1. Statistical Modeling in R

2. Descriptive statistics

3. R Packages: Regression (MASS package)

4. Distribution (STATS package)

5. ANOVA

6. Time Series Analysis.

1. Statistical Modeling in R

A **statistical model** is a mathematical model that embodies a set of statistical assumptions concerning the generation of some sample data and similar data from a larger population. A statistical model represents, often in considerably idealized form, the data-generating process.

There are several standard statistical models to fit the data using R. The key to modeling in R is the formula object, which provides a shorthand method to describe the exact model to be fit to the data. Modeling functions in R typically require a formula object as an argument. The modeling functions return a model object that contains all the information about the fit

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

2. Descriptive Statistics

Descriptive statistics are brief descriptive coefficients that summarize a given data set, which can be either a representation of the entire population or a sample of it. Descriptive statistics are broken down into measures of central tendency and measures of variability, or spread. Measures of central tendency include the mean, median and mode, while measures of variability include the standard deviation or variance, the minimum and maximum variables.

Measures of central tendency describe the center position of a distribution for a data set. A person analyzes the frequency of each data point in the distribution and describes it using the mean, median or mode, which measure the most common patterns of the data set being analyzed.

Measures of variability, or the measures of spread, aid in analyzing how spread-out the distribution is for a set of data. For example, while the measures of central tendency may give a person the average of a data set, it doesn't describe how the data is distributed within the set. So, while the average of the data may be 65 out of 100, there can still be data points at both 1 and 100. Measures of variability help communicate this by describing the shape and spread of the data set. Range, quartiles, absolute deviation and variance are all examples of measures of variability.

Mean

It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function mean() is used to calculate this in R.

Syntax

The basic syntax for calculating mean in R is ,

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

x -> is the input vector.

trim ->is used to drop some observations from both end of the sorted vector.

na.rm ->is used to remove the missing values from the input vector.

Median

The middle most value in a data series is called the median. The median()function is used in R to calculate this value.

Syntax

The basic syntax for calculating median in R is −

median(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Maximum:

It represents maximum value in the given data set.

The basic syntax for calculating Maximum in R is −

max(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Minimum:

It represents minimum value in the given data set.

The basic syntax for calculating Minimum in R is −

max(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Range:

The difference between the maximum and minimum data entries in the set.

Range = (Max. data entry) – (Min. data entry)

The basic syntax for calculating Range in R is −

range(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

The standard deviation

It measures variability and consistency of the sample or population. In most real-world applications, consistency is a great advantage. In statistical data analysis, less variation is often better.

The basic syntax for calculating Range in R is −

sd(x, na.rm = FALSE)

x ->is the input vector.

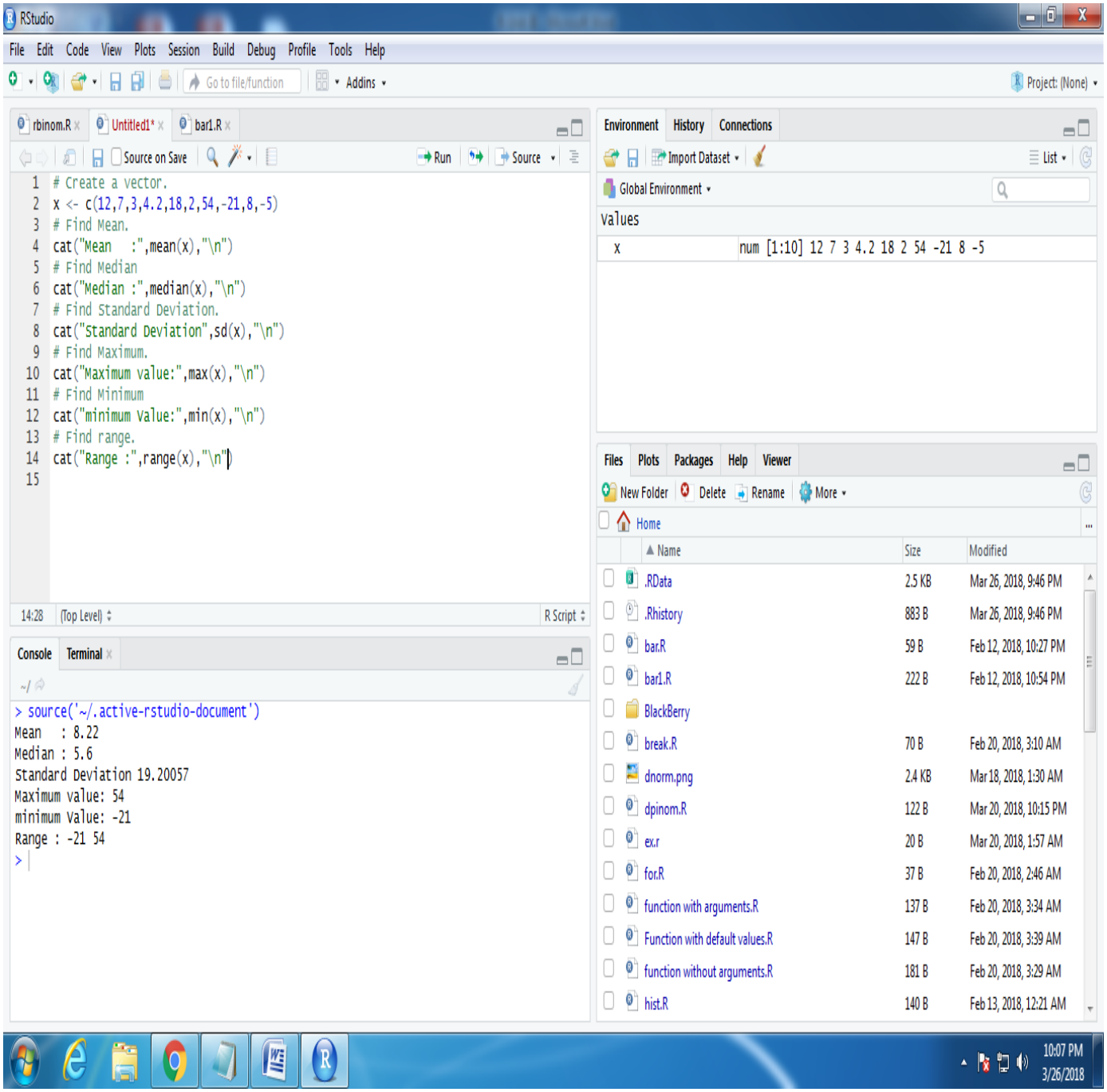
na.rm ->is used to remove the missing values from the input vector.

Example:

Create a vector.

```
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.

print("Mean   :",mean(x))

# Find Median

print("Median :",median(x))

# Find Standard Deviation.

print("Standard Deviation",sd(x))

# Find Maximum.

print("Maximum value:",max(x))

# Find Minimum

print("minimum Value:",min(x))

# Find range.

print("Range :",range(x))
```

Output:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
rbinom.R
Untitled1*
bar1.R
Source on Save
Run
Source
1 # Create a vector.
2 x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
3 # Find Mean.
4 cat("Mean :",mean(x),"\n")
5 # Find Median
6 cat("Median :",median(x),"\n")
7 # Find Standard Deviation.
8 cat("Standard Deviation",sd(x),"\n")
9 # Find Maximum.
10 cat("Maximum value:",max(x),"\n")
11 # Find Minimum
12 cat("minimum Value:",min(x),"\n")
13 # Find range.
14 cat("Range :",range(x),"\n")
15
14:28 (Top Level) R Script
Console Terminal
> source('~/.active-rstudio-document')
Mean : 8.22
Median : 5.6
Standard Deviation 19.20057
Maximum value: 54
minimum Value: -21
Range : -21 54
>
Environment History Connections
Import Dataset List
Global Environment
Values
x num [1:10] 12 7 3 4.2 18 2 54 -21 8 -5
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home
Name Size Modified
.RData 2.5 KB Mar 26, 2018, 9:46 PM
.Rhistory 883 B Mar 26, 2018, 9:46 PM
bar.R 59 B Feb 12, 2018, 10:27 PM
bar1.R 222 B Feb 12, 2018, 10:54 PM
BlackBerry
break.R 70 B Feb 20, 2018, 3:10 AM
dnorm.png 2.4 KB Mar 18, 2018, 1:30 AM
dpinom.R 122 B Mar 20, 2018, 10:15 PM
ex.r 20 B Mar 20, 2018, 1:57 AM
for.R 37 B Feb 20, 2018, 2:46 AM
function with arguments.R 137 B Feb 20, 2018, 3:34 AM
Function with default values.R 147 B Feb 20, 2018, 3:39 AM
function without arguments.R 181 B Feb 20, 2018, 3:29 AM
hist.R 140 B Feb 13, 2018, 12:21 AM
10:07 PM
3/26/2018

3.Regression

Linear Regression

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is −

```
y = ax + b
```

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.

- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.

Input Data

Below is the sample data representing the observations −

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is −

```
lm(formula,data)
```

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
```

```
# Apply the lm() function.

relation <- lm(y~x)

print(relation)
```

When we execute the above code, it produces the following result −

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
   -38.4551       0.6746
```

Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

print(summary(relation))
```

When we execute the above code, it produces the following result −

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3002  -1.6629  0.0412   1.8944  3.9775

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict() Function

Syntax

The basic syntax for predict() in linear regression is −

```
predict(object, newdata)
```

Following is the description of the parameters used −

- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

# Find weight of a person with height 170.

a <- data.frame(x = 170)

result <-  predict(relation,a)

print(result)
```

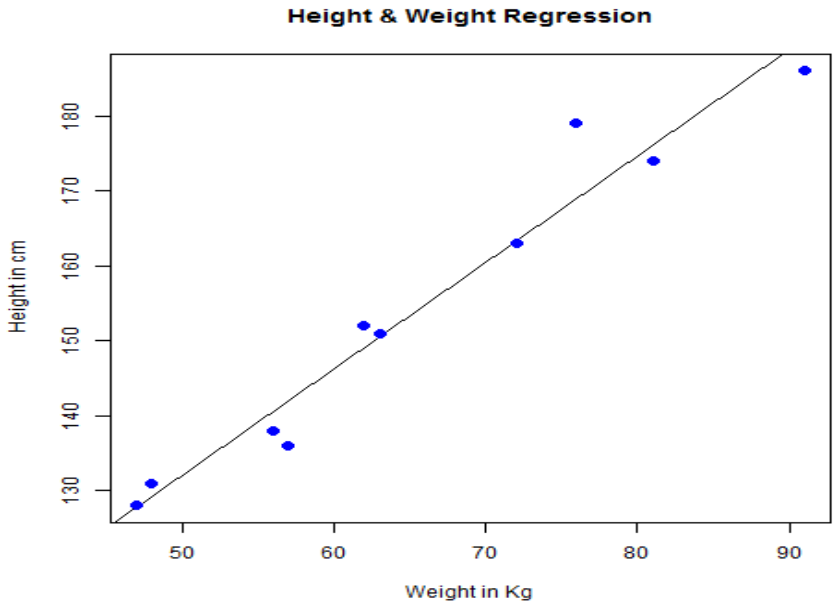
When we execute the above code, it produces the following result −

```
     1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

# Plot the chart.

plot(y,x,col = "blue",main = "Height & Weight Regression",

abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.

dev.off()
```

When we execute the above code, it produces the following result −



R - Multiple Regression

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

The general mathematical equation for multiple regression is −

```
y = a + b1x1 + b2x2 +...bnxn
```

Following is the description of the parameters used −

- **y** is the response variable.
- **a, b1, b2...bn** are the coefficients.
- **x1, x2, ...xn** are the predictor variables.

We create the regression model using the **lm()** function in R. The model determines the value of the coefficients using the input data. Next we can predict the value of the response variable for a given set of predictor variables using these coefficients.

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in multiple regression is −

```
lm(y ~ x1+x2+x3...,data)
```

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between the response variable and predictor variables.
- **data** is the vector on which the formula will be applied.

Example

Input Data

Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.

The goal of the model is to establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

```
input <- mtcars[,c("mpg","disp","hp","wt")]

print(head(input))
```

When we execute the above code, it produces the following result −

```
                  mpg   disp   hp    wt
```

```
Mazda RX4           21.0  160    110   2.620
Mazda RX4 Wag       21.0  160    110   2.875
Datsun 710          22.8  108     93   2.320
Hornet 4 Drive      21.4  258    110   3.215
Hornet Sportabout   18.7  360    175   3.440
Valiant             18.1  225    105   3.460
```

Create Relationship Model & get the Coefficients

```
input <- mtcars[,c("mpg","disp","hp","wt")]

# Create the relationship model.

model <- lm(mpg~disp+hp+wt, data = input)

# Show the model.

print(model)

# Get the Intercept and coefficients as vector elements.
```

When we execute the above code, it produces the following result −

```
Call:
lm(formula = mpg ~ disp + hp + wt, data = input)

Coefficients:
(Intercept)        disp          hp          wt
  37.105505     -0.000937      -0.031157   -3.800891
```

Create Equation for Regression Model

Based on the above intercept and coefficient values, we create the mathematical equation.

```
Y = a+Xdisp.x1+Xhp.x2+Xwt.x3
            or
Y = 37.15+(-0.000937)*x1+(-0.0311)*x2+(-3.8008)*x3
```

Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horse power and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is

```
Y = 37.15+(-0.000937)*221+(-0.0311)*102+(-3.8008)*2.91 = 22.7104
```

R - Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −

```
y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))
```

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are the coefficients which are numeric constants.

The function used to create the regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in logistic regression is −

```
glm(formula,data,family)
```

Following is the description of the parameters used −

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is binomial for logistic regression.

Example

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

```
# Select some columns form mtcars.

input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))
```

When we execute the above code, it produces the following result −

```
                  am   cyl  hp    wt
Mazda RX4          1   6    110   2.620
Mazda RX4 Wag      1   6    110   2.875
Datsun 710         1   4     93   2.320
Hornet 4 Drive     0   6    110   3.215
Hornet Sportabout  0   8    175   3.440
Valiant            0   6    105   3.460
```

Create Regression Model

We use the **glm()** function to create the regression model and get its summary for analysis.

```
input <- mtcars[,c("am","cyl","hp","wt")]

am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)

print(summary(am.data))
```

When we execute the above code, it produces the following result −

```
Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.17272  -0.14907  -0.01464   0.14116   1.27641

Coefficients:
             Estimate   Std. Error    z value    Pr(>|z|)
(Intercept) 19.70288    8.11637      2.428      0.0152 *
cyl          0.48760    1.07162      0.455      0.6491
hp           0.03259    0.01886      1.728      0.0840 .
wt          -9.14947    4.15332     -2.203      0.0276 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

   Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance:  9.8415  on 28  degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8
```

Conclusion

In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

4. DISTRIBUTION

Normal Distribution:

The normal distribution is the most widely known and used of all distributions. Because the normal distribution approximates many natural phenomena so well, it has developed into a standard of reference for many probability problems.

The normal distribution is defined by the following probability density function, $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$

where μ is the population mean

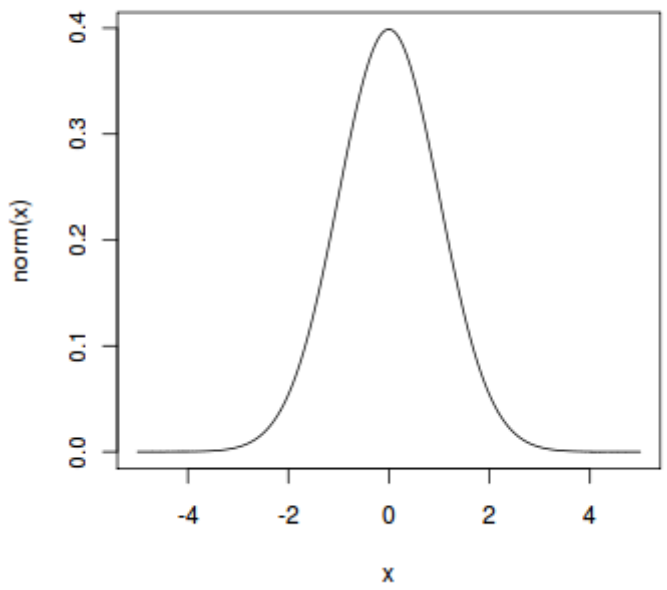
σ is the standard Deviation

Properties of a normal distribution

- The mean, mode and median are all equal.
- The curve is symmetric at the center (i.e. around the mean, μ).
- Exactly half of the values are to the left of center and exactly half the values are to the right.
- The total area under the curve is 1.

Standard Normal Distribution

In particular, the normal distribution with μ = 0 and σ = 1 is called the standard normal distribution, and is denoted as N (0,1). It can be graphed as follows.



Examples of data which follows a Normal Distribution:

- heights of people
- size of things produced by machines
- errors in measurements
- blood pressure
- marks on a test

R has four in built functions to generate normal distribution. They are described below.

```
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

Following is the description of the parameters used in above functions −

x is a vector of numbers.

p is a vector of probabilities.

n is number of observations(sample size).

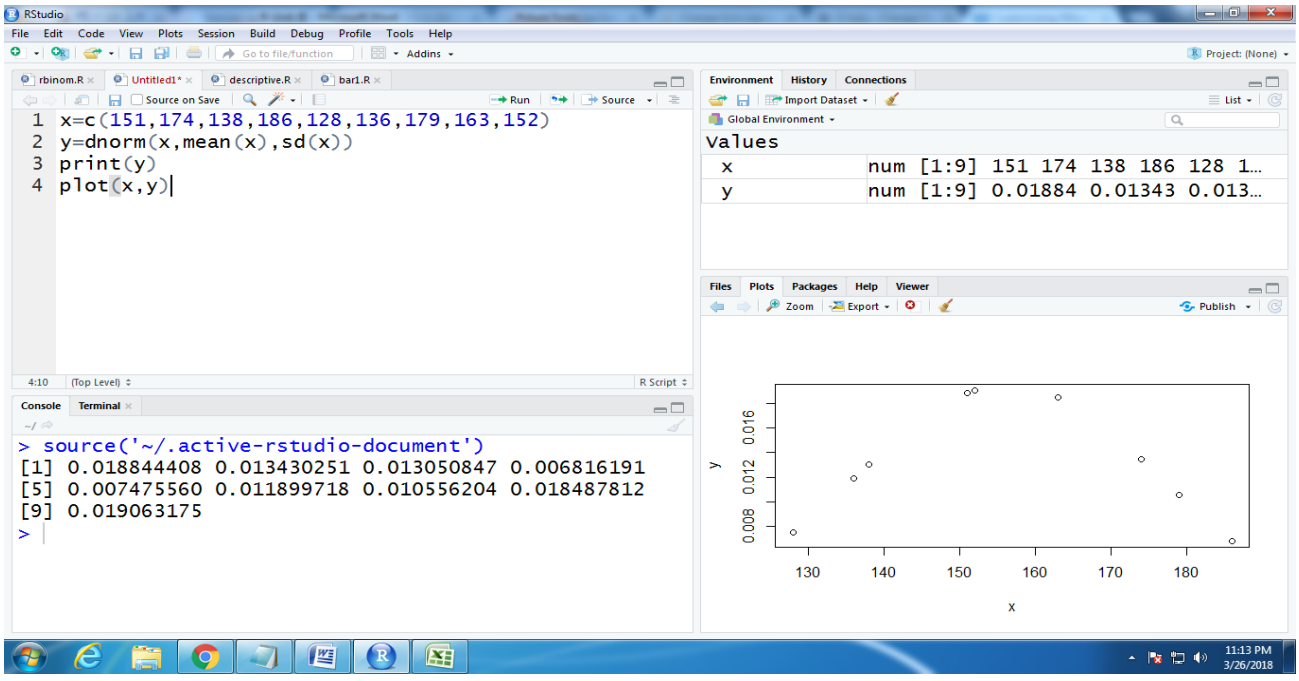
mean is the mean value of the sample data. It's default value is zero.

sd is the standard deviation. It's default value is 1.

dnorm()

This function gives height of the probability distribution at each point for a given mean and standard deviation.

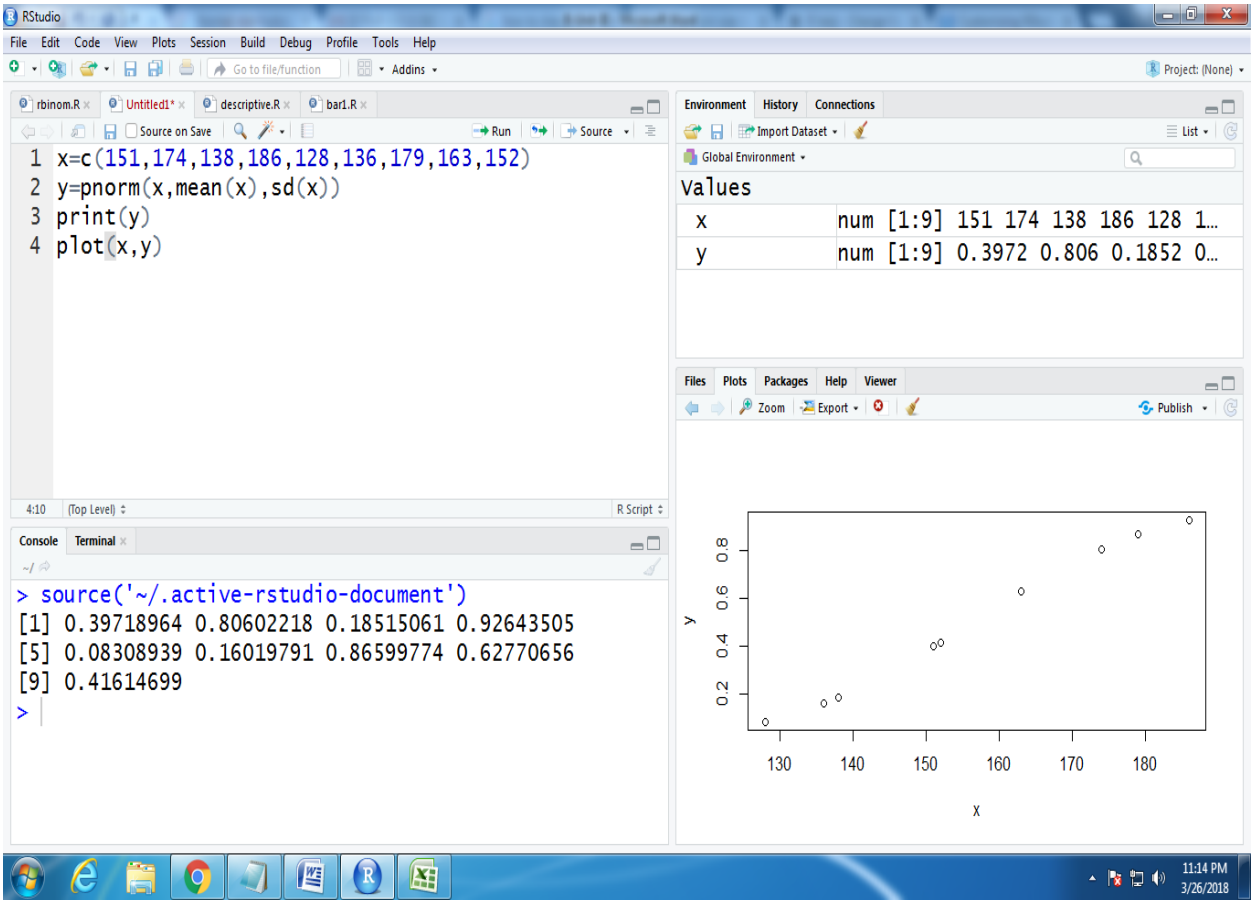
Example and Output:



pnorm()

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

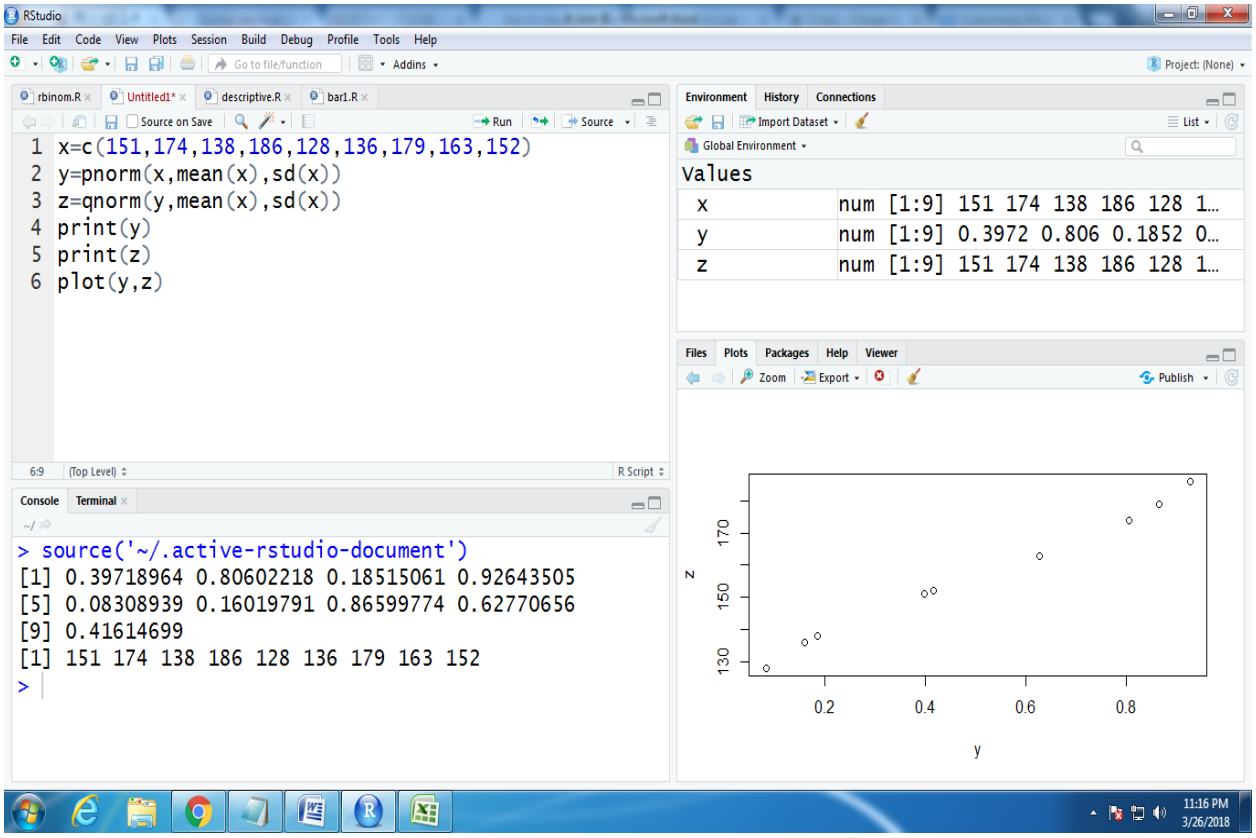
Example and Output:



qnorm()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

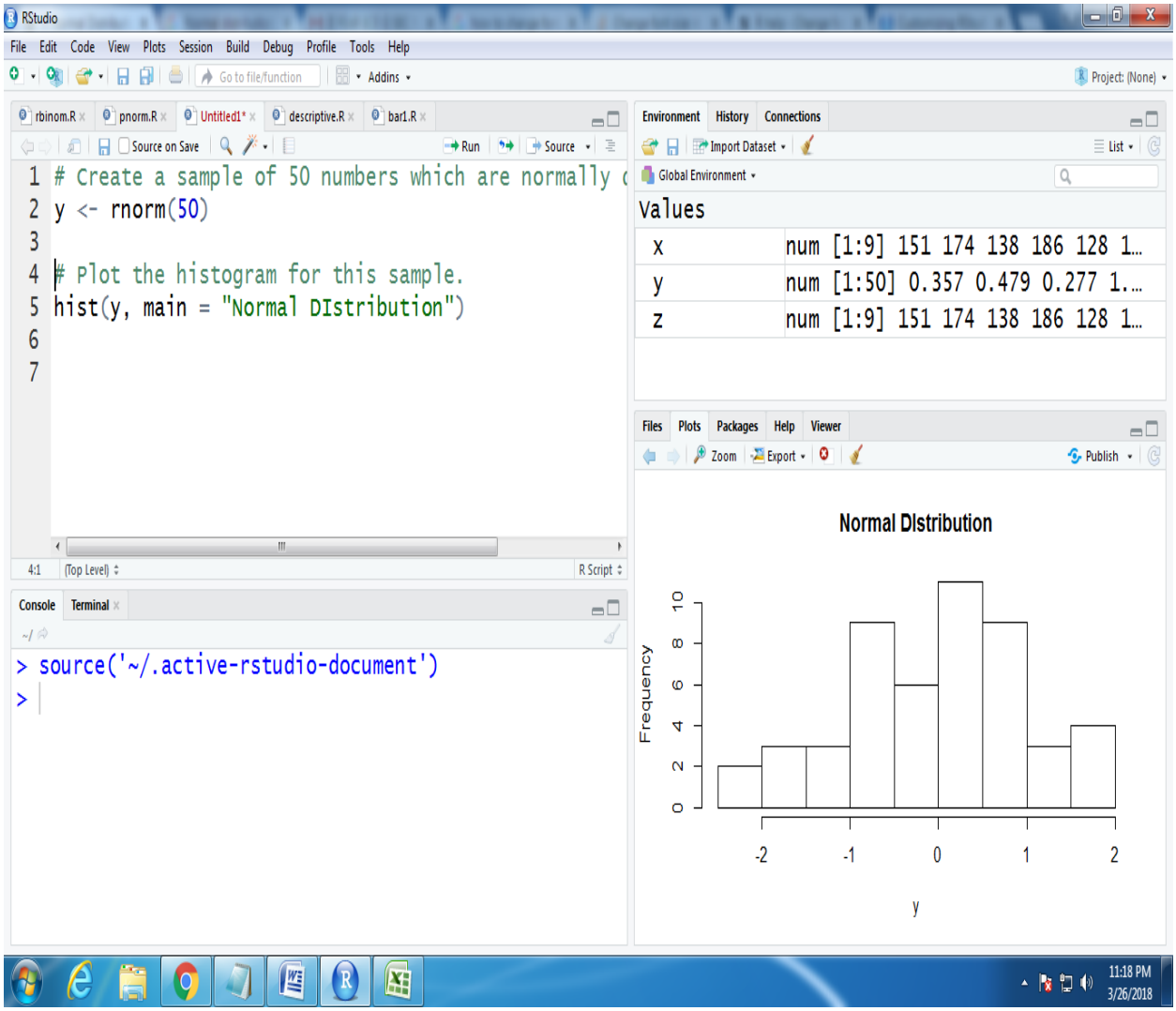
Example and Output:



morm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

Example and Output:



BINOMIAL DISTRIBUTION

A **binomial distribution** can be thought of as simply the probability of a SUCCESS or FAILURE outcome in an experiment or survey that is repeated multiple times. The binomial is a type of distribution that has **two possible outcomes** (the prefix "bi" means two, or twice). For example, a coin toss has only two possible outcomes: heads or tails and taking a test could have two possible outcomes: pass or fail.

Characteristics of Binomial Distribution

The number of observations or trials is fixed. In other words, you can only figure out the probability of something happening if you do it a certain number of times. This is common sense — if you toss a coin once, your probability of getting a tails is 50%. If you toss a coin a 20 times, your probability of getting a tails is very, very close to 100%.

Each observation or trial is independent. In other words, none of your trials have an effect on the probability of the next trial.

The **probability of success** (tails, heads, fail or pass) is exactly the same from one trial to another.

The binomial distribution formula is:

$b(x; n, P) = nCx * Px * (1 – P)n – x$

Where :

b=binomial probability

x=total number of "successes" (pass or fail, heads or tails etc.)

P=probability of a success on an individual trial

n= number of trials

Example:

A coin is tossed 10 times. What is the probability of getting exactly 6 heads?

$b(x; n, P) = {}_nC_x * P^x * (1 – P)^{n – x}$

The number of trials (n) = 10
The odds of success (p) = 0.5
q = 0.5
x = 6

$P(x=6) = {}_{10}C_6 * 0.5\text{^}6 * 0.5\text{^}4 = 210 * 0.015625 * 0.0625 = 0.205078125$

R has four in-built functions to generate binomial distribution. They are described below.

```
dbinom(x, size, prob)

pbinom(x, size, prob)

qbinom(p, size, prob)

rbinom(n, size, prob)
```

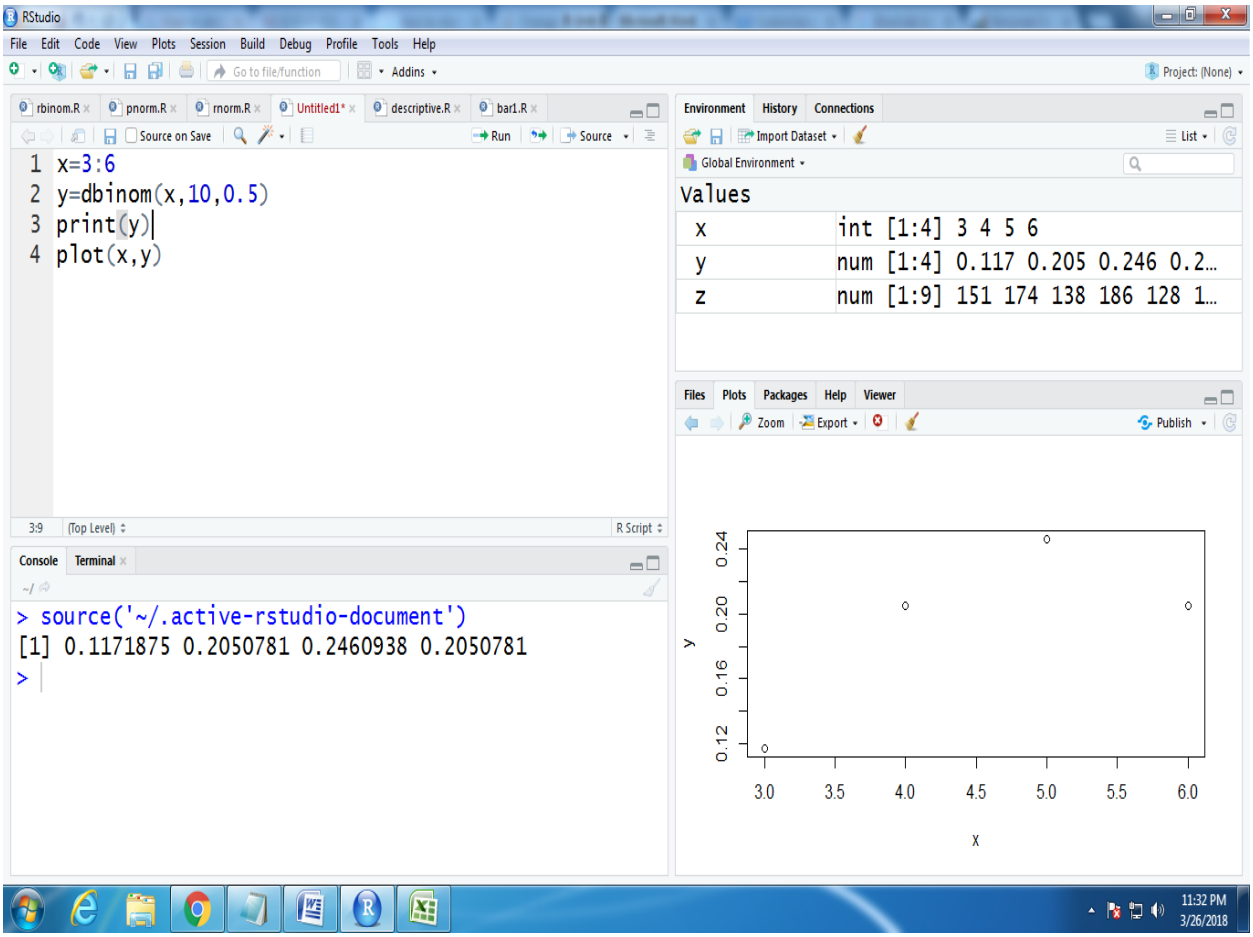
Following is the description of the parameters used −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations.
- **size** is the number of trials.
- **prob** is the probability of success of each trial.

dbinom()

This function gives the probability density distribution at each point.

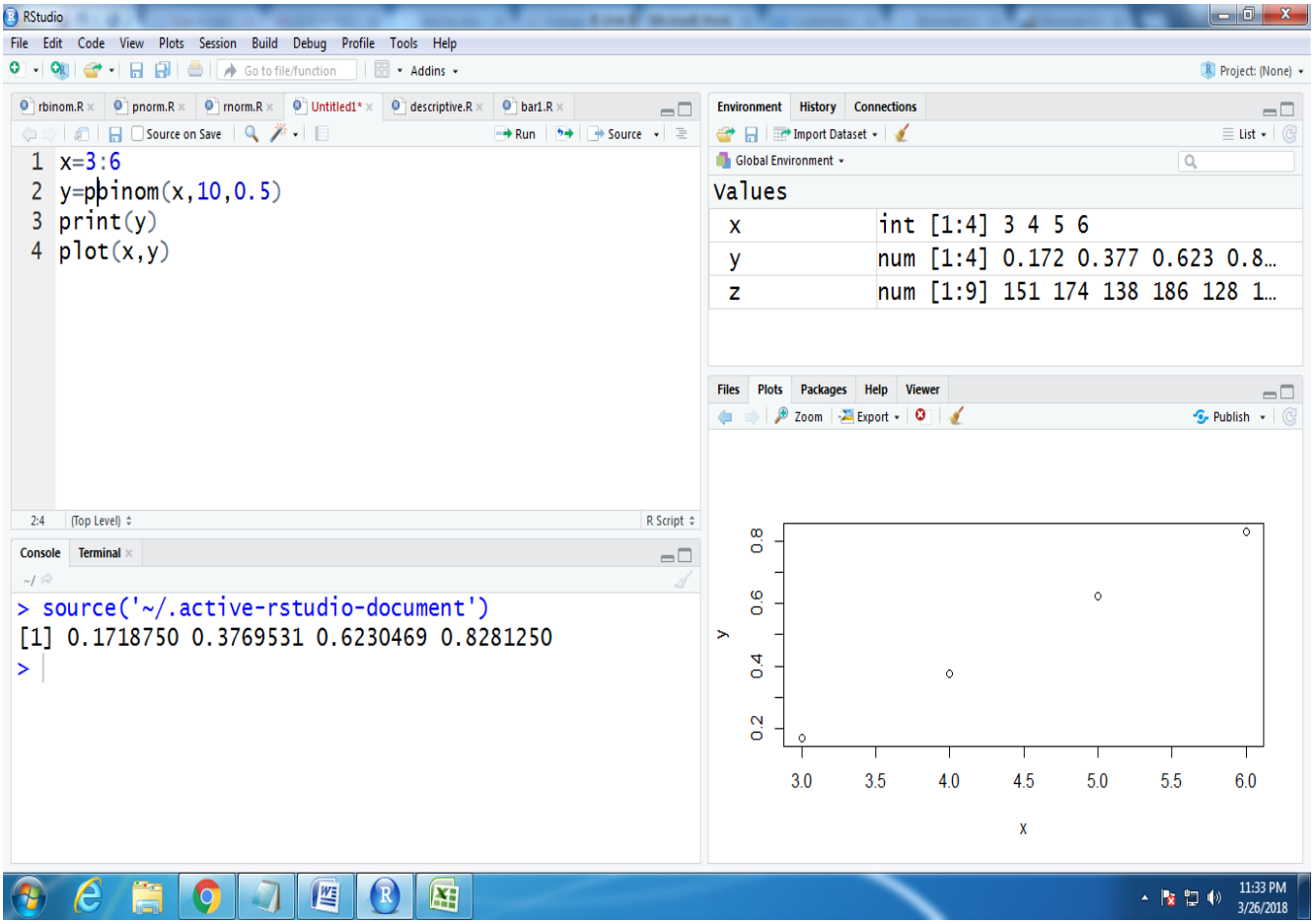
Example and Output:



pbinom()

This function gives the cumulative probability of an event. It is a single value representing the probability.

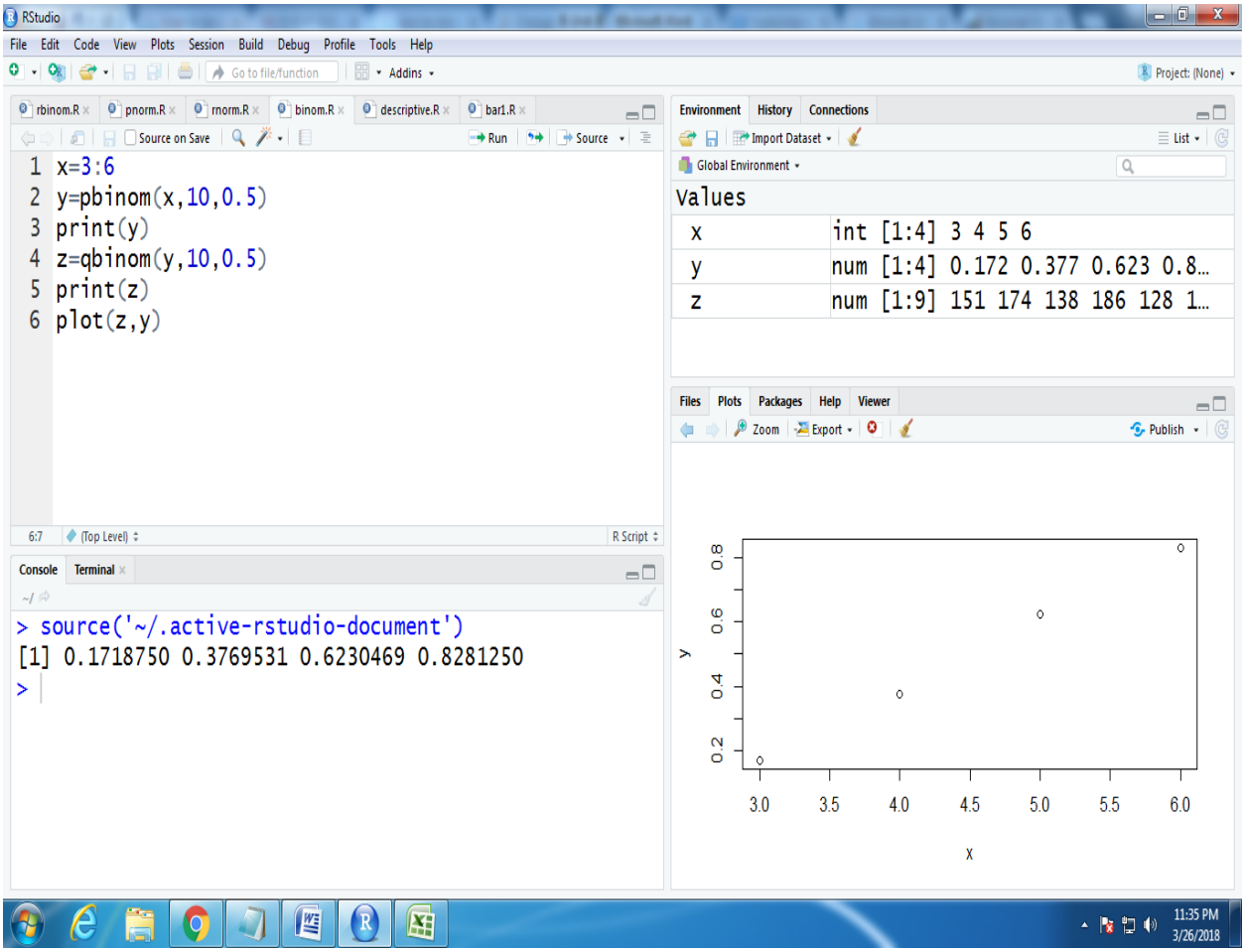
Example and Output:



qbinom()

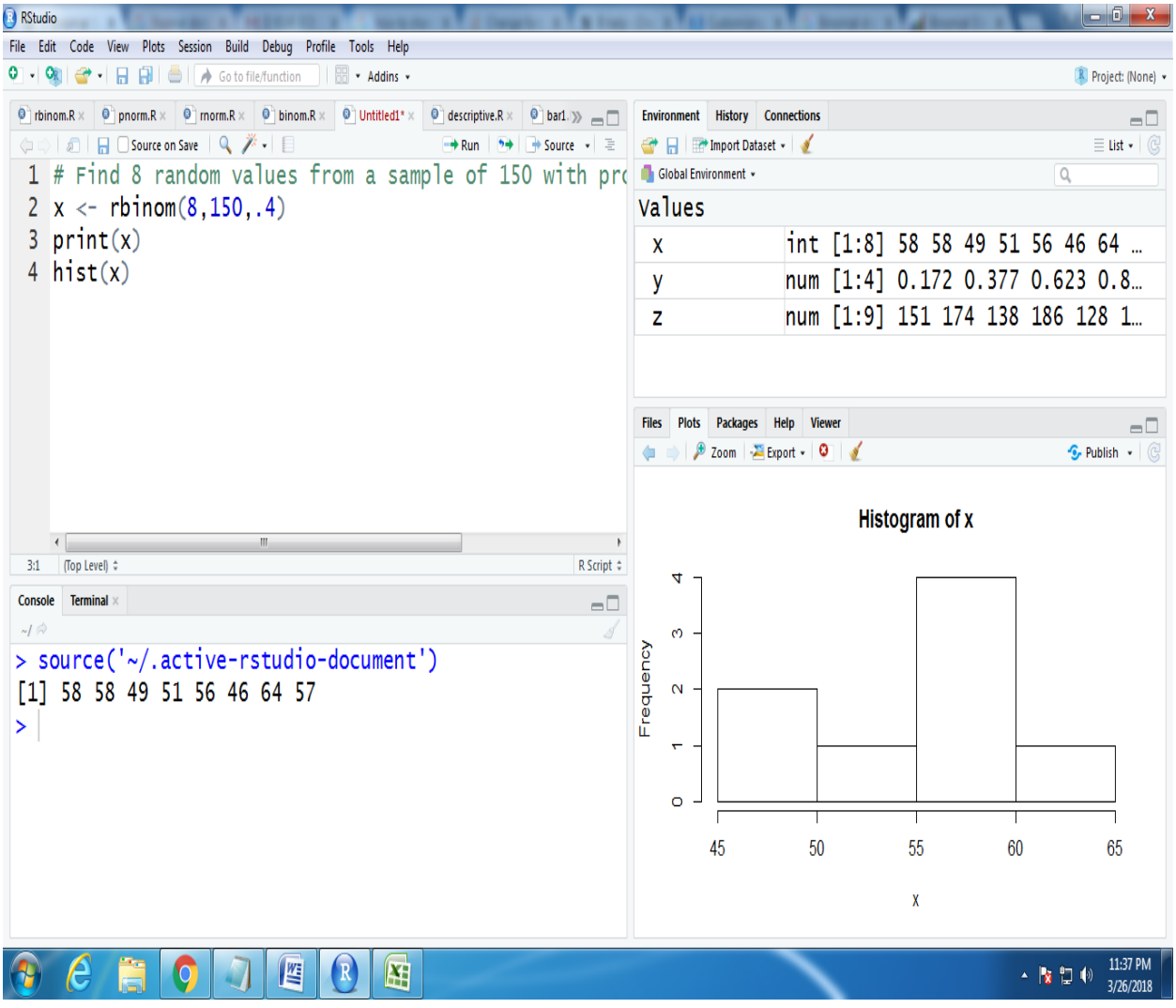
This function takes the probability value and gives a number whose cumulative value matches the probability value.

Example and output:



rbinom()

This function generates required number of random values of given probability from a given sample.



5. Time series Analysis

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called **time-series object**. It is also a R data object like a vector or data frame.

The time series object is created by using the **ts()** function.

Syntax

The basic syntax for **ts()** function in time series analysis is −

```
timeseries.object.name <- ts(data, start, end, frequency)
```

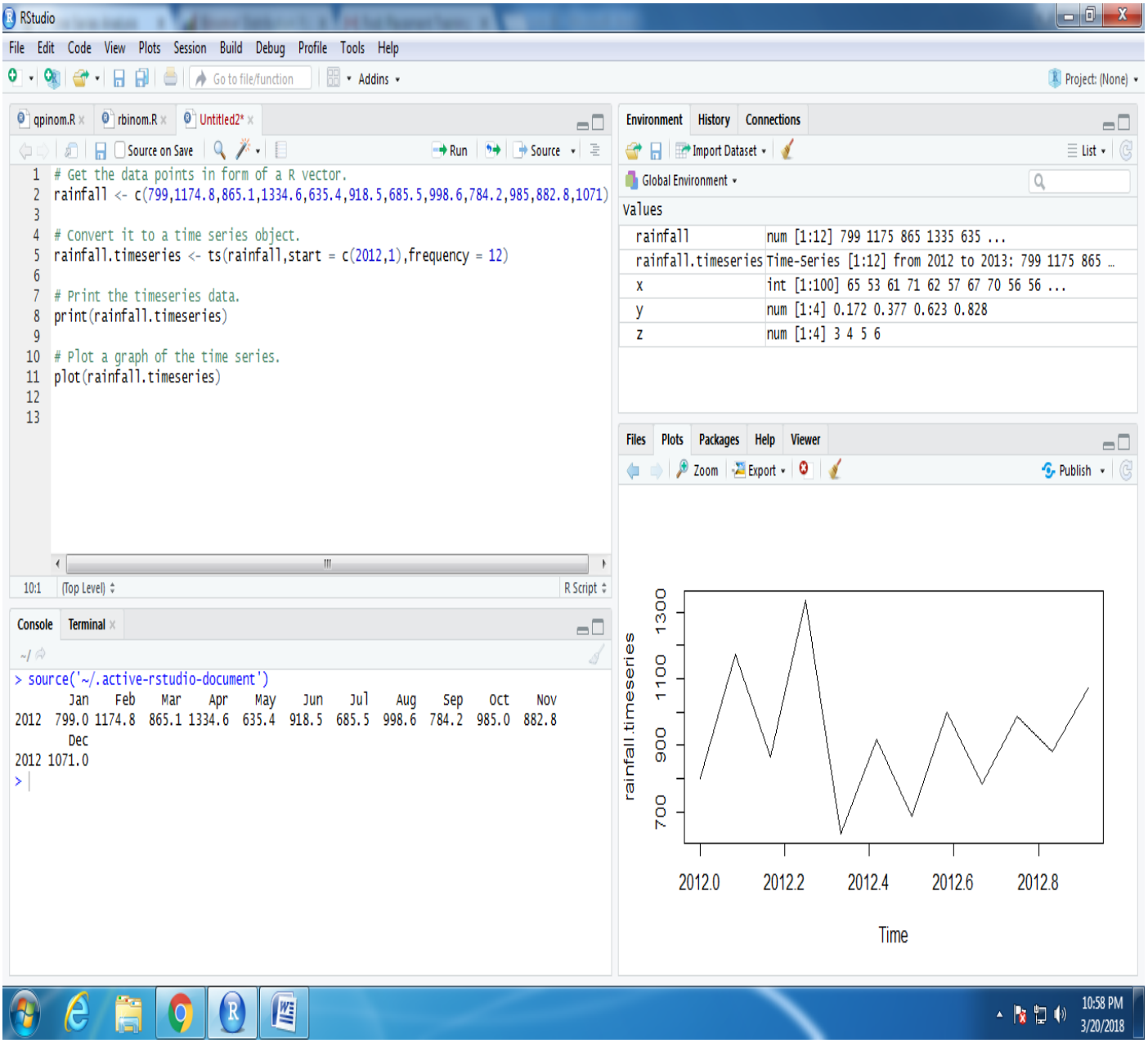
Following is the description of the parameters used −

- **data** is a vector or matrix containing the values used in the time series.
- **start** specifies the start time for the first observation in time series.
- **end** specifies the end time for the last observation in time series.
- **frequency** specifies the number of observations per unit time.

Except the parameter "data" all other parameters are optional.

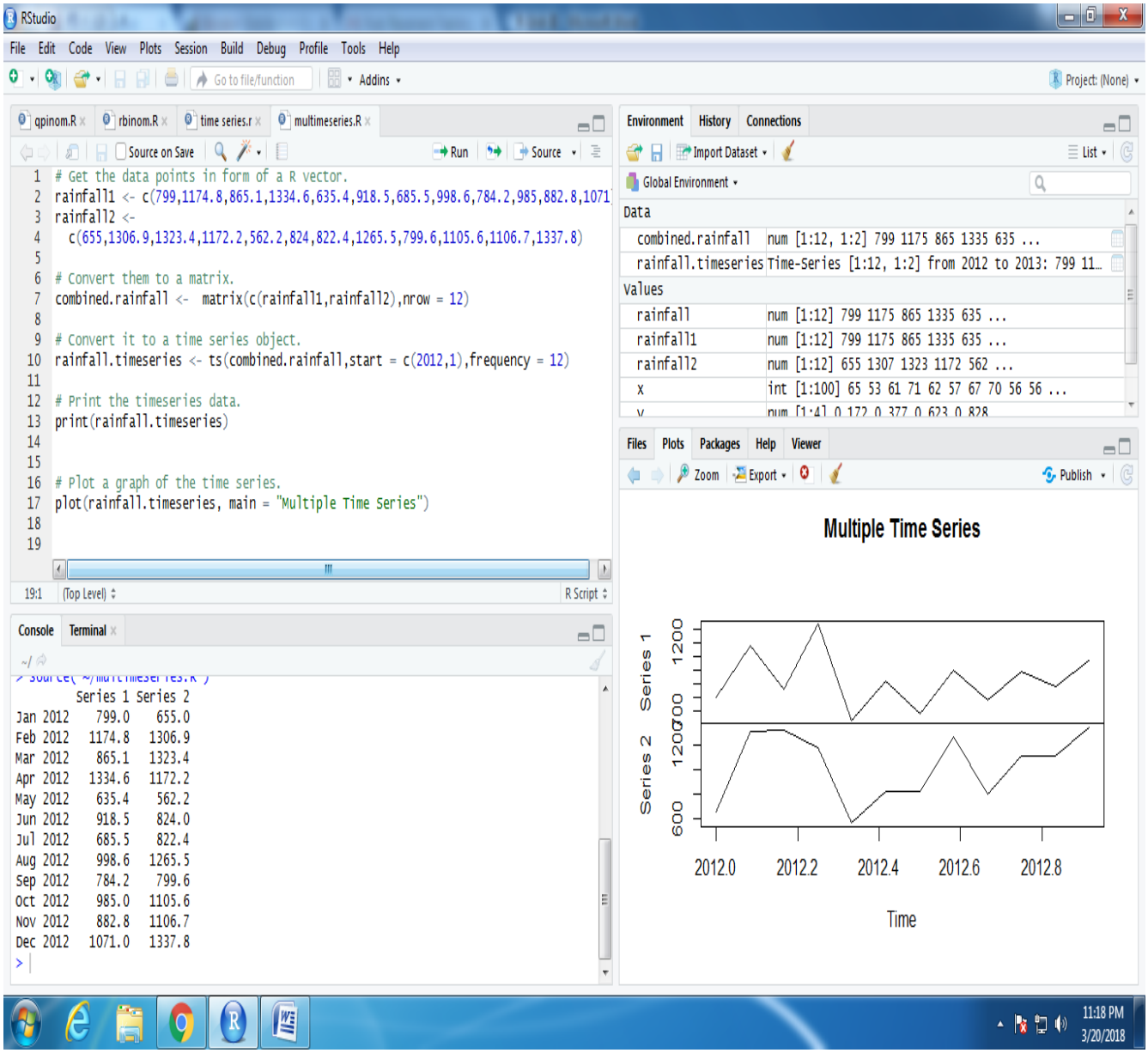
Example:

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
qpinom.R rbinom.R Untitled2*
1 # Get the data points in form of a R vector.
2 rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
4 # Convert it to a time series object.
5 rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
7 # Print the timeseries data.
8 print(rainfall.timeseries)
10 # Plot a graph of the time series.
11 plot(rainfall.timeseries)
Environment History Connections
Values
rainfall num [1:12] 799 1175 865 1335 635 ...
rainfall.timeseries Time-Series [1:12] from 2012 to 2013: 799 1175 865 ...
x int [1:100] 65 53 61 71 62 57 67 70 56 56 ...
y num [1:4] 0.172 0.377 0.623 0.828
z num [1:4] 3 4 5 6
Files Plots Packages Help Viewer
Console Terminal
> source('~/.active-rstudio-document')
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov
2012 799.0 1174.8 865.1 1334.6 635.4 918.5 685.5 998.6 784.2 985.0 882.8
Dec
2012 1071.0
rainfall.timeseries
700 900 1100 1300
2012.0 2012.2 2012.4 2012.6 2012.8
Time
10:58 PM
3/20/2018

Multiple Time Series

We can plot multiple time series in one chart by combining both the series into a matrix.



6. ANOVA

One Way Anova:

We are often interested in determining whether the means from more than two populations or groups are equal or not. To test whether the difference in means is statistically significant we can perform analysis of variance (ANOVA) using the R function aov(). If the ANOVA F-test shows there is a significant difference in means between the groups we may want to perform multiple comparisons between all pair-wise means to determine how they differ.

Analysis of Variance

The first step in our analysis is to graphically compare the means of the variable of interest across groups. It is possible to create side-by-side boxplots of measurements organized in groups using the function plot(). Simply type

plot(response ~ factor, data=data_name)

response -> name of the response variable

factor ->the variable that separates the data into groups.

Both variables should be contained in a data frame called data_name.

Ex. A drug company tested three formulations of a pain relief medicine for migraine headache sufferers. For the experiment 27 volunteers were selected and 9 were randomly assigned to one of three drug formulations. The subjects were instructed to take the drug during their next migraine headache episode and to report their pain on a scale of 1 to 10 (10 being most pain).

Drug A	4 5 4 3 2 4 3 4 4
Drug B	6 8 4 5 4 6 5 8 6
Drug C	6 7 6 6 7 5 6 5 5

To make side-by-side boxplots of the variable pain grouped by the variable drug we must first read in the data into the appropriate format.

```
> pain = c(4, 5, 4, 3, 2, 4, 3, 4, 4, 6, 8, 4, 5, 4, 6, 5, 8, 6, 6, 7, 6, 6, 7, 5, 6, 5, 5)
> drug = c(rep("A",9), rep("B",9), rep("C",9))
> migraine = data.frame(pain,drug)
```

Note the command rep("A",9) constructs a list of nine A‟s in a row. The variable drug is therefore a list of length 27 consisting of nine A‟s followed by nine B‟s followed by nine C‟s.

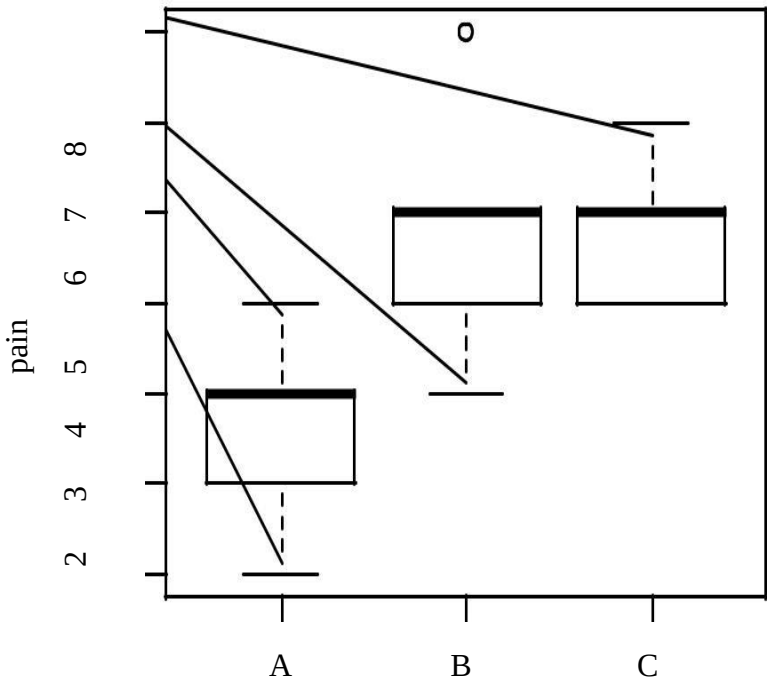
If we print the data frame migraine we can see the format the data should be on in order to make side-by-side boxplots and perform ANOVA (note the output is cut-off between observations 6-25 for space purposes).

```
> migraine
   pain drug
1     4   A
2     5   A
3     4   A
4     3   A
5     2   A
6     4   A
…

25    6   C
26    5   C
27    5   C
```

We can now make the boxplots by typing:

```
> plot(pain ~ drug, data=migraine)
```

The output of this program is shown below:



From the boxplots it appears that the mean pain for drug A is lower than that for drugs B and C.Next, the R function aov() can be used for fitting ANOVA models. The general form is

aov(response ~ factor, data=data_name)

Once the ANOVA model is fit, one can look at the results using the summary() function. This produces the standard ANOVA table.

Ex. Drug company example continued.

```
> results = aov(pain ~ drug, data=migraine)
> summary(results)
             Df Sum Sq  Mean Sq  F value   Pr(>F)
drug          2 28.222  14.1111  11.906 0.0002559 ***
Residuals 24    28.444   1.1852
---
Signif. codes:  0 „***“ 0.001 „**“ 0.01 „*“ 0.05 „.“ 0.1 „ “ 1
```

Studying the output of the ANOVA table above we see that the F-statistic is 11.91 with a p-value equal to 0.0003. We clearly reject the null hypothesis of equal means for all three drug groups.

B. Multiple comparisons

The ANOVA F-test answers the question whether there are significant differences in the K population means. However, it does not provide us with any information about how they differ. Therefore when you reject H0 in ANOVA, additional analyses are required to determine what is driving the difference in means. The function pairwise.t.test computes the pair-wise comparisons between group means with corrections for multiple testing. The general form is

```
pairwise.t.test(reponse, factor, p.adjust = method, alternative = c("two.sided", "less",
"greater"))
```

response ->is a vector of observations (the response variable)

factor -> a list of factors and

p.adjust ->is the correction method (e.g., “Bonferroni”).

Ex. Drug company example continued.

```
> pairwise.t.test(pain, drug, p.adjust="bonferroni")
```

Output:

```
     A       B
B 0.00119 -
C 0.00068 1.00000
```

The results state that the difference in means is not significantly different between drugs B and C (p-value = 1.00), but both are significantly different from drug A (p-values = 0.00119 and 0.00068, respectively). Hence, we can conclude that the mean pain is significantly different for drug A.

Another multiple comparisons procedure is Tukey‟s method (a.k.a. Tukey's Honest Significance Test). The function TukeyHSD() creates a set of confidence intervals on the differences between means with the specified family-wise probability of coverage. The general form is

```
TukeyHSD(x, conf.level = 0.95)
```

Here x is a fitted model object (e.g., an aov fit) and conf.level is the confidence level.

Ex. Drug company example continued.

```
> results = aov(pain ~ drug, data=migraine)
> TukeyHSD(results, conf.level = 0.95)

  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = pain ~ drug, data = migraine)

$drug
```

	diff	lwr	upr	p adj
B-A	2.1111111	0.8295028	3.392719	0.0011107
C-A	2.2222222	0.9406139	3.503831	0.0006453
C-B	0.1111111	-1.1704972	1.392719	0.9745173

These results show that the B-A and C-A differences are significant (p=0.0011 and p=0.00065, respectively), while the C-B difference is not (p=0.97). This confirms the results obtained using Bonferroni correction.