FINAL PROJECT: COVID-19 DASHBOARD

Algorithm and Programming

Alysha Maulidina

2502005906 | L1BC

Table of Contents

Introduction

Background

For Algorithm and Programming final project, students are expected to design a comprehensive application that solves a problem, using all the knowledge and skills we have gained in Python over the semester. Students are also encouraged to apply topics beyond the syllabus, which I have taken the challenge of doing so in my personal project. This report outlines the design and development of the resulting project and some analysis of the research conducted that influenced my decisions during the design process.

After ruminating over several ideas, I chose to create a COVID-19 dashboard, which I will further discuss its purpose in the function section. At the beginning, I had little experience with data science and manipulation on Python, as we only briefly covered it before the Christmas holiday. Thus, I spent every day of my break dedicated to learning, understanding, and processing my datasets, as well as testing at range of data visualization libraries to see which one I see fit for the dashboard.

Function

Taking various datasets, I plan to build an app that would allow users to visualize and analyze COVID-19 data easily. I intend to gather multiple data sources so that users can examine a detailed overview of COVID-19 statistics in one platform. During research, I came across datasets that showcase the impact of COVID-19 on certain socio-economic aspects, such as data on air quality in Jakarta during the pandemic, or areas of Jakarta that gather the most quarantine violation reports, and combined with the statistics COVID-19 cases, I believe that visualizing them can lead to some interesting discoveries. It was important for the users to be able to interact with the data and answer questions such as:

- *Are relationship between population of males and total cases?*
- *Which regions in Indonesia has the lowest number of deaths?*
- *Which regions in Jakarta are leading vaccination rollouts?*
- *How does the pandemic affect air quality?*

To achieve this, I plan to use the following libraries:

I. Streamlit

Deciding the library for my dashboard took a lot of consideration, as there are frameworks that are optimized for specific use cases involving data visualization. I

chose Streamlit, as it allows for rapid prototyping as well as a significant degree of interactivity. It also has native support for most Python plotting libraries, such as Plotly.

II. **Pandas**

Pandas is an essential package when it comes to data science projects. It provides powerful functions and commands to easily manipulate large sets of data. I will use this to "clean" and process my data where necessary before plotting them.

III. **Plotly**

I chose Plotly over other data visualization libraries as one of its advantages is that it allows for greater customization and interactivity for complex plots in a code efficient manner.

IV. **Pydeck**

Besides Plotly, I plan on using Pydeck, as some of my datasets come with spatial data that I can present using Pydeck. It supports rich geographical data visualization, such as extruding maps and 3D scatter plots.

V. **Prophet**

Prophet is designed for making forecasts for time series datasets. I will use this for COVID-19 predictions.

Solution Design Plan & Sketchboard

Steamlit does not allow flexibility in its layout options as the focus is primary on the ease of use. I have arranged containers and complements in such way that fits with the default layout, and adopted a standard dashboard design that can be easily understood by a regular user.

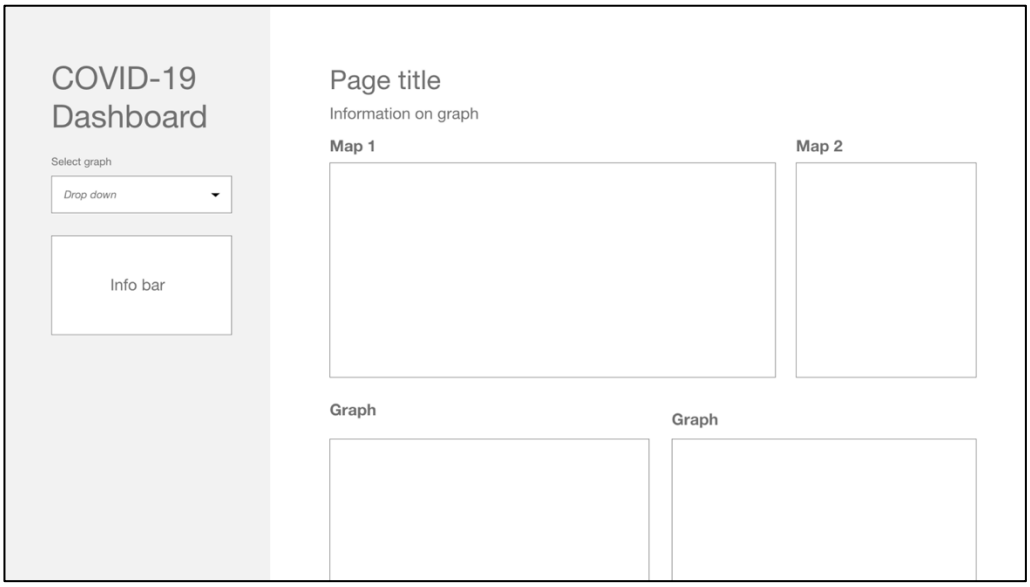


Figure 1: Sketchboard for dashboard interface

Structure of the app:

I. Pages
   a. *home.py*: landing page containing information on the app and a reference list in some form
   b. *indonesia.py*: dashboard for statistics in Indonesia
   c. *jakarta.py*: dashboard for statistics in Jakarta
   d. *world.py*: dashboard for statistics around the world
   e. *predictions.py:* dashboard for COVID-19 predictions

II. Sidebar (collapsible)
   a. Navigation in the form of a drop down menu where users can switch between pages
   b. Info bar contains a brief explanation of what the app and its purpose

Methodology and research

Data preparation and processing with Pandas

The first step was to find the necessary datasets which lays the foundation of the project. I obtained the datasets from multiple sources, as follows:

- Indonesia's official COVID-19 information site, *covid19.go.id*
- Datasets specifically for Jakarta, *corona.jakarta.go.id*
- Jakarta Smart City
- Our World in Data (OWID)

- John Hopkins University

These data sets will then go through a series of data processing, which I have done via Jupyter Notebook.

In this case, I will be examining the data sets *covid_province*, *covid_province_recovered*, and *covid_province_death* that is used for *Indonesia.py*.

```
def load_data(data):
        return pd.read_csv(data)

df_map = load_data("https://raw.githubusercontent.com/alyshapm/FP_AlgoProg/main/data/kasus_per_provinsi.csv")
df_jakarta = load_data("https://raw.githubusercontent.com/alyshapm/FP_AlgoProg/main/data/covid_jakarta.csv")
df_deaths = load_data("https://raw.githubusercontent.com/alyshapm/FP_AlgoProg/main/data/covid_province_death.csv")
df_confirmed = load_data("https://raw.githubusercontent.com/alyshapm/FP_AlgoProg/main/data/covid_province.csv")
df_recovered = load_data("https://raw.githubusercontent.com/alyshapm/FP_AlgoProg/main/data/covid_province_recovered.csv"
```

Figure 2: Loading csv files

```
[16] df_deaths.head(5)
```

	Province	Lat	Long	Population	2020-03-21	2020-03-22	2020-03-23	2020-03-24	2020-03-25	2020-03-26
0	Aceh	4.695135	96.749399	5274871	0	0	0	0	0	0
1	Bali	-8.409518	115.188916	4317404	1	2	2	2	2	2
2	Banten	-6.405817	106.064018	11904562	2	3	3	4	4	4
3	Babel	-2.741051	106.440587	1455678	0	0	0	0	0	0
4	Bengkulu	-3.577847	102.346388	2010670	0	0	0	0	0	0

5 rows × 563 columns

```
[17] df_deaths.shape

(34, 563)
```

Figure 3: First 5 rows of *df_deaths*

Upon observation, we learn that the data frame span 34 rows and 563 columns, with each column representing a day in the year (excluding the metric columns). Data visualization libraries, however, often expect data to be in a tidy or ‘long’ format so that the functions can correctly interpret and present the data. This can be achieved using Pandas’ Melt function.

```
[21] date_index = df_deaths.columns[4:] # specifies which columns to unpivot
     total_deaths = df_deaths.melt(
         id_vars=["Province", "Lat", "Long", "Population"], # columns to use as identifier variables
         value_vars=date_index,
         var_name="date", # name for the variable column
         value_name="deaths", # name for the value column
     )
     total_deaths
```

	Province	Lat	Long	Population	date	deaths
0	Aceh	4.695135	96.749399	5274871	2020-03-21	0
1	Bali	-8.409518	115.188916	4317404	2020-03-21	1
2	Banten	-6.405817	106.064018	11904562	2020-03-21	2
3	Babel	-2.741051	106.440587	1455678	2020-03-21	0
4	Bengkulu	-3.577847	102.346388	2010670	2020-03-21	0
...	...	...	...	...	...	...
19001	Papbar	-1.336115	133.174716	1134068	2021-09-30	352
19002	Papua	-4.269928	138.080353	4303707	2021-09-30	496
19003	Sulbar	-2.844137	119.232078	1419229	2021-09-30	336
19004	NTT	-8.657382	121.079371	5325566	2021-09-30	1296
19005	Gorontalo	0.699937	122.446724	1171681	2021-09-30	458

19006 rows × 6 columns

Figure 4: *total_deaths* dataframe after using the melt function

I tidied the rest of the aforementioned data sets and used the *pd.merge* function to combine the confirmed, deaths and recovered columns into one data frame. Additionally, I've included a column for active cases by subtracting deaths and recovered from confirmed. The same method is also applied for the CSSE data sets used in *world.py*.

```
[26] df_final = total_confirmed.merge(right=total_deaths, how="left", on=["Province", "Lat", "Long", "Population", "date"])
     df_final = df_final.merge(right=total_recovered, how="left", on=["Province", "Lat", "Long", "Population", "date"])
     df_final
```

	Province	Lat	Long	Population	date	confirmed	deaths	recovered
0	Aceh	4.695135	96.749399	5274871	2020-03-21	0	0	0
1	Bali	-8.409518	115.188916	4317404	2020-03-21	3	1	0
2	Banten	-6.405817	106.064018	11904562	2020-03-21	43	2	1
3	Babel	-2.741051	106.440587	1455678	2020-03-21	0	0	0
4	Bengkulu	-3.577847	102.346388	2010670	2020-03-21	0	0	0
...	...	...	...	...	...	...	...	...
19001	Papbar	-1.336115	133.174716	1134068	2021-09-30	22899	352	22461
19002	Papua	-4.269928	138.080353	4303707	2021-09-30	33746	496	31441
19003	Sulbar	-2.844137	119.232078	1419229	2021-09-30	12159	336	11567
19004	NTT	-8.657382	121.079371	5325566	2021-09-30	62716	1296	60747
19005	Gorontalo	0.699937	122.446724	1171681	2021-09-30	11735	458	11152

19006 rows × 8 columns

Figure 5: Output of df after merging

Another data type common to data science projects is datetime, which often cause problems without proper formatting. Pandas has a built-in function called *to_datetime()* that converts strings to datetime. This is crucial for our data visualization to work and ensures that the data is plotted correctly, given that we are mostly dealing with timeseries data sets.

```
# converting data column from str to date format
covid_data["date"] = pd.to_datetime(covid_data["date"])
```

Figure 6: Convert date column to datetime data type

Data forecasting with Prophet

In this section, I will be explaining how I generated 3-month forecast of confirmed COVID-19 cases using Prophet. The graphs will be presented in the *predictions.py* file. I first prepared the data and ensured that it complies with Prophet's expected input, a data frame consisting of two columns: **ds** (date stamp) and **y** (the value to be predicted).

```
df = pd.read_csv("data/covid_indonesia.csv") # read file
    df["Date"] = pd.to_datetime(df["Date"]) # ensure date is in the right format
    df = df.groupby("Date").sum()["total_infected"].reset_index() # groups table by Date and
total_infected, removes other columns

    df_prophet = df.rename(columns={"Date":"ds", "total_infected":"y"}) # rename column names for
prophet complience
```

Figure 7: Code for the Prophet df

	ds	y
652	2021-12-15	4259644
653	2021-12-16	4259857
654	2021-12-17	4260148
655	2021-12-18	4260380
656	2021-12-19	4260544

Figure 8: Prophet df output

Then, I built the data frame to fit the daily prediction for the next 3 months.

```
prophet = Prophet(
    changepoint_prior_scale=0.3,
    changepoint_range=0.99,
    yearly_seasonality=False,
    weekly_seasonality=False,
    daily_seasonality=True,
    seasonality_mode='additive'
)

prophet.fit(df_prophet)

build_forecast = prophet.make_future_dataframe(periods=(30*3), freq='D')
forecast = prophet.predict(build_forecast)

prophet.plot(forecast, xlabel='Date', ylabel='Confirmed')
plt.title('Covid')
plt.show()
```

Figure 9: Code for forecasting confirmed cases

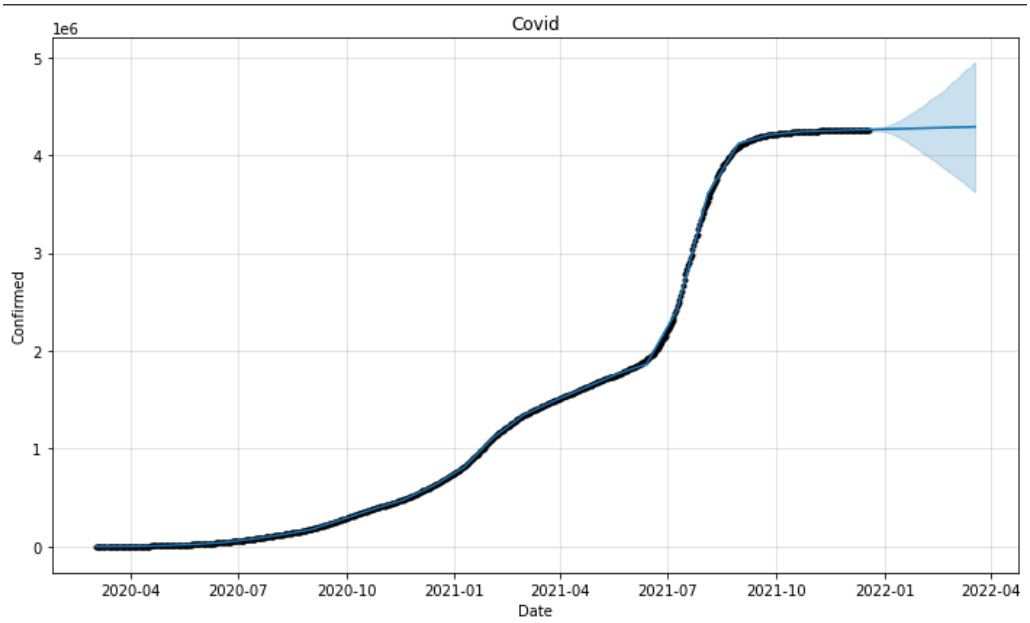


Figure 10: Graph for forecasted data

In the plot above, the black line indicates the actual measurements, the blue line displays the Prophet's forecast, and the light blue window indicates uncertainty intervals. At this point, the graph is ready to be rendered on the app, but I will discuss further about the forecast quality and accuracy.

According to Facebook (founder of Prophet), the framework uses **mean absolute percentage error (MAPE)** to determine the accuracy of a forecast. We can assess the MAPE values of the COVID-19 data using *cross_validation()*.

```
test = cross_validation(prophet, initial='365 days', period='30 days', horizon='90 days')
test2 = performance_metrics(test)
test2.tail(3)
```

```
INFO:fbprophet:Making 7 forecasts with cutoffs between 2021-03-24 00:00:00 and 2021-09-20
100% 7/7 [00:23<00:00, 3.54s/it]
```

	horizon	mse	rmse	mae	mape	mdape	coverage
79	88 days	2.319587e+12	1.523019e+06	1.096792e+06	0.277712	0.202274	0.285714
80	89 days	2.388322e+12	1.545420e+06	1.113455e+06	0.281432	0.203102	0.285714
81	90 days	2.458143e+12	1.567847e+06	1.130171e+06	0.285175	0.203102	0.285714

Figure 11: MAPE values of the forecasted data

Based on the output, Prophet achieved a 28% MAPE for a 90-day forecast of the COVID-19 data, which means that the projected values on the graph are therefore 28% inaccurate.

Setting up Steamlit

Multipage framework

As mentioned already, Streamlit does not have an implicit feature to add new a new page from a single app, but it is possible to build a framework that allows multi-app functionality using existing functions. I've implemented an object-oriented structure that adds pages as an instance of the class *MultiPage*. It's a simple class which contains two methods:

- *add()* is used add more pages to the entire application and each page is appended to the instance variable *apps*
- *run()* displays a dropdown menu from the list of pages where user can select the page. It also contains the code for sidebar interface.

MultiApp
+apps = []
-__init__(apps) +add_app() +run()

Figure 12: Class diagram for MultiApp

```
class MultiApp:
    # constructor class that generates a list which stores the pages as an instance variable
    def __init__(self):
        self.apps = []

    def add_app(self, title, func):
    # title (in str) is the title of the page which will be added to the list of apps that will be displayed in the
dropdown navigation
    # func displays the page in Streamlit
        self.apps.append({
            "title":title,
            "function":func
            })

    def run(self):
        # --- NAVIGATION ---
        st.sidebar.title("Navigation")

        # dropdown displays list of pages and runs the selected page
        app = st.sidebar.selectbox(
            "Select graph", # label
            self.apps,  # list of options
            format_func=lambda app: app["title"]) # receives the selected page as argument and returns it as str

        # runs the app function
        # app used to access the dictionary and return its corresponding value, or func
        app['function']()
```

Figure 13: Code for class MultiApp

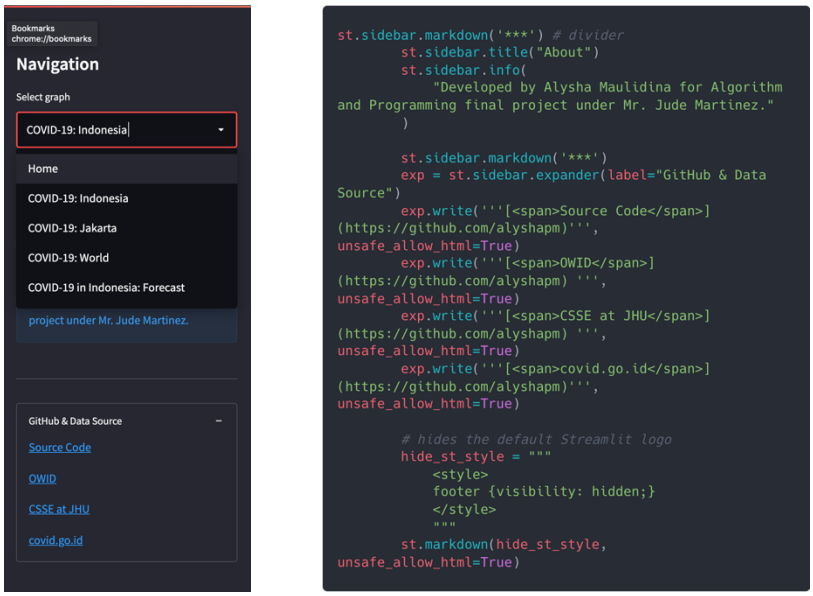


Figure 14a and 14b: Navigation sidebar (left) and the code (right)

Then, I created a separate file *app.py* which acts as a host app and "stores" the independent child pages. The *app* instance adds pages using the class function *add_app(title, func)* and renders the application with *run()*. In the individual pages, the codes are written inside the *app()* function as it will be called in the structure of the multipage application *app.py*.

```
from apps import (
    home,
    indonesia,
    jakarta,
    world,
    predictions)

# set up page layout
st.set_page_config(page_title='COVID-19 Dashboard',
                   page_icon=":bar_chart:",
                   layout="wide"
    )

# declare instance of the app
app = MultiApp()

# add new up using the class function add_app, which takes title and func as the argument
app.add_app("Home", home.app)
app.add_app("COVID-19: Indonesia", indonesia.app)
app.add_app("COVID-19: Jakarta", jakarta.app)
app.add_app("COVID-19: World", world.app)
app.add_app("COVID-19 in Indonesia: Forecast", predictions.app)

# run main app
app.run()
```

Figure 15: Code for *app.py*

Data caching

Throughout the program and in parts where the data processing and visualization takes place, I've added *st.experimental_memo* before a function. The decorator provides a caching

mechanism that allows the app to stay performant when loading large datasets or performing memory-extensive calculations.

```
@st.experimental_memo
    def load_data(data):
        return pd.read_csv(data)

    df_crossplot = load_data("data/covid_jakarta.csv")
    df_extrudingmap = load_data("data/pelanggaran_psbb.csv")
    df_histo = load_data("data/pelanggaran_psbb_timeseries.csv")
    df_airquality = load_data("data/avgcarspeed_jakarta.csv")
```

Figure 16: cache decorator before the load_data function in *Jakarta.py*

Layout: columns

To achieve the side-by-side containers, I've used the st.columns function where I can insert elements.

```
row3_1, row3_2 = st.columns((3,3))
    with row3_1:
        st.plotly_chart(plot_per_province("confirmed", "Confirmed"), use_container_width=True)
        st.plotly_chart(plot_per_province("recovered", "Recovered"), use_container_width=True)
    with row3_2:
        st.plotly_chart(plot_per_province("deaths", "Deaths"), use_container_width=True)
        st.plotly_chart(plot_per_province("active", "Active"), use_container_width=True)
```

Figure 17: Code to set up column containers for the graphs from *Indonesia.py*

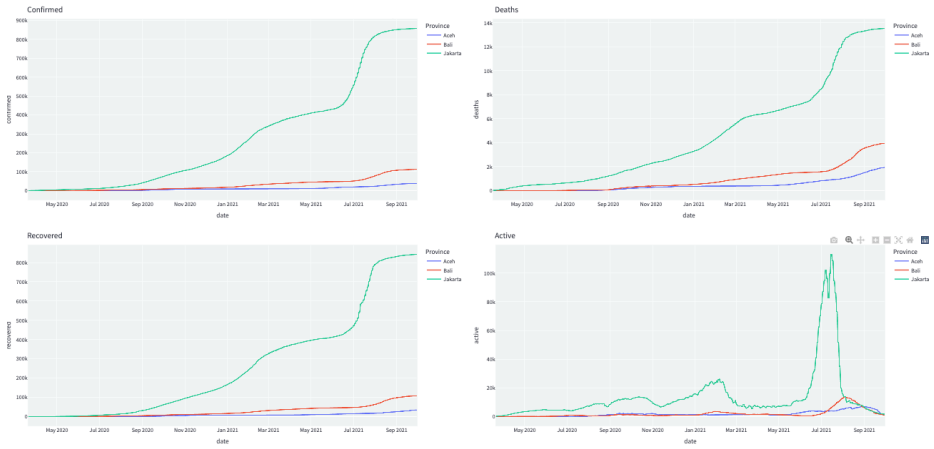


Figure 18: Plotted graphs in its respective containers

Data visualization with Plotly and Pydeck

As we are dealing with multiple types of datasets, there are different ways of analyzing the data which ultimately determines which chart types to use. Line charts, for instance, suit datasets that shows changes in value across continuous measurements, such as time series, and can display a trend to help the reader make predictions for future outcomes. I've used this charting for datasets such as *covid_indonesia.csv* and *covid_province.csv* that come with continuous variables.

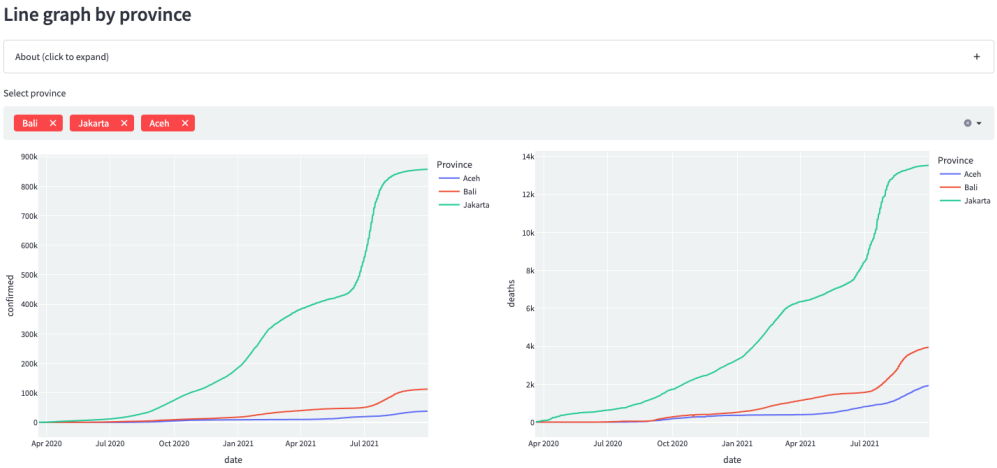


Figure 19: Line graphs of multiple data points based on selected regions

```
@st.experimental_memo
    def plot_per_province(y_axis):
        fig3 = px.line(
            df_final_query,
            x="date",
            y=y_axis,
            color="Province"
            )
        fig3.update_layout(
            margin=dict(l=0, r=0, b=0, t=0, pad=1)
        )
        return fig3

    row3_1, row3_2 = st.columns((3,3))
    with row3_1:
        st.plotly_chart(plot_per_province("confirmed"), use_container_width=True)
        st.plotly_chart(plot_per_province("recovered"), use_container_width=True)
    with row3_2:
        st.plotly_chart(plot_per_province("deaths"), use_container_width=True)
        st.plotly_chart(plot_per_province("active"), use_container_width=True)
```

Figure 20: Code for line graphs

On the other hand, scatter plots display two variables in the form of points positioned on two axes and can demonstrate the correlation between the variables. This type of charting is useful for datasets like *covid_jakarta.csv* with dependent (infections, deaths, vaccinations) and independent variables (median age, population numbers) where we can examine relationships between variables, especially when we consider risk factors and their impact on the pandemic.

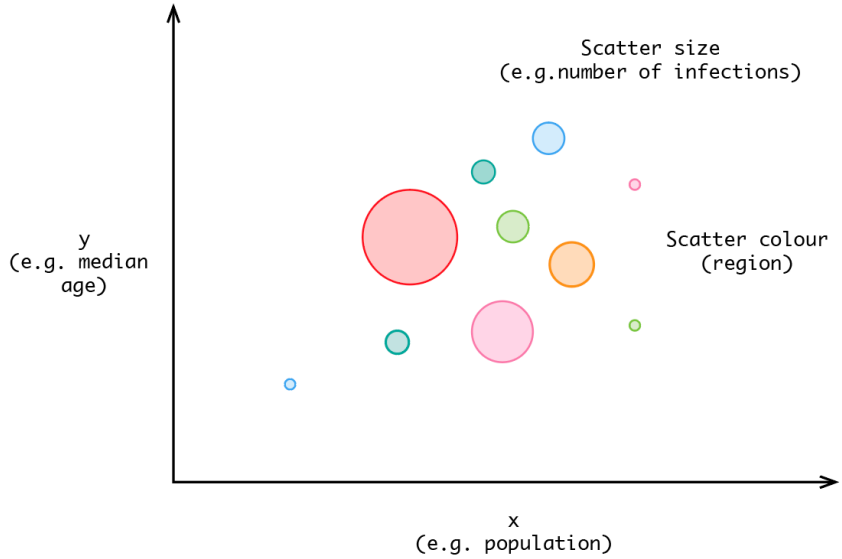


Figure 21: Illustration of a multi dimensional scatter plot

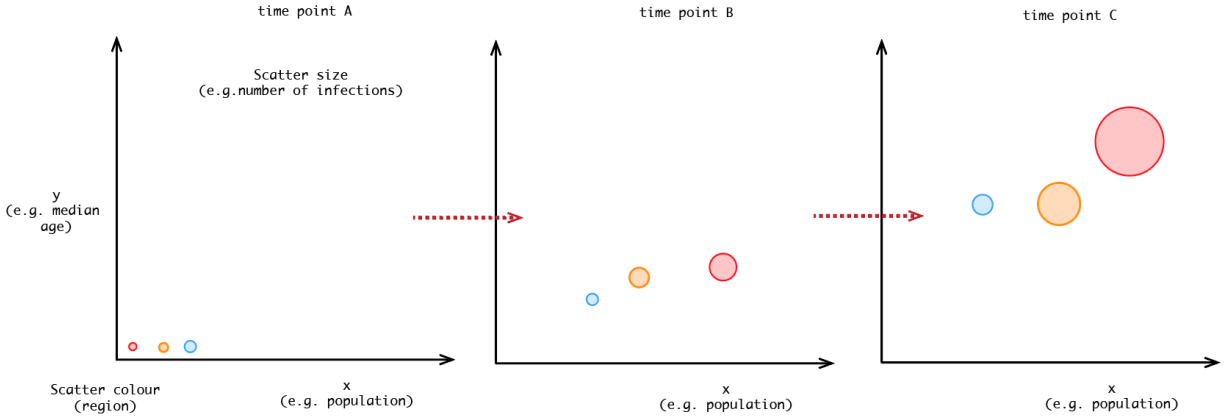
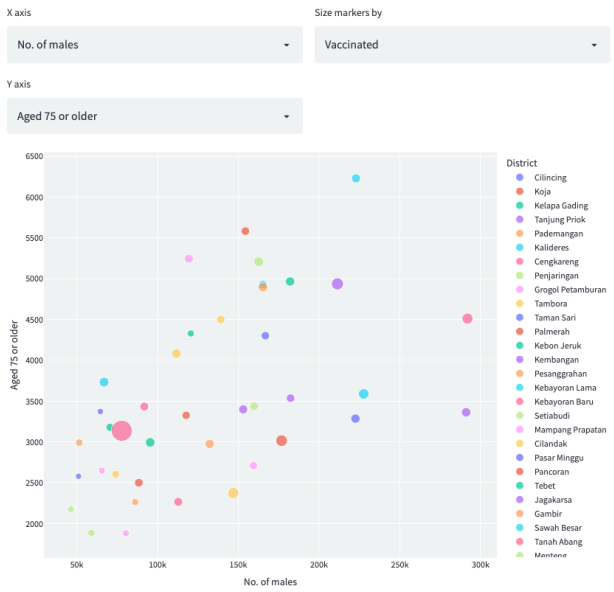


Figure 22: Illustration of multi dimension scatter plot for with time animation

In line with the purpose of this dashboard, I've also allowed customization for certain graphs using Streamlit widgets where users have the ability to select the variables displayed in the X and Y axis. This feature would encourage interactivity and therefore widen the scope of analysis.




```python
    x = row1_1.selectbox("X axis", list(metric_labels.keys()))
    x_ax = metric_labels[x]

    y = row1_1.selectbox("Y axis", list(metric_labels.keys()))
    y_ax = metric_labels[y]

    size_by = row2_1.selectbox("Size markers by",
list(sizing_labels.keys()))
    size_by = sizing_labels[size_by]

    color_up = df_melted["district"].drop_duplicates()

    fig = px.scatter(
        df_melted,
        x=df_melted[df_melted["variable"] == x_ax]["value"],
        y=df_melted[df_melted["variable"] == y_ax]["value"],
        color=color_up,
        size=df_melted[df_melted["variable"] == size_by]["value"],
        labels=dict(x=x, y=y, color="District")
    )
    fig.update_layout(
        autosize=True,
        height=600,
        margin=dict(l=0, r=0, b=20, t=10, pad=1)
    )
```

Figure 23a and 23b: Interactive scatter plot (left) and code (right)

Some datasets also come with geographical data points, such as coordinates or country names, which can be used to represent geographic areas and display spatial distribution. I've used choropleth and scatter maps to represent this.

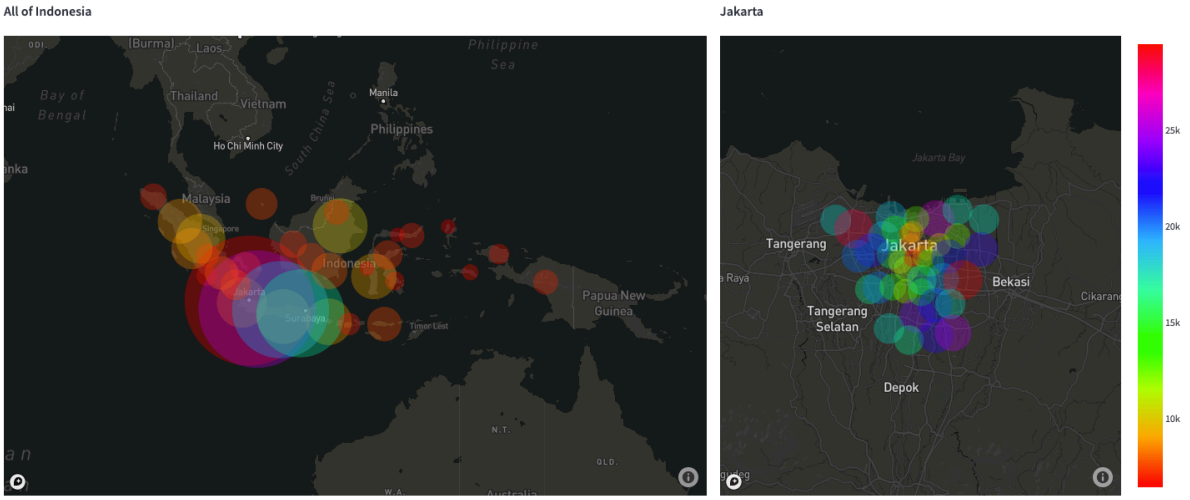


Figure 24: Heat map of confirmed cases for *Indonesia.py*

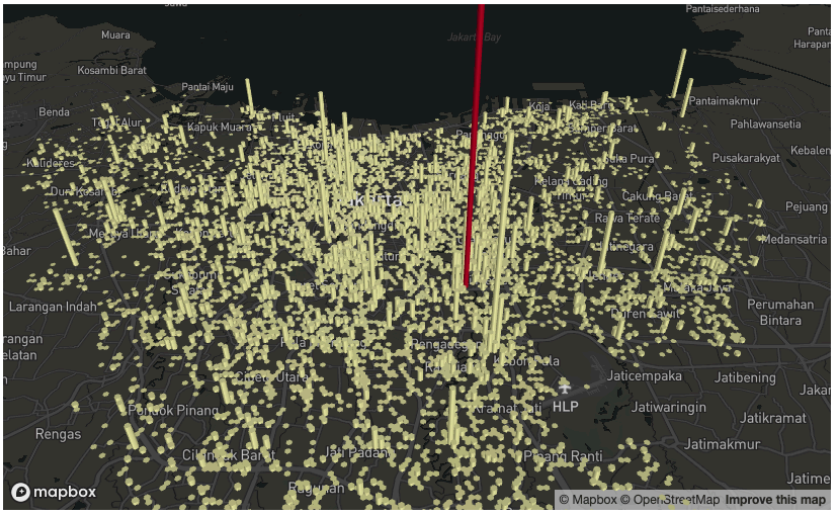


Figure 25: Extruded scatter map of PSBB violation reports using pydeck

```
st.write(pdk.Deck(
        map_style="mapbox://styles/mapbox/dark-v9",
        initial_view_state={
            "latitude": -6.2088,
            "longitude": 106.8456,
            "zoom": 11,
            "pitch": 50,
        },
        layers=[
            pdk.Layer(
                "HexagonLayer",
                data=df_extrudingmap,
                get_position=["long", "lat"],
                radius=100,
                elevation_scale=50,
                elevation_range=[0, 1000],
                pickable=True,
                extruded=True,
            ),
        ]
    ))
```

Figure 26: Code for extruded map in *Jakarta.py*

Other graphs that I have incorporated in the dashboard include histograms, pie chart, bar chart, and area chart.

Evaluation

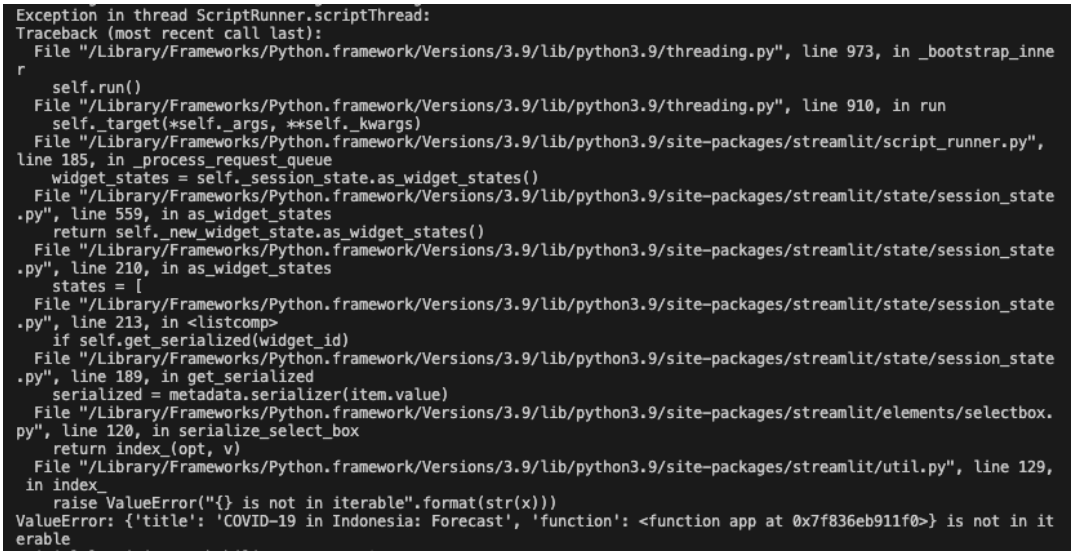
Final result

So far I've only launched the app locally and the final result works and looks just as expected, however I find that the app is running quite slow which I figured was due to the massive data sets that it is processing. I wanted to deploy the web app to see how it works under a different environment and whether it would affect the performance. I deployed the app using Streamlit Cloud and doing so has helped reduce the speed of which each page loads to a certain degree.

Otherwise, the app follows the original solution plan and structure, and it functions as I intended it to be. In terms of layout, however, I've added extra functionality using Streamlit's various widgets, such as expanders that I use to store information on each graph.

Errors/Bugs

When I run the app locally and refresh the page to see the changes I've made in the code, I receive the following traceback error on terminal.



```
Exception in thread ScriptRunner.scriptThread:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/threading.py", line 973, in _bootstrap_inne
r
    self.run()
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/threading.py", line 910, in run
    self._target(*self._args, **self._kwargs)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/script_runner.py",
line 185, in _process_request_queue
    widget_states = self._session_state.as_widget_states()
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/state/session_state
.py", line 559, in as_widget_states
    return self._new_widget_state.as_widget_states()
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/state/session_state
.py", line 210, in as_widget_states
    states = [
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/state/session_state
.py", line 213, in <listcomp>
    if self.get_serialized(widget_id)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/state/session_state
.py", line 189, in get_serialized
    serialized = metadata.serializer(item.value)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/elements/selectbox.
py", line 120, in serialize_select_box
    return index_(opt, v)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/streamlit/util.py", line 129,
 in index_
    raise ValueError("{} is not in iterable".format(str(x)))
ValueError: {'title': 'COVID-19 in Indonesia: Forecast', 'function': <function app at 0x7f836eb911f0>} is not in it
erable
```

Figure 27: Traceback error from terminal

However, it doesn't affect the app in anyway as it continues to perform normally. I did research and altered parts of MultiClass file which seems to be the source of the bug, but I have yet to solve the error. I found that re-running the page (i.e. pressing R) instead of refreshing the entire app (i.e. command/ctrl + R) helps to prevent the traceback error from occurring.

Future development & conclusion

There is no doubt that the final result leaves a lot of room for potential developments. I'd look into migrating the dashboard to Dash as it allows for more customization and certainly outperforms Streamlit in terms of speed. I also find that the code also requires a lot of fixing, perhaps by adding a helper function or another class for the data visualization components to improve readability and avoid repeating functions across pages. I would also like to further develop the forecast page, by implementing a more rigorous and comprehensive data

projection algorithm that considers variables such as vaccination, risk factors, re-infections and so on.

Overall, I am satisfied with the outcome of this project. The functionality related to the core purpose is working as desired. I've learnt above and beyond of the original scope and what I gathered in class, and this project is evidence that I was able to study and train myself in a subject that was once completely foreign to me – data science. I hope to sharpen my data science skills in my future learning endeavors.

Evidence of a working program

I. Home - home.py

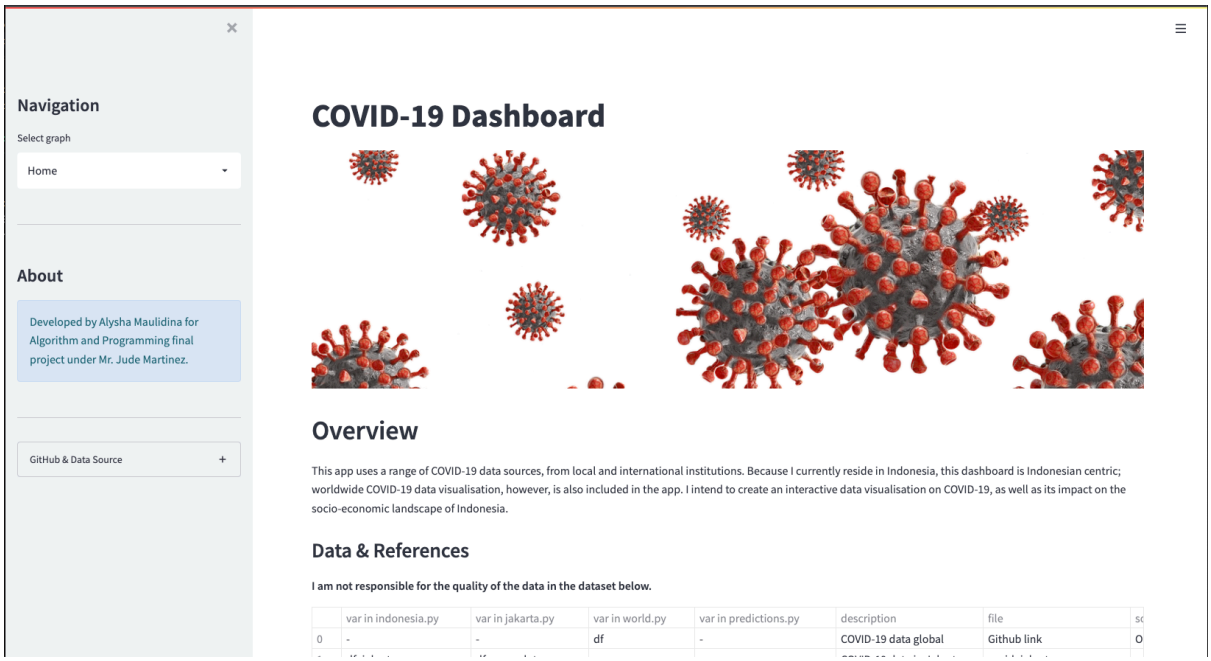


Figure 28: Homepage

II. COVID-19 in Indonesia - indonesia.py

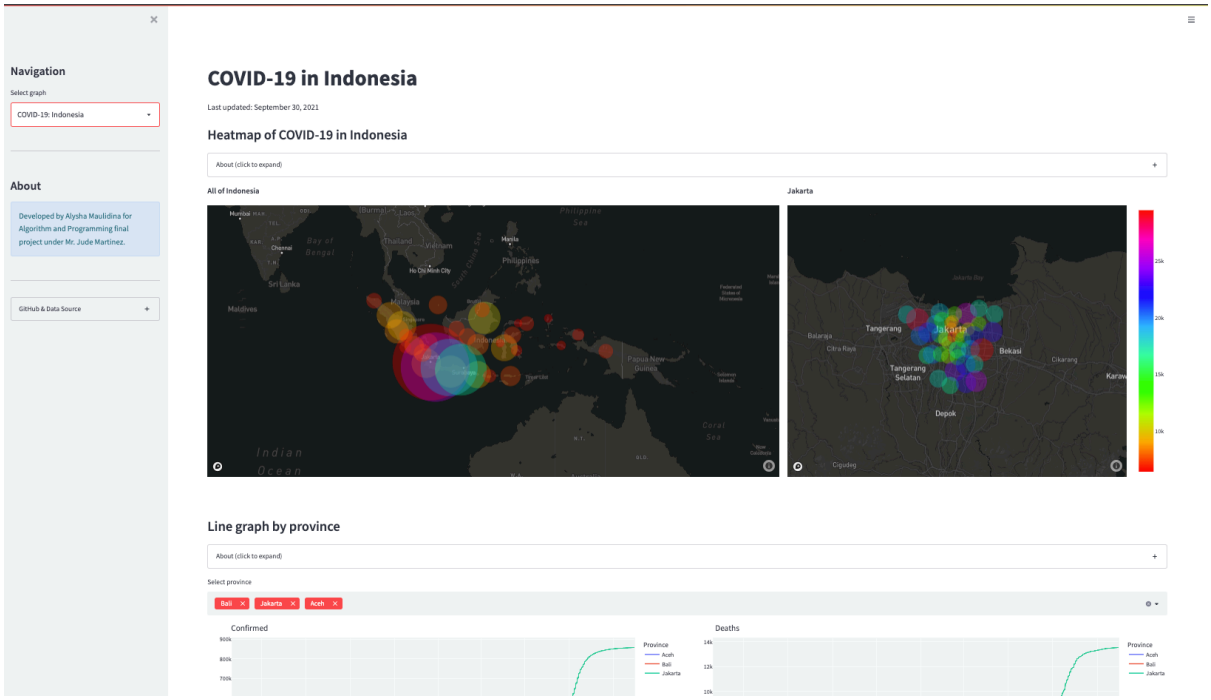


Figure 29: Heatmaps

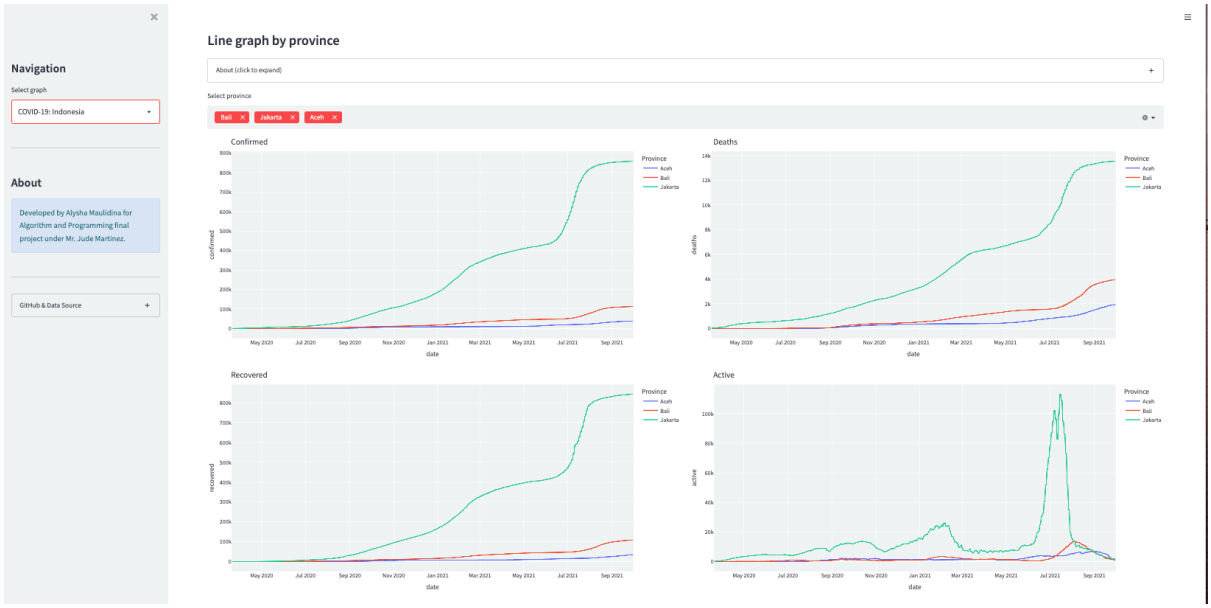


Figure 30: Line graphs of different data points for the selected regions, from the multi select widget

Figure 31a and 31b: Cross plot of different time points. The x and y axis plot independent data variables and the size of the scatter plots plot dependent data variables, which could be used to demonstrate how the former might be affected by the latter, e.g. how age might be a risk factor in COVID infections.

III. COVID-19 in Jakarta - jakarta.py

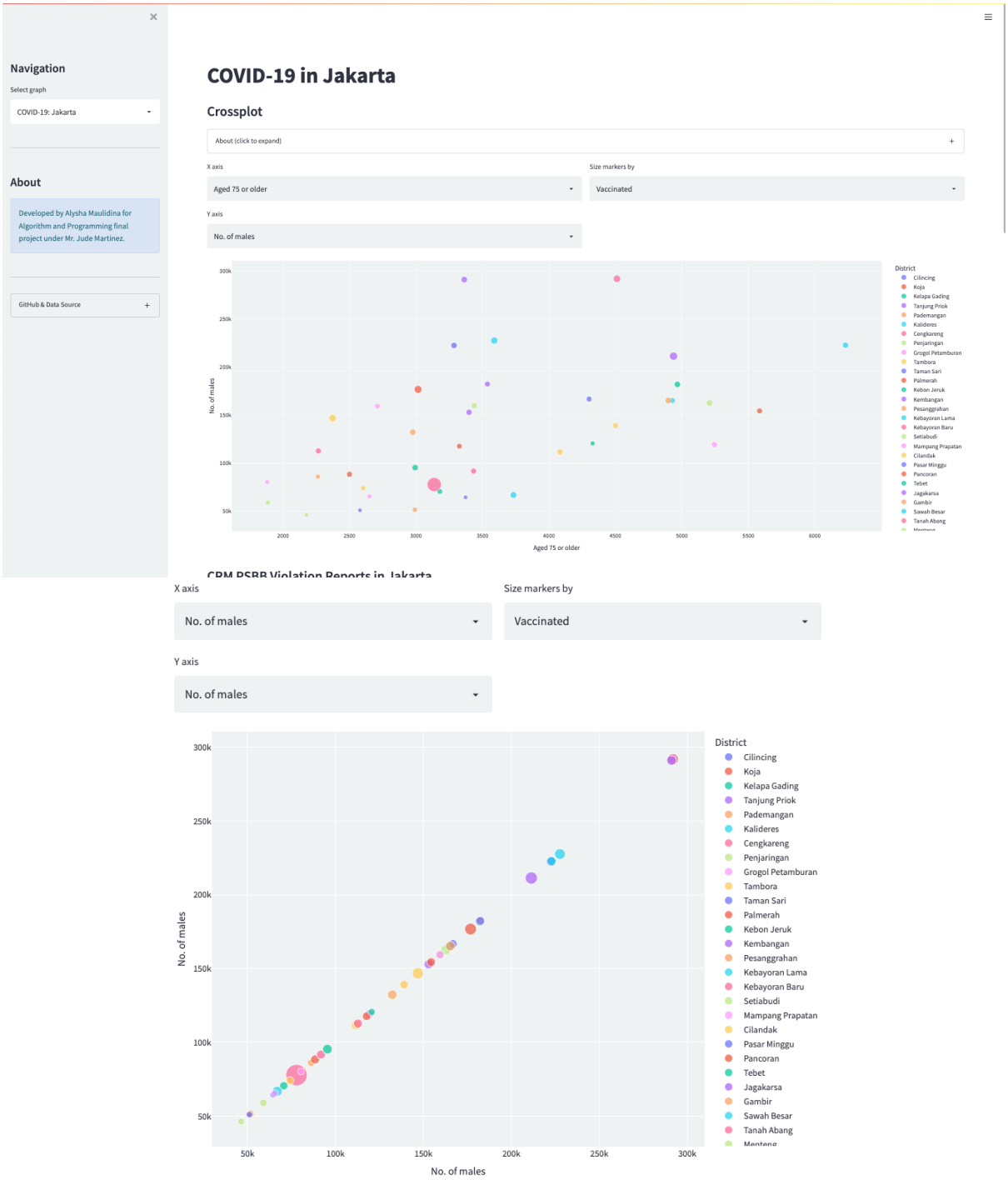


Figure 32a and 32b: Cross plot with different axes

Figure 33: Extruding map and histogram

Figure 34: Multi dimensional bar graph

IV. COVID-19 Worldwide - world.py

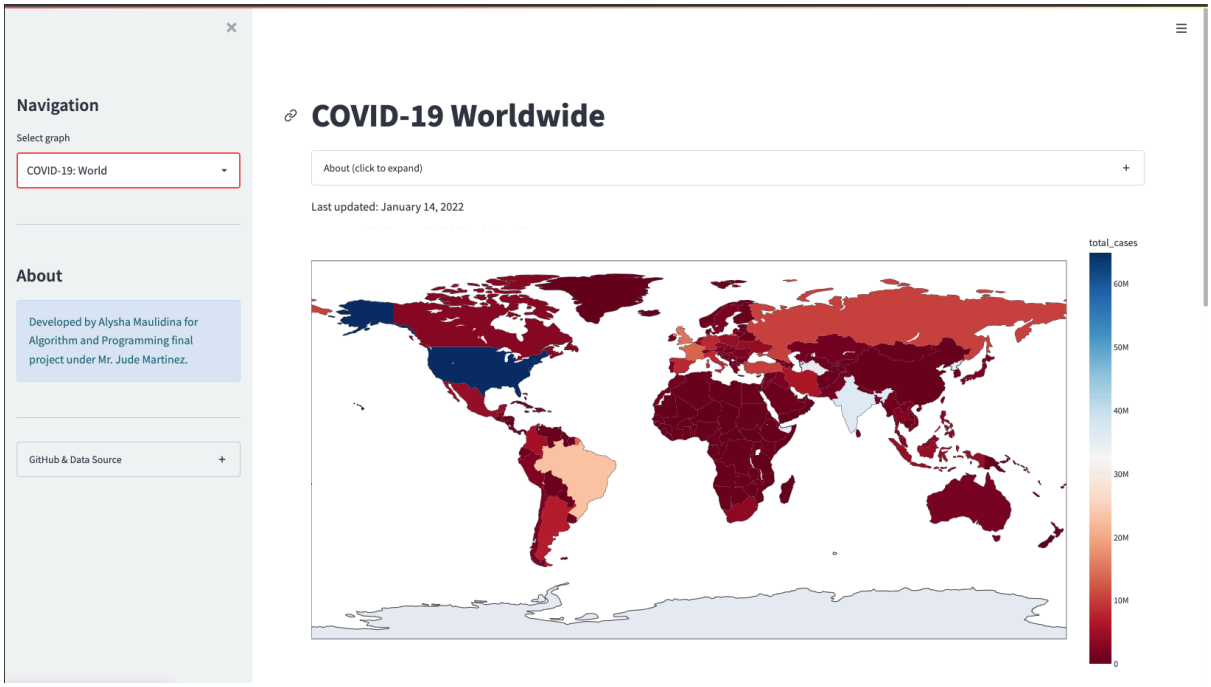


Figure 35: Chloropleth map of confirmed cases in the world

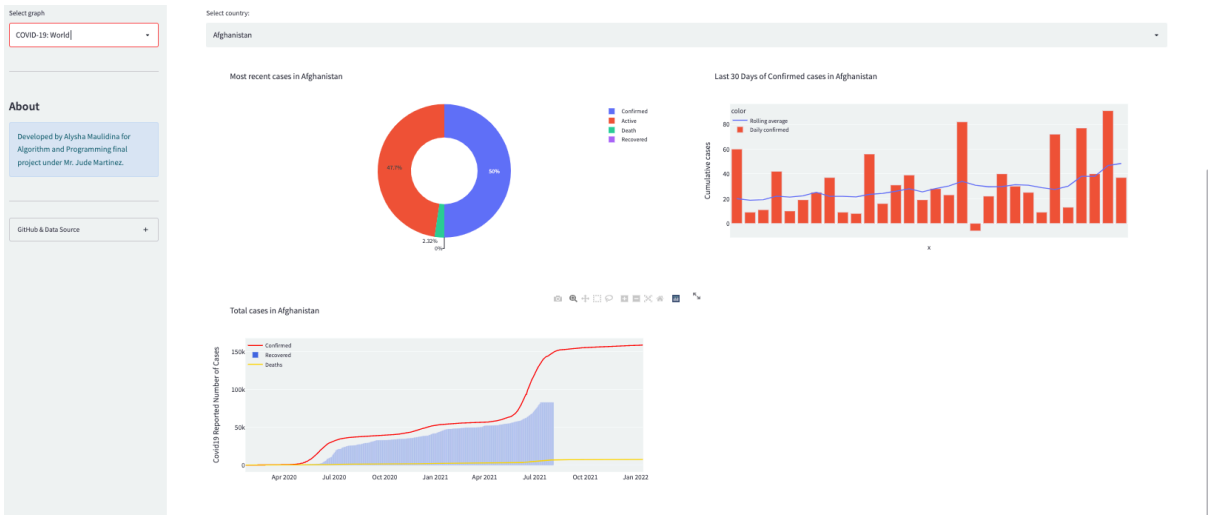


Figure 36: Different chart types showcasing data for Afghanistan - pie chart displays the most recent case, bar chart displays the rolling average of the last 30 days of confirmed cases, multiple plot chart displays the overall cases

V. COVID-19 Forecast in Indonesia - predictions.py

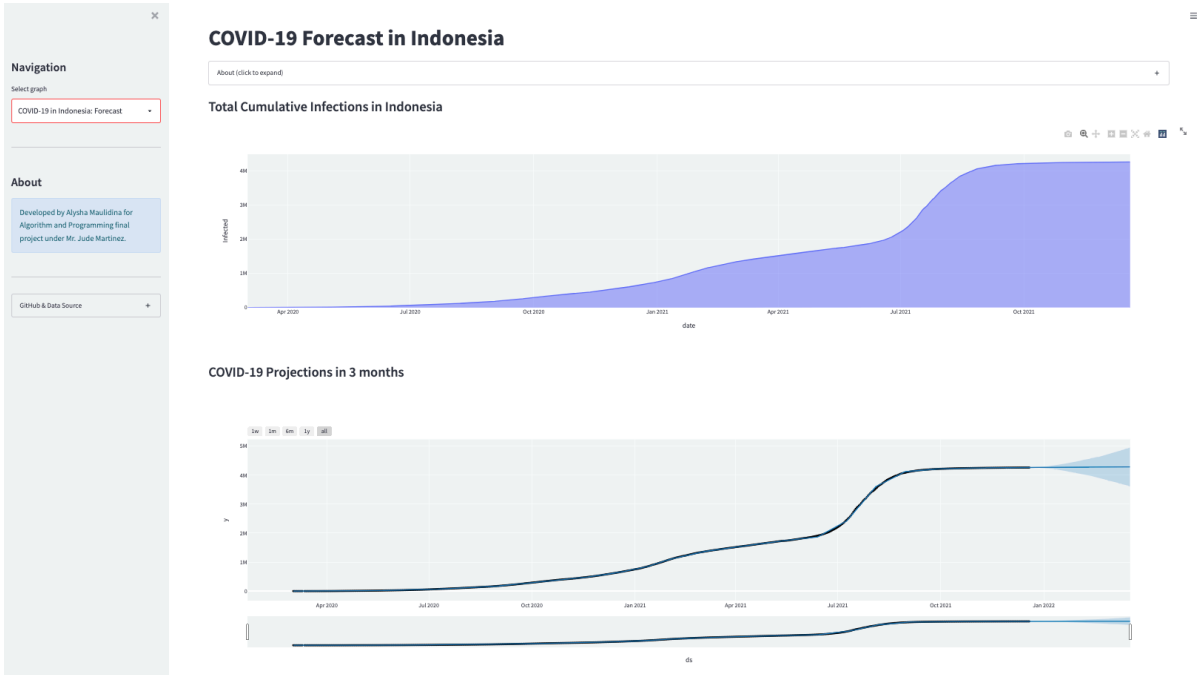


Figure 37: Area chart and line chart representing forecasted data

VI. Sidebar navigation

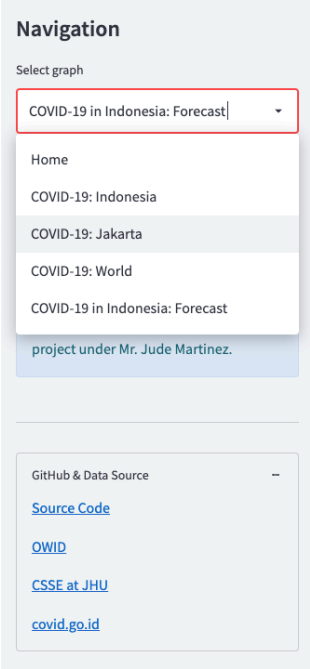


Figure 37: Side bar with drop down navigation system

References

Experimental cache primitives - Streamlit Docs. Docs.streamlit.io. (2022). Retrieved 17 January 2022, from https://docs.streamlit.io/library/advanced-features/experimental-cache-primitives.

Hwang, J. (2022). *Plotly Dash vs Streamlit—Which is the best library for building data dashboard web apps?*. Medium. Retrieved 17 January 2022, from https://towardsdatascience.com/plotly-dash-vs-streamlit-which-is-the-best-library-for-building-data-dashboard-web-apps-97d7c98b938c.

Peta Kesebaran COVID-19. Jakarta Tanggap COVID-19. (2022). Retrieved 17 January 2022, from https://corona.jakarta.go.id/id/statistik-covid-19-berbasis-kelurahan.

Robson, W. (2022). *Intro to Facebook Prophet*. Medium. Retrieved 17 January 2022, from https://medium.com/future-vision/intro-to-prophet-9d5b1cbd674e.

Nihar, P. (2022). *Building Multi Page Web App Using Streamlit*. Medium. Retrieved 17 January 2022, from https://medium.com/@u.praneel.nihar/building-multi-page-web-app-using-streamlit-7a40d55fa5b4.

Ritchie, H., Mathieu, E., Rodés-Guirao, L., Appel, C., Giattino, C., & Ortiz-Ospina, E. et al. (2022). *Coronavirus Pandemic (COVID-19)*. Our World in Data. Retrieved 17 January 2022, from https://ourworldindata.org/coronavirus#citation.

Dong, E., H, D., & Gardner, L. (2022). *GitHub - CSSEGISandData/COVID-19: Novel Coronavirus (COVID-19) Cases, provided by JHU CSSE*. GitHub. Retrieved 17 January 2022, from https://github.com/CSSEGISandData/COVID-19.