Python for biologists

Tutorial 6 - Statistical models

by Tobias Andermann

In the previous tutorials we have learned the basics of the numpy and pandas libraries, as well as some basic plotting with matplotlib. Now we will cover some basics of how to do statistics in Python. We will mostly use the scipy package, which has many useful functions for statistics.

Installation

Make sure to install scipy through your bash command line.

```
pip install scipy
```

We'll need all of the following packages in this tutorial, best is to load them all now at once.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy
from scipy import stats
```

Linear regression

Load the data in the file `data/co2_temp.txt` stored in this workshops GitHub folder using the pandas library.

In [21]:
```
```

Out[21]:

	year	co2	temperature
0	1880	289.469999	13.81
1	1881	289.736999	13.89
2	1882	290.018999	13.89
3	1883	290.262999	13.80
4	1884	290.511999	13.71

Store the values of the `year` , `co2` , and `temperature` columns in separate numpy arrays.

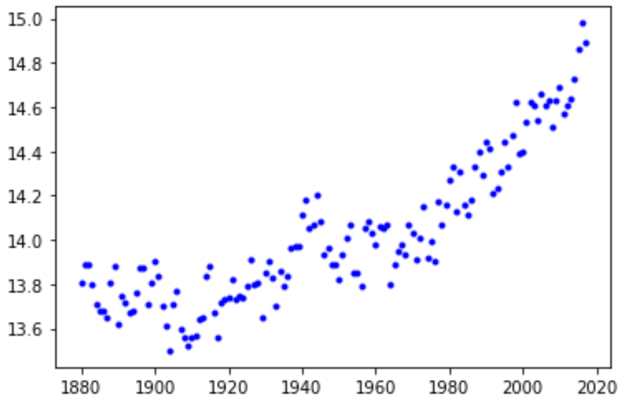
In [ ]:
```
year = ???
temp = ???
co2 = ???
```

In the following we will use a linear regression model to see if there is a linear trend in temperature through time. But first, we just plot the temperature data to get an idea of what the data looks like.

Tip: You can change the marker style in the plot function. By default it connects all point with a line, but there are other ways you can plot the data, which you can specify with the `marker=` setting in the plot command. For example choose `marker='.'`. See overview of available styles [here (https://matplotlib.org/api/markers_api.html#module-matplotlib.markers)](https://matplotlib.org/api/markers_api.html#module-matplotlib.markers). Additionally in order to turn off lines you can choose `linestyle=''`.

In [90]:
```
plt.plot(year,temp,marker='.',linestyle='',color='blue',label='Data')
```

Out[90]:
```
[<matplotlib.lines.Line2D at 0x1a17144780>]
```



Now we will fit a linear regression model to these data, using the `stats.linregress()` function. The function returns the slope and intercept of the best determined regression. It further returns the r-value (measure of how strong the linear correlation of the tested variables is) and p-value (probability that slope equals 0).

In [103]:
```
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(year,temp)
```

To view the fitted linear regression, we can predict temperature data under the given slope and intercept and plot it together with the actual data. The most elegant way of doing this is to write a short function that applies the linear formula `y = slope * x + intercept` to a given array of x values together with a provided slope and intercept, to calculate the predicted temperature. Try to complete the function by replacing the `???` with the appropiate formula.

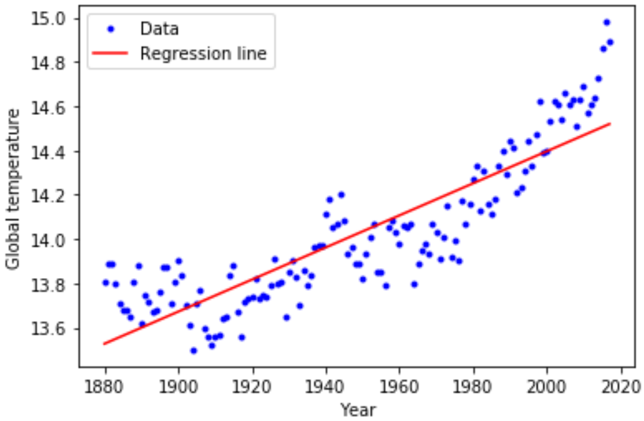
In [98]:
```
def linear_function(x,slope,intercept):
    y = ???
    return(y)
```

Now you can apply it to the `year` array to calculate the y values predicted by our regression function:

In [99]:
```
predicted_y = linear_function(year,slope,intercept)
```

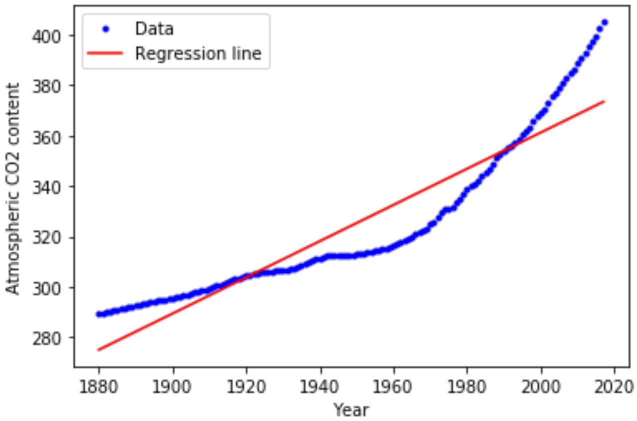
TASK: Plot the actual data in blue and the regression line in red in the same plot.

In [100]:
```
plt.plot(year,temp,marker='.',linestyle='',color='blue',label='Data')
plt.plot(year,predicted_y,color='red',label='Regression line')
plt.xlabel('Year')
plt.ylabel('Global temperature')
plt.legend();
```



TASK: Run a linear regression model for the CO2 data in the same way as we did for global temperature. Plot the data and the regression line in the same manner as above.

In [82]:



Instead of looking at the temperature and CO2 arrays separately through time, we can also use a linear regression model to test if the two are correlated with each other (testing this climate change stuff everyone is talking about).

In [220]:
```
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(co2,temp)
```

The R value represents the correlation coefficient. This shows how strong the linear relationship is between the two variables temperature and CO2 content. An R value of more than 0.8 signifies a high positive correlation. What do we get here?

In [223]:
```
r_value
```

Out[223]:
```
0.9397825066375002
```

The p-value is also interesting as it shows the probability of the data if the null hyothesis was true. The null hypothesis is that the slope equals 0, i.e. that there is no correlation whatsoever. What do we get here?

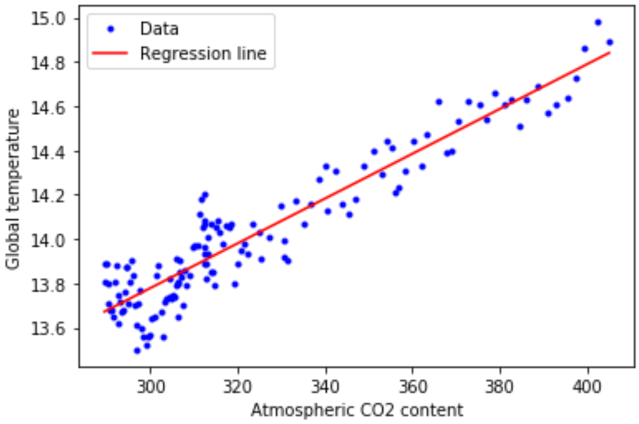
In [224]:
```
p_value
```

Out[224]: 2.813701898143606e-65

You can cite this p-value next time you are discussing with someone who claims that CO2 does not affect the global temperature. The p value we determined is the probability of them being right. However, to be fair, our analysis can not determine what is cause and what is effect, maybe there is another variable that causes CO2 and temperatures to increase equally, even though the two don't influence each other. But in this case there is also plenty of evidence outlining the physical reasons why CO2 in the atmosphere is increasing global temperature, so you are probably on the safe side claiming that CO2 is the cause and temperature the effect.

TASK: Plot the data and the regression line for CO2 against temperature.

In [225]:



Fitting other models

Particularly for the CO2 data you can see that a linear model is maybe not the most suitable, but that the CO2 content is increasing at increasing rates. Let's try to fit a power function instead. To fit any desired function to the data, we can use the `curve_fit()` function of scipy.

Let us first define the power function in the same way as we did above for the linear function. This function has two parameters that are called a and b (the linear function has two parameters as well which we called slope and intercept).

In [186]:
```
def power_function(x,a,b):
    y = float(a)*x**float(b)
    return y
```

Before fitting the power function it is important that we rescale our input data, i.e. they should start at the value 0. Think for a moment how you can accomplish that, given what you know about array math operations.

In [193]:
```
year_adjusted = ???
co2_adjusted = ???
```

After rescaling the two arrays, and after having defined the function to optimize, we can now fit the power function to the data using the `curve_fit()` function. The output of this function looks a bit different than that of the `linregress()` function we used earlier. This function outputs two lists, the ifrst of which contains our model parameters:

In [199]:
```
parameters, covariance = scipy.optimize.curve_fit(power_function,year_adjusted,co2
parameters
```

Out[199]:
```
array([1.34618012e-03, 2.29628011e+00])
```

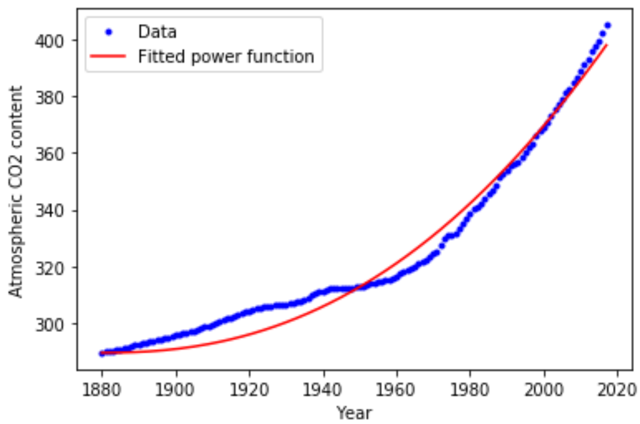
Assign the model parameters to the variables a and b:

In [201]:
```
a,b = parameters
print(a,b)
```

```
0.0013461801228692922 2.2962801086451843
```

TASK: Now plot the new fitted power function on top of the the data. First produce an array of predicted y-values for the target years, just as we did for the linear function. Preferably also reverse the rescaling of the x and y values, so the final plot contains actual years and CO2 values (not the rescaled values that we needed to fit the power function). The final plot should look like the one you see below.

In [209]:



Our model enables us e.g. to predict the expected CO2 content for future years.

TASK: Use the fitted power function to calculate the predicted value for the year 2050?

You are ready, young padawan!

If you want to dive deeper into doing statistics in Python, there are many great online tutorials. With the basic knowledge covered in the tutorials of this course, you should be able to get started exploring the Python world by yourself. You can for example try [this statistics tutorial here (https://scipy-lectures.org/packages/statistics/index.html)](https://scipy-lectures.org/packages/statistics/index.html).