Bentley University MA346 in R

Content extracted from the How to Data Website

This PDF generated on 18 January 2024

Contents

MA346 is an undergraduate data science course at Bentley University. The description from the course catalog can be found here. Topics included in the course are listed as tasks (on website) below.

Mathematical topics include functions and relations, a review of basic statistics, and (time permitting) networks, matrices, and an introduction to supervised learning.

Computing topics include Jupyter notebooks (locally and in the cloud), Python and pandas, abstraction, concatenation and merging, map-reduce, split-apply-combine, data munging, version control, and dashboards.

Communication topics include best practices for writing reports, documenting code and computational notebooks, and data visualization.

This page is not yet complete. More content will be added here over time.

Content last modified on 03 August 2023.

How to do basic mathematical computations

Description

How do we write the most common mathematical operations in a given piece of software? For example, how do we write multiplication, or exponentiation, or logarithms, in Python vs. R vs. Excel, and so on?

Solution in pure R

For those expressions that need the Python math package, use the code `import math` beforehand to ensure that package is loaded. Alternatively, you can write `from math import *` and thus drop the `math` prefixes in the table below.

Mathematical notation	R code
$x+y$	`x+y`
$x-y$	`x-y`
xy	`x*y`
$\frac{x}{y}$	`x/y`
x^y	`x^y`
$\|x\|$	`abs(x)`
$\ln x$	`log(x)`
$\log_a b$	`log(b,a)`
e^x	`exp(x)`
π	`pi`
$\sin x$	`sin(x)`
$\sin^{-1} x$	`asin(x)`
$\sqrt{x}$	`sqrt(x)`

Other trigonometric functions are also available besides just `sin`, including `cos`, `tan`, etc.

R naturally applies these functions across vectors. For example, you can square all the entries in a vector as in the example below.

```
example.vector <- c( -3, 2, 0.5, -1, 10, 9.2, -3.3 )
example.vector ^ 2
```

```
[1]   9.00   4.00   0.25   1.00 100.00  84.64  10.89
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to quickly load some sample data

Description

Sometimes you just need to try out a new piece of code, whether it be data manipulation, statistical computation, plotting, or whatever. And it's handy to be able to quickly load some example data to work with. There is a lot of freely available sample data out there. What's the easiest way to load it?

Solution in pure R

R comes with many datasets in its `datasets` package. Ensure that you have it installed as follows.

```
library(datasets)
```

Then you can load any one of them with the `data` function, as follows.

```
data(iris)  # Load the famous Fisher's irises dataset.
head(iris)  # It has been placed in a variable of the same name.
```

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

To page through a list of all available datasets, just call `data()` with no arguments.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to compute summary statistics

Description

The phrase "summary statistics" usually refers to a common set of simple computations that can be done about any dataset, including mean, median, variance, and some of the others shown below.

Related tasks:

- How to summarize a column (on website)
- How to summarize and compare data by groups (on website)

Solution in pure R

We first load a famous dataset, Fisher's irises, just to have some example data to use in the code that follows. (See how to quickly load some sample data.)

```
library(datasets)
data(iris)
```

How big is the dataset? The output shows number of rows then number of columns.

```
dim(iris)  # Short for "dimensions."
```

```
[1] 150   5
```

What are the columns and their data types? Can I see a sample of each column?

```
str(iris)  # Short for "structure."
```

```
'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 ...
```

What do the first few rows look like?

```
head(iris) # Gives 5 rows by default.  You can do head(iris,10), etc.
```

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

The easiest way to get summary statistics for an R `data.frame` is with the `summary` function.

```
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  
```

The columns from the original dataset are the column headings in the summary output, and the statistics computed for each are listed below those headings.

We can also compute these statistics (and others) one at a time for any given set of data points. Here, we let `xs` be one column from the above `data.frame` but you could use any vector or list.

```
xs <- iris$Sepal.Length

mean( xs )           # mean, or average, or center of mass
median( xs )         # 50th percentile
quantile( xs, 0.25 ) # compute any percentile, such as the 25th
var( xs )            # variance
sd( xs )             # standard deviation, the square root of the variance
sort( xs )           # data in increasing order
sum( xs )            # sum, or total
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to convert a text column into dates

Description

When loading data, many software systems make intelligent guesses about the format and data type of each column, but sometimes that is not sufficient. If you have a column of text that should be interpreted as dates, how can we ask the software to convert it?

Solution in pure R

Let's create a small example DataFrame to use here (using the method from how to create a data frame from scratch). Naturally, you would apply this solution to your own data instead.

```r
df <- data.frame(
    Date  = c( '5/7/19', '5/10/19',   '5/11/19' ),
    Event = c(   'Work',   'Party', 'More work' )
)
df
```

```
  Date    Event
1 5/7/19  Work
2 5/10/19 Party
3 5/11/19 More work
```

We use the `as.Date()` function to convert a text column into dates. If the input dates are not in the standard format, we can use the `format=` argument to change the format. Note the difference between `%y` and `%Y`: The `%y` code means a 2-digit year, but the `%Y` code means a 4-digit year.

```r
df$Date = as.Date( df$Date, format='%m/%d/%y' )
df
```

```
  Date       Event
1 2019-05-07 Work
2 2019-05-10 Party
3 2019-05-11 More work
```

It's often easier to handle date conversions while reading the data file. You can use the `read_csv()` function in the `readr` package, which will automatically recognize dates in some common formats.

Additionaly, you can use the `anytime()` function in the `anytime` package to automatically parse strings as dates regardless of the format.

```r
# Use anytime() to attempt to parse various formats:
library(anytime)
examples <- c( "Nov 01 2022", "2022-11-01", "22-11-01" )
anytime( examples )
```

```
[1] "2022-11-01 UTC" "2022-11-01 UTC" NA
```

Note that it succeeded in two cases, but not the third.

How to create a data frame from scratch

Description

Sometimes it is useful to create a small table of data directly in code, without first needing to store the data in a file and load it from there. This can be useful for creating small tables for testing purposes, or for creating small lookup tables that hold abbreviations, IDs, etc. What's the easiest way to build such a table?

Solution in pure R

In R, the `data.frame` function can be used to build data frames. Each column in your data should be a separate parameter, with its name provided, followed by an equals sign, followed by the vector of column contents, as shown in the example below.

```r
data <- data.frame(
    last.name =  c( 'Potter', 'Weasley', 'Granger', 'Malfoy' ),
    first.name = c( 'Harry', 'Ron', 'Hermione', 'Draco' ),
    house =      c( 'Griffindor', 'Griffindor', 'Griffindor', 'Slytherin' )
)
data
```

```
  last.name first.name house
1 Potter    Harry      Griffindor
2 Weasley   Ron        Griffindor
3 Granger   Hermione   Griffindor
4 Malfoy    Draco      Slytherin
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to compute covariance and correlation coefficients

Description

Covariance is a measure of how much two variables "change together." It is positive when the variables tend to increase or decrease together, and negative when they upward motion of one variable is correlated with downward motion of the other. Correlation normalizes covariance to the interval $[-1, 1]$.

Solution in pure R

We will construct some random data here, but when applying this, you would use your own data, of course.

```r
# Create a dataframe with random values between 0 and 1
set.seed(1)
df <- as.data.frame(matrix(runif(n=50,min=0,max=1),nrow = 10))
names(df) <- c('col1','col2','col3','col4','col5')
head(df)
```

```
  col1      col2      col3      col4      col5
1 0.2655087 0.2059746 0.9347052 0.4820801 0.8209463
2 0.3721239 0.1765568 0.2121425 0.5995658 0.6470602
3 0.5728534 0.6870228 0.6516738 0.4935413 0.7829328
4 0.9082078 0.3841037 0.1255551 0.1862176 0.5530363
5 0.2016819 0.7698414 0.2672207 0.8273733 0.5297196
6 0.8983897 0.4976992 0.3861141 0.6684667 0.7893562
```

In R, we can use the `cov()` function to calculate the covariance between two variables. The default method is Pearson.

```r
cov( df$col1, df$col2 )
```

```
[1] 0.0004115864
```

You can also compare all of a DataFrame's columns among one another, each as a separate variable.

```r
cov(df)
```

```
          col1          col2          col3          col4          col5
col1  0.0996382947  0.0004115864 -0.0287090091 -0.0052485522 -0.029944309
col2  0.0004115864  0.0731549057 -0.0255386673 -0.0112688616 -0.026535785
col3 -0.0287090091 -0.0255386673  0.0942522913  0.0009465216  0.050640298
col4 -0.0052485522 -0.0112688616  0.0009465216  0.0593140088 -0.008714775
col5 -0.0299443088 -0.0265357850  0.0506402980 -0.0087147752  0.055665077
```

The Pearson correlation coefficient can be computed with `cor()` in place of `cov()`.

```r
cor(df$col1,df$col2)
```

```
[1] 0.004820878
```

And you can compute correlation coefficients for all numeric columns in a DataFrame.

```
cor(df)
```

```
     col1         col2         col3        col4        col5
col1  1.000000000  0.004820878 -0.29625051 -0.06827280 -0.4020775
col2  0.004820878  1.000000000 -0.30756049 -0.17107229 -0.4158329
col3 -0.296250506 -0.307560491  1.00000000  0.01265919  0.6991315
col4 -0.068272803 -0.171072293  0.01265919  1.00000000 -0.1516653
col5 -0.402077472 -0.415832858  0.69913152 -0.15166527  1.0000000
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create basic plots

Description

Plotting is a huge topic with many options and variations, but the most foundational types of plots are a line plot and a scatterplot. How can we create those?

Related tasks:

- How to add details to a plot
- How to create a histogram
- How to create a box (and whisker) plot
- How to change axes, ticks, and scale in a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in pure R

We will create some fake data using vectors, for simplicity. But everything we show below works also if your data is in columns of a DataFrame.

```r
patient.id     <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
patient.height <- c(60,  64,  64,  65,  66,  66,  70,  72,  72,  76)
patient.weight <- c(141, 182, 169, 204, 138, 198, 180, 175, 244, 196)
```

We can make a line plot if we use the `type="l"` option (which is an "ell," not a number one).

```r
plot(patient.id, patient.height, main="Patient heights", type="l")
```

We can create a scatterplot if we have two numerical columns, such as the height and weight in the data above.

```r
plot(patient.height, patient.weight, main = "Height vs. Weight")
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to add details to a plot

Description

After making a plot, we might want to add axis labels, a title, gridlines, or text. Plotting packages provide tons of tools for this sort of thing. What are some of the essentials?

Related tasks:

- How to create basic plots
- How to create a histogram
- How to create a box (and whisker) plot
- How to change axes, ticks, and scale in a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in pure R

We will create some fake data using R vectors, for simplicity. But everything we show below works also if your data is in columns of a data frame, such as `df$age`.

```
patient_height <- c(  60,  64,  64,  65,  66,  66,  70,  72,  72,  76 )
patient_weight <- c( 141, 182, 169, 204, 138, 198, 180, 175, 244, 196 )
```

The following code creates a plot with many details added, but each is independent of the others, so you can take just the bit of code that you need.

```
plot( patient_height, patient_weight,
      main="This is the title.",
      xlab="This is the x axis label.",
      ylab="This is the y axis label." )   # Scatter plot with labels
grid()                                     # Turns on gridlines
text( 70, 200, "Text at (70,200)" )        # Add text
text( 65, 225, "Point 1", pos=2 )          # Text to the left of a point
text( 72, 244, "Point 2", pos=4 )          # Text to the right of a point
arrows( 65, 225, 72, 244, col='red' )      # Arrow from (65,225) to (72,244)
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to change axes, ticks, and scale in a plot

Description

The mathematical markings and measurements in a plot can make a big difference on its readability and usefulness. These include the range of each axis, which points on that axis are marked with tick marks, and whether the axes use linear or logarithmic scaling. How can we customize these options?

Related tasks:

- How to create basic plots
- How to create a histogram
- How to create a box (and whisker) plot
- How to add details to a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in R

How to Data does not yet contain a solution for this task in R.

How to create a histogram

Description

A histogram is a very common and useful data visualization. It displays an approximation of the distribution in single series of data points (one variable) by grouping the data into bins, each bin draw as a vertical bar. How can we create such a visualization?

Related tasks:

- How to create basic plots
- How to create a box (and whisker) plot
- How to add details to a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in pure R

We will create some random data, but that's just for demonstration purposes. You can apply the answer below to any data. Simply replace the data variable with your real data (a list, a column of a dataframe, etc.).

```r
data <- rnorm(1000)
```

We can use R's hist() function to create the histogram.

```r
hist(data)
```

The y axis in a histogram is frequency, or the number of occurences. You can change it to probabilities instead.

```r
hist(data, prob = TRUE)
```

You can also choose your own bin boundaries. You might specify the number of bin breaks you want, or you can choose the exact bin breaks that you want.

```r
hist(data, breaks = 8)                # Specify number of bin breaks
hist(data, breaks = c(seq(-5, 5, 1))) # Choose exact bin breaks
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create a box (and whisker) plot

Description

A box plot, or a box and whisker plot, shows the quartiles of a single variable from a dataset (one of which is the median) and may also show the outliers. It is a simplified way to see the distribution of a variable. Sometimes multiple box plots (one for each of several variables) are shown side-by-side on a plot, to compare the variables. How can we create such graphs?

Related tasks:

- How to create basic plots
- How to add details to a plot
- How to create a histogram
- How to change axes, ticks, and scale in a plot
- How to create bivariate plots to compare groups (on website)
- How to plot interaction effects of treatments (on website)

Solution in pure R

We will create some fake data using vectors, for simplicity. But everything we show below works also if your data is in columns of a DataFrame.

```
patient_id     <- c(0,   1,   2,   3,   4,   5,   6,   7,   8,   9)
patient_height <- c(60,  64,  64,  65,  66,  66,  70,  72,  72,  76)
patient_weight <- c(141, 182, 169, 204, 138, 198, 180, 175, 244, 196)
```

We can use R's boxplot() function to make the plot.

```
boxplot(patient_weight)
```

You can show more than one variable's box plot side-by-side by passing both variables into the boxplot() function.

```
boxplot(patient_height, patient_weight)
```

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to create a QQ-plot

Description

We often want to know whether a set of data is normally distributed, so that we can deduce what inference tests are appropriate to conduct. If we have a set of data and want to figure out if it comes from a population that follows a normal distribution, one tool that can help is a QQ plot. How do we make and interpret one?

Related tasks:

- How to test data for normality with Pearson's chi-squared test (on website)
- How to test data for normality with the D'Agostino-Pearson test (on website)
- How to test data for normality with the Jarque-Bera test (on website)

Solution in pure R

We're going to use some fake data here by generating random numbers, but you can replace our fake data with your real data in the code below.

```r
# Replace this with your data, such as a variable or column in a DataFrame
values <- c(4, 90, 85, 49, 34, 23, 17, 10, 20, 59, 100, 112, 46, 10, 4, 39, 24, 77, 63, 23, 67, 109, 70)
```

If the data is normally distributed, then we expect that the QQ plot will show the observed values (black circles) falling very clsoe to the red line (the quantiles for the normal distribution).

```r
# Make a QQ plot for the data
qqnorm(values, pch = 1)
# Add the reference line representing what the data should look like if normally distributed
qqline(values, col = "red", lwd = 2)
```

Our observed values fall pretty close to the reference line. In this case, we expected that, because we created fake data that was normally distributed. But for real data, it may not stay so close to the red line.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.