Basic Plots

Table of Contents

Estimated Time Needed: **20 min**

The importance of graphs

Data visualization is the presentation of data with graphics. It's a way to summarize your findings and display it in a form that facilitates interpretation and can help in identifying patterns or trends. Having great data visualizations will make your work more interesting and clear.

Introduction to data visualization

Why use data visualization?

- Represent data in the form of graphs, charts, plots, etc.
- A story conveyed through visuals
- Patterns, trends, and anomalies in data

What are the charts? Let talk about chart types.

Charts can be classified into six categories based on the chart's goals, aesthetics, or visual features.

- **Comparison charts**: designed for comparison and aim to visualize the differences between elements.
- **Trend charts**: represent data along with the time dimension.
- **Part to whole charts**: show the inner subdivision of a value among different categories or groups.
- **Correlation charts**: highlight the possible correlation between two or more indicators.
- **Relationships and connections charts**: represent hierarchies.
- **Maps**: communicate geolocated information.

In this course, you will learn how to use the ggplot2 library to create beautiful graphics and charts, customizing the look and feel of them as you wish.

The difference between R libraries

R has its own built in graphing library, `graphics`, which is often referred to as base R graphics. `ggplot2` on the other hand is a library created to make plotting elegant visuals easier. It is based on the book "The Grammar of Graphics", a book that describes the foundations for data plotting. Thus, "gg" stands for grammer of graphics.

The differences between the base R graphing library and ggplot2 are many. Let's look at the differences between them using a simple example.

Base R graphics

The `graphics` library is the base (default) R library for plotting graphs. It includes functions like `plot`, `hist`, and `boxplot`. The graphs are very simplistic in both syntax and aesthetics.

For example, to create a bar plot, you use the `barplot` function which takes in height (count of each bar to plot).

[1]:
```
count <- table(mtcars$cyl)  # count of each number of cylinder
barplot(count)
```

ggplot2

`ggplot2`, as mentioned above, is a specialized library made to create visually pleasing data visualizations.

There are some main components to create `ggplot2` graphs:

- **Data** - the dataset
- **Aesthetics** - maps data to visual variables like color and position
- **Geometry** - represents the items you see on the plot like points and lines

There are two ways to plot, both contain the three components.

1. `qplot()` offers a simpler syntax similar to the base R functions, but is limited in customization.
   - e.g. `qplot(data, x = feature1, geom = "bar")` where "x" is an example of the aesthetics and "bar" is an example of geometry.
2. `ggplot()` is the full-fledged function. It has far more possible customizations and is more widely used in practice.
   - e.g. `ggplot(data, aes(x = feature1)) + geom_bar()` would produce the same plot as the qplot example

In this course, we will start using `qplot()` and then change to `ggplot()` as you advance.

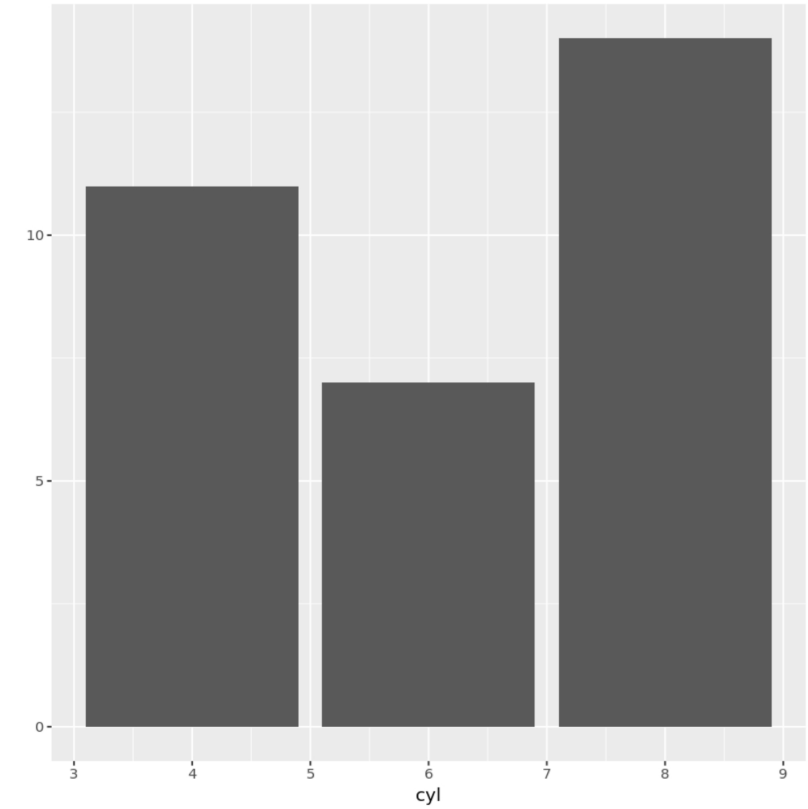
Before we can use `ggplot2`, we need to import it into the R environment. If you are running this in the online lab enviornment, it already has `ggplot2` installed. If running locally, you should first run `install.packages("ggplot2")`. Once the library is installed, use the `library` function to import `ggplot2`.

[2]:
```
library(ggplot2)
```

```
Registered S3 methods overwritten by 'ggplot2':
  method         from
  [.quosures     rlang
  c.quosures     rlang
  print.quosures rlang
```

Now, let's plot our graph. To plot a simple bar graph using `ggplot2`'s `qplot`:

[4]:
```
qplot(data = mtcars, x = cyl, geom = "bar")
```

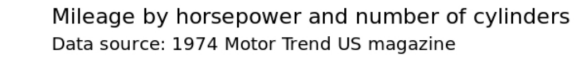


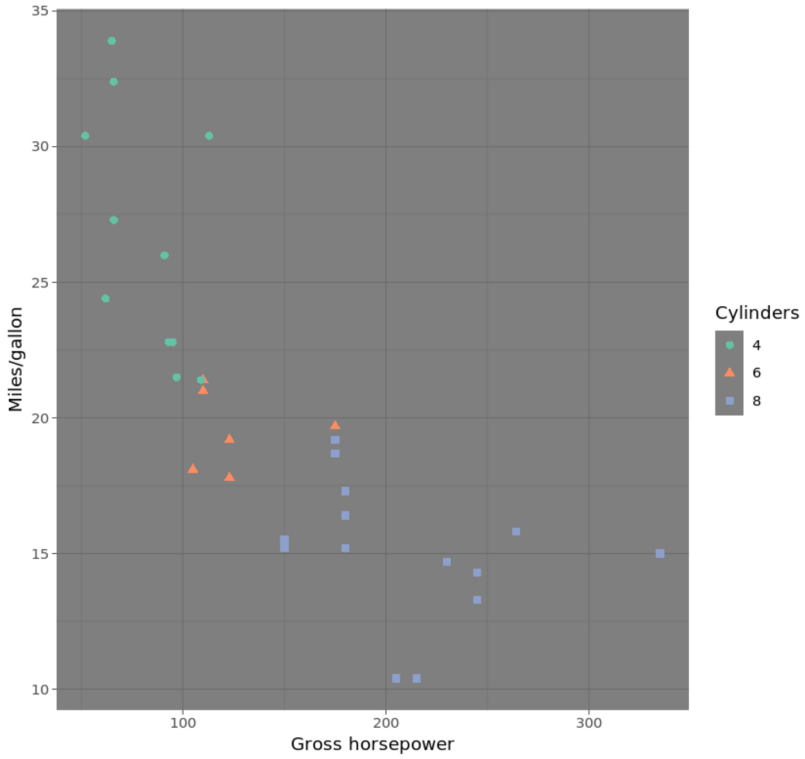
Don't worry - we will go back and learn these and other plotting methods during our lessons.

As you can see, `ggplot2` offers us a nicer-looking graph and more customization, but has a slightly more complex syntax than the base R library. However, there is a lot of documentation on how to use `ggplot2` to help you out. You can reference the `ggplot2` cheat sheet here, it is a handy resource to look up simple examples of the syntax.

The next graph is a demonstration of what is possible to do with `ggplot()`. In the end of this course you will be able to create graphs like this one below and even more complex ones.

[40]:
```
ggplot(mtcars, aes(x = hp, y = mpg, color = factor(cyl), shape = factor(cyl))) +
    geom_point(size=2) +
    labs(x = "Gross horsepower",
         y = "Miles/gallon",
         color = "Cylinders",
         shape = "Cylinders",
         title = "Mileage by horsepower and number of cylinders",
         subtitle = "Data source: 1974 Motor Trend US magazine") +
    theme_dark() +
    scale_color_brewer(palette = "Set2")
```





Qualitative vs Quantitative Data

One thing to always keep in mind is the type of data which you are trying to create graphs for. In general, we categorize data in two big groups: Qualitative data (also called Categorical data) and Quantitative data (also called Numerical data).

- What we refer to as **qualitative or categorical** data is data that refers to, as seen from its name, categories. This can include data for "Yes or No" questions, it could be the name of a location, it could be a person's favorite ice cream flavor, or it could be something else.
- **Quantitative or numerical** data is data that is, quite simply, numbers. It normally is a measurement of some sort, and can be manipulated using simple math. However, keep in mind that it is possible for seemingly numerical data to actually be categorical, as we will see in the "mtcars" dataset.

The plotting methods and the best types of chart for each type of data are different - choosing the best one will help you greatly in creating visually pleasant graphs.

The mtcars dataset

Before we begin exploring `ggplot` more, let's understand the main dataset we will be using to demostrate concepts. `mtcars` is an inbuilt dataset that contains data from the 1974 Motor Trend US magazine, and contains fuel consumption and 10 other aspects of automobile performance for 32 automobiles from 1973–74.

Since it is already included with R, there is no need to import `mtcars`. Let's check its structure - we can do so by calling it by its name, like so:

[5]:
```
mtcars
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2

Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

What kinds of insights can we get from this data? We have the cars' mileage per gallon of gas, the number of cylinders, and other attributes.

If there's any column that you don't know what it represents, you can use the `?` function to see the helpfile for this dataset, like so:

[6]:
```
?mtcars
```

Note that this is only possible because `mtcars` is an inbuilt dataset within R. Almost no datasets will have helpfiles. Some datasets that you find on the internet will have a readme file that will describe what every column is depicting. Some will include the name of each column in the dataset. Although these are good practices, you can find datasets with undescriptive names for its columns and datasets with no names at all.

As you may already know, we don't need to view the entire dataset to look how the data is structured. We can use the `head` function to see the first 6 rows of our data.

[7]:
```
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

And we can use `tail` to see the last 6 rows of our data.

[8]:
```
tail(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.7	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

Let's further analyze our data... We can get a quick summary of each column using the `summary` function:

[9]:
```
summary(mtcars)
```

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

We can get the average for any column using `mean(datasetname$columnname)`, like so:

[10]:
```
mean(mtcars$cyl)
```

6.1875

Making Bar Plots

Let's start our plotting with bar plots. As the name implies, it's a plot format that shows your data using bars. You probably have seen a lot of them already. Bar plots should be used on **categorical variables** to display a *count* of each category. These plots are a good way to do an intial visual explororation of categorical variables.

In our example, we will use `cyl` in `mtcars`. On first inspection, `cyl` may appear to numerical. However, `cyl` is the number of cylinders in cars and this is fixed to be 4, 6, or 8, so there are three categories. So, `cyl` is categorical and we can create a bar plot for it.

Before actually creating any bar plot, let's import our plotting library, `ggplot2`. There's no need to execute this block if you already executed the import in "Differences between R Libraries".

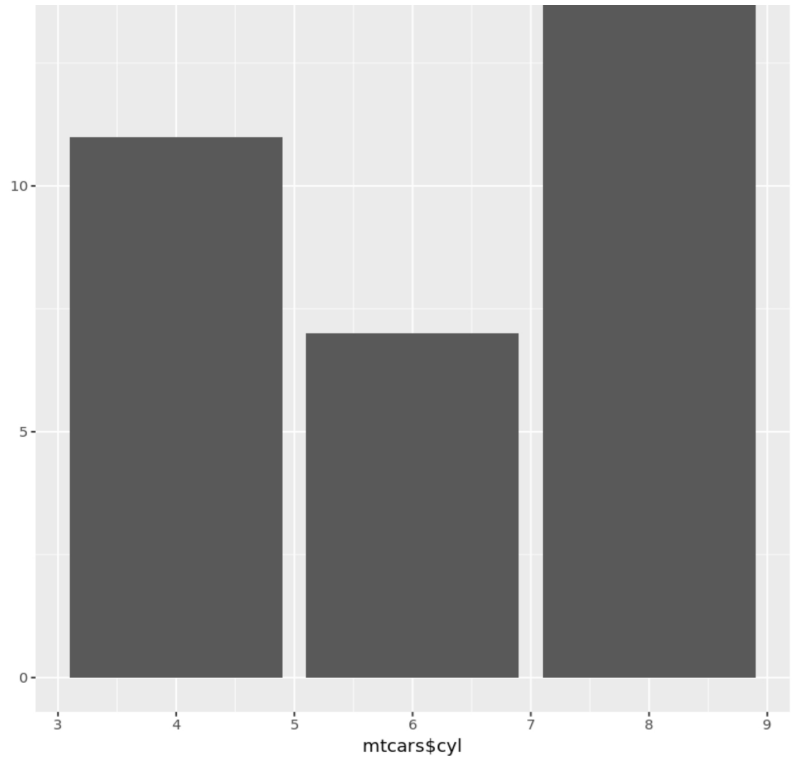
[11]:
```
library(ggplot2)
```

Now that we have loaded our libraries, let's start plotting. To begin, we can use the `qplot` function. As we introduced before, `qplot` follows a format like:

`qplot(data = mtcars, x = cyl, geom = "bar")`

However, you can also choose to not include `data` and instead explicitly grab the column like, `mtcars$cyl`. Since we are creating bar plots, set the geometry to "bar".

[12]:
```
qplot(mtcars$cyl, geom = "bar")
```



As we can see, we plotted a bar plot, consisting of the count of every element with the same value. We can now exploit some of the possibilities of `ggplot2`'s `qplot` function.

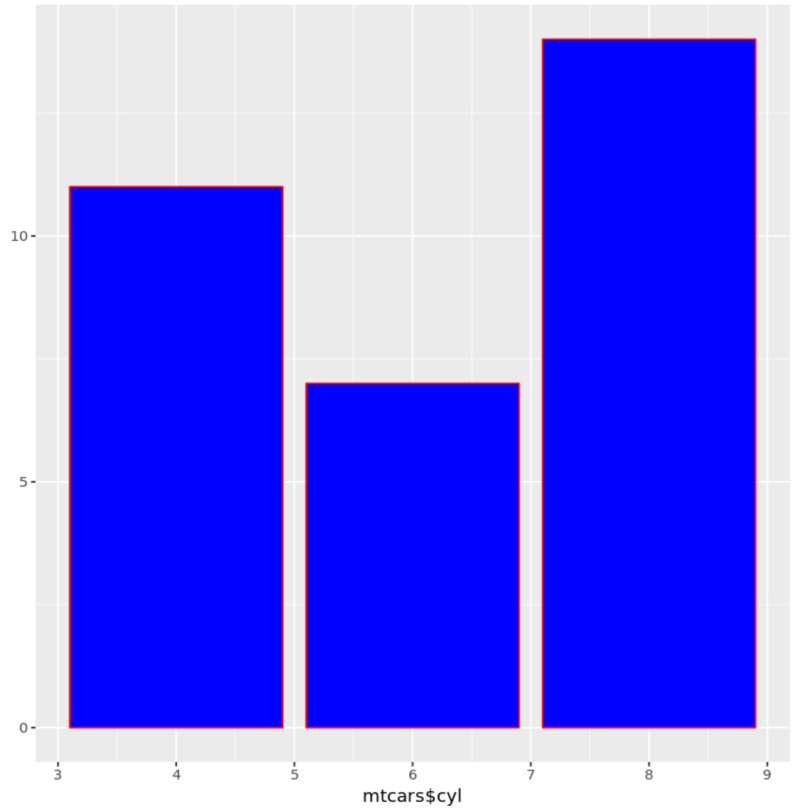
[13]:
```
?qplot
```

You don't need to understand how exactly each parameter of the `qplot` function works, but you can always go to the help file to try to find what you need.

Our graph is plain as it stands right now. A plain graph is an excellent choice for academic papers, but for Internet content simply being gray will not catch people's attention. Let's change the aesthetics by giving it some color using the `colour` (or `color`) and the `fill` parameters. `colour` will modify the color of the outline, while `fill` will change the color of the bars.

You'll notice that with `qplot`, you must include `I()` to denote the colors. The `I()` indicates that it should be treated "as is". Without `I()`, `qplot` would interpret the color names as factors. You'll see later on when using `ggplot`, this will not be needed.

[48]:
```
qplot(mtcars$cyl, geom = "bar", fill = I("blue"), colour = I("red"))
```

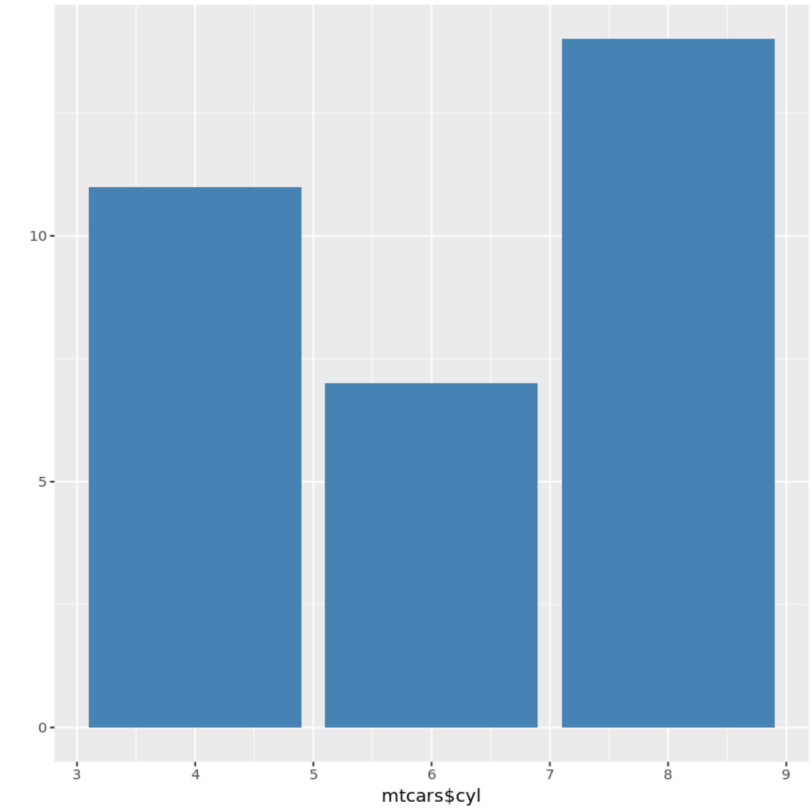


If only the `fill` parameter is changed, the outline will also change to that color.

You can change the colors to blue, pink, green, yellow...there's so many colors! You can find a list here.

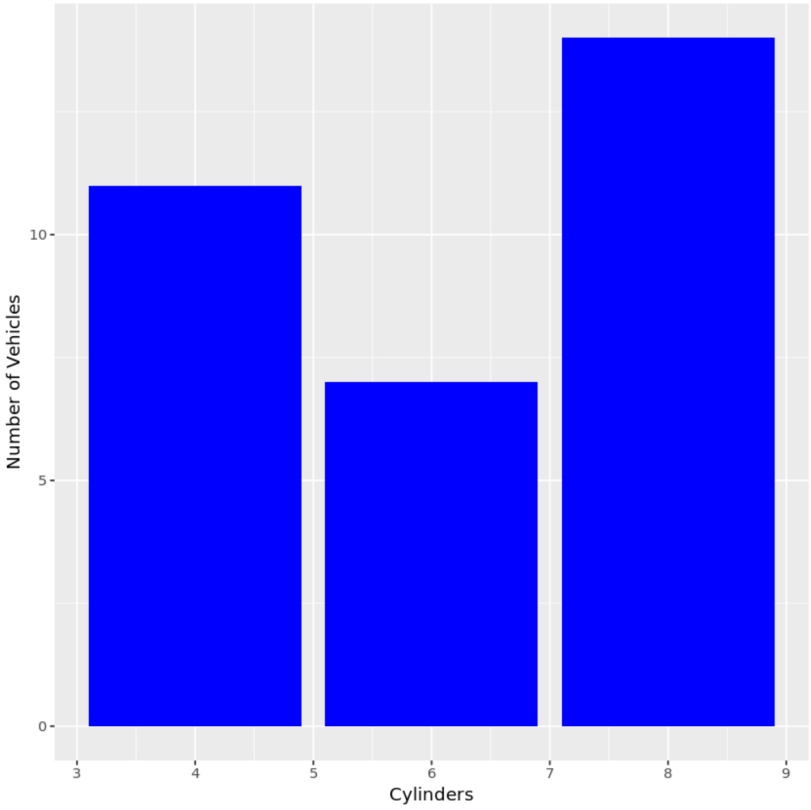
For example, if you would like to change the color of each bar, then customize the `fill` parameter like so:

[50]:
```
qplot(mtcars$cyl, geom = "bar", fill = I("steelblue"))
```



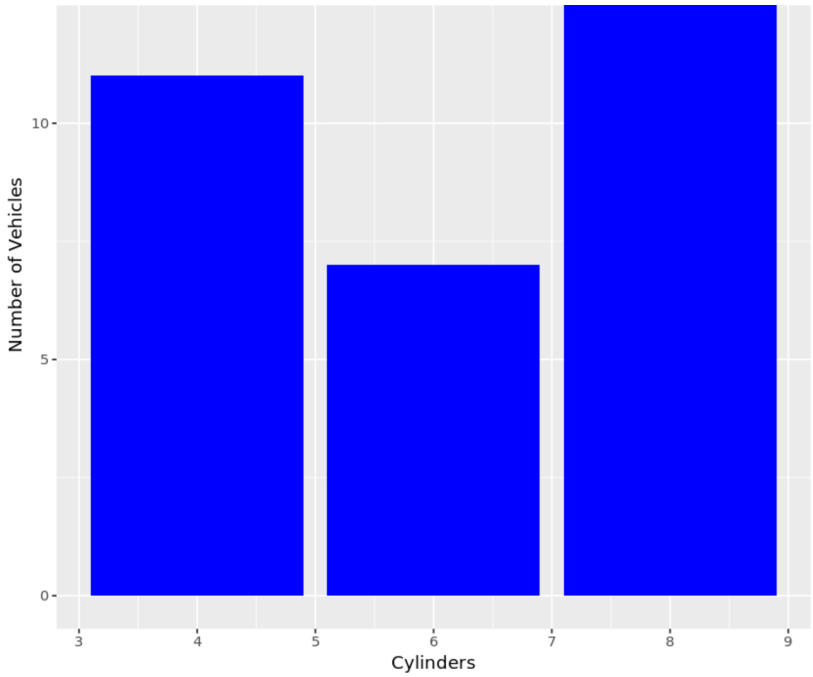
We can also change the name of our axes to make it more easily understandable by passing the `xlab` and `ylab` parameters (`lab` stands for "label"):

[16]:
```
qplot(mtcars$cyl, geom = "bar", fill = I("blue"), xlab = "Cylinders", ylab = "Number of Vehicles")
```



To finish our bar plot, we can give a name for our graph. We can do this using the `main` parameter.

[17]:
```
qplot(mtcars$cyl, geom = "bar", fill = I("blue"), xlab = "Cylinders", ylab = "Number of Vehicles", main = "Cylinders in mtcars")
```



Question #1:

According to the example above, let's make a bar plot by using the "carb" column in the "mtcars" dataset. You can customize the name of the graph and the x and y labels.

[18]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
# There are multiple ways to customize, this is just one example
qplot(mtcars$carb, geom = "bar", fill = I("blue"), xlab = "Carb", ylab = "Number of Vehicles", main = "Carb in mtcars")
```

We've focused on using `qplot()`, however like we said earlier, you can also use `ggplot()`.

With `ggplot()`, instead of specifying the geometry type with a paramter, you **add the geometry** function which normally begins with `geom_`. For example the following are equivalent graphs:

- With qplot: `qplot(data = mtcars, x = cyl, geom = "bar")`
- With ggplot: `ggplot(mtcars, aes(x = cyl)) + geom_bar()`

When we go over pie charts and in later lessons, we will use the `ggplot()` way. So for now, just start getting a grasp of this concept of adding on functions to create components.

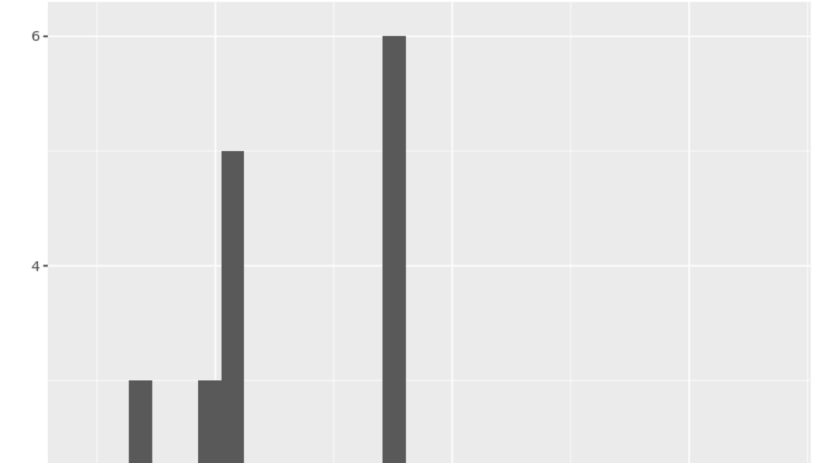
Histograms

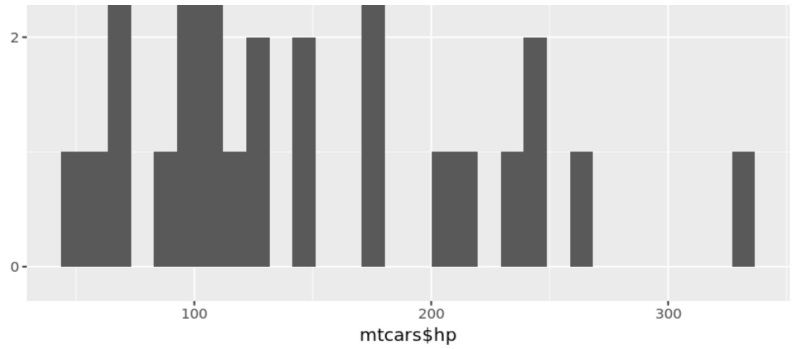
Histograms can be defined as a graphical visualization of data counts. It is similar to a bar plot, however histograms are used for **numerical** data. They are simple graphs commonly used to show the distribution of a numerical variable. Note that bar plots on the other hand are used for categorical data. Usually, there's no space between the columns of a histogram since the data is continuous. These plots are a good way to do an intial visual explororation of numerical variables.

Again, let's begin by using `qplot`. To create a histogram, you just need to pass in the data and the geometry "histogram". For example, let's look at the numerical variable, horsepower, from `mtcars`.

[19]:
```
qplot(mtcars$hp, geom = "histogram")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

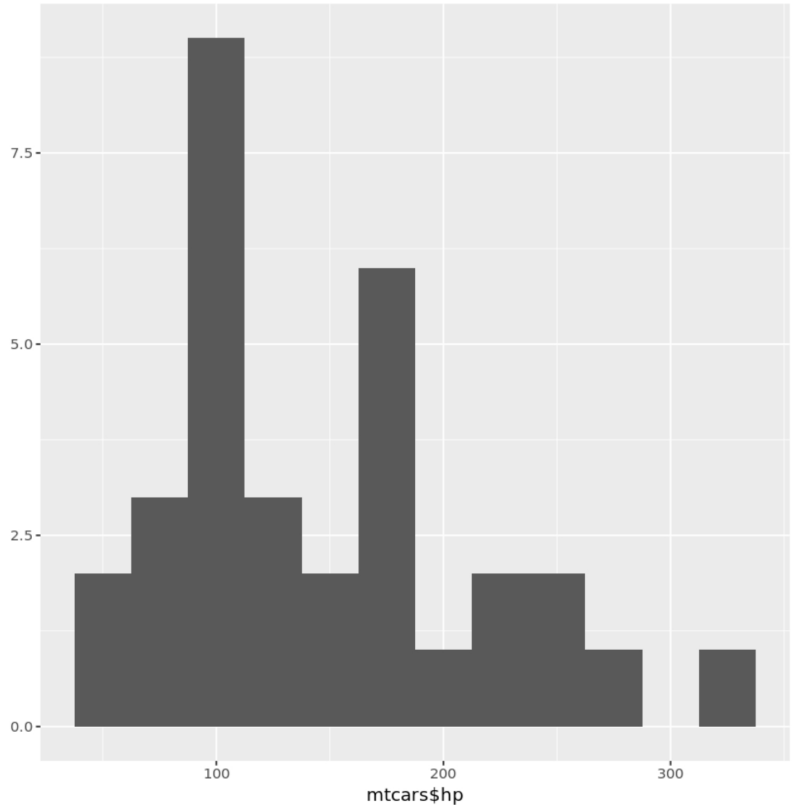




As you executed the code, you may receive the following error message: `stat_bin() using bins = 30. We can pick a better value with binwidth`.

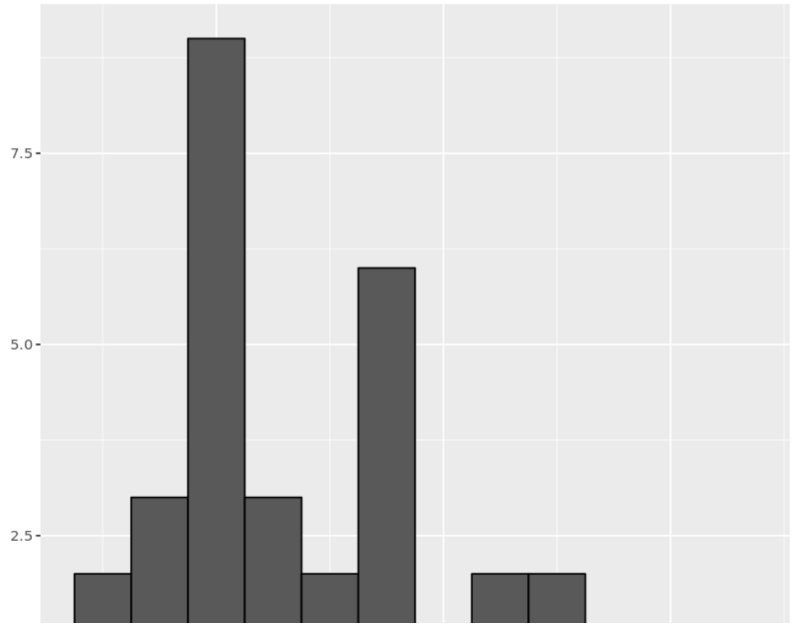
You could also change the number of bins with `bins`. However here, we will change the `binwidth`, which is the width of each bar.

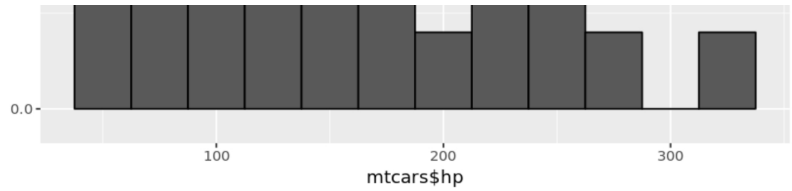
[20]:
```
qplot(mtcars$hp, geom="histogram", binwidth = 25)
```



Now, let's add a black outline to the histogram to make it better to visualize.

[21]:
```
qplot(mtcars$hp, geom="histogram", binwidth = 25, colour = I("black"))
```



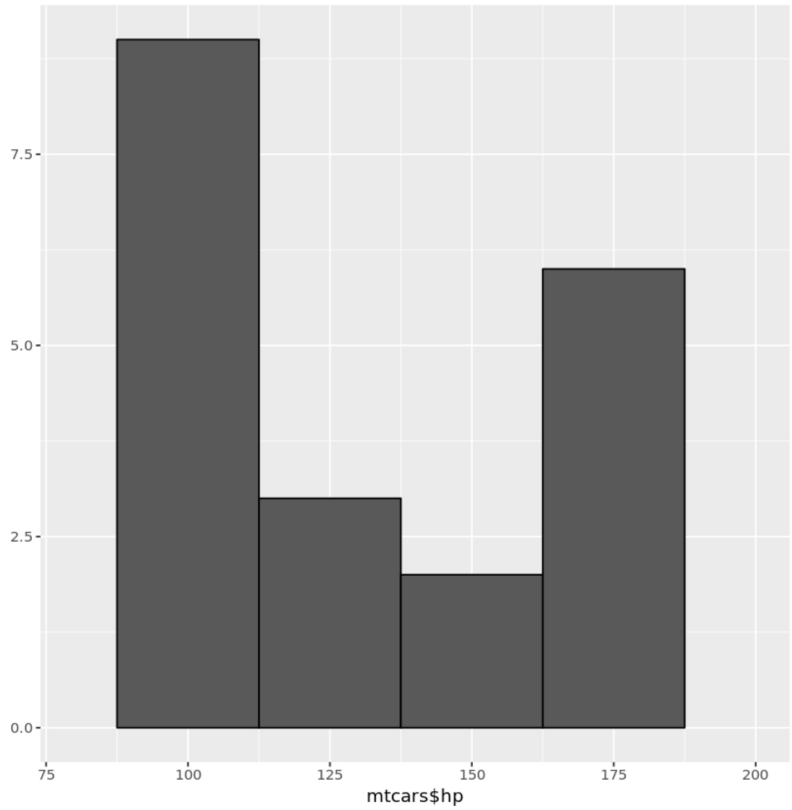


We can also define limits for our x axis. R will fit automatically the best values according to our data, so the result will not change when you use qplot and an adequate limit.

If it happens to "cut" any value, R will display an error message saying how many values are not being shown.

[23]:
```
qplot(mtcars$hp, geom="histogram", binwidth = 25, colour = I("black"), xlim = c(80,200))
```

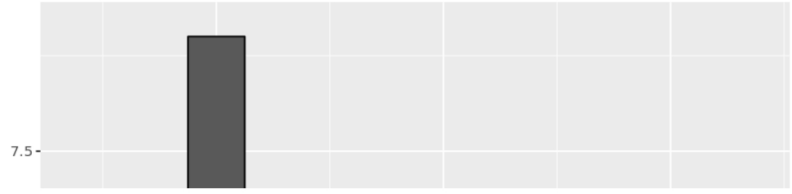
```
Warning message:
“Removed 12 rows containing non-finite values (stat_bin).”Warning message:
“Removed 2 rows containing missing values (geom_bar).”
```



In the above example, the `xlim` range is too narrow and leaves out some data. So let's just use the default `xlim`.

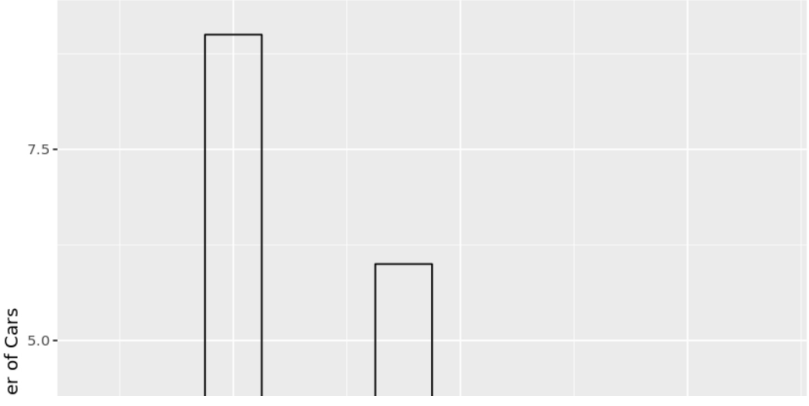
Now let's give a name for our axes. We use the `xlab` and `ylab` parameters to modify them.

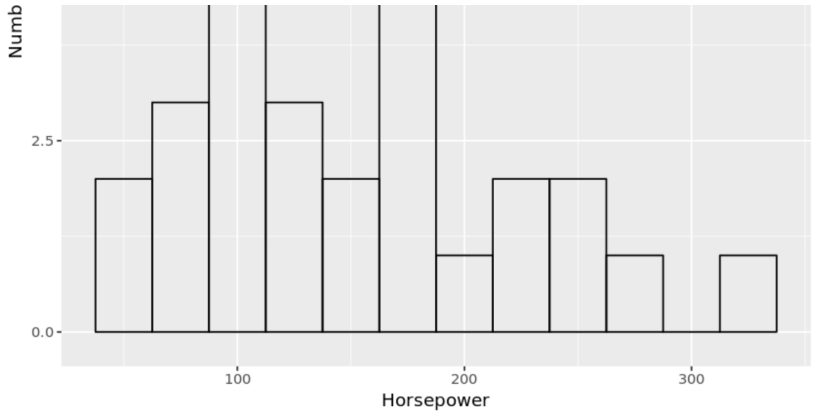
[24]:
```
qplot(mtcars$hp, geom="histogram", binwidth = 25, colour = I("black"), xlab = "Horsepower", ylab= "Number of Cars")
```



We can remove the color that fills the bars using the parameter `alpha`. Again, we use `I()` to make sure `qplot()` understands that we want the number 0.

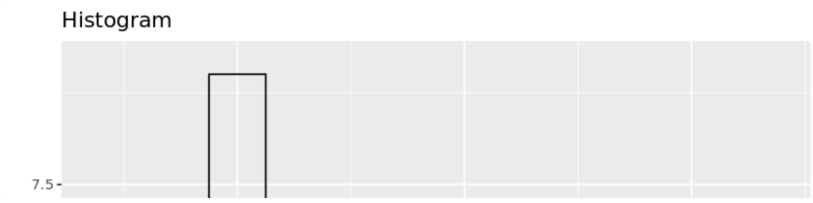
[25]:
```
qplot(mtcars$hp, geom="histogram", binwidth = 25, colour = I("black"), xlab = "Horsepower", ylab= "Number of Cars", alpha = I(0))
```





As we did with the bar plot, we can give a name for our graph. We can do this using the `main` parameter.

[26]:
```
qplot(mtcars$hp, geom="histogram", binwidth = 25, colour = I("black"), xlab = "Horsepower", ylab= "Number of Cars", alpha = I(0),
      main = "Histogram")
```



Question #2:

According to the example above, let's make a histogram by using the "qsec" column in the "mtcars" dataset and set the "binwidth" as 1. You can customize the name of the graph and the x and y labels.

[27]:
```
# Write your code below and press Shift+Enter to execute
```

▼ Click here for the solution.

```
qplot(mtcars$qsec, geom="histogram", binwidth = 1, xlab = "1/4 mile time", ylab= "Number of Cars", main = "Histogram")
```

Expert Tip: Sharpening your qplot skills

Do you want to check all the qplot function parameters? Click here to access the function documentation !

We've focused on using `qplot()` to create histograms so far. Let's try using `ggplot()` now. The geometry function to use for histograms is `geom_histogram()`.

For example, the following are equivalent graphs:

- With qplot: `qplot(data = mtcars, x = hp, geom = "histogram", fill = I("black"))`
- With ggplot: `ggplot(mtcars, aes(x = hp)) + geom_histogram(fill = "black")`

Notice that you don't need to add `I()` when using `ggplot()` and that the `fill` parameter goes into the geometry function since you are coloring that component. Again, this is just to get you more familiar with the `ggplot()` way of plotting, in later lessons and labs we will go into more details.

Pie charts

A pie chart is a circular graph, most commonly used in business. It shows the proportion that each part of data contributes to an overall total. Essentially, it is a bar plot but in circular form. Thus, like bar plots, pie charts represent the counts of **categorical** data.

Expert Tip: Choosing the right chart type

Unsure which type of chart to use for your data? Click here to learn more !

To create pie charts in `ggplot2`, we will will need to use the `ggplot()` function (not `qplot()`). As we introduced earlier, using the `ggplot()` method to create charts, you add on different functions instead of just passing in parameters.

Since pie charts and bar plots are the same except that pie charts are circles, in ggplot2, you first need to create a bar chart. So,

1. First, you will understand what a grouped bar chart is
2. Then, we will make the bar chart stacked
3. Finally, you can transform the stacked bar chart to polar coordinates (make it a circle)

Grouped bar chart

This bar chart uses the mtcars dataset and plots the categorical data in the column cyl, which represents the number of cylinders in the car. We will explain the following code in more detail:

1. Load the "tidyverse" library. If running locally, you may have to install the "tidyverse" library first
2. To create the grouped bar chart, you first convert the column `cyl` to a factor. Using `as.factor()` on a column makes it easier to work with categorical data because it converts the column to levels. In this case, cars typically have 4, 6, or 8 cylinders, so the values have three levels (or categories). Notice that this example uses the tidyverse way to create a new variable using the `mutate()` function. You can also do this using the base R language, which will be seen in the next lab.
3. The next step is to plot the chart using the `ggplot()` function.

- In the aesthetics function or `aes()`, you must set the `x` and `fill` arguments to the categorical variable, which is cyl_factor in this example. The `fill` will fill each category of `cyl` with a different color and automatically creates a legend.
- Next, recall that `geom_bar()` creates a bar chart (instead of using `geom = "bar"` in `qplot()`). To create a *grouped* bar chart, set the position argument to "dodge". This creates a different bar for each factor level (or subgroup) of `cyl_factor`.

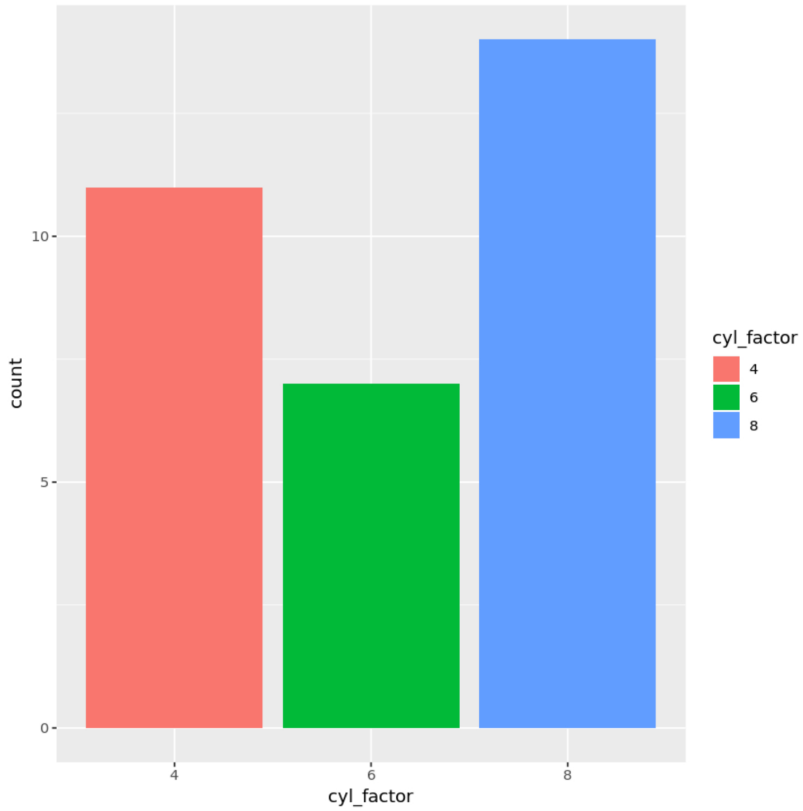
Something you may be wondering is that, isn't the `fill` parameter a valid paramter in the `geom_bar()` as well, why put it in `aes()`? We put it in the `aes()` because the aethetics function is *mapping* an aesthetic to a variable so that each level of `cyl_factor` will be a different fill. On the other hand, putting it in `geom_bar` will make *all* the bars be the same fill.

[5]:
```
# Load Tidyverse library
library(tidyverse)

# Convert cyl to factor
mtcars <- mtcars %>%
  mutate(cyl_factor = as.factor(cyl))

# Plot grouped bar chart
ggplot(data = mtcars,
       aes(x = cyl_factor,
           fill = cyl_factor)) +
  geom_bar(position = "dodge")
```

```
Registered S3 method overwritten by 'rvest':
  method            from
  read_xml.response xml2
── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 ──
✓ tibble  2.1.1       ✓ purrr   0.3.2
✓ tidyr   0.8.3       ✓ dplyr   0.8.0.1
✓ readr   1.3.1       ✓ stringr 1.4.0
✓ tibble  2.1.1       ✓ forcats 0.4.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✕ dplyr::filter() masks stats::filter()
✕ dplyr::lag()    masks stats::lag()
```



Stacked bar chart

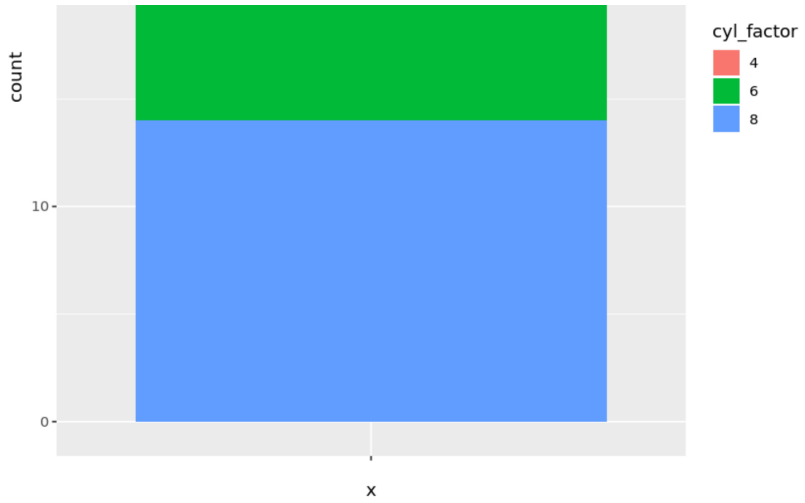
A pie chart in ggplot2 is a transformed stacked bar plot. A stacked bar plot is a plot that stacks all the values on the vertical axis, instead of creating separate bars for each different data point.

To create a stacked bar chart, there are two things you must change.

1. Create an empty string on the x-axis.
2. Change the position argument from “dodge” to “stack”.

Let's create a stacked bar plot.

[29]:
```
ggplot(data = mtcars,
       aes(x = " ",
           fill = cyl_factor)) +
  geom_bar(position = "stack")
```

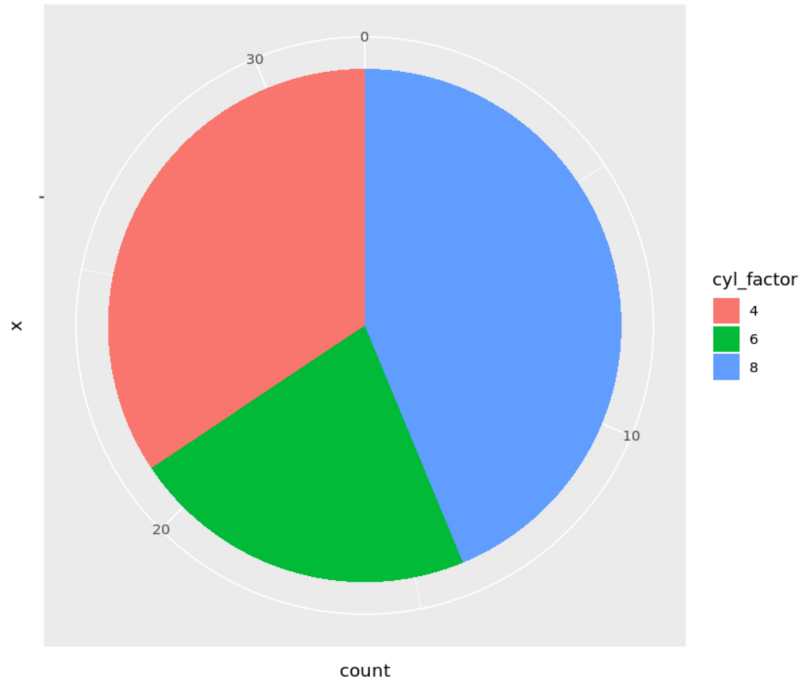


Pie chart

To create a pie chart using ggplot(), you transform the stacked bar chart into polar coordinates, which essentially means converting it to a circle. You can use `coord_polar()` to do this.

This example sets the theta argument of the `coord_polar()` method to "y". The theta is the angle, which determines how big each of the sections of the pie chart should be, and y is the count of each category of the cylinders variable.

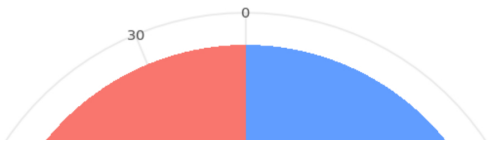
[30]:
```
ggplot(data = mtcars,
       aes(x = " ", fill = cyl_factor)) +
  geom_bar(position = "stack") +
  coord_polar(theta = "y")
```

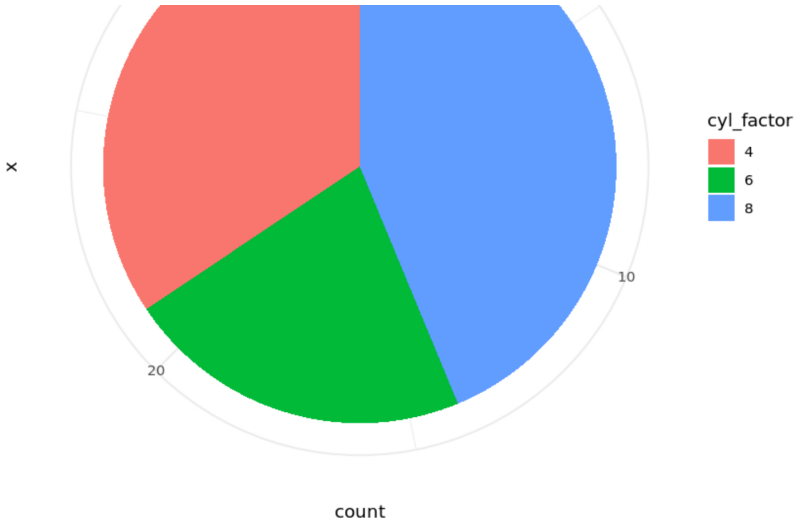


You can also customize how the chart looks by using the different themes built into ggplot(). Use the `theme_minimal()` helper function.

You can search the help documentation with `?theme_minimal` for more built-in themes. Additionally, in a later lesson we will go over more about themes.

[31]:
```
ggplot(data = mtcars,
       aes(x = " ", fill = cyl_factor)) +
  geom_bar(position = "stack") +
  coord_polar(theta = "y") +
  theme_minimal()
```

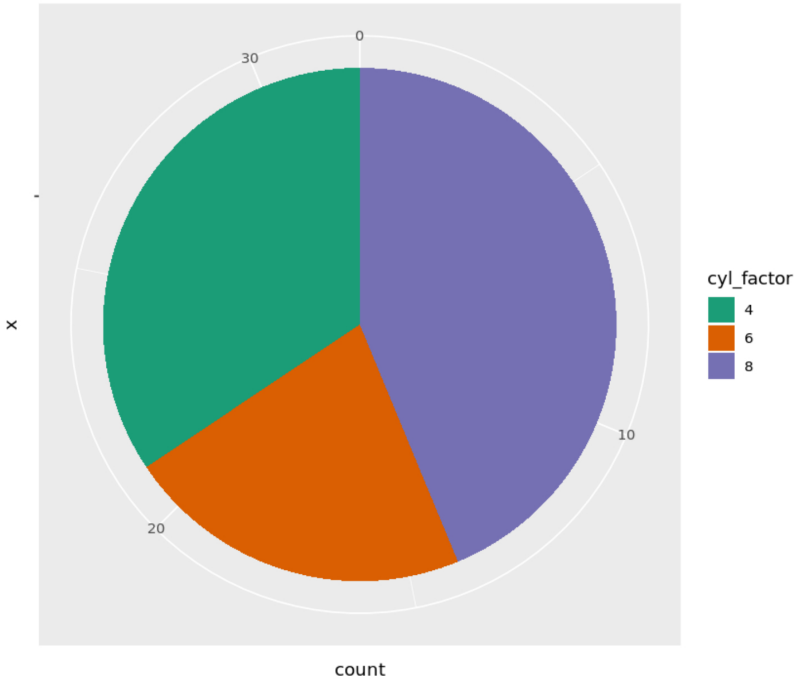




Additionally, you can change the color palette using the `scale_fill_brewer()` function. The palette of the "fill" aesthetics, which represents the color of the pie sections, is being changed. There is a similar function `scale_color_brewer()` to change the "color" aesthetic as well.

This example uses the palette "Dark2" from ColorBrewer, an online tool designed to help you select good color schemes for maps and other graphics. You can find more palette options here.

[32]:
```
ggplot(data = mtcars,
       aes(x = " ", fill = cyl_factor)) +
  geom_bar(position = "stack") +
  coord_polar(theta = "y") +
  scale_fill_brewer(palette  = "Dark2")
```



Question #3:

According to the example above, let's make a pie chart using the "carb" column in "mtcars" dataset.

First convert the "carb" column to factor and name it as "carb_factor", remember to use "as.factor()" function. And then, plot the pie chart using ggplot.

Feel free to customize the color of pie chart.

[33]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
# Convert carb to factor
mtcars <- mtcars %>%
  mutate(carb_factor = as.factor(carb))
```

```r
# Create pie chart
ggplot(data = mtcars,
       aes(x = " ", fill = carb_factor)) +
  geom_bar(position = "stack") +
  coord_polar(theta = "y")
```

About the Author:

Hi! It's Yiwen Li and Tiffany Zhu, the authors of this notebook. We hope you found R easy to learn! There's lots more to learn about R but you're well on your way. Feel free to connect with us if you have any questions.

Other contributors:

Francisco Magioli and Erich Natsubori Sato