**Julia Programming Language**

Julia is an general purpose programming language like C , C++ . Julia is developed mainly for numerical computation. As of we know how science has been changing in the area of computation because everything needs quick computation in-order to generate large data results in fraction of seconds , and more-ever we do get an question like despite having of many programming languages with good performance, compatibility like C, C++, Java, Python why Julia? Julia is developed mainly for numerical computation purpose and it will also eliminates the performance issues perhaps it will provide environment which is enough to develop for applications that required high performances.

Let we discuss in detail more about Julia programming language . The concepts are

* Installation
* Basics of Julia for Data Analysis
* Julia Data Structures
* Loops, Conditionals in Julia
* Exploratory analysis with Julia
* Introduction to DataFrames.jl
* Visualization in Julia using Plots.jl
* Data Munging in Julia
* Building a predictive ML model
* Logistic Regression
* Decision Tree
* Random Forest
* Calling R and Python libraries in Julia
* Using pandas with Julia
* Using ggplot2 in Julia

**Installation**

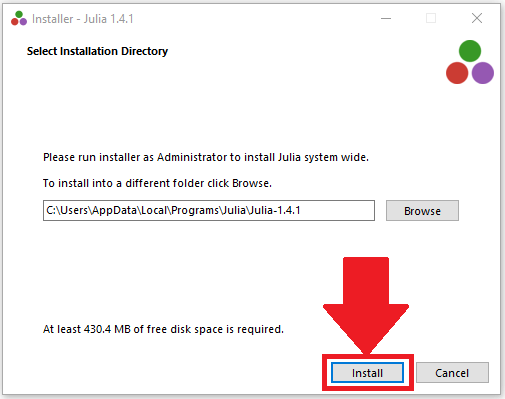
Here, we are going to see how the download and installation of Julia is done. The following steps like

Step1: To download Julia go through <https://julialang.org/downloads/> link or else you can search in the google download Julia

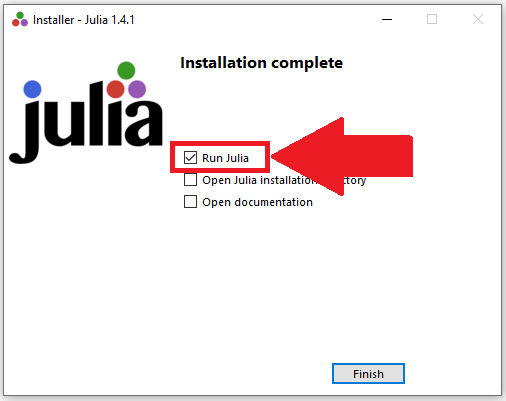
Step2: One more thing is that download as per your machine bit configuration i.e 32 or 64 bit

Step3: After download run the .exe file

Step4: click install buton and furtherly go with the picture shown below.



Step5: Click the checkbox inorder to run Julia and click **Finish**. As shown in the figure



Step6: Now you can see a command line prompt which is also known as REPL

(Read-Eval-Print-Loop)

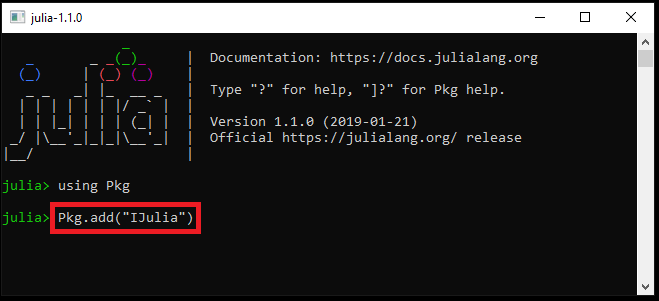
Before going into another topic lets see what are the packages of Julia that are used in data analysis or data science related projects.

We know about jupyter notebook and its popularity in the field of data science as well as ML which gives fast results and easy to handle the ide. Similarly we do have notebook for Julia i.e

Juno IDE but if you are familiar with notebook then go on with jupyter notebbok. Lets see how we can setup the package for julia notebook(IJulia).

Open the Julia prompt and then type following command:

Julia> Pkg.add(“IJulia”)

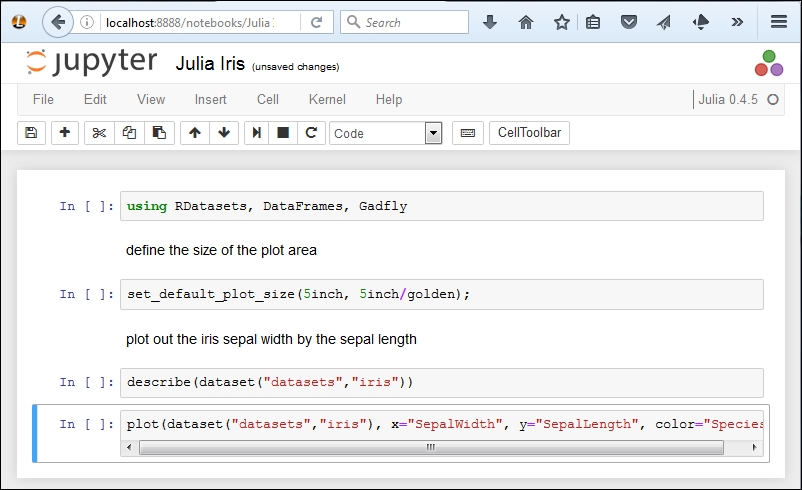


If u look above picture you can easily find and then al the packages will be added or updared.

After IJulia package is downloaded or updated you can type following code to run it:

Julia> using IJulia

Julia> notebook()



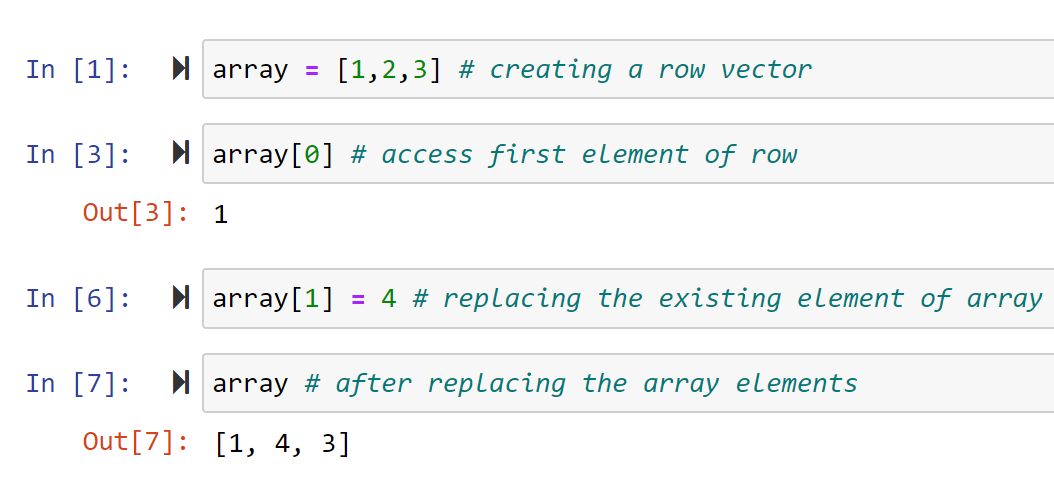
Then you will get by default notebook “dashboard” which opens in your home directory or the installation that you have done in the location. If you want to open dashboard in different directory then notebook(dir = “/some/path”).

**Data Structures Concept in Julia Programming Language**

Like Every programming language Julia also have data structure concepts . Lets see the concepts of data structures that are used for data analysis.

1. **Vector(Array) –** A vector is one dimensional array which means similar to normal array. In array we use numbers followed by comma as separator similarly in Julia also the vector(array) follows same.

Lets have look on piece of code



In Julia the index start with 1 whereas above code snippet is python and it starts with 0.

1. **Matrix Operations**

Matrix which is another data structure that is widely used in linear algebra. We know that matrix is of multi dimensional array. Lets see dome operation of matrix in Julia,

**A = [1 2 3; 4 5 6; 7 8 9] # semi-column is used to change rows**

**When we print it looks like : 1 2 3**

**4 5 6**

**7 8 9**

**In order access element, say A [1,2] = 2**

Now for transpose of a matrix, **A’** then the following result will be look like as

**A’ = 1 4 7**

**2 5 8**

**3 6 9**

1. **Dictionary**

Another data structure is dictionary which is an un ordered key : value pair and the keys are always unique in nature.

Let’s have a look on the dictionary implementation,

**D = Dict (“string1” => “Hello”, “length” => 5) #create dictionary**

It will get result : string => Hello

Length => 5

Suppose in-order to access the dictionary we will access the key of dictionary then the value will give us as result

**D[“length]**

**o/p: 5**

to get count of dictionary use object. Count i.e **D.count**

**Operations of Dictionary:**

1. **Creation =**  **Dict(“a” => 1, “b” => 2)**
2. **Addition = d[“c”] = 3**
3. **Removal = delete !(d, “b”)**
4. **Lookup = get(d,”a”, 1)**
5. **Update = d[“a”] = 10**

**Strings**

Next data structure is strings , strings are generally written within the quotes as {“ ”} i.e inverted commas. Similar to the python in Julia also once string is created it cannot be changes as they are immutable.

Lets have a look,

**Text = “Hello world”**

**print(Text[1]) # will gives first character of string as H**

**Print(Text.length) # will gives the length of string 11**

There are three key phases of data structures that are used in data analysis

1. **Data Exploration**

It’s all about finding the data more than what we have

1. **Data Munging**

Cleaning the data and use that data for making better statistical models

1. **Predictive Modelling**

Final thing is run the algorithm and have fun

**Loops, Conditions In Julia**

Like other programming languages Julia also uses the loops and conditional statements

For loop

While Loop

If condition

These are most commonly used loops and condition statement in Julia as well as other programming languages

**If and else**

In Julia we need not to worry about spaces, identation, semicolon, brackets etc instead just add end at the end of statement or condition. Lets have the syntax for if and else

Syntax: if condition

Statement

else

Statement

end

**if elseif and else**

It also follows same as if else block follows. Let’s have look on syntax

Syntax: if condition

Statement

elseif

Statement

else

Statement

End

Lets take an example to the above we discuused

If x > 0

“Positive”

else if x < 0

“Negative”

else

“Whole Number”

Lets talk about loops in Julia.

**For Loop**

The only difference to the loop for with other languages for loop is, in Julia for loop will have start and end counter.

Julia> for i in 0: 10: 100

Print(i)

end

will gives result as: 0 10 20 30 40 50 60 70 80 90 100

Julia> for a in [“red”, “green”, “yellow”]

Print(a, “ “)

end

Will give result as : red green yellow

Julia> for a in Dict(“name” => “orange”, “size” => 6)

Print(a)

end

Name => orange Size -=> 6

Similarly we can also iterate through 2D array, lets have look on that

A = reshape(1:50, (3, 3))

for I in A

Print(I, “ “)

end

The result will be as 1 2 3 4 5 6 7 8 9 …………..50

We can also use inside of functions

function()

for condition

Statement

end

return

end

We know that scope of an variable inside a method or function will exists until its life span is not yet done once method or function ends and comes out then the variable scope is zero or dead

Function()

K = 2

for I in 1:10 :50

K = k\*i

end

return

end

**if we want to persist the variable to be exist in the function or method then use keyword “global” before variable name.**

**continue and break** are the condition statements used in between the loops

for I 10:5:20

print(i)

continue

end

**comprehensions**

similar to python Julia also supports comprehensions

Julia> s = set([a for a in 1: 8])

Set([6,4,5,7,1,3,2,8])

Julia> [(a,b) for a in 1:5, c in 1:2]

(1,1) (1,2)

(2,1) (2,2)

(3,1) (3,2)

(4,1) (4,2)

(5,1) (5,2)

**Generator Expressions**

Like comprehensions generating expressions can also be used to produce result using iterable variable.

Let’s have a look on the example

**Julia> sum( x^2 for x in 1:10)**

**385**

**Nested Loops**

Nested loops in Julia is quite different as of writing loop inside another loop is known to be as nested loops. But, in Julia we need not make duplicate loops instead we can use

@show(var1, var2) variables with comma separated

Have a loop on the piece of code for better understanding

for a in 1 : 10, y in 1: 10

@show (x,y)

Result will be:

(x,y) = (1,1)

(x,y) = (1,2)

(x,y) = (1,3)

(x,y) = (1,4)

(x,y) = (1,5)

(x,y) = (1,6)

(x,y) = (1,7)

(x,y) = (1,8)

……………

(x,y) = (10,10)

**@show is an macro that prints the names and values**

**@time will gives the complexity of loops**

Julia> x = rand(1000);

Julia> function sum()

A = 0.0

For I in x

A + = i

End

Return A

End

Julia> @time sum()

0.017705 seconds (15. 28k allocations: 694. 484 kiB)

496.84883432553846

**While Loop**

Same as for loop while as performs only when condition is true. The following syntax is

While condition

Statements

End

Let’s have an example

Julia> x = 0

0

Julia> while x < 3

Print(x)

global x+ = 1

end

result: 0 1 2

And finally Exceptions with loops, like other programming language Julia also have try, catch blocks.

Julia> s = “apple”

try

S[1] = “a”

catch e

Print(“caught an error: $e”)

End

**Basics Of Julia For Data Analysis**

Till today many of us familiar with python or R language in the field of machine learning, data science. All those are good in their performances and predicting fasten results. Whereas Julia is such a language that can computate the large amount of data and give results in fraction of seconds.

It is very similar to the languages like python or R with respect to syntax. There won’t be no time taking for one to use Julia on data analysis. Moreever a lot of time is spent by data scientists in-order to transform the data into good format . For that purpose Julia will provides an extensive library in dealing with the raw data and to make into good format of data I,e structured data format . There are basic steps to be followed in data analysis

1. Always explore the given data sets or data tables and apply statistical methods to find patterns in numbers.
2. Second thing is plot the data for visualization.

As in Machine Learning the data has to convert into data frames similarly using Julia we can do that. The following package provide by the Julia on Data Frames is DataFrames.jl that will converts the data into matrix format with extensions like .csv, .xlsx etc

Julia> Pkg.add(“DataFrames.jl”)

Let’s take an example to demonstrate dataframes in Julia

**Using DataFrames**

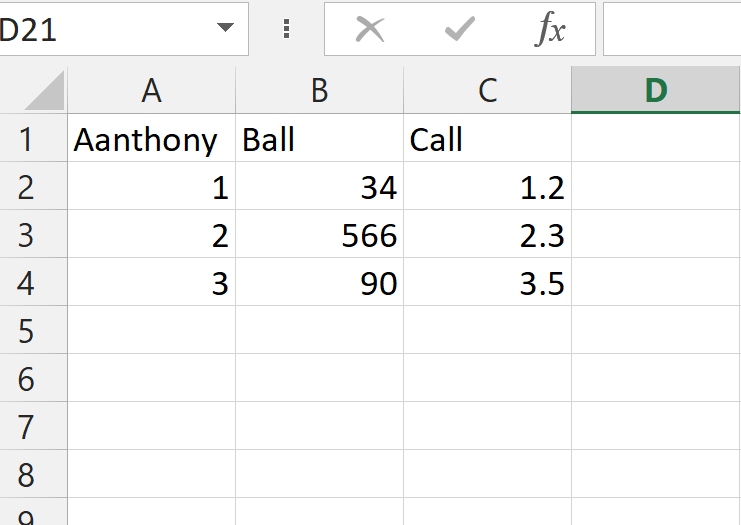
**#read the dataset**

**df = readtable(“demo.csv”, separator=’,’)**

---we have loaded the dataset into df variable and then we can print the dataset----

**Df**

Look at the demo dataset , this is just the view of dataset its not the dataframe view.

****

**Dataframe functions** likefinding size , column names, to know the first n rows of dataframe set

**size(df)** = given rows and columns (mXn)

output: [ 3, 3]

**Names(df)** = column names

Output: [‘Aanthony’, ‘Ball’, ‘Call’]

**head(df)** = say we give head(5) will results first five rows

output: first five rows

**Numerical Data** like describe() function which gives basic statistical data analysis such as mean, mode, sum, avg

**Categorial Data**  countmap() is function that maps the values to the no. of occurrence in the dataset.

**Dealing with Missing Data**

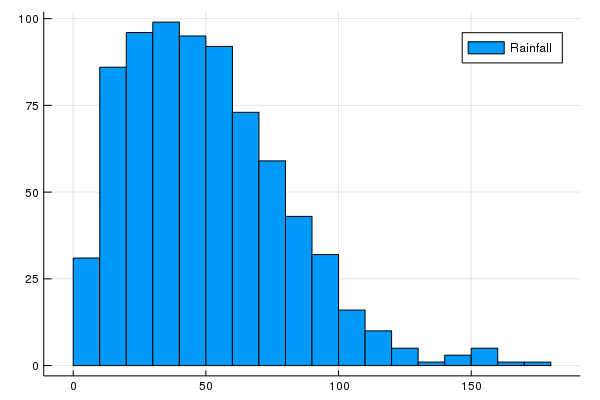
This is very important concept because entire game runs on this data only as of when there is loss of data obviously the predicted result will generates differ accuracy. So, in-order to maintain a good accuracy we should handle the missing data from the dataset

**showcols()** = to check for missing values in variables

And we can replace the empty values with some related values , lets say

**df.replace(df[‘Anthony’] == “ “ , : “some data to replace”)**

**Visualization** part that generalizes the entire data and their relation among them.

****

Above chart says that rainfall over a period of time interval keeps on increasing [cm’s]

**Point to remember**

Histogram charts should always be divide into bins i.e more bins more data analyzed .

**Exploratory Data Analysis With Julia**

Exploratory Data Analysis is used in understanding data in terms of data features, variables and their relationship among them. Always the main step to be do is understand the data set properly. There are some methods to be followed

**Methods to be followed on given dataset (explore)**

1. To the data table apply some statistics
2. Apply visual plot techniques over the data set or data table

**To the data table apply some statistics**

**Step1:** installing Data Frame Package

Using Julia over the data table or data set a data structure concept called Data Frames is used. As of data frame can handle multiple operations like speed , accuracy and compatibility

Data frames to be used in Julia should be installed first

The following command is used to install the data frame

Using Pkg

Pkg.add(“DataFrames”)

**Step2:** Next download the data set

**Step3:** Then install necessary packages, CSV packages, Data Frame etc

using DataFrames

using CSV

a = CSV.read(“sample.csv”)

**Step4:** Then have data exploration

Data exploration has to be done over the data set because it gives the relations among data variables, what are the functions ,column names, lists etc

using DataFrames

using CSV

a = CSV.read(“sample.csv”);

size(a)

names(a)

head(a, 10)

**Describe Function**

Describe function is used to give mean, mode, meadian, some basic statistical data over the data set

using DataFrames

using CSV

a = CSV.read(“sample.csv”);

describe(a)

describe(a, :all, cols = :SepalLength)

**Apply visual plot techniques over the data set**

Visual plotting in Julia can be achieved using plot libraries like Plots, StatPlots and Pyplot

**Plots :** it’s an high level plotting package which interfaces with other plotting packages called

‘**back-ends’** . Actually they behave like graphic engine that provides graphics

**StatPlots:** Itsan plotting package including with the Plots package especially these StatPlots are used for some statistics

**Pyplot:** Itsanpackage with Matplotlib which is library of python.

These libraries can be installed as follows:

Pkg.add(“Plots”)

Pkg.add(Statplots”)

Pkg.add(“Pyplot”)

**Distribution Analysis**

Here, in distribution Analysis Julia is performed using various plots such as histograms, scatterplot, boxplot

using DataFrames

using CSV

a = CSV.read(“sample.csv”);

using Plots

Plots.histogram(a[:SepalLength], bins = 50, xlabel = “Sepallength”,

Labels = “length in cm”)

Similarly we can plot graph using different formats like histogram etc

**Introduction To DataFrames.jl**

As we all know that Julia has the library that handles the data transformation like python and R does i.e DataFrames. This approach although looks similar to python or R but it differs during API call. For complex data tables DataFramesMeta concept is used

Lets see how to install and import the library

* To install library use command **Pkg.add(DataFrames**)
* To load the library use command **using DataFrame**

After doing above steps the next is to load the data set . The following way to read the data table is.

using CSV

Datatable = CSV.readtable(“sample.csv”)

**Fruits Sweet Sour**

**Apple 80% 10%**

**Orange 90% 10%**

**Pineapple 100% 0%**

After loading CSV file check for the missing values suppose if the column has missing values in the top most rows due to using type-auto recognization then there are chances of having error rate. Manually we have to remove the error tendancy from the data set.

To find missing value

Types = Dict(“Florida” => Union{Missing, Int64})

If we want to edit the values of imported dataframes then don not forget to use copycols = true

* Use the package from the stream HTTP:

Using DataFrame , HTTP, CSV

Resp = HTTP.request(“GET”, [https://somesite@domain.com?accesstyep = “Download](https://somesite@domain.com?accesstyep%20=%20%20%20%20%20%20%20))

df = CSV.read(IOBuffer(String(resp.body))

* Again create df from scratch

Df = DataFrame(

Color = [“red”, “yellow”, “orange”, “white”]

Shape = [ “circle”, “rhombus”, “vertical”]

Border = [“line”, “dotted”, “line”]

Area = [1.1,1.2,1.3,2.5])

* There are many possibilities with df like convert matrix form data to vector form :

For example:

df = DataFrame([[mat[ : , i]…] for I in a : size(mat, 2)], Symbol.(headerstrs))

Using dataframes package we can do a lot mpre with the data set or data table. Always the given dataset has to be converted into data frames i.e matrix conversions so that one can analyze the data properly and handle it regarding null values, missing values..

**Get Some Insights of Data**

* first(df, size)
* show(df, allrows=true, allcolls = true)
* last(df, size)
* describe(df)
* unique(df.fieldName)
* names(df)
* size(df)
* to iterate over each column [for a in eachcol(df)]
* to iterate over each row [for a in eachrow(df)]

**Filter**

In-order to refer to some columns there are two ways in data frame like referencing the stored values into the object or copying them into another new object

1. Myobject = df[ !, [cFruits]] {store values in object}
2. newObject = df[ :, [cFruits(s)] { Copying entire into new object }

You know we can also query using data frames let’s see how we can do

dfresult1 = @from I in df begin

@where i.col > 1

@select {aNewColName = i.col1, i.col3}

@collect DataFrame

end

dfresult2 = @from I in df begin

@where i.value != 1 && i.cat1 in [“red”, “yellow”]

@ select i

@collect DataFrame

end

**Replace Data**

We can replace the values of column with other data that to dictionary based values

df.col1 = map(key ->mydict[key], df.col1)

Can be concate the values of column using dot operation df.a = df.b

Appending rows : push! (df, [1 2 3])

Delete rows: deleterows !(df, rowIdx)

**Change the structure of data or holding object**

Here dataframe can be used to change name of column, data type of column , delete column, rename column or else replacing position of columns. Type casting which can be help to convert any kind of data type

From int to float: df.a = convert(Array{Float32, 1}, df.a)

**Sorting** sort ! (df, cols = (:col2, :col1), rev = (false, false))

So, Dataframes is most powerful library or package for data handling . It will handle missing values which cause a lot error tendancy . we can split the datasets and re combine them together and apply some statistical operations like aggregate functions,

**Visualization in Julia Using Plots.jl**

This is another way to explore the data and analysis i.e by doing visualization using various kinds of plot formats.

In Julia we can even plot the graph for the data using library. But, Julia does not provide direct library of its own instead it provides to use libraries of your own choice in Julia programs.

To have this functionality we need install some packages:

Julia> Pkg.add(“Plots.jl”)

Julia> Pkg.add(“StatPlots.jl”)

Julia> Pkg.add(“PyPlot.jl”)

This Plots.jl is act as interface to any plotting library such that using libraries in Julia we can plot data .

StatPlots.jl is supporting package for Plots.jl

PyPlot.jl will act as Matplotlib of python

Now, let’s see some data visualization plots using pyplot.jl and also we can get information about data table more using plots.

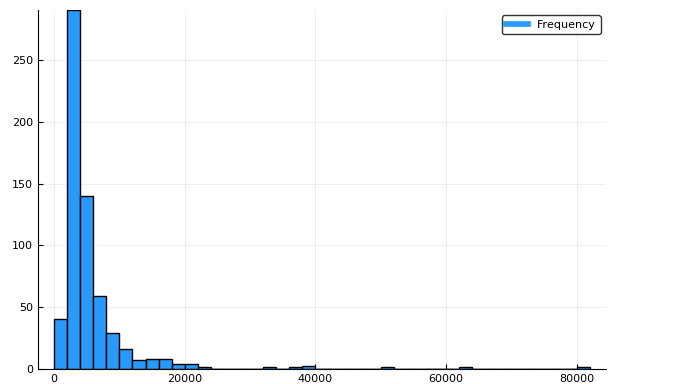
Using CSV

S = CSV.readtable(‘Venice.csv’)

using Plots, StatPlots

pyplot() #set backend as matplotlib package i.e matplotlib.pyplot

Plots.histogram(dropna(train[: ApplicationTax]), bins = 50, xlabel = “ApplicationTax”, labels = “Frequency”) # plot histogram



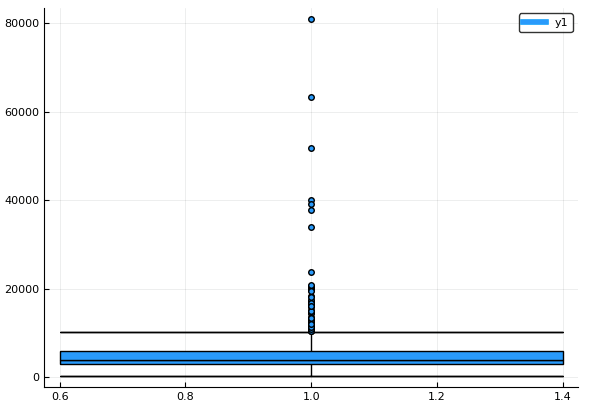
If you observe the plot we have different values with depriciation in the plot , so that is the reason why we need the bins as 50 or relevant to that

In other scenario we can look at box plots to understand the distributions of bins in the above graph clearly.

Lets see another way of visualizing the plot:

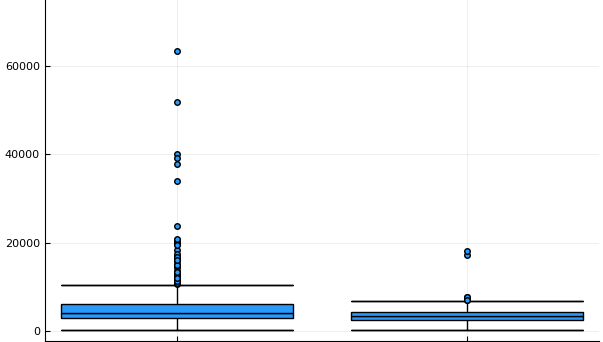
Plots.boxplot(dropna(train[: ApplicationTax]), xlabel = “”ApplicationTax”)

If u look the plot below it tells us the preence of extreme values . This can be attributed to the Tax in the society. And also we can segregate the part based on their profession in the society



Plots.boxplot(train[: Education], train[: ApllicationTax], label = “ApplicationTax”)

**ApplicationTax**



Now, if u see there is no difference between the Tax of the persons and also the Profession

of persons based on which the tax is paid i.e high or low tax .

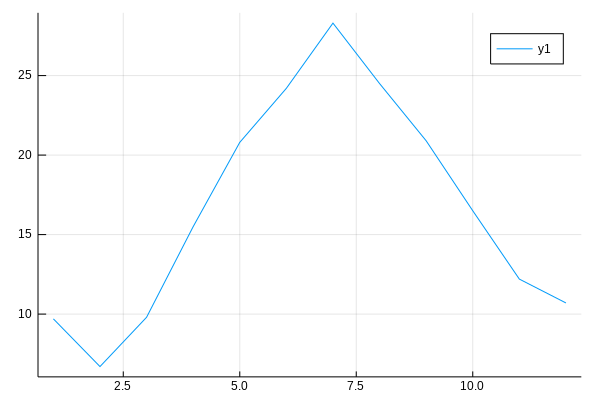
Lets have look on other charts like **line chart**, pie chart for rain data in a year/month

using CSV

a = CSV.read(“sample.csv”)

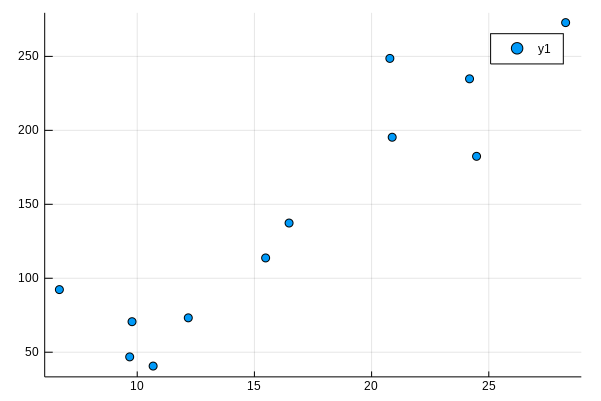
plot(a.month, a.max)

This graph will says that a month with maximum rain



Next, we will see **scatter chart** by using same data i.e rain data in a year/month

Scatter(a.Rain, label = “y1”)

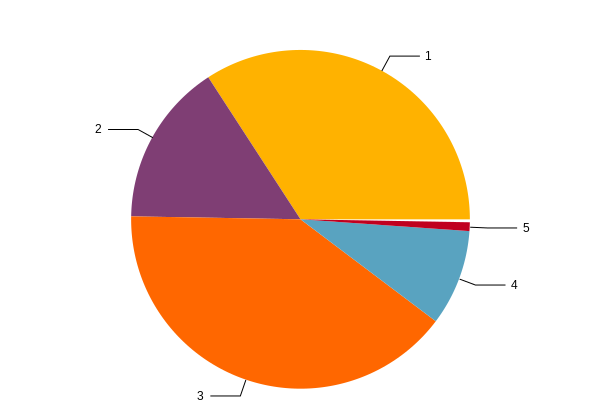


This chart says that the rainfall is vary’s on every year i.e increase as the year goes on increase

Similarly lets look on the **pie chart** also with same rain data in a year/month

W = 1:5; y = rand(5); #plotting data

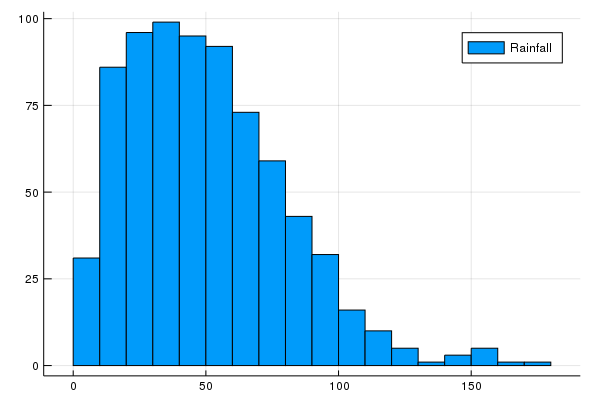
Pie(x,y)



The pie chart gives an analyzation of more area with rainfall followed by average and less rainfall per year or month.

**Histogram Chart**

Histogram(a.Rain, label = “Rainfall”)



We can easily find by histogram chart the rainfall is varies in a year with unequal distribution of rainfall.

So, I hope we learnt topic in Julia i.e plots. so far we completed all the basic charts that are used in Julia with plot library.

**Data Munging In Julia**

While we did analysis of data there are some problems that we encountered i.e missing values, null values all these problem has to be remove under data analysis step. To do so, data munging is a technique or process to handle the missing values in data table or data set i.e converting the raw data into some format that can be utilized for data analysis . It is also known as **Data Wrangling**

It is one of the most important component in data science .

The following packages that are required:

**RDataset** this packagae will load the data set generally used in R language since julia can also be access the libraries or packages of other languages like R it can be installed as follows

Julia> Pkg.add(‘RDatasets’)

As we know that inorder to convert into multidimensional array format to a data set in python or R we use data frames . similarly here in julia **DataFrames** and **DataFramesMeta** will provide the functionality

Julia> Pkg.add(‘DataFrames’)

Julia> Pkg.add(‘DataFramesMeta’)

Let’s load the data set

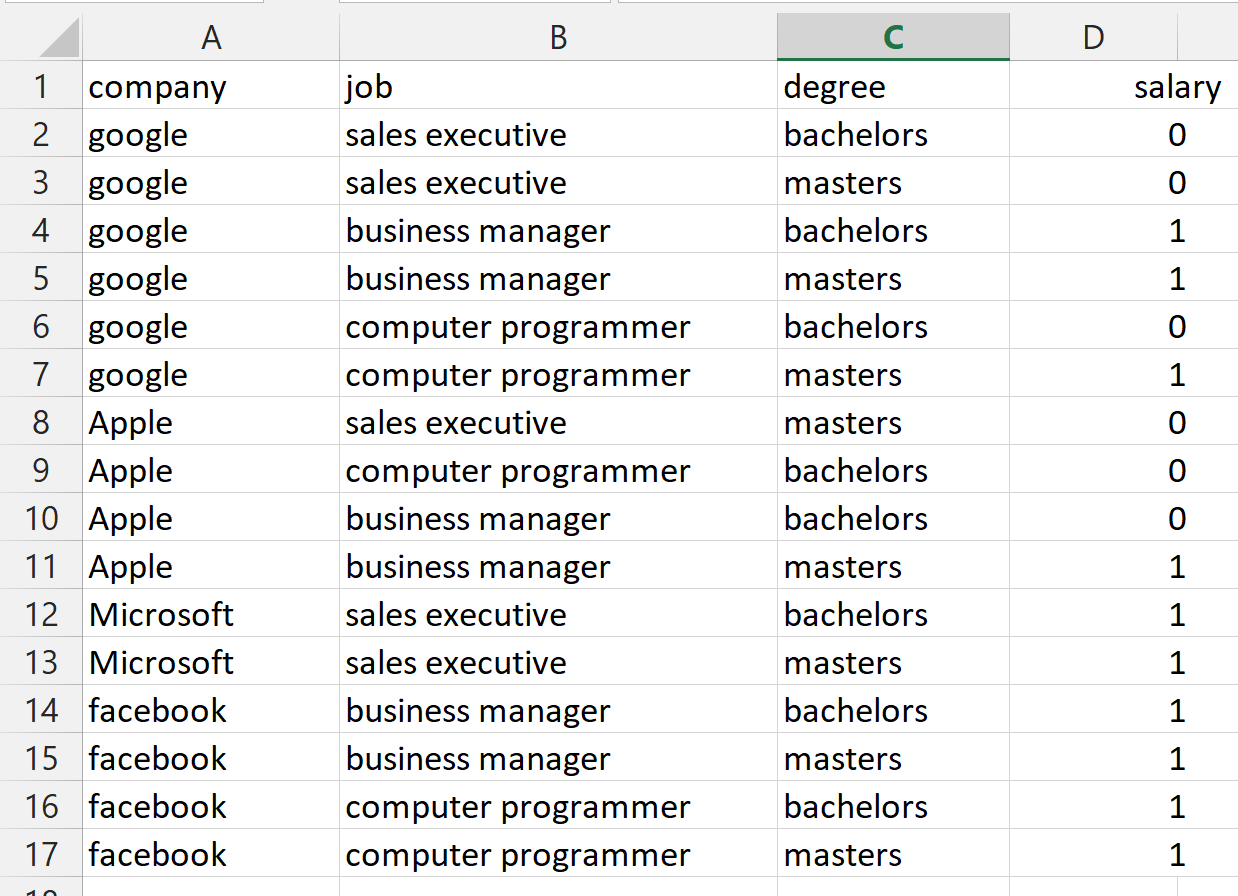
It contains columns

company

job

degree

salary



So, the analysis of this data set is if an employee having bachelors degree he or she can be promoted or salary can be increased and condition applys i.e varies with company.

using RDatasets

sal = dataset(“datasets”, “sample”)

head(sal)

**it gives the same dataset as we saw in the above figure**

**Using groupby():**

The groupby function will group the data in all the columns to a given value . It splits the datagrame and those split dataframes are again split into subsets then the function is used. The indices for data set starts from indices 1 when we use the groupby()

The following syntax:

groupby(a, :col\_names, sort = false, skipmissing = false)

Parameters are

a : dataframe

:col\_names: column names on which data set is split

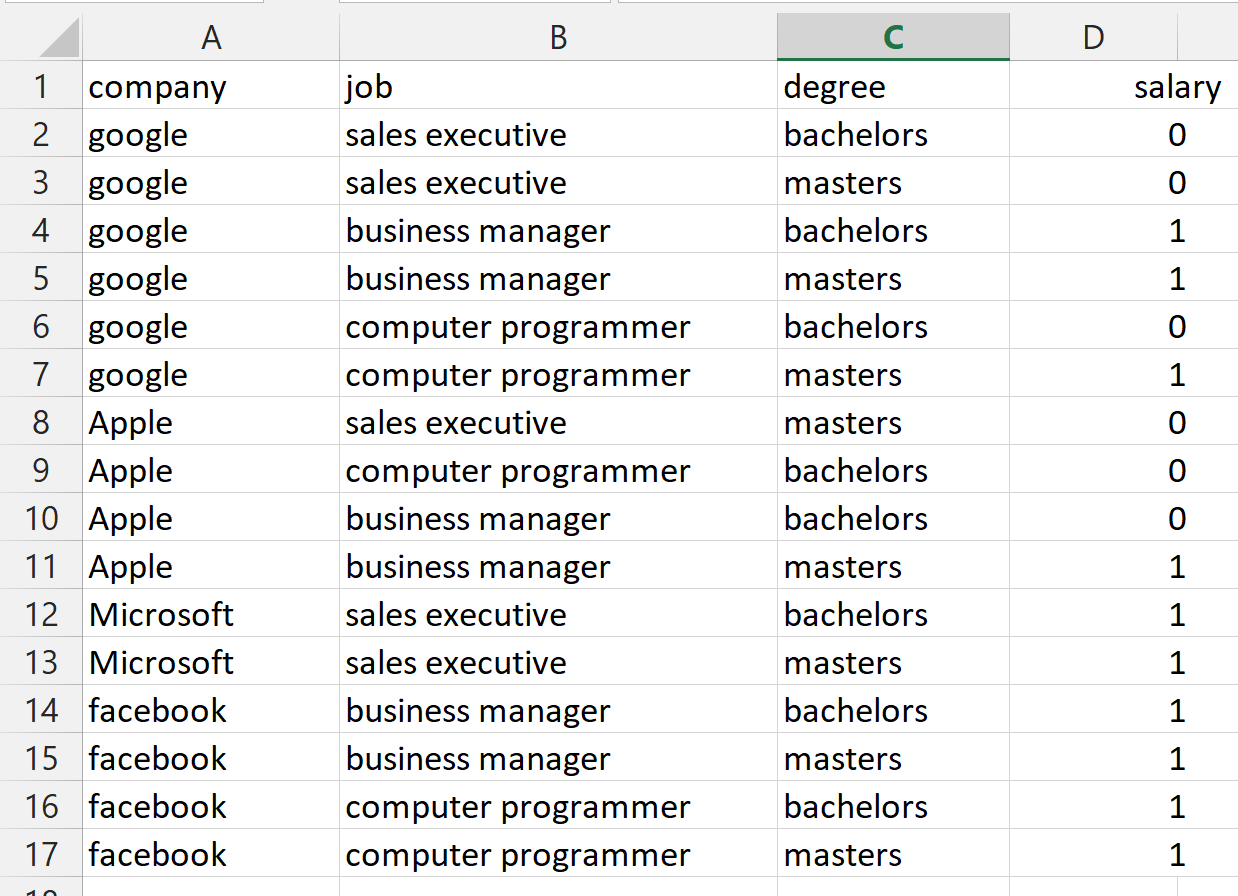
sort: to return the data set in sorted manner by default it is false

skipmissing: it will decides whether to skip the missing values or not , by default false

using RDatasets

sal = dataset(“datasets”, “sample”)

groupby(sal)



**by() function**

This by() function will performs **split-apply** method which means it will split the column and then apply the by() function . The syntax as follows:

by(a, :col\_names, function, sort = false)

The Parameters :

a: dataframe

col\_names: the split of columns

function: function applied on each column

sort: the dataframe to be return sort order by default it is false

lets split the dataframe and show the column who are eligible for salary promotion

using RDatasets

using Statistics

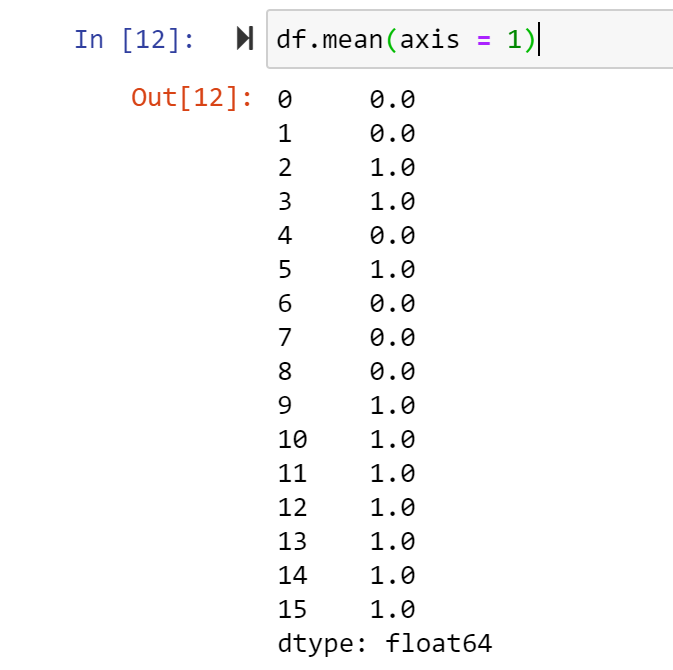
sal = dataset(“datasets”, “sample”)

by(sal, [:job, :degree]) do a DataFrame(Mean\_of\_Salary = mean(a[:Salary]),

Variance\_of\_Salary = var(a[:Salary])

End



 \* Mean of Column Salary

**aggregate() function**

aggregate function will also follows split- apply method . columns are split and then the function is applied to the specified column .

aggregate(a, :col\_names, function)

The Parameters are:

a: dataframe

col\_names: the split of columns

function: function applied on each column

using RDatasets

sal = dataset(“datasets”, “Sample”)

aggregate(sal, :job, degree)

**Missing**

In Julia the missing values are represented using special name i.e **missing** which is instance for the type Missing.

Julia> missing

missing

let’s see for the type of of missing

Julia> typeof(missing)

Missing

Missing type will allows users to create Vectors and DataFrame column with missing values.

Let we see an example :

Julia> x = [0, 1, missing]

3-element Array{Union{Missing, Int64}, 1}:

0

1

Missing

Julia> eltype(x)

Union{Missing, Int64}

Julia> Union{Missing, Int}

Union{Missing, Int64}

Julia> eltype(x) == Union{Missing, Int}

True

While performing some operations missing values can be excluded using a technique called as

“skipmissing”

Julia> skipmissing(x)

Base.Skipmissing{Array{Union{Union{Missing, Int64}, 1}}(Union{Missing, Int64}[0,1,missing].

Lets take an scenario i.e I want to find the average of all missing values.

Julia> avg(skipmissing(x))

4

Julia> collect(skipmissing(x))

2-element Array{Int64, 1}

**Coalesce is the function which is used to replace null value with some other values.**

Julia> coalesce(x, 0)

3-element Array{Int64, 1}

1

2

0

Similarly we may also have missing values or null values in rows . For that we can use dropmissing and dropmissing! to remove the missing values .

Julia> df = DataFrame(I = 1:4,

P = [missing, 3, missing, 2,1]

Q = [missing, missing, “c”,“d”,”e”])

4X3 DataFrame

Row | I x y

| Int64 Int 64 String?

1 | 1 missing missing

2 | 2 3 missing

3 | 3 missing c

4 | 4 2 d

Julia> dropmissing(df)

2X3 DataFrame

Row | I x y

| Int64 Int64 String

------------------------------------------------

1 | 4 2 d

2| 5 1 e

One more point i.e Missings.jl package provide the few functions inorder to work with missing values.

Julia> using Missing

Julia> Missings.replace(x,1)

Missings.EachReplaceMissing{Array{Union{Misssing, Int64}, 1}, Int64}(Union{Missing, Int64}[1,2,missing], 1)

These are some basic functions used to handle the data while analyzing i.e mainly to remove null and missing values from the data set. This is what data munging.

**Building a Predictive ML Model**

Till now, we have saw how the data set should be handle , how to overcome the problems especially like missing values in the data set or null values and more-ever visualizing the data using library plot.pl, StatPlots.

Now, we will see how to build an Machine Learning Model using Julia programming language.

In python scikitlearn is the package or library that will provides all the necessary models , similarly in Julia Scikitlearn package will provides.

Julia> Pkg.add(“Scikitlearn.jl”)

This package will act as interface to the python’s Scikitlearn package

*“ Since Julia can access Packages of Python”*

**Label Encoder**

In python labelencoder() is the package that can be found from **Scikitlearn.Preprocessing** which will converts data into numerical format data [0,1,2…………….]

In Julia also we will convert the data into numerical format. The one who are familiar with python they can understand why label encoder is used.(it becomes easy to access any column of data with numerical values).

Lets encode sample data

using ScikitLearn

@sk\_import preprocessing: LabelEncoder

encoder = LabelEncoder()

data = [“apple”, “orange”, “papaya”]

for col in data

train[data] = fit\_transform! (encoder, train[data])

end

Now, we will define generic classification function which takes model as input and gives us the accuracy and cross-validation scores.

using ScikitLearn : fit!, predict, @sk\_import, fit\_transform!

@sk\_import preprocessing : LabelEncoder

@sk\_import model\_selection : cross\_val\_score

@sk\_import metrics: accuracy\_score

@sk\_import linear\_model: LogisticRegression

@sk\_import ensemble: RandomForestClassifier

@sk\_import tree: DecisionTreeClassifier

function classification\_Model(model, predictions)

p = convert(Array, train[:13])

q = convert(Array, train[predictions])

r = convert(Array, test[predictions])

# check for fitness of model

fit! (model, p, 1)

#predicitons on training data set

Predictions = predict(model, p)

#accuracy

Accuracy = accuracy(Predictions, q)

#cross\_validation

Cross\_score = cross\_val\_score(model, p, q, cv = 5)

#print cross score

print(“cross score: “, mean(Cross\_score))

fit!(model, p, q)

Out = predict(model, r)

Return Out

End

**Logistic Regression**

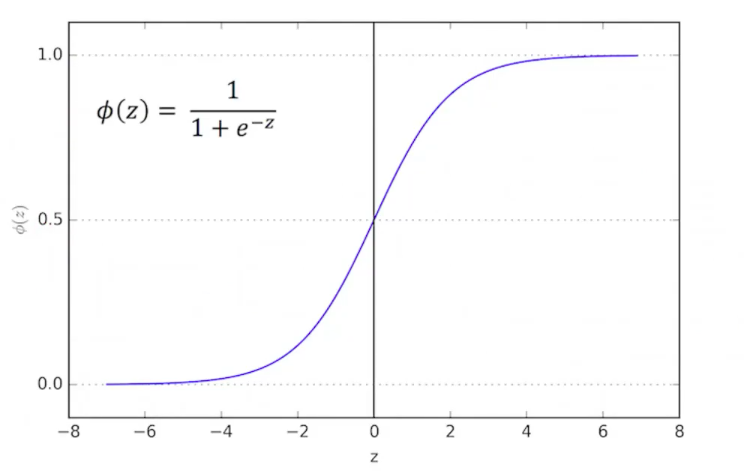
Using logistic regression we are going to predict the accuracy and cross validation scores like what we have done in the above classification\_Model function.

LogisticRegression in Julia is similar to Python. Logistic Regression in Machine Learning is an classification algorithm which is used to predict the probability of dependent categorial value. The dependent values will be either in 0 or 1.

Logistic Regression can be classifies into two classifications

1. Binary Classification
2. Multiclass Classification

Lets see the logistic regression plot in visual



Mathematical Equation For Logistic Regression : **1/ 1+ e^-x (or) 1/ 1 + e^-z**

lets make use of model and determine the accuracy for the persons obesity

model = LogisticRegression()

predict\_value = [:Obesity] => this code snippet add as

classification\_Model(model, predict\_value) continuation to above code

The result will be :

Accuracy: 80.9% Cross-Validation Score: 80%

The accuracy and cross\_score are good but if you need more accuracy then change the column or variables and apply model again.

Predict\_value = [:Obesity, :Age, :Weight]

Classification\_Model(model,predict\_value)

The result wil be :

Accuracy: 88% Cross-Validation-Score: 87.9%

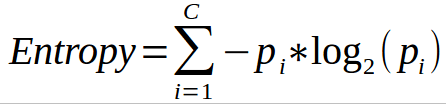
This how the prediction’s are done with logistic regression. Generally problems which are not ended at particular limit instead they tend to change frequently for those problems Logistic Regression Model is used to solve.

**Decision Tree**

Decision Tree is another Model under Classification. Decision Tree works on parent child scenario, always the child node will be consider as the result node vice-versa parent node is consider as root node which takes decisions. The working process of decision tree

* Decision tree selects best attribute using Attribute Selection Measure
* Selected attribute will be consider as root node
* Then again it divides into sub nodes until it reaches to leaf node

The mathematic equations or formulae used in decision tree are:

* **Information Gain (ig)** = **-p/s log(p/s) – n/s log(p/s)**
* 
* **Gini Index = ig – Entropy**

**Information Gain:**

This will gives us the information regarding an attribute i.e how important an attribute to the data set as of attribute posses feature od vectors through which we can identify the relations of parent and child nodes.

**Entropy**

Entropy , we can get this from information gain such that information gain will gives us the

entire relation of data set whereas the entropy will tells us the impurities from the data set.

The higher entropy the more information gain.

Let’s say two classes and we want to find the which class belongs to same category

Suppose class A belongs to some x category and B also same category x then it is not

a good entropy as 0. if it is like 50 – 50 % then it is good entropy and data set is good as 1

**Gini Index**

**Gini Index** will gives the pure impurity which means it will calculate the probability of s

Selected attribute if all are linked to same attribute then that attribute is pure attribute or

Belongs to same classs.

Decision tree gives higher accuracy than logistic regression , since decision tree follows the parent and child concept by taking exact decision.

Let’s see the implementation part for decision tree:

model = DecisionTreeClassifier()

predict\_value = [:Student, :Name, :Age]

classification\_Model(model, predict\_value)

The result will be as:

Accuracy: 81.95% Cross-Validation Score: 75.6%

We can again increase the accuracy to more extent by changing the input columns so that maximum accuracy can be obtained.

“***Always find maximum accuracy and score”***

Predict\_value = [:Student, :Name, :Class, :Age]

Classification\_Model(model, predict\_value)

The result will be as:

Accuracy: 85.78% Cross-Validation Score: 80.7%

**Random Forest**

Random Forest, it is an another algorithm that is capable of performing both regression as well as classification tasks with a technique called “Bootstrap” and “Aggregation” known as bagging.

Random Forest having multiple decision trees as its learning models then it performs random row sampling and feature sampling to the dataset by applying a model. This is called as Bootstrap.

Let’s see the approach or process involved to use random forest algorithm

* We should design a relevant question to the given information or data set
* And one more thing to make sure is convert all the data to accessible format or else convert into that format
* Develop a machine learning model
* categorize data set into training data and test data
* Apply model and find the accuracy or score for the testing data
* Repeatedly change the values so that accuracy will reach to max

Let’s see the implementation part of Random Forest

model = RandomForestClassifier(n\_value = 100)

predictions = [:Name, :Age, :Sex, :Loan]

classification\_Model(model, prediction)

Accuracy : 100% Cross-Validation Score : 80%

Here, we got 100% accuracy for the training data set. This is the problem overfitting and can be resolved in two ways

1. Reducing the number of predictions
2. Tuning the model parameters

model = RandomForestClassifier(n\_value = 100, min\_samples\_split = 50, max\_depth = 20,

n\_jobs = 1 )

classification\_Model(model, predictions)

The result will be :

Accuracy : 83% Cross-Validation Score : 80%

Here if you see even though accuracy is reduced the score is increased which means the model is doing well Random Forest will use multiple decision trees which in return gives different predictions.

As possible as avoid complex modelling technique as black box without understanding the concepts.

**Using R and Python Libraries in Julia**

Julia programming language is such a powerful language with many libraries and packages included as well as it also provides outside libraries to be accesses.

You may get doubt like if Julia is has such powerful libraries then why is needed to access from other languages especially Python and R because even the libraries are there but they might be very young to be used that’s the reason Julia provides ways to access libraries from R and python.

To call python libraries in the Julia PyCall is the package that will enables to call python libraries from Julia code

Julia> Pkg.add(“PyCall”).

**PyCall** provides many good functionality that helps in manipulating python in Julia using type **PyObject**

The following are the steps to be followed in order to call python packages

Step1: using Pkg

Step2: Pkg.add(“PyCall”)

Step3: using PyCall

Step4: @pyimport python\_library\_name

Lets see basic programe on how to import math package of python into Julia

using Pkg

Pkg.add(“PyCall”)

using PyCall

@pyimport math

Print(math.cos(90))

Second example to import Numpy package into Julia language

using Pkg

Pkg.add(“PyCall”)

using PyCall

@pyimport numpy

A = numpy.array([2,1,4,3,

5,7,6,8])

Print(A)

Output:

[2, 1, 4, 3, 5, 7, 6, 8]

**Using Pandas With Julia**

If you are familiar with the library pandas in python then it is same as Julia also. Using Pandas we can filter the data or analyze the data lot more. Especially converting the data into dataframes which is package of pandas library .

DataFrames will helps to visualize the data into multidimensional array i.e matrix format

Julia> Pkg.add(“Pandas”)

Lets see an example using pandas with Julia

using pandas

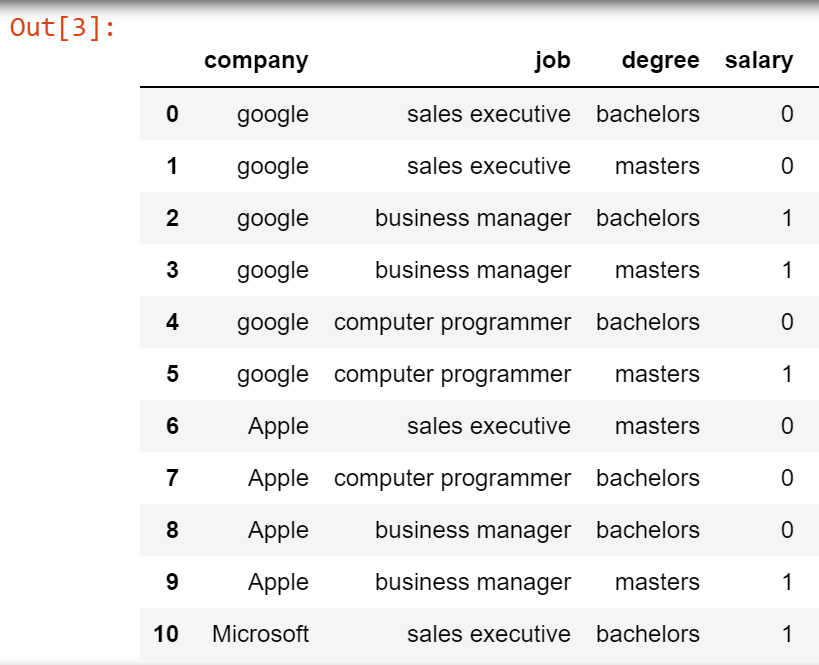
df = read\_csv(“job.csv”)



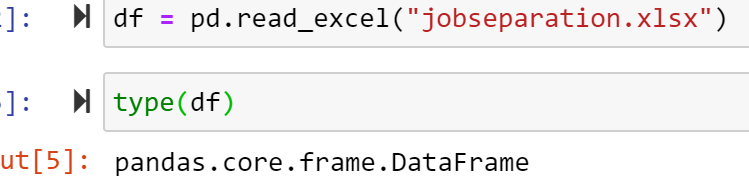
df = DataFrame(Dict(:company => [“google”, “Apple”, “Microsoft”], :job=>[“sales executive”,

“business manager”, “business manager”, “computer manager”],

:degree=>[“bachelors”, “masters”], :salary=>[0,1]))

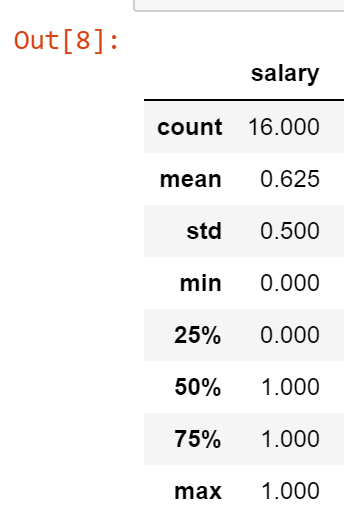


typeof(df)



head(df) # will gives first five rows of data

describe(df)



If df[“job”] == “computer manager”

df[“job”] = “manager”

end

df.mean(“salary”, axis = 1)

So, there are many operations which are basics of pandas and are used on data set as cleaning procedure .

Cleaning includes like removing null values, missing values replacement and modifying the data which is in appropriate .

Pandas is most powerful library not only in python but also in Julia .

**Using ggplot2 in Julia**

ggplot2 is an data visualization package used in statistical programming language R. ggplot will breaks the data into semantic components such as scales and layers.

Since, Julia can access the libraries of python and R so ggplot2 can be installed with Julia and include.

Lets see how to load R package into Julia

Using RCall

@rlibrary ggplot2

There might be question araise like having most powerful Julia with all packages include why to use R packages for data visualization ?

Plots.jl is powerful package but unfortunately its interface is similar to R language . If user wants to visualize the plot then it is very difficult to remember all the commands as there are more to remember .

So that’s the reason why Julia uses R packages for data visualization even python libraries too.

Lets consider an example with this scenario:

**Using Julia plot.jl package**

plot(plot\_data\_1, a = “a”, b = “b”, Geom.line,

layer(Geom.line, a = “a”, b = “text” , Theme(default\_color = “red”)),

layer(Geom.line, a = “a”, b = “a\_mc”, Theme(default\_color = “blue”)),

layer(Geom.line, a = “a”, b = “a\_mf”, Theme(default\_color = “orange”)),

}

**Using R ggplot package**

ggplot(plot\_data\_1, aes(a = “a”, b = “b”)) +

geom\_line(color = “red”) +

geom\_line(aes(b = :a\_mc), color = “green”) +

geom\_line(aes(b =:a\_mf), color = “violet”)

if u observe above piece of code using ggplot which is very simper when compared to Julia plots.jl . The user wont get frustrated on using R package as it is simpler than Julia package

The above code might be have some issues since, Gadfly do not follow grammer of graphics strictly like font size, data visualizing pattern, color pattern on the line etc.

By considering all these we can say at the end of day that packages of Julia are bit complex than the packages of R or python . R packages gives good interoperability and difficulty problems can be solved easily.

The package ggplot in Julia installed as follows:

Julia> Pkg.add(“RDatasets.jl”)

Julia> Pkg.add(“RCall.jl”)

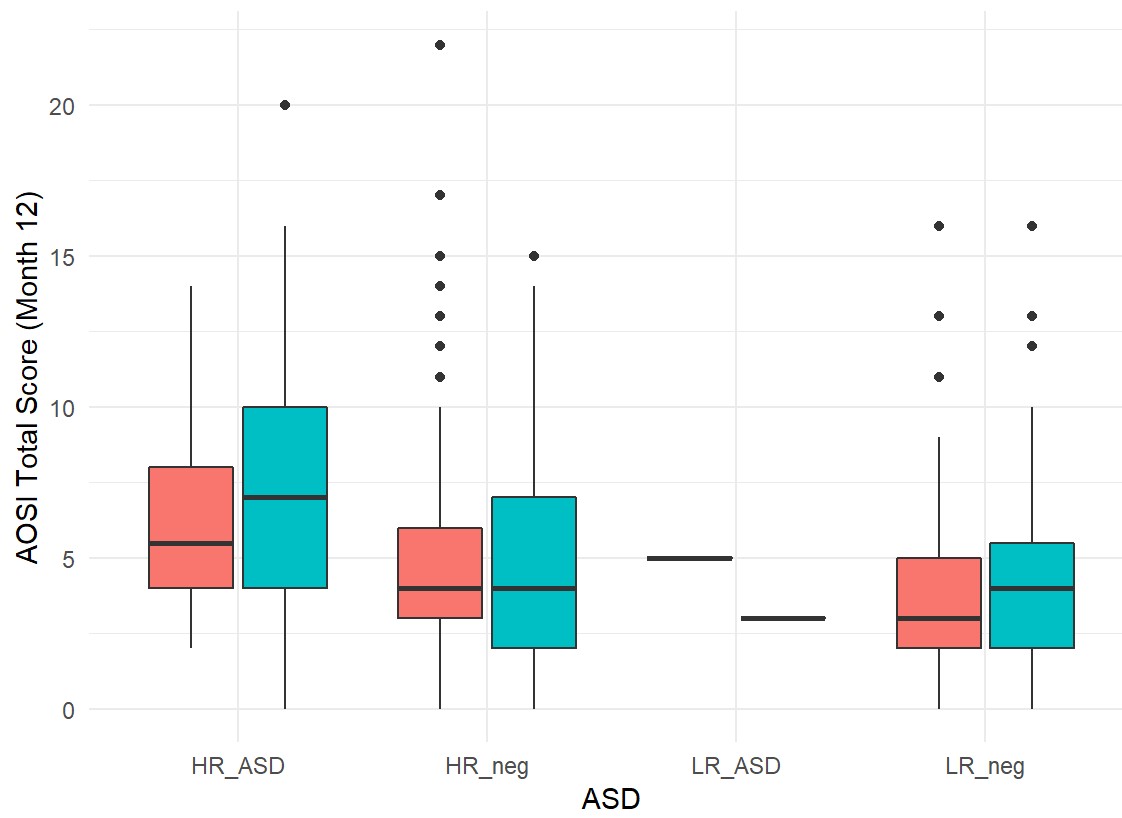
Lets look on the plot visualized using ggplot library

using Rcall, RDatasets

val = datasets(“datasets”, “demo”)

library(ggplot2)

ggplot($demo, aes(p =”ASD” , q =”AOSI Total Score(Month 12)” )) + geom\_print()



**Thoughts of Conclusion**

Finally Julia is such powerful language that provides accessability packages related to python and R by PyCall and RCall . Julia is ideal in its nature and its syntax too compared to python particularly when writing highly function code .

Julia is better programming language we can say . Strong reason might be its best for numerical computation

**“Technology Never Stops instead it flows like Water”**