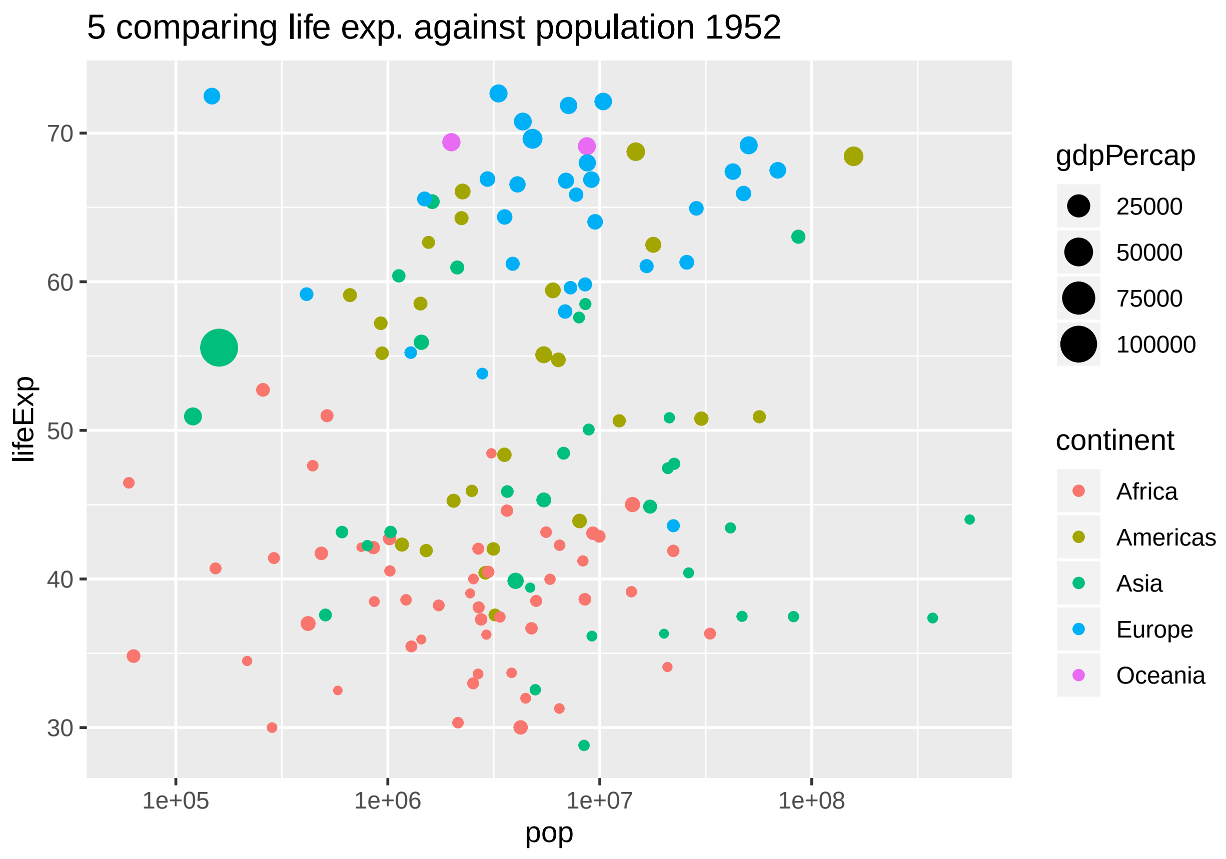
4) Complete this code to get this image.

```r
gapminder_1952 <- gapminder %>%
  filter( ___ )

ggplot(gapminder_1952, aes(x= ___ , y= ___ , colour= ___ )) +
  geom_????( ___ ) +
  scale_x_log10() +
  ggtitle( '4 comparing life exp. against population 1952')
```



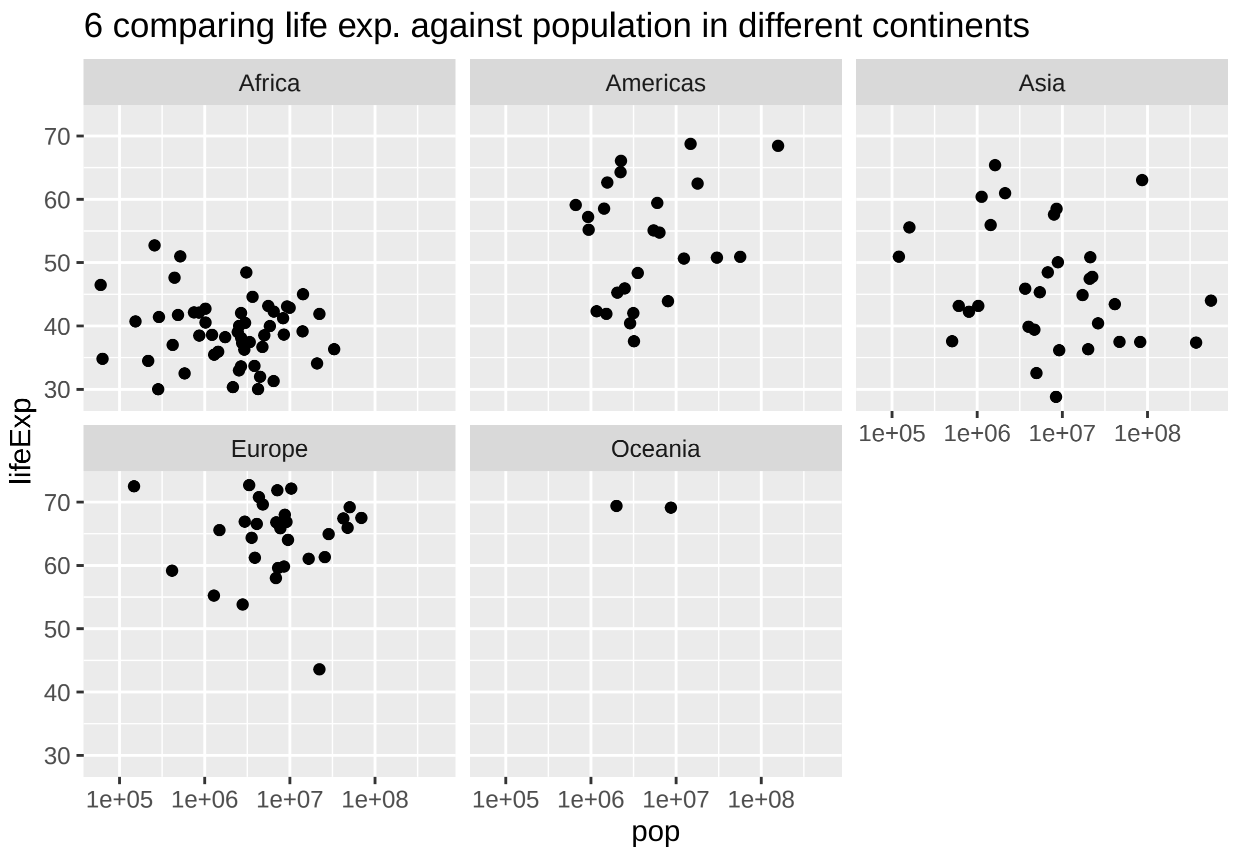
5) Complete the code to get this image

```
gapminder_1952 <- gapminder %>%
  filter( ___ )

ggplot(gapminder_1952, aes(x = ___ , y = ___ , colour = ___ , size = ___ )) +
  geom_????() +
  scale_x_log10() +
  ggtitle( '5 comparing life exp. against population 1952')
```



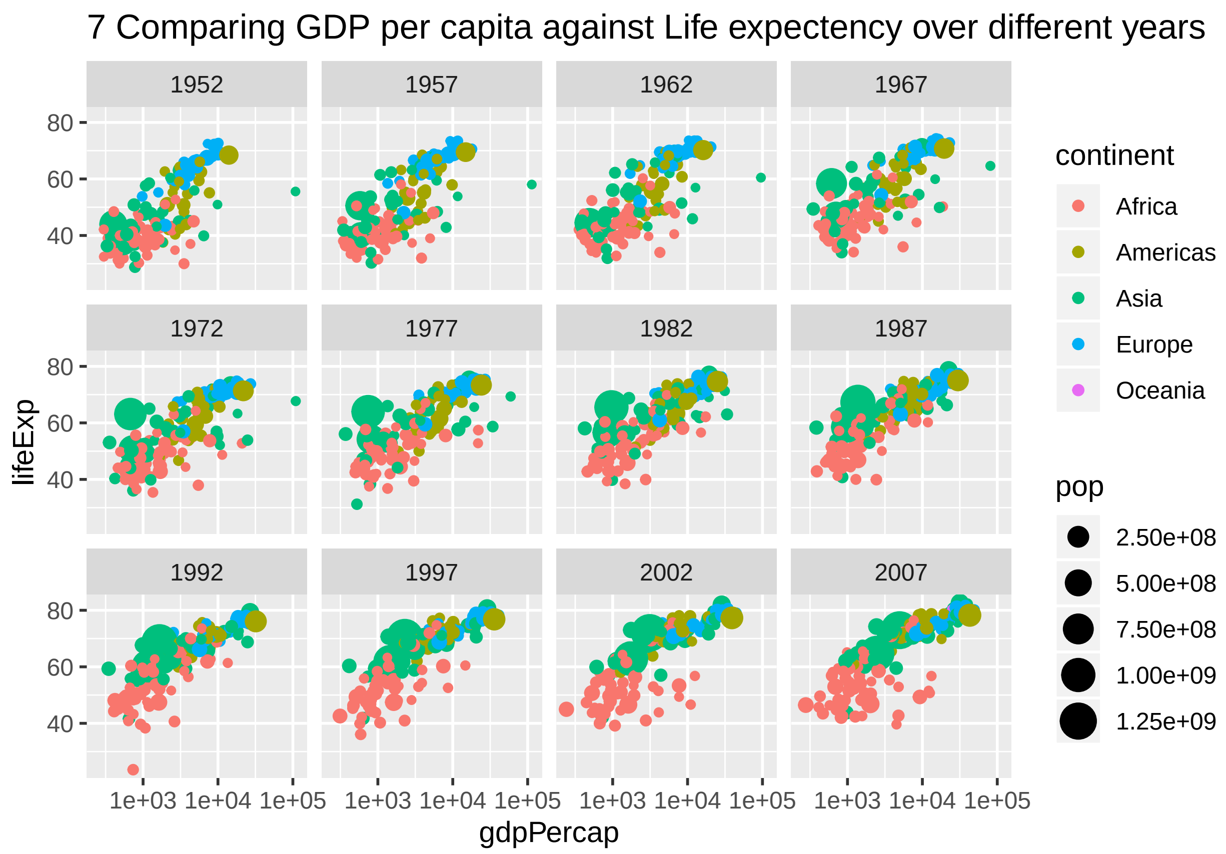
6) Complete the code to get this image

```
gapminder_1952 <- gapminder %>%
  filter( ___ )

ggplot(gapminder_1952, aes(x= ___ , y= ___ )) +
  geom_????() +
  scale_x_log10() +
  ???_????(~ ___ ) +
  ggtitle(  '6 comparing life exp. against population in different continents')
```



7) combining the techniques from question 6 and 5 we can now make an overview of Gapminder for all years to create the image below, what is the code for this?

```r
ggplot(gapminder, aes(x= ___ , y= ___ , colour= ___ , size= ___ )) +
  geom_????() +
  scale_x_log10() +
  ????_????(~  ___ ) +
  ggtitle( '7 Comparing GDP per capita against Life expectancy over different years')
```



As you see, in the last assignments we have been working with the ggplot2 library, let's dive deeper into this library.

Part 2

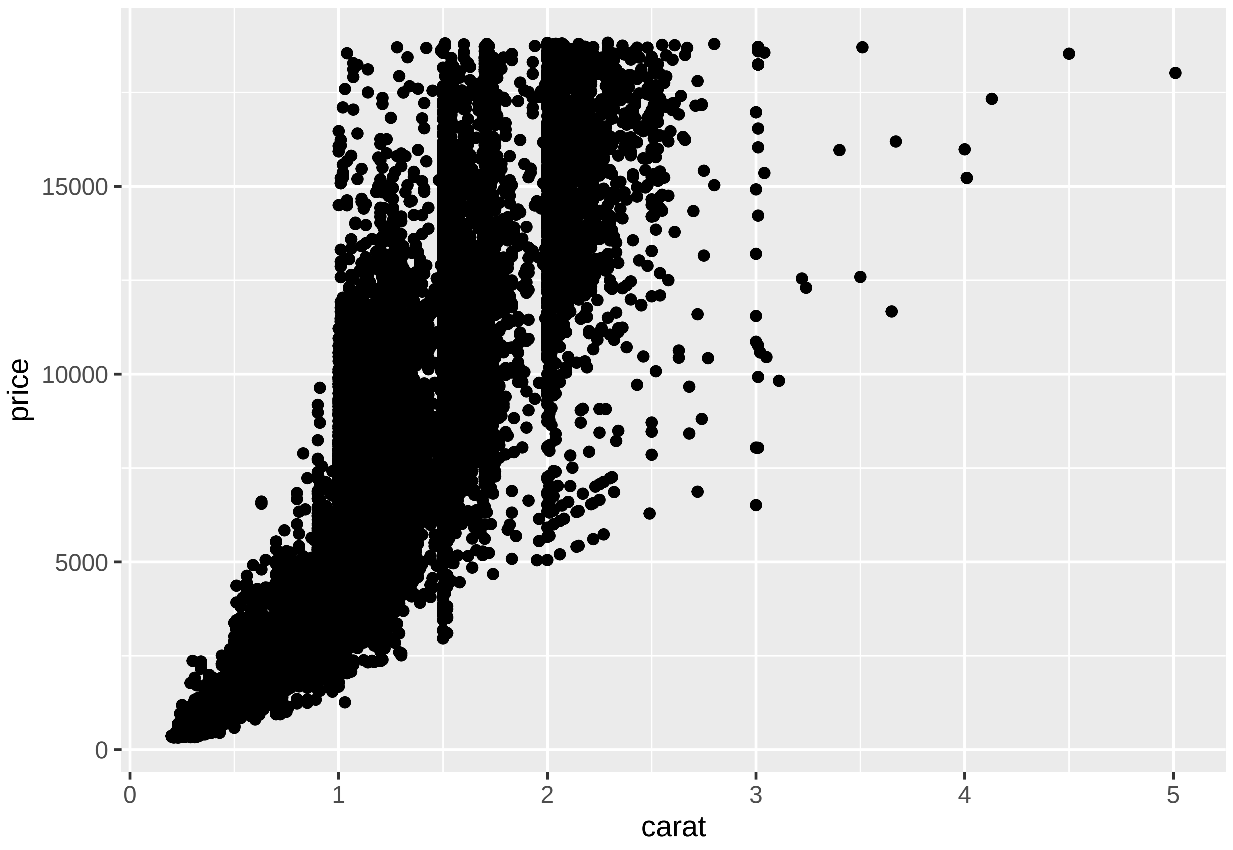
For this excercise we will look at the data set called Diamonds.

In this data set we have an overview of different diamonds on their size, price, colour, clarity, cut. In this excercise we will look only at the price, size and cut of the diamond. We want to know the relation between the weight and price for different cuts.

```
str(diamonds)
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
head(diamonds)
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

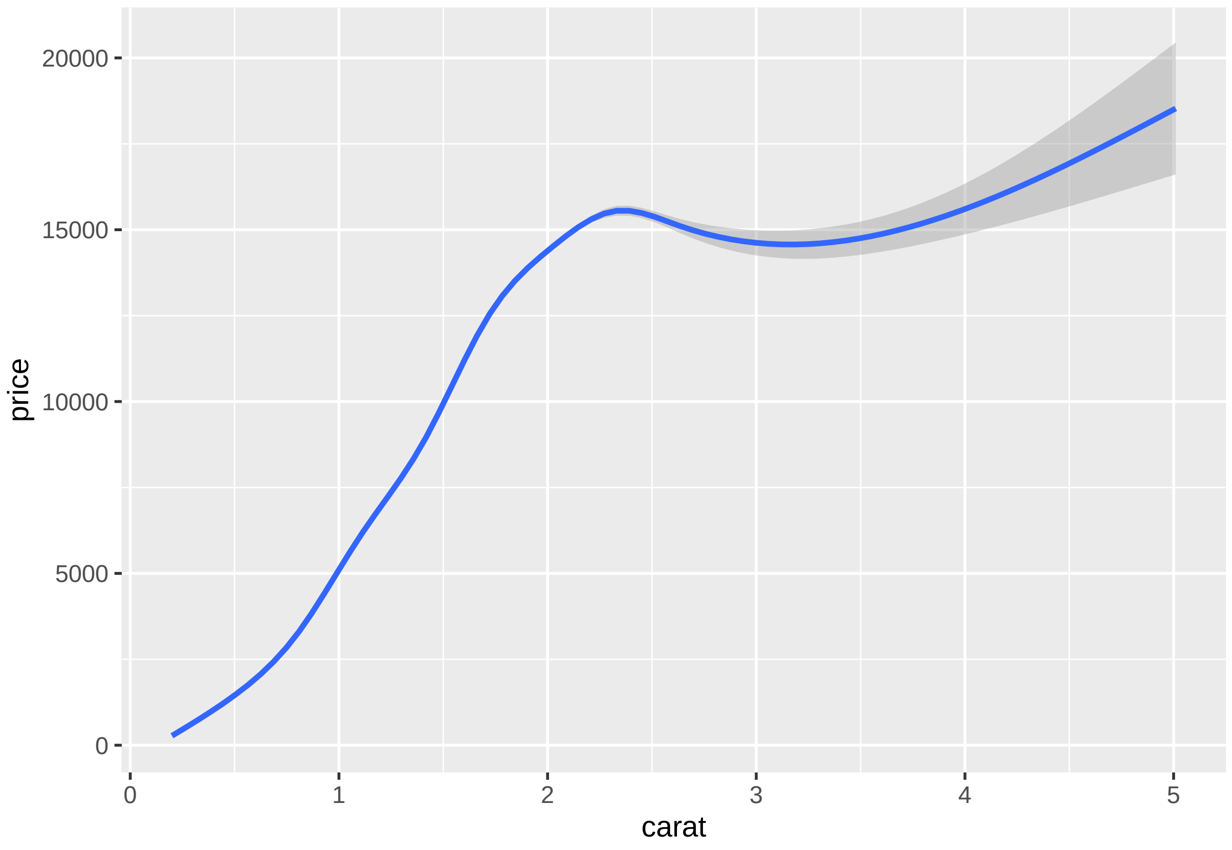
8) To understand the data set, we make a scatterplot of the dataset Diamonds, with the carat on the X axis and the price on the Y axis, as seen in the graph below.

```r
# - Make a scatter plot of the carat on the x axis and the price on the y axis.
ggplot( ___ , ___ ) + ????_??()
```



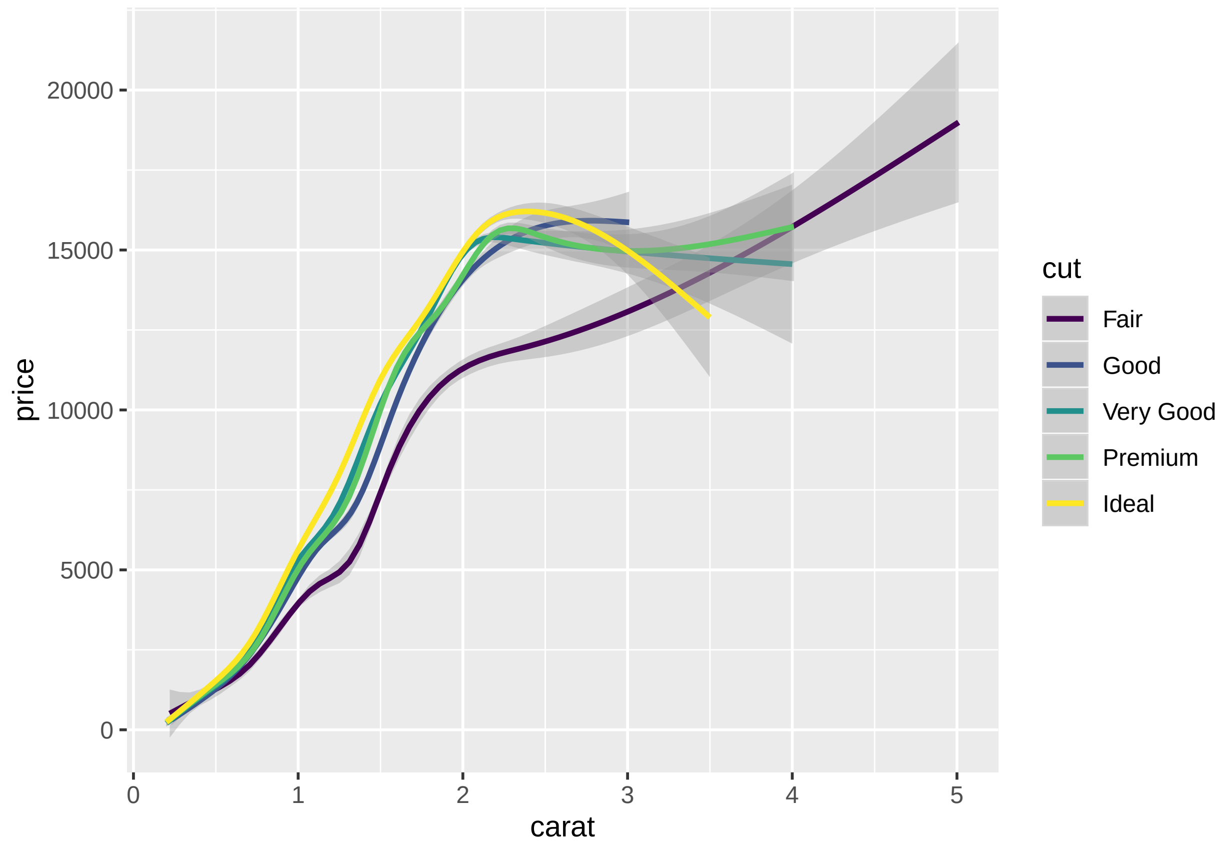
9) We now have a lot of black dots. One way to make this graph more useful is to add a smooth line through this. In the example we have added a smooth line. What is the code to do this?

```
# - Copy the above command but show only the smooth line
ggplot( ___  , ___ ) + ????_??()
```



10) Unfortunately this does not give the right information yet. To add more information we use some colour in the next graph. What is the code to get this graph?

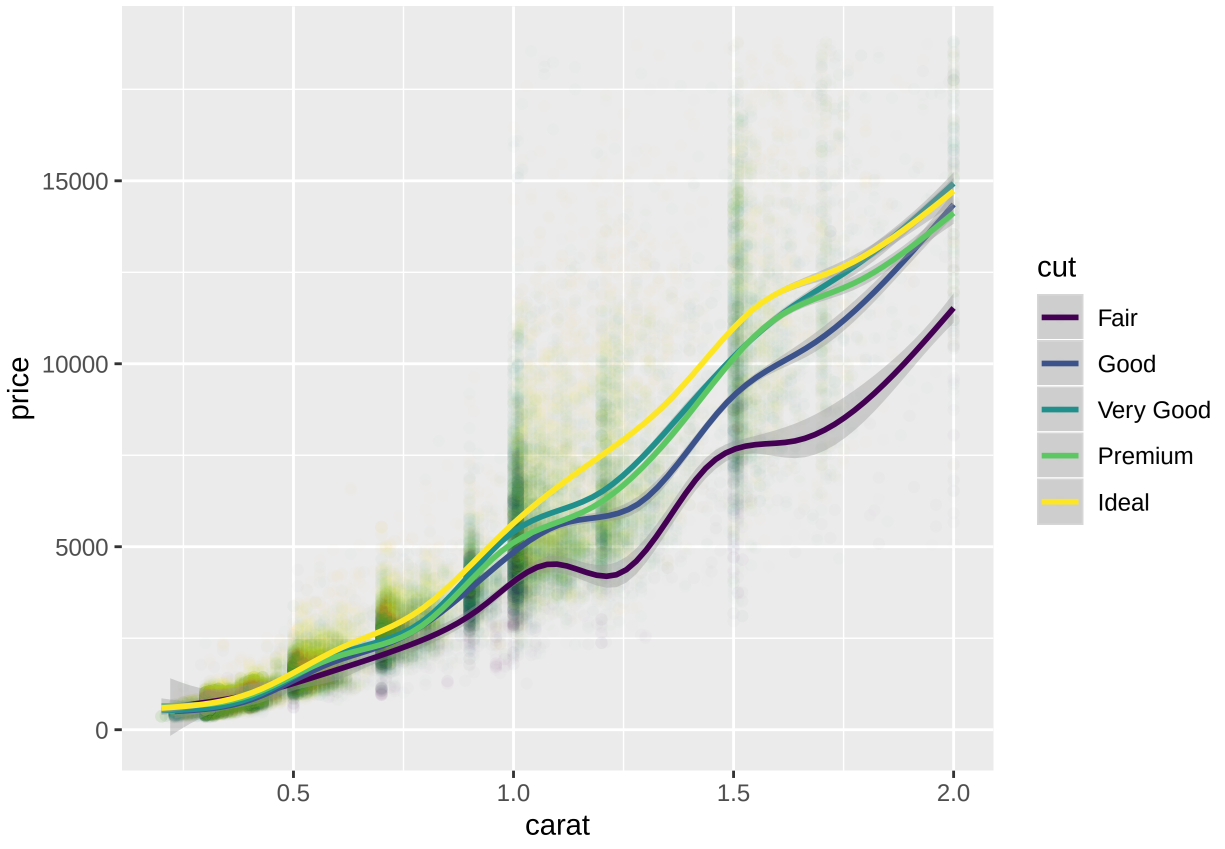
```
# - Based on the previous graph, make the separate lines for the different cuts.
ggplot( ___ , ___ , ___ ) +
  ????_??()
```



The last graph clearly shows the price for different diamond weights based on their cut. The scatter plot shows the build up of the smooth graphs. Especially the end of the graph is not so clear and can therefore be less predictive in its outcome. If we would have reduced the data point to only 2 carat we would get a completely different graph. With the filter function we used earlier we can do this:

11) Make the same graph, now only for the filtered version of only diamonds smaller than 2.0 carat.

```
# First we need to select the reduced data set of diamonds with a weight of only 2 carats.
diamond_short <- diamonds %>%
  ????( ___ )

ggplot( ___ , ___ , ___ ) +
  ????_??()
```



Part 3

Lastly we have been working with data. To get this data from a process into your project we have the readr library. For this part, let us have a look at importing data and how to tidy the data. For the first set the code will be given, the second data set you will have to import yourself.

```
bakeoff <- read_csv("bakeoff.csv", skip = 0,
                    na = c("", "NA", "UNKNOWN"))
## Parsed with column specification:
## cols(
##   series = col_double(),
##   episode = col_double(),
##   baker = col_character(),
##   signature = col_character(),
##   technical = col_double(),
##   showstopper = col_character(),
##   result = col_character(),
##   uk_airdate = col_date(format = ""),
##   us_season = col_double(),
##   us_airdate = col_date(format = "")
## )


#If we want to see the loaded data set, we can use the following two options:
#View(bakeoff)
#str(bakeoff)
```

12) Within the CSV that we have downloaded, some of the entries for showstopper are considered non applicable (na). To find all the entries that are NA we can use the filter option. Complete the filter argument for the bakeoff data to show all na for showstopper.

```
# Filter rows where showstopper is NA
bakeoff %>%
???(is.na( ___ ))
```

```
## # A tibble: 21 x 10
##    series episode baker signature technical showstopper result uk_airdate
##     <dbl>   <dbl> <chr> <chr>         <dbl> <chr>       <chr>  <date>
##  1      1       1 Edd   Caramel ~         1 <NA>        IN     2010-08-17
##  2      1       1 Jasm~ Fresh Ma~        NA <NA>        IN     2010-08-17
##  3      1       6 Mira~ Lemon Cu~        NA <NA>        RUNNE~ 2010-09-21
##  4      2       1 Ian   Apple an~        10 <NA>        IN     2011-08-16
##  5      2       1 Jason "Lemon M~         6 <NA>        IN     2011-08-16
##  6      2       1 Urva~ Cherry B~         7 <NA>        IN     2011-08-16
##  7      2       1 Yasm~ Cardamom~         5 <NA>        IN     2011-08-16
##  8      2       1 Holly "Cherry ~         1 <NA>        SB     2011-08-16
##  9      2       2 Ben   Chorizo,~         1 <NA>        IN     2011-08-23
## 10      2       2 Ian   "Stilton~         2 <NA>        IN     2011-08-23
## # ... with 11 more rows, and 2 more variables: us_season <dbl>,
## #   us_airdate <date>
```

13) with the function distinct() we can find all the unique entries for a certain column. Can you show the different values in the column of results by completing this code?

```
# View distinct results
bakeoff %>%
distinct( ___ )
```

```
## # A tibble: 6 x 1
##   result
##   <chr>
## 1 IN
## 2 OUT
## 3 RUNNER UP
## 4 WINNER
## 5 SB
## 6 LEFT
```

14) Not only knowing the distinct variable values in a column is useful, also knowing how often these appear. With which command can we find a count of all the values in the column result?

```
# Count rows for each result
  bakeoff %>%
  ????( ___ )
```

```
## # A tibble: 6 x 2
##   result        n
##   <chr>     <int>
## 1 IN          393
## 2 LEFT          1
## 3 OUT          70
## 4 RUNNER UP    16
## 5 SB           61
## 6 WINNER        8
```

15) We want to find the number of episodes per season, to get this we need to do two steps: first we make an overview of the count of series and episodes. You can use the same command as we used in the previous assignment, though now for the series and episodes.

```
bakeoff %>%
 ????( ___ ,___  )
```

```
## # A tibble: 74 x 3
##    series episode     n
##     <dbl>   <dbl> <int>
##  1      1       1    10
##  2      1       2     8
##  3      1       3     6
##  4      1       4     5
##  5      1       5     4
##  6      1       6     3
##  7      2       1    12
##  8      2       2    11
##  9      2       3    10
## 10      2       4     8
## # ... with 64 more rows
```

16) In question 15 you have made a new table. We will continue with this table in the next step. Count the number of episodes within each series. Complete the code below:

```
bakeoff %>%
  ????( ___ , ___ ) %>% ????( ___ )
```

```
## # A tibble: 8 x 2
##   series     n
##    <dbl> <int>
## 1      1     6
## 2      2     8
## 3      3    10
## 4      4    10
## 5      5    10
## 6      6    10
## 7      7    10
## 8      8    10
```

BONUS

17) Now that we know how many episodes there are in a series, we will look at the viewers that tuned in with each episode. In order to do this we need to load a new data set. We have the messy_ratings.csv file. Load this file into your notebook by completing this code:

```r
ratings <- ???.??(  ___ )

# Gather viewer columns and remove NA rows that are in the data set, you can use this code without alte
tidy_ratings <- ratings %>%
    gather(key = ___, value = ___, -series, na.rm = TRUE)
```

18) We need to clean our data to a tidy table. A tidy table means that we create a long table with the seasons as the leading value. Complete the code:

```r
# Adapt code to parse episode as a number
tidy_ratings <- ratings %>%
    gather(  ___, ____  , - ___  , na.rm = TRUE) %>%
    mutate(episode = parse_number(episode))
```

```
##   series episode viewers
## 1      1       1    2.24
## 2      2       1    3.10
## 3      3       1    3.85
## 4      4       1    6.60
## 5      5       1    8.51
## 6      6       1   11.62
```

19) We want to see only the number of viewers in episode 1 and in the last aired episode. For this you can complete the following code:

```r
# Fill in blanks to get first/last data
tidy_ratings %>%
    group_by(___) %>%
    filter(___ == ___ | episode == max(___)) %>%
    ungroup()
```

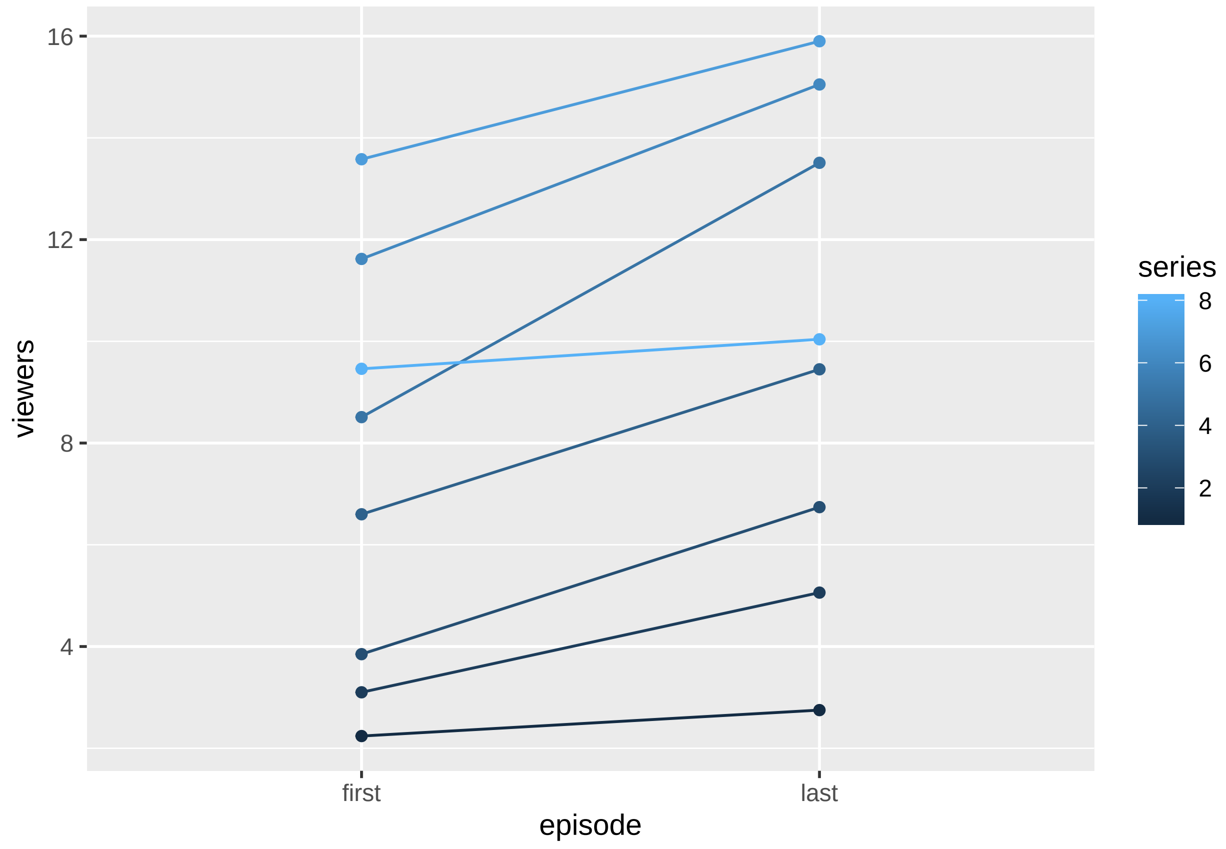
```
## # A tibble: 16 x 3
##    series episode viewers
##     <int>   <dbl>   <dbl>
##  1      1       1    2.24
##  2      2       1    3.1
##  3      3       1    3.85
##  4      4       1    6.6
##  5      5       1    8.51
##  6      6       1   11.6
##  7      7       1   13.6
##  8      8       1    9.46
##  9      1       6    2.75
## 10      2       8    5.06
## 11      3      10    6.74
## 12      4      10    9.45
## 13      5      10   13.5
## 14      6      10   15.0
## 15      7      10   15.9
## 16      8      10   10.0
```

As we now have the first and last episode in one tidy table, the next step is to recode episode 1 as "first" and the last as "last". We use the following code:

```
# Recode first/last episodes
first_last <- tidy_ratings %>%
  mutate(episode = recode(episode, `1` = "first", .default = "last"))

first_last
## # A tibble: 16 x 3
##    series episode viewers
##     <int> <chr>     <dbl>
##  1      1 first      2.24
##  2      2 first      3.1
##  3      3 first      3.85
##  4      4 first      6.6
##  5      5 first      8.51
##  6      6 first     11.6
##  7      7 first     13.6
##  8      8 first      9.46
##  9      1 last       2.75
## 10      2 last       5.06
## 11      3 last       6.74
## 12      4 last       9.45
## 13      5 last      13.5
## 14      6 last      15.0
## 15      7 last      15.9
## 16      8 last      10.0
```

20) the next step is to make a scatter plot with the different points. To show the jump in viewers a line is drawn through the points for each season.

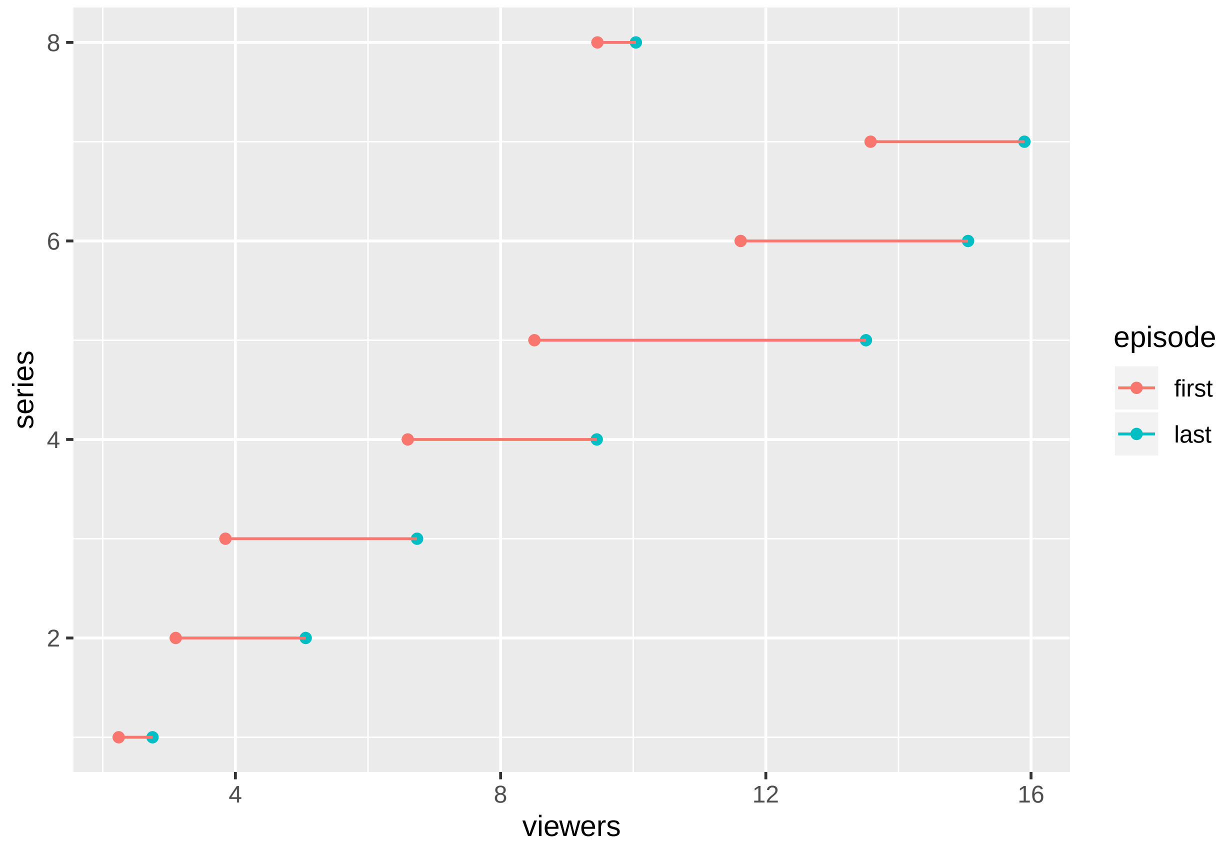
```
# Fill in to make slope chart
ggplot(___, aes(x = ___, y = ___, color = ___)) +
  geom_point() +
  geom_line(aes(group = ____))
```



```
# Let's reset the object for the first_last.
first_last <- tidy_ratings %>%
  mutate(episode = recode(episode, `1` = "first", .default = "last"))
```

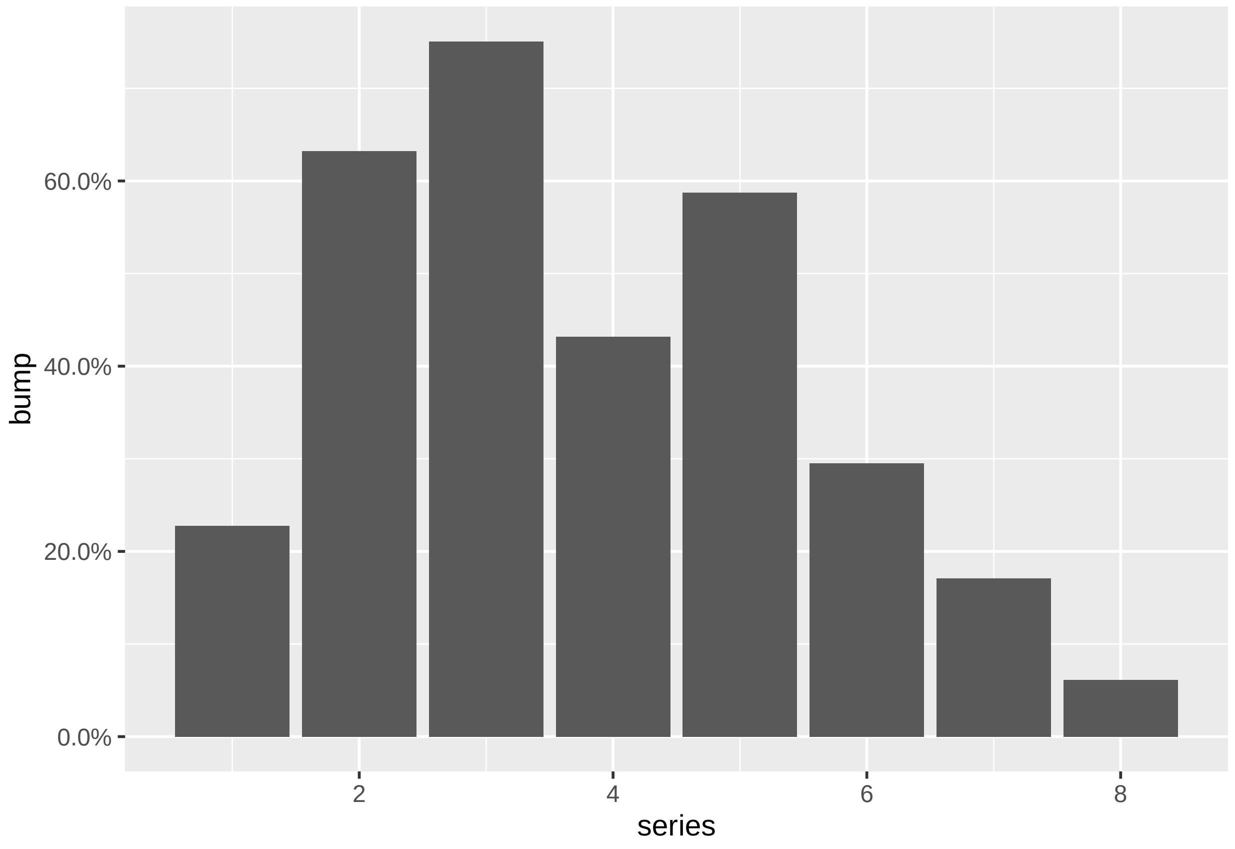
21) The next step is to flip the coordinates (hint, add the command to flip the coord. to the next chunk). Complete the code to get the next graph.

```r
# Switch the variables mapping x-axis and color
ggplot(___, aes(x = ___, y = ___, color = ___)) +
  geom_point() +
  geom_line(aes(group = ___)) +
  ____()
```



22) As the last step we want to make the graph showing the increase of viewers in each series. This will give some insight in the popularity of the show during that series. For the spread we are looking at the episodes and the viewers during that episode. We then want to know the increase in percentages, this increase needs to be calculated use the formula for percentage increase.

```
# Calculate relative increase in viewers
bump_by_series <- first_last %>%
  spread(___, ___) %>%
  mutate(bump = (___ - ___) / ____)


ggplot(bump_by_series, aes(x = ___, y = ___)) +
  geom_col() +
  scale_y_continuous(labels = scales::percent) # converts to %
```



\ \

23) to make this Notebook into a report, press “knit” in the upper bar.

End of exam

This notebook is now complete. You can make this into a HTML by pressing the “knit” option at the top of your workfield, or select the dropdown by clicking “preview” and “knit to HTML”. Or the key combination: “shift + ctrl + k” When complete you add the 2 .CSV files, your notebook and the HTML to the submission folder.

If you could not finish all questions, or if you could not knit your notebook, you don’t have to add an HTML to your submission.

The Bonus question is for the Good and Outstanding grades specifically.

Week 6 exam NJ2019, Course Based, Data Design Orienting, Semester 1.