REVOLUTION ANALYTICS

Data visualisation
Module 1: Preliminaries

Overview

In this session, we'll cover base plotting and graphics in R.

Objectives:

- Introduce the `plot()` function and graphical parameters.
- Demonstrate multi-plotting abilities.
- Introduce basic data visualization functions.

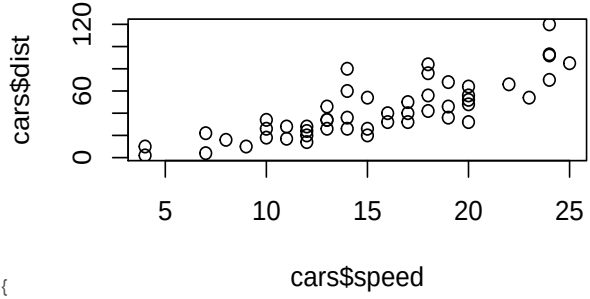
Outline

Introduction to `plot()`

We've been working a lot with the `cars` data set. Let's visualize it.

(Note: distance (`dist`) is the dependent variable, `speed` is the independent variable.)

```
plot(x = cars$speed, y = cars$dist)
```




```
{
}
```

Introduction to `plot()`

The function `plot()` is a generic function that has a variety of methods for different types of R objects.

We'll explore its more basic applications here.

See `help(plot)` for further details

Additional Arguments

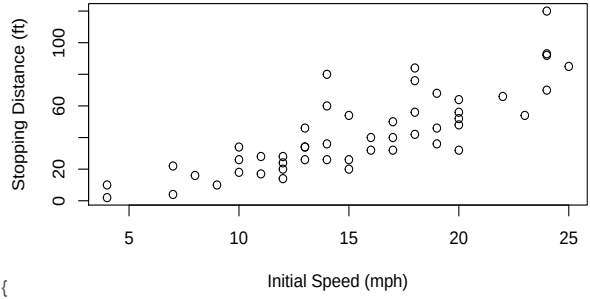
```
args(plot.default)
```

```
## function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
##     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
##     ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
##     panel.last = NULL, asp = NA, ...)
## NULL
```

Adding labels

We can add labels to the chart with the axis arguments:

```
plot(x = cars$speed, y = cars$dist, xlab = "Initial Speed (mph)",
  ylab = "Stopping Distance (ft)")
```



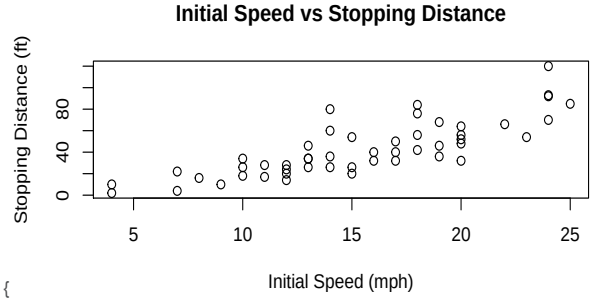
{

}

Adding a title

We can add a title using the *main* argument in plot() or a separate *title* argument

```
title("Initial Speed vs Stopping Distance")
```




```
{
}
```

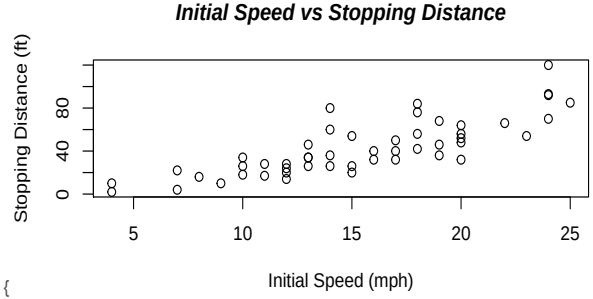
Changing your fonts

- You can change the font of your title by using the *font.main* argument.
- There are 5 different types of basic fonts for the title:
  - 1 = plain
  - 2 = bold
  - 3 = italic
  - 4 = bold and italic
  - 5 = symbol

Changing your fonts

In this plot we decided to bold and italicize our title

```
title("Initial Speed vs Stopping Distance", font.main = 4)
```

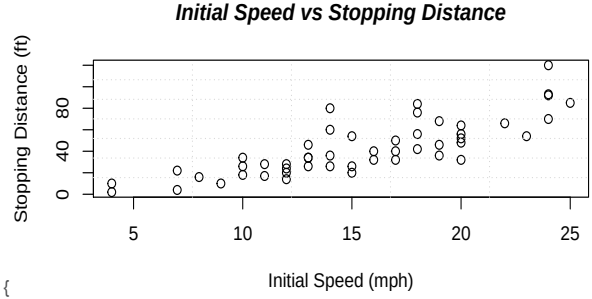



```
{
}
```

Adding Grids

You can further customize your plots by adding grids, using the *grid* command. In this plot, we added a basic grid to our scatter plot.

```
title("Initial Speed vs Stopping Distance", font.main = 4)
grid(5, 7, lwd = 1)
```



{

}

Creating a scatter plot

Recall the acceleration column that we added to cars.

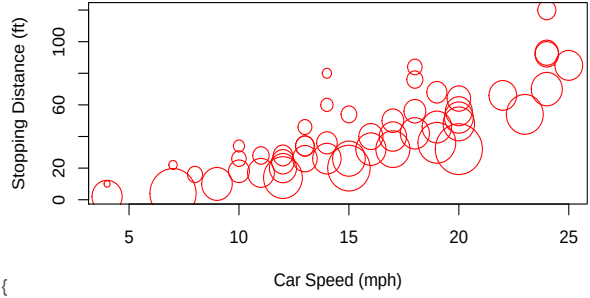
```
cars <- transform(cars, acceleration = -(speed^2)/(2 * dist))
head(cars)
```

```
##   speed dist acceleration
## 1     4    2       -4.000
## 2     4   10       -0.800
## 3     7    4       -6.125
## 4     7   22       -1.114
## 5     8   16       -2.000
## 6     9   10       -4.050
```

Adding additional information

We can use the new acceleration column to add extra information to the plot with `cex` (Note that are also using "col" to color circles red)

```
plot(cars$speed, cars$dist, xlab = "Car Speed (mph)", ylab = "Stopping Distance (ft)",
  col = "red", cex = abs(cars$acceleration))
```



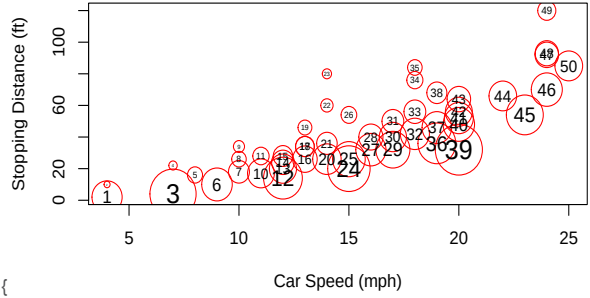
{

}

Adding labels to a scatter plot

We can add car numbers to the plot, as well, using `text()` (note scaling with `cex`):

```
par(mar = c(5, 4, 0.5, 1) + 0.1)
plot(cars$speed, cars$dist, xlab = "Car Speed (mph)", ylab = "Stopping Distance (ft)",
  col = "red", cex = abs(cars$acceleration))
text(x = cars$speed, y = cars$dist, labels = row.names(cars), cex = 1.5 *
  abs(cars$acceleration)/max(abs(cars$acceleration)))
```



{

}

Additional arguments

Many options to `plot()` functions

Some important ones:

`type` (point, line, etc)

`xlim`, `ylim` Limits on `x` and `y` axes, respectively

`log` log-scaled axes

`...` additional arguments from `par`

`pch` character used to plot (described in `par()`)

par()

Use par() to set many additional graphics parameters

Can control graph layout, default plotting options, etc.

Notable parameters:

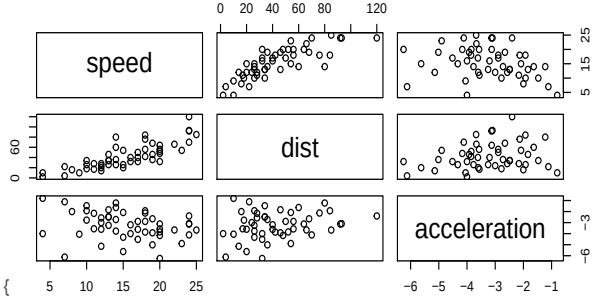
`mfrow`	Plot layout
`cex.axis`	size of axis information
`cex.lab`	size of axis labels

Using pairwise plots to inspect your data

You an use `plot()` on dataframes to generate pairwise displays.

It considers each pair of columns and plots the first column versus the second column.

```
plot(cars)
```



Exercise: Get familiar with plot

Your turn:

- Use `plot()` to explore some of the patterns in `reorg_stats`. Explore relationships between pairs of variables.
  - Create a pairwise plot of any two variables (Hint: You may find `jitter()` helpful. Jitter adds a small amount of noise to a numeric vector thus improving visualization of continuous numeric data with a limited number of unique values.)
  - Change the character used to a filled circle (pch=19)

Outline

Saving Plots

You may need to save a plot for future use such as a presentation.

There are many ways to do this, and we will talk about two.

1. Manually via the IDE
2. In code by creating an appropriate `device` to render output to.

Manually Saving Plots

In most IDEs, there is a mechanism by which you can export a current plot to an image or pdf.

In `RStudio`, you can see this as an `Export` button in the `Plots` tab. Simple walk through the set of menus to select the type of output you would like.

Graphics devices

Graphics work in R by creating `devices` that contain the rendered output. You can get a lot of help with these by viewing the help on the `grDevices` package.

```
library(help = grDevices)
```

Active Device

There are a number of `dev.xxx()` functions that help you manage devices

```
dev.cur()  ## tell which device is current
```

```
## pdf
##   2
```

```
dev.list()  ## list all active devices
```

```
## pdf
##   2
```

Copy to a pdf

The IDE typicaly operates by copying the currently active device to some other type of device.

In addition to doing it manually, we can specify the code to do that as well.

```
outputPath <- "../output"
if (!file.exists(outputPath)) dir.create(outputPath)
dev.copy2pdf(file = file.path(outputPath, "copiedscatter.pdf"))
```

```
## pdf
##   2
```

More Copying

There are additional approaches to this.

```
help(dev.copy)
```

Generating Graphics First

You might want to automate graphics generation. In this case, it's not a good idea to generate an on-screen device, and then copy. You should just render directly to the output of your choice.

```
help(Devices)
```

Example: pdf

```
pdf(file.path(outputPath,"Example.pdf"))
par(mar=c(5,4,0.5,1)+0.1)
plot(cars$speed, cars$dist,
     xlab = "Car Speed (mph)",
     ylab = "Stopping Distance (ft)",
     col = "red", cex = abs(cars$acceleration)
     )
text(x = cars$speed,
     y = cars$dist,
     labels = row.names(cars),
     cex = 1.5*abs(cars$acceleration) / max(abs(cars$acceleration))
     )
dev.off() ## have to turn the device off.
```

```
## pdf
##   2
```

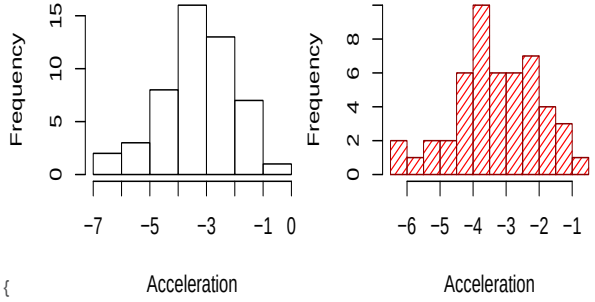
Questions?

Outline

Introduction to histogram

The function `hist()` plots histograms of the data. It is easier to alter the graphical specs on these plots as well.

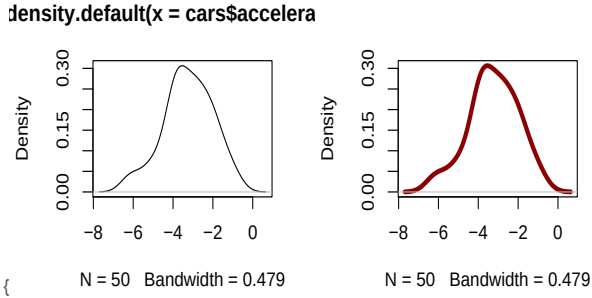
```
hist(cars$acceleration, xlab = "Acceleration", main = NA)
hist(cars$acceleration, xlab = "Acceleration", density = 20, border = "darkred",
  col = "red", breaks = 10, main = NA)
```



{ }

Density plots

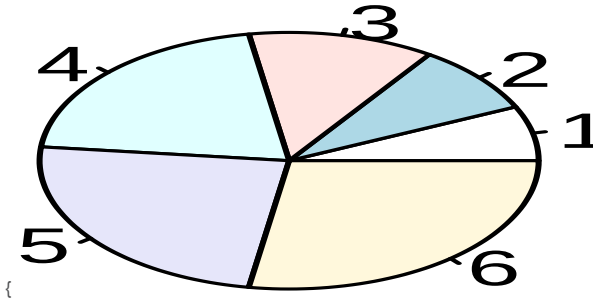
```
plot(density(cars$acceleration))
plot(density(cars$acceleration), lwd = 4, col = "darkred", main = "")
```



{

}

Pie Charts



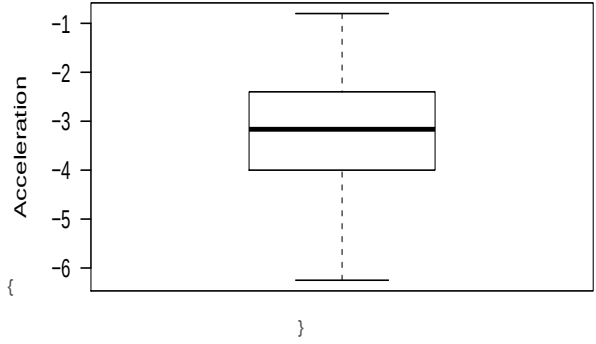
{

}

Box-and-Whisker plots

Box and Whisker plot

```
boxplot(cars$acceleration, las = 1, ylab = "Acceleration")
```



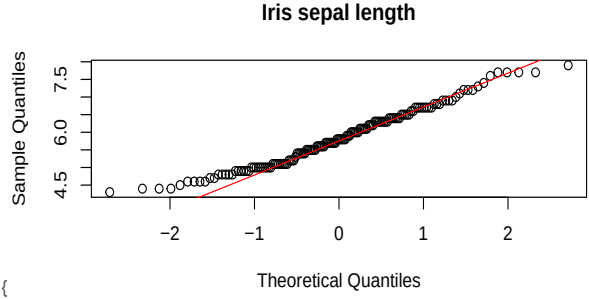
Adding additional components to graphs

`lines()`	add a line to a graph
`abline()`	add an ax + b formatted line to a graph by specifying `a` and `b`
`arrows`	add a set of lines with `ticks` at one or both ends (useful for error bars)
`polygon`	add arbitrary polygons to a plot (useful for `ribbons`)
`text`	add text to a particular `x`, `y` location

Testing for normal distribution

We can use QQ plots (background reading) to check the distribution of our data.

```
qqnorm(iris$Sepal.Length, main = "Iris sepal length")
qqline(iris$Sepal.Length, col = "red")
```



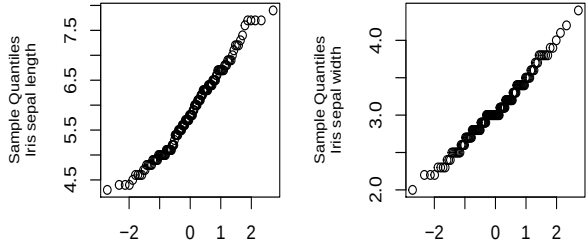
{

}

Multi-plotting using `par()`

The function `par()` sets the parameters of the graphical device. The argument `mfrow` controls the number of rows and columns, allowing you to put multiple plots on a page.

```
par(mfrow = c(1, 2), mar = c(5, 6, 1.5, 0) + 0.1)
qqnorm(iris$Sepal.Length, ylab = "Sample Quantiles\nIris sepal length",
  cex.lab = 0.8, main = NA)
qqnorm(iris$Sepal.Width, ylab = "Sample Quantiles\nIris sepal width",
  cex.lab = 0.8, main = NA)
```

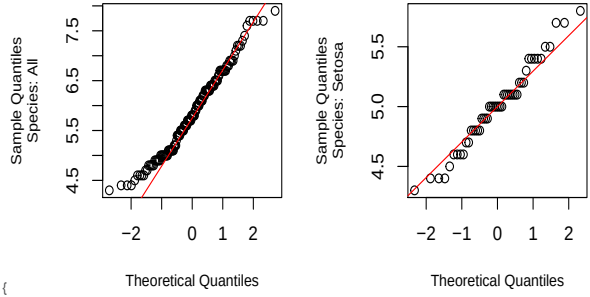


}

Side note on qqplot: Exploring samples

The `iris` dataset contains information on three different types of species. Compare all flowers with only one species:

```
par(mfrow = c(1, 2), mar = c(5, 6, 1.5, 0) + 0.1)
with(iris, qqnorm(Sepal.Length, ylab = "Sample Quantiles\nSpecies: All",
  main = NA, cex.lab = 0.8))
with(iris, qqline(Sepal.Length, col = "red"))
with(iris[1:50, ], qqnorm(Sepal.Length, ylab = "Sample Quantiles\nSpecies: Setosa",
  main = NA, cex.lab = 0.8))
with(iris[1:50, ], qqline(Sepal.Length, col = "red"))
```



Lab: Graph performance data

Your turn:

- Load the `performance.refmt.csv` file into your workspace (it should be there already). Explore the data using `str()` and reformat as necessary for plotting.
- Use `qqnorm()` and `density()` to examine the normality of `ROI`. What are your conclusions?
- What other variables in the data set would you like compare to `ROI`?
- Make some plots exploring these relationships.

Additional Plotting tools

Two additional popular tools

- `library(lattice)`
- `library(ggplot2)`

Module review questions

- What is the fundamental function to create plots in R?
- Name three different function to create special type of plot.
- How does the `plot()` function know what type of plot to make, given different classes input data?
- What is the function that controls what the plot device looks like?
- How can a user set an R graphics window to have multiple panes?

REVOLUTION ANALYTICS

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR