**SET-1**

**(a) Basic data structures and constructs:**

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values.

R’s base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they’re homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the six data types which are most frequently utilized in data analysis.

The most essential data structures used in R include:

* Vectors
* Lists
* Data-frames
* Matrices
* Arrays
* Factors

1. **Vectors:** A vector is an ordered

of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures.

# R program to illustrate Vector

# Vectors(ordered collection of same data type)

X **=** c(1, 3, 5, 7, 8)

# Printing those elements in console

print(X)

Output: [1] 1 3 5 7 8

1. **Lists:** A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous data structures. These are also one-dimensional data structures. A list can be a list of vectors, list of matrices, a list of characters and a list of functions and so on.

| R program to illustrate a List    # The first attributes is a numeric vector  # containing the employee IDs which is  # created using the 'c' command here  empId **=** c(1, 2, 3, 4)    # The second attribute is the employee name  # which is created using this line of code here  # which is the character vector  empName **=** c("Debi", "Sandeep", "Subham", "Shiba")    # The third attribute is the number of employees  # which is a single numeric variable.  numberOfEmp **=** 4    # We can combine all these three different  # data types into a list  # containing the details of employees  # which can be done using a list command  empList **=** list(empId, empName, numberOfEmp)    print(empList) |
| --- |

**Output:**

[[1]]

[1] 1 2 3 4

[[2]]

[1] "Debi" "Sandeep" "Subham" "Shiba"

[[3]]

[1] 4

**III Data frames:** Data frames are generic data objects of R which are used to store the tabular data. Data frames are the foremost popular data objects in R programming because we are comfortable in seeing the data within the tabular form. They are two-dimensional, heterogeneous data structures. These are lists of vectors of equal lengths.

Data frames have the following constraints placed upon them:

* A data-frame must have column names and every row should have a unique name.
* Each column must have the identical number of items.
* Each item in a single column must be of the same data type.
* Different columns may have different data types.

To create a data frame we use the data.frame() function.

**Example:**

| # R program to illustrate dataframe    # A vector which is a character vector  Name **=** c("Amiya", "Raj", "Asish")    # A vector which is a character vector  Language **=** c("R", "Python", "Java")    # A vector which is a numeric vector  Age **=** c(22, 25, 45)    # To create dataframe use data.frame command  # and then pass each of the vectors  # we have created as arguments  # to the function data.frame()  df **=** data.frame(Name, Language, Age)    print(df) |
| --- |

**Output:**

Name Language Age

1 Amiya R 22

2 Raj Python 25

3 Asish Java 45

**IV** Matrices: A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are two-dimensional, homogeneous data structures.  
Now, let’s see how to create a matrix in R. To create a matrix in R you need to use the function called matrix. The arguments to this matrix() are the set of elements in the vector. You have to pass how many numbers of rows and how many numbers of columns you want to have in your matrix and this is the important point you have to remember that by default, matrices are in column-wise order.

**Example:**

* Python3

| # R program to illustrate a matrix    A **=** matrix(      # Taking sequence of elements      c(1, 2, 3, 4, 5, 6, 7, 8, 9),        # No of rows and columns      nrow **=** 3, ncol **=** 3,        # By default matrices are      # in column-wise order      # So this parameter decides      # how to arrange the matrix      byrow **=** TRUE  )    print(A) |
| --- |

**Output:**

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 9

**V Arrays:**

Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dimensional data structures. For example, if we create an array of dimensions (2, 3,3) then it creates 3 rectangular matrices each with 2 rows and 3 columns. They are homogeneous data structures.

Now, let’s see how to create arrays in R. To create an array in R you need to use the function called array(). The arguments to this array() are the set of elements in vectors and you have to pass a vector containing the dimensions of the array.

**Example:**

* Python3

| # R program to illustrate an array    A **=** array(      # Taking sequence of elements      c(1, 2, 3, 4, 5, 6, 7, 8),        # Creating two rectangular matrices      # each with two rows and two columns      dim **=** c(2, 2, 2)  )    print(A) |
| --- |

**Output:**

, , 1

**[,1] [,2]**

**[1,]**  1 3

**[2,]** 2 4

, , 2

[,1] [,2]

[1,] 5 7

[2,] 6 8

**VI Factors**

Factors are the data objects which are used to categorize the data and store it as levels. They are useful for storing categorical data. They can store both strings and integers. They are useful to categorize unique values in columns like “TRUE” or “FALSE”, or “MALE” or “FEMALE”, etc.. They are useful in data analysis for statistical modeling.

Now, let’s see how to create factors in R. To create a factor in R you need to use the function called factor(). The argument to this factor() is the vector.

**Example:**

* Python3

| # R program to illustrate factors    # Creating factor using factor()  fac **=** factor(c("Male", "Female", "Male",                 "Male", "Female", "Male", "Female"))    print(fac) |
| --- |

**Output:**

[1] Male Female Male Male Female Male Female

Levels: Female Male

**(b) Available R Datasets:**

R has numerous datasets that are built-in and these datasets are available in a R package called “[R Datasets Package](https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html)“. This is maintained by R Core team and available with base installation of R. We can find the list of built-in datasets readily available in R using R function data().

Most used data sets in ‘R’

(i) mtcars: Motor Trend Car Road Tests

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models)

(ii) iris

**iris** data set gives the measurements in centimeters of the variables sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

(iii) ToothGrowth

ToothGrowth data set contains the result from an experiment studying the effect of vitamin C on tooth growth in 60 Guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, (orange juice or ascorbic acid (a form of vitamin C and coded as VC).

(iv) PlantGrowth

Results obtained from an experiment to compare yields (as measured by dried weight of plants) obtained under a control and two different treatment condition.

**Now Let’s learn the command to display the data from a particular data set** iris

#Loading

data("iris")

#for printing

**head**(iris)

# Number of rows (observations)

nrow(iris)

# Number of columns (variables)

ncol(mtcars)

If you want to learn more about *iris* type this:

?iris

(c) Null, NA, Missing Values:

NULL is an object and is returned when an expression or function results in an undefined value. In R language, NULL (capital letters) is a reserved word and can also be the product of importing data with unknown data type.

NA is a logical constant of length 1 and is an indicator for a missing value.NA (capital letters) is a reserved word and can be coerced to any other data type vector (except raw) and can also be a product when importing data. NA and “NA” (as presented as string) are not interchangeable. NA stands for Not Available.

(d) Basic Packages related to Statistics: e. g. stats, **stats4,** graphics, grDevices

* + One of the strengths of R is that the system can easily be extended. The system allows you to write new functions and package those functions in a so called `R package' (or `R library'). The R package may also contain other R objects, for example data sets or documentation. There is a lively R user community and many R packages have been written and made available on CRAN for other users. Just a few examples, there are packages for portfolio optimization, drawing maps, exporting objects to html, time series analysis, spatial statistics and the list goes on and on.
  + When you download R, already a number (around 30) of packages are downloaded as well. To use a function in an R package, that package has to be attached to the system. When you start R not all of the downloaded packages are attached, only seven packages are attached to the system by default. You can use the function search to see a list of packages that are currently attached to the system, this list is also called the search path.

> search()

[1] ".GlobalEnv" "package:stats" "package:graphics"

[4] "package:grDevices" "package:datasets" "package:utils"

[7] "package:methods" "Autoloads" "package:base"

SET-2

**Descriptive Statistics:**

(a) Compute Mean, Median, Quartiles, Percentile (use quantile() function)

**(i) Mean:**

**Command** mean()

The mean of a set of observations is an average. You simply add up all the values and then divide by the total number of values. If we were to compute the mean for cars$speed (or the variable speed our dataset called cars) we would simply sum the values in the column for speed and divide by 50.

Use the command mean(cars$speed)

**(ii) Median**

**Command** median()

Use the command median(cars$speed)

**(iii) Quartiles:**

**To find all Quartiles we use the command quantile()**

For eg. quantile(cars$speed)

## 0% 25% 50% 75% 100%

## 4 12 15 19 25

Or just to see the 25% (Q1) and 75% (Q3) we can type:

quantile(cars$speed, probs=c(.25, .75))

## 25% 75%

**(iv) Percentile : quantile() functions by default gives** 0% 25% 50% 75% 100% values. For specific percentile values, we can modified same command as follows:

how to find percentiles in r - quantile in r

> x = c(5,10,12,15,20,24,27,30,35)

> quantile(x, probs = c(0.125,0.375,0.625,0.875))

12.5% 37.5% 62.5% 87.5%

10 15 24 30

(b) Variance, Standard Deviation, IQR, Minimum & Maximum Values

**(i) Variance:** To compute the sample variance in R we would type the following:

**Command var()**

Eg var(cars$speed)

## [1] 27.95918

**(ii) Standard Deviation**: The square root of the variance is the standard deviation. Below is the formula for the sample standard deviation.

To compute the sample standard deviation in R, type the following:

sqrt(var(cars$speed))

## [1] 5.287644or you can use the sd function

OR

sd(cars$speed)

## [1] 5.287644

**(iii) IQR:** We can calculate the interquartile range (IQR) using IQR. This is the range spanned by the middle half of the data. For example this is the 75th quantile minus the 25th quantile.

**Command:** IQR()

IQR(cars$speed)

**(iv) Minimum & Maximum Values**

min(cars$speed)

## [1] 4

max(cars$speed)

## [1] 25

R has an even better function, range that outputs the minimum and maximum value in a vector

range(cars$speed)

## [1] 4 25

(b) Summary Statistics and its interpretation:

* There’s an easier way to compute some measures of central tendency and variability using the summary( ) function. The summary function provides the min( ), max( ), median( ), mean( ), the 75% and 25% quantiles. To compute all these measures for a single variable type:
* Syntax-summary(column name of a dataset/vector/dataframe)
* **summary**(cars$speed)
* ## Min. 1st Qu. Median Mean 3rd Qu. Max.
* ## 4.0 12.0 15.0 15.4 19.0 25.0
* **summary**(cars)
* ## speed dist
* ## Min. : 4.0 Min. : 2.00
* ## 1st Qu.:12.0 1st Qu.: 26.00
* ## Median :15.0 Median : 36.00
* ## Mean :15.4 Mean : 42.98
* ## 3rd Qu.:19.0 3rd Qu.: 56.00
* ## Max. :25.0 Max. :120.00

(c) Histogram, Scatter Plot, Box Plot, Density Plot of R data sets and interpretation:

(i**) Histogram**: The simplest display for the shape of a distribution of data can be done using a histogram- a count of how many observations fall within specified divisions ("bins") of the x-axis.

> hist(airquality$Temp)

**(ii) Scatter Plot:** One very commonly used tool in exploratory analysis of multivariate data is the scatterplot. We will look at this in more detail later when we discuss regression and correlation. The R command for drawing a scatterplot of two variables is a simple command of the form "plot(x,y)."

> plot(airquality$Temp, airquality$Ozone)

**(iii) Box Plot**: A "boxplot", or "box-and-whiskers plot" is a graphical summary of a distribution; the box in the middle indicates "hinges" (close to the first and third quartiles) and median. The lines ("whiskers") show the largest or smallest observation that falls within a distance of 1.5 times the box size from the nearest hinge. If any observations fall farther away, the additional points are considered "extreme" values and are shown separately. A boxplot can often give a good idea of the data distribution, and is often more useful to compare distributions side-by-side, as it is more compact than a histogram.

> boxplot(airquality$Ozone)

**(iv) Density Plot:** The function **density**() is used to estimate kernel density.

# Compute the density data

dens <- density(mtcars$mpg)

# plot density

plot(dens, frame = FALSE, col = "steelblue", main = "Density plot of mpg")

(d) Generate Frequency Distribution of data as a data frame:

To create a frequency table in R, we can simply use table function but the output of table function returns a horizontal table. If we want to read the table in data frame format then we would need to read the table as a data frame using as.data.frame function. For example, if we have a table called T then to convert it into a data frame format we can use the command as.data.frame(T).

> x1<-rpois(200,2)

> x1

## **Output**

[1] 2 0 2 3 2 3 1 2 1 4 0 0 4 4 1 3 1 2 1 3 2 3 2 1 4 1 4 1 1 1 2 2 0 2 1 1 1

[38] 1 5 1 1 2 3 0 5 3 3 2 0 1 2 1 3 2 1 5 3 2 2 2 3 3 0 3 0 3 1 3 3 4 0 3 3 0

[75] 2 3 0 2 2 1 3 1 4 0 1 1 5 1 3 2 0 2 4 1 1 2 2 1 3 0 2 3 3 1 4 1 4 4 1 3 5

[112] 0 0 2 1 0 1 0 3 3 4 6 3 3 0 2 0 2 1 3 1 1 1 1 2 4 1 0 1 3 2 2 0 2 3 2 4 4

[149] 1 2 0 5 3 3 3 1 4 5 5 1 3 1 2 4 4 5 0 3 3 1 2 1 1 0 3 3 5 1 1 6 1 2 3 6 1

[186] 2 0 1 4 1 1 1 2 3 5 1 1 1 0 4

## **Example**

> table\_x1<-table(x1)

> table\_x1

## **Output**

x1

0 1 2 3 4 5 6

27 58 39 42 20 11 3

## **Example**

> df1<-as.data.frame(table\_x1)

> df1

## **Output**

  x1 Freq

1 0   27

2 1   58

3 2   39

4 3   42

5 4   20

6 5   11

7 6   3

(e) Compute Correlation Coefficient and Covariance:

[**Covariance**](https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/) and [**Correlation**](https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/) are terms used in statistics to measure relationships between two random variables. Both of these terms measure linear dependency between a pair of random variables or bivariate data.

In this article, we are going to discuss **cov()**, **cor()** and **cov2cor()** functions in [R](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) which use covariance and correlation methods of statistics and probability theory.

* Aside from looking at the characteristics of our variables, we may want to see if there is a relationship between our variables in our data, or a correlation.

**cor**(cars$speed, cars$dist)

## [1] 0.8068949

Correlation shows the strength of a relationship between two variables and is expressed numerically by the correlation coefficient.

The correlation coefficient's values range between -1.0 and 1.0.

A perfect [positive correlation](https://www.investopedia.com/terms/p/positive-correlation.asp) means that the correlation coefficient is exactly 1.

This implies that as one security moves, either up or down, the other security moves in lockstep, in the same direction.

A perfect [negative correlation](https://www.investopedia.com/terms/n/negative-correlation.asp) means that two assets move in opposite directions, while a zero correlation implies no linear relationship at all.

You can also use the cor( ) function to calculate a complete correlation matrix between all pairs of variables in the data frame.

Syntax-cor(dataframe name)

**cor**(cars)

## speed dist

## speed 1.0000000 0.8068949

## dist 0.8068949 1.0000000

**Covariance:**

***Syntax:****cov(x, y, method)*

***where,***

* ***x****and****y****represents the data vectors*
* ***method****defines the type of method to be used to compute covariance. Default is “pearson”.*
* Data vectors
* x <- c(1, 3, 5, 10)
* y <- c(2, 4, 6, 20)
* # Print covariance using different methods
* print(cov(x, y))
* print(cov(x, y, method = "pearson"))
* print(cov(x, y, method = "kendall"))
* print(cov(x, y, method = "spearman"))
* **Output:**
* [1] 30.66667
* [1] 30.66667
* [1] 12
* [1] 1.666667

**SET-2**

Applications of Descriptive Statistics

● Analysis of iris data set.

1. Determine the correlation coefficient of (Sepal Length, Sepal Width) and (Petal Length, Petal Width).

corr <- **cor**(iris[,1:4])

**round**(corr,3)

output:

## Sepal.Length Sepal.Width Petal.Length Petal.Width

## Sepal.Length 1.000 -0.118 0.872 0.818

## Sepal.Width -0.118 1.000 -0.428 -0.366

## Petal.Length 0.872 -0.428 1.000 0.963

## Petal.Width 0.818 -0.366 0.963 1.000

Interpret the result and draw conclusions using all the rows: +1 means variables are correlated, -1 inversely correlated.

Characterize the 3 varieties of iris flowers:

**library**(datasets)

**str**(iris)

Output:

## 'data.frame': 150 obs. of 5 variables:

## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

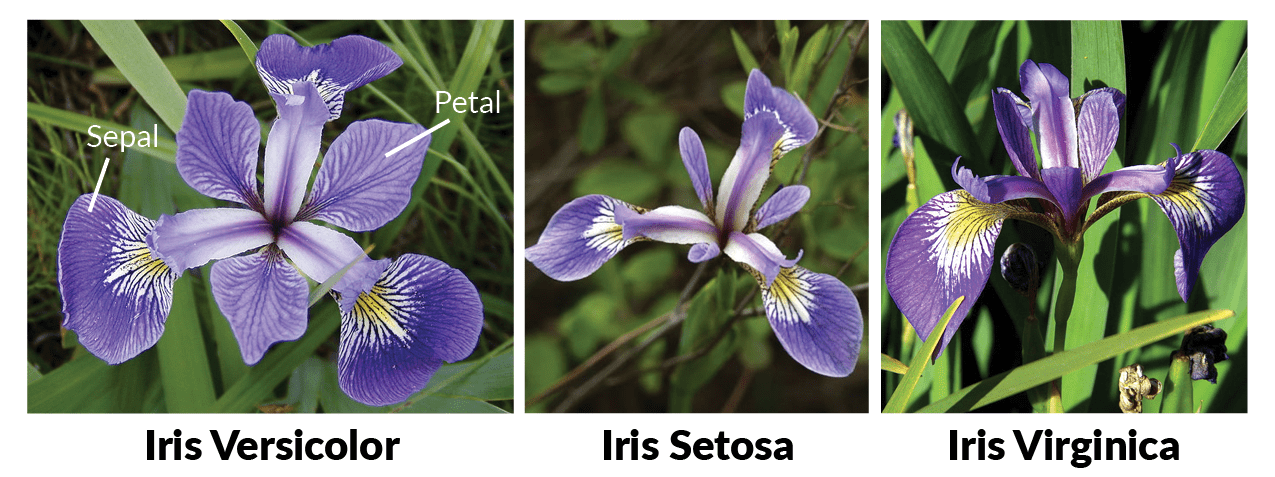
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

Determine the discriminating parameters for 3 varieties of iris flower:

Setosa, Vercicolor, Veginica



**summary**(iris)

# Sepal.Length Sepal.Width Petal.Length Petal.Width

## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100

## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300

## Median :5.800 Median :3.000 Median :4.350 Median :1.300

## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199

## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800

## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500

## Species

## setosa :50

## versicolor:50

## virginica :50

● Analysis of faithful data set to answer the following questions:

What is the minimum waiting time for a visitor to observe an eruption?

What will be the likely eruption duration?

What is the maximum duration of eruption?

If I want to see the eruption of maximum duration, how long will I have to wait?

**SET-3**

Probability and Probability Distributions

Many of the standard probability distributions have functions in R to calculate:

* probability density/mass function (d)
* cumulative distribution function (p)
* quantile function (q)

 R has in-built functions for almost all probability distributions (see below)

# Get the list of all probability

# distribution related functions

**help**(distributions)

All of these functions for probability distribution follows the same common scheme, the **root name** of the function prefixed by either of **p**, **d**, **q** and **r**. For example for the Normal distribution we have the four variants of function available - **pnorm**, **dnorm**, **qnorm** and **rnorm**.

| **Function type (prefix)** | **Meaning** | |
| --- | --- | --- |
| p- | (probability) gives the cumulative distribution function (cdf) |  |
| d- | (density) gives the probability density function (pdf) |  |
| q- | (quantiles) gives the inverse cumulative distribution function |  |
| r- | (random) returns randomly generated numbers for the distribution |  |

(a) Use pnorm(), pbinom(), ppois(), pexp() functions to compute probabilitiy

(b) Use dnorm(), dbinom(), dpois(), dexp() functions to compute probability density functions

(c) Use qnorm(), qbinom(), qpois, qexp() functions to get x value corresponding to given probability value

(d) Use different parameter values in 3 (a), and 3 (b) to observe the impact of different parameter values and prepare a note on that.

**Discrete distributions**

If the support of the random variable is a finite or countably infinite number of values, then the random variable is discrete. Discrete random variables have:

1. The [probability mass function (pmf)](https://en.wikipedia.org/wiki/Probability_mass_function): This pmf gives the probability that a random variable will take on each value in its support.
2. The [cumulative distribution function (cdf)](https://en.wikipedia.org/wiki/Cumulative_distribution_function) provides the probability the random variable is less than or equal to a particular value.
3. The quantile function is the inverse of the cumulative distribution function, i.e. you provide a probability and the quantile function returns the value of the random variable such that the cdf will return that probability.

**Binomial distribution:**

Whenever you repeat an experiment and you assume

1. the probability of success is the same each time and
2. each trial of the experiment is independent of the rest (as long as you know the probability of success).

Then, if you record the number of successes out of the total, you have a [binomial distribution](https://en.wikipedia.org/wiki/Binomial_distribution).

We can use the pmf to calculate the probability of a particular outcome of the experiment. For example, what is the probability of seeing 6 successes? We can use the dbinom function.

n <- 13

p <- 0.7

dbinom(6, size = n, prob = p)

## [1] 0.0441524

The entire pmf is

x <- 0:n

plot(x, dbinom(x, size = n, prob = p), main = "Probability mass function for Bin(13,0.7)")

If we want to calculate the probability of observing an outcome less than or equal to a particular value, we can use the cumulative distribution function. For example, what is the probability of observing 9 or fewer successes?

pbinom(9, size = n, prob = p)

## [1] 0.5793944

Here is the entire cdf.

plot(x, pbinom(x, size = n, prob = p), type="s", main = "Cumulative distribution function for Bin(13,0.7)")

**Poisson distribution**

While the binomial distribution has an upper limit n, we sometimes run an experiment and are counting *successes* without any **technical** upper limit. These experiments are usually run for some amount of time or over some amount of space or both. For example,

* the number of photos observed by a detector in a minute
* the number of times an a/c unit comes on in an hour
* the number of buildings in a square mile
* the number of transistors on a circuit board

When there is no technical upper limit (even if the probability of more is extremely small), then a [Poisson distribution](https://en.wikipedia.org/wiki/Poisson_distribution) can be used. The Poisson distribution has a single parameter, the rate that describes, on average, how many of the *things* are expected to be observed parameter.

Suppose we record the number of network failures in a day and on average we see 2 failures per day. The number of network failures in a day has no upper limit, so we’ll use the Poisson distribution.

Here is the pmf:

rate <- 2

x <- 0:10 *# with no upper limit we need to decide on an upper limit*

plot(x, dpois(x, lambda = rate), main = "Probability mass function for Po(2)")

Here is the cdf:

plot(x, ppois(x, lambda = rate), type="s", main = "Cumulative distribution function for Po(2)")

And random draws

draws <- rpois(100, lambda = rate)

hist(draws, breaks = (0:(max(draws)+1)) - 0.5, probability = TRUE, main = "Random draws from Po(2)")

points(x, dpois(x, lambda = rate), col="red")

| **Distribution** | **Density function: *P*(*X* = *x*)** | **Distribution function: *P*(*X* ≤ *x*)** |
| --- | --- | --- |
| Binomial | dbinom(\*x\*, \*size\*, \*prob\*) | pbinom(\*x\*, \*size\*, \*prob\*) |
| Poisson | dpois(\*x\*, \*lambda\*) | ppois(\*x\*, \*lambda\*) |

**Continuous distributions**

In contrast to discrete random variables, [continuous random variables](http://stattrek.com/probability-distributions/discrete-continuous.aspx?Tutorial=Stat) can take on an uncountably infinite number of values. The easiest way for this to happen is that the random variable can take on any value between two specified values (and infinity counts), i.e. an interval.

Continuous random variables have a [probability density function (pdf)](https://en.wikipedia.org/wiki/Probability_density_function) instead of a pmf. When integrated from a to b, this pdf gives the probability the random variable will take on a value between a and b. Continuous random variables still have a cdf, quantile function, and random generator that all still have the same interpretation.

| **Distribution** | **Distribution function: *P*(*X* ≤ *x*)** |
| --- | --- |
| Normal | pnorm(\*x\*, \*mean\*, \*sd\*) |
| Exponential | pexp(\*x\*, \*rate\*) |

**Normal distribution**

The most important distribution for this course is the [normal (Gaussian) distribution](https://en.wikipedia.org/wiki/Normal_distribution). The normal distribution has two parameters: the mean (*μ*) and the variance (*σ2*) and we write *X∼N(μ,σ2)*. The probability density function for the normal distribution is the well-known bell-shaped curve.

mu <- 0

sigma <- 1 *# standard deviation*

curve(dnorm(x, mean = mu, sd = sigma), *# notice the 3rd argument is the sd*

from = -4, to = 4,

main = "PDF for a standard normal")

The quantile function will be used later in the semester to help construct confidence/credible intervals.

curve(qnorm(x, mean = mu, sd = sigma),

from = 0, to = 1,

main = "Quantile function for a standard normal")

Likewise, we can use pexp to calculate the probability that an exponential variable with a mean of 40 could be less than 20:

**pexp**(20, rate = 1 / 40)

*#> [1] 0.393*

**Applications of Descriptive Statistics:**