LAB 6

STAT 131A

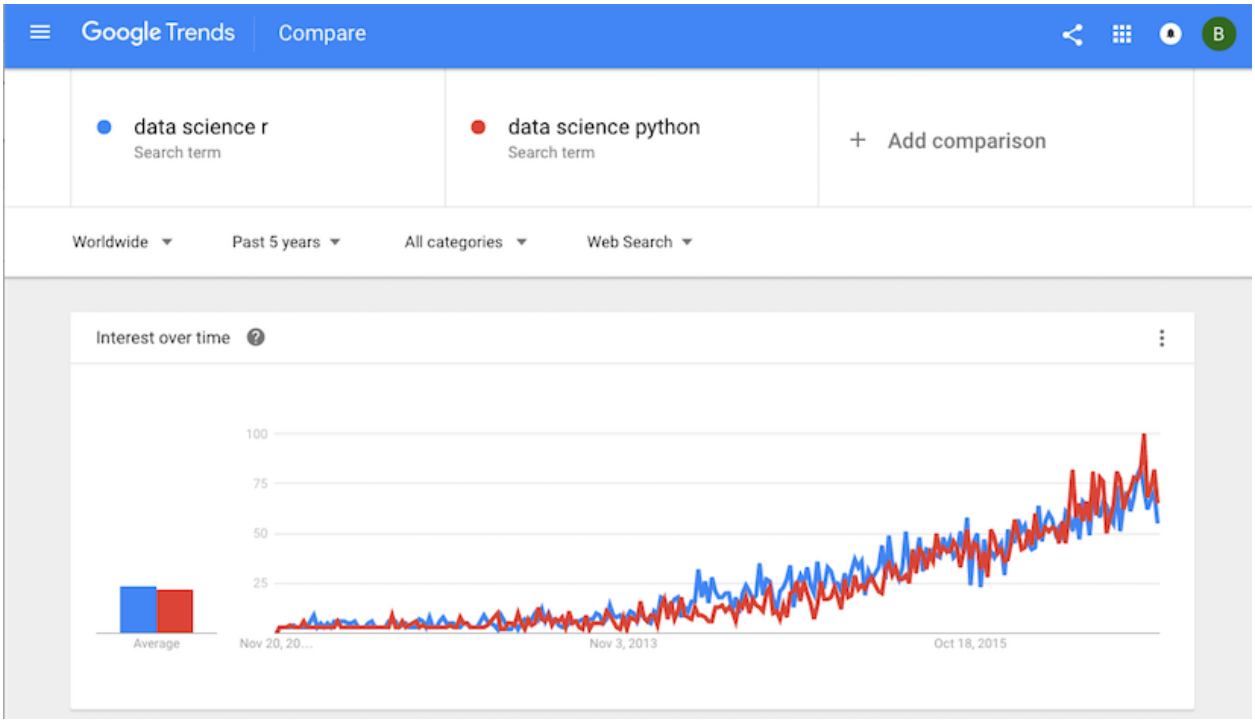
Oct 2, 2021

Welcome to LAB 6!

Linear models

We will continue working with the Google Trend dataset for R and Python. Recall the dataset we obtained contains the following variables:

- **week**: beginning date of the week (recent 5 years)
- **python**: trend of the search term **Data science Python**
- **r**: trend of the search term **Data science r**



Read the data.

```
data_science <- read.csv("data_science.csv")
# convert string to date object
data_science$week <- as.Date(data_science$week, "%Y-%m-%d")
# create a numeric column representing the time
data_science$time <- as.numeric(data_science$week)
data_science$time <- data_science$time - data_science$time[1] + 1
```

The plot in the Google Trend page looks somewhat linear. So we will try linear model first. Note that in the `lm` function for the model $y = \beta_0 + \beta_1 x + e$, you don't need to add the intercept term explicitly. Fitting the model with an intercept term is the default when you pass the formula as `y ~ x`. If you would like to fit a model without an intercept($y = \beta_1 x + e$,), you need the formula `y ~ x - 1`.

```
python.lm <- lm(python ~ time, data = data_science)
r.lm <- lm(r ~ time, data = data_science)
```

```
summary(r.lm)
```

```
##
## Call:
## lm(formula = r ~ time, data = data_science)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.348  -7.076  -0.648   5.873  40.142
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -10.034179   1.116414  -8.988   <2e-16 ***
## time          0.038423   0.001065  36.089   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.019 on 258 degrees of freedom
## Multiple R-squared:  0.8347, Adjusted R-squared:  0.834
## F-statistic:  1302 on 1 and 258 DF,  p-value: < 2.2e-16
```

Exercise 1.

Read the output of `lm`.

(a) Which of the folowing formula will you use to predict the search index of `data science r` at time `t`.

A. -10.034179 + 0.038423 t

B. 1.116414 + 0.001065 t

C. -10.034179 + 1.116414 t

D. 0.038423 + 0.001065 t

(b) Calculate the t-statistics for the intercept and the slope using the coefficient estimates and standard deviations. (Please copy and paste the numbers you need from the output of `summary` function.) Are your results consistent with the ones given in the summary?

```
# Insert your code here and save the t-statistic for the intercept
tstat_intercept = -10.034179/1.116414
tstat_intercept
```

```
## [1] -8.987866
```

```
# Insert your code here and save the t-statistic for the slope
tstat_slope = 0.038423/0.001065
tstat_slope
```

```
## [1] 36.07793
```

(c) Calculate the p-value for the intercept and the slope using the test statistics you get in (b). What are the null hypothesis and your conclusion? Please use the normal distribution to approximate the distribution of test statistics under the null. Are your results consistent with the ones given in the summary?

```
# Insert your code here and save the p-value for the intercept
pval_intercept = 2*pnorm(-abs(tstat_intercept))
pval_intercept
```

```
## [1] 2.52078e-19
```

```
# Insert your code here and save the t-statistic for the slope
pval_slope = 2*pnorm(-abs(tstat_slope))
pval_slope
```

```
## [1] 5.032119e-285
```

(d) Construct the confidence interval for the intercept and the slope using the coefficient estimates and standard deviations. (Please copy and paste the numbers you need from the output of `summary` function.) Will you accept or reject the null given the confidence interval you calculated?

```
# Insert your code here and save the confidence interval for the intercept
ci_intercept = c(-10.034179-1.96*1.116414, -10.034179+1.96*1.116414)
ci_intercept
```

```
## [1] -12.222350  -7.846008
```

```
# Insert your code here and save the confidence interval for the slope
ci_slope = c((0.038423-(1.96*0.001065)), (0.038423+(1.96*0.001065)))
ci_slope
```

```
## [1] 0.0363356 0.0405104
```

No, I will reject it since 0 is out of the CI.

More on plots in R

Function `plot`

The function `plot` is the most standard plotting function in R. When taking only one vector, it plots the vector against the index vector.

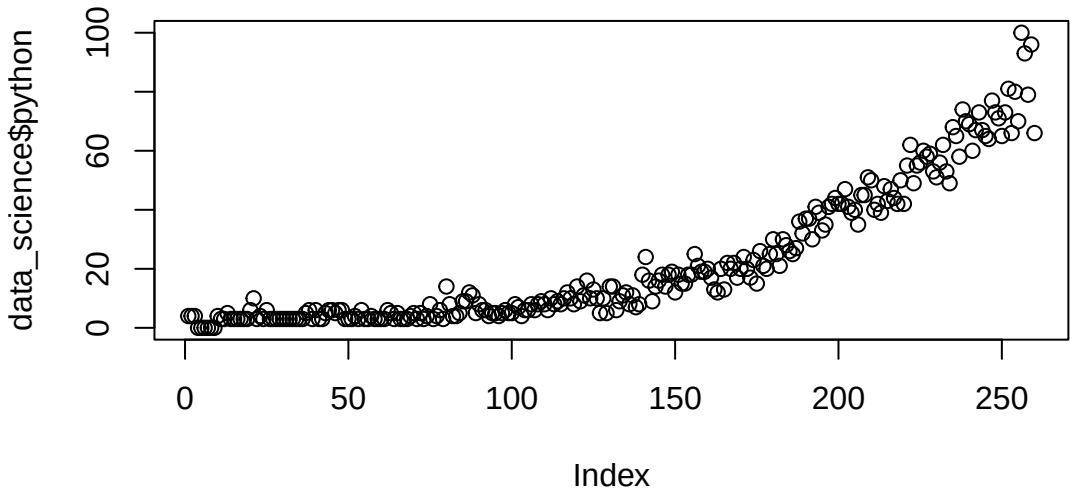
```
plot(data_science$python)
library(dplyr)
```

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

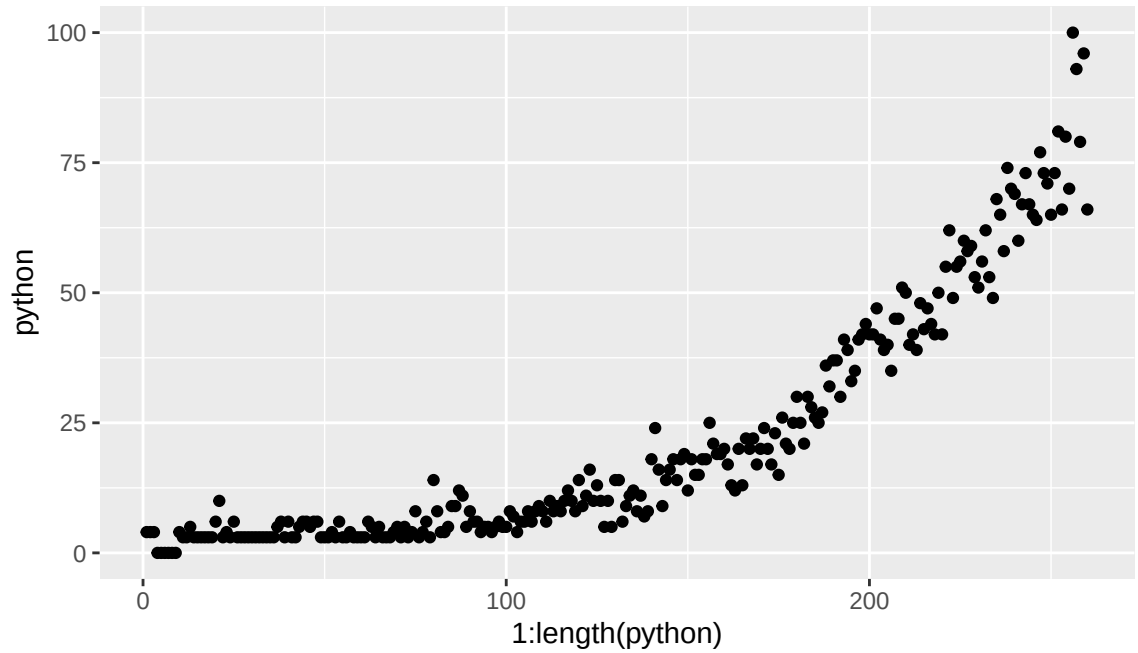
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

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

```
library(ggplot2)
```

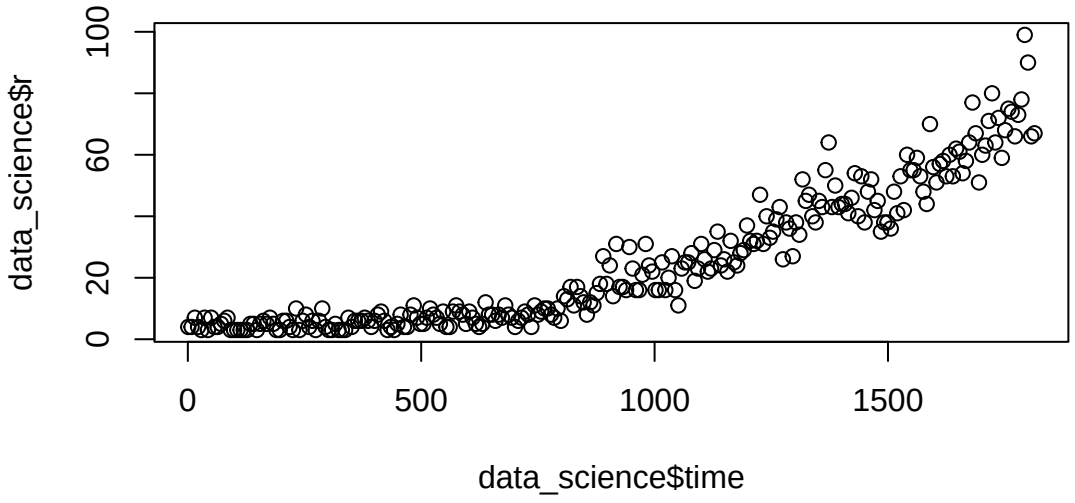



```
#tidyverse
data_science %>% ggplot()+geom_point(aes(x=1:length(python), y=python))
```



When taking two vectors, it plots the scatter plot.

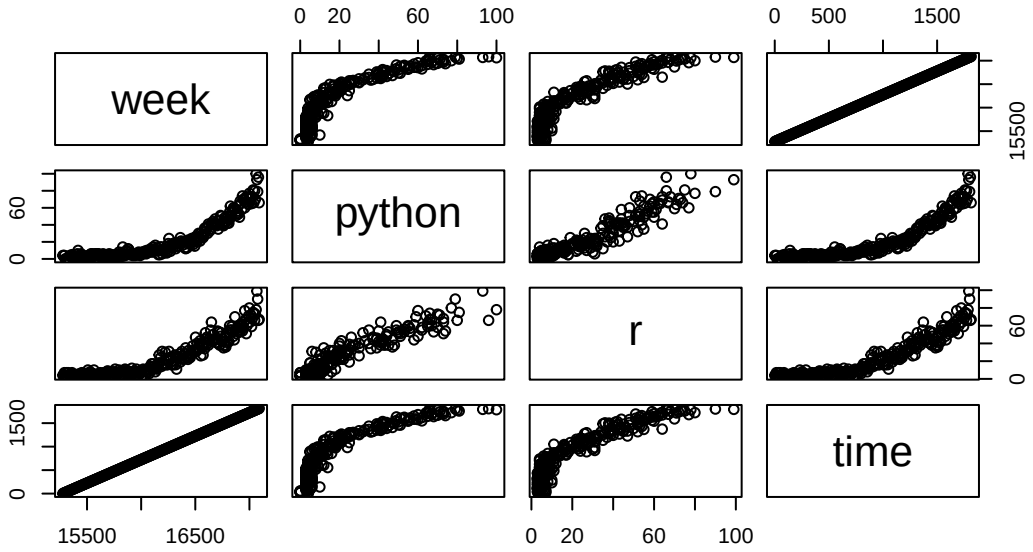
```
plot(data_science$time, data_science$r)
```




```
#tidyverse
#data_science %>% ggplot()+geom_point(aes(x=time, y=r))
```

The `plot` function can also accept a data frame as the parameter. And it plots all variables against each other.

```
plot(data_science)
```



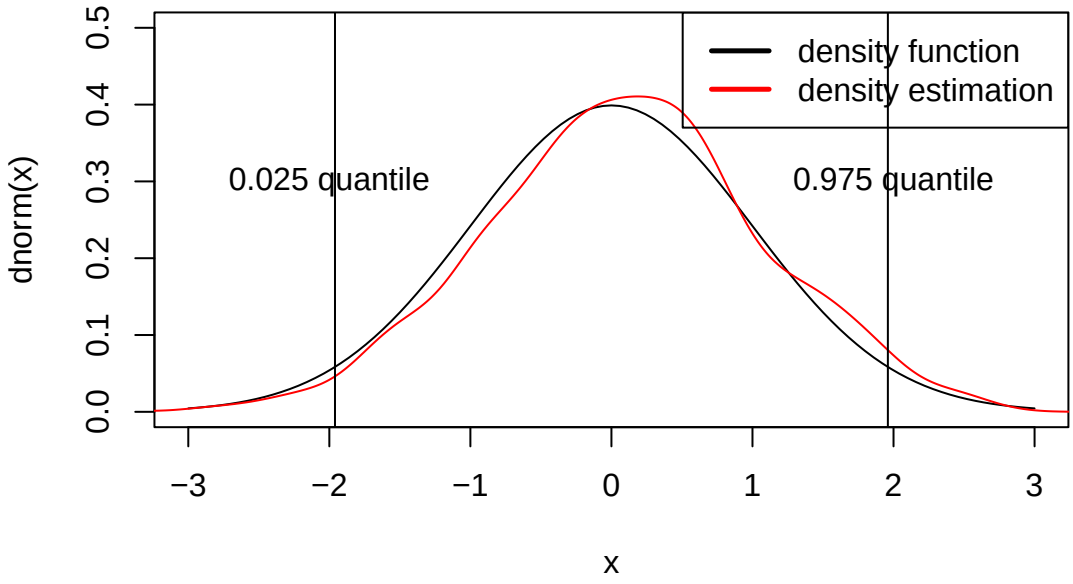
Low-level graphics functions

The `plot` function introduced above is a high-level plot function, which produces complete plots. There are also low-level functions that add further outputs to an existing plot. You have already seen some high-level functions such as `hist`, `boxplot` and `curve`. `lines` is a low-level function which can not be called directly.

Commenly used low-level functions includes:

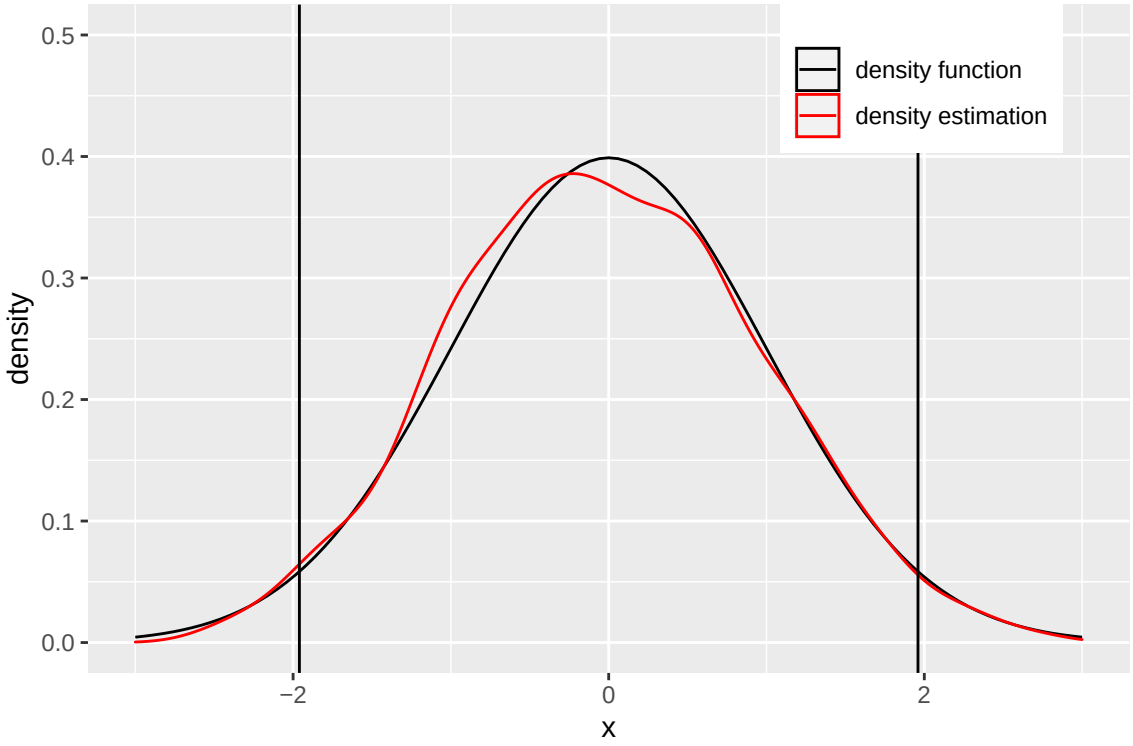
- `points(x, y)`: Adds points to the current plot. `x` and `y` are vectors of coordinates.
- `lines(x, y)`: Add line to the current plot by joining the points with line segments.
- `text(x, y, labels, ...)`: Add text to a plot at points given by x, y.
- `abline(a, b)`: Adds a line of slope b and intercept a to the current plot.
- `abline(h=y)`: Adds a horizontal line.
- `abline(v=x)`: Adds a vertical line.
- `legend(x, y, legend, ...)`: Add legends to plots

```
curve(dnorm, xlim = c(-3, 3), ylim = c(0, 0.5))  # create a plot for the density of normal distribution
lines(density(rnorm(1000)), col = "red") # add a line (kernel density estimation) to the plot created
abline(v=qnorm(0.025)) # add a vertical line (0.025 quanitle of standrad normal distribution)
abline(v=qnorm(0.975)) # add a vertical line (0.975 quanitle of standrad normal distribution)
text(-2, 0.3, "0.025 quantile") # add text
text(2, 0.3, "0.975 quantile") # add text
legend("topright", c("density function", "density estimation"),
       lwd=c(2.5,2.5), # Line widths in the legend
       col=c("black", "red")) # Add legend
```




```
#tidyverse
ggplot(data = data.frame(data = rnorm(1000)))+
        stat_function(fun = dnorm, mapping=aes(color = 'density function'))+
        geom_density(aes(x=data, color = 'density estimation'))+
        geom_vline(xintercept=qnorm(0.025))+
        geom_vline(xintercept=qnorm(0.975))+
        scale_colour_manual("",
                            breaks = c("density function", "density estimation"),
                            values = c("black", "red"))+
        xlab("x")+
        xlim(-3,3)+
        ylim(0,0.5)+
        theme(legend.position = c(0.8, 0.9))
```

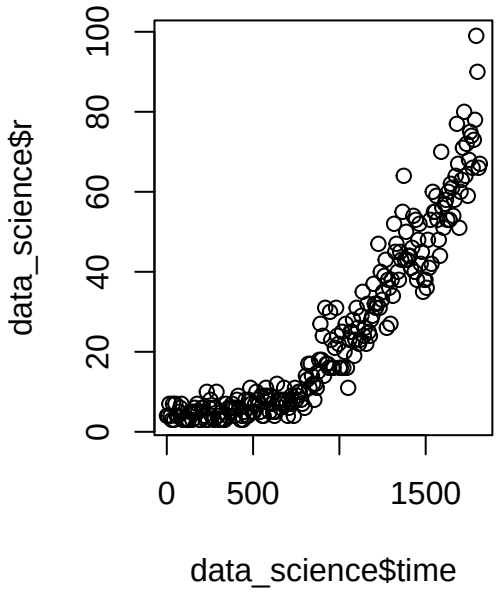
```
## Warning: Removed 2 rows containing non-finite values (stat_density).
```

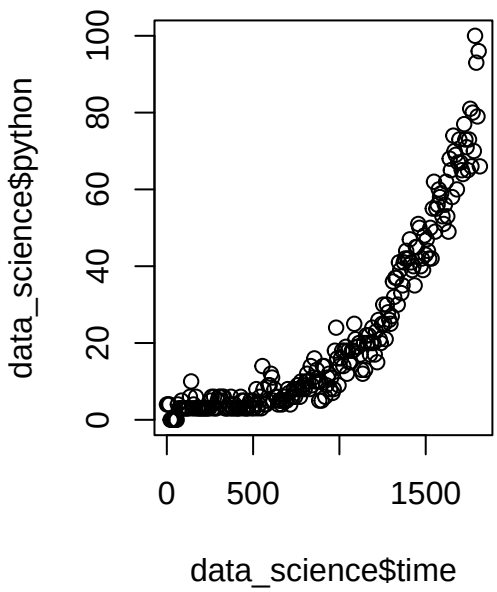


par()

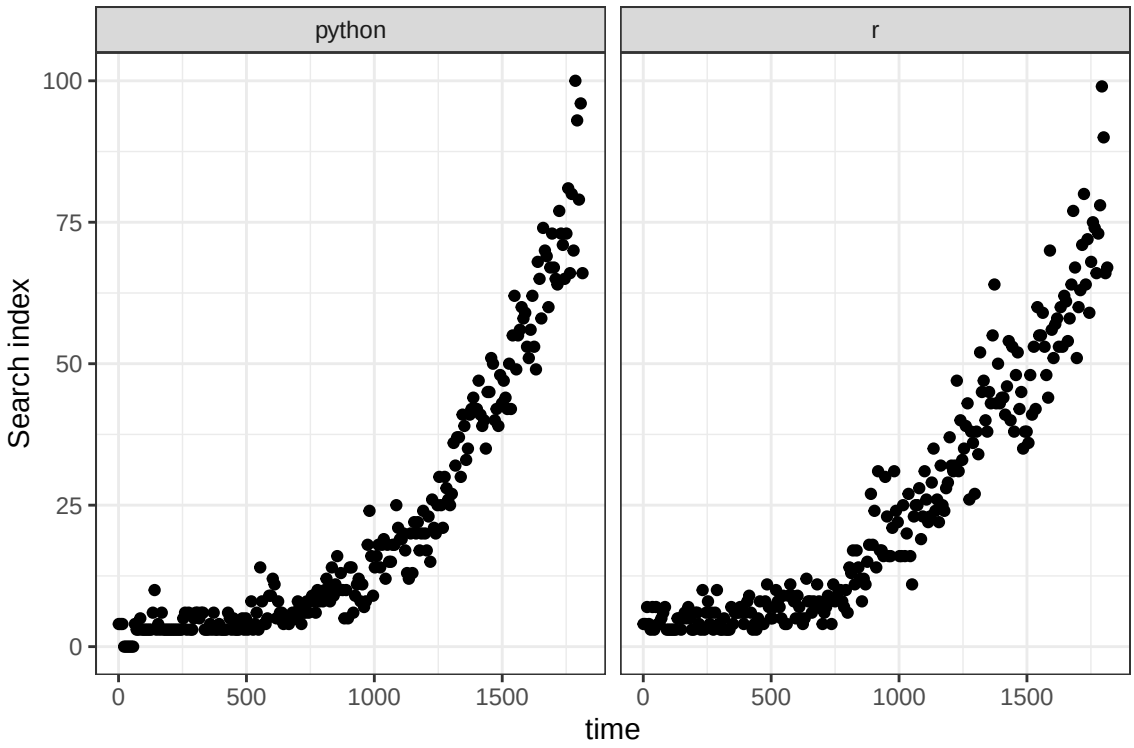
The par() function is used to access and modify the list of graphics parameters. For example, to put several plots in the same window, use `par(mfrow = c(a, b))`. `a` is the number of rows and `b` is the number of columns. This command will allow you to plot `a*b` plots in one window.

```
library(tidyr)
par(mfrow=c(1, 2))
plot(data_science$time, data_science$r)
plot(data_science$time, data_science$python)
```






```r
#tidyverse
data_science %>% pivot_longer(
    cols = c(r, python),
    names_to = "Language",
    values_to = "Search index"
) %>% ggplot(aes(x = time, y = `Search index`)) +
    geom_point() + facet_wrap(. ~ Language) +
    theme_bw()
```


python
r
Search index
100
75
50
25
0
0
500
1000
1500
time

Exercise 2.

Functions and plots for the lm fit.

(a) Get the coefficients of the linear models fit on `data science r` search index using function `coef`. (This is equivalent to the dollar sign plus "coefficients")

```
# Insert your code here and save the coefficients vector
coef(r.lm)
```

```
##  (Intercept)         time
## -10.03417888   0.03842291
```

(b) Get the confidence intervals of the linear models fit on `data science r` search index using function `confint`.

```
# Insert your code here and save the confidence intervals
confint(r.lm,level = 0.95)
```

```
##                    2.5 %      97.5 %
## (Intercept) -12.23262190 -7.83573586
## time          0.03632639  0.04051944
```

(c) Get the prediction of `data science r` search index on "2013-01-06" (time point 435) and "2015-01-04"(time point 1163), using function `predict`. What is the absolute error of the prediction?

HINT: absolute error = |prediction - true value|

```
# Insert your code here and save the coefficients vector and prediction error
predict1<- predict(r.lm, data.frame(time=435))
predict2<- predict(r.lm, data.frame(time=1163))
true1 <- data_science$r[data_science$time == 435]
true2 <- data_science$r[data_science$time == 1163]
abs1 <- abs(predict1-true1)
abs2 <- abs(predict2-true2)
abs1
```

```
##        1
## 2.679789
```

```
abs2
```

```
##       1
## 2.65167
```

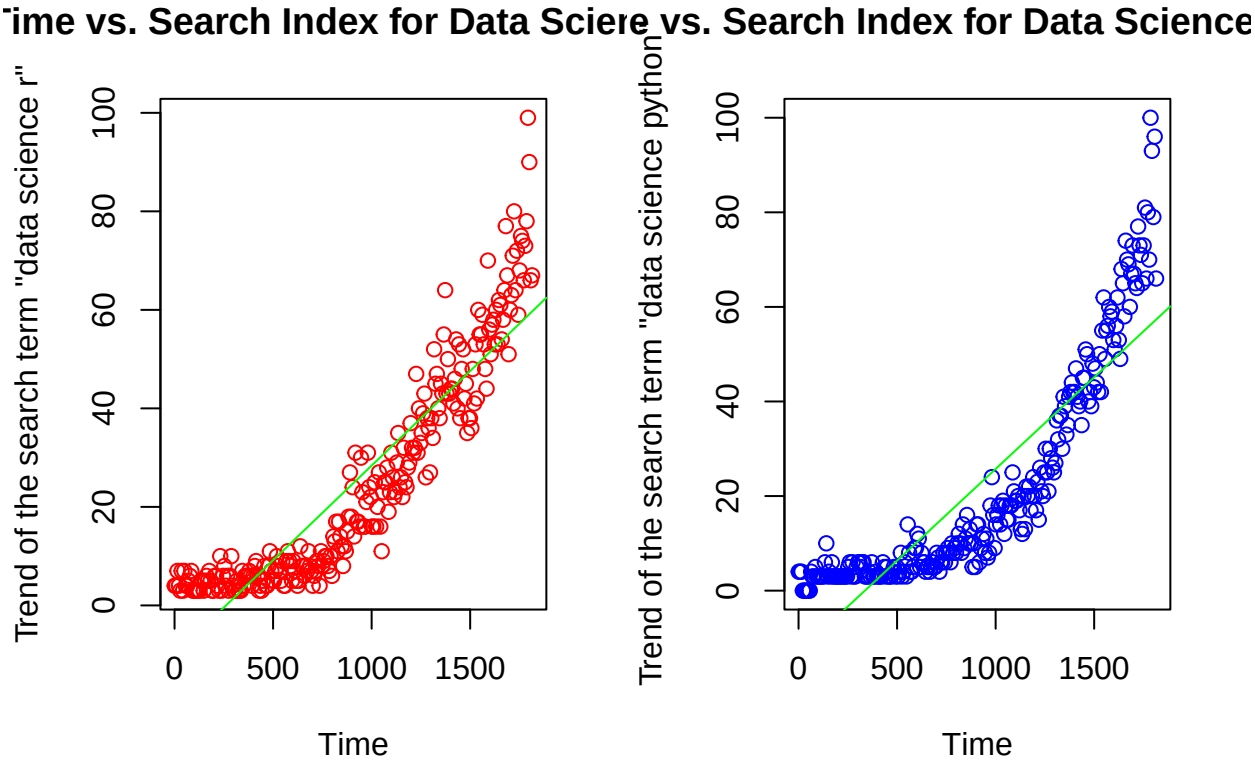
(d) Get the prediction intervals ofdata science rsearch index on "2013-01-06" (time point 435) and"2015-01-04"(time point 1163), using functionpredict. Do your perdiction intevals contains the actual values?

```
# Insert your code here and save the prediction interval as
#`pred.int.2013`
pred.int.2013 <- predict(r.lm, data.frame(time=435), intervel = "prediction")
# Insert your code here and save the prediction interval as
#`pred.int.2015`
pred.int.2015 <- predict(r.lm, data.frame(time=1163), intervel = "prediction")
```

(e) Plot the fittings. Plot the scatter plot of time versus the search index for `data science r` and `data science python`. Color the points with red and blue. Plot the fitted linear line with color.

```
# Insert your code here
par(mfrow=c(1,2))
plot(data_science$r~data_science$time, col = 'red', main='Time vs. Search Index for Data Science R', xl
abline(r.lm, col='green')
```

```
plot(data_science$python~data_science$time, col = 'blue', main='Time vs. Search Index for Data Science
abline(python.lm, col='green')
```



(Question Extra for Experts) (Not required) You might notice that in thedata_sciencedata frame, thereis another column calledweek, which gives the actual dates instead of the time point integers. The dateinformation is saved in another data type calledDate. When fitting linear models, we can not use theDatetype. However, the intergers are not informative for plotting. Can you figure out a way to plot the dates instead of integers for the x axis in (e)?

```
# Insert your code here
```

Polynomial models

Exercise 3. As you may discover, the linear model is not quite good for your dataset. In this exercise, you will fit a cubic curve to the data (use time to predict `r` and `python`). You may want to refer to the code in Page 23 and 24 in Lecture 3. (old lab, not sure if still these pages)

(a) Fit the cubic model.

```
# Insert your code here and save your fitted model as
# `python.poly` and `r.poly`
python.poly <- lm(
  python ~ time + I(time^2) + I(time^2) , data = data_science
)
 summary(python.poly)
```

```
##
```

```
## Call:
## lm(formula = python ~ time + I(time^2) + I(time^2), data = data_science)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.8714  -2.7568   0.5277   2.5325  24.1898
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.586e+00  8.911e-01   9.636   <2e-16 ***
## time        -3.294e-02  2.268e-03 -14.525   <2e-16 ***
## I(time^2)    3.952e-05  1.210e-06  32.669   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.816 on 257 degrees of freedom
## Multiple R-squared:  0.957,  Adjusted R-squared:  0.9567
## F-statistic:  2861 on 2 and 257 DF,  p-value: < 2.2e-16
```

```
r.poly <- lm(
  python ~ time + I(time^2) + I(time^2) , data = data_science
)
summary(r.poly)
```

```
##
## Call:
## lm(formula = python ~ time + I(time^2) + I(time^2), data = data_science)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.8714  -2.7568   0.5277   2.5325  24.1898
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.586e+00  8.911e-01   9.636   <2e-16 ***
## time        -3.294e-02  2.268e-03 -14.525   <2e-16 ***
## I(time^2)    3.952e-05  1.210e-06  32.669   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.816 on 257 degrees of freedom
## Multiple R-squared:  0.957,  Adjusted R-squared:  0.9567
## F-statistic:  2861 on 2 and 257 DF,  p-value: < 2.2e-16
```

(b) In the fitting for `r` and `python` search index, which of the following term is significant (not equal to zero)?

```
# Uncomment the line of your answer for this question: (r)
intercept.r.sig <- TRUE
time.r.sig <- TRUE
time.r.square.sig <- TRUE
# time.r.cubic.sig <- TRUE
```

```
# Uncomment the line of your answer for this question: (python)
intercept.python.sig <- TRUE
# time.python.sig <- TRUE
```

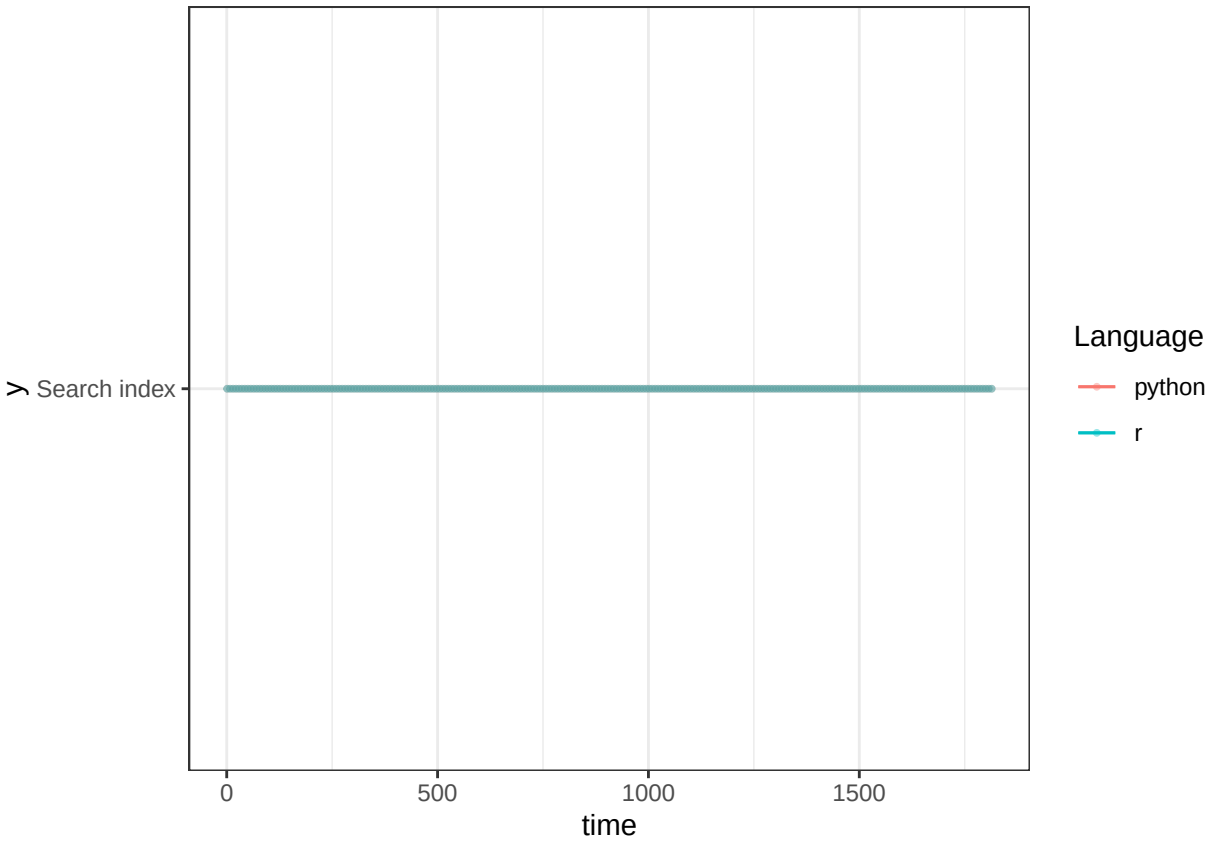
```
time.python.square.sig <- TRUE
time.python.cubic.sig <- TRUE
```

(c) Plot the scatter plot and the fitted line. Color the groups with red and blue. How will you describe the trend of r search and python search?

```
# Insert your code here for plotting
data_science %>% pivot_longer(
  cols = c(python, r),
  names_to = "Language",
  values_to = "Search index"
) %>% ggplot(aes(x=time,
                 y = 'Search index',
                 color = Language))+
  geom_point(size=0.7, alpha=0.4)+
  geom_function(aes(color="python"),
                fun = function(x){
                  coef(python.poly)[1]+
                  coef(python.poly)[2]*x+
                  coef(python.poly)[3]*x^2+
                  coef(python.poly)[4]*x^3
                })+
  geom_function(aes(color="r"),
                fun = function(x){
                  coef(r)[1]+
                  coef(r)[2]*x+
                  coef(r)[3]*x^2+
                  coef(r)[4]*x^3
                })+
  theme_bw()+
  theme(legend.position = "right")
```

```
## Warning: Computation failed in `stat_function()`:
## object 'r' not found
```

```
## Warning: Removed 101 row(s) containing missing values (geom_path).
```


Search index
0
500
1000
1500
time
Language
python
r

Reading - Caveat

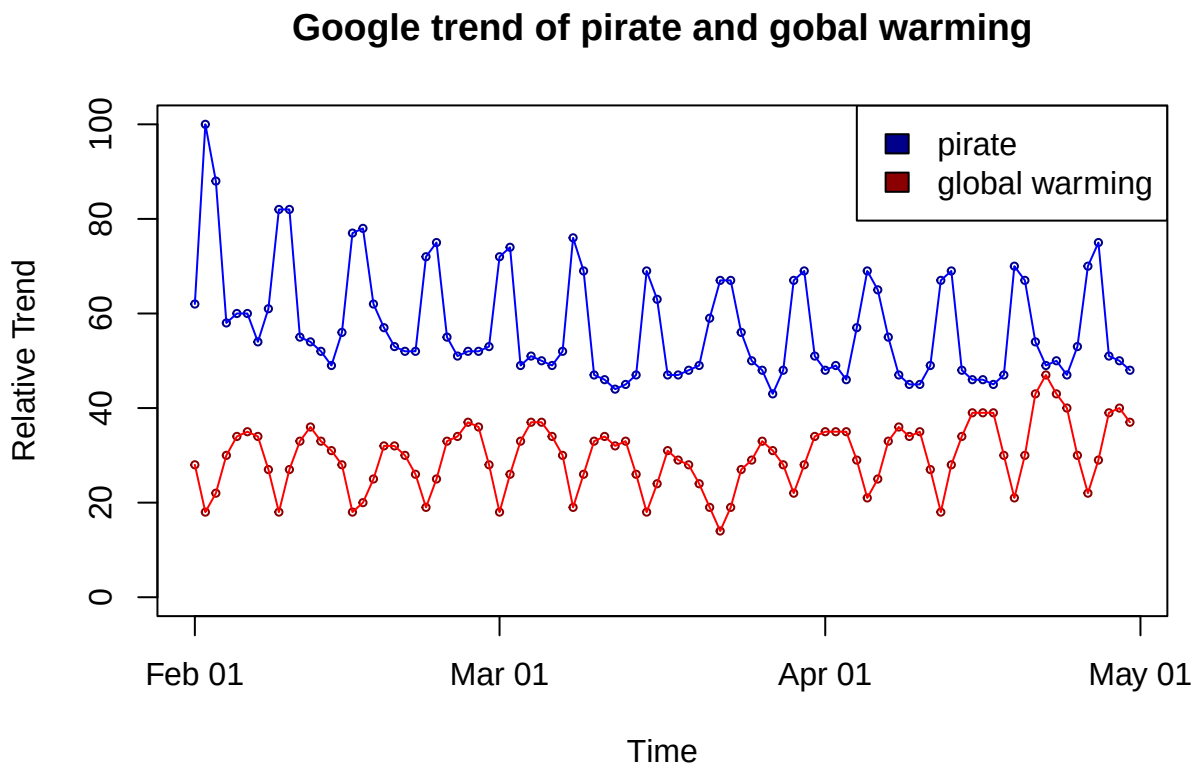
Google trend seems so powerful and accessible. However, when analyzing some topics, we would rather not to use it. Why? Consider the following two examples.

Global Warming & Pirates

Back to 2008, there is an obvious negative correlation between global warming and pirate searching, which may just be a coincidence. If you fit a linear model, the coefficients are significant.

```
pirate <- read.csv("pirate.csv")
pirate$Day <- as.Date(pirate$Day, "%Y-%m-%d")
data_science$time <- as.numeric(data_science$week)
data_science$time <- as.numeric(data_science$week) - as.numeric(data_science$week)[1]
```

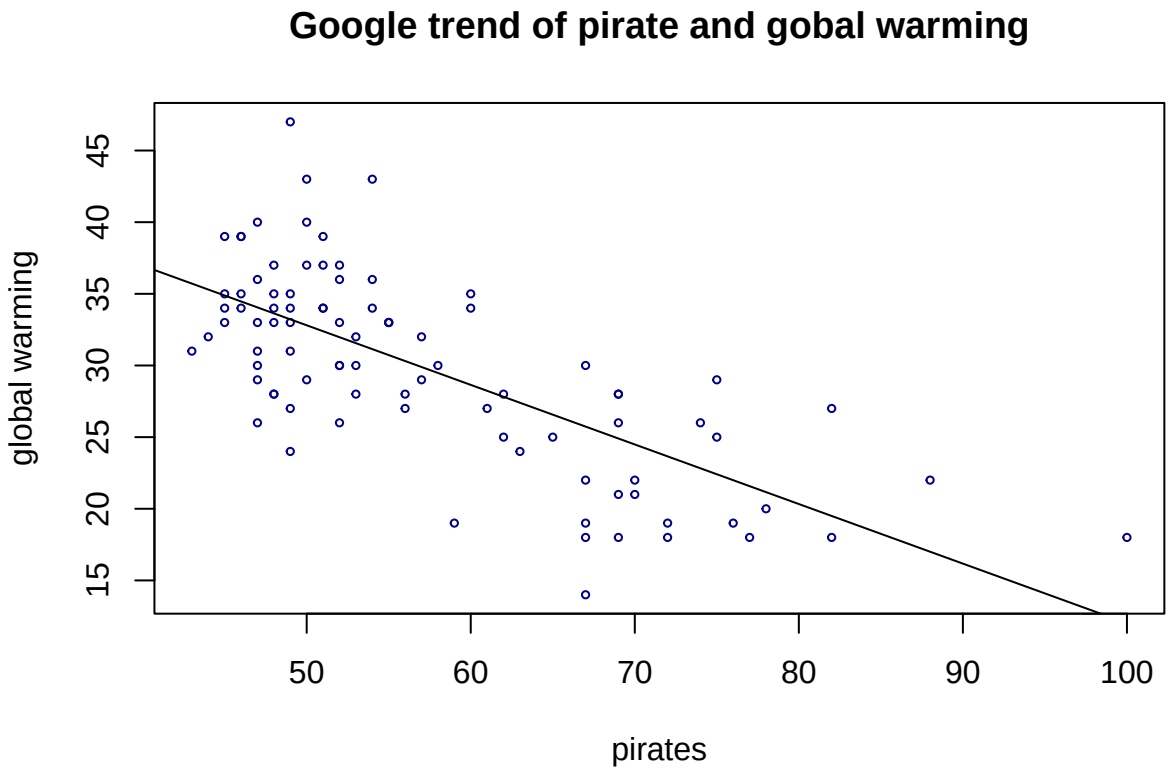
```
plot(pirate$Day, pirate$pirates, cex = 0.5, col = "blue4",
     xlab = "Time", ylab = "Relative Trend", ylim = c(0, 100),
     main = "Google trend of pirate and gobal warming")
points(pirate$Day, pirate$global.warming, cex = 0.5, col = "red4")
lines(pirate$Day, pirate$pirates, col = "blue")
lines(pirate$Day, pirate$global.warming, col = "red")
legend("topright",c("pirate", "global warming"), fill=c("blue4","red4"))
```




```
pirate.lm <- lm(global.warming ~ pirates, data = pirate)

plot(pirate$pirates, pirate$global.warming, cex = 0.5, col = "blue4",
     xlab = "pirates", ylab = "global warming",
     main = "Google trend of pirate and gobal warming")
```

```
abline(pirate.lm)
```




```
cor(pirate$pirates, pirate$global.warming)
```

```
## [1] -0.7097726
```

```
summary(pirate.lm)
```

```
##
## Call:
## lm(formula = global.warming ~ pirates, data = pirate)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.7361  -3.3770  -0.0522   3.1548  13.7794
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 53.59517    2.57293  20.830  < 2e-16 ***
## pirates     -0.41581    0.04399  -9.452 4.81e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.824 on 88 degrees of freedom
## Multiple R-squared:  0.5038, Adjusted R-squared:  0.4981
## F-statistic: 89.34 on 1 and 88 DF,  p-value: 4.814e-15
```

The Failure of Google Flu Trend

Google Flu Trend was first launched in 2008 (The Google research paper: Detecting influenza epidemics using search engine query data). It was widely recognized an exciting event in the big data application In the 2009 flu pandemic, Google Flu Trends tracked information about flu in the United States. In February 2010, the CDC (the U.S. Centers for Disease Control and Prevention) identified influenza cases spiking in the mid-Atlantic region of the United States. However, Google's data of search queries about flu symptoms was able to show that same spike two weeks prior to the CDC report being released.

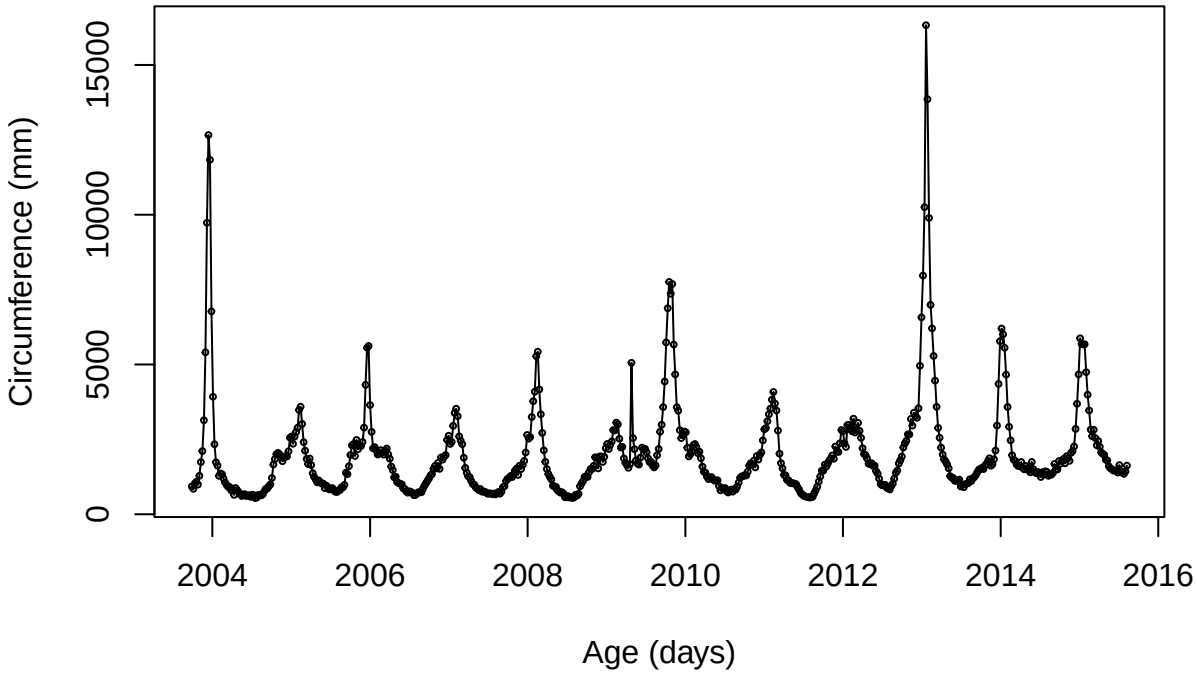
The model was initially based on the flu data from 2003-2008. Google Flu Trend prediction performs well at the beginning. However, it's been wrong since August 2011. The subsequent report continuously overestimates the flu prevalence. And now Google Flu Trends is no longer publishing current estimates of Flu based on search patterns.

In a Science article, a team of researchers described Google Flu Trend as "Big Data Hubris":

"The core challenge is that most big data that have received popular attention are not the output of instruments designed to produce valid and reliable data amenable for scientific analysis."

```
flu <- read.csv("flu.csv", stringsAsFactors = FALSE)
flu$Date <- as.Date(flu$Date)
```

```
plot(flu$Date, flu$California,
     type = "o", cex = 0.4,
     xlab="Age (days)",
     ylab="Circumference (mm)" )
```



Reference and further reading:

- The Parable of Google Flu: Traps in Big Data Analysis

- Google Flu Trends' Failure Shows Good Data > Big Data
- Google Flu Trend