Method for Generating Trendlines

Goal: Generate fake logarithmic trendlines for chosen states for use in the study.

Overview: The general process is fairly straightforward. Essentially, to generate both flatter and steeper trendlines, I use Python's random module to generate a series of weights, which I then use to interpolate new values for the given data **post-lockdown**. The primary difference here is that I fit to a linear curve for the flatter trendlines, but to an exponential curve for the steeper trendlines.

Detailed Walkthrough of Trendline Generation for New York

First, I clean up and format the data for our specific state. Below is a screenshot of an example dataframe for the state of New York.

[169]:

	Province_State	Confirmed	Date	Day	New_Cases	image_url	color
0	New York	173.0	2020-03-10	0	0.0		before
1	New York	220.0	2020-03-11	1	47.0		before
2	New York	328.0	2020-03-12	2	108.0		before
3	New York	421.0	2020-03-13	3	93.0		before
4	New York	525.0	2020-03-14	4	104.0		before
5	New York	732.0	2020-03-15	5	207.0		before
6	New York	967.0	2020-03-16	6	235.0		before
7	New York	1706.0	2020-03-17	7	739.0		before
8	New York	2495.0	2020-03-18	8	789.0		before
9	New York	5365.0	2020-03-19	9	2870.0		before
10	New York	8310.0	2020-03-20	10	2945.0		before
11	New York	11710.0	2020-03-21	11	3400.0		before
12	New York	15800.0	2020-03-22	12	4090.0	https://raw.githubusercontent.com/Murtz5253/co...	before
13	New York	20884.0	2020-03-23	13	5084.0		after
14	New York	25681.0	2020-03-24	14	4797.0		after
15	New York	30841.0	2020-03-25	15	5160.0		after
16	New York	37877.0	2020-03-26	16	7036.0		after
17	New York	44876.0	2020-03-27	17	6999.0		after
18	New York	52410.0	2020-03-28	18	7534.0		after
19	New York	59648.0	2020-03-29	19	7238.0		after

Now, let us look at the generation of the less steep trendlines. Below is the code:

```
seq = np.arange(.3, 1.0, .1)
weights = [random.choice(seq) for i in range(5)] # First the smaller slopes
weights.sort() # Ensures that charts are generates in correct order
print("Less steep trendlines:")
counter = 1
for val in weights:
    test = df.copy()
    original = df["Confirmed"].values[inflection_day:]
    step = original[1] - original[0]
    new_step = step * val
    # y = m(x - x1) + y1
    f = lambda x : new_step * (x - inflection_day) + original[0]
    updated = [f(x) for x in range(inflection_day, test.shape[0])]
    test.loc[inflection_day:, "Confirmed"] = np.random.normal(updated,  scale=500)
    test["Type"] = "less_steep_" + str(counter)
    trendlines.append(test)
    base = create_base_log_layer(test, 'Day', 'Confirmed')
    img = create_img_layer(test, 'Day', 'Confirmed', 'image_url')
    display(base + img)
```

I start by generating several random weights and sorting them for clarity. Then, I look at the step between the first two data points, and I multiply it by each weight to obtain a new step. I define a linear function using this new step as the slope, and I generate new y-values using this function for data points **after** the inflection day (date of the lockdown). I then display the corresponding trendlines using Altair.

Next, we look at the code for generating steeper trendlines:

```
seq = np.arange(1, 3, .1) # Range of exponents
weights = [random.choice(seq) for i in range(5)] # Now the larger, exponential slopes
weights.sort()
print("Steeper trendlines:")
counter = 1
def func(x, adj1,adj2):
    return ((x+adj1) ** pw) * adj2
seq = np.arange(.3, 1.0, .1)
for val in weights:
    test = df.copy()
    original = df["Confirmed"].values[inflection_day:]
    print(original)
    x = [inflection_day, inflection_day + 1]
    y = [original[0], original[1]]
    pw = val # the weight is the exponent this time
    A = np.exp(np.log(y[0]/y[1])/pw)
    a = (x[0] - x[1]*A)/(A-1)
    b = y[0]/(x[0]+a)**pw

    xf = np.arange(inflection_day, df.shape[0])
    updated = func(xf, a, b)
    print(updated)
    test.loc[inflection_day:, "Confirmed"] = np.random.normal(updated, scale=70)
    test["Type"] = "steeper_" + str(counter)
    trendlines.append(test)
    base = create_base_log_layer(test, 'Day', 'Confirmed')
    img = create_img_layer(test, 'Day', 'Confirmed', 'image_url')
    display(base + img)
```

Note that while previously I used weights strictly less than 1 (to get flatter trendlines), we now use weights that range from 1 to 3. The process we use is very similar to what we did above, except this time we fit to an exponential curve, using our weight as the exponent. The reason I did not use a linear fit for the steeper trendlines is that attempting to change the slope resulted in very jagged, unrealistic curves. The code for generating the exponential trendlines is taken from the following link, which also explains the method in more detail: https://stackoverflow.com/questions/33186740/fitting-exponential-function-through-two-data-points-with-scipy-curve-fit

For each new trendline I generate, I save its corresponding data into a dataframe. I then concatenate all these dataframes together into one and export the final dataframe as a CSV file. This CSV file is subsequently read into the Streamlit file that generates the main study, with each trendline bound to one option available for selection. Crucial here is the fact that we only generate the random trendlines **once**, so all users will receive the same random trendlines for each state.