A Brief Introduction to R

Computing Science and Mathematics Skill Sharing

Alexander E. I. Brownlee
Mila Goranova

UNIVERSITY of STIRLING

Rough content outline

1. basics of syntax: assignment, data types, printing, blocks, conditionals and loops, functions, help
2. data structures: scalar, vector, dataframes, loading from csv etc.
3. simple examples: multiply a vector, mean, sum, length, histogram; selecting data with which()
4. common stat tests
5. manipulating data
6. handy plots
7. ggplot
8. a linear model + syntax
9. machine learning: caret package
10. packages – e.g. `https://cran.r-project.org/web/packages/available_packages_by_name.html` `https://awesome-r.com`

Intro

R is a statistical programming language. It's free and has a rich variety of third party open-source libraries.

R – available from `https://www.r-project.org`

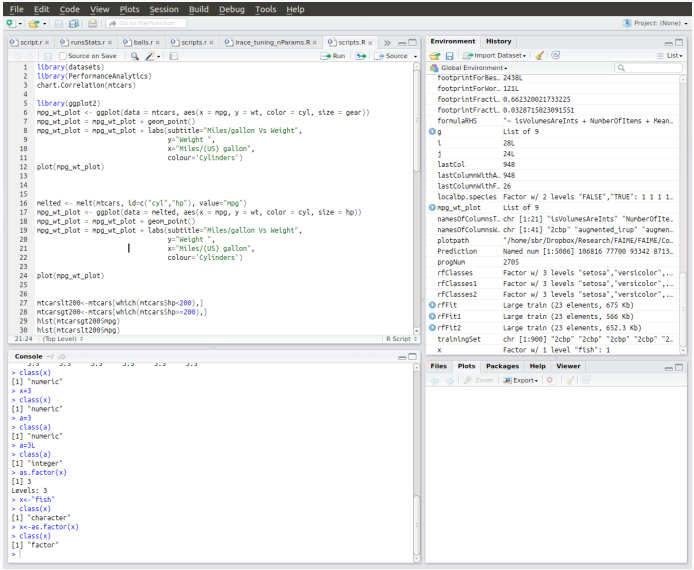
R Studio – IDE available from `https://www.rstudio.com`

This is a brief introduction covering the basics!

A lot of syntax and examples. Mainly to show what's possible. Slides will be available for reference.

* thanks to Kevin Swingler who prepared the BD6 R practical notes on which at least some of this is based

R Studio


File Edit Code View Plots Session Build Debug Tools Help
script.r
runnStats.r
balls.r
scripts.r
irace_tuning_nParams.R
scripts.R
Source on Save
Run
Source
library(datasets)
library(PerformanceAnalytics)
chart.Correlation(mtcars)
library(ggplot2)
mpg_wt_plot <- ggplot(data = mtcars, aes(x = mpg, y = wt, color = cyl, size = gear))
mpg_wt_plot = mpg_wt_plot + geom_point()
mpg_wt_plot = mpg_wt_plot + labs(subtitle="Miles/gallon Vs Weight",
y="Weight ",
x="Miles/(US) gallon",
colour='Cylinders')
plot(mpg_wt_plot)
melted <- melt(mtcars, id=c("cyl","hp"), value="mpg")
mpg_wt_plot <- ggplot(data = melted, aes(x = mpg, y = wt, color = cyl, size = hp))
mpg_wt_plot = mpg_wt_plot + geom_point()
mpg_wt_plot = mpg_wt_plot + labs(subtitle="Miles/gallon Vs Weight",
y="Weight ",
x="Miles/(US) gallon",
colour='Cylinders')
plot(mpg_wt_plot)
mtcarslt200<-mtcars[which(mtcars$hp<200),]
mtcarsgt200<-mtcars[which(mtcars$hp>=200),]
hist(mtcarsgt200$mpg)
hist(mtcarslt200$mpg)
21:24 (Top Level)
R Script
Environment
History
Import Dataset
List
Global Environment
footprintForBes… 2438L
footprintForWor… 121L
footprintFractL… 0.662320021733225
footprintFractL… 0.0328715023091551
formulaRHS "~ isVolumesAreInts + NumberOfItems + Mean…
g List of 9
i 28L
j 24L
lastCol 948
lastColumnWitha… 948
lastColumnWithF… 26
localbp.species Factor w/ 2 levels "FALSE","TRUE": 1 1 1 1…
mpg_wt_plot List of 9
namesOfColumnsT… chr [1:21] "isVolumesAreInts" "NumberOfIte…
namesOfColumnsW… chr [1:41] "2cbp" "augmented_irup" "augmen…
plotpath "/home/sbr/Dropbox/Research/FAIME/FAIME/Co…
Prediction Named num [1:5006] 106816 77700 93342 8713…
progNum 2705
rfClasses Factor w/ 3 levels "setosa","versicolor",…
rfClasses1 Factor w/ 3 levels "setosa","versicolor",…
rfClasses2 Factor w/ 3 levels "setosa","versicolor",…
rfFit Large train (23 elements, 675 Kb)
rfFit1 Large train (23 elements, 566 Kb)
rfFit2 Large train (23 elements, 652.3 Kb)
trainingSet chr [1:900] "2cbp" "2cbp" "2cbp" "2cbp" "2…
x Factor w/ 1 level "fish": 1
Console
> class(x)
[1] "numeric"
> x=3
> class(x)
[1] "numeric"
> a=3
> class(a)
[1] "numeric"
> a=3L
> class(a)
[1] "integer"
> as.factor(x)
[1] 3
Levels: 3
> x="fish"
> class(x)
[1] "character"
> x=as.factor(x)
> class(x)
[1] "factor"
Files
Plots
Packages
Help
Viewer
Zoom
Export

Basic syntax

Basic Syntax

Assignment: `x<-1`

(note R also supports = and -> but they can cause confusion! = is also used for equality, and right assignment is just hard to read)

Delete a variable: `rm(x)`

Print a variable: either just type its name and enter, or `print(x)`

Concatenate strings for output:

```
> cat(x,x,sep=",")
1,1
```

(note that functions have mandatory and optional arguments. You can also choose to explicitly name them, but don't have to.)

Commands can be separated by either a newline, or by `;`.

Blocks of code delimited by braces `{...}`

```
if (a==1) b=2 else b=3
```

For, While and Repeat loops are also supported, but we don't need them much because many operations can be applied to whole lists etc. in one operation.

```
> x<-c(3,4,5)
> for (a in x) print(a)
[1] 3
[1] 4
[1] 5
> for (a in 1:3) print(a)
[1] 1
[1] 2
[1] 3
```

Declaring a function

```
> multints<-function(n1,n2) {
>     result<-n1*n2;
>     result
> }
> multints(3,6)
[1] 18
```

You can also see the source of an existing function (not always available though...)

```
> multints
function(n1,n2) {
 result<-n1*n2;
 result
}
```

Help

e.g. `?mean` will load the documentation for a given function

e.g. `??histogram` will search the documentation for a given term

Types

Common data types: logical (TRUE/FALSE), numeric, character (string), factor (categorical)

Common variable types: scalar, vector, matrix, data frame

R is dynamically typed

You can change types at will:

```
> x<-3
> class(x)
[1] "numeric"
> x<-"fish"
> class(x)
[1] "character"
> x<-as.factor(x)
> class(x)
[1] "factor"
```

`summary(x)` will print a text summary of any variable (what you get depends on the type)

Vectors

Ordered lists of values of the same type.

```
> x<-c(1,3,54,7,89,5)
> x
[1] 1 3 54 7 89 5
> x[3]
[1] 54
> y<-x[2:4]
> y
[1] 3 54 7
> y<-x[-3]
> y
[1] 1 3 7 89 5
> y<-x[x>10]
> y
[1] 54 89
> x>10
[1] FALSE FALSE TRUE FALSE TRUE FALSE
> x<-seq(5,8,by=0.5)
> x
[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0
```

Note: indices start at 1 not 0!

Factors

Categorical or nominal variables. Limited set of values. Created automatically when we read in a file, or convert a character vector.

```
> x<-as.factor(c("apple","pear","pear","banana"))
> x[5]<-"orange"
Warning message:
In `[<-.factor`(`*tmp*`, 5, value = "orange") :
 invalid factor level, NA generated
> table(x)
 apple banana   pear
     1      1      2
> levels(x)
[1] "apple" "banana" "pear"
```

It is also possible to specify levels separately, and to give them an ordering so that e.g. `min(x)` makes sense.

Data frames

A table; multiple vectors of the same length addressable by name or index

```
> x<-c(2,4,7)
> y<-c("apples","pears","bananas")
> my_data<-data.frame(counts=x,fruit=y)
> summary(my_data)
```

```
     counts          fruit
 Min.   :2.000   apples :1
 1st Qu.:3.000   bananas:1
 Median :4.000   pears  :1
 Mean   :4.333
 3rd Qu.:5.500
 Max.   :7.000
```

Data frames

Indexing is `data[row,column]`; can also use `data$columnname` or `data["columnname"]`.

```
> my_data

  counts   fruit
1      2  apples
2      4   pears
3      7 bananas

> names(my_data)
[1] "counts" "fruit"
> my_data[2]
 fruit
1 apples
2 pears
3 bananas
> my_data[,2]
[1] apples pears bananas
Levels: apples bananas pears
> my_data[1,2]
[1] apples
Levels: apples bananas pears
```

Read/write files

```
> mydata = read.table("filename")
> mydata = read.csv("filename.csv")
```

`read.csv` is just a wrapper for `read.table` with some sensible defaults for CSVs. Both are highly configurable: delimiters, headers, nulls, etc.

Note: these will reformat column names (e.g. spaces changed to dots) by default.

Redirecting output

Use `sink(filename)` to redirect text output to a file. `sink()` will re-enable writing to stdout.

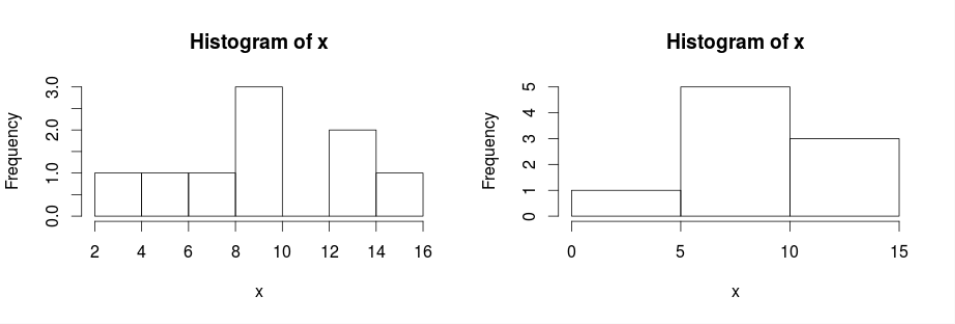
For graphics, use `pdf(filename)` or `png(filename)`. `dev.off()` will close the file.

Basic stats and plots

Basic operations and stats

```
> x<-c(3,6,8,10)
> x*2
[1] 6 12 16 20
> mean(x)
[1] 6.75
> median(x)
[1] 7
> sum(x)
[1] 27
> length(x)
[1] 4
```

```
> x<-c(3,6,8,10,10,10,13,14,15)
> hist(x)
> hist(x,breaks = 3)
```



`barplot()` exists for factors.

Basic operations and stats

```
> x<-c(2,5,8,10,12,14)
> y<-c(5,7,8,9,10,12)
> t.test(x,y)
 Welch Two Sample t-test
data: x and y
t = 0, df = 7.7253, p-value = 1
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:-4.811595 4.811595
sample estimates:
mean of x mean of y
 8.5 8.5
```

Actually the above returns an object that we can query...

```
> result<-t.test(x,y)
> result$p.value
[1] 1
> wilcox.test(x,y)
 Wilcoxon rank sum test with continuity correction
data: x and y
W = 19, p-value = 0.9357
alternative hypothesis: true location shift is not equal to 0
```

Basic operations and stats

For data frames, operations apply to the whole frame.

```
> x<-data.frame(a=c(1:10),b=c(31:40))
> sum(x)
[1] 410
> x*2
```

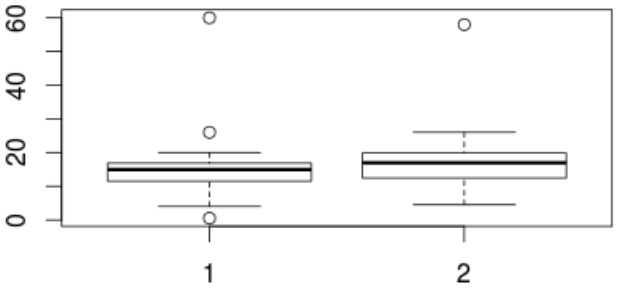
```
    a  b
1   2 62
2   4 64
3   6 66
4   8 68
5  10 70
6  12 72
7  14 74
8  16 76
9  18 78
10 20 80
```

Use `apply()` or one of its variants to apply a function to rows or columns separately. (MARGIN=1 is rows, 2 is columns)

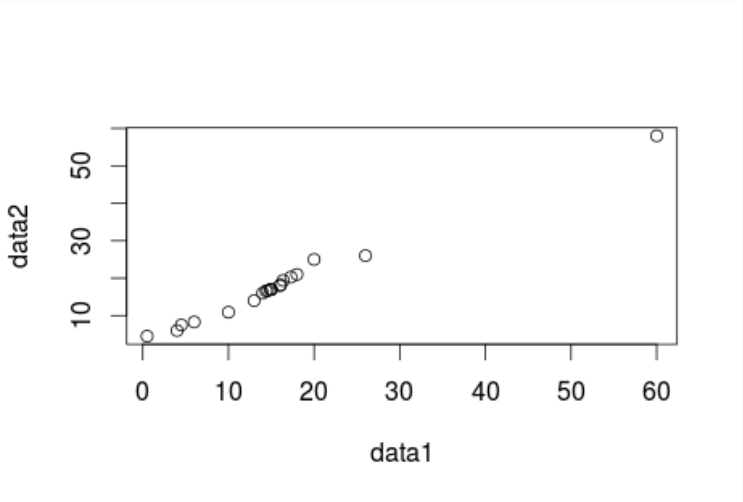
```
> apply(x,MARGIN=2,FUN=mean)
 a  b
 5.5 35.5
> apply(x,MARGIN=2,FUN=sd)
 a  b
 3.02765 3.02765
```

boxplot

```
data1 <- c
    (0.5,4,4.5,6,10,13,14,14.4,14.6,14.9,15,15,16,16.1,16.4,17.3,18,20,26,60)
data2 <- c
    (4.5,6,7.5,8.3,10.9,14,16,16.6,16.7,16.9,17,17,18,18.2,19.4,20.3,21,25,26,58)
boxplot(data1,data2)
```

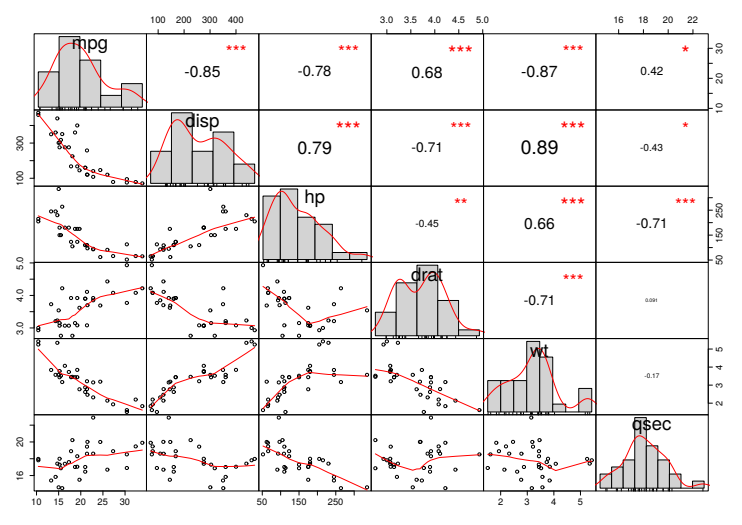


```
data1 <- c
    (0.5,4,4.5,6,10,13,14,14.4,14.6,14.9,15,15,16,16.1,16.4,17.3,18,20,26,60)
data2 <- c
    (4.5,6,7.5,8.3,10.9,14,16,16.6,16.7,16.9,17,17,18,18.2,19.4,20.3,21,25,26,58)
plot(data1,data2)
```



correlation matrix

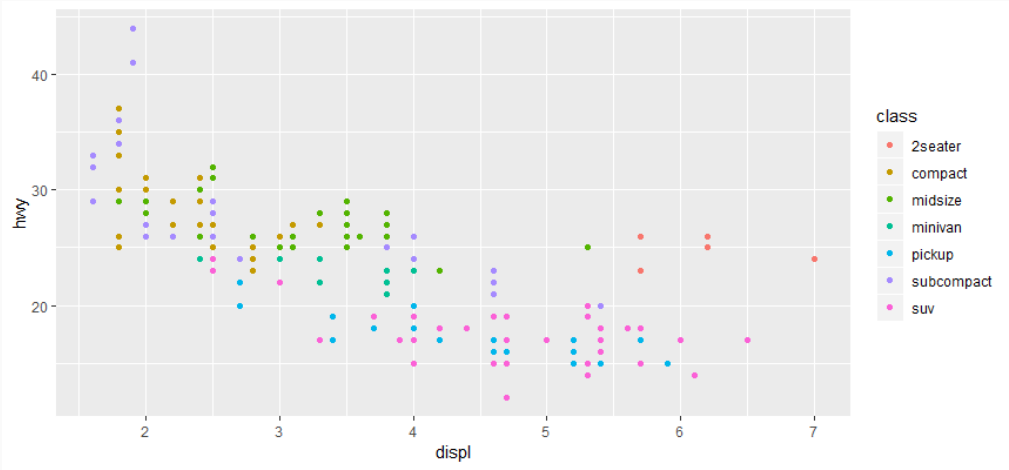
```
> install.packages("PerformanceAnalytics")
> library("PerformanceAnalytics")
> my_data <- mtcars[, c(1,3,4,5,6,7)]
> chart.Correlation(my_data, histogram=TRUE, pch=19)
```



The package ggplot2 is a powerful package that lets you create graphics. ggplot builds up plots in layers, which are combined using +. The basic plot and data are made using the ggplot() function, then we add the type of function we would like to use to plot our data.

```
install.packages("ggplot2")
library(ggplot2)

ggplot(mpg, aes(x = displ,  y = hwy, colour = class)) +
  geom_point()
```



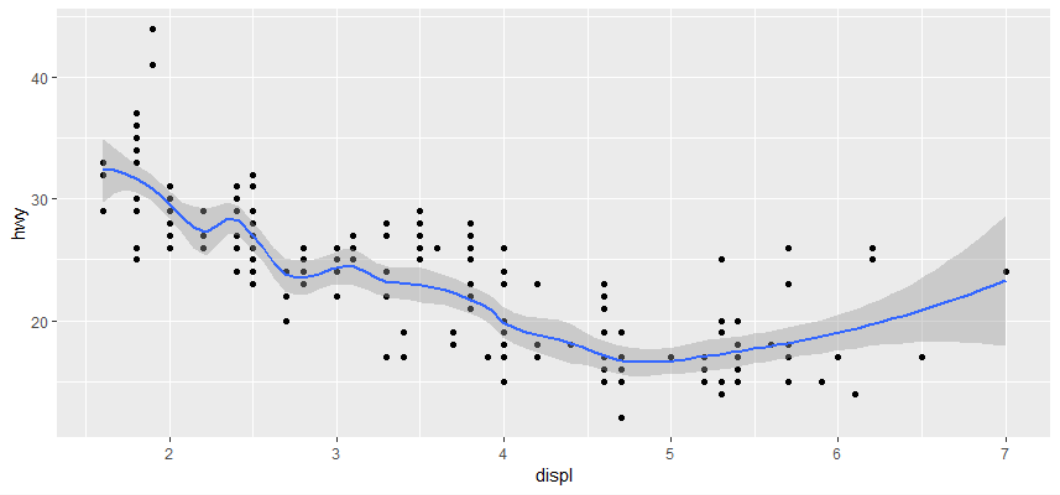
The ggplot() function takes the following parameters:

- dataset - in this example *mpg* is a build-in data set contains a subset of the fuel economy data.
- aes() function - aesthetic mapping specifying the variables and how to colour them. In our example we have chosen to visualize the *displ* and *hwy* variables from the data set and colour them by variable *class*.
- geom_point() function - a geom function to represent the data points. We have chose to create a scatterplots for the two variables using geom_point(). There are a lot of available function and a good summary could be found here - https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf .

```
ggplot(mpg, aes(x = displ,  y = hwy, colour = class)) +
  geom_point()
```

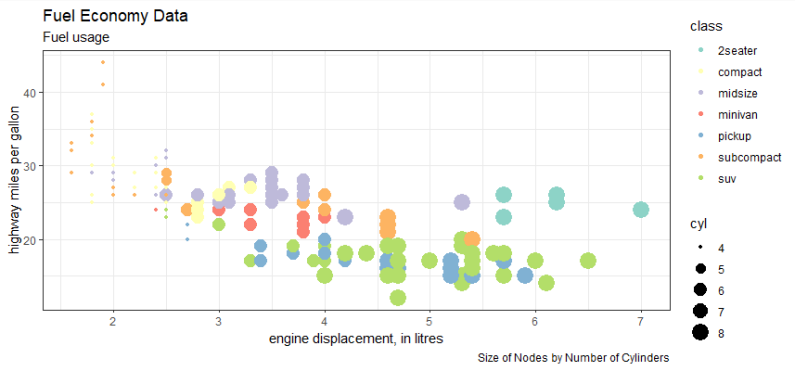
More ggplot

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.3)
```



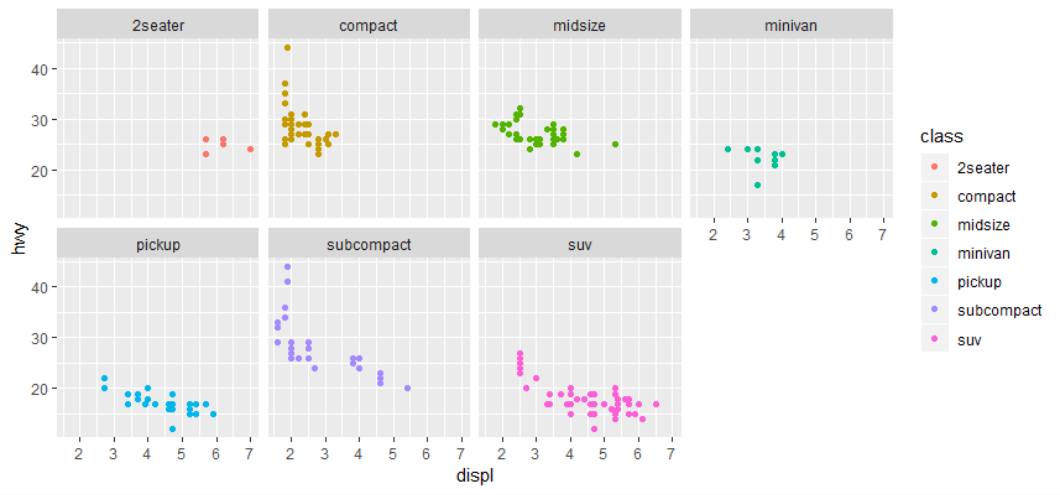
More ggplot

```
ggplot(mpg, aes(displ,hwy, colour = class, size = cyl)) +
  geom_point() +
  labs(subtitle="Fuel usage",
       y="highway miles per gallon",
       x="engine displacement, in litres",
       title="Fuel Economy Data",
       caption = "Size of Nodes by Number of Cylinders") +
  scale_color_brewer(palette="Set3") + theme_bw()
```



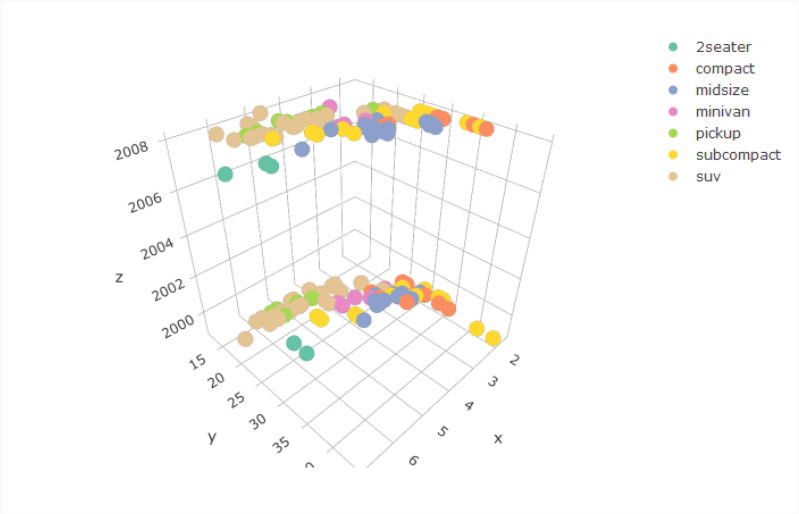
More ggplot

```
ggplot(mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color=class)) +
  facet_wrap(~ class, nrow = 2)
```



3D Scatter Plot

```
install.packages("plotly")
library(plotly)
plot_ly(x=mpg$displ, y=mpg$hwy, z=mpg$year, type="scatter3d", mode="markers",
    color=mpg$class)
```



Manipulating data

Selecting data

```
> x<-c(5,7,8,9,10,12)
> which(x>=10)
[1] 5 6
> x<-c(5,7,8,9,10,12,3,21,4)
> which(x>=10)
[1] 5 6 8
> x[which(x>=10)]
[1] 10 12 21
```

```
> x=data.frame(a=c(5,7,8,9,10,12),b=c("a","b","c","d","e","f"))
> x[which(x$a>=10),2]
[1] e f
Levels: a b c d e f
```

Merging datasets - cbind

Takes two sequences of vectors, matrices or data frames and combines them by columns.

```
my_data_one <- read.csv("filename-1.csv", header = T, sep = ",")
my_data_two <- read.csv("filename-2.csv", header = T, sep = ",")
my_data_both <- cbind(my_data_one, my_data_two)
```

The row number of the two must be equal.

Merging datasets - rbind

Takes two sequences of vectors, matrices or data frames and combines them by rows.

```
my_data_one <- read.csv("filename-1.csv", header = T, sep = ",")
my_data_two <- read.csv("filename-2.csv", header = T, sep = ",")
my_data_both <- rbind(my_data_one, my_data_two)
```

The column number of the two must be equal.

Melting

Sometimes we need to rearrange the data into what is called ”long form”. This is often needed for libraries like ggplot. The function to do this is `melt()` in the `reshape2` library.

```
        > dat
  FactorA FactorB     Group1     Group2     Group3      Group4
1     Low     Low -1.1616334 -0.5228371 -0.6587093  0.45064563
2  Medium     Low -0.5991478 -1.0461138 -0.1942979  2.47985577
3    High     Low  0.8420797 -1.5413266  0.6318852 -0.98948125
4     Low  Medium  1.6225569 -1.2706469 -0.8026467 -0.32332181
5  Medium  Medium -0.3450745 -1.3377985  1.4988363  0.36541918
6    High  Medium  1.6025044  0.7631882 -0.5375833  0.85028148
7     Low    High -1.2991011 -0.2223622 -0.6321478 -1.57284216
8  Medium    High -0.4906400 -1.1802192  0.1235253  0.09891793
9    High    High  0.3897769 -0.3832142  0.6671101  0.23407257
```

Melting

```
> melt(dat)
Using FactorA, FactorB as id variables
   FactorA FactorB variable       value
1      Low     Low   Group1 -1.16163338
2   Medium     Low   Group1 -0.59914783
3     High     Low   Group1  0.84207974
4      Low  Medium   Group1  1.62255690
5   Medium  Medium   Group1 -0.34507455
6     High  Medium   Group1  1.60250438

...
36    High    High   Group4  0.23407257
```

Models

```
linearmodel<-lm(data$TaxiTime ~ data$distance + data$angle_sum)
```

This will perform a simple linear regression, and store the model in the variable called `linearmodel`.

Syntax is a bit strange: before the ~ is the variable we want to predict, and after the ~ are the variables we are using as inputs.

```
> linearmodel

Call:
lm(formula = data$TaxiTime ~ data$distance + data$angle_sum)

Coefficients:
   (Intercept)   data$distance  data$angle_sum
      1.555937        0.002670        0.006377
```

```
> summary(linearmodel)

Call:
lm(formula = data$TaxiTime ~ data$distance + data$angle_sum)

Residuals:
   Min     1Q Median     3Q    Max
-8.442 -3.788 -0.440  2.453 40.504

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.5559371  0.5427447   2.867  0.00423 **
data$distance  0.0026700  0.0003437   7.768 1.92e-14 ***
data$angle_sum 0.0063771  0.0011696   5.452 6.22e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.011 on 1032 degrees of freedom
Multiple R-squared:  0.2091,^^IAdjusted R-squared:  0.2076
F-statistic: 136.4 on 2 and 1032 DF,  p-value: < 2.2e-16
```

Install the *caret* package (library), load it, and load some sample data (R includes some standard data sets for free!).

```
> install.packages("caret")
> library(caret)
> data(iris)
```

Split into training and test data:

```
set.seed(10)
inTrain <- createDataPartition(y=iris$Species, p=.6, list=FALSE)
training <- iris[ inTrain,]
testing <- iris[-inTrain,]
```

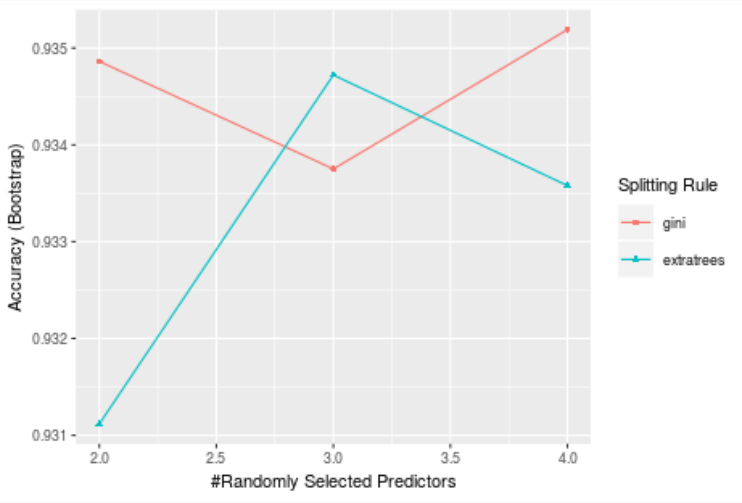
`set.seed()` sets the random number generator's seed, which ensures the results are repeatable. The remaining lines create two variables "training" and "test" that have our subsets of the data.

Train a model on all the training data.

- The first argument follows the notation we used for linear regression: ”Species” is the variable want to predict, then ~ then all the features we want in the model. A dot (.) means we want all the features.
- We then specify that we want to use the training data, and the model type is ”ranger”, a kind of random forest. Models supported by Caret are here: `https://topepo.github.io/caret/available-models.html`
- preproc specifies any preprocessing we want to do on the data
- Caret makes use of many other packages and will ask to install them if needed

```
rfFit1 <- train(
Species ~ .,
data = training,
method = "ranger",
## Center and scale the predictors for the training
## set and all future samples.
preProc = c("center", "scale")
)
```

By default, caret will explore the hyperparameters for the model, and you can see the results of that search with `ggplot(rfFit1)`:



Some machine learning...

Having got our model, we can then try predicting the values for our unseen data:

```
> rfClasses1 <- predict(rfFit1, newdata = testing)
> str(rfClasses1)
 Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> confusionMatrix(data = rfClasses1, testing$Species)

Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         20         2
  virginica       0          0        18

Overall Statistics

               Accuracy : 0.9667
                 95% CI : (0.8847, 0.9959)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.95
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            1.0000           0.9000
Specificity                 1.0000            0.9500           1.0000
Pos Pred Value              1.0000            0.9091           1.0000
...
```

Some handy packages

- igraph - network analysis and visualisation package
- tidyverse - collection of packages for data science
- latex integration...

Questions / discussion

R - available from `https://www.r-project.org`

R Studio - IDE available from `https://www.rstudio.com`