Lab 2: Julia Quickstart

Functions, Logic, and Packages

Taigen Hunter-Galvan

Fri., Jan. 19

Table of contents

## 1 First steps

We start by loading the packages we will use in this lab

using CSV  
using DataFrames  
using DataFramesMeta  
using Dates  
using Plots  
using StatsBase: mean  
using StatsPlots  
using Unitful

## 2 Defining a function

In <index.qmd>, we read in a CSV file from scratch. However, we’d like to repeat this process for each year of data, and to do it in a consistent way so that we can read in the data for all available years into a single file. To do this, we’ll write a *function* that we can use to read in the data for any year. Specifically, our function will take in the year as an argument, and return a DataFrame with the data for that year.

Before we do that, let’s define a function that will return the filename for a given year. It’s often valuable to stack several functions together.

get\_fname(year::Int) = "data/tidesandcurrents-8638610-$(year)-NAVD-GMT-metric.csv"

Now we’re ready to define our function:

function get\_fname(year::Int)  
 return "data/tidesandcurrents-8638610-$(year)-NAVD-GMT-metric.csv"  
end  
  
function read\_tides(year::Int)  
 # define the CSV file corresponding to our year of choice  
 fname = get\_fname(year)  
  
 # a constant, don't change this  
 date\_format = Dates.DateFormat("yyyy-mm-dd HH:MM")  
   
 # 1. read in the CSV file and save as a dataframe  
 df = CSV.read(fname, DataFrame)  
   
 # 2. convert the "Date Time" column to a DateTime object  
 df[!, "Date Time"] = Dates.DateTime.(df[!, "Date Time"], date\_format)  
   
 # 3. convert the " Water Level" column to meters  
 df[!, " Water Level"] .\*= 1u"m"  
   
 # 4. rename the columns to "datetime" and "lsl"  
 rename!(df, Symbol("Date Time") => :datetime, Symbol(" Water Level") => :lsl)  
  
  
   
 # 5. select the "datetime" and "lsl" columns  
 df = select(df, :datetime, :lsl)  
   
 # 6. return the dataframe  
 return df  
end  
  
# print out the first 10 rows of the 1928 data  
first(read\_tides(1928), 10)

|  |  |
| --- | --- |
|  | **Instructions**  Fill out this function. Your function should implement the six steps indicated in the instructions. Use the example code from <index.qmd> to help you. When it’s done, convert it to a live code block by replacing ```julia``` with ```{julia}```. When you run this code, it should print out the first 10 rows of the 1928 data. Make sure they look right! |

## 3 Building the dataset

Now that we have the ability to read in the data corresponding to any year, we can read them all in and combine into a single DataFrame. First, let’s read in all the data.

|  |  |
| --- | --- |
|  | **Instructions**   1. **Hint**: to *vectorize* a function means to apply it to each element of a vector. For example, f.(x) will apply the function f to each element of the vector x. This is a very common operation in Julia! 2. Update the code blocks below, then replace ```julia``` with ```{julia}```. |

years = 1928:2021 # all the years of data  
annual\_data = [read\_tides(year) for year in years] # call the read\_tides function on each year (see hint above!)  
typeof(annual\_data) # should be a vector of DataFrames

Vector{DataFrame} (alias for Array{DataFrame, 1})

Next, we’ll use the vcat function to combine all the data into a single DataFrame.

df = vcat(annual\_data...)  
first(df, 5)

And we can look at the last 5 rows

last(df, 5)

Finally, we’ll make sure we drop any missing data.

dropmissing!(df) # drop any missing data

## 4 Plots

1. Plot the hourly water levels for March 2020, using subsetting and plotting techniques from the instructions
2. In the instructions, we plotted the average monthly water level from each month using groupby. Repeat this analysis, using the full dataset (all years).
3. Now repeat the analysis, but group by day of the year. What do you notice? (**Hint**: use Dates.dayofyear to get the day of the year from a DateTime object)

# Create a new column called :month  
df[!, :month] = Dates.month.(df.datetime)  
  
# Discard rows with missing values in :lsl  
dropmissing!(df, :lsl)  
  
# Group the data by month  
df\_bymonth = groupby(df, :month)  
  
# Calculate the mean of the :lsl column for each month  
df\_climatology = combine(df\_bymonth, :lsl => mean => :lsl\_avg)  
  
# Plot the climatology  
plot(df\_climatology.month,  
 df\_climatology.lsl\_avg;  
 xticks=1:12,  
 xlabel="Month",  
 ylabel="Average Water Level",  
 linewidth=3,  
 label=false,)

# Group the data by day of the year  
df[!, :dayofyear] = Dates.dayofyear.(df.datetime)   
df\_byday = groupby(df, :dayofyear)   
  
# Calculate the mean of the :lsl column for each day  
daily\_avg\_data = combine(df\_byday, :lsl => mean => :avg\_lsl)   
  
# Plot the average daily water levels  
plot(daily\_avg\_data.dayofyear,   
 daily\_avg\_data.avg\_lsl;  
 xlabel="Day of the Year",  
 ylabel="Average Water Level (m)",  
 label="Average Daily Water Levels",  
 legend=:topleft)

# Subsetting for March 2020  
t\_start = DateTime(2020, 3, 1, 0)  
t\_end = DateTime(2020, 3, 31, 23)  
march\_2020\_df = @subset(df, t\_start .<= :datetime .<= t\_end)  
  
# Plotting hourly water levels for March 2020  
plot( march\_2020\_df.datetime,  
 march\_2020\_df.lsl;  
 xlabel="Date",  
 ylabel="Water Level (m)",  
 label="Hourly Water Levels for March 2020",  
 legend=:topleft)