03-06 Linear Regressions

1 - Purpose

- Convert vectors from string to numeric
- Create scatterplots between variables
- Simple linear regression
- Add regression lines to plots

2 - Concepts

3 - Reading in the weather data

There are two files you need to download for this lesson:
1) The R script
2) Weather data from Lansing in 2016: ***LansingNOAA2016.csv*** (note: this data file contains an expanded version of weather data that we used in previous lessons)

Ensure that the data file, ***LansingNOAA2016.csv,*** is saved to your ***data*** directory in the ***R Root*** folder.

In this lesson we are going to look at a year's worth of weather data and try to find relationships among weather conditions (e.g, humidity and precipitation) using linear modelling.

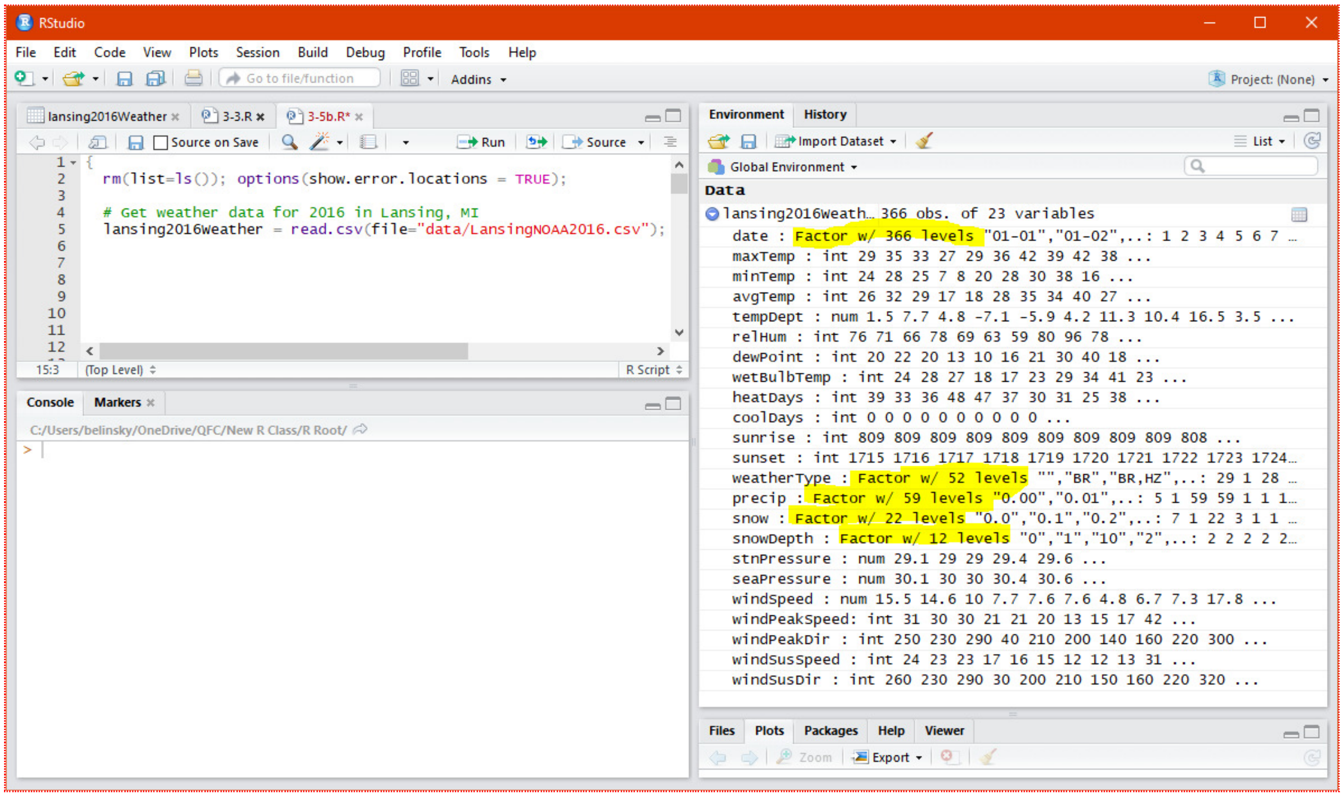
The first thing we do in the script is import the data from the CSV file and save it to the variable ***lansing2016Weather,*** a data frame

```
lansing2016Weather = read.csv(file="data/LansingNOAA2016.csv");
```

3.1 - Factors in a data frame

If you look at the column headers in ***lansing2016Weather*** (*Fig 1*), you will see that five of them (***date, weatherType, precip, snow, snowDepth***) are of type ***Factor***. By default, R will designate columns in a data frame as factor if there are any non-numeric characters in the column. *Extension: Factors in R*

Of these five column headers that are factors:
- ***precip***, ***snow,*** and ***snowDepth*** have numeric values but use **T** to represent trace amounts of precipitation
- ***date*** uses dashes (-) to separate the numeric month and day (02-20 is February 20th)
- ***weatherType*** contains 2-character strings that represent various weather conditions for the day (e.g., **HZ** is hazy, **FG** is foggy)



```r
{
  rm(list=ls());  options(show.error.locations = TRUE);

  # Get weather data for 2016 in Lansing, MI
  lansing2016Weather = read.csv(file="data/LansingNOAA2016.csv");
```

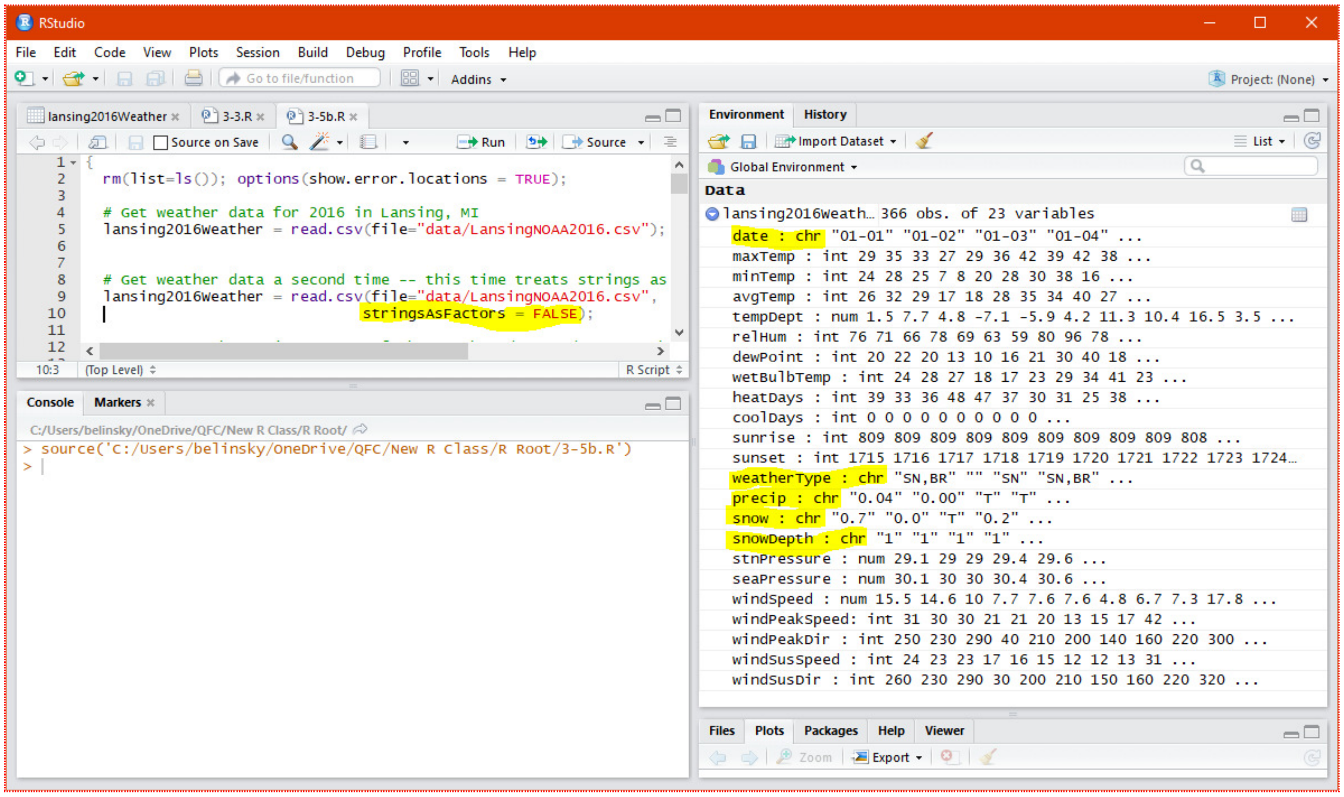
```
Data
lansing2016Weath… 366 obs. of 23 variables
  date : Factor w/ 366 levels "01-01","01-02",..: 1 2 3 4 5 6 7 …
  maxTemp : int 29 35 33 27 29 36 42 39 42 38 ...
  minTemp : int 24 28 25 7 8 20 28 30 38 16 ...
  avgTemp : int 26 32 29 17 18 28 35 34 40 27 ...
  tempDept : num 1.5 7.7 4.8 -7.1 -5.9 4.2 11.3 10.4 16.5 3.5 ...
  relHum : int 76 71 66 78 69 63 59 80 96 78 ...
  dewPoint : int 20 22 20 13 10 16 21 30 40 18 ...
  wetBulbTemp : int 24 28 27 18 17 23 29 34 41 23 ...
  heatDays : int 39 33 36 48 47 37 30 31 25 38 ...
  coolDays : int 0 0 0 0 0 0 0 0 0 0 ...
  sunrise : int 809 809 809 809 809 809 809 809 809 808 ...
  sunset : int 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724…
  weatherType : Factor w/ 52 levels "","BR","BR,HZ",..: 29 1 28 …
  precip : Factor w/ 59 levels "0.00","0.01",..: 5 1 59 59 1 1 1…
  snow : Factor w/ 22 levels "0.0","0.1","0.2",..: 7 1 22 3 1 1 …
  snowDepth : Factor w/ 12 levels "0","1","10","2",..: 2 2 2 2 2…
  stnPressure : num 29.1 29 29 29.4 29.6 ...
  seaPressure : num 30.1 30 30 30.4 30.6 ...
  windSpeed : num 15.5 14.6 10 7.7 7.6 7.6 4.8 6.7 7.3 17.8 ...
  windPeakSpeed: int 31 30 30 21 21 20 13 15 17 42 ...
  windPeakDir : int 250 230 290 40 210 200 140 160 220 300 ...
  windSusSpeed : int 24 23 23 17 16 15 12 12 13 31 ...
  windSusDir : int 260 230 290 30 200 210 150 160 220 320 ...
```

Fig 1: In R, data frame column with strings in them default to factors.

The problems is that we would not use four of the five "factor" columns as factors in an analysis and the fifth, ***weatherType***, needs to be better formatted. In this case, it would be best to have R treat these columns as strings. We can add the parameter ***stringsAsFactors*** to ***read.csv()*** to change this behavior:

```r
lansing2016Weather = read.csv(file="data/LansingNOAA2016.csv",
                                   stringsAsFactors = FALSE);
```

Changing the columns to string also makes the variables in the Environment Window easier to read (*Fig 2*).



```
{
  rm(list=ls()); options(show.error.locations = TRUE);

  # Get weather data for 2016 in Lansing, MI
  lansing2016Weather = read.csv(file="data/LansingNOAA2016.csv");


  # Get weather data a second time -- this time treats strings as
  lansing2016Weather = read.csv(file="data/LansingNOAA2016.csv",
                                stringsAsFactors = FALSE);
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-5b.R')
>
```

```
Data
lansing2016Weath… 366 obs. of 23 variables
  date : chr "01-01" "01-02" "01-03" "01-04" ...
  maxTemp : int 29 35 33 27 29 36 42 39 42 38 ...
  minTemp : int 24 28 25 7 8 20 28 30 38 16 ...
  avgTemp : int 26 32 29 17 18 28 35 34 40 27 ...
  tempDept : num 1.5 7.7 4.8 -7.1 -5.9 4.2 11.3 10.4 16.5 3.5 ...
  relHum : int 76 71 66 78 69 63 59 80 96 78 ...
  dewPoint : int 20 22 20 13 10 16 21 30 40 18 ...
  wetBulbTemp : int 24 28 27 18 17 23 29 34 41 23 ...
  heatDays : int 39 33 36 48 47 37 30 31 25 38 ...
  coolDays : int 0 0 0 0 0 0 0 0 0 0 ...
  sunrise : int 809 809 809 809 809 809 809 809 809 808 ...
  sunset : int 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724…
  weatherType : chr "SN,BR" "" "SN" "SN,BR" ...
  precip : chr "0.04" "0.00" "T" "T" ...
  snow : chr "0.7" "0.0" "T" "0.2" ...
  snowDepth : chr "1" "1" "1" "1" ...
  stnPressure : num 29.1 29 29 29.4 29.6 ...
  seaPressure : num 30.1 30 30 30.4 30.6 ...
  windSpeed : num 15.5 14.6 10 7.7 7.6 7.6 4.8 6.7 7.3 17.8 ...
  windPeakSpeed: int 31 30 30 21 21 20 13 15 17 42 ...
  windPeakDir : int 250 230 290 40 210 200 140 160 220 300 ...
  windSusSpeed : int 24 23 23 17 16 15 12 12 13 31 ...
  windSusDir : int 260 230 290 30 200 210 150 160 220 320 ...
```

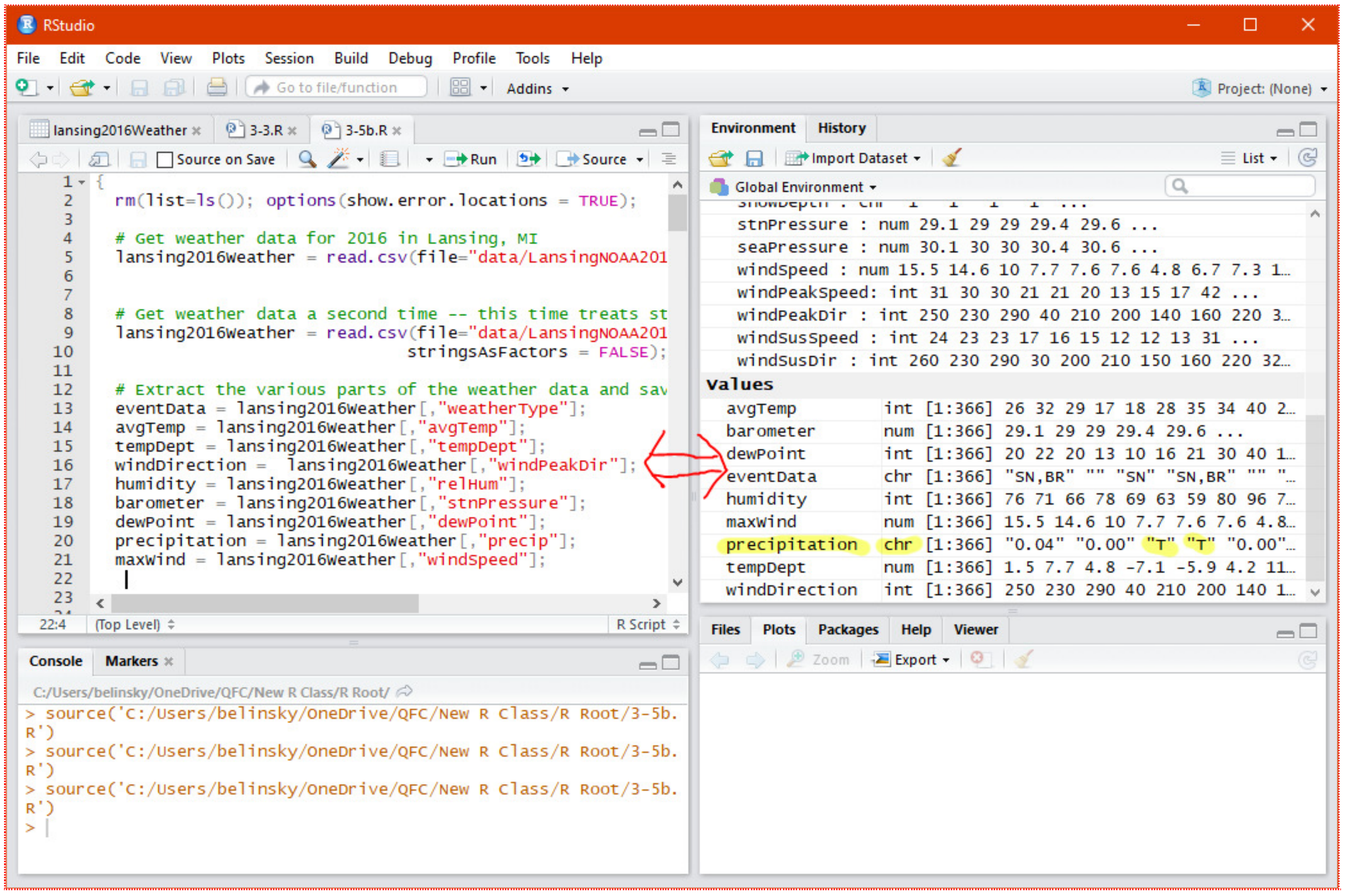
Fig 2: Strings (or characters) in a data frame.

Note: most functions in R will intelligently convert strings or numbers to factors when needed. This happened in lesson 3-4 (t-tests and ANOVAs) when the ***boxplot()*** and ***aov()*** functions converted the years (second column of the data frame) into factors.

3.2 - Getting data from the data frame

Next we are going to pull out 12 columns (representing 12 different weather measurements) from the ***lansing2016Weather*** data frame and save them to vectors.

```
date = lansing2016Weather[,"date"];
eventData = lansing2016Weather[,"weatherType"];
avgTemp = lansing2016Weather[,"avgTemp"];
tempDept = lansing2016Weather[,"tempDept"];
precipitation = lansing2016Weather[,"precip"];
humidity = lansing2016Weather[,"relHum"];
barometer = lansing2016Weather[,"stnPressure"];
dewPoint = lansing2016Weather[,"dewPoint"];
avgWind = lansing2016Weather[,"windSpeed"];
maxWind = lansing2016Weather[,"windPeakSpeed"];
windDirection = lansing2016Weather[,"windPeakDir"];
sunrise = lansing2016Weather[,"sunrise"];
sunset = lansing2016Weather[,"sunset"];
```



```
{
  rm(list=ls()); options(show.error.locations = TRUE);

  # Get weather data for 2016 in Lansing, MI
  lansing2016Weather = read.csv(file="data/LansingNOAA201


  # Get weather data a second time -- this time treats st
  lansing2016Weather = read.csv(file="data/LansingNOAA201
                                stringsAsFactors = FALSE);

  # Extract the various parts of the weather data and sav
  eventData = lansing2016Weather[,"weatherType"];
  avgTemp = lansing2016Weather[,"avgTemp"];
  tempDept = lansing2016Weather[,"tempDept"];
  windDirection =  lansing2016Weather[,"windPeakDir"];
  humidity = lansing2016Weather[,"relHum"];
  barometer = lansing2016Weather[,"stnPressure"];
  dewPoint = lansing2016Weather[,"dewPoint"];
  precipitation = lansing2016Weather[,"precip"];
  maxWind = lansing2016Weather[,"windSpeed"];
```

```
  stnPressure : num 29.1 29 29 29.4 29.6 ...
  seaPressure : num 30.1 30 30 30.4 30.6 ...
  windSpeed : num 15.5 14.6 10 7.7 7.6 7.6 4.8 6.7 7.3 1...
  windPeakSpeed: int 31 30 30 21 21 20 13 15 17 42 ...
  windPeakDir : int 250 230 290 40 210 200 140 160 220 3...
  windSusSpeed : int 24 23 23 17 16 15 12 12 13 31 ...
  windSusDir : int 260 230 290 30 200 210 150 160 220 32...
Values
  avgTemp         int [1:366] 26 32 29 17 18 28 35 34 40 2...
  barometer       num [1:366] 29.1 29 29 29.4 29.6 ...
  dewPoint        int [1:366] 20 22 20 13 10 16 21 30 40 1...
  eventData       chr [1:366] "SN,BR" "" "SN" "SN,BR" "" "...
  humidity        int [1:366] 76 71 66 78 69 63 59 80 96 7...
  maxWind         num [1:366] 15.5 14.6 10 7.7 7.6 7.6 4.8...
  precipitation   chr [1:366] "0.04" "0.00" "T" "T" "0.00"...
  tempDept        num [1:366] 1.5 7.7 4.8 -7.1 -5.9 4.2 11...
  windDirection   int [1:366] 250 230 290 40 210 200 140 1...
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-5b.
R')
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-5b.
R')
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-5b.
R')
> 
```

Fig 3: Saving columns from the data frame as vectors.

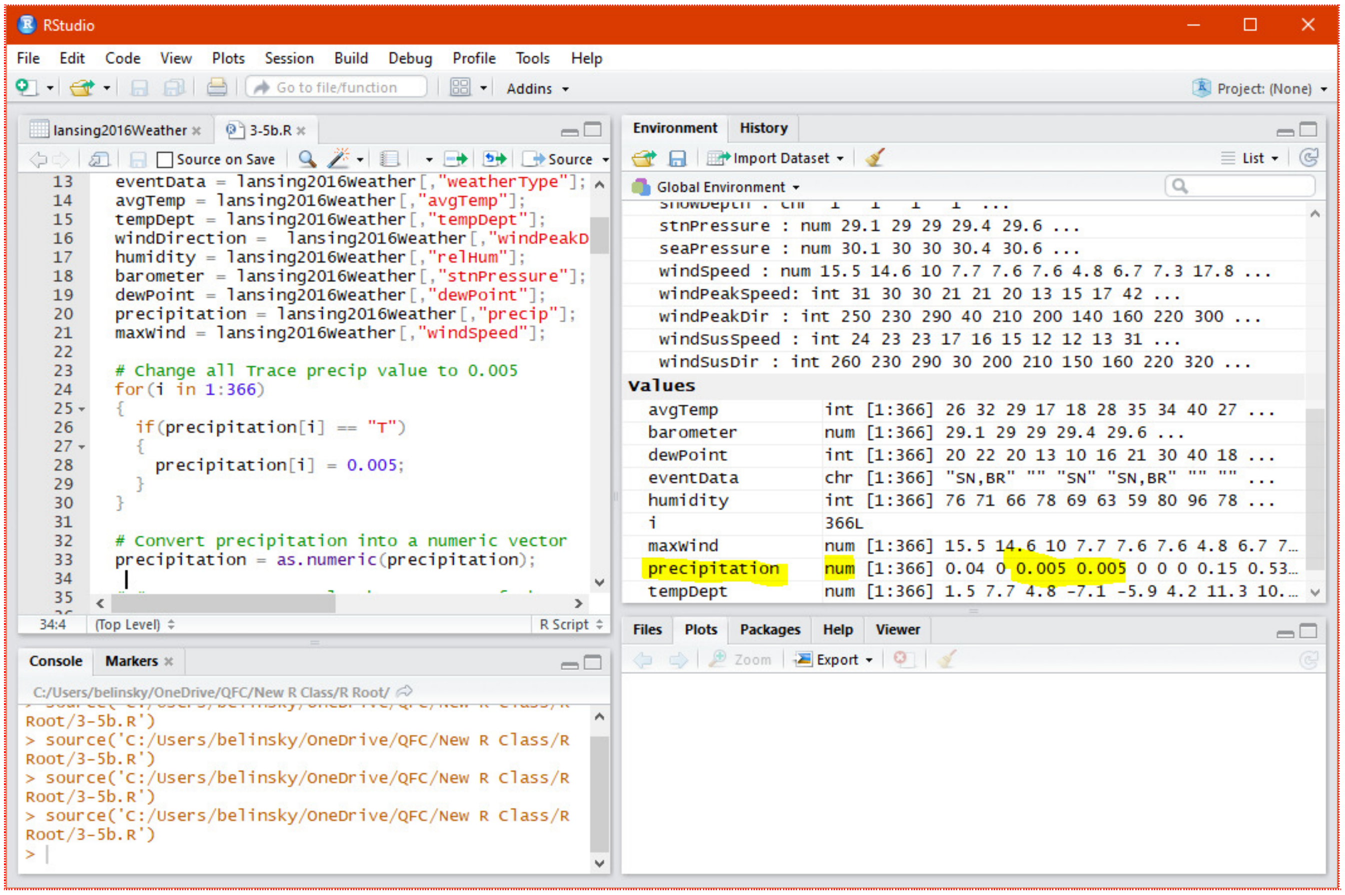
4 - Converting from characters to numbers

Precipitation is given as a character string instead of numeric (*Fig 3*). This is because it has "T" values, which means "trace" or that there were less than 0.01 inches of precipitation-- but not 0. It would be best to convert every value in the vector to a number so we can treat the whole vector as a number. In this case, we will convert "T" to 0.005 (halfway between 0 and 0.01).

```
for(i in 1:366)
{
  if(precipitation[i] == "T")
  {
    precipitation[i] = 0.005;
  }
}
```

We go through all 366 values in the vector and, if the value is "T", we change it to 0.005. However, the vector will remain a string vector until we change its type. We do this using ***as.numeric()***.

```
precipitation = as.numeric(precipitation);
```



```
eventData = lansing2016Weather[,"weatherType"];
avgTemp = lansing2016Weather[,"avgTemp"];
tempDept = lansing2016Weather[,"tempDept"];
windDirection =  lansing2016Weather[,"windPeakD
humidity = lansing2016Weather[,"relHum"];
barometer = lansing2016Weather[,"stnPressure"];
dewPoint = lansing2016Weather[,"dewPoint"];
precipitation = lansing2016Weather[,"precip"];
maxWind = lansing2016Weather[,"windSpeed"];

# Change all Trace precip value to 0.005
for(i in 1:366)
{
  if(precipitation[i] == "T")
  {
    precipitation[i] = 0.005;
  }
}

# Convert precipitation into a numeric vector
precipitation = as.numeric(precipitation);
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
Root/3-5b.R')
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R
Root/3-5b.R')
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R
Root/3-5b.R')
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R
Root/3-5b.R')
> 
```

```
  stnPressure : num 29.1 29 29 29.4 29.6 ...
  seaPressure : num 30.1 30 30 30.4 30.6 ...
  windSpeed : num 15.5 14.6 10 7.7 7.6 7.6 4.8 6.7 7.3 17.8 ...
  windPeakSpeed: int 31 30 30 21 21 20 13 15 17 42 ...
  windPeakDir : int 250 230 290 40 210 200 140 160 220 300 ...
  windSusSpeed : int 24 23 23 17 16 15 12 12 13 31 ...
  windSusDir : int 260 230 290 30 200 210 150 160 220 320 ...
Values
  avgTemp         int [1:366] 26 32 29 17 18 28 35 34 40 27 ...
  barometer       num [1:366] 29.1 29 29 29.4 29.6 ...
  dewPoint        int [1:366] 20 22 20 13 10 16 21 30 40 18 ...
  eventData       chr [1:366] "SN,BR" "" "SN" "SN,BR" "" "" ...
  humidity        int [1:366] 76 71 66 78 69 63 59 80 96 78 ...
  i               366L
  maxWind         num [1:366] 15.5 14.6 10 7.7 7.6 7.6 4.8 6.7 7...
  precipitation   num [1:366] 0.04 0 0.005 0.005 0 0 0 0.15 0.53...
  tempDept        num [1:366] 1.5 7.7 4.8 -7.1 -5.9 4.2 11.3 10....
```

Fig 4: Converting precipitation into a numeric vector (0.005 in place of "T").

5 - Scatterplots

Before doing any statistics, we can use scatterplots to see if there is evidence for a relationship between variables. The following code creates a scatterplot of ***humidity*** (y-axis) vs. ***avgTemp*** (x-axis).

```
plot(formula=humidity~avgTemp);
```

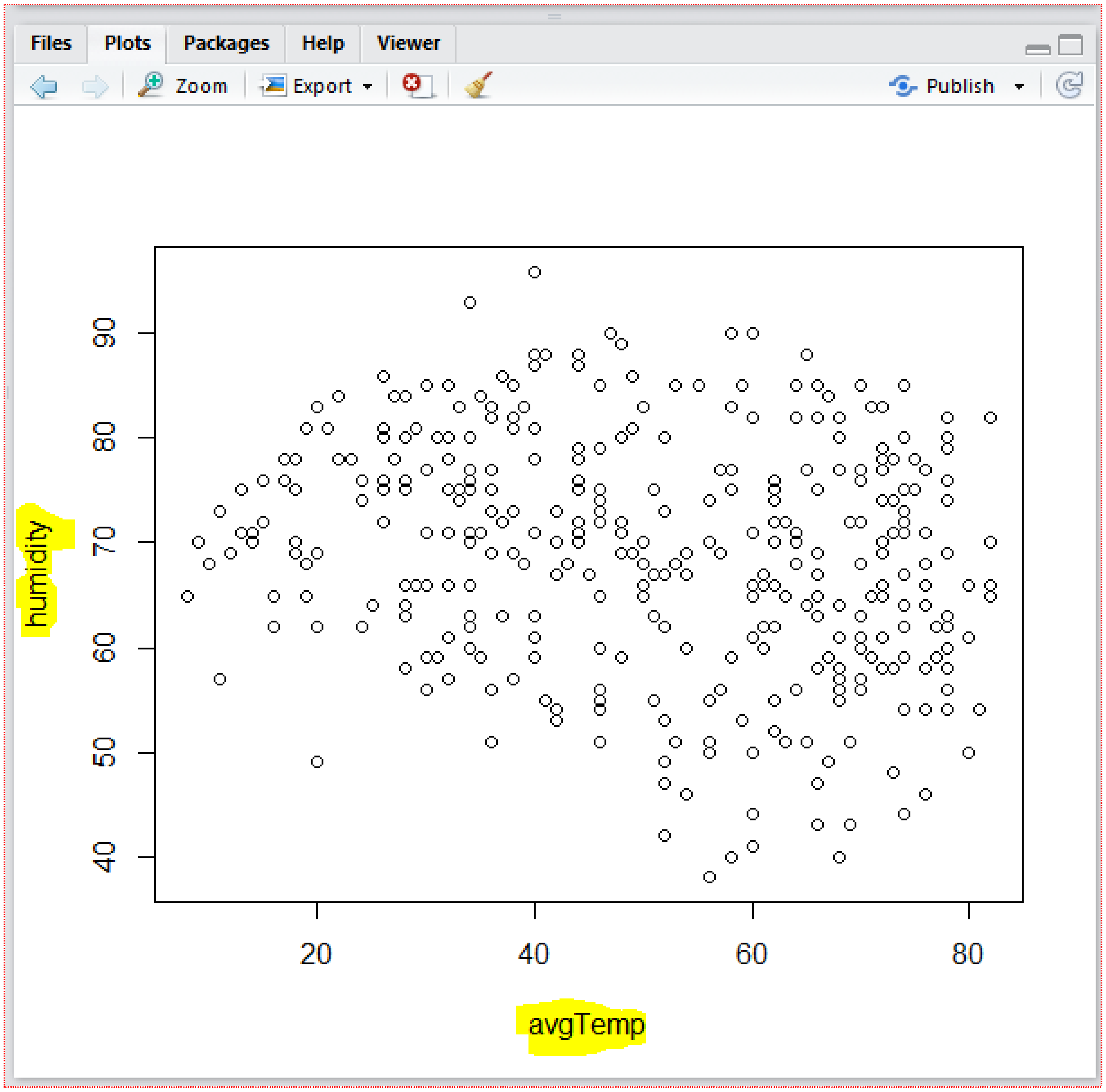


*Fig 5: **Humidity** vs **avgTemp** scatterplot.*

Trap: figure margins too large

We can make this plot look much better by changing some parameters:

- ***xlab, ylab***: x-axis and y-axis labels
- ***main***: title
- ***pch***: point character or type of point -- click here for a detailed list of pch types
- ***col***: color of the points
- ***cex***: character expansion or scaler for the points (so 0.7 means the points are scaled to 70% of default)

```
plot(formula=humidity~avgTemp,
     xlab="Average Temperature (F)", ylab="Humidity (%)",
     main="Humidity vs Average Temperature",
     pch=4, col="blue", cex=0.7 );
```

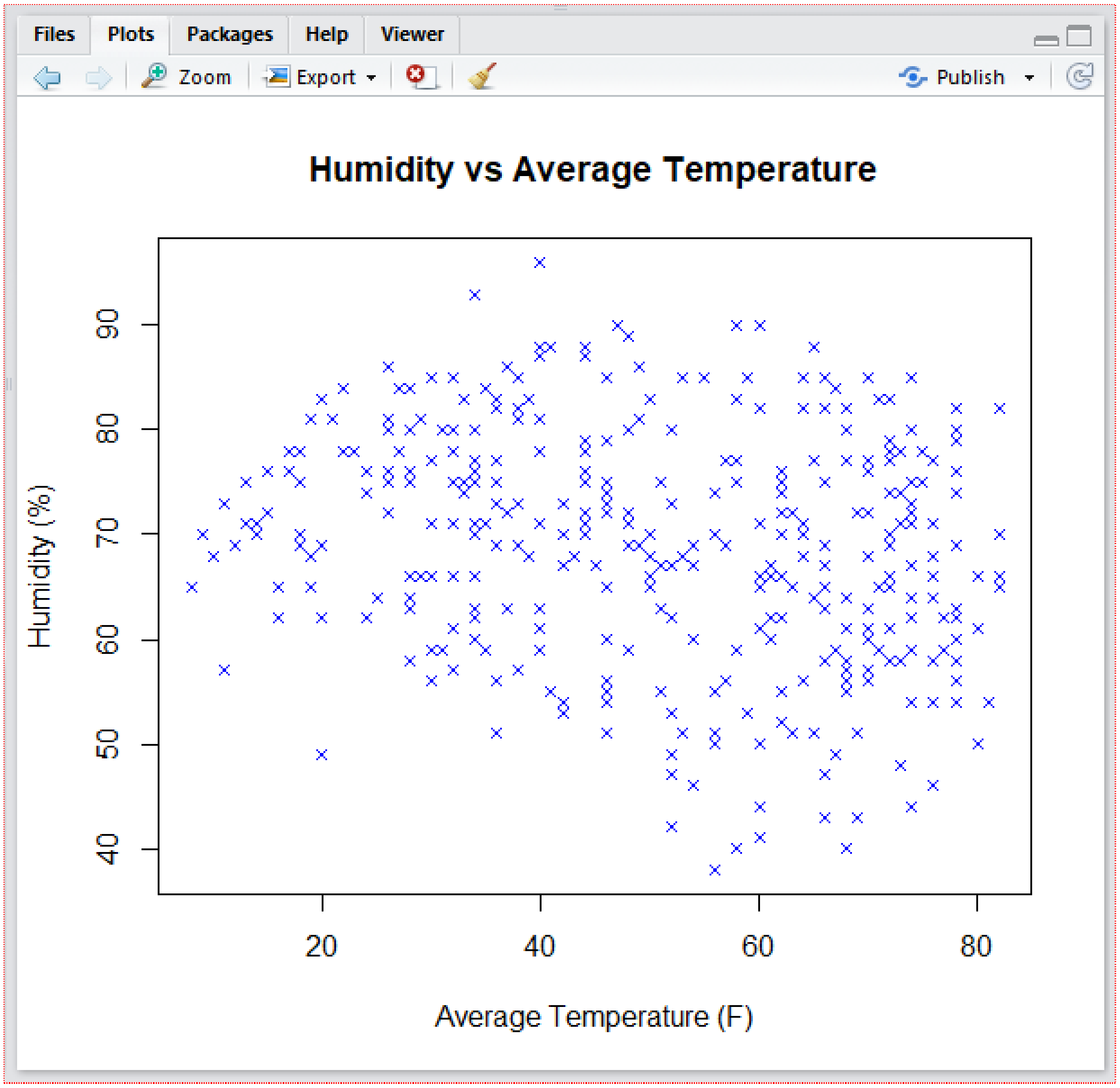


*Fig 6: **Humidity** vs **avgTemp** scatterplot with parameters changed*

Extension: Conditionally coloring the points
Extension: Numbering the points

6 - Multiple scatterplots using pairs()

Sometimes it is convenient to look at the relationship of multiple variables at once. In R this can be accomplished using the ***pairs()*** function. ***pairs()*** takes multiple vectors as input and outputs a scatterplot for each possible pair of vectors. The following code produces a plot of all six possible combinations of ***precipitation***, ***avgTemp***, and ***humidity***. Note: each plot will have a corresponding plot with the same data but reversed axes.

```
pairs(formula=~avgTemp+humidity+precipitation);
```

Extension: The + sign

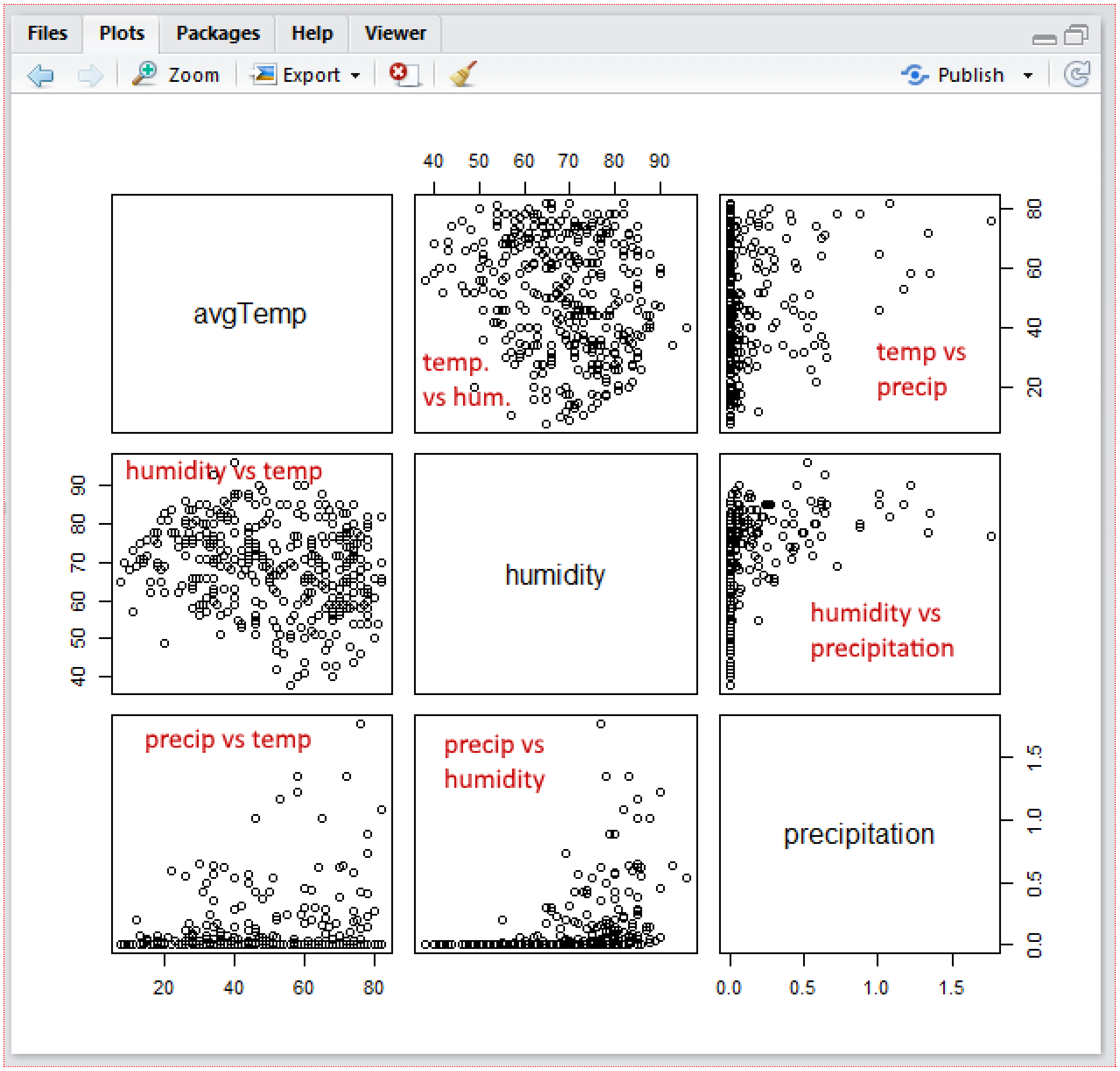


Fig 7: Multiple scatterplots -- red text added outside of R

6.1 - Even more pairs()

We can go crazy with ***pairs()*** and compare seven different variables

```
pairs(formula=~precipitation+avgTemp+maxWind+humidity+
                    barometer+dewPoint+windDirection);
```

The plots seem to show a strong relationship between ***avgTemp*** and ***dewPoint***. The two plots highlighted both represent a comparison between ***avgTemp*** and ***dewPoint***. The top plot has ***avgTemp*** on the y-axis, the bottom plot has ***dewPoint*** on the y-axis.

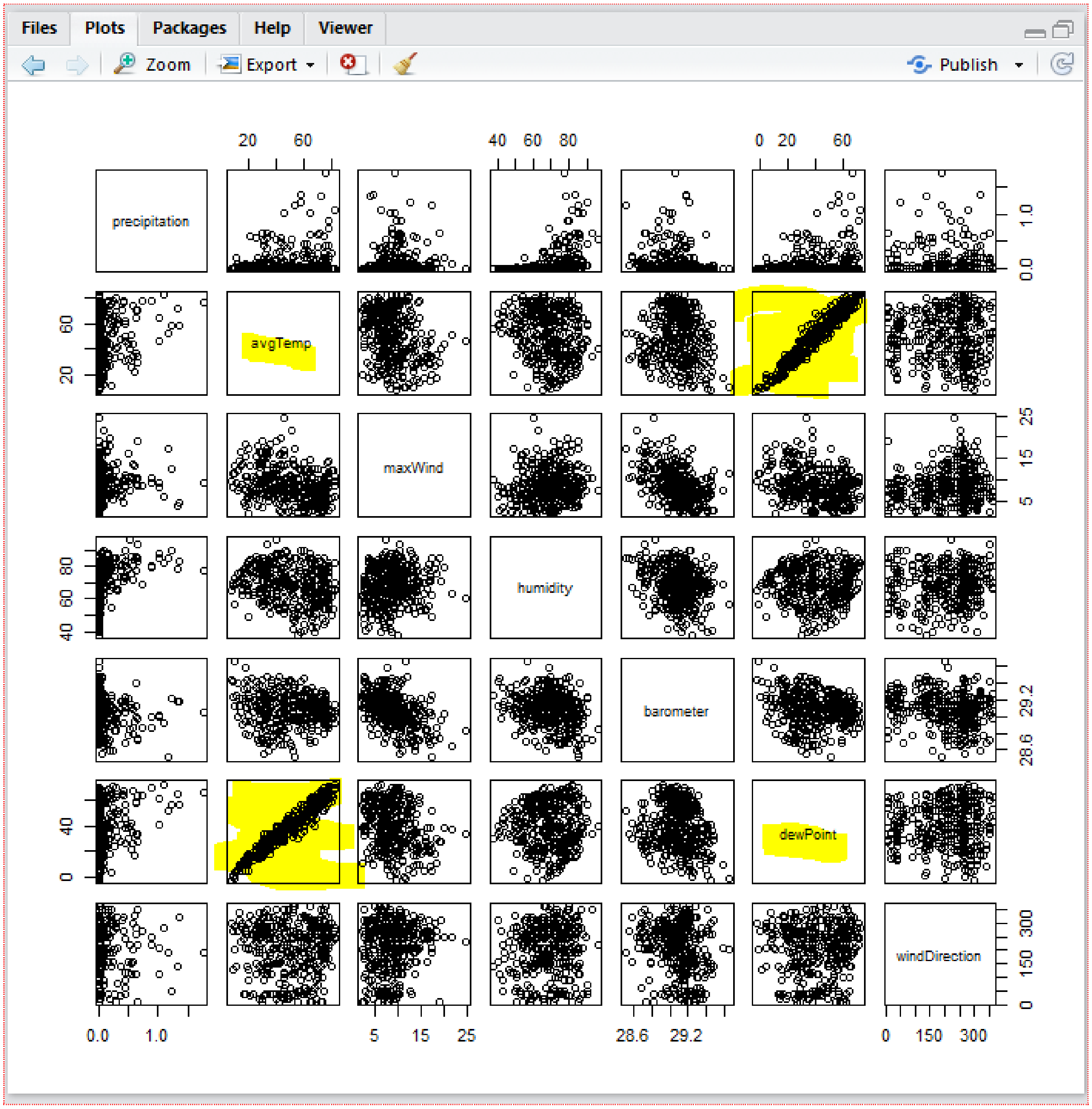


Fig 8: Scatterplot matrix relating seven different weather vectors.

7 - Linear Models

Taking a closer look at the ***avgTemp-dewPoint*** scatterplot seems to confirm a relationship (not a surprise if you know what dew point measures!):

```
plot(formula=avgTemp~dewPoint);
```

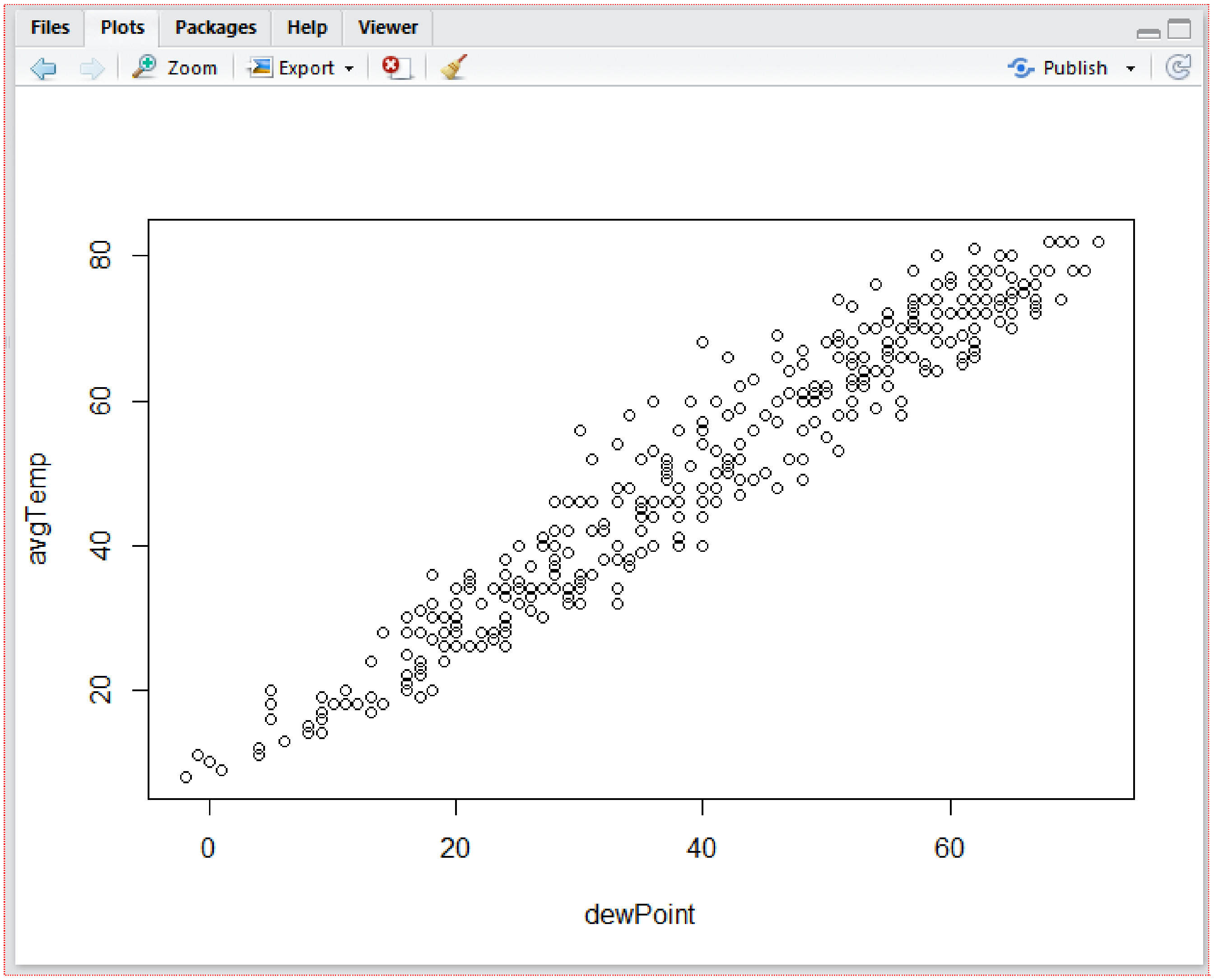


*Fig 9: **avgTemp** vs **dewPoint** scatterplot*

We want to quantify the relationship and we can do that by generating a linear model using ***avgTemp*** and ***dewPoint***.

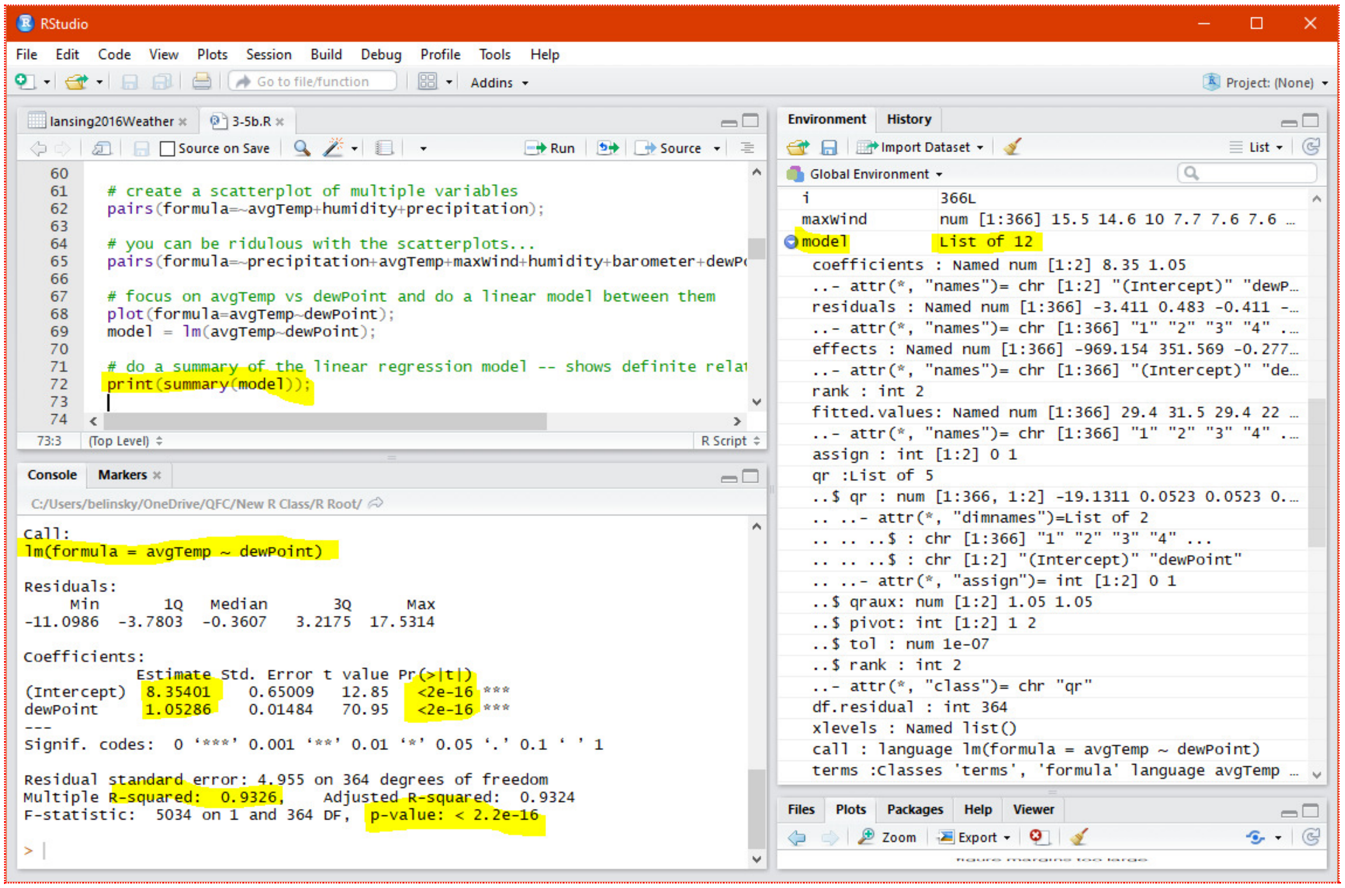
In R the function ***lm()*** performs a linear model. ***lm()*** generates a bunch of output that is in the form of a list. We will assign the list to the variable named ***model***.

```
model = lm(formula=avgTemp~dewPoint);
```

And, because viewing the list in the Environment Window is not especially helpful, we will use ***print()*** and ***summary()*** to get a *summary* of the results of the linear model and *print* the summary in the Console Window.

```
print(summary(model));
```

This is an example of nested functions, with ***summary()*** nested inside ***print()***. In other words, ***summary()*** is executed first and the results of ***summary()*** are used as inputs for the function ***print()***.



```
# create a scatterplot of multiple variables
pairs(formula=~avgTemp+humidity+precipitation);

# you can be ridulous with the scatterplots...
pairs(formula=~precipitation+avgTemp+maxWind+humidity+barometer+dewPo

# focus on avgTemp vs dewPoint and do a linear model between them
plot(formula=avgTemp~dewPoint);
model = lm(avgTemp~dewPoint);

# do a summary of the linear regression model -- shows definite relat
print(summary(model));
```

```
Call:
lm(formula = avgTemp ~ dewPoint)

Residuals:
     Min       1Q   Median       3Q      Max
-11.0986  -3.7803  -0.3607   3.2175  17.5314

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  8.35401    0.65009   12.85   <2e-16 ***
dewPoint     1.05286    0.01484   70.95   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.955 on 364 degrees of freedom
Multiple R-squared:  0.9326,	Adjusted R-squared:  0.9324
F-statistic:  5034 on 1 and 364 DF,  p-value: < 2.2e-16

>
```

```
i              366L
maxWind        num [1:366] 15.5 14.6 10 7.7 7.6 7.6 ...
model          List of 12
  coefficients : Named num [1:2] 8.35 1.05
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "dewP...
  residuals : Named num [1:366] -3.411 0.483 -0.411 -...
  ..- attr(*, "names")= chr [1:366] "1" "2" "3" "4" ...
  effects : Named num [1:366] -969.154 351.569 -0.277...
  ..- attr(*, "names")= chr [1:366] "(Intercept)" "de...
  rank : int 2
  fitted.values: Named num [1:366] 29.4 31.5 29.4 22 ...
  ..- attr(*, "names")= chr [1:366] "1" "2" "3" "4" ...
  assign : int [1:2] 0 1
  qr :List of 5
  ..$ qr : num [1:366, 1:2] -19.1311 0.0523 0.0523 0...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:366] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "dewPoint"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ qraux: num [1:2] 1.05 1.05
  ..$ pivot: int [1:2] 1 2
  ..$ tol : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
  df.residual : int 364
  xlevels : Named list()
  call : language lm(formula = avgTemp ~ dewPoint)
  terms :Classes 'terms', 'formula' language avgTemp ...
```

Fig 10: Information about the linear model in the Console Window

The output printed to the Console Window (*Fig 10*) is typical regression results such as parameter estimates (intercept: **8.354**, slope: **1.053**), p-values (**< 2e-16** means that it is lower than R can calculate), and R^2 statistic (0.9326).

7.1 - Adding the regression line to the scatterplot

The ***abline()*** function can read information from the linear model, ***model***, and add the relationship to the scatterplot.

```
abline(x=model, col="blue");
```

The code above adds a line to the scatterplot that has the x-axis intercept (**8.354**) and slope (**1.053**) given by the linear model.

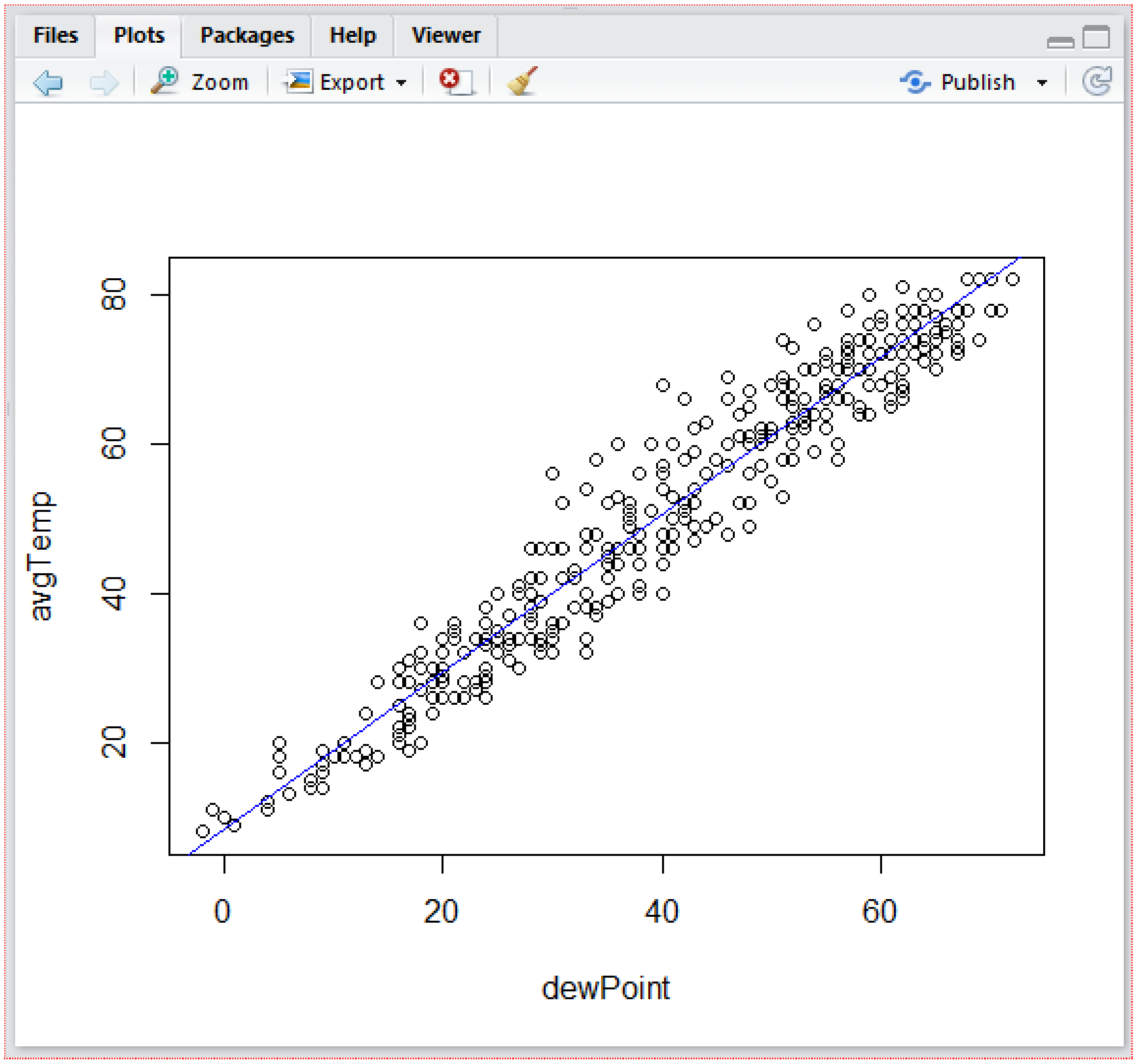


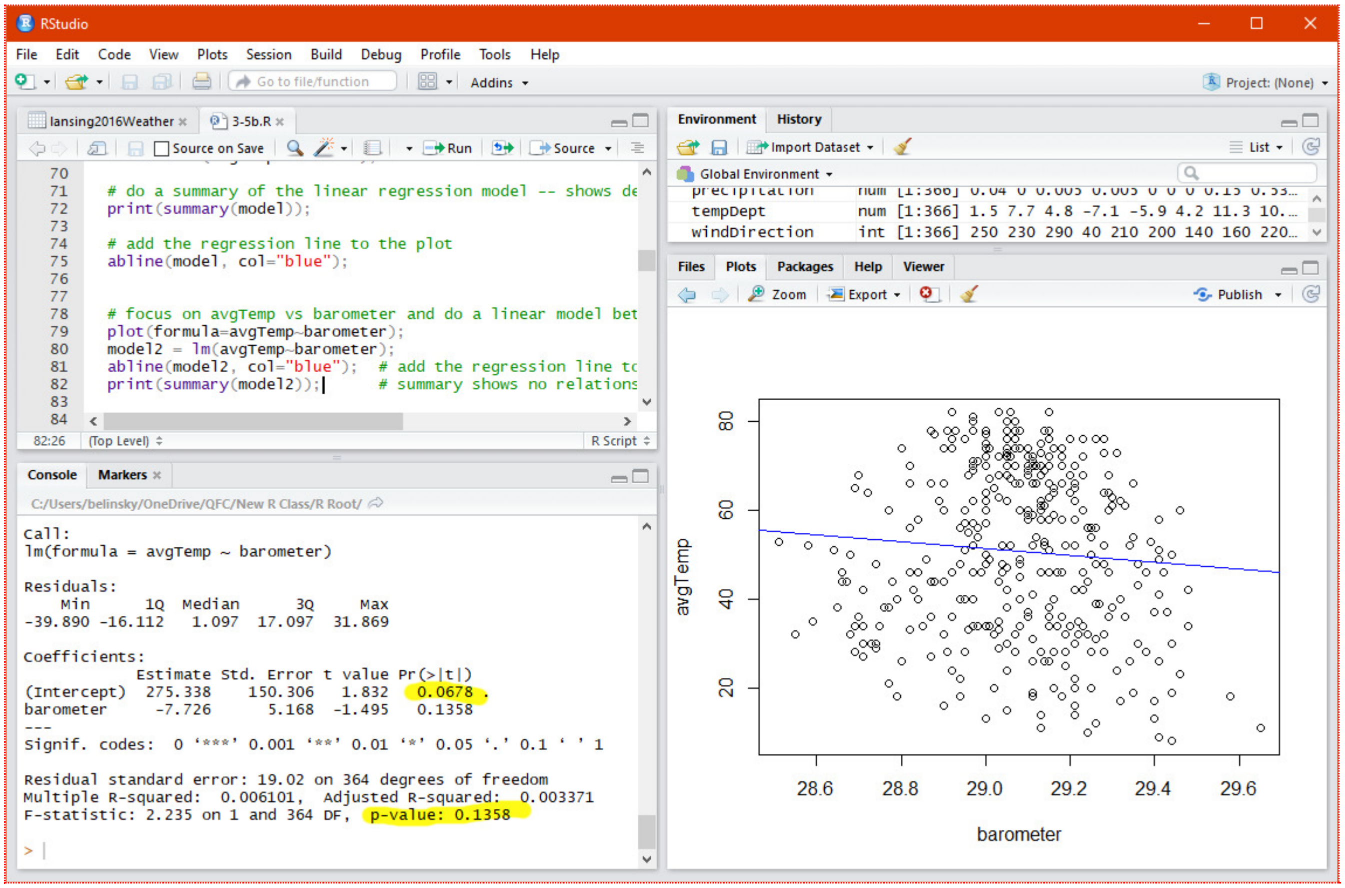
*Fig 11: Regression line added to **avgTemp** vs **highTemp** scatterplot using **abline()***

8 - Conflating variables in a linear regression: preview

We are going to perform a second regression, this time between barometric pressure and temperature. Like last time we will:

1) produce a scatterplot of ***avgTemp*** vs ***barometer*** using ***plot()***
2) create a linear model using ***lm()***
3) summarize the model in the Console Window using ***print(summary())***
4) use ***abline()*** *to* add the regression line from the model to the scatterplot

```
plot(formula=avgTemp~barometer);
model2 = lm(avgTemp~barometer);
abline(model2, col="blue");
print(summary(model2));
```



```
# do a summary of the linear regression model -- shows de
print(summary(model));

# add the regression line to the plot
abline(model, col="blue");


# focus on avgTemp vs barometer and do a linear model bet
plot(formula=avgTemp~barometer);
model2 = lm(avgTemp~barometer);
abline(model2, col="blue");  # add the regression line tc
print(summary(model2));      # summary shows no relations
```

```
precipitation      num [1:366] 0.04 0 0.005 0.005 0 0 0 0.15 0.33...
tempDept           num [1:366] 1.5 7.7 4.8 -7.1 -5.9 4.2 11.3 10....
windDirection      int [1:366] 250 230 290 40 210 200 140 160 220...
```

```
Call:
lm(formula = avgTemp ~ barometer)

Residuals:
    Min      1Q  Median      3Q     Max 
-39.890 -16.112   1.097  17.097  31.869 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept)  275.338    150.306   1.832   0.0678 .
barometer     -7.726      5.168  -1.495   0.1358  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 19.02 on 364 degrees of freedom
Multiple R-squared:  0.006101,	Adjusted R-squared:  0.003371 
F-statistic: 2.235 on 1 and 364 DF,  p-value: 0.1358

> 
```

Fig 12: Linear model showing no significant relationship between temperature and pressure

While there seems to be little evidence for a relationship (p=0.1358) between ***avgTemp*** and ***barometer***, we will find out in the next lesson that we can tease out a relationship if we subset the data and look at other conflating factors like wind direction.

9 - Saving vectors to a csv file

We will be using the data from the vectors in the application for this lesson and in future lessons so it would be best to save the vectors to a new CSV file. However, there is an intermediate step -- we first must save the vector to a new data frame and then save the data frame to a CSV file.

We can call ***data.frame()*** to create a new data frame, called ***formattedData***, from the 12 weather vectors.

```
formattedData = data.frame(date, eventData, avgTemp, tempDept,
                           precipitation, humidity, barometer,
                           dewPoint, avgWind, maxWind,
                           windDirection, sunrise, sunset);
```

And then we can write the data frame ***formattedData*** to a csv file called ***LansingNOAA2016Formatted.csv,*** which we will put in the ***data*** folder in the ***R Root*** directory.

```
write.csv(x=formattedData,
          file="data/LansingNOAA2016Formatted.csv",
          row.names = TRUE );
```

There should now be a file called ***LansingNOAA2016Formatted.csv*** in your data folder.

10 - Application

For the application, you are going to use the data found in ***LansingNOAA2016Formatted.csv***

1) place the function below, ***timeConvert()***, inside your ***toolbox.r*** file. This function takes a time in this format: HHMM and converts it to hours. So **1730 = 17.5, 445 = 4.75, 2012 = 20.2**

```
timeConvert = function(hoursMinutes)
{
  returnVector = c();
  for(i in 1:length(hoursMinutes))
  {
    numDigits = nchar(as.character(hoursMinutes[i]));
    minutes = substr(hoursMinutes[i], numDigits-1, numDigits);
    minutes = as.numeric(minutes)*(1/60);
    hours = floor(hoursMinutes[i]/100);
    returnVector[i] = round(hours+minutes, 2);
  }
  return(returnVector)
}
```

2) Using the function above, ***timeConvert()***, convert the ***sunrise*** and ***sunset*** vectors to hours.

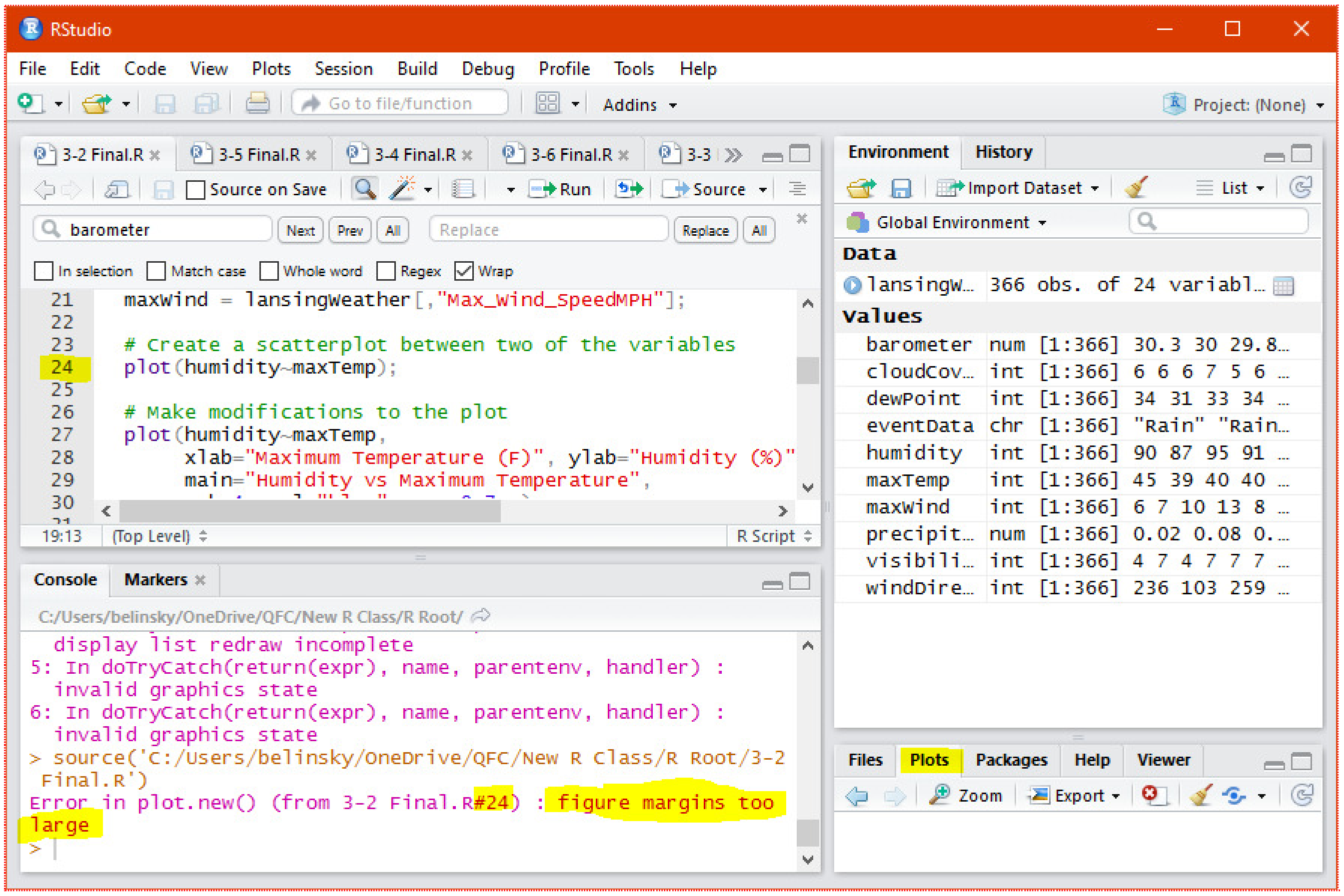
3) Create a new vector called ***hoursOfSun.*** ***hoursOfSun*** is the amount of time the sun is up on a particular day. It is sunset time minus sunrise time.

4) What variables does ***hoursOfSun*** correlate with?

- create scatterplots between ***hoursOfSun*** and other variables
- perform linear models between ***hoursOfSun*** and at least three other variables.
- print to the Console Window and explain the summary of the linear models
- place regression lines on the scatterplots (so, one regression line per scatterplot)

11 - Trap: figure margins too large

A common, and somewhat unintuitive, error you may get when executing a script that produces plots is: *figure margins too large*


```
maxWind = lansingWeather[,"Max_Wind_SpeedMPH"];

# Create a scatterplot between two of the variables
plot(humidity~maxTemp);

# Make modifications to the plot
plot(humidity~maxTemp,
     xlab="Maximum Temperature (F)", ylab="Humidity (%)"
     main="Humidity vs Maximum Temperature",
```

```
  display list redraw incomplete
5: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
6: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-2
 Final.R')
Error in plot.new() (from 3-2 Final.R#24) : figure margins too
 large
>
```

```
Data
lansingW…   366 obs. of 24 variabl…
Values
barometer   num [1:366] 30.3 30 29.8…
cloudCov…   int [1:366] 6 6 6 7 5 6 …
dewPoint    int [1:366] 34 31 33 34 …
eventData   chr [1:366] "Rain" "Rain…
humidity    int [1:366] 90 87 95 91 …
maxTemp     int [1:366] 45 39 40 40 …
maxWind     int [1:366] 6 7 10 13 8 …
precipit…   num [1:366] 0.02 0.08 0.…
visibili…   int [1:366] 4 7 4 7 7 7 …
windDire…   int [1:366] 236 103 259 …
```

Fig 13: Figure margins too large error

This error often has nothing to do with your script. Rather, the issue is that your plot window is too small to hold the plot you are trying to display. To fix this issue you just need to increase the size of the plot window.

12 - Extension: Conditionally coloring points on a plot

On the ***humidity*** vs. ***avgTemp*** plot, we have 366 point representing each day of the year but the plot tells you nothing about the days. We can add information about the points by controlling their color.

The colors indicate what day the point represents. In the plot below, yellow points represent early days in the year (e.g., January, February), red points represent middle days (e.g., June, July), and blue color represent later days (e.g., November, December). There is a spectrum so days in March and April are orangish and days in September and October are purpleish.

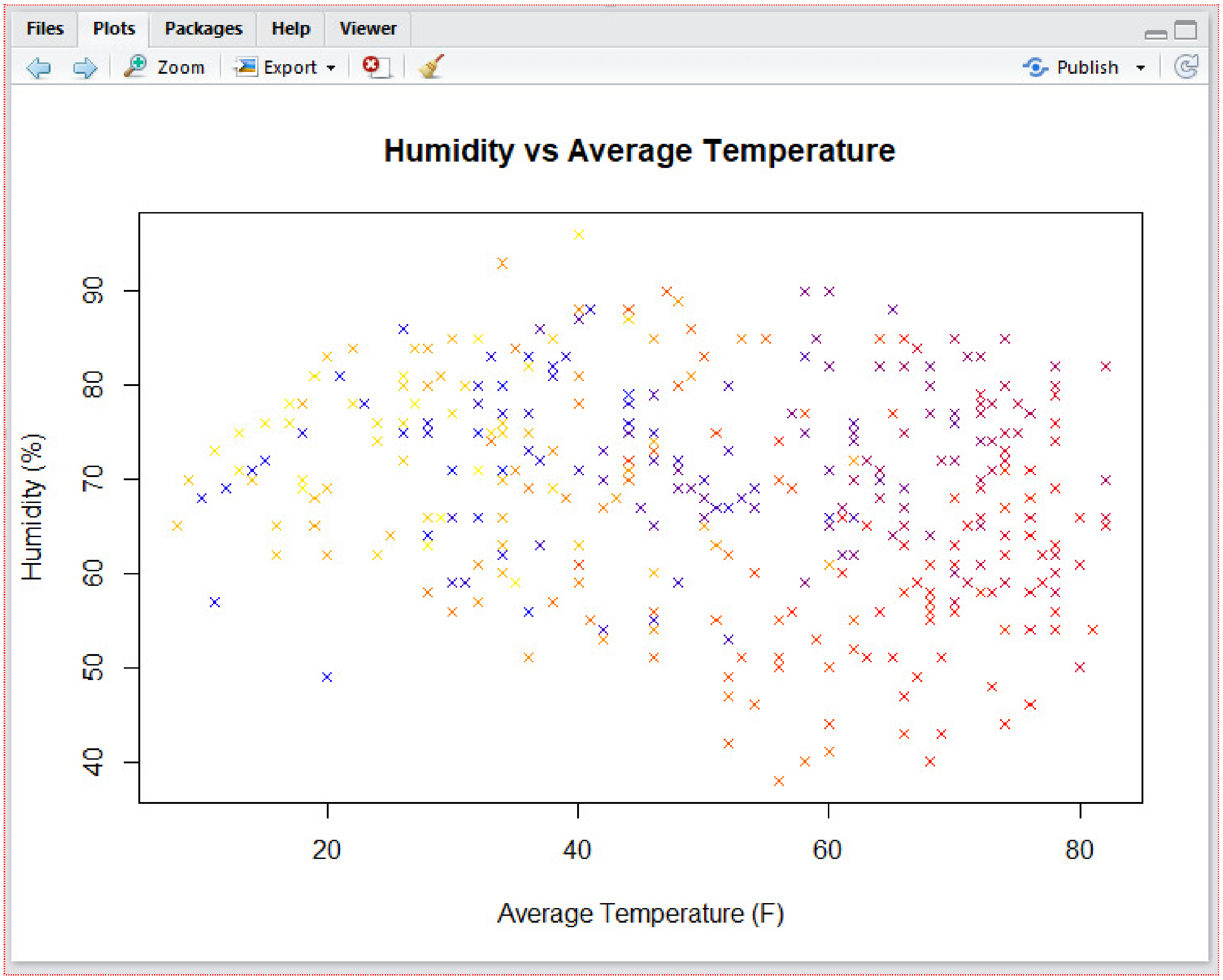


Fig 14: Adding color to the points to represent days.

The first step is to create a color scheme that goes from yellow to red to blue. We do that using the function ***colorRampPalette().***

```
colorByDay=colorRampPalette(color=c("yellow", "red", "blue"));
```

colorByDay is technically a function that holds information about a yellow-red-blue color scheme. ***colorByDay*** can be used to create color values in a plot:

```
plot(formula=humidity~avgTemp,
     xlab="Average Temperature (F)", ylab="Humidity (%)",
     main="Humidity vs Average Temperature",
     pch=4, col=colorByDay(366), cex=0.7 );
```

colorByDay is a function expecting a value. The value is the number of colors you want. Since I have 366 points, I want to use 366 colors in the yellow-red-blue color scheme. This means that there are 366 different colors where color 1 is yellow, 366 is blue, and halfway in between, 183, is red. All colors in between are calculated.

You can use as many colors as you want in the ***colorRampPalette.*** This will work for scenarios where you want evenly-spaced colors.

13 - Extension: Using text instead of points

We can add more information to the scatterplot by replacing the points on the plot with text. In this case, the text will represent the days of the year(1 though 366).

We will first make the figure without plotting the points. We do that by setting ***type*** to "n", which means *no* plot.

```
plot(formula=humidity~avgTemp, type="n",
     xlab="Average Temperature (F)", ylab="Humidity (%)",
     main="Humidity vs Average Temperature" );
```

This creates an empty plot. We need to "manually" add points to the plot. But, instead of points, we are going to add numbers that represent the 366 days. So 1 is Jan 1, 2 is Jan 2, etc.

We use the ***text()*** function to add text points to the scatterplot

```
text(formula=humidity~avgTemp, labels=1:366, cex= 0.7, col=colorByDay(366));
```

text() uses many of the same parameters as ***plot()***. ***cex*** is the size of text points, ***col*** uses the color scheme from the previous extension.
labels set the text starting with 1 and going up to 366.

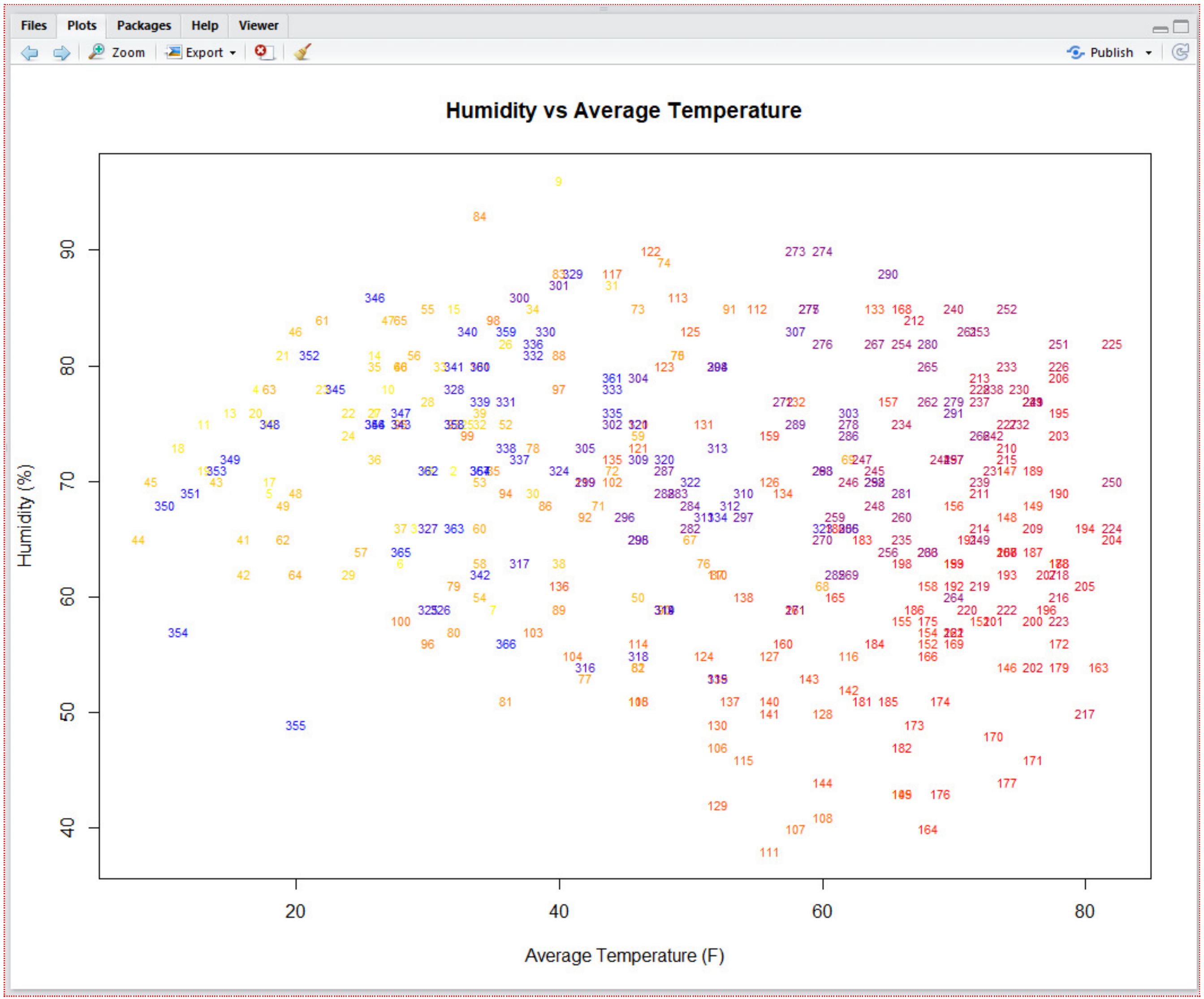


Fig 15: Using text as points in a scatterplot

Lastly, if you want to make the scatterplot bigger, then you can use the Zoom button (highlighted). Zoom will open the scatterplot in a new window and you can resize it to your full screen.

14 - Extension: Factors in R

R, by default, will treat any column from a CSV file that has any characters in it as a **factor**. Functionally, this means that R groups equivalent values and assigns a number to them. So, if my values were: c("llama", "alpaca", "guanaco", "llama", "llama", "guanaco, "alpaca"). R would create three factors ("llama", "alpaca, "guanaco") and assign them the *ordinal numbers* 1,2, and 3.

So, this vector:

```
{
  c("llama", "alpaca", "guanaco", "llama", "llama", "guanaco, "alpaca");
}
```

would, as a **factor**, look like this:

```
{
  c(1,2,3,1,1,3,2);
}
```

Note: these are ordinal number meant to express order, these numbers are not meant to be used in mathematical operators (i.e., 1 + 2 =3 or 1*3 = 3 does not make sense in this case)

The reality is that, at this level, students do not need to worry about the distinction between **strings** and **factor** in R. R will, for the most part, intelligently make the conversion from string to factor when a grouping is needed. We have already seen this happen when we did boxplots and ANOVA on temperature data across years in lesson 3-4.

I recommend, as a default, setting the parameter ***stringAsFactor =*** **FALSE** whenever you pull in data from a CSV file. In other words, treat strings as strings.

15 - Extension The + sign

R using the ***+*** sign in formula to declare multiple independent variables. So, if you wanted to look at how ***temperature*** depends on both ***humidity*** and ***windSpeed***, you would use the formula:

```
temperature ~ humidity +  windSpeed;
```

When doing scatterplots, the + sign has a slightly different meaning in that it declare all the independent variables that you are plotting.

```
pairs(formula = ~temperature + humidity + windSpeed);
```

In this case, ***temperature***, ***humidity***, and ***windSpeed*** are all plotted against each other so they all act as independent and dependent variables. But, in would be redundant to also declare each as dependent since that is what ***pairs()*** does.