

Working with Data in Julia

John Myles White

January 15, 2013

```
julia> Pkg.add("DataFrames")  
julia> Pkg.add("RDatasets")  
julia> using DataFrames  
julia> using RDatasets
```

How do we cope with missing data?

```
v = [x1, x2, x3, x4, x5]
```

```
mean(v)
```

```
v = [0.5, 0.6, 0.7, 0.8, 0.9]
```

```
mean(v)
```

```
v = [0.5, 0.6, 0.7, NA, 0.9]
```

```
mean(v)
```

The NA type:

- ▶ Represents a missing value
 - ▶ Like NULL in some systems
- ▶ Poisons other values
 - ▶ Like NaN for floating point numbers

```
julia> 1 + NA  
NA
```

```
julia> 1 > NA  
NA
```

```
julia> isna(NA)  
true
```


- ▶ `dataArray{T}` adds NA's to `Array{T}`
- ▶ `dataArray{T}` can store T or NA

```
julia> dv = DataArray([1, 2, 3])
```

```
3-element Int64 DataArray:
```

```
1
```

```
2
```

```
3
```

```
julia> dv[1] = NA
```

```
NA
```

```
julia> join(dv, "::")
```

```
"NA::2::3"
```

Convenience constructors:

- ▶ `datazeros()`
- ▶ `dataones()`
- ▶ `datafalses()`
- ▶ `datatrues()`
- ▶ `dataeye()`
- ▶ `datadiagm()`

```
julia> dm = dataeye(2)
2x2 Float64 DataArray:
 1.0  0.0
 0.0  1.0
```

```
julia> svd(dm)
(
 2x2 Float64 Array:
 1.0  0.0
 0.0  1.0,
 [1.0, 1.0],
 2x2 Float64 Array:
 1.0  0.0
 0.0  1.0)
```

Convenience converters:

- ▶ `dataint()`
- ▶ `datafloat()`
- ▶ `databool()`

```
julia> dataint([1.0, 2.0, 3.0])
```

```
3-element Int64 DataArray:
```

```
1
```

```
2
```

```
3
```

```
julia> databool([1, 0, 1])
```

```
3-element Bool DataArray:
```

```
true
```

```
false
```

```
true
```

How do we store data efficiently?

`PooledDataArray{T}` compresses `DataArray{T}`


```
julia> pda = PooledDataArray(["AA", "BB", "BB", "AA"])  
4-element ASCIIString PooledDataArray:
```

```
"AA"
```

```
"BB"
```

```
"BB"
```

```
"AA"
```

```
julia> pda[1] = NA  
NA
```

```
julia> levels(pda)
3-element ASCIIString DataArray:
 "AA"
 "BB"
 NA
```

```
julia> pda.refs  
4-element UInt16 Array:  
 0x0000  
 0x0002  
 0x0002  
 0x0001
```

```
julia> pda.pool  
2-element ASCIIString Array:  
 "AA"  
 "BB"
```

How do we cope with heterogeneous data?

Name	Height	Weight	Gender
John Smith	73.0	NA	Male
Jane Doe	68.0	130	Female

```
df = DataFrame()  
df["Name"] = DataVector["John Smith", "Jane Doe"]  
df["Height"] = DataVector[73.0, 68.0]  
df["Weight"] = DataVector[NA, 130]  
df["Gender"] = DataVector["Male", "Female"]
```

```
julia> df
```

```
2x4 DataFrame:
```

	Name	Height	Weight	Gender
[1,]	"John Smith"	73.0	NA	"Male"
[2,]	"Jane Doe"	68.0	130	"Female"

```
julia> df["Weight"]  
2-element Int64 DataArray:  
  NA  
 130
```


- ▶ A DataFrame is a list of DataVector's
- ▶ DataFrame's allow mixed indexing:
 - ▶ Columns by number
 - ▶ Columns by name
 - ▶ Rows + Columns by number + number
 - ▶ Rows + Columns by number + name

```
julia> df = DataFrame()
```

```
0x0 DataFrame:
```

```
julia> df["A"] = dataones(3)
```

```
3-element Float64 DataArray:
```

```
1.0
```

```
1.0
```

```
1.0
```

```
julia> df[2] = datazeros{Int, 3}
```

```
3-element Int64 DataArray:
```

```
0
```

```
0
```

```
0
```

```
julia> df
3x2 DataFrame:
      A  x2
[1,]  1.0  0
[2,]  1.0  0
[3,]  1.0  0

julia> df[1, 1]
1.0

julia> df[1, "A"]
1.0
```

```
julia> path = joinpath(julia_pkgdir(),  
                        "DataFrames",  
                        "test", "data", "types.csv")  
"/Users/johnmyleswhite/.julia/DataFrames/te...
```

```
julia> types = read_table(path)
```

```
3x5 DataFrame:
```

	IntColumn	IntlikeColumn	FloatColumn	BoolColumn...
[1,]	1	1.0	3.1	true...
[2,]	2	7.0	-3.1e8	false...
[3,]	-1	7.0	-3.1e-8	false...

```
julia> head(types, 2)
```

```
3x5 DataFrame:
```

	IntColumn	IntlikeColumn	FloatColumn	BoolColumn...
[1,]	1	1.0	3.1	true...
[2,]	2	7.0	-3.1e8	false...

```
julia> tail(types, 2)
```

```
3x5 DataFrame:
```

	IntColumn	IntlikeColumn	FloatColumn	BoolColumn...
[2,]	2	7.0	-3.1e8	false...
[3,]	-1	7.0	-3.1e-8	false...

```
julia> types[1:3, 1:5]
```

```
3x5 DataFrame:
```

	IntColumn	IntlikeColumn	FloatColumn	BoolColumn...
[1,]	1	1.0	3.1	true...
[2,]	2	7.0	-3.1e8	false...
[3,]	-1	7.0	-3.1e-8	false...


```
julia> iris = data("datasets", "iris")
```

```
150x6 DataFrame:
```

		Sepal.Length	Sepal.Width	Petal.Length...
[1,]	1	5.1	3.5	1.4...
[2,]	2	4.9	3.0	1.4...
	...			
[149,]	149	6.2	3.4	5.4...
[150,]	150	5.9	3.0	5.1...

```
julia> RDatasets.datasets()
570-element Any Array:
 ["COUNT", "affairs"]
 ["COUNT", "azdrg112"]
 ...
 ["vcd", "WeldonDice"]
 ["vcd", "WomenQueue"]
```

```
julia> colnames(iris)
6-element Union{ASCIIString,UTF8String} Array:
 ""
 "Sepal.Length"
 "Sepal.Width"
 "Petal.Length"
 "Petal.Width"
 "Species"
```

```
julia> coltypes(iris)
```

```
6-element Any Array:
```

```
Int64
```

```
Float64
```

```
Float64
```

```
Float64
```

```
Float64
```

```
UTF8String
```

```
julia> clean_colnames!(iris)
```

```
julia> colnames(iris)
```

```
6-element Union{ASCIIString,UTF8String} Array:
```

```
""
```

```
"Sepal_Length"
```

```
"Sepal_Width"
```

```
"Petal_Length"
```

```
"Petal_Width"
```

```
"Species"
```

```
julia> size(iris)
(150,6)
```

```
julia> nrow(iris)
150
```

```
julia> ncol(iris)
6
```

```
julia> vcat(iris, iris)
```

```
...
```

```
julia> hcat(iris, iris)
```

```
...
```

```
julia> rbind(iris, iris)
```

```
...
```

```
julia> cbind(iris, iris)
```

```
...
```

```
julia> iris[1, 1] = NA
NA
```

```
julia> head(iris)
6x6 DataFrame:
```

		Sepal_Length	Sepal_Width	Petal_Length...
[1,]	NA	5.1	3.5	1.4...
[2,]	2	4.9	3.0	1.4...
[3,]	3	4.7	3.2	1.3...
[4,]	4	4.6	3.1	1.5...
[5,]	5	5.0	3.6	1.4...
[6,]	6	5.4	3.9	1.7...


```
julia> complete_cases(iris)
150-element Bool Array:
 false
  true
   ...
  true
  true
```

```
julia> iris[complete_cases(iris), :]  
...
```

```
julia> complete_cases!(iris)
```

```
julia> head(iris)
```

6x6 DataFrame:

		Sepal_Length	Sepal_Width	Petal_Length...
[1,]	2	4.9	3.0	1.4...
[2,]	3	4.7	3.2	1.3...
[3,]	4	4.6	3.1	1.5...
[4,]	5	5.0	3.6	1.4...
[5,]	6	5.4	3.9	1.7...
[6,]	7	4.6	3.4	1.4...

```
julia> any(duplicated(iris))  
false
```

```
julia> any(duplicated(rbind(iris, iris)))  
true
```

```
julia> new_iris = rbind(iris, iris)
...
```

```
julia> drop_duplicates!(new_iris)
```

```
julia> nrow(new_iris)
149
```

```
julia> vector(iris["Species"])  
149-element UTF8String Array:  
 "setosa"  
 "setosa"  
 ...  
 "virginica"  
 "virginica"
```

```
julia> vector(iris["Species"], Any)
```

```
149-element Any Array:
```

```
"setosa"
```

```
"setosa"
```

```
...
```

```
"virginica"
```

```
"virginica"
```

```
julia> matrix(iris)
julia> matrix(iris[:, 2:3])
julia> matrix(iris[:, 1:3])
julia> matrix(iris[:, 1:3], Any)
```



```
julia> with(iris, :(Petal_Length .* Petal_Width))  
149-element Float64 DataArray:  
 0.28  
 0.26  
 ...  
12.42  
 9.18
```

```
julia> within!(iris,  
              :(Petal_Area = Petal_Length .* Petal_Width))
```

```
julia> head(iris)
```

6x7 DataFrame:

		Sepal_Length	Sepal_Width	Petal_Length...
[1,]	2	4.9	3.0	1.4...
[2,]	3	4.7	3.2	1.3...
[3,]	4	4.6	3.1	1.5...
[4,]	5	5.0	3.6	1.4...
[5,]	6	5.4	3.9	1.7...
[6,]	7	4.6	3.4	1.4...

Database operations on DataFrames:

- ▶ subset
- ▶ merge
- ▶ groupby

```
julia> subset(iris, :(Species .== "setosa"))  
julia> nrow(subset(iris, :(Species .== "setosa")))  
49
```

```
julia> df1 = DataFrame({"a" => [1, 2, 3],  
                        "b" => ["America", "Europe",  
                                "Africa"]})  
  
julia> df2 = DataFrame({"a" => [1, 2, 4],  
                        "c" => ["New World", "Old World",  
                                "New World"]})
```

```
julia> merge(df1, df2)
```

```
2x3 DataFrame:
```

	a	b	c
[1,]	1	"America"	"New World"
[2,]	2	"Europe"	"Old World"

```
julia> merge(df1, df2, "a", "inner")
```

```
2x3 DataFrame:
```

	a	b	c
[1,]	1	"America"	"New World"
[2,]	2	"Europe"	"Old World"

```
julia> merge(df1, df2, "a", "left")
```

```
3x3 DataFrame:
```

	a	b	c
[1,]	1	"America"	"New World"
[2,]	2	"Europe"	"Old World"
[3,]	3	"Africa"	NA


```
julia> merge(df1, df2, "a", "right")
```

3x3 DataFrame:

	a	b	c
[1,]	1	"America"	"New World"
[2,]	2	"Europe"	"Old World"
[3,]	NA	NA	"New World"

```
julia> merge(df1, df2, "a", "outer")
```

```
4x3 DataFrame:
```

	a	b	c
[1,]	1	"America"	"New World"
[2,]	2	"Europe"	"Old World"
[3,]	3	"Africa"	NA
[4,]	NA	NA	"New World"

The Split-Apply-Combine Strategy:

- ▶ Segment data into groups
- ▶ Apply a function to each group independently
- ▶ Combine results into a single DataFrame

```
julia> movies = data("ggplot2", "movies")  
...
```

```
julia> by(movies, "year", nrow)
```

```
113x2 DataFrame:
```

	year	x1
[1,]	1893	1
[2,]	1894	9
...		
[112,]	2004	1945
[113,]	2005	349

```
julia> subset(movies, :(year .== 1893))
```

```
1x25 SubDataFrame:
```

		title	year	length	budget...
[1,]	6076	"Blacksmith Scene"	1893	1	NA...

```
julia> by(movies, ["Action", "year"], nrow)
```

```
205x3 DataFrame:
```

	Action	year	x1
[1,]	0	1893	1
[2,]	0	1894	9
...			
[204,]	0	2005	306
[205,]	1	2005	43

Exercises

- ▶ Create some DataArray's
 - ▶ A DataArray containing the first five primes
 - ▶ A DataArray containing only NA's
 - ▶ A DataArray containing DataArray's

- ▶ Create some PooledDataArray's
 - ▶ A PooledDataArray of all strings
 - ▶ A PooledDataArray of repeated ComplexPair's

- ▶ Create some DataFrame's
 - ▶ A 4x3 DataFrame w/:
 - ▶ 4 String's
 - ▶ 4 Int's
 - ▶ 4 ComplexPair's

- ▶ Load more datasets from RDatasets:
 - ▶ Search using `RDatasets.datasets()`

- ▶ Do some Split-Apply-Combine Working on `iris`:
 - ▶ Find the median petal area
 - ▶ Find the variance of the petal area
 - ▶ Find the centroid of the sepal and petal dimensions

- ▶ Do some modeling with other packages
 - ▶ Clustering package
 - ▶ kNN package
 - ▶ DecisionTree package

```
Pkg.add("Clustering")  
using Clustering  
iris = data("datasets", "iris")  
k_means(matrix(iris[:, 2:5]), 3)  
iris["Cluster"] =  
    k_means(matrix(iris[:, 2:5]), 3).assignments  
by(iris, ["Cluster", "Species"], nrow)
```

```
Pkg.add("kNN")
Pkg.add("Resampling")
using kNN, Resampling
iris = data("datasets", "iris")
train, test = splitrandom(iris, 0.80)
test["Guess"] = knn(matrix(train[:, 2:5]),
                      matrix(test[:, 2:5]),
                      vector(train[:, 6]),
                      10)
by(test, ["Species", "Guess"], nrow)
mean(test["Guess"] .== test["Species"])
```

```
Pkg.add("DecisionTree")
using DecisionTree, Resampling
iris = data("datasets", "iris")
train, test = splitrandom(iris, 0.80)
train_features = matrix(train[:, 2:5])
train_labels = vector(train[:, "Species"])
t = build_tree(train_features, train_labels)
preds = apply_tree(t, matrix(test[:, 2:5]))
mean(preds .== test["Species"])
```