Learning Objectives: Comparison Charts

- **Create a column chart**
- **Create a bar chart**
- **Differentiate between a column chart and a bar chart**
- **Create a line chart**
- **Determine the best comparison chart to use based on the data provided**

definition

Assumptions

- Learners are comfortable reading and importing CSV data sets, extracting relevant data into data frames, and printing that data to the console.

Limitations

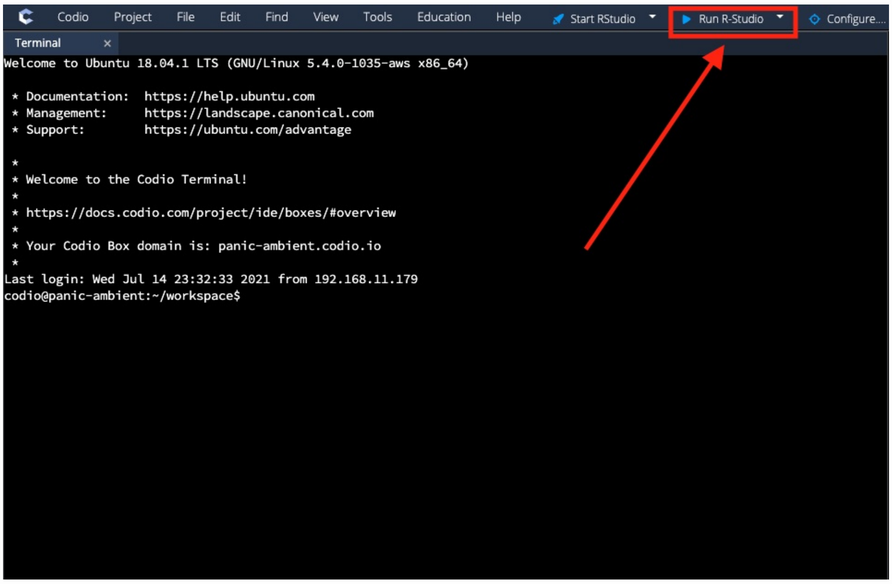
- This section will cover comparison charts in brief details only and will offer practical visualization functions for learners to start creating charts right away.

Visualization Basics

Starting RStudio

Up to this point, we've covered how to import data from CSV files, extract relevant information from them, conduct statistical functions and tests on them, and print that information. In this course, we will go over how to present that information through visualization models or **charts**.

First, let's start RStudio by clicking on the `Run R-Studio` button towards the top of the screen.

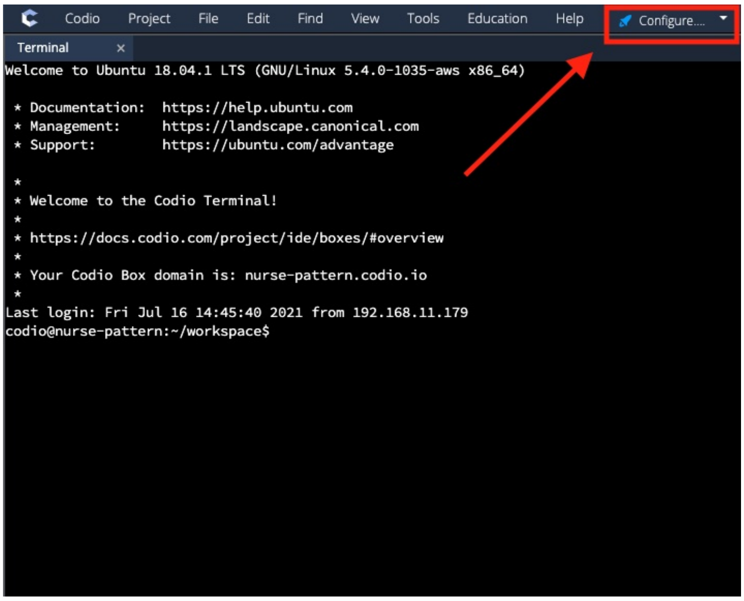


```
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: panic-ambient.codio.io
 *
Last login: Wed Jul 14 23:32:33 2021 from 192.168.11.179
codio@panic-ambient:~/workspace$
```

Click this link to enlarge the image.

▼ **Help, I don't see that button!**

If you do not see the `Run R-Studio` button, click on the `Configure...` button instead.

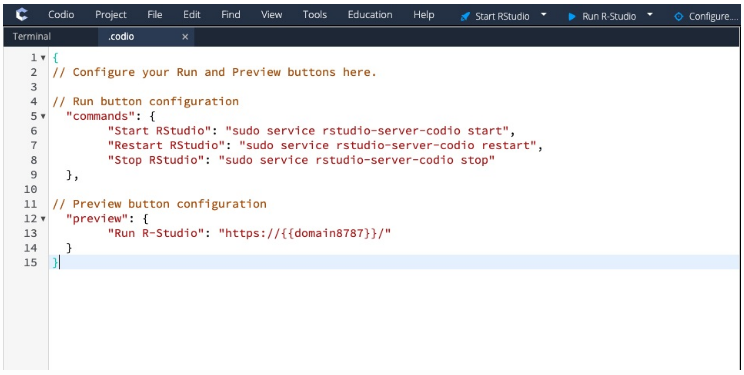


```
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: nurse-pattern.codio.io
 *
Last login: Fri Jul 16 14:45:40 2021 from 192.168.11.179
codio@nurse-pattern:~/workspace$
```

.guides/img/comp/configure

This will open up a file called `.codio`. Then copy the following code into the file:

```
{
// Configure your Run and Preview buttons here.

// Run button configuration
  "commands": {
        "Start RStudio": "sudo service rstudio-server-codio start",
        "Restart RStudio": "sudo service rstudio-server-codio restart",
        "Stop RStudio": "sudo service rstudio-server-codio stop"
  },

// Preview button configuration
  "preview": {
        "Run R-Studio": "https://{{domain8787}}/"
  }
}
```

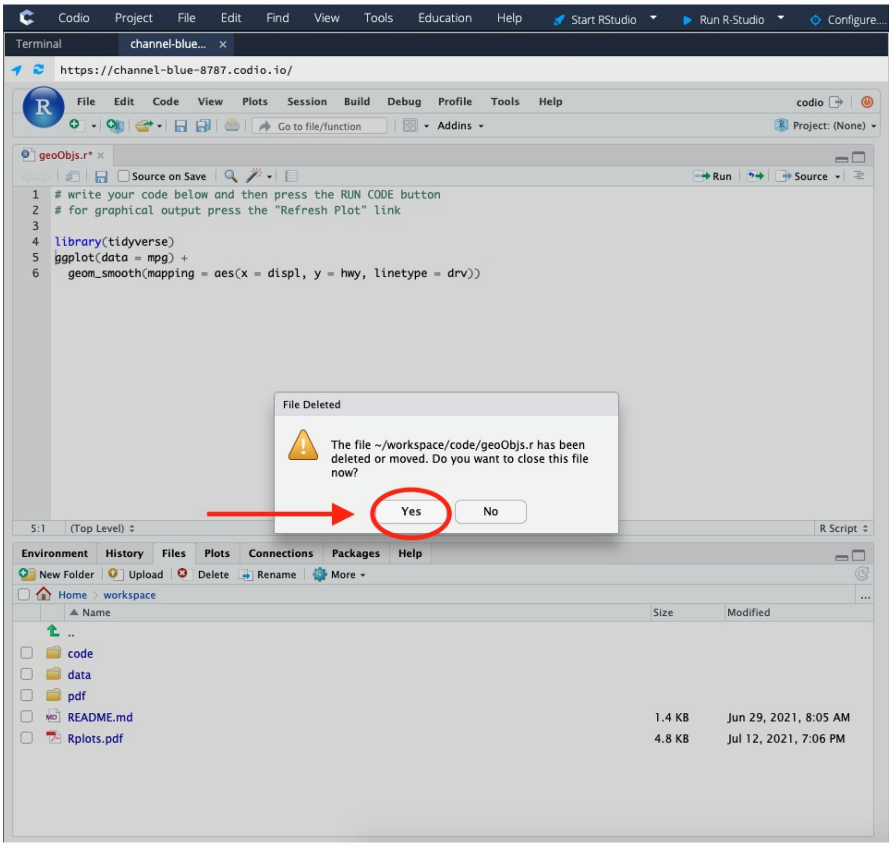


```
{
// Configure your Run and Preview buttons here.

// Run button configuration
  "commands": {
        "Start RStudio": "sudo service rstudio-server-codio start",
        "Restart RStudio": "sudo service rstudio-server-codio restart",
        "Stop RStudio": "sudo service rstudio-server-codio stop"
  },

// Preview button configuration
  "preview": {
        "Run R-Studio": "https://{{domain8787}}/"
  }
}
```

.guides/img/comp/configure-codio

Doing so will enable the `Run R-Studio` button to activate.

Once RStudio opens, you might encounter a message that says that a previous file has been deleted or moved and asks if you want to close the file. Select `Yes`.



https://channel-blue-8787.codio.io/

```
# write your code below and then press the RUN CODE button
# for graphical output press the "Refresh Plot" link

library(tidyverse)
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```

File Deleted

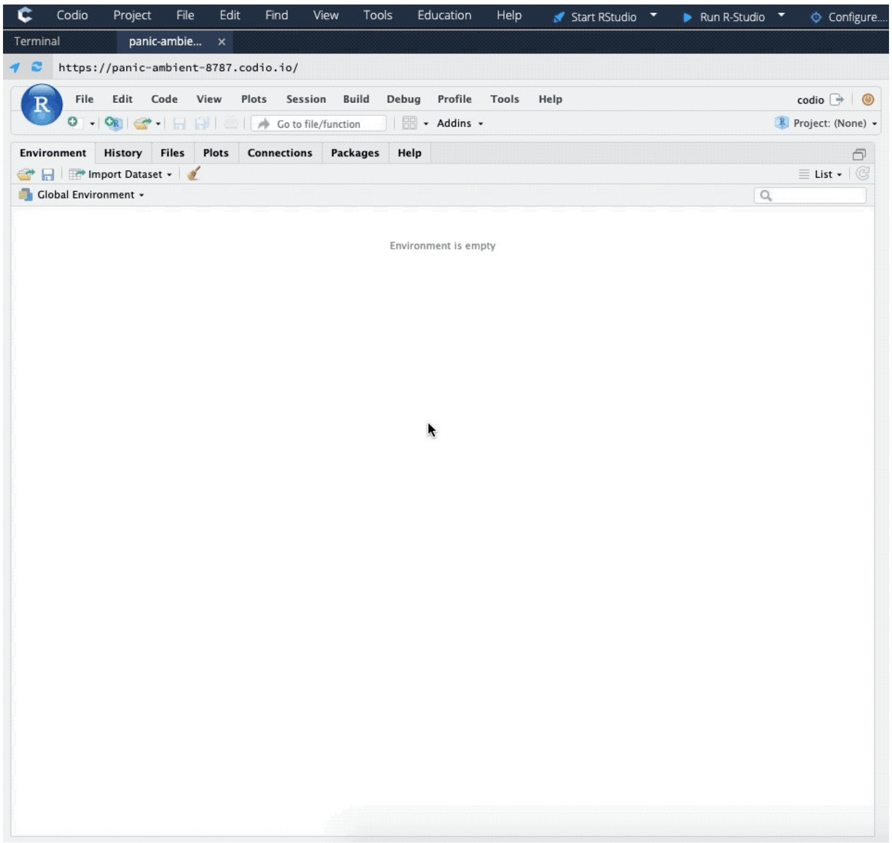
The file ~/workspace/code/geoObjs.r has been deleted or moved. Do you want to close this file now?

Yes No

Name	Size	Modified
..		
code		
data		
pdf		
README.md	1.4 KB	Jun 29, 2021, 8:05 AM
Rplots.pdf	4.8 KB	Jul 12, 2021, 7:06 PM

Click this link to enlarge the image.

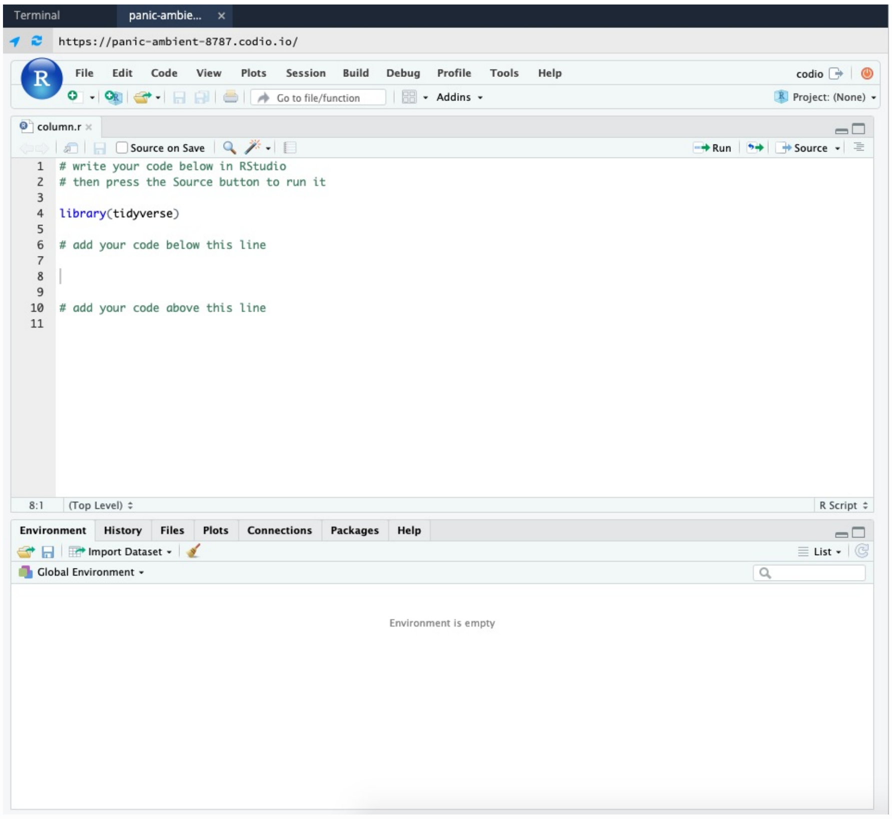
After you select `Yes`, you want to open the file `column.r` by selecting `File` from the top menu in RStudio, and then choosing `Open File...` —> `code` —> `comp` —> `column.r`



Click this link to enlarge the image.

Working in RStudio

Make sure your screen looks something like this:


Terminal
panic-ambie...
https://panic-ambient-8787.codio.io/
File Edit Code View Plots Session Build Debug Profile Tools Help
codio
Go to file/function
Addins
Project: (None)
column.r
Source on Save
Run
Source
8:1
(Top Level)
R Script
Environment History Files Plots Connections Packages Help
Import Dataset
List
Global Environment

```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)

# add your code below this line



# add your code above this line

```

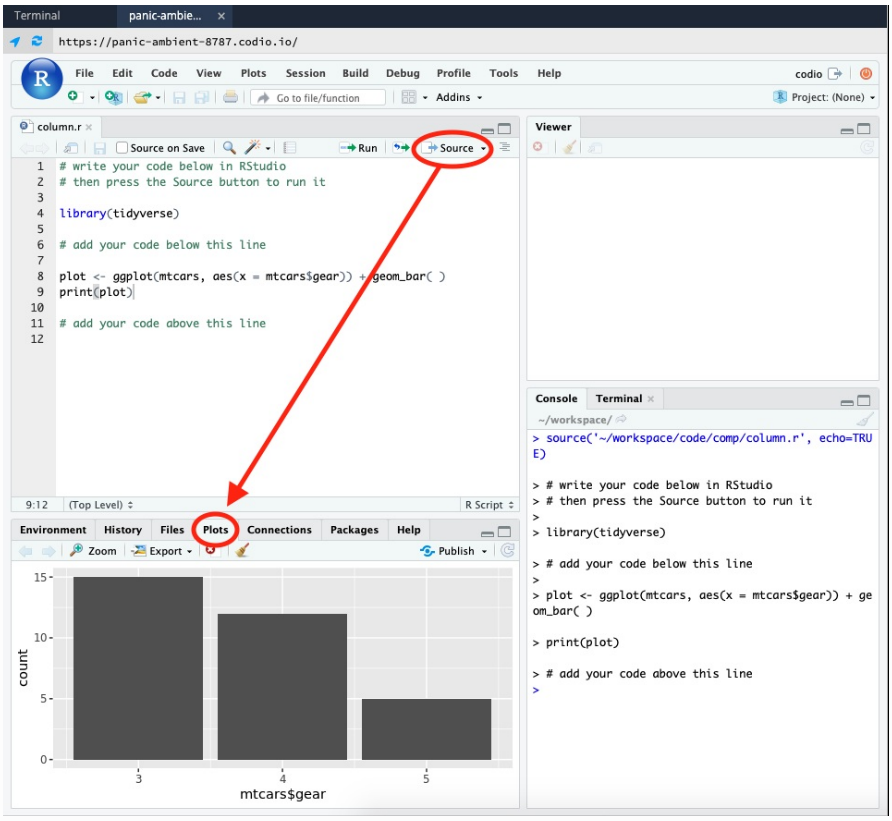
Environment is empty

Click this link to enlarge the image.

Once you've opened `column.r` successfully, you are ready to create your first visualization model. Add the following code into the text editor in RStudio as shown below. Then click the `Source` button to render the model.

```
plot <- ggplot(mtcars, aes(x = mtcars$gear)) + geom_bar( )
print(plot)
```

The image below depicts what happens when you include a visualization model function within your code and then click the `Source` button. The code will run as usual, but the `Plots` tab will activate and display your visualization model.

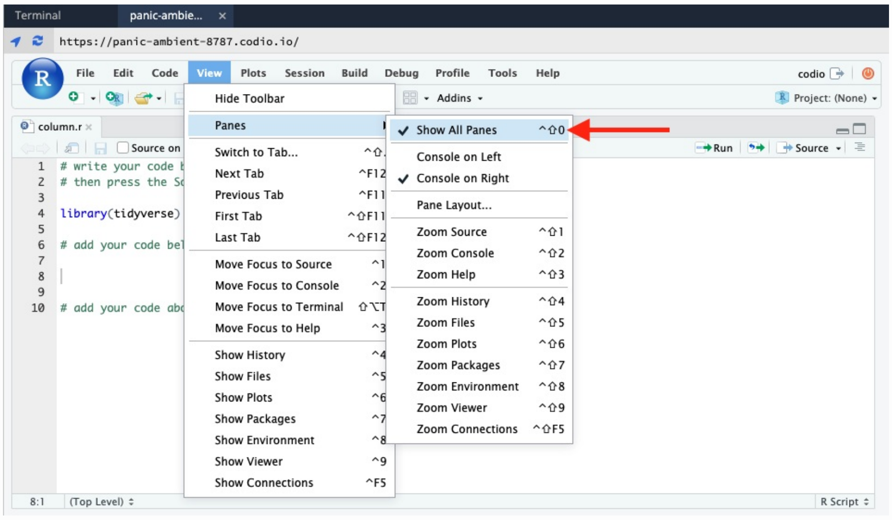


```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)

# add your code below this line

plot <- ggplot(mtcars, aes(x = mtcars$gear)) + geom_bar( )
print(plot)

# add your code above this line

```

```
> source('~/workspace/code/comp/column.r', echo=TRUE)

> # write your code below in RStudio
> # then press the Source button to run it
>
> library(tidyverse)

> # add your code below this line
>
> plot <- ggplot(mtcars, aes(x = mtcars$gear)) + geom_bar( )

> print(plot)

> # add your code above this line
>
```

Click this link to enlarge the image.

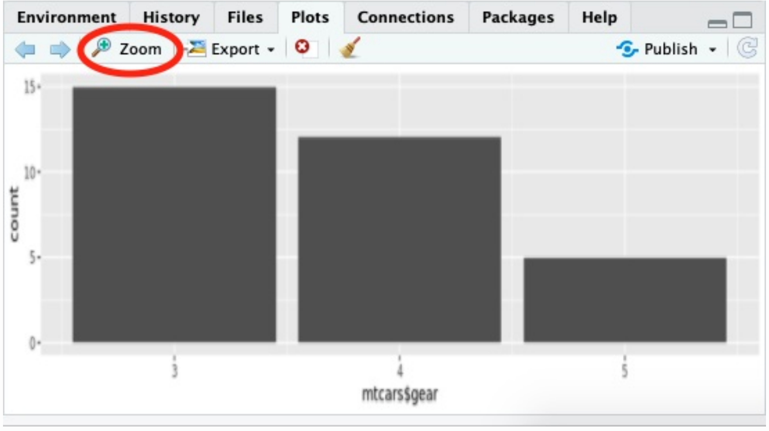
By default, only two panes are seen when you initially start RStudio, to show all available panes, go to `View` —> `Panes` —> `Show All Panes`. This will cause the `Console` and `Viewer` panes to appear.



```
# write your code b
# then press the So

library(tidyverse)

# add your code bel


# add your code abo
```

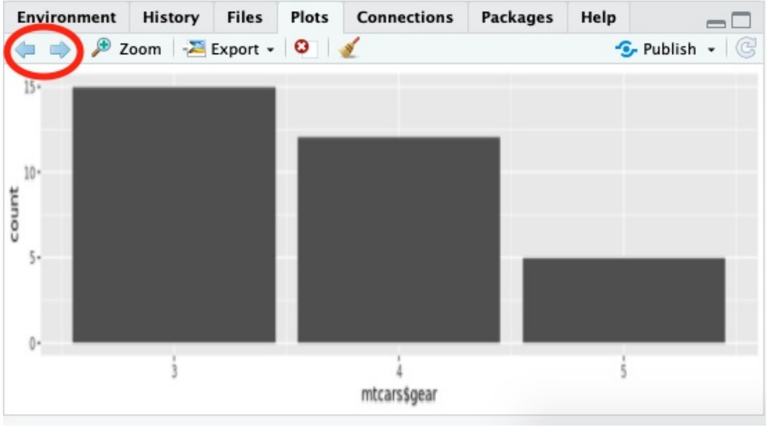
Click this link to enlarge the image.

Visualization models are useful because they can capture and present information through images. After your model has been rendered, you can click on `Zoom` to open a window that displays only your model.



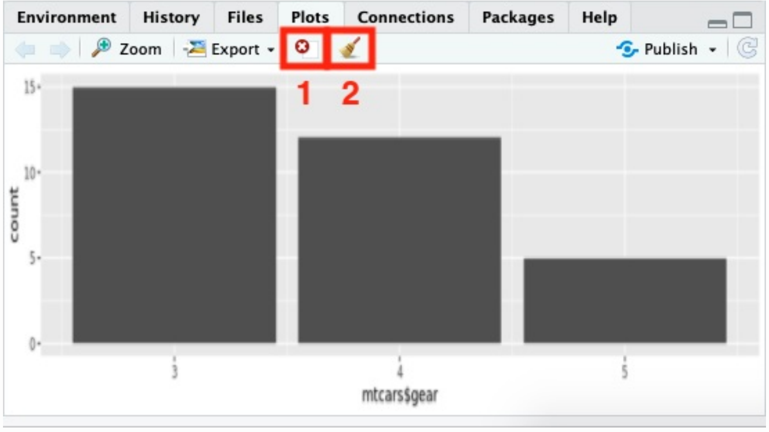
.guides/img/comp/zoom

Note that every time you generate a model using the `Source` button, RStudio will continuously store the models as images. To cycle through the images, click the left or right arrow.



.guides/img/comp/left-right

To delete the current visualization image, click on the red button with an `x` on it depicted as `1` in the image below. To delete or clear **all** model images that have been stored, click on the broom button depicted as `2` in the image below.



.guides/img/comp/clear-plot

Preparing Data for Visualization

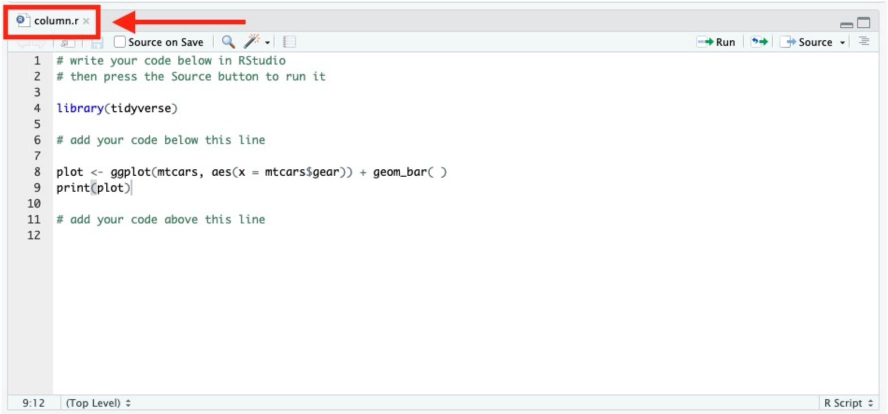
Importing Data for Visualization

Before we can create visualization models, we will need to apply some of the data wrangling techniques we learned previously. Load up RStudio by clicking on `Run R-Studio` at the top of the menu if it isn't already opened. Your current working file should be `column.r`. If it is not, follow the directions below to open it in RStudio.

info

Open the `column.r` file

Within RStudio, open the `column.r` file by selecting: `File` —> `Open File...` —> `code` —> `comp` —> `column.r`



```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)

# add your code below this line

plot <- ggplot(mtcars, aes(x = mtcars$gear)) + geom_bar( )
print(plot)

# add your code above this line
```

Click this link to enlarge the image.

First, replace the existing code in the text editor with the following command. This will import the data `all-ages.csv` and store it into `d`. This data comes from a study published in 2014 by Ben Casselman who accessed various statistics surrounding college majors (median salary, unemployment rate, etc.). Link to the study provided here.

```
d <- read.csv("data/all-ages.csv")
```

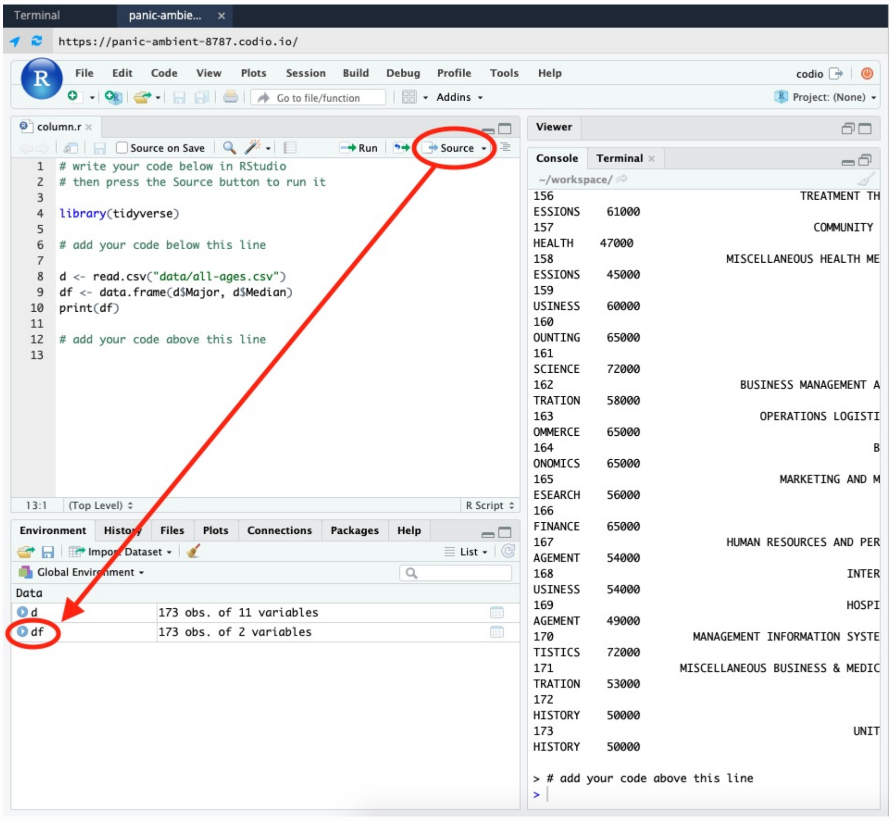
Then create a data frame with the data columns `Major` and `Median`. `Major` includes the names of the occupations categorized within the data and `Median` includes the median salaries associated with those occupations.

```
df <- data.frame(d$Major, d$Median)
```

You can print the data frame to see the data with the `print` command. Or you can view the data as a table using the `View()` command.

```
print(df)
View(df)
```

By clicking on the `Source` button and then the `d` variable within the environment tab, you'll be able to see the data that is stored.



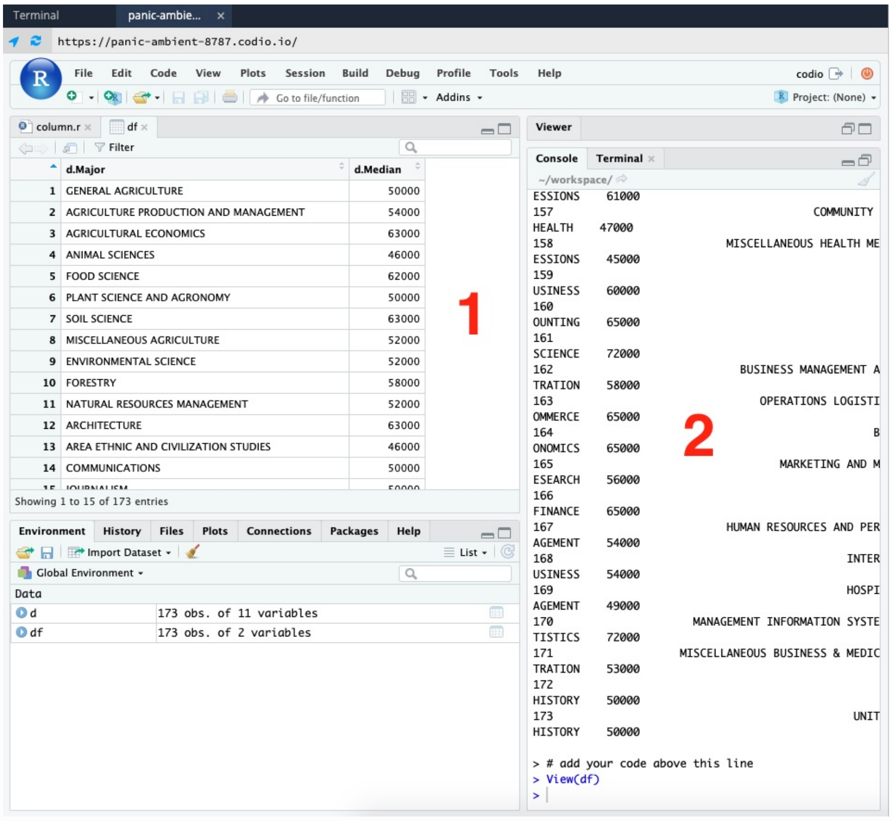
```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)

# add your code below this line

d <- read.csv("data/all-ages.csv")
df <- data.frame(d$Major, d$Median)
print(df)

# add your code above this line
```

d   173 obs. of 11 variables

df   173 obs. of 2 variables

```
156                                   TREATMENT TH
ESSIONS    61000
157                                      COMMUNITY
HEALTH   47000
158                         MISCELLANEOUS HEALTH ME
ESSIONS    45000
159
USINESS    60000
160
OUNTING    65000
161
SCIENCE    72000
162                           BUSINESS MANAGEMENT A
TRATION    58000
163                              OPERATIONS LOGISTI
OMMERCE    65000
164                                               B
ONOMICS    65000
165                                  MARKETING AND M
ESEARCH    56000
166
FINANCE    65000
167                         HUMAN RESOURCES AND PER
AGEMENT    54000
168                                           INTER
USINESS    54000
169                                           HOSPI
AGEMENT    49000
170                    MANAGEMENT INFORMATION SYSTE
TISTICS    72000
171                  MISCELLANEOUS BUSINESS & MEDIC
TRATION    53000
172
HISTORY    50000
173                                            UNIT
HISTORY    50000

> # add your code above this line
>
```

Click this link to enlarge the image.


Terminal
panic-ambie...
https://panic-ambient-8787.codio.io/
File Edit Code View Plots Session Build Debug Profile Tools Help
codio
Go to file/function
Addins
Project: (None)
column.r
df
Filter
1
Viewer
Console
Terminal
~/workspace/
2
Environment History Files Plots Connections Packages Help
Import Dataset
List
Global Environment

	d.Major	d.Median
1	GENERAL AGRICULTURE	50000
2	AGRICULTURE PRODUCTION AND MANAGEMENT	54000
3	AGRICULTURAL ECONOMICS	63000
4	ANIMAL SCIENCES	46000
5	FOOD SCIENCE	62000
6	PLANT SCIENCE AND AGRONOMY	50000
7	SOIL SCIENCE	63000
8	MISCELLANEOUS AGRICULTURE	52000
9	ENVIRONMENTAL SCIENCE	52000
10	FORESTRY	58000
11	NATURAL RESOURCES MANAGEMENT	52000
12	ARCHITECTURE	63000
13	AREA ETHNIC AND CIVILIZATION STUDIES	46000
14	COMMUNICATIONS	50000
15	JOURNALISM	50000

Showing 1 to 15 of 173 entries

```
Data
d     173 obs. of 11 variables
df    173 obs. of 2 variables
```

```
ESSIONS    61000
157                                     COMMUNITY
HEALTH    47000
158                       MISCELLANEOUS HEALTH ME
ESSIONS    45000
159
USINESS    60000
160
OUNTING    65000
161
SCIENCE    72000
162                         BUSINESS MANAGEMENT A
TRATION    58000
163                            OPERATIONS LOGISTI
OMMERCE    65000
164                                             B
ONOMICS    65000
165                               MARKETING AND M
ESEARCH    56000
166
FINANCE    65000
167                       HUMAN RESOURCES AND PER
AGEMENT    54000
168                                         INTER
USINESS    54000
169                                         HOSPI
AGEMENT    49000
170                  MANAGEMENT INFORMATION SYSTE
TISTICS    72000
171                MISCELLANEOUS BUSINESS & MEDIC
TRATION    53000
172
HISTORY    50000
173                                          UNIT
HISTORY    50000

> # add your code above this line
> View(df)
> 
```

Click this link to enlarge the image.

The images above show how the data is stored (labeled 1) and how it prints (labeled 2).

Extracting the Essential Data

Note that there is a lot of data that is stored in the data frame `df`. Let's condense it by rearranging and sorting it so that only the top 10 highest median salary occupation majors are printed (in descending order) and stored in `top_10`. First let's rename the column labels within our data frame `df` and then store those new labels back into `df`.

```
df <- rename(df, Median_Salary = d.Median, Major = d.Major)
```

The basic syntax for the `rename()` function is shown above where:
* `df` represents the data frame containing the column name you want to change
* `Median_Salary` represents the new name you want the column to adopt
* `d.Median` represents the old name you want to change
* additional column name changes can be performed by including a comma `,`

Next, let’s sort the data frame `df` so that it organizes the data in **descending** order based on the `Median_Salary`. Remember to include the minus sign `-` in front of the column name to sort in descending order. Then store this new data frame as `df_sorted`.

```
df_sorted <- df[order(-df$Median_Salary),]
```

To check if the sorting is done successfully, print the sorted data frame:

```
print(df_sorted)
```

The beginning of the sorted data frame should something look like this (scroll right to see more data):

```
        Major Median_Salary
60                                              PETROLEUM
        ENGINEERING        125000
155               PHARMACY PHARMACEUTICAL SCIENCES AND
        ADMINISTRATION        106000
58                          NAVAL ARCHITECTURE AND MARINE
        ENGINEERING         97000
56                                          METALLURGICAL
        ENGINEERING         96000
59                                                NUCLEAR
        ENGINEERING         95000
57                                     MINING AND MINERAL
        ENGINEERING         92000
98                                   MATHEMATICS AND COMPUTER
        SCIENCE         92000
49                                             ELECTRICAL
        ENGINEERING         88000
46                                               CHEMICAL
        ENGINEERING         86000
52                             GEOLOGICAL AND GEOPHYSICAL
        ENGINEERING         85000
42                                              AEROSPACE
        ENGINEERING         80000
48                                               COMPUTER
        ENGINEERING         80000
55                                             MECHANICAL
        ENGINEERING         80000
105                                        ASTRONOMY AND
        ASTROPHYSICS         80000
21                                                   COMPUTER
        SCIENCE         78000
44                                          ARCHITECTURAL
        ENGINEERING         78000
47                                                  CIVIL
        ENGINEERING         78000
```

Now, to store just the top 10 rows of data, we can use the `head()` function and specify `n` to be `10`. Then store that data in `top_10`.

```r
top_10 <- head(df_sorted, n = 10)
```

Print `top_10` to see the output.

```r
print(top_10)
```

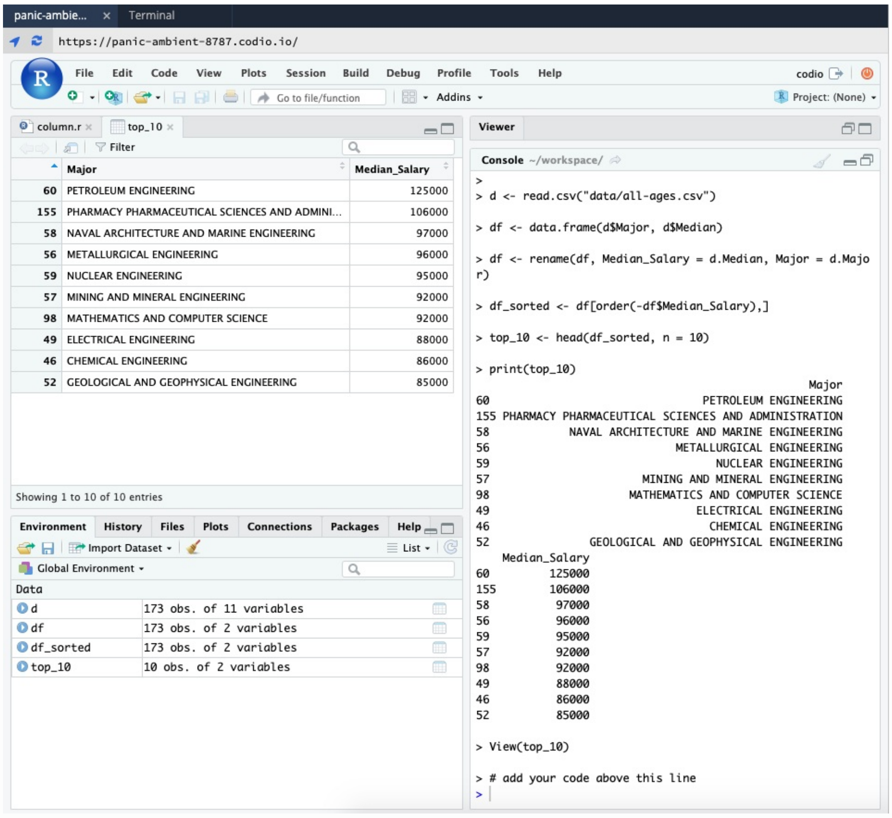
Result:

```
                                                  Major
        Median_Salary
60                                PETROLEUM ENGINEERING
        125000
155 PHARMACY PHARMACEUTICAL SCIENCES AND ADMINISTRATION
        106000
58            NAVAL ARCHITECTURE AND MARINE ENGINEERING
        97000
56                            METALLURGICAL ENGINEERING
        96000
59                                  NUCLEAR ENGINEERING
        95000
57                       MINING AND MINERAL ENGINEERING
        92000
98                     MATHEMATICS AND COMPUTER SCIENCE
        92000
49                               ELECTRICAL ENGINEERING
        88000
46                                 CHEMICAL ENGINEERING
        86000
52               GEOLOGICAL AND GEOPHYSICAL ENGINEERING
        85000
```

Now you’ll see that only the top 10 majors with the highest median salaries are displayed.

To view the data as a table for better organization, use the `View()` command.

```r
View(top_10)
```



panic-ambie... ×
Terminal
https://panic-ambient-8787.codio.io/
File Edit Code View Plots Session Build Debug Profile Tools Help
codio
Go to file/function
Addins
Project: (None)
column.r ×
top_10 ×
Filter
Viewer
Console ~/workspace/
Environment History Files Plots Connections Packages Help
Import Dataset
List
Global Environment

	Major	Median_Salary
60	PETROLEUM ENGINEERING	125000
155	PHARMACY PHARMACEUTICAL SCIENCES AND ADMINI...	106000
58	NAVAL ARCHITECTURE AND MARINE ENGINEERING	97000
56	METALLURGICAL ENGINEERING	96000
59	NUCLEAR ENGINEERING	95000
57	MINING AND MINERAL ENGINEERING	92000
98	MATHEMATICS AND COMPUTER SCIENCE	92000
49	ELECTRICAL ENGINEERING	88000
46	CHEMICAL ENGINEERING	86000
52	GEOLOGICAL AND GEOPHYSICAL ENGINEERING	85000

Showing 1 to 10 of 10 entries

Data	
d	173 obs. of 11 variables
df	173 obs. of 2 variables
df_sorted	173 obs. of 2 variables
top_10	10 obs. of 2 variables

```
>
> d <- read.csv("data/all-ages.csv")

> df <- data.frame(d$Major, d$Median)

> df <- rename(df, Median_Salary = d.Median, Major = d.Majo
r)

> df_sorted <- df[order(-df$Median_Salary),]

> top_10 <- head(df_sorted, n = 10)

> print(top_10)
                                                  Major
60                                PETROLEUM ENGINEERING
155 PHARMACY PHARMACEUTICAL SCIENCES AND ADMINISTRATION
58            NAVAL ARCHITECTURE AND MARINE ENGINEERING
56                            METALLURGICAL ENGINEERING
59                                  NUCLEAR ENGINEERING
57                       MINING AND MINERAL ENGINEERING
98                     MATHEMATICS AND COMPUTER SCIENCE
49                               ELECTRICAL ENGINEERING
46                                 CHEMICAL ENGINEERING
52               GEOLOGICAL AND GEOPHYSICAL ENGINEERING
    Median_Salary
60         125000
155        106000
58          97000
56          96000
59          95000
57          92000
98          92000
49          88000
46          86000
52          85000

> View(top_10)

> # add your code above this line
>
```

Click this link to enlarge the image.

Being able to wrangle or condense the data into the only most essential information is important for data visualization.

Column Charts

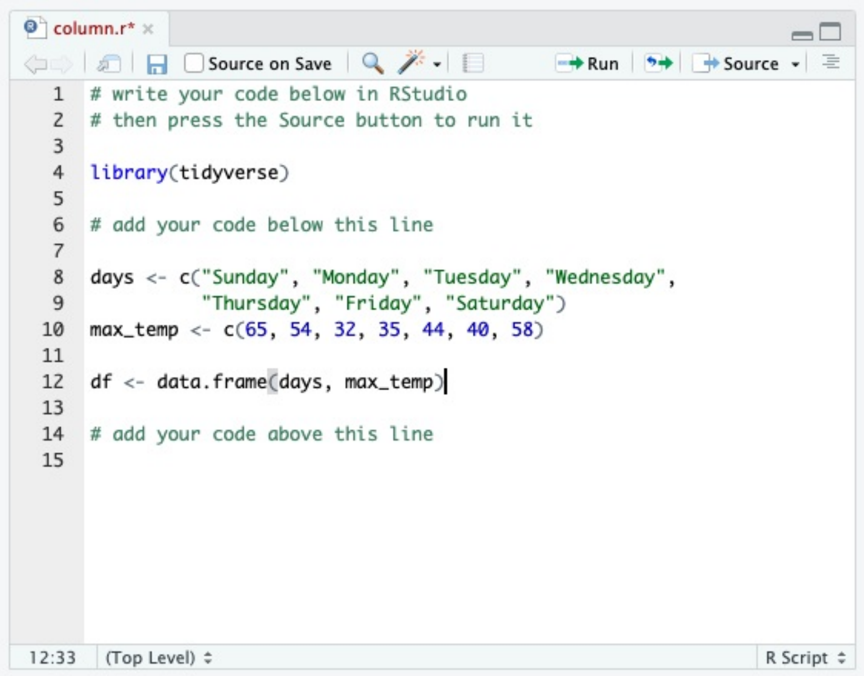
Creating Column Charts

We will revisit the data we wrangled from the previous pages later. For now, clear your code within the RStudio text editor. Then add the following code:

```
days <- c("Sunday", "Monday", "Tuesday", "Wednesday",
          "Thursday", "Friday", "Saturday")
max_temp <- c(65, 54, 32, 35, 44, 40, 58)

df <- data.frame(days, max_temp)
```

The code above creates two vectors `days` and `max_temp` and then stores them in a data frame `df`. This data can then be used to create a **column chart**. Column charts are one of the most common visualizations because they are very easy to make. Often, column charts and **bar charts** are used interchangeably but for the purposes of this course, we'll refer to column charts as charts containing **vertical** bars and bar charts as charts containing **horizontal** bars.

Make sure your `column.r` file looks like this now:



```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)

# add your code below this line

days <- c("Sunday", "Monday", "Tuesday", "Wednesday",
          "Thursday", "Friday", "Saturday")
max_temp <- c(65, 54, 32, 35, 44, 40, 58)

df <- data.frame(days, max_temp)

# add your code above this line
```

The basic syntax for column charts is:

```
ggplot($data, aes(x = $x, y = $y)) +
  geom_bar(stat = "identity")
```

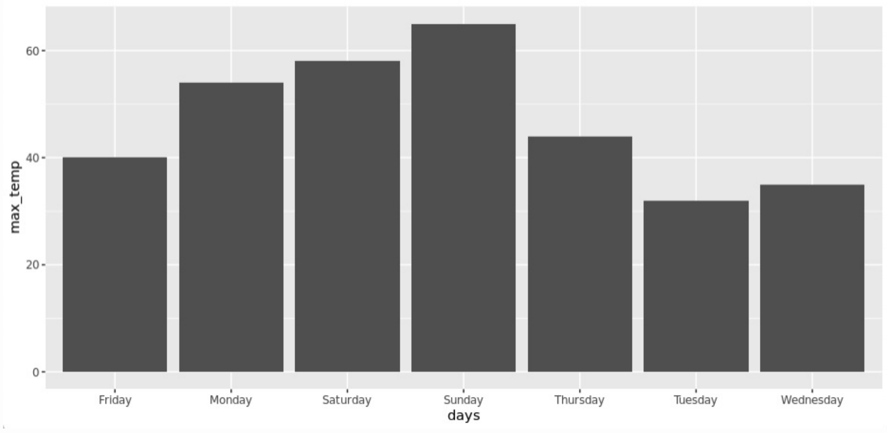
- `$data` represents the data frame being used.
- `$x` represents the data on the x-axis.
- `$y` represents the data on the y-axis.

Now we can replace the components above to create our column chart like so:

```
chart <- ggplot(df, aes(x = days, y = max_temp)) +
  geom_bar(stat = "identity")
print(chart)
```

Click the `Source` button to display your chart.

Chart Result:



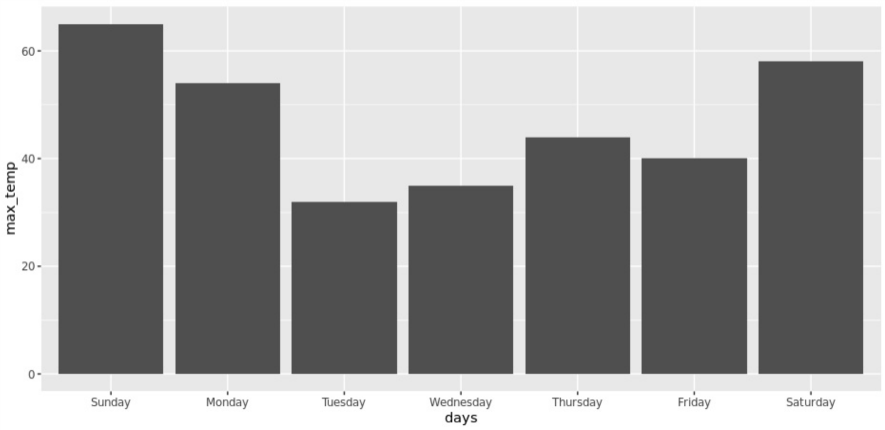
Click this link to enlarge the image.

Organizing the Chart Data

You’ll notice in the chart that the data is oddly organized where the x-axis sorts the `days` of the week in alphabetical order. To maintain the order of the data based on their occurrence (`Sunday` is first and `Saturday` is last), use `df$days <- factor(df$days, levels = df$days)` **before** plotting the chart. The `factor()` function helps to maintain the order of data within a column or vector.

```r
days <- c("Sunday", "Monday", "Tuesday", "Wednesday",
          "Thursday", "Friday", "Saturday")
max_temp <- c(65, 54, 32, 35, 44, 40, 58)

df <- data.frame(days, max_temp)
df$days <- factor(df$days, levels = df$days)

chart <- ggplot(df, aes(x = days, y = max_temp)) +
  geom_bar(stat = "identity")
print(chart)
```

Chart Result:



Click this link to enlarge the image.

Once the `factor()` function is applied to `days`, the data retains its order of occurrence.

Labeling the Chart Data

To change the x-axis and y-axis labels, and to give the chart a title, you must **add** the `labs()` function like below:

```r
labs(title = "Maximum Temperatures of the Week",
  x = "Day of the Week",
  y = "Maximum Temperature")
```

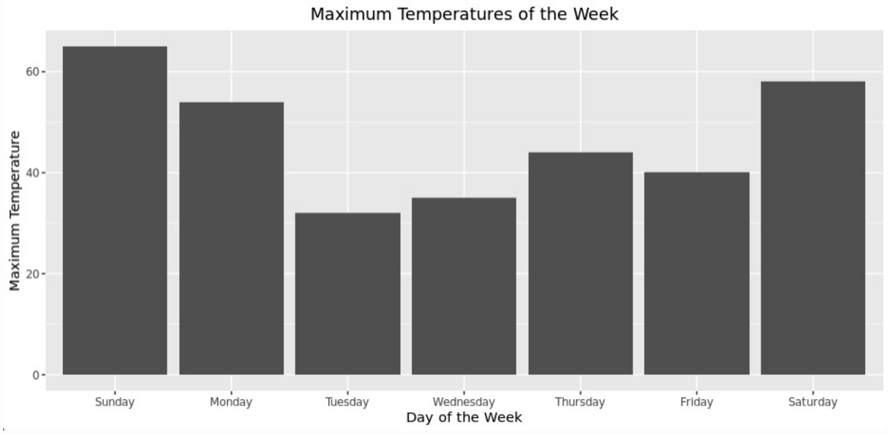
`labs` stands for labels where `title` becomes the title of the chart, `x` becomes the x-axis label, and `y` becomes the y-axis label.

Click `Source` on the entire code below to see the newest column chart. **Notice** how `labs` has been added using the `+` symbol.

```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)
theme_update(plot.title = element_text(hjust = 0.5))

# add your code below this line

days <- c("Sunday", "Monday", "Tuesday", "Wednesday",
          "Thursday", "Friday", "Saturday")
max_temp <- c(65, 54, 32, 35, 44, 40, 58)

df <- data.frame(days, max_temp)
df$days <- factor(df$days, levels = df$days)

chart <- ggplot(df, aes(x = days, y = max_temp)) +
  geom_bar(stat = "identity") +
  labs(title = "Maximum Temperatures of the Week",
    x = "Day of the Week",
    y = "Maximum Temperature")
print(chart)

# add your code above this line
```

Note: By default, titles in RStudio are left-aligned. To center the title, `theme_update(plot.title = element_text(hjust = 0.5))` was included in the header of the provided code underneath `library(tidyverse)` to center all generated titles.

Chart Result:



Click this link to enlarge the image.

Bar Charts

Creating Bar Charts

The next chart we will learn about is the **bar chart**. Remember that we differentiate between a column chart and a bar chart based on the orientation of their bars. Column charts contain **vertical** bars while bar charts contain **horizontal** bars. Follow the directions below to open up the `bar.r` file in RStudio.

info

Open the `bar.r` file

Within RStudio, open the `bar.r` file by selecting: `File` —> `Open File...` —> `code` —> `comp` —> `bar.r`

We will revisit the data we wrangled earlier by re-adding the following code into the text editor:

```
d <- read.csv("data/all-ages.csv")
df <- data.frame(d$Major, d$Median)
df <- rename(df, Median_Salary = d.Median, Major = d.Major)
df_sorted <- df[order(-df$Median_Salary),]
top_10 <- head(df_sorted, n = 10)
```

The basic syntax for bar charts is:

```
ggplot($data, aes(x = $vertical, y = $horizontal)) +
  geom_bar(stat = "identity") +
  coord_flip()
```

- `$data` represents the data frame being used.
- `$vertical` represents the data on the x-axis.
- `$horizontal` represents the data on the y-axis.

important

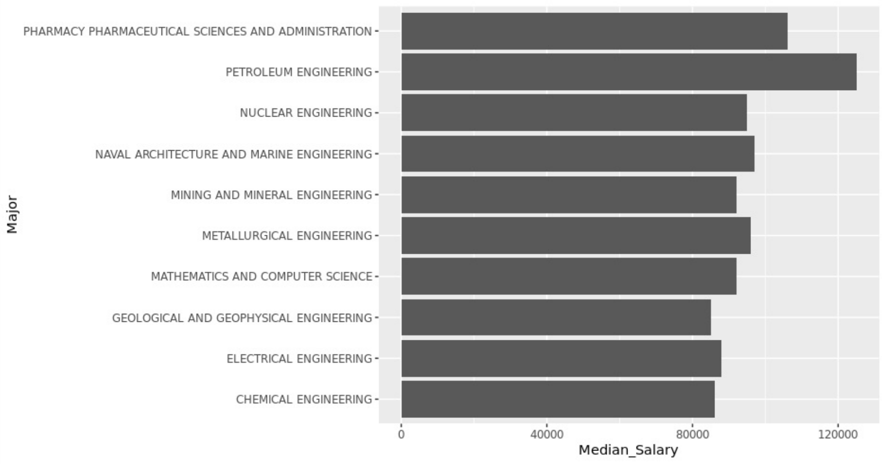
IMPORTANT

Note how the syntax for bar charts is almost identical to the one for column charts. The only difference is the addition of the function `coord_flip()` which will flip the x and y axes. When talking about bar charts we will refer to the x-axis as the y-axis and the y-axis as the x-axis.

Add on the following code into the text editor and then click the `Source` button to see the result.

```
chart <- ggplot(top_10, aes(x = Major, y = Median_Salary)) +
  geom_bar(stat = "identity") +
  coord_flip()
print(chart)
```

Chart Result:



Click this link to enlarge the image.

Organizing the Chart Data

Notice how the resulting chart does not organize the data according to the median salaries. Instead, the system organizes the occupation majors in alphabetical order along the y-axis in an ascending fashion (bottom to top).

challenge

Try this variation(s):

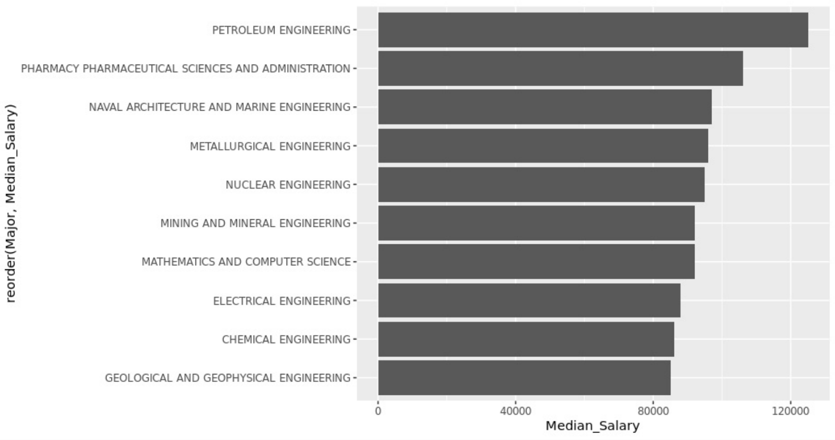
- Replace

```
chart <- ggplot(top_10, aes(x = Major, y = Median_Salary)) +
  geom_bar(stat = "identity") +
  coord_flip()
print(chart)
```

with

```
chart <- ggplot(top_10, aes(x = reorder(Major,
        Median_Salary), y = Median_Salary)) +
  geom_bar(stat = "identity") +
  coord_flip()
print(chart)
```

▼ **Chart Result**



Click this link to enlarge the image.

To organize `Major` according to the order of `Median_Salary`, we substitute `x = Major` with `x = reorder(Major, Median_Salary)`. Doing so causes `Major` to be organized in descending order of highest median salary to lowest (`Median_Salary`).

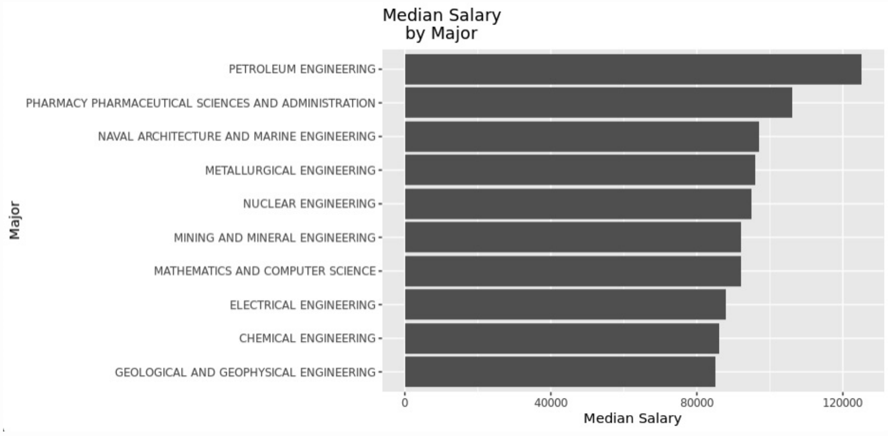
Labeling the Chart Data

To rename the x-axis and y-axis labels and to provide a title for the bar chart, replace `chart` with:

```
chart <- ggplot(top_10, aes(x = reorder(Major, Median_Salary), y = Median_Salary)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Median Salary \n    by Major",
       x = "Major",
       y = "Median Salary")
print(chart)
```

Again, the syntax is identical to the one for column charts with the exception of `coord_flip()` being implemented. However, the x-axis and y-axis labels are **NOT** flipped. This is important to keep in mind when working with bar charts.

Chart Result:



Click this link to enlarge the image.

Note that the `\n` within `title` creates a newline. In addition, you can adjust the title by adding whitespaces. Or you can include `theme_update(plot.title = element_text(hjust = 0.5))` in the header of your code underneath `library(tidyverse)` to center all titles just like how it was done in column charts.

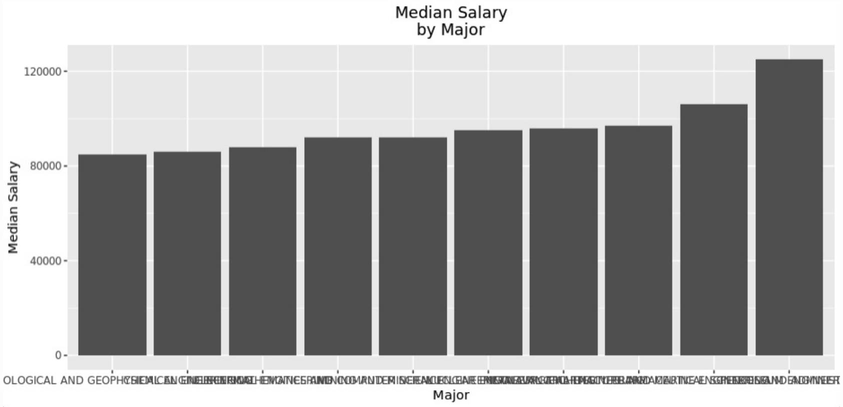
challenge

Try this variation(s):

- Remove `coord_flip() +` and modify the code so that it looks like this:

```
# write your code below in RStudio
# then press the Source button to run it

library(tidyverse)
theme_update(plot.title = element_text(hjust = 0.5))

# add your code below this line

d <- read.csv("data/all-ages.csv")
df <- data.frame(d$Major, d$Median)
df <- rename(df, Median_Salary = d.Median, Major = d.Major)
df_sorted <- df[order(-df$Median_Salary),]
top_10 <- head(df_sorted, n = 10)

chart <- ggplot(top_10, aes(x = reorder(Major,
        Median_Salary), y = Median_Salary)) +
  geom_bar(stat = "identity") +
  labs(title = "Median Salary\nby Major",
       x = "Major",
       y = "Median Salary")
print(chart)

# add your code above this line
```

▼ Chart Result



Click this link to enlarge the image.

important

IMPORTANT

The **advantage** of using bar charts over column charts is that bar charts can display the categorical variable names more efficiently. When `coord_flip()` was removed in the example above, the variable names overlapped causing them to be unreadable. This would require the user to either rename the data, or they can simply use a bar chart instead.

Line Charts

Creating Line Charts

Follow the directions below to open up the `line.r` file in RStudio.

info

Open the `line.r` file

Within RStudio, open the `line.r` file by selecting: `File` —> `Open File...` —> `code` —> `comp` —> `line.r`

Data Import

```
d <- read.csv("data/all-ages.csv")
df <- data.frame(d$Major, d$Unemployment_rate)
df <- rename(df, Unemployment_Rate = d.Unemployment_rate, Major = d.Major)
df_sorted <- df[order(-df$Unemployment_Rate),]
top_10 <- head(df_sorted, n = 10)
```

The data above showcases the top 10 highest unemployment rates in 2014 from Ben Casselman's study. Notice how we swapped the data on median salaries with the data on unemployment rates.

The basic syntax for line charts is:

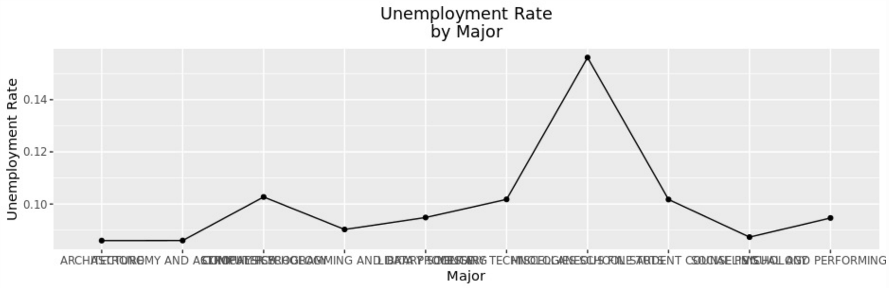
```
ggplot($data, aes(x = $vertical, y = $horizontal, group = 1)) +
  geom_line() +
  geom_point()
```

- `$data` represents the data frame being used.
- `$vertical` represents the data on the x-axis.
- `$horizontal` represents the data on the y-axis.
- `group = 1` enables the plots to be connected by a line.

Add on the following code into the text editor and then click the `Source` button to see the result.

```
chart <- ggplot(top_10, aes(x = Major, y = Unemployment_Rate,
        group = 1)) +
  geom_line() +
  geom_point() +
  labs(title = "Unemployment Rate\nby Major",
    x = "Major",
    y = "Unemployment Rate")
print(chart)
```

Plot Result:



Click this link to enlarge the image.

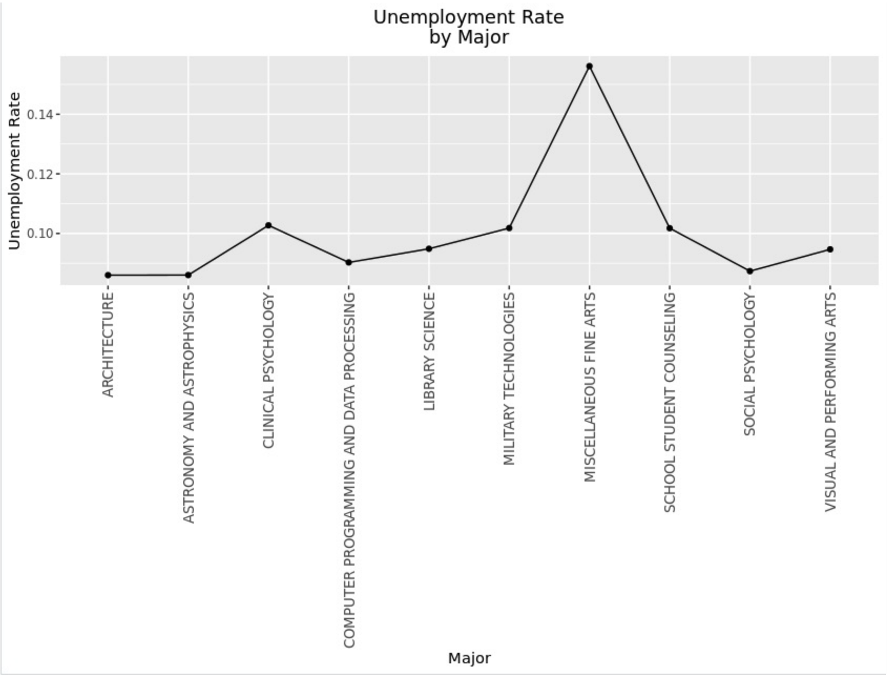
Unfortunately, the x-axis labels overlap each other making it difficult to read. One strategy is to rename each x-axis label prior to printing the chart. Alternatively, we can make use of the functions `theme()` and `scale_x_discrete()` to alter the way the labels are displayed.

Modify `chart` so that it incorporates the `theme()` function:

```
chart <- ggplot(top_10, aes(x = Major, y = Unemployment_Rate,
        group = 1)) +
  geom_line() +
  geom_point() +
  labs(title = "Unemployment Rate\nby Major",
    x = "Major",
    y = "Unemployment Rate") +
  theme(axis.text.x = element_text(size = 10, angle = 90, vjust
        = 0.5, hjust = 1))
print(chart)
```

In the `theme()` function, `size` refers to the text size of the x-axis label, `angle` refers to how much the text is rotated, and `vjust` and `hjust` refer to the vertical and horizontal adjustments imposed on the text.

Plot Result:



Click this link to enlarge the image.

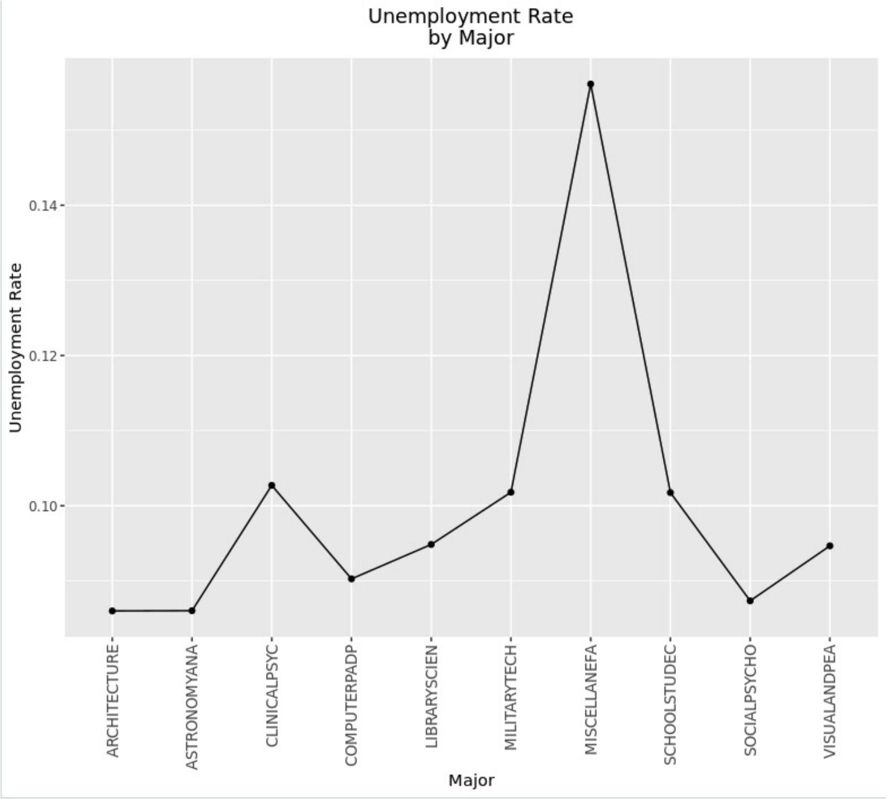
The chart looks much clearer, however, the labels are still too long and compete for space with the data.

To shorten the labels, we can use the `scale_x_discrete()` function like below:

```
chart <- ggplot(top_10, aes(x = Major, y = Unemployment_Rate,
        group = 1)) +
  geom_line() +
  geom_point() +
  labs(title = "Unemployment Rate\nby Major",
    x = "Major",
    y = "Unemployment Rate") +
  theme(axis.text.x = element_text(size = 10, angle = 90, vjust
        = 0.5, hjust = 1)) +
  scale_x_discrete(label = function(x) abbreviate(x, minlength =
        12))
print(chart)
```

The `abbreviate()` function within `scale_x_discrete()` helps to abbreviate the labels and `minlength` determines how many characters to display (12 in this case).

Plot Result:



Click this link to enlarge the image.

The chart now highlights the data better without sacrificing the labels too much.

challenge

Try this variation(s):

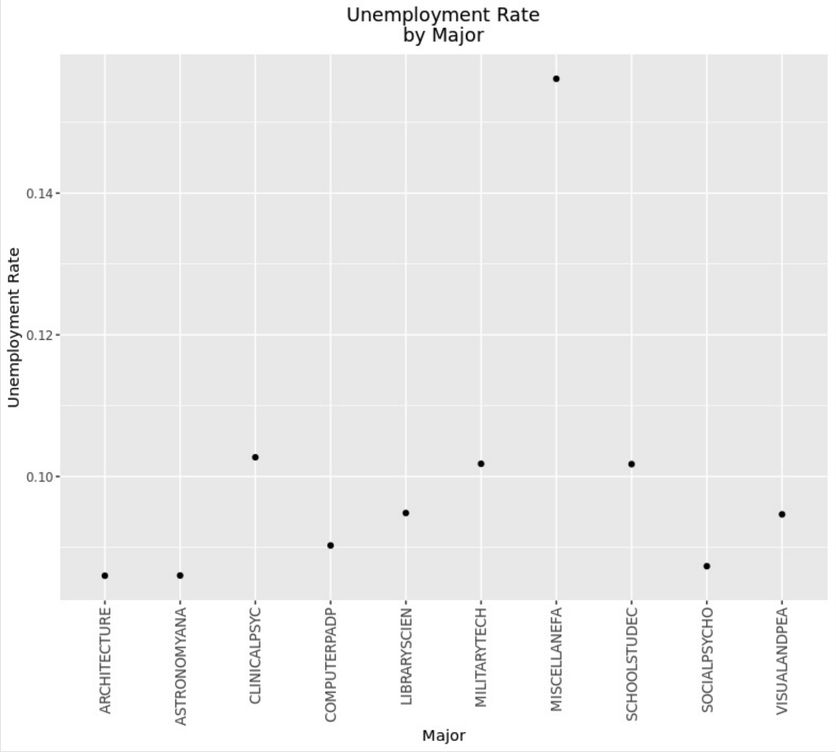
- Remove

```
group = 1
```

from

```r
chart <- ggplot(top_10, aes(x = Major, y =
        Unemployment_Rate, group = 1)) +
  geom_line() +
  geom_point() +
  labs(title = "Unemployment Rate\nby Major",
    x = "Major",
    y = "Unemployment Rate") +
  theme(axis.text.x = element_text(size = 10, angle = 90,
        vjust = 0.5, hjust = 1)) +
  scale_x_discrete(label = function(x) abbreviate(x,
        minlength = 12))
print(chart)
```

▼ Plot Result



Click this link to enlarge the image.