

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Introduction to R Workshop

Presented by Merry Chu

UNSW BINFSOC x UNSW MathSoc

Slides by John Kameas, Raymond Li and Gorden Zhuang

6th July, 2021

Table of Contents

The Basics
Functions

Probability
and Statistics

Data Frames

Visualisation

1 The Basics

2 Functions

3 Probability and Statistics

4 Data Frames

5 Visualisation

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

The Basics

The Bare Essentials



The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Before we get into the interesting stuff in , we need to make sure we've covered the bare essentials. That is, making sure how to do some basic arithmetic, making use of the most common functions and actually familiarising ourselves with the  interface!

The Interface

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- **Console:** for quickly testing out code. Can use UP arrow to run previous commands.
- **Script:** create a new script by clicking on the plus sign in the top left corner.
- **Environment:** lists the variables in the workspace and their values.

Example: Arithmetic

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Try typing a couple of the following commands in the console.

■ `1 + 2`

■ `5.5 - 1`

■ `3 * 2`

■ `10 / 2`

■ `2 ^ 3`

■ `exp(1)`

■ `pi`

■ `exp(1000)`

■ `Inf * 0`

Comparison and Logical Operators

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Comparison operators compare a pair of values and gives either **TRUE** or **FALSE** . For example

- `5 > 3` is **TRUE**
- `-2 <= 5` is **TRUE**
- `0 == 1` is **FALSE** (`==` is *comparison* not *assignment*)
- `1+1 != 3` is **TRUE**

Logical operators can be used to combine the **TRUE** and **FALSE** .

- `(1 > 3) & (4 > 2)` is **FALSE**
- `(1 > 3) | (4 > 2)` is **TRUE**

Variables

The Basics
Functions

Probability
and Statistics

Data Frames

Visualisation

Often you would want the value to be saved so that it can be used later.

- A value can be assigned to a variable with the arrow `<-` . For example `x <- 5` assigns the value 5 to the variable `x` . The value can be printed by running `x` .
- Variable names need to start with a letter and can contain letters, numbers and underscores. However, they are case sensitive.
- If a value is assigned to a variable name that already exists, the old value is replaced with the new value.

Task: Working Out the Value of a Variable

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Guess what you get when you run the following code.

```
a <- 5
```

```
x <- 3
```

```
x <- a + x
```

```
a <- 10
```

```
x <- x + 1
```

```
x
```

Solution: Working Out the Value of a Variable

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

```
a <- 5           # a is 5
x <- 3           # x is 3
x <- a + x       # x is now 5 + 3 = 8
a <- 10          # this does not affect x
x <- x + 1       # x is now 9
x
```

Data Types

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

The following are some data types.

- **numeric** : numbers like `1.2`, `pi`, `1` .
- **character** : contain letters, numbers and or other characters separated by `"` . Examples are `"a"`, `"1"`, `"1.5"`, `"hello"`, `"a_b c1 2 ^ 1"` .
- **logical** : `TRUE` and `FALSE` .

Example: Data Types

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

You can work out the type of a variable by using the function `class(var)` . Try running the following:

- `class("this is a character")`
- `class(1)`

You can convert a variable to another data type with `as.type(var)` . Try running the following.

- `x <- 1`
 `x <- as.character(x)`
 `class(x)`
- `as.numeric("hello")`
- `as.numeric(TRUE)`

Vectors

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Although vectors might conjure up horrors of maths problems, in R, they are simply just a collection of the same type of data! For example $(1, 2, 3, 4)$ is a vector in R and so is $(\text{"cats"}, \text{"are"}, \text{"better"}, \text{"than"}, \text{"dogs"})$.

- To assign a variable as a vector you simply need to use the `c()` function.
- Examples are:
 - `x <- c(1, 5, 4, 9, 0)`
 - `x <- c("cats", "are", "better", "than", "dogs")`

Vectors Cont.

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- Elements of a vector can be accessed using square brackets, e.g. `x[2]` gives the second element in the vector `x`. New values can be assigned in this way.
- You can use the `typeof()` command to determine what kind of data is stored in a vector. For the two examples above, `typeof(x)` would return `double` and `character` respectively.
- There is also the `length()` command which does, just what you might expect, it returns the length of a vector. Use the `length()` command on the three examples above.

Task: Creating Vectors

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Let's bend the rules a bit. If we assigned

```
x<-c(1, 5.4, TRUE, "hello")
```

what would `typeof(x)` return? Why?

Solution: Creating Vectors

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- The output of `typeof(x)` gives `"character"` .
- Investigating further, printing `x` in the console gives:
`[1] "1" "5.4" "TRUE" "hello"`
- All the values have been converted to characters!
- Remember a vector can only contain values of a single data type.

Creating Numerical Vectors

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

There are many other ways to create vectors than by just specifying each and every element. A very useful command is the `:` operator.

- For example `x <- 1:7` will produce the vector `(1,2,3,4,5,6,7)` .
- Additionally, `x <- -2:2` will produce the vector `(-2,-1,0,1,2)` .

Creating Numerical Vectors Cont.

The Basics
Functions

Probability
and Statistics

Data Frames

Visualisation

Another useful command is the `seq()` function. The `seq` function is a little more tricky than the the previous command since you need to input three values into `seq()` for it to know what to do.

- `x <- seq(1,4,by = 0.5)` will produce a vector which *starts* at 1, *finishes* at 4 and goes up in *increments* by 0.5. So, in this case, `x` would be the vector `(1,1.5,2,2.5,3,3.5,4)`.
- You can also change the third value to `length.out()` which will intelligently produce a vector of the size specified from the starting and finishing values specified. For example, `seq(1, 5, length.out=4)` will produce the size 4 vector `(1.000000,2.333333,3.666667,5.000000)`.

Example: Creating Numerical Vectors

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

What vectors are produced by the following commands?

- `1.5:6`
- `1.5:2`
- `6:-2.4`
- `seq(-pi, pi, length.out=100)`
- `seq(20, 0, by=-4)`

Task: Creating Vectors II

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Create a vector `x` that contains values from 100 to 0, counting down by 2, i.e. 100, 98,...,2,0.
- 2 Add the element 1 to the end of the vector `x`.
- 3 Change the 2nd element of `x` to 1.

Solution: Creating Vectors II

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

```
1 x <- seq(100,0,-2)
```

```
2 x <- c(x,1)
```

```
3 x[2] <- 1
```

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Functions

Introduction to Functions

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Functions are a piece of code that does a certain task and can easily be reused. For example, the function `class` returns the data type.

- **Argument:** An input to the function.
 - **Positional arguments:** these are ordered and must be specified when calling the function.
 - **Keyword arguments:** these are optional and have a default value.
- **Return value:** the output of the function.

help

The `help(func)` command can be used to obtain useful information about the function like the parameters and the return value. Try running `help(log)` .

- `log(x, base = exp(1))` : the positional argument `x` does not have a default value and must be specified. Base is a optional-keyword argument that does not need to be specified unless a different base is needed.
- The arguments section indicates the type of inputs required for the function to work.

Useful Functions

The Basics
Functions

Probability
and Statistics

Data Frames

Visualisation

There are many useful built-in functions in R. Often there is one that does what you want - just search online for the correct function.

- Maths functions:
 - `abs(x)`
 - `sqrt(x)`
 - `round(x,digits=n)`
 - `log(x)`
 - `exp(x)`
 - `sin(x)`

Defining Your Own Function

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Repeating code should be avoided as it makes it more difficult to make changes. A new function should be made, which is called whenever the chunk of code is required. A function can be defined in the following way:

```
my_func <- function(arg1, arg2, kwarg1, kwarg2) {  
  # code  
  return(return_val)  
}
```

Example: Writing a New Function

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

The following function takes in two parameters *a* and *b* and calculates the geometric mean (square root of the product). Note that *a* and *b* have a *default value* of 1. (i.e. if *a* or *b* are not specified, then they are assumed to be 1)

```
gm <- function(a=1,b=1) {  
  product <- a * b  
  return(sqrt(product))  
}
```

Now the function can be used, e.g.

- `gm(1,2)`
- `gm()`

Task: Writing Your Own Function

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Write a function that does the following:

- 1 takes the arguments x_1 , y_1 , x_2 , y_2 which represent the coordinates of two points
- 2 returns the distance between them
- 3 x_2 and y_2 have a default value of 0.

Note: the distance is calculated using the following:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Solution: Writing Your Own Function

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

This is an example of a function that returns the distance between two points.

```
distance <- function(x1,y1,x2=0,y2=0) {  
  return(sqrt( (x1 - x2)^ 2 + (y1 - y2)^ 2 )) }  
}
```

You can now run your function e.g.

- `distance(1,0)` returns 1.
- `distance(0,1,3,1)` returns 3.

Think about a way you could adapt the code above to use vectors instead of coordinates. For example, what arguments would the function have now? How would the return line change to deal with vectors?

Loops and Conditionals

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Often you might want the code to run multiple times, or only if a condition holds. One way to do this is by using

- If statements
- For loops
- While loops.

If Statement

Often you want a line of code to run if a condition holds.

```
if (condition) {  
    # code  
} else {  
    # code  
}
```

The following example prints the square root of x if x is nonnegative, otherwise 0.

```
if (x < 0) {  
    print(0)  
} else {  
    print(sqrt(x))  
}
```

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

For Loop

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Sometimes you might want a piece of code to run many times.

```
for (i in vector) {  
  # code  
}
```

The following example prints the first 10 square numbers.

```
for (i in 1:10) {  
  print(i ^ 2)  
}
```


While Loop

Sometimes you might want a piece of code to keep running until some condition fails.

```
while (condition) {  
    # code  
}
```

The following example prints integers from 1 to 9.

```
i <- 1  
while (i < 10) {  
    print(i)  
    i <- i + 1  
}
```

Here, the code doesn't run anymore since after 10 loops, `i` would be 10 and so `10 < 10` would be a `FALSE` statement. It is much more preferred to use a for-loop rather than a while-loop; *but*, sometimes they are necessary.

The Basics

Functions

**Probability
and Statistics**

Data Frames

Visualisation

Probability and Statistics

Probability and Statistics

R has many useful functions for analysing large volumes of data. Suppose we have the vector

```
x <- c(70,74,81,60,91,80,55,89,86,62)
```

- `mean(x)` returns the mean of the values.
- `median(x)` returns the median of the values.
- `var(x)` returns the variance of the values.
- `sd(x)` returns the standard deviation of the values.
- `quantile(x,probs)` returns the quantile corresponding to the given probability. e.g. `quantile(x,0.5)` returns the median.
- `max(x)` and `min(x)` give the maximum and minimum values respectively.
- `summary(x)` returns a table of the min, 1st quartile, median, mean, 3rd quartile and max.

Probability Distributions

The Basics
Functions

Probability
and Statistics

Data Frames

Visualisation

Common probability distributions have four useful functions associated with them. Here we consider the standard normal distribution.

- `dnorm(x, mean = 0, sd = 1)` gives the mass/density of the distribution at x .
- `pnorm(q, mean = 0, sd = 1)` gives the cumulative distribution at q .
- `qnorm(p, mean = 0, sd = 1)` gives the p th quantile.
- `rnorm(n, mean = 0, sd = 1)` generates a random sample of size n .

Run `help(distributions)` for a list of other distributions and their names.

Example: Using Probability Distributions

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Below are examples where you might use these functions.

- 1 Suppose $X_1 \sim \text{Poisson}(\lambda = 3)$. Find $\mathbb{P}(X_1 = 1)$.

`dpois(1, 3)`

- 2 Suppose $X_2 \sim \text{Normal}(\mu = 10, \sigma^2 = 16)$. Find $\mathbb{P}(X_2 < 20)$.

`pnorm(20, 10, sd=4)`

- 3 Find x such that $\mathbb{P}(X_2 < x) = 0.975$.

`qnorm(0.975, 10, sd=4)`

- 4 Simulate 10 unbiased coin flips using the *Binomial*(1, 0.5) distribution.

`rbinom(10, 1, 0.5)`

Task: Using Probability Distributions

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Find the 95th percentile of an exponential distribution with rate parameter of 2.

- 1 By directly using the appropriate function.
- 2 By generating a large sample.

Solution: Using Probability Distributions

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

1 `qexp(0.95, rate = 2)` gives 1.497866 .

2 `x <- rexp(100000, rate = 2)`
`quantile(x,0.95)`

This is an approximation. This may be useful for more complex calculations where exact calculations are not possible/difficult, e.g. calculating the expected value of $g(X)$ with X following a known distribution.

The 'integrate' Function

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- `integrate(f, lower, upper)` performs numerical integration on a function.
- This can be useful for calculating probabilities and expectations of non-standard distributions.

Example: Using 'integrate'

The Basics
Functions

Probability
and Statistics

Data Frames

Visualisation

Say we wish to evaluate the following:

$$\int_0^{\infty} 2xe^{-2x} dx.$$

- 1 First define the integrand as a function:

```
f <- function(x){  
  return(2 * x * exp(-2*x))  
}
```

- 2 Use `integrate` , with the bounds as the arguments:
`integrate(f, 0, Inf)`

Statistical Tests

Statistical tests can be conducted by computing the test statistic yourself. Often there is a function available that does all of this. Here are two examples:

- `t.test(x, mu = 0, conf.level = 0.95)` where `mu` is the mean under the null hypothesis.
- `prop.test(x, n, p, conf.level)` where `n` is the number of trials, `p` the probability of success (under null hypothesis) and `x` the number of successes.

These functions output the test statistic, degrees of freedom, p-value and a confidence interval.

The Basics
Functions
Probability
and Statistics
Data Frames
Visualisation

Data Frames

What Are Data Frames?

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- A data frame is a structure that is used for storing data in R. It is a list of vectors, which are of equal length but can be of different data types.
- The vectors are presented as columns, each with a name, and represent variables. The rows represent observations and can be named or unnamed.
- While they can be constructed from scratch, data frames are typically produced from importing data sets.

Example: 'mtcars'

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 R has several built-in data frames which you can experiment with. Load the `mtcars` data frame and view it using the following code:

```
View(mtcars)
```

- 2 See what the variables and numbers mean by using the `help` function:

```
help(mtcars)
```

- 3 To see the full list of built-in data sets, use the following:

```
data()
```

Example: Importing Data

There are a few steps to importing data into R.

- 1 View the data set outside of R for inspection. Make sure to not make any changes.
- 2 Check that the file is saved in the same folder as the R file. Alternatively, you can change the working directory using the `setwd` command.
- 3 Import the data and assign it using the line (replacing "foo.csv" with the actual file name):

```
df <- read.csv("foo.csv", header = TRUE,  
stringsAsFactors = TRUE)
```
- 4 Check the data frame in R using:

```
View(df)
```

About 'read.csv'

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

A CSV file is a text file commonly used to store data.

`read.csv` loads it into R and converts it into a data frame.

- `header = TRUE` tells R that the first row of the data consists of headers. Change this to `FALSE` if this is not the case.
- `stringsAsFactors = TRUE` converts character variables into factors, which is useful for categorical data. Change this to `FALSE` if you want strings to be treated as character data.
- Remember to assign the newly created data frame to a variable.

Inspecting the Data

In addition to `View`, there are other useful functions to use once the data has been imported:

- `fix(df)` lets you manually edit the data in a separate window. Note that the edits are not recorded and must be repeated if the session is closed.
- `str(df)` returns the structure of the data frame.
- `summary(df)` gives a statistical summary of each variable in `df`.
- `plot(df)` will create a matrix plot of all the variables in `df`. This works better for data frames with less variables.

Task: Inspect A Data Frame

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Enter `library(MASS)` to access more pre-installed data frames.
- 2 Enter `data()` and scroll down until you reach “Data sets in package ‘MASS’”.
- 3 Choose one of the data frames and try out the four commands on the previous slide (`fix` , `str` , `summary` and `plot`) on it.

Working With Data Frames

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- `names(df)` will return a list of variable names in the data frame.
- To reference a variable, use a dollar sign (e.g. `df$var1`). This can also be used to add new variables to the data frame or overwrite existing ones.
- The data can also be extracted using square brackets (e.g. `df[row, col]`) and entering either the row and column names or the numerical coordinates of the cells. If only one value is supplied without a comma, R will look for a column.

Example: Extracting Data From Data Frames

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 To extract a single variable:

```
mtcars$mpg
```

- 2 The following will create a new variable:

```
mtcars$Lp100km <- 235/mtcars$mpg  
View(mtcars)
```

- 3 To extract different combinations of data, use square brackets:

```
mtcars["Datsun 710",]  
mtcars["Datsun 710", "hp"]  
mtcars[3,4]  
mtcars[1:5,c(1,4)]
```

Task: Extracting Data From Data Frames

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Use the 'mtcars' data frame to complete the following.

- 1 Create a new variable which contains the weight of each car in tonnes. Find the average of this variable. (Note: $1000 \text{ lbs} \approx 0.4536 \text{ tonnes}$)
- 2 Find the average horsepower in the first 10 cars.

Solution: Extracting Data From Data Frames

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

Use the following:

- 1 `mtcars$tonnes <- mtcars$wt * 0.4536`
`mean(mtcars$tonnes)`
- 2 `mean(mtcars[1:10, "hp"])`

You should get the following answers:

- 1 1.459345
- 2 122.8

Example: More Useful Functions

- 1 `attach(df)` allows you to reference the variables within 'df' directly. `detach` will undo this action.

```
attach(mtcars)  
mpg
```

- 2 `table(var)` returns a frequency table based on unique observations. This can be stored and plotted.

```
table(cyl)
```

- 3 `na.omit(df)` removes all rows containing NA values. Remember to assign this to a new variable name.

```
View(airquality)  
newairquality <- na.omit(airquality)  
View(newairquality)
```

Example: Packages

Packages are bundles of code that can be used to save writing lots of code yourself. A lot of tasks can be done by using a package which contains a function that does what you need.

- 1 Before using a package for the first time, it needs to be installed using the `install.packages` command:
`install.packages("dplyr")`
- 2 In order to use the functions in a package, it must be loaded using the `library` function:
`library(dplyr)`
`help(package="dplyr")`

The Basics
Functions
Probability
and Statistics
Data Frames
Visualisation

Visualisation

Using 'plot'

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- The `plot` function takes either one or two vectors of values and plots the points on a graph. This can be made into a scatter plot or a line plot.
- `points` will add a graph on top of the existing plot. This can be a line graph.
- `abline` is used to draw straight lines on the existing plot.
- `grid` adds a grid.
- `dev.off` will clear all plots.

Example: Using 'plot'

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Generate some values:

```
x <- seq(-pi, pi, length.out = 100)
y1 <- sin(x)
y2 <- cos(x)
```

- 2 Create the plot:

```
plot(x,y1)
grid()
points(x,y2)
abline(a=0, b=1)
```

- 3 Click "Zoom" to enlarge the image.

Arguments of 'plot'

The appearance of the plot can be modified by adding arguments to the `plot` function. Some of the more useful ones are below:

- `main="..."` - changes the title of the plot.
- `xlab="..."` and `ylab="..."` - changes the labels for the x-axis and y-axis respectively.
- `xlim=c(x1, x2)` and `ylim=c(y1, y2)` - specifies the horizontal and vertical boundaries of the plot respectively.
- `type="l"` - graphs a line graph instead of a scatter plot.
- `col="..."` - changes the colour of the graph.

Example: Using 'plot' again

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

1 Add additional arguments to the plot:

```
plot(x, y1, main="y = sin(x)",  
      ylab="sin(x)", ylim=c(-pi, pi),  
      type="l", col="red")  
grid()  
abline(a=0, b=1)
```

Controlling the Plot Window

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- `dev.off()` will clear the plot window.
- `par(mfrow=c(m,n))` will prepare an m by n panel to hold multiple plots. When plots are generated, they will be added row by row and from left to right.
- The `layout` function is also used for multipanel plots and allows for more customisation with regards to the placement and relative size of plots.

Task: Visualising the Normal Distribution

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Enter `par(mfrow=c(1,2))` to set up a 1 by 2 panel.
- 2 Plot the density function and cumulative distribution function of a standard normal distribution for values from -3 to 3.
Hint: start by creating the values for x. Use `dnorm` and `pnorm` for the density and cumulative functions.
- 3 Clear the plot window afterwards.

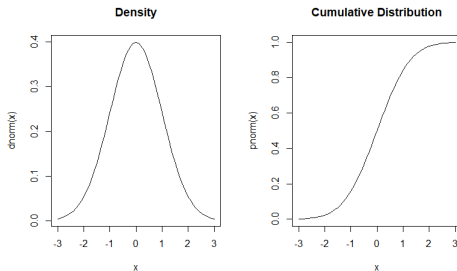
Solution: Visualising the Normal Distribution

```
1 par(mfrow=c(1,2))
```

```
2 x <- seq(-3,3,by=0.1)
```

```
plot(x, dnorm(x), main="pdf", type="l")
```

```
plot(x, pnorm(x), main="cdf", type="l")
```



```
3 dev.off()
```

Other Types of Plots

There are many other types of plots you can make on R. The functions below can be used to make some of them:

- `boxplot` creates a box plot, is used to summarise the distribution of values.
- `hist` creates a histogram, which is used to show the distribution of values.
- `barplot` creates a column graph, which is used to visualise the magnitudes of values.
- `pie` creates a pie chart, which is used to show the composition of a variable.

Similarly to `plot`, these can all be customised using arguments.

Example: Box Plots

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Creating a single box plot is simple:
`boxplot(mtcars$mpg)`
- 2 To split the box plot using another variable, use a tilde (values~categories). Add the data frame as the second argument to save typing it in the first argument
`boxplot(hp~vs, mtcars)`
- 3 Add arguments to customise the plot:
`boxplot(hp~vs, mtcars,
main="Horsepower by Engine Shape",
names=c("V-Shaped", "Straight"),
col=c("lightblue", "lightgreen"))`

Task: Box Plots

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Inspect the `ToothGrowth` data frame.
- 2 Create a series of box plots on tooth lengths grouped by dose amount.

Solution: Box Plots

1 `View(ToothGrowth)`

2 `boxplot(len~dose, ToothGrowth,
main="Tooth Length by Dose",
ylab="Length", xlab="Dose (mg/day)")`

The Basics

Functions

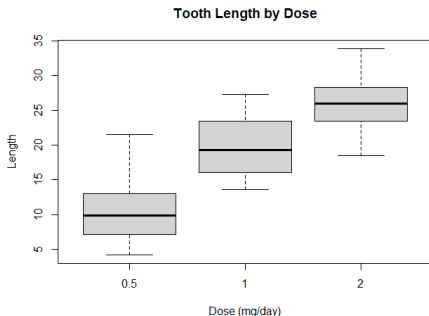
Probability
and Statistics

Data Frames

Visualisation

Solution: Box Plots

- 1 `View(ToothGrowth)`
- 2 `boxplot(len~dose, ToothGrowth,
main="Tooth Length by Dose",
ylab="Length", xlab="Dose (mg/day)")`



Example: Histograms

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 `hist` is also quite simple to use:

```
View(airquality)  
hist(airquality$Temp)
```

- 2 Use the 'breaks' argument to change the bins. Either supply a number of bins or a vector to indicate where the breaks should be. For example:

```
hist(airquality$Temp, breaks=42)  
hist(airquality$Temp, breaks=seq(55, 100,  
by=5))
```

Example: Bar Plots

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Bar plots are good for displaying the comparative magnitudes of different values. The 'VADeaths' data frame contains some numbers that can be compared.
`View(VADeaths)`
- 2 Create a bar plot for death rates in the 50-54 age group:
`barplot(VADeaths[1,], main="VA Death Rates For 50-54 Year Olds")`

Example: Pie Charts

The Basics

Functions

Probability
and Statistics

Data Frames

Visualisation

- 1 Compare the world distribution of phones in 1951 and 1961 using the 'WorldPhones' data set:

```
View(WorldPhones)  
par(mfrow=c(1,2))  
pie(WorldPhones["1951",], main="1951")  
pie(WorldPhones["1961",], main="1961")
```