Linear Modeling with a Non-Categorical Predictor Variable

ANTH 3720-001 Archaeological and Forensic Science Lab Methods: Data Analysis with R

Elic Weitzel

Jan. 29-31, 2021

If you would like the original R Markdown file, find it on my GitHub page at https://github.com/weitzele/Basic-R-Tutorial

1 Linear modeling with non-categorical predictors

Up to this point, we have built linear models with categorical predictor variables - groups of data with no inherent order to them. Now, let's build some linear models with non-categorical predictors. In these models, the predictor variable (commonly on the x-axis) will be either continuous or a ranked discrete variable (groups which have an order to them).

1.1 Prepare the Data

Instead of using the `archdata` package, let's load some other data for this modeling exercise. We'll use some zooarchaeological data from a 1994 paper by Jack Broughton (Broughton, J.M. (1994) Journal of Arch. Sci. 21(4):501-514) on the abundance of various animal species through time.

So let's load this .csv file, which is provided to you along with this tutorial, and assign it to the new object `sac.data`.

```
sac.data <- read.csv("Broughton1994JAS_SacramentoValley.csv")
```

Let's now inspect this dataset and see what we're dealing with.

```
head(sac.data) #check the first six rows for each column
```

```
##   Site Date Sylvilagus Lepus    LS_Index Mammals Fish     FM_Index
## 1   68 3665         10    31  0.51219512     213    7 -0.936363636
## 2  105 2875          7     8  0.06666667     336   19 -0.892957746
## 3  101 2650         27    13 -0.35000000     441   53 -0.785425101
## 4  288 1650          9     5 -0.28571429     385   51 -0.766055046
## 5   99 1470        671   263 -0.43683083    3749 3699 -0.006713212
## 6   12  400         10     5 -0.33333333     102 1849  0.895438237
```

```
colnames(sac.data) #check the column names
```

```
## [1] "Site"       "Date"       "Sylvilagus" "Lepus"      "LS_Index"
## [6] "Mammals"    "Fish"       "FM_Index"
```

```
nrow(sac.data) #see how many rows this data frame contains
```

```
## [1] 9
```

It appears that we have a data frame with 8 columns and 9 rows. Each row represents one of nine sites in the Sacramento Valley of California. In the second column, we have the mean dates for each site.

First, let's take a look at the last column, "FM_Index". This "FM_Index" column stands for *Fish-Mammal Index*. I calculated it based on the specimen counts for fish and mammals reported in Broughton (1994). It represents the relative abundance of fish versus mammals in each site. When the index value is positive, there are more fish bones than mammals. When the index is negative, there are more mammals than fish.

Let's pull up the data from this column.

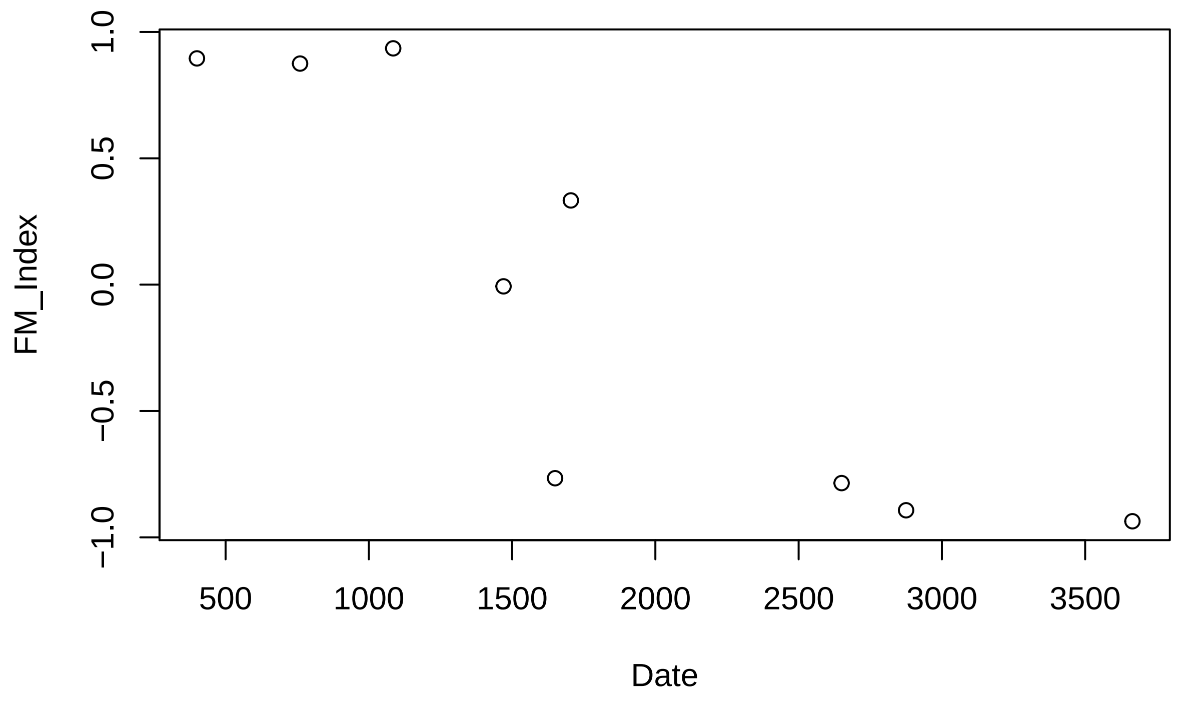
```
sac.data$FM_Index
```

```
## [1] -0.936363636 -0.892957746 -0.785425101 -0.766055046 -0.006713212
## [6]  0.895438237  0.935188715  0.875000000  0.333333333
```

This column contains continuous values (they have decimals) ranging both above and below zero (possibly unbounded). These data therefore appear to fit a normal/Gaussian distribution, so we ought to be able to use the `lm()` function to model them.

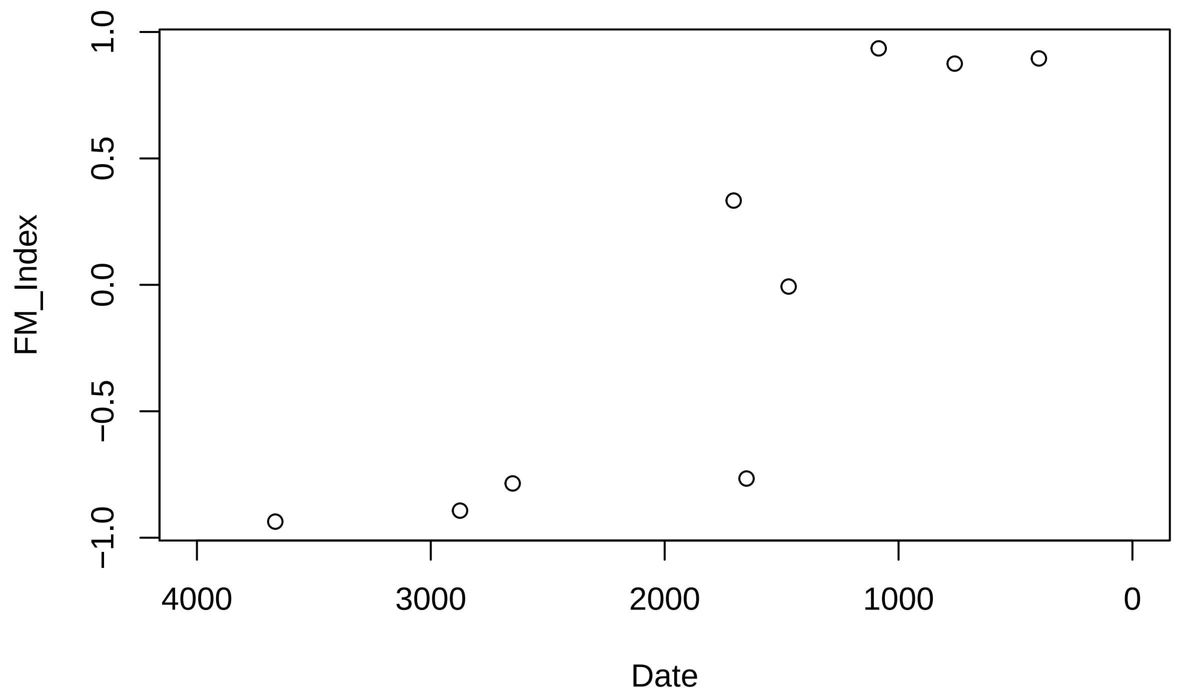
But before we model anything, we should always visualize our data. Let's plot the FM Index as a function of time (the "Date" column).

```
plot(FM_Index ~ Date, data = sac.data)
```



Based on this plot, we can see that our index value is high from dates of around 500 to 1000 BP (before present). Around the dates 2500 to 3500 BP, it is quite low. If you're struggling to interpret this plot because the dates are running from the present to the past, we can flip the x axis using the `xlim` argument.

```
plot(FM_Index ~ Date, data = sac.data, xlim = c(4000, 0))
```



Now our x-axis is oriented in a more conventional way for archaeologists, if you prefer that, with the past on the left and the present on the right.

It therefore appears that there is a positive trend in the FM Index as we move towards the present.

But is this trend strong? Statistically significant? Let's model the data to find out!

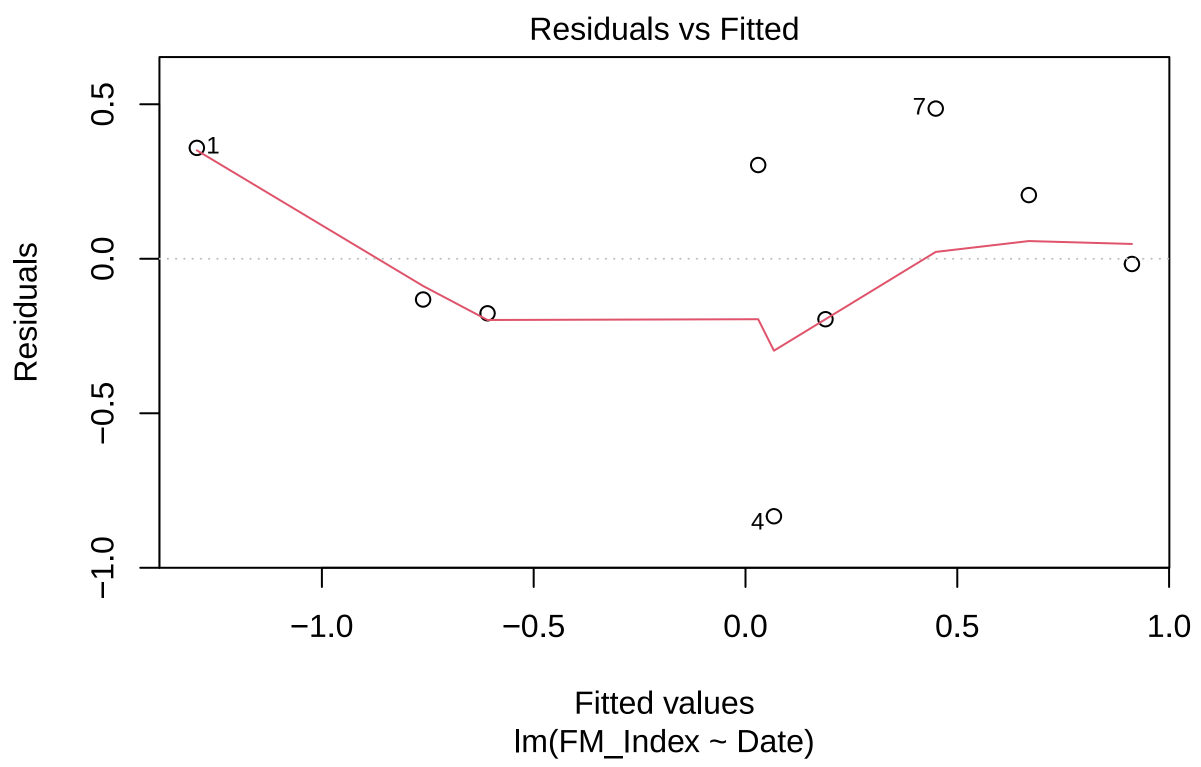
1.2 Build the model and inspect the diagnostic plots

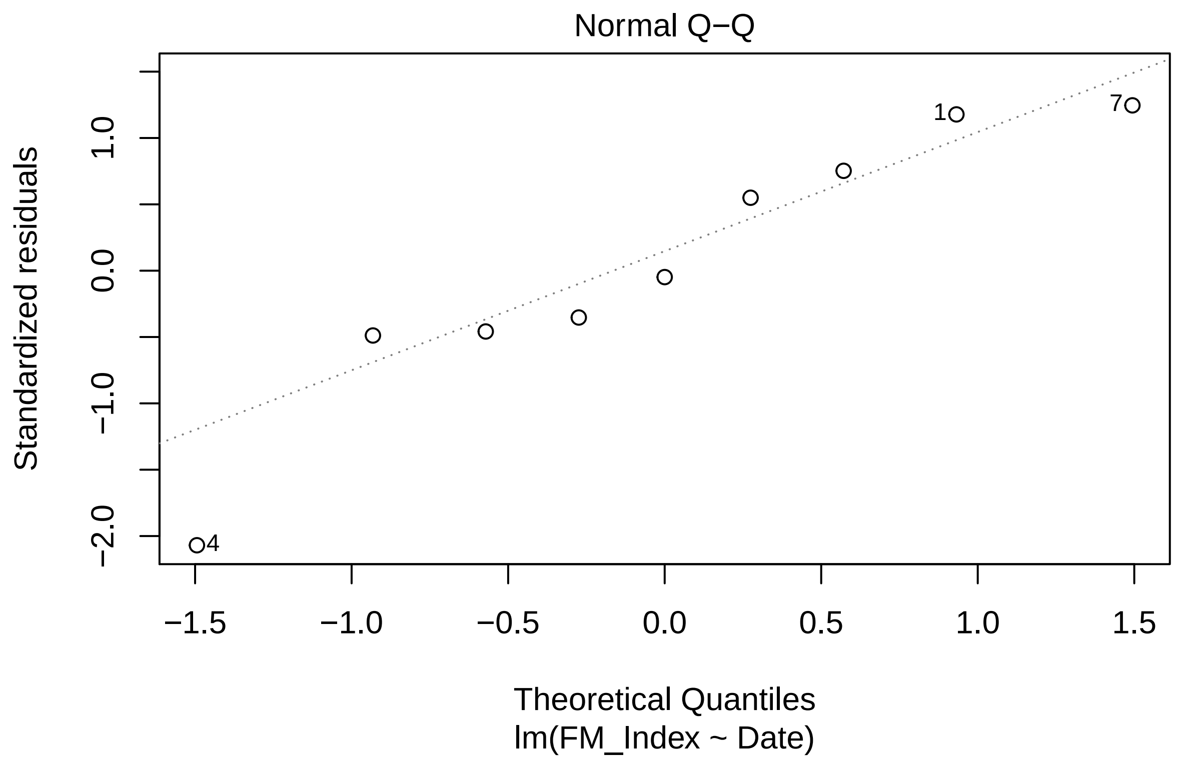
Let's use the `lm()` function to create a model of the FM index as a function of time.

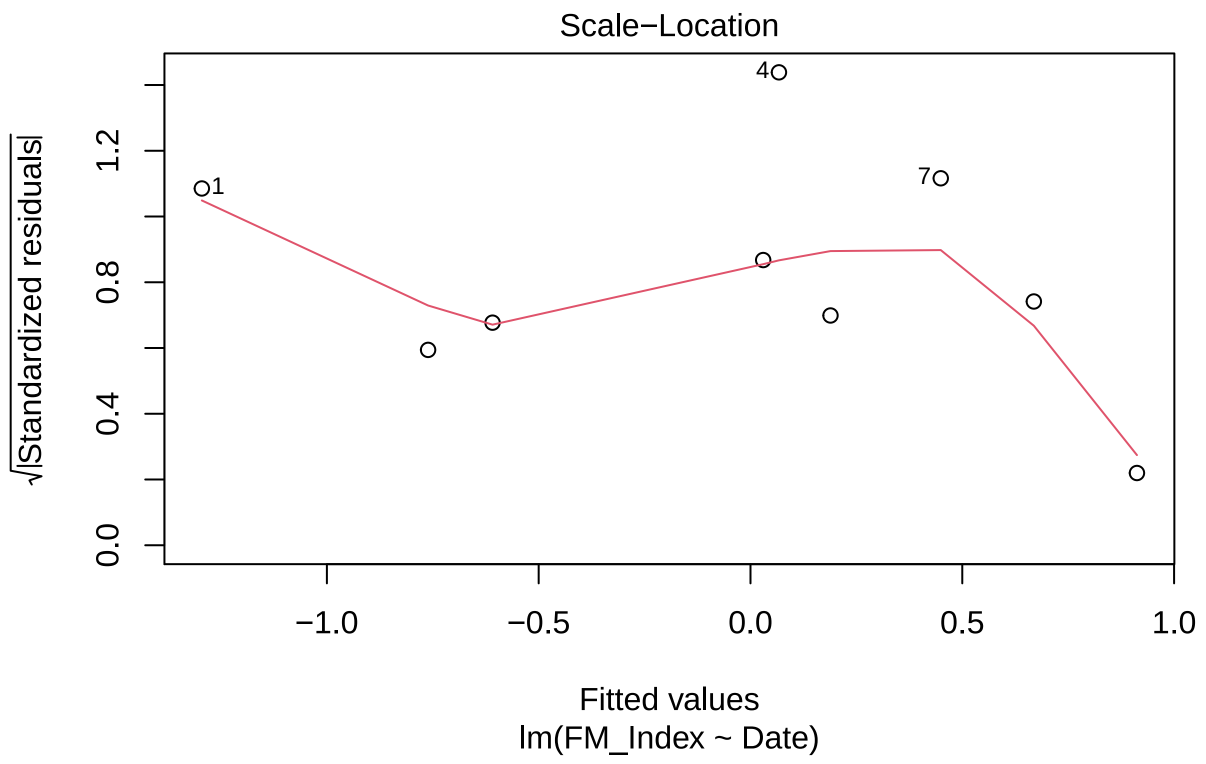
```
sac.mod <- lm(FM_Index ~ Date, data = sac.data)
```

With our model built, let's now check the diagnostic plots. Remember to put your cursor in the R Console and click through each of them.

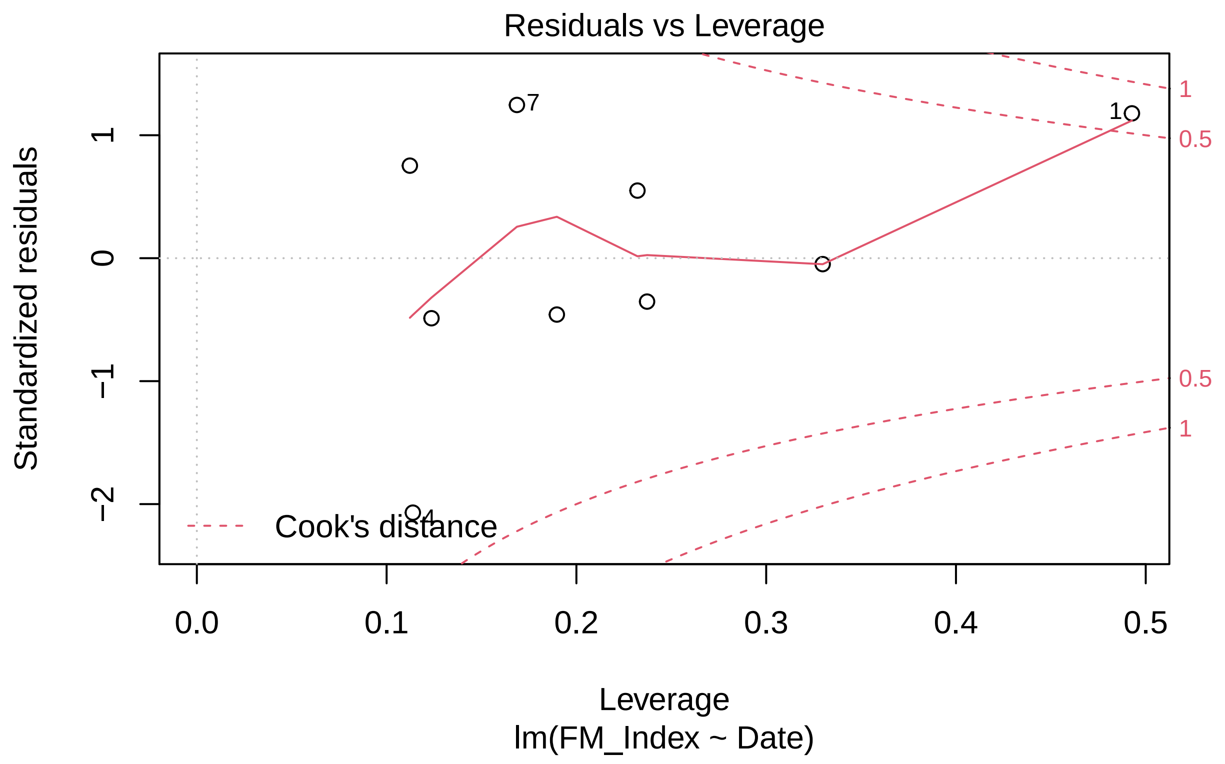
```r
plot(sac.mod)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(FM_Index ~ Date)
1
7
4
1.0
0.0
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5



Scale−Location
√|Standardized residuals|
0.0
0.4
0.8
1.2
−1.0
−0.5
0.0
0.5
1.0
Fitted values
lm(FM_Index ~ Date)
1
4
7



We can see that our residuals are more or less scattered randomly around the dotted line, which is good. Our qq-plot also looks good, with all of the residuals falling along the dotted line, indicating that they're normally distributed. There's a bit of a downward trend in the standardized residuals in the third plot, but it's really just driven by that last point at the highest fitted value, so I wouldn't consider this to be a problem.

That same point also appears to have a somewhat outsized influence on our model, as the fourth plot shows. Point 1 is located outside the 0.5 dotted red line, which is a representation of a metric called Cook's Distance. As before, you don't need to know how Cook's Distance is calculated in most cases, but this plot is simply telling us that point 1 has a really big impact on the model. This means that it will have a strong influence on the parameters we estimate for our regression model, and that if we eliminated this point, our model could be quite different.

But, as a general rule, I would not remove outliers from my data. It's a slippery slope from removing an outlier to p-hacking… If you have good reason to do so, you can, but generally it's not a good idea. And this Cook's distance plot is not meant to tell you what points to remove from your data, it's simply meant to inform you that some might have an outsized impact on the regression for some reason - that might not always be a bad thing. Looking at the main plot of our actual data through time, no points appear to be unreasonable. I would say we are fine to proceed.

1.3 Interpret the model

So now, let's inspect the model summary.

```
summary(sac.mod)
```

```
##
## Call:
## lm(formula = FM_Index ~ Date, data = sac.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.83325 -0.17650 -0.01691  0.30332  0.48598
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.1827966  0.2938171   4.026  0.00502 **
## Date        -0.0006761  0.0001422  -4.755  0.00207 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4278 on 7 degrees of freedom
## Multiple R-squared:  0.7636, Adjusted R-squared:  0.7298
## F-statistic: 22.61 on 1 and 7 DF,  p-value: 0.002072
```

This model output looks pretty similar to those we've seen before, but there is a big difference that isn't immediately obvious.

We're no longer running a linear model on a categorical predictor variable. Before, our coefficient estimates represented the effect of belonging to a particular group. But now, they represent the effect of time on our index. This will become clear in a moment.

So as before, our model equation is $y = \beta_0 + \beta_1 * x$. Our intercept, β_0, is 1.183 in this case. Our slope, β_0, which is the coefficient for date, is -0.00067. Remember that this date coefficient is negative because R thinks that our x-axis is running from 0 to 4000, even if we plotted it in reverse. Our response variable, y, is what we're trying to predict, while x is our predictor variable: Date. So when we're using this model to predict the response variable y, which is the FM Index, we multiply our coefficient for Date, β_1 by the date, then add that to the intercept, β_0.

So in the case of this model, if we wanted to estimate the FM Index for a date that we don't have in our dataset, we could use this model. For example, there's a bit of a gap between around 1800-2500 BP. Let's use this model equation to predict what the FM Index would be for a date of 2200 BP. We can use the model object we created to run this calculation because it contains the coefficients in it.

```
sac.mod$coefficients #inspect the coefficients within the model object
```

```
##   (Intercept)          Date
##  1.1827965527 -0.0006761208
```

```
sac.mod$coefficients[1] #the intercept
```

```
## (Intercept)
##    1.182797
```

```
sac.mod$coefficients[2] #the coefficient for date (i.e. slope)
```

```
##          Date
## -0.0006761208
```

```
#now take the date coefficient times 2200 BP, and add the intercept
FMIndex_2200BP <- sac.mod$coefficients[1] + (sac.mod$coefficients[2] * 2200)

FMIndex_2200BP
```

```
## (Intercept)
##  -0.3046691
```

And there we have it! The estimated FM Index value for 2200 BP is -0.305.

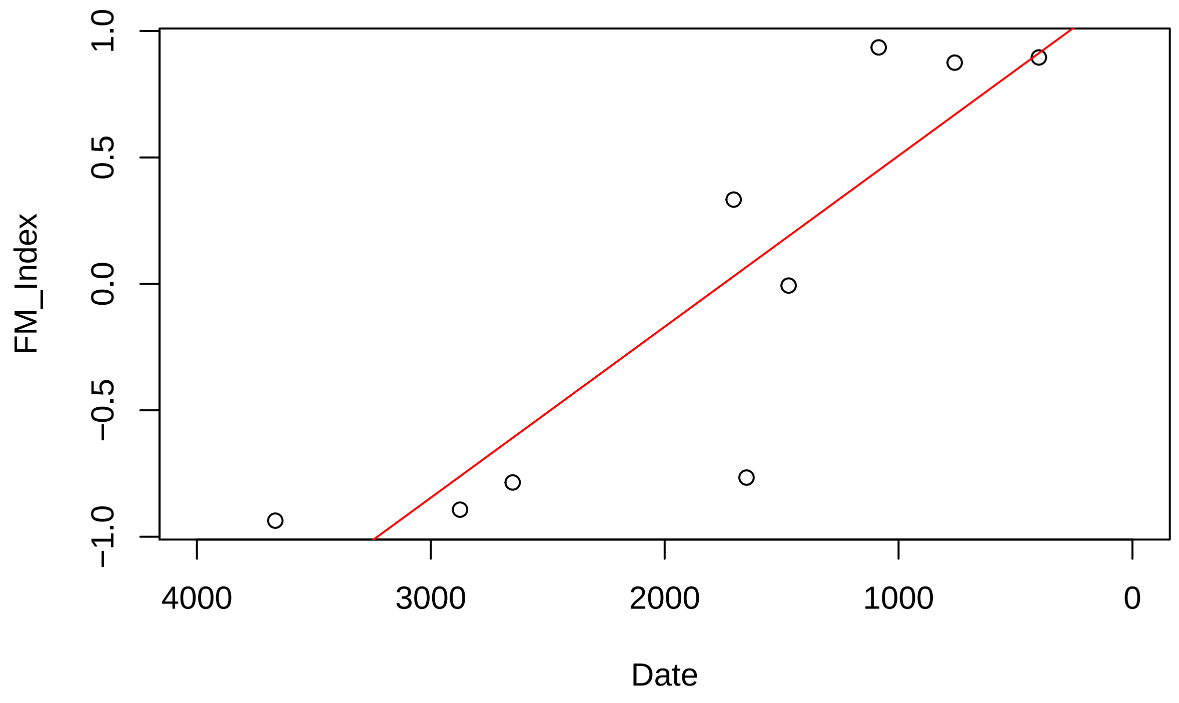
1.4 Plot the model fit

Now, very rarely will you be interested in calculating a value for a certain date, like this. More often, you'll want to simply plot the trendline: the model fit. This model fit will visually illustrate the relationship between Date and the FM Index.

We can do this quite easily with an `lm` model object using the `abline()` function, which simply draws our model fit for us.

```
plot(FM_Index ~ Date, data = sac.data, xlim = c(4000, 0)) #plot the data

abline(sac.mod, col = "red") #add the model fit
```



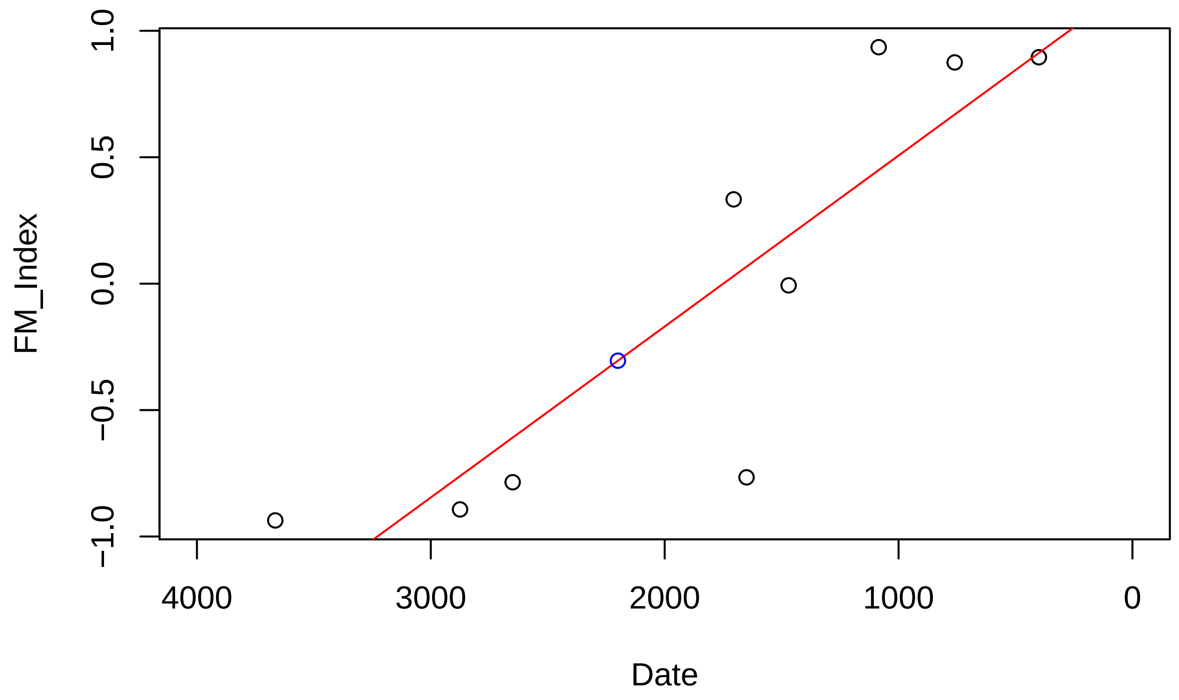
We now see quite clearly that there is a positive relationship between Date and the FM Index, with the index increasing through time as we move towards the present.

And just to illustrate that this model fit is calculated the same way that we calculated the FM index for 2200 BP, let's plot our result on this figure.

```
plot(FM_Index ~ Date, data = sac.data, xlim = c(4000, 0)) #plot the data

abline(sac.mod, col = "red") #add the model fit

points(x = 2200, y = FMIndex_2200BP, col = "blue") #add the 2200 BP value
```



The value that we calculated is simply taken from this model fit! When the regression line here is calculated, R is simply calculating the FM Index for every value of Date and using those values to draw this red line.

We should also plot a confidence interval around our model fit. Remember that a model is made up of two parts: a deterministic part and a stochastic part. What we plotted so far is just the deterministic part. Let's include a visual representation of the stochastic part, as we should.

We can do this using the `predict()` function.

The `predict()` function requires you to provide a model object, which will be `sac.mod`. It also requires an argument called `newdata`, which takes a data frame which contains a column with a range of values from the predictor variable. The `predict()` function is going to use this range of predictor values (i.e. Date) to predict the values of the response variable (i.e. FM Index) based on the model object (i.e. `sac.mod`). This data frame that you provide to the `newdata` argument must contain a column *named identically to the original predictor variable.* Failing to name this data frame appropriately is a common source of error messages. This means that our `newdata` object will be a data frame with a column named `Date` which will contain a sequence of values spanning the range of dates in our dataset.

So let's make this data frame over which we will predict values of FM Index.

```r
new.x <- data.frame("Date" = seq(4000, 0, length.out = 100))
```

The `seq()` function allows us to create a sequence of values from 4000 to 0, in this case. And I've also used the `length.out` argument to specify that we want 100 values between 4000 and 0.

Now let's plug this `new.x` object into the `newdata` argument of the `predict()` function, and then use the `sac.mod` object to predict a confidence interval for our model fit.

```r
pred.y <- predict(sac.mod, newdata = new.x, interval = "confidence", level = 0.95)
```

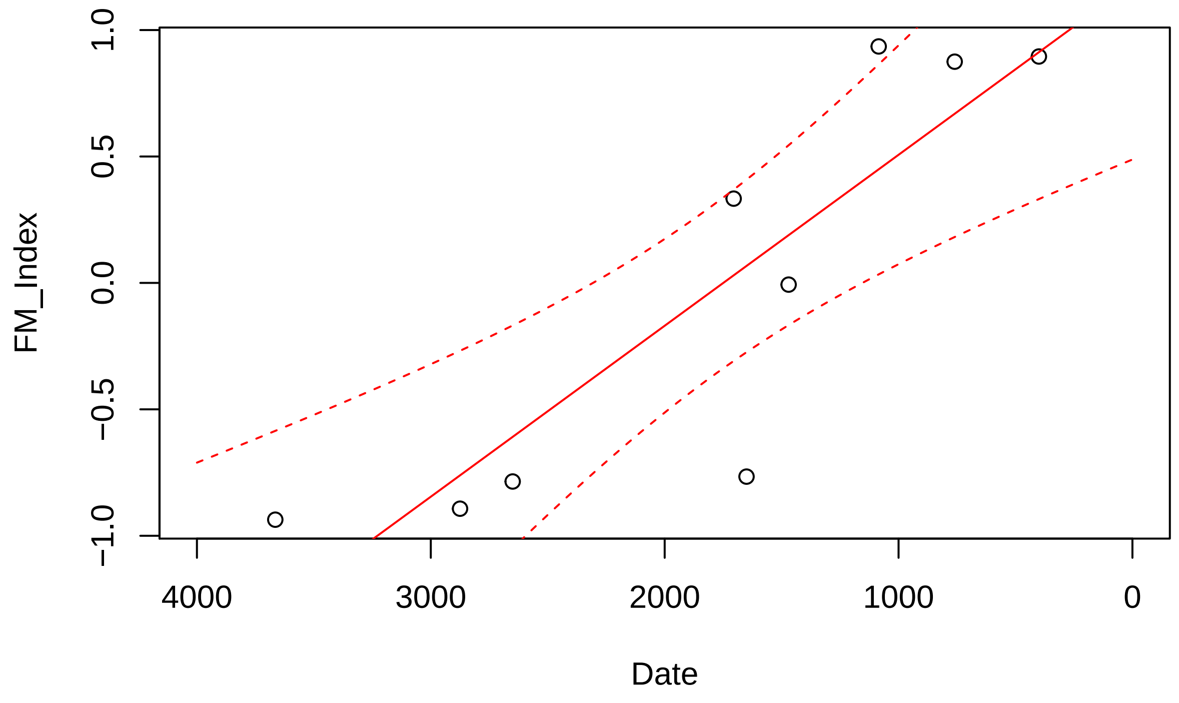
We now have an object called `pred.y` that contains the model fit, the lower confidence interval, and the upper confidence interval. You can inspect the structure of this object to see for yourself, and note that these confidence interval values are stored in the 2nd and 3rd columns of this object.

So now let's use the `lines()` function to plot these lower and upper confidence intervals. Remember that they are stored in the second and third columns of `pred.y` so we'll need to index the `pred.y` object to get them out.

```r
plot(FM_Index ~ Date, data = sac.data, xlim = c(4000, 0)) #plot the data

abline(sac.mod, col = "red") #add the model fit

lines(pred.y[, 2] ~ new.x$Date, lty = 2, col = "red") #add the CIs
lines(pred.y[, 3] ~ new.x$Date, lty = 2, col = "red") #add the CIs
```



Now we have a plot of our model fit (the deterministic part of our model) along with our 95% confidence intervals (the stochastic part of our model). Looking good!

1.5 Reporting model results

Now that we've got a plotted model and understand our coefficients, let's further explore our model output.

```r
summary(sac.mod)
```

```
##
## Call:
## lm(formula = FM_Index ~ Date, data = sac.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.83325 -0.17650 -0.01691  0.30332  0.48598
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.1827966  0.2938171   4.026  0.00502 **
## Date        -0.0006761  0.0001422  -4.755  0.00207 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4278 on 7 degrees of freedom
## Multiple R-squared:  0.7636, Adjusted R-squared:  0.7298
## F-statistic: 22.61 on 1 and 7 DF,  p-value: 0.002072
```

Just as in prior examples, our model coefficients have associated error estimates of 0.2938 and 0.0001. Each of those then has a t value associated with it that is used to calculate a p-value in the Pr(>|t|) column. However, again, this is not the model's p-value! This is simply an expression of whether the coefficient estimate is significantly different from zero. We should really be looking for the p-value reported in the very last row of this `summary` output. In this particular case it's the same as the p-value for the Date coefficient, but this won't always be so.

Another important aspect of this model is the r-squared value. An r^2 value is a measure of what is known as goodness of fit. Essentially, an r^2 value represents the proportion of the variance in the response variable that is explained by including the predictor variable in our model. So here, it's the proportion of variance in FM Index that is explained by Date. In our `summary` output, we can see our r^2 value in next to the words "Multiple R-squared" and see that it is 0.7636.

You'll also see that there is an "Adjusted R-squred" value reported as well. This is only used when you have multiple predictor variables, not just one like we do. For most purposes, you'll use the "Multiple R-squared".

An r^2 value of 0.0 to 0.3 is often considered a very weak, or no effect. Values from 0.3 to 0.5 are considered weak, values from 0.5 to 0.7 are considered moderately strong, and values from 0.7 to 1.0 are considered strong.

When you report the results of this regression analysis, you would say something like the following:

The relationship between date and the FM index is strongly negative and significant, with the FM index increasing towards the present ($r^2 = 0.76$, F(1, 7) = 22.61, $p < 0.01$).

Note: Remember that as far as R is concerned, this is a negative relationship because Date runs from 0 to 4000, not from 4000 to 0 as we've plotted it in reverse. If we wanted to actually have this be a positive relationship in the model, as we're visualizing it, we'd need to make our Date values negative so that our axis is really running from -4000 to 0, and then re-create our model.

You could also say things like this:

- Date explains 76% of the variance in FM Index (r^2 = 0.76)
- Date has a negative effect on FM Index (β = -0.00067)