UDACITY

< Return to Classroom

Communicate Data Findings

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations

You have met all of the requirements for this project! This is an exceptional submission *(one of the best that I have seen)*. Your data wrangling, to generate some very well chosen plots, is impressive. And your attention to detail is especially impressive. Very nice work!!

For data analysts, one of the most difficult parts of the job is explaining the implications of their studies to non-statisticians. With your choice of visualizations, and your data wrangling to generate those visualizations, you have highlighted that you have the skill to do this with ease. It is a skill that translates to any *data exploration*. You should look forward with confidence to applying what you have learned here, to any of the interesting data analyses that you will face in your career.

*(I have made some suggestions below, if you are going to use this report as evidence of your abilities in **data exploration**, you should make these changes where relevant. These suggestions don't detract from your work, they are just included so that you can add the finishing touches to your excellent report).*

Congrats again and best wishes for your next project (or your next endeavor, if this is your last project)!!

Code Quality

All code is functional (i.e. no errors are thrown by the code). Warnings are okay, as long as they are not a result of poor coding practices.

The code in both of your notebooks evaluate as expected (without errors). Nice work!

NOTE

- To help develop your skills using python, I highly recommend working through the examples in this free text

The project uses functions and loops where possible to reduce repetitive code. Comments and docstrings are used as needed to document code functionality.

- pandas is used for most data wrangling tasks, so vectorized operations rather than loops are used. Nice work.
- The comments in your code, and in your Markdown that relate to the code, makes it easy to follow your code. Very nice work!
- Excellent work using a function to avoid repetitive coding *(* `set_continent_dtype()` *, e.g)*

NOTES

- In data analysis, in a work environment, commenting code, so that your colleagues understand the intent of the code that you write, is a basic necessity.
    - *(It is always more difficult to read other people's code).*
    - Commenting code is a good habit to develop, because it communicates to colleagues, or to yourself at some future time, the intent of the code that you have written (in a succinct way that avoids you having to examine the code line by line). It is difficult to overstate the importance of clear code commenting, in a work environment, in data analysis.

Again, nice work!!

- If you want to explore functions in python, this free online text/tutorial will help you to develop this area of your coding skills

Exploratory Data Analysis

The project appropriately uses univariate, bivariate, and multivariate plots to explore many relationships in the data set. Reasoning is used to justify the flow of the exploration.

Your plot selection is excellent (as are your aesthetic choices for those plots). Also, your data wrangling to generate those plots is exceptional (which helps capture patterns that would otherwise have not been so evident).

You have a wide range of visualizations to choose from for your slide deck/show. Very nice work!

Questions and observations are placed regularly throughout the report, after each plot or set of related plots.

Your commentary makes it easy to follow your thought processes as you work through your data exploration. It is clear, concise and engaging. The questions that you answer throughout your report uncovers interesting patterns in the data. Excellent work!

Visualizations made in the project depict the data in an appropriate manner that allows plots to be readily interpreted. This includes choice of appropriate plot type, data encodings, transformations, and labels as needed.

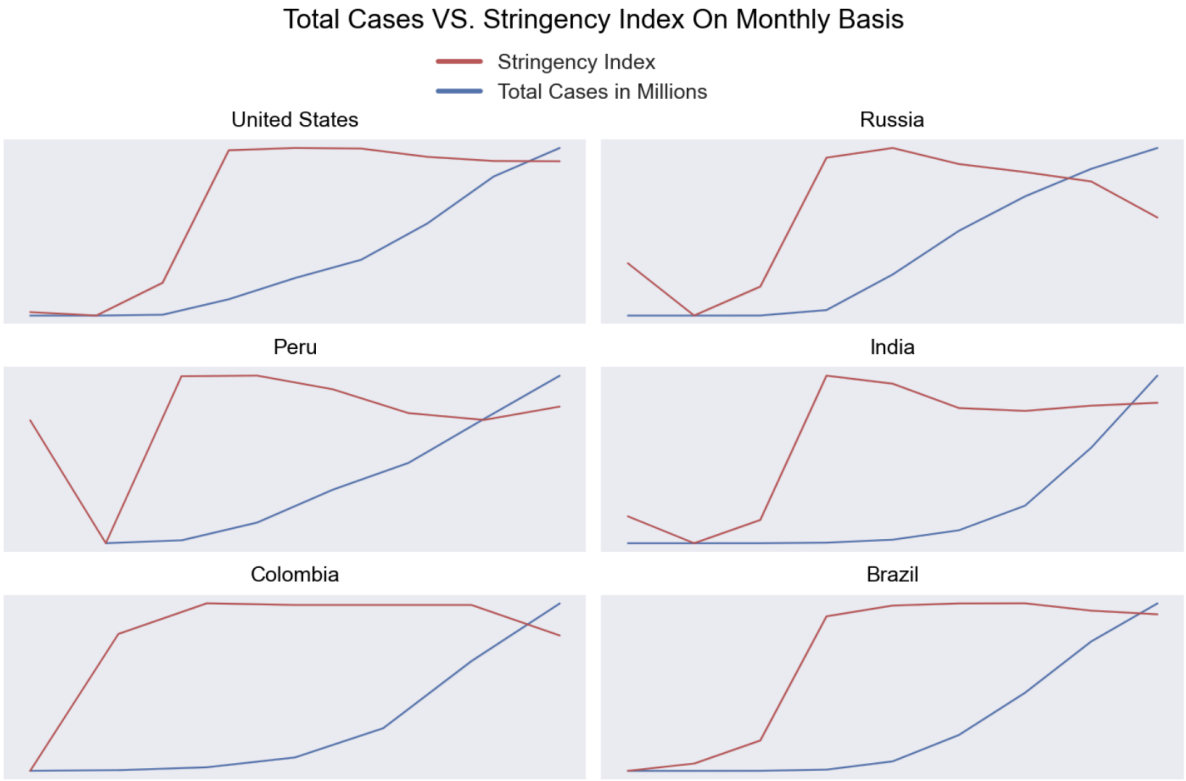
As noted above, you include a nice selection of univariate, bivariate, and multivariate plots. Also, your aesthetic choices for those plots make them easy to interpret. Very nice work!

TIPS

1. FACE COLOR

On re-running your code, the face color on some of your plots changed (which ruins your carefully chosen aesthetic) - this could be due to a different default on different OSs. You can easily set them, to avoid this issue, as explained in this post: set facecolor.

For example:



Here I used:

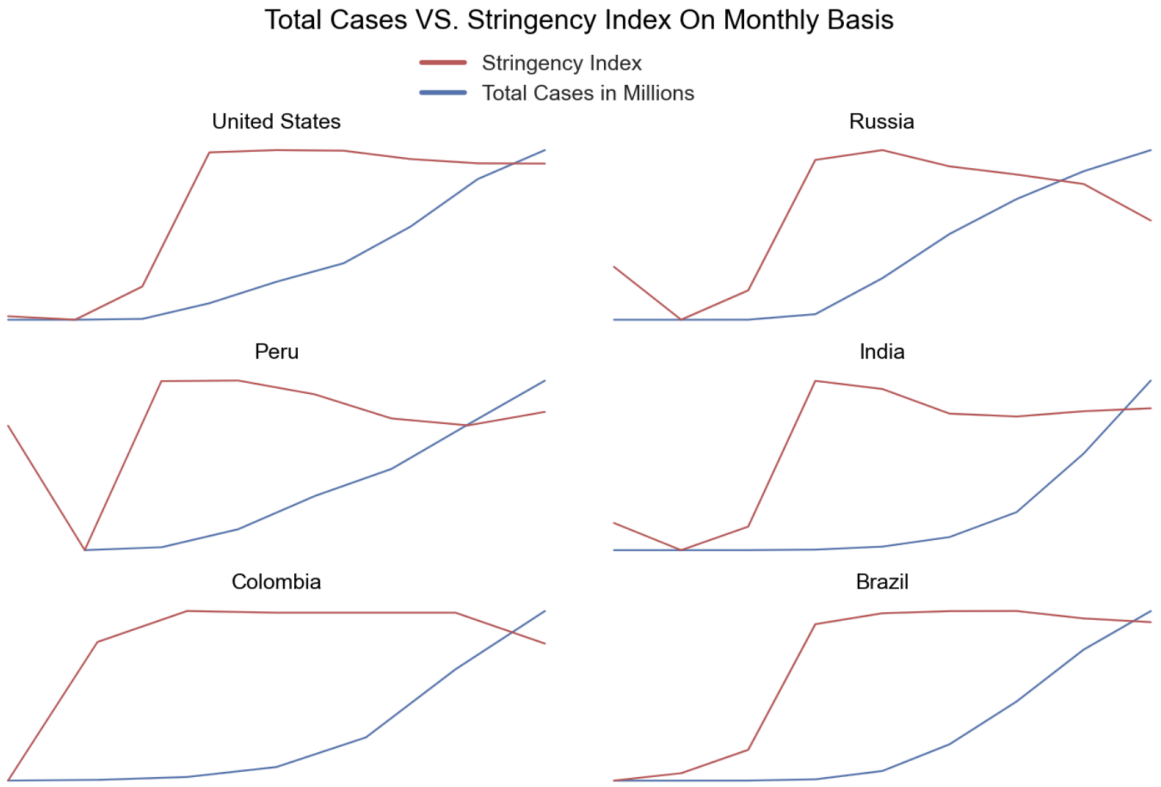
```
ax.set_facecolor('xkcd:white')
```

```
    ax.set_facecolor('xkcd:white')

    sb.despine(left=True, bottom=True) # Remove plot borders

plt.tight_layout() # Improve appearance

plt.savefig('src/cases_vs_string.png', transparent=True, bbox_inches='tight', pad_inches=1);
```



SCATTERPLOT: DISCRETE DATA

Typically it is not a good idea to use scatterplots with discrete data.

```
tot_cases_cont_no_0 = tot_cases_cont_no_0[tot_cases_cont_no_0['date'] > '2019-12-31']

g = sb.FacetGrid(data = tot_cases_cont_no_0, row = 'continent', height=4, aspect=2.5) # Create FacetGrid class
g.map(plt.scatter, 'month', 'new_cases', alpha=1/2); # add x and y axes
g.fig.set_size_inches(14.70, 8.27);
```

This is because the points on the plot will be ***overplotted*** (i.e. points in the visualizations plotting on top of each other). That is, because there are only a finite number of values that *X* can take, all of the points will lie along those lines. ***Which means, that it is not possible to interpret the information in this plot.***

If you do decide to use scatterplot, you should use `jitter`. _

- **Standard scatterplot:** Seaborn's_ `.stripplot()` allows you to use `jitter`.

```
g = sb.FacetGrid(data = tot_cases_cont_no_0, row = 'continent', height=4, aspect=2.5) # Create FacetGrid class
g.map(sb.stripplot, 'month', 'new_cases', jitter = 0.35); # add x and y axes
g.fig.set_size_inches(14.70, 10.27);
plt.tight_layout();
```

(where the `jitter` *value is the maximum value that the points can be jittered around the original position, and the* `dodge` *option separates the categories. That is, for that plot, add the option* `jitter = 0.35` *(**Seaborn** allocates a space of* `1` *for each discrete value, jittering by* `0.35` *randomly moves the points around their original location **(both sides)**, so that is a space of* `0.7` *the remaining* `0.3` *allows enough room between categories so that you have a space between each).).*

As you can see, the distribution of the data, for each continent, can now be clearly seen.

Here is a discussion of using jitter

Explanatory Data Analysis

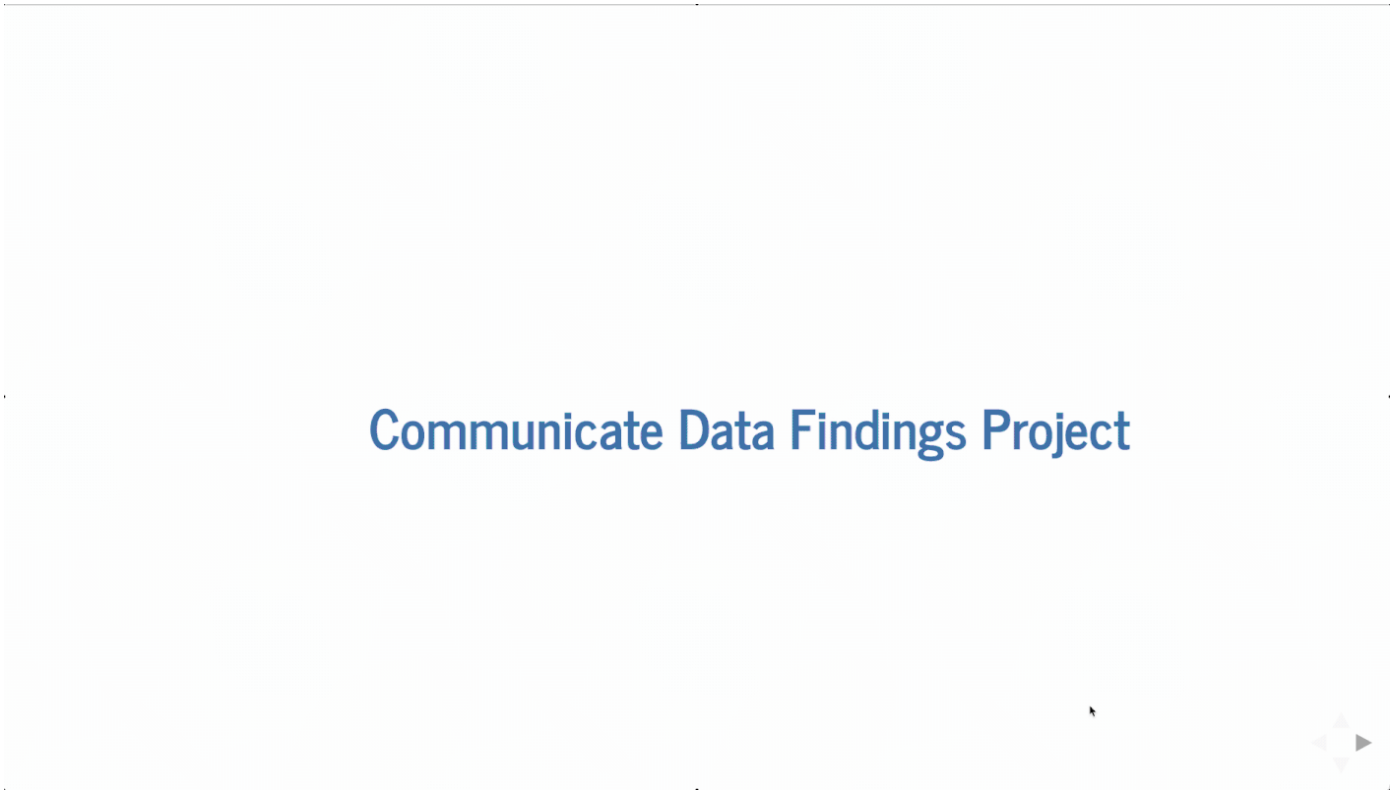
A section in the submitted materials includes a summary of main findings that reflects on the steps taken during the data exploration. The section also describes the key insights that are conveyed by the explanatory presentation.

You summarize the contents of your project files in your `readme.md` file.

After reading the readme file, the reader should have a clear sense of what to expect in your *data exploration* and slide deck/show. Your summary is excellent is this regard. Very nice work!

A slideshow is provided, with at least three visualizations used in the presentation to convey key insights. These key insights match those documented in the summary. Each visualization is associated with comments that accurately depict their purpose.

You include a slide show notebook and a slide show *HTML* file in your submission.



Your plot selection, and commentary, are excellent. Very nice work!

All plots in the presentation have an appropriate title with labeled axes and legends. Labels include units as needed. Plot type,

encodings, and transformations are all appropriate.

All *(relevant)* plots in the presentation have an appropriate title with labeled axes and legends *(you specifically chose to omit labels on some plots, for aesthetic reasons)*.

TIP

As can be seen in the gif above, one of your visualizations is too large to fit on a slide.

1. PLOT DIMENSIONS

A. *MATPLOTLIB PLOTS*

The plot dimensions change from plot to plot in your slide show.

Using a fixed plot dimension for all of your visualizations helps (so that the reader doesn't have to re-focus on each slide (and the increased width means that you don't have to angle some of your tick labels)).

An A4 (portrait) page size is:

```
plt.figure(figsize=[ 11.69, 8.27])
```

However, the aspect ratio for most computer screens (and projectors) is 16/9. So, to use the A4 (portrait) height with a 16/9 aspect ratio, you would use

```
plt.figure(figsize=[14.70, 8.27])
```

Note: **The statements have to be placed *before* any other code for your plot**

B. ***SOME*** *SEABORN PLOTS*

Some *Seaborn* plots *(called figure level plots, like* `.catplot()` *,* `.FacetGrid()` *,* `.PairGrid()` *, etc)* use `height` and `aspect`, so changing the plot dimensions is slightly different. *(That is, for these plots, using* `plt.figure()` ***does not work*** *(it only creates a python message with no effect on the visualization))*.

As noted, they use `height` (formerly `size`, which is deprecated - *(If* `height` *throws an error, either update* ***Seaborn*** *or use* `size` *)* and `aspect` - which are for *each individual* facet. That is, the width (`14.70` , for example) and height (`8.27` , for example) has to be divided amongst all of the facets.

For example, if a plot is 4 x 3 (4 rows, 3 columns), so you would use:

```
height = 8.27/4,  aspect = (14.70/3)/(8.27/4)
```

replace the divisors with the appropriate row and column values for your plot.

For `1 x 1` plots, this simplifies to

```
height = 8.27,  aspect = 14.70/8.27
```

Here is an example (using a different file):

Click On Images To Enlarge Them

This is explained in more detail in this post

C. *PANDAS PLOTS*

For pandas plots, you put the figure size in the plot call:

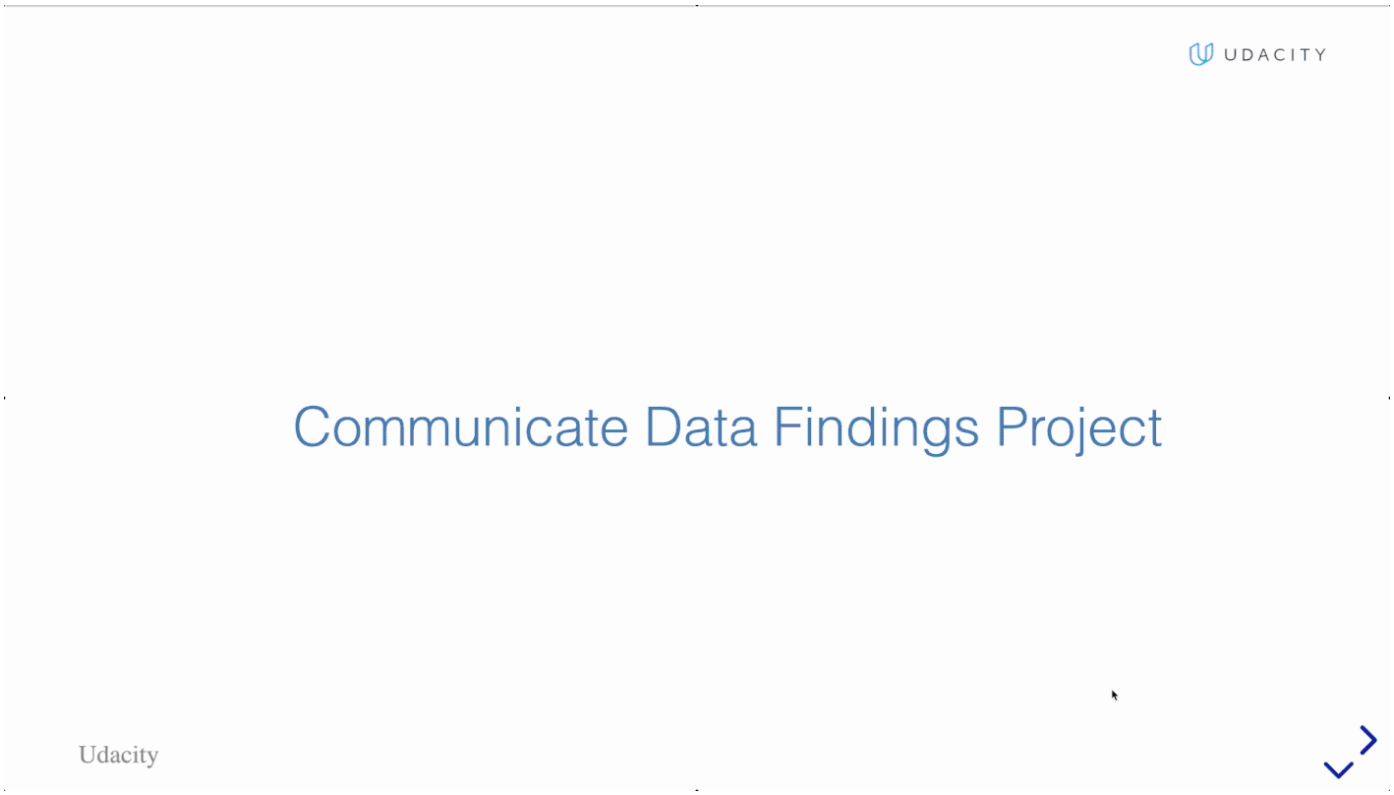
...plot(figsize=[14.70, 8.27])

D. *VISUALIZATIONS WITH SUBPLOTS*

Also, if a plot has subplots, using:

```
plt.tight_layout();
```

optimizes the distances between the visualizations



DOWNLOAD PROJECT

RETURN TO PATH