COVID-19 Data Visualization with Python

Aahana Sapra

February 1, 2022

PROBLEM & SOLUTION

Problem Statement

Design a data visualization solution that is cost-effective and displays the COVID-19 status for all counties in Oregon.

Real Life Application

This design can be used by decision makers at various levels—school districts, city governments, state governments, and businesses—to make informed decisions about in-person and online meetings. It can also be used for recommendations such as mask wearing, social distancing, and getting vaccinated. Businesses, including retailers, restaurants, etc., can use visualizations to plan for physical store openings, curbside pick-up, and delivery. Data visualization can also be used by the general public to understand the current spread of COVID and take precautionary actions to minimize the risk of getting infected.

Criteria

1. The graphs, maps, and charts must be visually-appealing and easy to understand
    - Graphs must be colorful, labeled, and simple
2. Low code
    - Involves minimal programming
3. Programmable & Reusable

    - Easy to program and replicate
    - Can be shared

4. Automated to retrieve the latest data

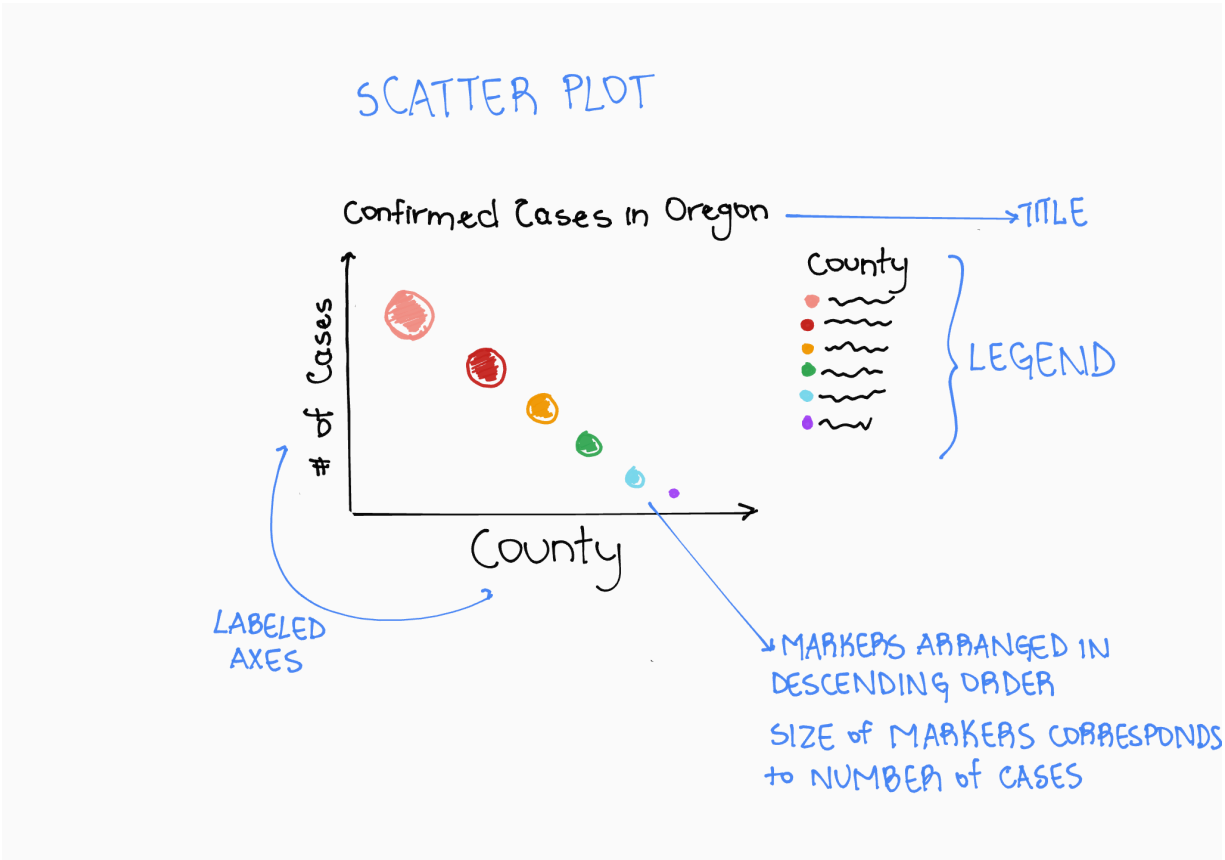
Constraints

1. Cost: the model should be built using free or open-source tools, so that it is easily accessible to everyone
2. Time to learn: use tools that are easy to learn because I have to prior experience with coding

Background Information

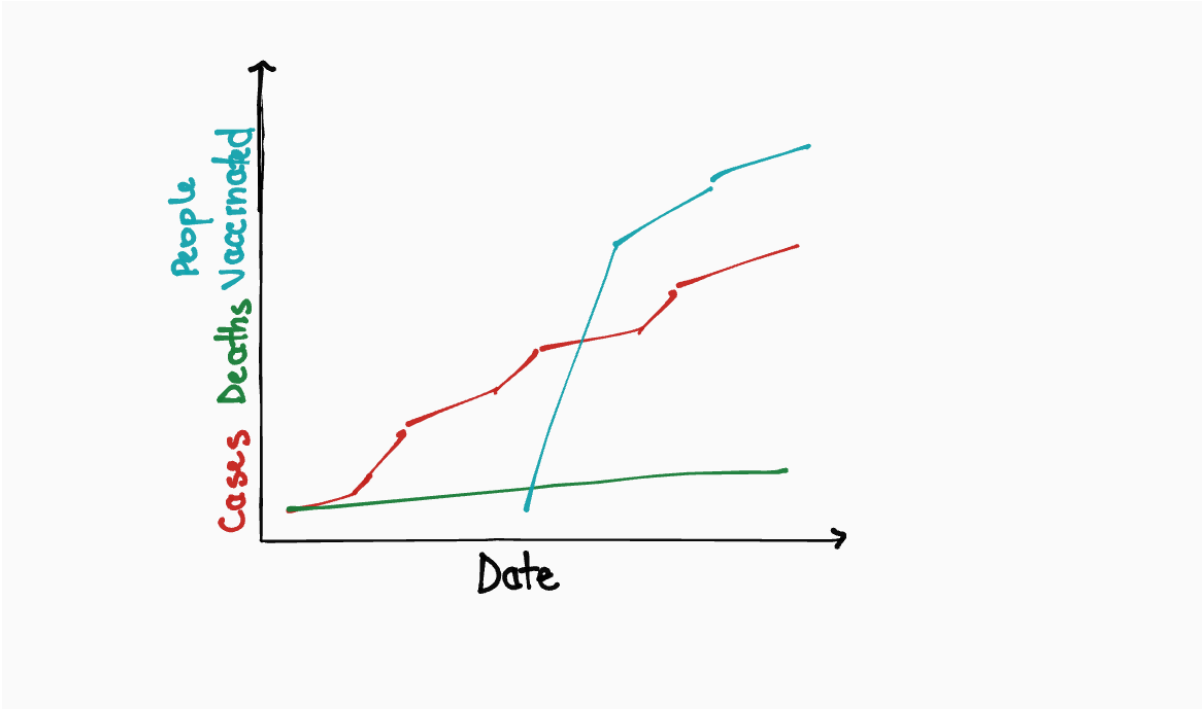
DESIGNING THE SOLUTION

Prototype 1



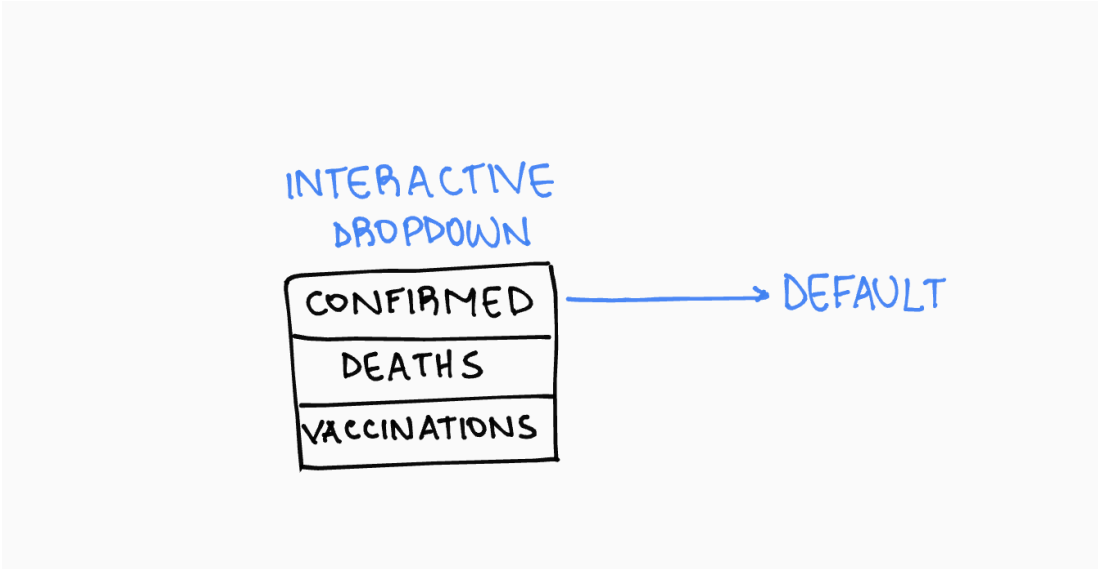
Pros	Cons
- Visually-appealing - Low code - Automated to retrieve latest data	- Can't be shared

Prototype 2



Pros	Cons
- Visually-appealing - Low code - Automated to retrieve latest data	- Can't be shared

Prototype 3



Interactive Dash Application that includes the two graphs above

Pros	Cons
- Visually-appealing - Low code - Automated to retrieve latest data - Can be shared	- Time-consuming

Chosen Prototype: Prototype 3

Materials List

- 1 Computer
- Anaconda (installed)
- Jupyter Notebook (opened through Anaconda)
- John Hopkins University (JHU) Time Series COVID-19 Data for Confirmed US Cases, US Deaths, and US Vaccinations

Procedure

Choropleth Map

1. Import the following libraries: Pandas and NumPy and the module Plotly Express from Plotly

    a. Plotly isn't included in Anaconda, so it has to be pip installed

```
In [4]: !pip install plotly

        Collecting plotly
          Downloading plotly-5.6.0-py2.py3-none-any.whl (27.7 MB)
        Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from plotly) (1.16.0)
        Collecting tenacity>=6.2.0
          Downloading tenacity-8.0.1-py3-none-any.whl (24 kB)
        Installing collected packages: tenacity, plotly
        Successfully installed plotly-5.6.0 tenacity-8.0.1
```

```
In [1]: import pandas as pd
        import numpy as np
        import plotly.express as px
```

2. Download the data for confirmed cases in the US from the John Hopkins University COVID-19 data repository

```
In [236]: confirmed_us_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_
          confirmed_us_df.head()
```

Out[236]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_	...	3/17/22	3/18/22	3/19/22	3/20/22	3/21/22	3/22/22
0	84001001	US	USA	840	1001.0	Autauga	Alabama	US	32.539527	-86.644082	...	15578	15582	15582	15582	15586	15593
1	84001003	US	USA	840	1003.0	Baldwin	Alabama	US	30.727750	-87.722071	...	55218	55327	55327	55327	55353	55355
2	84001005	US	USA	840	1005.0	Barbour	Alabama	US	31.868263	-85.387129	...	5465	5467	5467	5467	5471	5475
3	84001007	US	USA	840	1007.0	Bibb	Alabama	US	32.996421	-87.125115	...	6408	6411	6411	6411	6411	6411
4	84001009	US	USA	840	1009.0	Blount	Alabama	US	33.982109	-86.567906	...	14883	14889	14889	14889	14891	14892

5 rows × 806 columns

3. Filter the confirmed cases data frame for Oregon counties

```
In [237]: # Filter Data for Oregon
          confirmed_oregon_df = confirmed_us_df[confirmed_us_df.Province_State == 'Oregon']
          confirmed_oregon_df.head()
```

Out[237]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_	...	3/17/22	3/18/22	3/19/22	3/20/22	3/21/22
2293	84041001	US	USA	840	41001.0	Baker	Oregon	US	44.709156	-117.674988	...	3203	3204	3204	3204	3204
2294	84041003	US	USA	840	41003.0	Benton	Oregon	US	44.491673	-123.431699	...	14974	14976	14976	14976	14989
2295	84041005	US	USA	840	41005.0	Clackamas	Oregon	US	45.187874	-122.217963	...	59652	59674	59674	59674	59715
2296	84041007	US	USA	840	41007.0	Clatsop	Oregon	US	45.997129	-123.660711	...	4564	4564	4564	4564	4570
2297	84041009	US	USA	840	41009.0	Columbia	Oregon	US	45.944643	-123.089090	...	7521	7526	7526	7526	7532

5 rows × 806 columns

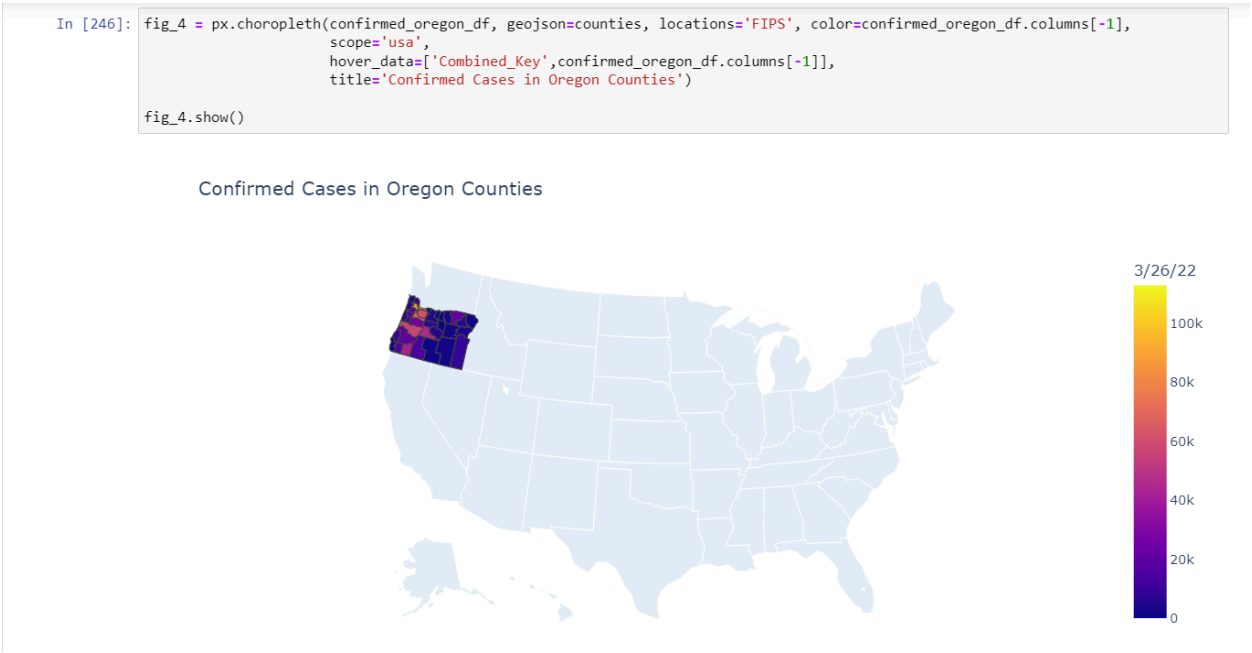
4. Read the GeoJSON data from the Plotly GitHub repository and load it for use. The properties of the GeoJSON data include the FIPS code and geometry—multiple latitude and longitude coordinates for boundaries—of US counties.

```
In [110]: from urllib.request import urlopen
          import json
          with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
              counties = json.load(response)
```

5. Plot a choropleth map for confirmed cases in Oregon. Give the following parameters: the confirmed cases data frame, GeoJSON, location, color, scope, hover data (information that is displayed when you hover your cursor over a certain region), and title.

```
In [246]: fig_4 = px.choropleth(confirmed_oregon_df, geojson=counties, locations='FIPS', color=confirmed_oregon_df.columns[-1],
                                scope="usa",
                                hover_data=['Combined_Key',confirmed_oregon_df.columns[-1]],
                                title='Confirmed Cases in Oregon Counties')

          fig_4.show()
```

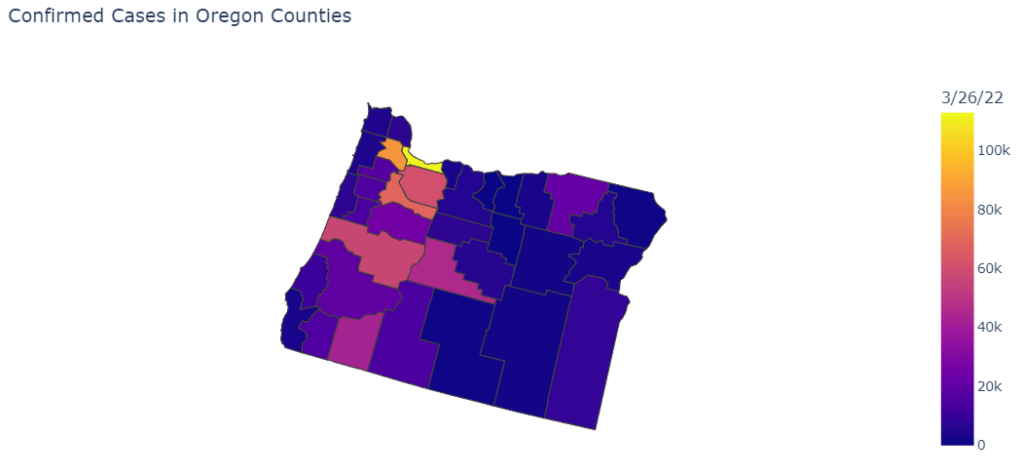


6. Update the geographical data: use fitbounds to only show Oregon in the map

```
In [247]: fig_5 = px.choropleth(confirmed_oregon_df, geojson=counties, locations='FIPS', color=confirmed_oregon_df.columns[-1],
                                scope='usa',
                                hover_data=['Combined_Key',confirmed_oregon_df.columns[-1]],
                                title='Confirmed Cases in Oregon Counties')

          fig_5.update_geos(fitbounds = 'locations', visible=False)

          fig_5.show()
```



Dash

7. Import the following modules from the dash library: Dash, html, and dcc. Dash is a library that can be used to make interactive apps and dashboards.

```
In [248]: !pip install jupyter-dash
```

```
In [118]: from dash import Dash, html, dcc
```

8. Initiate the Dash application

```
In [119]: app = Dash(__name__)
```

9. Give the layout of the Dash web page and run the web server on your local computer

```
In [120]: app.layout = html.Div([
              html.H4('Confirmed Covid-19 Cases in Oregon'),
              dcc.Graph(id='oregon_graph', figure=fig_5),
          ])
          if __name__ == '__main__':
              app.run_server(debug=True, use_reloader=False)

          Dash is running on http://127.0.0.1:8050/

          Dash is running on http://127.0.0.1:8050/

           * Serving Flask app "__main__" (lazy loading)
           * Environment: production
             WARNING: This is a development server. Do not use it in a production deployment.
             Use a production WSGI server instead.
           * Debug mode: on
```

Bar Graph

10. Drop all columns in the confirmed cases data frame except for the Combined_Key column and the date columns for the confirmed cases

```
In [26]: confirmed_oregon_reduced = confirmed_oregon_df.copy()
         confirmed_oregon_reduced.drop(confirmed_oregon_reduced.iloc[:, 0:11], inplace = True, axis = 1)
         confirmed_oregon_reduced.head()
```

Out[26]:

	Combined_Key	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/20	...	3/17/22	3/18/22	3/19/22	3/20/22	3/21/22	3/22/22	3/23/22
2293	Baker, Oregon, US	0	0	0	0	0	0	0	0	0	...	3203	3204	3204	3204	3204	3204	3204
2294	Benton, Oregon, US	0	0	0	0	0	0	0	0	0	...	14974	14976	14976	14976	14989	14994	15003
2295	Clackamas, Oregon, US	0	0	0	0	0	0	0	0	0	...	59652	59674	59674	59674	59715	59731	59747
2296	Clatsop, Oregon, US	0	0	0	0	0	0	0	0	0	...	4564	4564	4564	4564	4570	4572	4573
2297	Columbia, Oregon, US	0	0	0	0	0	0	0	0	0	...	7521	7526	7526	7526	7532	7534	7536

5 rows × 796 columns

11. Transpose the confirmed cases data frame. Transposing a data frame turns the columns into rows and rows into columns. By doing this, the the dates for the confirmed cases become their own rows

```
In [29]: or_cfm_transposed = confirmed_oregon_reduced.T
         or_cfm_transposed.head()
```

Out[29]:

	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	...	2321	2322	2323	2324
Combined_Key	Baker, Oregon, US	Benton, Oregon, US	Clackamas, Oregon, US	Clatsop, Oregon, US	Columbia, Oregon, US	Coos, Oregon, US	Crook, Oregon, US	Curry, Oregon, US	Deschutes, Oregon, US	Douglas, Oregon, US	...	Sherman, Oregon, US	Tillamook, Oregon, US	Umatilla, Oregon, US	Unassigned, Oregon, US
1/22/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1/23/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1/24/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1/25/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

5 rows × 38 columns

12. Remove the first row (Combined_Key) of the confirmed cases data frame and save it into a new series. This makes turns the index into the column headers

```
In [38]: new_headers = or_cfm_transposed.iloc[0] # get the first row as a new column names
         or_cfm_transposed = or_cfm_transposed[1:] # take the data frame except the first row
         or_cfm_transposed.head()
```

Out[38]:

	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	...	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330
1/22/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1/23/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1/24/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1/25/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1/26/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 38 columns

13. Make the series created in the previous step into the column names. This gets rid of the index and makes the Combined_Key the column headers

```
In [45]: or_cfm_transposed.columns = new_headers # set the fist row as the new column names
         or_cfm_transposed.head()
```

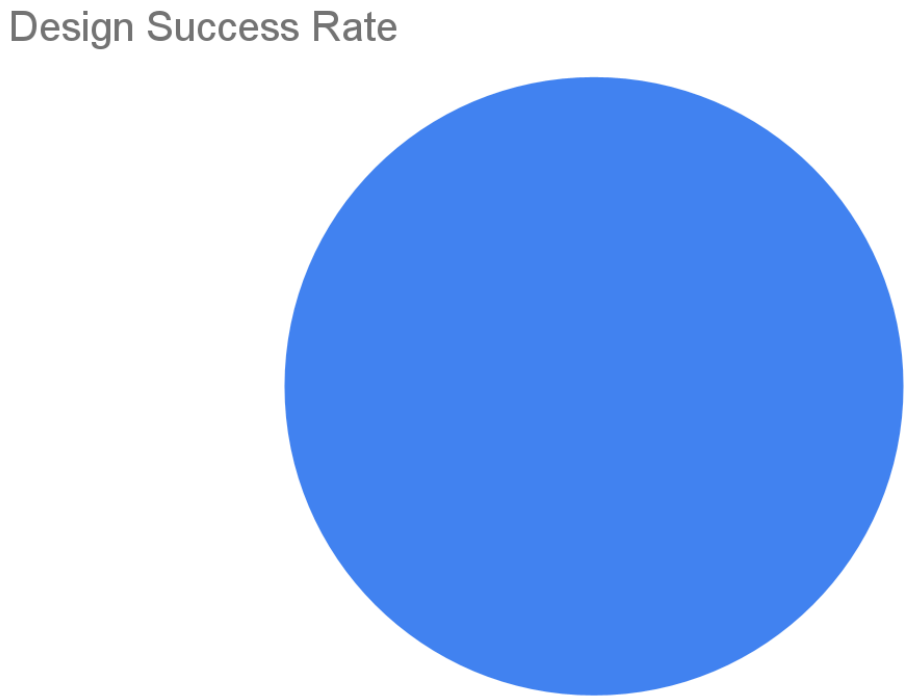
Out[45]:

Combined_Key	Baker, Oregon, US	Benton, Oregon, US	Clackamas, Oregon, US	Clatsop, Oregon, US	Columbia, Oregon, US	Coos, Oregon, US	Crook, Oregon, US	Curry, Oregon, US	Deschutes, Oregon, US	Douglas, Oregon, US	...	Sherman, Oregon, US	Tillamook, Oregon, US	Umatilla, Oregon, US	Unas Ore
1/22/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1/23/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1/24/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1/25/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1/26/20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 38 columns

ANALYZE DATA

Trial Number	Trial 1	Trial 2	Trial 3	Success Rate
Success: Yes or No	Yes	Yes	Yes	100%



Blue: Success Rate

FINAL DESIGN

Evaluation of Solution

Ploty Express choropleth maps are visually appealing because they are similar to heat maps where the intensity of the color changes according to the metric of the data. In this design, the metric is the number of cases, and as the number of cases increases, the shade of each region becomes brighter. Plotly choropleth maps support the GeoJSON format, which helps plot the boundaries of counties, states, and countries on a map. GeoJSON is an open-standard format that includes multiple latitude

and longitude points that allow it to plot county and state boundaries on a map. The Plotly dataset also includes the FIPS code for US counties, which helps in correlating the data from JHU.

Python Pandas has in-built functions for data manipulation, including reading data in multiple formats, organizing the data into a dataframe, dropping columns in dataframes, transposing dataframes, and more. Plotly Express has in-built functions to create a variety of plots, including line charts, bar charts, scatter plots, bubble maps, and choropleth maps. Using Pandas and Plotly Express together makes it easy to create visualizations with minimal code.

This model can be reused for other simple applications, such as representing student grades in a scatter plot or the population density for a region in a choropleth map.

Automated: the model uses time series data, which is updated daily. The code is designed such that it can read the latest data and update the graphs accordingly.

There were two constraints in my project: cost and time. The model follows the cost constraints as it uses open-source Python libraries—Pandas, NumPy, and Plotly—and the free version of Anaconda. In addition, I met the time constraint by using Python to program this design. Python is a beginner-friendly programming language and there are a lot of tutorials available online. However, I have yet to fully develop an interactive application. I started a prototype in Plotly Dash, but I want to expand on that design by making it interactive.

Strengths & Weaknesses

One weakness of the design is that I can't come up with a conclusive statement about the impact of vaccinations on deaths. As per the line graphs, the number of deaths increased at the greatest

rate of change in the last quarter of 2021, after many people were already vaccinated. It also isn't clear if the data includes people who got their booster vaccine, or just received two doses.

Improve Design

I can improve my design by fully developing an interactive application in Plotly Dash, enabling others to personalize what data they see. I could also use datasets from different places to model the spread of COVID in other regions. Another thing I could do to better my design would be to use different graphs that can be used to make more certain conclusions about the effect of vaccinations on death and such.