ASS x Cainz R Workshop 2021 - Day 1

R is a programming language and environment for statistical computing and graphing. R provides a wide variety of statistical and graphical techniques, and is highly extensible with many open-sourced packages. It is widely seen as go-to language for statistical computation or data analysis along with other software packages such as `SAS`, `SPSS`, or `Stata` but importantly, `R` is **free**.

In the business world many data-based careers expect graduates to have experience in `R`.

This first workshop is meant to serve as an introduction to `R` with much of the content borrowed from Communicate Data with R by Benjamin Avanzi. A Professor in Actuarial Studies here at the University.

Section 1, Introduction and Basic operations

Intended Learning Outcomes (ILOs):

- Perform basic calculations with `R`
- Define variables in `R`
- Perform relational and logical operations in `R`
- Recognise and create data of different types in `R`

There is no Prerequisite Knowledge for this section.

Basic calculations

```r
# R can replace all the functionality of a (scientific) calculator
1 + 1
```

```
## [1] 2
```

```r
10^2
```

```
## [1] 100
```

```r
sqrt(100)
```

```
## [1] 10
```

```r
125^(1/3)
```

```
## [1] 5
```

```r
tan(pi/4)
```

```
## [1] 1
```

```r
log(1)
```

```
## [1] 0
```

Assigning variables

```r
x <- 1
x
```

```
## [1] 1
```

```r
x = 2
x
```

```
## [1] 2
```

```r
3 -> y
y
```

```
## [1] 3
```

```r
#2 = y #this doesn't work
(z = 3)
```

```
## [1] 3
```

Data Types

Data type	Type in R	Display
real number (integer or not)	numeric (double)	3.14159
integer	numeric (integer)	3
logical (T/F)	logical	TRUE or FALSE
missing	logical	NA
text (string)	character	"text"

There are many other data types, but these are the main ones you'll encounter.

```r
rm(list = ls())
#By default, numbers are 'double' even if they are whole numbers
(a <- 5)
```

```
## [1] 5
```

```r
typeof(a)
```

```
## [1] "double"
```

```r
c(is.numeric(a), is.integer(a))
```

```
## [1]  TRUE FALSE
```

```r
#Integers must be explicitly defined
b <- as.integer(a)
typeof(b)
```

```
## [1] "integer"
```

```r
#There isn't a huge difference between doubles and integers,
#generally there is no need to explicitly change them.
#The difference is the amount of memory allocated to the computer (32bit vs 64bit)

#Missing data
x <- c(1,2,3,NA)
is.na(x)
```

```
## [1] FALSE FALSE FALSE  TRUE
```

```r
sum(x)
```

```
## [1] NA
```

```r
sum(x, na.rm = TRUE)
```

```
## [1] 6
```

```r
#
string <- "Hello world"
is.character(string)
```

```
## [1] TRUE
```

Logical Operators

All very common across programming languages:

- `==` is exactly equal to
- `<` greater than, `<=` greater or equal to
- `>` less than, `>=` less or equal to
- `!=` not equal to

```r
c(4 == 4, 2 > 1, 4 < 0, 5 != 5, 3 >= 5)
```

```
## [1]  TRUE  TRUE FALSE FALSE FALSE
```

The AND operators: `&` and `&&` The OR operators: `|` and `||`

```r
#element-wise evaluation
c(TRUE, TRUE) & c(FALSE, TRUE)
```

```
## [1] FALSE  TRUE
```

```r
#short-circuit evaluation
#only evaluates the first element of each vector and returns one boolean value
c(TRUE, TRUE) && c(FALSE, TRUE)
```

```
## [1] FALSE
```

```r
#element-wise evaluation
c(TRUE, TRUE) | c(FALSE, TRUE)
```

```
## [1] TRUE TRUE
```

```r
c(TRUE, TRUE) || c(FALSE, TRUE)
```

```
## [1] TRUE
```

Section 2, Data structures and manipulations

Intended Learning Outcomes (ILOs):

- Recognise and performing basic operations on the `vector`, `matrix`, and `dataframe` data structures
- Import and export data using `read.csv()` and `write.csv()`
- Control execution flow using `if` statements, `for`, and `while` loops
- Defining and calling functions

Prerequisite knowledge is the content from the previous section

Vectors

```r
#the basic data structure, most often used to encode a sequence of data points
#or a list of numbers
#getwd()
#setwd()
```

```r
#read.csv automatically reads as a dataframe so we need to retreive the first
#column for the vector
x <- read.csv("vector.csv")[,1]

length(x)
sort(x)
rev(x)
rank(x)

#indexing - one dimension
x[1]
x[1:10]

#How do I take the last element
x[-1] # This removes the first element of the vector ?
x[length(x)]
```

Matrices

This is a generalisation of a vector in two dimensions

```r
(X <- matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE))

#indexing - two dimensions
#index by row
X[1,]
X[1,1:2]
#index by column
X[,1]
#retreive a single element
X[1,1]

(Y <- matrix(1:4, nrow = 2, ncol = 2))

#Transpose
t(X)
t(X) == Y

#Invert
solve(X)

#Diag
diag(X)

# Operations
X + Y
X*Y
X/Y
X %*% Y
```

Comments:

- If we want to do math using data, we store using matrices
- Do not confuse 'data structure' (vector, matrix, array,...) with 'data type' (numerics, characters, logicals,...).
- A 'data type' refers to the type of information (numerical, string, logical, etc.) while a 'data structure'

refers to how we store the information (in a vector, in a matrix, etc.)

The `Data frame` data structure is also very important which will be touched on in Section 3.

Importing and exporting data

```
#Make sure your working directory is correct
getwd()
```

```
## [1] "C:/Users/antho/Desktop/CAINZ/Documents/2021/Sem 1/R Workshop"
```

```
setwd("C:/Users/antho/Desktop/CAINZ/Documents/2021/Sem 1/R Workshop")
heights_data <- read.csv("heights.csv")
#we can do all sorts of things with it

mean(heights_data$heights)
```

```
## [1] 175.06
```

```
sd(heights_data$heights)
```

```
## [1] 8.650124
```

```
#write.csv(heights_data, file="heights_copy.csv")
write.csv(heights_data, file="heights_copy.csv", row.names = FALSE)
```

Control flows

if statements - will execute segments of code if some condition is satisfied *for loops* - *while loops* - will execute segments of code if some condition is satisfied

```
#if else statement
x <- 3
y <- 2
if (x<=y) {
  print("x smaller or equal to than y")
} else if (x > y) {
  print("x larger than y")
} else {
  print("x is equal to y")
}
```

```
## [1] "x larger than y"
```

```
#===================================
#read.csv automatically stores the data as a dataframe
#In our loop we want x as a vector so we retreive the first column as a vector
x <- read.csv("vector.csv")[,1]

summation <- 0
for (i in 1:length(x)) {
  summation <- summation + x[i]
}
summation
```

```
## [1] 5256
```

```
#this emulates the sum() function
sum(x)
```

```
## [1] 5256
```

```r
#====================================
#while loop
summation <- 0
i <- 1
while (i <= length(x)) {
  summation <- summation + x[i]
  i <- i + 1
}
summation
```

```
## [1] 5256
```

Defining functions

```r
say_hello <- function(name) {
  print(paste("Hello my name is", name))
}

#say_hello(Anthony) #won't work
say_hello("Anthony")
```

```
## [1] "Hello my name is Anthony"
```

```r
my.dnorm <- function(x, mu, var) {
  return((1/sqrt(2*pi*var)) * exp((-0.5/var)*(x - mu)^2))
}

#density function of normal dist.
my.dnorm(2,0,1)
```

```
## [1] 0.05399097
```

```r
dnorm(2,0,1)
```

```
## [1] 0.05399097
```

Functions can be very, very complicated and incorporate many types of control flows.

Section 3, Working with Data

After completing these section, participants should be able to:

- Retrieve summary statistics of a dataset
- Work with and manipulate a Data frame in Base R, including:
  - Indexing and sampling using conditional formulas
  - Indexing using the which() function, and understanding its differences with regular index-matching
  - Handling missing values

Prerequisite Knowledge for this section:

- Basic extraction of data using indices from matrices and vectors
- The concept of logical and relational operators

Summary Statistics

```r
is.data.frame(iris)
```

```
## [1] TRUE
```

```r
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
tail(iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```r
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
##                 
```

Data frames

```r
#Slicing via logicals
apple <- c("a", "p", "p", "l", "e")
bool <- c(T, F, T, F, T)
apple[bool]

#every second row
iris[c(T,F),]

iris$Petal.Width > 1.5
iris[iris$Petal.Width > 1.5,]
# Petal.Width can be used as a pretty good identifier for the virginica species

iris[iris$Species == "virginica",]

#Slicing via indicies
iris[,c(1,3,5)]
```

```r
which(iris$Petal.Width > 1.5)
iris[which(iris$Petal.Width > 1.5), ]
```

```r
set.seed(1)
nrow(iris)
```

```
## [1] 150
```

```r
jumble.indicies <- sample(nrow(iris))
jumbled.iris <- iris[jumble.indicies,]

#What is the difference between these two methods
which(jumbled.iris$Petal.Width > 1.5)
```

```
##  [1]   2   8  14  15  17  18  21  23  26  29  32  34  36  38  39  43  48  49  51
## [20]  57  58  62  65  66  68  70  73  77  81  87  88  92  95  96 100 103 104 105
## [39] 107 110 114 115 118 119 126 130 133 134 137 145 147 149
```

```r
#returns the rows in the corresponding to the condition in the new dataset
as.numeric(rownames(jumbled.iris[jumbled.iris$Petal.Width > 1.5,]))
```

```
##  [1] 129 106 105 110 143 126  84 142 111 124 149 121 119 146 127 147  78 102 115
## [20] 118 132 130 104 123 131 144 140 113 148 136 112 114 145  86 150  71 101 108
## [39] 125 138 141 133 122 116 107 103 109 137 139 128 117  57
```

```r
#returns the rows in the corresponding to the condition in the original dataset
```

Handling missing values

```r
set.seed(1)
iris.missing <- iris
#creating data with missing values
missing_matrix <- matrix(as.logical(rbinom(5*150, 1, 0.05)), nrow = 150)
iris.missing[missing_matrix] <- NA

# Easiest way to handle is simply to remove the rows with missing data
iris.cleaned1 <- na.omit(iris.missing)

#Imputation - filling the missing values with the mean of the column
#the natural way to do this with this dataset is to replace with the mean
#of the species
```

Section 4, Mathematics and Statistics

After completing these section, participants should be able to:

- Learn four common functions for probability distributions.
- Draw basic plots in `R`.
- Construct a simple linear model.
- Construct confidence intervals and prediction intervals.

Probability distributions

`rnorm(n)`: returns a vector of random observations from the distribution

```r
set.seed(1)
rnorm(10,mean=5,sd=10)
```

```
##  [1] -1.264538  6.836433 -3.356286 20.952808  8.295078 -3.204684  9.874291
##  [8] 12.383247 10.757814  1.946116
```

`dnorm(x)`: finds the pdf/pmf evaluated at value x

```r
dnorm(5, mean=5,sd=10)
```

```
## [1] 0.03989423
```

`pnorm(q)`: finds the cdf of the distribution for quantile q

```r
pnorm(7, mean=5,sd=10)
```

```
## [1] 0.5792597
```

`qnorm(p)`: finds the quantile of the distribution that returns the cdf p

```r
qnorm(0.95, mean=5,sd=10)
```

```
## [1] 21.44854
```

Common probability distributions:

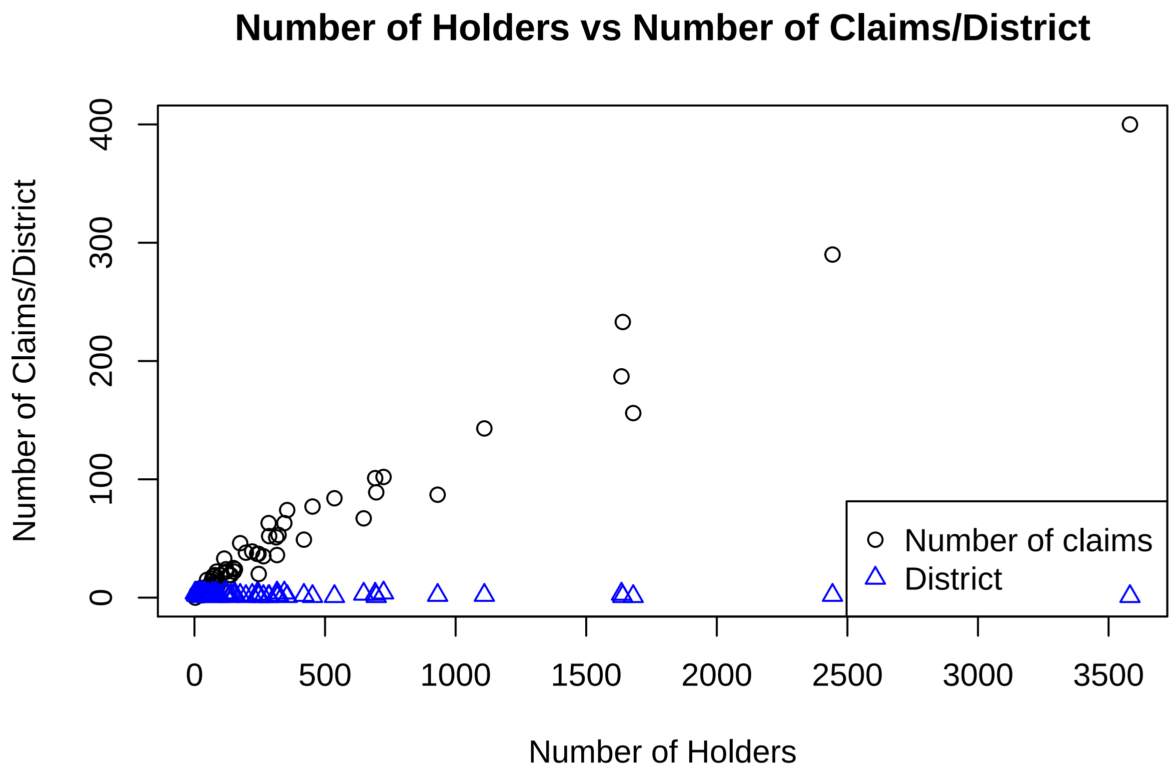
- `_unif()` - Continuous uniform
- `_norm()` - Normal distribution
- `_binom()` - Binomial distribution
- `_geom()` - Geometric distribution
- `_pois()` - Poisson distribution
- `_exp()` - Exponential distribution
- `_gamma()` - Gamma distribution
- `_weibull()` - Weibull distribution
- `_lnorm()` - Log-normal distribution
- `_lgamma()` - Log-gamma distribution
- `_pareto()` - Pareto distribution
- `_t()` - t distribution
- `_f()` - F distribution
- `_chisq()` - Chi-square distribution

Data visualisation

```r
Insurance <- read.csv("insurance.csv")
```

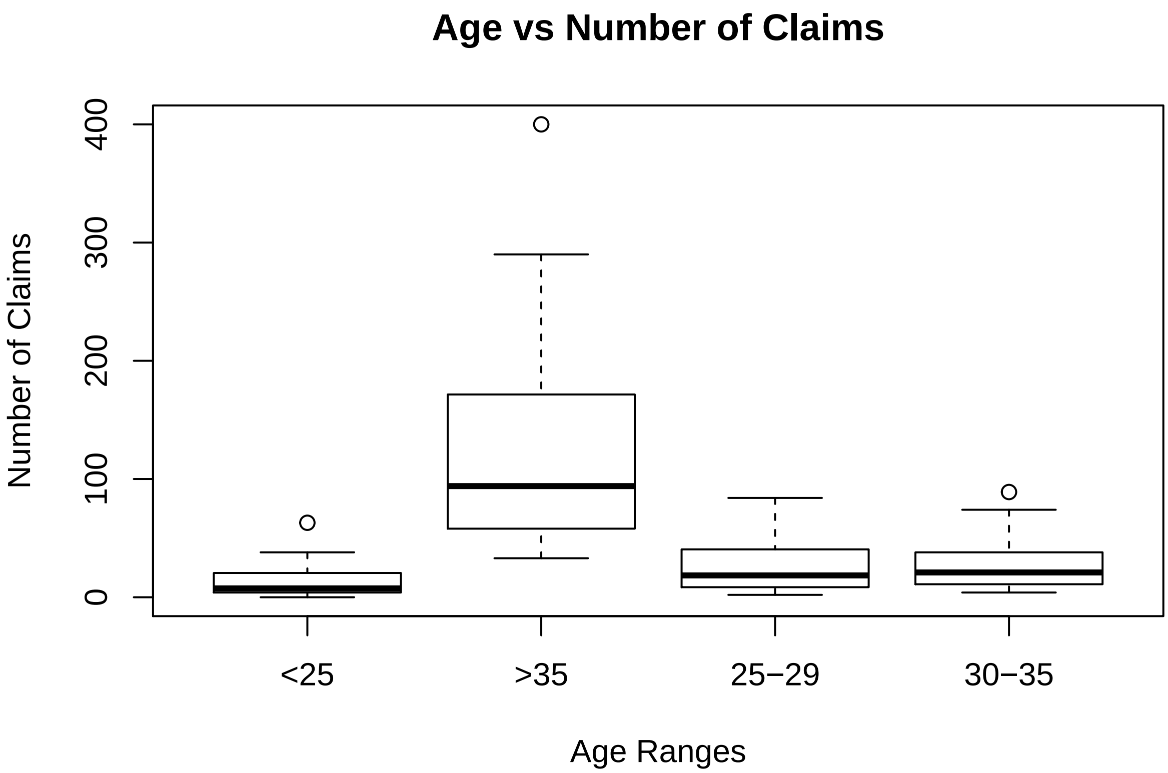
1. **Dotplot**

```r
plot(Insurance$Holders, Insurance$Claims,
     main = "Number of Holders vs Number of Claims/District",pch=1, col="black",
     xlab="Number of Holders", ylab="Number of Claims/District")
points(Insurance$Holders, Insurance$District, col="blue",pch=2)
#points() adds points to the previously created plot.
#It has same basic argument as plot()
legend("bottomright",legend=c("Number of claims", "District"),pch=c(1,2),col=c("black","blue"))
```

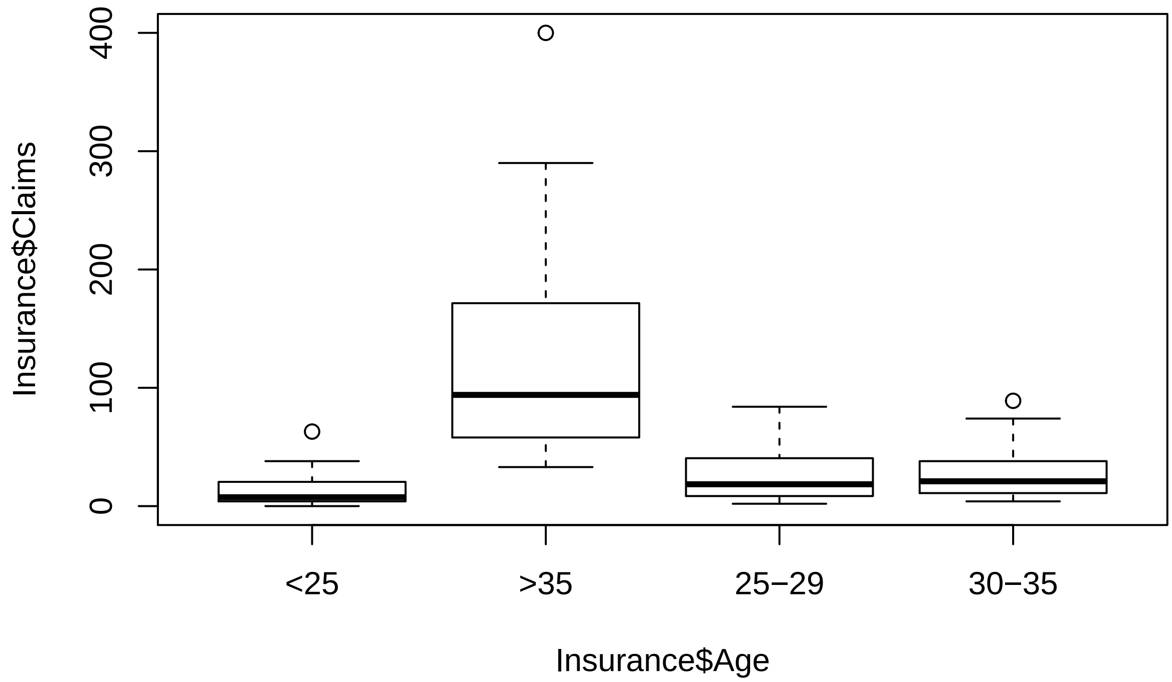


2. **Boxplot**

```
plot(Insurance$Age, Insurance$Claims, main = "Age vs Number of Claims",
     xlab= "Age Ranges", ylab= "Number of Claims")
```



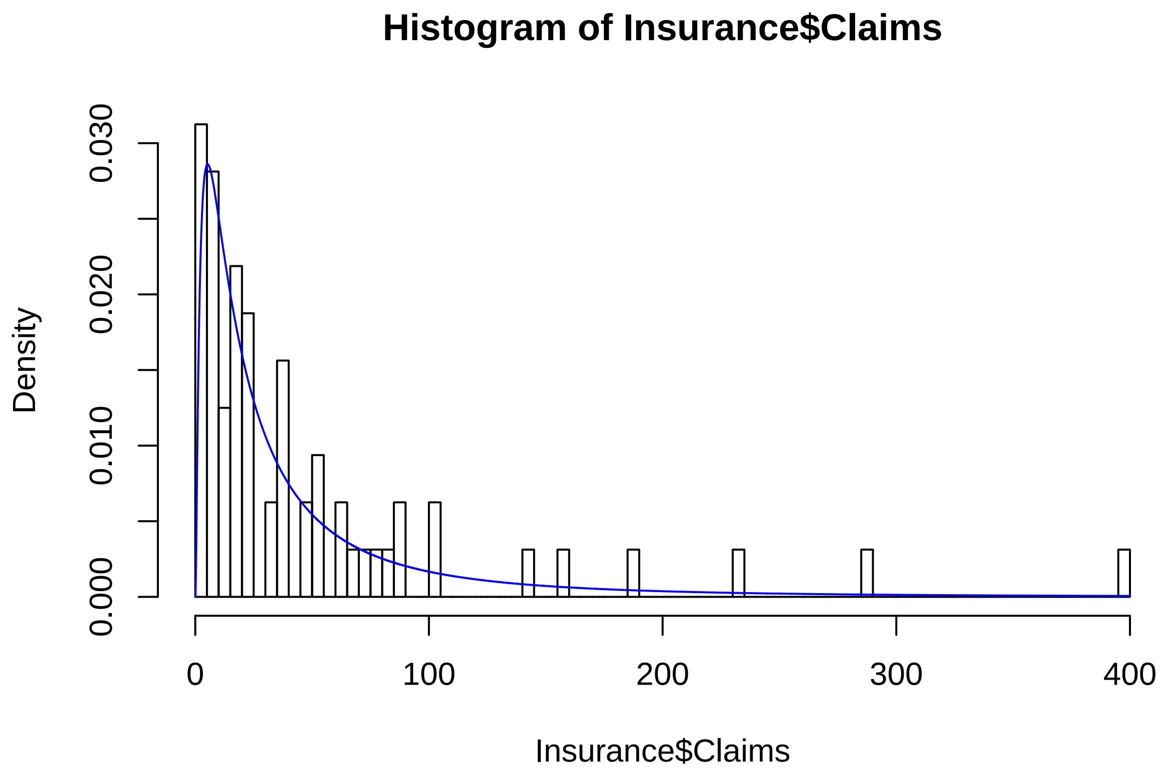

```
boxplot(Insurance$Claims~Insurance$Age)
```



Plot defaults to boxplot when x axis is factor and y axis is numeric. Alternatively, you can use `boxplot()`

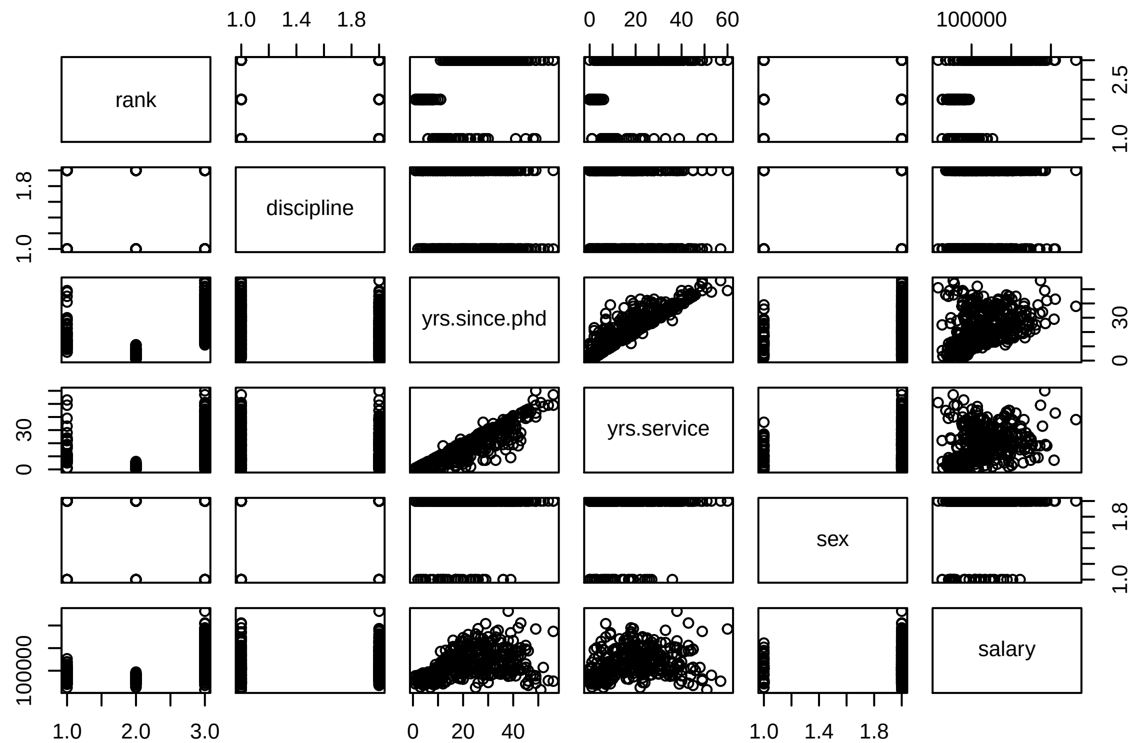
3. **Histogram**

```
#find the best estimation
hist(Insurance$Claims, freq = F,breaks = 100)
x_axis=seq(0,400, 0.01)
#these parameters come from the maximum likelihood estimators (MLE)
y_lnorm=dlnorm(x_axis,3.185221,1.231108)
lines(x_axis,y_lnorm, col="blue",lwd =1)
legend(250,0.2, legend="dlnorm", col="blue",lty=1, cex=0.8)
```



Simple Linear Regression

```r
pairs(salaries)
```



From the scatterplot, we can observe that variables **yrs.since.phd** and **salary** are highly positive correlated

```
lm.fit=lm(salary~yrs.since.phd, data=salaries)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = salary ~ yrs.since.phd, data = salaries)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -84171 -19432  -2858  16086 102383
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    91718.7     2765.8  33.162   <2e-16 ***
## yrs.since.phd    985.3      107.4   9.177   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27530 on 395 degrees of freedom
## Multiple R-squared:  0.1758, Adjusted R-squared:  0.1737
## F-statistic: 84.23 on 1 and 395 DF,  p-value: < 2.2e-16
```

Confidence and prediction intervals

```
predict(lm.fit,data.frame(yrs.since.phd=(c(2,15,40))),
        interval="confidence",level=0.99)
```

```
##         fit       lwr      upr
## 1  93689.37  87006.26 100372.5
## 2 106498.82 102384.78 110612.9
## 3 131132.37 125053.98 137210.8
```

```r
predict(lm.fit,data.frame(yrs.since.phd=(c(2,15,40))),
        interval="confidence")
```

```
##         fit       lwr       upr
## 1  93689.37  88613.16  98765.58
## 2 106498.82 103373.97 109623.66
## 3 131132.37 126515.49 135749.26
```

```r
predict(lm.fit,data.frame(yrs.since.phd=(c(2,15,40))),
        interval="prediction")
```

```
##         fit      lwr      upr
## 1  93689.37 39321.18 148057.6
## 2 106498.82 52278.00 160719.6
## 3 131132.37 76805.14 185459.6
```