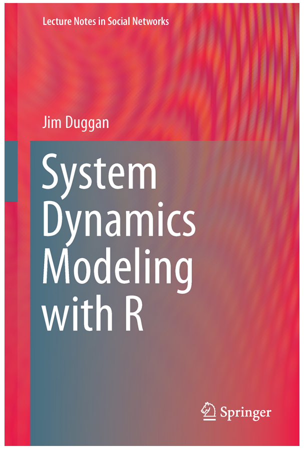
Session 1: Introduction to R



Dr. Jim Duggan,

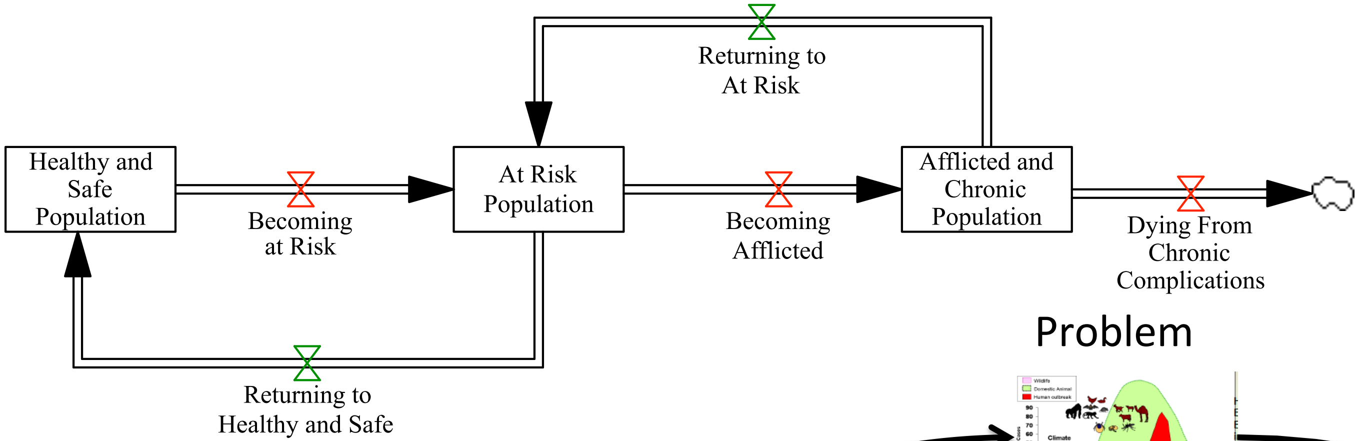
School of Engineering & Informatics

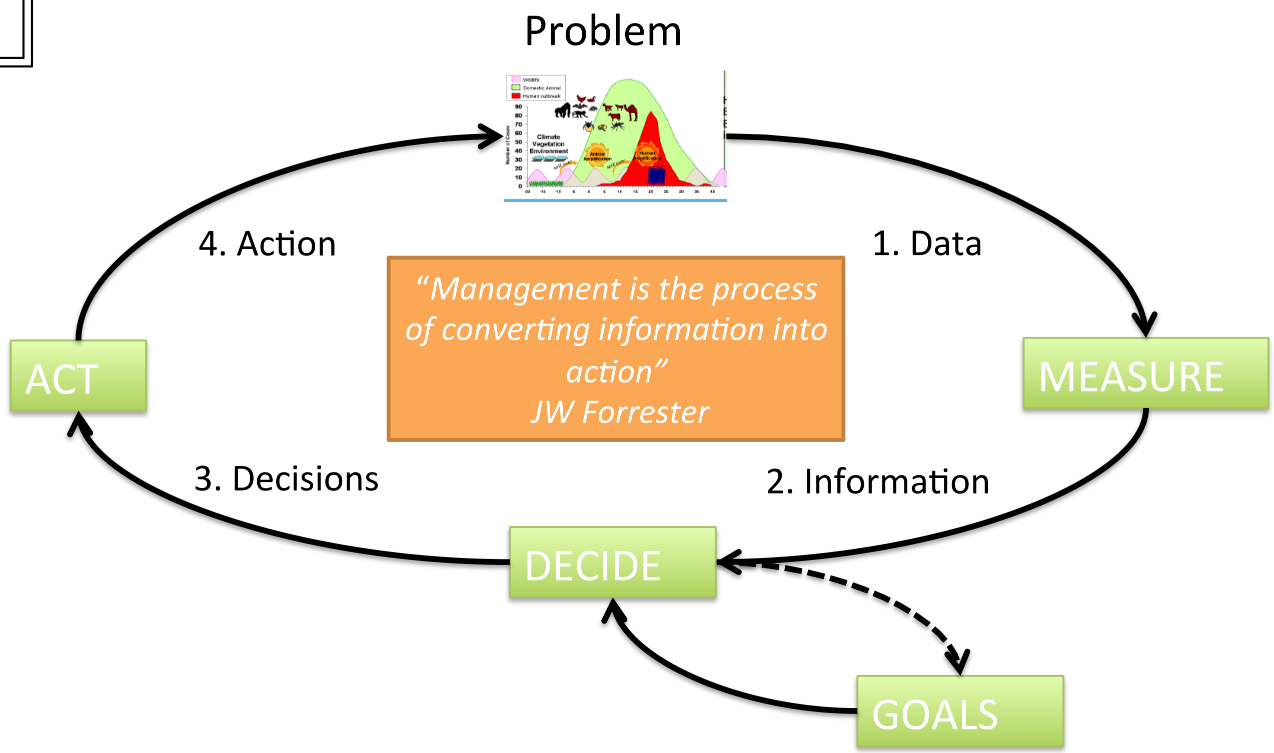
National University of Ireland Galway.

https://github.com/JimDuggan/SDMR

https://twitter.com/_jimduggan

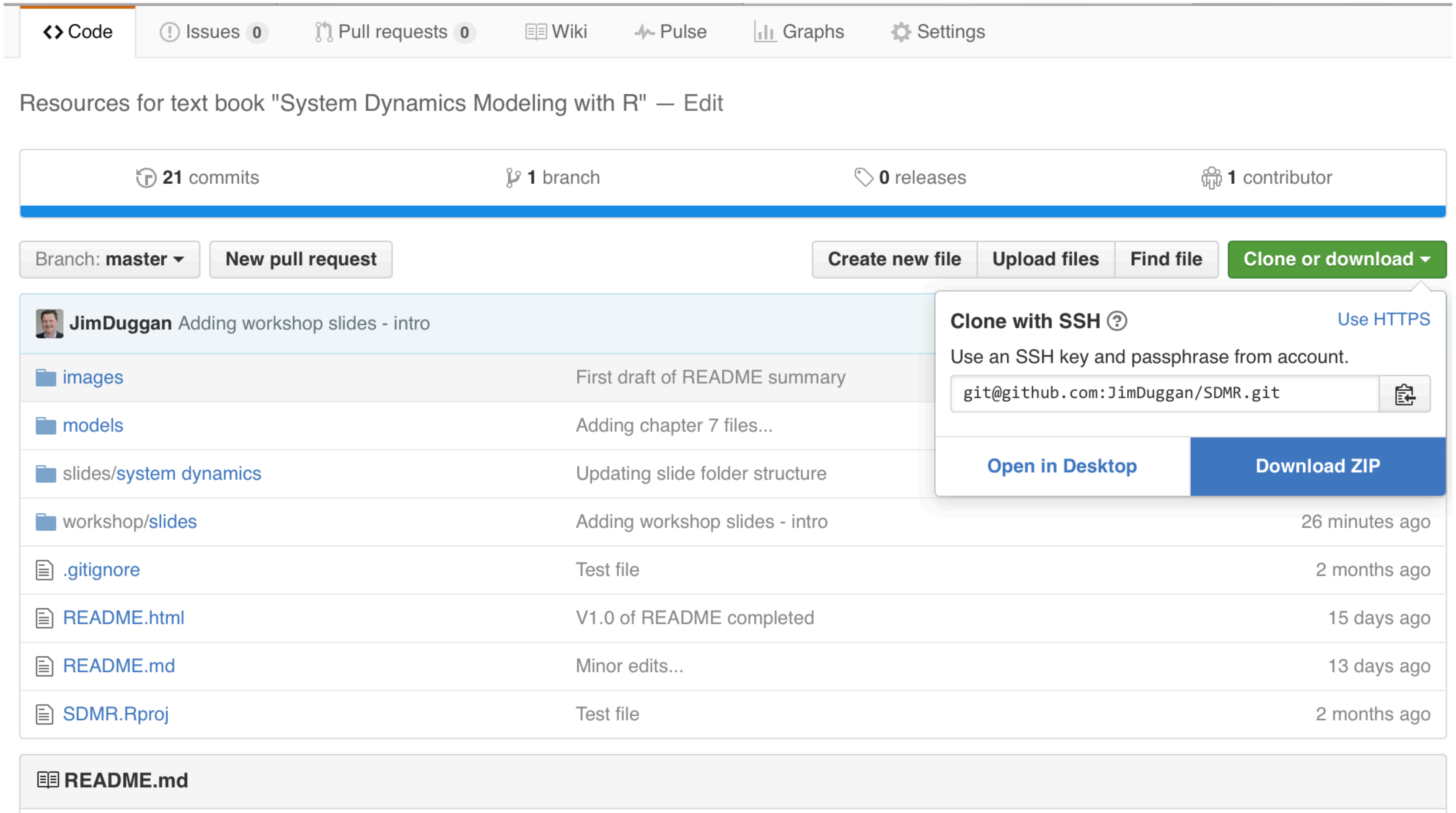
Research Interests: System Dynamics – Data Science – Population Health





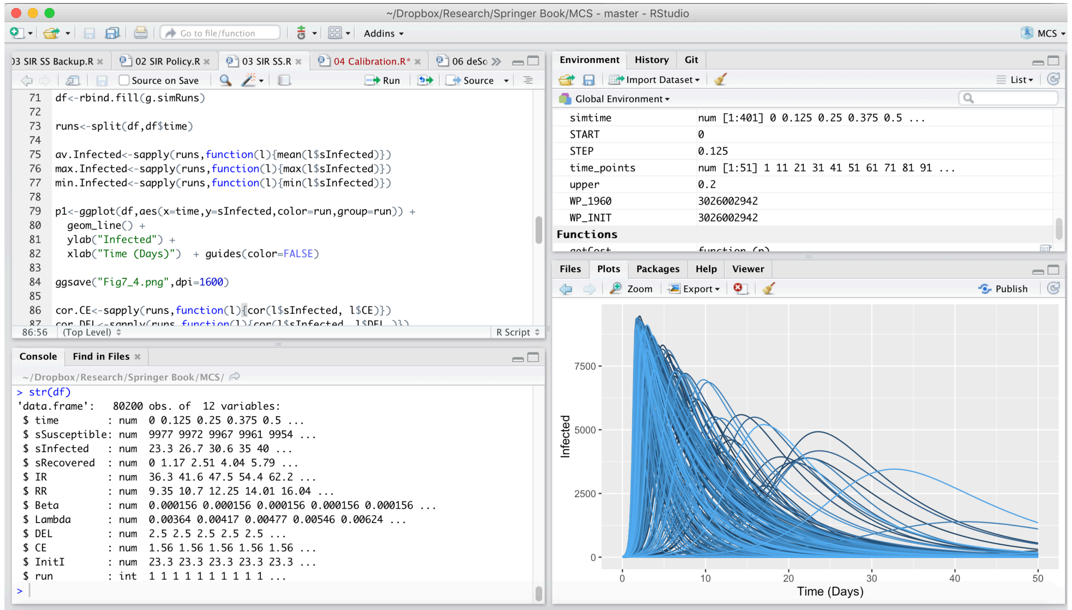
First step… download materials…

https://github.com/JimDuggan/SDMR



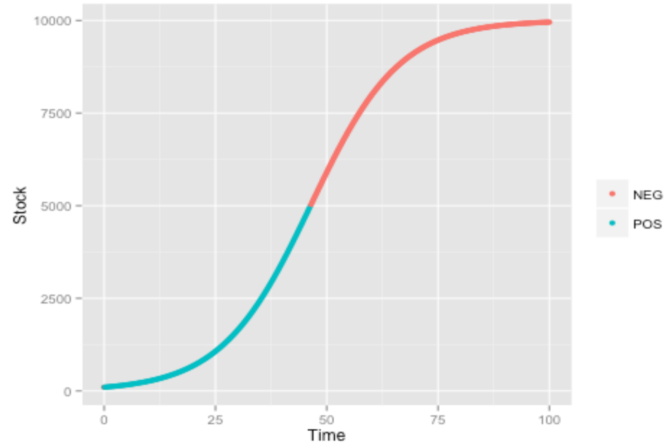
Outline

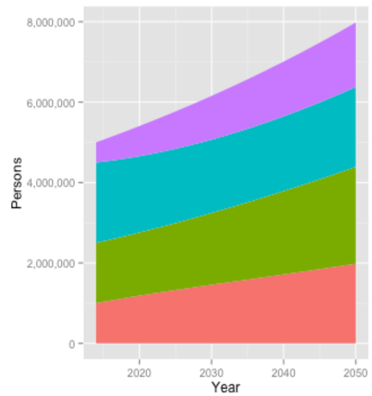
- Introduction to R
- Storing data
  - Vectors
  - Lists
  - Data Frames
- Functions
- Apply Function
- Visualisation

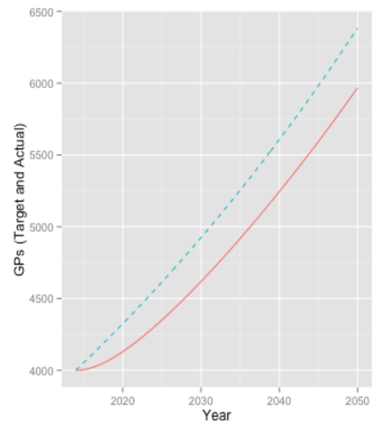


What is R?

- It is a dialect of the S language, developed at Bell Laboratories
- Open source, functional & object-oriented language
- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).

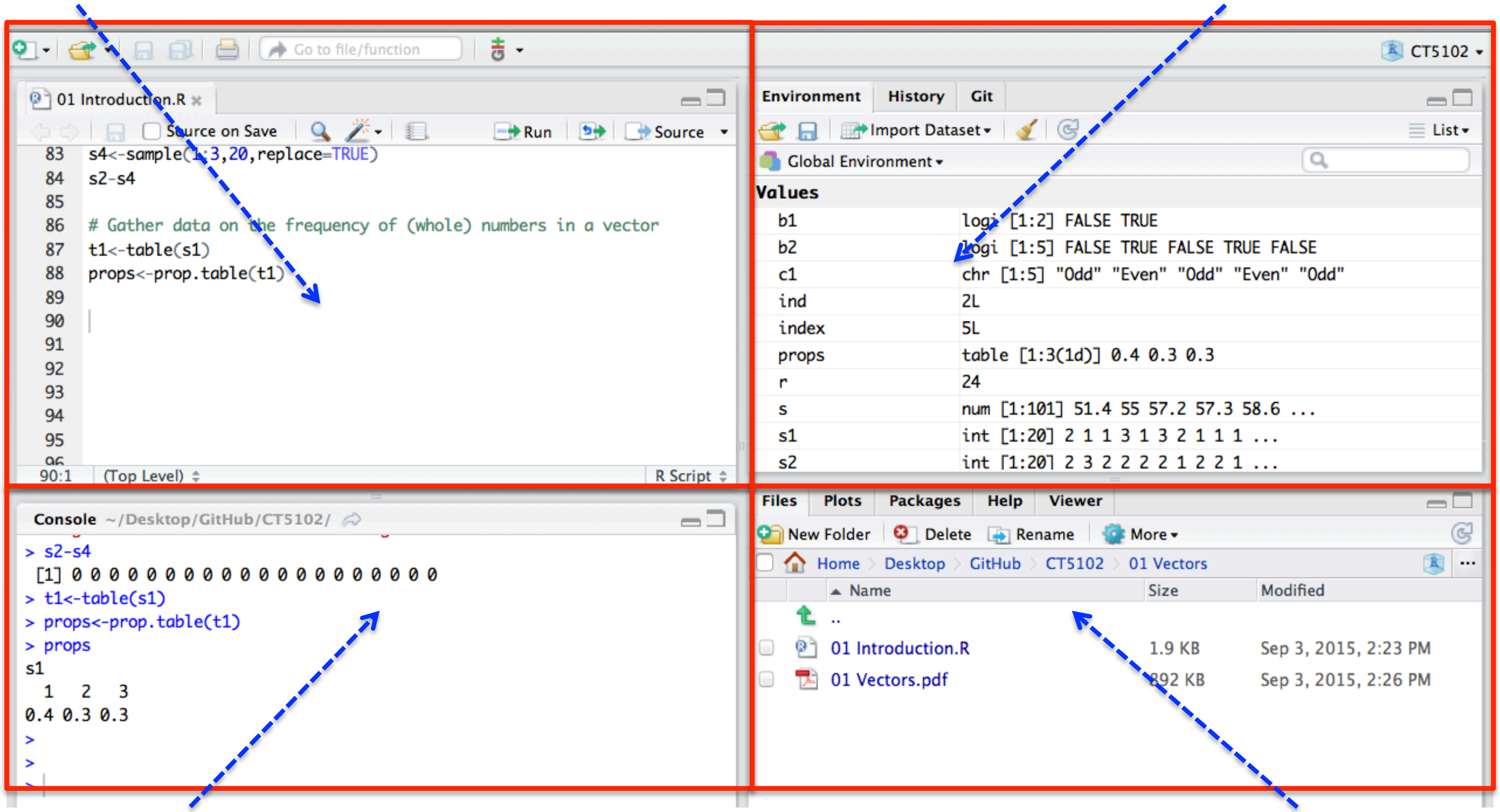






R Studio Workbench

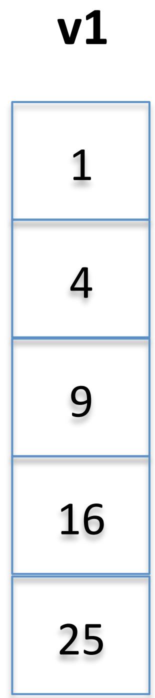
R Code **Environment/State**



Interactive console **File System**

(1) Data Structures - Vector

- The fundamental data type in R is the vector,
- A variable that contains a sequence of elements that have the same data type (Matloff 2009).
- Create using $c(e_1, \ldots, e_n)$
- Assignment statement <-




```
> v1<-c(1,4,9,16,25)
> v1
[1]  1  4  9 16 25
```

Four main types of atomic vectors

- logical
- integer
- double
- character

```
> v5<-c(v1,v2)
> v5
[1]  1  0 10 20

> typeof(v5)
[1] "integer"
```

```
> v1<-c(T,FALSE)
> v2<-c(10L,20L)
> v3<-c(3.5,12.2)
> v4<-c("system","dynamics")
>
> typeof(v1)
[1] "logical"
> typeof(v2)
[1] "integer"
> typeof(v3)
[1] "double"
> typeof(v4)
[1] "character")
```

Index

- The concept of an index is powerful in R, as it allows access to individual data elements of a vector, using the square brackets notation.
- In R, unlike programming languages such as C and Java, the index for a vector starts at 1.

```
> v1
[1]  1  4  9 16 25

> v1[1]
[1] 1

> v1[2]
[1] 4

> v1[5]
[1] 25
```

Creating sequences

- The colon operator (:) generates regular sequences within a specified range.

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10

> v2<-3:13
> v2
 [1]  3  4  5  6  7  8  9 10 11 12 13
```

Sequences as indices

- Sequences can be used an indices for vectors, with the minus sign used for exclusion

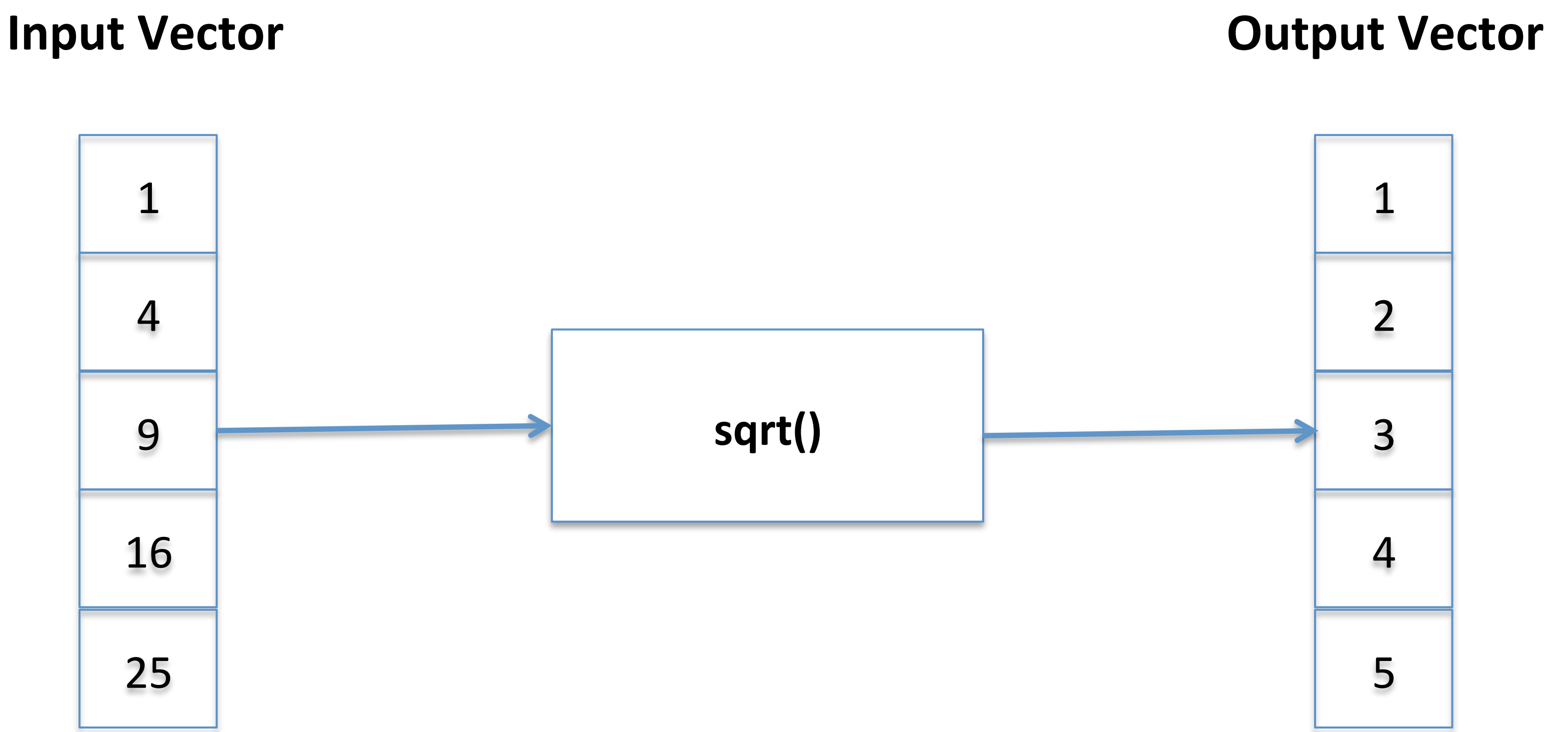
```
> v1
[1]  1  4  9 16 25

> v1[1:2]
[1] 1 4

> v1[-1]
[1]  4  9 16 25

> v1[-(1:3)]
[1] 16 25
```

Vectorization

- A powerful feature of R is that it supports *vectorization*
- Functions can operate on every element of a vector, and return the results of each individual operation in a new vector.

```
> v1
[1] 1 2 3 4 5
> r<-sqrt(v1)
> r
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Key Idea



Conditional Expressions on Vectors

- Conditional expressions can be applied to vectors, and this operation returns a boolean vector

```
> v1
[1]  1  4  9 16 25

> v1 %% 2==0
[1] FALSE  TRUE FALSE  TRUE FALSE
```

Operators	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	not x
x | y	x OR y
x & y	x AND y

http://www.statmethods.net/management/operators.html

Boolean vectors used as indices

- A target the vector can be filtered by the TRUE locations in the boolean vector.

```
> v1
[1]  1  4  9 16 25

> b1<-v1 %% 2 == 1

> b1
[1]  TRUE FALSE  TRUE FALSE  TRUE

> v1[b1]
[1]  1  9 25
```

Challenge 1

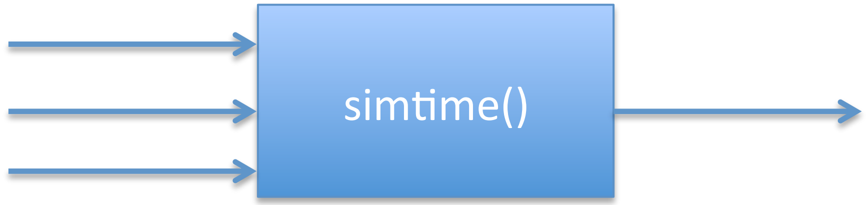
- Create an R vector of squares of 1 to 10
- Find the minimum
- Find the maximum
- Display all those greater than the mean
- Display all those less than the mean
- *Display every second vector element*
- *Find the index location of the maximum value*

(2) Functions

- A function is a group of instructions that:
  - takes input,
  - uses the input to compute other value, and
  - returns a result.
- Functions are declared:
  - using the **function** reserved word
  - are objects

function(*arguments*){
    expression
}

Example




```
simtime<-function(start, finish, DT=1){
  seq(start,finish,by = DT)
}
```

```
> simtime(1,10,0.25)
 [1]  1.00  1.25  1.50  1.75  2.00  2.25  2.50  2.75  3.00  3.25  3.50  3.75  4.00
[14]  4.25  4.50  4.75  5.00  5.25  5.50  5.75  6.00  6.25  6.50  6.75  7.00  7.25
[27]  7.50  7.75  8.00  8.25  8.50  8.75  9.00  9.25  9.50  9.75 10.00
```

Challenge 2

- Write a function that takes in a vector and returns the number of odd numbers
- Write a function that takes in a vector and returns a vector of even numbers
- *Write a function that returns the unique values in a vector (hint: use the R function* **duplicated***)*

(3) Lists

- A list is a vector that can contain different types
- Flexible for returning function results
- Lists can be recursive (a list can contain a list)

```
v1<-c(1,2,3)
v2<-c("One","Two","Three")

l<-list(Numbers=v1,Names=v2)
```

```
> l
$Numbers
[1] 1 2 3

$Names
[1] "One"   "Two"   "Three"
```

Indexing: [], [[]], $

```
> str(l)
List of 2
 $ Numbers: num [1:3] 1 2 3
 $ Names  : chr [1:3] "One" "Two" "Three"
```

```
> l[1]
$Numbers
[1] 1 2 3

> l[[1]]
[1] 1 2 3
```

```
> l[2]
$Names
[1] "One"   "Two"   "Three"

> l[[2]]
[1] "One"   "Two"   "Three"
```

```
> l$Numbers
[1] 1 2 3
```

```
> l$Names
[1] "One"   "Two"   "Three"
```

(4) Data Frames

- Can be viewed as a set of vectors, organised into a column format
- Each column can have a different data type.
- Ideal for storing simulation output

```
time<-seq(1,7)
v1<-rep(10,length(time))
sm<-cumsum(v1)

df<-data.frame(Time=time,Var1=v1,
               SumVar=sm)
```

```
> df
  Time Var1 SumVar
1    1   10     10
2    2   10     20
3    3   10     30
4    4   10     40
5    5   10     50
6    6   10     60
7    7   10     70
```

Filtering Data Frame using matrix notation [row,col]

```
> df[1,]
  Time Var1 SumVar
1    1   10     10

> df[1:4,]
  Time Var1 SumVar
1    1   10     10
2    2   10     20
3    3   10     30
4    4   10     40
```

```
> df[,1]
[1] 1 2 3 4 5 6 7

> df[,2:3]
  Var1 SumVar
1   10     10
2   10     20
3   10     30
4   10     40
5   10     50
6   10     60
7   10     70
```

Subset data using conditionals

```
> b<-df$SumVar<mean(df$SumVar)
> b
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
> df[b,]
  Time Var1 SumVar
1    1   10     10
2    2   10     20
3    3   10     30
```

```
> df[df$SumVar<mean(df$SumVar),]

  Time Var1 SumVar
1    1   10     10
2    2   10     20
3    3   10     30
```

Adding new columns

```
> df$Category<-ifelse(df$SumVar<mean(df$SumVar),"Lower","Higher")

> df
  Time Var1 SumVar Category
1    1   10     10    Lower
2    2   10     20    Lower
3    3   10     30    Lower
4    4   10     40   Higher
5    5   10     50   Higher
6    6   10     60   Higher
7    7   10     70   Higher
```

subset() function

- Return subsets of vectors, matrices or data frames which meet conditions

```
> s<-subset(df,df$SumVar>mean(df$SumVar))
> s
  Time Var1 SumVar Category
5    5   10     50   Higher
6    6   10     60   Higher
7    7   10     70   Higher
```

Reading from Excel (install gdata)

```
library(gdata)
sim <- read.xls("workshop/R code/01 session/SimData.xlsx",
            stringsAsFactors=FALSE)
```

```
> head(sim)
     Time Age.0.14 Age.15.39 Age.40.64 Age.Over.65
1 2014.00  1000000   1500000   2000000      500000
2 2014.13  1004170   1500830   1997500      505625
3 2014.25  1008320   1501700   1995020      511230
4 2014.38  1012460   1502590   1992550      516816
5 2014.50  1016580   1503520   1990100      522383
6 2014.63  1020690   1504470   1987670      527930
```

Summary Statistics

```
> summary(sim[,-1])
   Age.0.14          Age.15.39          Age.40.64          Age.Over.65
 Min.   :1000000   Min.   :1500000   Min.   :1824300   Min.   : 500000

 1st Qu.:1271170   1st Qu.:1624240   1st Qu.:1835170   1st Qu.: 859393

 Median :1509380   Median :1837660   Median :1867720   Median :1145590

 Mean   :1503914   Mean   :1874865   Mean   :1881764   Mean   :1114595

 3rd Qu.:1740660   3rd Qu.:2104030   3rd Qu.:1920900   3rd Qu.:1386170

 Max.   :1980660   Max.   :2408280   Max.   :2000000   Max.   :1604080
```

Challenge 3

- Add the total population for year time as a column to the data frame
- *Subset the data frame so that only the actual yearly values are shown (i.e. the interval between data points is 1 year)*

(5) Apply Function

Usage

apply(X, MARGIN, FUN, ...)

Arguments

- X an array, including a matrix (and data frame). X has named dimnames, it can be a character vector selecting dimension names
- MARGIN a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns..
- FUN the function to be applied

... optional arguments to FUN.

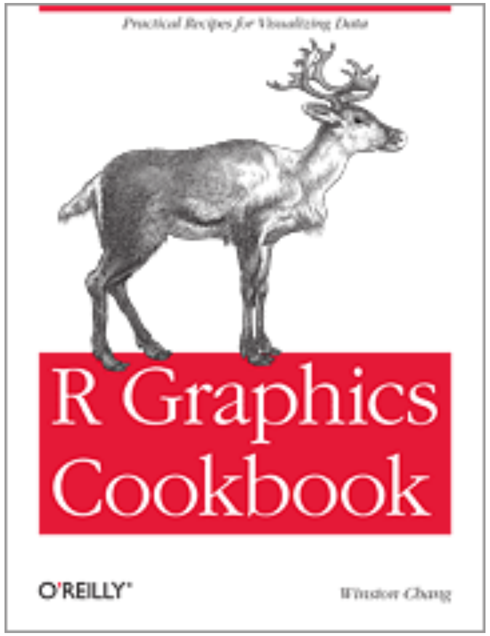
Previous Example

```
library(gdata)
sim <- read.xls("workshop/R code/01 session/SimData.xlsx",
          stringsAsFactors=FALSE)

# Add the total population for year time as a column to the data frame
Sim$Total<-apply(sim[,2:5],MARGIN = 1,sum)
```

```
> head(sim)
   Time Age.0.14 Age.15.39 Age.40.64 Age.Over.65   Total
1 2014.000  1000000   1500000   2000000     500000 5000000
2 2014.125  1004170   1500830   1997500     505625 5008125
3 2014.250  1008320   1501700   1995020     511230 5016270
4 2014.375  1012460   1502590   1992550     516816 5024416
5 2014.500  1016580   1503520   1990100     522383 5032583
6 2014.625  1020690   1504470   1987670     527930 5040760
```

(6) Plotting (library ggplot2)

- Name based on Leland Wilkinson's *grammar of graphics,* which provides a formal, structured perspective on how to describe data graphics
- ggplot package developed by Hadley Wickham
- *Data must be stored in data frames*



http://www.cookbook-r.com/Graphs/

Terminology

- The *data* is what we want to visualise. It consists of variables, which are stored as columns in the data frame. *The data should be in tidy data format.*
- *Geoms* are the geometric objects that are drawn to represent the data, such as bars, lines and points
- *Aesthetic attributes* are visual properties of geoms, such as x and y position, line colour, point shapes etc.
- *Mappings* from data values to aesthetics
- *Scales* that control the mappings from values in the data space to values in the aesthetic space.
- *Guides*, for example, tick marks and labels on an axis.

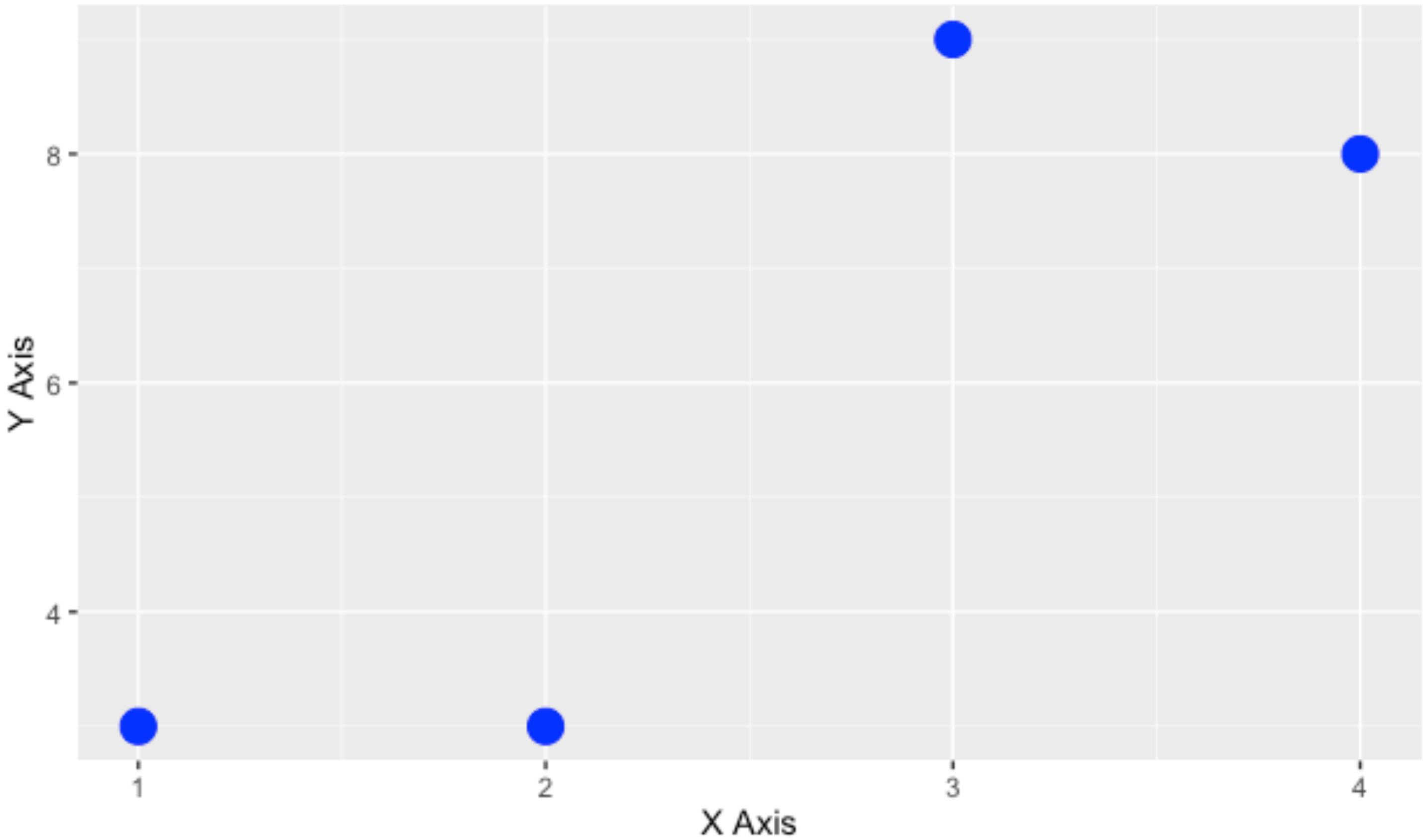
A simple example…

```
df<-data.frame(xval=1:4,
               yval=c(3,3,9,8),
               group=c("A","A","B","B"))
```

```
> df
  xval yval group
1    1    3     A
2    2    3     A
3    3    9     B
4    4    8     B
```

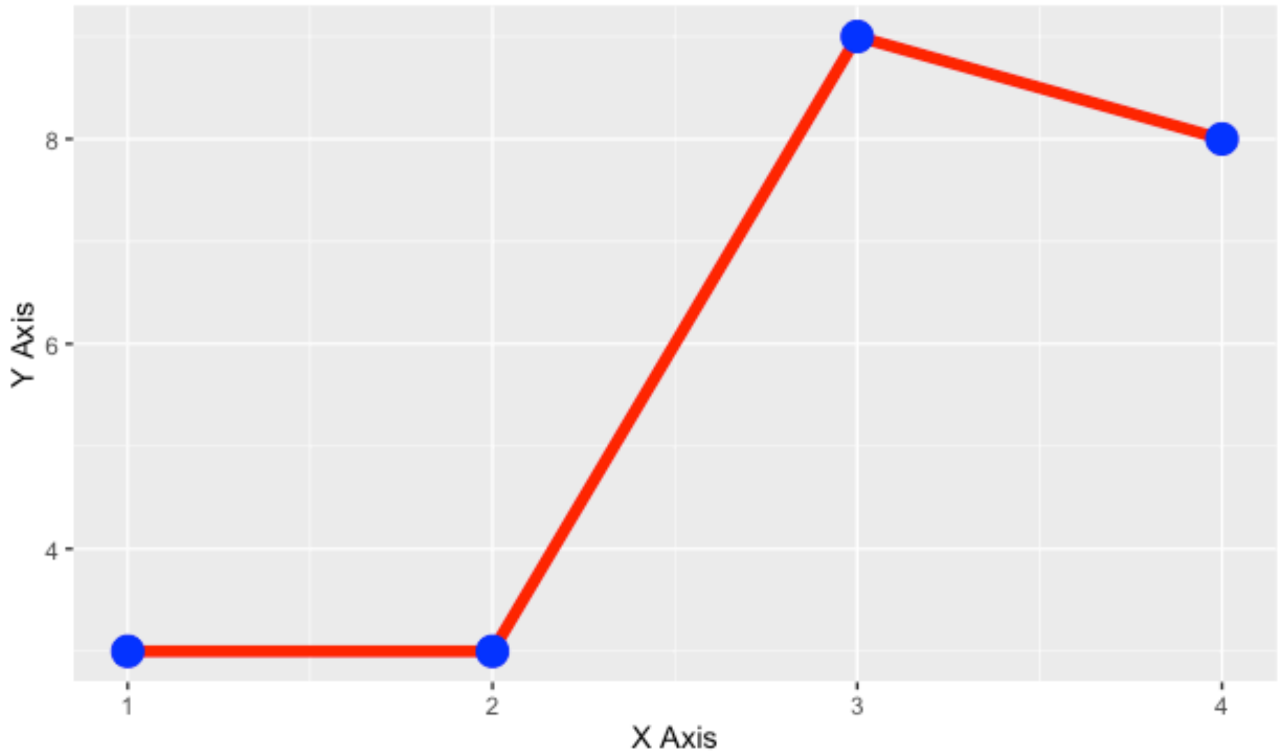
Call to ggplot()

```
ggplot(data=df,aes(x=xval,y=yval))+ geom_point(colour="blue",size=5)+
      xlab("X Axis") + ylab("Y Axis")
```



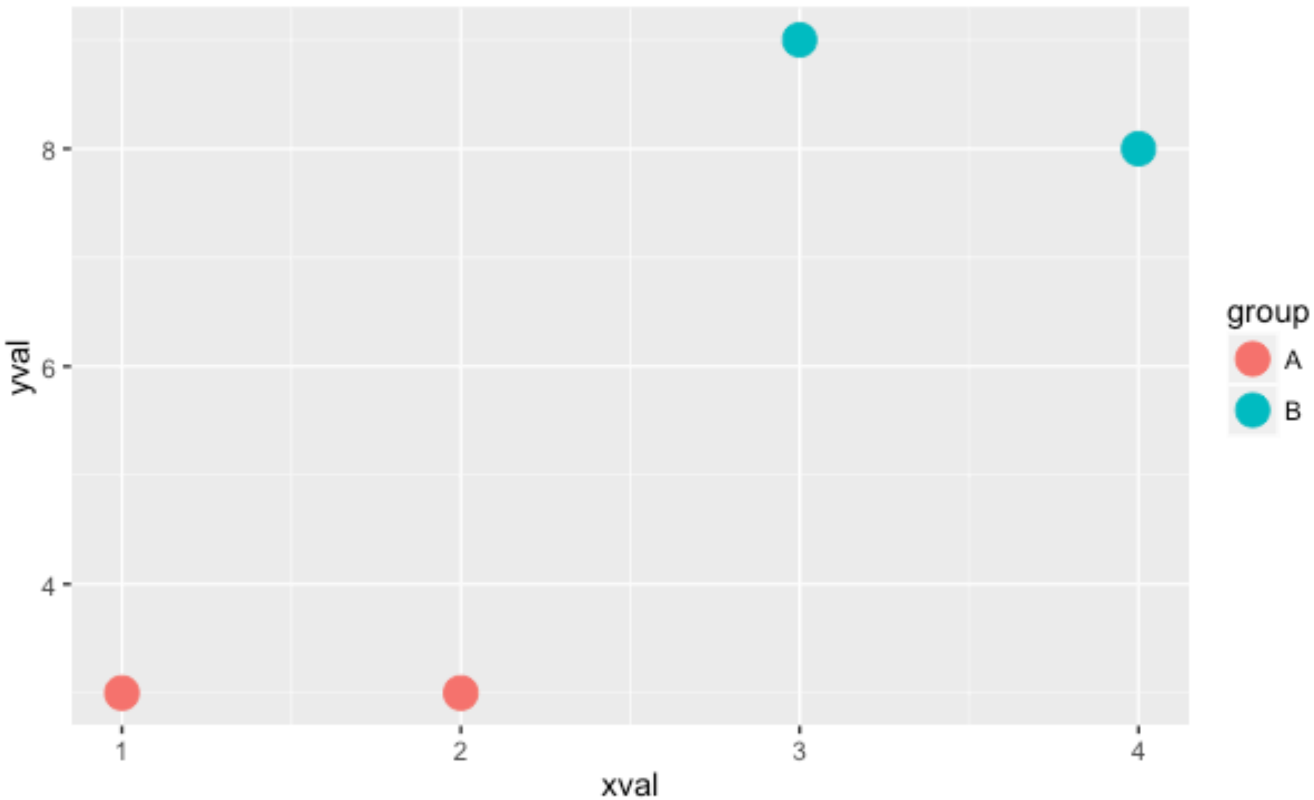
Layer different geoms…

```
ggplot(data=df,aes(x=xval,y=yval))+geom_point(colour="blue",size=5)+
  geom_line(colour="red")+ xlab("X Axis") + ylab("Y Axis")
```



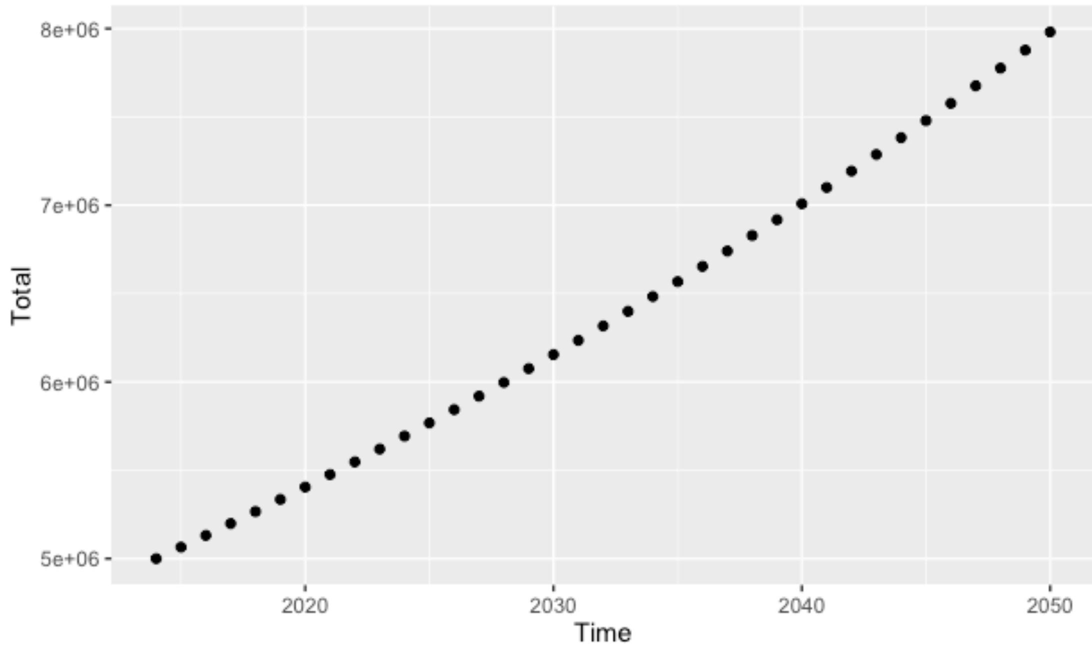
Using categorical information…

```
ggplot(df,aes(x=xval,y=yval))+geom_point(aes(colour=group),size=5)
```



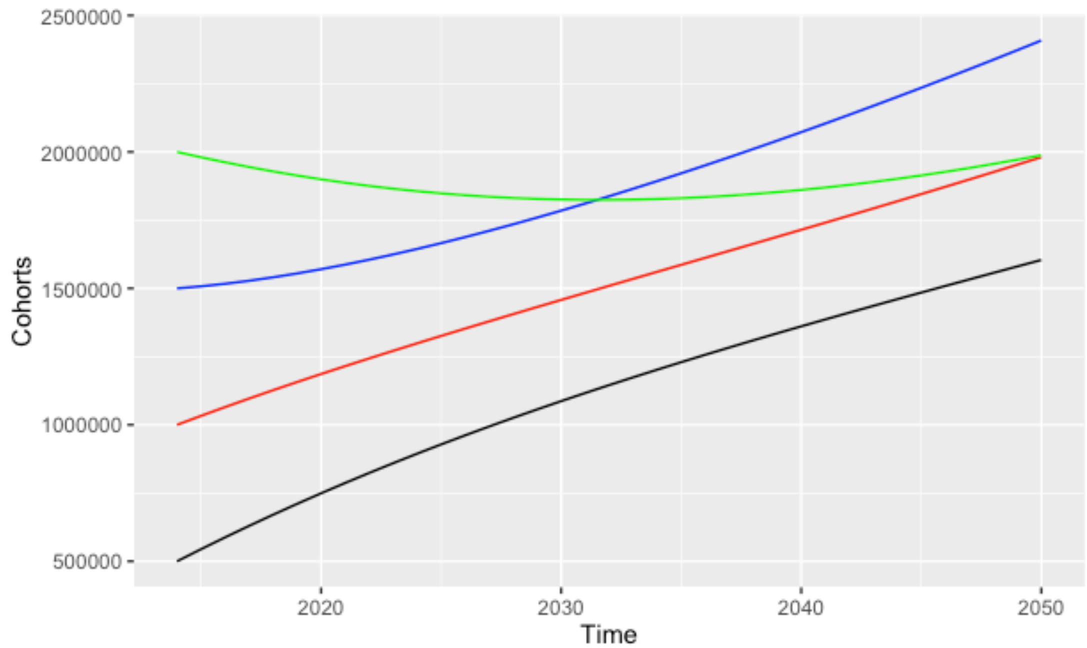
Simulation Time Series

```
sim <- read.xls("workshop/R code/01 session/SimData.xlsx”,
                    stringsAsFactors=FALSE)
sim$Total<-apply(sim[,2:5],MARGIN = 1,sum)
sub<-sim[c( TRUE, rep(FALSE,7)),]
ggplot(data=sub,aes(x=Time,y=Total))+geom_point()
```



Multiple plots (slightly cumbersome)

```
ggplot(data=sub)+geom_line(aes(x=Time,y=Age.0.14),color="red")+
      geom_line(aes(x=Time,y=Age.15.39),color="blue")+
      geom_line(aes(x=Time,y=Age.40.64),color="green")+
      geom_line(aes(x=Time,y=Age.Over.65),color="black")+
      ylab("Cohorts")
```



Tidy Data

Time	Age 0-14	Age 15-39	Age 40-64	Age Over 65
2014.000	1000000.00	1500000.00	2000000.00	500000.00
2014.125	1004170.00	1500830.00	1997500.00	505625.00
2014.250	1008320.00	1501700.00	1995020.00	511230.00
2014.375	1012460.00	1502590.00	1992550.00	516816.00
2014.500	1016580.00	1503520.00	1990100.00	522383.00
2014.625	1020690.00	1504470.00	1987670.00	527930.00

- Variables in Columns
- Observations in Rows

Year	Cohort	Population
2014.000	Age 0-14	1000000.00
2014.000	Age 15-39	1500000.00
2014.000	Age 40-64	2000000.00
2014.000	Age Over 65	500000.00

melt() function – reshape library

melt(data, id.vars, measure.vars)

- data   Data set to melt
- id.vars   Id variables. If blank, will use all non measure.vars variables. Can be integer (variable position) or string (variable name)
- measure.vars Measured variables. If blank, will use all non id.vars variables. Can be integer (variable position) or string (variable name)

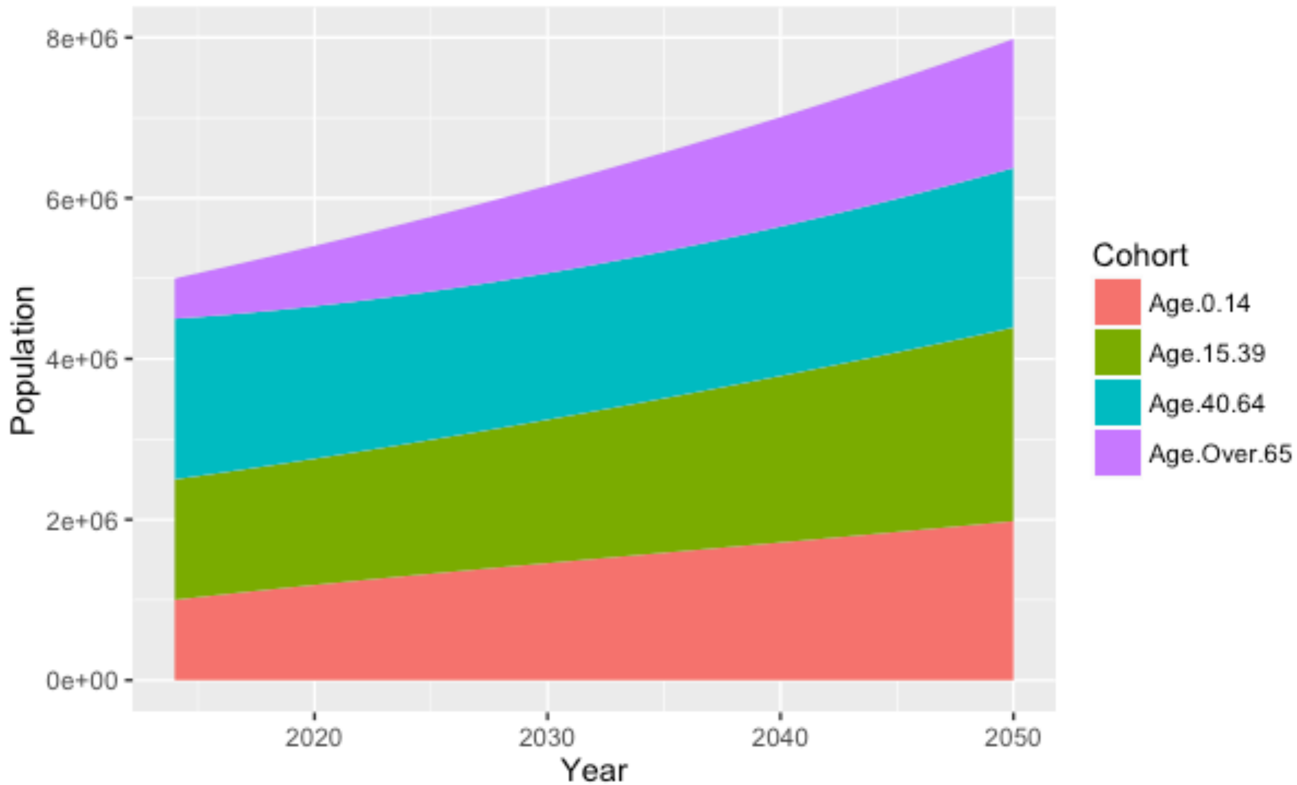
Population Example

```
sub<-sim[c( TRUE, rep(FALSE,7)),]
sub$Total<-NULL
msub<-melt(sub,id.vars = "Time")
names(msub)<-c("Year","Cohort","Population")
```

```
> head(msub)
  Year   Cohort Population
1 2014 Age.0.14    1000000
2 2015 Age.0.14    1032940
3 2016 Age.0.14    1065020
4 2017 Age.0.14    1096320
5 2018 Age.0.14    1126900
6 2019 Age.0.14    1156830
```

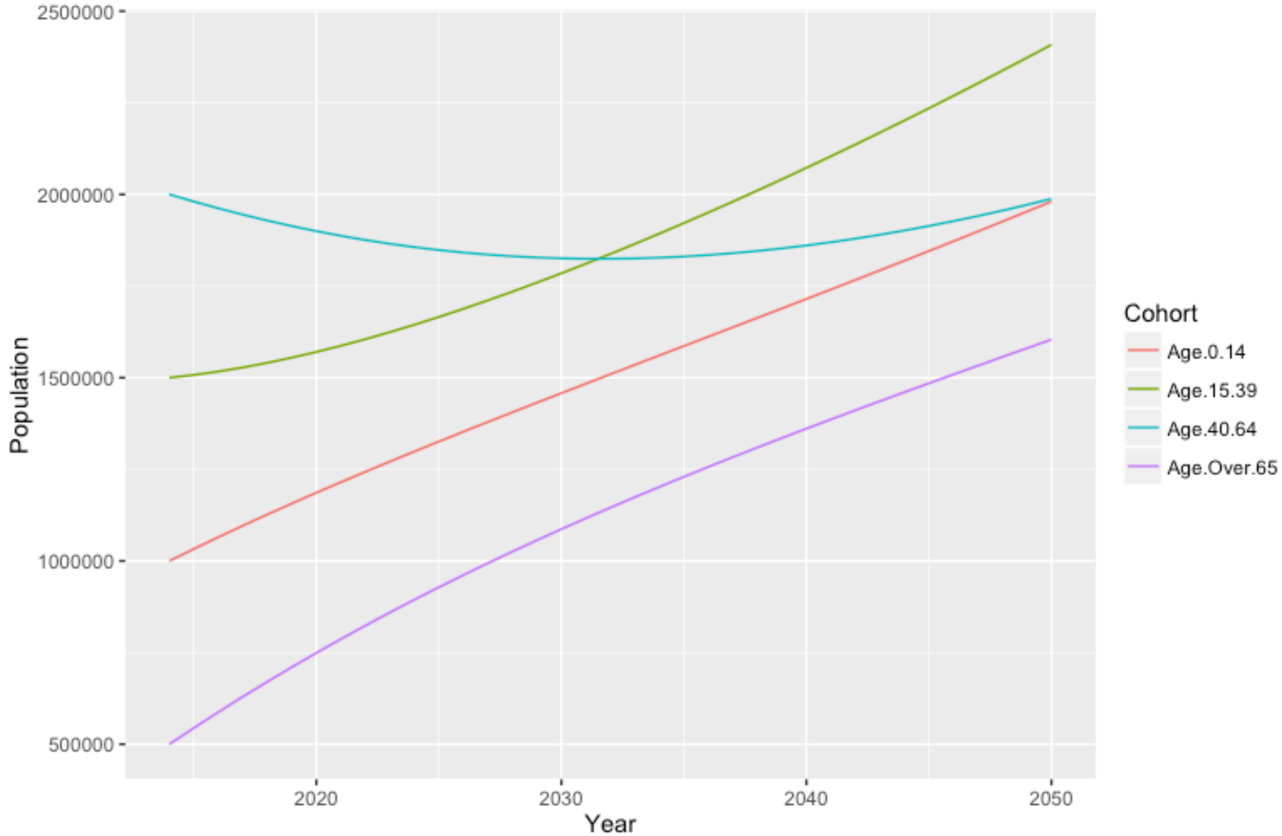
ggplot “likes” tidy data

```
ggplot(data=msub)+geom_area(aes(x=Year,y=Population,fill=Cohort))
```



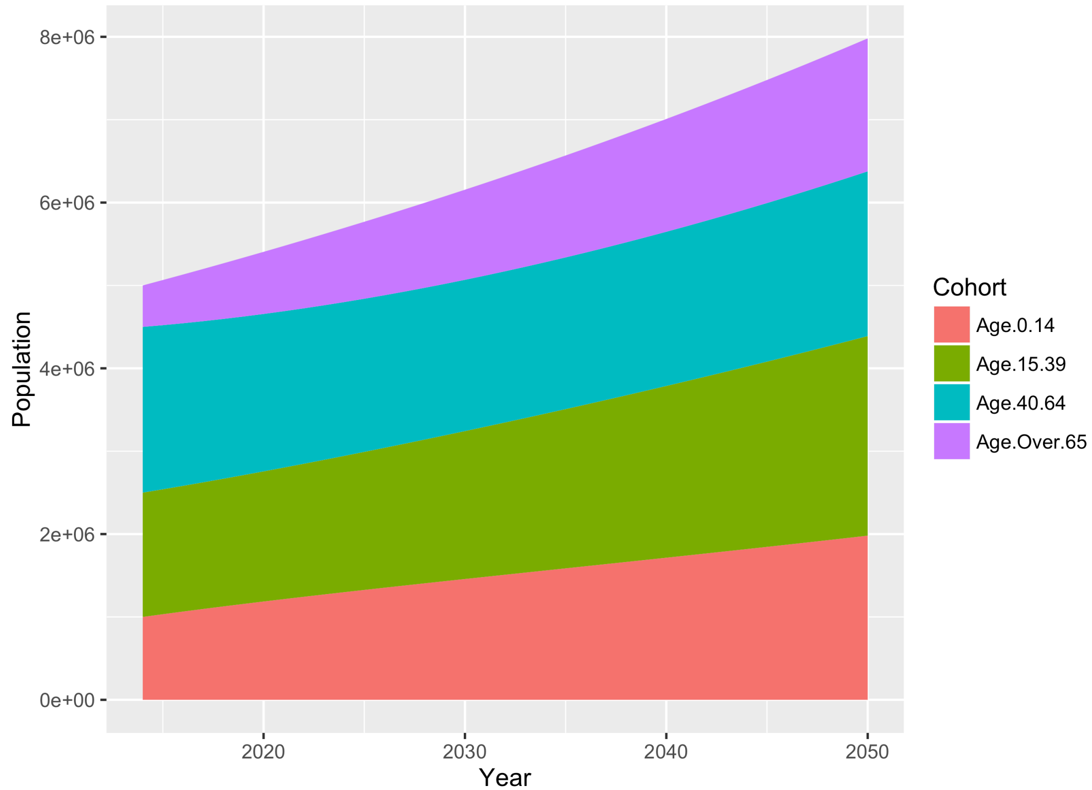
Another example…

```
ggplot(data=msub)+geom_line(aes(x=Year,y=Population,colour=Cohort))
```



Controlling Plot Resolution ggsave()

```
p1<-ggplot(data=msub)+geom_area(aes(x=Year,y=Population,fill=Cohort))
ggsave(file="workshop/models/01 session/hq_plot.png", height=5,  width=7,
        units="in",dpi=400,p1)
```



Challenge 4

- Plot the population data on an area chart showing the percentages for each cohort over the simulation time.