Reading From Data Files

Arpit Bhatia

June 15, 2019

In many cases we might need to read data available in an external file rather than type it into Julia ourselves. This tutorial is concerned with reading tabular data into Julia and using it for a JuMP model. We'll be reading data using the DataFrames.jl package and some other packages specific to file types.

Note: There are multiple ways to read the same kind of data intto Julia. However, this tutorial only focuses on DataFrames.jl as it provides the ecosystem to work with most of the required file types in a straightforward manner.

0.1 DataFrames.jl

The DataFrames package provides a set of tools for working with tabular data. It is available through the Julia package system.

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

0.2 What is a DataFrame?

A DataFrame is a data structure like a table or spreadsheet. You can use it for storing and exploring a set of related data values. Think of it as a smarter array for holding tabular data.

1 Reading Tabular Data into a DataFrame

We will begin by reading data from different file formats into a DataFrame object. The example files that we will be reading are present in the data folder.

1.1 Excel Sheets

Excel files can be read using the ExcelFiles.jl package.

```julia
Pkg.add("ExcelFiles")
```

To read a Excel file into a DataFrame, we use the following julia code. The first arguement to the load function is the file to be read and the second arguement is the name of the sheet.

```
using DataFrames
using ExcelFiles
excel_df = DataFrame(load("data/SalesData.xlsx", "SalesOrders"))
```

	OrderDate	Region	Rep	Item	Units	Unit Cost	Total
	Dates	String	String	String	Float64	Float64	Float64
1	2018-01-06T00:00:00	East	Jones	Pencil	95.0	1.99	189.05
2	2018-01-23T00:00:00	Central	Kivell	Binder	50.0	19.99	999.5
3	2018-02-09T00:00:00	Central	Jardine	Pencil	36.0	4.99	179.64
4	2018-02-26T00:00:00	Central	Gill	Pen	27.0	19.99	539.73
5	2018-03-15T00:00:00	West	Sorvino	Pencil	56.0	2.99	167.44
6	2018-06-08T00:00:00	East	Jones	Binder	60.0	8.99	539.4
7	2018-06-25T00:00:00	Central	Morgan	Pencil	90.0	4.99	449.1
8	2018-07-12T00:00:00	East	Howard	Binder	29.0	1.99	57.71
9	2019-07-21T00:00:00	Central	Morgan	Pen Set	55.0	12.49	686.95
10	2019-08-07T00:00:00	Central	Kivell	Pen Set	42.0	23.95	1005.9
11	2019-08-24T00:00:00	West	Sorvino	Desk	3.0	275.0	825.0
12	2019-09-10T00:00:00	Central	Gill	Pencil	7.0	1.29	9.03
13	2019-09-27T00:00:00	West	Sorvino	Pen	76.0	1.99	151.24
14	2019-10-14T00:00:00	West	Thompson	Binder	57.0	19.99	1139.43
15	2019-10-31T00:00:00	Central	Andrews	Pencil	14.0	1.29	18.06
16	2019-11-17T00:00:00	Central	Jardine	Binder	11.0	4.99	54.89
17	2019-12-04T00:00:00	Central	Jardine	Binder	94.0	19.99	1879.06
18	2019-12-21T00:00:00	Central	Andrews	Binder	28.0	4.99	139.72

1.2 CSV Files

CSV and other delimited text files can be read the CSV.jl package.

```
Pkg.add("CSV")
```

To read a CSV file into a DataFrame, we use the `CSV.read` function.

```
using CSV
csv_df = CSV.read("data/StarWars.csv")
```

	Name	Gender	Height	Weight	Eyecolor	Haircolor	Skincolor	Homeland
	String	String	Float64	String	String	String	String	String
1	Anakin Skywalker	male	1.88	84	blue	blond	fair	Tatooine
2	Padme Amidala	female	1.65	45	brown	brown	light	Naboo
3	Luke Skywalker	male	1.72	77	blue	blond	fair	Tatooine
4	Leia Skywalker	female	1.5	49	brown	brown	light	Alderaan
5	Qui-Gon Jinn	male	1.93	88.5	blue	brown	light	unk_planet
6	Obi-Wan Kenobi	male	1.82	77	bluegray	auburn	fair	Stewjon
7	Han Solo	male	1.8	80	brown	brown	light	Corellia
8	Sheev Palpatine	male	1.73	75	blue	red	pale	Naboo
9	R2-D2	male	0.96	32	NA	NA	NA	Naboo
10	C-3PO	male	1.67	75	NA	NA	NA	Tatooine
11	Yoda	male	0.66	17	brown	brown	green	unk_planet
12	Darth Maul	male	1.75	80	yellow	none	red	Dathomir
13	Dooku	male	1.93	86	brown	brown	light	Serenno
14	Chewbacca	male	2.28	112	blue	brown	NA	Kashyyyk
15	Jabba	male	3.9	NA	yellow	none	tan-green	Tatooine
16	Lando Calrissian	male	1.78	79	brown	blank	dark	Socorro
17	Boba Fett	male	1.83	78	brown	black	brown	Kamino
18	Jango Fett	male	1.83	79	brown	black	brown	ConcordDawn
19	Grievous	male	2.16	159	gold	black	orange	Kalee
20	Chief Chirpa	male	1.0	50	black	gray	brown	Endor

1.3 Other Delimited Files

We can also use the CSV.jl package to read any other delimited text file format. By default, CSV.File will try to detect a file's delimiter from the first 10 lines of the file. Candidate delimiters include `','`, `'\t'`, `' '`, `'|'`, `';'`, and `':'`. If it can't auto-detect the delimiter, it will assume `','`. Let's take the example of space separated data.

```
ss_df = CSV.read("data/Cereal.txt")
```

	Name	Cups	Calories	Carbs	Fat	Fiber	Potassium	Protein	Sodium
	String	Float64	Int64	Float64	Int64	Float64	Int64	Int64	Int64
1	CapnCrunch	0.75	120	12.0	2	0.0	35	1	220
2	CocoaPuffs	1.0	110	12.0	1	0.0	55	1	180
3	Trix	1.0	110	13.0	1	0.0	25	1	140
4	AppleJacks	1.0	110	11.0	0	1.0	30	2	125
5	CornChex	1.0	110	22.0	0	0.0	25	2	280
6	CornFlakes	1.0	100	21.0	0	1.0	35	2	290
7	Nut&Honey	0.67	120	15.0	1	0.0	40	2	190
8	Smacks	0.75	110	9.0	1	1.0	40	2	70
9	MultiGrain	1.0	100	15.0	1	2.0	90	2	220
10	CracklinOat	0.5	110	10.0	3	4.0	160	3	140
11	GrapeNuts	0.25	110	17.0	0	3.0	90	3	179
12	HoneyNutCheerios	0.75	110	11.5	1	1.5	90	3	250
13	NutriGrain	0.67	140	21.0	2	3.0	130	3	220
14	Product19	1.0	100	20.0	0	1.0	45	3	320
15	TotalRaisinBran	1.0	140	15.0	1	4.0	230	3	190
16	WheatChex	0.67	100	17.0	1	3.0	115	3	230
17	Oatmeal	0.5	130	13.5	2	1.5	120	3	170
18	Life	0.67	100	12.0	2	2.0	95	4	150
19	Maypo	1.0	100	16.0	1	0.0	95	4	0
20	QuakerOats	0.5	100	14.0	1	2.0	110	4	135
21	Muesli	1.0	150	16.0	3	3.0	170	4	150
22	Cheerios	1.25	110	17.0	2	2.0	105	6	290
23	SpecialK	1.0	110	16.0	0	1.0	55	6	230

We can also specify the delimiter by passing the `delim` arguement.

```
delim_df = CSV.read("data/Soccer.txt", delim = "::")
```

	Team	Played	Wins	Draws	Losses	Goals_for	Goals_against
	String	Int64	Int64	Int64	Int64	String	String
1	Barcelona	38	30	4	4	110 goals	21 goals
2	Real Madrid	38	30	2	6	118 goals	38 goals
3	Atletico Madrid	38	23	9	6	67 goals	29 goals
4	Valencia	38	22	11	5	70 goals	32 goals
5	Seville	38	23	7	8	71 goals	45 goals
6	Villarreal	38	16	12	10	48 goals	37 goals
7	Athletic Bilbao	38	15	10	13	42 goals	41 goals
8	Celta Vigo	38	13	12	13	47 goals	44 goals
9	Malaga	38	14	8	16	42 goals	48 goals
10	Espanyol	38	13	10	15	47 goals	51 goals
11	Rayo Vallecano	38	15	4	19	46 goals	68 goals
12	Real Sociedad	38	11	13	14	44 goals	51 goals
13	Elche	38	11	8	19	35 goals	62 goals
14	Levante	38	9	10	19	34 goals	67 goals
15	Getafe	38	10	7	21	33 goals	64 goals
16	Deportivo La Coruna	38	7	14	17	35 goals	60 goals
17	Granada	38	7	14	17	29 goals	64 goals
18	Eibar	38	9	8	21	34 goals	55 goals
19	Almeria	38	8	8	22	35 goals	64 goals
20	Cordoba	38	3	11	24	22 goals	68 goals

Note that by default, are read-only. If we wish to make changes to the data read, we pass the `copycols = true` arguement in the function call.

```
ss_df = CSV.read("data/Cereal.txt", copycols = true)
```

2 Working with DataFrames

Now that we have read the required data into a DataFrame, let us look at some basic operations we can perform on it.

2.1 Querying Basic Information

The `size` function gets us the dimensions of the DataFrame.

```
size(ss_df)
```

```
(23, 10)
```

We can also us the `nrow` and `ncol` functions to get the number of rows and columns respectively.

```
nrow(ss_df), ncol(ss_df)
```

```
(23, 10)
```

The `describe` function gives basic summary statistics of data in a DataFrame.

```
describe(ss_df)
```

	variable	mean	min	median	max	nunique	nmissing	eltype
	Symbol	Union	Any	Union	Any	Union	Nothing	DataType
1	Name		AppleJacks		WheatChex	23		String
2	Cups	0.823043	0.25	1.0	1.25			Float64
3	Calories	113.043	100	110.0	150			Int64
4	Carbs	15.0435	9.0	15.0	22.0			Float64
5	Fat	1.13043	0	1.0	3			Int64
6	Fiber	1.56522	0.0	1.5	4.0			Float64
7	Potassium	86.3043	25	90.0	230			Int64
8	Protein	2.91304	1	3.0	6			Int64
9	Sodium	189.957	0	190.0	320			Int64
10	Sugars	7.52174	1	7.0	15			Int64

Names of every column can be obtained by the `names` function.

```
names(ss_df)
```

```
10-element Array{Symbol,1}:
 :Name
 :Cups
 :Calories
 :Carbs
 :Fat
 :Fiber
 :Potassium
 :Protein
 :Sodium
 :Sugars
```

Correspong data types are obtained using the `eltypes` function.

```
eltypes(ss_df)
```

```
10-element Array{DataType,1}:
 String
 Float64
 Int64
 Float64
 Int64
 Float64
 Int64
 Int64
 Int64
 Int64
```

2.2 Accessing the Data

Similar to regular arrays, we use numerical indexing to access elements of a DataFrame.

```
csv_df[1,1]
```

```
"Anakin Skywalker"
```

The following are different ways to access a column.

```
csv_df[1]
```

```
20-element CSV.Column{String,String}:
 "Anakin Skywalker"
 "Padme Amidala"
 "Luke Skywalker"
 "Leia Skywalker"
 "Qui-Gon Jinn"
 "Obi-Wan Kenobi"
 "Han Solo"
 "Sheev Palpatine"
 "R2-D2"
 "C-3PO"
 "Yoda"
 "Darth Maul"
 "Dooku"
 "Chewbacca"
 "Jabba"
 "Lando Calrissian"
 "Boba Fett"
 "Jango Fett"
 "Grievous"
 "Chief Chirpa"
```

```
csv_df[:Name]
```

```
20-element CSV.Column{String,String}:
 "Anakin Skywalker"
 "Padme Amidala"
 "Luke Skywalker"
 "Leia Skywalker"
 "Qui-Gon Jinn"
 "Obi-Wan Kenobi"
 "Han Solo"
 "Sheev Palpatine"
 "R2-D2"
 "C-3PO"
 "Yoda"
 "Darth Maul"
 "Dooku"
 "Chewbacca"
 "Jabba"
 "Lando Calrissian"
 "Boba Fett"
 "Jango Fett"
 "Grievous"
 "Chief Chirpa"
```

```
csv_df.Name
```

```
20-element CSV.Column{String,String}:
 "Anakin Skywalker"
 "Padme Amidala"
 "Luke Skywalker"
 "Leia Skywalker"
 "Qui-Gon Jinn"
 "Obi-Wan Kenobi"
 "Han Solo"
 "Sheev Palpatine"
 "R2-D2"
 "C-3PO"
 "Yoda"
```

```
 "Darth Maul"
 "Dooku"
 "Chewbacca"
 "Jabba"
 "Lando Calrissian"
 "Boba Fett"
 "Jango Fett"
 "Grievous"
 "Chief Chirpa"
```

```
csv_df[:, 1] # note that this creates a copy
```

```
20-element WeakRefStrings.StringArray{String,1}:
 "Anakin Skywalker"
 "Padme Amidala"
 "Luke Skywalker"
 "Leia Skywalker"
 "Qui-Gon Jinn"
 "Obi-Wan Kenobi"
 "Han Solo"
 "Sheev Palpatine"
 "R2-D2"
 "C-3PO"
 "Yoda"
 "Darth Maul"
 "Dooku"
 "Chewbacca"
 "Jabba"
 "Lando Calrissian"
 "Boba Fett"
 "Jango Fett"
 "Grievous"
 "Chief Chirpa"
```

The following are different ways to access a row.

```
csv_df[1:1, :]
```

	Name	Gender	Height	Weight	Eyecolor	Haircolor	Skincolor	Homeland	B
	String	String	Float64	String	String	String	String	String	St
1	Anakin Skywalker	male	1.88	84	blue	blond	fair	Tatooine	41.9

```
csv_df[1, :] # this produces a DataFrameRow
```

	Name	Gender	Height	Weight	Eyecolor	Haircolor	Skincolor	Homeland	B
	String	String	Float64	String	String	String	String	String	St
1	Anakin Skywalker	male	1.88	84	blue	blond	fair	Tatooine	41.9

We can change the values just as we normally assign values.

Assign a range to scalar.

```
excel_df[1:3, 5] = 1
```

```
1
```

Vector to equal length vector.

```
excel_df[4:6, 5] = [4, 5, 6]
```

```
3-element Array{Int64,1}:
 4
 5
 6
```

Subset of the DataFrame to another data frame of matching size.

```
excel_df[1:2, 6:7] = DataFrame([-2 -2; -2 -2])
```

	x1	x2
	Int64	Int64
1	-2	-2
2	-2	-2

```
excel_df
```

	OrderDate	Region	Rep	Item	Units	Unit Cost	Total
	Dates	String	String	String	Float64	Float64	Float64
1	2018-01-06T00:00:00	East	Jones	Pencil	1.0	-2.0	-2.0
2	2018-01-23T00:00:00	Central	Kivell	Binder	1.0	-2.0	-2.0
3	2018-02-09T00:00:00	Central	Jardine	Pencil	1.0	4.99	179.64
4	2018-02-26T00:00:00	Central	Gill	Pen	4.0	19.99	539.73
5	2018-03-15T00:00:00	West	Sorvino	Pencil	5.0	2.99	167.44
6	2018-06-08T00:00:00	East	Jones	Binder	6.0	8.99	539.4
7	2018-06-25T00:00:00	Central	Morgan	Pencil	90.0	4.99	449.1
8	2018-07-12T00:00:00	East	Howard	Binder	29.0	1.99	57.71
9	2019-07-21T00:00:00	Central	Morgan	Pen Set	55.0	12.49	686.95
10	2019-08-07T00:00:00	Central	Kivell	Pen Set	42.0	23.95	1005.9
11	2019-08-24T00:00:00	West	Sorvino	Desk	3.0	275.0	825.0
12	2019-09-10T00:00:00	Central	Gill	Pencil	7.0	1.29	9.03
13	2019-09-27T00:00:00	West	Sorvino	Pen	76.0	1.99	151.24
14	2019-10-14T00:00:00	West	Thompson	Binder	57.0	19.99	1139.43
15	2019-10-31T00:00:00	Central	Andrews	Pencil	14.0	1.29	18.06
16	2019-11-17T00:00:00	Central	Jardine	Binder	11.0	4.99	54.89
17	2019-12-04T00:00:00	Central	Jardine	Binder	94.0	19.99	1879.06
18	2019-12-21T00:00:00	Central	Andrews	Binder	28.0	4.99	139.72

There are a lot more things which can be done with a DataFrame. See the docs for more information.

3 A Complete Modelling Example - Passport Problem

Let's now apply what we have learnt to solve a real modelling problem.

The Passport Index Dataset lists travel visa requirements for 199 countries, in .csv format. Our task is to find out the minimum number of passports required to visit all countries.

In this dataset, the first column represents a passport (=from) and each remaining column represents a foreign country (=to). The values in each cell are as follows:

- 3 = visa-free travel
- 2 = eTA is required

- 1 = visa can be obtained on arrival
- 0 = visa is required
- -1 is for all instances where passport and destination are the same

Our task is to find out the minimum number of passports needed to visit every country without requiring a visa. Thus, the values we are interested in are -1 and 3. Let us modify the data in the following manner -

```
passportdata = CSV.read("data/passport-index-matrix.csv", copycols = true)

for i in 1:nrow(passportdata)
    for j in 2:ncol(passportdata)
        if passportdata[i,j] == -1 || passportdata[i,j] == 3
            passportdata[i,j] = 1
        else
            passportdata[i,j] = 0
        end
    end
end
```

The values in the cells now represent:

- 1 = no visa required for travel
- 0 = visa required for travel

Let us assossciate each passport with a decision variable $pass_{cntr}$ for each country. We want to minize the sum $\sum pass_{cntr}$ over all countries.

Since we wish to visit all the countries, for every country, we should own atleast one passport that lets us travel to that country visa free. For one destination, this can be mathematically represented as $\sum_{cntr \in world} passportdata_{cntr,dest} \cdot pass_{cntr} \geq 1$.

Thus, we can represent this problem using the following model:

$$
\begin{aligned}
\min \quad & \sum_{cntr \in World} pass_{cntr} & \\
s.t. \quad & \sum_{cntr \in World} passportdata_{cntr,dest} \cdot pass_{cntr} \geq 1 & \forall dest \in World \\
& pass_{cntr} \in \{0, 1\} & \forall cntr \in World
\end{aligned}
$$

We'll now solve the problem using JuMP.

```
using JuMP, GLPK

# Finding number of countries
n = ncol(passportdata) - 1 # Subtract 1 for column representing country of passport

model = Model(with_optimizer(GLPK.Optimizer))
@variable(model, pass[1:n], Bin)
@constraint(model, [j = 2:n], sum(passportdata[i,j] * pass[i] for i in 1:n) >= 1)
@objective(model, Min, sum(pass))
```

```julia
optimize!(model)

println("Minimum number of passports needed:  ", objective_value(model))

countryindex = findall(value.(pass) .== 1 )

print("Countries:  ")
for i in countryindex
    print(names(passportdata)[i+1], " ")
end
```

```
Minimum number of passports needed: 23.0
Countries: Afghanistan Angola Australia Austria Comoros Congo Eritrea Gambi
a Georgia Hong Kong India Iraq Kenya Madagascar Maldives North Korea Papua
New Guinea Singapore Somalia Sri Lanka Tunisia United Arab Emirates United
States
```