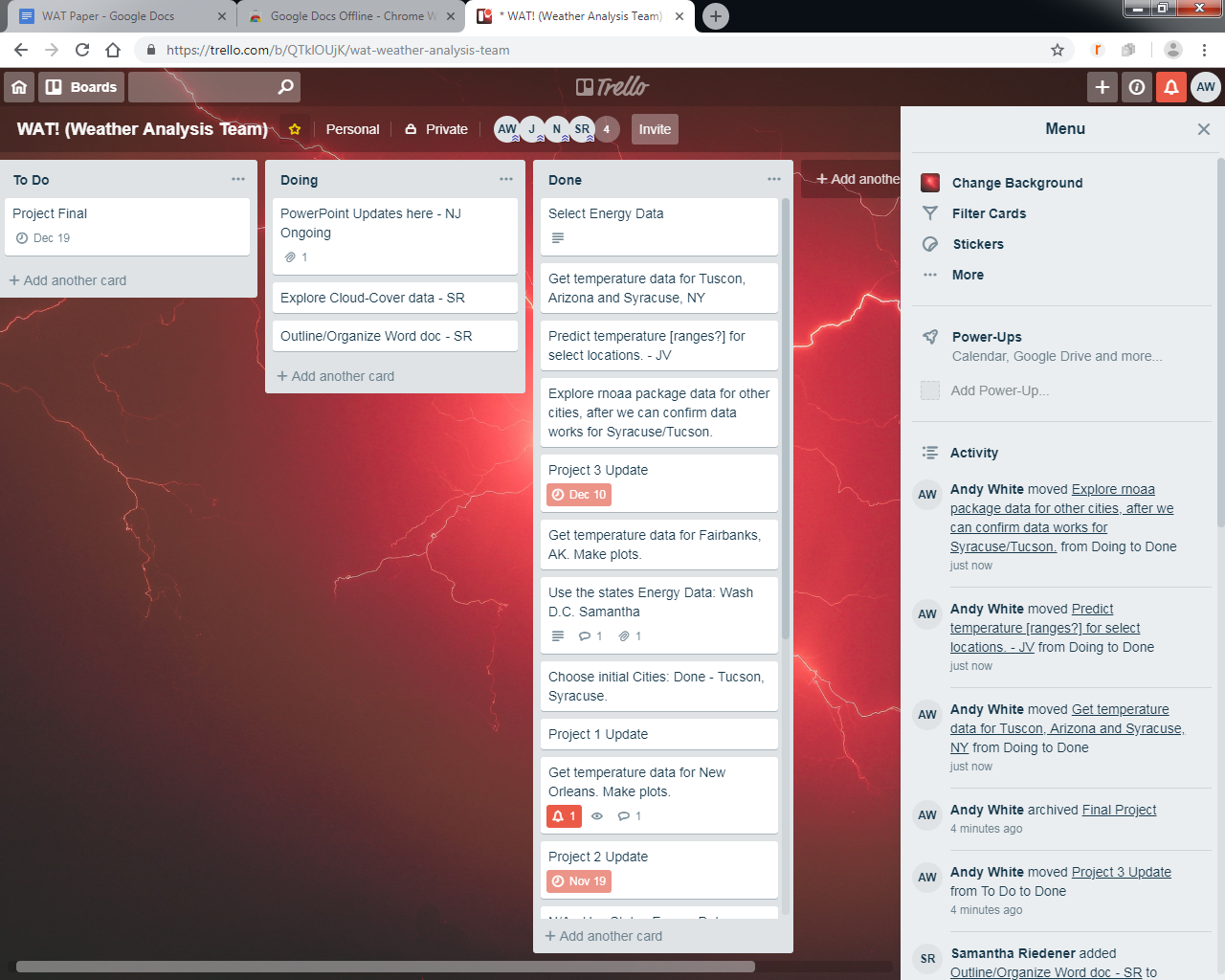
Introduction

As the Weather Analysis Team, our goal is to capture and analyze existing weather data that includes temperature, precipitation, and wind speed. We are accessing an online data source, National Oceanic Atmospheric Administration (NOAA), and locally running base routine scripts that grant access to the data source online, and will help predict temperature readings for different locations on any given date of the year.

Organization

We used trello.com for a free version of the Kanban methodology. We used this method during our weekly meetings held after class and on Friday evenings. This allowed us to visualize our remaining items, track Work-In-Progress, and complete tasks in an efficient manner. See below for a screenshot:



Data Collection

Data collection was complicated by a number of different constraints that occurred throughout the process. At the beginning of the collection process, we collected NOAA data through an online request system that would email an excel document with data based on the user’s requirements. This system took far too long to receive the data required and eventually we located an R Package called “rnoaa” that allowed us to access the NOAA database through an R request detailed below:

prcepitation <- ncdc(datasetid = "GHCND", stationid='GHCND:USW00023160', datatypeid='PRCP', startdate = '2010-05-01', enddate = '2010-10-31', limit=500)

To successfully run the code a user needed to obtain a personal “noaakey” and execute the following code:

library(rnoaa)

options(noaakey = "aRDlwGcDCHNulBwzeTmxCRBqhirzTKWK")

An important part of the code above was the Global Historical Climatology Network Daily (GHCND) access. This is the daily data collected from each individual station throughout the world. There were a number of other databases to choose from, GHCNM (monthly data), buoy data (data from buoys in the ocean), etc. We selected the daily data due to the granularity of weather patterns that we’re witnessed and potential trends we could isolate. For purposes of this project we focused solely on U.S. data due to reliability of data reported at each station. We then isolated a number of different stations with highly different weather patterns: Fairbanks, Alaska; Tucson, Arizona; Honolulu, HI; Syracuse, NY. Next, we had to locate the “Stationid” to pull the data we wanted. Each station that collects weather data has a unique ID associated with it, which means that this ID must be known for us to be able to call a particular data set. We would have liked to pull a large list from NOAA, but we could not locate a consolidated list of station-id’s like “USW00023160”. This was done individually through searching. However, we could pull different stations through use of the “ncdc\_stations” function as well.

“Datatypeid” allows a user to select the type of data being pulled. For instance, “PRCP” data is precipitation in tenths of millimeters. There is a range of other types of metadata, such as snowfall or maximum temperature on a given day. Please see the appendix for a complete list of these parameters. Finally, the start- and end-date needed to be input by the user as a string and in International Organization for Standardization (ISO) date format. However, NOAA limited users to only one year of data pull per execution. If a user wanted to pull more than one year of data (which we did), an error message was generated.

Next we needed to create a function that allowed a user to select the years and code based on a user’s requirements. This required a loop and a “dummy” dataframe.

data.df.index.start <- c(1900:2018)

data.df.index.start<- data.frame(data.df.index.start,"01","01")

colnames(data.df.index.start)<-c("Year","Month","Day")

data.df.index.start$startdate<-as.character(paste(data.df.index.start$Year,data.df.index.start$Month,data.df.index.start$Day,sep="-"))

data.df.index.start$enddate<-as.character(paste(data.df.index.start$Year,12,31,sep="-"))

startend <- data.df.index.start

The code above created a dataframe with a year, an ISO start-date (January 1st of each year) and an ISO end-date (December 31st of each year). Now we had a dummy dataframe with which we needed to be able to “attach” different datasets. Luckily, rnoaa already had a built in function for that which we tested:

tavg <- ncdc(datasetid = "GHCND", stationid='GHCND:USW00023160', datatypeid='TAVG', startdate = data.df.index.start[118,4], enddate = data.df.index.start[118,5], limit=500)

tavg2 <- ncdc(datasetid = "GHCND", stationid='GHCND:USW00023160', datatypeid='TAVG', startdate = data.df.index.start[119,4], enddate = '2018-12-31', limit=500)

df.tavg <- ncdc\_combine(tavg,tavg2)

Next, we needed to create a loop and function to bring everything together

noaapull1 <- function(year1,datatype,stationid1){

year1 <- match(year1,startend$Year)

pullstart <- ncdc(datasetid = "GHCND", stationid=stationid1, datatypeid=datatype, startdate = startend[year1,4], enddate = startend[year1,5], limit=500)

for (i in (year1+1):nrow(startend)) {

yearx <- ncdc(datasetid = "GHCND", stationid=stationid1, datatypeid=datatype, startdate = startend[i,4], enddate = startend[i,5], limit=500)

pullstart <- ncdc\_combine(pullstart,yearx)

}

return(pullstart)

}

We could plot the data (which was in the form of a list) with the following function from rnoaa:

test <- noaapull(2010,'AWND','GHCND:USW00023160')

ncdc\_plot(test)

However, this data was in the form of tibbles, and for this class we needed it to be in the form of a dataframe, so we had to add an additional workaround, where we would save the file as a csv, and then reread the data back into R. Final function:

noaapull1 <- function(year1,datatype,stationid1){

year1 <- match(year1,startend$Year)

pullstart <- ncdc(datasetid = "GHCND", stationid=stationid1, datatypeid=datatype, startdate = startend[year1,4], enddate = startend[year1,5], limit=500)

for (i in (year1+1):nrow(startend)) {

yearx <- ncdc(datasetid = "GHCND", stationid=stationid1, datatypeid=datatype, startdate = startend[i,4], enddate = startend[i,5], limit=500)

pullstart <- ncdc\_combine(pullstart,yearx)

}

pullstart<- pullstart[[1]]

write.csv(pullstart,file = paste(datatype,year1,stationid1))

return(read.csv(paste(datatype,year1,stationid1)))

}

Now we had access to any type of weather data we wanted, and we began analyzing.

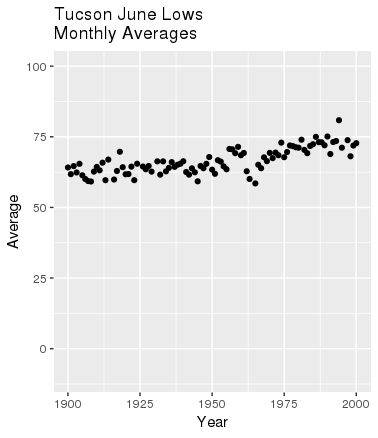
Analytic Techniques

Overview

Our goal was the ability to predict future temperatures for a variety of locations, using data from past years. We used the archive of temperature data at NOAA, as described above. We used linear regression and predictive tools that are available in R to perform analysis and predictions. We also analyzed the standard deviation of the data to understand the range of temperatures that were likely, around any prediction. Finally, we used ggplot2 to visually communicate our analysis and conclusions.

Details

The only temperature data that is publicly available, over a long period of time, for many locations, is high and low daily extreme temperatures. We chose those as a starting point.

Obviously, we expect January 24, 2015 to have a different high temperature than June 1, 1950, for the same location. But January 24, 2015 should have a high temperature that’s similar to January 24, 1950. To smooth out the regular irregularities, we chose to average the high temperatures for a particular month of a year, and store that average along with the similar averages for each year in the 20th century. For example, we create a data point that was the average of the high temperatures for Tucson, AZ in June, 1900, and for 1901, 1902, and so on, through 1999.

We used the ggplot2 package to create a graph depicting these monthly averages, one dot per year.

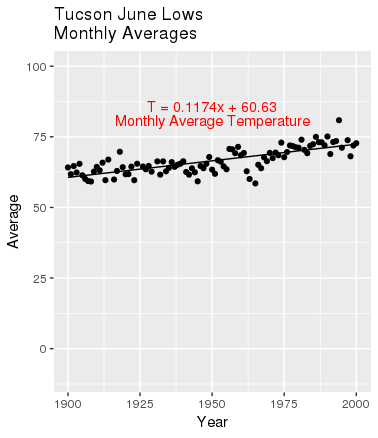
ggplot(statsMonDF, mapping=aes(x=Year))+

geom\_point(aes(y=Average), na.rm=TRUE)

With that set of data, we used linear regression to approximate the change in high temperatures - for March, for example - during the 20th century. We used the R function lm() for this:

avgYear <- statsMonDF$Year

avgLM <- lm(statsMonDF$Average~avgYear)



Adding a line to represent that regression on the graph required a dataframe. Because the code would do this several times, we created a function:

# lm2LineDF converts an lm and two x-values

# into a 2x2 DF of coordinate pairs.

lm2LineDF <- function(lm, year1, year2) {

icept <- lm$coefficients[1]

slope <- lm$coefficients[2]

yearEnds <- c(year1, year2)

valueEnds <- c(slope\*year1+icept, slope\*year2+icept)

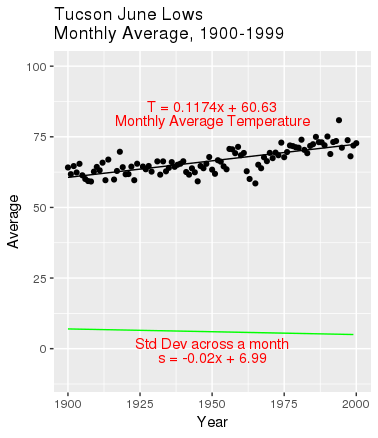
return (data.frame(yearEnds, valueEnds)) }

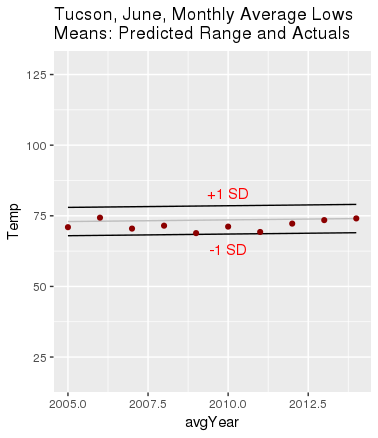
From this, we learned that the daily low temperatures in June, in Tucson, rose at an average of 12 degrees Fahrenheit per century during the 20th century!

We know that it is impossible to correctly predict the *exact* atmospheric temperature in advance. To acknowledge that there is always natural variability in temperature, from one day to the next, we wanted a range. For this we used standard deviation, planning to predict that the average high temperature in June, 2005 would fall within a certain range: one standard deviation. So we used tapply() to calculate the standard deviation of the high temperatures in one month of each year:

statsSD <- tapply(myMonth$EXTR[-1], myMonth$Year[-1], sd)

It was easy to add that to our plot:



To complete our prediction process, we used the predict() function to predict the high temperature for Tucson in June 2005 - and other years in the 21st century. We created a range of values around those predictions: the predicted value, plus or minus one standard deviation. That became our prediction range.

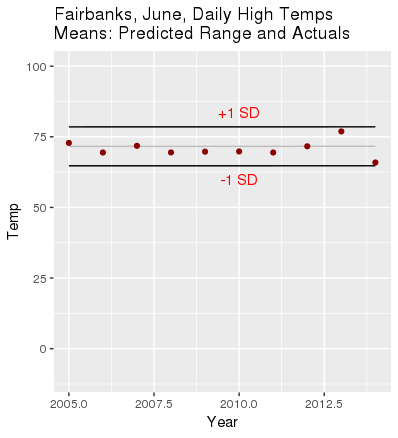
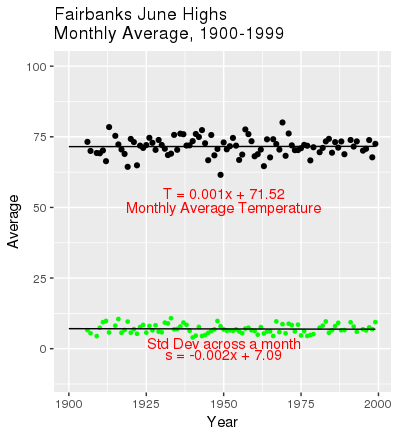
The plot to the right is typical for the locations and months that we reviewed. On almost every prediction plot, all of the actual values - shown as red dots - fall within the range bars - one standard deviation of the prediction centerline.

Results

Our results show interesting trends across the 20th century, and demonstrate the ability to correctly predict future air temperature values - within a range.

We think our results have good visual clarity. The historical plots show two modestly complicated concepts. The prediction plot drives home the predictability of atmospheric temperatures.

The rnoaa package greatly eases access to data, and we are thrilled to see this becoming common.



Complications

We attempted to collect data pertaining to energy consumption, energy usage, and/or energy production, however, this data was only available monthly, produced on a state level, and unreliable (many entries had no data and the data was not self consistent between years). Additionally, putting the data in a useable format required manual entry by downloading large PDF files that only reported two months worth of data over a 20 year time span. Given the number of constraints we decided to move forward with the NOAA data, which was plentiful and reliable.

We also ran into issues integrating the rnoaa code with the existing data analysis and prediction code. This was due to rnoaa producing data types known as tibbles, which were unfamiliar and incompatible with the existing framework of code. To work around this, the rnoaa generated information was exported as a csv file and then reread into R where it could then be analyzed.

Conclusion

To predict temperature data based on past values proved to be a reliable form of prediction. The amount by which temperatures changed, or stayed relatively the same, was surprising for each location as we expected to see a positive trend for each location. What was most shocking was seeing how the variability among the monthly averages decreased or stayed relatively stable over time, as we had anticipated the variability to increase in all cases.

We learned a lot along the way, especially how to find the correct data to utilize for the project. Locating station id’s was a particular challenge, we attempted to utilize the station id list noted in the appendix, however we learned along the way that data published at any number of stations was at times “hit-or-miss”. Dealing with NOAA or rnoaa data presented a number of challenges that sometimes could be overcome with code and sometimes with a rather manual effort. We found that we could limit the manual through frustration. As we ran into challenges, we had to combine skills from class learnings and utilize our frustrations to springboard into a workable solution.

In future models it would be beneficial to add additional predictor values such as precipitation amount, humidity, or atmospheric pressure, to get a more refined predictor of temperature. Additionally, being able to fully integrate the rnoaa code with the prediction code would make the process much more automated and streamlined.

Appendix

“Datatypeid” metadata information: <https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt>

We also utilized this station list to help locate different stations to choose from: <https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/ghcnd-inventory.txt>

Data collection code:  
#Part 1: Get data into ISO date to be ingested by noaa into dummy dataframe

data.df.index.start <- c(1900:2018)

data.df.index.start<- data.frame(data.df.index.start,"01","01")

colnames(data.df.index.start)<-c("Year","Month","Day")

data.df.index.start$startdate <- as.character(paste(data.df.index.start$Year,data.df.index.start$Month,data.df.index.start$Day,sep="-"))

data.df.index.start$enddate<- as.character(paste(data.df.index.start$Year,12,31,sep="-"))

strte

#first rename data df:

startend <- data.df.index.start

#the function we utilized to pull data

noaapull1 <- function(year1,datatype,stationid1){

year1 <- match(year1,startend$Year)

pullstart <- ncdc(datasetid = "GHCND", stationid=stationid1, datatypeid=datatype, startdate = startend[year1,4], enddate = startend[year1,5], limit=500)

for (i in (year1+1):nrow(startend)) {

yearx <- ncdc(datasetid = "GHCND", stationid=stationid1, datatypeid=datatype, startdate = startend[i,4], enddate = startend[i,5], limit=500)

pullstart <- ncdc\_combine(pullstart,yearx)

}

pullstart<- pullstart[[1]]

write.csv(pullstart,file = paste(datatype,year1,stationid1))

return(read.csv(paste(datatype,year1,stationid1)))

}

Data Analysis Code

#

# This code uses the ggplot2 library to draw graphs.

#

library(ggplot2)

# Directory is the directory ("folder") that contains the weather data,

# grouped first by location.

Directory='/users/jeff/DS/SyracuseU/IST-687-IntroDS/Work/Final Project/Data/Weather'

# "Location" is the location from which the weather data was collected.

Location='Syracuse'

myFile="SYR-AIR-20th.csv"

recentFile="SYR-AIR-20xx.csv"

MonthText="November"

MonthNum="-11-"

FirstYear=1900

MiddleYear=1950

LastYear=1999

PredBegin=2005

PredEnd=2014

ExtremeLabel="Highs" # Can be "Highs" or "Lows"

# lm2LineDF converts an lm and two x-values into a 2x2 DF of coordinate pairs.

lm2LineDF <- function(lm, year1, year2) {

icept <- lm$coefficients[1]

slope <- lm$coefficients[2]

yearEnds <- c(year1, year2)

valueEnds <- c(slope\*year1+icept, slope\*year2+icept)

return (data.frame(yearEnds, valueEnds))

}

WLocation <- paste(Directory, Location, sep='/')

setwd(WLocation)

myData <- read.csv(file=myFile, header=TRUE, sep=",")

# Find data for the month we want.

myData <- myData[grep(MonthNum, myData$DATE),]

# Remove the second column (station long name).

myMonth <- myData[-2]

# Extract 4-digit Year number.

myMonth$Year = as.numeric(substr(myMonth$DATE, 1, 4))

# Create new, temporary vector for highs or lows. (as.num(as.char()) handles negatives).

if (ExtremeLabel == "Highs") {

extreme <- myMonth$TMAX

} else {

extreme <- myMonth$TMIN

}

# Convert the string-numbers in quote marks (as.n(as.c()) needed to fix negatives).

myMonth$EXTR = as.numeric(as.character(extreme))

# Create a new DF, one row per year, with mean and StdDev, for ggplot.

statsYear <- unique(myMonth$Year)

statsExtr <- tapply(myMonth$EXTR[-1], myMonth$Year[-1], mean)

statsExtr <- unname(statsExtr)

statsSD <- tapply(myMonth$EXTR[-1], myMonth$Year[-1], sd)

statsSD <- unname(statsSD)

# Remove NA's because it trips up some functions.

statsSD <- statsSD[which(!is.na(statsExtr))]

statsYear <- statsYear[which(!is.na(statsExtr))]

statsExtr <- statsExtr[which(!is.na(statsExtr))]

statsMonDF <- data.frame("Year"=statsYear, "Average"=statsExtr, "SD"=statsSD)

# Perform linear regression on monthly means.

# (Unfortunately, predict() wants a clean name for a vector so we copy it out.)

avgYear <- statsMonDF$Year

avgLM <- lm(statsMonDF$Average~avgYear)

# Create 2x2 DF of (x,y) coordinate pairs, give the columns names, for ggplot.

dfAvgLR <- lm2LineDF (avgLM, 1900, 1999)

names(dfAvgLR)[1]<-"Year"

names(dfAvgLR)[2]<-"Averages"

# Text of formula to display on plot, and its position, near but not on dots.

tempSlope<-((dfAvgLR[2,2]-dfAvgLR[1,2])/(dfAvgLR[2,1]-dfAvgLR[1,1]))

tempFormula <- paste ("T = ", round(tempSlope, digits=4),

"x + ", round(dfAvgLR[1,2], digits=2), sep="")

if (dfAvgLR[1,2]<70) {

tempLabelY <- dfAvgLR[1,2]+25

} else {

tempLabelY <- dfAvgLR[1,2]-15

}

# Perform linear regression on monthly StdDev's, create DF for regressed line.

sdLM <- lm(statsMonDF$SD~statsMonDF$Year)

dfSDLR <- lm2LineDF (sdLM, 1900, 1999)

names(dfSDLR)[1]<-"Year"

names(dfSDLR)[2]<-"StdDev"

# Text of formula to display on plot, and its position, near but not on dots.

SDSlope<-((dfSDLR[2,2]-dfSDLR[1,2])/(dfSDLR[2,1]-dfSDLR[1,1]))

SDFormula <- paste ("s = ", round(SDSlope, digits=4),

"x + ", round(dfSDLR[1,2], digits=2), sep="")

if (dfSDLR[1,2] > dfAvgLR[1,2]) {

sdLabelY <- dfSDLR[1,2]+15

} else {

sdLabelY <- dfSDLR[1,2]-5

}

# Title for the historic plot.

myTitle <- paste(MonthText, " ", ExtremeLabel, "\nMonthly Average, ", FirstYear, "-", LastYear, sep="")

# Automatically choose minimum and maximum values for y-axis,

# preferring (-10:100).

yScaleLow=-10

yScaleHigh=100

if (dfAvgLR[1,2] >95) {

yScaleHigh=dfAvgLR[1,2]+20

}

if (dfAvgLR[1,2] <5) {

yScaleLow=dfAvgLR[1,2]-20

yScaleHigh=yScaleLow+110

}

ggplot(statsMonDF, mapping=aes(x=Year)) +

geom\_point(aes(y=Average), na.rm=TRUE) +

geom\_line(data = dfAvgLR, mapping=aes(x=Year, y=Averages) ) +

ggtitle(paste(Location, myTitle)) +

annotate(geom="text", x=MiddleYear, y=tempLabelY, label=tempFormula,

color="red", angle=0) +

annotate(geom="text", x=MiddleYear, y=tempLabelY-5,

label="Monthly Average Temperature", color="red", angle=0) +

geom\_line(data = dfSDLR, mapping=aes(x=Year, y=StdDev), color="green" ) +

annotate(geom="text", x=MiddleYear, y=sdLabelY,

label="Std Dev across a month", color="red", angle=0) +

annotate(geom="text", x=MiddleYear, y=sdLabelY-5, label=SDFormula,

color="red", angle=0) +

ylim(yScaleLow,yScaleHigh)

#########

# Now predict the extreme temperatures in 2005-2014.

# 1. Define a function that will create a df of two (x, y) pairs that represent a line.

# It needs the 2 x-values of the ends of the line segment, and a previously

# regressed lm.

mkPred <- function(pLM, yearBegin, yearEnd, yOffset=0) {

yearDF <- data.frame(avgYear=c(PredBegin, PredEnd))

retDF <- data.frame(yearDF[,1], unname(predict(pLM, yearDF)))

retDF[1,2] = retDF[1,2] + yOffset

retDF[2,2] = retDF[2,2] + yOffset

colnames(retDF) <- c("avgYear", "Temp")

colnames(retDF)[2] <- "Temp"

return (retDF)

}

# 2. Create the predicted centerline.

dfPredLR <- mkPred(avgLM, PredBegin, PredEnd)

# 3. Create the upper limit of predictions.

dfPredUpper <- mkPred(avgLM, PredBegin, PredEnd, yOffset=dfSDLR[2,2])

# 4. Create the lower limit of predictions.

dfPredLower <- mkPred(avgLM, PredBegin, PredEnd, yOffset=-1\*dfSDLR[2,2])

# 4. Read and scrub recent data (like above).

newData <- read.csv(file=recentFile, header=TRUE, sep=",")

newData <- newData[grep(MonthNum, newData$DATE),]

# Remove the second column (station long name).

newMonth <- newData[-2]

# newMonth[200:255,]

### Extract 4-digit Year number.

newMonth$Year = as.numeric(substr(newMonth$DATE, 1, 4))

# Create new column for highs, (as.num(as.char()) handles negatives.).

newMonth$HIGH = as.numeric(as.character(newMonth$TMAX))

# Create new, temporary vector for highs or lows (as.num(as.char()) handles negatives).

if (ExtremeLabel == "Highs") {

extreme <- newMonth$TMAX

} else {

extreme <- newMonth$TMIN

}

# Convert the string-numbers in quote marks (as.n(as.c()) needed to fix negatives).

newMonth$EXTR = as.numeric(as.character(extreme))

# Create a new DF, one row per year, with mean and StdDev.

newStatsYear <- unique(newMonth$Year)

newStatsExtr <- tapply(newMonth$EXTR, newMonth$Year, mean)

newStatsExtr <- unname(newStatsExtr)

# Remove NA's because it trips up some functions.

newStatsYear <- newStatsYear[which(!is.na(newStatsExtr))]

newStatsExtr <- newStatsExtr[which(!is.na(newStatsExtr))]

newStatsMonDF <- data.frame("Year"=newStatsYear, "Average"=newStatsExtr)

# Limit range of years to be predicted.

newStatsMonDF <- newStatsMonDF[which(newStatsMonDF$Year>=2005 & newStatsMonDF$Year<=2014),]

myTitle <- paste(Location, ", ", MonthText, ", Monthly Average ", ExtremeLabel,

"\nMeans: Predicted Range and Actuals", sep="")

# Automatically choose minimum and maximum values for y-axis, preferring -10:100.

yScaleLow=-10

yScaleHigh=100

if (dfPredUpper[1,2] >90) {

yScaleHigh=dfPredUpper[1,2]+20

yScaleLow=yScaleHigh-110

}

if (dfPredLower[1,2] <5) {

yScaleLow=dfPredLower[1,2]-20

yScaleHigh=yScaleLow+110

}

### 5. Plot predicted data and actual data.

ggplot() +

geom\_line(data = dfPredLR, mapping=aes(x=avgYear, y=Temp), color="gray" ) +

geom\_line(data = dfPredUpper, mapping=aes(x=avgYear, y=Temp) ) +

geom\_line(data = dfPredLower, mapping=aes(x=avgYear, y=Temp) ) +

geom\_point(data= newStatsMonDF, aes(x=Year, y=Average),

color="red4", na.rm=TRUE) +

ylim(dfPredLR[1,2]-55,dfPredLR[1,2]+55) +

annotate(geom="text", x=2010, y=dfPredUpper[1,2]+5,

label="+1 SD", color="red", angle=0) +

annotate(geom="text", x=2010, y=dfPredLower[1,2]-5,

label="-1 SD", color="red", angle=0) +

ggtitle(myTitle)