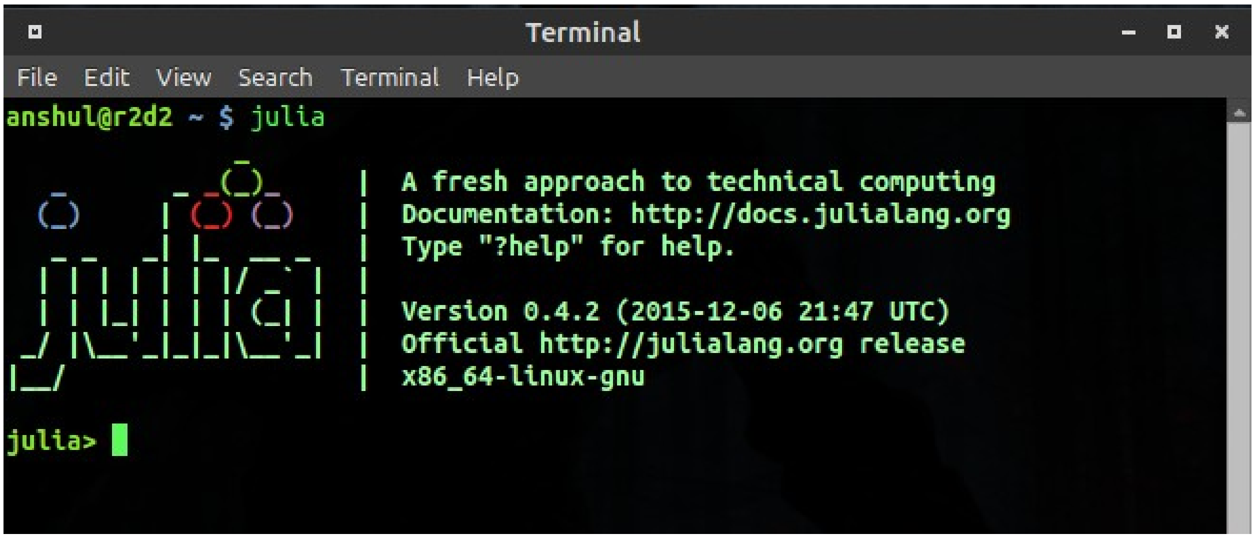
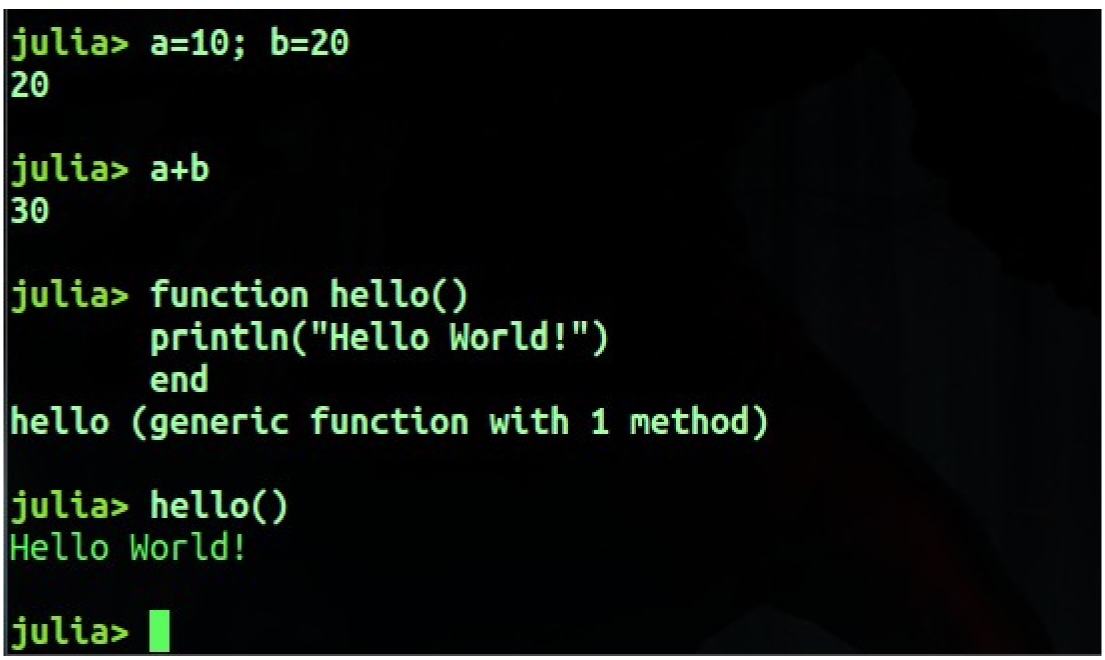
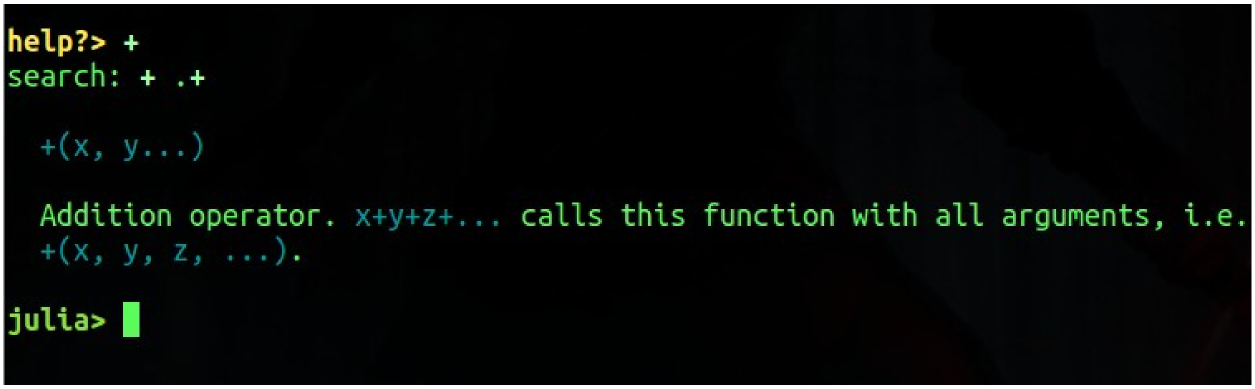
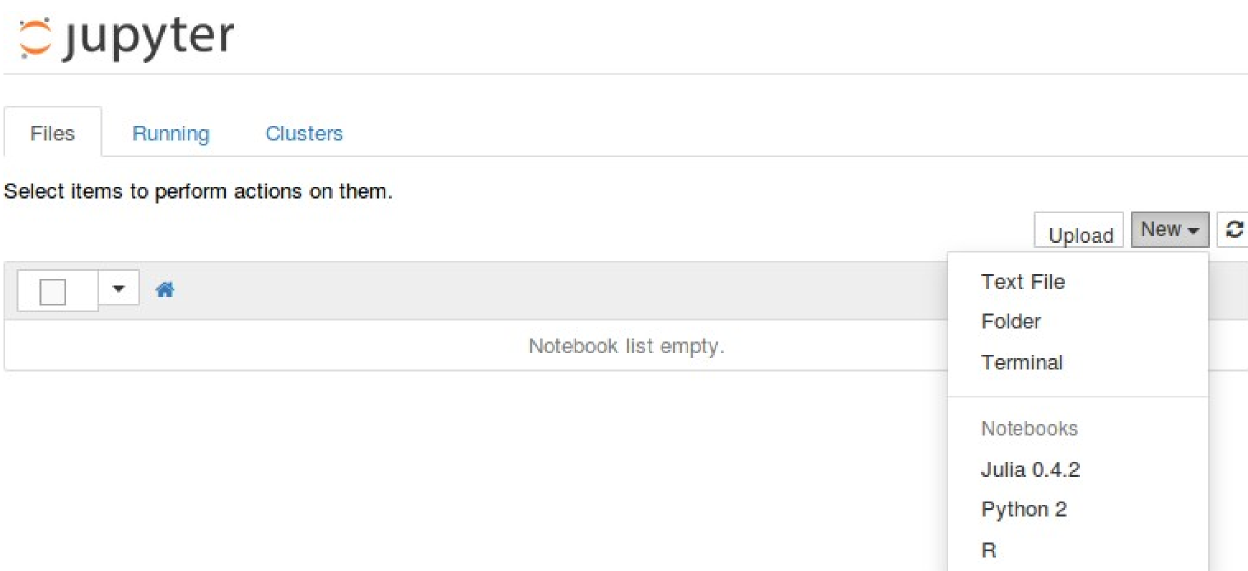
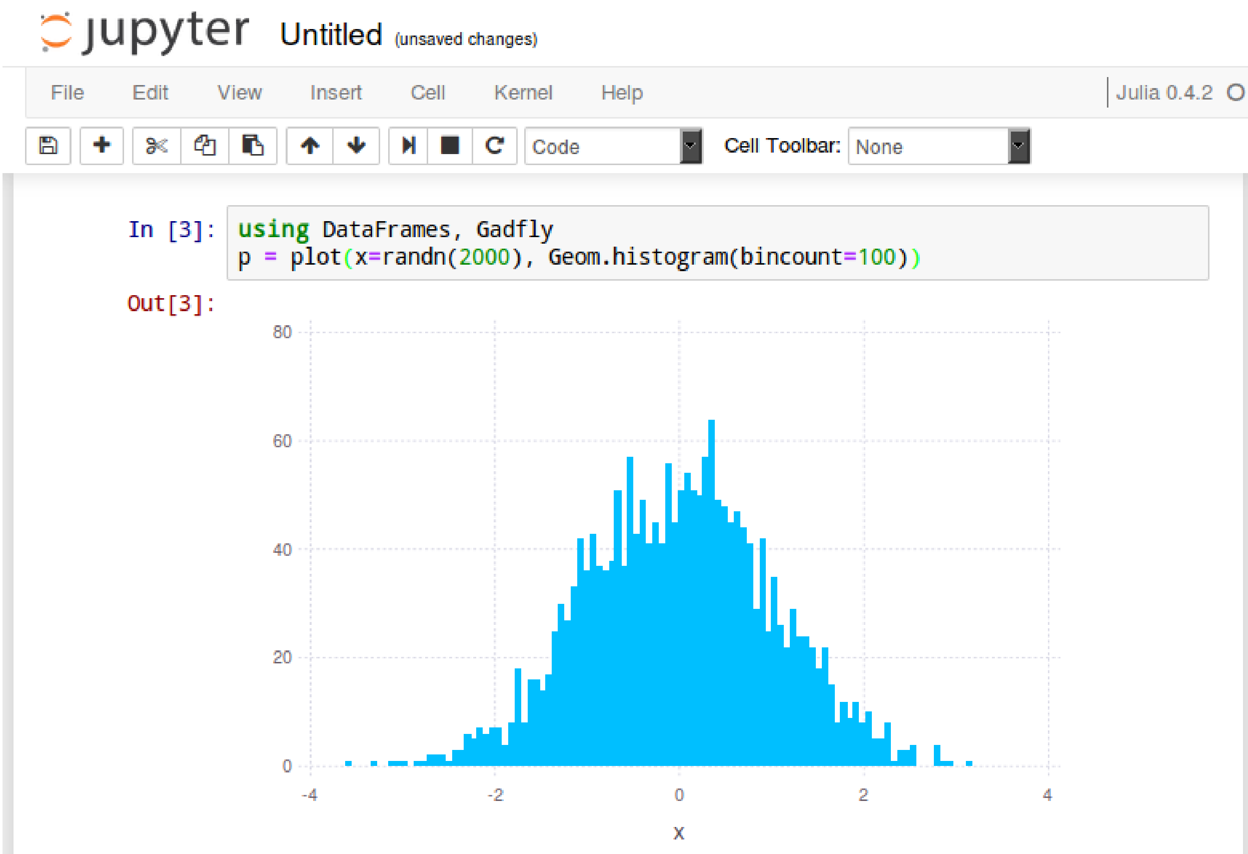
Chapter 1: The Groundwork—Julia's Environment






help?> +
search: + .+
+(x, y...)
Addition operator. x+y+z+... calls this function with all arguments, i.e.
+(x, y, z, ...).
julia>


jupyter
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Text File
Folder
Terminal
Notebooks
Julia 0.4.2
Python 2
R
Notebook list empty.




```
julia> Pkg.installed()
Dict{ASCIIString,VersionNumber} with 43 entries:
  "ImmutableArrays"    => v"0.0.11"
  "ZMQ"                => v"0.3.1"
  "ArrayViews"         => v"0.6.4"
  "DataStructures"     => v"0.4.0"
  "Compat"             => v"0.7.8"
  "Calculus"           => v"0.1.14"
  "GZip"               => v"0.2.18"
  "Measures"           => v"0.0.1"
  "StatsFuns"          => v"0.2.0"
  "DataFrames"         => v"0.6.10"
```

```
julia> nheads = @parallel (+) for i=1:100000000
         Int(rand(Bool))
       end
50001992
```

```
julia> M = {rand(500,500) for i=1:10}; pmap(svd, M)
```

```
julia> f(x::Float64, y::Float64)= x+y
f (generic function with 1 method)

julia> f(10.0,14.0)
24.0

julia> f(2,10.0)
ERROR: MethodError: `f` has no method matching f(::Int64, ::Float64)
Closest candidates are:
  f(::Float64, ::Float64)
```

```
julia> f(x::Float64, y::Float64)= x+y
f (generic function with 1 method)

julia> f(x::Number, y::Number)=2x+2y
f (generic function with 2 methods)

julia> f(24.0,4.0)
28.0

julia> f(10,11)
42
```

Chapter 2: Data Munging

```
julia> df
2x3 DataFrames.DataFrame
| Row | Name               | Count | OS       |
|-----|--------------------|-------|----------|
| 1   | "Ajava Rhodiumhi"  | 14.04 | "Ubuntu" |
| 2   | "Las Hushjoin"     | 17.3  | "Mint"   |
```

```
julia> head(df2)
6x2 DataFrames.DataFrame
| Row | X | Y      |
|-----|---|--------|
| 1   | 1 | "Head" |
| 2   | 2 | "Tail" |
| 3   | 3 | "Head" |
| 4   | 4 | "Head" |
| 5   | 5 | "Tail" |
| 6   | 6 | "Head" |
```

```
julia> head(iris_dataset)
6x5 DataFrames.DataFrame
| Row | SepalLength | SepalWidth | PetalLength | PetalWidth |
|-----|-------------|------------|-------------|------------|
| 1   | 5.1         | 3.5        | 1.4         | 0.2        |
| 2   | 4.9         | 3.0        | 1.4         | 0.2        |
| 3   | 4.7         | 3.2        | 1.3         | 0.2        |
| 4   | 4.6         | 3.1        | 1.5         | 0.2        |
| 5   | 5.0         | 3.6        | 1.4         | 0.2        |
| 6   | 5.4         | 3.9        | 1.7         | 0.4        |

| Row | Species  |
|-----|----------|
| 1   | "setosa" |
| 2   | "setosa" |
| 3   | "setosa" |
| 4   | "setosa" |
| 5   | "setosa" |
| 6   | "setosa" |
```

```
head(DfTRoadSafety_Accidents_2015)
```

	_Accident_Index	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitu
1	201501BS70001	525130	180050	-0.198465	51.50
2	201501BS70002	526530	178560	-0.178838	51.49
3	201501BS70004	524610	181080	-0.20559	51.51
4	201501BS70005	524420	181080	-0.208327	51.51
5	201501BS70008	524630	179040	-0.206022	51.49
6	201501BS70009	525480	179530	-0.19361	51.50

```
head(full_DfTRoadSafety_2015)
```

	_Accident_Index	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitu
1	201501BS70001	525130	180050	-0.198465	51.50
2	201501BS70002	526530	178560	-0.178838	51.49
3	201501BS70004	524610	181080	-0.20559	51.51
4	201501BS70005	524420	181080	-0.208327	51.51
5	201501BS70008	524630	179040	-0.206022	51.49
6	201501BS70008	524630	179040	-0.206022	51.49

	ID	City	RandomValue1	RandomValue2
1	2	Amsterdam	0.45225250816056284	100
2	3	Amsterdam	0.45097306910048696	107
3	8	Amsterdam	0.5567617467537034	102
4	9	Amsterdam	0.29952715400087837	107
5	4	Delhi	0.9703172728242426	106
6	5	Delhi	0.7235992085381457	100
7	6	Delhi	0.9517456514707969	101
8	7	Delhi	0.32919783621458265	103
9	10	Delhi	0.5552632497124872	101
10	1	Hamburg	0.2965785054730816	110

```
julia> by(DfTRoadSafety_Accidents_2014, :Location_Northing_OSGR, size)
96296x2 DataFrames.DataFrame
| Row    | Location_Northing_OSGR | x1     |
|--------|------------------------|--------|
| 1      | 10304                  | (1,32) |
| 2      | 10620                  | (1,32) |
| 3      | 13264                  | (1,32) |
| 4      | 16554                  | (1,32) |
| 5      | 17181                  | (1,32) |
| 6      | 19800                  | (1,32) |
| 7      | 21245                  | (1,32) |
| 8      | 22410                  | (1,32) |
⋮
```

```
julia> iris_dataframe = dataset("datasets", "iris")
150x5 DataFrames.DataFrame
| Row | SepalLength | SepalWidth | PetalLength | PetalWidth | Species     |
|-----|-------------|------------|-------------|------------|-------------|
| 1   | 5.1         | 3.5        | 1.4         | 0.2        | "setosa"    |
| 2   | 4.9         | 3.0        | 1.4         | 0.2        | "setosa"    |
| 3   | 4.7         | 3.2        | 1.3         | 0.2        | "setosa"    |
| 4   | 4.6         | 3.1        | 1.5         | 0.2        | "setosa"    |
| 5   | 5.0         | 3.6        | 1.4         | 0.2        | "setosa"    |
| 6   | 5.4         | 3.9        | 1.7         | 0.4        | "setosa"    |
| 7   | 4.6         | 3.4        | 1.4         | 0.3        | "setosa"    |
| 8   | 5.0         | 3.4        | 1.5         | 0.2        | "setosa"    |
⋮
```

```
julia> iris_stackdf = stackdf(iris_dataframe)
600x3 DataFrames.DataFrame
| Row | variable    | value | Species     |
|-----|-------------|-------|-------------|
| 1   | SepalLength | 5.1   | "setosa"    |
| 2   | SepalLength | 4.9   | "setosa"    |
| 3   | SepalLength | 4.7   | "setosa"    |
| 4   | SepalLength | 4.6   | "setosa"    |
| 5   | SepalLength | 5.0   | "setosa"    |
| 6   | SepalLength | 5.4   | "setosa"    |
| 7   | SepalLength | 4.6   | "setosa"    |
| 8   | SepalLength | 5.0   | "setosa"    |
⋮
```

	variable	value	Species	id
1	SepalLength	5.1	setosa	1
2	SepalLength	4.9	setosa	2
3	SepalLength	4.7	setosa	3
4	SepalLength	4.6	setosa	4
5	SepalLength	5.0	setosa	5
6	SepalLength	5.4	setosa	6

```
| Row | variable      | value |
|-----|---------------|-------|.
| 1   | SepalLength | 5.1   |
| 2   | SepalLength | 4.9   |
| 3   | SepalLength | 4.7   |
```

```
julia> iris_dataframe = stack(iris, [:PetalLength, :PetalWidth], :Species)
300x3 DataFrames.DataFrame
| Row | variable      | value | Species      |
|-----|---------------|-------|--------------|
| 1   | PetalLength | 1.4   | "setosa"     |
| 2   | PetalLength | 1.4   | "setosa"     |
| 3   | PetalLength | 1.3   | "setosa"     |
| 4   | PetalLength | 1.5   | "setosa"     |
| 5   | PetalLength | 1.4   | "setosa"     |
| 6   | PetalLength | 1.7   | "setosa"     |
| 7   | PetalLength | 1.4   | "setosa"     |
| 8   | PetalLength | 1.5   | "setosa"     |
⋮
```

```
600x4 DataFrames.DataFrame
| Row | variable    | value | Species     | id  |
|-----|-------------|-------|-------------|-----|
| 1   | SepalLength | 5.1   | "setosa"    | 1   |
| 2   | SepalLength | 4.9   | "setosa"    | 2   |
| 3   | SepalLength | 4.7   | "setosa"    | 3   |
| 4   | SepalLength | 4.6   | "setosa"    | 4   |
| 5   | SepalLength | 5.0   | "setosa"    | 5   |
| 6   | SepalLength | 5.4   | "setosa"    | 6   |
| 7   | SepalLength | 4.6   | "setosa"    | 7   |
| 8   | SepalLength | 5.0   | "setosa"    | 8   |
⋮
| 592 | PetalWidth  | 2.3   | "virginica" | 142 |
| 593 | PetalWidth  | 1.9   | "virginica" | 143 |
```

```
julia> unstack(iris_melt, :id, :variable, :value)
150x5 DataFrames.DataFrame
| Row | variable | PetalLength | PetalWidth | SepalLength | SepalWidth |
|-----|----------|-------------|------------|-------------|------------|
| 1   | 1        | 1.4         | 0.2        | 5.1         | 3.5        |
| 2   | 2        | 1.4         | 0.2        | 4.9         | 3.0        |
| 3   | 3        | 1.3         | 0.2        | 4.7         | 3.2        |
| 4   | 4        | 1.5         | 0.2        | 4.6         | 3.1        |
| 5   | 5        | 1.4         | 0.2        | 5.0         | 3.6        |
| 6   | 6        | 1.7         | 0.4        | 5.4         | 3.9        |
| 7   | 7        | 1.4         | 0.3        | 4.6         | 3.4        |
| 8   | 8        | 1.5         | 0.2        | 5.0         | 3.4        |
⋮
```

```
julia> iris_stackdf = stackdf(iris_dataframe)
600x3 DataFrames.DataFrame
| Row | variable    | value | Species      |
|-----|-------------|-------|--------------|
| 1   | SepalLength | 5.1   | "setosa"     |
| 2   | SepalLength | 4.9   | "setosa"     |
| 3   | SepalLength | 4.7   | "setosa"     |
| 4   | SepalLength | 4.6   | "setosa"     |
| 5   | SepalLength | 5.0   | "setosa"     |
| 6   | SepalLength | 5.4   | "setosa"     |
| 7   | SepalLength | 4.6   | "setosa"     |
| 8   | SepalLength | 5.0   | "setosa"     |
⋮
```

```
julia> dump(stack(iris_dataframe))
DataFrames.DataFrame  600 observations of 3 variables
  variable: Array(Symbol,(600,)) [:SepalLength,:SepalLength,:Sep
alLength,:SepalLength,:SepalLength,:SepalLength,:SepalLength,:Se
palLength,:SepalLength,:SepalLength  …  :PetalWidth,:PetalWidth,
:PetalWidth,:PetalWidth,:PetalWidth,:PetalWidth,:PetalWidth,:Pet
alWidth,:PetalWidth,:PetalWidth]
  value: DataArrays.DataArray{Float64,1}(600) [5.1,4.9,4.7,4.6]
  Species: DataArrays.PooledDataArray{ASCIIString,UInt8,1}(600)
ASCIIString["setosa","setosa","setosa","setosa"]
```

```
julia> dump(stackdf(iris_dataframe))
DataFrames.DataFrame  600 observations of 3 variables
  variable: DataFrames.RepeatedVector{Symbol}
    parent: Array(Symbol,(4,)) [:SepalLength,:SepalWidth,:Peta
lLength,:PetalWidth]
    inner: Int64 150
    outer: Int64 1
  value: DataFrames.StackedVector
    components: Array(Any,(4,))
      1: DataArrays.DataArray{Float64,1}(150) [5.1,4.9,4.7,4.6
]
      2: DataArrays.DataArray{Float64,1}(150) [3.5,3.0,3.2,3.1
]
      3: DataArrays.DataArray{Float64,1}(150) [1.4,1.4,1.3,1.5
]
      4: DataArrays.DataArray{Float64,1}(150) [0.2,0.2,0.2,0.2
]
  Species: DataFrames.RepeatedVector{ASCIIString}
    parent: DataArrays.PooledDataArray{ASCIIString,UInt8,1}(15
0) ASCIIString["setosa","setosa","setosa","setosa"]
    inner: Int64 1
    outer: Int64 4

julia> iris_mean_stack = by(iris_stack, [:variable, :Species],
 df -> DataFrame(iris_mean = mean(df[:value])))
12x3 DataFrames.DataFrame
| Row | variable    | Species      | iris_mean |
|-----|-------------|--------------|-----------|
| 1   | PetalLength | "setosa"     | 1.462     |
| 2   | PetalLength | "versicolor" | 4.26      |
| 3   | PetalLength | "virginica"  | 5.552     |
| 4   | PetalWidth  | "setosa"     | 0.246     |
| 5   | PetalWidth  | "versicolor" | 1.326     |
| 6   | PetalWidth  | "virginica"  | 2.026     |
| 7   | SepalLength | "setosa"     | 5.006     |
| 8   | SepalLength | "versicolor" | 5.936     |
| 9   | SepalLength | "virginica"  | 6.588     |
| 10  | SepalWidth  | "setosa"     | 3.428     |
| 11  | SepalWidth  | "versicolor" | 2.77      |
| 12  | SepalWidth  | "virginica"  | 2.974     |
```

```
julia> unstack(iris_mean_stack, :Species, :iris_mean)
4x4 DataFrames.DataFrame
| Row | variable    | setosa | versicolor | virginica |
|-----|-------------|--------|------------|-----------|
| 1   | PetalLength | 1.462  | 4.26       | 5.552     |
| 2   | PetalWidth  | 0.246  | 1.326      | 2.026     |
| 3   | SepalLength | 5.006  | 5.936      | 6.588     |
| 4   | SepalWidth  | 3.428  | 2.77       | 2.974     |

julia> sort!(iris_dataframe)
150x5 DataFrames.DataFrame
| Row | SepalLength | SepalWidth | PetalLength | PetalWidth |
|-----|-------------|------------|-------------|------------|
| 1   | 4.3         | 3.0        | 1.1         | 0.1        |
| 2   | 4.4         | 2.9        | 1.4         | 0.2        |
| 3   | 4.4         | 3.0        | 1.3         | 0.2        |
| 4   | 4.4         | 3.2        | 1.3         | 0.2        |
| 5   | 4.5         | 2.3        | 1.3         | 0.3        |
| 6   | 4.6         | 3.1        | 1.5         | 0.2        |
| 7   | 4.6         | 3.2        | 1.4         | 0.2        |
| 8   | 4.6         | 3.4        | 1.4         | 0.3        |
⋮

julia> sort!(iris_dataframe, cols = [order(:Species, by = uppe
rcase), order(:PetalLength, rev = true)])
150x5 DataFrames.DataFrame
| Row | SepalLength | SepalWidth | PetalLength | PetalWidth |
|-----|-------------|------------|-------------|------------|
| 1   | 4.8         | 3.4        | 1.9         | 0.2        |
| 2   | 5.1         | 3.8        | 1.9         | 0.4        |
| 3   | 5.1         | 3.3        | 1.7         | 0.5        |
| 4   | 5.4         | 3.4        | 1.7         | 0.2        |
| 5   | 5.4         | 3.9        | 1.7         | 0.4        |
| 6   | 5.7         | 3.8        | 1.7         | 0.3        |
| 7   | 4.7         | 3.2        | 1.6         | 0.2        |
| 8   | 4.8         | 3.1        | 1.6         | 0.2        |
⋮
```

```
julia> random_dataframe = DataFrame(A = randn(5), B = randn(5), C
 = randn(5))
5x3 DataFrames.DataFrame
| Row | A         | B        | C         |
|-----|-----------|----------|-----------|
| 1   | 0.610386  | 0.39672  | 0.843678  |
| 2   | 0.386281  | 1.53446  | -0.199888 |
| 3   | -0.118111 | -1.17061 | -1.44164  |
| 4   | 0.203097  | 1.3115   | 1.03606   |
| 5   | -0.856892 | 1.68626  | 0.149367  |

julia> random_modelframe = ModelFrame(A ~ B + C, random_dataframe)
DataFrames.ModelFrame(5x3 DataFrames.DataFrame
| Row | A         | B         | C         |
|-----|-----------|-----------|-----------|
| 1   | 1.03875   | -0.698513 | 0.664952  |
| 2   | 0.500446  | 1.97565   | 0.43762   |
| 3   | 1.70717   | 0.424157  | -0.846524 |
| 4   | -0.869665 | 0.182574  | -0.703025 |
| 5   | 0.801253  | 0.311777  | 2.08523   |,DataFrames.Terms(Any[:B,
:C],Any[:A,:B,:C],3x3 Array{Int8,2}:
 1  0  0
 0  1  0
 0  0  1,[1,1,1],true,true),Bool[true,true,true,true,true])


julia> random_modelmatrix = ModelMatrix(ModelFrame(A ~ B + C, ran
dom_dataframe))
DataFrames.ModelMatrix{Float64}(5x3 Array{Float64,2}:
 1.0   0.39672   0.843678
 1.0   1.53446  -0.199888
 1.0  -1.17061  -1.44164
 1.0   1.3115    1.03606
 1.0   1.68626   0.149367,[0,1,2])
```

```
julia> datavector = @data(["A", "A", "A","B", "B", "B"])
6-element DataArrays.DataArray{ASCIIString,1}:
 "A"
 "A"
 "A"
 "B"
 "B"
 "B"
```

```
julia> pooleddatavector = @pdata(["A", "A", "A","B", "B", "B"])
6-element DataArrays.PooledDataArray{ASCIIString,UInt32,1}:
 "A"
 "A"
 "A"
 "B"
 "B"
 "B"
```

```
julia> levels(pooleddatavector)
2-element Array{ASCIIString,1}:
 "A"
 "B"
```

```
julia> dataframe_notpooled = DataFrame(A = [10, 10, 10, 20, 20, 2
0], B = ["X", "X", "X", "Y", "Y", "Y"])
6x2 DataFrames.DataFrame
| Row | A  | B   |
|-----|----|-----|
| 1   | 10 | "X" |
| 2   | 10 | "X" |
| 3   | 10 | "X" |
| 4   | 20 | "Y" |
| 5   | 20 | "Y" |
| 6   | 20 | "Y" |
```

```
julia> pooleddf = pool!(dataframe_notpooled, [:A, :B])
```

```
julia> allPosts = []
0-element Array{Any,1}

julia> for record in 1:counter
julia> for record in 1:counter
       url = dataReceived["data"]["children"][record]["data"]["url"]
       redditrecord_id  = dataReceived["data"]["children"][record]["data"]
["id"]
       redditrecord_title  = dataReceived["data"]["children"][record]["dat
a"]["title"]
       author  = dataReceived["data"]["children"][record]["data"]["author"
]
       created = dataReceived["data"]["children"][record]["data"]["created
"]
       push!(allPosts, (url, redditrecord_id, redditrecord_title, author,
created))
       end

julia> allPosts
26-element Array{Any,1}:
 ("http://juliacon.org/","3ztvre","SAVE THE DATE: JuliaCon 2016 - Boston, MA
","Mr_You",1.452170599e9)


 ("https://www.reddit.com/r/Julia/comments/41iz6o/native_plotting_function_i
n_julia/","41iz6o","Native plotting function in Julia","shivaramkrs",1.45315
3237e9)

julia> for post in allPosts
           println(post[3])
       end
SAVE THE DATE: JuliaCon 2016 - Boston, MA
Native plotting function in Julia
A Speed Comparison Of C, Julia, Python, Numba, and Cython on LU Factorizatio
n
Julia 0.4.3 released
Julia IDE work in Atom
RBM written from scratch in Julia and trained with persistent states -- 98%
on MNIST without fine-tuning
Looking for a couple people to test my Julia editor
Vetting of a package
How to initialize columns of a matrix with a function?
Gave these as Christmas presents this year
RStudio equivalent for Julia
Julia for robotics programming
PyData Amsterdam CFP, we'd love to have somebody talk about Julia!
Why is this loop in Julia slower than the Python equivalent? What am I doing
 wrong?
Deep neural network written from scratch in Julia
```

Chapter 3: Data Exploration

```
julia> exam = dataset("mlmRev", "Exam")
4059x10 DataFrames.DataFrame
| Row | School | NormExam  | SchGend | SchAvg   | VR        | Intake
|-----|--------|-----------|---------|----------|-----------|----------
| 1   | "1"    | 0.261324  | "mixed" | 0.166175 | "mid 50%" | "bottom 2
| 2   | "1"    | 0.134067  | "mixed" | 0.166175 | "mid 50%" | "mid 50%"
| 3   | "1"    | -1.72388  | "mixed" | 0.166175 | "mid 50%" | "top 25%"
| 4   | "1"    | 0.967586  | "mixed" | 0.166175 | "mid 50%" | "mid 50%"
| 5   | "1"    | 0.544341  | "mixed" | 0.166175 | "mid 50%" | "mid 50%"
| 6   | "1"    | 1.7349    | "mixed" | 0.166175 | "mid 50%" | "bottom 2
| 7   | "1"    | 1.03961   | "mixed" | 0.166175 | "mid 50%" | "top 25%"
| 8   | "1"    | -0.129085 | "mixed" | 0.166175 | "mid 50%" | "mid 50%"
| 9   | "1"    | -0.939378 | "mixed" | 0.166175 | "mid 50%" | "mid 50%"
⋮
```

```
julia> describe(exam)
School
Length  4059
Type    Pooled ASCIIString
NAs     0
NA%     0.0%
Unique  65

NormExam
Min       -3.666072
1st Qu.   -0.699505
Median    0.0043222
Mean      -0.00011380542005424873
3rd Qu.   0.6787592
Max       3.6660912
NAs       0
NA%       0.0%
```

```
julia> a = [123,4234,23423,1231231,1432432423,1341413413]
6-element Array{Int64,1}:
        123
       4234
      23423
    1231231
 1432432423
 1341413413

julia> geomean(a)
553833.3901002567
```

```
julia> harmmean(a)
713.4557870657444
```

```
julia> a = [123,4234,23423,1231231,1432432423,1341413413]
6-element Array{Int64,1}:
        123
       4234
      23423
    1231231
 1432432423
 1341413413

julia> trimmean(a,0.1)
2.685344848e8
```

```
julia> a
6-element Array{Int64,1}:
        123
       4234
      23423
    1231231
 1432432423
 1341413413

julia> wv = rand(6)
6-element Array{Float64,1}:
 0.79903
 0.131471
 0.951132
 0.248691
 0.631604
 0.186289

julia> mean(a, weights(wv))
3.917448913086356e8
```

$$\text{var}_x = \frac{\sum_{i=1}^{n} (x_i - x)^2}{n}$$

```
julia> a
6-element Array{Int64,1}:
        123
       4234
      23423
    1231231
 1432432423
 1341413413

julia> var(a)
5.1354392444543296e17
```

```
julia> a = [1 2;3 4;5 6;7 8;9 10]
5x2 Array{Int64,2}:
 1   2
 3   4
 5   6
 7   8
 9  10

julia> var(a, 2)
5x1 Array{Float64,2}:
 0.5
 0.5
 0.5
 0.5
 0.5
```

```
julia> std(a)
3.0276503540974917
```

```
julia> mean_and_var(a)
(5.5,9.166666666666666)
```

```
julia> mean_and_std(a)
(5.5,3.0276503540974917)
```

```
julia> a = [12,234,567,1234,535,335,19]
7-element Array{Int64,1}:
   12
  234
  567
 1234
  535
  335
   19

julia> skewness(a)
0.9763073577410081

julia> kurtosis(a)
0.04885930438714192
```

```
julia> moment(a,3)
5.80616226472303le7
```

```
julia> a
7-element Array{Int64,1}:
   12
  234
  567
 1234
  535
  335
   19

julia> span(a)
12:1234
```

```
julia> variation(a)
1.0051933013705867
```

```
julia> a = [12,23,45,68,99,72,61,39,21,71]
10-element Array{Int64,1}:
 12
 23
 45
 68
 99
 72
 61
 39
 21
 71

julia> mad(a)
27.428099999999997

julia> mad(a,5)
71.1648
```

```
julia> zscore(a)
10-element Array{Float64,1}:
 -1.4102
 -1.01347
 -0.220005
  0.609522
  1.72758
  0.753788
  0.357057
 -0.436403
 -1.0856
  0.717721
```

```
julia> using Distributions

julia> d = Dirichlet([1.0, 3.0, 5.0])
Distributions.Dirichlet(alpha=[1.0,3.0,5.0])
```

```
julia> arr=rand(d)
3-element Array{Float64,1}:
 0.190511
 0.80904
 0.000449442
```

```
julia> sum(arr)
1.0000000000000002

julia> entropy(arr)
0.4907814135561367
```

```
julia> entropy(arr,2)
0.7080479114979139
```

```
julia> a = rand(10)
10-element Array{Float64,1}:
 0.256684
 0.0760744
 0.959692
 0.933633
 0.170989
 0.371441
 0.123852
 0.959958
 0.552251
 0.999725
```

```
julia> quantile(a)
5-element Array{Float64,1}:
 0.0760744
 0.192413
 0.461846
 0.953178
 0.999725
```

```
julia> iqr(a)
0.7607643393430641
```

```
julia> percentile(a,0.5)
0.07822436710303723
```

```
julia> nquantile(a,2)
3-element Array{Float64,1}:
 0.0760744
 0.461846
 0.999725
```

```
julia> mode(a)
0.2566843440628257
```

```
julia> summarystats(a)
Summary Stats:
Mean:          0.540430
Minimum:       0.076074
1st Quartile:  0.192413
Median:        0.461846
3rd Quartile:  0.953178
Maximum:       0.999725
```

```
julia> a = rand(4)
4-element Array{Float64,1}:
 0.462513
 0.340506
 0.269411
 0.283305
```

```
julia> ordinalrank(a)
4-element Array{Int64,1}:
 4
 3
 1
 2
```

```
julia> a = rand([1:5],30)
30-element Array{Int64,1}:
 4
 1
 3
 1
 1
 4
```

```
julia> counts(a)
5-element Array{Int64,1}:
  7
  1
  5
 11
  6
```

```
julia> proportions(a,1:3)
3-element Array{Float64,1}:
 0.233333
 0.0333333
 0.166667
```

```
julia> countmap(a)
Dict{Int64,Int64} with 5 entries:
  4 => 11
  2 => 1
  3 => 5
  5 => 6
  1 => 7
```

```
julia> proportionmap(a)
Dict{Int64,Float64} with 5 entries:
  4 => 0.36666666666666664
  2 => 0.03333333333333333
  3 => 0.16666666666666666
  5 => 0.2
  1 => 0.23333333333333334
```

```
h = fit(Histogram, (rand(100),rand(100)),nbins=10)
```

```
StatsBase.Histogram{Int64,2,Tuple{FloatRange{Float64},FloatRange{Float64}}}
edges:
  0.0:0.1:1.0
  0.0:0.1:1.0
weights: 10x10 Array{Int64,2}:
 1  1  0  1  1  0  0  0  0  1
 1  0  2  0  3  1  0  1  0  0
 1  1  3  0  2  3  0  1  1  0
 1  1  2  1  3  1  3  0  0  2
 1  1  0  2  0  0  1  1  1  0
 1  1  2  1  1  1  1  2  2  1
 0  1  1  2  0  1  1  0  0  3
 0  1  2  0  3  0  2  1  0  3
 2  0  0  1  2  0  2  2  1  0
 1  0  1  3  1  1  0  2  1  0
closed: right
```

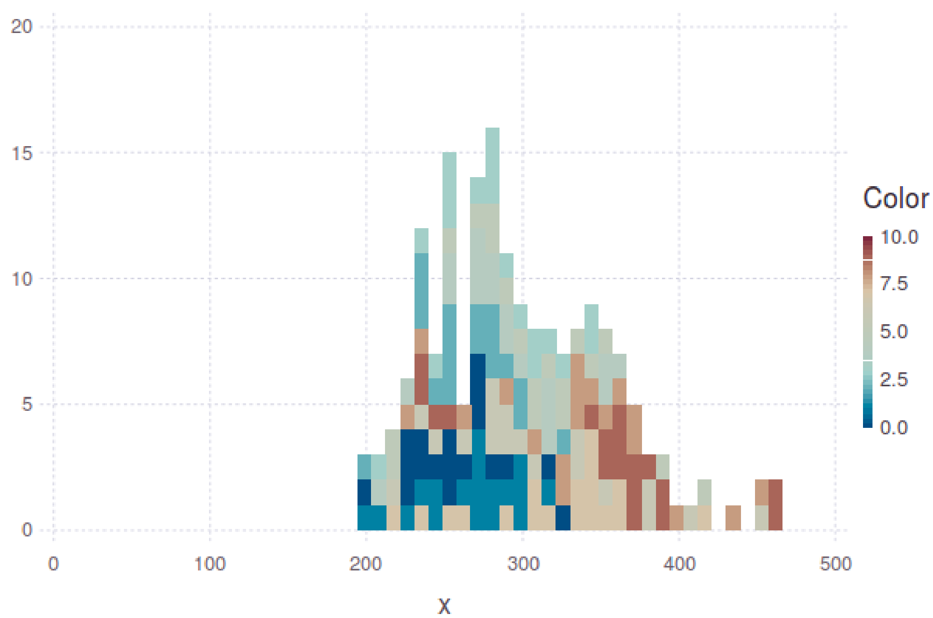
In [2]:
```
using RDatasets
using Distributions
using StatsBase
using Gadfly
```

In [3]:
```
sleep = dataset("lme4","sleepstudy")
```

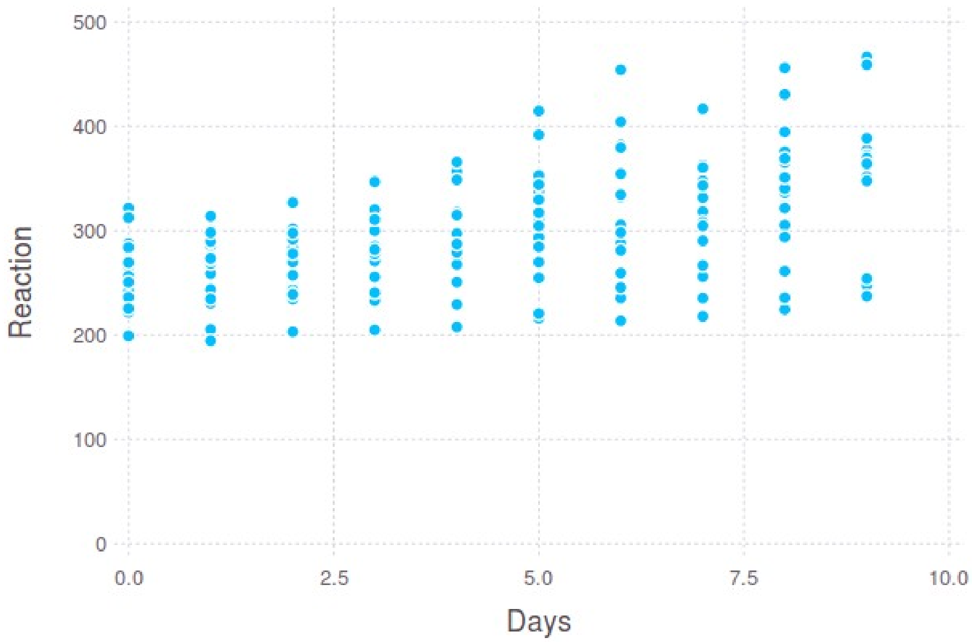
Out[3]:

	Reaction	Days	Subject
1	249.56	0	308
2	258.7047	1	308
3	250.8006	2	308
4	321.4398	3	308

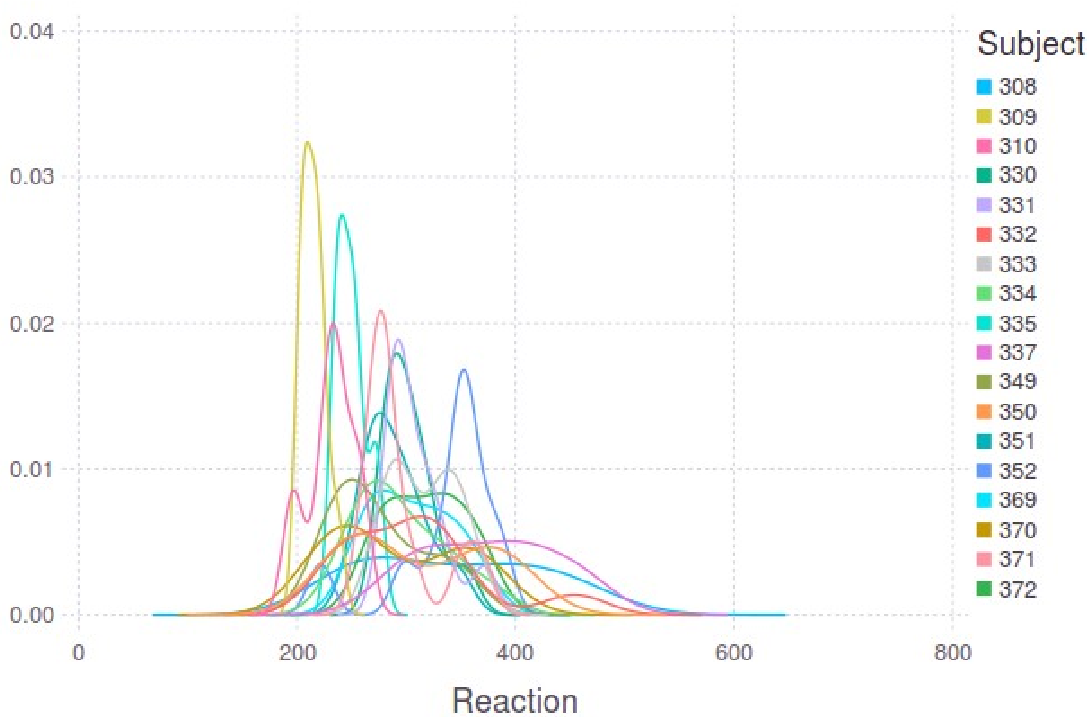
```
plot(x = sleep[:Reaction], Geom.histogram(bincount = 30), color = sleep[:Days])
```



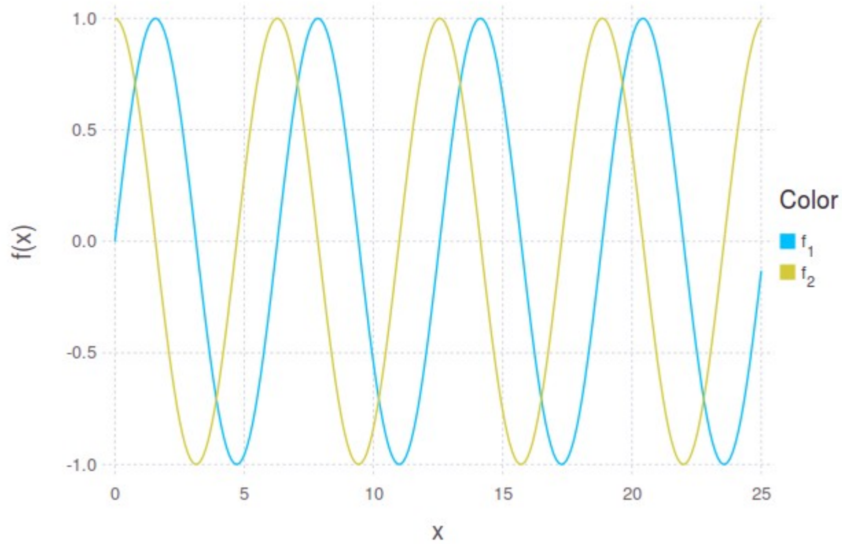
```
plot(sleep, x = "Days", y = "Reaction", Geom.point)
```




```
plot(sleep, x = "Reaction", Geom.density, color = "Subject")
```



```
In [11]: plot([sin, cos], 0, 25)
Out[11]:
```




```
julia> a = rand(6)
6-element Array{Float64,1}:
 0.167763
 0.015309
 0.681381
 0.842937
 0.894316
 0.432843

julia> autocov(a)
6-element Array{Float64,1}:
  0.109268
  0.0402556
 -0.0301791
 -0.0528897
 -0.0159282
  0.00410752
```

```
julia> autocor(a)
6-element Array{Float64,1}:
  1.0
  0.368411
 -0.276194
 -0.484037
 -0.145772
  0.0375912

julia> crosscor(a,b)
11-element Array{Float64,1}:
  0.0637495
 -0.324786
 -0.391401
 -0.160508
  0.576313
```

Chapter 4: Deep Dive into Inferential Statistics

```
using DataFrames
using RDatasets
using Distributions
using Gadfly
```

In [22]:

```
srand(619)
```

```
super(Normal)
```

```
Distributions.Distribution{Distributions.Univariate,Distributi
ons.Continuous}
```

```
names(Normal)
```

```
WARNING: names(t::DataType) is deprecated, use fieldnames(t) i
nstead.
```

```
2-element Array{Symbol,1}:
 :μ
 :σ
```

```
dist1 = Normal(1.0, 3.0)
```

```
Distributions.Normal(μ=1.0, σ=3.0)
```

```
params(dist1)
```

```
(1.0,3.0)
```

```
dist1.μ
```

```
1.0
```

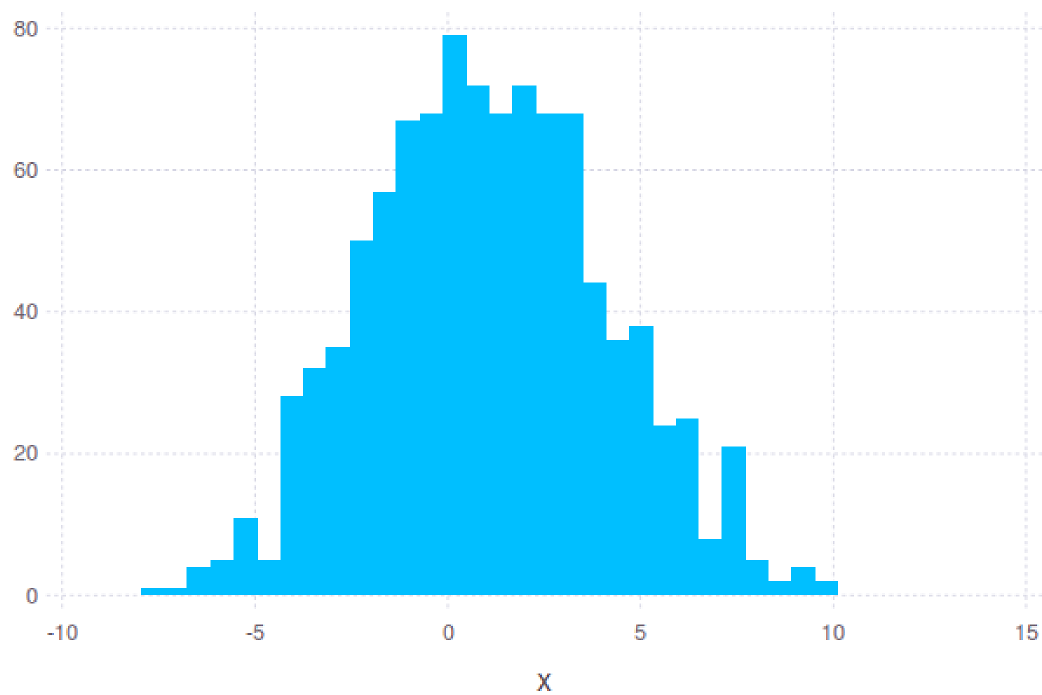
```
dist1.σ
```

```
3.0
```

```
x = rand(dist1, 1000)
```

```
1000-element Array{Float64,1}:
  7.53922
 -4.27887
  1.54685
 -1.3515
 -2.68357
  3.62544
```

```
plot(x = rand(dist1, 1000), Geom.histogram(bincount = 30))
```



```
fit(Normal, x)
```

```
Distributions.Normal(μ=1.0038374150247216, σ=2.99929
17508924752)
```

```
abstract Distributions.Sampleable{F<:Distributions.
VariateForm,S<:Distributions.ValueSupport} <: Any
```

```
abstract Distributions.Distribution{F<:Distributions
.VariateForm,S<:Distributions.ValueSupport} <: Distr
ibutions.Sampleable{F<:Distributions.VariateForm,S<:
Distributions.ValueSupport}
```

```
julia> using Distributions

julia> Binomial()
Distributions.Binomial(n=1, p=0.5)

julia> n=5
5

julia> Binomial(n)
Distributions.Binomial(n=5, p=0.5)

julia> p=0.3
0.3

julia> Binomial(n,p)
Distributions.Binomial(n=5, p=0.3)
```

```
julia> Cauchy()
Distributions.Cauchy(μ=0.0, σ=1.0)

julia> Cauchy(u,s)
Distributions.Cauchy(μ=0.2, σ=1.5)
```

$$f(x|a < X <= b) = \frac{g(x)}{F(b) - F(a)} = TruncatedD(x)$$

Summary:

```
immutable Distributions.Truncated{D<:Distributions
.Distribution{Distributions.Univariate,S<:Distribu
tions.ValueSupport},S<:Distributions.ValueSupport}
 <: Distributions.Distribution{Distributions.Univa
riate,S<:Distributions.ValueSupport}
```

Fields:

```
untruncated :: D<:Distributions.Distribution{Distr
ibutions.Univariate,S<:Distributions.ValueSupport}
lower       :: Float64
upper       :: Float64
lcdf        :: Float64
ucdf        :: Float64
tp          :: Float64
logtp       :: Float64

Truncated TruncatedNormal truncate
```

`Distributions.MultivariateDistribution` is of type `TypeConstructor`:

Summary:

```
immutable TypeConstructor <: Type{T}
```

Fields:

```
parameters :: SimpleVector
body       :: Any
```

```
MultivariateDistribution
DiscreteMultivariateDistribution
```

```
  immutable Distributions.MvNormal{Cov<:PDMats.AbstractPDMat,Mean<:Uni
on{Array{Float64,1},Distributions.ZeroVector{Float64}}} <: Distributio
ns.AbstractMvNormal
```

Fields:

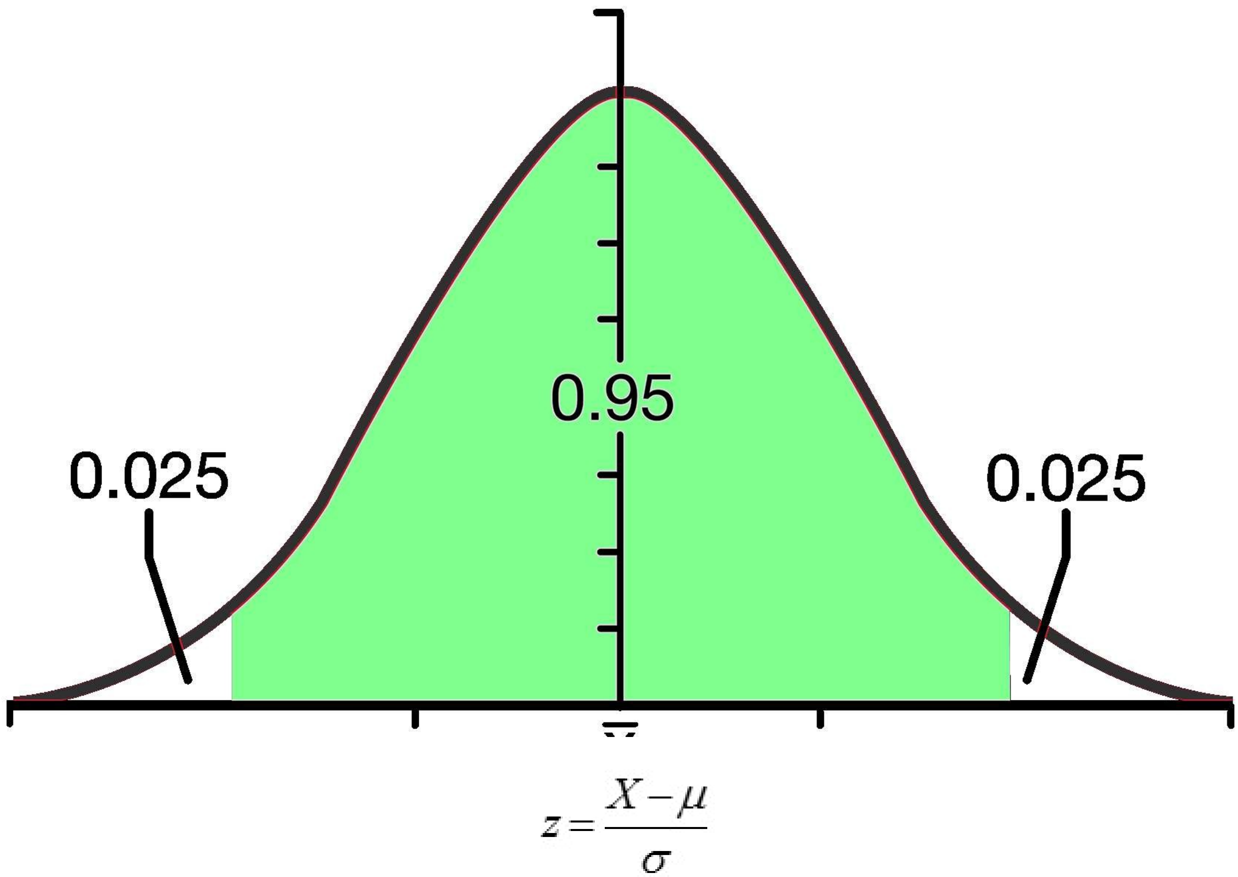
```
μ :: Mean<:Union{Array{Float64,1},Distributions.ZeroVector{Float64}}
Σ :: Cov<:PDMats.AbstractPDMat
```

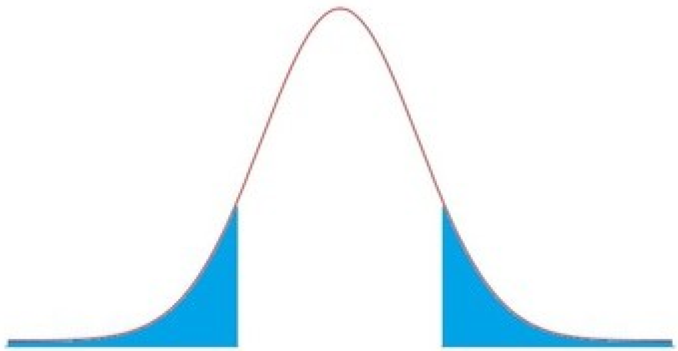
Summary:

```
  immutable Distributions.Dirichlet <: Distributions.Distribution{Distributions.
Multivariate,Distributions.Continuous}
```

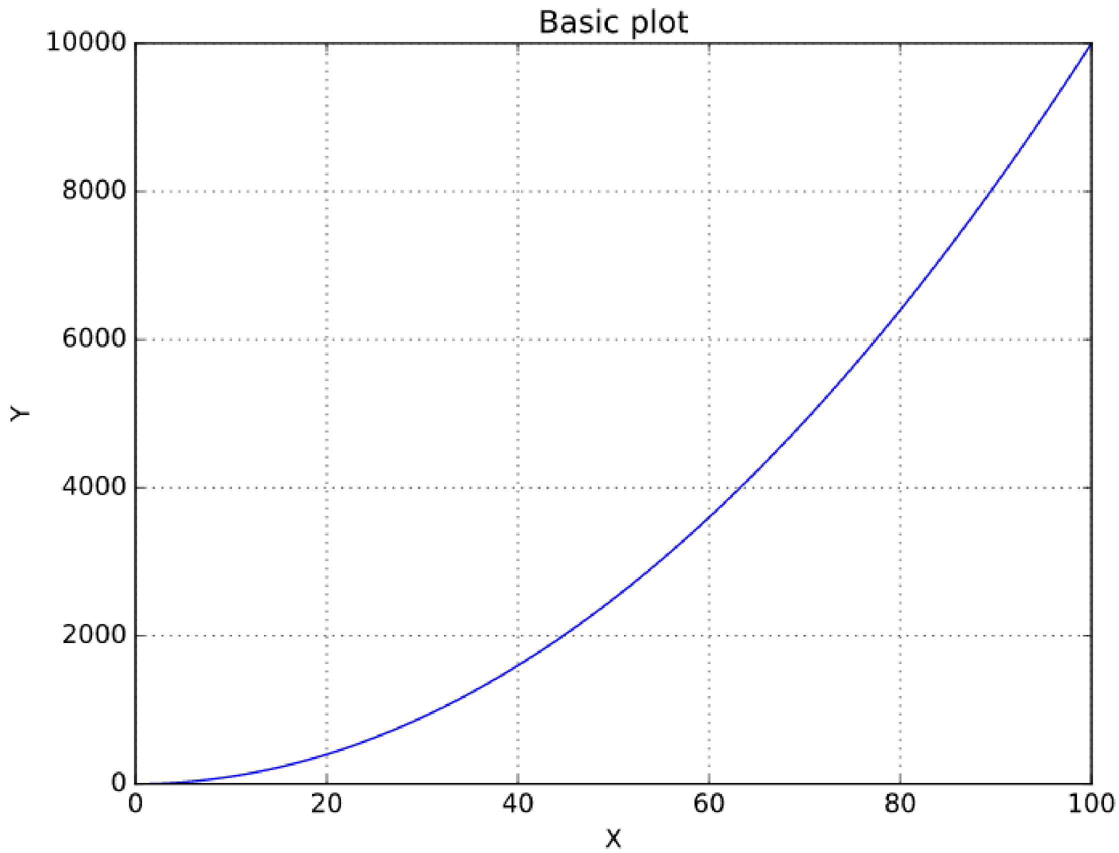
Fields:

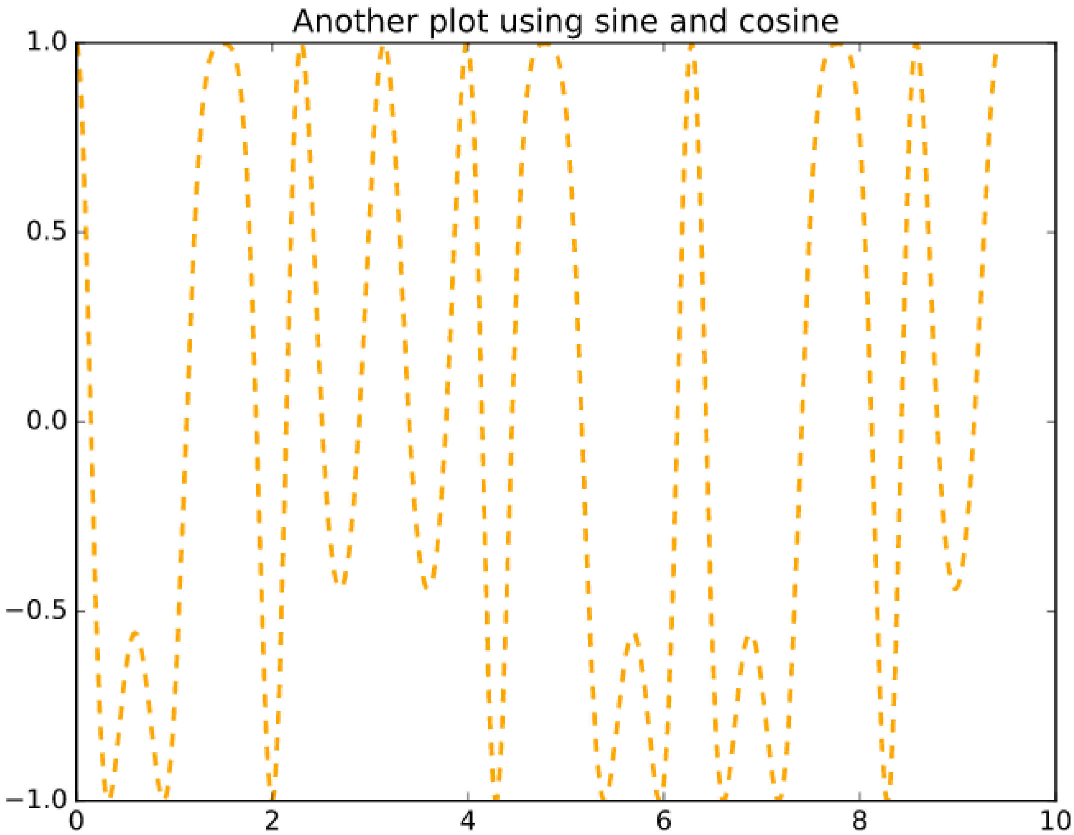
```
alpha  :: Array{Float64,1}
alpha0 :: Float64
lmnB   :: Float64
```



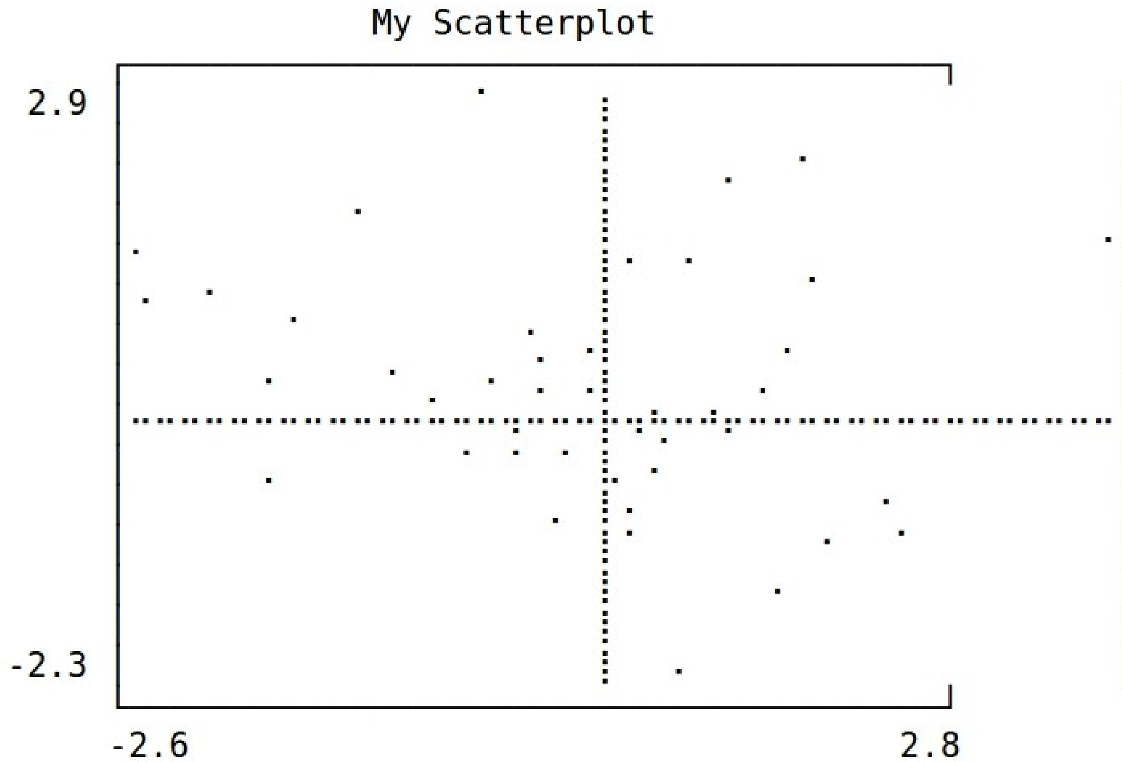


Chapter 5: Making Sense of Data Using Visualization

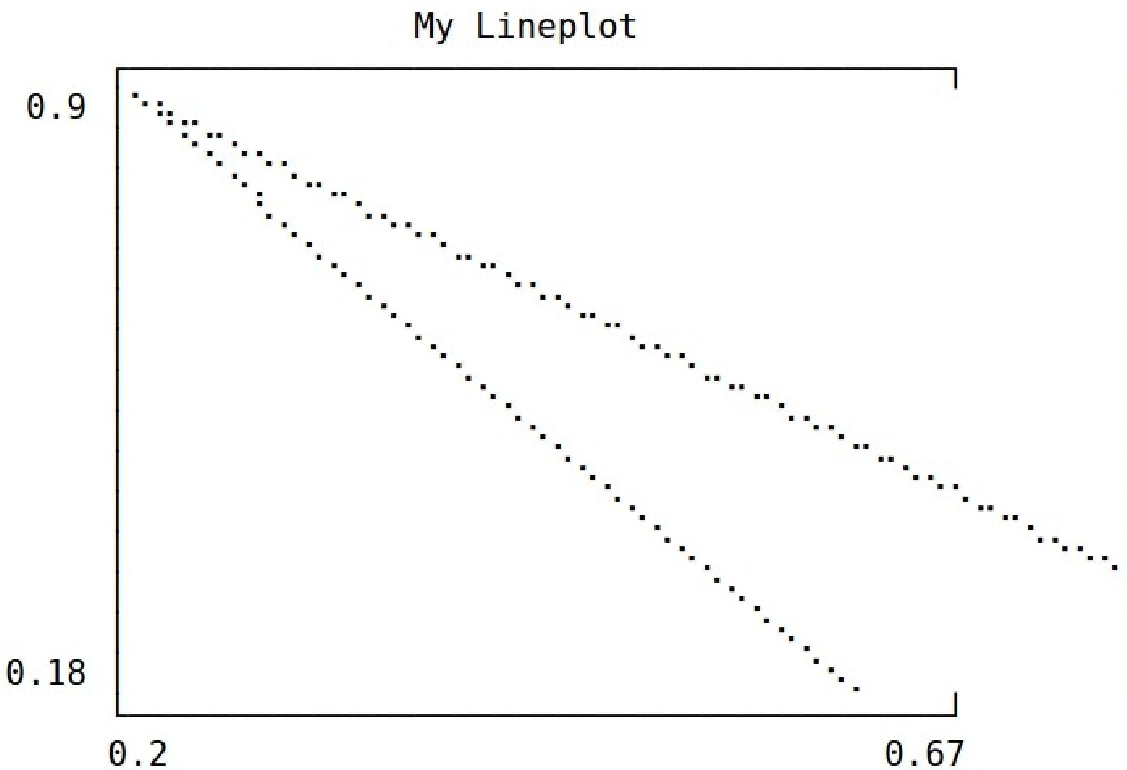


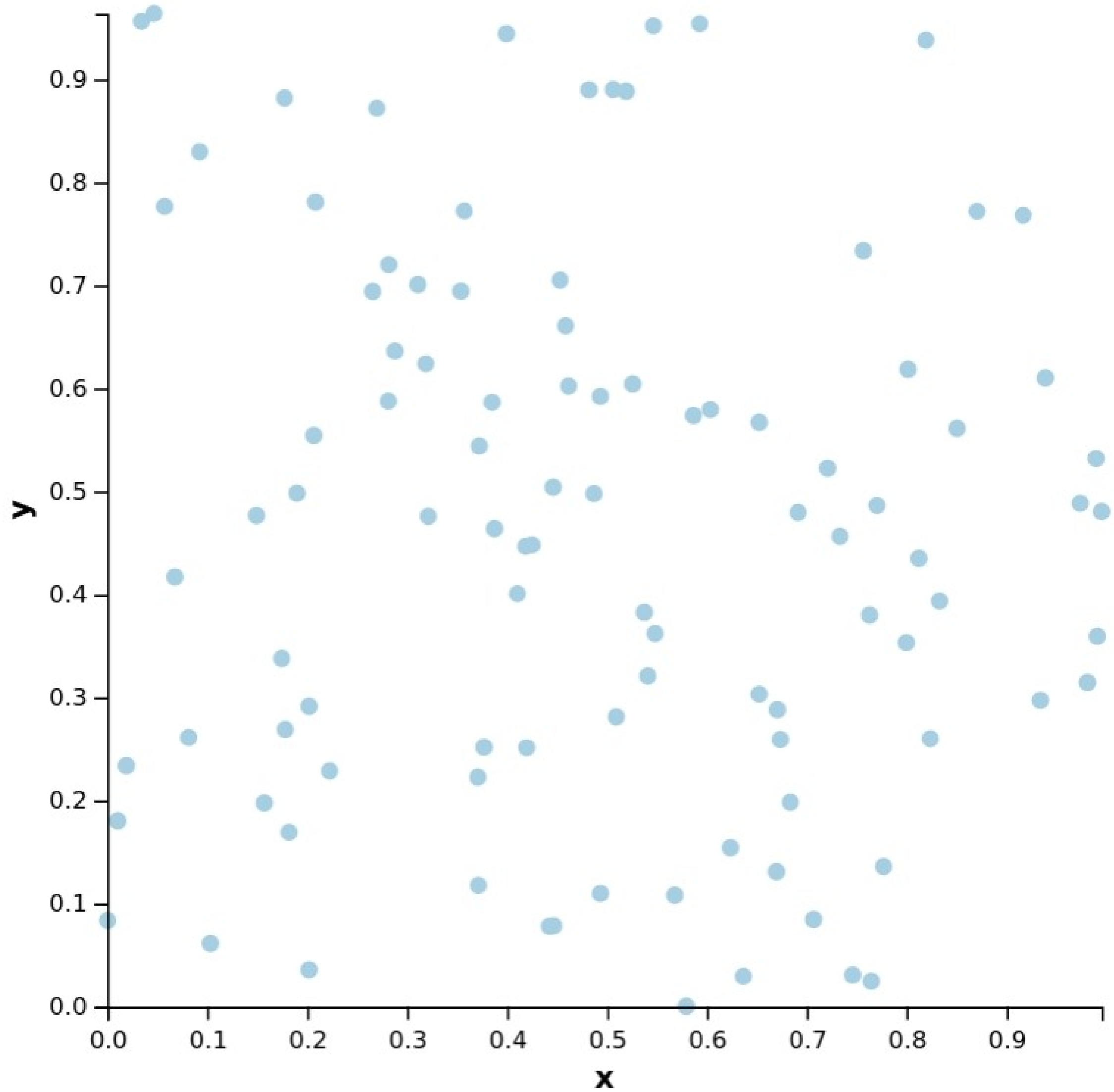

Another plot using sine and cosine
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10

```
scatterplot(randn(50), randn(50),
    title = "My Scatterplot")
```

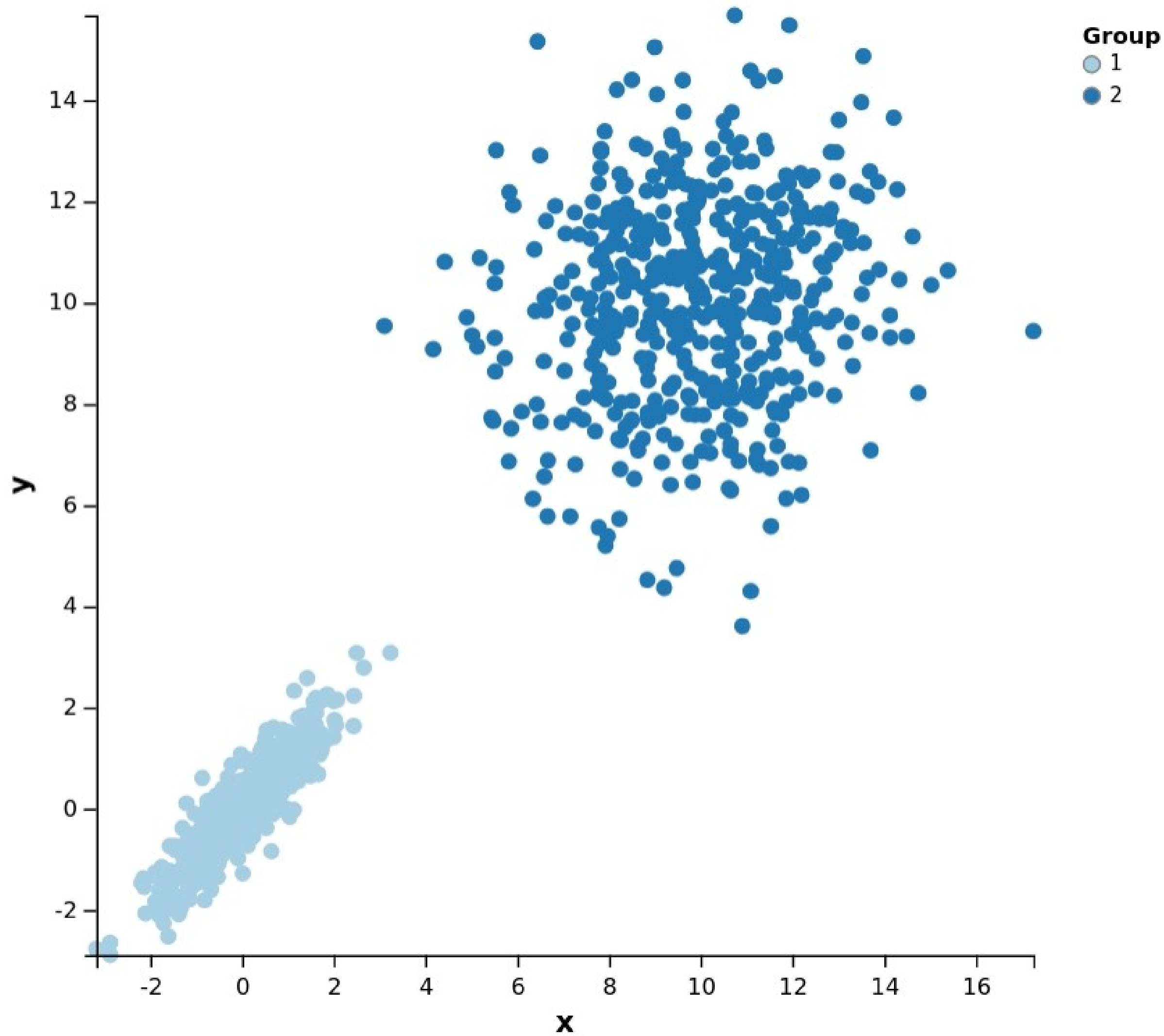


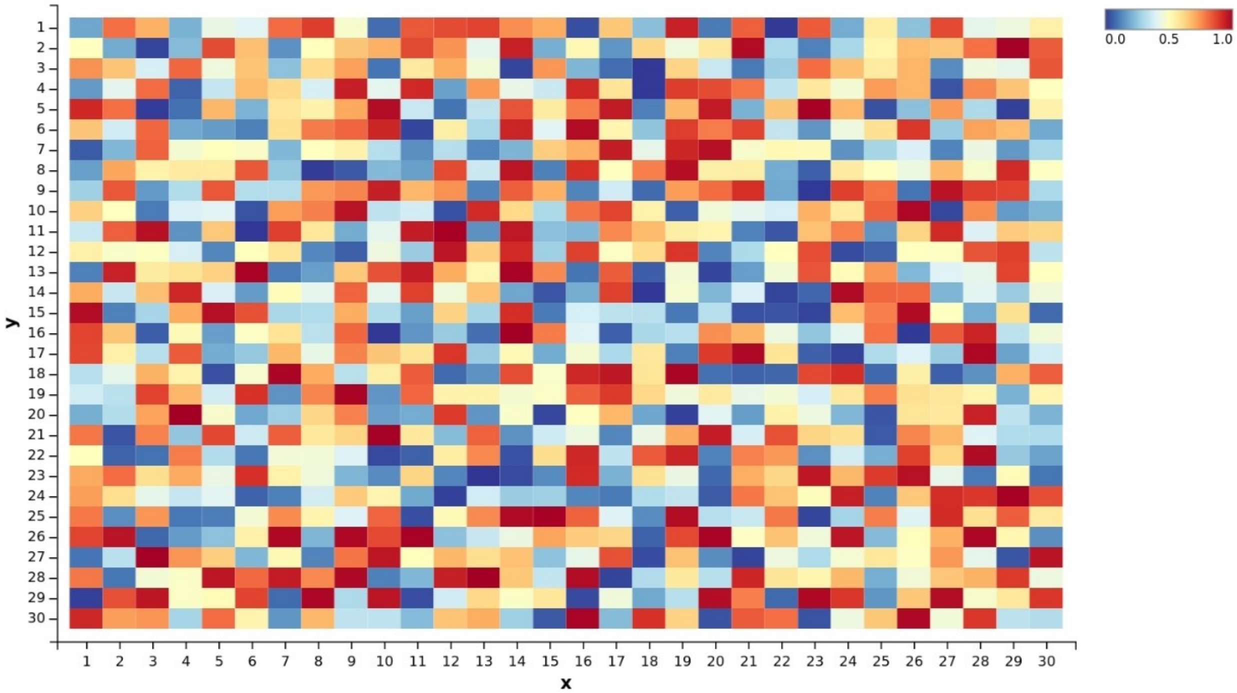
```
lineplot(rand(3), rand(3), title = "My Lineplot")
```



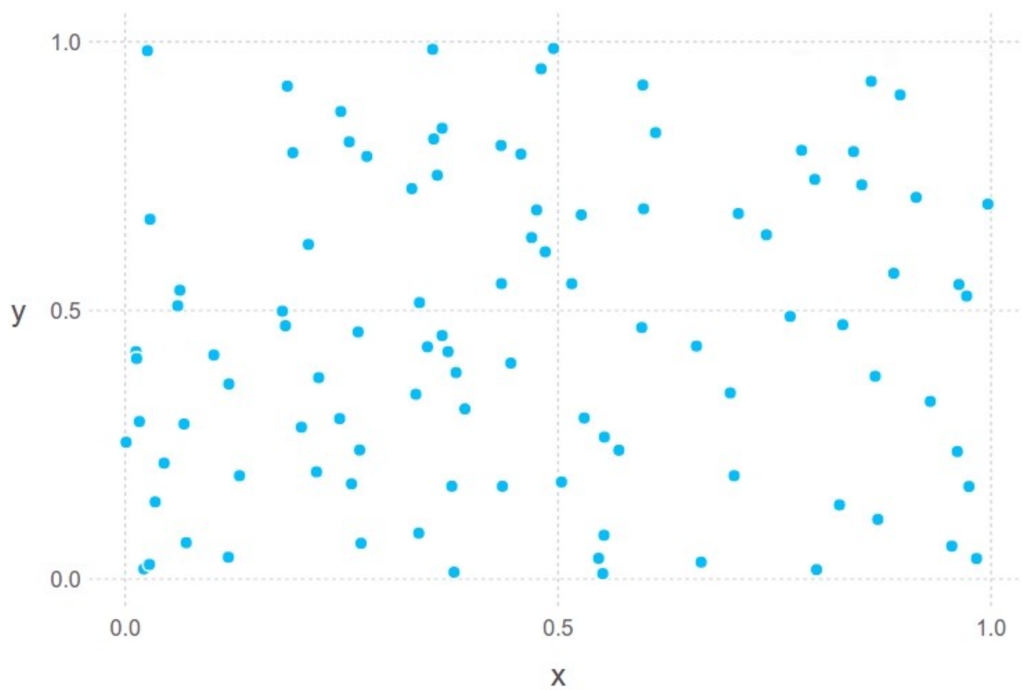



```
using Vega, Distributions

d1 = MultivariateNormal([0.0, 0.0], [1.0 0.9; 0.9 1.0])
d2 = MultivariateNormal([10.0, 10.0], [4.0 0.5; 0.5 4.0])
points = vcat(rand(d1, 500)', rand(d2, 500)')

x = points[:, 1]
y = points[:, 2]
group = vcat(ones(Int, 500), ones(Int, 500) + 1)

scatterplot(x = x, y = y, group = group)
```

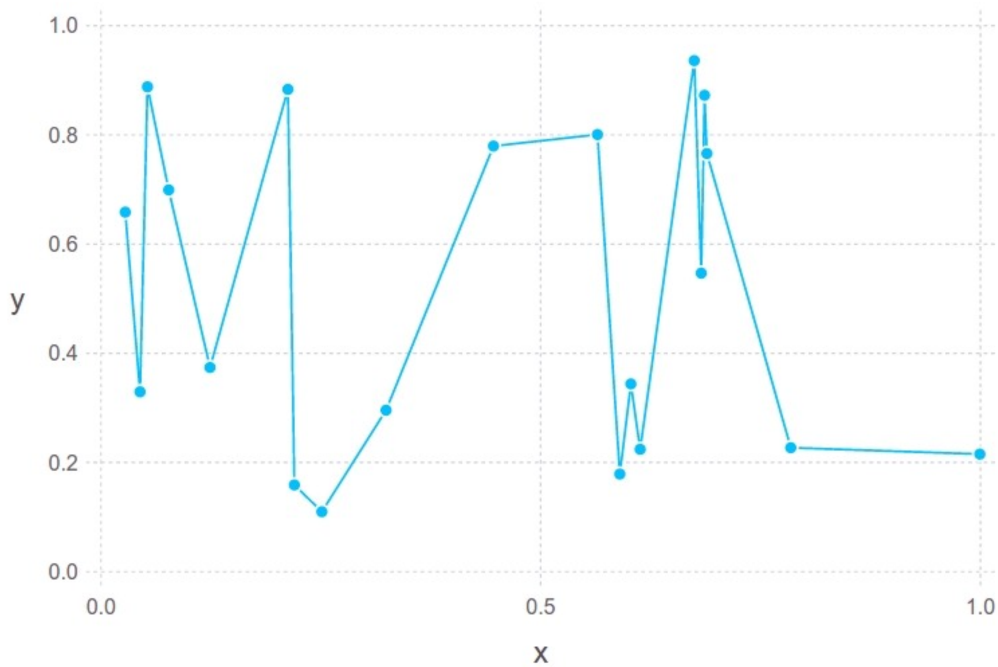

Group
1
2
y
14
12
10
8
6
4
2
0
-2
x
-2
0
2
4
6
8
10
12
14
16



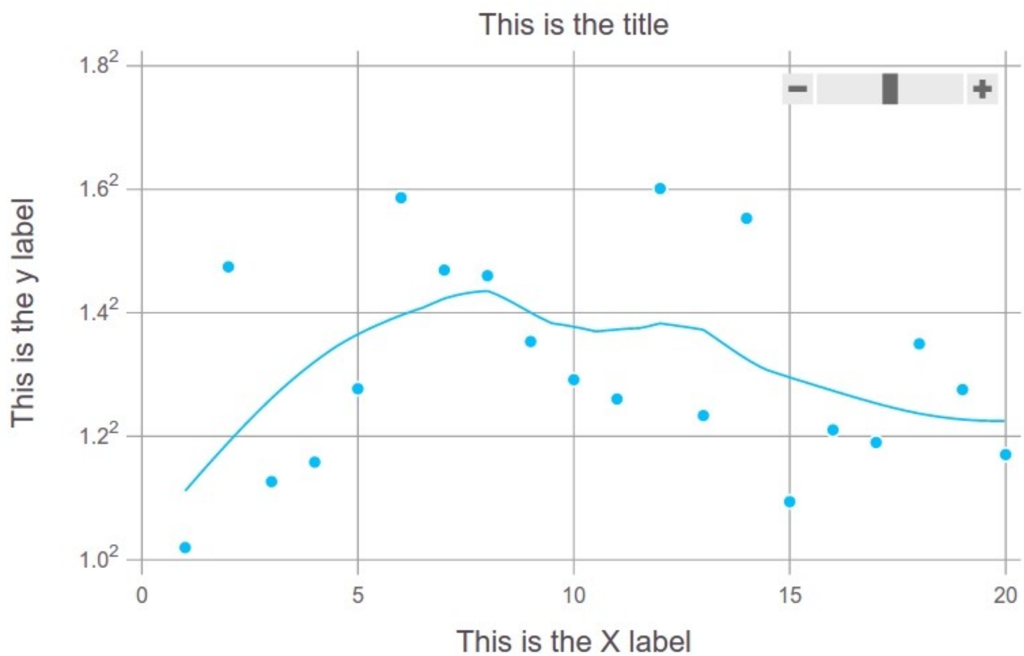

```
plot(x=rand(100), y=rand(100))
```

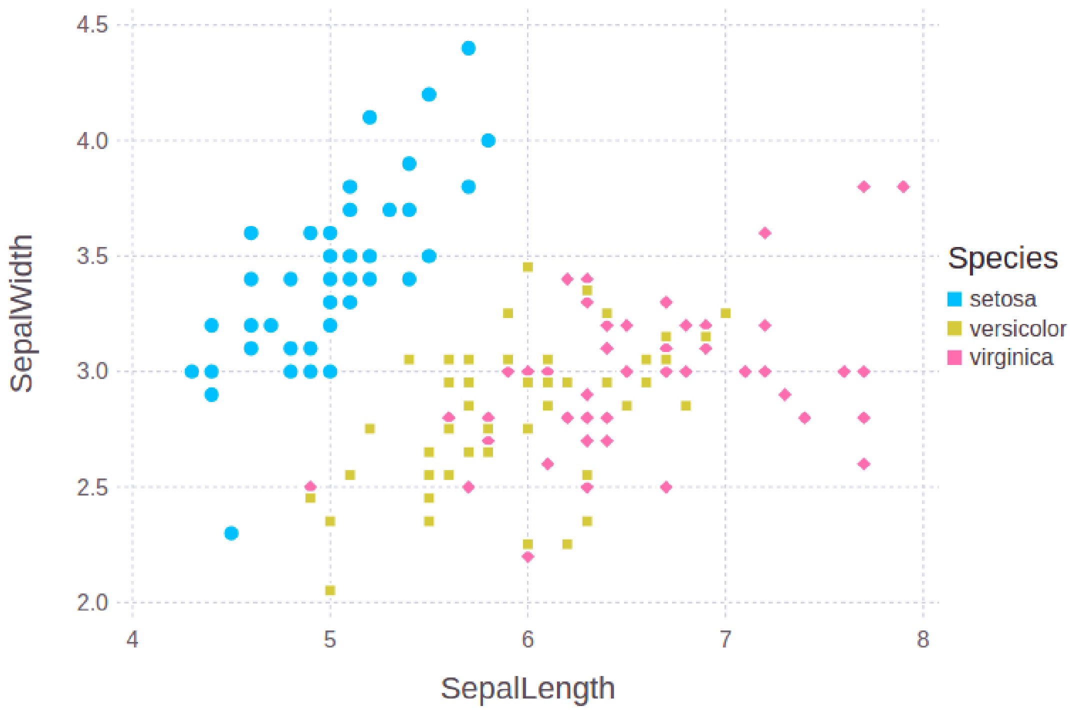


```
plot(x=rand(20), y=rand(20), Geom.point, Geom.line)
```

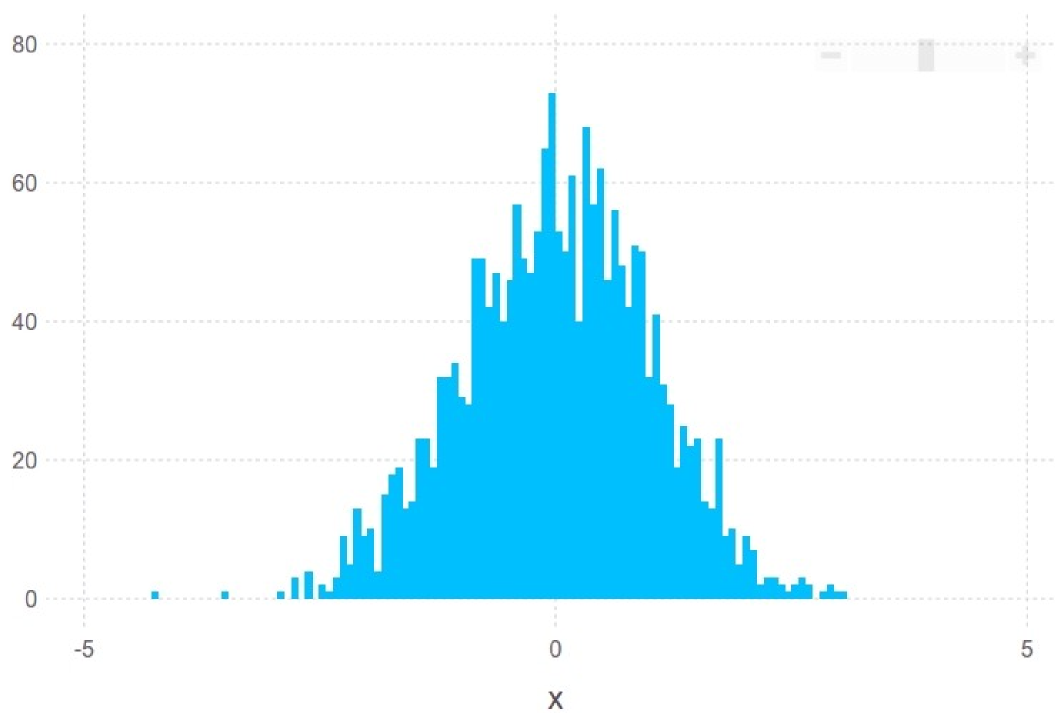


```
plot(x=1:20, y=3.^rand(20),
     Scale.y_sqrt, Geom.point, Geom.smooth,
     Guide.xlabel("This is the X label"),
     Guide.ylabel("This is the y label"),
     Guide.title("This is the title"))
```

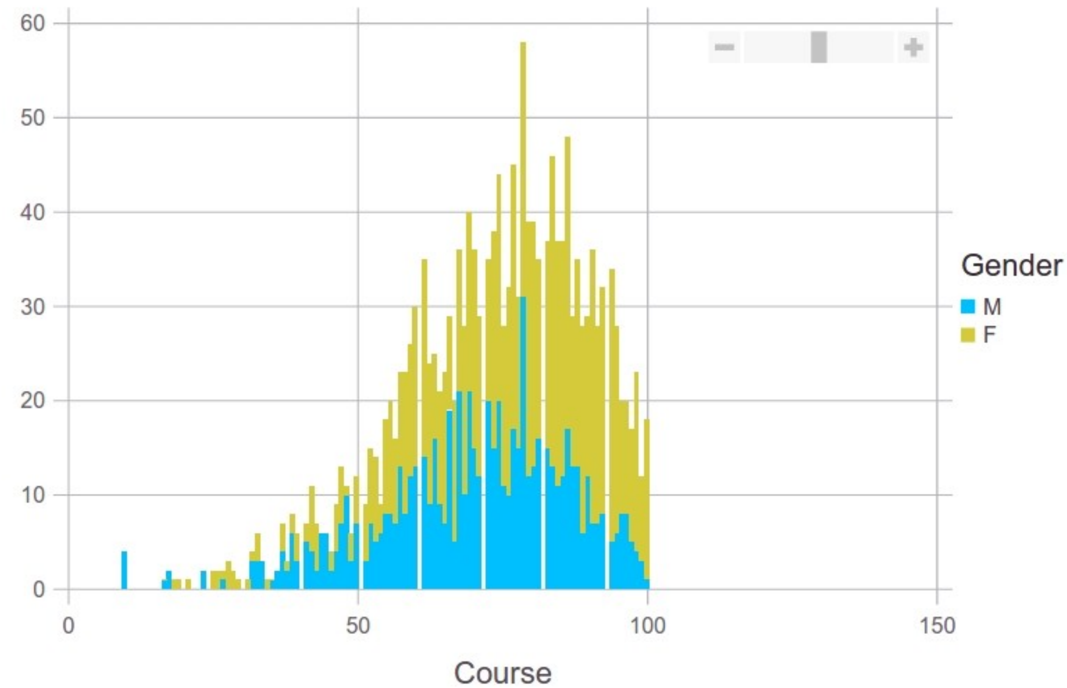




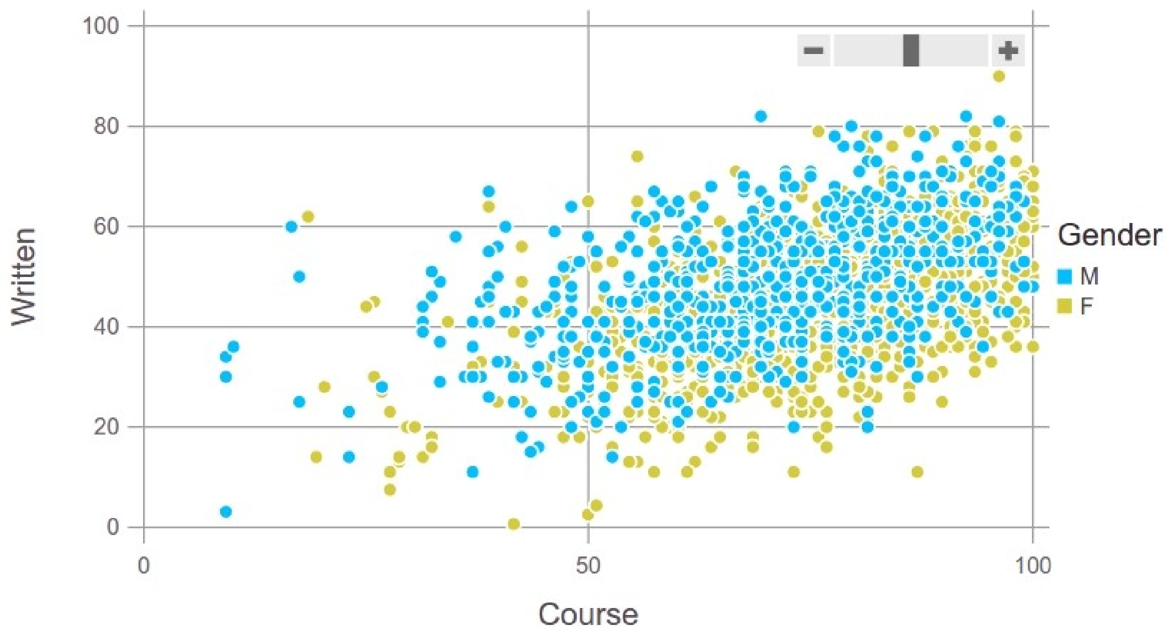

```
p = plot(x=randn(2000), Geom.histogram(bincount=100))
```



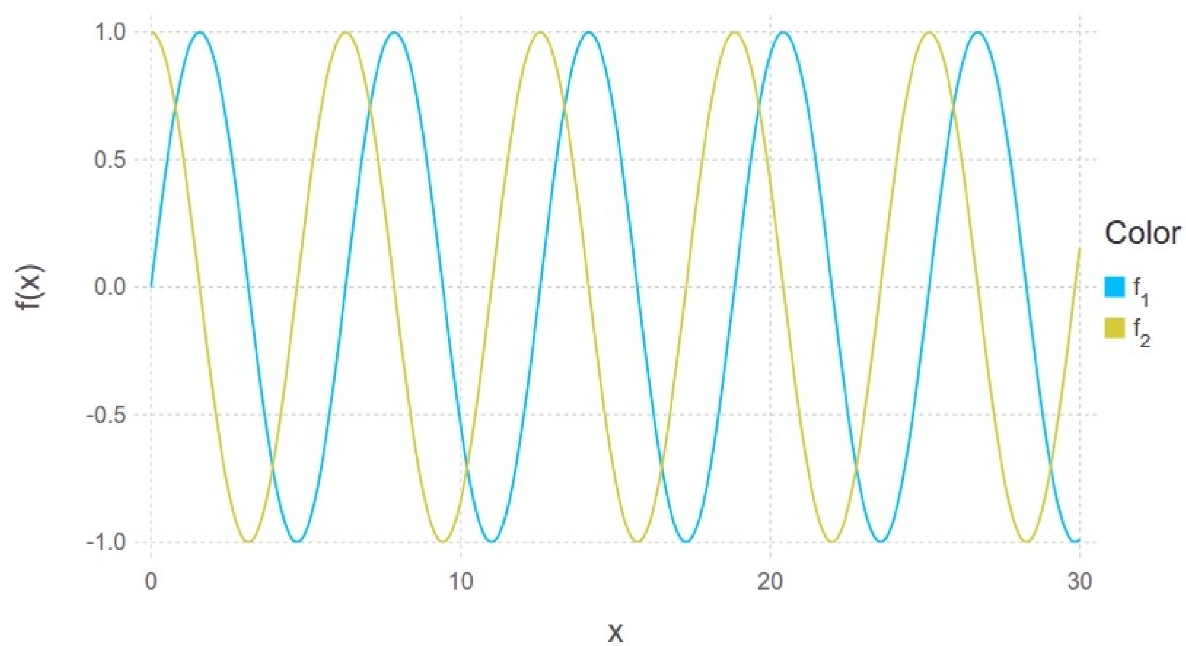
```
plot(dataset("mlmRev","Gcsemv"),
    x="Course", color="Gender", Geom.histogram)
```



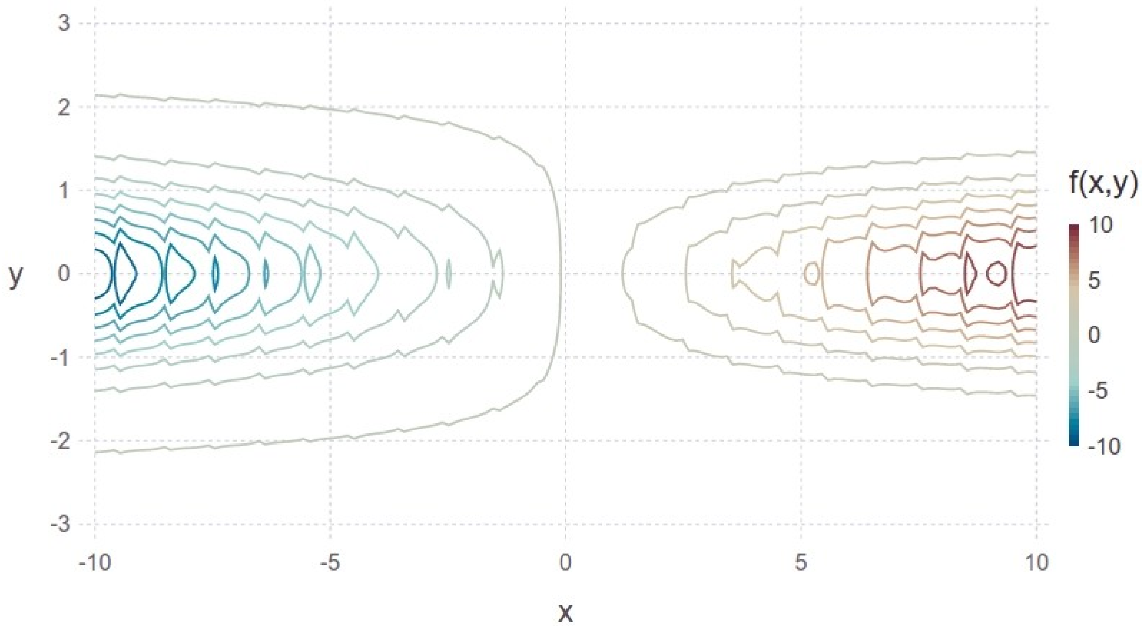
```
set_default_plot_size(15cm, 9cm);
mlmf = dataset("mlmRev","Gcsemv")
df = mlmf[complete_cases(mlmf), :]
names(df)
plot(df, x="Course", y="Written", color="Gender")
```



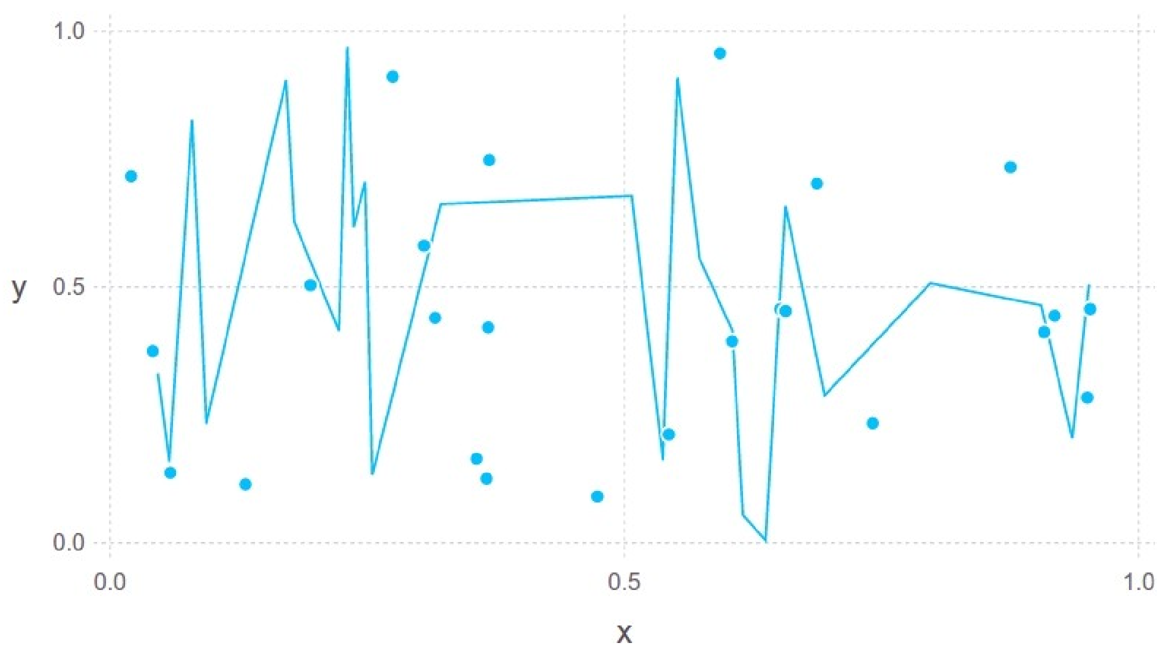

```
plot([sin, cos], 0, 30)
```



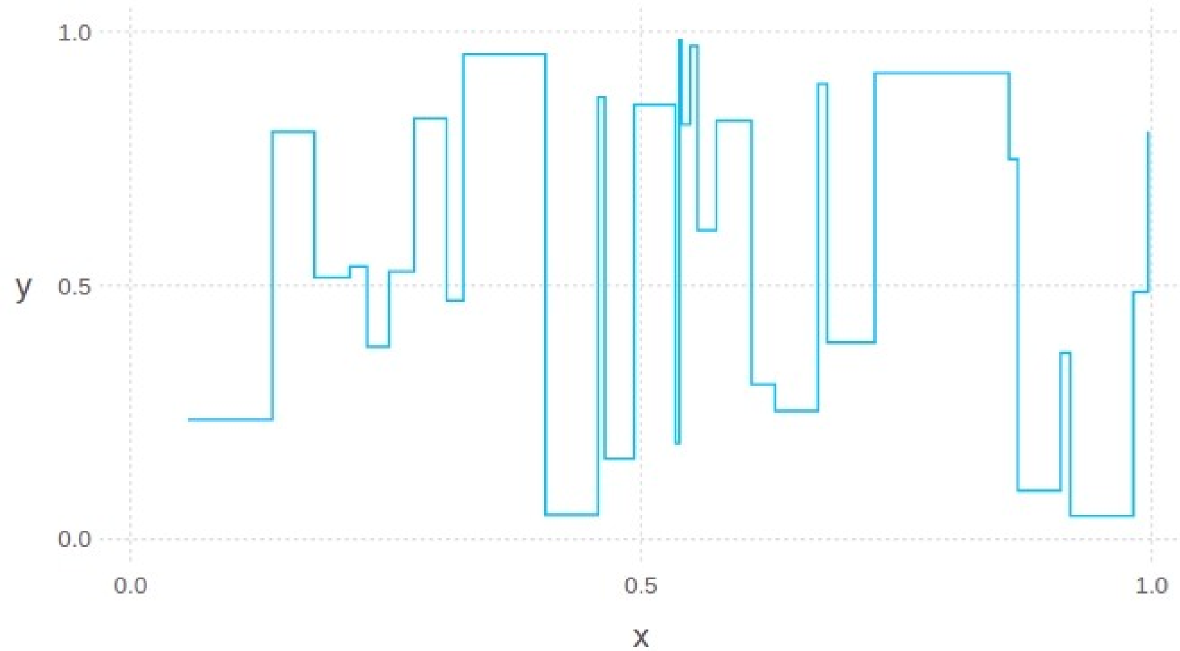
```
plot((x,y) -> x*exp(-(x-int(x))^3-y^2), -10., 10, -3., 3)
```



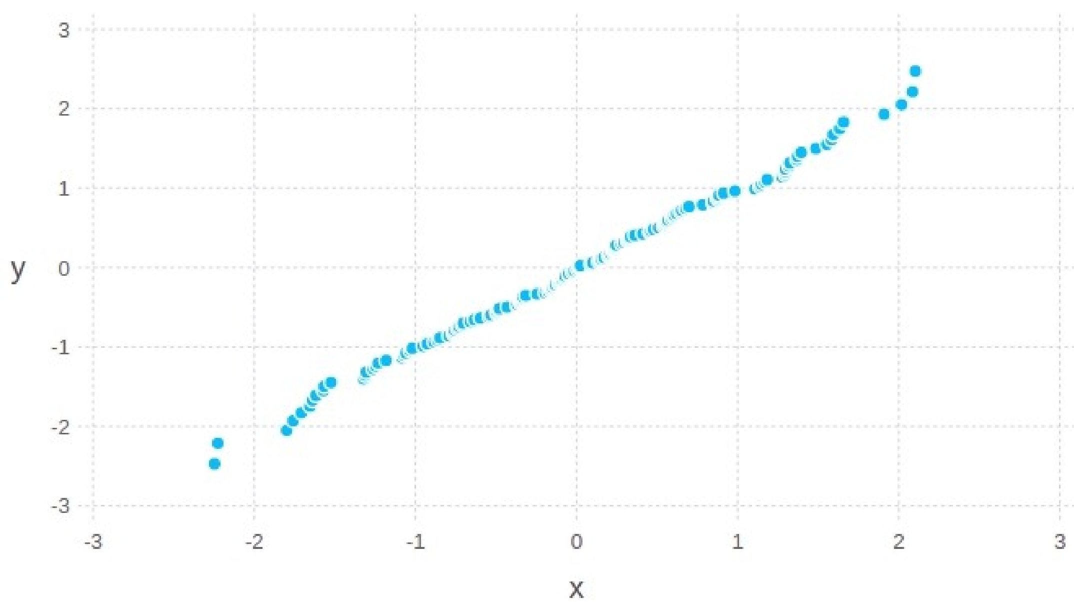

```
plot(layer(x=rand(25), y=rand(25), Geom.point),
        layer(x=rand(25), y=rand(25), Geom.line))
```



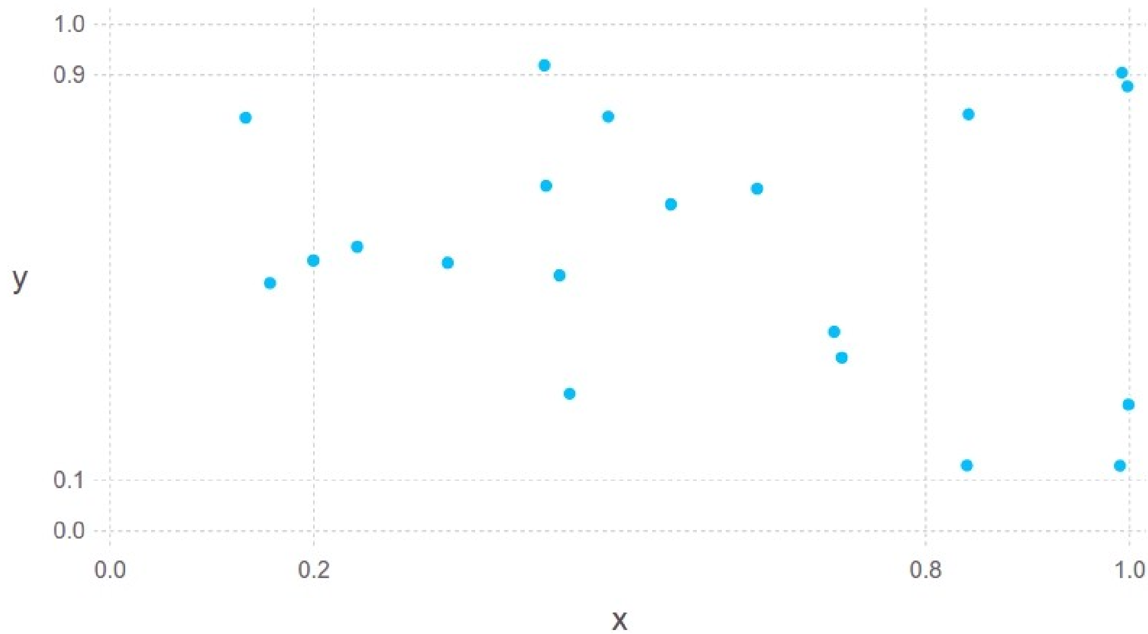
```
plot(x=rand(30), y=rand(30), Stat.step, Geom.line)
```



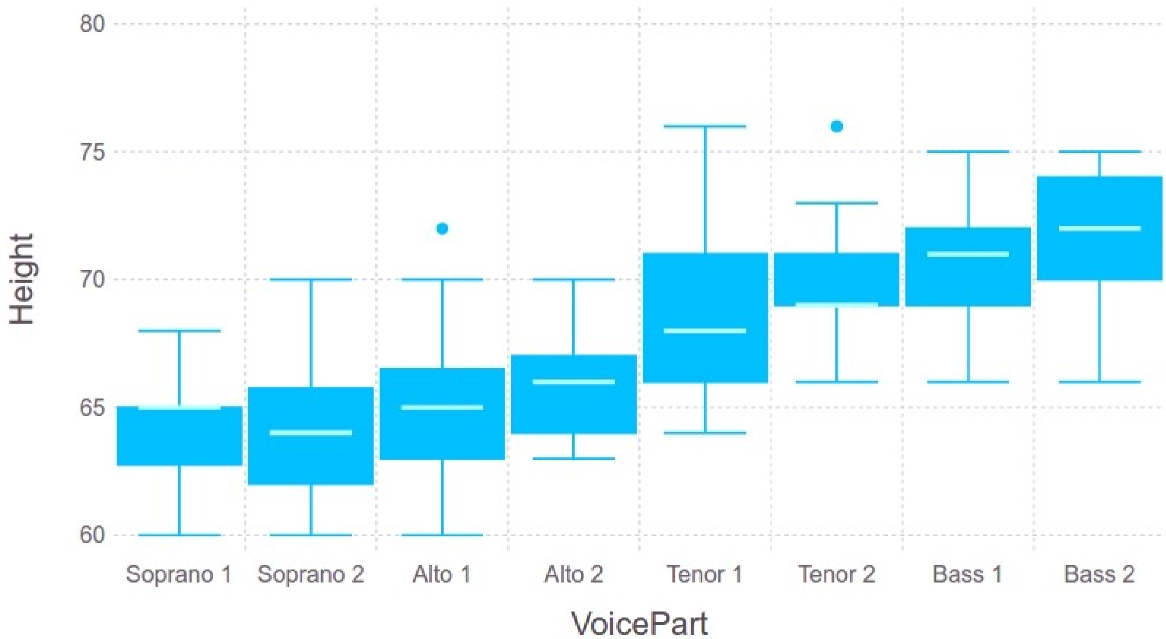

```
using Distributions
plot(x=rand(Normal(), 150), y=rand(Normal(), 150), Stat.qq, Geom.point)
plot(x=rand(Normal(), 150), y=Normal(), Stat.qq, Geom.point)
```



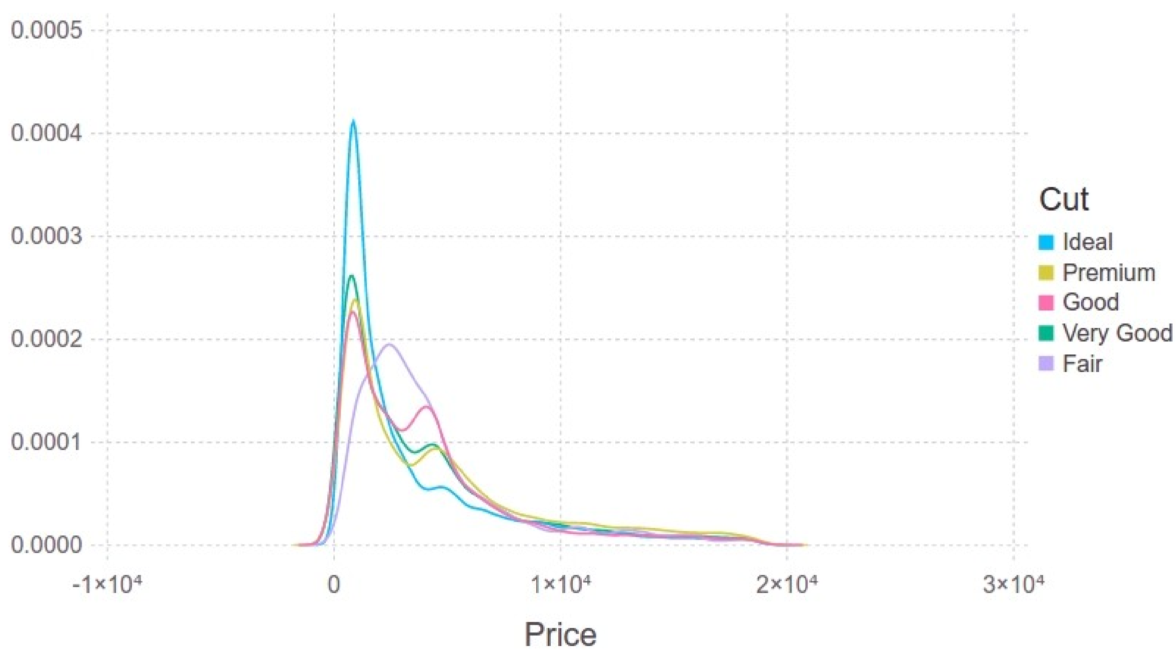
```
# Providing a fixed set of ticks
plot(x=rand(20), y=rand(20),
     Stat.xticks(ticks=[0.0, 0.2, 0.8, 1.0]),
     Stat.yticks(ticks=[0.0, 0.1, 0.9, 1.0]),
     Geom.point)
```



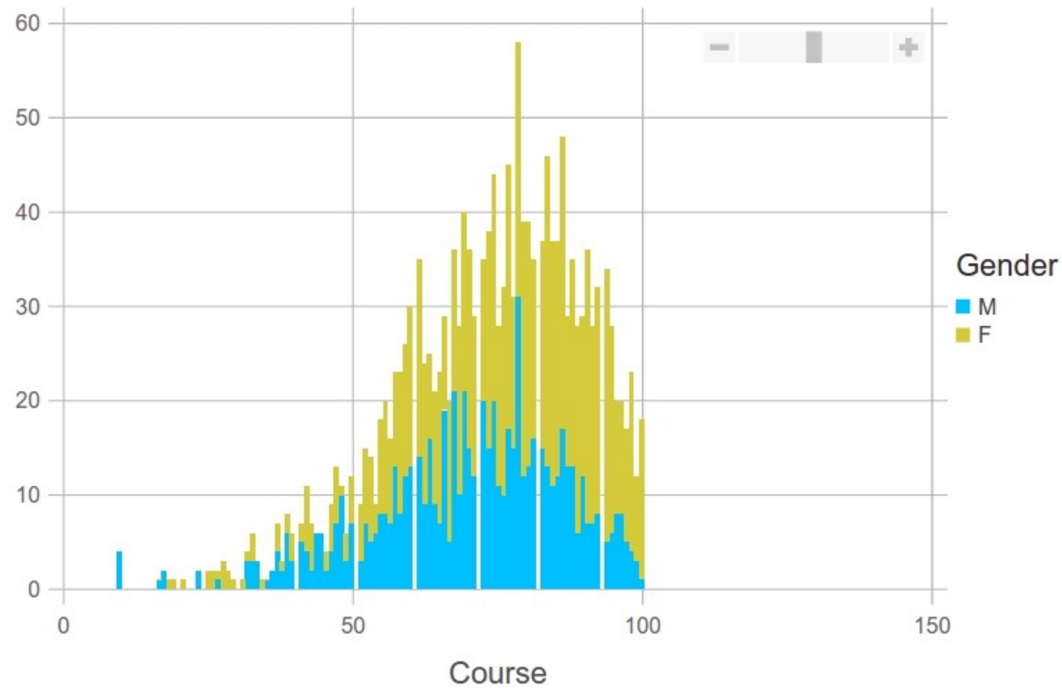
```
plot(dataset("lattice", "singer"),
    x="VoicePart", y="Height", Geom.boxplot)
```



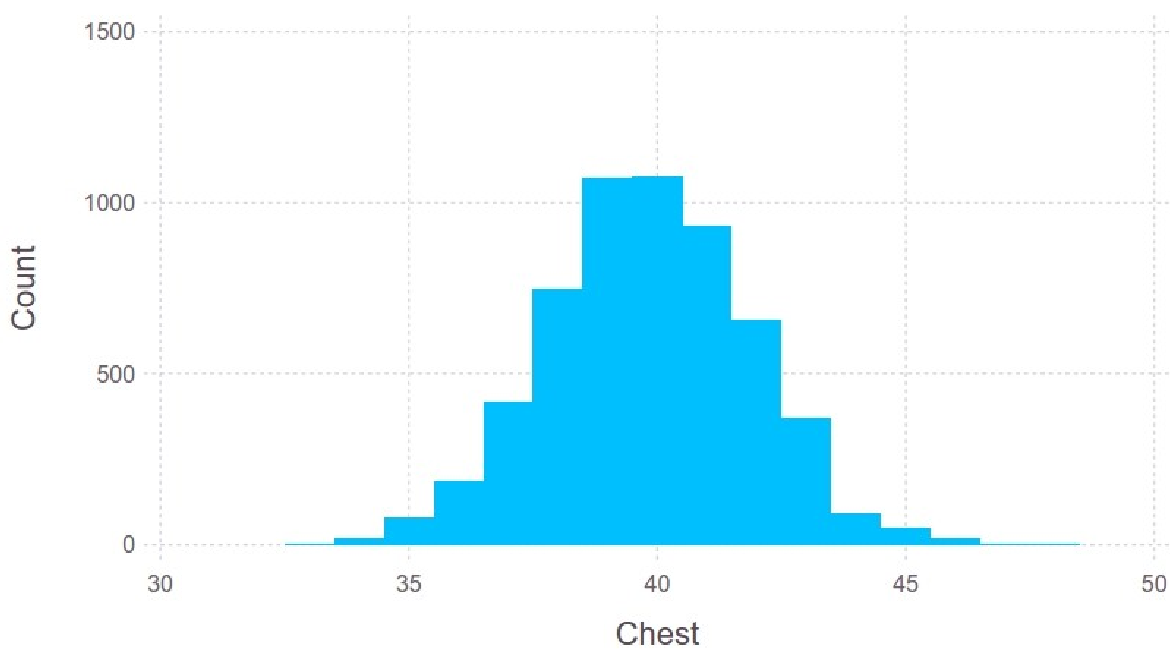

```
plot(dataset("ggplot2", "diamonds"),
    x="Price", color="Cut", Geom.density)
```



```
plot(dataset("mlmRev","Gcsemv"),
    x="Course", color="Gender", Geom.histogram)
```

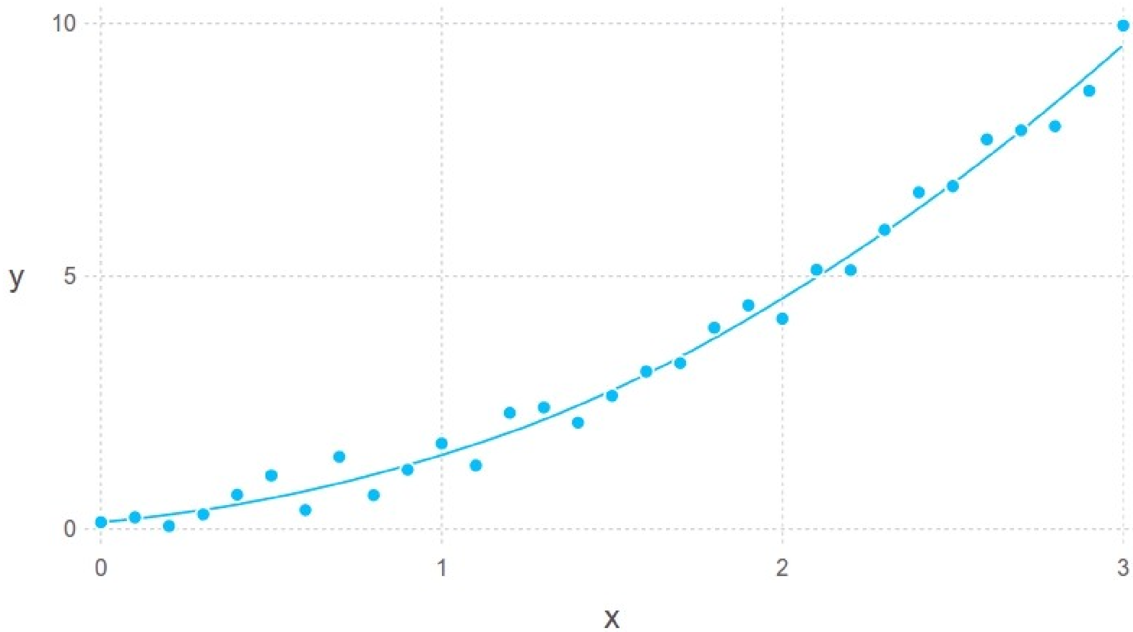



```
plot(dataset("HistData", "ChestSizes"),
    x="Chest", y="Count", Geom.bar)
```

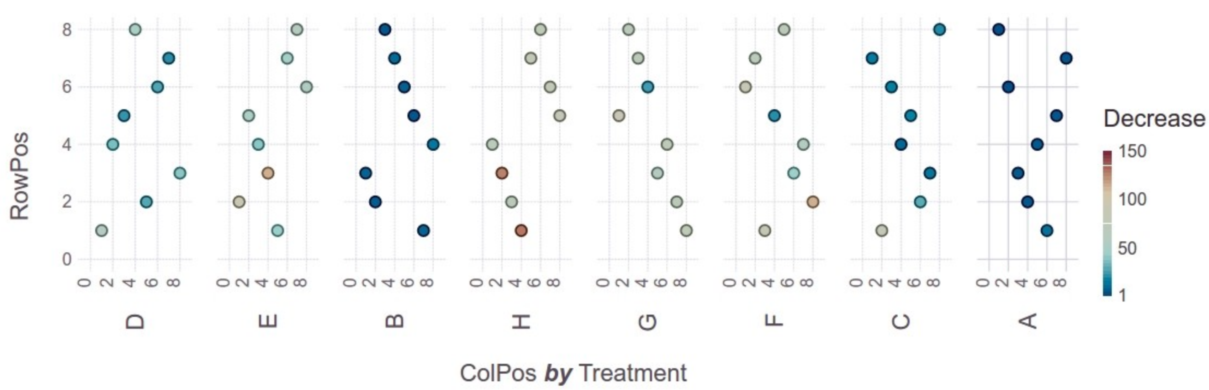


```
# Explicitly setting the number of bins
plot(dataset("car", "UN"), x="GDP", y="InfantMortality",
     Scale.x_log10, Scale.y_log10,
     Geom.histogram2d(xbincount=20, ybincount=20))
```

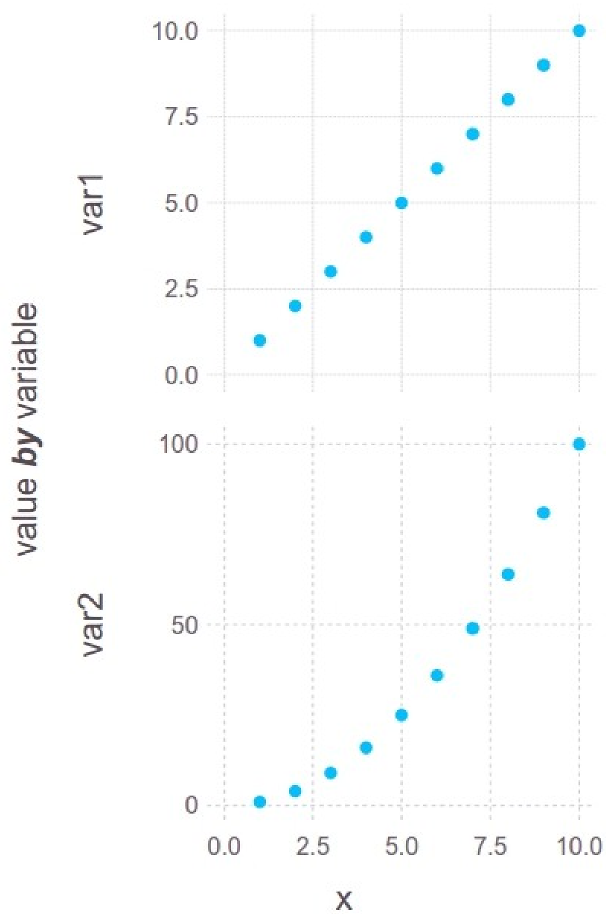
```
x_data = 0.0:0.1:3.0
y_data = x_data.^2 + rand(length(x_data))
plot(x=x_data, y=y_data,
    Geom.point,
    Geom.smooth(method=:loess,smoothing=0.9))
```



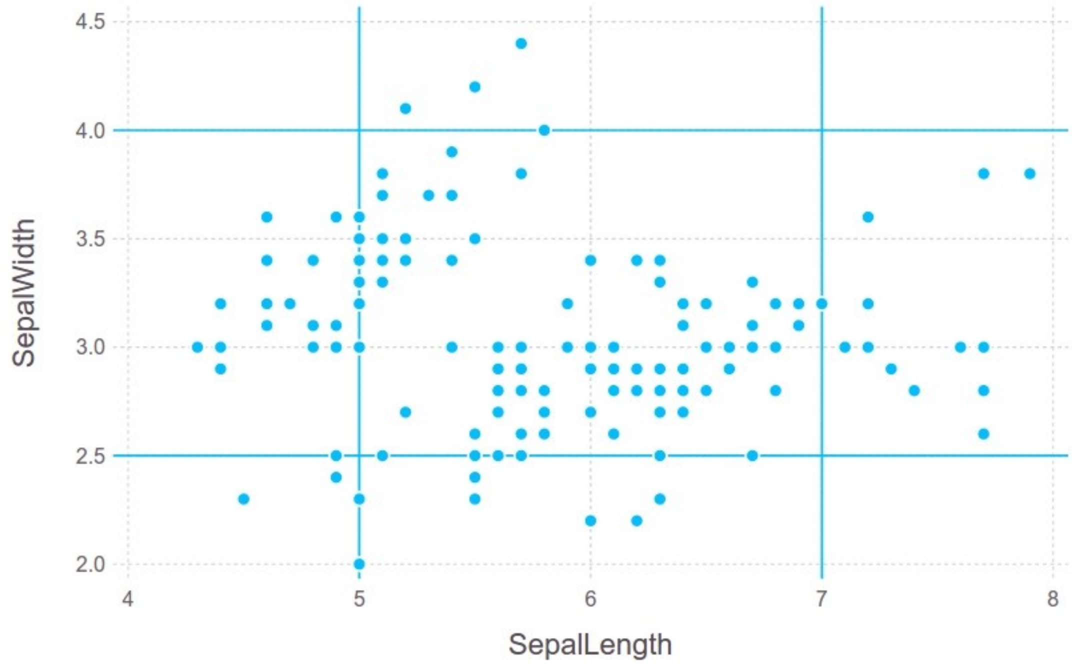

```
set_default_plot_size(20cm, 7.5cm)
plot(dataset("datasets", "OrchardSprays"),
     xgroup="Treatment", x="ColPos", y="RowPos", color="Decrease",
     Geom.subplot_grid(Geom.point))
```

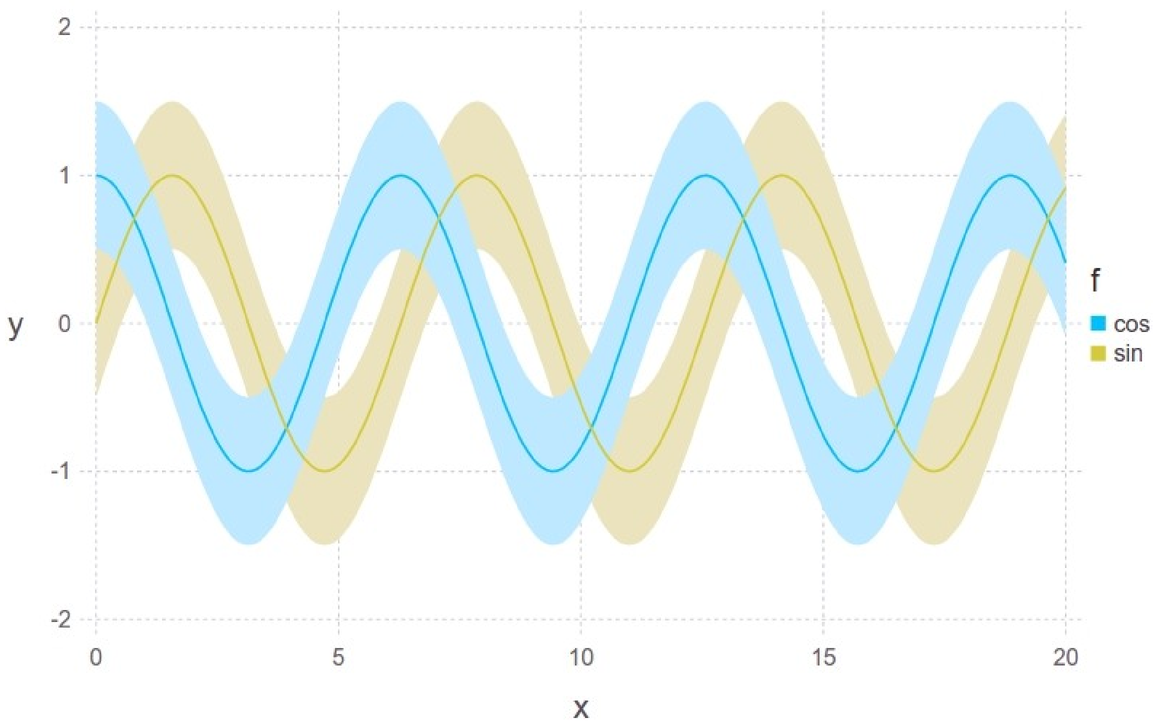


```
using DataFrames
set_default_plot_size(8cm, 12cm)
widedf = DataFrame(x = [1:10], var1 = [1:10], var2 = [1:10].^2)
longdf = stack(widedf, [:var1, :var2])
plot(longdf, ygroup="variable", x="x", y="value",
    Geom.subplot_grid(Geom.point, free_y_axis=true))
```

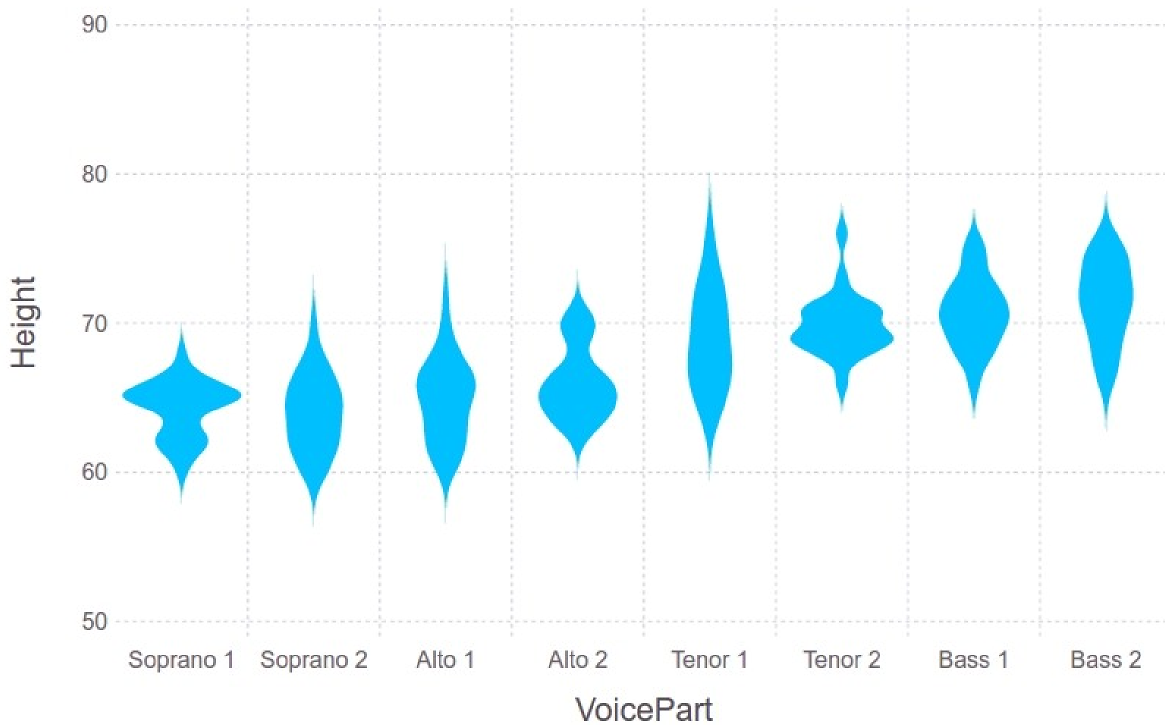


```
plot(dataset("datasets", "iris"), x="SepalLength", y="SepalWidth",
     yintercept=[2.5, 4.0], Geom.point, Geom.hline,
     xintercept=[5.0, 7.0], Geom.point, Geom.vline)
```

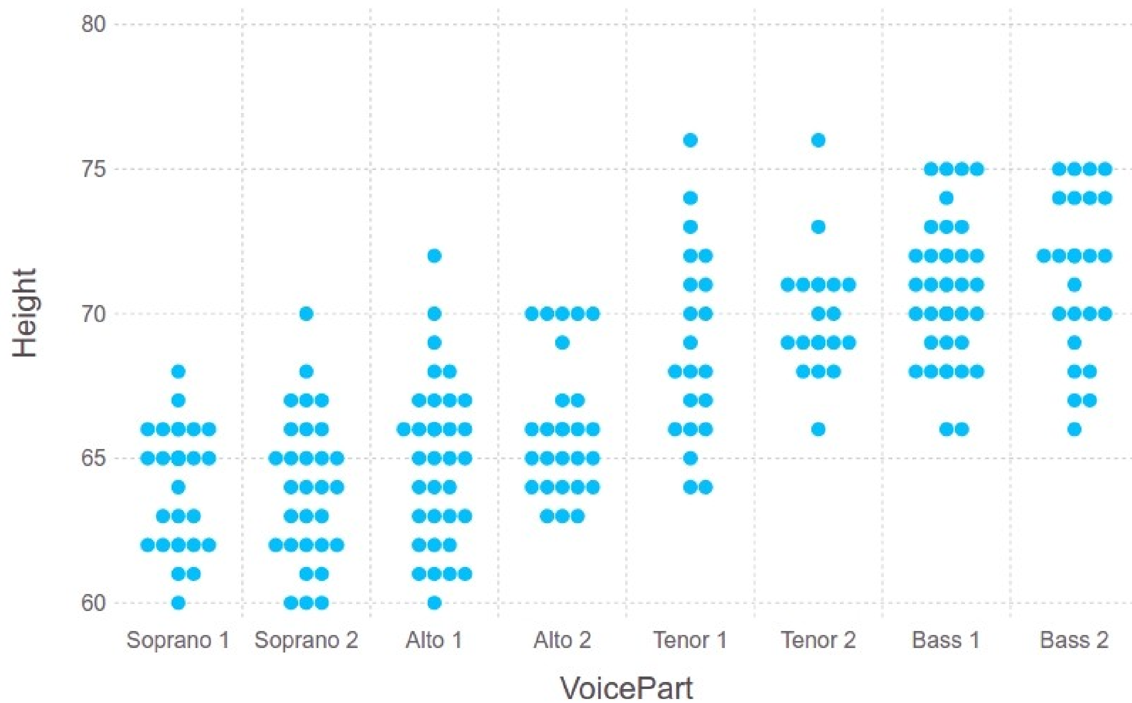




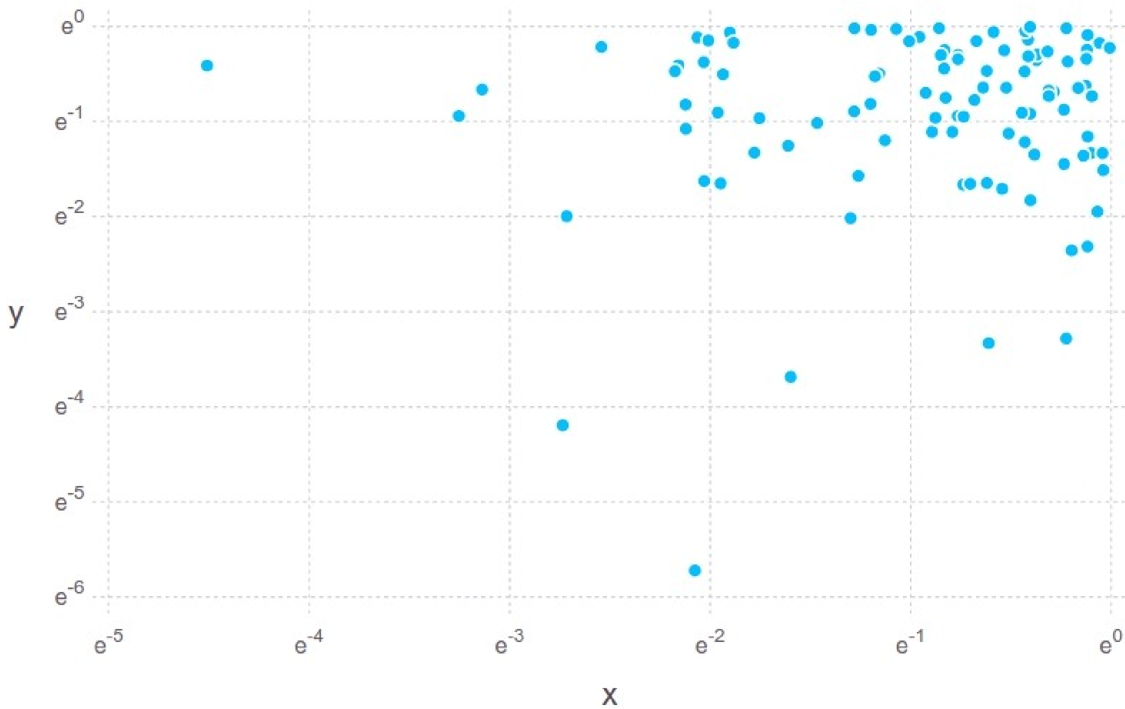
```
plot(dataset("lattice", "singer"),
    x="VoicePart", y="Height", Geom.violin)
```



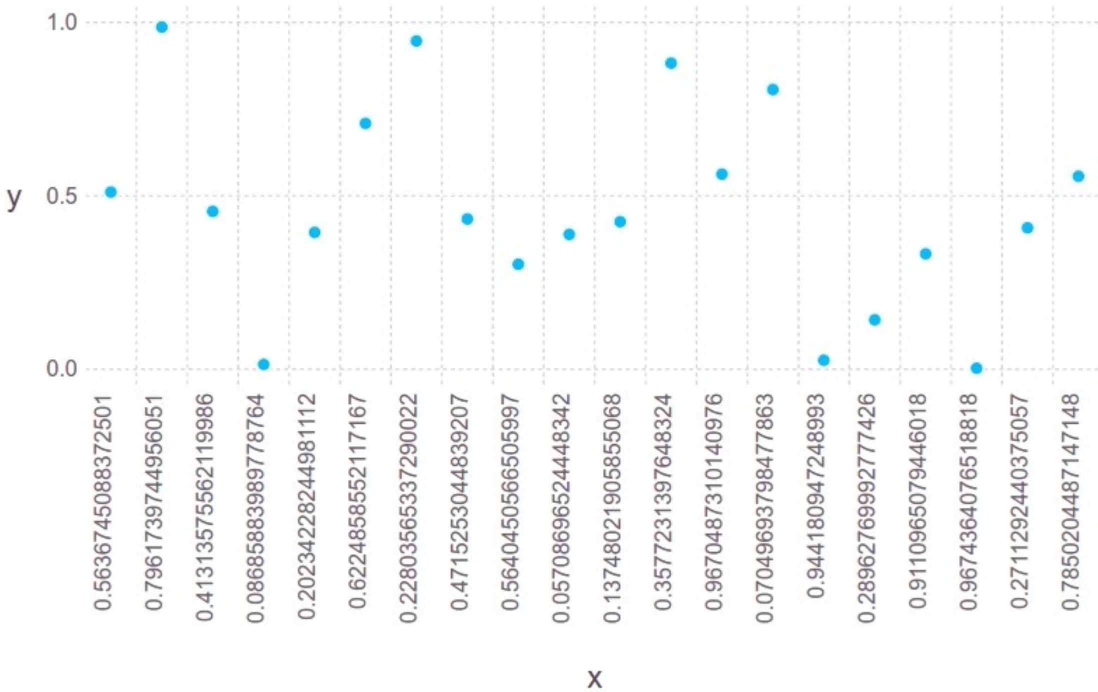
```
# Binding categorial data to x
plot(dataset("lattice", "singer"),
    x="VoicePart", y="Height", Geom.beeswarm)
```



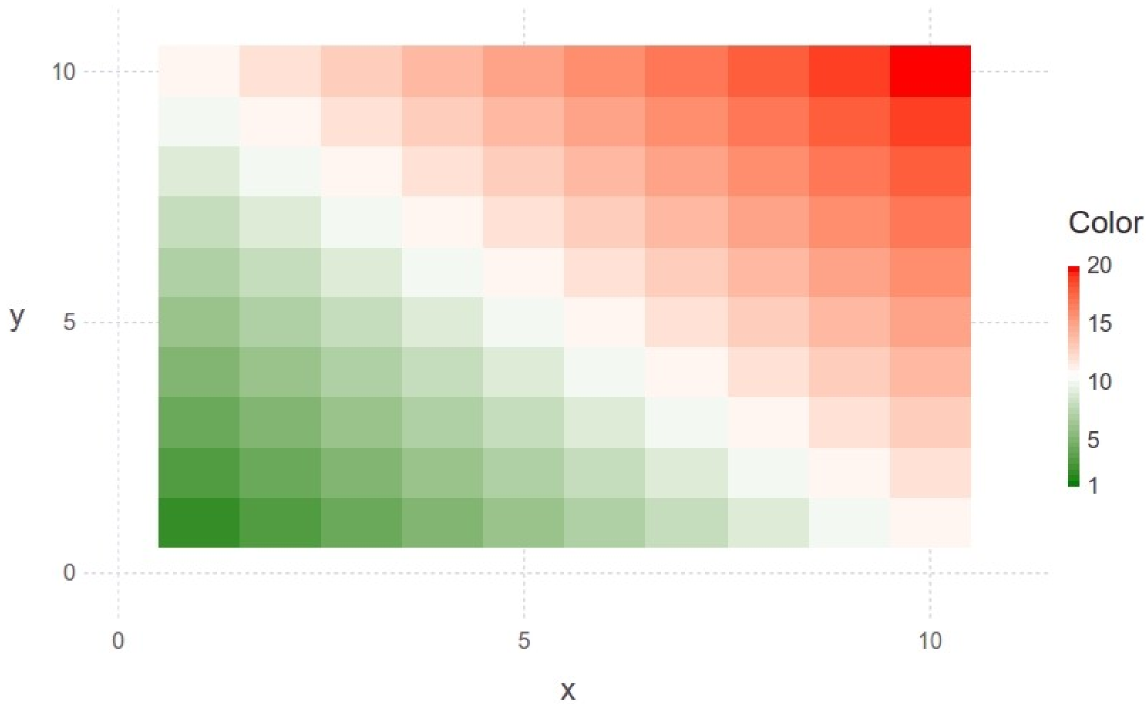
```
# Transform both dimensions
plot(x=rand(100), y=rand(100),
    Scale.x_log, Scale.y_log)
```



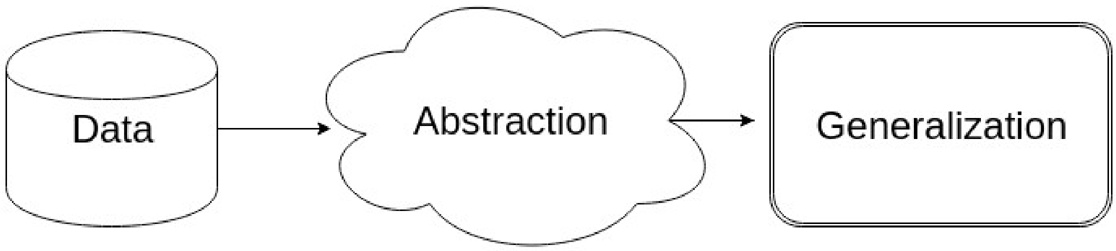
```
# Treat numerical y data as categories
plot(x=rand(20), y=rand(20),
    Scale.x_discrete)
```

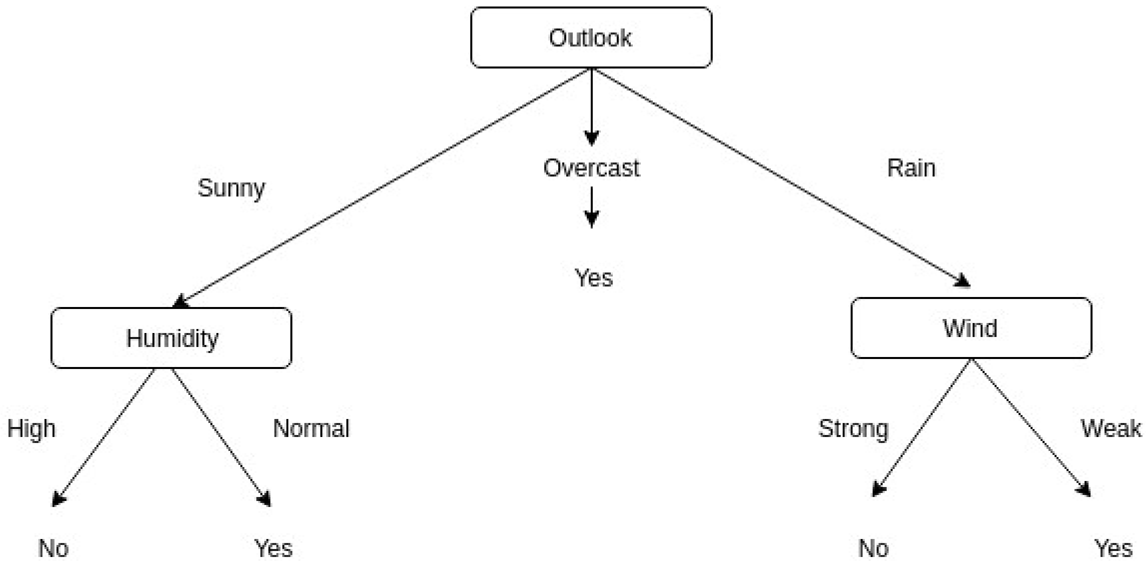


```
using Colors
x = repeat([1:10], inner=[10])
y = repeat([1:10], outer=[10])
plot(x=x,y=y,color=x+y, Geom.rectbin,
     Scale.ContinuousColorScale(Scale.lab_gradient(colorant"green",
                                                   colorant"white",
                                                   colorant"red")))
```



Chapter 6: Supervised Machine Learning





$$\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \, log_2(p_i)$$

$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2)$$

```
# Fit regression model
regr_1 = DecisionTreeRegressor()
regr_2 = DecisionTreeRegressor(pruning_purity_threshold=0.05)
```

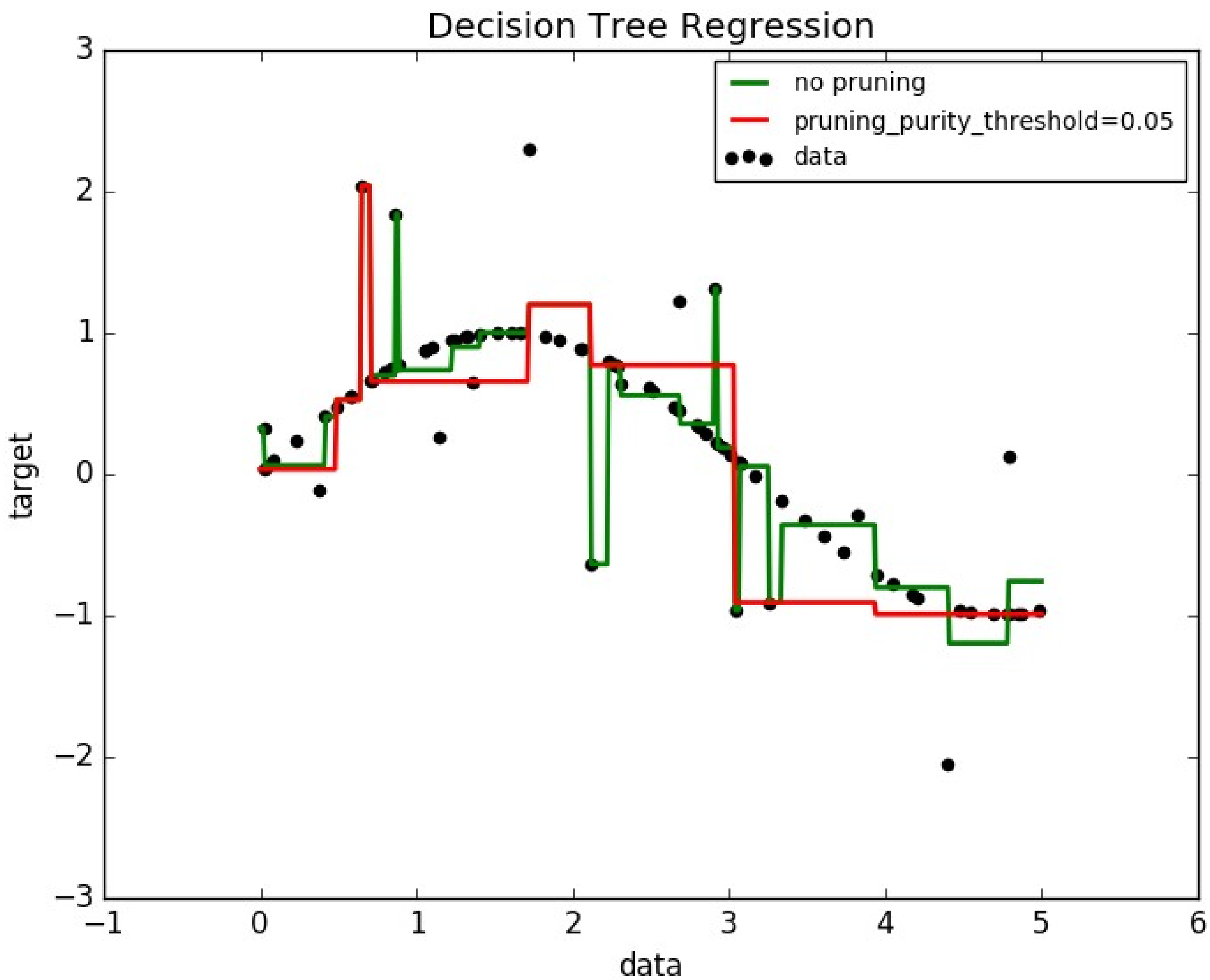
```
DecisionTree.DecisionTreeRegressor(Nullable(0.05),5,0,#undef)
```

```
fit!(regr_1, XX, y)
```

```
DecisionTree.DecisionTreeRegressor(Nullable{Float64}(),5,0,Decision Tree
Leaves: 25
Depth:  8)
```

```
fit!(regr_2, XX, y)
```

```
DecisionTree.DecisionTreeRegressor(Nullable(0.05),5,0,Decision Tree
Leaves: 6
Depth:  4)
```



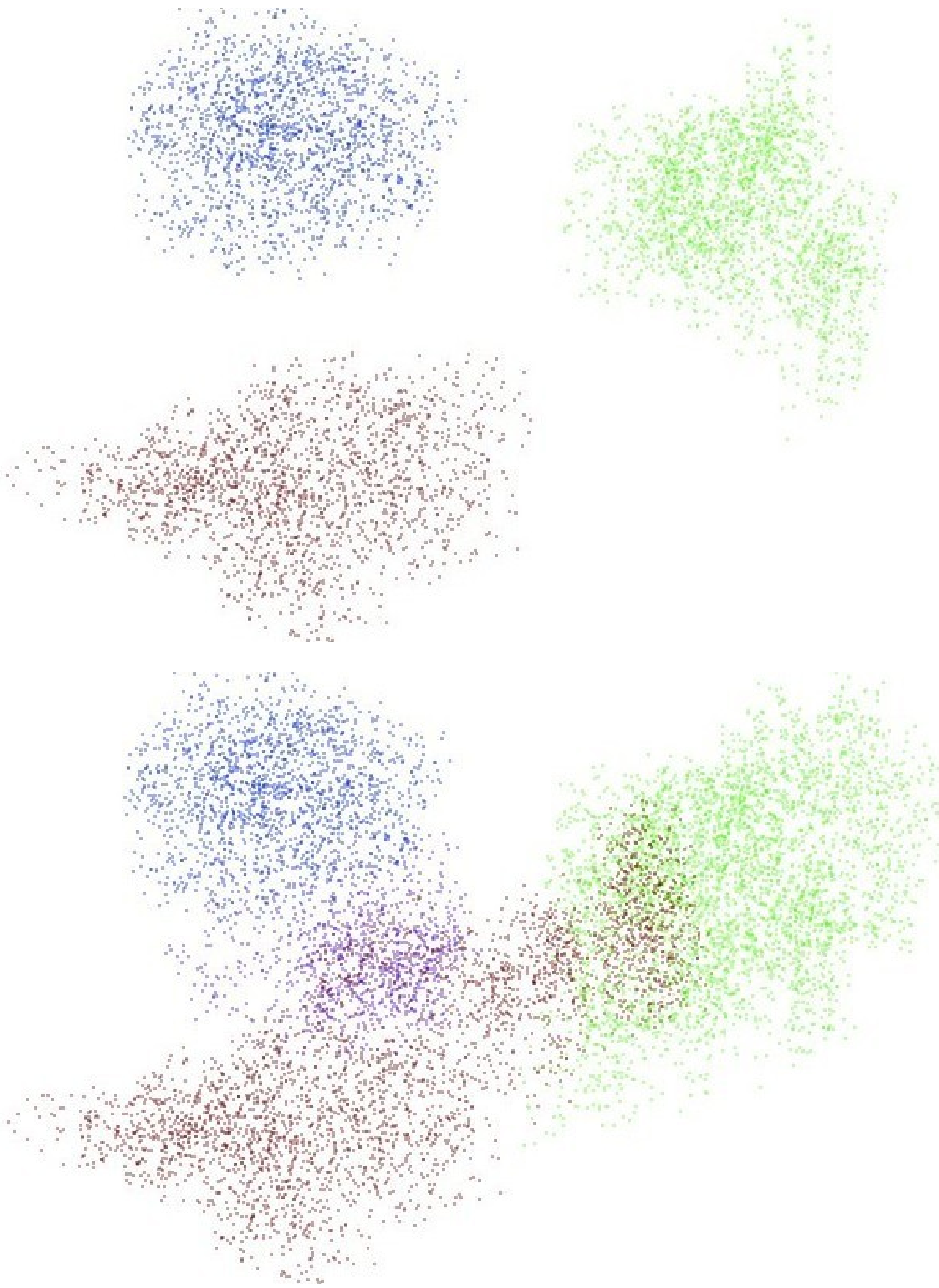
$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

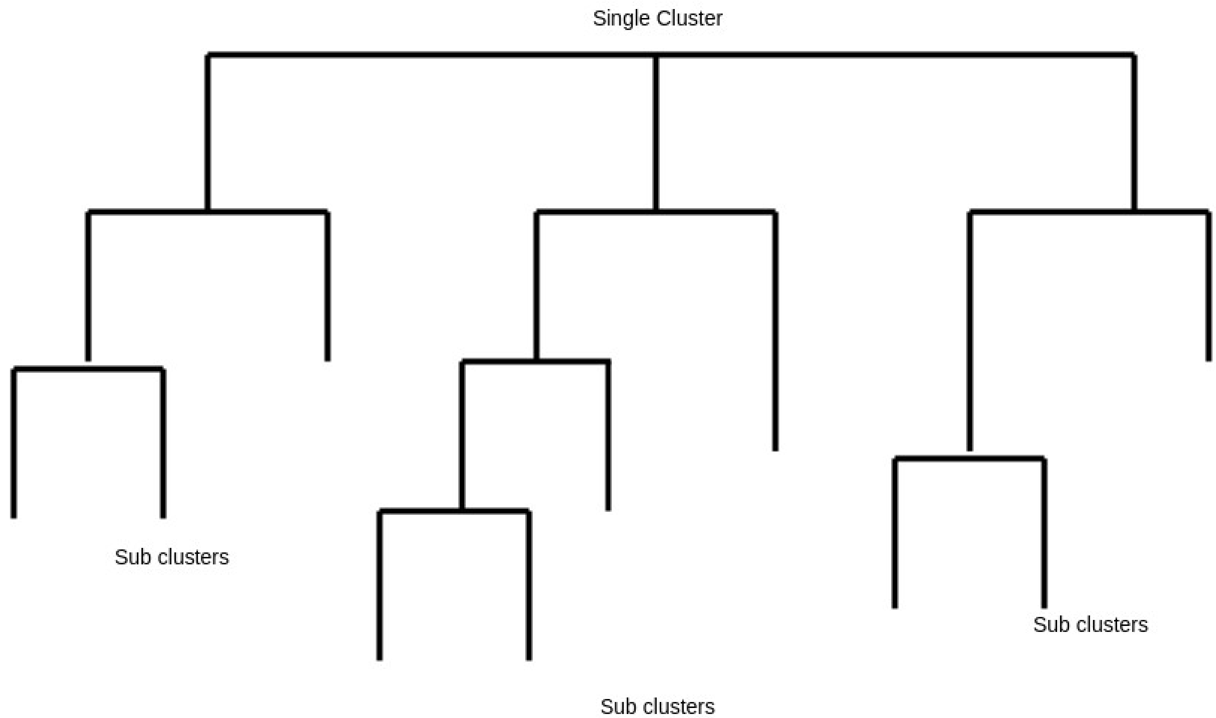
```
# how much data use for training
train_frac = 0.9
k = int(floor(train_frac * n))
idxs = randperm(n)
train = idxs[1:k]
test = idxs[k+1:end]
```

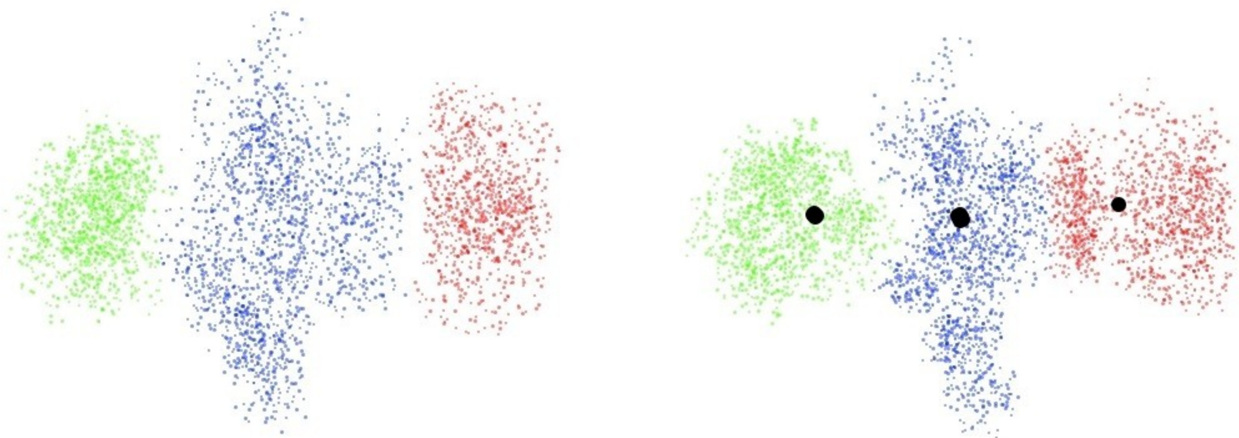
```
model = GaussianNB(unique(y), p)
fit(model, X[:, train], y[train])

accuracy = countnz(predict(model, X[:,test]).==
            y[test]) / countnz(test)

println("Accuracy: $accuracy")
```

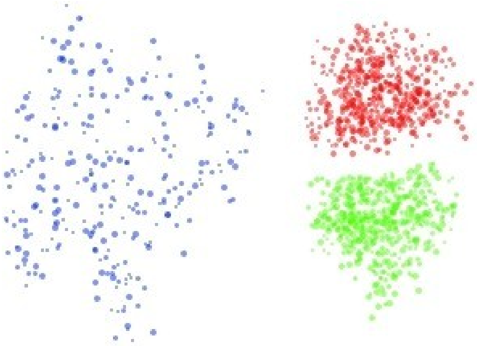
```
Accuracy: 1.0
```

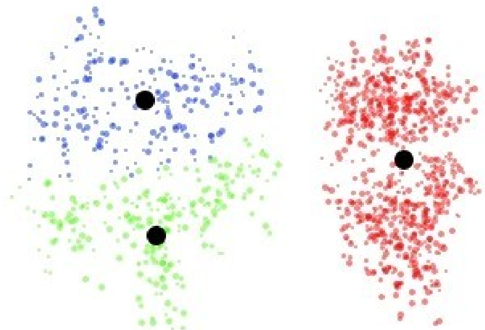
Chapter 7: Unsupervised Machine Learning

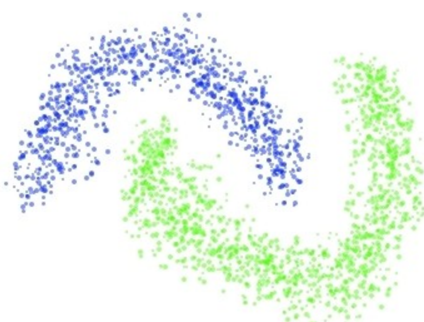


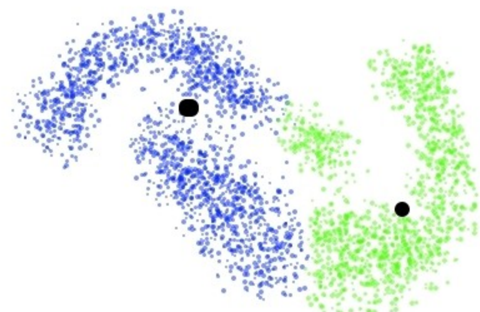

Single Cluster
Sub clusters
Sub clusters
Sub clusters



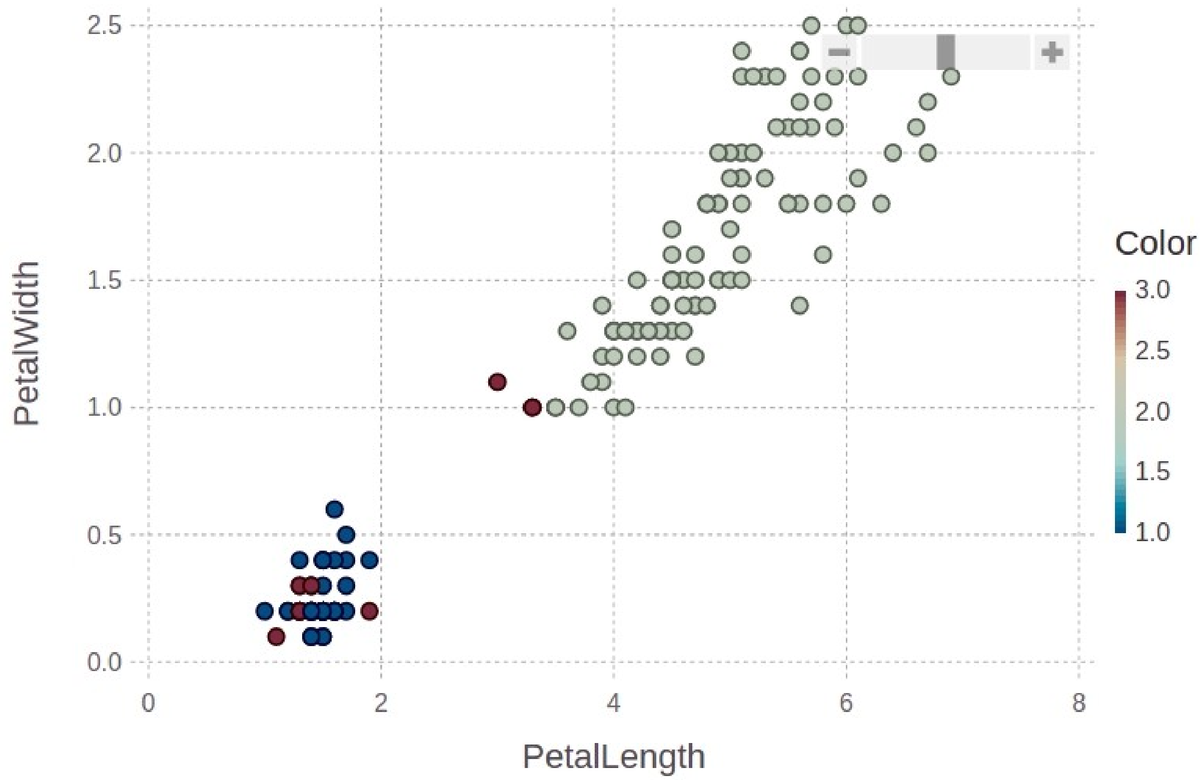


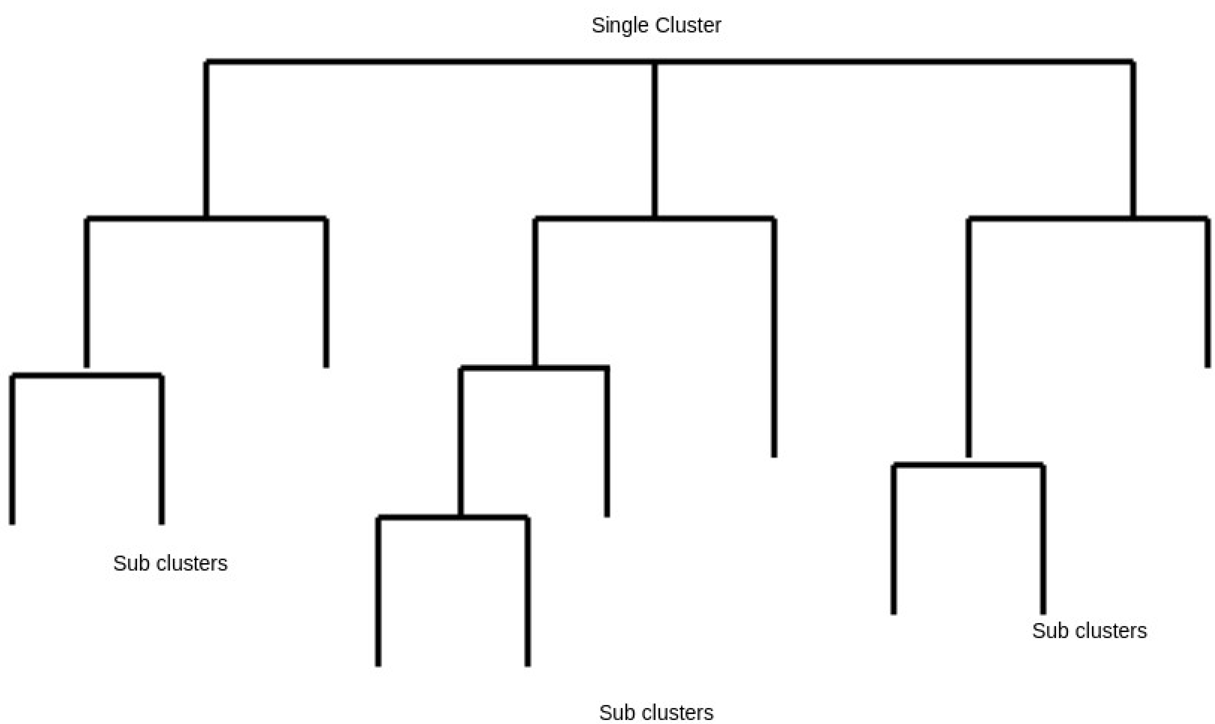


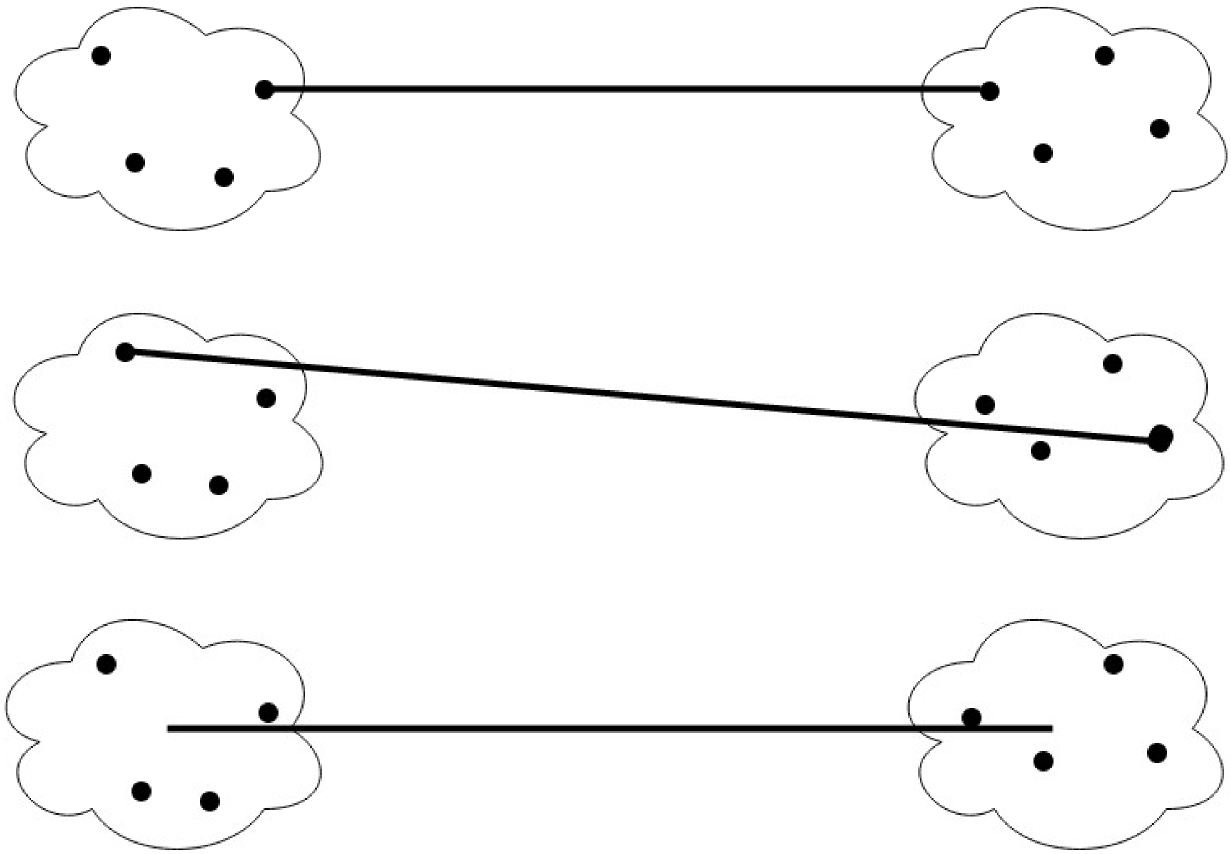


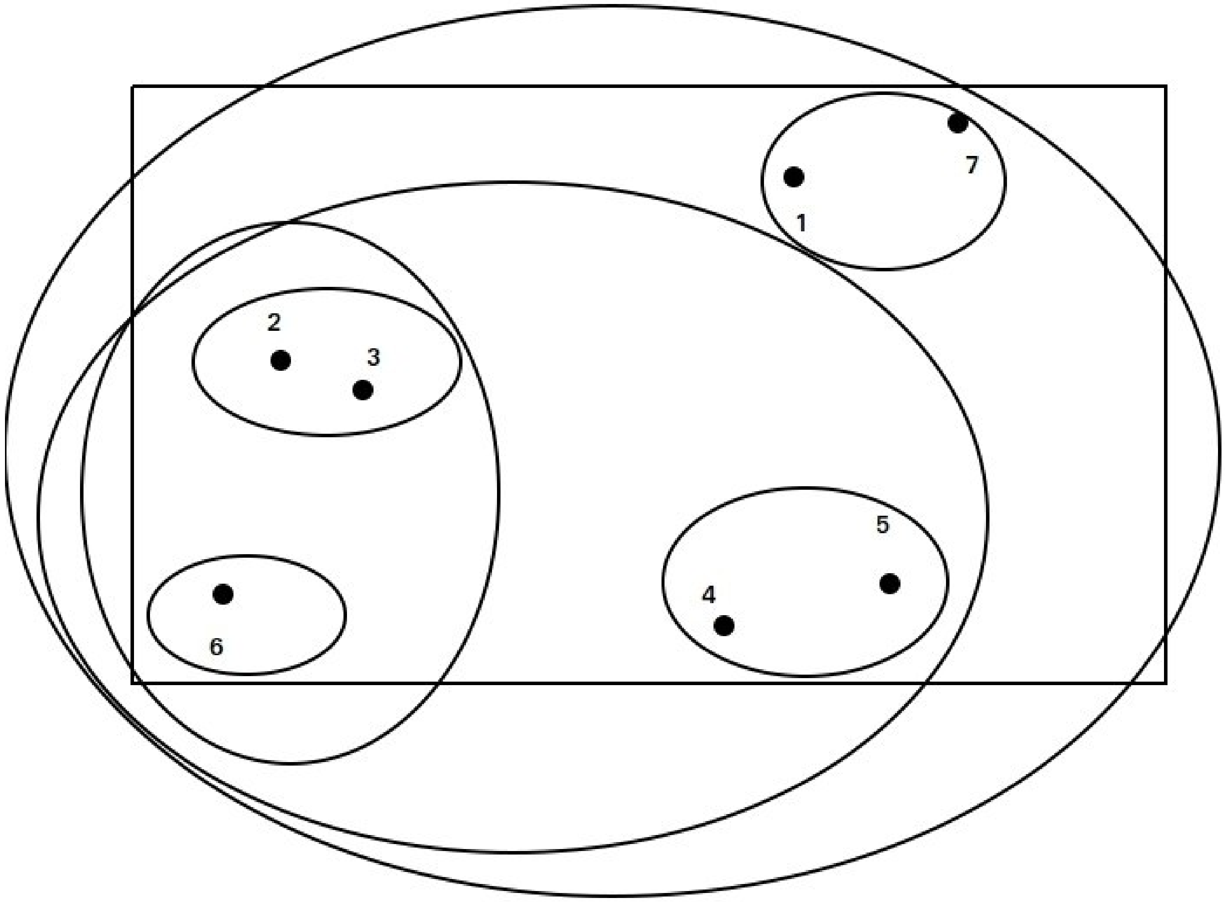


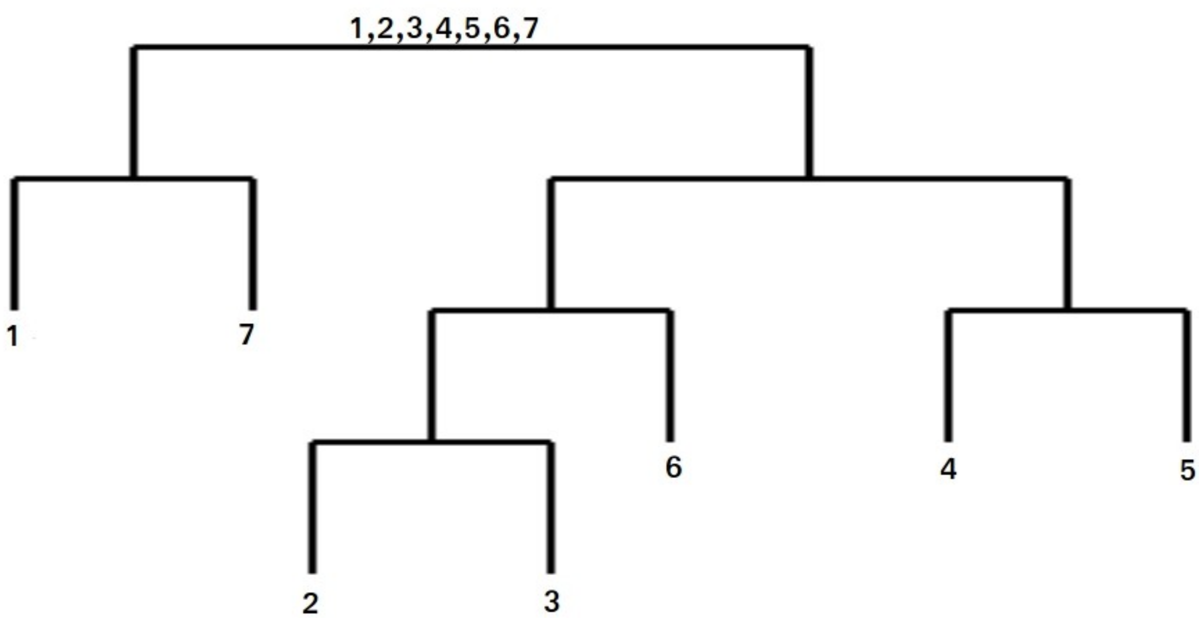
```
using Clustering
using Gadfly
iris = dataset("datasets", "iris")
features = array(iris[:, 1:4])'
# group the data onto 3 clusters
result = kmeans( features, 3 )
plot(iris, x = "PetalLength", y = "PetalWidth",
         color = result.assignments, Geom.point)
```


2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
PetalWidth
PetalLength
Color
3.0
2.5
2.0
1.5
1.0


Single Cluster
Sub clusters
Sub clusters
Sub clusters



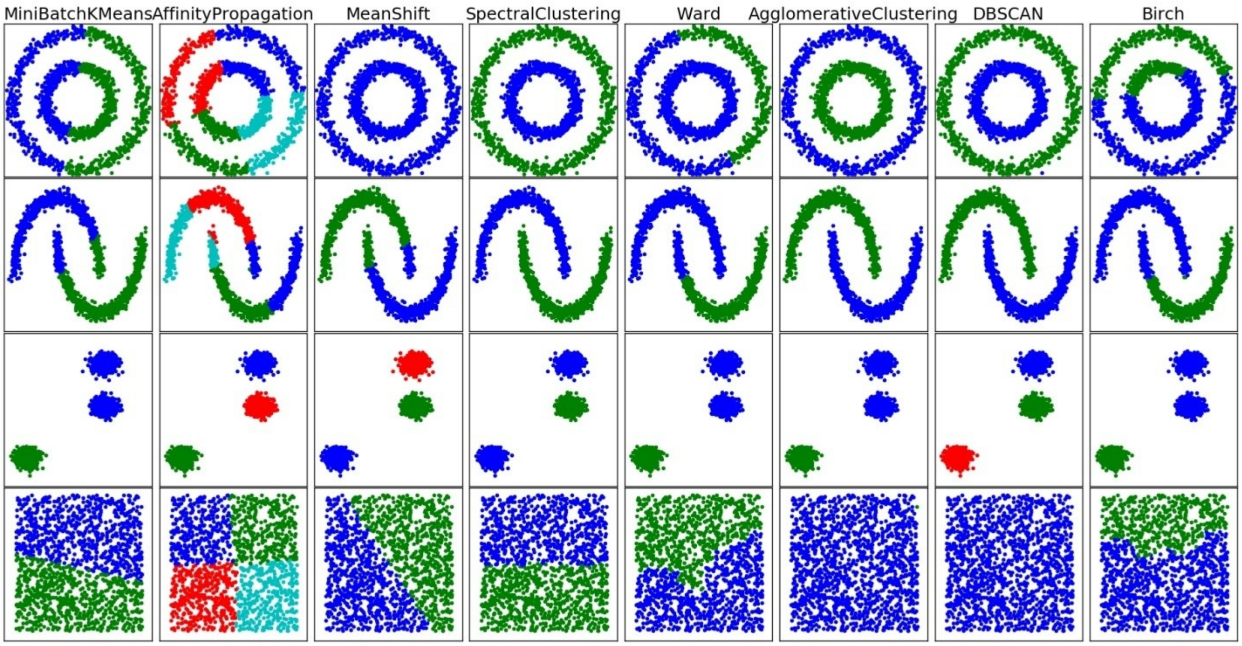

1
7
2
3
6
4
5


1,2,3,4,5,6,7
1
7
6
4
5
2
3

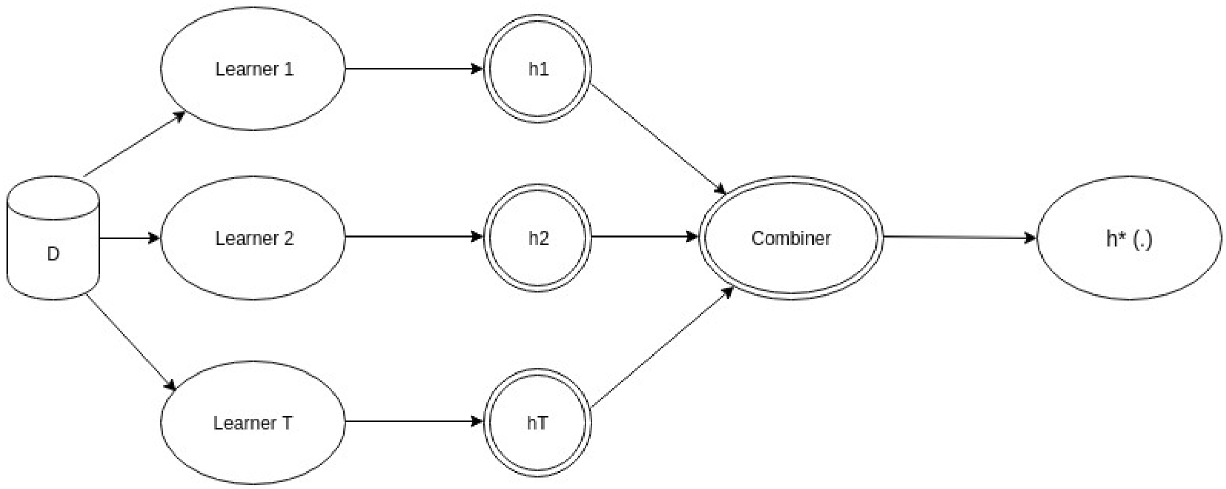
```julia
for (i_dataset, dataset) in enumerate(datasets)

    X, y = dataset
    # normalize dataset for easier parameter selection
    X = fit_transform!(StandardScaler(), X)

    # estimate bandwidth for mean shift
    bandwidth = estimate_bandwidth(X, quantile=0.3)

    # connectivity matrix for structured Ward
    connectivity = kneighbors_graph(X, n_neighbors=10,
                        include_self=false)[:todense]()
```

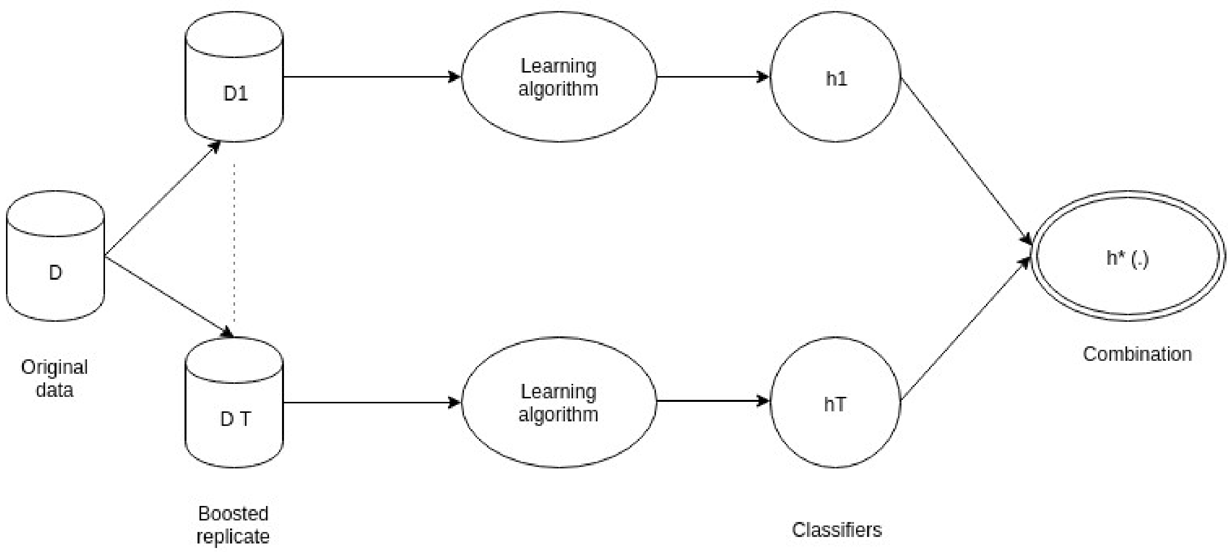
```julia
# PyCall does not support numpy sparse matrices
# make connectivity symmetric
connectivity = 0.5 * (connectivity + connectivity')

# create clustering estimators
ms = MeanShift(bandwidth=bandwidth, bin_seeding=true)
two_means = MiniBatchKMeans(n_clusters=2)
ward = AgglomerativeClustering(n_clusters=2, linkage="ward",
                               connectivity=connectivity)
spectral = SpectralClustering(n_clusters=2,
                              eigen_solver="arpack",
                              affinity="nearest_neighbors")
```

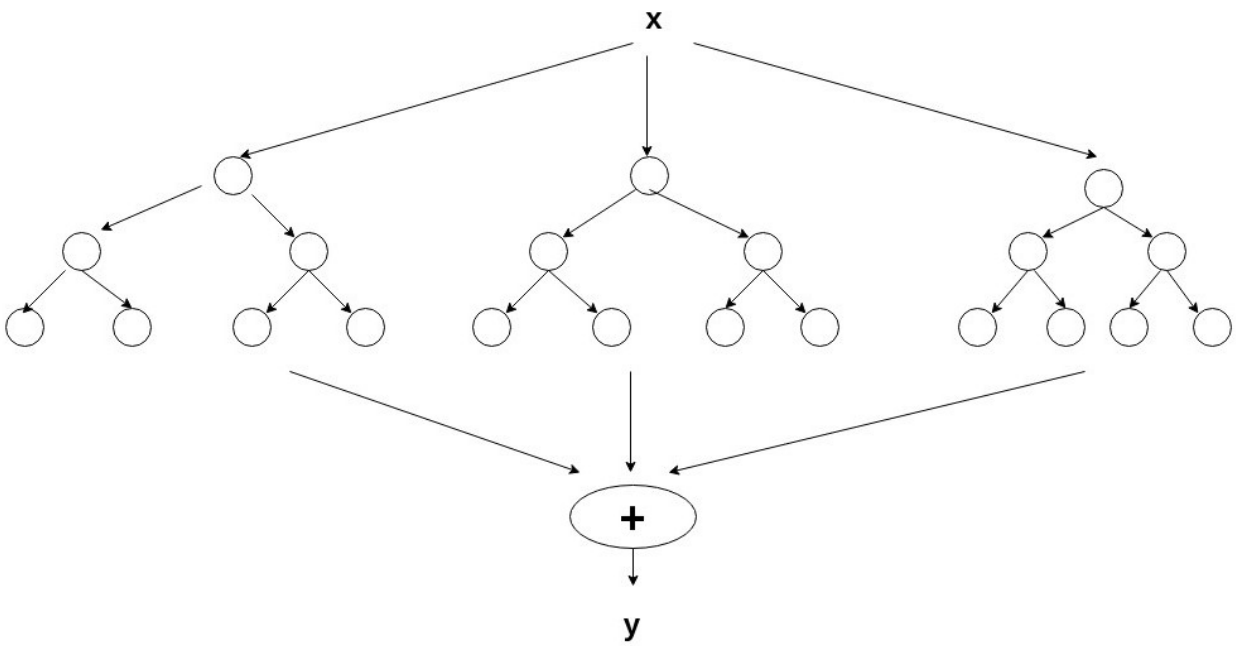
```julia
dbscan = DBSCAN(eps=.2)
affinity_propagation = AffinityPropagation(damping=.9, preference=-200)

average_linkage = AgglomerativeClustering(
    linkage="average", affinity="cityblock", n_clusters=2,
    connectivity=connectivity)

birch = Birch(n_clusters=2)
clustering_algorithms = [
    two_means, affinity_propagation, ms, spectral, ward, average_linkage,
    dbscan, birch]
```


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
AgglomerativeClustering
DBSCAN
Birch

Chapter 8: Creating Ensemble Models

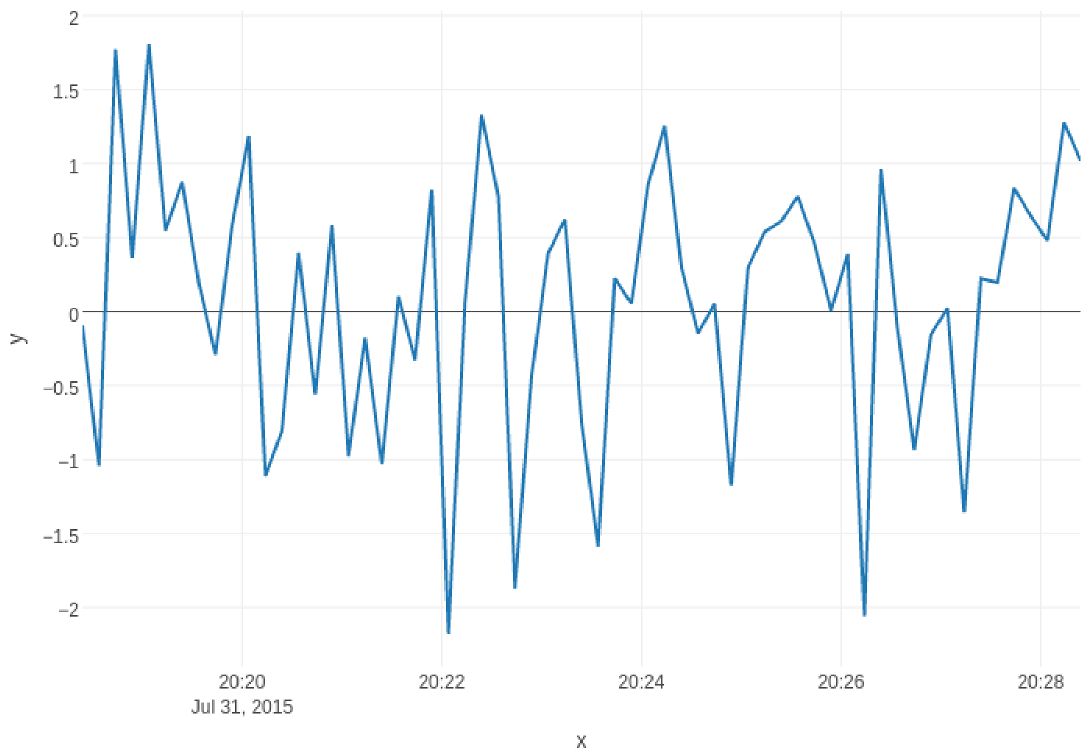


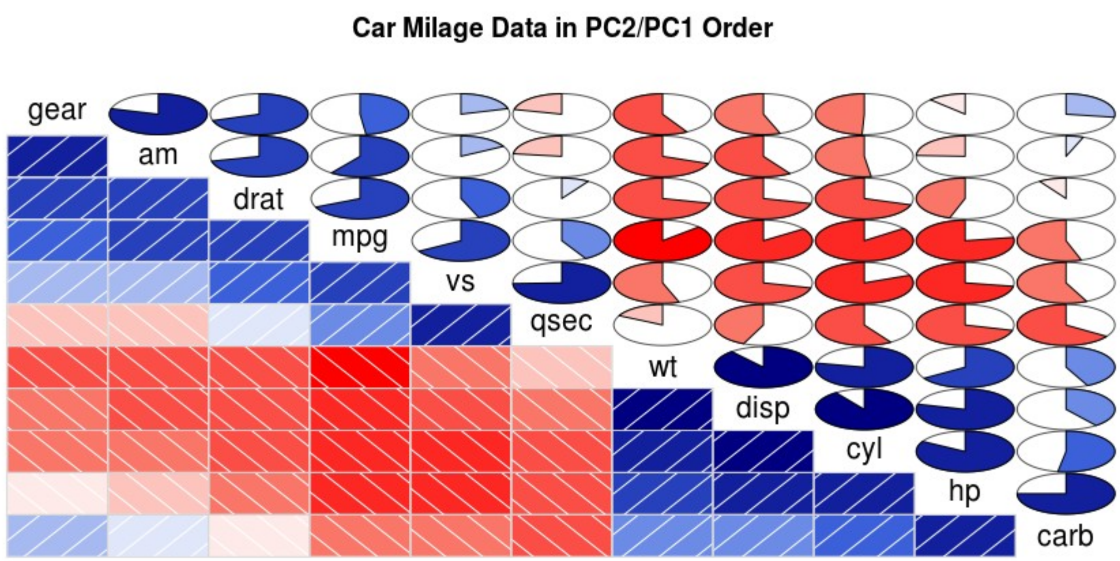




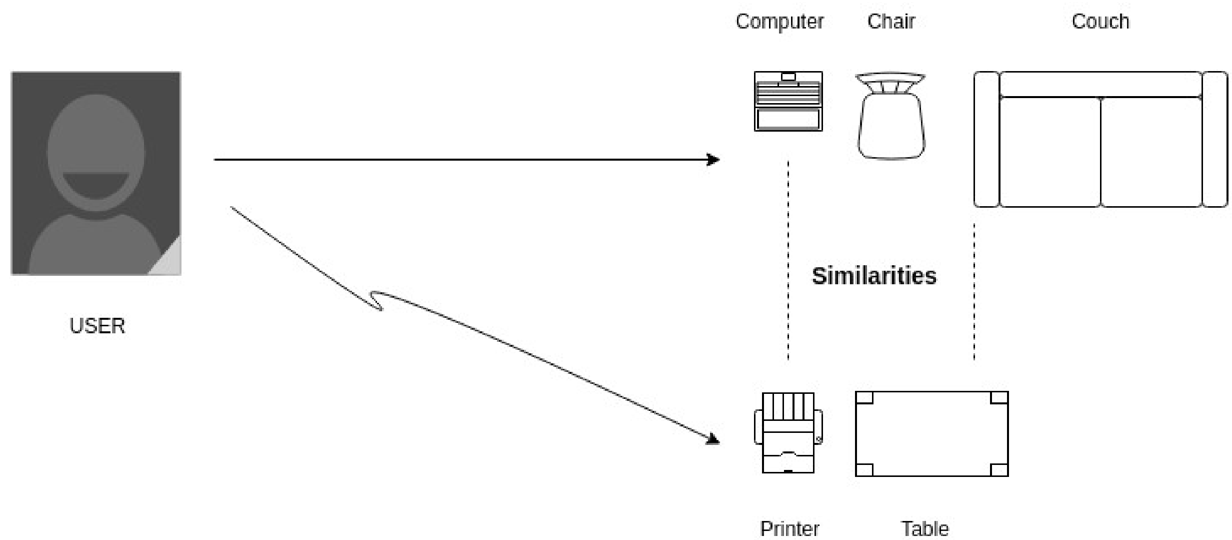

```
Feature 3, Threshold 3.0
L-> setosa : 50/50
R-> Feature 4, Threshold 1.8
    L-> Feature 3, Threshold 5.0
        L-> versicolor : 47/48
        R-> Feature 4, Threshold 1.6
            L-> virginica : 3/3
            R-> Feature 1, Threshold 7.2
                L-> versicolor : 2/2
                R-> virginica : 1/1
    R-> Feature 3, Threshold 4.9
        L-> Feature 1, Threshold 6.0
            L-> versicolor : 1/1
            R-> virginica : 2/2
        R-> virginica : 43/43
```

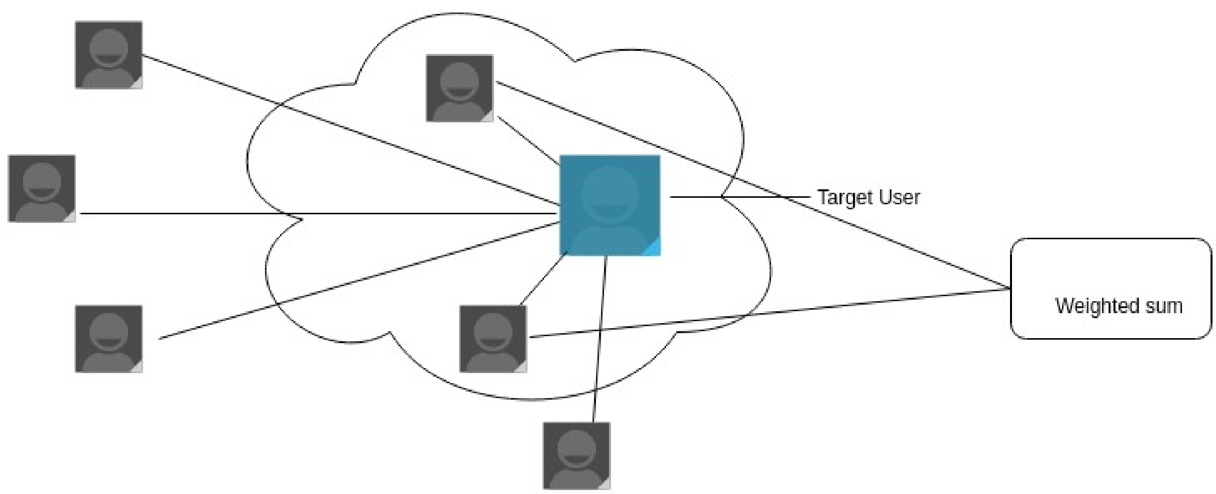
Chapter 9: Time Series





Chapter 10: Collaborative Filtering and Recommendation System



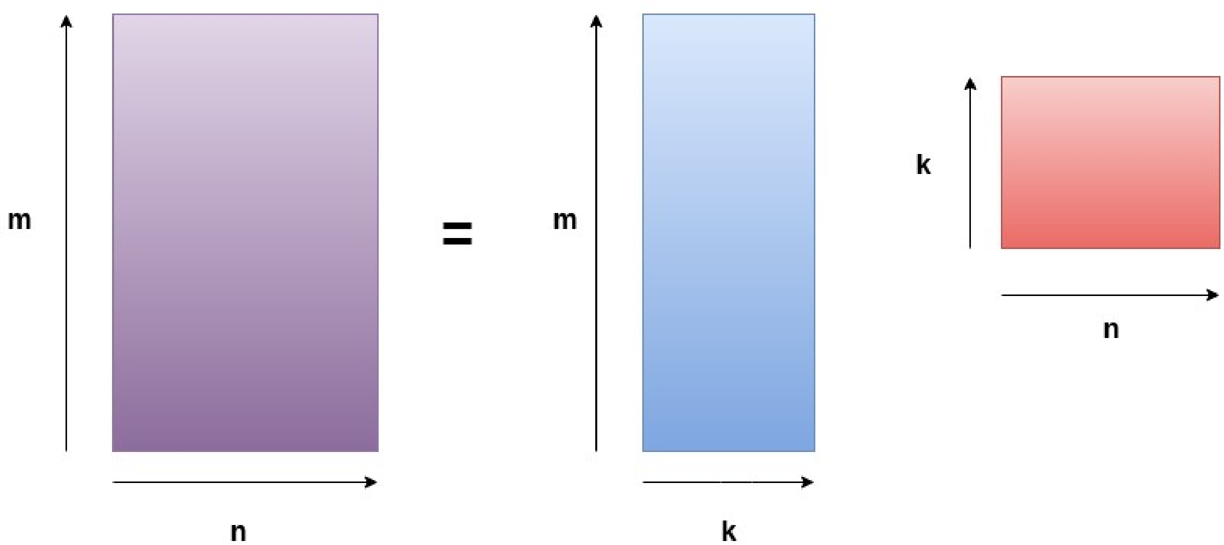


$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}}$$

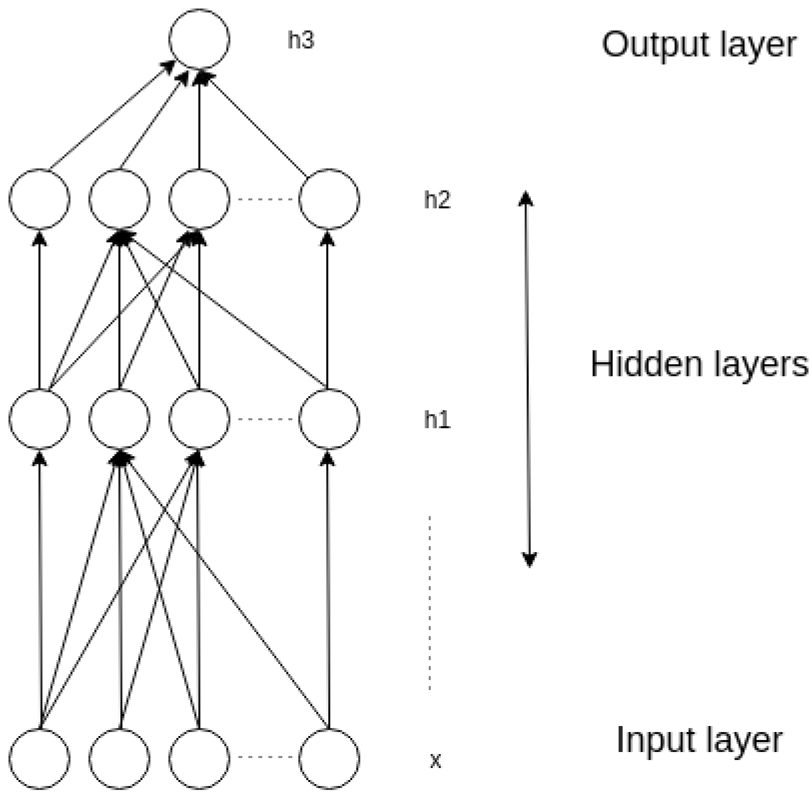
$$s(u,v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\|_2 \|\mathbf{r}_v\|_2} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2}\sqrt{\sum_i r_{v,i}^2}}$$

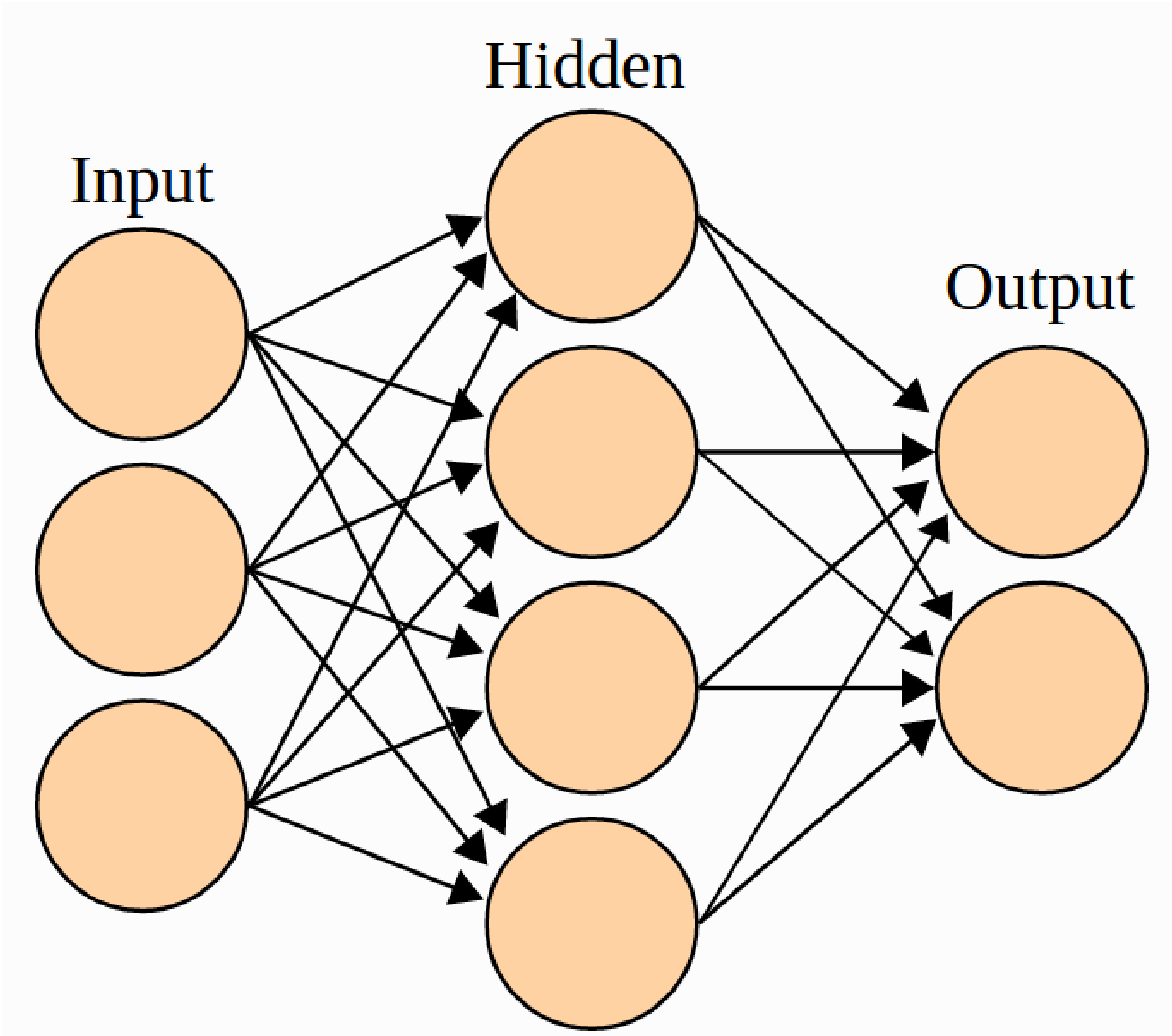
$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u,u')(r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} |s(u,u')|}$$

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j)(r_{u,j} - b_{u,i})}{\sum_{j \in S} |s(i,j)|} + b_{u,i}$$



Chapter 11: Introduction to Deep Learning




Hidden
Input
Output

Graph for (1-e-2*x)/(1+e-2*x)