Ecological theory for the biodiversity crisis

Henrique Miguel Pereira

2024-10-08

Table of contents

Preface

We live in the midst of the biodiversity crisis. Biodiversity science, at first mostly a spin off from Ecology, has developed tremendously in the last two decades, becoming a truly interdisciplinary field, with contributions from Geography, Economy, Social Sciences and many other disciplines. Big data and the development of statistical ecology has also been remarkable. As I write these lines, the Global Biodiversity Information Facility or GBIF (https://gbif.org) is about to hit 3 billion records. The availability of large amounts of data and the availability of the open source R software (https://www.r-project.org) has made statistical approaches take the main stage in much of ecological research. However, at the same time, as a biodiversity researcher, practitioner, and a teacher, I find again and again the need for a good understanding of ecological theory to be able to carry out my work. This book if for all of you that have also felt a similar need.

Programming and mathematics are key tools of ecological theory. The programming language I use in this book is R (version 4). If you have familiarly with programming in another computer language, a fast skimming of Appendix A will be enough to get you going with the book. If you are new to programming, I recommend you spend a few hours working through the short course in Appendix A, but you may also wanna consult some web resources (e.g. https://ourcodingclub.github.io/course). With the advent of Large Language Models (LLM) such as ChatGPT there may be the temptation to think programming knowledge will no longer be needed. I think LLMs will accelerate our writing of computer code, but they will not replace the need for being able to read and understand the code.

Explain choice or R and open software/science. blabla

But programming is not enough. In order to be able to effectively develop theory and models of biodiversity, a good grasp of mathematics, including of probability and statistics is essential. One challenge is that, and this at least the case in Europe where I have lectured the most, most biology students receive very little mathematical training. This book cannot address by itself such gap and assumes some basic familiarity with probability theory (e.g. what is a probability distribution function), some calculus training (e.g. what area a derivative and an integral), a little algebra (e.g. multiplication of a matrix by a vector). I try to take the reader forward from that level. There are excellent books out there for those that need this basic background (e.g. Otto's and Day's A Biologist's Guide to Mathematical Modelling). And Wikipedia (https://www.wikipedia.org) is often your best friend when you wanna a fast refresh of any mathematical concept or even for many of the models and ecological concepts presented in this book.

Structure of the book blabla

This book can be used as the support for a semester course in Ecological Theory or Ecological Modelling. I have used many of these materials over the last decade in a similar course at iDiv/University of Halle-Wittenberg. But the book can also be used as self-learning tool or even as a reference tool. It takes a lot of experience from the Primer of Ecological Theory of Joan Roughgarden, that I was lucky to have as a mentor. As she used to say, one writes papers for the reviewers and books for the readers. This book is for you.

Part I

Ecology of individuals

Chapter 1

Ecophysiology and the climate space: when to bask in the sun?

We are now almost daily bombarded with news about climate change and its impacts on people and ecosystems. But how does climate mechanistically affects organisms? The basic foundations to address this question were laid out by ecophysiology research. Interestingly, at the time that some of seminal research on ecophysiology was carried out, back in the 1960's, climate change was not yet in the radar of most people. The research was driven by the interest in the basic understanding of how biophysical conditions affect organisms. Today the knowledge gained from this research has acquired new importance. This is an area whether the mechanistic knowledge learnt from theory and experiments is now sometimes overlooked because of the massive datasets and machine learning approaches available. But let's start with the basics.

1.1 A model for the body temperature of an animal

Animals can be divided in two big groups in regard to the way they regulate their temperature: ecoctherms and endotherms (Figure 1.1). Endotherms such as mammals and birds are able to regulate their temperature by producing heat through metabolism. Ectotherms in contrast must regulate their temperature by obtaining heat from the environment. By obtaining energy from the environment, ectotherms have lower energy demands than endotherms, which have to be burning energy all the time to keep their bodies warm. Both do have to avoid getting too hot because indeed there can bee too much from a good thing.

To build a model of the body temperature of an animal, let's consider a lizard that is perched in a rock basking in the sun (Figure 1.2). The lizard receives heat from the direct solar radiation than can be measured in for instance calories per hour. It can also receive indirect solar radiation, reflected for instance by the ground. The lizard also exchanges heat with the surrounding air through convection. If the lizard body temperature is higher than the surrounding air temperature it will lose heat, while if the lizard body temperature is lower than it will gain heat. The rate at which this exchange of heat through convection happens depends on the body of the animal

Figure 1.1: Shrews (left, *Crocidura russula*) are ectotherms and have high energy demands, eating almost their body weight every day. In contrast, vipers (right, *Vipera seonae*) can go days without eating, or even hibernate or aestivate as needed.

and the properties of its skin. These properties are captured in the heat transfer coefficient, which can be measured as calories per hour per degree Celsius. Similarly, the lizard can receive heat by conduction from being in contact with a warm rock, or lose heat to the rock if the rock is colder than the lizard's body temperature. Finally the animal can lose heat through evapotranspiration, this is by transpiring water that has a cooling effect when it evaporates. I also like to think about this model as representing our own experience on the beach. We can be laying on a towel receiving heat by conduction from the sand and solar radiation from the sun. If the air temperature is warm, we can become uncomfortable in the sun and we may seek a shade to reduce the input from solar radiation, but if there is a cool brise we may be able to stay in the sun a bit longer. If the air temperature is really cold and it's really windy our skin hair may rise to reduce the convection coefficient.

The total flow of energy into the lizard, also known as the heat exchange equation, can be written as

$$f = q - k(b - a) \tag{1.1}$$

where

- f is the energy flow (cal/h),
- q is the quantity of heat in the solar radiation (cal/h),
- k is the convection coefficient (cal/h/^{o}C),
- b is the body temperature (^{o}C) and
- a is the air temperature (^{o}C).

If the energy flow is positive, then the lizard is warming, while if it is negative then the lizard is

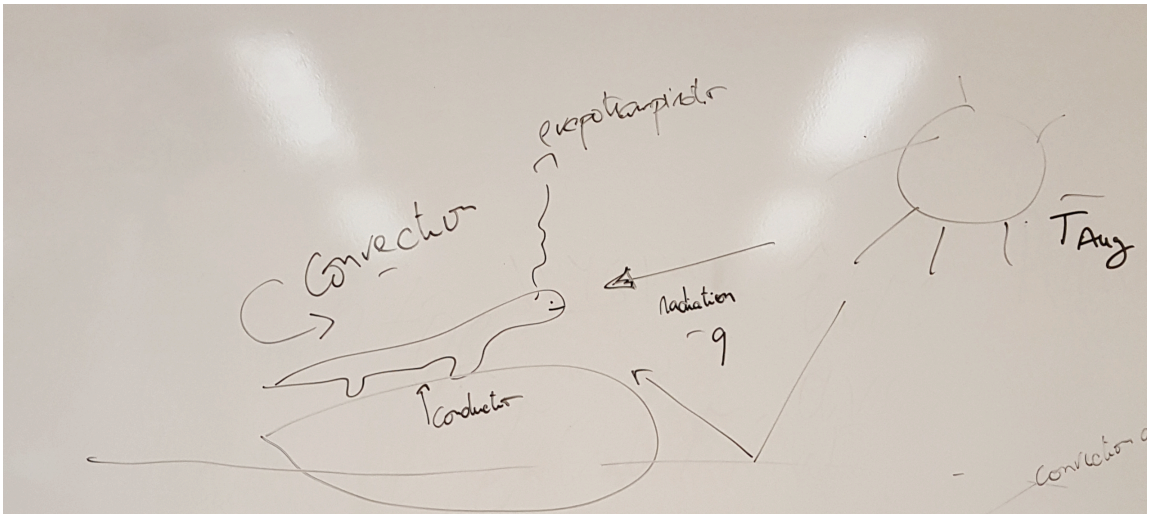


Figure 1.2: The energy flows of a lizard basking in the sun.

cooling. The equilibrium[1] happens when the energy flow is zero and so the the lizard is neither cooling nor warming. If we solve Equation 1.1 for equilibrium by replacing f with zero and expanding the right-hand side of the equation,

$$0 = q - k\ b + k\ a$$

and rearranging for b we find the equilibrium body temperature that we denote with a hat,

$$\hat{b} = q/k + a. \tag{1.2}$$

Let's see if this equation makes sense. It says that the equilibrium body temperature is the sum of the solar radiation divided by the convection coefficient with the air temperature. So, at minimum the equilibrium body temperature is equal to the air temperature when the solar radiation is zero (e.g. during the night). But when the solar radiation is greater than zero then the equilibrium body temperature is higher than the air temperature as one would expect. How much higher? Well that depends on the convection coefficient. If the convection coefficient is very high then the body temperature is mainly determined by the air temperature. In contrast, if the convection coefficient is very low then the solar radiation can contribute significantly to the body temperature.

It's time to start using **R** to explore this model. For instance, we can use **R** to plot the relationship between the body temperature and the solar radiation. First we create variables for the heat coefficient and the air temperature and assign some valuues:

[1]The concept of equilibrium is very important in ecological theory. It is often introduced in the context of differential equations and corresponds to the point at which the derivative of the variable of interest is zero. The heat equation can also be seen as a differential equation where f corresponds to the derivative of the amount of heat Q in the lizard through time t, i.e. dQ/dt.

```
k = 50  #convection coefficient (cal/h)
a = 18  #air temperature (ºC)
```

We want to plot the equilibrium body temperature for a range of solar radiation values. So we create a vector with radiation values, for instance ranging from 0 cal/h to 1500 cal/h in steps of 500. I am going to use big letters to denote vectors in **R** code, in contrast with scalars which I will denote with small letters.

```
Q = seq(0,1500,by=500)  #vector with radiation values
```

Now we can write Equation 1.2 in **R** to produce a vector of the equilibrium body temperatures for each value of radiation.

```
B_eq = Q/k+a   #vector with equilibrium body temperatures
```

Let's examine the values of the vector, by binding the two vectors as a matrix,

```
rbind(Q,B_eq)
```

```
     [,1] [,2] [,3] [,4]
Q       0  500 1000 1500
B_eq   18   28   38   48
```

This is nice as we can see the values of the equilibrium body temperature for each value of solar radiation. But let's visualize this as a graph, by plotting these vectors in **R**,

```
plot(Q,B_eq,type="l")
```

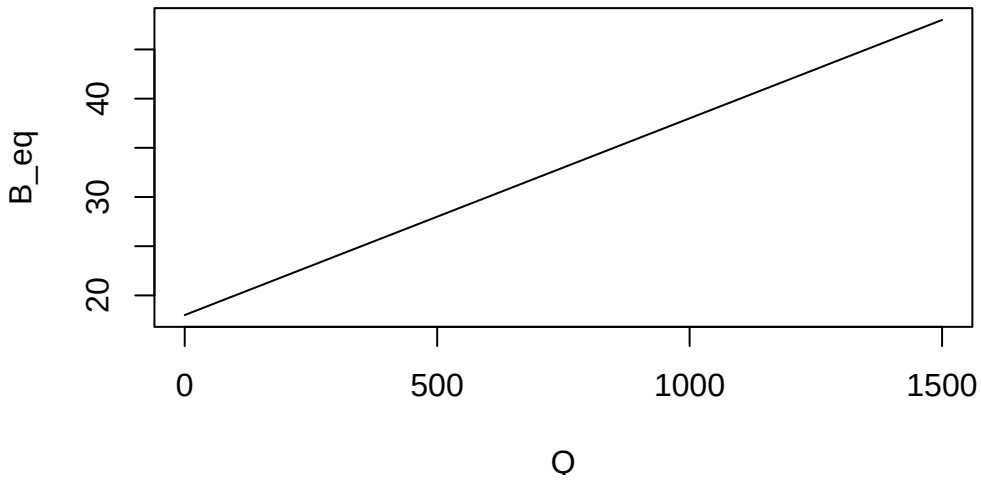


Figure 1.3: Equilibrium body temperature as a function of solar radiation.

Note that we added `type="l"` as a parameter of the plot to have a line drawn instead of a sequence of dots in the plot.

1.2 Understanding the climate space

Another way of looking at the heat exchange equation Equation 1.1 is to look at what combinations of air temperature and solar radiation values are livable for the lizard. This is also known as the climate space of an organism. In order to explore the climate space of the lizard we need to first assess what are the maximum and minimum body temperature that the lizard can experience. Let's assume those are respectively 36 and 24ºC and store them in **R,**

```
b_max = 36  #maximum body temperature
b_min = 24  #minimum body temperature
```

We now want to solve Equation 1.1 at equilibrium for the air temperature as a function of the solar radiation and body temperature. We can do this by rearranging Equation 1.2,

$$a = b - q/k. \tag{1.3}$$

This equation can be then used in **R** to calculate vectors of the maximum and minimum survivable air temperatures for each value of the solar radiation in vector `Q`,

```
A_max = b_max - Q/k  #vector with maximum air temperatures
A_min = b_min - Q/k  #vector with minimum air temperatures
```

We can now visualize the climate space, this is, the combination of solar radiation and air temperatures in which the lizard can survive, by plotting these two equations (i.e. by plotting lines with x coordinates given by the vector Q and the y coordinates given by the vectors A_max and A_min),

```
plot(Q, A_max,type="l", #plots the maximum survivable air temperature
     xlab="Solar radiation (cal/h)", #adds x and y axis labels to the plot
     ylab="Air temperature (ºC)")
lines(Q,A_min) #adds line for the minimum survivable air temperature
```

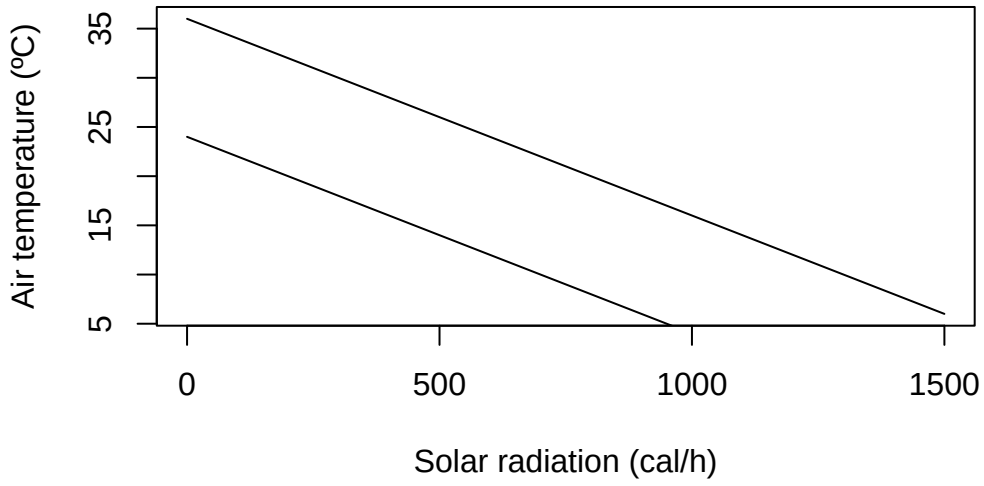


Figure 1.4: The climate space of a lizard.

Let's paint the area between the two lines which corresponds to the climate space of the lizard. We can use the **R** function `polygon`, to which we need to give the set of `x` and `y` coordinates delimiting the climate space as vectors. For instance, the upper lower corner can be the first coordinate, being 0 for the `x` vector and `b_min` for the `y` vector. The right upper corner is 1500 for `x` and `b_max-1500/k` for `y`.

```
X<-c(0,0,1500,1500)
Y<-c(b_min,b_max,b_max-1500/k,b_min-1500/k)
polygon(X,Y,col="green")
```

Now that we have the climate space we can plot on top of it some empirical data of how the conditions are in the field during the day. We can for instance consider two micro-habitats, a rock (in the sun) and a bush (in the shade). Suppose we obtain some data from temperature loggers that were installed in each micro-habitat , recording every three hours the values of temperature and radiation as tabled below.

Table 1.1: Hypothetical radiation values in two micro-habitats and air temperatures at different times of the day.

Time (hh:mm)	Radiation rock (cal/h)	Radiation bush (cal/h)	Temperature (ºC)
00:00	150	150	18
03:00	150	150	13
06:00	800	450	10
09:00	1100	600	14
12:00	1300	650	21
15:00	1200	650	24
18:00	800	350	22
21:00	400	200	20

We can overlay these values of temperatures and radiations on the plot to understand which micro-habitat should the lizard choose at each time of the day. First we create a vector for each column of Table 1.1. Note that we must enclose the times of the day in commas as they are strings. We also append at the end of the vector the values for 0:00 in order to close the lines (otherwise there would be a gap between 21:00 and 0:00).

```
#Times of the day
T <- c("00:00","03:00","06:00","09:00",
       "12:00","15:00","18:00","21:00","00:00")

#Solar radiation in the rock habitat
Rock_q <- c(150,150,800,1100,1300,1200,800,400,150)

#Air temperature in the rock habitat
Rock_a <- c(18,13,10,14,21,24,22,20,18)

#Solar radiation in the bush habitat
```

```
Bush_q <- c(150,150,450,600,650,650,350,200,150)

#   Air temperature in the bush habitat
Bush_a <- c(18,13,10,14,21,24,22,20,18)
```

We overlay these vectors in the figure by invoking the function `lines(Xcoor,Ycoord)` for each micro-habitat. We add labels to each point to show the correspondence between each point and the time of the day.

```
lines(Rock_q,Rock_a,col="orange")
text(Rock_q,Rock_a,T)

lines(Bush_q,Bush_a,col="blue")
text(Bush_q,Bush_a,T)
```

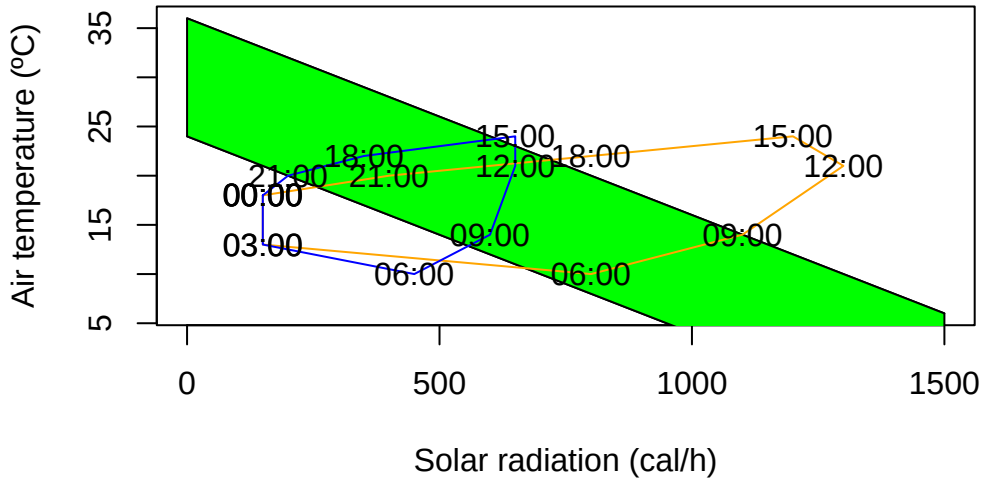
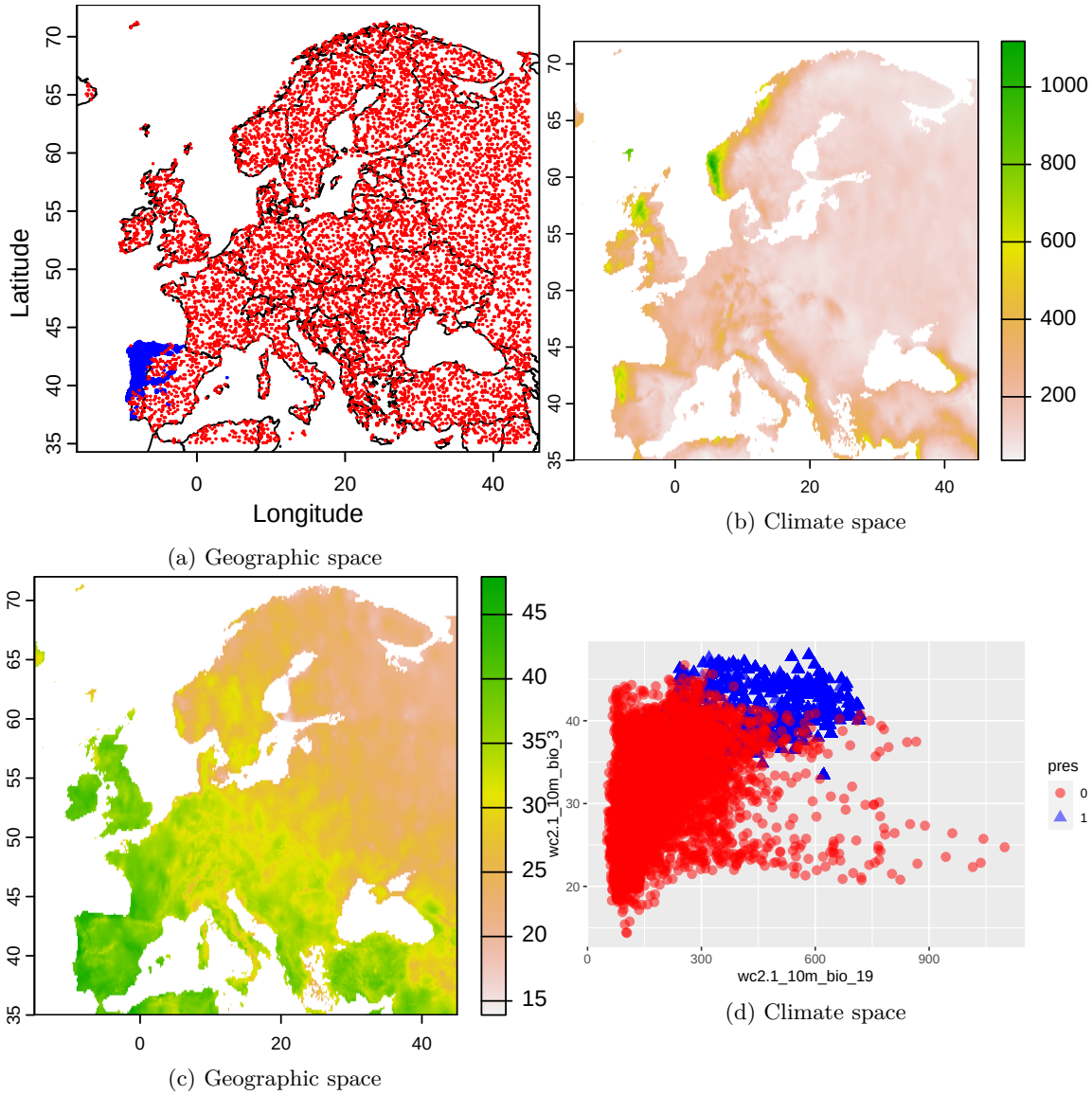


Figure 1.5: The climate space of a lizard with the conditions of two microhabitats ploted at different times of the day: rock (orange), bush (blue).

1.3 From ecophysiological models to species distribution models

In the previous section we developed a simple mechanistic model for the climate space of an organism. This is a micro-habitat level climate space. But one can also infer the climate space from the distribution of a species at the macro-habitat level. They are different but they are conceptually similar. The inference of such macro climate space is an area where species distribution models excel. The idea is relatively simple. One start with a bunch of locations of a species in space.



(a) Geographic space

(b) Climate space

(c) Geographic space

(d) Climate space

Figure 1.6: The geographic space and climate space of Lacerta schreiberi.

1.4 Computational Lab 1: Least squaring the climate

1.5 Statistical confrontation: Linear regression

Part II

Ecology of populations

Chapter 2

Demography: boom or bust

A brief R tutorial

If you are new to **R** you can have a short dive into its main features by working through this tutorial. If you had learnt programming in another computer language, you will be able to skim over this tutorial to find the main differences from what you have learnt to how things are done in **R.**

Variables, vectors and matrices

```
# Introduction to variables
# Variables can be any sequence of letter and numbers, but
# it cannot start with a number
x = 2
x <- 4
2+2
```

```
[1] 4
```

```
y <- x^5
y
```

```
[1] 1024
```

```
# Introduction to vectors
v1 <- c(2,3,6,12)
v2 <- 1:100
length(v2)
```

```
[1] 100
```

```
v2
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

```
v3 <- seq(1,100,5)  # call without naming arguments
v3
```

```
 [1]  1  6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
v3 <- seq(from=1,to=100,by=5) # call with names of arguments
v3
```

```
 [1]  1  6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
v3 <- seq(to=100,by=5) # call skipping the first argument
#and using the default value 1 - see help(seq)
v3
```

```
 [1]  1  6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
v3 <- seq(by=5,to=100) # call by arguments and change order or arguments

# Indexing vectors
v3[3] #uses square brackets to obtain the third element of the vector
```

```
[1] 11
```

```
v3>20 # produce a vector of boolean values that are TRUE when
```

```
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
      #v3 is greater than 20
v3[v3>20] # select from v3 all the values that are greater than 20
```

```
 [1] 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
v4<-c(1,2,3,4,5)
v4[c(FALSE,TRUE,FALSE,TRUE,FALSE)] #select from v4 the second and fourth element
```

```
[1] 2 4
```

```
v3[1:10] # first ten elements
```

```
 [1]  1  6 11 16 21 26 31 36 41 46
```

```
v3[-1] # dropping first element
```

```
 [1]  6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
head(v2) # prints the first few elements of v2
```

```
[1] 1 2 3 4 5 6
```

```
tail(v2) # prints the last few elements of v2
```

```
[1]  95  96  97  98  99 100
```

```
which(v3 == 26) # returns the position of v3 that equals 26
```

```
[1] 6
```

```
#Numerical operations with vectors
2^v2
```

```
  [1] 2.000000e+00 4.000000e+00 8.000000e+00 1.600000e+01 3.200000e+01
  [6] 6.400000e+01 1.280000e+02 2.560000e+02 5.120000e+02 1.024000e+03
 [11] 2.048000e+03 4.096000e+03 8.192000e+03 1.638400e+04 3.276800e+04
 [16] 6.553600e+04 1.310720e+05 2.621440e+05 5.242880e+05 1.048576e+06
 [21] 2.097152e+06 4.194304e+06 8.388608e+06 1.677722e+07 3.355443e+07
 [26] 6.710886e+07 1.342177e+08 2.684355e+08 5.368709e+08 1.073742e+09
 [31] 2.147484e+09 4.294967e+09 8.589935e+09 1.717987e+10 3.435974e+10
 [36] 6.871948e+10 1.374390e+11 2.748779e+11 5.497558e+11 1.099512e+12
 [41] 2.199023e+12 4.398047e+12 8.796093e+12 1.759219e+13 3.518437e+13
 [46] 7.036874e+13 1.407375e+14 2.814750e+14 5.629500e+14 1.125900e+15
 [51] 2.251800e+15 4.503600e+15 9.007199e+15 1.801440e+16 3.602880e+16
 [56] 7.205759e+16 1.441152e+17 2.882304e+17 5.764608e+17 1.152922e+18
 [61] 2.305843e+18 4.611686e+18 9.223372e+18 1.844674e+19 3.689349e+19
 [66] 7.378698e+19 1.475740e+20 2.951479e+20 5.902958e+20 1.180592e+21
 [71] 2.361183e+21 4.722366e+21 9.444733e+21 1.888947e+22 3.777893e+22
 [76] 7.555786e+22 1.511157e+23 3.022315e+23 6.044629e+23 1.208926e+24
 [81] 2.417852e+24 4.835703e+24 9.671407e+24 1.934281e+25 3.868563e+25
 [86] 7.737125e+25 1.547425e+26 3.094850e+26 6.189700e+26 1.237940e+27
 [91] 2.475880e+27 4.951760e+27 9.903520e+27 1.980704e+28 3.961408e+28
 [96] 7.922816e+28 1.584563e+29 3.169127e+29 6.338253e+29 1.267651e+30
```

```
log(v2)
```

```
  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
  [8] 2.0794415 2.1972246 2.3025851 2.3978953 2.4849066 2.5649494 2.6390573
 [15] 2.7080502 2.7725887 2.8332133 2.8903718 2.9444390 2.9957323 3.0445224
 [22] 3.0910425 3.1354942 3.1780538 3.2188758 3.2580965 3.2958369 3.3322045
 [29] 3.3672958 3.4011974 3.4339872 3.4657359 3.4965076 3.5263605 3.5553481
 [36] 3.5835189 3.6109179 3.6375862 3.6635616 3.6888795 3.7135721 3.7376696
 [43] 3.7612001 3.7841896 3.8066625 3.8286414 3.8501476 3.8712010 3.8918203
 [50] 3.9120230 3.9318256 3.9512437 3.9702919 3.9889840 4.0073332 4.0253517
 [57] 4.0430513 4.0604430 4.0775374 4.0943446 4.1108739 4.1271344 4.1431347
 [64] 4.1588831 4.1743873 4.1896547 4.2046926 4.2195077 4.2341065 4.2484952
 [71] 4.2626799 4.2766661 4.2904594 4.3040651 4.3174881 4.3307333 4.3438054
 [78] 4.3567088 4.3694479 4.3820266 4.3944492 4.4067192 4.4188406 4.4308168
 [85] 4.4426513 4.4543473 4.4659081 4.4773368 4.4886364 4.4998097 4.5108595
 [92] 4.5217886 4.5325995 4.5432948 4.5538769 4.5643482 4.5747110 4.5849675
 [99] 4.5951199 4.6051702
```

```
v5 <- 101:200
v5-v2
```

```
  [1] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
 [19] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
 [37] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
 [55] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
 [73] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
 [91] 100 100 100 100 100 100 100 100 100 100
```

```
v5/v2
```

```
  [1] 101.000000  51.000000  34.333333  26.000000  21.000000  17.666667
  [7]  15.285714  13.500000  12.111111  11.000000  10.090909   9.333333
 [13]   8.692308   8.142857   7.666667   7.250000   6.882353   6.555556
 [19]   6.263158   6.000000   5.761905   5.545455   5.347826   5.166667
 [25]   5.000000   4.846154   4.703704   4.571429   4.448276   4.333333
 [31]   4.225806   4.125000   4.030303   3.941176   3.857143   3.777778
 [37]   3.702703   3.631579   3.564103   3.500000   3.439024   3.380952
 [43]   3.325581   3.272727   3.222222   3.173913   3.127660   3.083333
 [49]   3.040816   3.000000   2.960784   2.923077   2.886792   2.851852
 [55]   2.818182   2.785714   2.754386   2.724138   2.694915   2.666667
 [61]   2.639344   2.612903   2.587302   2.562500   2.538462   2.515152
 [67]   2.492537   2.470588   2.449275   2.428571   2.408451   2.388889
 [73]   2.369863   2.351351   2.333333   2.315789   2.298701   2.282051
 [79]   2.265823   2.250000   2.234568   2.219512   2.204819   2.190476
 [85]   2.176471   2.162791   2.149425   2.136364   2.123596   2.111111
 [91]   2.098901   2.086957   2.075269   2.063830   2.052632   2.041667
 [97]   2.030928   2.020408   2.010101   2.000000
```

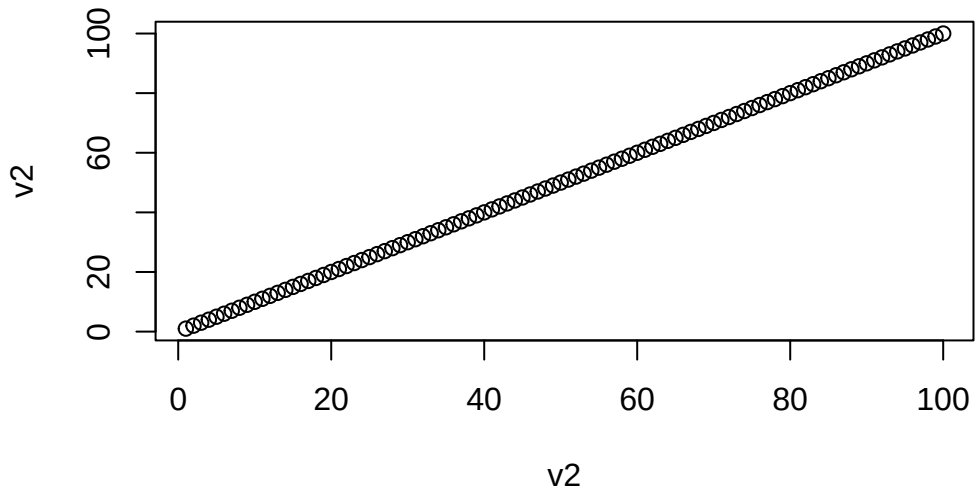
```
v1/v2
```

```
  [1] 2.00000000 1.50000000 2.00000000 3.00000000 0.40000000 0.50000000
  [7] 0.85714286 1.50000000 0.22222222 0.30000000 0.54545455 1.00000000
 [13] 0.15384615 0.21428571 0.40000000 0.75000000 0.11764706 0.16666667
 [19] 0.31578947 0.60000000 0.09523810 0.13636364 0.26086957 0.50000000
 [25] 0.08000000 0.11538462 0.22222222 0.42857143 0.06896552 0.10000000
 [31] 0.19354839 0.37500000 0.06060606 0.08823529 0.17142857 0.33333333
 [37] 0.05405405 0.07894737 0.15384615 0.30000000 0.04878049 0.07142857
 [43] 0.13953488 0.27272727 0.04444444 0.06521739 0.12765957 0.25000000
 [49] 0.04081633 0.06000000 0.11764706 0.23076923 0.03773585 0.05555556
 [55] 0.10909091 0.21428571 0.03508772 0.05172414 0.10169492 0.20000000
 [61] 0.03278689 0.04838710 0.09523810 0.18750000 0.03076923 0.04545455
 [67] 0.08955224 0.17647059 0.02898551 0.04285714 0.08450704 0.16666667
 [73] 0.02739726 0.04054054 0.08000000 0.15789474 0.02597403 0.03846154
 [79] 0.07594937 0.15000000 0.02469136 0.03658537 0.07228916 0.14285714
 [85] 0.02352941 0.03488372 0.06896552 0.13636364 0.02247191 0.03333333
 [91] 0.06593407 0.13043478 0.02150538 0.03191489 0.06315789 0.12500000
 [97] 0.02061856 0.03061224 0.06060606 0.12000000
```

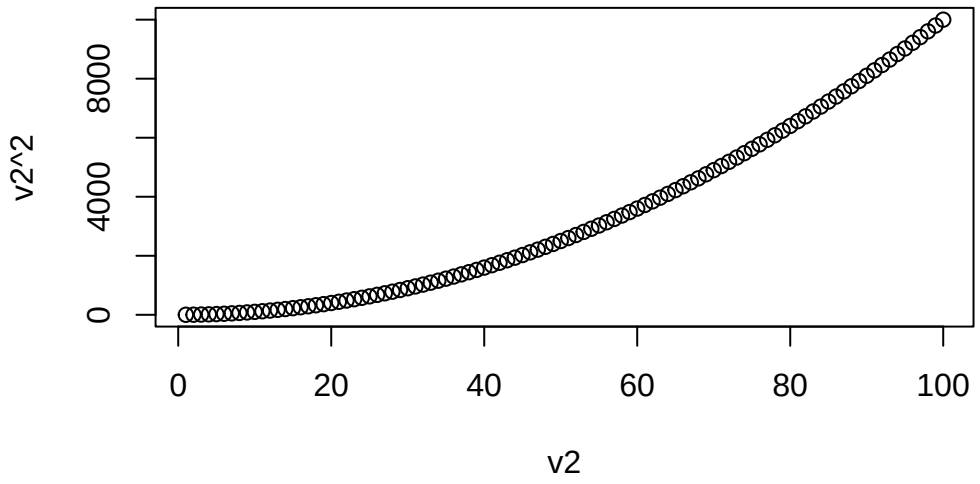
```
#Using strings in R
mystring <- "Ecology"
vstrg <- c("Anna", "Peter", "Xavier")
vstrg[2]
```

```
[1] "Peter"
```

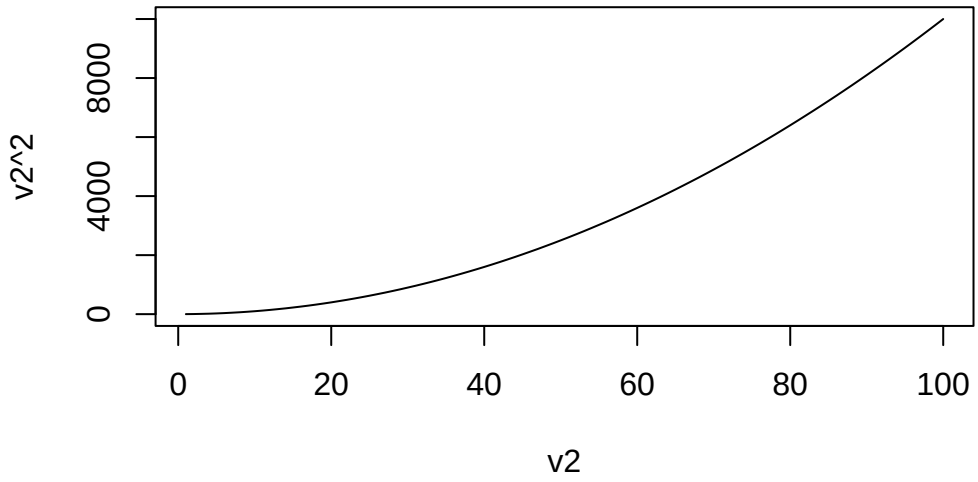
```
#making plots in R
plot(v2,v2)
```



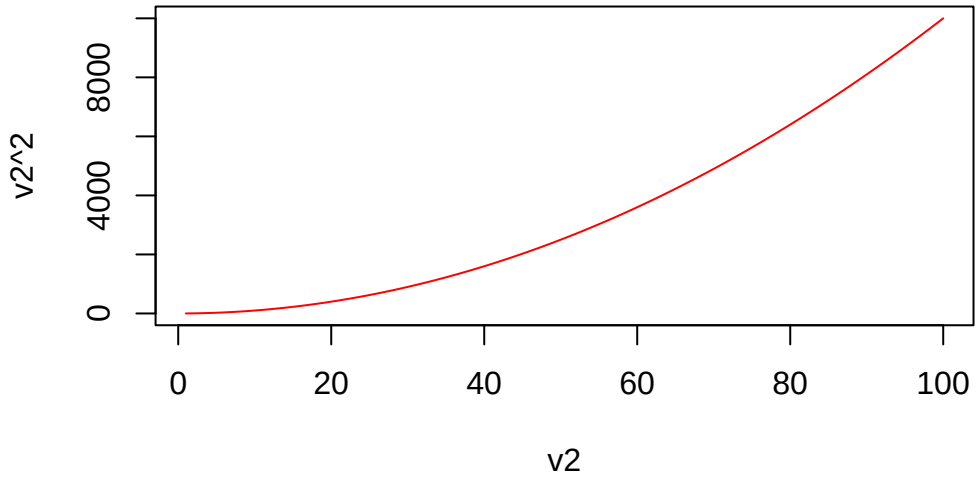

```
plot(v2,v2^2)
```



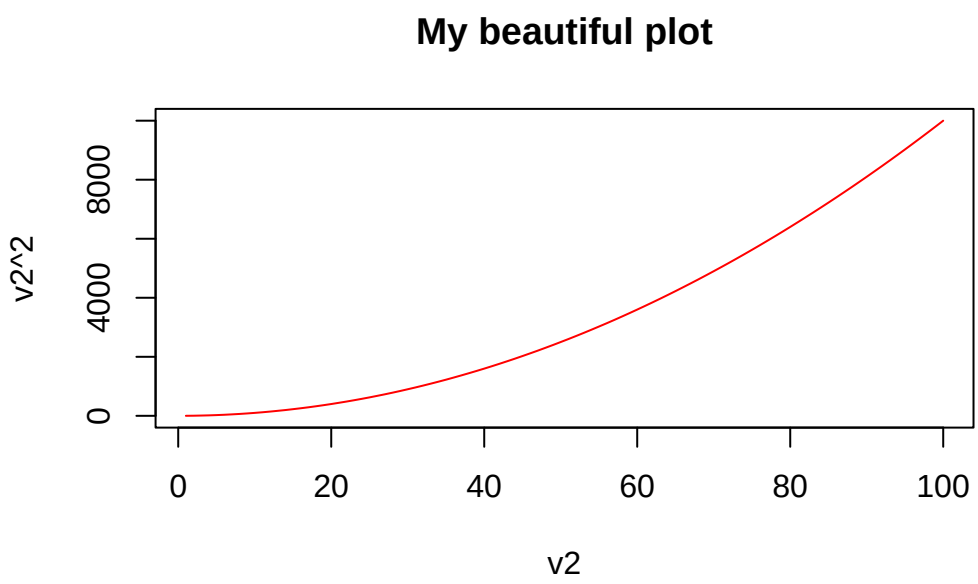

```
plot(v2,v2^2,type="l")
```



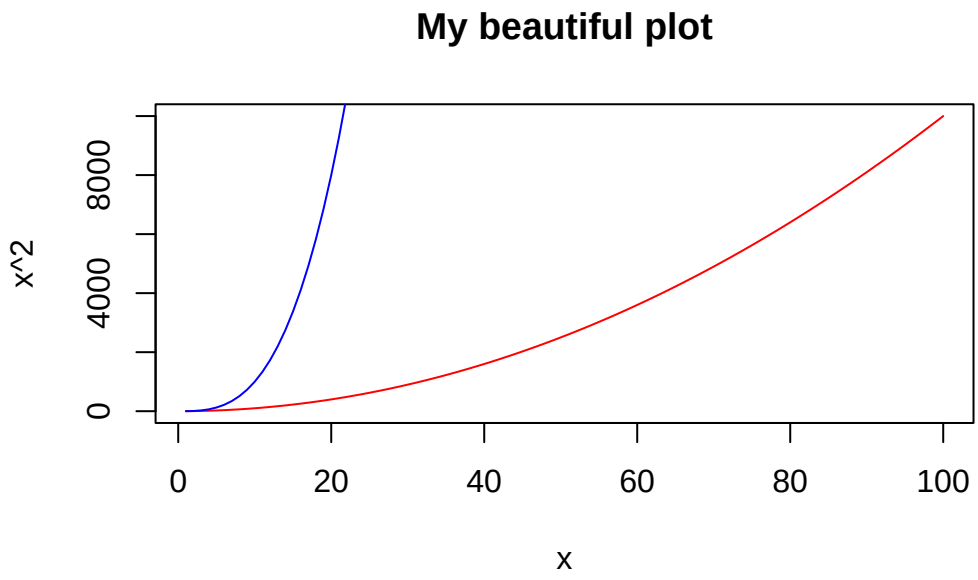

```
plot(v2,v2^2,type="l",col="red")
```




```
plot(v2,v2^2,type="l",col="red",main="My beautiful plot")
```




```
plot(v2,v2^2,type="l",col="red",main="My beautiful plot",xlab="x",
     ylab="x^2")
lines(v2,v2^3,col="blue")
```




```
# Matrices in R
m <- matrix(5,3,2)
m
```

```
     [,1] [,2]
[1,]    5    5
```

```
[2,]    5    5
[3,]    5    5
```

```
m2 <- matrix(1:6,3,2)
m2
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
t(m2) # transposes matrix
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

```
x <- 1:4
y <- 5:8

m3<-cbind(x,y)
m3
```

```
     x y
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
```

```
m4<-rbind(x,y)
m4
```

```
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    5    6    7    8
```

```
# Indexing matrices
m3[3,2] #element in row 3 and column 2
```

```
y
7
```

```
m3[1,] #entire first row
```

```
x y
1 5
```

```
m3[,1] #entire first column
```

```
[1] 1 2 3 4
```

```
colnames(m3)<-c("col1","col2")
m3
```

```
     col1 col2
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

```
m3[,"col2"]
```

```
[1] 5 6 7 8
```

```
# Lists in R
mylist <- list(elem1=m,elem2=v2,elem3="my list")
mylist$elem2
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

```
# Dataframes
df <- as.data.frame(m3)
df$col1
```

```
[1] 1 2 3 4
```

Iterations and conditional expressions

```
# FOR loops

for (k in 1:10)  # for k =1, 2, 3, 4, 5,...10
  print (k^2)   #do this
```

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

```
R <- 1.2
n <- 1
print(n[1])
```

```
[1] 1
```

```
for (t in 1:100)
{
  n[t+1] <- R*n[t]
  print(n[t+1])
}
```

```
[1] 1.2
[1] 1.44
[1] 1.728
[1] 2.0736
[1] 2.48832
[1] 2.985984
[1] 3.583181
[1] 4.299817
[1] 5.15978
[1] 6.191736
[1] 7.430084
[1] 8.9161
[1] 10.69932
[1] 12.83918
[1] 15.40702
[1] 18.48843
[1] 22.18611
[1] 26.62333
[1] 31.948
[1] 38.3376
[1] 46.00512
[1] 55.20614
[1] 66.24737
[1] 79.49685
[1] 95.39622
[1] 114.4755
[1] 137.3706
[1] 164.8447
[1] 197.8136
[1] 237.3763
[1] 284.8516
[1] 341.8219
[1] 410.1863
[1] 492.2235
[1] 590.6682
```

```
[1] 708.8019
[1] 850.5622
[1] 1020.675
[1] 1224.81
[1] 1469.772
[1] 1763.726
[1] 2116.471
[1] 2539.765
[1] 3047.718
[1] 3657.262
[1] 4388.714
[1] 5266.457
[1] 6319.749
[1] 7583.698
[1] 9100.438
[1] 10920.53
[1] 13104.63
[1] 15725.56
[1] 18870.67
[1] 22644.8
[1] 27173.76
[1] 32608.52
[1] 39130.22
[1] 46956.26
[1] 56347.51
[1] 67617.02
[1] 81140.42
[1] 97368.5
[1] 116842.2
[1] 140210.6
[1] 168252.8
[1] 201903.3
[1] 242284
[1] 290740.8
[1] 348889
[1] 418666.7
[1] 502400.1
[1] 602880.1
[1] 723456.1
[1] 868147.4
[1] 1041777
[1] 1250132
[1] 1500159
[1] 1800190
[1] 2160228
[1] 2592274
```

```
[1] 3110729
[1] 3732875
[1] 4479450
[1] 5375340
[1] 6450408
[1] 7740489
[1] 9288587
[1] 11146304
[1] 13375565
[1] 16050678
[1] 19260814
[1] 23112977
[1] 27735572
[1] 33282687
[1] 39939224
[1] 47927069
[1] 57512482
[1] 69014979
[1] 82817975
```

```
R <- 1.2
n <- 1
for (t in 1:100)
  n[t+1] <- R*n[t]

# IF conditional statement

# logical operators
# == equal to
# > greater than
# < smaller than
# >= greater or equal
# <= smaller or equal
# != different from
# && and
# || or

if (3>2) print ("yes")
```

```
[1] "yes"
```

```
if (3==2) print ("yes") else print("no")
```

```
[1] "no"
```

```
if ((3>2)&&(4>5)) print ("yes")

for (k in 1:10)  # for k =1, 2, 3, 4, 5,...10
```

```
  if (k^2>20) print (k^2)
```

```
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

Writing functions

```
# creating FUNCTIONS in r

pythagoras <- function (c1,c2)
{
  h <- sqrt (c1^2 + c2^2)
  return (h)
}

pythagoras(1,1)
```

```
[1] 1.414214
```

```
pythagoras(5,5)
```

```
[1] 7.071068
```

```
pythagoras(10,1)
```

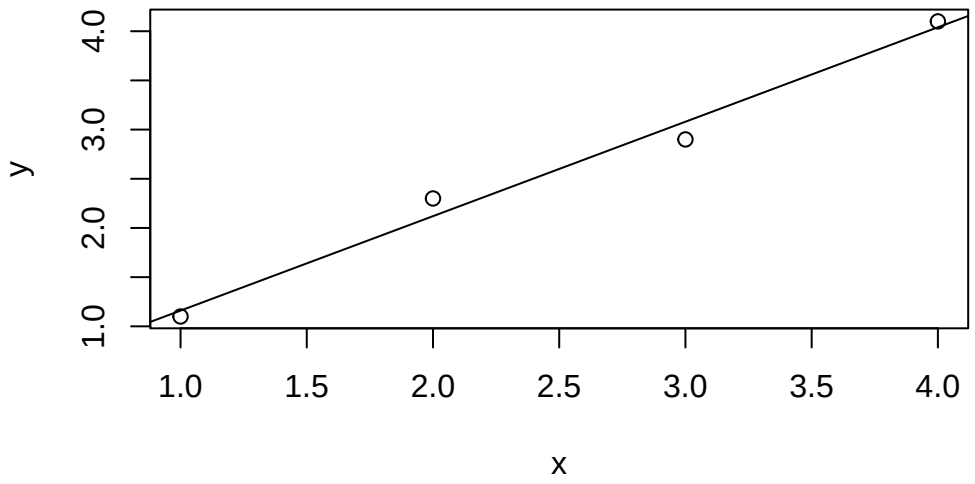
```
[1] 10.04988
```

```
# regression in R

help(lm)
x <- c(1,2,3,4)
y <- c(1.1,2.3,2.9,4.1)
plot(x,y)
myreg<-lm(y ~ x)
summary(myreg)
```

```

Call:
lm(formula = y ~ x)

Residuals:
    1     2     3     4
-0.06  0.18 -0.18  0.06
```

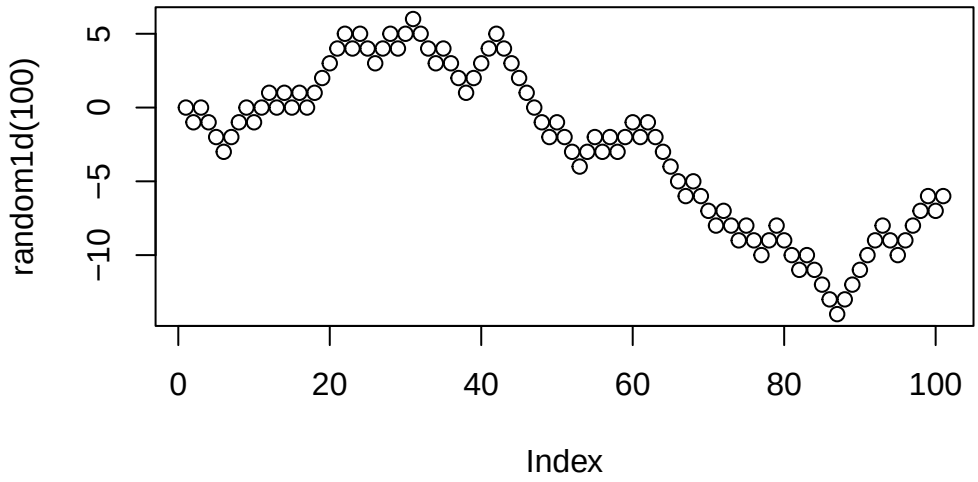
```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.20000    0.23238   0.861  0.48012
x            0.96000    0.08485  11.314  0.00772 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1897 on 2 degrees of freedom
Multiple R-squared:  0.9846,    Adjusted R-squared:  0.9769
F-statistic:   128 on 1 and 2 DF,  p-value: 0.007722
```

```
abline(myreg)
```



Random numbers and statistical distributions

```
random1d<-function(tmax)
{
x<-0
for (t in 1:tmax)
{
  r<-runif(1)
  if (r<1/2)
    x[t+1]<-x[t]+1 else
      x[t+1]<-x[t]-1
}
return(x)
}

plot(random1d(100))
```

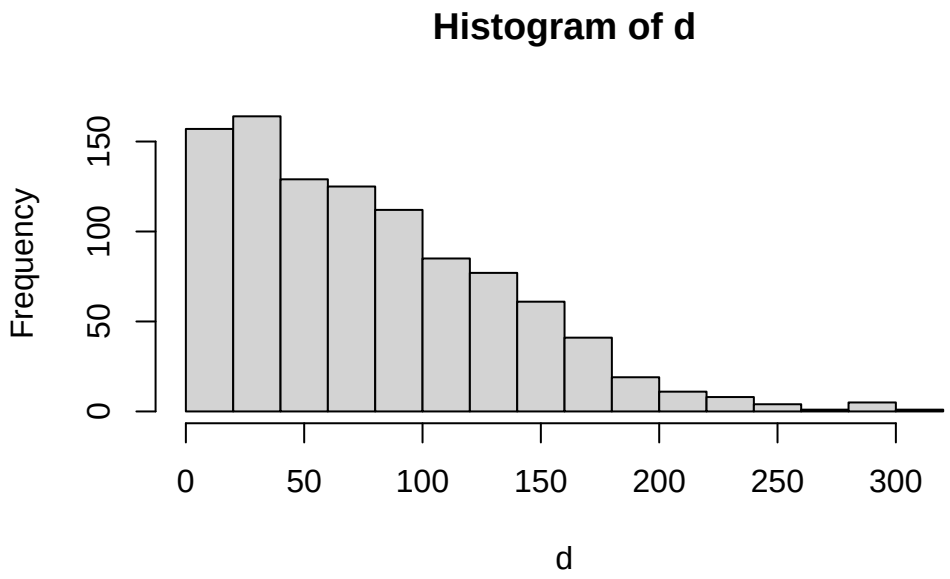



```
tmax<-10000
lastx<-0
for (i in 1:1000)
{
  x<-random1d(tmax)
  lastx[i]<-x[tmax]
}

hist(lastx)
```


```
mean(lastx)
```

```
[1] 1.368
```

```
d<-sqrt(lastx^2)
hist(d)
```




```
mean(d)
```

```
[1] 77.754
```

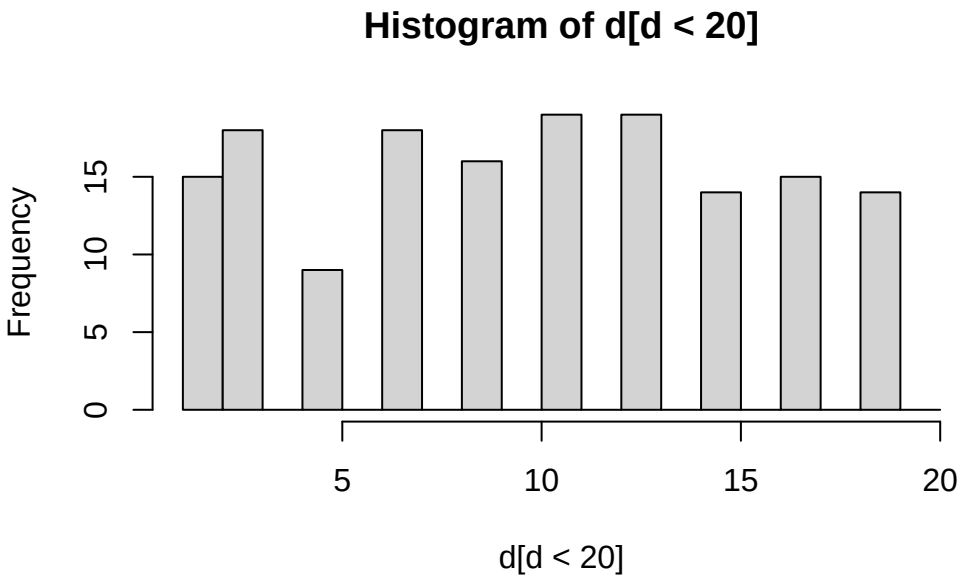
```
median(d)
```

```
[1] 67
```

```
max(d)
```

```
[1] 305
```

```
hist(d[d<20],breaks=c(1:20))
```



Spatial analysis

Shapefiles and Raster (Isabel Rosa)

When you work with spatial data, essentially you use two types of data:

1) vector data (i.e., shapefiles): stores the geometric location and attribute information of geographic features. These can be represented by points, lines, or polygons (areas).
2) matricial data (i.e., raster): consists of a matrix of cells (or pixels) organized into rows and columns (or a grid) where each cell contains a value representing information. They can be categorical or continuous and have multiple bands.

For more information on the tree cover datasets, please see: https://earthenginepartners.appspot.com/science-2013-global-forest/download_v1.2.html

```
# read in shapefile using rgdal
sc <- readOGR(".", "SantaCatarina")

# import municipalities and settlements shapefiles
sc_mun <- readOGR(".", "SantaCatarina_mun")
br_sett <- readOGR(".", "Brazil_settlements")

# always good to check the contents of your dat
#str(br_sett)

# visualize one of the variables
spplot(sc_mun, z="Shape_Area", main = "Municipality Area (km2)")
```

```
# read in raster
tc<-raster("tree_cover.tif")

# import loss and gain rasters here
tl<-raster("loss.tif")
tg<-raster("gain.tif")

# for multiple band rasters, you can choose to import just one or all bands
#r2 <- raster("tree_cover_multi.tif", band=2)

# note that the value 255, which is Hansen's nodata value was not recognized as such
NAvalue(tg) # check first
NAvalue(tc)<-255 #fix it by forcing 255 to be the nodata
NAvalue(tl)<-255 #fix it by forcing 255 to be the nodata
NAvalue(tg)<-255 #fix it by forcing 255 to be the nodata

# visualize one of the rasters
par(mfrow=c(1,3))
plot(tc, main = "Tree Cover (%)")
plot(tl, main = "Tree Cover Loss (binary)")
plot(tg, main = "Tree Cover Gain (binary)")
```

Reference systems

Coordinate systems are essential to understand when working with spatial data. Some reading material on this can be found here: Essentially, if one wants to know which position of the Earth we refer to, coordinates of geospatial data require a reference system:

1) geodesic/geographic coordinates need an order (lat/long), a unit (e.g., degrees) and a datum (a reference ellipsoid: e.g. WGS84)
2) cartesian/projected coordinates (e.g. UTM, web Mercator) need also measurement units (e.g., meters), and some way of encoding how they relate to geodesic coordinates, in which datum (this is handled by the GIS system)

Operations with Shapefiles

Clip: in R you can clip using the command "intersect", so that intersect(feature to be clipped, clip feature) Select: you can use a boolean selection to subset the features of your shapefile, for instance if you just want to look at settlements with a mininum number of habitants, so that Population > median(Population) There are several options, have a look at this great tutorial: http://www.rspatial.org/spatial/rst/7-vectmanip.html

```
# Clip the settlement features using the Santa Catarina shapefile
sc_sett<-intersect(br_sett, sc)

#sc_sett$med <- sc_sett$population > median(sc_sett$population) # oops! annoyingly our population va
```

```
# convert to original numerical values
sc_sett$population<-as.numeric(as.vector(sc_sett$population))
# careful! applying as.numeric alone it will not work!!

# visualize results
plot(sc_sett, main = "Settlements in Santa Catarina")
spplot(sc_sett, z="population", main = "Population per Settlement (people)")

# select settlements larger then the median value
sc_sett$med <- sc_sett$population > median(sc_sett$population)
sc_largesett <- sc_sett[sc_sett$med == 1, ]

# visualize results
par(mfrow=c(1,2))
plot(sc_sett, main = "All Settlements")
plot(sc_largesett, main = "Largest Settlements")
```

Operations with Rasters

There are many operations you can do with rasters, and these are more frequently used in spatial analyses than shapefiles. Here I will just illustrate a couple of simple operations: - Global/Raster statistics - obtain a value that summarizes the whole raster layer - Cell statistics (pixel-by-pixel operation): obtains a value per pixel - Focal statistics (operation that takes into account neighborhood of central cell) - results in raster of same of different size - Zonal statistics - calculates summary statistics of a give raster (e.g., elevation) based on pre-defined zones (e.g., admnistrative boundaries, biomes). Outputs a table with the values per zone. For more great examples, have a look here: http://www.rspatial.org/spatial/rst/8-rastermanip.html

```
# sum the loss and gain rasters to know where there was simultaneous loss and gain in Santa Catarina
tclg<-tl+tg
par(mfrow=c(1,3))
plot(tl, main = "Forest Loss")
plot(tg, main = "Forest Gain")
plot(tclg, main = "Forest Loss and Gain")

# you can also try to create three new rasters and work with them
# create a new raster
r <- raster(ncol=10, nrow=10, xmx=-80, xmn=-150, ymn=20, ymx=60)
values(r) <- runif(ncell(r)) # assign random values
#plot(r)

# create two more rasters based on the first one
r2 <- r * r
r3  <- sqrt(r)

# either stack or brick them
```

```
s <- stack(r, r2, r3)
#b <- brick(s)

# Raster statistics - calculate several statistics per raster layer (i.e., sum, mean, median)
cellStats(s, "sum") # outputs a value per raster

# Cell statistics - calculate several statistics per pixel  (i.e., sum, mean, median)
par(mfrow=c(2,2))
plot(r, main ="Random 1")
plot(r2, main ="Random 2")
plot(r3, main ="Random 3")
plot(overlay(s, fun="mean"), main="Average Values") # outputs a new raster

# Focal statistics - calculate mean accounting for the neighborhood values, compare with previous ou
f1 <- focal(tc, w=matrix(1,nrow=5,ncol=5) , fun=mean)
plot(f1, main = "Average forest cover 5x5")
# sum the loss, vary window size
f2 <- focal(tl, w=matrix(1,nrow=5,ncol=5) , fun=sum)
plot(f2, main = "Total forest loss 5x5")
# sum the gain, vary window size
f3 <- focal(tg, w=matrix(1,nrow=5,ncol=5) , fun=sum)
plot(f3, main = "Total forest gain 5x5")

# plot 4 maps with different window sizes
par(mfrow=c(2,2))
for(i in c(5,15,25,55)){
  f_w <- focal(tc, w=matrix(1,nrow=i,ncol=i) , fun=sum)
  plot(f_w, main = paste0("Window size: ", i))
}

# Zonal Statistics - using two rasters
sc_tc_mean_loss <- zonal(tc, tl, fun=mean) #average tree cover in loss areas
sc_tc_mean_gain <- zonal(tc, tg, fun=mean) #average tree cover in gain areas

# average tree cover loss
sc_tc_mean_loss

# average tree cover gain
sc_tc_mean_gain
```

Operations with both Rasters and Shapefiles

Here I'll show a couple of examples of operation that use feature data as inputs and output rasters: Distance to features - calculates the euclidean distance from each cell/pixel to the closest feature (e.g., roads, settlements). Outputs a raster file with these distances. Interpolation: a world in itself!

Very vey short example provided here (based on a single method, IDW), please see more here: http://www.rspatial.org/analysis/rst/4-interpolation.html To better understand interpolation I advise you to read first about spatial autocorrelation: http://www.rspatial.org/analysis/rst/3-spauto.html

To use interpolation metrics you need to load another packaged called gstat Inverse distance weighted (IDW) - See more also here: http://desktop.arcgis.com/en/arcmap/10.3/tools/3d-analyst-toolbox/how-idw-works.htm

```
# create an empty raster (little trick using existing raster)
dist_sett<-tc*0
# or you can create an empty one like before
# dist_sett <- raster(ncol=ncol(tc), nrow=nrow(tc), xmx=extent(tc)@xmax, xmn=extent(tc)@xmin, ymn=ex

# Distance to points
dist_sett <- distanceFromPoints(dist_sett, sc_sett)

# you can then mask the outside area of Santa Catarina
dist_sett <- mask(dist_sett, tc)

# plot results
plot(dist_sett, main = "Distance to settlements (m)")

# load gstat
library(gstat)
idw_sett<-tc*0

# compute the model, see reference for more detail
gs <- gstat(formula=population~1, locations=sc_sett, nmax=5, set=list(idp = 2))
idw_out <- interpolate(idw_sett, gs)

## [inverse distance weighted interpolation]
sc_pop <- mask(idw_out, tc)
plot(sc_pop, main = "Santa Catarina Population")
```

Export Shapefiles and Rasters

It's very easy to export both shapefiles and rasters from R to be visualized in QGIS or ArcMap.

```
# Save feature layers (point, polygon, polyline) to shapefile
writeOGR(sc_largesett, dsn=".", layer="SC_largeSett", driver="ESRI Shapefile" )

# or
#shapefile(sc_largesett, "SC_largeSett.shp", overwrite=TRUE)

#Exporting raster
writeRaster(sc_pop, filename="SC_popmap", format="GTiff" )
```

Working with biodiversity data: GBIF, EBV Portal (Corey Callaghan, Luise Quoss)First we load the library rgbif.

```
library(rgbif)
library(tidyverse)
```

Now we will download observations of a species. Let's download observations of the common toad "Bufo bufo".

```
matbufobufo<-occ_search(scientificName="Bufo bufo", limit=500, hasCoordinate = TRUE, hasGeospatialIs
```

Let's examine the object *matbufobufo*

```
class(matbufobufo)
matbufobufo
```

Let's download data about octupusses. They are in the order "Octopoda". First we need to find the GBIF search key for Octopoda.

```
a<-name_suggest(q="Octopoda",rank="Order")
key<-a$data$key
```

```
octopusses<-occ_search(orderKey=key,limit=2000, hasCoordinate = TRUE, hasGeospatialIssue = FALSE)
```

Show the result

```
octmat<-octopusses$data
head(octmat)
```

Count the number of observations per species using tidyverse and pipes

```
#class(octmat)
octmat %>%
  group_by(scientificName) %>%
  summarise(sample_size=n()) %>%
  arrange(desc(sample_size)) %>%
  mutate(sample_size_log=log(sample_size,2)) %>%
  ggplot(aes(x = sample_size_log)) + geom_histogram()
```

Plot the records on an interactive map. First load the leaflet package.

```
library(leaflet)
leaflet(data=octmat) %>% addTiles() %>%
  addCircleMarkers(lat= ~decimalLatitude, lng = ~decimalLongitude,popup=~scientificName)
```

Version 2 (Isabel Rosa)

Here are the packages we'll need.

```
library(rgbif)
library(tidyverse)
library(raster)
library(maps)
library(leaflet)
library(sdmpredictors)
```

First let's pick an example species to download data for. We will only download 500 observations to keep it simple for now. If you were doing this for real, you would download all data for that species (see notes at the end). I will choose the European Robin: https://en.wikipedia.org/wiki/European_robin.

```
species <- occ_search(scientificName="Erithacus rubecula", limit=500, hasCoordinate = TRUE, hasGeosp
```

What does this object look like?

```
class(species)

species
```

It is a special object of class `gbif` which allows for the metadata and the actual data to all be included, as well as taxonomic hierarchy data, and media metadata. We won't worry too much about the details of this object now. But we do want to get a dataframe we can work with! To do this, we have one extra step.

```
sp_dat <- species$data

class(sp_dat)

head(sp_dat)
```

So this was just for one species. Lets broaden this out a little bit. What if we were interested in many species of a given order/class? Here, we will choose an entire order to download. I will choose owls! https://en.wikipedia.org/wiki/Owl. First, we need to find the 'key' that gbif uses for that order and then pass it to our GBIF download function. Again, we are only getting a small number of records for illustration purposes.

```
a <- name_suggest(q='Strigiformes')

key <- a$data$key

order <- occ_search(orderKey=key, limit=1000, hasCoordinate = TRUE, hasGeospatialIssue=FALSE)
```

What kind of object is 'order'? As with species, we need to turn it into a dataframe to work with.

```
order_dat <- order$data

class(order_dat)
```

```
head(order_dat)
```

Count the number of observations by species

```
order_dat %>%
  group_by(scientificName) %>%
  summarize(sample_size=n()) %>%
  arrange(desc(sample_size))
```

Plot the records on an interactive map. First for our chosen species.

```
leaflet(data = sp_dat) %>%
    addTiles() %>%
    addCircleMarkers(lat = ~decimalLatitude, lng = ~decimalLongitude,popup = ~scientificName)
```

Then for the order we chose.

```
leaflet(data = order_dat) %>%
    addTiles() %>%
    addCircleMarkers(lat = ~decimalLatitude, lng = ~decimalLongitude, popup = ~scientificName)
```

Climate data

Let's play with some global climate data and overlay that with our GBIF observations.

```
mean_temp_map <- getData(name="worldclim", res=10, var="tmean")
plot(mean_temp_map)
```

Each month has separate values for each cell. To combine to a yearly value, we just take the mean.

```
annual_mean_temp <- mean(mean_temp_map)/10 #data comes as degrees * 10
```

Now plot this.

```
plot(annual_mean_temp)
```

To get out the values for the organism of interest, we use `extract`.

```
org_temp <- extract(annual_mean_temp, cbind(x=sp_dat$decimalLongitude, y=sp_dat$decimalLatitude))
```

Now we will visualize how the global distribution of temperature values compares with the species' distribution of temperature values. This shows the distribution of temps where robins are found versus the global distribution of temps.

```
temp <- tibble(mean_temp=getValues(annual_mean_temp)) %>%
  filter(!is.na(mean_temp))

temp_org <- tibble(organism_temp=org_temp) %>%
  filter(!is.na(organism_temp))
```

```
ggplot(temp, aes(x=mean_temp))+
  geom_density(fill="blue")+
  geom_density(data=temp_org, aes(x=organism_temp), fill="red")+
  theme_bw()
```

Intro to apply, pipes, ggplot2, tidyverse.

Introduction to gpplot

```
cars
library(ggplot2)

ggplot(data=cars, mapping=aes(x=speed,y=dist)) + geom_point(colour="red")
```

```
plot(cars$speed,cars$dist)
```

```
#Introduction to R - 9


# a recursive function that calculates a factorial
myfun <- function(x)
{
  if (x==1)
    return (1)
  else return(x*myfun(x-1))
}

myfun(1:10) # does not work

#option1 - with a for loop
start_time <- Sys.time()
y<-0
for (i in 1:100)
  y[i]<-myfun(i)
end_time <- Sys.time()
end_time-start_time
y
plot(y,type="l")

#option 2 - with apply
start_time <- Sys.time()
y<-sapply(1:100,myfun)
end_time <- Sys.time()
end_time-start_time
y
```

```r
# selecting a subset from a matrix and applying a function to a column of that subset

setwd("~/iDiv Dropbox/Henrique Pereira/Teaching/Spatial Ecology/Spatial Ecology 2022/2_Lab_assignmen
Florida <- read.csv("Florida.csv")

# number of species for year 1970 and route 20
tapply(Florida$Abundance,Florida$Route==20 & Florida$Year==1970, length)

# matrix with number of species per route and per year
out<-tapply(Florida$Abundance,list(Florida$Route,Florida$Year), length)

names(out[,1])
plot(out[10,])
plot(out[20,])


shannon<-function(x)
{
  p<-x/sum(x)
  - sum(p*log(p))
}

out<-tapply(Florida$Abundance,list(Florida$Route,Florida$Year), shannon)
plot(out[10,])

library(tidyverse)

#our first pipe
x<-rnorm(1000)
hist(x)

rnorm(1000) %>%  hist

t<-1:ncol(out)
myreg<-lm(out[10,]~t)
summary(myreg)
plot(out[10,])
abline(myreg)

lm(out[10,]~t) %>% summary
plot(out[10,])
lm(out[10,]~t) %>% abline

#ggplot
mat=cbind(t,out[10,])
```

```
data(cars)
colnames(mat)<-c("time","shannon")
mat<-as.data.frame(mat)

myplot <-  ggplot(mat, aes(time,shannon))+
  geom_point()
myplot

myplot <-  ggplot(mat, aes(time,shannon))+
  geom_line()
myplot

data(cars)
myplot <-  ggplot(cars, aes(speed,dist))+
  geom_point()+geom_line()
myplot

data(cars)
myplot <-  ggplot(cars, aes(speed,dist))+
  geom_point()+geom_smooth(method="lm")
myplot

data(cars)
myplot <-  ggplot(cars, aes(speed,dist))+
  geom_point()+geom_smooth(method="lm")+scale_x_log10()+scale_y_log10()
myplot
```