Data Visualization (Using Plotly)

Import data from importdata.ipynb

In [ ]:
```
import numpy as np
import pandas as pd
import import_ipynb
import sys
sys.path.append(r"C:\Users\user\Desktop\wbez")
import importdata
```

```
importing Jupyter notebook from importdata.ipynb
<string>:9: DtypeWarning: Columns (3) have mixed types. Specify dtype option on import or set low_memory=False.
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41786043 entries, 0 to 41786042
Data columns (total 14 columns):
 #   Column     Dtype  
---  ------     -----  
 0   id         int64  
 1   race       object 
 2   sex        object 
 3   zip        object 
 4   year       int64  
 5   date       object 
 6   reason     object 
 7   outcome    object 
 8   agency_id  float64
 9   age        int64  
 10  code       float64
 11  agency     object 
 12  geoid      float64
 13  county     object 
dtypes: float64(3), int64(3), object(8)
memory usage: 4.4+ GB
2019
                              Race  Population
0                            White   5853518.0
1                            Black   1260534.0
2                         Hispanic   1387742.0
3                            Asian    508855.0
4  Native Hawaiian/Pacific Islander      2310.0
6                           Others      9471.0
2020
                              Race  Population
0                            White   5619291.0
1                            Black   1207387.0
2                         Hispanic   1337232.0
3                            Asian    493163.0
4  Native Hawaiian/Pacific Islander      1979.0
6                           Others     11062.0
2021
                              Race  Population
0                            White   5060436.0
1                            Black   1884014.0
2                         Hispanic   1826702.0
3                            Asian    545762.0
4  Native Hawaiian/Pacific Islander      5122.0
6                           Others     28354.0
2022
                              Race  Population
0                            White   5047912.0
1                            Black   1975167.0
2                         Hispanic   1884763.0
3                            Asian    539009.0
4  Native Hawaiian/Pacific Islander      5051.0
6                           Others     29113.0
```

1. Time-Series Plot on Year Basis

In [ ]:
```
# Convert 'year' column to numeric, handle missing values
importdata.df['year'] = pd.to_numeric(importdata.df['year'], errors='coerce')
importdata.df.sort_values(by='year', ascending=False).head()
```

Out[ ]:

	id	race	sex	zip	year	date	reason	outcome	agency_id	age	code	agency	geoi
41786042	42450285	Hispanic	Male	60625.0	2022	2022-12-31	Moving Violation	Verbal Warning	155.0	48	13194.0	CHICAGO POLICE	1714000
40446382	41108457	White	Female	60044.0	2022	2022-07-29	Moving Violation	Written Warning	460.0	38	13478.0	LAKE BLUFF POLICE	1740910
40446384	41108459	White	Female	60045.0	2022	2022-07-29	Moving Violation	Written Warning	460.0	23	13478.0	LAKE BLUFF POLICE	1740910
40446385	41108460	White	Male	60064.0	2022	2022-07-30	Moving Violation	Citation	460.0	22	13478.0	LAKE BLUFF POLICE	1740910
40446386	41108461	White	Male	60045.0	2022	2022-07-30	Moving Violation	Citation	460.0	51	13478.0	LAKE BLUFF POLICE	1740910

In [ ]:
```
import plotly.express as px

def plot_time_series_counts(df, group_col):
    """
    Plots time series of counts for different columns over different years.

    Parameters:
    - df: The full_stops_cleaned data.
    - group_col: The column by which the data should be grouped for counting.
    """
    # Group by 'Month' and another specified column, and count occurrences
    counts_df = df.groupby(['year', group_col]).size().reset_index(name='Counts')

    # Plot time series with counts as lines
    fig = px.line(counts_df, x='year', y='Counts', color=group_col,
                  title="Times Seires:{} vs. Year".format(group_col), line_group=group_col)
    fig.show()

plot_time_series_counts(importdata.df, 'sex')
plot_time_series_counts(importdata.df, 'race')
plot_time_series_counts(importdata.df, 'reason')
plot_time_series_counts(importdata.df, 'outcome')
```

Time-Series Plot on Month/Year Basis

In [ ]:
```
# importdata.df['Mon_yr'] = importdata.df['DateOfStop'].dt.to_period('M')
# def plot_time_series_counts(df, group_col):

    #counts_df = df.groupby(['Mon_yr', group_col]).size().reset_index(name='Counts')
    #counts_df['Mon_yr'] = counts_df['Mon_yr'].astype(str)
    #fig = px.line(counts_df, x='Mon_yr', y='Counts', color=group_col,
    #              title="Times Seires:{} vs. Month/Year".format(group_col), line_group=group_col)
    #fig.show()

#plot_time_series_counts(importdata.df, 'sex')
#plot_time_series_counts(importdata.df, 'race')
#plot_time_series_counts(importdata.df, 'reason')
#plot_time_series_counts(importdata.df, 'outcome')
```

2. Histogram

In [ ]:
```python
def generate_histogram(df, feature, value):
    """
    Generates a histogram based on specified feature (driver race) and value (police variable).

    Parameters:
    - df: The full_stops_cleaned data.
    - feature: The column by which the data should be grouped for the histogram.
    - value: The column representing the specific value for color-coding in the histogram.
    """
    # Group by specified columns and calculate counts
    counts_df = df.groupby([feature,value]).size().reset_index(name='Counts')

    # Create histogram with text annotations
    fig = px.histogram(counts_df, x=feature, y='Counts', color=value,
                       text_auto=True, title='{} of Stop by Driver Race'.format(value))

    return fig

generate_histogram(importdata.df, 'race', 'outcome')
```

In [ ]:
```python
generate_histogram(importdata.df, 'race', 'reason')
```

3. Pie Chart

In [ ]:
```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

def plot_pie_chart(df, value, race_categories):
    """
    Plots several pie charts based on different police variables and race categories.

    Parameters:
    - df: The full_stops_cleaned data.
    - value: a police variable
    - race_categories: A list of race categories for which pie charts will be plotted.
    """
    # Group by race and a specified police variable and calculate counts
    counts_df = df.groupby(['race',value]).size().reset_index(name='Counts')

    # Create subplots
    fig = make_subplots(rows=2, cols=3, specs=[[{'type': 'domain'},
                                                {'type': 'domain'}, {'type': 'domain'}],
                                               [{'type': 'domain'},
                                                {'type': 'domain'}, {'type': 'domain'}]],
                        subplot_titles=race_categories)

    # Add traces for each race category
    for race in race_categories:
        race_df = counts_df[counts_df['race'] == race]
        fig.add_trace(go.Pie(labels=race_df[value], values=race_df['Counts'], name=race),
                      (race_categories.index(race) // 3) + 1,
                      (race_categories.index(race) % 3) + 1)

    # Update layout and display the plot
    fig.update_layout(
        title_text='{} by Driver Race (%)'.format(value),
        height=600,
        width=1000,
    )
    fig.show()

# Display
plot_pie_chart(importdata.df, 'reason',
               ['Asian', 'Black', 'Hispanic', 'Native American',
                'Native Hawaiian/Pacific Islander', 'White'])
plot_pie_chart(importdata.df, 'outcome',
               ['Asian', 'Black', 'Hispanic', 'Native American',
                'Native Hawaiian/Pacific Islander', 'White'])
```

4. Comparison between Different Dataframes

In [ ]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

def compare_estimate(df, value):
    """
    Compares the distribution of race among different dataframes
    (number of stops, the benchmark for driving population,
    and census data of total Illinois population.)

    Parameters:
    - df: The full_stops_cleaned data.
    - value (str): The column 'race' is used for pie chart categorization.

    """
    counts_df = df.groupby(value).size().reset_index(name='Counts')

    fig = make_subplots(rows=1, cols=3, specs=[[{'type': 'domain'},
                                                {'type': 'domain'}, {'type': 'domain'}]],
                        subplot_titles=['Stop Numbers', 'Driving Population', 'Census Data'])

    # Add traces for stop numbers, driving population, and census data
    fig.add_trace(go.Pie(labels=counts_df[value], values=counts_df['Counts']),1,1)
    fig.add_trace(go.Pie(labels=importdata.df_driv_2020['Race'],
                         values=importdata.df_driv_2020['Population']),1,2)
    fig.add_trace(go.Pie(labels=importdata.df_census['Race'],
                         values=importdata.df_census['Population']),1,3)


    fig.update_layout(
        title_text='Race by Proportion in 2020',
        height=600,
        width=1000,
    )

    fig.show()

compare_estimate(importdata.df_2020, 'race')
```

5. Normalized Histogram

In [ ]:
```
def normalized_histogram(df, df2, title):
    """
    Compares the normalized proportion of stop counts and driving population counts by race.

    Parameters:
    - df: From the full_stops_cleaned data.
    - df2: DataFrame containing driving population data.
    - title: The year for which the comparison is performed.
    """
    # Group by specified columns and calculate counts
    counts_df = df.groupby(['race']).size().reset_index(name='Counts')
    counts_df['Percentage'] = counts_df['Counts'] / counts_df['Counts'].sum()

    counts_df['data_type']= 'Stops'
    counts_df = counts_df.rename(columns={'race': 'Race'})

    df2['data_type'] = 'Driving Population'
    df2  = df2.rename(columns={'Population': 'Counts'})
    df2['Percentage'] = df2['Counts'] / df2['Counts'].sum()

    # Concatenate the two DataFrames
    combined_df = pd.concat([counts_df, df2], ignore_index=True)

    # Create histogram with text annotations
    fig = px.histogram(combined_df, x='data_type', y='Percentage',
                       color='Race', title='Nomralized Proportion by Driver Race in {}'.format(title))
```

```
    return fig

normalized_histogram(importdata.df_2019, importdata.df_driv_2019, 2019)
normalized_histogram(importdata.df_2020, importdata.df_driv_2020, 2020)
normalized_histogram(importdata.df_2021, importdata.df_driv_2021, 2021)
normalized_histogram(importdata.df_2022, importdata.df_driv_2022, 2022)
```

6. In Chicago or not

In [ ]:
```
importdata.df['chicago'] = importdata.df['county'].apply(lambda x: 1 if x == 'COOK' else 0)
```

In [ ]:
```
plot_time_series_counts(importdata.df, 'chicago')
```

In [ ]:
```
df_chicago = importdata.df[importdata.df['county']=='COOK']
df_other = importdata.df[importdata.df['county'] !='COOK']
```

In [ ]:
```
plot_pie_chart(df_chicago, 'reason', ['Asian', 'Black', 'Hispanic',
                                      'Native American',
                                      'Native Hawaiian/Pacific Islander', 'White'])
plot_pie_chart(df_chicago, 'outcome', ['Asian', 'Black', 'Hispanic', 'Native American',
                                       'Native Hawaiian/Pacific Islander', 'White'])
plot_pie_chart(df_other, 'reason', ['Asian', 'Black', 'Hispanic', 'Native American',
                                    'Native Hawaiian/Pacific Islander', 'White'])
plot_pie_chart(df_other, 'outcome', ['Asian', 'Black', 'Hispanic', 'Native American',
                                     'Native Hawaiian/Pacific Islander', 'White'])
```

7. Interactive Graphs

In [ ]:
```
# App Link: http://127.0.0.1:8050/

from dash import Dash, dcc, html, Input, Output
import dash_bootstrap_components as dbc
import plotly.express as px

# Create the Dash app
app = Dash(__name__)

# Define the layout of the app
app.layout = html.Div(
    dbc.Container(
        style={'backgroundColor': '#f4f4f4'},
        children=[
        html.H2(children='Exploring Driver Race and Policing Data Through Visualization',
                style={'color': "#0C2D48", 'font-weight': 'bold'}),

        # Section 1: Time Series Plots
        html.H3(children='1. Time Series Plots for Key Variables in Stops Dataframe on Year Basis'),
        html.P("Select Variable:"),
        dcc.Dropdown(
            id="selected_variable",
            options=[
                {'label': 'Driver Race', 'value': 'race'},
                {'label': 'Driver Sex', 'value': 'sex'},
                {'label': 'Outcome', 'value': 'outcome'},
                {'label': 'Reason for Stop', 'value': 'reason'},
            ],
            value='race',
            clearable=False,
        ),
        html.P("Select Area:"),
        dcc.Dropdown(
            id="selected_area",
            options=[
```

```
                {'label': 'All', 'value': 'All'},
                {'label': 'Chicago (Cook County)', 'value': 'Chicago'},
                {'label': 'Other Counties in Illinois', 'value': 'Others'},
            ],
            value='All',
            clearable=False,
        ),
        dcc.Graph(id="time-series-chart"),
        html.Br(),

        # Section 2: Stacked Histogram
        html.H3(children='2. Stacked Histogram: Police Variables by Driver Race'),
        html.P("Select Year Range:"),
        dcc.RangeSlider(
            id="selected_year",
            min=2004,
            max=2023,
            step=1,
            marks={str(year): str(year) for year in range(2004, 2024)},
            value=[2004, 2023],
        ),
        html.P("Select Race:"),
        dcc.Dropdown(
            id="selected_race",
            options=[
                {'label': race, 'value': race} for race in ['Asian', 'Black', 'Hispanic',
                                                            'Native American', 'White',
                                                            'Native Hawaiian/Pacific Islander']
            ],
            value=["Asian", 'Black', 'Hispanic', 'White'],
            multi=True,
            clearable=False,
        ),
        dcc.Graph(id="histogram"),
        html.Br(),

        # Section 3: Pie Charts
        html.H3(children='3. Pie Charts: Proportions of Police Related Variables by Driver Race'),
        html.P("Select Year Range:"),
        dcc.RangeSlider(
            id="selected_year2",
            min=2004,
            max=2023,
            step=1,
            marks={str(year): str(year) for year in range(2004, 2024)},
            value=[2004, 2023],
        ),
        html.P("Select Police Variable:"),
        dcc.Dropdown(
            id="selected_police_variable",
            options=[
                {'label': 'Outcome', 'value': 'outcome'},
                {'label': 'Reason for Stop', 'value': 'reason'},
               ],
            value='outcome',
            clearable=False,
        ),
        html.P("Select Area:"),
        dcc.Dropdown(
            id="selected_area2",
            options=[
                {'label': 'All', 'value': 'All'},
                {'label': 'Chicago (Cook County)', 'value': 'Chicago'},
                {'label': 'Other Counties in Illinois', 'value': 'Others'},
            ],
            value='All',
            clearable=False,
        ),
        dcc.Graph(id="pie-chart"),
        html.Br(),

        # Section 4: Normalization
        html.H3(children='4. Normalization'),
        html.P("Select Year:"),
```

```
        dcc.Dropdown(
            id="selected_year3",
            options=[{'label': str(year), 'value': str(year)} for year in range(2019, 2023)],
            value='2019',
            clearable=False
        ),
        dcc.Graph(id="normalized-proportion"),
        html.Br()
    ]))



# Callback to update time series chart based on selected variable
@app.callback(
    Output("time-series-chart", "figure"),
    Input("selected_variable", "value"),
    Input("selected_area", "value")
)
def plot_time_series_counts(selected_variable, selected_area):
    if selected_area == 'All':
        df = importdata.df
    elif selected_area == 'Chicago':
        df = df_chicago
    else:
        df = df_other
    # Group by 'Month' and another specified column, and count occurrences
    counts_df = df.groupby(['year', selected_variable]).size().reset_index(name='Counts')

    # Plot time series with counts as lines
    fig = px.line(counts_df, x='year', y='Counts', color=selected_variable,
                  title="Times Seires:{} vs. Year in Area {}".format(selected_variable, selected_area),
                 line_group= selected_variable)
    return fig



# Callback to generate stacked histogram
@app.callback(
    Output("histogram", "figure"),
    [Input("selected_year", "value"),
     Input("selected_race", "value")]
)
def generate_histogram(selected_year, selected_race):
    # Filter DataFrame based on selected_year
    counts_df = importdata.df.groupby(['year', 'race','reason']).size().reset_index(name='Counts')
    start_year, end_year = selected_year
    filtered_df = counts_df[(counts_df['year'] >= int(start_year)) &
                            (counts_df['year'] <= int(end_year))]

    # Filter DataFrame based on selected_race
    if selected_race:
        filtered_df = filtered_df[filtered_df['race'].isin(selected_race)]

    # Create the histogram figure
    fig = px.histogram(filtered_df, x="race", y="Counts", color="reason", text_auto=True,
                       title=f'Reason for Stop by Driver Race from {start_year} to {end_year}')
    return fig



# Callback to plot pie chart
@app.callback(
    Output("pie-chart", "figure"),
    [Input("selected_police_variable", "value"),
     Input("selected_year2", "value"),
     Input("selected_area2", "value")]
)
def plot_pie_chart(selected_variable, selected_year2, selected_area2):
    if selected_area2 == 'All':
        df = importdata.df
    elif selected_area2 == 'Chicago':
        df = df_chicago
    else:
        df = df_other
```

```
    # Filter year
    start_year, end_year = selected_year2
    df = df[(df['year'] >= int(start_year)) & (df['year'] <= int(end_year))]

    # Group by selected_variable and race
    counts_df = df.groupby(['race', selected_variable]).size().reset_index(name='Counts')
    race_categories = ['Asian', 'Black', 'White', 'Hispanic',
                       'Native American', 'Native Hawaiian/Pacific Islander']

    # Create pie charts
    fig = make_subplots(rows=2, cols=3, specs=[[{'type': 'domain'},
                                                {'type': 'domain'}, {'type': 'domain'}],
                                               [{'type': 'domain'},
                                                {'type': 'domain'}, {'type': 'domain'}]],
                        subplot_titles= race_categories)

    for race in  race_categories:
        race_df = counts_df[counts_df['race'] == race]
        fig.add_trace(go.Pie(labels=race_df[selected_variable],
                             values=race_df['Counts'], name=race),
                             (race_categories.index(race) // 3) + 1,
                             (race_categories.index(race) % 3) + 1)

    fig.update_layout(
        title_text=f'{selected_variable} Proportions by Driver Race (%)
                    from {start_year} to {end_year} in Area {selected_area2}',
        height=600,
        width=1000,
    )
    return fig


# Callback to generate normalized histogram
@app.callback(
    Output("normalized-proportion", "figure"),
    [Input("selected_year3", "value")]
)
def normalized_histogram(selected_year3):
    # Filter year
    df = importdata.df[importdata.df['year'] == int(selected_year3)]

    # Group by race and calculate counts and percentage
    counts_df = df.groupby(['race']).size().reset_index(name='Counts')
    counts_df['Percentage'] = counts_df['Counts'] / counts_df['Counts'].sum()

    counts_df['data_type']= 'Stops'
    counts_df = counts_df.rename(columns={'race': 'Race'})

    # Years with the benchmark estimate of driving population
    year_to_df = {
    '2019': importdata.df_driv_2019,
    '2020': importdata.df_driv_2020,
    '2021': importdata.df_driv_2021,
    '2022': importdata.df_driv_2022
    }
    # Filter year
    df2 = year_to_df.get(selected_year3, None)
    df2['data_type'] = 'Driving Population'
    df2  = df2.rename(columns={'Population': 'Counts'})
    df2['Percentage'] = df2['Counts'] / df2['Counts'].sum()

    # combine the stops dataframe and driving population dataframe
    combined_df = pd.concat([counts_df, df2], ignore_index=True)
    fig = px.histogram(combined_df, x='data_type', y='Percentage',
                       color='Race', text_auto=True,
                       title='Nomralized Proportion by Driver Race in {}'.format(selected_year3))
    return fig



if __name__ == "__main__":
    app.run_server(debug=True)
```