testing-healthcare.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

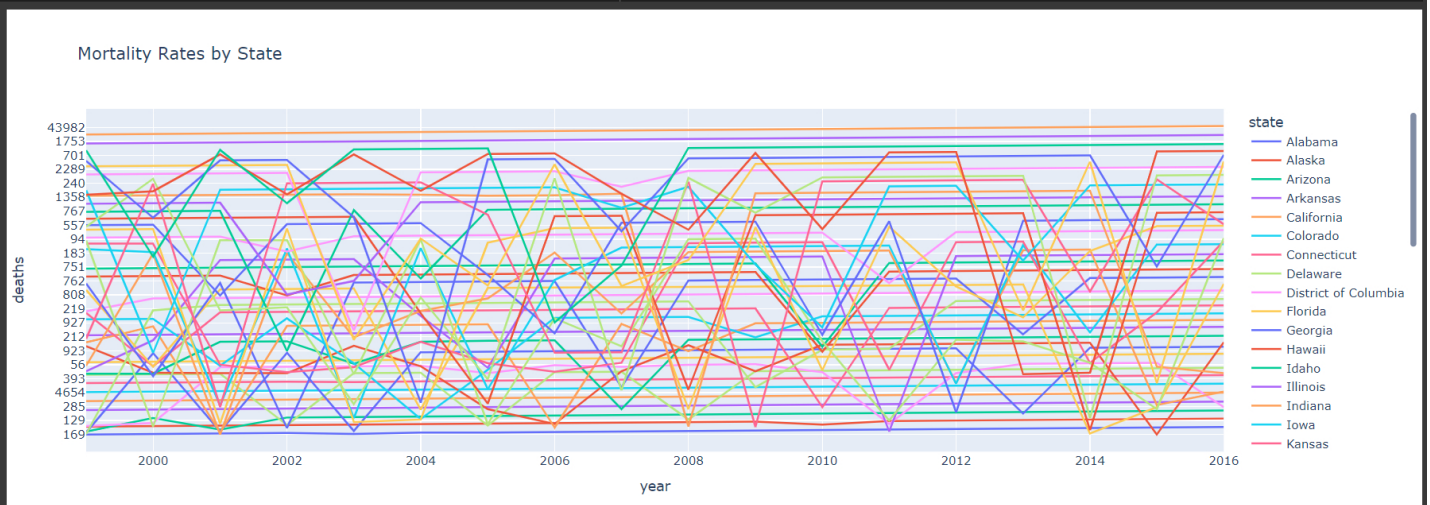
RAM Disk Gemini

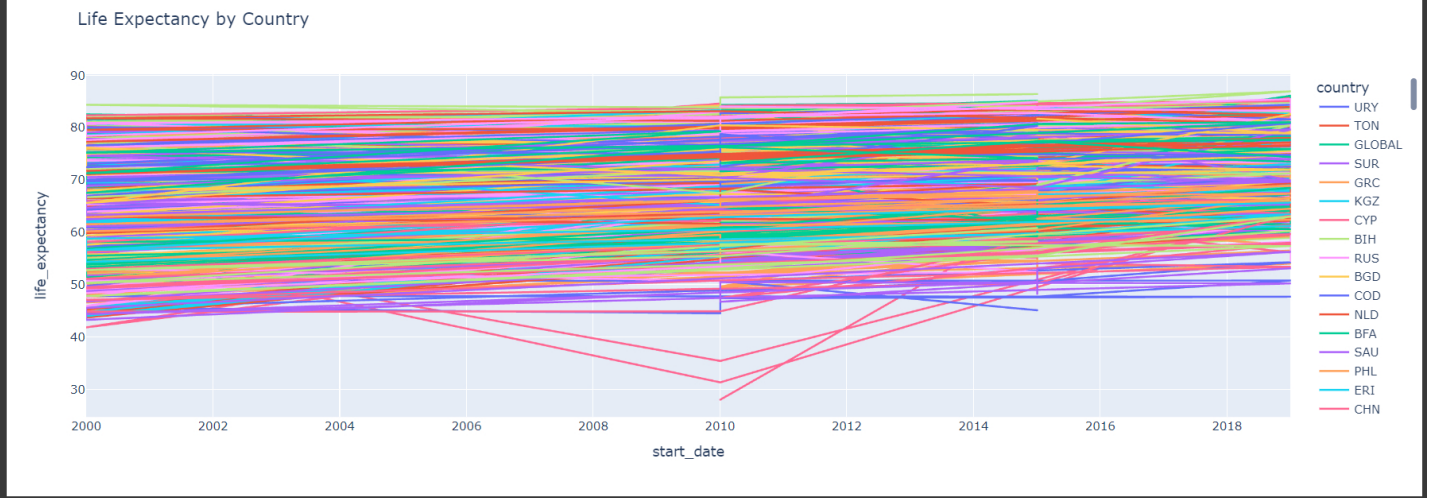
[7]
```python
# Import libraries
import plotly.express as px
import pandas as pd
import geopandas as gpd
import requests

# Define CDC API endpoints
CDC_BASE_URL = "https://data.cdc.gov/resource/"
CDC_API_ENDPOINTS = {
    "mortality": "xbxb-epbu.json",  # Example endpoint for mortality data
    "chronic_diseases": "5h56-n989.json",  # Example endpoint for chronic diseases
}

# Define WHO API endpoints
WHO_BASE_URL = "https://ghoapi.azureedge.net/api/"
WHO_API_ENDPOINTS = {
    "life_expectancy": "WHOSIS_000001",  # Example endpoint for life expectancy
}

# Function to fetch data from CDC API
def fetch_cdc_data(endpoint, params=None):
    url = f"{CDC_BASE_URL}{endpoint}"
    response = requests.get(url, params=params)
    response.raise_for_status()  # Raise an exception for HTTP errors
    return pd.DataFrame(response.json())

# Function to fetch data from WHO API
def fetch_who_data(endpoint, params=None):
    url = f"{WHO_BASE_URL}{endpoint}"
    response = requests.get(url, params=params)
    response.raise_for_status()  # Raise an exception for HTTP errors
    return pd.DataFrame(response.json()['value'])

# Fetching metadata from CDC mortality data endpoint
cdc_mortality_metadata = fetch_cdc_data(CDC_API_ENDPOINTS["mortality"], {"$limit": 1})
print("CDC Mortality Metadata:")
print(cdc_mortality_metadata.columns)
print(cdc_mortality_metadata.dtypes)

# Fetching metadata from WHO life expectancy data endpoint
who_life_expectancy_metadata = fetch_who_data(WHO_API_ENDPOINTS["life_expectancy"], {"$top": 1})
print("WHO Life Expectancy Metadata:")
print(who_life_expectancy_metadata.columns)
print(who_life_expectancy_metadata.dtypes)

# Fetch actual data
cdc_mortality_data = fetch_cdc_data(CDC_API_ENDPOINTS["mortality"])
who_life_expectancy_data = fetch_who_data(WHO_API_ENDPOINTS["life_expectancy"])

# Example data cleaning steps for CDC mortality data
def clean_cdc_data(df):
    # Convert 'year' to datetime
    df['year'] = pd.to_datetime(df['year'], format='%Y')
    # Filter out unwanted columns
    df = df[['year', 'state', 'sex', 'age_group', 'race_and_hispanic_origin', 'deaths', 'age_adjusted_rate']]
    # Handle missing values
    df.dropna(inplace=True)
    return df

cdc_mortality_data_cleaned = clean_cdc_data(cdc_mortality_data)
print("Cleaned CDC Mortality Data:")
print(cdc_mortality_data_cleaned.head())

# Example data cleaning steps for WHO life expectancy data
def clean_who_data(df):
    # Convert 'TimeDimensionBegin' and 'TimeDimensionEnd' to datetime
    df['TimeDimensionBegin'] = pd.to_datetime(df['TimeDimensionBegin'])
    df['TimeDimensionEnd'] = pd.to_datetime(df['TimeDimensionEnd'])
    # Filter out unwanted columns
    df = df[['TimeDimensionBegin', 'TimeDimensionEnd', 'SpatialDim', 'Dim1', 'NumericValue']]
    df = df.rename(columns={'TimeDimensionBegin': 'start_date', 'TimeDimensionEnd': 'end_date', 'SpatialDim': 'country', 'Dim1': 'sex', 'NumericValue': 'life_expectancy'})
    # Handle missing values
    df.dropna(inplace=True)
    return df

who_life_expectancy_data_cleaned = clean_who_data(who_life_expectancy_data)
print("Cleaned WHO Life Expectancy Data:")
print(who_life_expectancy_data_cleaned.head())
```

```
CDC Mortality Metadata:
Index(['state', 'year', 'sex', 'age_group', 'race_and_hispanic_origin',
       'deaths', 'population', 'crude_death_rate',
       'standard_error_for_crude_rate',
       'lower_confidence_limit_for_crude_rate',
       'upper_confidence_limit_for_crude_rate', 'age_adjusted_rate',
       'standard_error_for_age_adjusted_rate',
       'lower_confidence_limit_for_age_adjusted_rate',
       'upper_confidence_limit_for_age_adjusted_rate',
       'state_crude_rate_in_range', 'us_crude_rate', 'us_age_adjusted_rate',
       'unit'],
      dtype='object')
state                                   object
year                                    object
sex                                     object
age_group                               object
race_and_hispanic_origin                object
deaths                                  object
population                              object
crude_death_rate                        object
standard_error_for_crude_rate           object
```

```
lower_confidence_limit_for_crude_rate           object
upper_confidence_limit_for_crude_rate           object
age_adjusted_rate                               object
standard_error_for_age_adjusted_rate            object
lower_confidence_limit_for_age_adjusted_rate    object
upper_confidence_limit_for_age_adjusted_rate    object
state_crude_rate_in_range                       object
us_crude_rate                                   object
us_age_adjusted_rate                            object
unit                                            object
dtype: object
WHO Life Expectancy Metadata:
Index(['Id', 'IndicatorCode', 'SpatialDimType', 'SpatialDim', 'TimeDimType',
       'ParentLocationCode', 'ParentLocation', 'Dim1Type', 'Dim1', 'TimeDim',
       'Dim2Type', 'Dim2', 'Dim3Type', 'Dim3', 'DataSourceDimType',
       'DataSourceDim', 'Value', 'NumericValue', 'Low', 'High', 'Comments',
       'Date', 'TimeDimensionValue', 'TimeDimensionBegin', 'TimeDimensionEnd'],
      dtype='object')
Id                      int64
IndicatorCode          object
SpatialDimType         object
SpatialDim             object
TimeDimType            object
ParentLocationCode     object
ParentLocation         object
Dim1Type               object
Dim1                   object
TimeDim                 int64
Dim2Type               object
Dim2                   object
Dim3Type               object
Dim3                   object
DataSourceDimType      object
DataSourceDim          object
Value                  object
NumericValue          float64
Low                    object
```

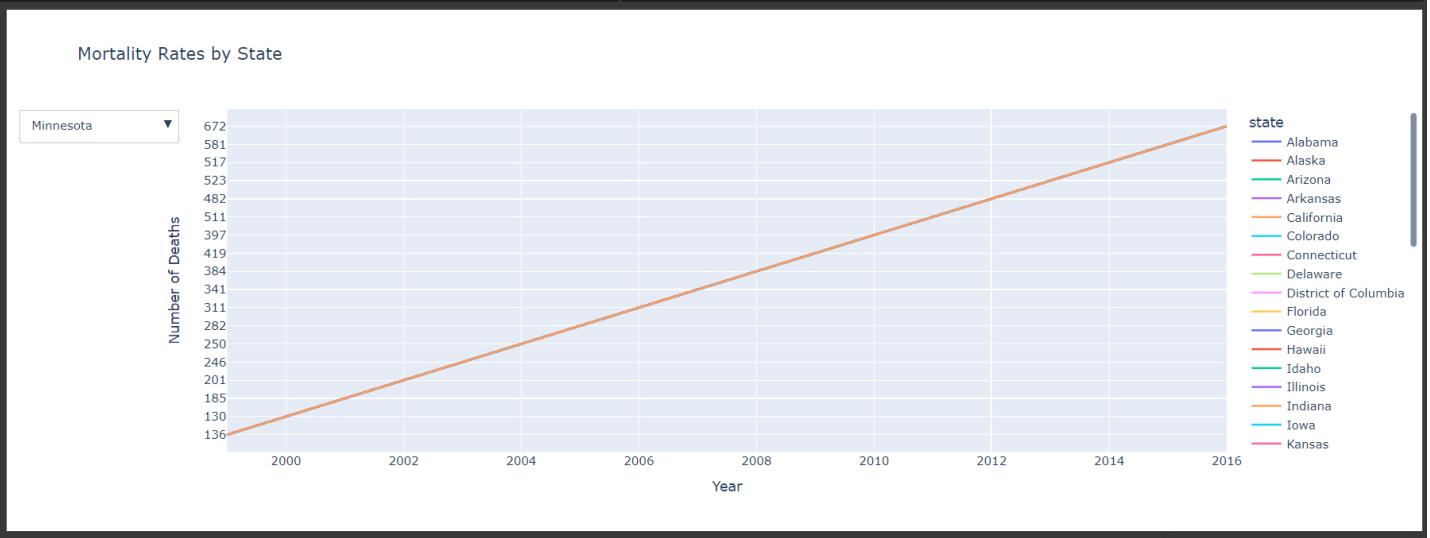
```
# Plot CDC mortality data
fig = px.line(cdc_mortality_data_cleaned, x='year', y='deaths', color='state', title='Mortality Rates by State')
fig.show()

# Plot WHO life expectancy data
fig = px.line(who_life_expectancy_data_cleaned, x='start_date', y='life_expectancy', color='country', title='Life Expectancy by Country')
fig.show()
```



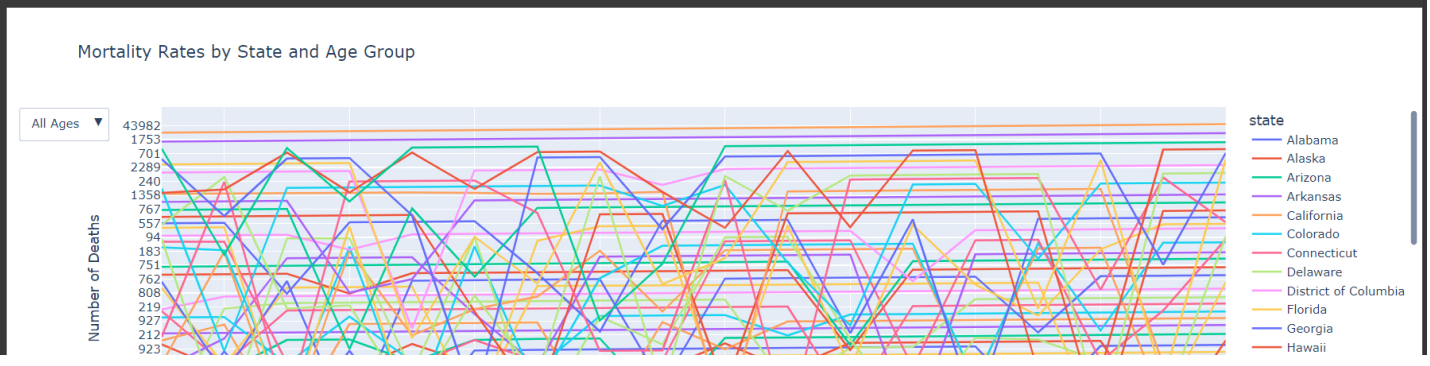



```
# Enhanced interactive plot for CDC mortality data
fig = px.line(
    cdc_mortality_data_cleaned,
    x='year',
    y='deaths',
```

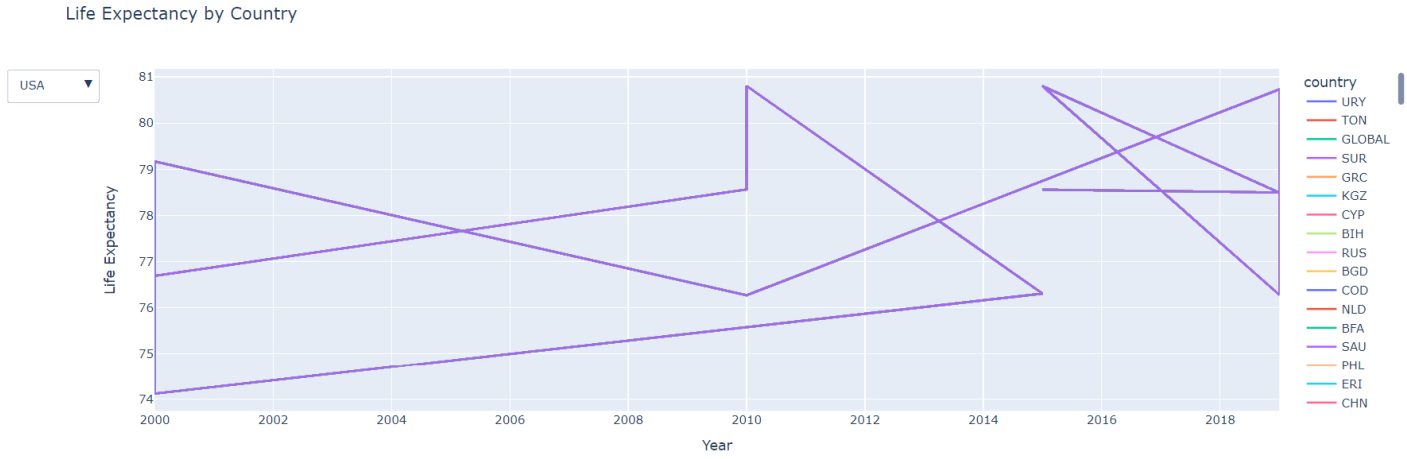
```
    color='state',
    title='Mortality Rates by State',
    labels={'deaths': 'Number of Deaths', 'year': 'Year'}
)
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=[{"y": [cdc_mortality_data_cleaned[cdc_mortality_data_cleaned["state"] == state]["deaths"] for state in cdc_mortality_data_cleaned["state"].unique()],
                           "x": [cdc_mortality_data_cleaned[cdc_mortality_data_cleaned["state"] == state]["year"] for state in cdc_mortality_data_cleaned["state"].unique()],
                           "type": "line"}],
                    label="All States",
                    method="restyle"
                )
            ] + [
                dict(
                    args=[{"y": [cdc_mortality_data_cleaned[cdc_mortality_data_cleaned["state"] == state]["deaths"]],
                           "x": [cdc_mortality_data_cleaned[cdc_mortality_data_cleaned["state"] == state]["year"]],
                           "type": "line"}],
                    label=state,
                    method="restyle"
                ) for state in cdc_mortality_data_cleaned["state"].unique()
            ])
        )
    ]
)
fig.show()
```



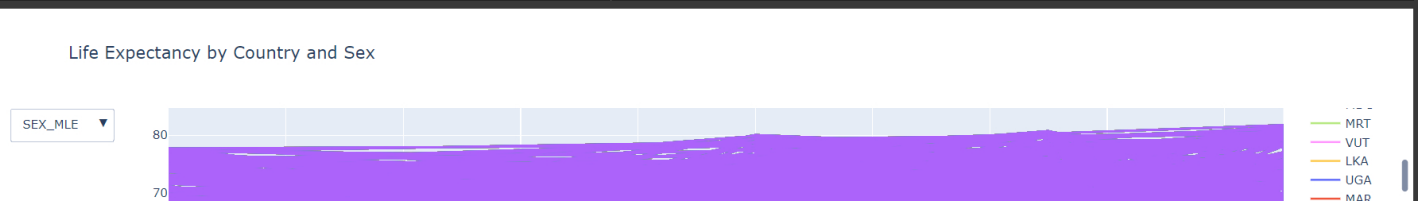

```
# Enhanced interactive plot with age group filter
fig = px.line(
    cdc_mortality_data_cleaned,
    x='year',
    y='deaths',
    color='state',
    line_group='age_group',
    title='Mortality Rates by State and Age Group',
    labels={'deaths': 'Number of Deaths', 'year': 'Year'}
)
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=[{"y": [cdc_mortality_data_cleaned[cdc_mortality_data_cleaned["age_group"] == age]["deaths"]],
                           "x": [cdc_mortality_data_cleaned[cdc_mortality_data_cleaned["age_group"] == age]["year"]],
                           "color": cdc_mortality_data_cleaned["state"],
                           "type": "line"}],
                    label=age,
                    method="restyle"
                ) for age in cdc_mortality_data_cleaned["age_group"].unique()
            ])
        )
    ]
)
fig.show()
```

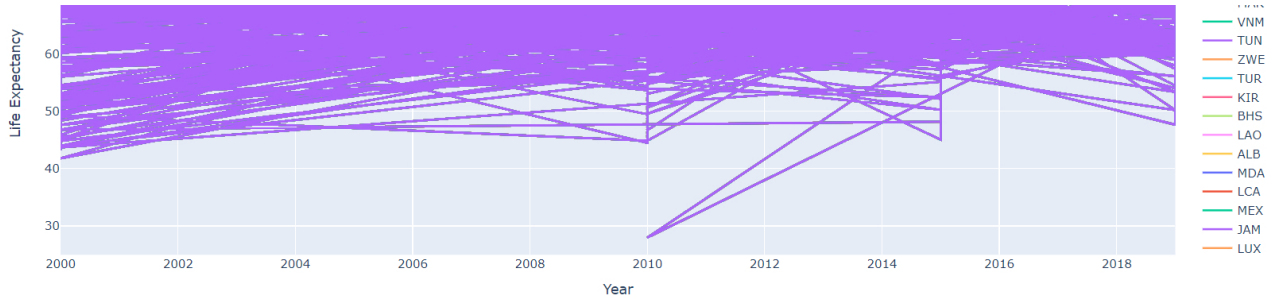


[11]
```
# Enhanced interactive plot for WHO life expectancy data
fig = px.line(
    who_life_expectancy_data_cleaned,
    x='start_date',
    y='life_expectancy',
    color='country',
    title='Life Expectancy by Country',
    labels={'life_expectancy': 'Life Expectancy', 'start_date': 'Year'}
)
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=[{"y": [who_life_expectancy_data_cleaned[who_life_expectancy_data_cleaned["country"] == country]["life_expectancy"]],
                           "x": [who_life_expectancy_data_cleaned[who_life_expectancy_data_cleaned["country"] == country]["start_date"]],
                           "type": "line"}],
                    label=country,
                    method="restyle"
                ) for country in who_life_expectancy_data_cleaned["country"].unique()
            ])
        )
    ]
)
fig.show()
```



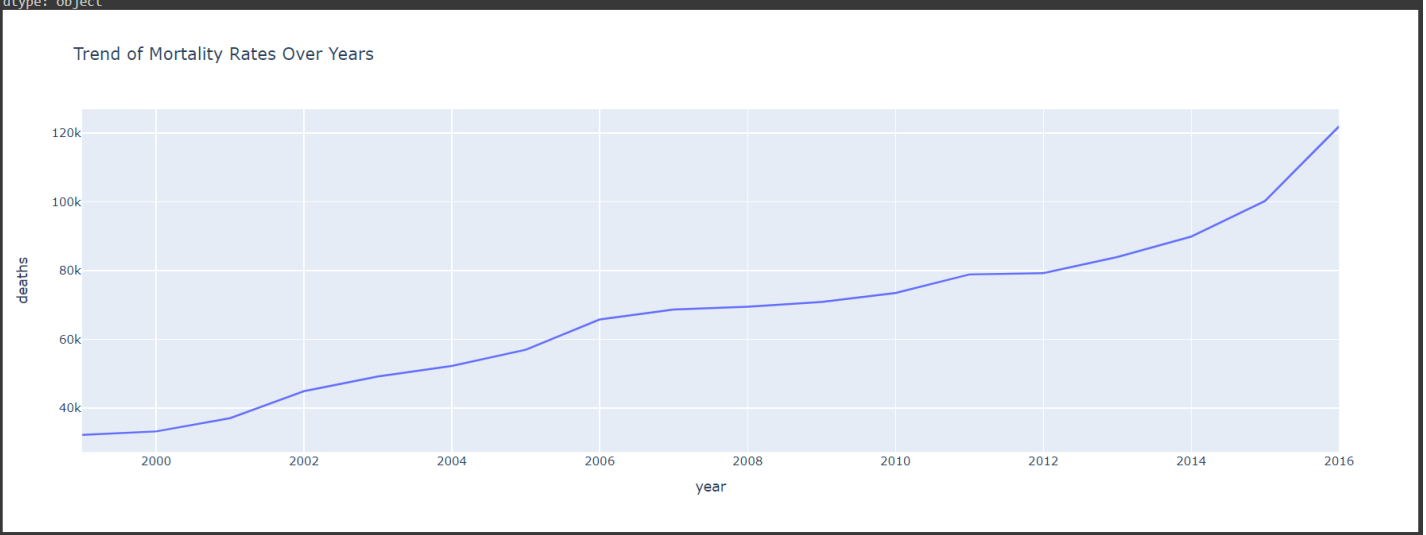
[12]
```
# Enhanced interactive plot with sex filter
fig = px.line(
    who_life_expectancy_data_cleaned,
    x='start_date',
    y='life_expectancy',
    color='country',
    line_group='sex',
    title='Life Expectancy by Country and Sex',
    labels={'life_expectancy': 'Life Expectancy', 'start_date': 'Year'}
)
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=[{"y": [who_life_expectancy_data_cleaned[who_life_expectancy_data_cleaned["sex"] == sex]["life_expectancy"]],
                           "x": [who_life_expectancy_data_cleaned[who_life_expectancy_data_cleaned["sex"] == sex]["start_date"]],
                           "color": who_life_expectancy_data_cleaned["country"],
                           "type": "line"}],
                    label=sex,
                    method="restyle"
                ) for sex in who_life_expectancy_data_cleaned["sex"].unique()
            ])
        )
    ]
)
fig.show()
```





[34]
```
# Ensure the 'year' column is in datetime format and then extract the year
cdc_mortality_data_cleaned['year'] = pd.to_datetime(cdc_mortality_data_cleaned['year'], format='%Y')
cdc_mortality_data_cleaned['year'] = cdc_mortality_data_cleaned['year'].dt.year

# Convert 'deaths' column to numeric if it's not already
cdc_mortality_data_cleaned['deaths'] = pd.to_numeric(cdc_mortality_data_cleaned['deaths'], errors='coerce')

# Check data types to ensure they are correct
print(cdc_mortality_data_cleaned.dtypes)

# Group by year and calculate the sum of deaths for each year
mortality_trend = cdc_mortality_data_cleaned.groupby('year')['deaths'].sum().reset_index()

# Plot the trend of mortality rates over the years
fig = px.line(mortality_trend, x='year', y='deaths', title='Trend of Mortality Rates Over Years')
fig.show()
```

```
year                    int32
state                  object
sex                    object
age_group              object
race_and_hispanic_origin    object
deaths                  int64
age_adjusted_rate     float64
dtype: object
```



[14]
```
# Display column names and data types in cdc_mortality_data_cleaned
print("Columns in cdc_mortality_data_cleaned:")
print(cdc_mortality_data_cleaned.columns)
print("\nData types in cdc_mortality_data_cleaned:")
print(cdc_mortality_data_cleaned.dtypes)

# Display the first few rows to understand the structure
print("\nFirst few rows of cdc_mortality_data_cleaned:")
print(cdc_mortality_data_cleaned.head())
```

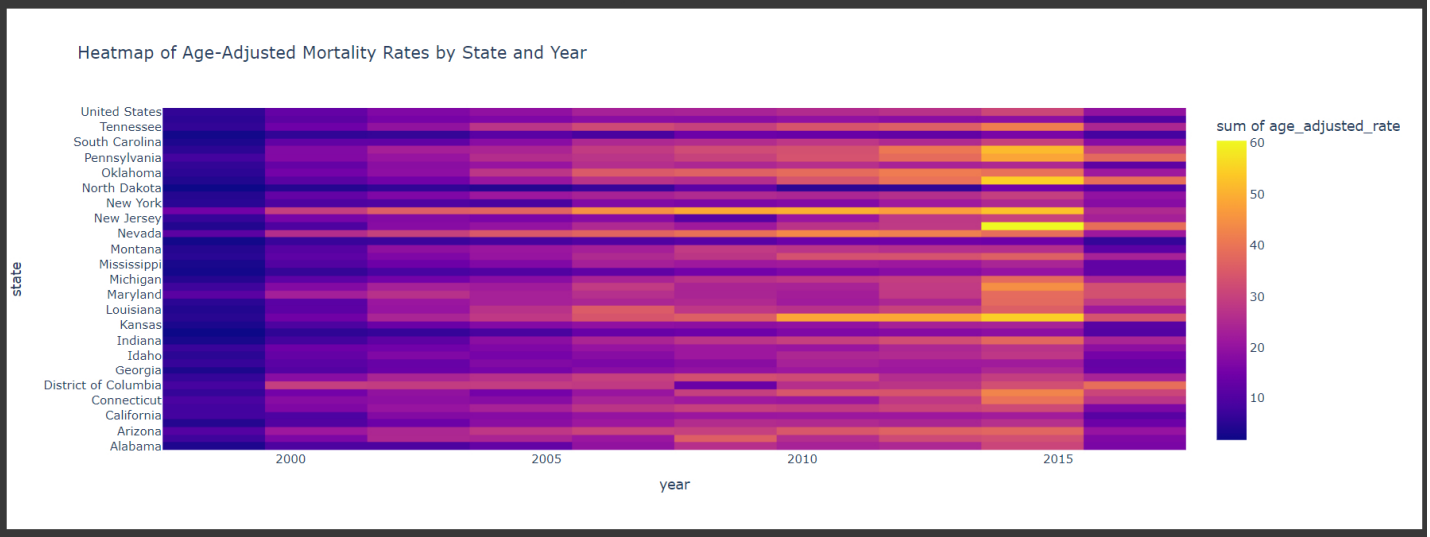
```
Columns in cdc_mortality_data_cleaned:
Index(['year', 'state', 'sex', 'age_group', 'race_and_hispanic_origin',
       'deaths', 'age_adjusted_rate'],
      dtype='object')

Data types in cdc_mortality_data_cleaned:
year                    int32
state                  object
sex                    object
age_group              object
race_and_hispanic_origin    object
deaths                  int64
age_adjusted_rate      object
dtype: object

First few rows of cdc_mortality_data_cleaned:
   year    state         sex age_group race_and_hispanic_origin  deaths  \
0  1999  Alabama  Both Sexes  All Ages    All Races-All Origins     169
1  2000  Alabama  Both Sexes  All Ages    All Races-All Origins     197
2  2001  Alabama  Both Sexes  All Ages    All Races-All Origins     216
3  2002  Alabama  Both Sexes  All Ages    All Races-All Origins     211
4  2003  Alabama  Both Sexes  All Ages    All Races-All Origins     197

  age_adjusted_rate
```

```
0           3.8521
1           4.4857
2           4.8915
3           4.7619
4           4.4333
```

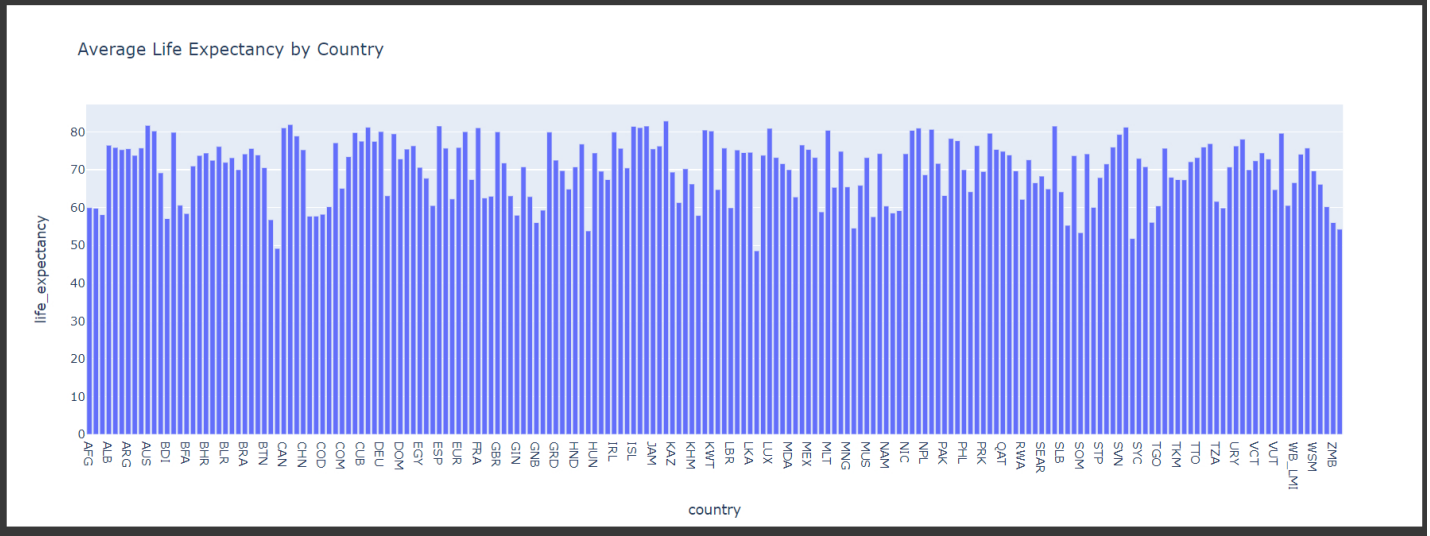
[15]
```
# Convert 'age_adjusted_rate' to numeric
cdc_mortality_data_cleaned['age_adjusted_rate'] = pd.to_numeric(cdc_mortality_data_cleaned['age_adjusted_rate'], errors='coerce')

# Drop rows with missing values in 'age_adjusted_rate' or 'deaths'
cdc_mortality_data_cleaned = cdc_mortality_data_cleaned.dropna(subset=['age_adjusted_rate', 'deaths'])

# Analyze correlation between deaths and age_adjusted_rate
correlation = cdc_mortality_data_cleaned[['age_adjusted_rate', 'deaths']].corr()
print("Correlation between age-adjusted rate and deaths:")
print(correlation)
```

```
Correlation between age-adjusted rate and deaths:
                    age_adjusted_rate    deaths
age_adjusted_rate           1.000000  0.065453
deaths                      0.065453  1.000000
<ipython-input-15-ad299beb859c>:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[16]
```
# Plot heatmap for age-adjusted mortality rates
fig = px.density_heatmap(cdc_mortality_data_cleaned, x='year', y='state', z='age_adjusted_rate',
                         title='Heatmap of Age-Adjusted Mortality Rates by State and Year')
fig.show()
```



[17]
```
# Calculate average life expectancy by country
avg_life_expectancy = who_life_expectancy_data_cleaned.groupby('country')['life_expectancy'].mean().reset_index()

# Plot bar chart for average life expectancy by country
fig = px.bar(avg_life_expectancy, x='country', y='life_expectancy', title='Average Life Expectancy by Country')
fig.show()
```



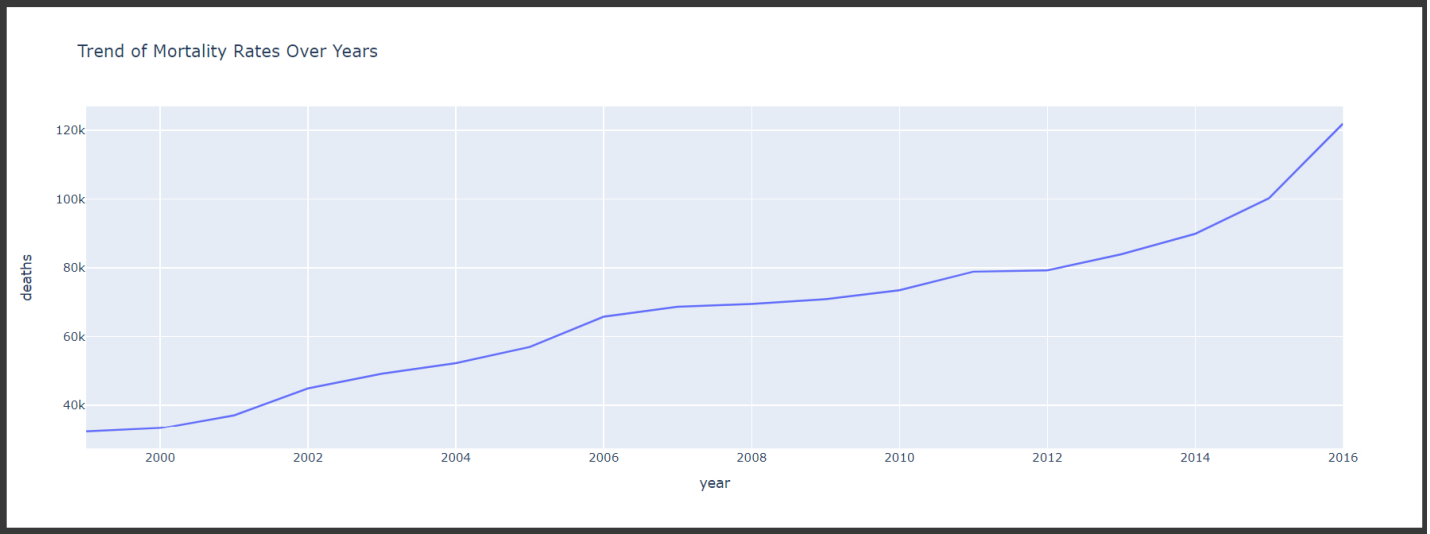
[19]
```
!pip install dash
```

```
Collecting dash
```

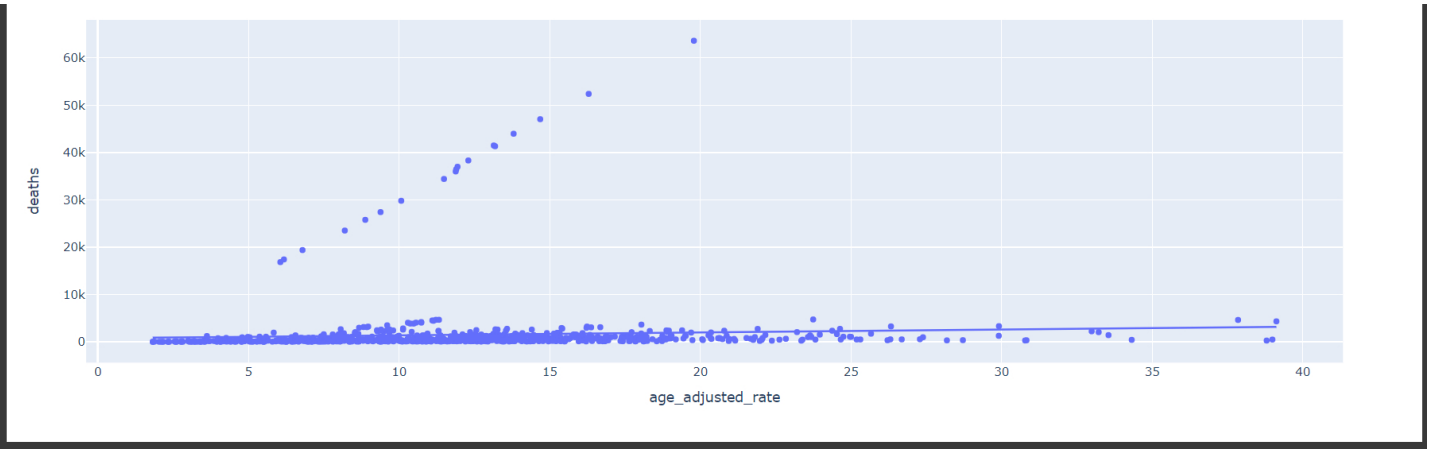
```
  Downloading dash-2.17.1-py3-none-any.whl (7.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 18.2 MB/s eta 0:00:00
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from dash) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (from dash) (3.0.3)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash) (5.15.0)
Collecting dash-html-components==2.0.0 (from dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from dash) (8.0.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash) (4.12.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from dash) (2.31.0)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dash) (67.7.2)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (8.1.7)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash) (8.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash) (24.1)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from Werkzeug<3.1->dash) (2.1.5)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata->dash) (3.19.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (2024.7.4)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from retrying->dash) (1.16.0)
Installing collected packages: dash-table, dash-html-components, dash-core-components, retrying, dash
Successfully installed dash-2.17.1 dash-core-components-2.0.0 dash-html-components-2.0.0 dash-table-5.0.0 retrying-1.3.4
```

[21]
```python
# Import necessary libraries
from dash import Dash, dcc, html
import plotly.express as px
import pandas as pd

# Initialize the Dash app
app = Dash(__name__)

# Ensure that the necessary dataframes are available
# Assuming `cdc_mortality_data_cleaned` and `who_life_expectancy_data_cleaned` are already defined
```

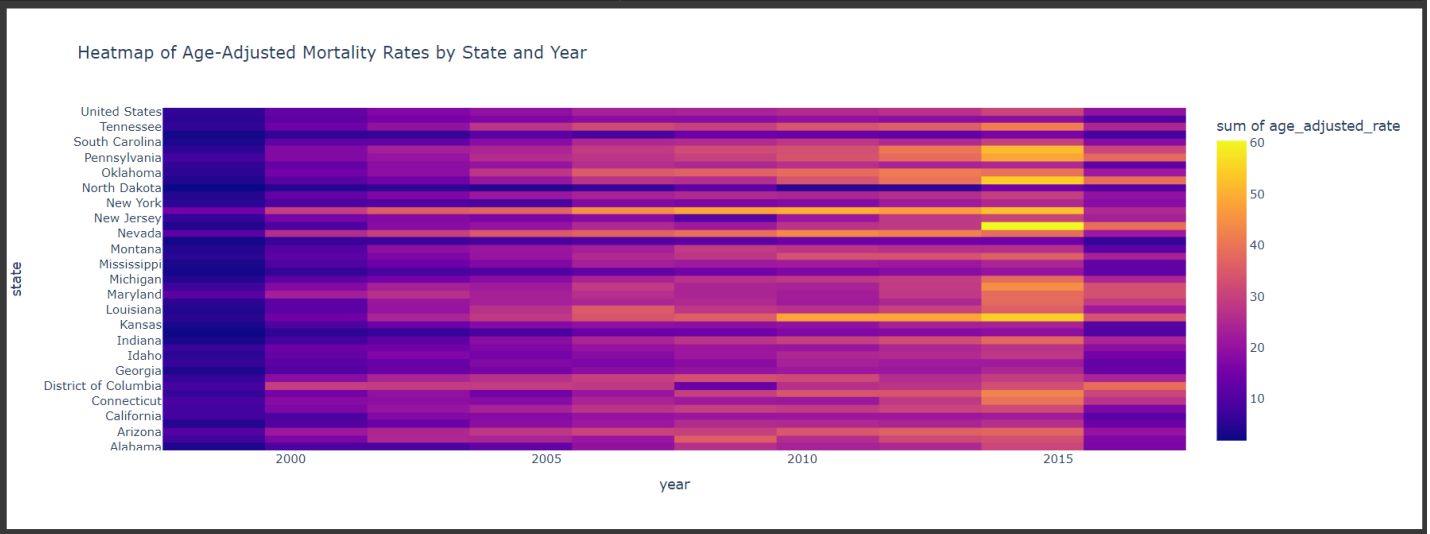
[22]
```python
# Ensure 'year' column is in datetime format and extract year
cdc_mortality_data_cleaned['year'] = pd.to_datetime(cdc_mortality_data_cleaned['year'], format='%Y')
cdc_mortality_data_cleaned['year'] = cdc_mortality_data_cleaned['year'].dt.year

# Convert 'deaths' column to numeric if not already
cdc_mortality_data_cleaned['deaths'] = pd.to_numeric(cdc_mortality_data_cleaned['deaths'], errors='coerce')

# Group by year and calculate the sum of deaths for each year
mortality_trend = cdc_mortality_data_cleaned.groupby('year')['deaths'].sum().reset_index()

# Create figure for trend analysis
fig_mortality_trend = px.line(mortality_trend, x='year', y='deaths', title='Trend of Mortality Rates Over Years')
fig_mortality_trend.show()
```



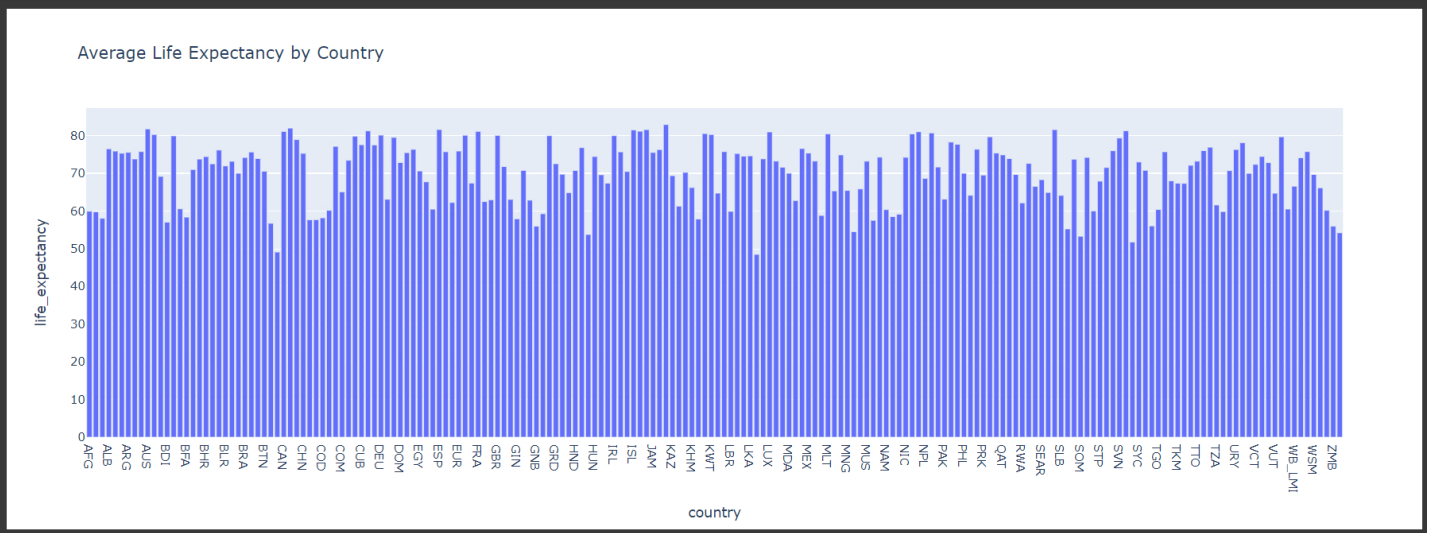
[23]
```python
# Convert 'age_adjusted_rate' to numeric
cdc_mortality_data_cleaned['age_adjusted_rate'] = pd.to_numeric(cdc_mortality_data_cleaned['age_adjusted_rate'], errors='coerce')

# Drop rows with missing values in 'age_adjusted_rate' or 'deaths'
cdc_mortality_data_cleaned = cdc_mortality_data_cleaned.dropna(subset=['age_adjusted_rate', 'deaths'])

# Analyze correlation between deaths and age_adjusted_rate
correlation = cdc_mortality_data_cleaned[['age_adjusted_rate', 'deaths']].corr()

# Create scatter plot for visualizing correlation
fig_correlation = px.scatter(cdc_mortality_data_cleaned, x='age_adjusted_rate', y='deaths', trendline='ols',
                             title='Correlation between Age-Adjusted Mortality Rate and Deaths')
fig_correlation.show()
```



[24]
```
# Plot heatmap for age-adjusted mortality rates
fig_heatmap = px.density_heatmap(cdc_mortality_data_cleaned, x='year', y='state', z='age_adjusted_rate',
                                 title='Heatmap of Age-Adjusted Mortality Rates by State and Year')
fig_heatmap.show()
```



[25]
```
# Calculate average life expectancy by country
avg_life_expectancy = who_life_expectancy_data_cleaned.groupby('country')['life_expectancy'].mean().reset_index()

# Plot bar chart for average life expectancy by country
fig_life_expectancy = px.bar(avg_life_expectancy, x='country', y='life_expectancy', title='Average Life Expectancy by Country')
fig_life_expectancy.show()
```



[26]
```
# Display columns and first few rows of who_life_expectancy_data_cleaned
print("Columns in who_life_expectancy_data_cleaned:")
print(who_life_expectancy_data_cleaned.columns)
print("\nFirst few rows of who_life_expectancy_data_cleaned:")
print(who_life_expectancy_data_cleaned.head())
```

```
Columns in who_life_expectancy_data_cleaned:
Index(['start_date', 'end_date', 'country', 'sex', 'life_expectancy'], dtype='object')

First few rows of who_life_expectancy_data_cleaned:
```

```
                start_date                 end_date country       sex  \
0 2015-01-01 00:00:00+01:00 2015-12-31 00:00:00+01:00     URY  SEX_BTSX
1 2019-01-01 00:00:00+01:00 2019-12-31 00:00:00+01:00     TON  SEX_BTSX
2 2015-01-01 00:00:00+01:00 2015-12-31 00:00:00+01:00  GLOBAL  SEX_BTSX
3 2000-01-01 00:00:00+01:00 2000-12-31 00:00:00+01:00     SUR  SEX_BTSX
4 2000-01-01 00:00:00+01:00 2000-12-31 00:00:00+01:00     GRC   SEX_MLE

   life_expectancy
0         77.03117
1         72.57066
2         72.27778
3         69.94162
4         75.51034
```

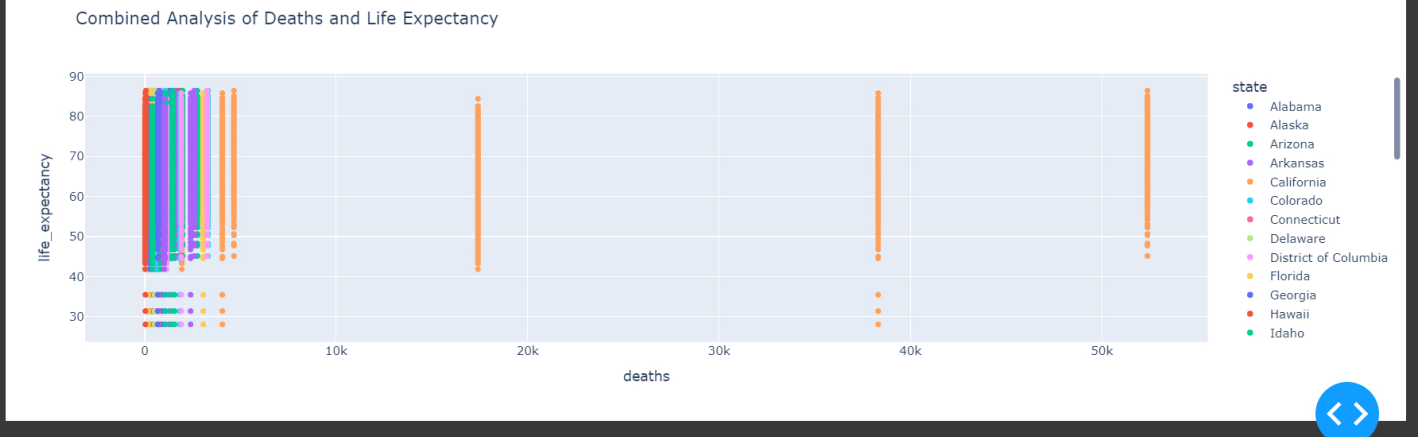
[27]
```
# Convert 'start_date' to datetime and extract the year
who_life_expectancy_data_cleaned['year'] = pd.to_datetime(who_life_expectancy_data_cleaned['start_date']).dt.year

# Merge CDC and WHO data based on year and state/country
combined_data = pd.merge(cdc_mortality_data_cleaned, who_life_expectancy_data_cleaned, on='year', how='inner')

# Select relevant columns for combined analysis
combined_data = combined_data[['year', 'state', 'country', 'deaths', 'life_expectancy']]

# Display the first few rows of the combined data to verify
print("Combined Data:")
print(combined_data.head())

# Create scatter plot for combined data analysis
fig_combined = px.scatter(combined_data, x='deaths', y='life_expectancy', color='state',
                          title='Combined Analysis of Deaths and Life Expectancy')
```

```
Combined Data:
   year    state country  deaths  life_expectancy
0  2000  Alabama     SUR     197         69.94162
1  2000  Alabama     GRC     197         75.51034
2  2000  Alabama     KGZ     197         65.95020
3  2000  Alabama     BGD     197         64.07050
4  2000  Alabama     SAU     197         69.58871
```

[28]
```
# Define the layout of the app
app.layout = html.Div([
    html.H1("Health Inequities Dashboard"),
    dcc.Tabs([
        dcc.Tab(label='Trend Analysis', children=[
            dcc.Graph(id='mortality-trend-graph', figure=fig_mortality_trend)
        ]),
        dcc.Tab(label='Correlation Analysis', children=[
            dcc.Graph(id='correlation-graph', figure=fig_correlation)
        ]),
        dcc.Tab(label='Heatmap Analysis', children=[
            dcc.Graph(id='heatmap-graph', figure=fig_heatmap)
        ]),
        dcc.Tab(label='Life Expectancy', children=[
            dcc.Graph(id='life-expectancy-graph', figure=fig_life_expectancy)
        ]),
        dcc.Tab(label='Combined Data Analysis', children=[
            dcc.Graph(id='combined-data-graph', figure=fig_combined)
        ])
    ])
])

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

Health Inequities Dashboard

Trend Analysis | Correlation Analysis | Heatmap Analysis | Life Expectancy | Combined Data Analysis



[29]
```
app.layout = html.Div([
    html.H1("Health Inequities Dashboard"),
    dcc.Tabs([
        dcc.Tab(label='Trend Analysis', children=[
```

```
                dcc.Graph(id='mortality-trend-graph', figure=fig_mortality_trend)
            ]),
            dcc.Tab(label='Correlation Analysis', children=[
                dcc.Graph(id='correlation-graph', figure=fig_correlation)
            ]),
            dcc.Tab(label='Heatmap Analysis', children=[
                dcc.Dropdown(
                    id='state-dropdown',
                    options=[{'label': state, 'value': state} for state in cdc_mortality_data_cleaned['state'].unique()],
                    value=cdc_mortality_data_cleaned['state'].unique()[0]
                ),
                dcc.Graph(id='heatmap-graph')
            ]),
            dcc.Tab(label='Life Expectancy', children=[
                dcc.Graph(id='life-expectancy-graph', figure=fig_life_expectancy)
            ]),
            dcc.Tab(label='Combined Data Analysis', children=[
                dcc.Graph(id='combined-data-graph', figure=fig_combined)
            ])
        ])
    ])
```

0s

```
[30] from dash.dependencies import Input, Output

    @app.callback(
        Output('heatmap-graph', 'figure'),
        Input('state-dropdown', 'value')
    )
    def update_heatmap(selected_state):
        filtered_data = cdc_mortality_data_cleaned[cdc_mortality_data_cleaned['state'] == selected_state]
        fig_heatmap = px.density_heatmap(filtered_data, x='year', y='state', z='age_adjusted_rate',
                                         title=f'Heatmap of Age-Adjusted Mortality Rates in {selected_state}')
        return fig_heatmap
```

```
[ ]  Start coding or generate with AI.
```