Workshop on Julia - Introduction to Julia for Statistics

ANZ Statistics Conference, University of Canterbury, November 2015

Arindam Basu MB BS MPH PhD

School of Health Sciences,

University of Canterbury

Email: arindam.basu@canterbury.ac.nz

Sessions

Hello and welcome to the Julia workshop. The purpose of this workshop and indeed this document is to introduce Julia, a programming language that is flexible, easy to use and quite intuitive.

Session Timings

- 9-10:00 AM - Session 1: About, Download, Installation
- 10-10:30 AM - Hands on Installation with different options
- 10:30 AM - Morning Tea
- 10:40-12:00 - Session 2: Introduction to Julia Language
- 12:00-1:00 PM - Session 3: Data Analysis and Graphics using Julia
- 1:00 PM - Questions and answers, we break for lunch

Session 1: About Julia, Download, Installation

Getting started

What is Julia?

From the Julia website,

"Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Julia's Base library, largely written in Julia itself, also integrates mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing. In addition, the Julia developer community is contributing a number of external packages through Julia's built-in package manager at a rapid pace. IJulia, a collaboration between the IPython and Julia communities, provides a powerful browser-based graphical notebook interface to Julia."

Why Might We be Interested in Julia

- Good for custom work and writing your own programme
- Useful Built-in functions and Packages
- Very extensible
- Everything is written in Julia itself
- Great for Reproducible Research
- Interfaces with other programmes
- It can call Python by PyCall
- It can call R by calling RCall

Performance Metrics of Julia

- To check performance,see (http://speed.julialang.org)
- To check performance, see also (https://Github.com/Julialang/julia/tree/master/test/perf/micro)

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathematica 10.2.0	JavaScript V8 3.28.71.19
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

Julia Sources - Where to go for Julia

- First point: Julia Home Page (http://www.julialang.org)
- Julia Documentation (http://docs.julialang.org)

Help with Julia

- Google and Search Engines in general
- Mailing LIst (http://julialang.org/community/)
- Julia Stack Overflow Site (http://stackoverflow.com/questions/tagged/julia-lang)
- I have included the Julia current documentation in PDF in the working directory
- Juliabloggers (http://www.juliabloggers.com) is a site for blogs
- Books
- Mastering Julia (http://www.amazon.com/Mastering-Julia-Contemporary-Challenges-Programming-ebook/dp/B010T266RY)
- Getting Started with Julia Programming Language (http://www.amazon.com/Getting-Started-Julia-Ivo-Balbaert-ebook/dp/B00U2MI8OG/ref=pd_sim_351_1?ie=UTF8&dpID=517X1-b9loL&dpSrc=sims&preST=_AC_UL160_SR135%2C160_&refRID=02RBCKAPCQQHVMRRPPR8)
- A good compilation of Julia related resources (https://github.com/svaksha/Julia.jl/blob/master/Resources.md)

How to install and work with Julia

There are several ways to work with Julia

1. No installation needed (as we shall use in this workshop) Visit: http://www.juliabox.org (http://www.juliabox.org) to get started
2. You really do not need to download Julia to your machine to get started. Now there are two web based sources where you can use Julia without downloading any other software:
3. You can use sagemathcloud
4. You can use Juliabox.org (http://juliabox.org) and directly use your Google username to log in and work with Julia

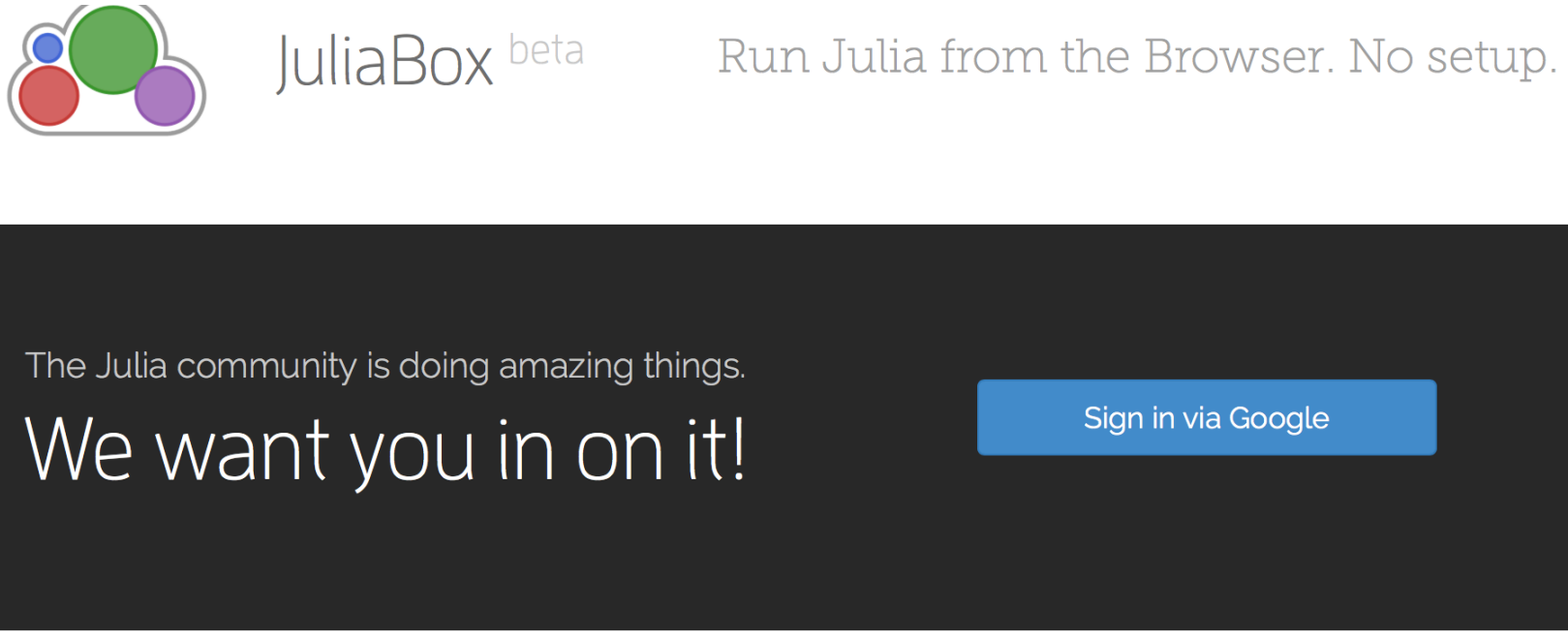
In this workshop, this is what we intend to do. So, just log on to juliabox.org and start working.

- You can use Juno to work with Julia. Juno is based on LightTable and can be ontained from: Juno Website (http://junolab.org)

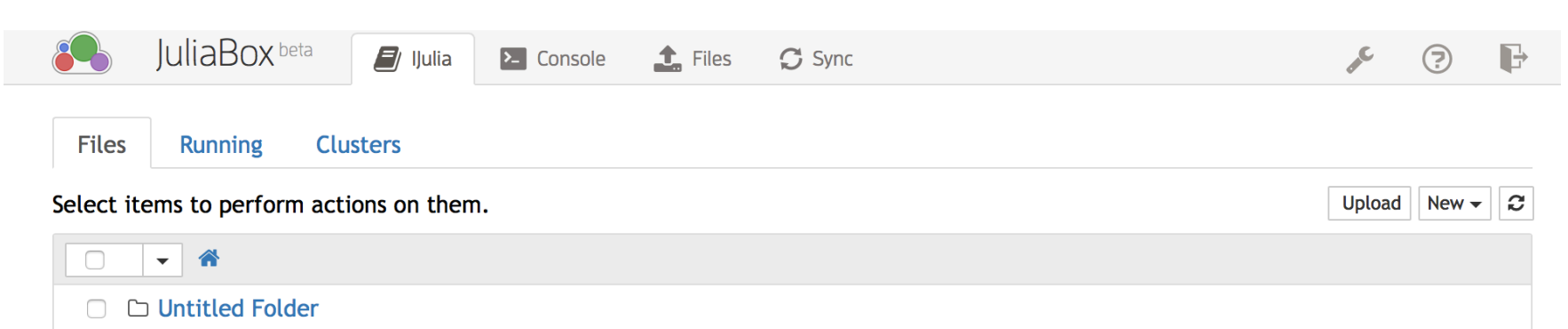
If you do not want to install anything, use Juliabox.org

Here are the steps.

1. Visit the Juliabox.org site (https://www.juliabox.org)
2. You will see something like this:



1. Click on sign in via Google (if you have a google account), this will bring you to the following window (it will look different on your computer as this is my machine view):



1. From here there are two paths:
2. Either, click on sync, and then choose workshop-2015 from the shared Google Drive folder, OR
3. Upload the contents of this folder.
4. Then click on the juliaworkshop2015.ipynb file. You are all set.
5. Just work with the codes presented in this file

Installing Julia

1. Obtain the binaries from the julia downloads site (http://julialang.org/downloads/), here:
2. Here is the list of different binaries available:

Julia (command line version)

Windows Self-Extracting Archive (.exe)	32-bit		64-bit
Mac OS X Package (.dmg)	10.7+ 64-bit		
Ubuntu packages (.deb)	32/64-bit		
Fedora/RHEL/CentOS/SL packages (.rpm)	32/64-bit		
Generic Linux binaries	32-bit (GPG)		64-bit (GPG)
Source	Tarball (GPG)	Full Tarball (GPG)	GitHub

1. From here, just double clicking on the relevant file for your system will install Julia on your system.
2. On Mac for instance, you start Julia.App from the Applications folder and will see something like this:

1. On Mac, if you want to start Julia from the commandn prompt, open a terminal and write:

```
sudo ln -s /Applications/Julia-0.4.1.app/Contents/Resources/julia/bin/julia /usr/local/bin/julia
```

After this, if you type julia next to $ prompt, you will be taken to the julia REPL window.

Installing Juno

Another way you can install Julia is to install Juno. Juno is an IDE and works on LightTable. To install Juno, visit: Junolab.org (http://junolab.org), download, and install from the binaries. It will install the needed binaries.

Installing IJulia

This is one of the best ways to work with Julia. Here you interact with Julia on IPython environment. Follow these steps to get IJulia working on your computer (these are for Mac OSx and Windows, but the steps might be similar to your OS please check with OS specific system):

1. Download Anaconda Python from http://continuum.io/downloads (http://continuum.io/downloads)
2. After it is installed, open Julia from your applications.
3. Type `Pkg.add("IJulia")` and wait for IJulia installation.
4. Open a terminal and type `jupyter notebook` in the desired folder/directory to start

If you use Linux, you already have Python installed. Type:

```
pip install ipython
```

After this, from Julia, type `Pkg.add("IJulia")`

1. To get started, type "jupyter notebook" at the command prompt.

After you get started, you will need to install a few packages to get working with Julia, so we shall focus on these next.

Package Management

- Julia uses git as a repository for itself and for all packages
- Installation has a built in package manager
- Below, we have issued a command on package update
- Following this, we can check the status of different installed packages using Pkg.status()
- Also, note that in Jupyter or in Juliabox.org, when you ate after you enter the codes, you hit shift+enter to run the codes.
- You can find a list of packages at the docs.julialang.org (http://docs.julialang.org)
- At the console you can also find the available packages by issuing Pkg.available()
- At the time of writing this, you have about 756 packages made available for this version of Julia
- If you want to add a registered pacakge, use Pkg.add()
- If you want to add an unregistered package, use Pkg.clone(URL of the package), where hte URL is a Git url

General Principle of Using a Package

1. First, install a package by using Pkg.add("Packagename")
2. If the package is not in Julia repository, use Pkg.clone("URL of the package")
3. Then, call the package by issuing: `using Packagename`

In [ ]:

```
## List of the available Julia packages

Pkg.available()
```

In [ ]:

```
## Update your package repository
Pkg.update()
```

In [ ]:

```
## Check the status of your installed packages
Pkg.status()
```

In [ ]:

```
## Which packages are installed
Pkg.installed()
```

In [ ]:

```
# Install a package that is not in Julia repository
Pkg.clone("https://github.com/joemliang/RegTools.jl.git")
```

Exercise 1: We shall use the following packages to work in Julia

'

Task: Please install (Pkg.add) the following packages and check that they are correctly installed on your system

- Stats
- StatsBase
- DataFrames
- Distributions
- HypothesisTests
- GLM
- Multivariate
- Gadfly
- PyCall
- Winston
- RDatasets

In each case, when you install, issue:

```
Pkg.add("PackageName.jl")
```

When you call them, type:

using Packagename

Session 2 - About Julia - Getting to know it

- Strongly typed language
- Does not require the type to be declared
- Use "?topic" to find help, example: "?which"
- For finding functions, use method("topic")

Variable in Julia

- Any combination of upper case, lower case, digits, underscore, exclamation
- must start with a letter OR underscore
- Use lowercase with underscore don't use CamelCase
- To find variable type, use typeof()
- In the next box, we have a few examples

In [ ]:

```
#= You can use multiline comments in Julia
Just keep them within =#
x = 2; typeof(x)
```

In [ ]:

```
x = 2.9; typeof(x)
## to suppress output, add a semicolon ;
```

Types in Julia

- Adding type information is optional
- Type annotation:
- x::ASCIIString
- var::TypeName
- Used in fnctions arguments

In [ ]:

```
"This"::AbstractString
#Type comes after a variable
```

In [ ]:

```
## use the function isa() to test the type of variable
x = 7;
typeof(x)
isa(x, Int)
```

In [ ]:

```
## Maximum value for Int64 data type
typemax(Int64)
```

In [ ]:

```
## Minimum value for Int64 data type
typemin(Int64)
```

Features

- Division of two integers produce a real
- to get integer divisor, use div(x,y)
- To get remainder use %, or use rem(x,y)
- x\y == y/x
- Let's check out:

In [ ]:

```
x = 5; y = 3;
z = x/y;
typeof(z)
```

In [ ]:

```
## Dividing two integers
z1 = div(x,y); z1
```

In [ ]:

```
typeof(z1)
```

In [ ]:

```
## finding remainders

r1 = rem(x,y)
```

In [ ]:

```
## finding remainders, another way
r2 = x % y
```

In [ ]:

```
## reverse
x \ y
```

Booleans

- Logical type Bool
- Dynamically assigned

In [ ]:

```
t = (2 < 3)
```

In [ ]:

```
typeof(t)
```

Arrays

- Index starts at 1 not 0
- Easy to create index A = [1,2,3,4] (see below)
- You can create range B = [1:4]
- You can create start, step, range as start:step:range C = [1:2:10] (see examples below)

In [ ]:

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

In [ ]:

```
A1 = 1:10000
sum(A1)
```

In [ ]:

```
B = [1:2:10]
```

Arrays - II

- How to populate Arrays
- Use list comprehension
- A = Array(Int64, 10) ... Array of 10 integers
- Create an empty array A = Int64[]
- YOu can add elements to this array by push!() function
- You can remove elements from the end of this array by using the pop!() function
- The bang sign (!) is to change an element (see the examples below)

In [ ]:

```
## creating a 10 element array
A = Array(Int64, 10)
```

In [ ]:

```
## We can change the above array by assigning values to it

A[1] = 5; A[2] = 10; A
```

In [ ]:

```
# Create empty array
B = [];
# then populate
push!(B, 2) # the array comes and then value
push!(B, 3)
push!(B, 4)
B
# remove or pop 4 from B
pop!(B)
B
```

In [ ]:

```
## Array dimensions
## One-dimensional array
A = [1,2,3]
A
# Produces one column and three rows
```

In [ ]:

```
## Three column  array
A = [1 2 3]
A
# produces 1 row and 3 columns
```

In [ ]:

```
## Adding rows to arrays
A = [1 2 3; 4 5 6]
## will produce array with two rows and three columns
## indexing is in column order
```

In [ ]:

```
## illustration how columns order is run in matrices

for i in (1:length(A)) @printf("%1d: %1d\n", i, A[i]); end
```

In [ ]:

```
## You can reshape a matrix from 2 x 3 to 3 x 2

B = reshape(A, 3, 2);
B
```

In [ ]:

```
## Then we can transpose B, two ways:

C = transpose(B)
C
```

In [ ]:

```
## Another way to transpose matrix B
C1 = B'
```

In [ ]:

```
## Can add matrices

D = A + C;
D
```

In [ ]:

```
## Matrix multiplication
E = A * B; E
```

In [ ]:

```
## Elemental operations, note the dot
## Multiplication
F = A .* C;
F
## Equality check

A .== C
```

Strings in Julia

- Within double quotes
- Strings are immutable, cannot change values
- Strings are indexable
- ## Data Arrays and Data frames
- Package DataFrames
- using DataFrames
- NA removed by removeNA()

Types and dispatch

Functions

- General structure
- function () ... end
- One way: sq(x) = x * x
- Another, function sq(x) y = x * x return y end
- this y is not needed
- Functions are first class objects

Metaprogramming

- Possible to create runtime macros
- Check out the Lazy.jl package source code
- Symbols in Julia represent the homoiconity

Symbols in Julia

- d = :(a + b + c)

In [ ]:

```
d = :(a + b + c)
```

- Note when we do something like this, there is no error
- We did not specify what was a, b, or c, so d could not be estimated at all
- the : operator made sure that the expression was saved not evaluated
- What is the type of d?
- The type of d is expression
- Now if we specify the values of a, b, c, we can evaluate the expression (see below in a separate example)

In [ ]:

```
typeof(d)
```

In [ ]:

```
z = :(x + y);
# this is just an expression
# Next, specify the values of x and y
x = 10
y = 12
eval(z)

## what is an expression?
names(z)
```

- So, if we use expression we have two parts
- We store the expression
- Then we evaluate it
- What is an expression then?

In [ ]:

```
## what is an expression?
fieldnames(z)
## What are the arguments of z?
z.args
```

Testing in Julia

- assert() function which raises exception
- @assert macro = evaluate condition and output custom message

In [ ]:

```
using Base.Test
x = 1;
@test x == 1
```

In [ ]:

```
@test x == 3
ErrorException("test failed")
```

Parallel Operations in Julia

- Based on message passing
- Remote calls: call a function with arguments on a specific processor and return a remote reference to the call. Raise exception if the call fails, else, runs the task asynchronously. You can wait for the remote task to finish. You can fetch the value of the result by fetch() ## Set up the mulitprocessor system:
- Use julia -p n where n = number of processors
- the process that has the interactive julia prompt has an id of 1
- All other processes are workers. If one process you have one worker
- You can create extra processors by addproc(n) where n number of processors
- See the illustration below

In [ ]:

```
## start multiprocessors

addprocs(3);
```

In [ ]:

```
## check how many processors
nprocs()
```

In [ ]:

```
## how many workers? (should be 3)
nworkers()
```

In [ ]:

```
## select one of them to execute some code
r = remotecall(1, randn, 5)
fetch(r)
```

Why parallelism matters

- Facilitates analysis of big data
- We can spawn worker processeson aribtrary machines
- Macros that use parallel processes @spawn, @async, @spawnat (see example below)

In [ ]:

```
addprocs(3) # add three 2 workers
@everywhere fib(n) = (n < 2) ? n :(fib(n-2) + fib(n-2))
# @everywhere ensures function definition available on all processors
```

In [ ]:

```
@elapsed fib(40) ## gets the elapsed time for running this function
```

In [ ]:

```
r = @spawn @elapsed fib(40)
@spawn selects any machine at random
```

In [ ]:

```
## Simple MapReduce
## generate a matrix of 300 x 300
## normally distributed random numbers, mean = 0, sd= 1
## add three processors

addprocs(3)
d = drand(300, 300)
nd = length(d.chunks)
```

In [ ]:

```
Pkg.add("DistributedArrays")
?drand
```

In [ ]:

```
?drand
```

In [ ]:

```
using DistributedArrays
d = drand(1000);
```

In [ ]:

```
typeof(d)
```

In [ ]:

```
names(d)
```

In [ ]:

```
d.dims # we asked for 1000 points
```

In [ ]:

```
d.chunks # provides the remote references
```

In [ ]:

```
length(d.chunks) # number of workers that hold the data
```

In [ ]:

```
d.pids
```

In [ ]:

```
d.indexes
```

How Julia interacts with the system

In [ ]:

```
## print the working directory
pwd()
```

In [ ]:

```
## Working with the filesystem
# read files
readdir()
```

Working with Data in Julia

In [ ]:

```
write(STDOUT, "Hello World")
```

- 11 is the number of bytes written
- print absorbs the byte count

Structured datasets

In [ ]:

```
readdir()
```

In [ ]:

```
pwd()
```

In [ ]:

```
write(STDOUT, "hello")
```

Uploading csv and dlm files

In [ ]:

```
(mydata, cols) = readcsv("wolfriver.csv", header=true)
mydata
```

In [ ]:

```
typeof(mydata)
```

In [ ]:

```
cols
```

- We can see that there are four fields per row, and the
- last column is left blank
- the data itself is a matrix of 30 rows and 4 columns
- Data is in the form of any array

In [ ]:

```
mydata[:, 1:3]
## all rows, first three columns
```

DataFrames and RDatasets

- NA represents a missing value
- DataArray is a type that emulates julia's Array type but can store missing values
- Dataframe can represent tabular datasets
- DataArray has DataVector, and DataMatrix
- (see the examples)

In [ ]:

```
using DataFrames
da = @data([NA, 3.1, 2.3, 5.7, NA, 4.4])
```

In [ ]:

```
## Let's check what happens if we take mean
thismean = mean(da)
```

In [ ]:

```
## We need to drop the NA values,
## use dropna()
meanafterdroppingna = mean(dropna(da))
```

In [ ]:

```
## DataFrames is a real world tabular dataset
## Create a dataframe with x linearly spaced 0 to 10
## y randomly distributed values

df = DataFrame(X=linspace(0,10,101), Y = randn(101))
```

In [ ]:

```
typeof(df)
```

In [ ]:

```
## finding mean
mean(df[:X])
mean(df[:Y])
```

In [ ]:

```
var(df[:Y])
```

In [ ]:

```
using DataFrames
mydatax = DataFrame(readcsv("wolfriver.csv"))
```

In [ ]:

```
threept = readtable("threept.csv")
```

In [ ]:

```
describe(threept)
```

In [ ]:

```
gap = threept[:inc03] - threept[:inc07]
Histogram(gap)
```

RDatasets

- start with using RDatasets
- to install, do Pkg.add("RDatasets")
- To load data, use dataset() function
- dataset(names of the data group, specific dataset

In [ ]:

```
## Here is a complete list of all datasets
RDatasets.available_datasets()
```

In [ ]:

```
using RDatasets
mlmf = dataset("mlmRev", "Gcsemv")
```

In [ ]:

```
## lets summarise the data

summary(mlmf)
```

In [ ]:

```
## You can see it has five columns and 1905 rows
## Let's get the header data
head(mlmf)
```

In [ ]:

```
## We can select only one school data
mlmf[mlmf[:School] .== "20920", :]
# We want school number 20920, all rows
```

In [ ]:

```
## group the data by School
groupby(mlmf, :School)
```

In [ ]:

```
sort!(mlmf, cols = [:Written])
```

In [ ]:

```
## Let's remove all missing values
df = mlmf[complete_cases(mlmf), :]
```

In [ ]:

```
## let's calculate the difference in the scores

diff = df[:Course] - df[:Written]
```

In [ ]:

```
## what is the mean of diff?
mean(df[:Course] - df[:Written])
```

In [ ]:

```
df=mlmf[complete_cases(mlmf[[:Written, :Course]]), :]
cor(df[:Written], df[:Course])
```

In [ ]:

```
?dropna
```

In [ ]:

```
?meanafterdroppingna
```

In [ ]:

```
mean(complete_cases(mlmf[:Written]))
```

In [ ]:

```
using DataArrays

diff = dropna(mlmf[:Course] - mlmf[:Written])
```

In [ ]:

```
summarystats(df[:Written])
```

In [ ]:

```
df[isnan(df)] = 0
```

In [ ]:

```
?isnan
```

In [ ]:

```
isnan(mlmf)
```

In [ ]:

```
## we removed the Nans
dfnew = mlmf[!isnan(mlmf[:Course]), :]
dfnew = dfnew[!isnan(dfnew[:Written]), :]
diff = dfnew[:Course] - dfnew[:Written]
mean(diff)
cor(dfnew[:Written], dfnew[:Course])
linreg(convert(Array, dfnew[:Written]), convert(Array, dfnew[:Course]))
```

In [ ]:

```
typeof(dfnanremoved)
```

In [ ]:

```
mean(dfnew[:Course])
```

In [ ]:

```
diff = dfnew[:Course] - dfnew[:Written]
```

In [ ]:

```
dropna(mean(diff))
```

In [ ]:

```
mean(diff[!isna(diff)])
```

In [ ]:

```
loss = diff[!isna(diff),:]
```

In [ ]:

```
loss
```

In [ ]:

```
## Statistics
using StatsBase, PyCall
```

In [ ]:

```
readdir()
```

In [ ]:

```
thisdata = readtable("threept.csv")
```

In [ ]:

```
myd = thisdata[!isna(thisdata[:peakhrtuseyr]), :]
```

In [ ]:

```
summarystats(myd[:inc96])
```

In [ ]:

```
describe(myd)
```

In [ ]:

```
using HypothesisTests
```

In [ ]:

```
names(myd)
```

In [ ]:

```
UnequalVarianceTTest(float64(myd[:inc03]), float64(myd[:inc96]))
```

In [ ]:

```
## So there was an increase in 2003 compared with 1996
## How abotu between 2003 and 2007
UnequalVarianceTTest(float64(myd[:inc03]), float64(myd[:inc07]))
```

In [ ]:

```
lm1 = fit(LinearModel, inc07 ~ inc96, myd)
```

In [ ]:

```
using DataFrames
lm1 = fit(LinearModel, inc07 ~ inc96, myd)
```

In [ ]:

```
Pkg.add("GLM")
```

In [ ]:

```
Pkg.update()
```

In [ ]:

```
using GLM
```

In [ ]:

```
inchrt = fit(GeneralizedLinearModel, inc03 ~ inc96 + peakhrtusepct, myd, Norma
l())
```

In [ ]:

```
deviance(inchrt)
```

In [ ]:

```
Pkg.add("MultivariateStats")
using MultivariateStats
```

Graphics in Julia

- Julia has no in built graphics command
- Basic Graphics
- Graphic Engiines
- Web graphics

In [ ]:

```
Pkg.add("Cairo")
```

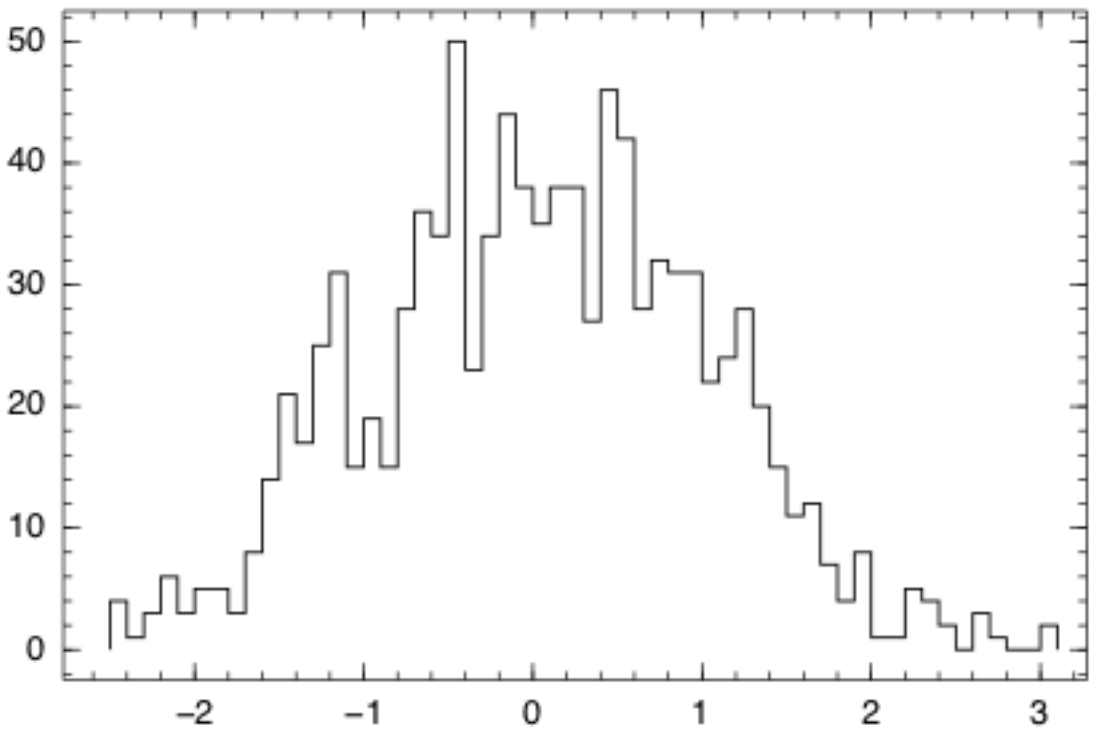
In [ ]:

```
using Winston
```

In [8]:

```
## How to plot a histogram using Winston
a = randn(1000)
ha = hist(a, 100)
plothist(ha)
```

Out[8]:



In [ ]:

```
## Let's look at a framed plot
p = FramedPlot(aspect_ratio = 1, xrange = (0, 100),
yrange = (0, 100))
n = 21
x = linspace(0, 100, n)
# Let's create a set of random variates
y1 = 40 .+ 10 * randn(n)
y2 = x .+ 5 * randn(n)

## Labels and symbol styles
a = Points(x, y1, kind = "circle")
setattr(a, label = "Points of A")
b = Points(x, y2)
setattr(b, label = "B Points")
style(b, kind = "filled circle")

## We plot a line which fits through y2
## Add a legend on the top part of the graph

s = Slope(1, (0,0), kind = "dashed")
setattr(s, label = "slope")
lg = Legend(0.1, 0.9, Any[a, b, s])
add(p, s, a, b, lg)
display(p)
```

In [9]:

```
## Use of Gadfly
using Gadfly
```

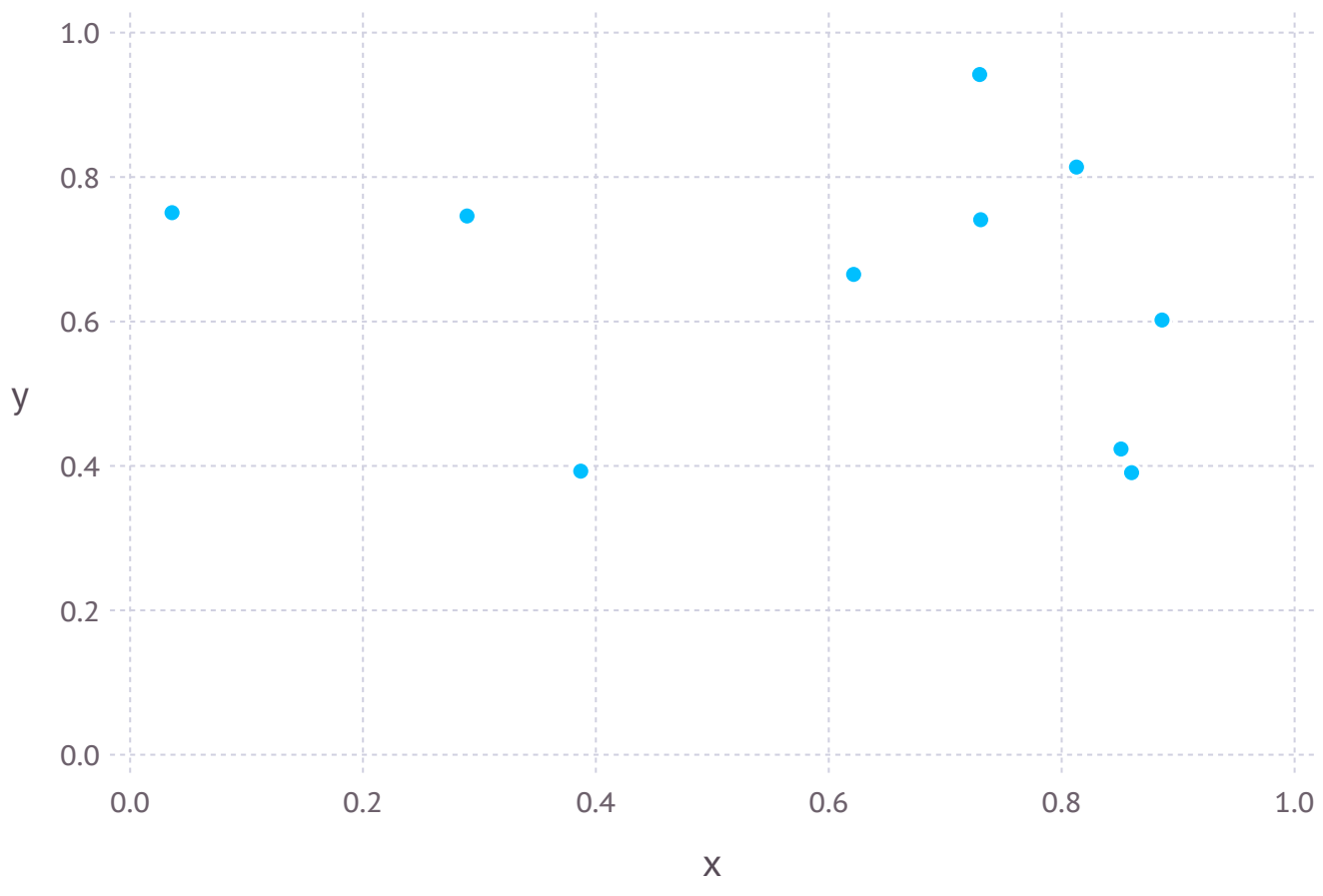
Use of gadfly

- Large, complex, flexible
- Equivalent to ggplot2
- Grammar of Graphics (Leland Wilkinson)
- More information on gadflyjl.org (http://www.gadflyjl.org)
- use plot(), assign plot() to a variable, and call with display()
- Can output to svg, pdf, png, etc

In [10]:

```
## illustration
using Gadfly
dd = Gadfly.plot(x = rand(10), y = rand(10))
# draw(PNG("this.png", 12cm, 6cm), dd)
```

Out[10]:



In [ ]:

```
dd
```

In [ ]:

```
## Where Gadfly can work
* Dataframes
* Functions and expressions
* Arrays and collections
```

In [ ]:

```
## Let's see how gadfly works on a data frame
## we shall use the threept dataset
using DataFrames
mydata = readtable("threept.csv")
# let's get a clean data set
mydatamod = mydata[!isna(mydata[:peakhrtuseyr]), :]
# we are going to work on mydatamod
```

In [ ]:

```
## here are the variables for mydatamod
names(mydatamod)
```

In [ ]:

```
## Let's plot the association between incidence in 96
## and incidence in peakhrtuse percent
## for different age ranges of mammography screening

plot(mydatamod, x = "peakhrtusepct", y = "inc96",
color = "agerange")
```

In [ ]:

```
## Let's plot multiline plots

x = [1:100;]
y1 = 1 - 2*rand(100)
y2 = randn(100)
```

General motif

plot(datasource, element1, element2, mappings) datasource = dataframe, fnction, array elements = themes, scales, markers, labels, titles, guides, mappings = symbols and assignments

Plot elements in Gadfly

- Aesthetics. -- functions, such as Stat.boxplot
- Arguments. -- Plotting arguments
- Geometry (Geom.) -- Actually does the drawing, eg Geom.line, Geom.point etc
- Guides (Guides.) -- similar to geometries

Programming Concepts with Julia Part II

In [ ]:

```
## Integers in Julia
# Find the default type
WORD_SIZE
# Find the minimum and maximum values
typemin(Int)
x = 10
x = big(x)
x^10000
## but if we did not declare big, then
y = 10
y^10000
# would return a wrong value
# +. -, * apply for integers
# / will lead to floating point
# use div and rem for integer divisor and remainder
```

In [ ]:

```
## Rounding off
x = 100
y = 3
#Use of round
round(x/y, 2)
```

In [ ]:

```
## working with strings
typeof("hello")
## see it understands utf
typeof("μ")
## strings are immutable
s = "Hello"
#s[2] = "Z"
## String is an array of characters, so all rules of Arrays apply
## start with 1
s[1]
## index of last character
endof(s)
## Number of characters
length(s)
## For substring, use indices
s[1:2]
s[2:end]
## String interpolation
x = 4
y = 5
# use of $ will replace the value of the var
w = "$x + $y in words, is nine"
#you can use expressions ($(x+y)) to be evaluated
w2 = "$x + $y in digits is $(x+y)"
## YOu can catenate strings
w3 = "ABC"
w4 = "DEF"
## catenate w3 and w4
## three ways
w5cat1 = *(w3, w4)
w5c2 = w3 * w4
w5c3 = string(w3, w4)
##:string is symbol, use it for ID, key,
## cannot catenate symbols
## list of methods or functions associated with Strings
# methodswith(AbstractString)
## about 400 functions
hw = "Hello World"

## search within string
search(hw, 'H')
## will return the position of the character
## Note character is given with ''
# replace
hw1 = replace(hw, "Hello", "Hi")
# Split a string on a character
hw2 = split(hw, 'H')
## if no character specified, then splits on space
hw3 = split(hw)
```

In [ ]:

```
## Formatting Numbers and Strings
## @printf takes format string, variables, substitues
x = "world"
#@printf("Hello %s", x)
## it took a string and added to Hello
## the string was in the variable x
# d for integers
y = 10
#@printf("%d", y)
z = 10.2345
# f for floating point numbers with 0.x where
# x = number of decimal places
#@printf("%0.2f", z)
# this is going truncate to two decimal places
# s for srtrings
@printf("%s", x)
```

In [ ]:

```
## Regular expressions
# Search for patterns in text and data
# use double quoted strings preceded by r
email = r".+@.+"
#input = "arin.basu@gmail.com"
#ismatch(email, input)
# ismatch is boolean
#=if ismatch(email, input)
    println("found")
else
    println("not found")
end =#

# An example of using shortcut with regular exp
#ismatch(email, input) ? "found" : "not found"
## ismatch is boolean
# for detailed, use "match"
input = "arin.basu#email.com"
myfind = match(email, input)
# will result in a blank
input1 = "arin.basu@email.com"
myfind2 = match(email, input1)
# will contain the entire string that matches
# offset
myfind2.offset
## starts at the position where the match begins
## offsets
myfind2.offsets
## capture part of the string
# use parentheses
emailpattern = r"(.+)@(.+)"
# we want to capture the username and hostname
input2 = "arin.basu@email.com"
m = match(emailpattern, input2)
m.captures
```

In [ ]:

```
## Ranges and arrays
typeof(1:100)
```

In [ ]:

```
# get a set of numbers between 1 to 9 that increases
# by units of 2
numset = collect(1:2:9)
## Ways to initialise arrays
x = 1:10 # this is a range, not array
x[1]
x1 = collect(1:10) # this is an array with
typeof(x1)
x1
## you can also initiailise with comma separation
x2 = [1,2,3,4,5]
## Specify type and number of elements
x3 = Array(Int64, 5)
## If you want to create 0 element arrays
```

In [ ]:

```
## to populate the array, use push!()
x4 = Int64[]; push!(x4, 1)
push!(x4, 2)
# Create arrays from range
x5 = collect(1:6)
# indexed from 1
#x4[1]
# join array elements
join(x4, ";")
x4
```

In [ ]:

```
# zero array
z1 = zeros(3)
z2 = ones(3)
```

In [ ]:

```
## Some more
myarray = collect(1:100)
typeof(myarray)
sum(myarray); std(myarray)
```

Functions in Julia

- General format function **functionname**(argument list) function logic return value end

  Or, leave the arguments empty, and functionname()
  - Value of the last expression is returned

In [ ]:

```
## functions

function mult(x,y)
    x*y
end
mult(3,4) # will return 12
```

In [ ]:

```
## functions return multiple values
function multi(x::Float64,y::Float64)
    x*y, x+y, x-y
end
multi(2,3)

## Arguments to functions are passed by references
# values are not copied
# they can be changed by the function

function insert_elements(anarray)
    push!(anarray, -11) # add -11 to an array
end
myarray = collect(1:5)
insert_elements(myarray)
# see this array got changed from a five element to a
# six element array
# Must define a function by the time you call it
# Indicate the argument type
# Julia is a strongly typed language.


function push_eleven(array::Array)
    push!(array, 11)

end
push_eleven(myarray)

# if we define a function without types,
# then this function becomes generic

insert_elements # should return generic function
multi

# Function defines its own scope
# you can nest one function within another
# functions with related functionality can have own file
```

In [ ]:

```
## concept of optional positional arguments
f(a, b = 5) = a + b
f(1)
# optional positional arguments should always occur at the end
# optional keyword arguments
k(a; b = 5) = a * b
k(6)
k(6, b = 10)
## Anonymous functions
function myadd1(a,b)
    a + b
end

# OR
function (a, b)
    a+ b
end
## tbis above function is called anonymous function
## Use a stab character
(a,b) -> a+b
typeof(mult)
```

In [ ]:

```
## first class functions
# Functions are their own type
# You can assign a function to a variable
m = mult
m(1,2)
```

In [ ]:

```
# Consider this, where it becomes similar to R
addtwo = function (x) x + 2 end
```

In [ ]:

```
# the above is an anonymous function, but
# you can call like
addtwo(2)
```

In [ ]:

```
# counter function that returns a tuple of two
# anonymous functions
function counter()
    n = 0
    () -> n += 1, () -> n = 0
end
(addthis, clear) = counter()

## both of these are anonymous functions
```

In [ ]:

```
# if we call addthis
addthis() # we get one, as n increases
```

In [ ]:

```
# then we can reset this,
clear()
```

In [ ]:

```
# so n here is captured by addthis and clear functions
# addthis and clear are closed over n
# hence these are called closures

## Currying functions
function add(x)
    return function f(y)
        return x + y
    end
end
add(10)(10)
## Need both parameters to run this function
# You can pass functions around
```

In [ ]:

```
## Recursive functions
# You can nest one fnction wihin another
function myfn(x)
    a = x - 2
    function y(a)
        a += 1
    end
    y(a)
end

# we nested function y(a) within myfn(x)
myfn(10)
```

In [ ]:

```
## REcursive conditions, ternary functions or
# short cut fnctions

# expr ? t : e
# expr = evaluate the expression
# ? if true,
# t = do true
# e = do else

sumup(n) = n > 1 ? sumup(n-1) + n : n

# what this function does is:
# 1. Test is n more than 1?
# if yes, then recursively run this function by deducting
# one from the parameter and add the n to it
# Otherwise, just mention the n and exit
# So, with 2, it will first test 2 > 1, returns true
# then sumup(1) will fail, so it will revert back to
# print 1, then add 2 to it, so the answer is 3
# thus it will recursively keep adding previous
# numbers to it
sumup(4)
```

Quiz

Write a recursive function to print out the the 10th fibonacci number [0,1,1,2,3,5,8]

In [ ]:

```
#Solution
fib(n) = n < 2 ? n: fib(n-1) + fib(n-2)
fib(10)
```

In [ ]:

```
## Map, filter and list
# map(func, coll)
# func = function that is successively applied to
# every element of a collection
# returns a new collection
myarray = collect(1:4)
addtwo = function (x) x+2 end
ymap = map(addtwo, myarray )
# the map function in ymap takes addtwo function
# and applies it to the array myarray
```

In [ ]:

```
## Filter
## takes the form filter(func, coll) as before
# Example
#myarray2 = collect(1:10)
#filter(x -> iseven(x), myarray2)
# can be used to return only evens
# So here is a difference between map and filter
#=
map(myarray2) do x
    if iseven(x) return "male"
        else return "female"
    end
end
=#

map(x -> iseven(x) ? "male": "female", myarray2)

# filter in this situation will not do anything
```

In [ ]:

```
## List comprehension

somearray = Float64[x^2 for x = 1:5]
# creates a five element array with squared numbers
somearray
# Create a 10 x 10 array filled with random numbers
mymat = [rand(x*y) for x in 1:10, y in 1:10]
```

In [ ]:

```
methodswith(Array)
```

In [ ]:

```
# this function will produce a fibonacci sequence till 10
function fibprod(x)
    a, b = (0,1)
    for i in 1:x
        produce(b)
        a, b = (b, a+b)
    end
end
fibprod(2)
# it does not do anything or does not produce any
# output, it just produces them
#
task1 = @task fibprod(5)
```

In [ ]:

```
## types of collection
# Arrays
# Matrices (multidimensional arrays)
# dictionary
# Set

## Matrices
## row vector, two dimension (1 row x 3 columns)
# matrix is a two dimensional array
newmat = [1 2 3]
# Ways to create vectors and matrix
myvec = Array{Int64, 1}
mymat = Array{Int64, 2}
```

In [ ]:

```
myvec
```

In [ ]:

```
mymat2 = [1 2; 3 4]
## this is a matrix with two rows and two columns
```

In [ ]:

```
## index by row and columns
mymat2[2,1]
```

In [ ]:

```
## products of matrices
# product of 1 x 2 with 2 x 1 matrix
[1 2] * [3;4]
```

In [ ]:

```
## matrix of random numbers
mymat3 = rand(3, 5)
# matrix of 3 rows and 5 columns
## find the number of dimensions
ndims(mymat3) # two dimensions
# find the number of rows
size(mymat3, 1) # 1 == rows 2 = columns
# find the number of columns
size(mymat3, 2)
# find the number of elements
length(mymat3)
# identity matri
identity_matrix = eye(3)
```

In [ ]:

```
## Matrix slicing

identity_matrix[1:end, 2] # only the second column
```

In [ ]:

```
## alternatively,
identity_matrix[:, 2]
## the second row,
identity_matrix[2, :]
## last four elements
identity_matrix[2:end, 2:end]
#3 set the second row to zero
identity_matrix[2, :] = 0
# check
identity_matrix
#change the values in the identity matrix
identity_matrix[2:end, 2:end] = [4 6; 7 9]
#check
identity_matrix
```

In [ ]:

```
## jagged array
jagged_array = fill(Array(Int64, 1), 3)
jagged_array[1] = [1,2,3]
jagged_array[2]=[4,5,6]
jagged_array

## Matrix transpose
mymat4 = [1 2; 3 4]
transposed_mymat = mymat4'
#check
mymat4
#check transposed
transposed_mymat
# matrix multiplication
mymat4 * transposed_mymat
## element wise multiplication
mymat4 .* 2
# inverse of a matrix
inverse_matrix = inv(mymat4)
# identity matrix as multiplication of inverse and matrix
identity_matrix_created = mymat4 * inverse_matrix
```

In [ ]:

```
# horizontal catenation
m1 = [1, 2, 3]; m2 = [4, 5, 6]
# want to horizontally catenate the two matrices
# so that you produce a 3 x 2 matrix (3 rows 2 cols)
horizontal_cat = hcat(m1, m2)
# want to vertically catenate two matrices
vertical_cat = vcat(m1, m2)
# this is same as append!
appended_m = append!(m1, m2)
## produces a one-dimensional array

## simpler solutions
m3 = [1 2; 3 4]; m4 = [5 6; 7 8]
side_side = [m3 m4]
top_bottom = [m3; m4]
```

In [ ]:

```
## change the dimensions of the array
new_array = [1:10]
reshaped_array = reshape(new_array, 2,5)
# created a new matrix with 2 rows 5 cols
# create a random number matrix 4 x 4
fourbyfour = rand(4, 4)
# reshape it to an 8 by 2
eightbytwo = reshape(fourbyfour, (2, 8))
```

In [ ]:

```
# Tuples
# group of values
# separated by comma
# surrounded by ()
# Same or differnt types, arrays always same type
#
a_tuple = (1,2,3, "old")
a, b, c, d = (1, 4, 7, "Hello")
typeof(a_tuple)
# argument list of a function is a tuple
# index tuples in eh same way as array
a_tuple[end-1]
# iterate over a tuple
for i = a_tuple
    println(i)
end
```

In [ ]:

```
## Dictionaries
# type Dict
# two element tuples with (k,v)
dict1 = Dict(:Author => 100, :year => 2000)
dict1
## extract author
dict1[:Author]
in((:Author=>100), dict1)
# dictionaries are mutable, change
dict1[:Entry]=2010
dict1
empty_dictionary = Dict()
```

In [ ]:

```
## get a listing of hte keys
#for k in keys(dict1) println(k) end
## get a listing of the values

#for v in values(dict1) println(v) end

## two ways to test if a key exists
# :A in keys(dict1) # will return false
# or use haskey
# haskey(dict1, :A) # will also return false
# If you want to get the keys
key_list = collect(keys(dict1))
# if you want to get the values
value_list = collect(values(dict1))

## creating and using dictionaries
keys1 = ["A", "B", "C"]; vals1 = [1, 2, 3]
dict2 = Dict(zip(keys1, vals1))
dict2
## Loop over
for d in dict2
    println("$(d[1]) is valued at $(d[2])")
end
```

In [ ]:

```
## Sets
# can contain duplicate elements
s= Set() # create an empty Set
s = Set([1, 2, 3, 4, 1, 4, 3, 2])
```

In [ ]:

```
## How to convert between data types
x = 7
Float64(x) # converts x from Int to Float, also
convert(Float64, x) # do the same
```

In [ ]:

```
## You can create your own type
## Let's say we want to create type Human for epi
# we need age, and gender for that type

type Human
    age::Int64
    gender::AbstractString
end

## let's say we have data,
h = Human(32, "f")
typeof(h)

## Julia has no classes
# what fields?
names(Human)
methods(Human)
```

In [ ]:

```
## Are the two values equal or identical

x = 3; y = 3.0
x == y # will be true, one way to check
is(x,y) # will be false, as x is Int and y is Float
```

In [ ]:

```
## Modules and Paths
# code of julia packages (such as GLM.jl) contained
# in module
# name of module starts with uppercase
# Epi
# module Epi
# code
# end

## type of Main
typeof(Main)
```

In [ ]:

```
names(Main)
```

In [ ]:

```
whos
```

In [ ]:

```
whos()
```

In [ ]:

```
whos(Gadfly)
```

In [ ]:

```
using Gadfly
```

In [ ]:

```
whos(Gadfly)
```

In [ ]:

```
LOAD_PATH
```

Metaprogramming in Julia

- Everything in Julia is an expression
- All expressions return values when executed
- Every piece of programme code is represented
- Julia data structure
- Expression = Julia data structure

Concept of Homoiconicity

- We work on Julia expressions
- then we generate new code
- we can transform these expressions
- "Code is data and data is code"

Code blocks

((c * 9)/5 + 32)

- Julia cannot evaluate that as we donot know yet what is c
- if we want to block evaluation but keep expression,
- we do: :((c * 9)/5 + 32)
- So, this argument is data, not code
- If want to have more than one line for expression, se do:
- quote [write the expressions] end

In [ ]:

```
function celstof(c::Int64)
    c = Float64(c)
    return ((c * 9)/5) + 32
end

celstof(29)

quote
    c = Float64(c)
    return ((c * 9)/5) + 32
end

## we can assign a name to it

e1 = quote
    c = Float64(c)
    return ((c * 9)/5) + 32
end

## what are the fields of e1?
names(e1) ## you will see :head, :args, :typ
# let's check out
e1.head # returns a block expression
e1.args
dump(e1)
```

In [ ]:

```
## concept of expression
## if we type this:
# (cx * 9)/5 + 32
# this will result in error as julia does not know
# what is cx
# if we would like to stop Julia evaluate this, we do:
e2 = :((cx * 9) / 5  + 32)
# now julia knows it is an expression
```

In [ ]:

```
## Eval and interpolation
e3 = Expr(:call, *, 3, 4)
eval(e3)
## But
e4 = Expr(:call, *, 3, aaaa)
#eval(e4) # will return error, because aaaa is not known
aaaa = 5 # will now yield value
#eval(e4)
aaaa
eval(e4)

## Interpolation

www = 4; yyy = 6;
e5 = :(www + yyy) # is an expression
## what about
e6 = :($www + yyy) ## will be :(4 + yyy)
```

In [11]:

```
## Macros
# Macros are functions
# Instead of values, they take expressions
# Macro returns a modified expression
## macro mname
## code here ...
## end

@which Histogram # @which is a macro and Histogram is a function name
```

Out[11]:

```
Winston
```

In [ ]:

```
## Built in macros in Julia
@assert 1 != 3
@which sort # tells you where a method can be found
@which Array
@show π * 5^2 # shows the expression and result
@time [x*2 for x in 1:100] # how much time elapsed
@elapsed [x*2 for x in 1:100] # same as above
#hello = @async
# starting a task, then consume()
consume(hello)
```

Data Frames in Julia

- Allows for NA
- DataArray -- can contain missing values
- DataFrame for tabular data sets
- Each column of the DataFrame is a DataArray

In [ ]:

```
using DataArrays, DataFrames, RDatasets
```

In [15]:

```
dv = @data [NA, 3, 2, 5, 4];
dv = dropna(dv);
mean(dv)

dm = @data [NA 0.0; 0.0 1.0]
dm * dm

## Build a data frame
df = DataFrame() # initialise an empty data frame
df[:A] = 1:8 # add variable A 1...8
df[:B] = ["m", "f", "m", "f", "m", "f", "m", "f",]
nrows = size(df, 1)
describe(df)
mean(df[:A]); median(df[:A])

df2 = DataFrame(A = 1:4, B = 2:5); colwise(cumsum, df2)
#df2

iris = DataFrame(dataset("datasets", "iris"))
head(iris)

## select SepalWidth > 2
## Remember to use element wise operation
# otherwise doesn't work

iris2 = iris[iris[:SepalWidth] .> 3.0, :]

## joining data bases
a = DataFrame(ID = [1,2], Name = ["A", "B"])
b = DataFrame(ID = [1, 3], Job = ["Doc", "Law"])
join(a, b, on = :ID, kind = :inner) #joins only on common ID
join(a, b, on = :ID, kind = :left)
join(a, b, on = :ID, kind = :right)
join(a, b, on = :ID, kind = :outer)
join(a, b, on = :ID, kind = :semi)
join(a, b, on = :ID, kind = :anti)
join(a, b, kind = :cross) # doesn't use a key

## Split-Apply-Combine Strategy
# use the by function
# DataFrame, column to split dataframe,
# function or expression
```

```
# function or expression

by(iris, :Species, size) #how many rows and columns of each
by(iris, :Species, pl -> mean(pl[:PetalLength]))
# after splitting up the iris data, compute mean
by(iris, :Species, df -> DataFrame(N = size(df, 1)))
# after splitting up iris by Species,
# how many rows of data did you get

## use a do block
by(iris, :Species) do df
    DataFrame(m = mean(df[:PetalLength]),
    sd = std(df[:PetalLength]))
end

## aggregate
# takes dataframe, column or cols, function or functions

aggregate(iris, :Species, [std, mean])

## if you want to split the data into subgroups

for subdf in groupby(iris, :Species)
    println(size(subdf, 1))
end

## Reshape data

iris[:id] = 1:size(iris, 1) #create an id variable
d = stack(iris, [:SepalLength, :SepalWidth], :Species)
#d1 = melt(iris, :Species)
d = stackdf(iris)
unstack(d)

d = stackdf(iris)
x = by(d, [:variable, :Species],
df -> DataFrame(vsum = mean(df[:value])))
unstack(x, :Species, :vsum)

## Model Matrix

df = DataFrame(X = randn(10),
Y = randn(10), Z = randn(10))
mf = ModelFrame(Z ~ X + Y, df)
mm = ModelMatrix(mf)

## ModelMatrix witih interaction terms
mf2 = ModelFrame(Z ~ X * Y, df)
mm2 = ModelMatrix(mf2)

## Pool Data
dv = @data(["gra", "gra", "gra",
"grb", "grb", "grb"])
## Pooled data
pdv = @pdata(["gra", "gra", "gra",
"grb", "grb", "grb"])
levels(pdv)
pdv = compact(pdv)
```

```
pdv1 = pool(dv)
```

```
A

Out[15]:

6-element DataArrays.PooledDataArray{ASCIIString,UInt8,1}:
 "gra"
 "gra"
 "gra"
 "grb"
 "grb"
 "grb"

Min       1.0
1st Qu.   2.75
Median    4.5
Mean      4.5
3rd Qu.   6.25
Max       8.0
NAs       0
NA%       0.0%

B
Length  8
Type    ASCIIString
NAs     0
NA%     0.0%
Unique  2
```

```
50
50
50
```

In [13]:

```
## GLM in Julia

## Call the GLM package
using GLM, RDatasets

# Needs formula, data, family, link
# Methods: coef, deviance, df_residual,
# glm, lm, stderr, vcov = variance covariance matrix,
# predict = predicted values

## Simple linear regression model

form = dataset("datasets", "formaldehyde")
lm1 = fit(LinearModel, OptDen ~ Carb, form)
ols = glm(OptDen ~ Carb, form, Normal(), IdentityLink())
```

```
WARNING: Base.String is deprecated, use AbstractString instead.
  likely near /Users/arindambose/.julia/v0.4/RDatasets/src/dataset
.jl:1
WARNING: Base.String is deprecated, use AbstractString instead.
  likely near /Users/arindambose/.julia/v0.4/RDatasets/src/dataset
.jl:1
WARNING: Base.String is deprecated, use AbstractString instead.
  likely near /Users/arindambose/.julia/v0.4/RDatasets/src/dataset
s.jl:1
```

Out[13]:

```
DataFrames.DataFrameRegressionModel{GLM.GeneralizedLinearModel{GLM
.GlmResp{Array{Float64,1},Distributions.Normal,GLM.IdentityLink},G
LM.DensePredChol{Float64,Base.LinAlg.Cholesky{Float64,Array{Float6
4,2}}}},Float64}:

Coefficients:
              Estimate  Std.Error  z value Pr(>|z|)
(Intercept)  0.00508571 0.00783368 0.649211   0.5162
Carb           0.876286  0.0135345  64.7444   <1e-99
```

In [14]:

```
# Get the predicted values and confidence intervals
predicted_values = predict(lm1)
conf_int = confint(lm1)
regression_coefficients = coef(lm1)

predict(ols)
coeftable(lm1)
residuals(lm1)
```

```
WARNING: could not attach metadata for @simd loop.
```

Out[14]:

```
6-element Array{Float64,1}:
 -0.00671429
  0.00102857
  0.00277143
  0.00714286
  0.00751429
 -0.0117429
```

In [40]:

```
## Statsbase
# Basic support for statistics
# Calculation of mean
x = randn(100)
#w = rand(10)
#y = mean(x, weights(w))

mean_and_variance = mean_and_var(x)
zscore(x)

#Histogram

fit(Histogram, iris[:SepalLength])
```

Out[40]:

```
StatsBase.Histogram{Int64,1,Tuple{FloatRange{Float64}}}
edges:
  4.0:0.5:8.0
weights: [5,27,27,30,31,18,6,6]
closed: right
```

In [15]:

```
## Using Gadfly in Julia


## Call Gadfly and Cairo so that you can plot
## to backend as well
using Gadfly, Cairo
## most interaction is with the plot function
## aesthetics, scales, coordinates, guides, geometries
```

In [ ]:

```
## Point geometry default
## simplest
Gadfly.plot(x = rand(10), y = rand(10)) # We need to specify this as we have b
oth Winston
# and Gadfly open. Otherwise just plot would do
## Now we add a line to it
plot(x = rand(10), y = rand(10),
Geom.point, Geom.line)
# these results are layered
```

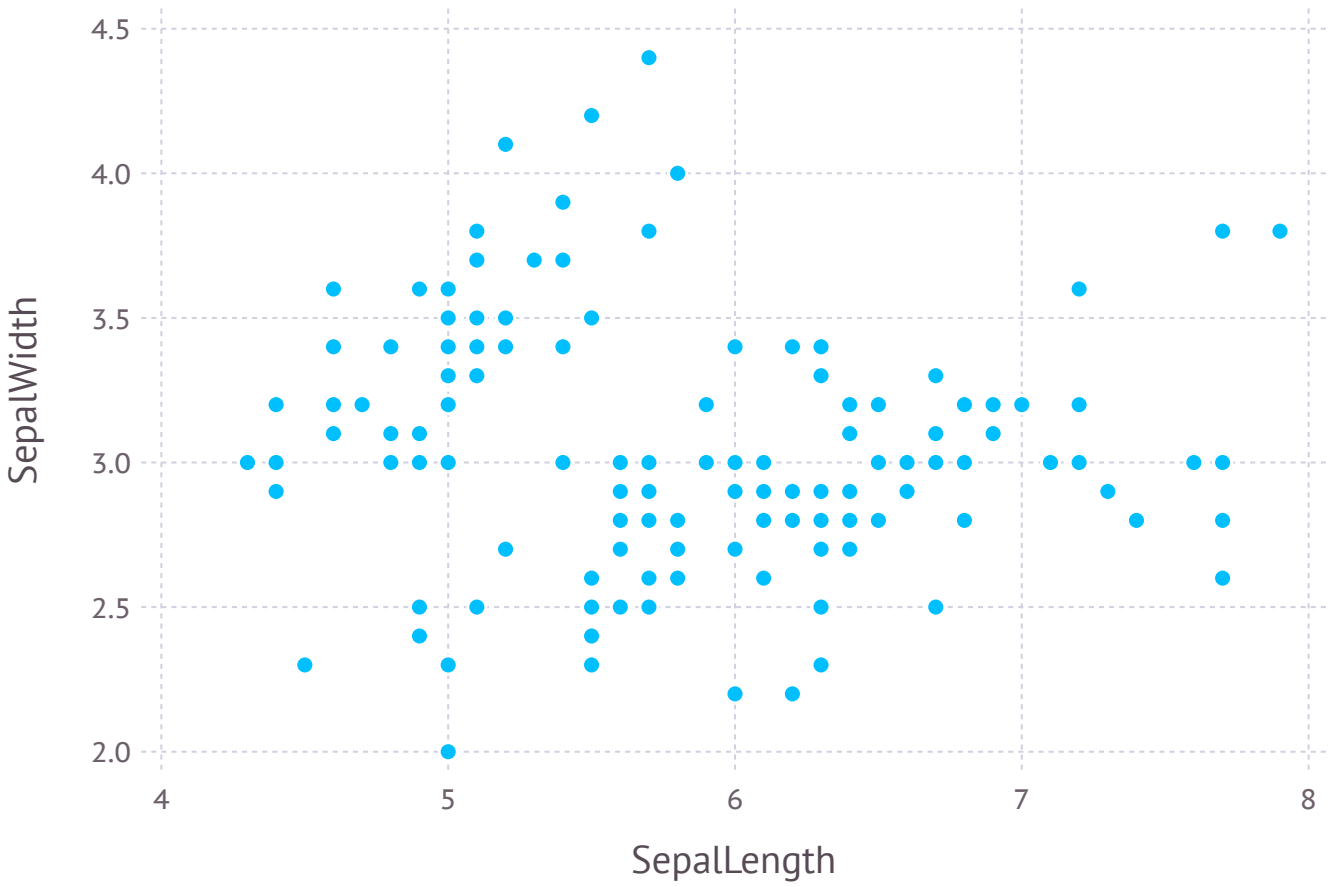
In [ ]:

```
## Add a title, xlabel, ylabel, fit line
plot(x = rand(10), y = rand(10),
Geom.point, Geom.smooth(method=:lm),
Guide.title("Random plot"),
Guide.ylabel("Y values"),
Guide.xlabel("X values"))

# push this plot to an image
# first, save it to an object
myplot = plot(x = rand(10), y = rand(10),
Geom.point, Geom.smooth(method=:lm),
Guide.title("Random plot"),
Guide.ylabel("Y values"),
Guide.xlabel("X values"))

## then use the draw fnction

draw(PNG("myplot.png", 4inch, 3inch), myplot)
```

In [17]:

```
## How to plot data frames

using RDatasets
## we shall read iris

iris = dataset("datasets", "iris")

## Simple plotting of a scatterplot
myplot2 = Gadfly.plot(iris, x = "SepalLength",
y = "SepalWidth", Geom.point)
```

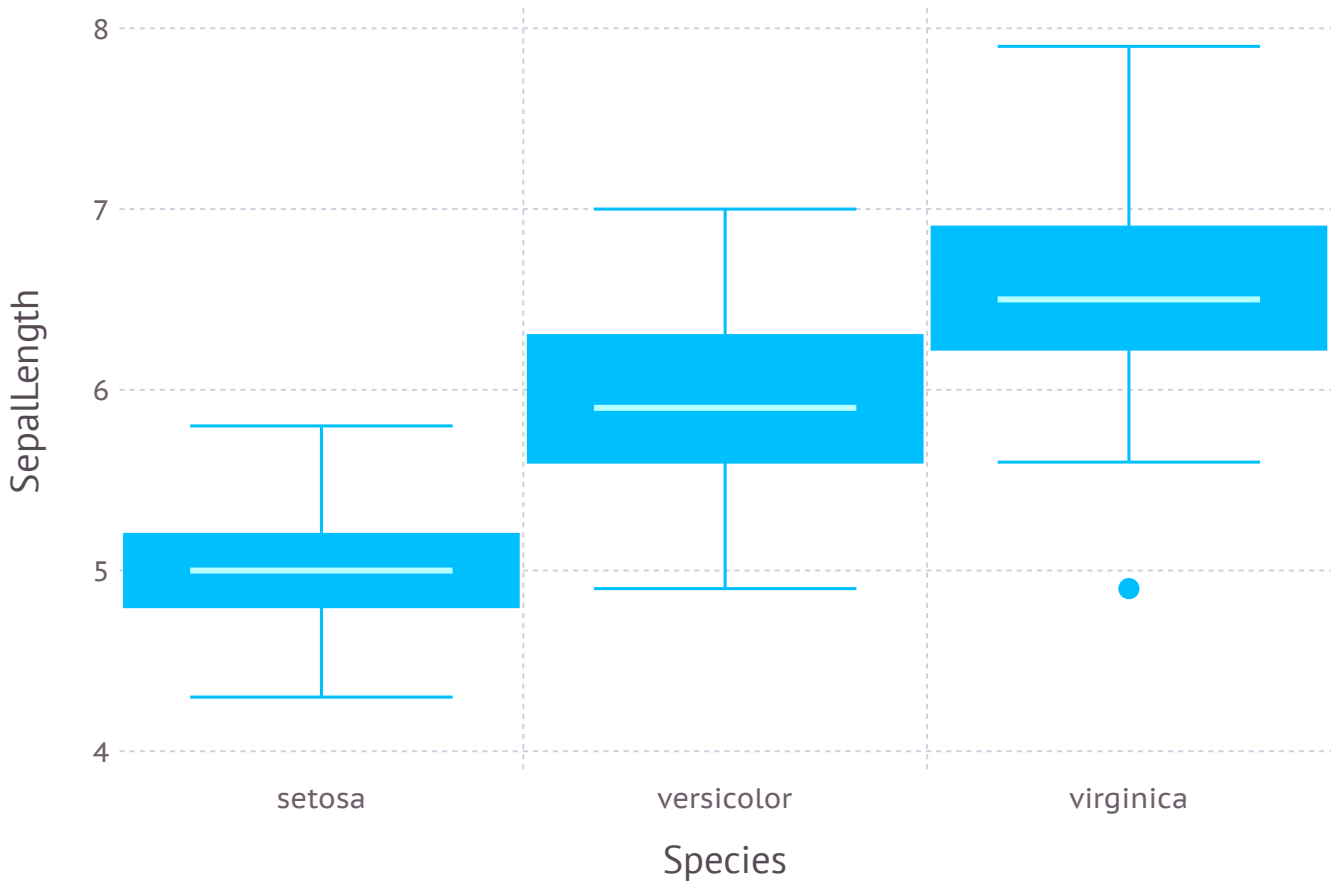
Out[17]:



In [18]:

```
## Boxplots
myplot2 = Gadfly.plot(iris, x = "Species",
y="SepalLength", Geom.boxplot)
```

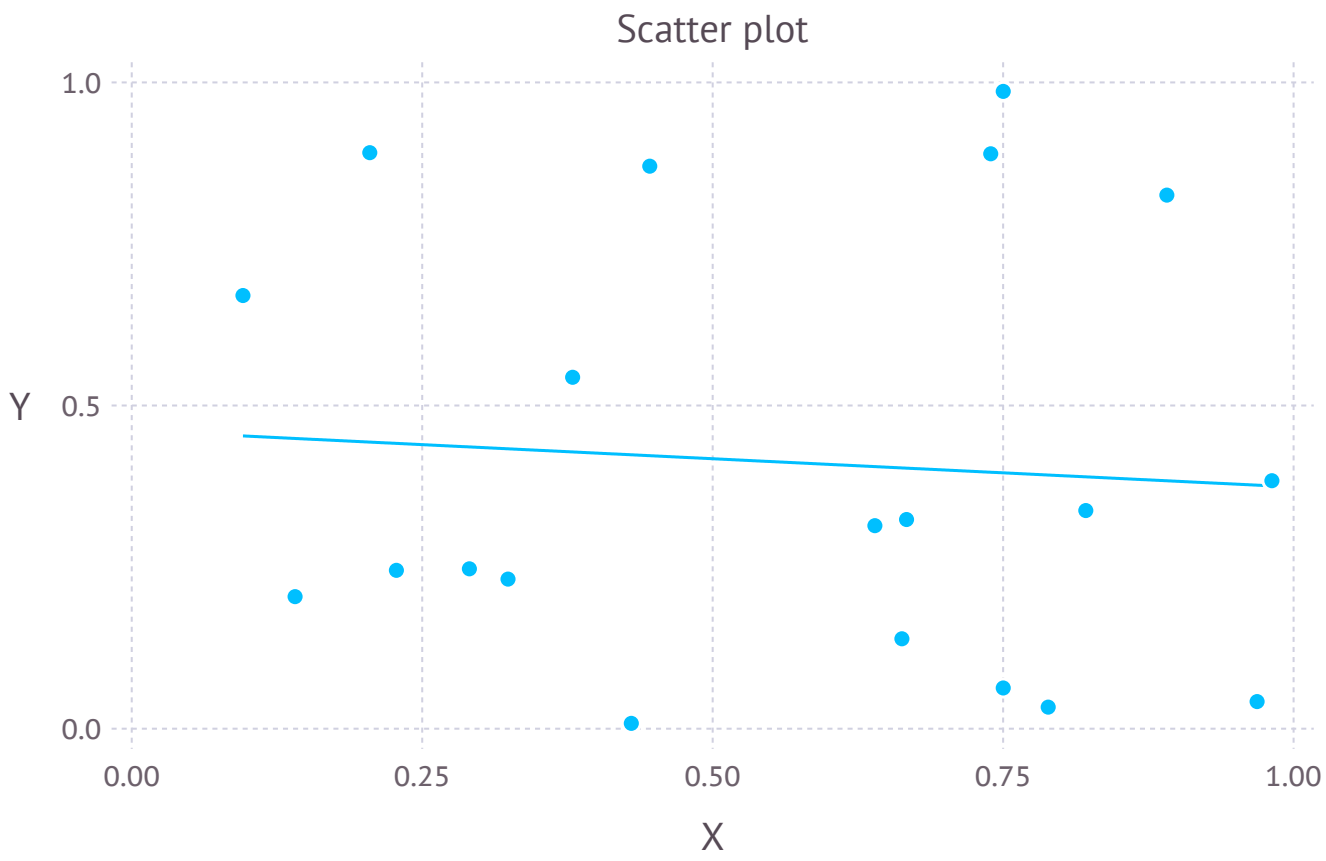
Out[18]:



In [20]:

```
## Plotting functions
x = rand(20)
Gadfly.plot((y=2*x +1), x = rand(20),
y = rand(20), Geom.point, Geom.smooth(method=:lm),
Guide.title("Scatter plot"),
Guide.xlabel("X"),
Guide.ylabel("Y"))
```

Out[20]:



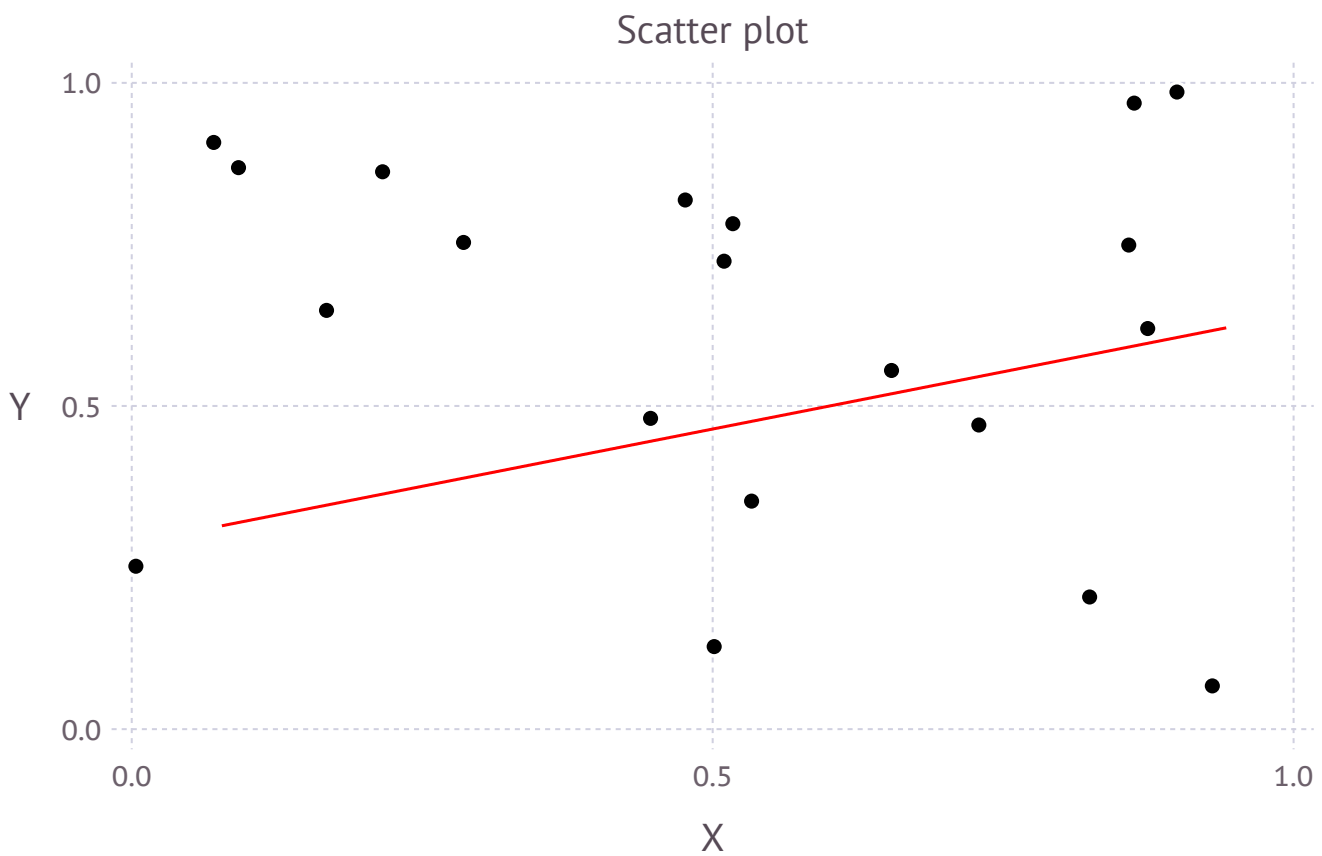
In [ ]:

```
## We can customise the plot appearance with Theme
## Change from blue to black

plot((y=2*x +1), x = rand(20),
y = rand(20), Geom.point, Geom.smooth(method=:lm),
Guide.title("Scatter plot"),
Guide.xlabel("X"),
Guide.ylabel("Y"),
Theme(default_color=colorant"Black"))
```

In [22]:

```
## We can add layers to turn the fit line to blue
## for the first layer we have black theme
## for the second layer, we have red theme

Gadfly.plot(layer((y=2*x +1), x = rand(20),
y = rand(20), Geom.point,
Theme(default_color=colorant"Black")),

layer((y=2*x +1), x = rand(20),
y = rand(20), Geom.smooth(method=:lm),
Theme(default_color=colorant"Red")),

Guide.title("Scatter plot"),
Guide.xlabel("X"),
Guide.ylabel("Y")

)
```

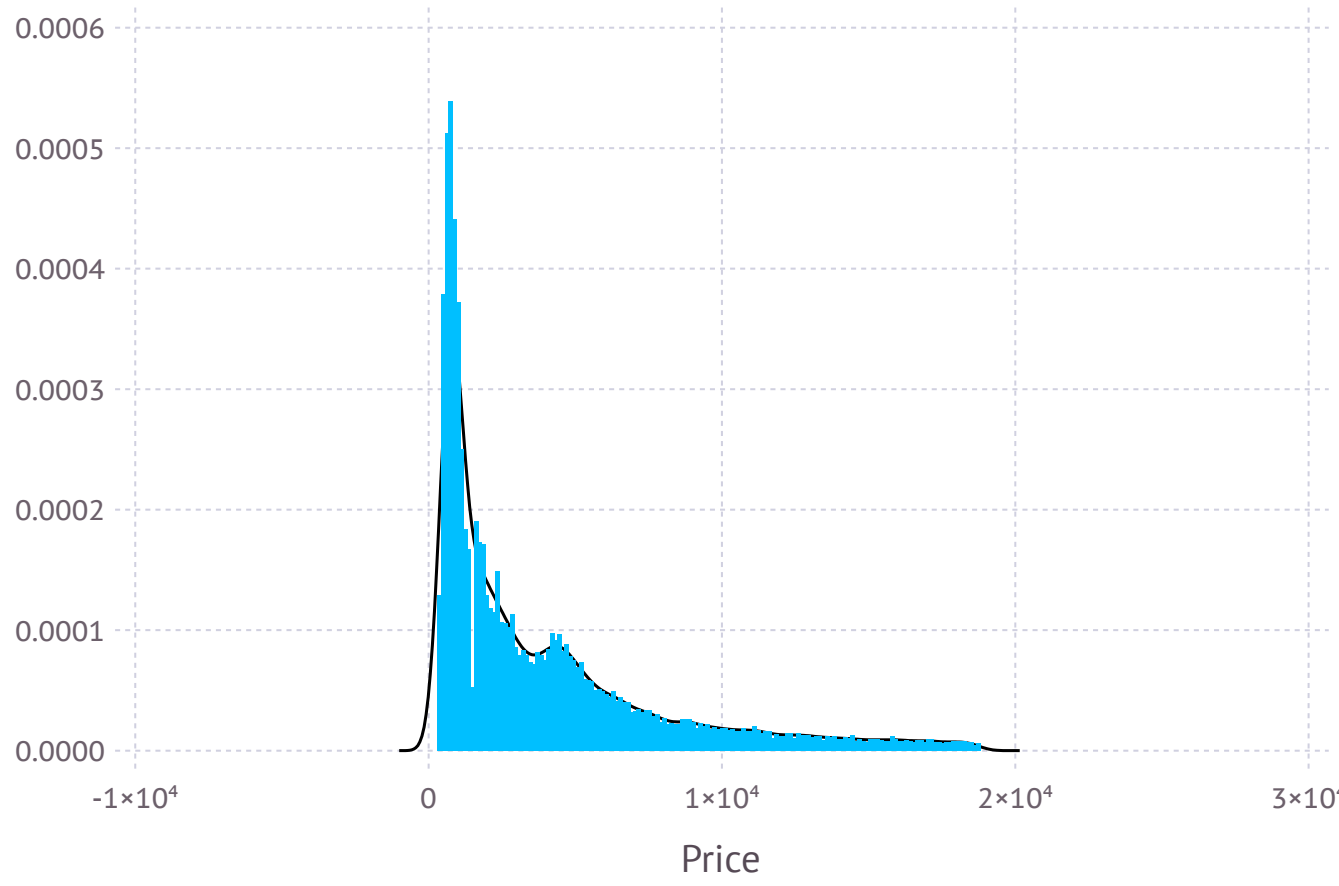
Out[22]:



In [97]:

```
## Histogram with density plot

plot(layer(dataset("ggplot2", "diamonds"),
x="Price", Geom.histogram(density=:true)),
layer(dataset("ggplot2", "diamonds"),
x="Price", Geom.density,
Theme(default_color=colorant"Black")))
```

Out[97]:



In [ ]: