Import libraries

In [2]:
```
import plotly.express as px
import numpy as np
import pandas as pd
```

Read the Data

In [3]:
```
# read the data
dfCar = pd.read_excel('Car_Data.xlsx')
dfCar.head()
```

Out[3]:

	CustomerName	CustomerGender	CustomerJobTitle	CarMaker	CarModel	CarCountry	CarColor
0	Louie Hinsche	Male	Mechanical Systems Engineer	Dodge	Ram 2500	USA	Goldenrod
1	Alexandros Manuel	Male	Structural Engineer	Toyota	Tundra	Japan	Crimson
2	Alvie Weighell	Male	Systems Administrator III	GMC	Savana 1500	USA	Black
3	Flint Gunston	Male	Operator	Volkswagen	Cabriolet	Germany	Fuscia
4	Alyssa Filpi	Female	Software Engineer III	Mercury	Mariner	USA	Teal

5 rows × 22 columns

In [4]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import dash
import dash_bootstrap_components as dbc
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

# Load your dataset
dfCar = pd.read_excel('Car_Data.xlsx')

# Aggregate the total price for each car model of each car maker in each country
dfCar['TotalPrice'] = dfCar.groupby(['CarCountry', 'CarMaker', 'CarModel'])['CarPrice'

# Dashboard 1
top_carmakers = dfCar['CarMaker'].value_counts().index.tolist()
top_carmakers = [carmaker for carmaker in top_carmakers if carmaker != 'Chevrolet' and

# Dashboard 2
top_states = dfCar['State'].value_counts().nlargest(10).index.tolist()
```

```
# Dashboard 3
countries = dfCar['CarCountry'].unique()
years = sorted(dfCar['CarModelYear'].unique())
years = [year for year in years if 2002 <= year <= 2014]

# Define selected car makers
car_makers = sorted(['Ford', 'Chevrolet', 'Toyota', 'Volkswagen', 'Mazda'])

app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP], suppress_callba

# Create a navigation bar
sidebar = html.Div(
    [
        html.H2("Navigation", style={'textAlign': 'center'}),
        html.Hr(),
        dbc.Nav(
            [
                dbc.NavLink("Dashboard 1", href="/page-1", id="page-1-link"),
                dbc.NavLink("Dashboard 2", href="/page-2", id="page-2-link"),
                dbc.NavLink("Dashboard 3", href="/page-3", id="page-3-link"),
                dbc.NavLink("Dashboard 4", href="/page-4", id="page-4-link"),
                dbc.NavLink("Dashboard 5", href="/page-5", id="page-5-link"),
            ],
            vertical=True,
            pills=True,
        ),
    ],
    style={'width': '25%', 'display': 'inline-block', 'float': 'right'}
)

# Create a container for the content
content = html.Div(id="page-content", style={'width': '75%', 'display': 'inline-block'

app.layout = html.Div([
    html.H1("Dashboard Driven: Steering Through Car Sales Data", style={'textAlign': '
    dcc.Location(id="url"),
    sidebar,
    content
])


# Callbacks to update the content based on the selected option in the sidebar
@app.callback(
    [Output(f"page-{i}-link", "active") for i in range(1, 6)],
    [Input("url", "pathname")],
)
def toggle_active_links(pathname):
    if pathname == "/":
        # Treat page 1 as the homepage / index
        return True, False, False, False, False
    return [pathname == f"/page-{i}" for i in range(1, 6)]

@app.callback(Output("page-content", "children"), [Input("url", "pathname")])
def render_page_content(pathname):
    if pathname in ["/", "/page-1"]:
        return html.Div([
            html.H2("Price and Sales Analysis"),
            html.Div(id='description1'),
            dcc.Dropdown(
```

```
                id='carmaker-dropdown',
                options=[{'label': i, 'value': i} for i in top_carmakers],
                value=top_carmakers[0]  # Default value
            ),
            dcc.Graph(id='graph1'),
            dcc.Graph(id='graph2')
        ])
    elif pathname == "/page-2":
        return html.Div([
            html.H2("Navigating the Landscape of State-wise Car Sales"),
            dcc.RangeSlider(
                id='year-slider-2',
                min=min(years),
                max=max(years),
                value=[min(years), max(years)],
                marks={str(year): str(year) for year in years},
                step=None
            ),
            dcc.Checklist(
                id='state-checklist',
                options=[{'label': i, 'value': i} for i in top_states],
                value=top_states  # Default values
            ),
            dcc.Graph(id='treemap')
        ])
    elif pathname == "/page-3":
        return html.Div([
            html.H2("Car Ratings Snapshot"),
            html.Div(id='description3'),
            dcc.RadioItems(
                id='carmaker-radio',
                options=[{'label': i, 'value': i} for i in car_makers],
                value=car_makers[0]  # Default value
            ),
            dcc.Graph(id='lollipop')
        ])
    elif pathname == "/page-4":
        return html.Div([
            html.H2("Exploring Car Sales Across US Cities"),
            html.Div(id='description4'),
            dcc.Dropdown(
                id='state-dropdown',
                options=[{'label': i, 'value': i} for i in sorted(dfCar['State'].uniqu
                value=sorted(dfCar['State'].unique())[0]
            ),
            dcc.Dropdown(id='city-dropdown'),
            dcc.Graph(id='graph4')
        ])
    elif pathname == "/page-5":
        return html.Div([
            html.H2("A Comprehensive View of Worldwide Car Markets"),
            html.Div(id='description5'),
            dcc.Dropdown(
                id='country-dropdown-5',
                options=[{'label': i, 'value': i} for i in countries],
                value=countries[0]  # Default value
            ),
            dcc.RangeSlider(
                id='year-slider-5',
                min=min(years),
```

```
                max=max(years),
                value=[min(years), max(years)],
                marks={str(year): str(year) for year in years},
                step=None
            ),
            dcc.Graph(id='sunburst5')
        ])

# Callbacks for Dashboard 1
@app.callback(
    Output('description1', 'children'),
    [Input('carmaker-dropdown', 'value')]
)
def update_description1(carmaker):
    return f"Examining {carmaker} sales trends by model"

@app.callback(
    [Output('graph1', 'figure'),
     Output('graph2', 'figure')],
    [Input('carmaker-dropdown', 'value')]
)
def update_graphs1(carmaker):
    filtered_df = dfCar[dfCar['CarMaker'] == carmaker]
    total_price_by_model = filtered_df.groupby('CarModel')['CarPrice'].sum().reset_inc
    quantity_by_model = filtered_df.groupby('CarModel')['Quantity(CarSold)'].sum().res
    fig1 = px.bar(total_price_by_model, x="CarModel", y="CarPrice", color="CarModel",
    fig2 = px.pie(quantity_by_model, values='Quantity(CarSold)', names='CarModel', tit
    return fig1, fig2

# Callbacks for Dashboard 2
@app.callback(
    Output('treemap', 'figure'),
    [Input('year-slider-2', 'value'),
     Input('state-checklist', 'value')]
)
def update_treemap(selected_years, selected_states):
    filtered_df = dfCar[(dfCar['CarModelYear'] >= selected_years[0]) &
                        (dfCar['CarModelYear'] <= selected_years[1]) &
                        (dfCar['State'].isin(selected_states))]
    fig = px.treemap(filtered_df, path=['Country', 'State', 'CarMaker'], color='Sales'
    return fig

# Callbacks for Dashboard 3
@app.callback(
    [Output('lollipop', 'figure'),
     Output('description3', 'children')],
    [Input('carmaker-radio', 'value')]
)
def update_lollipop(selected_carmaker):
    filtered_df = dfCar[dfCar['CarMaker'] == selected_carmaker]

    if filtered_df.empty:
        # If no data for the selected car maker, return empty figure and description
        empty_fig = go.Figure()
        empty_description = f"No data available for {selected_carmaker}."
        return empty_fig, empty_description

    # Calculate AverageRating
    filtered_df['AverageRating'] = filtered_df.groupby('CarModel')['CarRating'].transf
```

```
    top_model = filtered_df.loc[filtered_df['AverageRating'].idxmax()]['CarModel']
    low_model = filtered_df.loc[filtered_df['AverageRating'].idxmin()]['CarModel']

    fig = go.Figure()
    for model in filtered_df['CarModel'].unique():
        model_df = filtered_df[filtered_df['CarModel'] == model]
        fig.add_trace(go.Bar(
            x=model_df['CarModel'],
            y=model_df['AverageRating'],
            width=0.1,
            marker=dict(color=model_df['AverageRating'], colorbar=dict(title='Average
            opacity=0.6,
            name=model,
            showlegend=False,  # Do not show 'Car Model' in the legend
            hovertemplate='<i>Average Rating</i>: %{y}',
            hoverinfo='skip'
        ))
        fig.add_trace(go.Scatter(
            x=model_df['CarModel'],
                        y=model_df['AverageRating'] + 0.1,
            mode='markers+text',
            text=model_df['CarRating'],
            textposition='middle center',
            marker=dict(size=50, color=model_df['AverageRating']),
            showlegend=False,
            hovertemplate='<i>Car Rating</i>: %{text}',
            hoverinfo='skip'
        ))

    fig.update_layout(
        xaxis_title='Car Model',
        yaxis_title='Average Rating',
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        font=dict(color='black', size=12),
        hovermode='x'
    )

    description = f"The top-rated car model for {selected_carmaker} is {top_model}, ar
    return fig, description

# Callbacks for Dashboard 4
@app.callback(
    Output('city-dropdown', 'options'),
    Input('state-dropdown', 'value')
)
def set_cities_options(selected_state):
    return [{'label': i, 'value': i} for i in dfCar[dfCar['State'] == selected_state][

@app.callback(
    Output('city-dropdown', 'value'),
    Input('city-dropdown', 'options')
)
def set_cities_value(available_options):
    return available_options[0]['value']

@app.callback(
    Output('graph4', 'figure'),
    Input('state-dropdown', 'value'),
    Input('city-dropdown', 'value')
```

```
)
def update_graph(selected_state, selected_city):
    filtered_df = dfCar[(dfCar['State'] == selected_state) & (dfCar['City'] == selecte
    fig = px.bar(filtered_df, x='CarModel', y='Quantity(CarSold)', color='CarModel',
                 color_discrete_sequence=px.colors.qualitative.Dark24)
    fig.update_layout(
        autosize=False,
        width=700,
        height=500,
        bargap=0.5,
        bargroupgap=0.1,
    )
    return fig

@app.callback(
    Output('description4', 'children'),
    Input('state-dropdown', 'value'),
    Input('city-dropdown', 'value')
)
def update_description(selected_state, selected_city):
    return f"Analyzing car sales by model and maker in {selected_city}, {selected_stat

# Callbacks for Dashboard 5
@app.callback(
    [Output('sunburst5', 'figure'),
     Output('description5', 'children')],
    [Input('country-dropdown-5', 'value'),
     Input('year-slider-5', 'value')]
)
def update_sunburst5(selected_country, selected_years):
    filtered_df = dfCar[(dfCar['CarCountry'] == selected_country) & (dfCar['CarModelYe
    fig = px.sunburst(filtered_df, path=[px.Constant(selected_country), 'CarMaker', 'C
    description = f"The chart provides a hierarchical view of sales revenue data for {
    return fig, description

if __name__ == '__main__':
    app.run_server(debug=True, port=5555)
```

```
C:\Users\harig\AppData\Local\Temp\ipykernel_15532\4204424170.py:6: UserWarning:
The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`
  import dash_core_components as dcc
C:\Users\harig\AppData\Local\Temp\ipykernel_15532\4204424170.py:7: UserWarning:
The dash_html_components package is deprecated. Please replace
`import dash_html_components as html` with `from dash import html`
  import dash_html_components as html
```

```
C:\Users\harig\AppData\Local\Temp\ipykernel_15532\4204424170.py:200: SettingWithCopyW
arning:


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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]: