In [ ]:
```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

# Load the dataset
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/historical_automobile_
df = pd.read_csv(url)
```

In [ ]:
```
df.describe()
```

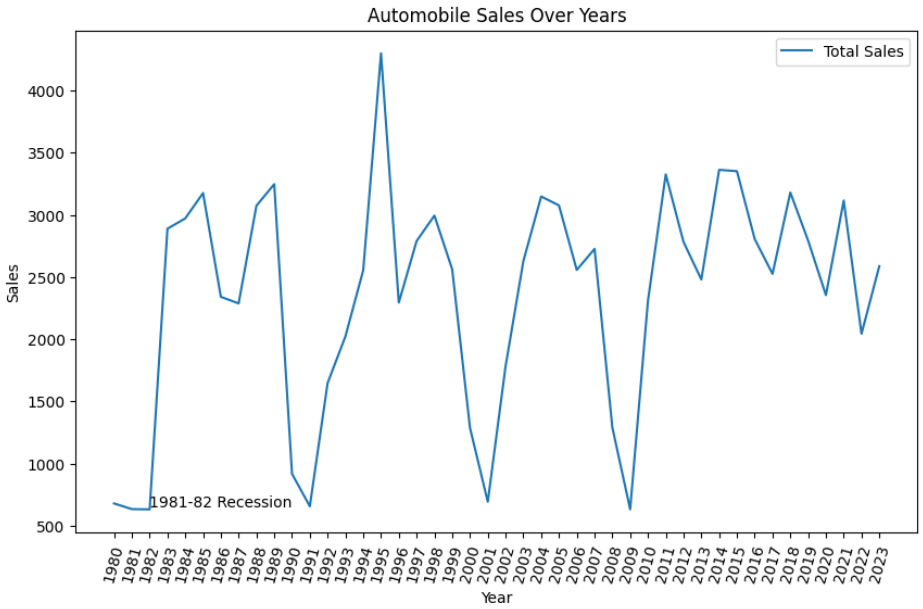
Out[ ]:

	Year	Recession	Consumer_Confidence	Seasonality_Weight	Price	Advertising_Expenditure	Competition	GDP	Growth_Rate	unemplo
count	528.000000	528.000000	528.000000	528.000000	528.000000	528.000000	528.000000	528.000000	528.000000	
mean	2001.500000	0.214015	101.140170	0.575795	24964.991956	3067.456439	6.064394	40.073903	-0.242001	
std	12.710467	0.410526	10.601154	0.454477	4888.073433	1139.564637	1.968350	16.249714	0.861268	
min	1980.000000	0.000000	73.900000	0.000000	8793.663000	1009.000000	3.000000	12.508000	-4.227601	
25%	1990.750000	0.000000	94.035000	0.250000	21453.300500	2083.500000	4.000000	27.237500	-0.574049	
50%	2001.500000	0.000000	100.740000	0.500000	25038.691500	3072.000000	6.000000	39.214500	-0.013162	
75%	2012.250000	0.000000	108.240000	0.750000	28131.684750	4067.250000	8.000000	53.506500	0.388932	
max	2023.000000	1.000000	131.670000	1.500000	44263.657000	4983.000000	9.000000	70.374000	0.815074	

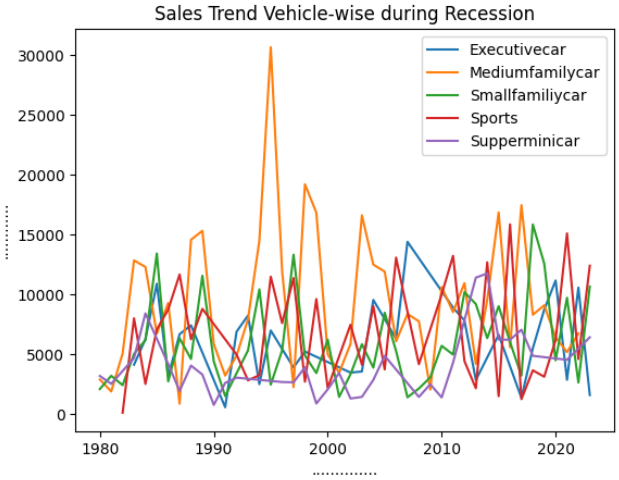
In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Date', 'Year', 'Month', 'Recession', 'Consumer_Confidence',
       'Seasonality_Weight', 'Price', 'Advertising_Expenditure', 'Competition',
       'GDP', 'Growth_Rate', 'unemployment_rate', 'Automobile_Sales',
       'Vehicle_Type', 'City'],
      dtype='object')
```

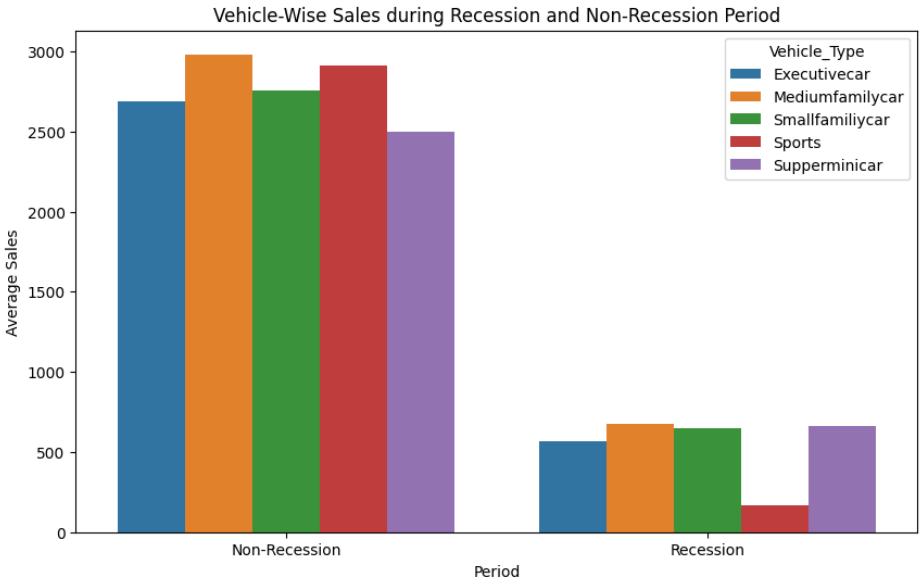
In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Assuming you have a DataFrame df_line with columns 'Year', 'Sales', and 'VehicleType'
# Example: df_line = pd.read_csv('your_dataset_line.csv')

# Replace 'YourXColumn' and 'YourYColumn' with the actual column names you want to use
df_line.plot(x='Year', y='Sales', kind='line', label='Total Sales')
plt.xticks(list(range(1980, 2024)), rotation=75)
plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Automobile Sales Over Years')
plt.text(1982, 650, '1981-82 Recession')
# Provide the correct coordinates and text for additional annotations
# Example: plt.text(1990, 500, 'Some additional annotation')
plt.legend()
plt.savefig('Line_plot_1.png')  # Save the plot as an image
plt.show()
```



In [ ]:
```
df_Mline = df.groupby(['Year','Vehicle_Type'], as_index=False)['Automobile_Sales'].sum()
df_Mline.set_index('Year', inplace=True)
df_Mline = df_Mline.groupby(['Vehicle_Type'])['Automobile_Sales']
df_Mline.plot(kind='line')
plt.xlabel('.............')
plt.ylabel('.............')
plt.title('Sales Trend Vehicle-wise during Recession')
plt.legend()
plt.show()
```



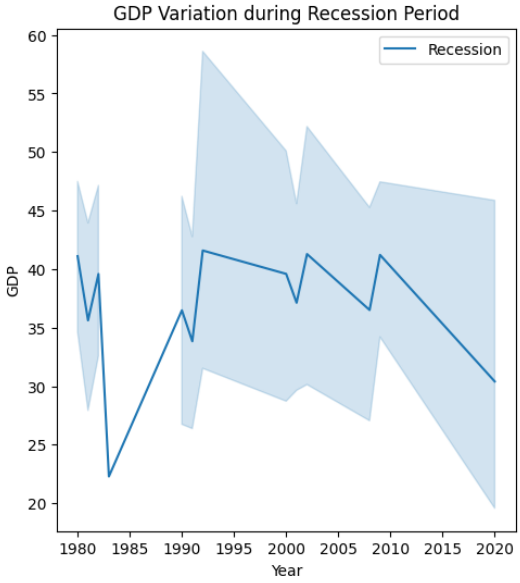
In [ ]:
```
recession_data = df[df['Recession'] == 1]

dd=df.groupby(['Recession','Vehicle_Type'])['Automobile_Sales'].mean().reset_index()

# Calculate the total sales volume by vehicle type during recessions
#sales_by_vehicle_type = recession_data.groupby('Vehicle_Type')['Automobile_Sales'].sum().reset_index()

# Create the grouped bar chart using seaborn
plt.figure(figsize=(10, 6))
sns.barplot(x='Recession', y='Automobile_Sales', hue='Vehicle_Type', data=dd)
plt.xticks(ticks=[0, 1], labels=['Non-Recession', 'Recession'])
plt.xlabel('Period')
plt.ylabel('Average Sales')
plt.title('Vehicle-Wise Sales during Recession and Non-Recession Period')

plt.show()
```

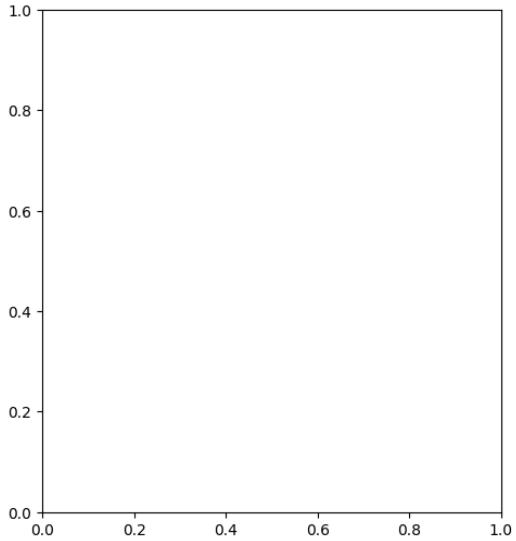


In [ ]:
```
# Create dataframes for recession and non-recession periods
rec_data = df[df['Recession'] == 1]
non_rec_data = df[df['Recession'] == 0]

# Figure
fig = plt.figure(figsize=(12, 6))

# Create different axes for subplotting
ax0 = fig.add_subplot(1, 2, 1)  # add subplot 1 (1 row, 2 columns, first plot)
ax1 = fig.add_subplot(1, 2, 2)  # add subplot 2 (1 row, 2 columns, second plot).

# Subplot 1
sns.lineplot(x='Year', y='GDP', data=rec_data, label='Recession', ax=ax0)
ax0.set_xlabel('Year')
ax0.set_ylabel('GDP')
ax0.set_title('GDP Variation during Recession Period')

# Subplot 2
sns.lineplot(x='Year', y='Sales', data=non_rec_data, label=f'Random_Label_{random.randint(1, 100)}', ax=ax1)
ax1.set_xlabel('Year')
ax1.set_ylabel('Sales')
ax1.set_title('Sales Variation')

plt.tight_layout()
plt.show()
```

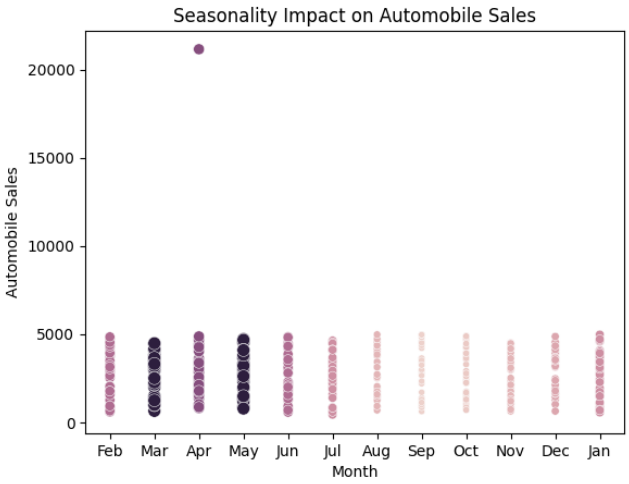
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
Cell In[30], line 19
     16 ax0.set_title('GDP Variation during Recession Period')
     18 # Subplot 2
---> 19 sns.lineplot(x='Year', y='Sales', data=non_rec_data, label=f'Random_Label_{random.randint(1, 100)}', ax=ax1)
     20 ax1.set_xlabel('Year')
     21 ax1.set_ylabel('Sales')

File c:\Users\siranjeevi\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\relational.py:479, in lineplot(data, x, y, hue, size, style, units, palette, hue_order, hue_norm, sizes, size_order, size_norm, dashes, markers, style_order, estimator, errorbar, n_boot, seed, orient, sort, err_style, err_kws, legend, ci, ax, **kwargs)
    465 def lineplot(
    466     data=None, *,
    467     x=None, y=None, hue=None, size=None, style=None, units=None,
   (...)
    475
    476     # Handle deprecation of ci parameter
    477     errorbar = _deprecate_ci(errorbar, ci)
--> 479     p = _LinePlotter(
    480         data=data,
    481         variables=dict(x=x, y=y, hue=hue, size=size, style=style, units=units),
    482         estimator=estimator, n_boot=n_boot, seed=seed, errorbar=errorbar,
    483         sort=sort, orient=orient, err_style=err_style, err_kws=err_kws,
    484         legend=legend,
    485     )
    487     p.map_hue(palette=palette, order=hue_order, norm=hue_norm)
    488     p.map_size(sizes=sizes, order=size_order, norm=size_norm)

File c:\Users\siranjeevi\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\relational.py:215, in _LinePlotter.__init__(self, data, variables, estimator, n_boot, seed, errorbar, sort, orient, err_style, err_kws, legend)
    201 def __init__(
    202     self, *,
    203     data=None, variables={},
   (...)
    209     # the kind of plot to draw, but for the time being we need to set
    210     # this information so the SizeMapping can use it
    211     self._default_size_range = (
    212         np.r_[.5, 2] * mpl.rcParams["lines.linewidth"]
    213     )
--> 215     super().__init__(data=data, variables=variables)
    217     self.estimator = estimator
    218     self.errorbar = errorbar

File c:\Users\siranjeevi\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_base.py:634, in VectorPlotter.__init__(self, data, variables)
    629 # var_ordered is relevant only for categorical axis variables, and may
    630 # be better handled by an internal axis information object that tracks
    631 # such information and is set up by the scale_* methods. The analogous
    632 # information for numeric axes would be information about log scales.
    633 self._var_ordered = {"x": False, "y": False}  # alt., used DefaultDict
--> 634 self.assign_variables(data, variables)
    636 # TODO Lots of tests assume that these are called to initialize the
    637 # mappings to default values on class initialization. I'd prefer to
    638 # move away from that and only have a mapping when explicitly called.
    639 for var in ["hue", "size", "style"]:

File c:\Users\siranjeevi\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_base.py:679, in VectorPlotter.assign_variables(self, data, variables)
    674 else:
    675     # When dealing with long-form input, use the newer PlotData
    676     # object (internal but introduced for the objects interface)
    677     # to centralize / standardize data consumption logic.
    678     self.input_format = "long"
--> 679     plot_data = PlotData(data, variables)
    680     frame = plot_data.frame
    681     names = plot_data.names

File c:\Users\siranjeevi\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_core\data.py:58, in PlotData.__init__(self, data, variables)
     51 def __init__(
     52     self,
     53     data: DataSource,
     54     variables: dict[str, VariableSpec],
     55 ):
     57     data = handle_data_source(data)
---> 58     frame, names, ids = self._assign_variables(data, variables)
     60     self.frame = frame
     61     self.names = names

File c:\Users\siranjeevi\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_core\data.py:232, in PlotData._assign_variables(self, data, variables)
    230     else:
    231         err += "An entry with this name does not appear in `data`."
--> 232     raise ValueError(err)
    234 else:
    235
    236     # Otherwise, assume the value somehow represents data
    237
    238     # Ignore empty data structures
    239     if isinstance(val, Sized) and len(val) == 0:

ValueError: Could not interpret value `Sales` for `y`. An entry with this name does not appear in `data`.
```

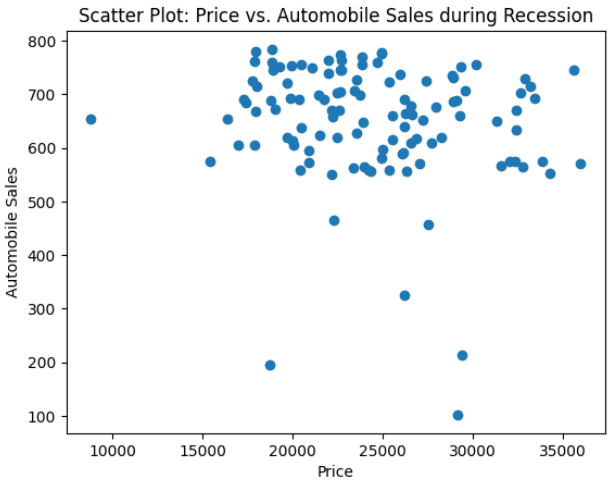




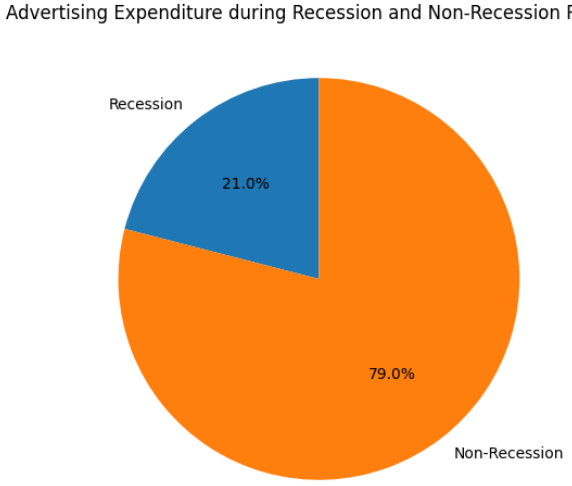
In [ ]:
```
non_rec_data = df[df['Recession'] == 0]

size = non_rec_data['Seasonality_Weight']  # for bubble effect

sns.scatterplot(data=non_rec_data, x='Month', y='Automobile_Sales', size=size, hue='Seasonality_Weight', legend=False)

plt.xlabel('Month')
plt.ylabel('Automobile Sales')
plt.title('Seasonality Impact on Automobile Sales')

plt.show()
```



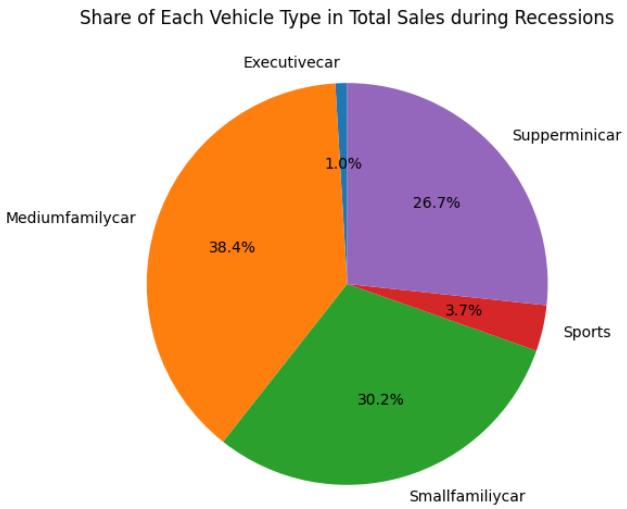
In [ ]:
```
# Create dataframes for recession and non-recession periods
rec_data = df[df['Recession'] == 1]

plt.scatter(rec_data['Price'], rec_data['Automobile_Sales'])

plt.xlabel('Price')
plt.ylabel('Automobile Sales')
plt.title('Scatter Plot: Price vs. Automobile Sales during Recession')

plt.show()
```



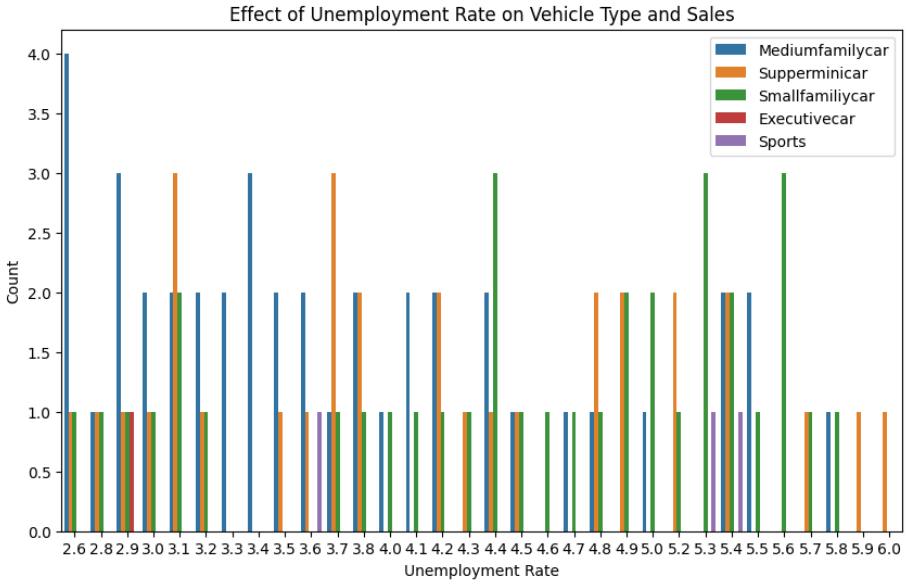
In [ ]:
```
# Filter the data
Rdata = df[df['Recession'] == 1]
NRdata = df[df['Recession'] == 0]

# Calculate the total advertising expenditure for both periods
RAtotal = Rdata['Advertising_Expenditure'].sum()
NRtotal = NRdata['Advertising_Expenditure'].sum()  # Fix the variable name here

# Create a pie chart for the advertising expenditure
plt.figure(figsize=(8, 6))

labels = ['Recession', 'Non-Recession']
sizes = [RAtotal, NRtotal]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Advertising Expenditure during Recession and Non-Recession Periods')

plt.show()
```



In [ ]:
```
# Filter the data
Rdata = df[df['Recession'] == 1]

# Calculate the sales volume by vehicle type during recessions
VTsales = Rdata.groupby('Vehicle_Type')['Advertising_Expenditure'].sum()

# Create a pie chart for the share of each vehicle type in total sales during recessions
plt.figure(figsize=(8, 6))

labels = VTsales.index
sizes = VTsales.values
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Share of Each Vehicle Type in Total Sales during Recessions')

plt.show()
```



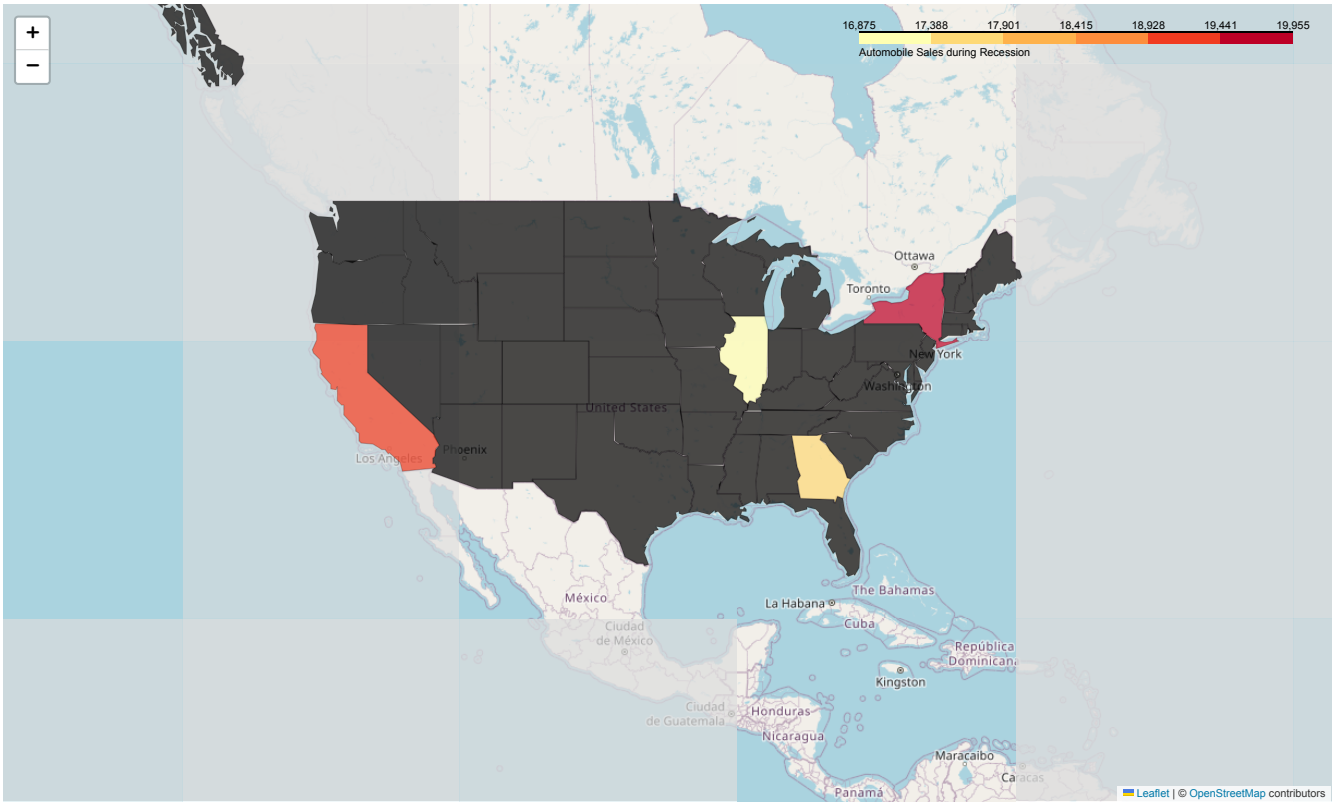
In [ ]:
```
data= df[df['Recession'] == 1]

plt.figure(figsize=(10, 6))

sns.countplot(data=data, x='unemployment_rate', hue='Vehicle_Type')

plt.xlabel('Unemployment Rate')
plt.ylabel('Count')
plt.title('Effect of Unemployment Rate on Vehicle Type and Sales')
plt.legend(loc='upper right')
plt.show()
```



In [ ]:
```
import requests
import folium

# Download GeoJSON file
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/us-states.json'
response = requests.get(url)
with open("us-states.json", "wb") as f:
    f.write(response.content)

# Filter the data for the recession period and specific cities
recession_data = data[data['Recession'] == 1]

# Calculate the total sales by city
sales_by_city = recession_data.groupby('City')['Automobile_Sales'].sum().reset_index()

# Create a base map centered on the United States
map1 = folium.Map(location=[37.0902, -95.7129], zoom_start=4)

# Create a choropleth layer using Folium
choropleth = folium.Choropleth(
    geo_data='us-states.json',  # GeoJSON file with state boundaries
    data=sales_by_city,
    columns=['City', 'Automobile_Sales'],
    key_on='feature.properties.name',
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Automobile Sales during Recession'
).add_to(map1)
```

```
# Add tooltips to the choropleth layer
choropleth.geojson.add_child(
    folium.features.GeoJsonTooltip(['name'], labels=True)
)

# Display the map
map1
```

Out[ ]:



In [ ]:
```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px

app = dash.Dash(__name__)

# Task 2.1 - Give your Dash application a meaningful title
app.title = "Statistical Analysis Dashboard"
```

In [ ]:
```
# Task 2.2 - Add drop-downs to your dashboard with appropriate titles and options
app.layout = html.Div
html.H1("Statistical Analysis Dashboard"),

dcc.Dropdown(
    id='statistic-dropdown',
    options=[
        {'label': 'Option 1', 'value': 'opt1'},
        {'label': 'Option 2', 'value': 'opt2'},
        # Add more options as needed
    ],
    value='opt1',  # Set a default value
    style={'width': '50%'}
),

dcc.Dropdown(
    id='year-dropdown',
    options=[
        {'label': '2020', 'value': '2020'},
        {'label': '2021', 'value': '2021'},
        # Add more options as needed
    ],
    value='2020',  # Set a default value
    style={'width': '50%'}
),
```

Out[ ]:
```
(Dropdown(options=[{'label': '2020', 'value': '2020'}, {'label': '2021', 'value': '2021'}], value='2020', style={'width': '50%'}, id='year-dropdown'),)
```

In [ ]:
```
# Task 2.3 - Add a division for output display with appropriate 'id' and 'classname' properties
html.Div(id='output-container', className='output-div'),
```

Out[ ]:
```
(Div(id='output-container', className='output-div'),)
```

In [ ]:
```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

app = dash.Dash(__name__)
app.title = "Statistical Analysis Dashboard"

# Sample data (replace this with your actual data)
```

```
data = pd.DataFrame({
    'Year': [2020, 2020, 2021, 2021],
    'Statistic': ['Opt1', 'Opt2', 'Opt1', 'Opt2'],
    'Value': [10, 15, 8, 12]
})

app.layout = html.Div([
    html.H1("Statistical Analysis Dashboard"),

    dcc.Dropdown(
        id='statistic-dropdown',
        options=[
            {'label': 'Option 1', 'value': 'Opt1'},
            {'label': 'Option 2', 'value': 'Opt2'},
        ],
        value='Opt1',  # Set a default value
        style={'width': '50%'}
    ),

    dcc.Dropdown(
        id='year-dropdown',
        options=[
            {'label': '2020', 'value': 2020},
            {'label': '2021', 'value': 2021},
        ],
        value=2020,  # Set a default value
        style={'width': '50%'}
    ),

    dcc.Graph(id='recession-graph'),

    html.Div(id='output-container', className='output-div'),
])

# Callback to update the graph
@app.callback(
    Output('recession-graph', 'figure'),
    [Input('statistic-dropdown', 'value'),
     Input('year-dropdown', 'value')]
)
def update_recession_graph(selected_statistic, selected_year):
    filtered_data = data[(data['Statistic'] == selected_statistic) & (data['Year'] == selected_year)]
    fig = px.bar(filtered_data, x='Statistic', y='Value', color='Statistic', barmode='group', title='Recession Report')
    return fig

# Callback to update the output container
@app.callback(
    Output('output-container', 'children'),
    [Input('statistic-dropdown', 'value'),
     Input('year-dropdown', 'value')]
)
def update_output(selected_statistic, selected_year):
    return f"Selected Statistic: {selected_statistic}, Selected Year: {selected_year}"

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