**Explanation third question:**

Best future price for non fossil energy

First step:

Ofcourse we import all necesseary modules for the last code. Yes i import allot of codes, but that is a choice of your own. For me you create more possibilities and you learn how to work with different types of modules.

Second step:

Here i also import a single csv file, use .set\_index() “Entity” and replace this column. Also use .drop() in the same cel to skip column “Code” from our Dataframe.

Third step:

Using .drop\_duplicates() i look if there are duplicates and skip them aswell if inplace= True.

Fourth step:

The .dropna() method removes the rows that contains NULL values and returns a new Dataframe object unless the parameter is set to True! In that case the .dropna() method does the removing in the original Dataframe instead.

Fifth step:

After i remove the NULL values i check the columns to see what we have. Maybe i want to rename them, so let’s use .columns.

Sixth step:

Yes, i decide to rename the columns by using the .rename() function. Now i changed the names into “year”, “geothermal”, “seawindpower”, “pv-celpower”, “solarpower”, “hydropower”, “landwindpower”.

Sixth step:

I also use .corr() function to find the pairwise correlation of all columns in our Dataframe.

Seventh step:

Here we use a single plot and start with column “biopower”. I use the .scatter() function, in the function i use (x=”year”, y=”biopower” and color= “red”).

Eight step:

Making the dataset fort he linear regression calculation, i change the variable into dbB.

Nineth step:

Here i use the .apply() function and inside the function i use .coerce and if it’s good i twill returns a Tuple consisting the two numeric arguments converted to a common type.

Nineth step:

You can choose to fill those values with some default, so let’s do that by using .fillna() function.

Tenth step:

I have to create the regression line and in this case i use .fit() function and inside this function i use the variable df\_year and dfB.

Eleventh step:

We need a prediction, so let’s predict by using .predict() function with variable df\_year inside.

Twelveth step:

Here we visualize the results of this column by plotting this. Here you see the regression line is quite straight and the lines of this power resource show that this is the case.

Thirteenth step:

Like the previous step i repeated the same steps and run it.

Fourteenth step:

Here i go put all the subplots together and show all the regression lines. By using .subplot() i can put all the 7 .regplots() under the subplot function. Inside this funtion i put df\_year variable(dfB), regressor.predict(df\_year), color = ….)

Plt.title(“graph regression lines cost natural energy over a period of ten years”)

Plt.xlabel(“YEAR”)

Plt.ylabel(“regression lines”)

And plt.legend()

After that you see a very nice regression lines plot including the spots and colors.

Fifteenth step:

Here i make a text with the explanation of the beautifull plot i make in step fourteenth.

Sixteenth step:

Here i use .concat() function to merge columns from the variable df\_energy\_names. If you want to concatenation to ignore existing indices, you can set the argument ignore\_index= True. Then, the resulting Dataframe index will be labeled with 0,…, n-1. But i choose False!

Seventeeth step:

Let’s checkout data. For this we use the .describe() function, this method returns description of the data in the Dataframe. If the Dataframe contains numerical data, the description contains these information for each column: count – The number of not-empty values. mean – The average (mean) value.

Eighteenth step:

I import some more modules for the last few cels.

Nineteeth step:

Let’s check this Dataframe again .columns how it looks now, and it’s still like we want it to be.

Twenteenth step:

Train\_test\_split helps us to split the data into a training and a test dataset. For this we use the fit\_transform() function. This fit method is calculating the mean and variance of each of the features present in our data.

Twenty-first step:

Let’s try the LinearRegression() again.

And it is a LinearRegression if we run this, that is what we want to see so we continue the steps.

Twenty-second step:

Now we are going to see if our model performs on previously unseen data. Here we use .predict() and .sqrt() function, this function returns the square root of a number. Note: The number must be greater than or equal to 0!

Twenty-third step:

And if you print lin\_reg\_model.coef\_, then you see the linear regression model’s values for B1 and B2.

Twenty-fourth step:

Here i use .intercept\_ function, this allows you to intercept any function call in Python and handle it in any manner you choose. For example, you can pre-process the inputs to a function, or apply post-processing on its output. Intercepts also allows you to completely replace a function with a custom implementation.

Twenty\_fifth step:

If we now print poly\_reg\_model.coef\_ we’ll get the values for five coefficients (*ß1, ß2, ß3, ß4, ß5).*

Twenty-sixth step:

Let’s go back to comparing our model’s performances by printing lin\_reg\_rmse.