final_stage_v

November 22, 2024

```python
[1]: import pandas as pd
     import numpy as np
     import plotly.graph_objs as go
     import dash
     from dash import dcc, html
     from dash.dependencies import Input, Output
     from sklearn.linear_model import LinearRegression
     from sklearn.preprocessing import PolynomialFeatures
```

```python
[3]: # Load dataset for cases
     df_cases = pd.read_csv('covid-19 data/covid_confirmed_usafacts.csv')

     # Read case dataset
     df_cases.head()

     # Load dataset for deaths
     df_deaths = pd.read_csv('covid-19 data/covid_deaths_usafacts.csv')

     # Read deaths dataset
     df_deaths.head()
```

```
[3]:    countyFIPS           County Name  State  StateFIPS  2020-01-22  2020-01-23  \
     0           0  Statewide Unallocated     AL          1           0           0
     1        1001         Autauga County     AL          1           0           0
     2        1003         Baldwin County     AL          1           0           0
     3        1005         Barbour County     AL          1           0           0
     4        1007            Bibb County     AL          1           0           0

        2020-01-24  2020-01-25  2020-01-26  2020-01-27  ...  2023-07-14  \
     0           0           0           0           0  ...           0
     1           0           0           0           0  ...         235
     2           0           0           0           0  ...         731
     3           0           0           0           0  ...         104
     4           0           0           0           0  ...         111

        2023-07-15  2023-07-16  2023-07-17  2023-07-18  2023-07-19  2023-07-20  \
     0           0           0           0           0           0           0
     1         235         235         235         235         235         235
```

```
2         731       731        731       731        731        731
3         104       104        104       104        104        104
4         111       111        111       111        111        111

      2023-07-21  2023-07-22  2023-07-23
0          0           0           0
1        235         235         235
2        731         731         731
3        104         104         104
4        111         111         111

[5 rows x 1269 columns]
```

```python
# cleaning and formatting the dataset for cases
selected_columns = [col for col in df_cases.columns if '2020-01-22' <= col <=
                    '2023-07-23' or col == 'State']
df_cases = df_cases[selected_columns]
df_cases.head()

# cleaning and formatting the dataset for deaths
selected_columns = [col for col in df_deaths.columns if '2020-01-22' <= col <=
                    '2023-07-23' or col == 'State']
df_deaths = df_deaths[selected_columns]
df_deaths.head()
```

```
[5]:   State  2020-01-22  2020-01-23  2020-01-24  2020-01-25  2020-01-26  \
0        AL           0           0           0           0           0
1        AL           0           0           0           0           0
2        AL           0           0           0           0           0
3        AL           0           0           0           0           0
4        AL           0           0           0           0           0

      2020-01-27  2020-01-28  2020-01-29  2020-01-30  ...  2023-07-14  \
0          0           0           0           0    ...        0
1          0           0           0           0    ...      235
2          0           0           0           0    ...      731
3          0           0           0           0    ...      104
4          0           0           0           0    ...      111

      2023-07-15  2023-07-16  2023-07-17  2023-07-18  2023-07-19  2023-07-20  \
0          0           0           0           0           0           0
1        235         235         235         235         235         235
2        731         731         731         731         731         731
3        104         104         104         104         104         104
4        111         111         111         111         111         111

      2023-07-21  2023-07-22  2023-07-23
```

```
0           0           0           0
1         235         235         235
2         731         731         731
3         104         104         104
4         111         111         111

[5 rows x 1266 columns]
```

```python
[7]:  dropdown_options = []
      states = df_cases['State'].unique()

      for state in states:
          option = { 'label': state, 'value': state }
          dropdown_options.append(option)

      cases_long = pd.melt(df_cases, id_vars=['State'], var_name='Date',
                    value_name='Cases')
      cases_long['Date'] = pd.to_datetime(cases_long['Date'])

      deaths_long = pd.melt(df_deaths, id_vars=['State'], var_name='Date',
                    value_name='Deaths')
      deaths_long['Date'] = pd.to_datetime(deaths_long['Date'])
```

```python
[9]:  import pandas as pd
      import numpy as np
      from dash import Dash, html, dcc, Input, Output
      from sklearn.preprocessing import PolynomialFeatures
      from sklearn.linear_model import LinearRegression
      import plotly.graph_objects as go

      # Load data
      cases = pd.read_csv('covid-19 data/covid_confirmed_usafacts.csv')
      deaths = pd.read_csv('covid-19 data/covid_deaths_usafacts.csv')

      dropdown_options = [{'label': state, 'value': state} for state in
                    sorted(cases['State'].unique())]

      # Reusable component creation functions
      def create_header(title):
          return html.H1(title, style={'textAlign': 'center', 'color': '#2F4F4F',
                    'margin-bottom': '20px'})

      def create_date_picker(id):
          return dcc.DatePickerRange(
              id=id,
              start_date=pd.to_datetime('2020-01-22'),
              end_date=pd.to_datetime('2023-07-23'),
```

```python
            display_format='YYYY-MM-DD',
            min_date_allowed=pd.to_datetime('2020-01-22'),
            max_date_allowed=pd.to_datetime('2023-07-23'),
            style={'width': '300px', 'margin': '0 auto'}
        )

def create_radio_items(id):
    return dcc.RadioItems(
        id=id,
        options=[
            {'label': 'Linear Scale', 'value': 'linear'},
            {'label': 'Logarithmic Scale', 'value': 'log'},
        ],
        value='linear',
        labelStyle={'margin': '10px'},
        style={'display': 'flex', 'justify-content': 'center', 'align-items':
 'center'}
    )

def create_dropdown(id):
    return dcc.Dropdown(
        id=id,
        options=dropdown_options,
        multi=True,
        value=[],
        style={'width': '150px', 'margin': '0 auto'}
    )

def create_graph(id):
    return dcc.Graph(id=id, style={
        'border': '1px solid #ccc',
        'border-radius': '10px',
        'box-shadow': '2px 2px 10px rgba(0, 0, 0, 0.1)',
        'padding': '10px',
        'margin': '0 auto'
    })

# Initialize Dash app
mainDashboard = Dash(__name__)

mainDashboard.layout = html.Div([
    create_header("COVID-19 Cases Dashboard"),
    create_date_picker('cases-date-picker-range'),
    create_radio_items('cases-y-axis-scale'),
    create_dropdown('cases-state-selection'),
    create_graph('cases-graph'),
    create_header("COVID-19 Deaths Dashboard"),
```

```python
    create_date_picker('deaths-date-picker-range'),
    create_radio_items('deaths-y-axis-scale'),
    create_dropdown('deaths-state-selection'),
    create_graph('deaths-graph'),
])

# Reusable callback logic
def create_graph_callback(data_long, value_col):
    def callback(start_date, end_date, y_axis_scale, selected_states):
        data_long['Date'] = pd.to_datetime(data_long['Date'])
        filtered_data = data_long[(data_long['Date'] >= pd.
 to_datetime(start_date)) &
                                  (data_long['Date'] <= pd.
 to_datetime(end_date))]

        if selected_states:
            filtered_data = filtered_data[filtered_data['State'].
 isin(selected_states)]

        daily_data = filtered_data.groupby('Date').sum()[value_col]

        # Polynomial regression predictions
        days = np.arange(len(daily_data))
        poly = PolynomialFeatures(degree=4)
        X_poly = poly.fit_transform(days.reshape(-1, 1))
        poly_reg = LinearRegression()
        poly_reg.fit(X_poly, daily_data)
        poly_predictions = poly_reg.predict(X_poly).clip(min=0)

        # Linear regression predictions
        linear_reg = LinearRegression()
        linear_reg.fit(days.reshape(-1, 1), daily_data)
        linear_predictions = linear_reg.predict(days.reshape(-1, 1))

        # Create figure
        fig = go.Figure()
        fig.add_trace(go.Scatter(x=daily_data.index, y=daily_data,
 mode='lines', name='Actual'))
        fig.add_trace(go.Scatter(x=daily_data.index, y=poly_predictions,
 mode='lines', name='Non-Linear Prediction', line={'dash': 'dash'}))
        fig.add_trace(go.Scatter(x=daily_data.index, y=linear_predictions,
 mode='lines', name='Linear Prediction', line={'dash': 'dash'}))
        fig.add_trace(go.Scatter(x=daily_data.index, y=daily_data.
 rolling(window=7).mean(), mode='lines', name='7-Day Moving Avg',
 line={'dash': 'dot'}))
```

```python
        # State-specific data
        for state in selected_states:
            state_data = filtered_data[filtered_data['State'] == state].
↪groupby('Date').sum()
            y_values = np.log(state_data[value_col] + 1) if y_axis_scale ==
↪'log' else state_data[value_col]
            fig.add_trace(go.Scatter(x=state_data.index, y=y_values,
↪mode='lines', name=f'{state}'))

        # Update layout
        fig.update_layout(
            title=f'Actual and Predicted COVID-19 {value_col.capitalize()}',
            xaxis_title='Date',
            yaxis_title=f'Number of {value_col.capitalize()}',
            yaxis_type='log' if y_axis_scale == 'log' else 'linear'
        )
        return fig
    return callback

# Set callbacks
mainDashboard.callback(
    Output('cases-graph', 'figure'),
    [Input('cases-date-picker-range', 'start_date'),
     Input('cases-date-picker-range', 'end_date'),
     Input('cases-y-axis-scale', 'value'),
     Input('cases-state-selection', 'value')]
)(create_graph_callback(cases_long, 'Cases'))

mainDashboard.callback(
    Output('deaths-graph', 'figure'),
    [Input('deaths-date-picker-range', 'start_date'),
     Input('deaths-date-picker-range', 'end_date'),
     Input('deaths-y-axis-scale', 'value'),
     Input('deaths-state-selection', 'value')]
)(create_graph_callback(deaths_long, 'Deaths'))

# Run server
def start_server(app, mode='inline', port=8050):
    app.run_server(mode=mode, port=port)

if __name__ == '__main__':
    start_server(mainDashboard)
```
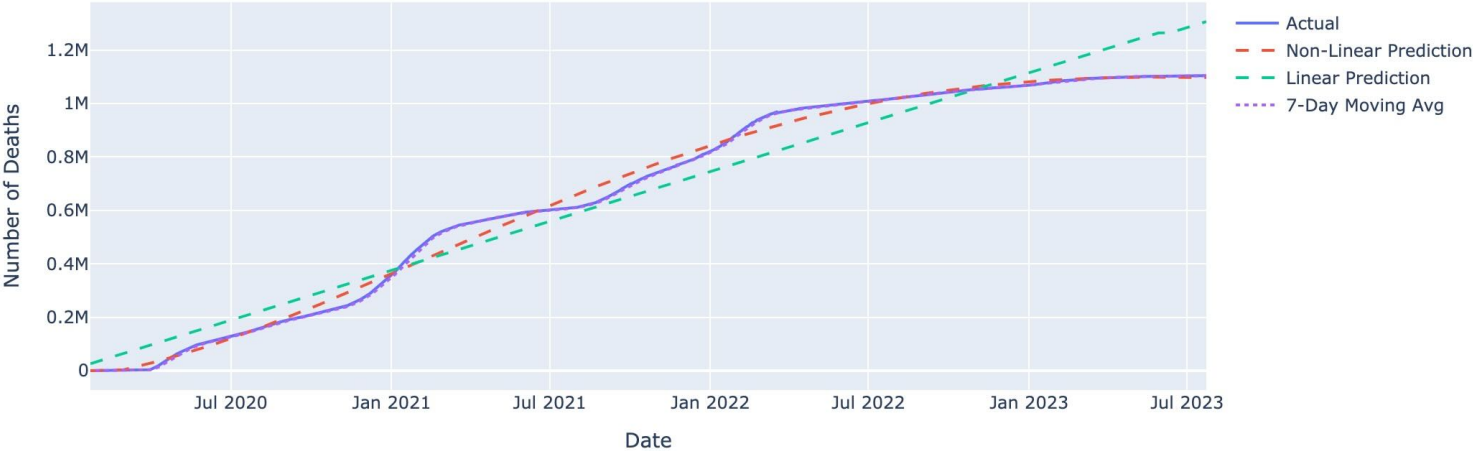
<IPython.lib.display.IFrame at 0x13cfb5fd0>

COVID-19 Deaths Dashboard

2020-01-22 → 2023-07-23

◉ Linear Scale   ○ Logarithmic Scale

Select... ▾

Actual and Predicted COVID-19 Deaths

Legend:
- Actual
- Non-Linear Prediction
- Linear Prediction
- 7-Day Moving Avg

COVID-19 Deaths Dashboard

2020-01-22 → 2023-07-23

○ Linear Scale   ○ Logarithmic Scale

× FL
× NC

Actual and Predicted COVID-19 Deaths

Legend:
- Actual
- Non-Linear Prediction
- Linear Prediction
- 7-Day Moving Avg
- FL
- NC

Y-axis: Number of Deaths (0, 50k, 100k)
X-axis: Date (Jul 2020, Jan 2021, Jul 2021, Jan 2022, Jul 2022, Jan 2023, Jul 2023)
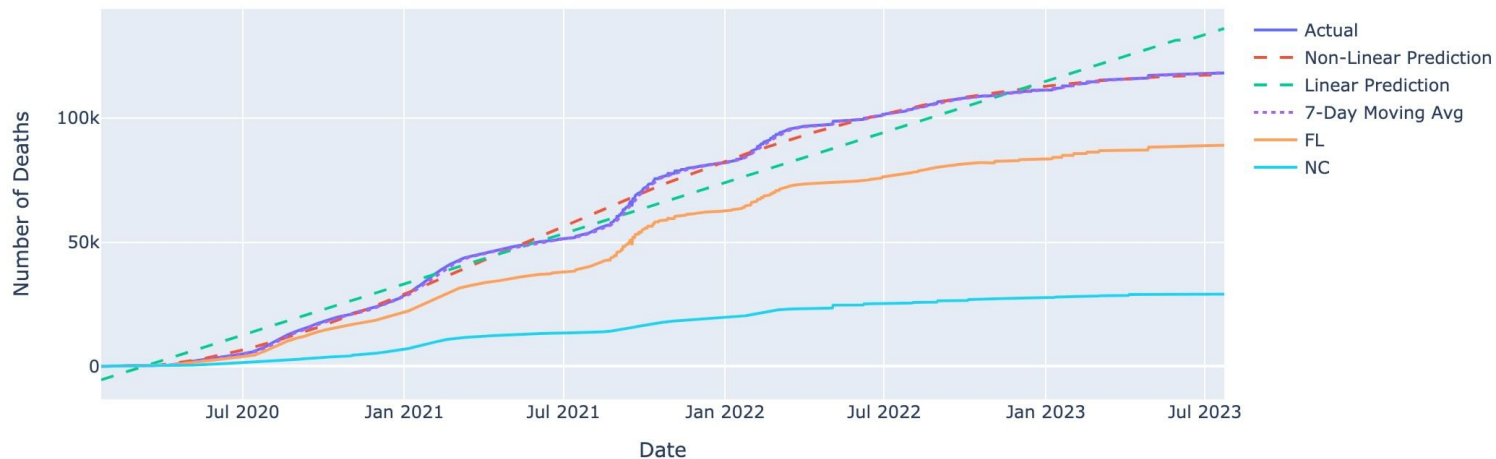
COVID-19 Cases Dashboard

2020-01-22 → 2023-07-23

○ Linear Scale   ○ Logarithmic Scale

× FL
× NC

Actual and Predicted COVID-19 Cases

Legend:
- Actual
- Non-Linear Prediction
- Linear Prediction
- 7-Day Moving Avg
- FL
- NC

Y-axis: Number of Cases (0, 5M, 10M)
X-axis: Date (Jul 2020, Jan 2021, Jul 2021, Jan 2022, Jul 2022, Jan 2023, Jul 2023)