Introduction to Statistical Programming

Timothy Wong[1]
James Gammerman[2]

March 19, 2018

[1] timothy.wong@hotmail.co.uk
[2] person@organisation.com

Outline

What is the R language?

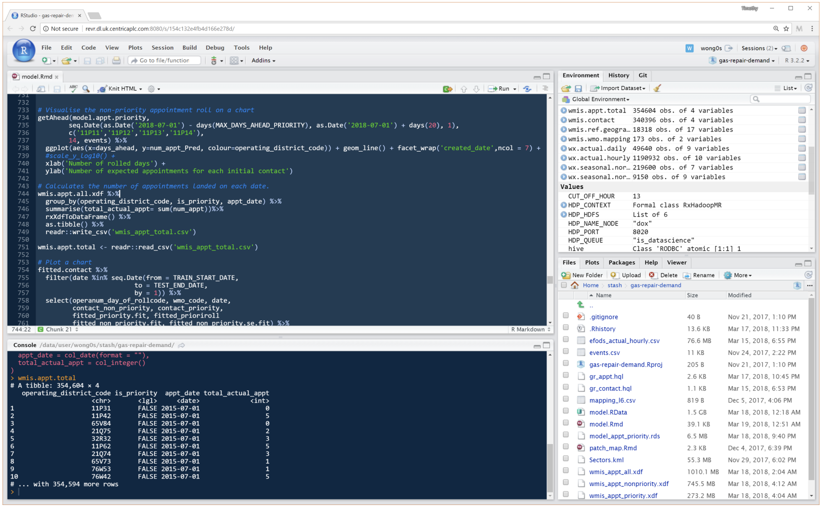
- Offers modern and sophisticated statistical algorithms
- Used by millions of analysts and researchers worldwide
- Has a thriving open-source community
- Enables big Data analytics

Easy to Use

- `PROC REG` = `lm()` or `glm()`
- `PROC SQL` = `%>%`
- `PROC SORT` = `arrange()`
- `PROC MEANS` = `mean()`, `sd()`
- `PROC GPLOT` = `plot()`, `ggplot()`, `autoplot()`

RStudio Server Pro

- RStudio is your integrated development environment (IDE)



```r
# Visualise the non-priority appointment roll on a chart
getAhead(model.appt.priority,
         seq.Date(as.Date('2018-07-01') - days(MAX_DAYS_AHEAD_PRIORITY), as.Date('2018-07-01') + days(20), 1),
         c('11P11','11P12','11P13','11P14'),
         14, events) %>%
  ggplot(aes(x=days_ahead, y=num_appt_Pred, colour=operating_district_code)) + geom_line() + facet_wrap('created_date',ncol = 7) +
  #scale_y_log10() +
  xlab('Number of rolled days') +
  ylab('Number of expected appointments for each initial contact')

# Calculates the number of appointments landed on each date.
wmis.appt.all.xdf %>%
  group_by(operating_district_code, is_priority, appt_date) %>%
  summarise(total_actual_appt= sum(num_appt))%>%
  rxXdfToDataFrame() %>%
  as.tibble() %>%
  readr::write_csv('wmis_appt_total.csv')

wmis.appt.total <- readr::read_csv('wmis_appt_total.csv')

# Plot a chart
fitted.contact %>%
  filter(date %in% seq.Date(from = TRAIN_START_DATE,
                            to = TEST_END_DATE,
                            by = 1)) %>%
  select(operanum_day_of_rollcode, wmo_code, date,
         contact_non_priority, contact_priority,
         fitted_priority.fit, fitted_prioriroll
         fitted_non_priority.fit, fitted_non_priority.se.fit) %>%
```

Console /data/user/wong0s/stash/gas-repair-demand/

```
  appt_date = col_date(format = ""),
  total_actual_appt = col_integer()
)
> wmis.appt.total
# A tibble: 354,604 × 4
   operating_district_code is_priority  appt_date total_actual_appt
                     <chr>       <lgl>     <date>             <int>
1                    11P31       FALSE 2015-07-01                 0
2                    11P42       FALSE 2015-07-01                 5
3                    65V84       FALSE 2015-07-01                 0
4                    21Q75       FALSE 2015-07-01                 2
5                    32R32       FALSE 2015-07-01                 3
6                    11P62       FALSE 2015-07-01                 5
7                    21Q74       FALSE 2015-07-01                 3
8                    65V73       FALSE 2015-07-01                 1
9                    76W53       FALSE 2015-07-01                 1
10                   76W42       FALSE 2015-07-01                 5
# ... with 354,594 more rows
> 
```

wmis.appt.total	354604 obs. of 4 variables
wmis.contact	340396 obs. of 4 variables
wmis.ref.geogra…	18318 obs. of 17 variables
wmis.wmo.mapping	173 obs. of 2 variables
wx.actual.daily	49640 obs. of 9 variables
wx.actual.hourly	1190932 obs. of 10 variables
wx.seasonal.nor…	219600 obs. of 7 variables
wx.seasonal.nor…	9150 obs. of 9 variables
Values	
CUT_OFF_HOUR	13
HDP_CONTEXT	Formal class RxHadoopMR
HDP_HDFS	List of 6
HDP_NAME_NODE	"dox"
HDP_PORT	8020
HDP_QUEUE	"is_datascience"
hive	Class 'RODBC' atomic [1:1] 1

Name	Size	Modified
..		
.gitignore	40 B	Nov 21, 2017, 1:10 PM
.Rhistory	13.6 KB	Mar 17, 2018, 11:33 PM
efods_actual_hourly.csv	76.6 MB	Mar 15, 2018, 6:55 PM
events.csv	11 KB	Nov 24, 2017, 2:22 PM
gas-repair-demand.Rproj	205 B	Nov 21, 2017, 1:10 PM
gr_appt.hql	2.6 KB	Mar 17, 2018, 10:45 PM
gr_contact.hql	1.1 KB	Mar 15, 2018, 6:53 PM
mapping_l6.csv	819 B	Dec 5, 2017, 4:06 PM
model.RData	1.5 GB	Mar 18, 2018, 12:18 AM
model.Rmd	39.1 KB	Mar 19, 2018, 12:51 AM
model_appt_priority.rds	6.5 MB	Mar 18, 2018, 9:40 PM
patch_map.Rmd	2.3 KB	Dec 4, 2017, 6:39 PM
Sectors.kml	55.3 MB	Nov 29, 2017, 6:02 PM
wmis_appt_all.xdf	1010.1 MB	Mar 18, 2018, 2:04 AM
wmis_appt_nonpriority.xdf	745.5 MB	Mar 18, 2018, 4:12 AM
wmis_appt_priority.xdf	273.2 MB	Mar 18, 2018, 4:04 AM

Packages

- **CRAN** is the Comprehensive R Archive Network.
- User-contributed packages: source code, binaries, documentation.

```r
# Install a new package with all its dependencies
install.packages('ggplot2', dependencies = TRUE)
# Load an installed package
# (both lines are identical)
library(ggplot2)
library('ggplot2')
```

Packages

- **CRAN Task View** is a curated list of packages.
- `https://cran.r-project.org/web/views/`

```
library(ctv)
# Install a CRAN Task View
install.views("Econometrics")
# Update a CRAN Task View
update.views("Econometrics")
```

Vectors

- R is a vectorised programming language.
- Vector contains objects of the same data type.

```
# Create a vector of integers one to ten
myVec1 <- 1:10
# Find out the length of vector
length(myVec1)

## [1] 10

# Reverse the vector
rev(myVec1)

##  [1] 10  9  8  7  6  5  4  3  2  1

# Create a custom vector of 10, 15, 20, 25, 30
myVec2 <- c(10, 15, 20, 25, 30)
myVec2

## [1] 10 15 20 25 30

# Create a vector of sequential numbers
myVec3 <- seq(from = -10, to = 10, by = 0.5)
myVec3

##  [1] -10.0  -9.5  -9.0  -8.5  -8.0  -7.5  -7.0  -6.5  -6.0  -5.5  -5.0
## [12]  -4.5  -4.0  -3.5  -3.0  -2.5  -2.0  -1.5  -1.0  -0.5   0.0   0.5
## [23]   1.0   1.5   2.0   2.5   3.0   3.5   4.0   4.5   5.0   5.5   6.0
## [34]   6.5   7.0   7.5   8.0   8.5   9.0   9.5  10.0
```

Subsetting a Vector

- You can subset members from a vector.

```r
# Select the second member of the vector
myVec1[2]

## [1] 2

# Subset a range from the vector
myVec1[2:4]

## [1] 2 3 4

# Subset specified elements
myVec1[c(4,2,3)]

## [1] 4 2 3
```

Vectorised Operations

- Operations in R are vectorised.

```r
# Arithmetic operations
myVec1 + 10

##  [1] 11 12 13 14 15 16 17 18 19 20

myVec1 - 10

##  [1] -9 -8 -7 -6 -5 -4 -3 -2 -1  0

myVec1 * 2

##  [1]  2  4  6  8 10 12 14 16 18 20

myVec1 / 2

##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

myVec1 ^ 2

##  [1]   1   4   9  16  25  36  49  64  81 100

log(myVec1)

##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

Looping

- Looping can be slow.
- Always try to vectorise your code.

```r
# Vectorised operation is fast
system.time({
  myResult <- 1:100000 * 2
})

##    user  system elapsed
##       0       0       0

# Looping is quite slow
system.time({
  myResult <- sapply(1:100000, function(x){ x * 2 })
})

##    user  system elapsed
##    0.05    0.02    0.06

# Appending to vector is much slower
system.time({
  myResult <- c()
  for(i in 1:100000){
    myResult <- c(myResult, i * 2)
  }
})

##    user  system elapsed
##   16.01    0.15   16.23
```

Functions

- Functions are vectorised.

```r
# Defines a custom function
myFunc <- function(x) {
  x * 2
}
# Execute the function with one input
myFunc(5)

## [1] 10

# Execute the function with an integer vector
myFunc(1:10)

##  [1]  2  4  6  8 10 12 14 16 18 20
```

Character Vectors

- Vector can also contain character objects.

```
# Vector can contain character objects
myVec4 <- c('Bill', 'Mark', 'Steve', 'Jeff', 'Larry')
myVec4

## [1] "Bill"  "Mark"  "Steve" "Jeff"  "Larry"

# Constant character vectors in R
LETTERS

##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

letters

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"

month.name

##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"

month.abb

##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
## [12] "Dec"
```

Vectors - Other Data Types

- Vector can contain objects of any data type.

```r
# This is a vector of character objects
myVec5 <- c('2017-07-13',
            '2017-10-11',
            '2017-11-21',
            '2018-01-16',
            '2018-03-27')
# This is a vector of date objects
myVec6 <- as.Date(myVec5)
# Compute the day of week - returns a vector of characters
# Notice that these are all vectorised functions
weekdays(myVec6)

## [1] "Thursday"  "Wednesday" "Tuesday"   "Tuesday"   "Tuesday"
```

List

- List is a generic container for objects of different data types

```
myFavBook <- list(title = 'R for Data Science',
                  authors = c('Garrett Grolemund', 'Hadley Wickham'),
                  publishDate = as.Date('2016-12-12'),
                  price = 18.17,
                  currency = 'USD',
                  edition = 1,
                  isbn = 1491910399)
myFavBook
```

```
## $title
## [1] "R for Data Science"
##
## $authors
## [1] "Garrett Grolemund" "Hadley Wickham"
##
## $publishDate
## [1] "2016-12-12"
##
## $price
## [1] 18.17
##
## $currency
## [1] "USD"
##
## $edition
## [1] 1
##
## $isbn
## [1] 1491910399
```

Subsetting a List

- You can subset a particular member from a list

```
# Select a named member of a list
# Using the dollar sign, followed by name without bracket
myFavBook$title

## [1] "R for Data Science"

# Using double squared brackets with member's name as string
myFavBook[['authors']]

## [1] "Garrett Grolemund" "Hadley Wickham"

# Select the fourth member in the list
myFavBook[[4]]

## [1] 18.17
```

Special Numbers in R

```r
# Pi is constant 3.14159...
pi

## [1] 3.141593

# One divided by zero is infinity
1/0

## [1] Inf

# Negative number divided by zero is negative infinity
-1/0

## [1] -Inf

# Infinity divided by infinity is Not-a-Number (NaN)
Inf/Inf

## [1] NaN

# Not available (NA) plus one is still NA
NA + 1

## [1] NA

# Effects of different special numbers
c(5, 10, 15, NA, 25, 30, NaN, 35, 40, Inf, 50, -Inf, 60) / 5

##  [1]    1    2    3   NA    5    6  NaN    7    8  Inf   10 -Inf   12
```

Data Frame

- Table with rows (observations) and columns (variables).
- Analogous to an Excel workbook.

```r
myFavMovies1 <- data.frame(title = c('Dr. No',
                                     'Goldfinger',
                                     'Diamonds are Forever',
                                     'Moonraker',
                                     'The Living Daylights',
                                     'GoldenEye',
                                     'Casino Royale'),
                           year = c(1962, 1964, 1971, 1979,
                                    1987, 1995, 2006),
                           box = c(59.5, 125, 120, 210.3,
                                   191.2, 355, 599),
                           bondActor = c('Sean Connery',
                                         'Sean Connery',
                                         'Sean Connery',
                                         'Roger Moore',
                                         'Timothy Dalton',
                                         'Pierce Brosnan',
                                         'Daniel Craig'))
```

Data Frame

```
myFavMovies1

##                      title year   box      bondActor
## 1                   Dr. No 1962  59.5   Sean Connery
## 2               Goldfinger 1964 125.0   Sean Connery
## 3 Diamonds are Forever 1971 120.0   Sean Connery
## 4                Moonraker 1979 210.3    Roger Moore
## 5 The Living Daylights 1987 191.2 Timothy Dalton
## 6                GoldenEye 1995 355.0 Pierce Brosnan
## 7            Casino Royale 2006 599.0   Daniel Craig
```

Tibble

- Similar to traditional `data frame`.
- `tibble` is the modern standard in R.

```r
library(dplyr)
myFavMovies2 <- tibble(title = c('Dr. No',
                                 'Goldfinger',
                                 'Diamonds are Forever',
                                 'Moonraker',
                                 'The Living Daylights',
                                 'GoldenEye',
                                 'Casino Royale'),
                       year = c(1962, 1964, 1971, 1979,
                                1987, 1995, 2006),
                       box = c(59.5, 125, 120, 210.3,
                               191.2, 355, 599),
                       bondActor = c('Sean Connery',
                                     'Sean Connery',
                                     'Sean Connery',
                                     'Roger Moore',
                                     'Timothy Dalton',
                                     'Pierce Brosnan',
                                     'Daniel Craig'))
```

Tibble

```
myFavMovies2

## # A tibble: 7 x 4
##                    title  year   box       bondActor
##                    <chr> <dbl> <dbl>           <chr>
## 1                 Dr. No  1962  59.5    Sean Connery
## 2             Goldfinger  1964 125.0    Sean Connery
## 3 Diamonds are Forever  1971 120.0    Sean Connery
## 4              Moonraker  1979 210.3     Roger Moore
## 5 The Living Daylights  1987 191.2 Timothy Dalton
## 6              GoldenEye  1995 355.0 Pierce Brosnan
## 7          Casino Royale  2006 599.0    Daniel Craig
```

Subsetting a Tibble

```r
# Get one column by name
myFavMovies2[['title']]
myFavMovies2$title
# Get a range of columns by position ID
myFavMovies2[, 1:2]
myFavMovies2[1:2]
# Get rows 1 to 3
myFavMovies2[1:3, ]
# Get the 'year' variable of row 1-3
myFavMovies2[1:3, 'year']
# Get the 'title' and 'year' variables of row 4-7
myFavMovies2[4:7, c('title','year')]
```

Simple Regression

Univariate linear regression model

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

- Analogous to a straight line $y = mx + c$
- Can be chained with M dependent variables (Multivariate)

$$\hat{y}_i = \beta_0 + \sum_{m=1}^{M} \beta_m x_{m,i}$$

- Residual term $\epsilon_i = y_i - \hat{y}_i$ assumed to be Gaussian

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

- Also known as ordinary least squared (OLS) regression

Linear Regression in R

```
# Build a univariate linear model
# These two lines are equivalent
myModel1 <- lm(mpg ~ wt, mtcars)
myModel1 <- lm(formula = mpg ~ wt, data = mtcars)
# Read the model summary
summary(myModel1)

##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528,Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

More Linear Regression Models

Multivariate linear model Additional independent variables can be chained using the + symbol. Categorical variables can be encoded as dummy on-the-fly using `factor()`.

```
myModel2 <- lm(mpg ~ wt + hp + qsec + factor(am), mtcars)
```

Polynomial term Model can become more flexible when an independent variable is converted into polynomial terms. Use the `poly()` function.

```
myModel3 <- lm(mpg ~ wt + qsec + factor(am) +
                 poly(hp, 3), mtcars)
```

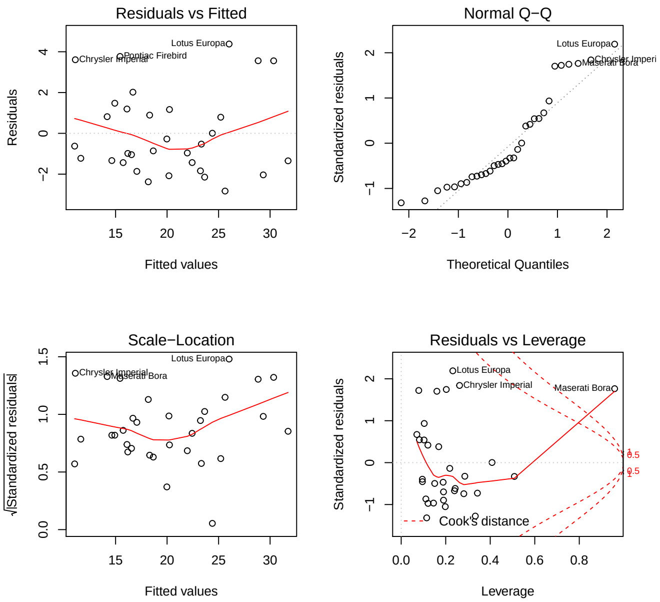
Interaction term Two variables can be combined to create synergy effect. The * symbol is used to combine variables together.

```
myModel4 <- lm(mpg ~ wt * hp + qsec + factor(am), mtcars)
```

Regression Diagnostics

- **Residuals vs Fitted** Checks for non-linear relationship. Look for a near horizontal line.
- **Normal Quantile-Quantile** It aligns model residuals against a theoretical normal distribution. If the residuals spread along a straight diagonal line on the Q-Q plot, it suggests that the residuals are normally distributed.
- **Scale-Location** Checks for homoscedasticity and heteroscedasticity. It is homoscedastic if observations scatter without any observable pattern.
- **Residual vs Leverage (Cook's Distance)** Identifies observations having strong influence to the model.

Diagnostics Plots

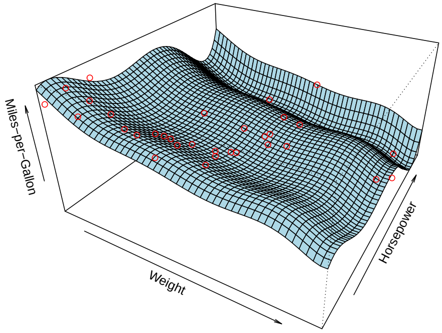


Overfitting

- Flexible models are prone to overfitting.
- Overfitting makes the model less generalisable.
- Solution
  - Use less flexible methods.
  - Impose regularisation.

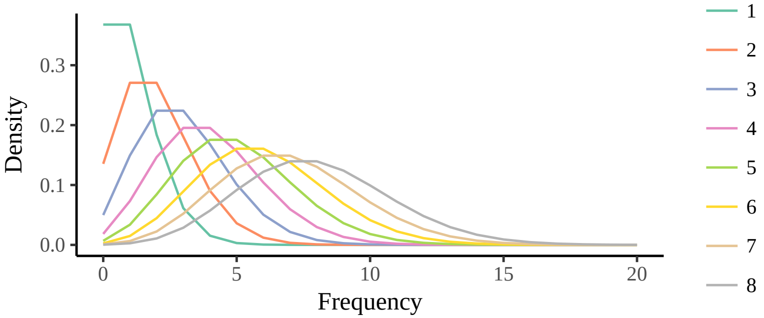
Overfitting: Visual Explaination

$$\hat{y} = \beta_0 + \sum_{j=1}^{8} \beta_{wt_j} x_{wt}^j + \sum_{k=1}^{5} \beta_{hp_k} x_{hp}^k$$



Poisson Distribution

- Count of distinct events are drawn from Poisson distribution.
  - Always positive.
  - In most cases they are integers.
- e.g. Number of people in a room, number of flights delayed per day... etc.



Testing for Poisson Distribution

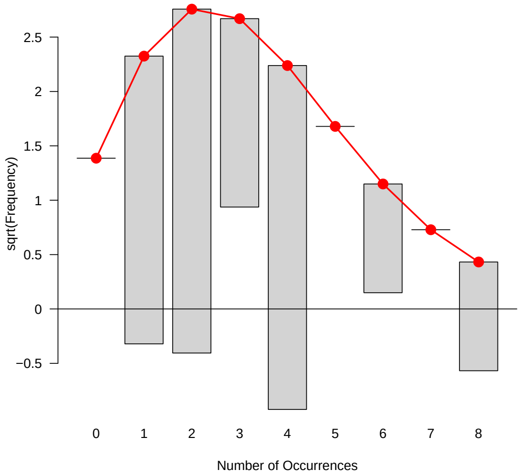
- Chi-square goodness of fit test.

```r
# Performs the Chi-squared goodness-of-fit test.
# It checks whether the variable is drawn from a Poisson distribution.
library(vcd)
gf <- goodfit(mtcars$carb, type= "poisson", method= "ML")
# Checks the statistical p-value of the goodness-of-fit test.
# If p<=0.05 then it is safe to say that the variable is Poisson.
summary(gf)
```

```
##
##   Goodness-of-fit test for poisson distribution
##
##                       X^2 df     P(> X^2)
## Likelihood Ratio 20.53973  4 0.0003906369
```

Goodness of Fit Plot

```r
# Plots the observed frequency vs theoretical Poisson distribution.
# The hanging bars should fill the space if it was perfectly Poisson.
plot(gf)
```



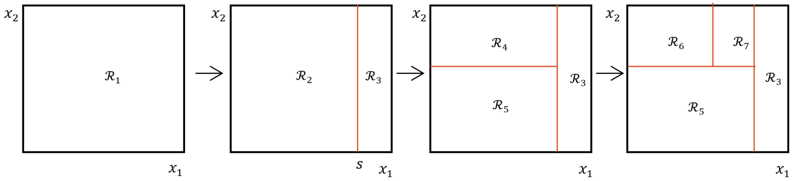
Poisson Regression

```
# Build a Poisson model to predict the number of carburetors in a car.
myPoissonModel <- glm(carb ~ hp + wt + factor(am),
                      family="poisson",
                      data=mtcars)
# Read the model summary
summary(myPoissonModel)

##
## Call:
## glm(formula = carb ~ hp + wt + factor(am), family = "poisson",
##     data = mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.91420  -0.48423  -0.07246   0.19252   1.26155
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.418081   0.604211  -0.692   0.4890
## hp           0.004316   0.001880   2.296   0.0217 *
## wt           0.179583   0.191352   0.938   0.3480
## factor(am)1  0.393750   0.324978   1.212   0.2257
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 27.043  on 31  degrees of freedom
## Residual deviance: 10.798  on 28  degrees of freedom
## AIC: 108.16
##
## Number of Fisher Scoring iterations: 4
```

Recursive Partitioning

- Cut off point is denoted as s.
- Divides data into regions (leaves) $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, ...$ recursively.
- Works with real values as well as categorical variables.
- Large tree risks overfitting
  - Removes weaker leaves.
  - Regularisation.



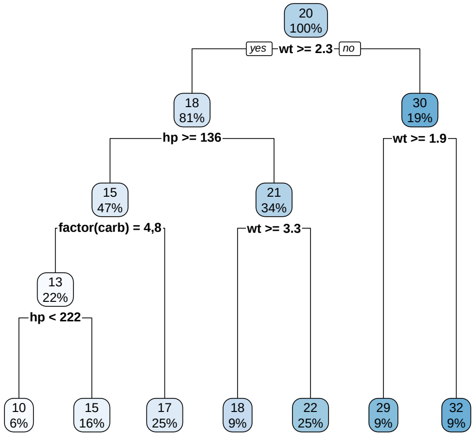
Decision Tree

- Trees can be trained with a formula and optional control parameters.

```r
# Load the rpart package for recursive partitioning
library(rpart)
# Build a decision tree to predict mpg
myTree <- rpart(formula = mpg ~ wt + hp +
                  factor(carb) +
                  factor(am),
                data = mtcars,
                control = rpart.control(minsplit=5))
# Read the detailed summary of the tree
summary(myTree)
```

Decision Tree: Visualisation

```r
# Load the rpart.plot package for tree visualisation
library(rpart.plot)
rpart.plot(myTree)
```

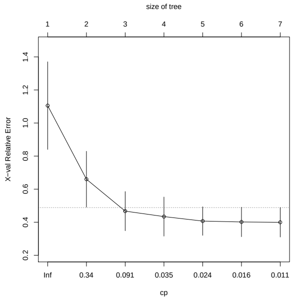


Tree Pruning

```
printcp(myTree)

##
## Regression tree:
## rpart(formula = mpg ~ wt + hp + factor(carb) + factor(am), data = mtcars,
##     control = rpart.control(minsplit = 5))
##
## Variables actually used in tree construction:
## [1] factor(carb) hp           wt
##
## Root node error: 1126/32 = 35.189
##
## n= 32
##
##         CP nsplit rel error  xerror     xstd
## 1 0.652661      0  1.000000 1.10529 0.265228
## 2 0.178618      1  0.347339 0.66000 0.169220
## 3 0.046269      2  0.168721 0.46702 0.118747
## 4 0.026109      3  0.122453 0.43399 0.118450
## 5 0.022593      4  0.096343 0.40681 0.086571
## 6 0.011989      5  0.073751 0.40145 0.089376
## 7 0.010000      6  0.061762 0.39935 0.089194

plotcp(myTree)
```

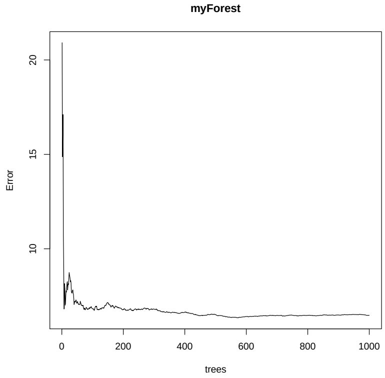


Random Forest

- Consists of many decision trees
  - Randomly selected variables will be used in each split
  - Usually no need to prune them (all trees are allowed to grow big)
- M trees in a random forest produces M predictions
  - Final prediction is calculated as mean value for regression problem
  - Classification problem will use most the common label (majority voting)

Training a Random Forest

```r
library(randomForest)
library(dplyr)
# Build a random forest with 1000 trees
# Each tree has 2 randomly selected variables
# You can change the parameters
myForest <- randomForest(mpg ~ wt + hp + carb + am,
                         ntree = 1000,
                         mtry = 2,
                         data = mtcars %>% mutate(carb = factor(carb),
                                                  am = factor(am)))
# Plot the error as the forest expands
plot(myForest)
```


myForest
Error
trees