Weather_Mini_Project

paul

March 23, 2017

Description:

This weather analysis is to scrape the Environment Canada website for the hourly updated weather and 24 hour forecast using an automated script.

The overall purpose is to be introduced to web scraping as well as basic data exploration.

Learning to WebScrape:

- rvest
- rvest CRAN
- selecting css tags
- xml2

Automated Script:

___ Script for scraping the web is a R file combining step 1 and step 2. It is saved in the same folder as this report, under `WeatherScript.R`.___

1: Set up a csv file for collecting data, made into a function. Which initializes the file and returns the file path (Saved under working directory).

```
    InitializeFile = function(forecast_Hours=24, forecast_Title="temp",
                   unit="(C)", sep=","){

        file_Path<-paste(forecast_Title, ".csv", sep="")

        if(!file.exists(file_Path)){
            file.create(file_Path)

            ## making the column names:
            col_Names <-c("Date",paste(
                "current ",forecast_Title,unit, sep=""
            ))
            for(i in 1:forecast_Hours){
                    forecast_Name = paste("forcast ",forecast_Title," in ",
                    i,"h ",unit,sep="")
                    col_Names <- append(col_Names,forecast_Name)
            }

            ## adding title to the file:
```

```
        titles = as.data.frame(as.list(col_Names),stringsAsFactors = FALSE,
                               col.names = NULL, fix.empty.names = FALSE)
        write.table(titles, file_Path, sep=sep,
                    append=TRUE,col.names = FALSE, row.names = FALSE)
    }


     return(file_Path)
 }
```

2: getting the data and appending it to the file: * The file is named `temperature.csv`

```
library(rvest)
library(xml2)

file_Path = normalizePath(InitializeFile(forecast_Title = "temperature"))


forecast <- "https://weather.gc.ca/forecast/hourly/on-118_metric_e.html"
current_Weather <- "https://weather.gc.ca/city/pages/on-118_metric_e.html"
sep <- ","

forecast_xml <- read_html(x=forecast)

#list of temperature forecast
temperature_css <- ".text-center:nth-child(2)"
temperature_xml <- rvest::html_nodes(forecast_xml,temperature_css)
temperature_Forecast <- lapply(temperature_xml, function(x){
    rvest::html_text(x)
})

# Getting current Temperature

current_Temp_css <-"div.col-xs-6 p.lead span.wxo-metric-hide"
current_Temp <- read_html(current_Weather) %>%
    rvest::html_nodes(current_Temp_css) %>%
    rvest::html_text()

# Formatting the captured results
temperature_Forecast <- as.numeric(as.vector(
    temperature_Forecast))
current_Temp <- as.numeric(gsub("[^-0-9]+","",
            current_Temp))
time <- Sys.time()
data <- append(current_Temp,temperature_Forecast)
data <- as.data.frame(as.list(data),stringsAsFactors = FALSE,
                    col.names = NULL, fix.empty.names = FALSE)
data <- cbind(time,data)

# writing to the file

write.table(data, file = file_Path,append=TRUE,
            col.names = FALSE, row.names = FALSE,
```

```
                sep=sep)
```

Scheduling Tasks:

- taskScheduleR
  - provides basic functionality to schedule R scripts.
  - includes a plugin accessible under the *addins* at the menu bar.
  - be sure to first stop the task and then delete it.
  - be sure to change the time format to local time format on the computer.
- windows schtasks.ext
  - command prompt tool that provides more control.
  - taskScheduleR is based on this tool.
- scripting
  - Essentially comes down to modifying a xml file outlining the task.
  - Default location: `C:\Windows\System32\Tasks`.
  - task name must be unique.
- xml2 package

3: Trying to directly use taskscheduleR

- It was observed that when the laptop was unplugged from the power source, the script to collect data was not run. The solution is to modify the `StopIfGoingOnBatteries` and `DisallowStartIfOnBatteries` parameters in the script file from the initial setting `true` to `false`. The default location for the windows task scheduler files is `C:\Windows\System32\Tasks`

```
library(taskscheduleR)
file_Path = normalizePath(InitializeFile(forecast_Title="temperature1"))
script_Path = ".\\WeatherScript.R"
task_Name= "testing"

##  MAKE SURE THAT THE DATE FORMAT IS CHANGED TO LOCAL COMPUTER FORMAT
taskscheduler_create(taskname = task_Name, rscript=script_Path,
                     schedule = "MINUTE", startdate=format(Sys.Date(), "%d/%m/%Y"),
                     modifier = 2)
```

```
## [1] "SUCCESS: The scheduled task \"testing\" has successfully been created."
```

```
## modifying task file
library(xml2)
script_Path = paste("C:\\Windows\\System32\\Tasks\\",task_Name,sep="")

##formating xml parameters
## Name search does not seem to be working.
changed_File = read_xml(x=script_Path, econding="UTF-16")
xml_Nodes =xml_children(xml_children(changed_File))
xml_text(xml_Nodes[6])<- "false"
xml_text(xml_Nodes[7])<- "false"


## Stoping old task so a modified version can begin.
taskcheduler_stop(task_Name)
taskscheduler_delete(task_Name)
```

```
write_xml(changed_File,script_Path,encoding="UTF-16")

## running the task scheduler with new configured file
command = paste("schtasks.exe /create /tn",task_Name,
                "/XML",script_Path)
system(command,intern=TRUE)
```

```
## [1] "SUCCESS: The scheduled task \"testing\" has successfully been created."
```

```
    taskscheduler_stop(task_Name)
    taskscheduler_delete(task_Name)
```

Simple Processing of Collected Data:

Produce a simple graph to visualize the data.

1: Lag Columns: * The raw collected data is stored under `oneDay.csv`. Which contains the collected temperature and forecast data for a single day. This will be used to test best processing and plotting of raw data.

- The desired data would contain the current temperature followed by the current temperature that was predicted 1 hour ago, 2 hours ago.., 24 hours ago in the same row. Essentially, lag the forecast1hour column by 1, forecast2hour column by 2... forecast24hour column by 24. This will introduce NA values as place holders.

```
    rm(list = ls())
    library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
    # Reading in dataframe
    file_Path = "oneDay.csv"
    data = read.table(file = file_Path, header=TRUE,
                      sep=",",stringsAsFactors = FALSE)


    # Data was collected every half hour,Seperate into
    # 2 pieces: one data_frame for the first half hour
    # and another for the second half hour

    length=dim(data)[1]
    first = data[seq(from=1,to=length,by=2),]
    second = data[seq(from=2,to=length,by=2),]
```

```
# Lagging each column, starting from 3rd column
# made into a quick function.
LagData = function(data,start_Col=3, lag=1, lag_Increment=1){

    for( i in start_Col: length(colnames(data))){
        data[,start_Col] = data.table::shift(
            x= data[,start_Col], n=lag
        )
        start_Col = start_Col+1
        lag =lag+lag_Increment
    }

    return(data)
}

##  Took the data for the first half hours and lag the data
lagged = LagData(data=first, lag_Increment = 1)
```

2: Getting Complete Data (Removing rows containing NA):

```
library(dplyr)

##  Select All Full Length Data (exclude the first 24 hours):

##  number indicating how many rows to skip
skip= 25

## For simplicity, just the first half hour data will
## be analyazed.


data = lagged%>%
    filter(!is.na(forcast.temperature.in.24h..C.))
```

3: Learning about ggplot2:

- ggplot2
- ggplot2 cheat sheet
- Interesting Course On ggplot2

4: Making Organized Data Tables:

- The organized data table should contain the following in each row:
  - *Date:* the time and day of the collected temperature.
  - *current:* the actual temperature at that date.
  - *for1h,for2h,. . . ,for24h:* The forecasted temperature for the current temperature 1 hour ago, 2 hours ago,. . . 24hours ago.
  - *maxPredict:* Maximum temperature predicted.
  - *minPredict:* Minimum temperature predicted.
  - *erUp:* The difference between maxPredict and current.
  - *erDown:* The difference between minPredict and current.
  - *erUpFreq:* The number of predications that lies above the current temperature.
  - *erDownFreq:* The number of predications that lies below the current temperature.
- The organized data is saved under `finished.csv`.

```
##  Changing the column names:
col_Names = c("Date","current")
for(i in 3:length(names(data))){
    col_Names = append(col_Names, paste("For",i-2,"h",sep=""))
}
colnames(data) = col_Names

##  Getting Formatted Data:
library(dplyr)

##  Max and Min predictions
max = apply(select(data,-(Date)),1,function(x){ max(x)} )
min = apply(select(data,-(Date)),1,function(x){ min(x)} )


formated = data %>%
    mutate(maxPredict = max) %>%
    mutate(minPredict = min) %>%
##  Error Bars
    mutate(erUp = pmax(0,(maxPredict-current))) %>%
    mutate(erDown=pmin(0,(minPredict-current)))


## frequencies count for how many predictions lay above
##  or below the actual temeperature

arbConstant = 0.1
arbConstant2 = 0.2 ## Used to offset the
## boundaries for the cut function such that the actual
## temperature is not included. Since only whole numbers,
##  a decimal to tenth place is sufficient

current=grep("current", names(formated))-1 ## subtract 1 from the index
##because date will be discarded later.

max = grep("maxPredict", names(formated))-1
min = grep("minPredict", names(formated))-1
for1h = grep("For1h", names(formated))-1
for24h = grep("For24h", names(formated))-1


##  Special Attention needs to be given because
##  lower bound is not included, but upper is for
##  the cut function.
erUpFreq = apply(formated, 1, function(x){
    x = as.numeric(x[2:length(x)])
    table(cut(x[for1h:for24h],breaks=
         c(x[current]+arbConstant,
           max(x[max],x[current]) )))
})


erDownFreq = apply(formated, 1, function(x){
```

```
    x = as.numeric(x[2:length(x)])
    min = min(x[min], x[current])-arbConstant2
   table= table(cut(x[for1h:for24h],
        breaks = c(min, x[current]-arbConstant)))
})

## Saving finished data

finished = formated %>%
    mutate(erUpFreq = erUpFreq) %>%
    mutate(erDownFreq = erDownFreq)

write.table(x=finished, file = "finished.csv",
            col.names = TRUE,row.names = FALSE,
            sep=",")
```

5: Plotting Graph:

- Basic time vs temperature series.

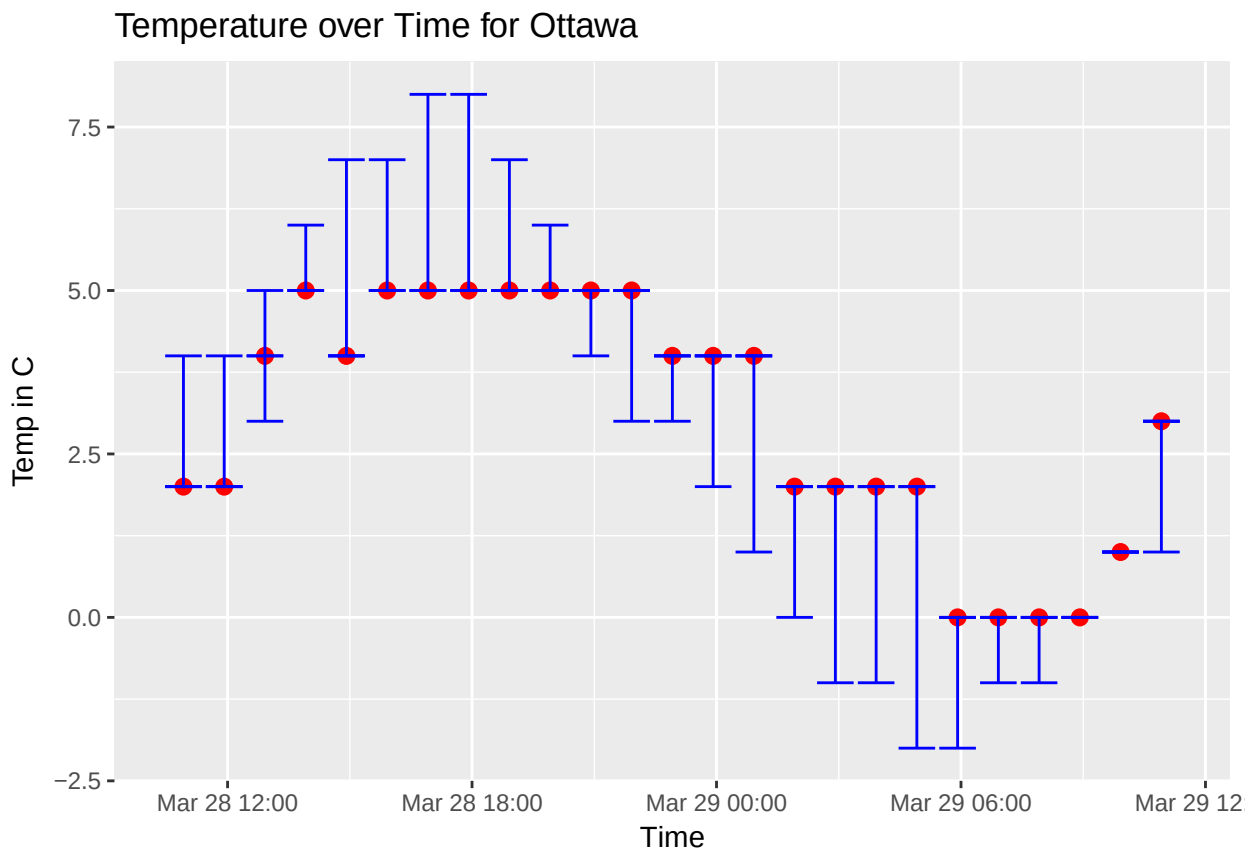
```
library(ggplot2)

table = read.table(file="finished.csv", header = TRUE,
                sep= ",", stringsAsFactors = FALSE)

sizeFactor = 20
size = table$erUpFreq
size2 = table$erDownFreq

plot = ggplot(data=table, aes(x=strptime(table$Date,format="%Y-%m-%d %H:%M:%S"),
        y=table$current) ) + geom_point(color="red",size=2.5)+
    scale_x_datetime() +
    ##  titles and things
    xlab("Time")+ylab("Temp in C")+
    ggtitle("Temperature over Time for Ottawa")+
    ##  Error Bar
    geom_errorbar(aes(ymin=table$current,
                      ymax = table$maxPredict),
                  color = "blue")+
        geom_errorbar(aes(ymin=table$minPredict,
                      ymax = table$current),
                      color="blue")

plot
```


Temperature over Time for Ottawa
Temp in C
7.5
5.0
2.5
0.0
−2.5
Mar 28 12:00
Mar 28 18:00
Mar 29 00:00
Mar 29 06:00
Mar 29 12
Time