Evaluating Monthly Trends using CHCN-Daily

Marc Los Huertos

February 10, 2021

Contents

Abstract

This SOP will provide the tools to analyze the CDO climate data for long-term trends in the daily data. We will also create monthly averages and determine if some months have a stronger trend than others.

1 Introduction

1.1 Goals for This Document

This document provides EA students with the methods to analyze climate data based on monthly averages and evaluate if these data are reliable compared to the CHCN-Monthly and investigate sources of uncertainty.

2 Creating a Best Fit Line for Daily TMAX data

2.1 To Begin

You should have R code that generates a plot of daily TMAX data for a site with a best fit line overlaid. If not, please go back to “Using NOAA Clamate Records.”

2.2 Generalized Steps

In this SOP you will...

1. plot the temperature vs. date and overlay a bestfit line
2. create new variables for date and month;
3. create a new dataframe with monthly averages;
4. model and estimate average trend for each month;
5. evaluate the validity of the models; and
6. interpret the trend data

2.3 Regression and Climate Change

One of the ourcomes of the linear regression is to estimate the best fit line

$$y = mx + b + \epsilon, \tag{1}$$

where ϵ is an estimate of the error. In addition, two other estimates are provided, one for the slope, m, and the y-intercept, b.

But these estimates are also hypotheses, where the null hypothesis is:

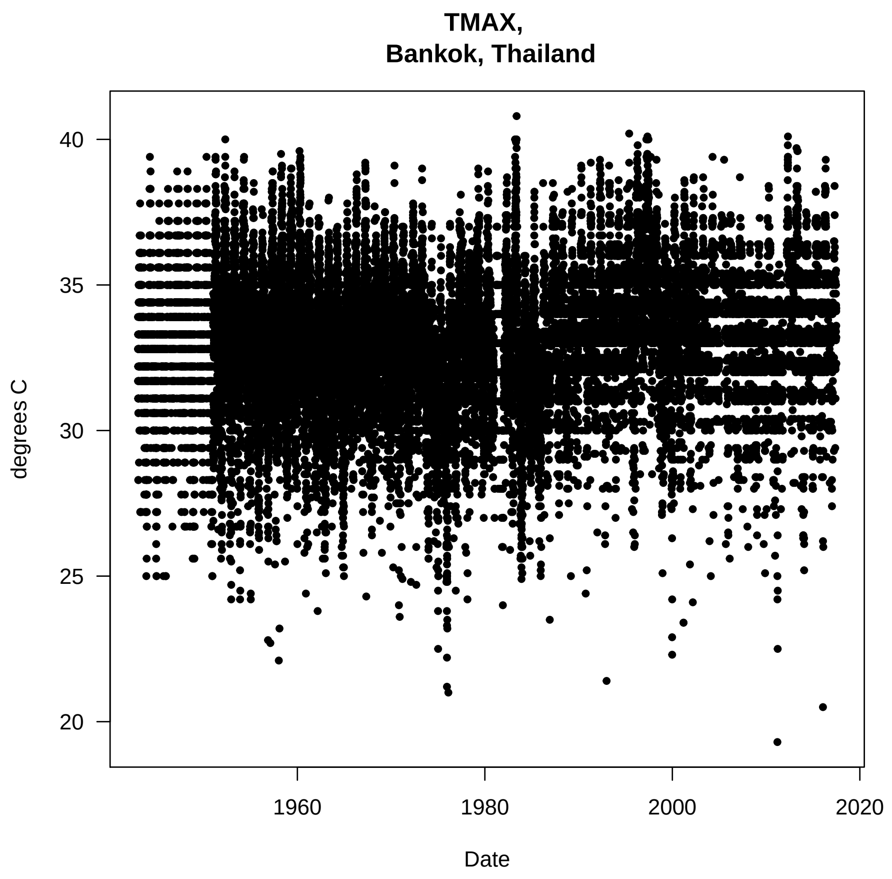
slope is zero Rejecting the null hypothesis would be support the alternative hypothesis, or the estimate of the slope.

y-intercept is zero Rejecting the null hypothesis would support the alternative hypothesis, the estimate of the y-intercept.

Okay, let’s see if we can do this for our Bangkok data. Let’s test if there is a significant change of daily maximum temperatures (TMAX) with time. Thus, in general terms, Maximum temperature is a function of time, or $TMAX = f(Time)$.

$$TMAX \sim \alpha + \beta * time + \epsilon \tag{2}$$

Figure 1: Maximum daily temperatures for Bangkok, Thailand.



Translating this in R will take some additional tricks besides just getting the code figured out. First, we need to identify the predictor variable, 'NewDate', in the data frame which we created in "2 Using NOAA Climate Records".

For the response variable, we will use the daily maximum temeratures, TMAX. Remember there are some missing data, it will be interesting to note how R deals with that.

2.4 Creating a plot and linear model

First, let's create a plot of data using `plot()`, whose format is `plot(x, y)` or `plot(y ~ x)`. We will use the later for now, e.g. plot(TMAX ~ NewDate, data=climate_data) (Figure 1).

We use the `lm()` function that arrange the results in-line with a regression model. This syntax is straight forward,

```
lm(TMAX ~ NewDate, data=climate_data)

##
## Call:
## lm(formula = TMAX ~ NewDate, data = climate_data)
##
## Coefficients:
## (Intercept)      NewDate
##   3.289e+01    2.702e-05
```

Notice that we had define the data source as we created the linearm model. In addition, the format is nearly identical to the `plot()` function. That's convenient!

From this model, we learn that the change in $TMAX$ is 2.7×10^{-5} ° C year^{-1}. Figure 2 shows a trend of increasing maximum temperatures.

We know tha the slope and intercept can used to define a line – so, next we can add a line to the plot. First, let's see how we can create a linear model object using `lm()` and extract the coefficients from linear model, usinig `coef()`.

I am going to name the linear model with an ".lm" suffix to keep track of the type of object created. Then name an object "slopeintercept" to be clear where those two values lie.

```
TMAX.lm = lm(TMAX ~ NewDate, data= climate_data)
slopeintercept = coef(TMAX.lm)
```

Now, we can plot the the data with the best fit line.

```
plot(TMAX ~ NewDate, data= climate_data, las=1)
abline(slopeintercept, col="red", lwd=3)
```

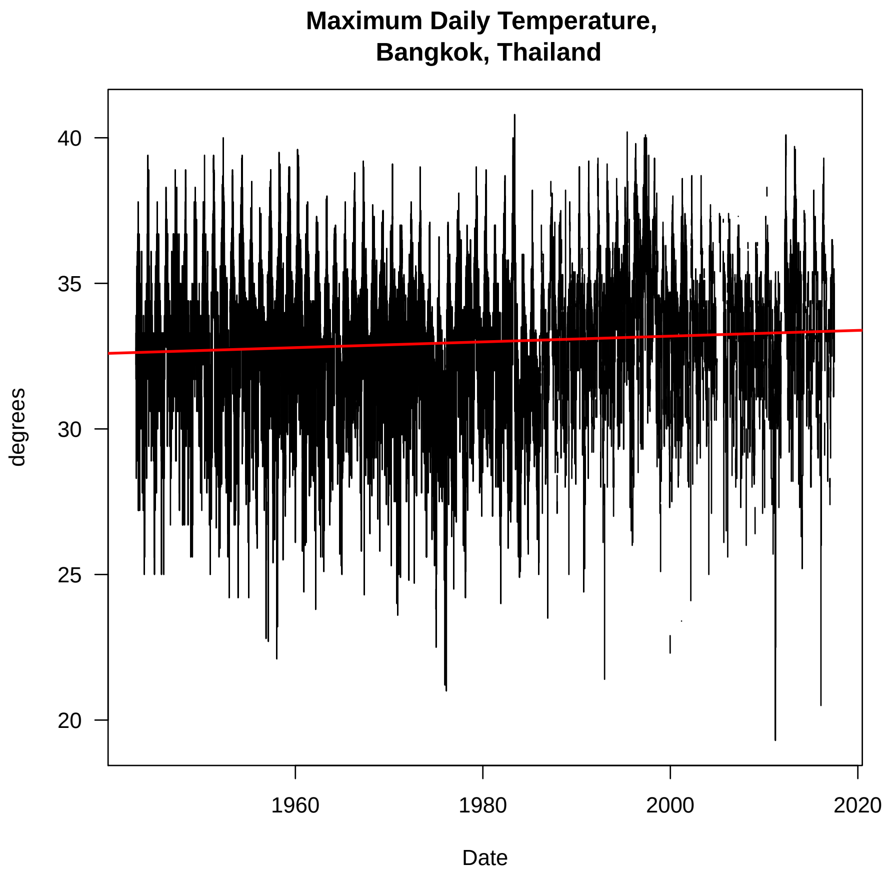
Notice that I added some arguments in the function to improve the graphics. However, I added others in Figure 2, like x- and y-axis labels, that I am not showing to keep coding easier. I encourage you to look up these other arguments by searching online.

2.5 Interpreting the Results

Now determine test the null hypotheses and use the `summary()` function to display many of the important regression results.

```
(TMAX.lm.sum = summary(lm(TMAX ~ NewDate, data=climate_data)))

##
## Call:
## lm(formula = TMAX ~ NewDate, data = climate_data)
##
```

Figure 2: Maximum Daily Temperatures in Bangkok, Thailand.



```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.9974  -1.3193   0.0782   1.4717   7.7771
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.289e+01  1.631e-02 2016.35   <2e-16
## NewDate     2.702e-05  2.140e-06   12.62   <2e-16
##
## (Intercept) ***
## NewDate     ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.329 on 21958 degrees of freedom
##   (5066 observations deleted due to missingness)
## Multiple R-squared:  0.007202,Adjusted R-squared:  0.007157
## F-statistic: 159.3 on 1 and 21958 DF,  p-value: < 2.2e-16
```

Based on the results, we reject the null hypotheses, i.e. the events that this might occur by chance is small: $< 2\text{x}10^{-16}$ for the slope is zero and $p < 2\text{x}10^{-16}$ for the y-intercept is zero.

Because these data are in a time series, they are serially correlated, meaning that the June sample will be more like the July sample than the August sample. In addition, the June 2010 sample will be similar to the June 2009 sample. These correlations violate the assumption of independence, but for now, we will ignore this violation and just create a linear model in bliss.

In addition, we have some information on the residuals and r^2 estimates, which are important to interpret the model – basically how much of the variation is explained by x (date). In our case, we obtain 0.72%, which is pretty low, generally, we shouldn't get excited about a regression unless we approach 30-40%! At some point, we will sort out why the r^2 is so low, but for now, keep moving.

For now, we can appreciate the the temperature is changing, i.e. increasing, with a slope of 2.7015606×10^{-5}°C per year or if we multiply by 100 to make it more "relateable", we obtain a change of 0.0027°C/100 years–still pretty low! It begs the question, how important is climate change in Bangkok?

3 Aggregating Data into Monthly Means

3.1 Monthly Means of Daily Maximum Temperatures

One of the first things to note is how messy the data look and there are lots of sources of variation. For example, we expect months to respond differently to the climate change. To assess this, we will now analyze the data for monthly means of the maximum temperatures.

3.2 Creating Monthly Means

To create monthly means, we need to disagragate the NewDate variable into a month and year variables.

First we can use the `as.Date()` function to extract a portion of the date, where %m is for month and %Y is for a four digit year. Then, we create new variables in our dataframe, one for month and one for year.

```
climate_data$Month = format(as.Date(climate_data$NewDate), format = "%m")
climate_data$Year = format(climate_data$NewDate, format="%Y")
```

Note: Some data sets might not show the century in front of the year, which confuses R when plotting graphs, since it cannot distinguish between different centuries. For example, if under Year in your data set you only see 10,11,12 and your data runs from 1910 to 2010, R will not know how to distinguish the 10 from 1910 and the 10 from 2010, and your plot will jump back and forward a century. Therefore, you need to format your years so that it displays the century. This is how it is done:

1) Open up your climate_data table from your files in R studio.

2) Find the point at which the entries jump from one century to the next. (eg. 1999 to 2000)

3) Write down the row number of the last entry of the earlier century (1999) and the first entry of the new century (2000).

4) Format your code in R using [1:(last entry of 19th century)], [(first entry of 19th century):(last entry)]

```
climate_data$NewDate[1:23344] = format(as.Date(strDates[1:23344], "%m/%d/%y"), "19%y-%m-%d"

## Error in as.Date(strDates[1:23344], "%m/%d/%y"):  object 'strDates'
not found

climate_data$NewDate[23345:45332] = format(as.Date(strDates[23345:45332], "%m/%d/%y"),"20%y-

## Error in as.Date(strDates[23345:45332], "%m/%d/%y"):  object 'strDates'
not found
```

After creating the month and year as separate variables, we can use them to caculate the mean using the `aggregate()` function. In the code below, we can also calculate the standard deviation too, although I haven't used this measure in this document, several students have asked for this for their analysis.

```
MonthlyTMAXMean = aggregate(TMAX ~ Month + Year, climate_data, mean)
```

Note: I always check to see what I have created to make sure I am getting what I expected!

For example, when we look at the structure of the MonthlyTMAXMean, I learn that the 'Year' and 'Month' format – which are defined as character strings.

```
str(MonthlyTMAXMean)

## 'data.frame': 888 obs. of  3 variables:
##  $ Month: chr  "01" "02" "03" "04" ...
##  $ Year : chr  "1943" "1943" "1943" "1943" ...
##  $ TMAX : num  32.2 33.2 34.9 33.5 33.8 ...
```

We need to change this to numeric formats so we can further process the data.

```
MonthlyTMAXMean$Year.num = as.numeric(MonthlyTMAXMean$Year)
MonthlyTMAXMean$Month.num = as.numeric(MonthlyTMAXMean$Month)
```

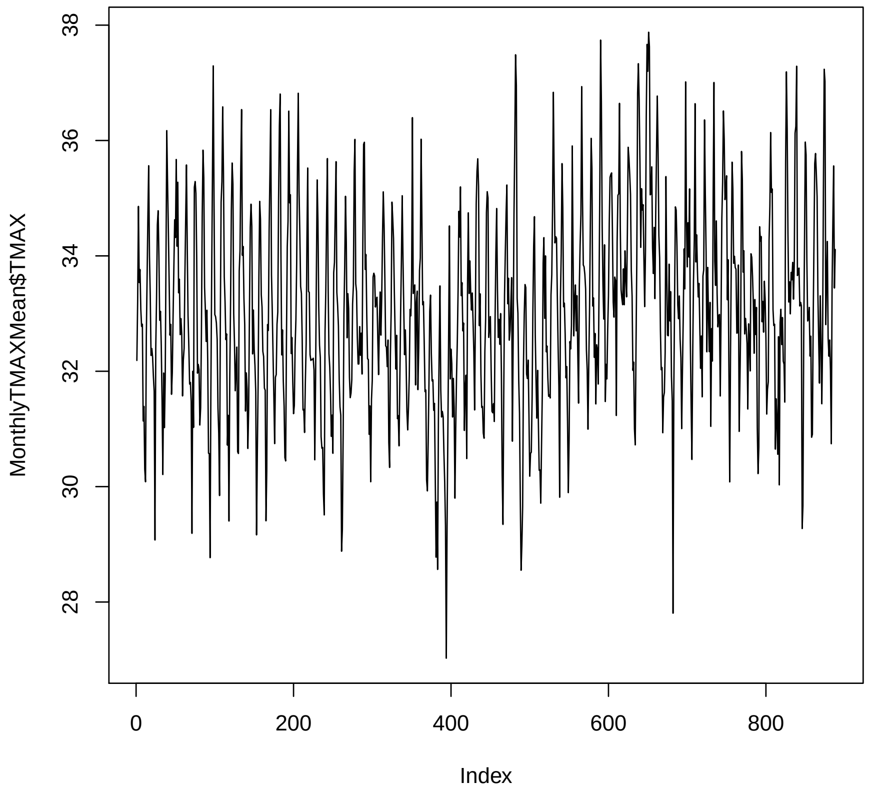
And checking the data...

```
str(MonthlyTMAXMean)

## 'data.frame': 888 obs. of  5 variables:
##  $ Month    : chr  "01" "02" "03" "04" ...
##  $ Year     : chr  "1943" "1943" "1943" "1943" ...
##  $ TMAX     : num  32.2 33.2 34.9 33.5 33.8 ...
##  $ Year.num : num  1943 1943 1943 1943 1943 ...
##  $ Month.num: num  1 2 3 4 5 6 7 8 9 10 ...
```

It worked!

Now, let's see what plot looks like. Should look pretty different than the daily data we had before.

```
plot(MonthlyTMAXMean$TMAX, ty='l')
```



Reminder: When we define a plot type, (ty =), we are defining it as a line which is abreviated as the letter 'l' and abrevation for 'line', not the number '1'. Altenatively, you could make a "point" plot with ty = 'p'.

3.3 Selecting for 1 Month – May

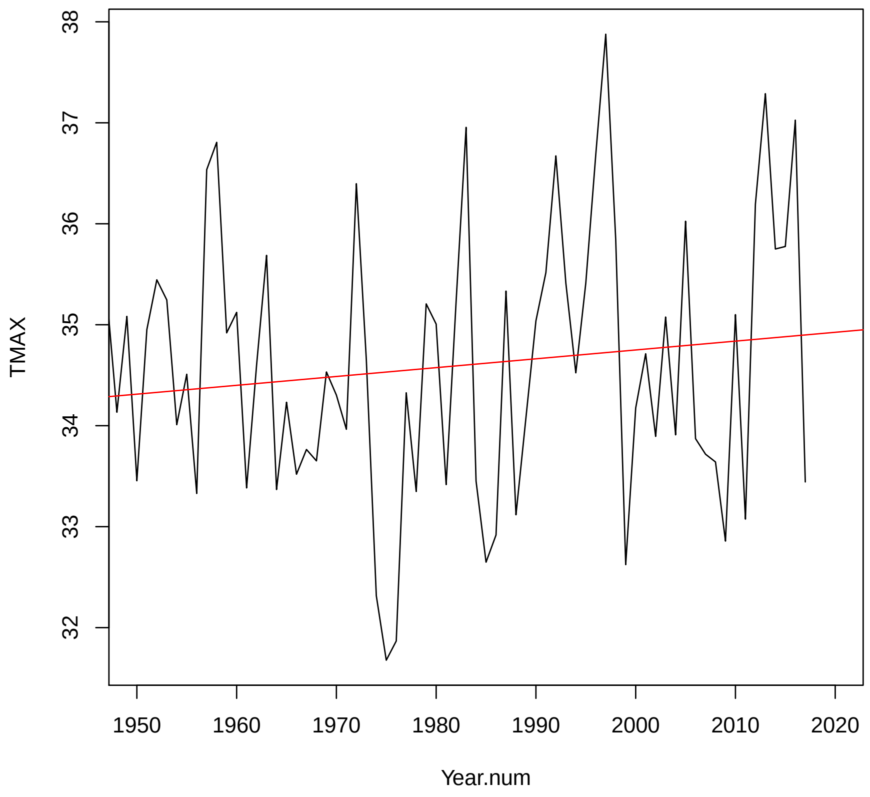
Perhaps, we can get a better handle on this stuff if we analyze for just one month at a time – certainly easier to visualize!

Note: I have limited the data limits to evaluate the a subset of the years from 1950–2020. Also, I can use the numbers because the Year.num is a numeric variable.

In addition, I have selected the 5th month of the year, i.e. May! Note that I am referencing the variable that is a character string, thus need to put "05" in quotes.

```
plot(TMAX~Year.num, data=MonthlyTMAXMean[MonthlyTMAXMean$Month=="05",],
     ty='l', xlim=c(1950, 2020))
```

```
May.lm <- lm(TMAX~Year.num, data=MonthlyTMAXMean[MonthlyTMAXMean$Month=="05",])
summary(May.lm)

##
## Call:
## lm(formula = TMAX ~ Year.num, data = MonthlyTMAXMean[MonthlyTMAXMean$Month ==
##     "05", ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.85376 -0.93210 -0.04633  0.81231  3.15347
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.238621  13.753642   1.253    0.214
## Year.num     0.008756   0.006946   1.261    0.211
##
## Residual standard error: 1.302 on 73 degrees of freedom
## Multiple R-squared:  0.0213,Adjusted R-squared:  0.007897
## F-statistic: 1.589 on 1 and 73 DF,  p-value: 0.2115

abline(coef(May.lm), col="red")
```



Now, the change is 0.0088 degress C/year or 0.876 degress C/100 years with a probability of 0.2115. Although we can't reject the null hypothesis, we find the method to be fairly straightforward!

Note: if you are getting an error saying: error in plot.window(need finite 'ylim' values), you may need to check your data format and see whether the months in your data set have a 0 infront of them or not. If not, try "5" instead of "05".

3.4 Testing all the Months

This section for advanced users and specific questions. Please do not proceed without consulting Marc.

I think you should evaluate every month and see what happens. You might also consider looking at the TMIN as well. Could be important![1]

Below, I have create code to evaluate all of the months at once, but you may

[1]What about multiple hypotheses in one dataset!

prefer to go through each month manually and change the number from 5 to other months of the year – but if you do you the code below, try to figure out what each line is doing.

3.5 Next Steps

3.5.1 Analyzing Minimum Daily Temperatures

Alternatively, it might be important to evaluate changes to the daily mininum temperatures. Following the same steps we used before but using the TMIN instead of TMAX, let's analyze the monthly average of daily minimum temperatures by following these steps:

1. First, let's plot the daily minimum temperatures, and as with the daily maximum temperatures, find tons of scatter (Table 1).

   There appears to be a trend, but it's clouded with lots of variation.

2. We create a monthly TMIN mean for each month.

```
MonthlyTMINMean = aggregate(TMIN ~ Month + Year, climate_data, mean)

# Fixing the Format of Month and Year as numeric
MonthlyTMINMean$Year.num = as.numeric(MonthlyTMINMean$Year)
MonthlyTMINMean$Month.num = as.numeric(MonthlyTMINMean$Month)
head(MonthlyTMINMean)

##   Month Year     TMIN Year.num Month.num
## 1    01 1943 18.54828     1943         1
## 2    02 1943 20.73077     1943         2
## 3    03 1943 23.39655     1943         3
## 4    04 1943 23.79259     1943         4
## 5    05 1943 24.87692     1943         5
## 6    06 1943 24.76429     1943         6
```

3. Create a plot of the monthly average of the daily minimum temperatures.

```
plot(MonthlyTMINMean$TMIN, ty='l')
```

```r
# First I create a vector of months
Months = c("January", "February", "March", "April",
    "May", "June", "July", "August", "September", "October",
    "November", "December")

# Create a panel so I can see all the figures at
# once.
par(mfrow = c(4, 3), mar = c(5, 4, 3, 2) + 0.1)
TMAXresult <- NA
for (i in 1:12) {
    plot(TMAX ~ Year.num, data = MonthlyTMAXMean[MonthlyTMAXMean$Month.num ==
        i, ], ty = "l", las = 1, xlim = c(1940, 2020),
        main = Months[i])
    Month.lm <- lm(TMAX ~ Year.num, data = MonthlyTMAXMean[MonthlyTMAXMean$Month.num ==
        i, ])
    summary(Month.lm)
    abline(coef(Month.lm), col = "red")

    TMAXresult <- rbind(TMAXresult, cbind(Months[i],
        round(coef(Month.lm)[2], 4), round(summary(Month.lm)$coefficients[2,
            4], 4), round(summary(Month.lm)$r.squared,
            3)))
}
```

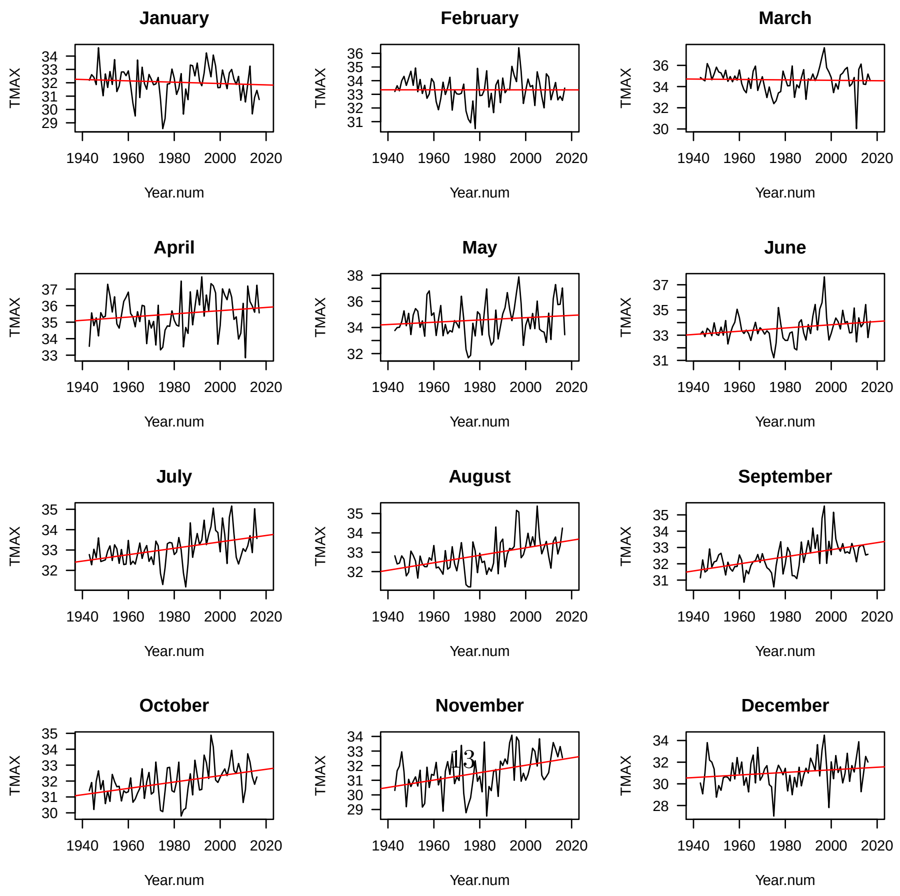
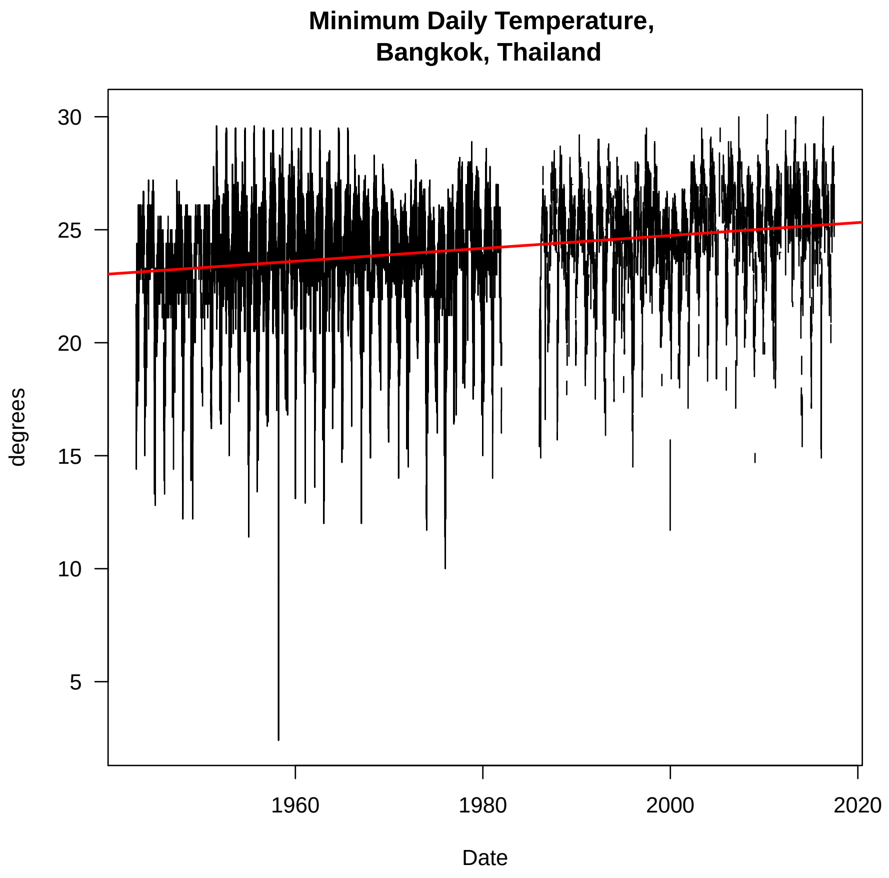
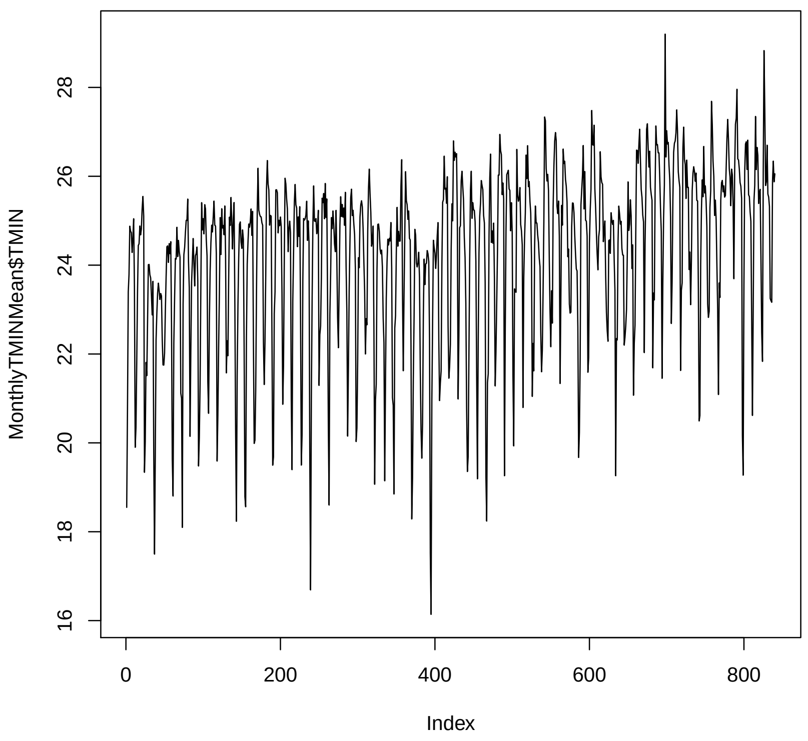


Figure 3: Minimum Daily Temperatures in Bangkok, Thailand.

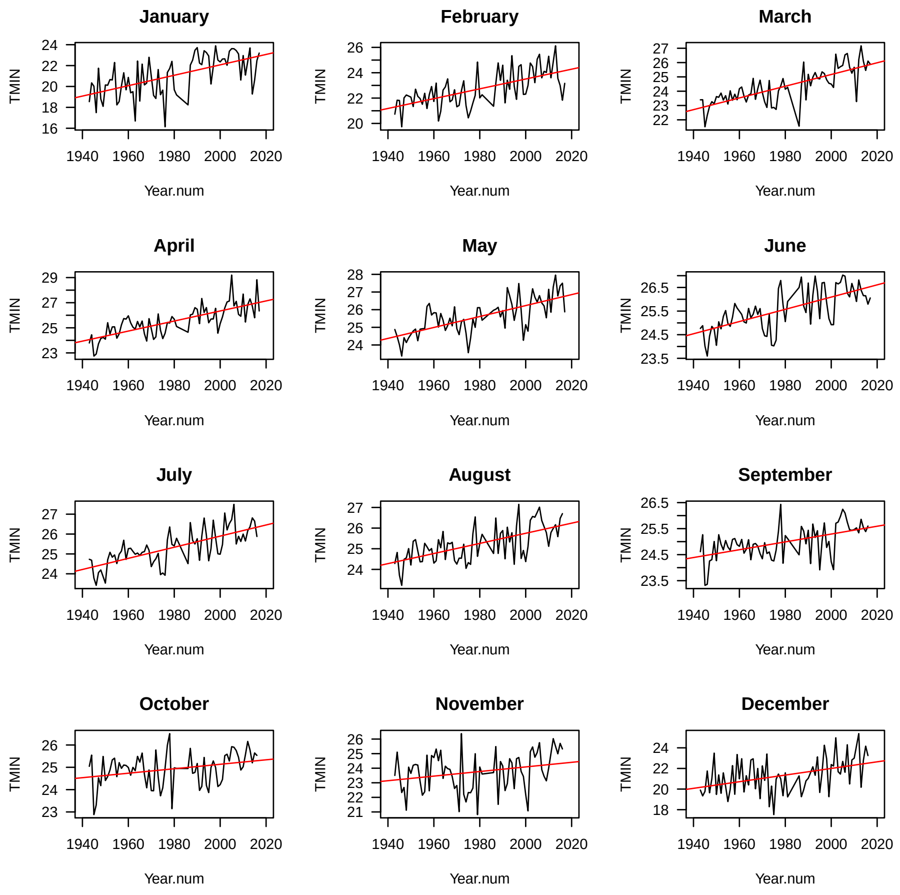




There is still lots of scatter and now we can subset our data by month.

4. Using the example above, we'll plot all 12 months at once to look for patterns (Table 4).

Figure 4: Twelve Months of Monthly Average Daily Minimum Temperatures, Bangkok, Thailand



5. The change in minimum temperatures seems to be even more compelling than the maximum temperatures. To compare, look at the Table **??** to appreciate estimated slopes and their associated null hypothesis probabilities.

```r
# Creating a dataframe to be used in a table.
Results <- data.frame(Month = TMINresult[c(2:13),1],
    TMINSlope = TMINresult[c(2:13),2],
    TMIN_P = as.numeric(TMINresult[c(2:13),3]),
    TMINRsq = TMINresult[c(2:13),4],
    TMAXSlope = TMAXresult[c(2:13),2],
    TMAX_P = as.numeric(TMAXresult[c(2:13),3]),
    TMAXRsq = TMAXresult[c(2:13),4])
Results$starTMAX <- ifelse(
  Results$TMAX_P <= .001, "***",
    ifelse(Results$TMAX_P <= .01, "**",
      ifelse(Results$TMAX_P <= .05, "*", "NS")))
Results$starTMIN <- ifelse(
  Results$TMIN_P <= .001, "***",
    ifelse(Results$TMIN_P <= .01, "**",
      ifelse(Results$TMIN_P <= .05, "*", "NS")))
Results$TMINslope=paste(Results$TMINSlope, Results$starTMIN)
Results$TMAXslope=paste(Results$TMAXSlope, Results$starTMAX)
colnames(Results) <- c("Month", "2", "3", "R^2", "5", "6",
                       "R^2", "8", "9", "Slope TMIN", "Slope TMAX")

library(xtable)
print(xtable(Results[,c(1, 10, 4, 11, 7)]))
```

	Month	Slope TMIN	R^2	Slope TMAX	R^2.1
1	January	0.0498 ***	0.341	-0.0051 NS	0.009
2	February	0.0387 ***	0.385	-1e-04 NS	0
3	March	0.0409 ***	0.567	-0.002 NS	0.001
4	April	0.04 ***	0.551	0.0097 NS	0.035
5	May	0.031 ***	0.48	0.0088 NS	0.021
6	June	0.0259 ***	0.448	0.0129 *	0.078
7	July	0.028 ***	0.493	0.0157 ***	0.173
8	August	0.0245 ***	0.395	0.0194 ***	0.245
9	September	0.015 ***	0.279	0.0217 ***	0.257
10	October	0.01 *	0.093	0.02 ***	0.174
11	November	0.0158 *	0.067	0.0253 ***	0.169
12	December	0.0322 ***	0.178	0.0119 NS	0.034

Based on the results above, the slopes are greatest during the dry season (starting in May) for the maximum temperatures – but the minimum temperatures show the largest slopes (change) and peaking between January and April.

In addition, the r^2 values signify the amount of the variance explained by the predictor – in the case of TMIN, most of the values are over 20% meaning that over 20% of the variance is explained by time. While in March and April over time explains 50% of the variance.

This is very high for uncontrolled experiments. However, we should be cognizant that in many cases, especially for the maximum temperatures, it is less than 10%. This means the the variation in temperature are not predicted by time – thus, as a modeler, I would work very hard to capture other sources to better understand what is going on in Thailand.

Finally, we should also be very concerned about testing 2 dozen hypotheses with our little R code. It's easy to do, but based on change alone, with a critical value of 0.05, we should expect 1 in 20 tests to give us a Type I error, a signal when one doesn't exists. Since we did 12 tests, we should expect a good chance that one or more of our tests will reject the null hypothesis incorrectly. Yikes! Please keep this in mind and be careful to avoid this potential problem.

As we might expect, the a small amount of the variance is explained by the "Month." Many things predict temerpature, that year is one, is quite problematic.

6. What we have not determined is the cause. So, be careful when you describe the results, cause and effect cannot be analyzed using this method.

3.5.2 Precipitation: Departure from Mean

Precipiation might depend more on the departure from the mean (often referred as as normal, whatever that means!). I think it's worth pursuing, but have found some datasets have problems.

Precipitation is something that might increase or decrease due to climate change. So, to analyze this, we will evaluate how much precipitation has deveated from the mean, by plotting the rainfall and the mean in a time-series plot.

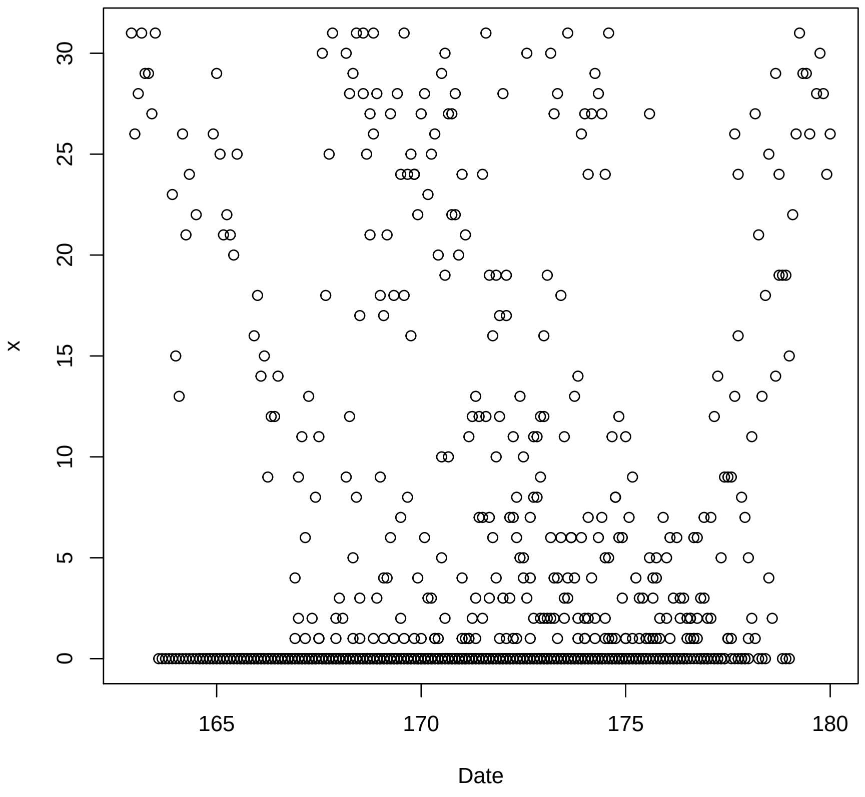
Second, we need to remove the missing values and evaluate which years that are complete – why do you think missing variables might be an issue analyzing precipiation?

If you are missing rainy months, then the whole year should be thrown out – but what about partial years in the drought season? We'll need to be consistent – assuming that missing data are not zeros, we'll define complete years as over 300 days of data.

NOTE: The missing values have not been converted to NAs!

```r
climate_data$PRCP[climate_data$PRCP==-9999] <- NA
```

```r
Missing <- aggregate(is.na(climate_data$PRCP),
          list(climate_data$Month, climate_data$Year), sum)

# The aggregate command is used to create a simplified dataset. In this case
# we are creating a sum of PRCP based on each month and year.

Missing$Date = as.numeric(Missing$Group.1) + as.numeric(Missing$Group.2)/12

plot(x ~ Date, data=Missing)
```



Third, we will need to decide what level of aggredation – monthly, yearly, etc. Let's aggreate by month and year to get monthly totals.

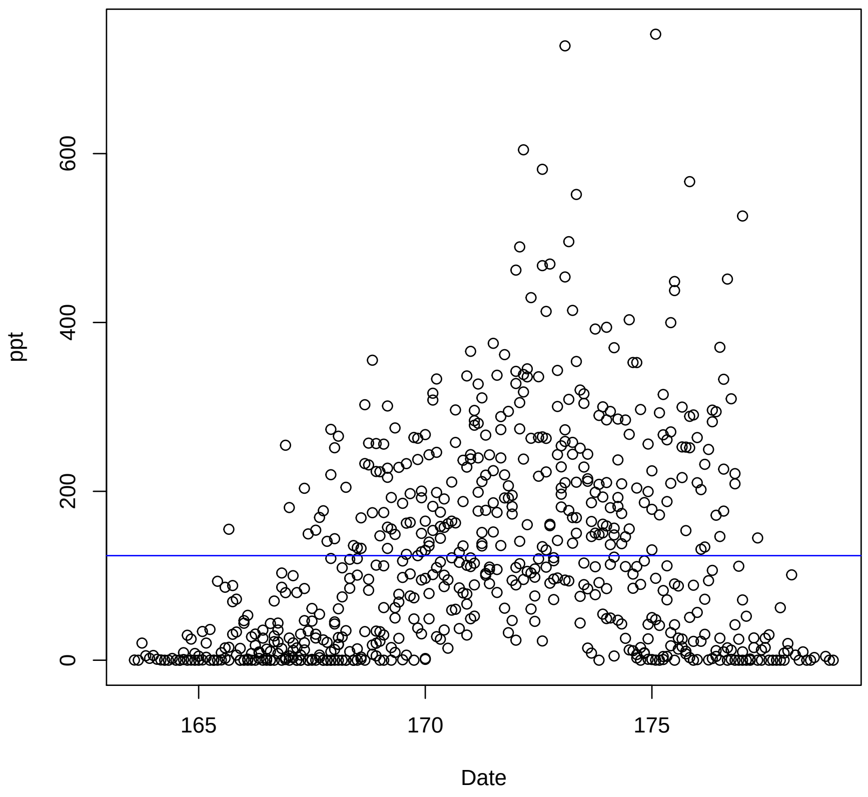
There are loads of missing values in many months. Let's cut of the months that have more than 4 missing days.

```
TotalPPT <- aggregate(climate_data$PRCP,
            list(climate_data$Month, climate_data$Year), sum, na.rm=T)

# Check to see what you created.

names(TotalPPT) = c("Group.1", "Group.2", "ppt")

NonMissing <- Missing[Missing$x < 5, c(1:3)]
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

PPT <- merge(TotalPPT, NonMissing, all.y=TRUE)
PPT$Date <- as.numeric(PPT$Group.1) + as.numeric(PPT$Group.2)/12
head(PPT)

##   Group.1 Group.2  ppt x     Date
## 1      01    1951  0.2 0 163.5833
## 2      01    1952  0.0 0 163.6667
## 3      01    1953 20.3 0 163.7500
## 4      01    1954  5.3 0 163.8333
## 5      01    1955  2.2 0 163.9167
## 6      01    1956  5.6 0 164.0000
```

First, we need a "mean" – The IPCC uses 1961-1990 as a norm for temperature, I don't know what is the standard for rainfall or Thailand, so we should look that up. For now, we'll use our filtered records to generate a mean.

```
PRCP_mean = mean(PPT$ppt)
```

```
plot(ppt~Date, data=PPT)
abline(h=PRCP_mean, col="blue")
```

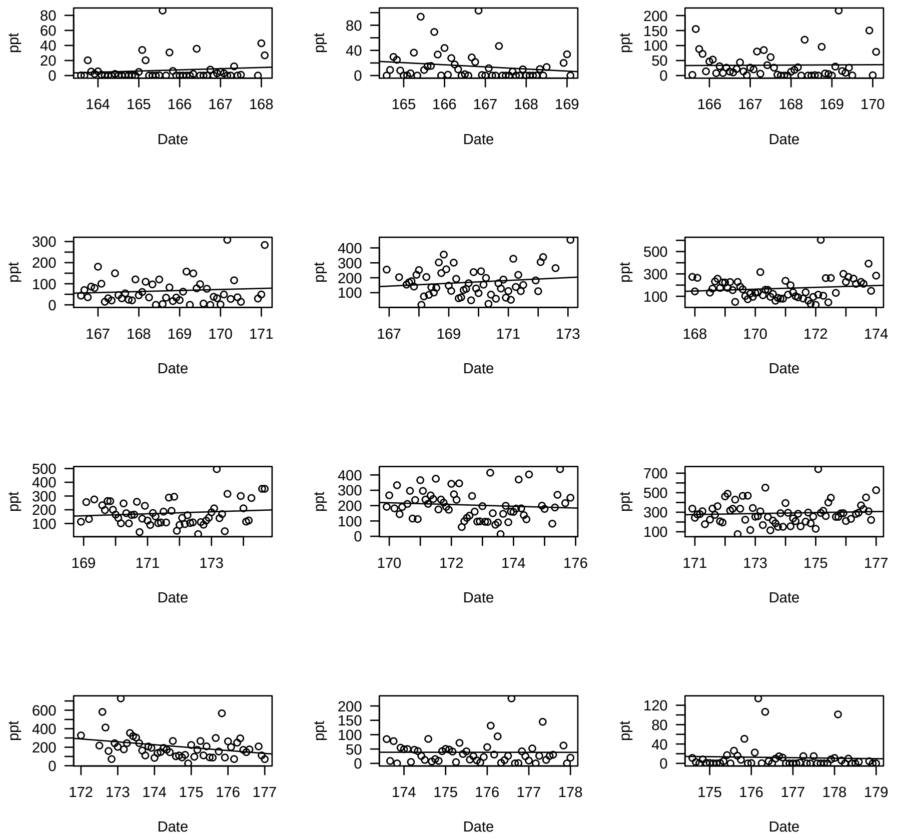


Wow, these data look terrible – the mean looks meaningless given the biased data set. I don't think we can do more analysis with this. But let's look at a few months and see what we can decipher.

```
#LosAngeles£PRCP[LosAngeles£PRCP==-9999] <- NA
#YearlySum = aggregate(PRCP ~ Year, LosAngeles, sum)
#YearlySum£YEAR = as.numeric(YearlySum£Year)
#YearlyMean = mean(YearlySum£PRCP)
```

A yearly mean, based on the annual sum for the entire records. Not sure this is appropriate.

Figure has points of the yearly sum of rainfall and the blue line mean. The greenline is the trend and red line is a five year running average, I think! I am still trying to understand what the code is doing.


ppt
Date

```
#plot(PRCP~YEAR, data=YearlySum, las=1, ty="p")
#abline(h=YearlyMean, col="blue")
#YearlySum.lm = lm(PRCP~YEAR, data=YearlySum)
#abline(coef(YearlySum.lm), col="green")

#n <- 5
#k <- rep(1/n, n)
#k

#y_lag <- stats::filter(YearlySum£PRCP, k, sides=1)
#lines(YearlySum£YEAR, y_lag, col="red")
```

```
#summary(YearlySum.lm)
```

3.6 Assumptions of the Linear Regression

Regression models, like all statistics, rely on certain assumptions. Violations of these assumptions reduces the validity of the model. If the violations are serious, then the model could be misleading or even incorrect.

TBD

3.6.1 Assumptions about ϵ

The error term should have

E(et) = 0 , zero mean

E(et) = s , constant variance

E(et, Xt) = 0 , no correlation with X

TBD E(e , e), no autocorrelation. t t-1

e **Normally distributed** (for hypothesis testing).

Assumption four is especially important and most likely not to be met when using time series data.

Autocorrelation.

1. It is not uncommon for errors to track themselves; that is, for the error a time t to depend in part on its value at t - m, where m is a prior time period.

Figure 5: Default diagnostic plots for a linear model in R.

```
par(mfrow=c(2,2))
plot(lm(TMIN ~ YEAR, data=MonthlyTMINMean[MonthlyTMINMean$MONTH==1,]))

## Error in eval(predvars, data, env):  object 'YEAR' not found
```

3.6.2 Model Diagnostics

With every statistical test done, researchers validate their model in some way or anther. Often this entails the use of diagnostics, a standardize battery of procedures to check to see if the data are following the assumptions.

In R four plots are created by default. To see them all at the same time, we need to change the graphical parameters, using the par() function. In this case, we use `par(mfrow=c(2,2))` to create alter the graphics window expects four panels, in this case a 2 rows and two columns.

Try not to get bogged down in the code at this point. But noting this function can be handy in a number of ways to improve one's graphics.

To determine the validity of linear model assumptions (e.g. normality or heterogeneity of variance), you have probably used statistical tests; in contrast statisticians almost exclusively look at diagnostic plots. Why? When assumptions are violated the tests to determine violations do not perform well. So, let's see how to look at these assumptions graphically with these diagnostic plots. Linear models should have diagnostic plots that do not have any obvious structure or pattern. In this case, Figure 3.6.2 should show a great deal remaining structure in the residuals. Although for today, we are not going to try to interpret these figures, but you should notice there is a ton of unaccounted structure, i.e. variance, in the model. This is due, in part, to a violation of independence; these data are serially correlated and the model does not account for that and is inappropriate because of this. It also appears that a straight-line model does not fit well and a curvilinear should be investigated.

A properly specified model is shown in