Statistics with Julia from the ground up

A JuliaCon 2021 (https://juliacon.org/2021/) workshop by Yoni Nazarathy (https://yoninazarathy.com/)

Many of the code examples for this workshop are adapted from Statistics with Julia: Fundamentals for Data Science, Machine Learning and Artificial Intelligence by Yoni Nazarathy and Hayden Klok (https://statisticswithjulia.org/).

Also related (and recommended in this JuliaCon):

- Dataframes tutorial (https://github.com/bkamins/JuliaCon2021-DataFrames-Tutorial) or here (https://pretalx.com/juliacon2021/talk/FXZXMB/) by Bogumił Kamiński.
- Introduction to Bayesian Data Analysis (https://pretalx.com/juliacon2021/talk/J7BFBM/) by Kusti Skytén.
- Dozens of other very exciting talks...

Table of Contents

7. We love `DataFrames`.
8. Gotta have some basic inference.
9. Linear models at our core.
10. Basic Machine learning.

Before we start

The tutorial was developed and tested under Julia 1.6.0. It is best to run it with the `Project.toml` and `Manifest.toml` files present in the working directory of the notebook. It also uses the following data files:

In [1]:
```
readdir("./data")
```

Out[1]:
```
8-element Vector{String}:
 "L1L2data.csv"
 "fertilizer.csv"
 "machine1.csv"
 "machine2.csv"
 "machine3.csv"
 "purchaseData.csv"
 "temperatures.csv"
 "weightHeight.csv"
```

You will find all these files and this notebook in the Github repo for this workshop (https://github.com/yoninazarathy/JuliaCon2021-StatisticsWithJuliaFromTheGroundUp). You can either "clone" the repo or download a zip file.

We use a Jupyter Notebook (https://jupyter.org/). Here is a qjuuick reference sheet (edureka.co/blog/wp-content/uploads/2018/10/Jupyter_Notebook_CheatSheet_Edureka.pdf). Many other reserouces on the web as well for Jupyter - many of which use Python and not Julia, but Jupyter is the same. BTW the "J" in "Jupyter" is for "Julia".

Load the packages we will use:

In [2]:
```
using Pkg
Pkg.activate(".")
Pkg.instantiate()
```

```
  Activating environment at `~/git/mine/JuliaCon2021-StatisticsWithJuli
```

In [3]:
```
Pkg.status()
```

```
      Status `~/git/mine/JuliaCon2021-StatisticsWithJuliaFromTheGroundU
  [336ed68f] CSV v0.8.5
  [aaaa29a8] Clustering v0.14.2
  [861a8166] Combinatorics v1.0.2
  [a93c6f00] DataFrames v1.2.0
  [31c24e10] Distributions v0.25.11
  [587475ba] Flux v0.12.5
  [38e38edf] GLM v1.5.1
  [09f84164] HypothesisTests v0.10.4
  [5ab0869b] KernelDensity v0.6.3
  [b964fa9f] LaTeXStrings v1.2.1
  [b4fcebef] Lasso v0.6.2
  [eb30cadb] MLDatasets v0.5.7
  [442fdcdd] Measures v0.3.1
  [dbeba491] Metalhead v0.5.3
  [6f286f6a] MultivariateStats v0.8.0
  [91a5bcdd] Plots v1.19.2
  [ce6b1742] RDatasets v0.7.5
  [f2b01f46] Roots v1.0.10
  [276daf66] SpecialFunctions v1.5.1
  [2913bbd2] StatsBase v0.33.8
  [f3b207a7] StatsPlots v0.14.25
```

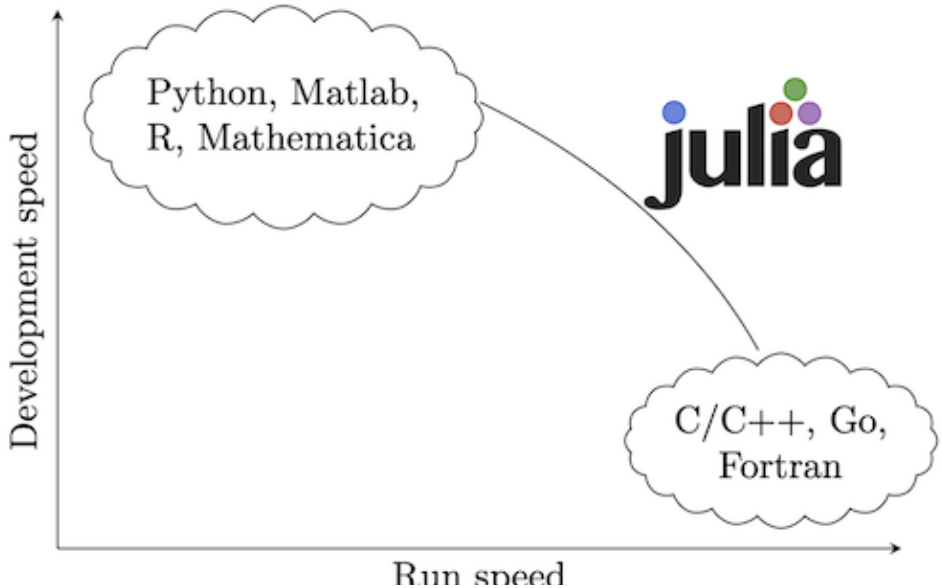
In [4]:
```
using Random, Statistics, LinearAlgebra, Dates #Shipped with Julia
using Distributions, StatsBase #Core statistics
using CSV, DataFrames #Basic Data
using Plots, StatsPlots, LaTeXStrings, Measures #Plotting and Output
using HypothesisTests, KernelDensity, GLM, Lasso, Clustering, Multivaria
using Flux, Metalhead #Deep learning
using Combinatorics, SpecialFunctions, Roots #Mathematical misc.
using RDatasets, MLDatasets #Example datasets
#uncomment if using R:  using RCall #Interface with R
```

In [5]:
```
# We run this before many examples for reproducibility
fix_seed!() = Random.seed!(0)
```

Out[5]:
```
fix_seed! (generic function with 1 method)
```

Why Julia?

home



Some ways to run Julia

- REPL
  - As an application
  - Out of your shell
  - As part of an IDE
- Jupyter (IJulia)
  - In your web browser
  - Jupyter Lab
- Google collab
- Pluto
- Visual Studio Code
- Legacy: Atom (Juno)
- JuliaHub
- In RMarkdown with IJulia (e.g. in R studio)
- ...

Key resources (My favorites)

- Main Julia Page: https://julialang.org/ (https://julialang.org/)
- Docs: https://docs.julialang.org/ (https://docs.julialang.org/)
- Julia Express: https://github.com/bkamins/The-Julia-Express (https://github.com/bkamins/The-Julia-Express)
- Think Julia: https://www.oreilly.com/library/view/think-julia/9781492045021/ (https://www.oreilly.com/library/view/think-julia/9781492045021/)
- MIT Course, computational thinking: https://computationalthinking.mit.edu/Spring21/ (https://computationalthinking.mit.edu/Spring21/)
- A University of Queensland Course: https://courses.smp.uq.edu.au/MATH2504/ (https://courses.smp.uq.edu.au/MATH2504/)
- Statistics with Julia: https://statisticswithjulia.org/ (https://statisticswithjulia.org/) (use image gallary)
- Package documentation: Searching for the package, e.g. `Plots.jl` , typically gets you to GitHub. From there find the docs.
- Julia Discourse: https://discourse.julialang.org/ (https://discourse.julialang.org/)
- Julia Slack: https://julialang.org/slack/ (https://julialang.org/slack/)
- Your local Julia "club": E.g. in my area: https://www.meetup.com/en-AU/brisbane-julia-language-meetup/ (https://www.meetup.com/en-AU/brisbane-julia-language-meetup/)
- YouTube...

What do you `mean`?

home

In [6]:
```
fix_seed!()
data = rand(Normal(),5)
```

Out[6]:
```
5-element Vector{Float64}:
  0.6791074260357777
  0.8284134829000359
 -0.3530074003005963
 -0.13485387193052173
  0.5866170746331097
```

```
In [7]: n = length(data)
Out[7]: 5

In [8]: sum(data)/n
Out[8]: 0.3212553422675611

In [9]: +(data...)/n
Out[9]: 0.3212553422675611
```

In [10]:
```
? +
```

```
search: +
```

Out[10]:
```
+(x, y...)
```

Addition operator. `x+y+z+...` calls this function with all arguments, i.e. `+(x, y, z, ...)`.

Examples

```
jldoctest
julia> 1 + 20 + 4
25

julia> +(1, 20, 4)
25
```

```
dt::Date + t::Time -> DateTime
```

The addition of a `Date` with a `Time` produces a `DateTime`. The hour, minute, second, and millisecond parts of the `Time` are used along with the year, month, and day of the `Date` to create the new `DateTime`. Non-zero microseconds or nanoseconds in the `Time` type will result in an `InexactError` being thrown.

```
bb1 + bb2 -> bb
```

Compute the BoundingBox `bb` that minimally contains `bb1` and `bb2`

In [11]:
```
"""
This is a function that takes in data and returns its sum.
"""
function my_sum(data)
    s = 0.0
    for d in data
        s += d
    end
    return s
end
```

Out[11]:
```
my_sum
```

In [12]:
```
? my_sum
```

```
search: my_sum
```

Out[12]: This is a function that takes in data and returns its sum.

In [13]:
```
my_sum(data)/length(data)
```

Out[13]:
```
0.3212553422675611
```

In [14]:
```
mean(data)
```

Out[14]:
```
0.3212553422675611
```

In [15]:
```
data'*ones(n)/n
```

Out[15]:
```
0.3212553422675611
```

In [16]:
```
dot(data,ones(n))/n
```

Out[16]:
```
0.3212553422675611
```

In [17]:
```
? dot
```

```
search: dot dotplot dotplot! @__dot__ stdout dropterm DEPOT_PATH groupe
```

Out[17]:
```
dot(x, y)
x · y
```

Compute the dot product between two vectors. For complex vectors, the first vector is conjugated.

`dot` also works on arbitrary iterable objects, including arrays of any dimension, as long as `dot` is defined on the elements.

`dot` is semantically equivalent to `sum(dot(vx,vy) for (vx,vy) in zip(x, y))`, with the added restriction that the arguments must have equal lengths.

`x · y` (where `·` can be typed by tab-completing `\cdot` in the REPL) is a synonym for `dot(x, y)`.

Examples

```
jldoctest
julia> dot([1; 1], [2; 3])
5

julia> dot([im; im], [1; 1])
0 - 2im

julia> dot(1:5, 2:6)
70

julia> x = fill(2., (5,5));

julia> y = fill(3., (5,5));

julia> dot(x, y)
150.0
```

```
dot(x, A, y)
```

Compute the generalized dot product `dot(x, A*y)` between two vectors `x` and `y`, without storing the intermediate result of `A*y`. As for the two-argument `dot(_,_)` (@ref), this acts recursively. Moreover, for complex vectors, the first vector is conjugated.

!!! compat "Julia 1.4" Three-argument `dot` requires at least Julia 1.4.

Examples

```
jldoctest
julia> dot([1; 1], [1 2; 3 4], [2; 3])
26

julia> dot(1:5, reshape(1:25, 5, 5), 2:6)
4850

julia> ⋅(1:5, reshape(1:25, 5, 5), 2:6) == dot(1:5, reshape(1:25, 5,
5), 2:6)
true
```

Doing it a little differently with the "running mean" formula

[Math Processing Error]

In [18]:
```
mn = 0
for i in 1:length(data)
    global mn = (1/i)*data[i] + (i-1)/i*mn
end
mn
```

Out[18]:
```
0.32125534226756114
```

In [19]:
```
mn = 0
for (i,d) in enumerate(data)
    global mn = (1/i)*d + (i-1)/i*mn #Note that in Jupyter `global` isn
end
mn
```

Out[19]:
```
0.32125534226756114
```

In [20]:
```
function my_mean(data)
    mn = 0
    for (i,d) in enumerate(data)
        mn = (1/i)*d + (i-1)/i*mn
    end
    return mn
end
```

Out[20]:
```
my_mean (generic function with 1 method)
```

In [21]:
```
my_mean(data)
```

Out[21]:
```
0.32125534226756114
```

In [22]:
```
fix_seed!()
data = rand(Normal(),n) + im*rand(Normal(),n)
```

Out[22]:
```
5-element Vector{ComplexF64}:
   0.6791074260357777 + 0.29733585084941616im
   0.8284134829000359 + 0.06494754854834232im
  -0.3530074003005963 - 0.10901738508171745im
 -0.13485387193052173 - 0.514210390833322im
   0.5866170746331097 + 1.5743302021369892im
```

In [23]:
```
mean(data)
```

Out[23]:
```
0.3212553422675611 + 0.2626771651239416im
```

In [24]:
```
methods(mean)
```

Out[24]: # 105 methods for generic function **mean**:

- mean(M::**PPCA**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/ppca.jl:15 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/ppca.jl)
- mean(d::**Geometric**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/geometric.jl:55 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/geometric.j
- mean(M::**MulticlassLDA**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/lda.jl:131 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/lda.jl)
- mean(d::**InverseGamma{T}**) *where T* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/inversegamm (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/inverseg
- mean(d::**Normal**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/normal.jl:67 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/normal.j
- mean(d::**PoissonBinomial**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/poissonbinomial (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/poissonbin
- mean(d::**FisherNoncentralHypergeometric**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/noncentralhyper (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/noncentrall
- mean(M::**PCA**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/pca.jl:31 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/pca.jl)
- mean(d::**Chi**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/chi.jl:51 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/chi.jl)
- mean(d::**EdgeworthSum**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/edgeworth.jl:87 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/edgeworth.jl)
- mean(d::**Triweight**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/triweight.jl:36 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/triweigh

- mean(d::**Product**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/product.jl:38 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/product.jl)
- mean(d::**NormalCanon**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/normalcanon. (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/normalc
- mean(d::**Laplace**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/laplace.jl:63 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/laplace.
- mean(d::**Biweight**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/biweight.jl:27 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/biweigh
- mean(r::**AbstractRange{var"#s832"} where var"#s832"<:Real**) in Statistics at /Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/stdlib/v1.6/Statistics/src/Statistics.jl:185 (file:///Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/v1.6/Statisti
- mean(d::**TDist{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/tdist.jl:50 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/tdist.jl)
- mean(S::**MulticlassLDAStats**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/lda.jl:74 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/lda.jl)
- mean(d::**Epanechnikov**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/epanechniko (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/epanecl
- mean(d::**Beta**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/beta.jl:63 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/beta.jl)
- mean(d::**Poisson**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/poisson.jl:52 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/poisson.jl)
- mean(M::**SubspaceLDA**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/lda.jl:222 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/lda.jl)
- mean(f::**Function**, a::**StaticArrays.StaticArray**; *dims*) in StaticArrays at /Users/uqjnazar/.julia/packages/StaticArrays/AHT47/src/mapreduce.jl:275 (file:///Users/uqjnazar/.julia/packages/StaticArrays/AHT47/src/mapreduce.jl)

- mean(d::**Cauchy{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/cauchy.jl:62 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/cauchy.
- mean(d::**PGeneralizedGaussian**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/pgeneralized((file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/pgenera
- mean(d::**MvNormalCanon**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvnormalcanon.jl:151 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvnormalcanon.jl
- mean(d::**Skellam**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/skellam.jl:60 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/skellam.jl)
- mean(d::**UnivariateGMM**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/mixtures/unigmm.jl:26 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/mixtures/unigmm.jl)
- mean(d::**MultivariateMixture{S, C} where {S<:ValueSupport, C<:Distribution}**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/mixtures/mixturemodel.jl:179 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/mixtures/mixturemodel.jl)
- mean(d::**UnivariateMixture{S, C} where {S<:ValueSupport, C<:Distribution}**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/mixtures/mixturemodel.jl:173 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/mixtures/mixturemodel.jl)
- mean(f::**Whitening**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/whiten.jl:36 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/whiten.jl)
- mean(d::**MatrixReshaped**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixreshaped.jl:54 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixreshaped.jl)
- mean(d::**Binomial**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/binomial.jl:68 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/binomial.jl)
- mean(d::**MatrixTDist**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixtdist.jl:106 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixtdist.jl)
- mean(d::**InverseGaussian**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/inversegauss

(file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/inverseg

- mean(d::**Distributions.GenericMvTDist**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvtdist.jl:89 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvtdist.jl)
- mean(d::**SymTriangularDist**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/symtriangular (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/symtriar
- mean(d::**LogNormal**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/lognormal.jl:6 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/lognorm
- mean(d::**Levy{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/levy.jl:55 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/levy.jl)
- mean(d::**Erlang**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/erlang.jl:61 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/erlang.jl
- mean(d::**VonMises**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/vonmises.jl:5 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/vonmise
- mean(d::**LocationScale**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/locationscale.jl:85 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/locationscale.jl)
- mean(a::**StaticArrays.StaticArray**; *dims*) in StaticArrays at /Users/uqjnazar/.julia/packages/StaticArrays/AHT47/src/mapreduce.jl:274 (file:///Users/uqjnazar/.julia/packages/StaticArrays/AHT47/src/mapreduce.jl)
- mean(d::**DirichletMultinomial**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/dirichletmultinomial.jl:2 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/dirichletmultinomi
- mean(d::**EdgeworthZ**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/edgeworth.jl:23 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/edgeworth.jl)
- mean(d::**Rayleigh**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/rayleigh.jl:55 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/rayleigh
- mean(d::**Bernoulli**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/bernoulli.jl:60 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/bernoulli.jl)

- mean(d::**Exponential**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/exponential.jl (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/exponer
- mean(d::**MatrixFDist**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixfdist.jl:90 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixfdist.jl)
- mean(d::**GeneralizedPareto{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/generalizedpa (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/generali
- mean(d::**Wishart**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/wishart.jl:108 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/wishart.jl)
- mean(d::**MvNormal**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvnormal.jl:237 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvnormal.jl)
- mean(d::**Semicircle**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/semicircle.jl:3 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/semicirc
- mean(d::**MatrixBeta**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixbeta.jl:84 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixbeta.jl)
- mean(d::**Truncated{var"#s464", Continuous, T} where {var"#s464"<:Exponential, T<:Real}**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/truncated/exponential.jl:5 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/truncated/exponential.jl)
- mean(d::**Truncated{Normal{T}, Continuous, T1} where T1<:Real**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/truncated/normal.jl:96 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/truncated/normal.jl)
- mean(d::**NoncentralT{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/noncentralt.jl: (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/noncent
- mean(d::**Chisq**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/chisq.jl:50 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/chisq.jl)
- mean(d::**Dirac**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/dirac.jl:33

(file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/dirac.jl)

- mean(d::**Cosine**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/cosine.jl:47 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/cosine.j
- mean(d::**Uniform**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/uniform.jl:58 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/uniform
- mean(d::**LKJ**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/lkj.jl:71 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/lkj.jl)
- mean(d::**SkewNormal**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/skewnormal.j (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/skewno
- mean(d::**NoncentralChisq**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/noncentralchi (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/noncent
- mean(d::**Gamma**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/gamma.jl:60 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/gamma
- mean(d::**NoncentralF{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/noncentralf.jl: (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/noncent
- mean(M::**FactorAnalysis**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/fa.jl:13 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/fa.jl)
- mean(d::**WalleniusNoncentralHypergeometric**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/noncentralhyper (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/noncentrall
- mean(d::**NegativeBinomial{T}**) *where T* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/negativebinomia (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/negativebir
- mean(d::**IntervalSets.AbstractInterval**) in IntervalSets at /Users/uqjnazar/.julia/packages/IntervalSets/VeVgo/src/IntervalSets.jl:100 (file:///Users/uqjnazar/.julia/packages/IntervalSets/VeVgo/src/IntervalSets.jl)
- mean(d::**MvLogNormal**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvlognormal.jl:207 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/mvlognormal.jl)

- mean(d::**Multinomial**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/multinomial.jl:59 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/multinomial.jl)
- mean(d::**Gumbel**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/gumbel.jl:70 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/gumbel.
- mean(d::**BetaPrime{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/betaprime.jl:6 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/betapri
- mean(d::**TriangularDist**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/triangular.jl:65 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/triangul
- mean(d::**Frechet{T}**) *where T* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/frechet.jl:62 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/frechet.
- mean(d::**Logistic**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/logistic.jl:64 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/logistic.
- mean(d::**GeneralizedExtremeValue{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/generalizedex (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/generali
- mean(d::**InverseWishart**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/inversewishart.jl:81 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/inversewishart.jl)
- mean(d::**Dirichlet**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/dirichlet.jl:69 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/multivariate/dirichlet.jl)
- mean(d::**NormalInverseGaussian**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/normalinverse (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/normali
- mean(M::**ICA**) in MultivariateStats at /Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/ica.jl:12 (file:///Users/uqjnazar/.julia/packages/MultivariateStats/HTpHt/src/ica.jl)
- mean(d::**MatrixNormal**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixnormal.jl:88 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/matrix/matrixnormal.jl)

- mean(d::**FDist{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/fdist.jl:57 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/fdist.jl)
- mean(qvec::**AbstractArray{Rotations.UnitQuaternion{T}, 1}**) *where T* in Rotations at /Users/uqjnazar/.julia/packages/Rotations/QaTBi/src/mean.jl:32 (file:///Users/uqjnazar/.julia/packages/Rotations/QaTBi/src/mean.jl)
- mean(qvec::**AbstractArray{Rotations.UnitQuaternion{T}, 1}**, method::**Integer**) *where T* in Rotations at /Users/uqjnazar/.julia/packages/Rotations/QaTBi/src/mean.jl:32 (file:///Users/uqjnazar/.julia/packages/Rotations/QaTBi/src/mean.jl)
- mean(vec::**AbstractVector{R}**) *where R<:Rotations.Rotation* in Rotations at /Users/uqjnazar/.julia/packages/Rotations/QaTBi/src/mean.jl:49 (file:///Users/uqjnazar/.julia/packages/Rotations/QaTBi/src/mean.jl)
- mean(A::**AbstractArray{T, N} where N**, w::**AbstractWeights{W, T, V} where {T<:Real, V<:AbstractVector{T}}**, dims::**Int64**) *where {T<:Number, W<:Real}* in StatsBase at deprecated.jl:70 (file:///Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/base/deprecated.jl)
- mean(A::**AbstractArray**; *dims*) in Statistics at /Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/stdlib/v1.6/Statistics/src/Statistics.jl:164 (file:///Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/v1.6/Statisti
- mean(A::**AbstractArray**, w::**UnitWeights**; *dims*) in StatsBase at /Users/uqjnazar/.julia/packages/StatsBase/DU1bT/src/weights.jl:625 (file:///Users/uqjnazar/.julia/packages/StatsBase/DU1bT/src/weights.jl)
- mean(A::**AbstractArray**, w::**AbstractWeights**; *dims*) in StatsBase at /Users/uqjnazar/.julia/packages/StatsBase/DU1bT/src/weights.jl:618 (file:///Users/uqjnazar/.julia/packages/StatsBase/DU1bT/src/weights.jl)
- mean(d::**DiscreteNonParametric**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/discretenonpara (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/discretenor
- mean(d::**EdgeworthMean**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/edgeworth.jl:103 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/edgeworth.jl)
- mean(d::**BetaBinomial**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/betabinomial.jl:5 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/betabinom
- mean(d::**Pareto{T}**) *where T<:Real* in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/pareto.jl:57 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/pareto.j

- mean(d::**Arcsine**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/arcsine.jl:64 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/arcsine.
- mean(d::**Weibull**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/weibull.jl:63 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/weibull.
- mean(d::**Chernoff**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/chernoff.jl:22 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/chernof
- mean(d::**Hypergeometric**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/hypergeometric. (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/hypergeom
- mean(d::**Kolmogorov**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/kolmogorov.jl (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/continuous/kolmog
- mean(Ω::**Soliton**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/soliton.jl:108 (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/soliton.jl)
- mean(d::**DiscreteUniform**) in Distributions at /Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/discreteuniform. (file:///Users/uqjnazar/.julia/packages/Distributions/fXTVC/src/univariate/discrete/discreteuni
- mean(itr) in Statistics at /Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/stdlib/v1.6/Statistics/src/Statistics.jl:44 (file:///Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/v1.6/Statisti
- mean(f, A::**AbstractArray**; *dims*) in Statistics at /Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/stdlib/v1.6/Statistics/src/Statistics.jl:104 (file:///Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/v1.6/Statisti
- mean(f, itr) in Statistics at /Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/stdlib/v1.6/Statistics/src/Statistics.jl:61 (file:///Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/v1.6/Statisti

In [25]:
```
@which mean(data)
```

Out[25]: mean(A::**AbstractArray**; *dims*) in Statistics at /Applications/Julia-1.6.app/Contents/Resources/julia/share/julia/stdlib/v1.6/Statistics/src/Statistics.jl:164 (file:///Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/v1.6/Statistics/s

In [26]:
```
methods(my_mean)
```

Out[26]: # 1 method for generic function **my_mean**:

- my_mean(data) in Main at In[20]:1

In [27]:
```
function my_mean(data::Vector{Float64})::Float64
    mn = 0.0
    for (i,d) in enumerate(data)
        mn = (1/i)*d + (i-1)/i*mn
    end
    return mn
end
```

Out[27]: `my_mean (generic function with 2 methods)`

In [28]:
```
methods(my_mean)
```

Out[28]: # 2 methods for generic function **my_mean**:

- my_mean(data::**Vector{Float64}**) in Main at In[27]:1
- my_mean(data) in Main at In[20]:1

In [29]:
```
@which my_mean(data)
```

Out[29]: my_mean(data) in Main at In[20]:1

In [30]:
```
@which my_mean([1,2])
```

Out[30]: my_mean(data) in Main at In[20]:1

In [31]:
```
@which my_mean([1.0,2.0])
```

Out[31]: my_mean(data::**Vector{Float64}**) in Main at In[27]:1

In [32]:
```
@which my_mean([1.0,2])
```

Out[32]: my_mean(data::**Vector{Float64}**) in Main at In[27]:1

In [33]:
```
function my_mean(data::Vector{T})::T where T
    mn = zero(data[begin])
    for (i,d) in enumerate(data)
        mn = (1/i)*d + (i-1)/i*mn
    end
    return mn
end
```

Out[33]:
```
my_mean (generic function with 3 methods)
```

In [34]:
```
methods(my_mean)
```

Out[34]:
3 methods for generic function **my_mean**:

- my_mean(data::**Vector{Float64}**) in Main at In[27]:1
- my_mean(data::**Vector{T}**) *where T* in Main at In[33]:1
- my_mean(data) in Main at In[20]:1

The mean of other types

In [35]:
```
fix_seed!()
data = [rand(Normal(),2,2) for _ in 1:5]
```

Out[35]:
```
5-element Vector{Matrix{Float64}}:
 [0.6791074260357777 -0.3530074003005963; 0.8284134829000359 -0.1348538
 [0.5866170746331097 0.06494754854834232; 0.29733585084941616 -0.109017
 [-0.514210390833322 -0.6889071278256981; 1.5743302021369892 -0.7628038
 [0.39748240921816347 -0.34635460427879816; 0.8116296225068749 -0.18757
 [-1.6072563241277753 2.2762328327845243; -2.48079273065994 0.219693467
```

In [36]:
```
mean(data)
```

Out[36]:
```
2×2 Matrix{Float64}:
 -0.091652   0.190582
  0.206183  -0.194911
```

In [37]:
```
my_mean(data)
```

Out[37]:
```
2×2 Matrix{Float64}:
 -0.091652   0.190582
  0.206183  -0.194911
```

In [38]:
```
@which my_mean(data)
```

Out[38]: my_mean(data::**Vector{T}**) *where T* in Main at In[33]:1

A glimpse under the hood

In [39]:
```
@code_lowered my_mean(data)
```

Out[39]:
```
CodeInfo(
1 ─ %1  = $(Expr(:static_parameter, 1))
│   %2  = Base.firstindex(data)
│   %3  = Base.getindex(data, %2)
│         mn = Main.zero(%3)
│   %5  = Main.enumerate(data)
│         @_3 = Base.iterate(%5)
│   %7  = @_3 === nothing
│   %8  = Base.not_int(%7)
└──       goto #4 if not %8
2 ┄ %10 = @_3
│   %11 = Core.getfield(%10, 1)
│   %12 = Base.indexed_iterate(%11, 1)
│         i = Core.getfield(%12, 1)
│         @_5 = Core.getfield(%12, 2)
│   %15 = Base.indexed_iterate(%11, 2, @_5)
│         d = Core.getfield(%15, 1)
│   %17 = Core.getfield(%10, 2)
│   %18 = 1 / i
│   %19 = %18 * d
│   %20 = i - 1
│   %21 = %20 / i
│   %22 = %21 * mn
│         mn = %19 + %22
│         @_3 = Base.iterate(%5, %17)
│   %25 = @_3 === nothing
│   %26 = Base.not_int(%25)
└──       goto #4 if not %26
3 ─       goto #2
4 ┄ %29 = Base.convert(%1, mn)
│   %30 = Core.typeassert(%29, %1)
└──       return %30
)
```

In [40]:
```
@code_llvm my_mean(data)
```

```
;  @ In[33]:1 within `my_mean'
define nonnull {}* @japi1_my_mean_4985({}* %0, {}** %1, i32 %2) #0 {
top:
  %3 = alloca [2 x {}*], align 8
  %gcframe70 = alloca [4 x {}*], align 16
  %gcframe70.sub = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcfram
  %.sub = getelementptr inbounds [2 x {}*], [2 x {}*]* %3, i64 0, i64 0
  %4 = bitcast [4 x {}*]* %gcframe70 to i8*
  call void @llvm.memset.p0i8.i32(i8* nonnull align 16 dereferenceable(
  %5 = alloca {}**, align 8
  store volatile {}** %1, {}*** %5, align 8
  %6 = call {}*** inttoptr (i64 4480861040 to {}*** ()*)() #5
  %7 = bitcast [4 x {}*]* %gcframe70 to i64*
  store i64 8, i64* %7, align 16
  %8 = bitcast {}*** %6 to i64*
  %9 = load i64, i64* %8, align 8
  %10 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe70, i64 0,
  %11 = bitcast {}** %10 to i64*
  store i64 %9, i64* %11, align 8
  %12 = bitcast {}*** %6 to {}***
  store {}** %gcframe70.sub, {}*** %12, align 8
  %13 = load {}*, {}** %1, align 8
;  @ In[33]:2 within `my_mean'
; ┌ @ array.jl:801 within `getindex'
   %14 = bitcast {}* %13 to { i8*, i64, i16, i16, i32 }*
   %15 = getelementptr inbounds { i8*, i64, i16, i16, i32 }, { i8*, i64
   %16 = load i64, i64* %15, align 8
   %.not = icmp eq i64 %16, 0
   br i1 %.not, label %oob, label %idxend

L18:                                              ; preds = %scalar.ph,
   %value_phi3 = phi i64 [ %19, %L18 ], [ %bc.resume.val, %scalar.ph ]
; └
; ┌ @ abstractarray.jl:1085 within `zero'
; │┌ @ array.jl:335 within `fill!'
; ││┌ @ array.jl:839 within `setindex!'
     %17 = add nsw i64 %value_phi3, -1
     %18 = getelementptr inbounds double, double* %58, i64 %17
     store double 0.000000e+00, double* %18, align 8
; ││└
; ││┌ @ range.jl:674 within `iterate'
```

```
; │││┌ @ promotion.jl:410 within `=='
      %.not43.not = icmp eq i64 %value_phi3, %56
; │││└
     %19 = add nuw nsw i64 %value_phi3, 1
; ││└
    br i1 %.not43.not, label %L34, label %L18

L34:                                              ; preds = %middle.blo
; └└
;  @ In[33]:3 within `my_mean'
; ┌ @ iterators.jl:158 within `iterate' @ iterators.jl:159 @ array.jl:7
; │┌ @ array.jl:197 within `length'
    %20 = load i64, i64* %15, align 8
; │└
; │┌ @ int.jl:448 within `<' @ int.jl:441
    %.not44 = icmp eq i64 %20, 0
; │└
   br i1 %.not44, label %L106, label %L43

L43:                                              ; preds = %L34
; │┌ @ array.jl:801 within `getindex'
    %21 = load {}**, {}*** %43, align 8
    %22 = load atomic {}*, {}** %21 unordered, align 8
    %.not45 = icmp eq {}* %22, null
    br i1 %.not45, label %fail8, label %L63

L63:                                              ; preds = %39, %L43
    %value_phi19 = phi i64 [ %value_phi21, %39 ], [ 1, %L43 ]
    %value_phi20 = phi {}* [ %38, %39 ], [ %22, %L43 ]
    %value_phi21 = phi i64 [ %40, %39 ], [ 2, %L43 ]
    %value_phi23 = phi {}* [ %33, %39 ], [ %53, %L43 ]
; └└
;  @ In[33]:4 within `my_mean'
; ┌ @ int.jl:93 within `/'
; │┌ @ float.jl:206 within `float'
; ││┌ @ float.jl:191 within `AbstractFloat'
; │││┌ @ float.jl:94 within `Float64'
      %23 = sitofp i64 %value_phi19 to double
; │└└└
; │ @ int.jl:93 within `/' @ float.jl:335
   %24 = fdiv double 1.000000e+00, %23
   %25 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe70, i64 0
   store {}* %value_phi23, {}** %25, align 8
   %26 = getelementptr inbounds [4 x {}*], [4 x {}*]* %gcframe70, i64 0
```

```
   store {}* %value_phi20, {}** %26, align 16
; └
  %27 = call nonnull {}* @"j_*_4986"(double %24, {}* %value_phi20)
; ┌ @ int.jl:86 within `-'
   %28 = add nsw i64 %value_phi19, -1
; └
; ┌ @ int.jl:93 within `/'
; │┌ @ float.jl:206 within `float'
; ││┌ @ float.jl:191 within `AbstractFloat'
; │││┌ @ float.jl:94 within `Float64'
      %29 = sitofp i64 %28 to double
; │└└└
; │ @ int.jl:93 within `/' @ float.jl:335
   %30 = fdiv double %29, %23
   store {}* %27, {}** %26, align 16
; └
  %31 = call nonnull {}* @"j_*_4987"(double %30, {}* %value_phi23)
  store {}* %31, {}** %25, align 8
  store {}* %27, {}** %.sub, align 8
  %32 = getelementptr inbounds [2 x {}*], [2 x {}*]* %3, i64 0, i64 1
  store {}* %31, {}** %32, align 8
  %33 = call nonnull {}* @"j1_+_4988"({}* inttoptr (i64 4598481024 to {
; ┌ @ iterators.jl:159 within `iterate' @ array.jl:777
; │┌ @ int.jl:923 within `-' @ int.jl:86
    %34 = add nsw i64 %value_phi21, -1
; │└
; │┌ @ array.jl:197 within `length'
    %35 = load i64, i64* %15, align 8
; │└
; │┌ @ int.jl:448 within `<' @ int.jl:441
    %.not46 = icmp ult i64 %34, %35
; │└
   br i1 %.not46, label %L86, label %L106

L86:                                              ; preds = %L63
; │┌ @ array.jl:801 within `getindex'
    %36 = load {}**, {}*** %43, align 8
    %37 = getelementptr inbounds {}*, {}** %36, i64 %34
    %38 = load atomic {}*, {}** %37 unordered, align 8
    %.not47 = icmp eq {}* %38, null
    br i1 %.not47, label %fail24, label %39

39:                                               ; preds = %L86
    %40 = add nuw i64 %value_phi21, 1
```

```
; └┘
  br label %L63

L106:                                             ; preds = %L63, %L34
  %value_phi35 = phi {}* [ %53, %L34 ], [ %33, %L63 ]
  %41 = load i64, i64* %11, align 8
  store i64 %41, i64* %8, align 8
;  @ In[33]:6 within `my_mean'
  ret {}* %value_phi35

oob:                                              ; preds = %top
;  @ In[33]:2 within `my_mean'
; ┌ @ array.jl:801 within `getindex'
   %42 = alloca i64, align 8
   store i64 1, i64* %42, align 8
   call void @jl_bounds_error_ints({}* %13, i64* nonnull %42, i64 1)
   unreachable

idxend:                                           ; preds = %top
   %43 = bitcast {}* %13 to {}***
   %44 = load {}**, {}*** %43, align 8
   %45 = load atomic {}*, {}** %44 unordered, align 8
   %.not41 = icmp eq {}* %45, null
   br i1 %.not41, label %fail, label %pass

fail:                                             ; preds = %idxend
   call void @jl_throw({}* inttoptr (i64 4636188672 to {}*))
   unreachable

pass:                                             ; preds = %idxend
; └
; ┌ @ abstractarray.jl:1085 within `zero'
; │┌ @ array.jl:354 within `similar'
; ││┌ @ array.jl:132 within `size'
     %46 = bitcast {}* %45 to {}**
     %47 = getelementptr inbounds {}*, {}** %46, i64 3
     %48 = bitcast {}** %47 to i64*
     %49 = load i64, i64* %48, align 8
     %50 = getelementptr inbounds {}*, {}** %46, i64 4
     %51 = bitcast {}** %50 to i64*
     %52 = load i64, i64* %51, align 8
; ││└
; ││┌ @ boot.jl:450 within `Array'
     %53 = call nonnull {}* inttoptr (i64 4479306021 to {}* ({}*, i64, 
```

```
; │└└
; │ ┌ @ array.jl:334 within `fill!'
; │ │┌ @ abstractarray.jl:301 within `eachindex' @ abstractarray.jl:311
; │ ││┌ @ array.jl:197 within `length'
      %54 = bitcast {}* %53 to { i8*, i64, i16, i16, i32 }*
      %55 = getelementptr inbounds { i8*, i64, i16, i16, i32 }, { i8*, i
      %56 = load i64, i64* %55, align 8
; │ ││└└
; │ ││ ┌ @ range.jl:670 within `iterate'
; │ ││ │┌ @ range.jl:519 within `isempty'
; │ ││ ││┌ @ operators.jl:305 within `>'
; │ ││ │││┌ @ int.jl:83 within `<'
        %.not42.not = icmp eq i64 %56, 0
; │ ││ ││└└└└
    br i1 %.not42.not, label %L34, label %L18.preheader

L18.preheader:                                    ; preds = %pass
; │ ┌ @ array.jl:335 within `fill!'
; │ │┌ @ array.jl within `setindex!'
     %57 = bitcast {}* %53 to double**
     %58 = load double*, double** %57, align 8
; │ │└
    %min.iters.check = icmp ult i64 %56, 16
    br i1 %min.iters.check, label %scalar.ph, label %vector.ph

vector.ph:                                        ; preds = %L18.prehea
    %n.vec = and i64 %56, 9223372036854775792
    %ind.end = or i64 %n.vec, 1
    br label %vector.body

vector.body:                                      ; preds = %vector.bod
    %index = phi i64 [ 0, %vector.ph ], [ %index.next, %vector.body ]
; │ ┌ @ array.jl:839 within `setindex!'
     %59 = getelementptr inbounds double, double* %58, i64 %index
     %60 = bitcast double* %59 to <4 x double>*
     store <4 x double> zeroinitializer, <4 x double>* %60, align 8
     %61 = getelementptr inbounds double, double* %59, i64 4
     %62 = bitcast double* %61 to <4 x double>*
     store <4 x double> zeroinitializer, <4 x double>* %62, align 8
     %63 = getelementptr inbounds double, double* %59, i64 8
     %64 = bitcast double* %63 to <4 x double>*
     store <4 x double> zeroinitializer, <4 x double>* %64, align 8
     %65 = getelementptr inbounds double, double* %59, i64 12
```

```
     %66 = bitcast double* %65 to <4 x double>*
     store <4 x double> zeroinitializer, <4 x double>* %66, align 8
     %index.next = add i64 %index, 16
     %67 = icmp eq i64 %index.next, %n.vec
     br i1 %67, label %middle.block, label %vector.body

middle.block:                                     ; preds = %vector.body
; ||└
    %cmp.n = icmp eq i64 %56, %n.vec
    br i1 %cmp.n, label %L34, label %scalar.ph

scalar.ph:                                        ; preds = %middle.block
    %bc.resume.val = phi i64 [ %ind.end, %middle.block ], [ 1, %L18.preh
    br label %L18

fail8:                                            ; preds = %L43
; └└
;  @ In[33]:3 within `my_mean'
; ┌ @ iterators.jl:158 within `iterate' @ iterators.jl:159 @ array.jl:7
; │┌ @ array.jl:801 within `getindex'
    call void @jl_throw({}* inttoptr (i64 4636188672 to {}*))
    unreachable

fail24:                                           ; preds = %L86
; └└
;  @ In[33]:4 within `my_mean'
; ┌ @ iterators.jl:159 within `iterate' @ array.jl:777
; │┌ @ array.jl:801 within `getindex'
    call void @jl_throw({}* inttoptr (i64 4636188672 to {}*))
    unreachable
; └└
}
```

In [41]:
```
@code_native my_mean(data)
```

```
        .section        __TEXT,__text,regular,pure_instructions
; ┌ @ In[33]:1 within `my_mean'
        pushq   %rbp
        movq    %rsp, %rbp
        pushq   %r15
        pushq   %r14
        pushq   %r13
        pushq   %r12
        pushq   %rbx
        andq    $-32, %rsp
        subq    $128, %rsp
        movq    %rsp, %rbx
        movq    %rsi, %r14
        movabsq $jl_alloc_array_2d, %r12
        vxorpd  %xmm0, %xmm0, %xmm0
        vmovapd %ymm0, (%rbx)
        movq    %rsi, 96(%rbx)
        leaq    1555019(%r12), %rax
        vzeroupper
        callq   *%rax
        movq    %rax, %r13
        movq    $8, (%rbx)
        movq    (%rax), %rax
        movq    %rax, 8(%rbx)
        movq    %rbx, %rax
        movq    %rax, (%r13)
        movq    (%r14), %r14
; │ @ In[33]:2 within `my_mean'
; │┌ @ array.jl:801 within `getindex'
        cmpq    $0, 8(%r14)
        je      L527
        movq    (%r14), %rax
        movq    (%rax), %rax
        testq   %rax, %rax
        je      L565
; │└
; │┌ @ abstractarray.jl:1085 within `zero'
; ││┌ @ array.jl:354 within `similar'
; │││┌ @ array.jl:132 within `size'
        movq    24(%rax), %rsi
        movq    32(%rax), %rdx
```

```
        movabsq $jl_system_image_data, %rdi
; ┌┌┌└
; │└└└ @ boot.jl:450 within `Array'
        callq   *%r12
        movq    %rax, %r12
; ┌└└└
; │┌┌ @ array.jl:334 within `fill!'
; ┌└└└ @ abstractarray.jl:301 within `eachindex' @ abstractarray.jl:311
; ┌└└└ @ array.jl:197 within `length'
        movq    8(%rax), %rax
; ┌└└└
; │┌┌ @ range.jl:670 within `iterate'
; ┌└└└ @ range.jl:519 within `isempty'
; ┌└└└ @ operators.jl:305 within `>'
; ┌└└└ @ int.jl:83 within `<'
        testq   %rax, %rax
; ┌└└└
        je      L256
; ┌└└
; │┌┌ @ array.jl:335 within `fill!'
; │└└ @ array.jl within `setindex!'
        movq    (%r12), %rcx
        movl    $1, %edx
; ┌└└
        cmpq    $16, %rax
        jb      L229
        movq    %rax, %rsi
        andq    $-16, %rsi
        leaq    1(%rsi), %rdx
        xorl    %edi, %edi
        vxorpd  %xmm0, %xmm0, %xmm0
        nopl    (%rax,%rax)
; ┌└└ @ array.jl:839 within `setindex!'
L192:
        vmovupd %ymm0, (%rcx,%rdi,8)
        vmovupd %ymm0, 32(%rcx,%rdi,8)
        vmovupd %ymm0, 64(%rcx,%rdi,8)
        vmovupd %ymm0, 96(%rcx,%rdi,8)
        addq    $16, %rdi
        cmpq    %rdi, %rsi
        jne     L192
; ┌└└
        cmpq    %rsi, %rax
        je      L256
```

```
L229:
        decq    %rdx
        nopl    (%rax,%rax)
; ┌┌┌ @ array.jl:839 within `setindex!'
L240:
        movq    $0, (%rcx,%rdx,8)
; ┌┌└
; ┌┌┌ @ range.jl:674 within `iterate'
; ┌┌┌┌ @ promotion.jl:410 within `=='
        incq    %rdx
        cmpq    %rdx, %rax
; └└└└
        jne     L240
; ┌└└
;  @ In[33]:3 within `my_mean'
; ┌┌ @ iterators.jl:158 within `iterate' @ iterators.jl:159 @ array.jl:
; ┌┌┌ @ int.jl:448 within `<' @ int.jl:441
L256:
        cmpq    $0, 8(%r14)
; ┌└
        je      L476
; ┌┌ @ iterators.jl:158 within `iterate' @ iterators.jl:159 @ array.jl:
        movq    %r13, 48(%rbx)
        movq    %r14, 56(%rbx)
; ┌┌ @ iterators.jl:158 within `iterate' @ iterators.jl:159 @ array.jl:
; ┌┌┌ @ array.jl:801 within `getindex'
        movq    (%r14), %rax
        movq    (%rax), %rdi
        testq   %rdi, %rdi
        je      L587
; ┌└└
;  @ array.jl within `my_mean'
        movl    $1, %r14d
        movabsq $5909782808, %rax               ## imm = 0x160402118
        vmovsd  (%rax), %xmm0                   ## xmm0 = mem[0],zero
        vmovsd  %xmm0, 64(%rbx)
        movabsq $"*", %r15
        nopw    %cs:(%rax,%rax)
;  @ In[33]:4 within `my_mean'
; ┌ @ int.jl:93 within `/'
; ┌┌ @ float.jl:206 within `float'
; ┌┌┌ @ float.jl:191 within `AbstractFloat'
; ┌┌┌┌ @ float.jl:94 within `Float64'
L336:
```

```
        vcvtsi2sd       %r14, %xmm2, %xmm1
        vmovsd  %xmm1, 72(%rbx)
        vmovsd  64(%rbx), %xmm0                 ## xmm0 = mem[0],zero
; ||└└└
; ||  @ int.jl:93 within `/' @ float.jl:335
        vdivsd  %xmm1, %xmm0, %xmm0
        movq    %r12, 24(%rbx)
        movq    %rdi, 16(%rbx)
; |└
        vzeroupper
        callq   *%r15
        movq    %rax, %r13
; |┌ @ int.jl:93 within `/'
; ||┌ @ float.jl:206 within `float'
; |||┌ @ float.jl:191 within `AbstractFloat'
; ||||┌ @ float.jl:94 within `Float64'
        leaq    -1(%r14), %rax
        vcvtsi2sd       %rax, %xmm2, %xmm0
; ||└└└
; ||  @ int.jl:93 within `/' @ float.jl:335
        vdivsd  72(%rbx), %xmm0, %xmm0
        movq    %r13, 16(%rbx)
; |└
        movq    %r12, %rdi
        callq   *%r15
        movq    %rax, 24(%rbx)
        movq    %r13, 80(%rbx)
        movq    %rax, 88(%rbx)
        movabsq $jl_system_image_data, %rdi
        leaq    80(%rbx), %rsi
        movl    $2, %edx
        movabsq $"+", %rax
        callq   *%rax
        movq    %rax, %r12
        movq    56(%rbx), %rax
; |┌ @ iterators.jl:159 within `iterate' @ array.jl:777
; ||┌ @ int.jl:448 within `<' @ int.jl:441
        cmpq    8(%rax), %r14
; ||└
        jae     L472
; ||┌ @ array.jl:801 within `getindex'
        movq    (%rax), %rax
        movq    (%rax,%r14,8), %rdi
        testq   %rdi, %rdi
```

```
        je      L505
; ┌└
        incq    %r14
        jmp     L336
; │ @ In[33] within `my_mean'
L472:
        movq    48(%rbx), %r13
L476:
        movq    8(%rbx), %rax
        movq    %rax, (%r13)
; │ @ In[33]:6 within `my_mean'
        movq    %r12, %rax
        leaq    -40(%rbp), %rsp
        popq    %rbx
        popq    %r12
        popq    %r13
        popq    %r14
        popq    %r15
        popq    %rbp
        vzeroupper
        retq
; │ @ In[33]:4 within `my_mean'
; │┌ @ iterators.jl:159 within `iterate' @ array.jl:777
; ││┌ @ array.jl:801 within `getindex'
L505:
        movabsq $jl_throw, %rax
        movabsq $jl_system_image_data, %rdi
        callq   *%rax
; │└└
; │ @ In[33]:2 within `my_mean'
; │┌ @ array.jl:801 within `getindex'
L527:
        movq    %rsp, %rax
        leaq    -16(%rax), %rsi
        movq    %rsi, %rsp
        movq    $1, -16(%rax)
        movabsq $jl_bounds_error_ints, %rax
        movl    $1, %edx
        movq    %r14, %rdi
        callq   *%rax
L565:
        movabsq $jl_throw, %rax
        movabsq $jl_system_image_data, %rdi
        callq   *%rax
```

```
; │└
; │ @ In[33]:3 within `my_mean'
; │┌ @ iterators.jl:158 within `iterate' @ iterators.jl:159 @ array.jl:
; ││┌ @ array.jl:801 within `getindex'
L587:
        movabsq $jl_throw, %rax
        movabsq $jl_system_image_data, %rdi
        vzeroupper
        callq   *%rax
; └└└
```

An array of arrays

In [42]:
```
fix_seed!()
data = [rand(Normal(μ,1),5) for μ in 1:5] #\mu + [TAB]
```

Out[42]:
```
5-element Vector{Vector{Float64}}:
 [1.6791074260357777, 1.8284134829000358, 0.6469925996994037, 0.865146
 [2.297335850849416, 2.0649475485483424, 1.8909826149182825, 1.485789
 [2.311092872174302, 2.2371961835895418, 3.3974824092181635, 3.811629
 [3.81242731805484, 2.3927436758722247, 1.5192072693400598, 6.276232
 [4.882862469267065, 4.398746405814617, 6.14227643894132, 4.911383681
```

In [43]:
```
mean.(data)
```

Out[43]:
```
5-element Vector{Float64}:
 1.321255342267561
 2.262677165123942
 2.8822092966420163
 3.644060912718838
 5.122947049582486
```

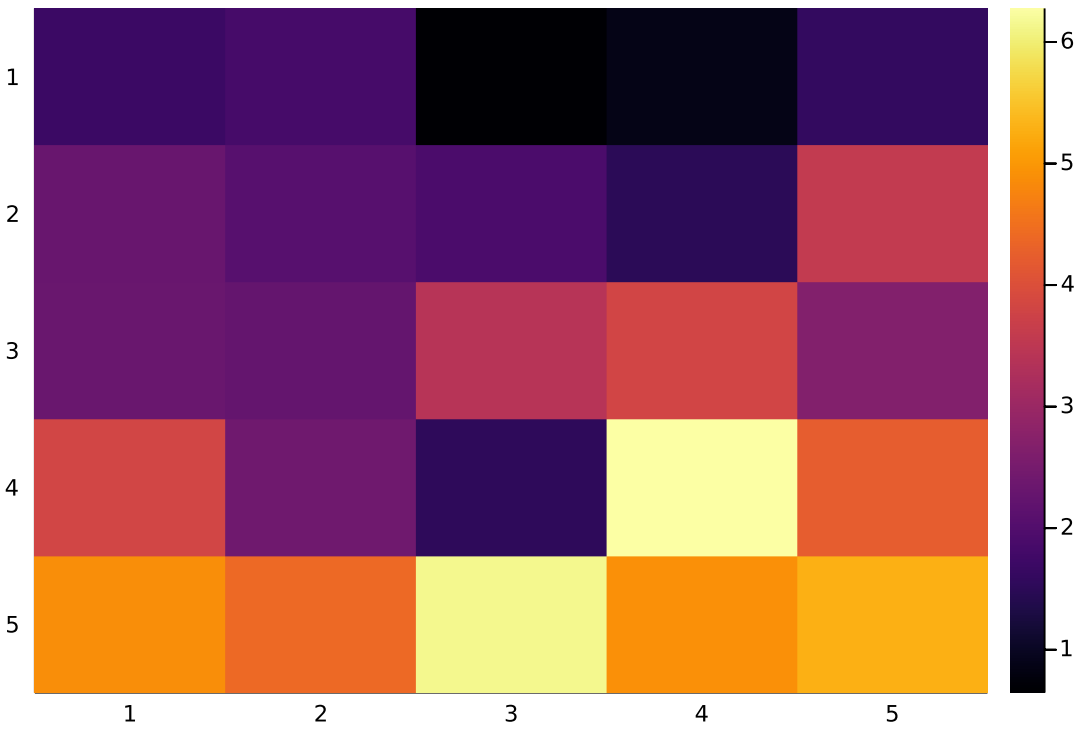
A matrix of data

In [44]:
```
data = hcat(data...)
```

Out[44]:
```
5×5 Matrix{Float64}:
 1.67911   2.29734  2.31109  3.81243  4.88286
 1.82841   2.06495  2.2372   2.39274  4.39875
 0.646993  1.89098  3.39748  1.51921  6.14228
 0.865146  1.48579  3.81163  6.27623  4.91138
 1.58662   3.57433  2.65365  4.21969  5.27947
```

In [45]:
```
heatmap(data',yflip=true)
```

Out[45]:



In [46]:
```
mean.(data) #does nothing useful
```

Out[46]:
```
5×5 Matrix{Float64}:
 1.67911   2.29734  2.31109  3.81243  4.88286
 1.82841   2.06495  2.2372   2.39274  4.39875
 0.646993  1.89098  3.39748  1.51921  6.14228
 0.865146  1.48579  3.81163  6.27623  4.91138
 1.58662   3.57433  2.65365  4.21969  5.27947
```

In [47]:
```
mean(data,dims=1) #better!
```

Out[47]:
```
1×5 Matrix{Float64}:
 1.32126  2.26268  2.88221  3.64406  5.12295
```

Dictionaries

In [48]:
```
data = Dict()
data[:cats] = rand(Normal(2.3,1),5)
data["dogs"] = rand(Normal(5.2,1),5)
data[25] = rand(Normal(7.8,1),5)
data
```

Out[48]:
```
Dict{Any, Any} with 3 entries:
  "dogs" => [6.62305, 5.60839, 5.78862, 4.90372, 5.89111]
  25     => [8.30687, 7.74307, 6.02898, 9.39062, 9.19706]
  :cats  => [2.41142, 1.94212, 2.77371, 2.60023, 1.53732]
```

In [49]:
```
keys(data)
```

Out[49]:
```
KeySet for a Dict{Any, Any} with 3 entries. Keys:
  "dogs"
  25
  :cats
```

In [50]:
```
mean(data[:cats])
```

Out[50]:
```
2.2529618709854136
```

In [51]:
```
mean(data["dogs"])
```

Out[51]:
```
5.762977963045932
```

In [52]:
```
mean(data["cats"])
```

```
KeyError: key "cats" not found

Stacktrace:
 [1] getindex(h::Dict{Any, Any}, key::String)
   @ Base ./dict.jl:482
 [2] top-level scope
   @ In[52]:1
 [3] eval
   @ ./boot.jl:360 [inlined]
 [4] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code:
   @ Base ./loading.jl:1094
```

In [53]:
```
mean(data[Symbol("cats")])
```

Out[53]:
```
2.2529618709854136
```

In [54]:
```
data[Symbol("cats")] |> mean
```

Out[54]:
```
2.2529618709854136
```

Tuples

In [55]:
```
fix_seed!()
rn() = rand(Normal())
data = (rn(), rn(),rn())
```

Out[55]:
```
(0.6791074260357777, 0.8284134829000359, -0.3530074003005963)
```

In [56]:
```
typeof(data)
```

Out[56]:
```
Tuple{Float64, Float64, Float64}
```

In [57]:
```
mean(data)
```

Out[57]:
```
0.3848378362117391
```

In [58]:
```
data[2]
```

Out[58]:
```
0.8284134829000359
```

In [59]:
```
data[2] = 7.8 #Tuples are immutable!
```

```
MethodError: no method matching setindex!(::Tuple{Float64, Float64, Flo

Stacktrace:
 [1] top-level scope
   @ In[59]:1
 [2] eval
   @ ./boot.jl:360 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code:
   @ Base ./loading.jl:1094
```

Structs

In [60]:
```
struct Person
    weight::Float64
    height::Float64
end
```

In [61]:
```
someone = Person(102, 180)
```

Out[61]:
```
Person(102.0, 180.0)
```

In [62]:
```
someone.weight = 103 #Structs are immutable unless you state immutable
```

```
setfield! immutable struct of type Person cannot be changed

Stacktrace:
 [1] setproperty!(x::Person, f::Symbol, v::Int64)
   @ Base ./Base.jl:34
 [2] top-level scope
   @ In[62]:1
 [3] eval
   @ ./boot.jl:360 [inlined]
 [4] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code:
   @ Base ./loading.jl:1094
```

In [63]:
```
me = Person(102, 180)
you = Person(83, 170)
me + you #can't add persons just like that
```

```
MethodError: no method matching +(::Person, ::Person)
Closest candidates are:
  +(::Any, ::Any, ::Any, ::Any...) at operators.jl:560
  +(::T, ::Any) where T<:Intervals.Interval at /Users/uqjnazar/.julia/p
  +(::ChainRulesCore.Tangent{P, T} where T, ::P) where P at /Users/uqjn
  ...

Stacktrace:
 [1] top-level scope
   @ In[63]:3
 [2] eval
   @ ./boot.jl:360 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code:
   @ Base ./loading.jl:1094
```

In [64]:
```
import Base: +

function +(x::Person, y::Person)
    Person(x.weight + y.weight, x.height + y.height)
end
```

Out[64]:
```
+ (generic function with 472 methods)
```

In [65]:
```
me + you #Now you can
```

Out[65]:
```
Person(185.0, 350.0)
```

In [66]:
```
import Base: /
function /(x::Person, n::Real)
    Person(x.weight/n, x.height/n)
end
```

Out[66]:
```
/ (generic function with 209 methods)
```

In [67]:
```
mean([me,you]) #With + and / (by number) we can compute the mean of pers
```

Out[67]:
```
Person(92.5, 175.0)
```

Dataframes

In [68]:
```
df = CSV.File("./data/temperatures.csv") |> DataFrame
```

Out[68]: 777 rows × 5 columns

	Year	Month	Day	Brisbane	GoldCoast
	Int64	Int64	Int64	Float64	Float64
1	2015	1	1	31.3	30.9
2	2015	1	2	30.5	30.1
3	2015	1	3	28.9	30.1
4	2015	1	4	30.2	30.1
5	2015	1	5	28.1	28.0
6	2015	1	6	29.5	29.3
7	2015	1	7	27.1	26.4
8	2015	1	8	28.4	27.7
9	2015	1	9	29.5	29.6
10	2015	1	10	30.2	29.5
11	2015	1	11	28.2	28.0
12	2015	1	12	30.4	29.1
13	2015	1	13	28.4	27.2
14	2015	1	14	30.3	29.4
15	2015	1	15	36.0	36.5
16	2015	1	16	32.5	30.9
17	2015	1	17	35.3	31.6
18	2015	1	18	36.7	32.2
19	2015	1	19	31.8	30.8
20	2015	1	20	29.2	27.9
21	2015	1	21	30.0	30.2
22	2015	1	22	28.6	28.9
23	2015	1	23	24.8	25.7

	Year	Month	Day	Brisbane	GoldCoast
	Int64	Int64	Int64	Float64	Float64
24	2015	1	24	30.8	29.4
25	2015	1	25	36.7	33.4
26	2015	1	26	33.1	30.8
27	2015	1	27	30.2	30.3
28	2015	1	28	28.1	27.9
29	2015	1	29	28.9	29.4
30	2015	1	30	29.0	28.8
⋮	⋮	⋮	⋮	⋮	⋮

In [69]:
```
data = df.GoldCoast
mean(data)
```

Out[69]:
```
26.163835263835264
```

All kind of other descriptive statistics

In [70]:
```
println("Sample Mean: ", mean(data))
println("Harmonic ≤ Geometric ≤ Arithmetic ",      #\le + [TAB]
            (harmmean(data), geomean(data), mean(data)))

println("Sample Variance: ",var(data))
println("Sample Standard Deviation: ",std(data))
println("Minimum: ", minimum(data))
println("Maximum: ", maximum(data))
println("Median: ", median(data))
println("95th percentile: ", percentile(data, 95))
println("0.95 quantile: ", quantile(data, 0.95))
println("Interquartile range: ", iqr(data),"\n")

summarystats(data)
```

```
Sample Mean: 26.163835263835264
Harmonic ≤ Geometric ≤ Arithmetic (25.65423319190783, 25.91556340978906
Sample Variance: 12.367253570765161
Sample Standard Deviation: 3.516710618001595
Minimum: 15.4
Maximum: 36.5
Median: 26.8
95th percentile: 30.9
0.95 quantile: 30.9
Interquartile range: 5.5
```

Out[70]:
```
Summary Stats:
Length:         777
Missing Count:  0
Mean:           26.163835
Minimum:        15.400000
1st Quartile:   23.400000
Median:         26.800000
3rd Quartile:   28.900000
Maximum:        36.500000
```

In [71]:
```
min(5,2,-3) #Minimum of individual function arguments
```

Out[71]:
```
-3
```

In [72]:
```
minimum([5,2,-3]) #Minimum of an array (AbstractArray)
```

Out[72]:
```
-3
```

In [73]:
```
min([5,2,-3]) #doesn't work
```

```
MethodError: no method matching min(::Vector{Int64})
Closest candidates are:
  min(::Any, ::Missing) at missing.jl:127
  min(::T1, ::DataValues.DataValue{T2}) where {T1, T2} at /Users/uqjnaza
  min(::Any, ::Any) at operators.jl:433
  ...

Stacktrace:
 [1] top-level scope
   @ In[73]:1
 [2] eval
   @ ./boot.jl:360 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code:
   @ Base ./loading.jl:1094
```

In [74]:
```
min([5,2,-3]...) #The splat operator passes the arguments
```

Out[74]:
```
-3
```

In [75]:
```
fix_seed!()
data = rand(Normal(),10^6);
```

In [76]:
```
@time begin
    minimum(data)
end
```

```
  0.000859 seconds (1 allocation: 16 bytes)
```

Out[76]:
```
-4.846415885152558
```

In [77]:
```
@time begin
    min(data...)
end
```

```
  0.491609 seconds (2.10 M allocations: 82.025 MiB, 11.12% gc time, 8.9
```

Out[77]:
```
-4.846415885152558
```

Something `rand`.

home

The Monty Hall Problem

In [78]:
```
fix_seed!()


function montyHall(switch_policy)
    prize, choice = rand(1:3), rand(1:3)
    if prize == choice
        revealed = rand(setdiff(1:3,choice))
    else
        revealed = rand(setdiff(1:3,[prize,choice]))
    end

    if switch_policy
        choice = setdiff(1:3,[revealed,choice])[1]
    end
    return choice == prize
end

N = 10^6
println("Success probability with policy I (stay): ",  sum([montyHall(fa
println("Success probability with policy II (switch): ", sum([montyHall(
```

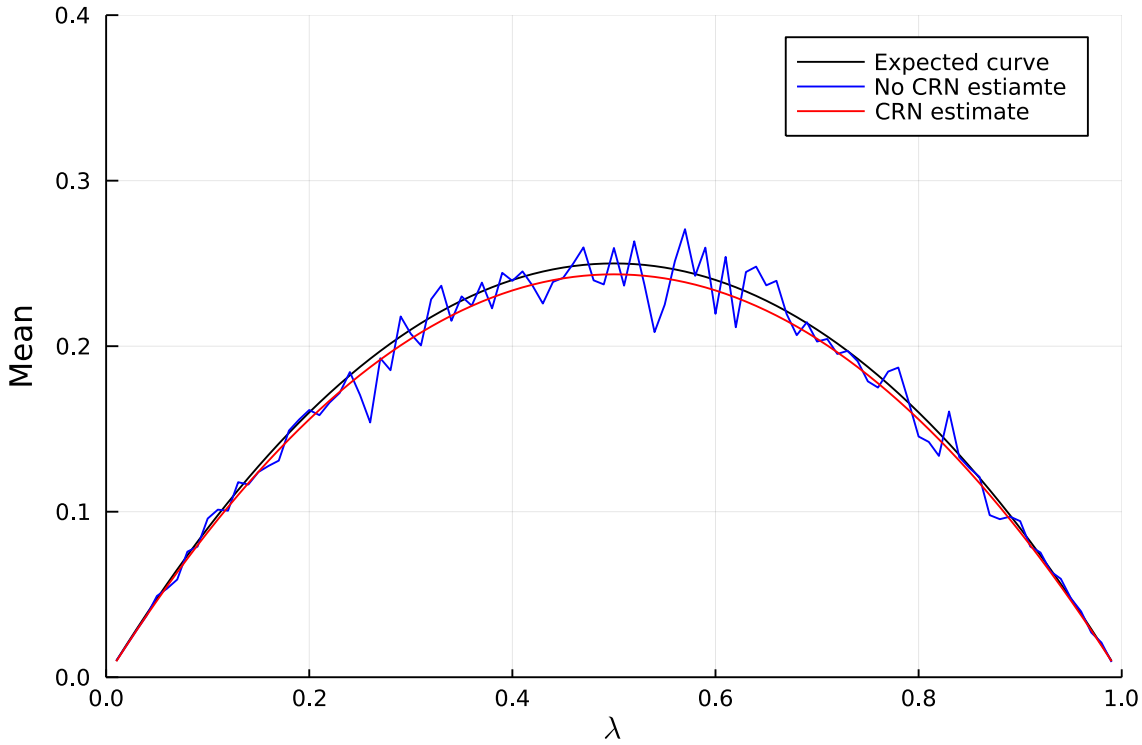
```
Success probability with policy I (stay): 0.334146
Success probability with policy II (switch): 0.666056
```

Common random numbers

[Math Processing Error][Math Processing Error]

In [79]:
```
seed = 1
N = 100
λ_grid = 0.01:0.01:0.99

theorM(λ) = mean(Uniform(0,2λ*(1-λ)))
estM(λ) = mean(rand(Uniform(0,2λ*(1-λ)),N))

function estM(λ, seed)
    Random.seed!(seed)
    estM(λ)
end

trueM = theorM.(λ_grid)
estM0 = estM.(λ_grid)
estMCRN = estM.(λ_grid,seed)

plot(λ_grid, trueM, c=:black, label="Expected curve")
plot!(λ_grid, estM0, c=:blue, label="No CRN estiamte")
plot!(λ_grid, estMCRN, c=:red, label="CRN estimate",
    xlims=(0,1), ylims=(0,0.4), xlabel=L"\lambda", ylabel = "Mean")
```

Out[79]:



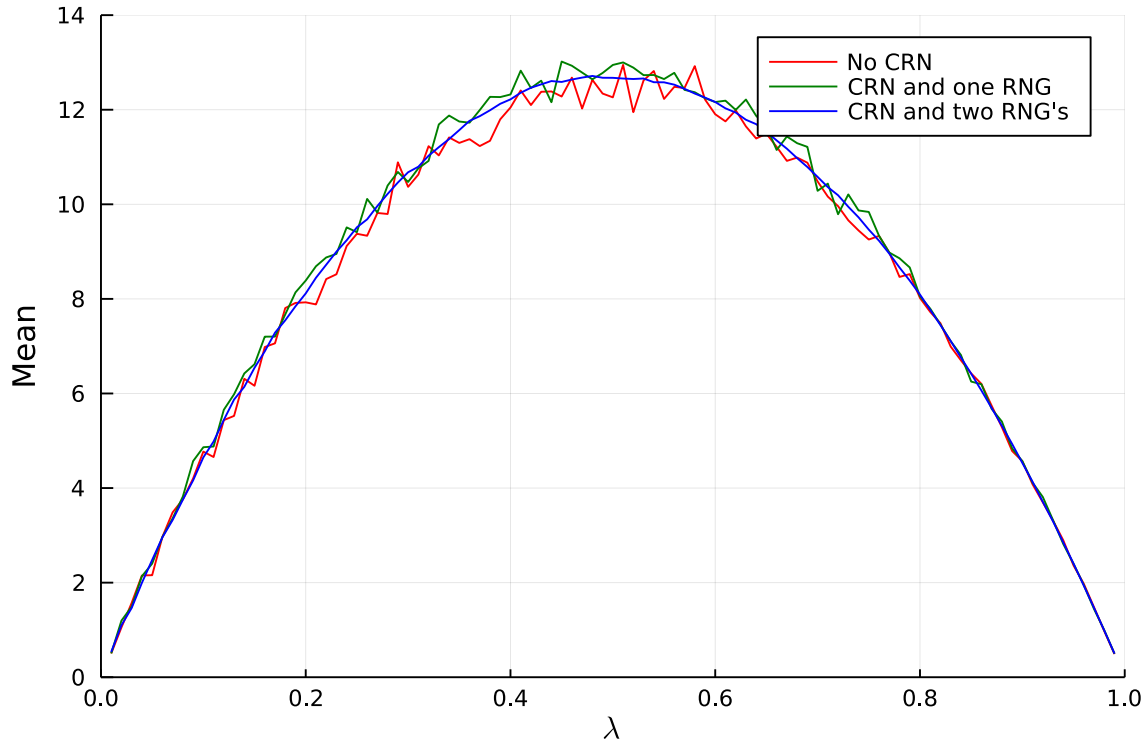
Multiple RNGs

Say we want to estimate the mean of *[Math Processing Error]* with *[Math Processing Error]* and *[Math Processing Error]* with *[Math Processing Error]* and *[Math Processing Error]*.

[Math Processing Error]

In [80]:
```
K = 50

prn(λ,rng) = quantile(Poisson(λ),rand(rng))
zDist(λ) = Uniform(0,2*(1-λ))

rv(λ,rng) = sum([rand(rng,zDist(λ)) for _ in 1:prn(K*λ, rng)])
rv2(λ,rng1,rng2) = sum([rand(rng1,zDist(λ)) for _ in 1:prn(K*λ,rng2)])

mEst(λ,rng) = mean([rv(λ,rng) for _ in 1:N])
mEst2(λ,rng1,rng2) = mean([rv2(λ,rng1,rng2) for _ in 1:N])

#No Common Random Numbers (CRN)
function mGraph0(seed)
    singleRng = MersenneTwister(seed)
    [mEst(λ,singleRng) for λ in λ_grid]
end

#CRN but single random number generator (RNG)
mGraph1(seed) = [mEst(λ,MersenneTwister(seed)) for λ in λ_grid]

#CRN with two random number generators
mGraph2(seed1,seed2) = [mEst2(λ,MersenneTwister(seed1), MersenneTwister(

plot(λ_grid,mGraph0(1987), c=:red, label="No CRN")
plot!(λ_grid,mGraph1(1987), c=:green, label="CRN and one RNG")
plot!(λ_grid,mGraph2(1987,1988), c=:blue, label="CRN and two RNG's", xli
    xlabel=L"\lambda", ylabel = "Mean")
```

Out[80]:



In [81]:
```
argMaxλ(graph) = λ_grid[findmax(graph)[2]]

M = 10^3
std0 = std([argMaxλ(mGraph0(seed)) for seed in 1:M])
std1 = std([argMaxλ(mGraph1(seed)) for seed in 1:M])
std2 = std([argMaxλ(mGraph2(seed,seed+M)) for seed in 1:M])

println("Standard deviation with no CRN: ", std0)
println("Standard deviation with CRN and single RNG: ", std1)
println("Standard deviation with CRN and two RNGs: ", std2)
```

```
Standard deviation with no CRN: 0.03708052002015299
Standard deviation with CRN and single RNG: 0.03411444555309959
Standard deviation with CRN and two RNGs: 0.014645353747396703
```

Do you still miss R? So Just `RCall`.

home

See the docs for RCall.jl (https://juliainterop.github.io/RCall.jl/stable/)

In [82]:
```
using RCall

data1 = DataFrame(CSV.File("./data/machine1.csv", header=false))[:,1]
data2 = DataFrame(CSV.File("./data/machine2.csv", header=false))[:,1]
data3 = DataFrame(CSV.File("./data/machine3.csv", header=false))[:,1]

function R_ANOVA(allData)
    data = vcat([ [x fill(i, length(x))] for (i, x) in enumerate(allDat
    df = DataFrame(data, [:Diameter, :MachNo])
    @rput df

    R"""
    df$MachNo <- as.factor(df$MachNo)
    anova <- summary(aov( Diameter ~ MachNo, data=df))
    fVal <- anova[[1]]["F value"][[1]][1]
    pVal <- anova[[1]]["Pr(>F)"][[1]][1]
    """
    println("R ANOVA f-value: ", @rget fVal)
    println("R ANOVA p-value: ", @rget pVal)
end

R_ANOVA([data1, data2, data3])
```

```
R ANOVA f-value: 10.516968568709089
R ANOVA p-value: 0.00014236168817139574
```

To use Julia from R, use JuliaCall (https://cran.r-project.org/web/packages/JuliaCall/index.html)

Some `Plots.`

home

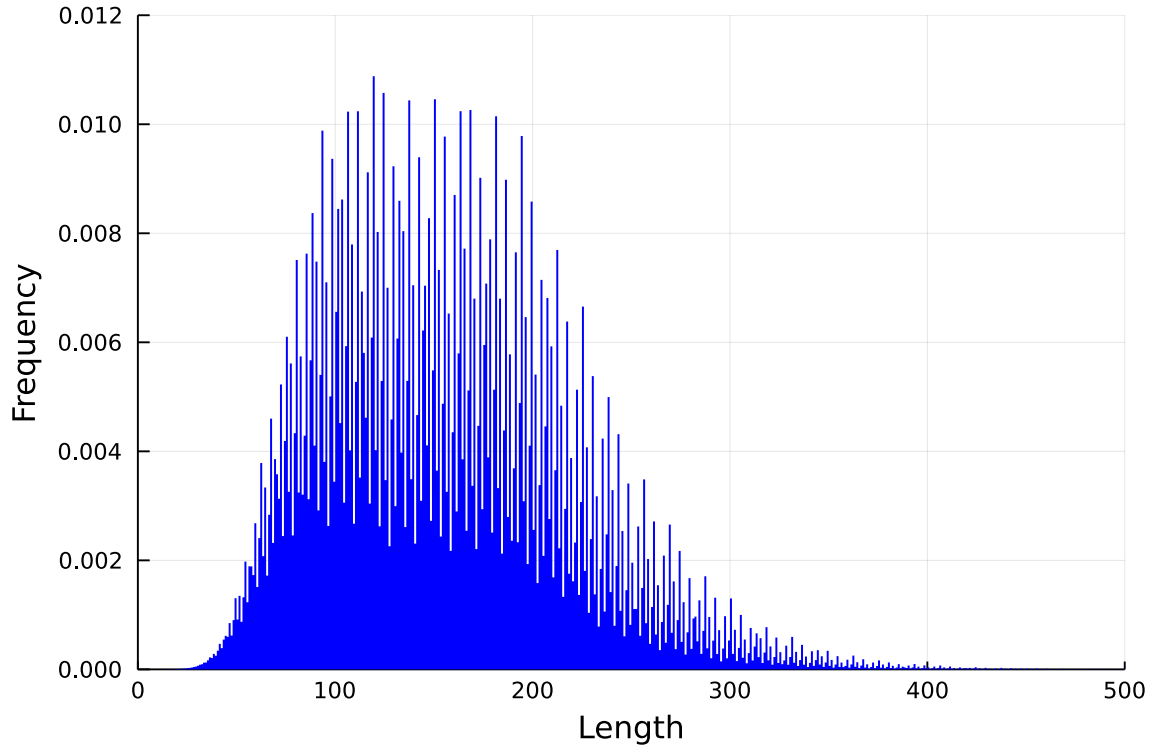
See example image gallery (https://statisticswithjulia.org/gallery.html) with code - part of "Statistics with Julia" book.

See docs for Plots.jl (http://docs.juliaplots.org/latest//) and StatsPlots.jl (https://github.com/JuliaPlots/StatsPlots.jl)

In [83]:

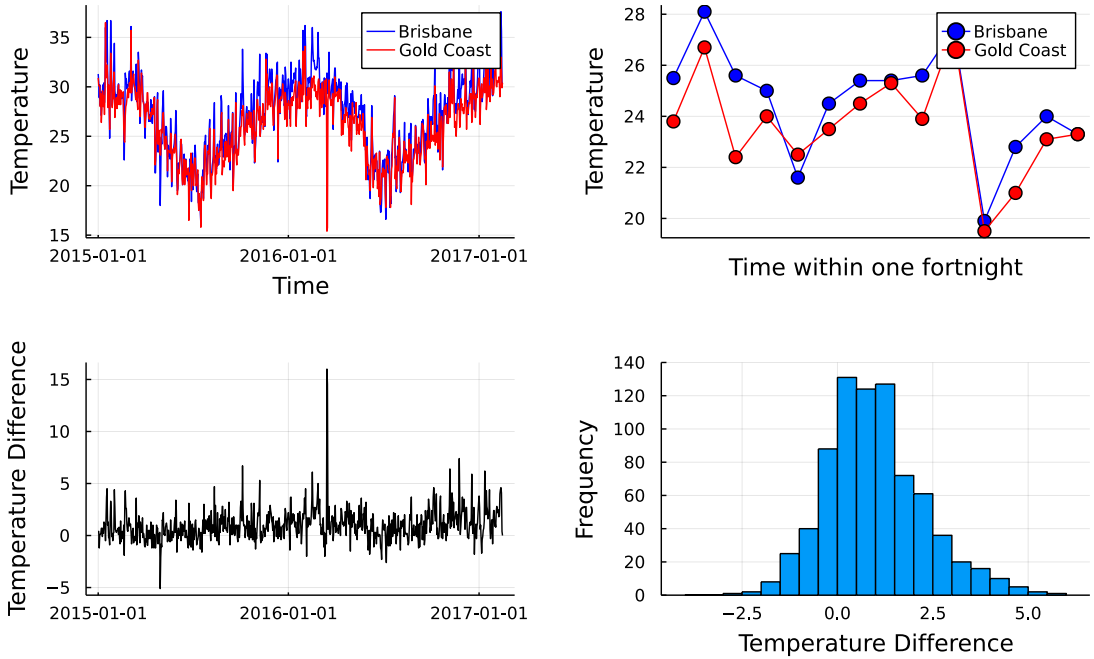
```
function hailLength(x::Int)
    n = 0
    while x != 1
        if x % 2 == 0
            x = x ÷ 2 #\div + [TAB]
        else
            x = 3x +1
        end
        n += 1
    end
    return n
end

lengths = [hailLength(x₀) for x₀ in 2:10^7] #\_0 + [TAB]

histogram(lengths, bins=1000, normed=:true,
    fill=(:blue, true), la=0, legend=:none,
    xlims=(0, 500), ylims=(0, 0.012),
    xlabel="Length", ylabel="Frequency")
```

Out[83]:



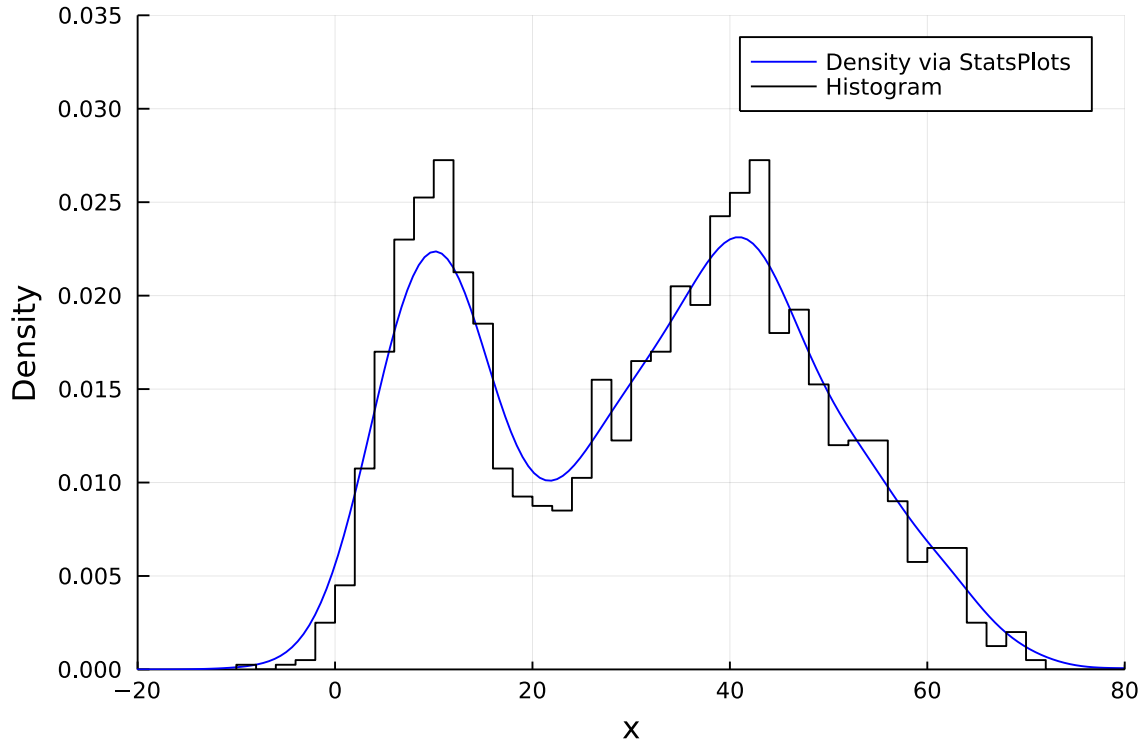
In [84]:
```
data = CSV.File("./data/temperatures.csv") |> DataFrame
brisbane = data.Brisbane
goldcoast = data.GoldCoast

diff = brisbane - goldcoast
dates = [Date(
            Year(data.Year[i]),
            Month(data.Month[i]),
            Day(data.Day[i])
        ) for i in 1:nrow(data)]

fortnightRange = 250:263
brisFortnight = brisbane[fortnightRange]
goldFortnight = goldcoast[fortnightRange]

p1 = plot(dates, [brisbane goldcoast],
        c=[:blue :red], xlabel="Time", ylabel="Temperature", label=["Bri
p2 = plot(dates[fortnightRange], [brisFortnight goldFortnight], xticks=:
        c=[:blue :red], m=(:dot, 5, Plots.stroke(1)), ylabel="Temperatur
        label=["Brisbane" "Gold Coast"], xlabel="Time within one fortnig
p3 = plot(dates, diff,
        c=:black, ylabel="Temperature Difference",legend=false)
p4 = histogram(diff, bins=-4:0.5:6,
        ylims=(0,140), legend = false,
        xlabel="Temperature Difference", ylabel="Frequency")
plot(p1,p2,p3,p4, size = (800,500), margin = 5mm)
```

Out[84]:


Temperature
Time
Brisbane
Gold Coast
2015-01-01
2016-01-01
2017-01-01
Temperature
Time within one fortnight
Brisbane
Gold Coast
Temperature Difference
Frequency
Temperature Difference

In [85]:
```
fix_seed!()

μ₁, σ₁ = 10, 5 #\mu + [TAB] \_1 + [TAB] etc...
μ₂, σ₂ = 40, 12
dist1, dist2 = Normal(μ₁,σ₁), Normal(μ₂,σ₂)
p = 0.3
mixRv() = (rand() <= p) ? rand(dist1) : rand(dist2)

n = 2000
data = [mixRv() for _ in 1:n]

density(data, c=:blue, label="Density via StatsPlots", xlims=(-20,80), y
stephist!(data, bins=50, c=:black, norm=true, label="Histogram", xlabel=
```

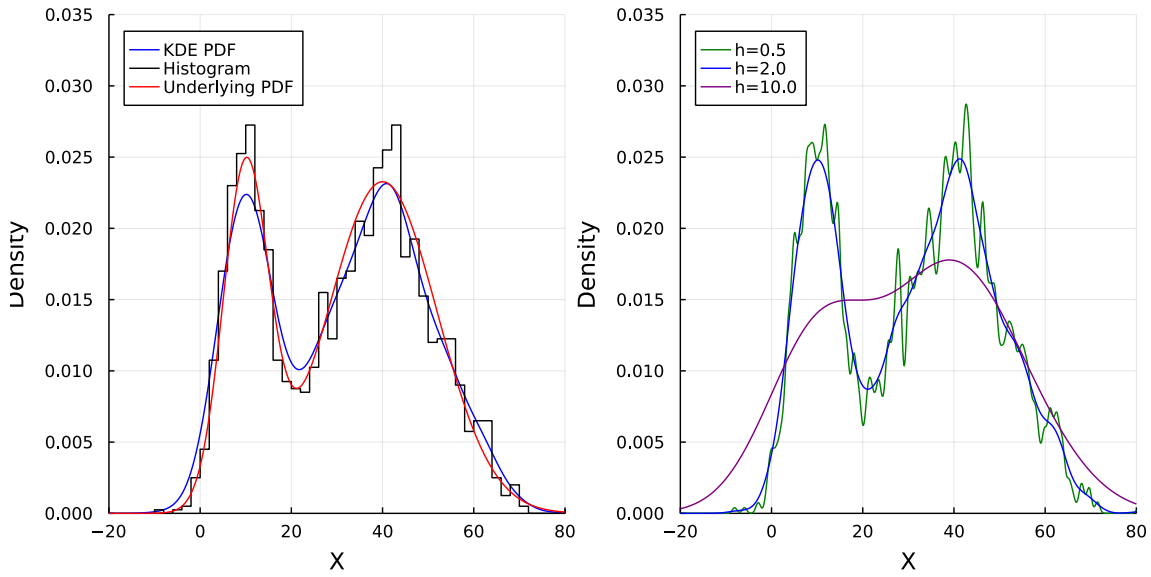
Out[85]:



In [86]:

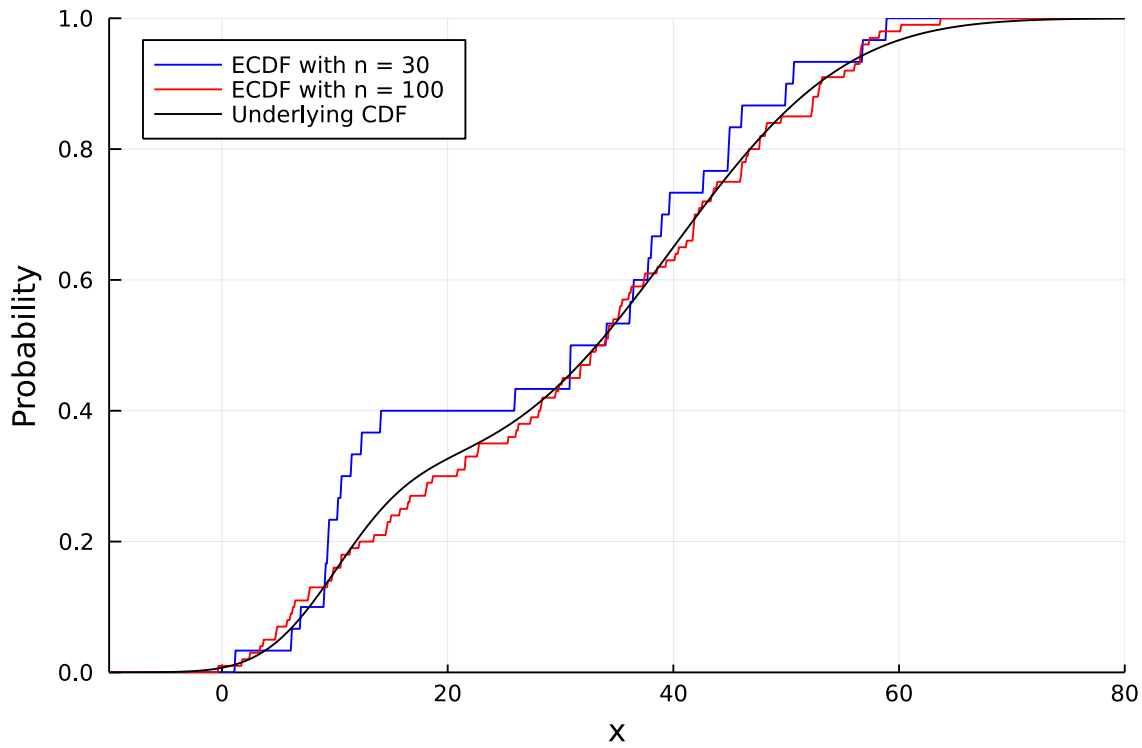
```
mixPDF(x) = p*pdf(dist1,x) + (1-p)*pdf(dist2,x)

kdeDist = kde(data)

xGrid = -20:0.1:80
pdfKDE = pdf(kdeDist,xGrid)

p1 = plot(xGrid, pdfKDE, c=:blue, label="KDE PDF")
stephist!(data, bins=50, c=:black, normed=:true, label="Histogram")
plot!(xGrid, mixPDF.(xGrid), c=:red, label="Underlying PDF",
    xlims=(-20,80), ylims=(0,0.035), legend=:topleft,
    xlabel="X", ylabel = "Density")

hVals = [0.5,2,10]
kdeS = [kde(data,bandwidth=h) for h in hVals]

p2 = plot(xGrid, pdf(kdeS[1],xGrid), c = :green, label= "h=$(hVals[1])")
plot!(xGrid, pdf(kdeS[2],xGrid), c = :blue, label= "h=$(hVals[2])")
plot!(xGrid, pdf(kdeS[3],xGrid), c = :purple, label= "h=$(hVals[3])",
    xlims=(-20,80), ylims=(0,0.035), legend=:topleft,
    xlabel="X", ylabel = "Density")

plot(p1,p2,size = (800,400))
```

Out[86]:



In [87]:
```
fix_seed!()

mixCDF(x) = p*cdf(dist1,x) + (1-p)*cdf(dist2,x)

n = [30, 100]
data1 = [mixRv() for _ in 1:n[1]]
data2 = [mixRv() for _ in 1:n[2]]

empiricalCDF1 = ecdf(data1)
empiricalCDF2 = ecdf(data2)

xGrid = -10:0.1:80
plot(xGrid,empiricalCDF1.(xGrid), c=:blue, label="ECDF with n = $(n[1])'
plot!(xGrid,empiricalCDF2.(xGrid), c=:red, label="ECDF with n = $(n[2])'
plot!(xGrid, mixCDF.(xGrid), c=:black, label="Underlying CDF",
    xlims=(-10,80), ylims=(0,1),
    xlabel="x", ylabel="Probability", legend=:topleft)
```

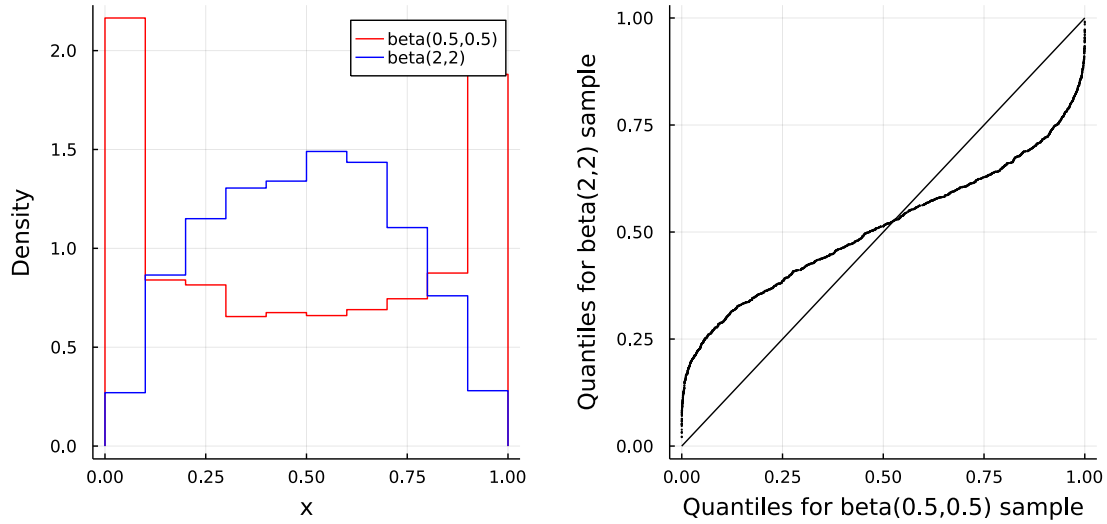
Out[87]:



In [88]:

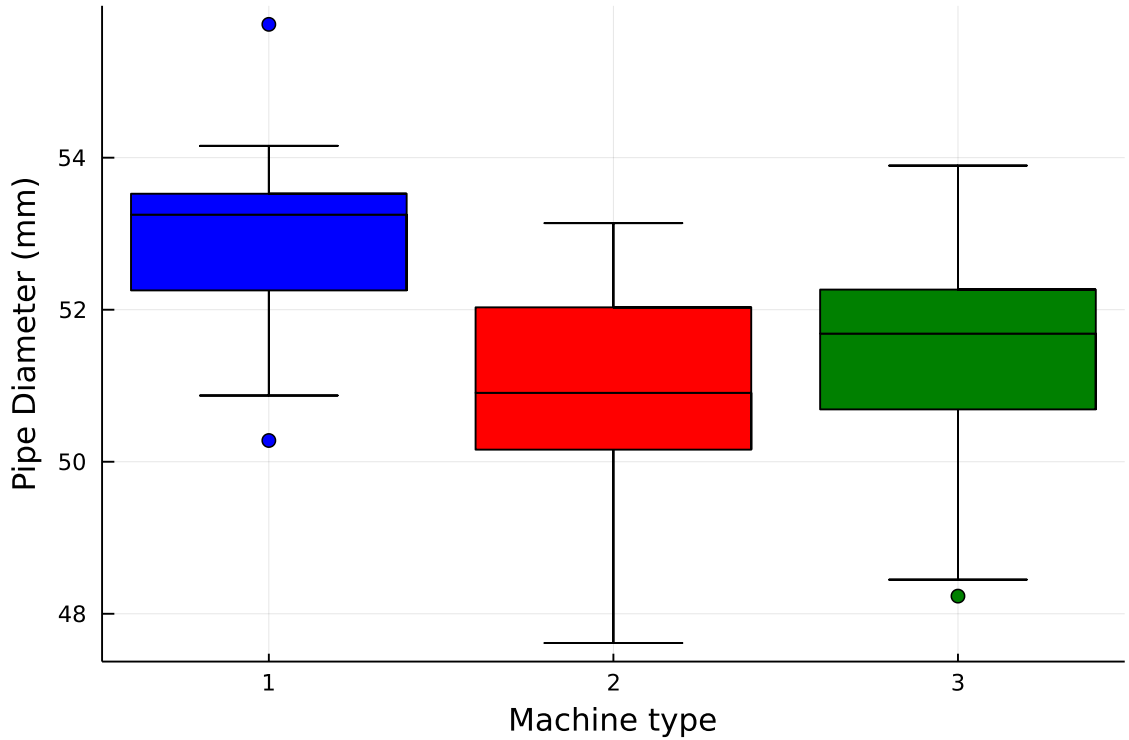
```
fix_seed!()

b1, b2 = 0.5 , 2
dist1, dist2, = Beta(b1,b1), Beta(b2,b2)

n = 2000
data1 = rand(dist1,n)
data2 = rand(dist2,n)

stephist(data1, bins=15, label = "beta($b1,$b1)", c = :red, normed = tru
p1 = stephist!(data2, bins=15, label = "beta($b2,$b2)",
        c = :blue, xlabel="x", ylabel="Density",normed = true)

p2 = qqplot(data1, data2, c=:black, ms=1, msw =0,
        xlabel="Quantiles for beta($b1,$b1) sample",
        ylabel="Quantiles for beta($b2,$b2) sample",
        legend=false)

plot(p1, p2, size=(800,400), margin = 5mm)
```

Out[88]:



In [89]:
```
data1 = parse.(Float64, readlines("./data/machine1.csv"))
data2 = parse.(Float64, readlines("./data/machine2.csv"))
data3 = parse.(Float64, readlines("./data/machine3.csv"))

boxplot([data1,data2,data3], c=[:blue :red :green], label="",
    xticks=([1:1:3;], ["1", "2", "3"]), xlabel="Machine type",
    ylabel="Pipe Diameter (mm)")
```

Out[89]:



Your favorite `Distribution`.

home

See Docs for `Distributions.jl` (https://juliastats.org/Distributions.jl/latest/)

In [90]:
```
dist = Normal(2.5,1.2)
```

Out[90]:
```
Normal{Float64}(μ=2.5, σ=1.2)
```

In [91]:
```
mean(dist), var(dist)
```

Out[91]:
```
(2.5, 1.44)
```

In [92]:
```
fix_seed!()
rand(dist,3,2,5) #a 3-tensor
```

Out[92]:
```
3×2×5 Array{Float64, 3}:
[:, :, 1] =
 3.31493  2.33818
 3.4941   3.20394
 2.07639  2.8568

[:, :, 2] =
 2.57794  4.3892
 2.36918  1.67331
 1.88295  1.58464

[:, :, 3] =
 2.97698   2.27491
 3.47396   0.571292
 2.08437  -0.476951

[:, :, 4] =
 5.23148  1.7785
 2.76363  3.87073
 2.35943  2.39366

[:, :, 5] =
 2.83536  3.06846
 2.63371  2.86028
 2.07054  1.58479
```

In [93]:
```
? NegativeBinomial
```

```
search: NegativeBinomial NegativeBinomialLink
```

Out[93]:
```
NegativeBinomial(r,p)
```

A *Negative binomial distribution* describes the number of failures before the `r` th success in a sequence of independent Bernoulli trials. It is parameterized by `r`, the number of successes, and `p`, the probability of success in an individual trial.

[Math Processing Error]

The distribution remains well-defined for any positive `r`, in which case

[Math Processing Error]

```
NegativeBinomial()        # Negative binomial distribution with r = 1 a
NegativeBinomial(r, p)    # Negative binomial distribution with r succe

params(d)       # Get the parameters, i.e. (r, p)
succprob(d)     # Get the success rate, i.e. p
failprob(d)     # Get the failure rate, i.e. 1 - p
```

External links:

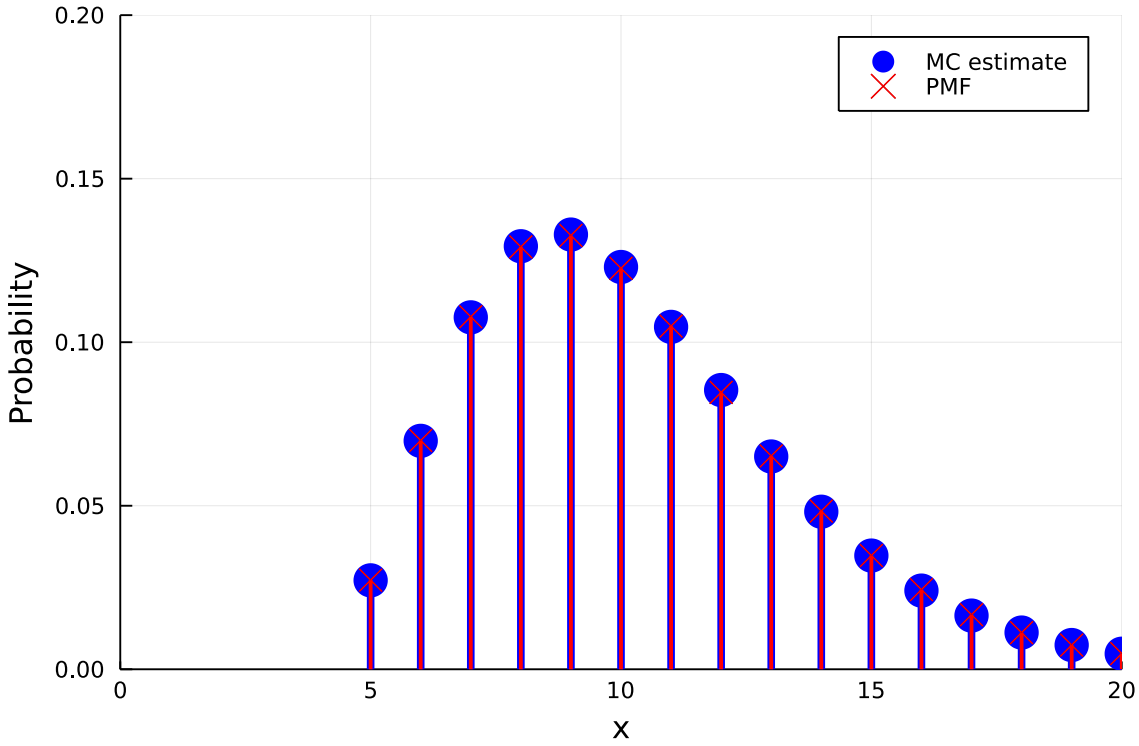
- Negative binomial distribution on Wolfram (https://reference.wolfram.com/language/ref/NegativeBinomialDistribution.html)

Note: The definition of the negative binomial distribution in Wolfram is different from the Wikipedia definition (https://en.wikipedia.org/wiki/Negative_binomial_distribution). In Wikipedia, `r` is the number of failures and `k` is the number of successes.

In [94]:

```
fix_seed!()

function rouletteSpins(r,p)
    x = 0
    wins = 0
    while true
        x += 1
        if rand() < p
            wins += 1
            if wins == r
                return x
            end
        end
    end
end

r, p, N = 5, 18/37,10^6
xGrid = r:r+15

mcEstimate = counts([rouletteSpins(r,p) for _ in 1:N],xGrid)/N

nbDist = NegativeBinomial(r,p)
nbPmf = [pdf(nbDist,x-r) for x in xGrid]

plot( xGrid, mcEstimate,
    line=:stem, marker=:circle, c=:blue,
    ms=10, msw=0, lw=4, label="MC estimate")
plot!( xGrid, nbPmf, line=:stem,
    marker=:xcross, c=:red, ms=6, msw=0, lw=2, label="PMF",
    xlims=(0,maximum(xGrid)), ylims=(0,0.2),
    xlabel="x", ylabel="Probability")
```

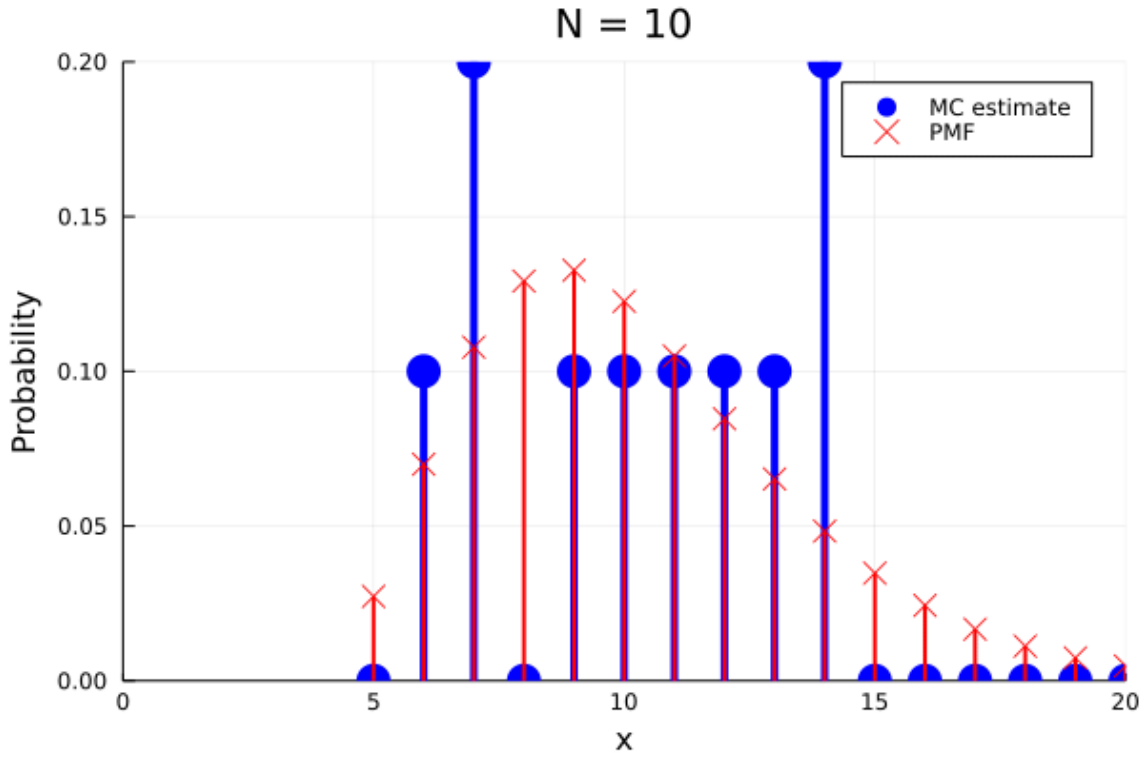
Out[94]:


0.20
0.15
0.10
0.05
0.00
Probability
0
5
10
15
20
x
MC estimate
PMF

In [95]:
```
function plot_it(N)
    fix_seed!()
    mcEstimate = counts([rouletteSpins(r,p) for _ in 1:N],xGrid)/N

    nbDist = NegativeBinomial(r,p)
    nbPmf = [pdf(nbDist,x-r) for x in xGrid]

    plot( xGrid, mcEstimate,
        line=:stem, marker=:circle, c=:blue,
        ms=10, msw=0, lw=4, label="MC estimate")
    plot!( xGrid, nbPmf, line=:stem,
        marker=:xcross, c=:red, ms=6, msw=0, lw=2, label="PMF",
        xlims=(0,maximum(xGrid)), ylims=(0,0.2),
        xlabel="x", ylabel="Probability",
        title="N = $N")
end

anim = Animation()

for N in union(10:10:100,200:100,1000,2000:1000:10^4)
    plot_it(N)
    frame(anim)
end

gif(anim,"sample_animation.gif",fps = 3)
```

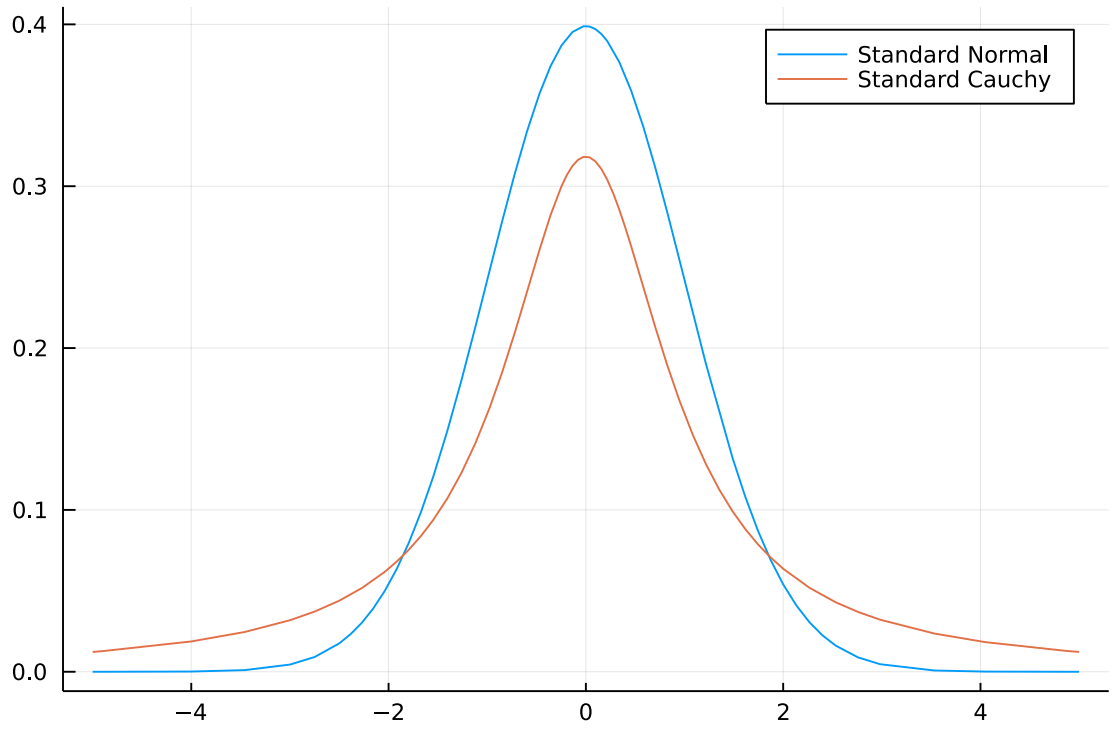
```
┌ Info: Saved animation to
│   fn = /Users/uqjnazar/git/mine/JuliaCon2021-StatisticsWithJuliaFromT
└ @ Plots /Users/uqjnazar/.julia/packages/Plots/S2aH5/src/animation.jl:
```

Out[95]:


N = 10
MC estimate
PMF
Probability
x
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20

In [96]:
```
plot((x)->pdf(Normal(),x),label = "Standard Normal")
plot!((x)->pdf(Cauchy(),x),label = "Standard Cauchy")
```

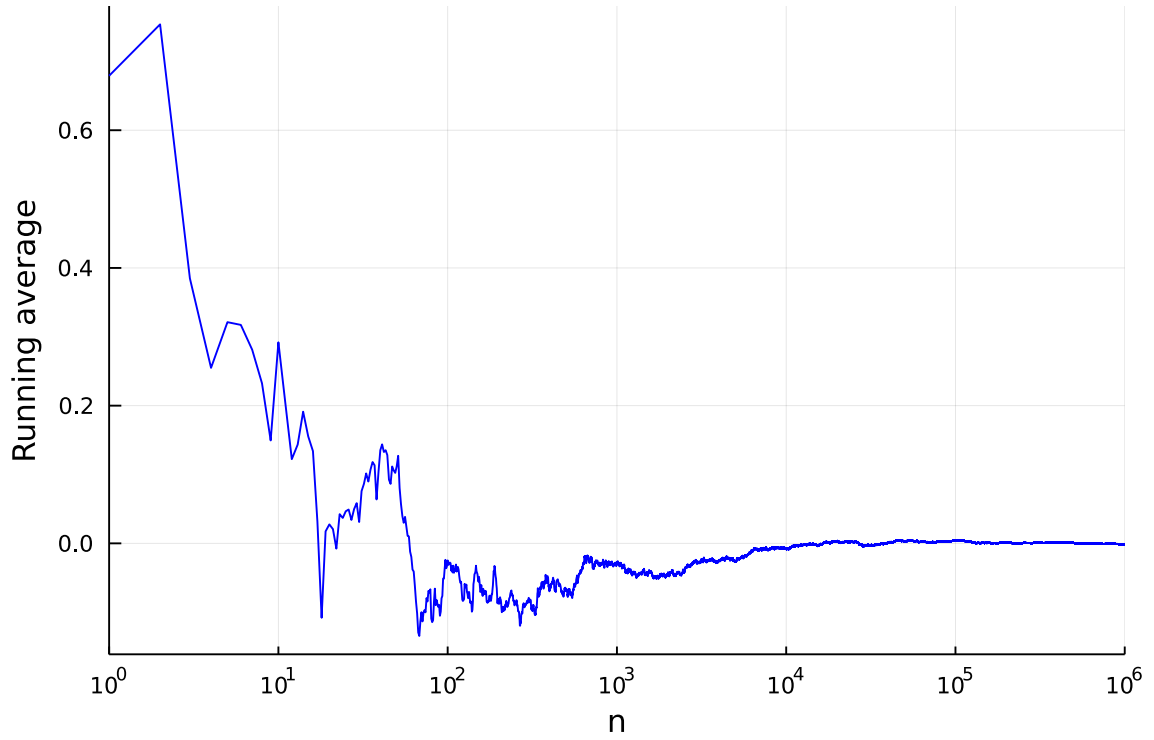
Out[96]:



In [97]:

```
fix_seed!()

n = 10^6
data = rand(Normal(), n)
#data = rand(Cauchy(),n) #Try this instead
averages = accumulate(+,data) ./ (1:n)

plot( 1:n, averages,
    c=:blue, legend=:none, xscale=:log10, xlims=(1,n), xlabel="n", ylabe
```

Out[97]:

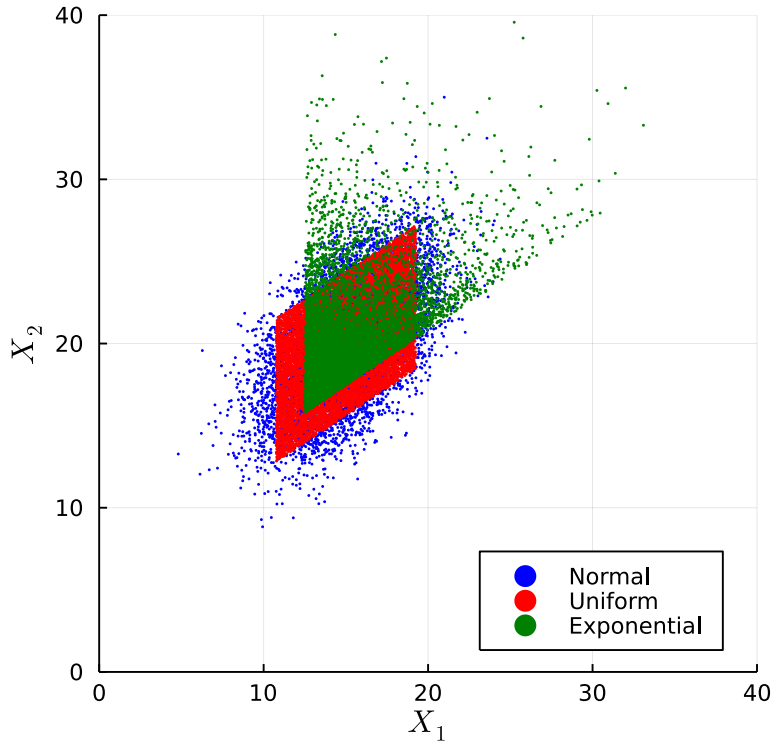


In [98]:

```
fix_seed!()

N = 10^4

Σ = [ 6 4 ;
      4 9]
μ = [15 ;
     20]
A = cholesky(Σ).L

rngGens = [()->rand(Normal()),
           ()->rand(Uniform(-sqrt(3),sqrt(3))),
           ()->rand(Exponential())-1]

rv(rg) = A*[rg(),rg()] + μ

data = [[rv(r) for _ in 1:N] for r in rngGens]

stats(data) = begin
    data1, data2 = first.(data),last.(data)
    println(round(mean(data1),digits=2), "\t",round(mean(data2),digits=2
            round(var(data1),digits=2), "\t", round(var(data2),digits=2)
            round(cov(data1,data2),digits=2))
end

println("Mean1\tMean2\tVar1\tVar2\tCov")
for d in data
    stats(d)
end

scatter(first.(data[1]), last.(data[1]), c=:blue, ms=1, msw=0, label="No
scatter!(first.(data[2]), last.(data[2]), c=:red, ms=1, msw=0, label="Un
scatter!(first.(data[3]), last.(data[3]), c=:green, ms=1, msw=0,label="E
    xlims=(0,40), ylims=(0,40), legend=:bottomright, ratio=:equal,
    xlabel=L"X_1", ylabel=L"X_2")
```

```
Mean1   Mean2   Var1    Var2    Cov
15.03   20.0    6.07    8.98    4.08
15.01   19.97   6.08    9.09    4.05
15.01   20.0    6.02    9.28    4.04
```

Out[98]:



In [99]:
```
dist = MvNormal([1,1],[2 2.3; 2.3 4])
```

Out[99]:
```
FullNormal(
dim: 2
μ: [1.0, 1.0]
Σ: [2.0 2.3; 2.3 4.0]
)
```

In [100]:
```
cor(dist)
```

Out[100]:
```
2×2 Matrix{Float64}:
 1.0       0.813173
 0.813173  1.0
```

```
In [101]: rand(dist,20)
```

```
Out[101]: 2×20 Matrix{Float64}:
 -1.14987  -0.000710292  0.734429  …  1.78702  2.84194  2.41636  1.1962
 -1.87928   0.0846214    1.79441      1.42384  4.54479  3.26142  1.5882
```

See also TruncatedDistributions.jl (https://github.com/yoninazarathy/TruncatedDistributions.jl) - still in progress as of July 2021.

We love `DataFrames.`

home

See also:

- Yesterday's (recorded) Dataframes tutorial (https://github.com/bkamins/JuliaCon2021-DataFrames-Tutorial) or here (https://pretalx.com/juliacon2021/talk/FXZXMB/) by Bogumił Kamiński.
- Docs for DataFrames.jl (https://dataframes.juliadata.org/stable/).

In [102]:
```
ENV["LINES"] = 10
data = CSV.File("./data/purchaseData.csv") |> DataFrame
```

Out[102]: 200 rows × 4 columns

	Name	Date	Grade	Price
	String?	String?	String?	Int64?
1	MARYANNA	14/09/2008	A	79700
2	REBECCA	11/03/2008	B	*missing*
3	ASHELY	5/08/2008	E	24311
4	KHADIJAH	2/09/2008	*missing*	38904
5	TANJA	1/12/2008	C	47052
6	JUDIE	17/05/2008	D	34365
7	NOE	15/08/2008	A	79344
8	JACALYN	7/10/2008	E	21474
9	SANDY	5/04/2008	D	*missing*
10	ALEJANDRA	16/12/2008	E	20916
⋮	⋮	⋮	⋮	⋮

In [103]:
```
size(data)
```

Out[103]:
```
(200, 4)
```

In [104]:
```
names(data)
```

Out[104]:
```
4-element Vector{String}:
 "Name"
 "Date"
 "Grade"
 "Price"
```

In [105]:
```
first(data, 6)
```

Out[105]: 6 rows × 4 columns

	Name String?	Date String?	Grade String?	Price Int64?
1	MARYANNA	14/09/2008	A	79700
2	REBECCA	11/03/2008	B	*missing*
3	ASHELY	5/08/2008	E	24311
4	KHADIJAH	2/09/2008	*missing*	38904
5	TANJA	1/12/2008	C	47052
6	JUDIE	17/05/2008	D	34365

In [106]:
```
describe(data)
```

Out[106]: 4 rows × 7 columns

	variable Symbol	mean Union…	min Any	median Union…	max Any	nmissing Int64	eltype Union
1	Name		ABBEY		ZACHARY	17	Union{Missing, String}
2	Date		1/07/2008		9/10/2008	4	Union{Missing, String}
3	Grade		A		E	13	Union{Missing, String}
4	Price	39702.0	8257	38045.5	79893	14	Union{Missing, Int64}

In [107]:
```
println("Grade of person 1: ", data[1, 3],
        ", ", data[1,:Grade],
        ", ", data.Grade[1])
```

```
Grade of person 1: A, A, A
```

In [108]:
```
data[[1,2,4], :]
```

Out[108]: 3 rows × 4 columns

	Name	Date	Grade	Price
	String?	String?	String?	Int64?
1	MARYANNA	14/09/2008	A	79700
2	REBECCA	11/03/2008	B	*missing*
3	KHADIJAH	2/09/2008	*missing*	38904

In [109]:
```
data[13:15, :Name]
```

Out[109]:
```
3-element Vector{Union{Missing, String}}:
 "SAMMIE"
 missing
 "STACEY"
```

In [110]:
```
data.Name[13:15]
```

Out[110]:
```
3-element Vector{Union{Missing, String}}:
 "SAMMIE"
 missing
 "STACEY"
```

In [111]:
```
data[13:15, [:Name]]
```

Out[111]: 3 rows × 1 columns

	Name String?
1	SAMMIE
2	*missing*
3	STACEY

Mising values

In [112]:
```
mean(data.Price)
```

Out[112]:
```
missing
```

In [113]:
```
ismissing.(data.Price) |> sum
```

Out[113]:
```
14
```

In [114]:
```
mean(skipmissing(data.Price))
```

Out[114]:
```
39702.01075268817
```

In [115]:
```
data.Grade[1:4]
```

Out[115]:
```
4-element PooledArrays.PooledVector{Union{Missing, String}, UInt32, Vec
 "A"
 "B"
 "E"
 missing
```

In [116]:
```
coalesce.(data.Grade, "QQ")[1:4]
```

Out[116]:
```
4-element Vector{String}:
 "A"
 "B"
 "E"
 "QQ"
```

In [117]:
```
dropmissing(data,:Price)
```

Out[117]: 186 rows × 4 columns

	Name	Date	Grade	Price
	String?	String?	String?	Int64
1	MARYANNA	14/09/2008	A	79700
2	ASHELY	5/08/2008	E	24311
3	KHADIJAH	2/09/2008	*missing*	38904
4	TANJA	1/12/2008	C	47052
5	JUDIE	17/05/2008	D	34365
6	NOE	15/08/2008	A	79344
7	JACALYN	7/10/2008	E	21474
8	ALEJANDRA	16/12/2008	E	20916
9	GARY	8/05/2008	C	40081
10	ROSELEE	29/06/2008	*missing*	37661
⋮	⋮	⋮	⋮	⋮

In [118]:
```
completecases(data) |> sum
```

Out[118]:
```
153
```

In [119]:
```
full_obs = completecases(data) |> findall
```

Out[119]:
```
153-element Vector{Int64}:
   1
   3
   5
   ⋮
 199
 200
```

In [120]:
```
data[setdiff(1:size(data,1), full_obs),:]
```

Out[120]: 47 rows × 4 columns

	Name String?	Date String?	Grade String?	Price Int64?
1	REBECCA	11/03/2008	B	*missing*
2	KHADIJAH	2/09/2008	*missing*	38904
3	SANDY	5/04/2008	D	*missing*
4	ROSELEE	29/06/2008	*missing*	37661
5	*missing*	12/12/2008	E	17328
6	RASHIDA	*missing*	C	44597
7	MINERVA	31/07/2008	*missing*	45751
8	JARRETT	30/11/2008	*missing*	41182
9	*missing*	1/10/2008	E	20229
10	*missing*	24/03/2008	E	24633
⋮	⋮	⋮	⋮	⋮

Gotta have some basic inference.

home

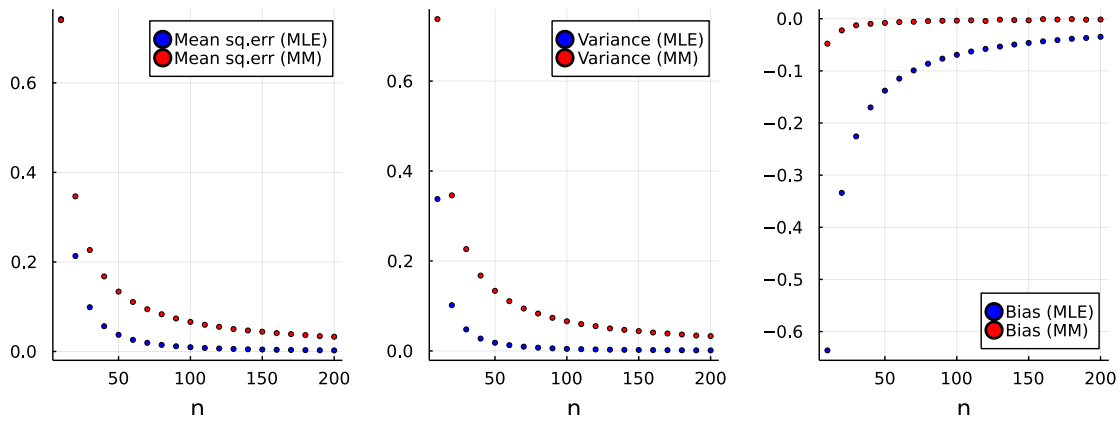
Point estimation

In [121]:

```
fix_seed!()

N = 10^5
nMin, nStep, nMax = 10, 10, 200
nn = Int(nMax/nStep)
sampleSizes = nMin:nStep:nMax
trueB = 5
trueDist = Uniform(-2, trueB)

#MLE for the upper bound
MLEest(data) = maximum(data)

#Method of moments estimator for the upper bound
MMest(data)  = mean(data) + sqrt(3)*std(data)

res = Dict{Symbol,Array{Float64}}( ((sym) -> sym => Array{Float64}(undef
                                  [:MSeMLE,:MSeMM, :VarMLE,:VarMM,:BiasMLE
for (i, n) in enumerate(sampleSizes)
    mleEst, mmEst = Array{Float64}(undef, N), Array{Float64}(undef, N)
    for j in 1:N
        sample    = rand(trueDist,n)
        mleEst[j] = MLEest(sample)
        mmEst[j]  = MMest(sample)
    end
    meanMLE, meanMM = mean(mleEst), mean(mmEst)
    varMLE, varMM = var(mleEst), var(mmEst)

    res[:MSeMLE][i] = varMLE + (meanMLE - trueB)^2
    res[:MSeMM][i] = varMM + (meanMM - trueB)^2
    res[:VarMLE][i] = varMLE
    res[:VarMM][i] = varMM
    res[:BiasMLE][i] = meanMLE - trueB
    res[:BiasMM][i] = meanMM - trueB
end

p1 = scatter(sampleSizes, [res[:MSeMLE] res[:MSeMM]], c=[:blue :red],
    label=["Mean sq.err (MLE)" "Mean sq.err (MM)"])
p2 = scatter(sampleSizes, [res[:VarMLE] res[:VarMM]], c=[:blue :red],
    label=["Variance (MLE)" "Variance (MM)"])
p3 = scatter(sampleSizes, [res[:BiasMLE] res[:BiasMM]], c=[:blue :red],
    label=["Bias (MLE)" "Bias (MM)"],legend = :bottomright)
```

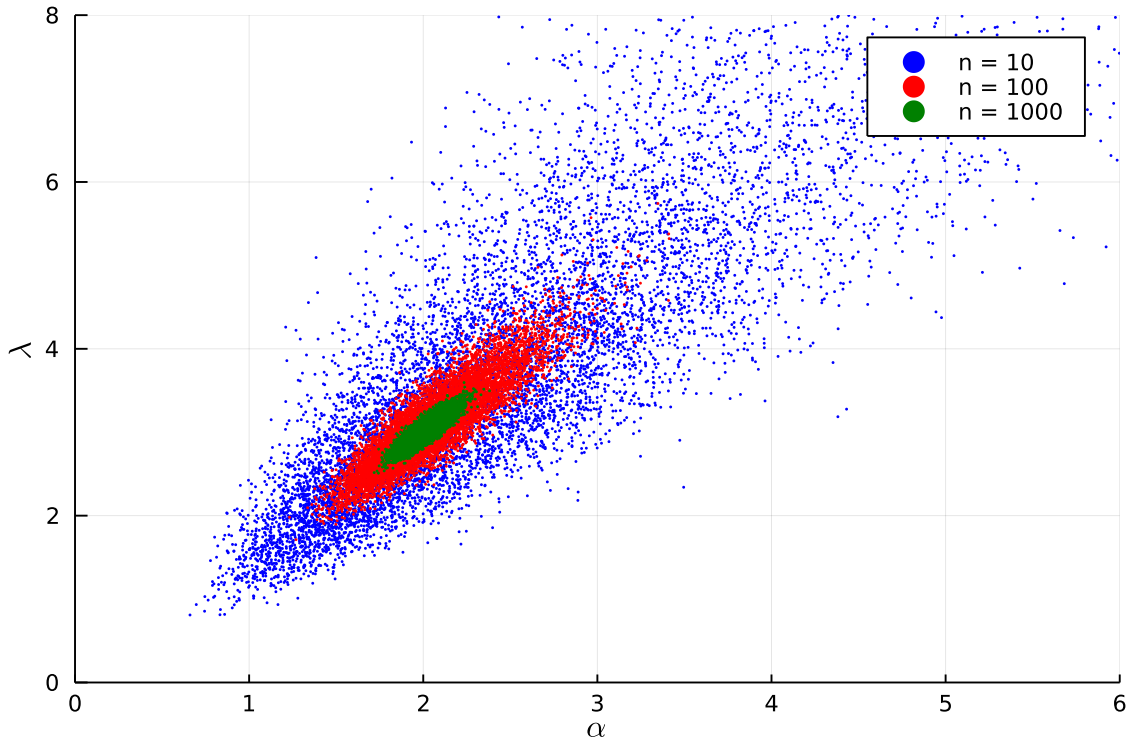
```
plot(p1, p2, p3, ms=2, shape=:circle, xlabel="n",
    layout=(1,3), size=(800, 300), margin = 2mm)
```

Out[121]:



In [122]:
```
fix_seed!()

eq(alpha, xb, xbl) = log(alpha) - digamma(alpha) - log(xb) + xbl

actualAlpha, actualLambda = 2, 3
gammaDist = Gamma(actualAlpha,1/actualLambda)

function mle(sample)
    alpha  = find_zero( (a)->eq(a,mean(sample),mean(log.(sample))), 1)
    lambda = alpha/mean(sample)
    return [alpha,lambda]
end

N = 10^4

mles10   = [mle(rand(gammaDist,10)) for _ in 1:N]
mles100  = [mle(rand(gammaDist,100)) for _ in 1:N]
mles1000 = [mle(rand(gammaDist,1000)) for _ in 1:N]

scatter(first.(mles10), last.(mles10),
    c=:blue, ms=1, msw=0, label="n = 10")
scatter!(first.(mles100), last.(mles100),
    c=:red, ms=1, msw=0, label="n = 100")
scatter!(first.(mles1000), last.(mles1000),
    c=:green, ms=1, msw=0, label="n = 1000",
    xlims=(0,6), ylims=(0,8), xlabel=L"\alpha", ylabel=L"\lambda")
```

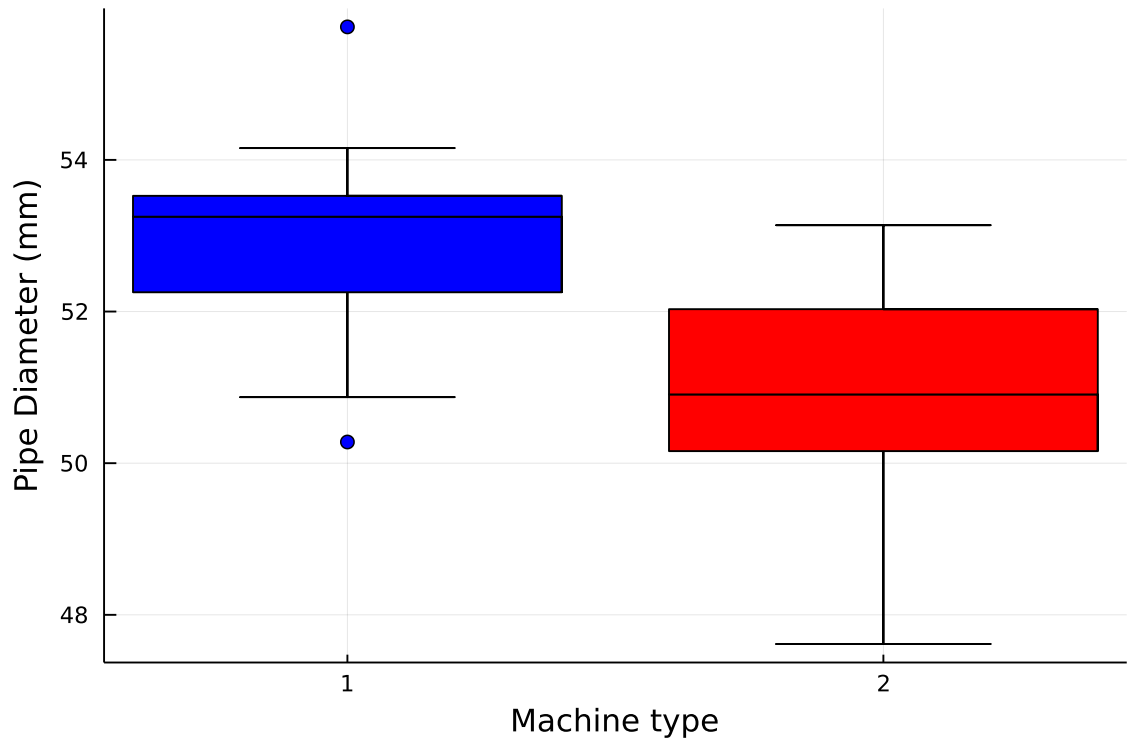
Out[122]:



Confidence Intervals

In [123]:
```
data1 = (CSV.File("./data/machine1.csv", header=false) |> DataFrame)[:,1
data2 = (CSV.File("./data/machine2.csv", header=false) |> DataFrame)[:,1
boxplot([data1,data2], c=[:blue :red], label="",
    xticks=([1:2;], ["1", "2"]), xlabel="Machine type",
    ylabel="Pipe Diameter (mm)")
```

Out[123]:



In [124]:
```
α = 0.05 #\alpha + [TAB]
confint(EqualVarianceTTest(data1,data2),α)
```

Out[124]:
```
(1.1127539574575822, 2.90486485574026)
```

In [161]:
```
#Doing it manually
xBar1, xBar2 = mean(data1), mean(data2)
n1, n2 = length(data1), length(data2)
t = quantile(TDist(n1+n2-2),1-α/2)

s1, s2 = std(data1), std(data2)
sP = sqrt(((n1-1)*s1^2 + (n2-1)*s2^2) / (n1+n2-2))

(xBar1 - xBar2 - t*sP* sqrt(1/n1 + 1/n2), xBar1 - xBar2 + t*sP* sqrt(1/r
```

Out[161]:
```
(1.1127539574575822, 2.90486485574026)
```

Hypothesis Tests

In [162]:
```
δ₀ = 0
test_result =UnequalVarianceTTest(data1, data2, δ₀)
```

Out[162]:
```
Two sample t-test (unequal variance)
------------------------------------
Population details:
    parameter of interest:   Mean difference
    value under h_0:         0
    point estimate:          2.00881
    95% confidence interval: (1.096, 2.9216)

Test summary:
    outcome with 95% confidence: reject h_0
    two-sided p-value:           <1e-04

Details:
    number of observations:   [20,18]
    t-statistic:              4.483705005611673
    degrees of freedom:       31.82453144280282
    empirical standard error: 0.4480244360600785
```

In [163]:
```
test_result.df
```

Out[163]:
```
31.82453144280282
```

In [164]:
```
pvalue(test_result)
```

Out[164]:
```
8.936189820682969e-5
```

In [165]:
```
methods(pvalue)
```

Out[165]: # 35 methods for generic function **pvalue**:

- pvalue(t::**HypothesisTests.VarianceEqualityTest{TD}**; *tail*) *where TD<: (ContinuousDistribution{F} where F<:VariateForm)* in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/var_equality.jl:30 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/var_equality.jl)
- pvalue(test::**CorrelationTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/correlation.jl:84 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/correlation.jl)
- pvalue(test::**WaldWolfowitzTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/wald_wolfowitz.jl:15 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/wald_wolfowitz.jl)
- pvalue(x::**ApproximateMannWhitneyUTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/mann_whitney.jl:225 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/mann_whitney.jl)
- pvalue(x::**ExactMannWhitneyUTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/mann_whitney.jl:136 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/mann_whitney.jl)
- pvalue(apt::**HypothesisTests.PermutationTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/permutation.jl:43 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/permutation.jl)
- pvalue(dist::**Distribution{Univariate, Continuous}**, x::**Number**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/HypothesisTests.jl:72 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/HypothesisTests.jl)
- pvalue(dist::**Distribution{Univariate, Discrete}**, x::**Number**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/HypothesisTests.jl:83 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/HypothesisTests.jl)
- pvalue(x::**HypothesisTests.ZTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/z.jl:31 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/z.jl)
- pvalue(x::**KSampleADTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/anderson_darling.jl:241 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/anderson_darling.jl)
- pvalue(T::**HypothesisTests.HotellingT2TestTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/hotelling.jl:8

(file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/hotelling.jl)

- pvalue(x::**ExactSignedRankTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/wilcoxon.jl:135 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/wilcoxon.jl)
- pvalue(x::**ApproximateSignedRankTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/wilcoxon.jl:225 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/wilcoxon.jl)
- pvalue(x::**FisherTLinearAssociation**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/circular.jl:138 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/circular.jl)
- pvalue(x::**HypothesisTests.TTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/t.jl:31 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/t.jl)
- pvalue(x::**HypothesisTests.ExactKSTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kolmogorov_smirnov.jl:80 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kolmogorov_smirnov.jl)
- pvalue(x::**LjungBoxTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/box_test.jl:123 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/box_test.jl)
- pvalue(x::**BreuschGodfreyTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/breusch_godfrey.jl:79 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/breusch_godfrey.jl)
- pvalue(x::**RayleighTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/circular.jl:63 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/circular.jl)
- pvalue(B::**BartlettTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/bartlett.jl:53 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/bartlett.jl)
- pvalue(x::**DurbinWatsonTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/durbin_watson.jl:180 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/durbin_watson.jl)
- pvalue(x::**ADFTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/augmented_dickey_fuller.jl:220 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/augmented_dickey_fuller.j
- pvalue(x::**BoxPierceTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/box_test.jl:74 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/box_test.jl)

- pvalue(t::**OneSampleADTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/anderson_darling.jl:55 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/anderson_darling.jl)
- pvalue(x::**ApproximateTwoSampleKSTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kolmogorov_smirnov.jl:181 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kolmogorov_smirnov.jl)
- pvalue(x::**FisherExactTest**; *tail, method*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/fisher.jl:123 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/fisher.jl)
- pvalue(t::**WhiteTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/white.jl:120 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/white.jl)
- pvalue(x::**ApproximateOneSampleKSTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kolmogorov_smirnov.jl:126 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kolmogorov_smirnov.jl)
- pvalue(x::**BinomialTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/binomial.jl:67 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/binomial.jl)
- pvalue(x::**VarianceFTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/f.jl:66 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/f.jl)
- pvalue(x::**KruskalWallisTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kruskal_wallis.jl:95 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/kruskal_wallis.jl)
- pvalue(x::**JammalamadakaCircularCorrelation**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/circular.jl:221 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/circular.jl)
- pvalue(x::**SignTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/binomial.jl:215 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/binomial.jl)
- pvalue(x::**PowerDivergenceTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/power_divergence.jl:44 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/power_divergence.jl)
- pvalue(x::**JarqueBeraTest**) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/jarque_bera.jl:87 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/jarque_bera.jl)

In [166]:
```
@which pvalue(test_result)
```

Out[166]: pvalue(x::**HypothesisTests.TTest**; *tail*) in HypothesisTests at /Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/t.jl:31 (file:///Users/uqjnazar/.julia/packages/HypothesisTests/V7PST/src/t.jl)

In [167]:
```
xBar1, s1, n1 = mean(data1), std(data1), length(data1)
xBar2, s2, n2 = mean(data2), std(data2), length(data2)

v = ( s1^2/n1 + s2^2/n2 )^2 / ( (s1^2/n1)^2/(n1-1) + (s2^2/n2)^2/(n2-1)
testStatistic = ( xBar1-xBar2 - δ₀ )  / sqrt( s1^2/n1 + s2^2/n2 )
pVal = 2*ccdf(TDist(v), abs(testStatistic))

println("Manually calculated degrees of freedom, v: ", v)
println("Manually calculated test statistic: ", testStatistic)
println("Manually calculated p-value: ", pVal, "\n")
```

```
Manually calculated degrees of freedom, v: 31.82453144280283
Manually calculated test statistic: 4.483705005611673
Manually calculated p-value: 8.936189820683007e-5
```

In [168]:
```
data = CSV.File("./data/fertilizer.csv") |> DataFrame

control = data.Control
fertilizer = data.FertilizerX

subGroups = collect(combinations([control;fertilizer],10))

meanFert = mean(fertilizer)
pVal = sum([mean(i) >= meanFert for i in subGroups])/length(subGroups)
println("p-value = ", pVal)
```

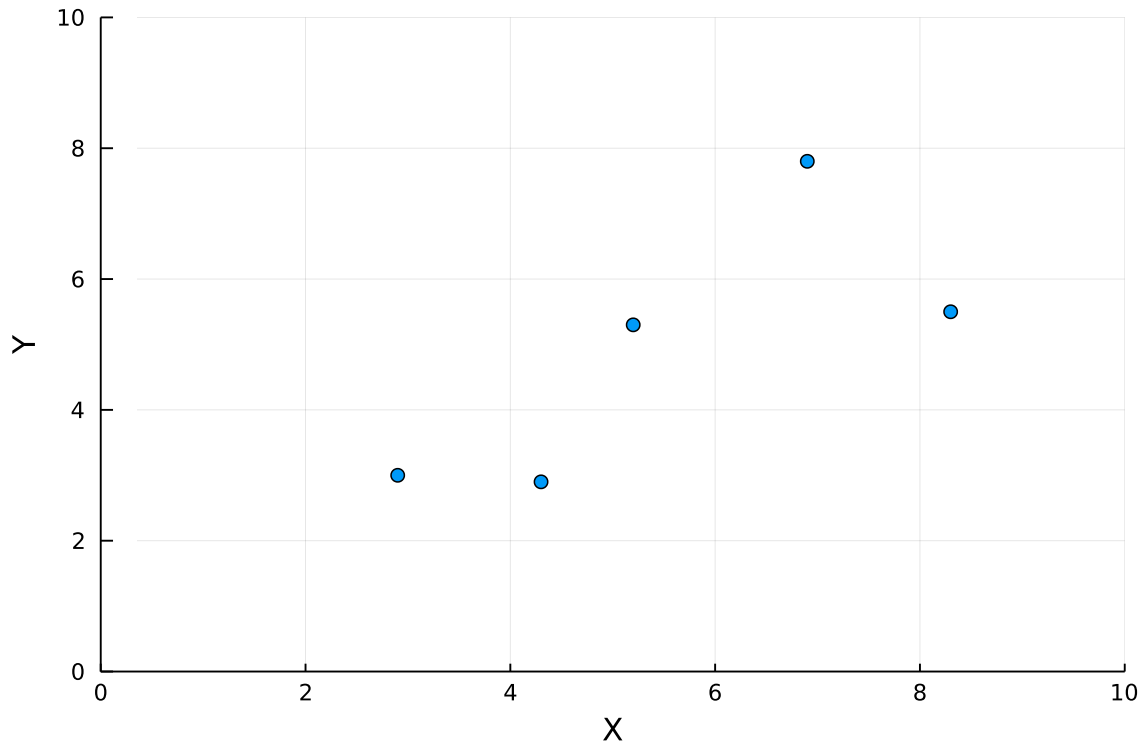
```
p-value = 0.023972157873086666
```

Linear models at our core.

home

In [169]:
```
data = CSV.File("./data/L1L2data.csv") |> DataFrame
xVals, yVals = data[:,1], data[:,2]
scatter(xVals,yVals,label=false,xlim=(0,10),ylim=(0,10),xlabel="X", ylab
```

Out[169]:



In [170]:
```
n = length(xVals)
A = [ones(n) xVals] #The "design matrix"
```

Out[170]:
```
5×2 Matrix{Float64}:
 1.0  2.9
 1.0  4.3
 1.0  5.2
 1.0  6.9
 1.0  8.3
```

In [171]:

```
# Approach A
xBar, yBar = mean(xVals),mean(yVals)
sXX, sXY = ones(n)'*(xVals.-xBar).^2 , dot(xVals.-xBar,yVals.-yBar)
b1A = sXY/sXX
b0A = yBar - b1A*xBar

# Approach B
b1B = cor(xVals,yVals)*(std(yVals)/std(xVals))
b0B = yBar - b1B*xBar

# Approach C
b0C, b1C = A'A \ A'yVals

# Approach D
Adag = inv(A'*A)*A'
b0D, b1D = Adag*yVals

# Approach E
b0E, b1E = pinv(A)*yVals

# Approach F
b0F, b1F = A\yVals

# Approach G
F = qr(A)
Q, R = F.Q, F.R
b0G, b1G = (inv(R)*Q')*yVals

# Approach H
F = svd(A)
V, Sp, Us = F.V, Diagonal(1 ./ F.S), F.U'
b0H, b1H = (V*Sp*Us)*yVals

# Approach I
eta, eps = 0.002, 10^-6.
b, bPrev = [0,0], [1,1]
while norm(bPrev-b) >= eps
    global bPrev = b
    global b = b - eta*2*A'*(A*b - yVals)
end
b0I, b1I = b[1], b[2]

# Approach J
```

```
modelJ = lm(@formula(Y ~ X), data)
b0J, b1J = coef(modelJ)

# Approach K
modelK = glm(@formula(Y ~ X), data, Normal())
b0K, b1K = coef(modelK)

println(round.([b0A,b0B,b0C,b0D,b0E,b0F,b0G,b0H,b0I,b0J,b0K],digits=3))
println(round.([b1A,b1B,b1C,b1D,b1E,b1F,b1G,b1H,b1I,b1J,b1K],digits=3))
```

```
[0.945, 0.945, 0.945, 0.945, 0.945, 0.945, 0.945, 0.945, 0.944, 0.945, 
[0.716, 0.716, 0.716, 0.716, 0.716, 0.716, 0.716, 0.716, 0.717, 0.716, 
```

In [172]:
```
data = CSV.File("./data/weightHeight.csv") |> DataFrame

lm1 = lm(@formula(Height ~ Weight), data)
lm2 = fit(LinearModel,@formula(Height ~ Weight), data)

glm1 = glm(@formula(Height ~ Weight), data, Normal(), IdentityLink())
glm2 = fit(GeneralizedLinearModel,@formula(Height ~ Weight), data, Norma

println("***Output of LM Model:")
println(lm1)
println("\n***Output of GLM Model:")
println(glm1)

pred(x) = coef(lm1)'*[1, x]

println("\n***Individual methods applied to model output:")
println("Deviance: ",deviance(lm1))
println("Standard error: ",stderror(lm1))
println("Degrees of freedom: ",dof_residual(lm1))
println("Covariance matrix: ",vcov(lm1))

yVals = data.Height
SStotal = sum((yVals .- mean(yVals)).^2)

println("R squared (calculated in two ways):",r2(lm1),
        ",\t", 1 - deviance(lm1)/SStotal)

println("MSE (calculated in two ways: ",deviance(lm1)/dof_residual(lm1),
        ",\t",sum((pred.(data.Weight) - data.Height).^2)/(size(data)[1]

xlims = [minimum(data.Weight), maximum(data.Weight)]
scatter(data.Weight, data.Height, c=:blue, msw=0)
plot!(xlims, pred.(xlims),
    c=:red, xlims=(xlims),
    xlabel="Weight (kg)", ylabel="Height (cm)", legend=:none)
```

```
***Output of LM Model:
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}

Height ~ 1 + Weight

Coefficients:
─────────────────────────────────────────────────────────────────────────
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper
(Intercept)	135.793	1.95553	69.44	<1e-99	131.937	139.6
Weight	0.532299	0.0293556	18.13	<1e-43	0.474409	0.5

***Output of GLM Model:
StatsModels.TableRegressionModel{GeneralizedLinearModel{GLM.GlmResp{Vec

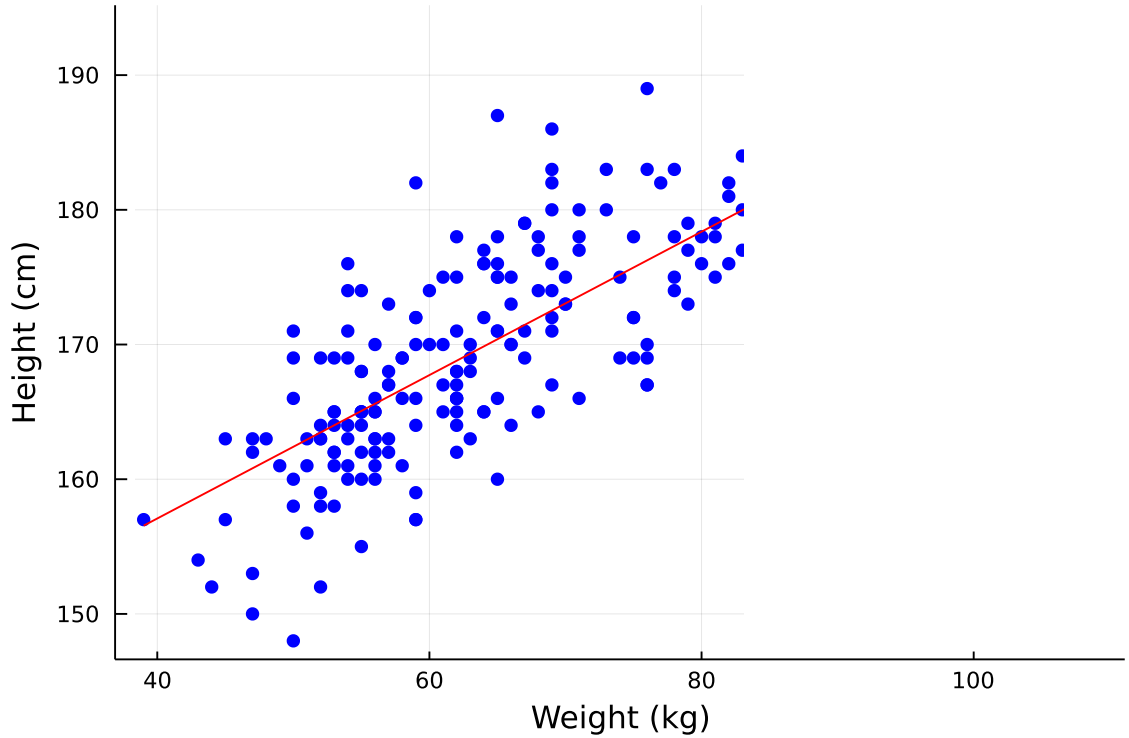
Height ~ 1 + Weight

Coefficients:

	Coef.	Std. Error	z	Pr(>|z|)	Lower 95%	Upper
(Intercept)	135.793	1.95553	69.44	<1e-99	131.96	139.6
Weight	0.532299	0.0293556	18.13	<1e-72	0.474763	0.5

***Individual methods applied to model output:
Deviance: 5854.05714276554
Standard error: [1.9555309971174046, 0.029355593998568793]
Degrees of freedom: 198.0
Covariance matrix: [3.824101480686991 -0.05628525996876029; -0.05628525
R squared (calculated in two ways):0.624144340084702, 0.624144340084
MSE (calculated in two ways: 29.56594516548253, 29.565945165482532

Out[172]:



Multiple variables

In [173]:
```
df = RDatasets.dataset("MASS", "cpus")
df.Freq = map( x->10^9/x , df.CycT)

model = lm(@formula(Perf ~ MMax + Cach + ChMax + Freq), df)
pred(x) = round(coef(model)'*vcat(1,x),digits = 3)

println("n = ", size(df)[1])
println("(Avg,Std) of observed performance: ", (mean(df.Perf),std(df.Per
println(model)
println("Estimated performance for computer A: ", pred([32000, 32, 32, 4
println("Estimated performance for computer B: ", pred([32000, 16, 32, 6
```

```
n = 209
(Avg,Std) of observed performance: (105.61722488038278, 160.83058719907
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}

Perf ~ 1 + MMax + Cach + ChMax + Freq

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	
(Intercept)	-46.5763	7.62382	-6.11	<1e-08	-61.6078	
MMax	0.00841457	0.000639214	13.16	<1e-28	0.00715426	
Cach	0.872508	0.152825	5.71	<1e-07	0.571189	
ChMax	0.96736	0.234847	4.12	<1e-04	0.504321	
Freq	9.74951e-7	5.5502e-7	1.76	0.0805	-1.1936e-7	

```
Estimated performance for computer A: 320.564
Estimated performance for computer B: 326.103
```

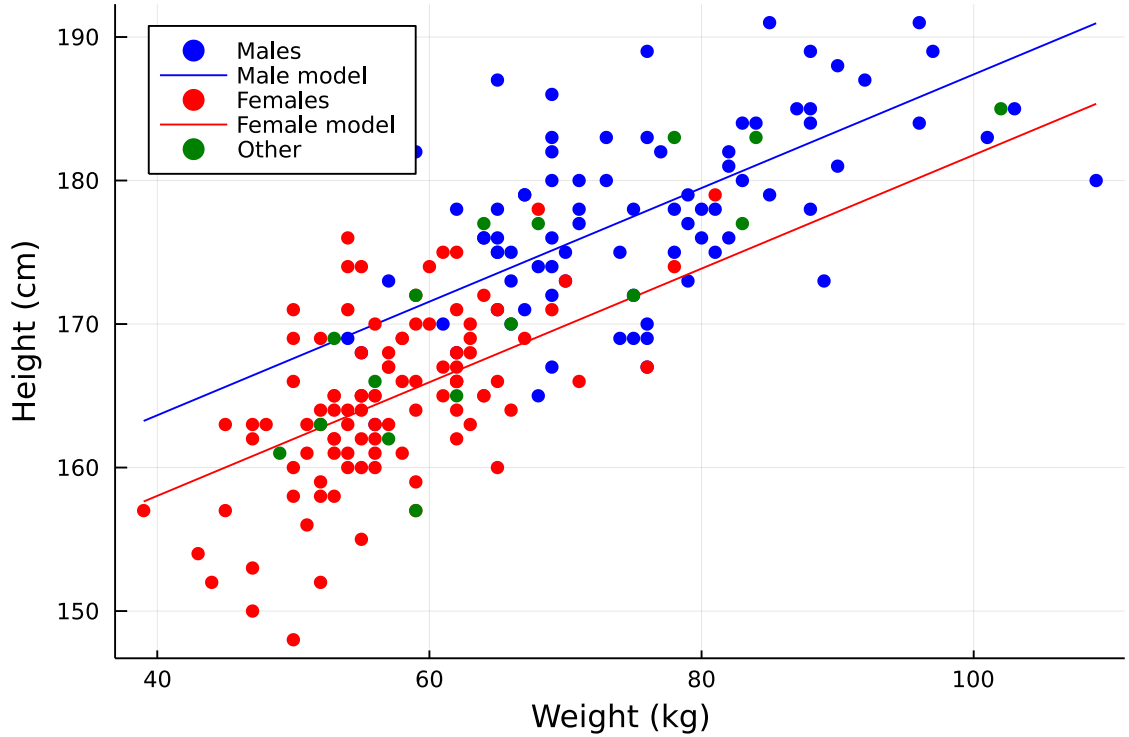
Categorical variables

In [174]:
```
df = CSV.File("./data/weightHeight.csv") |> DataFrame
n = size(df)[1]
df[shuffle(1:n),:] = df
df[[10,40,60,130,140,175,190,200],:Sex] .= "O1"
df[[9,44,63,132,138,172,192,199],:Sex] .= "O2"

model = lm(@formula(Height ~ Weight + Sex), df,
        contrasts=Dict(:Sex=>DummyCoding(base="F",levels=["M","O1","O2",
b0, b1, b2, b3, b4  = coef(model)
pred(weight,sex) = b0+b1*weight+b2*(sex=="M")+b3*(sex=="O1")+b3*(sex=="O
println(model)

males = df[df.Sex .== "M",:]
females = df[df.Sex .== "F",:]
other = df[(df.Sex .!= "M") .& (df.Sex .!= "F"),:]

xlim = [minimum(df.Weight), maximum(df.Weight)]
scatter(males.Weight, males.Height, c=:blue, msw=0, label="Males")
plot!(xlim, pred.(xlim,"M"), c=:blue, label="Male model")

scatter!(females.Weight, females.Height, c=:red, msw=0, label="Females")
plot!(xlim, pred.(xlim,"F"),
    c=:red, label="Female model", xlims=(xlim),
    xlabel="Weight (kg)", ylabel="Height (cm)", legend=:topleft)

scatter!(other.Weight, other.Height, c=:green, msw=0, label="Other")
```

```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}

Height ~ 1 + Weight + Sex

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper
(Intercept)	142.194	2.17705	65.32	<1e-99	137.901	146.48
Weight	0.395869	0.037162	10.65	<1e-20	0.322578	0.46
Sex: M	5.61479	1.01905	5.51	<1e-06	3.60501	7.62
Sex: O1	3.12116	1.88874	1.65	0.1000	-0.603814	6.84
Sex: O2	2.0663	1.92299	1.07	0.2839	-1.72624	5.85

Out[174]:

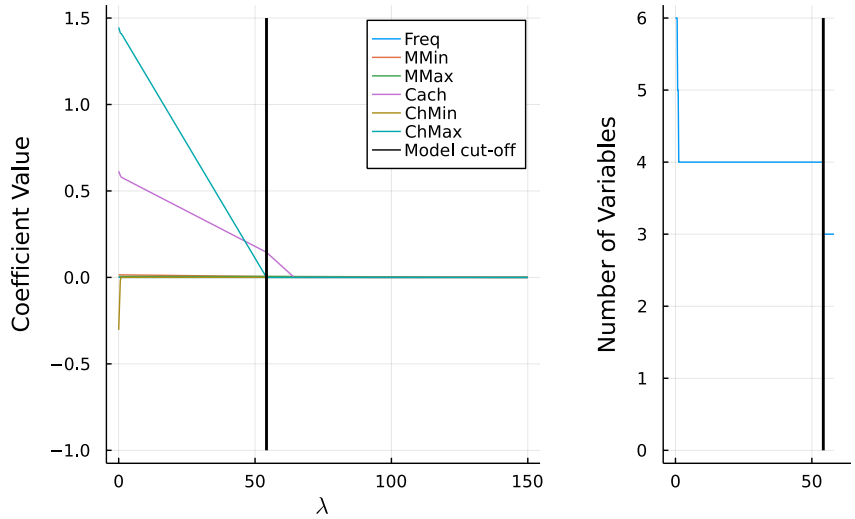


Variable selection with LASSO

In [177]:
```
df = RDatasets.dataset("MASS", "cpus")
df.Freq = map( x->10^9/x , df.CycT)
df = df[:, [:Perf, :Freq, :MMin, :MMax, :Cach, :ChMin, :ChMax]]
X = [df.Freq df.MMin df.MMax df.Cach df.ChMin df.ChMax]
Y = df.Perf

targetNumVars = 3

lambdaStep = 0.2
lamGrid = collect(0:lambdaStep:150)
lassoFit = fit(LassoPath,X, Y, λ = lamGrid);
dd = Array(lassoFit.coefs)'
nV = sum(dd .!= 0.0 ,dims=2)

goodLambda = lamGrid[findfirst((n)->n==targetNumVars,nV)]
newFit = fit(LassoPath,X, Y, λ = [goodLambda - lambdaStep, goodLambda])
println(newFit)
println("Coefficients: ", Array(newFit.coefs)'[2,:])

p1 = plot(lassoFit.λ, dd, label = ["Freq" "MMin" "MMax" "Cach" "ChMin" "
    ylabel = "Coefficient Value")
plot!([goodLambda,goodLambda],[-1,1.5],c=:black, lw=2, label = "Model cu

p2 = plot(lassoFit.λ,nV, ylabel = "Number of Variables",legend = false)
plot!([goodLambda,goodLambda],[0,6],c=:black, lw=2, label = "Model cut-o

plot(p1,p2,xlabel= L"\lambda", margin = 5mm, size = (800,400))
```

```
LassoPath (2) solutions for 7 predictors in 84 iterations):
──────────────────────────────
        λ   pct_dev  ncoefs
──────────────────────────────
[1]  54.0  0.673402       4
[2]  54.2  0.67203        3
──────────────────────────────
Coefficients: [0.0, 0.005294014227461394, 0.0056046927214597085, 0.1452
```

Out[177]:



Generalized linear models

In [178]:
```
df = RDatasets.dataset("MASS", "cpus")
n = size(df)[1]
df = df[shuffle(1:n),:]

pTest = 0.2
lastTindex = Int(floor(n*(1-pTest)))
numTest = n - lastTindex

train = df[1:lastTindex,:]
test = df[lastTindex+1:n,:]

form = @formula(Perf~CycT+MMin+MMax+Cach+ChMin+ChMax)
model1 = glm(form, train, Normal(),  IdentityLink())
model2 = glm(form, train, Poisson(), LogLink())
model3 = glm(form, train, Gamma(),  InverseLink())

invIdenityLink(x) = x
invLogLink(x) = exp(x)
invInverseLink(x) = 1/x

A = [ones(numTest) test.CycT test.MMin test.MMax test.Cach test.ChMin te
pred1 = invIdenityLink.(A*coef(model1))
pred2 = invLogLink.(A*coef(model2))
pred3 = invInverseLink.(A*coef(model3))

actual = test.Perf
lossModel1 = norm(pred1 - actual)
lossModel2 = norm(pred2 - actual)
lossModel3 = norm(pred3 - actual)

println("Model 1: ", coef(model1))
println("Model 2: ", coef(model2))
println("Model 3: ", coef(model3))
println("\nLoss of models 1,2,3: ",(lossModel1 ,lossModel2, lossModel3))
```

```
Model 1: [-64.40562610624185, 0.05281609072369714, 0.014630060511348993
Model 2: [4.022167583738135, -0.0019725826574401167, 1.2202420632006302
Model 3: [0.007807965416699389, 7.008659356350106e-5, 1.053052307550004

Loss of models 1,2,3: (281.9822744152483, 231.55851725675828, 397.19740
```

Basic Machine learning.

home

Some resources:

- Docs for MLJ.jl (https://alan-turing-institute.github.io/MLJ.jl/dev/) - not used here.
- Docs for Scikitlearn.jl (https://github.com/cstjean/ScikitLearn.jl) - not used here.
- Docs for Flux.jl (https://fluxml.ai/Flux.jl/stable/)
- The Mathematical Engineering of Deep Learning (https://deeplearningmath.org/amsi-summer-school-course-2021.html)
- ...

Applying off the shelf deep neural networks (see https://github.com/FluxML/Metalhead.jl (https://github.com/FluxML/Metalhead.jl))

In [179]:
```
vgg = VGG19(); #A neural network model VGG19 (about 500Mb of parameters,
vgg.layers
```

Out[179]:
```
Chain(
  Conv((3, 3), 3 => 64, relu, pad=1),   # 1_792 parameters
  Conv((3, 3), 64 => 64, relu, pad=1),  # 36_928 parameters
  MaxPool((2, 2)),
  Conv((3, 3), 64 => 128, relu, pad=1),  # 73_856 parameters
  Conv((3, 3), 128 => 128, relu, pad=1),  # 147_584 parameters
  MaxPool((2, 2)),
  Conv((3, 3), 128 => 256, relu, pad=1),  # 295_168 parameters
  Conv((3, 3), 256 => 256, relu, pad=1),  # 590_080 parameters
  Conv((3, 3), 256 => 256, relu, pad=1),  # 590_080 parameters
  Conv((3, 3), 256 => 256, relu, pad=1),  # 590_080 parameters
  MaxPool((2, 2)),
  Conv((3, 3), 256 => 512, relu, pad=1),  # 1_180_160 parameters
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  MaxPool((2, 2)),
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  Conv((3, 3), 512 => 512, relu, pad=1),  # 2_359_808 parameters
  MaxPool((2, 2)),
  Metalhead.var"#44#45"(),
  Dense(25088, 4096, relu),             # 102_764_544 parameters
  Dropout(0.5),
  Dense(4096, 4096, relu),              # 16_781_312 parameters
  Dropout(0.5),
  Dense(4096, 1000),                    # 4_097_000 parameters
  NNlib.softmax,
)                   # Total: 38 arrays, 143_667_240 parameters, 548.052
```

In [180]:
```
#download an arbitrary image and try to classify it
download("https://deeplearningmath.org/data/images/appleFruit.jpg","appl
img = load("appleFruit.jpg")
```

Out[180]:

In [181]:
```
@time begin
    classify(vgg,img) #vgg is the model
end
```

```
  0.348432 seconds (1.31 k allocations: 480.540 MiB)
```

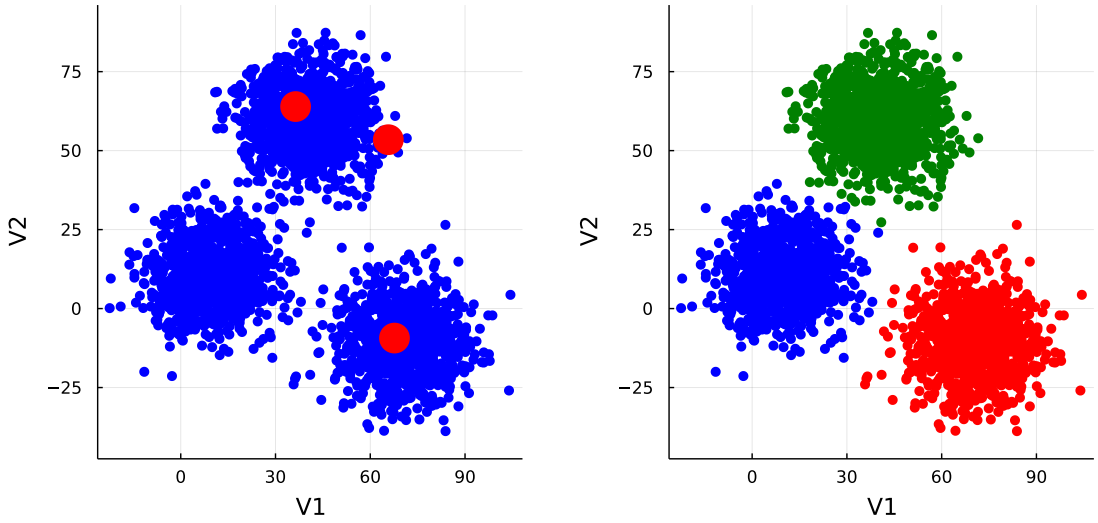
Out[181]:
```
"Granny Smith"
```

Clustering with *[Math Processing Error]*-means

In [182]:
```
fix_seed!()

K = 3
df = RDatasets.dataset("cluster", "xclara")
data = Matrix(df)'

seeds = initseeds(:rand, data, K)
xclaraKmeans = kmeans(data, K, init = seeds)

println("Number of clusters: ", nclusters(xclaraKmeans))
println("Counts of clusters: ", counts(xclaraKmeans))

df.Group  = assignments(xclaraKmeans)

p1 = scatter(df[:, :V1], df[:, :V2], c=:blue, msw=0)
     scatter!(df[seeds, :V1], df[seeds, :V2], markersize=12, c=:red, msv

p2 = scatter( df[df.Group .== 1, :V1], df[df.Group .== 1, :V2], c=:blue,
     scatter!( df[df.Group .== 2, :V1], df[df.Group .== 2, :V2], c=:red,
     scatter!( df[df.Group .== 3, :V1], df[df.Group .== 3, :V2], c=:gree

plot(p1,p2,legend=:none,ratio=:equal, size=(800,400), xlabel="V1", ylabe
```

```
Number of clusters: 3
Counts of clusters: [899, 952, 1149]
```

Out[182]:



The MNIST digits dataset

In [183]:
```
using MLDatasets
```

In [184]:
```
train_data = MLDatasets.MNIST.traindata(Float64)

imgs = train_data[1]
@show typeof(imgs)
@show size(imgs)

labels = train_data[2]
@show typeof(labels);
```

```
typeof(imgs) = Array{Float64, 3}
size(imgs) = (28, 28, 60000)
typeof(labels) = Vector{Int64}
```

In [185]:
```
test_data = MLDatasets.MNIST.testdata(Float64)
test_imgs = test_data[1]
test_labels = test_data[2]
@show size(test_imgs);
```

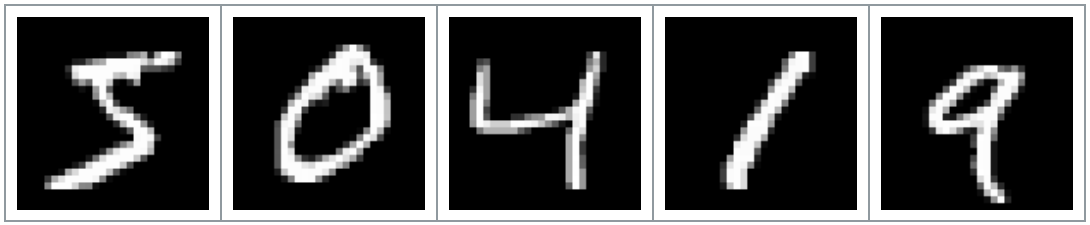
```
size(test_imgs) = (28, 28, 10000)
```

In [186]:
```
n_train, n_test = length(labels), length(test_labels)
```

Out[186]:
```
(60000, 10000)
```

In [187]:
```
[Gray.(train_data[1][:,:,k]') for k in 1:5]
```

Out[187]:

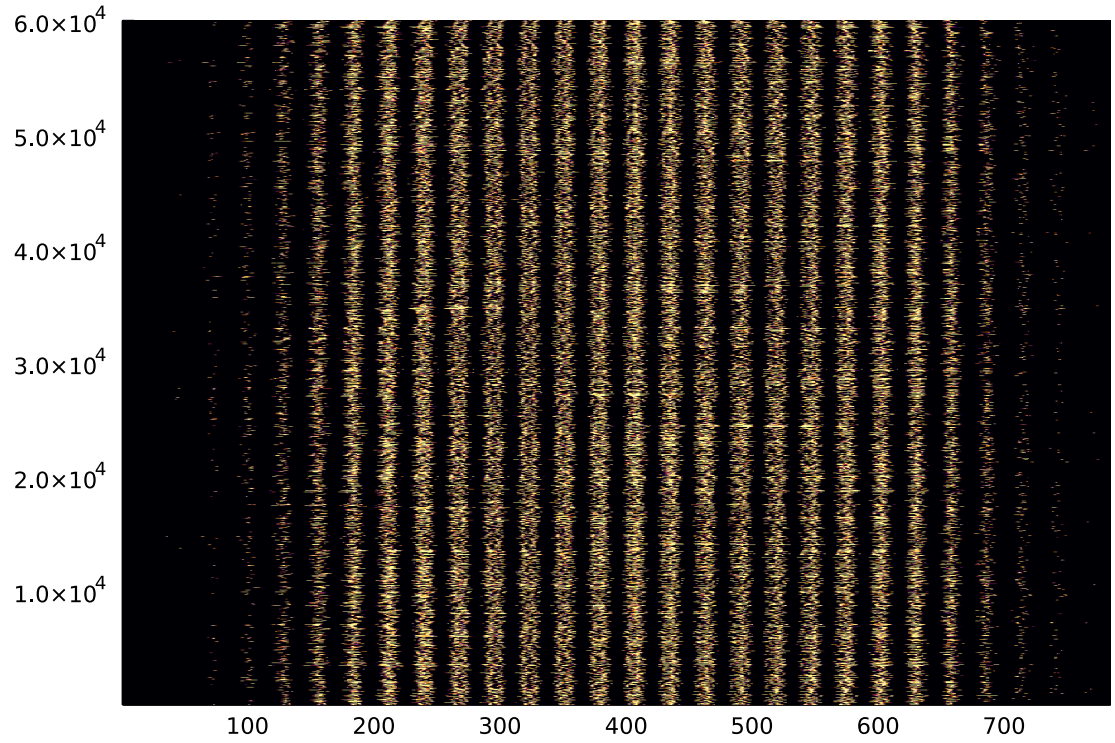


(a vector displayed as a row to save space)

In [188]:
```
X = vcat([vec(imgs[:,:,k])' for k in 1:last(size(imgs))]...)
@show size(X)
heatmap(X,legend=false)
```

```
size(X) = (60000, 784)
```

Out[188]:

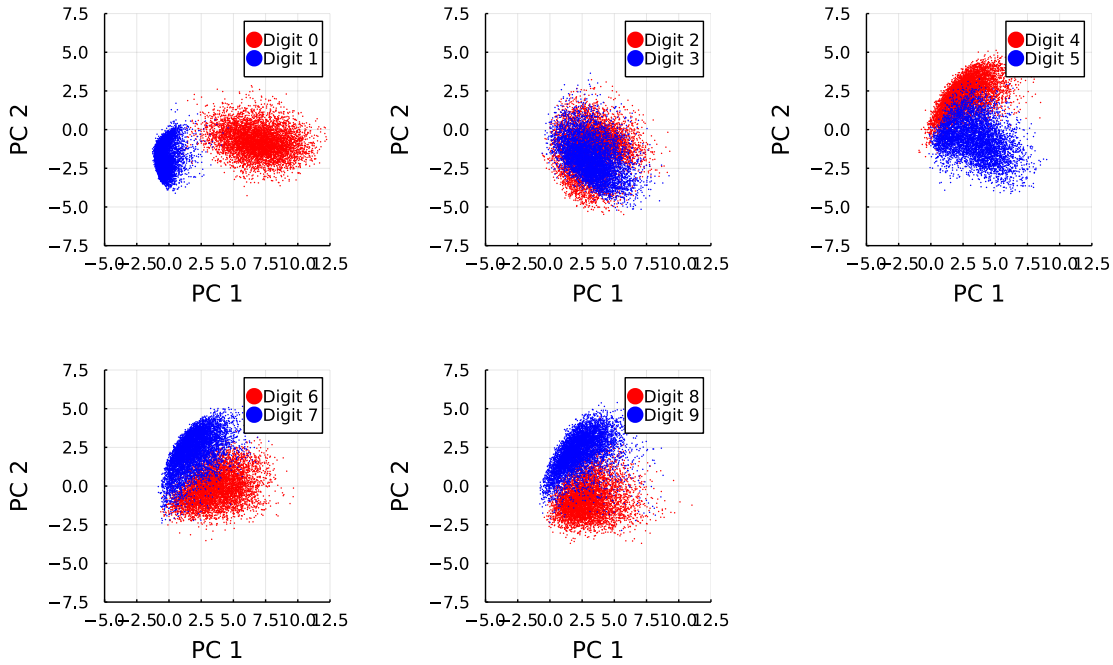


Principal component analysis (PCA)

In [189]:

```
pca = fit(PCA, X'; maxoutdim=2)
M = projection(pca)

args = (ms=0.8, msw=0, xlims=(-5,12.5), ylims=(-7.5,7.5),
            legend = :topright, xlabel="PC 1", ylabel="PC 2")

function compareDigits(dA,dB)
    xA = X[labels .== dA, :]'
    xB = X[labels .== dB, :]'
    zA, zB = M'*xA, M'*xB

    scatter(zA[1,:], zA[2,:], c=:red, label="Digit $(dA)"; args...)
    scatter!(zB[1,:], zB[2,:], c=:blue, label="Digit $(dB)"; args...)
end

plots = []
for k in 1:5
    push!(plots,compareDigits(2k-2,2k-1))
end
plot(plots...,size = (800, 500), margin = 5mm)
```

Out[189]:



A linear classifier

In [190]:
```
using Flux: onehotbatch

A = [ones(n_train) X]
Adag = pinv(A)
tfPM(x) = x ? +1 : -1
yDat(k) = tfPM.(onehotbatch(labels,0:9)'[:,k+1])
bets = [Adag*yDat(k) for k in 0:9]

linear_classify(square_image) = findmax([([1 ; vec(square_image)])'*bets
```

Out[190]:
```
linear_classify (generic function with 1 method)
```

In [191]:
```
predictions = [linear_classify(test_imgs[:,:,k]) for k in 1:n_test]
confusionMatrix = [sum((predictions .== i) .& (test_labels .== j)) for i
acc = sum(diag(confusionMatrix))/n_test

println("Accuracy: ", acc, "\nConfusion Matrix:")
show(stdout, "text/plain", confusionMatrix)
```

```
Accuracy: 0.8603
Confusion Matrix:
10×10 Matrix{Int64}:
 944     0   18    4    0   23   18    5   14   15
   0  1107   54   17   22   18   10   40   46   11
   1     2  813   23    6    3    9   16   11    2
   2     2   26  880    1   72    0    6   30   17
   2     3   15    5  881   24   22   26   27   80
   7     1    0   17    5  659   17    0   40    1
  14     5   42    9   10   23  875    1   15    1
   2     1   22   21    2   14    0  884   12   77
   7    14   37   22   11   39    7    0  759    4
   1     0    5   12   44   17    0   50   20  801
```

Training neural networks

In [192]:
```
train_range, validate_range = 1:5000, 5001:10000
batch_size = 1000

train_imgs = imgs[:,:,train_range]
train_labels = labels[train_range]
mb_idxs = Iterators.partition(1:length(train_range), batch_size)

validate_imgs = imgs[:,:,validate_range]
validate_labels = labels[validate_range];
```

In [193]:
```
function minibatch(x, y, index_range)
#     xBatch = Array{Float32}(undef, size(x[1])..., 1, length(indexRange
    x_batch = Array{Float32}(undef, 28,28, 1, length(index_range))
    for i in 1:length(index_range)
        x_batch[:, :, :, i] = Float32.(x[:,:,index_range[i]])
    end
    return (x_batch, onehotbatch(y[index_range], 0:9))
end

train_set = [minibatch(train_imgs, train_labels, bi) for bi in mb_idxs];
@show length(train_set)
validate_set = [minibatch(validate_imgs,validate_labels, 1:length(valida
@show length(validate_set);
```

```
length(train_set) = 5
length(validate_set) = 1
```

In [194]:
```
using Flux: onehotbatch, onecold, crossentropy, flatten, params
```

In [195]:
```
#A dense model
model1 = Chain(flatten,
               Dense(784, 200, relu),
               Dense(200, 100, tanh),
               Dense(100, 10, sigmoid),
               softmax)
```

Out[195]:
```
Chain(
  Flux.flatten,
  Dense(784, 200, relu),                # 157_000 parameters
  Dense(200, 100, tanh),                # 20_100 parameters
  Dense(100, 10, σ),                    # 1_010 parameters
  NNlib.softmax,
)                   # Total: 6 arrays, 178_110 parameters, 696.117 KiB.
```

In [196]:
```
#A convolutional model
model2 = Chain(
            Conv((5, 5), 1=>8, relu), MaxPool((2,2)),
            Conv((3, 3), 8=>16, relu), MaxPool((2,2)),
            flatten,
            Dense(400, 10),
            softmax)
```

Out[196]:
```
Chain(
  Conv((5, 5), 1 => 8, relu),           # 208 parameters
  MaxPool((2, 2)),
  Conv((3, 3), 8 => 16, relu),          # 1_168 parameters
  MaxPool((2, 2)),
  Flux.flatten,
  Dense(400, 10),                       # 4_010 parameters
  NNlib.softmax,
)                   # Total: 6 arrays, 5_386 parameters, 22.070 KiB.
```

In [197]:

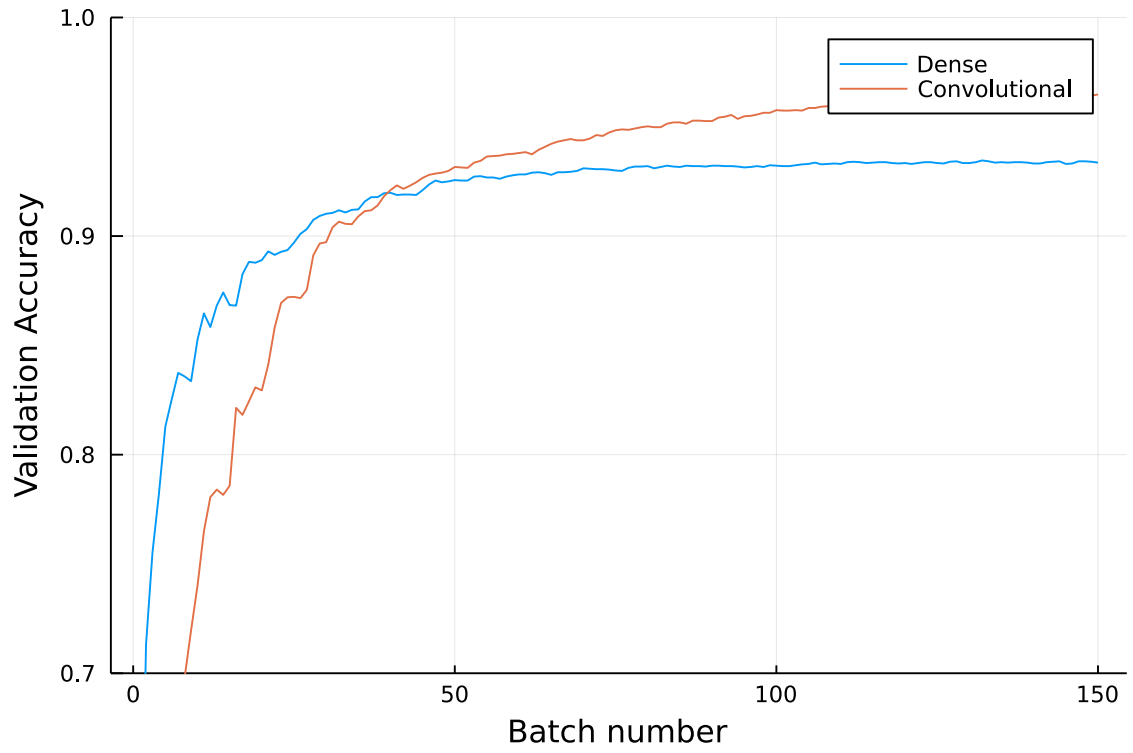
```
fix_seed!()

epochs = 30

η = 5e-3 #learning rate
opt1, opt2 = ADAM(η), ADAM(η)

accuracyPaths = [[],[]]

accuracy(x, y, model) = mean(onecold(model(x)) .== onecold(y))
loss(x, y, model) = crossentropy(model(x), y)

cbF1() = begin
            acc = accuracy(first(validate_set)..., model1)
            print("$acc, ")
            push!(accuracyPaths[1],acc)
        end

cbF2() = begin
            acc = accuracy(first(validate_set)..., model2)
            print("$acc, ")
            push!(accuracyPaths[2],acc)
        end

model1(train_set[1][1]); model2(train_set[1][1])

for ep in 1:epochs
    print("\nEpoch $ep \n  Model 1 accuracy: ")
    Flux.train!((x,y)->loss(x,y,model1), params(model1), train_set, opt1
    print("  Model 2 accuracy: ")
    Flux.train!((x,y)->loss(x,y,model2), params(model2), train_set, opt2
end

println("\n\nFinal Model1 (Dense) accuracy = ", accuracy(first(validate_
println("Final Model2 (Convolutional) accuracy = ", accuracy(first(valic

plot(accuracyPaths,label = ["Dense" "Convolutional"],
    ylim=(0.7,1.0), xlabel="Batch number", ylabel = "Validation Accuracy
```

```
Epoch 1
  Model 1 accuracy: 0.567, 0.7132, 0.7552, 0.782, 0.8128,   Model 2 accu
Epoch 2
  Model 1 accuracy: 0.8254, 0.8374, 0.8358, 0.8336, 0.8526,   Model 2 ac
```

```
Epoch 3
  Model 1 accuracy: 0.8646, 0.8584, 0.8682, 0.8742, 0.8684,   Model 2 ac
Epoch 4
  Model 1 accuracy: 0.8682, 0.8826, 0.8882, 0.8878, 0.889,   Model 2 acc
Epoch 5
  Model 1 accuracy: 0.893, 0.8914, 0.8928, 0.8936, 0.897,   Model 2 accu
Epoch 6
  Model 1 accuracy: 0.901, 0.9032, 0.9074, 0.9092, 0.9102,   Model 2 acc
Epoch 7
  Model 1 accuracy: 0.9106, 0.9118, 0.9108, 0.912, 0.9122,   Model 2 acc
Epoch 8
  Model 1 accuracy: 0.9158, 0.9178, 0.9178, 0.9196, 0.9198,   Model 2 ac
Epoch 9
  Model 1 accuracy: 0.9188, 0.919, 0.919, 0.9188, 0.921,   Model 2 accu
Epoch 10
  Model 1 accuracy: 0.9236, 0.9254, 0.9246, 0.925, 0.9256,   Model 2 acc
Epoch 11
  Model 1 accuracy: 0.9254, 0.9254, 0.9272, 0.9274, 0.9268,   Model 2 ac
Epoch 12
  Model 1 accuracy: 0.9268, 0.9262, 0.9272, 0.9278, 0.9282,   Model 2 ac
Epoch 13
  Model 1 accuracy: 0.9282, 0.929, 0.9292, 0.9288, 0.928,   Model 2 accu
Epoch 14
  Model 1 accuracy: 0.9292, 0.9292, 0.9294, 0.9298, 0.931,   Model 2 acc
Epoch 15
  Model 1 accuracy: 0.9308, 0.9306, 0.9306, 0.9304, 0.93,   Model 2 accu
Epoch 16
  Model 1 accuracy: 0.9298, 0.9312, 0.9318, 0.9318, 0.932,   Model 2 acc
Epoch 17
  Model 1 accuracy: 0.931, 0.9316, 0.9322, 0.9318, 0.9316,   Model 2 acc
Epoch 18
  Model 1 accuracy: 0.9322, 0.932, 0.932, 0.9318, 0.9322,   Model 2 accu
Epoch 19
  Model 1 accuracy: 0.9322, 0.932, 0.932, 0.9318, 0.9314,   Model 2 accu
Epoch 20
  Model 1 accuracy: 0.9316, 0.932, 0.9316, 0.9324, 0.9322,   Model 2 acc
Epoch 21
  Model 1 accuracy: 0.932, 0.932, 0.9324, 0.9328, 0.933,   Model 2 accu
Epoch 22
  Model 1 accuracy: 0.9336, 0.9328, 0.933, 0.9332, 0.933,   Model 2 accu
Epoch 23
  Model 1 accuracy: 0.9338, 0.934, 0.9338, 0.9334, 0.9336,   Model 2 acc
Epoch 24
  Model 1 accuracy: 0.9338, 0.9338, 0.9334, 0.9332, 0.9334,   Model 2 ac
```

```
Epoch 25
  Model 1 accuracy: 0.933, 0.9334, 0.9338, 0.9338, 0.9334,   Model 2 ac
Epoch 26
  Model 1 accuracy: 0.9332, 0.934, 0.9342, 0.9334, 0.9334,   Model 2 ac
Epoch 27
  Model 1 accuracy: 0.9338, 0.9346, 0.9342, 0.9336, 0.9338,   Model 2 a
Epoch 28
  Model 1 accuracy: 0.9336, 0.9338, 0.9338, 0.9336, 0.9332,   Model 2 a
Epoch 29
  Model 1 accuracy: 0.9332, 0.9338, 0.934, 0.9342, 0.933,   Model 2 acc
Epoch 30
  Model 1 accuracy: 0.9332, 0.9342, 0.9342, 0.934, 0.9336,   Model 2 ac

Final Model1 (Dense) accuracy = 0.9336
Final Model2 (Convolutional) accuracy = 0.9648
```

Out[197]:



The End

I hope you enjoyed the tutorial.

Many more of these types of examples + statistical background are at statisticswithjulia.org (https://statisticswithjulia.org/).