The Art and Science of Transportation Research in the AI Era

Data Visualization in Transportation with Plotly

Ali Bektas

Lecture Structure

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

#1 An overview on Plotly

#2 Basic Chart Types

#3 Geographic Visualizations

#1 An overview on Plotly

- **Plotly** is an open-source **Python** library for creating interactive and visually appealing data visualizations.
- It can handle complex datasets and **present** them in an engaging and user-friendly manner.
- Unlike static plots, **Plotly** visualizations allow for interaction such as **zooming**, **hovering over data points**, and **filtering** datasets directly within the chart.
- In **Transportation**, whether you are analyzing traffic flows, public transport efficiency or accident patterns, it is essential to present your findings in an interactive and visually engaging way.

#1 What will we learn today?

In this presentation, we will explore how to use Plotly to visualize transportation data effectively.

- We will start with **basic chart types** to understand the fundamentals.
- Then, we will move to **geographic visualizations**, showcasing how to map data interactively.
- Along the way, we will work with **real-world transportation datasets** to build insightful visualizations step by step.

#1 An overview on Plotly

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

There are two primary interfaces:

- **Plotly Express:** *plotly.express* is the high-level interface for creating simple plots quickly. It's designed for beginners or when you need a quick visualization.
- **Graph Objects:** *plotly.graph_objects* is the lower-level API for full customization, more control and advanced use cases.

```
import plotly.express as px

# List the built-in datasets for demonstration, educational and test purposes
print(dir(px.data))
```

```
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__
', 'carshare', 'election', 'election_geojson', 'experiment', 'gapminder', 'iris', 'medals_long', 'medals_wide', 'stocks', 'tips
', 'wind']
```

```
df = px.data.tips()

# List the columns of the data.tips()
print(df.columns)
```

```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'], dtype='object')
```

```
df = px.data.tips()

# List the first 5 rows of the data.tips()
print(df.head())
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

For more plotly.express.data sets: https://plotly.com/python-api-reference/generated/plotly.express.data.html

#2 Basic Chart Types: Understanding the Fundamentals

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

1. Line Charts: *px.line()*

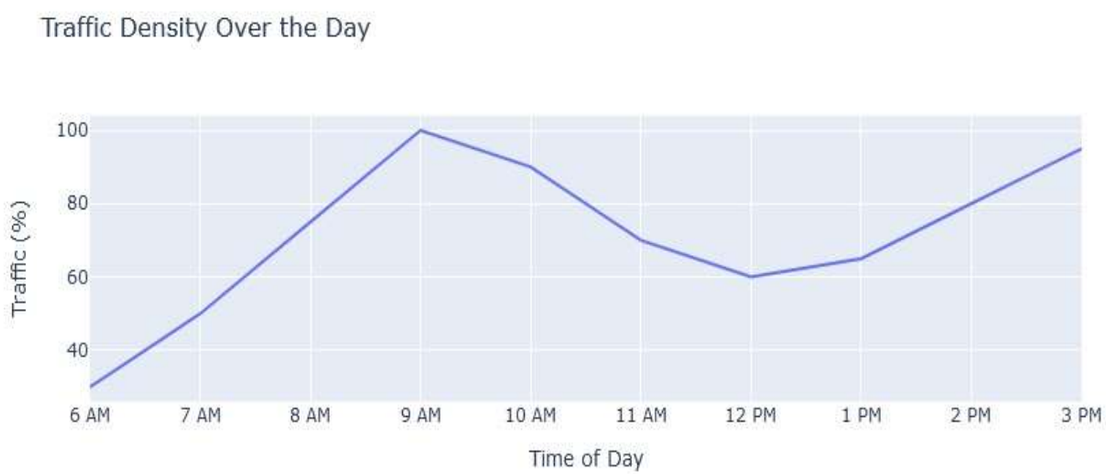
Ideal for time-series data, such as traffic flow over the course of a day or week.

```python
import plotly.express as px

# Create a Data Set
time_of_day = ["6 AM", "7 AM", "8 AM", "9 AM", "10 AM", "11 AM", "12 PM", "1 PM", "2 PM", "3 PM"]
traffic_density = [30, 50, 75, 100, 90, 70, 60, 65, 80, 95]

# Create Line Chart
fig = px.line(x=time_of_day, y=traffic_density,
              title="Traffic Density Over the Day",
              labels={"x": "Time of Day", "y": "Traffic (%)"})
fig.show()
```

Example: A line graph monitoring traffic density throughout the day:



#2 Basic Chart Types: Understanding the Fundamentals

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

2. Bar Charts: *px.bar()*

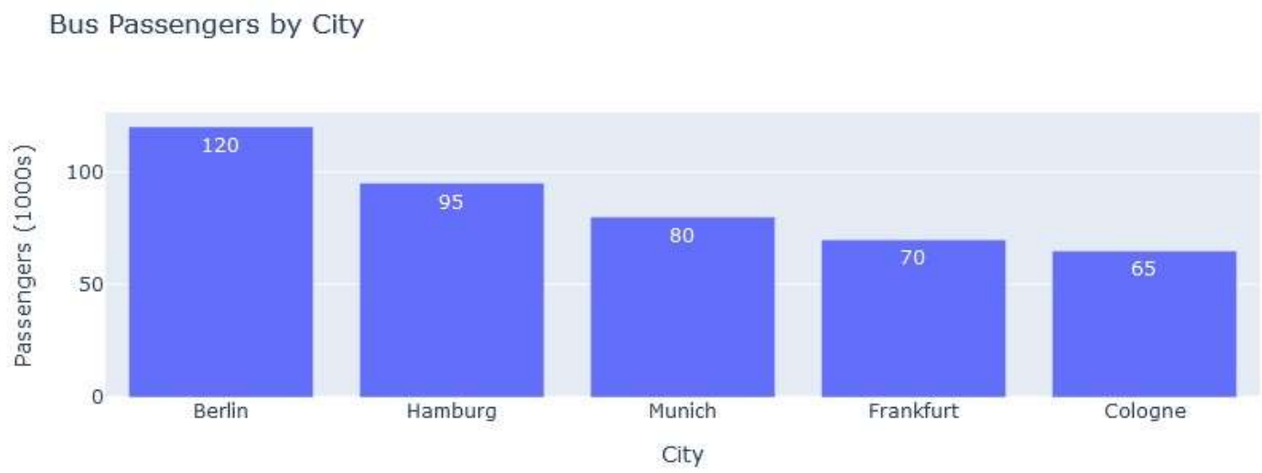
Best for comparing quantities, like accident counts by type or vehicle type proportions.

Example: Comparing bus passenger numbers across different cities.

```python
import plotly.express as px

# Example Data: cities and the numbers of passengers
cities = ["Berlin", "Hamburg", "Munich", "Frankfurt", "Cologne"]
passengers = [120, 95, 80, 70, 65]

# Create bar chart
fig = px.bar(x=cities, y=passengers,
             title="Bus Passengers by City",
             labels={"x": "City", "y": "Passengers (1000s)"},
             text=passengers)
fig.show()
```



#2 Basic Chart Types: Understanding the Fundamentals

TECHNISCHE UNIVERSITÄT DARMSTADT

IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

3. Scatter Plots: *px.scatter()*

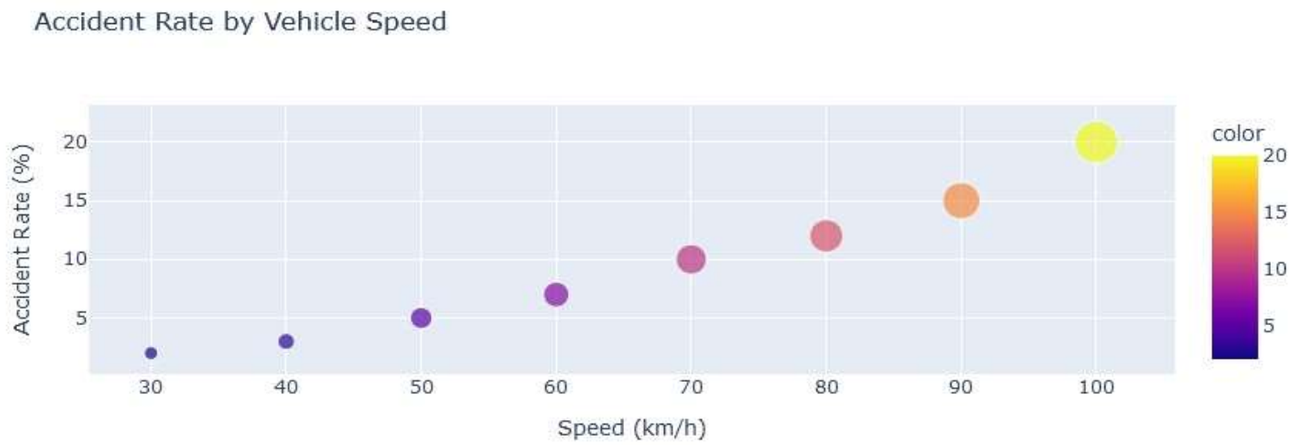
Perfect for examining relationships between variables, such as speed vs. accident severity.

Example: A scatter plot showing the relationship between vehicle speeds and accident rates:

```python
import plotly.express as px

# Data sample:
speeds = [30, 40, 50, 60, 70, 80, 90, 100]
accident_rates = [2, 3, 5, 7, 10, 12, 15, 20]

# Create a scatter plot
fig = px.scatter(x=speeds, y=accident_rates,
                 title="Accident Rate by Vehicle Speed",
                 labels={"x": "Speed (km/h)", "y": "Accident Rate (%)"},
                 size=accident_rates, color=accident_rates)
fig.show()
```



#2 Basic Chart Types: Understanding the Fundamentals

TECHNISCHE UNIVERSITÄT DARMSTADT

IVV Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

4. Pie Chart: *px.pie()*

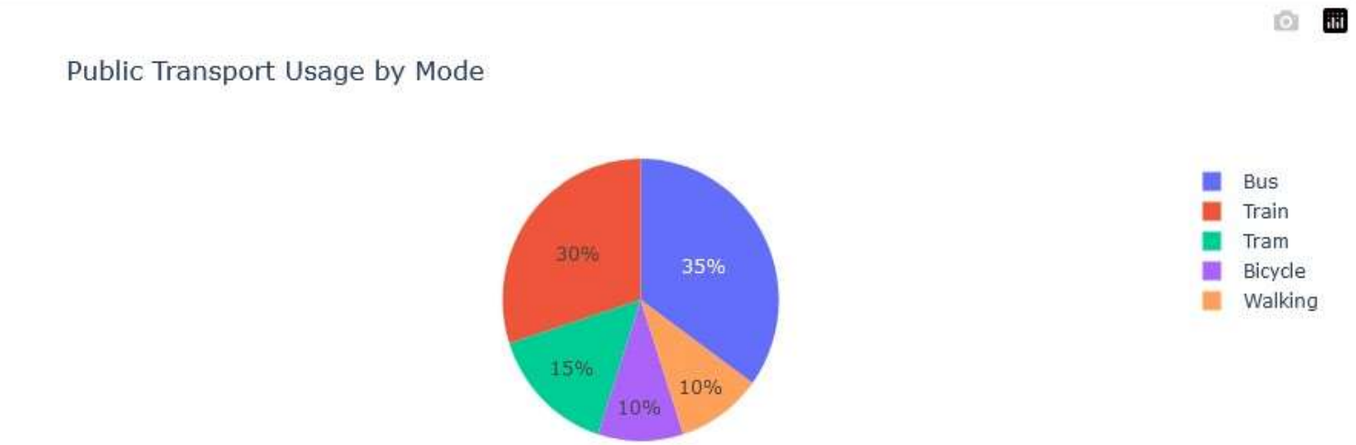
Useful for illustrating proportions, like the percentage of different modes of transport (e.g., buses, trains, bikes).

Example: A pie chart showing the usage rates of different types of public transport:

```python
import plotly.express as px

# Data Set
transport_modes = ["Bus", "Train", "Tram", "Bicycle", "Walking"]
usage_rates = [35, 30, 15, 10, 10]

# Create pie chart
fig = px.pie(names=transport_modes, values=usage_rates,
             title="Public Transport Usage by Mode")
fig.show()
```



#2 Basic Chart Types: Understanding the Fundamentals

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

5. Histogram: *px. histogram()*

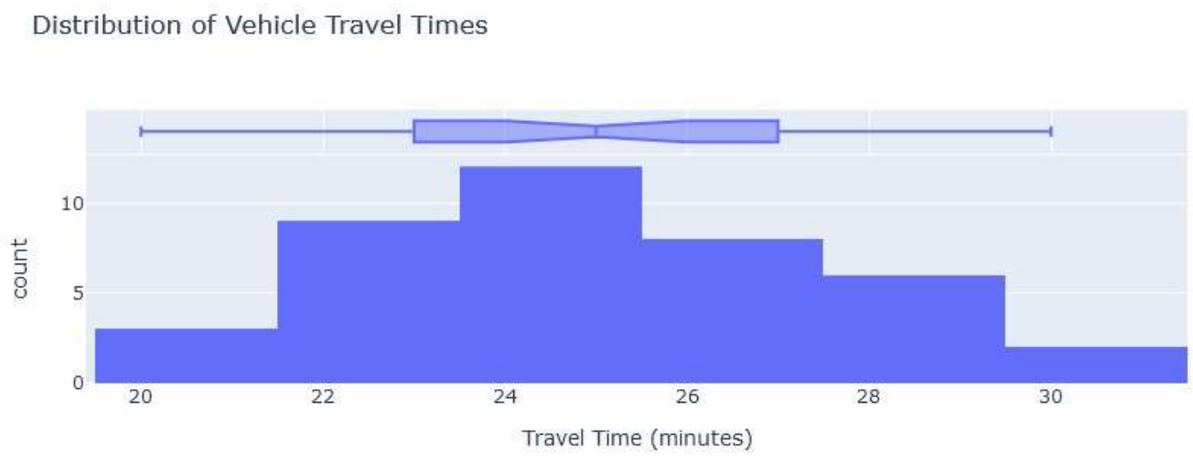
Examining the distribution of data (e.g. seeing how vehicle speeds are distributed over certain ranges.)

Example: A histogram showing the distribution of vehicle travel times in a city:

```python
import plotly.express as px

# Data set
travel_times = [20, 22, 25, 27, 24, 23, 30, 28, 26, 25, 21, 29, 22, 24, 23, 25, 27, 26, 24, 25,
                23, 28, 22, 26, 24, 21, 30, 29, 25, 27, 23, 24, 25, 28, 26, 22, 27, 23, 24, 29]

# Create a histogram
fig = px.histogram(x=travel_times, nbins=8,
                   title="Distribution of Vehicle Travel Times",
                   labels={"x": "Travel Time (minutes)"},
                   marginal="box")  # Displays the box plot above the histogram
fig.show()
```



#3 Geographic Visualizations: Mapping Data Interactively

Geographic visualizations are crucial in transportation research, as spatial analysis provides valuable insights into traffic patterns, accident hotspots, or public transit coverage.

Key Features of Geographic Visualizations in Plotly:

- **Interactive and Intuitive:** Users can zoom, pan, and hover over data points to reveal additional information, making the visualizations highly engaging.
- **Versatility:** Supports various map types, including scatter plots over maps, choropleth maps (color-coded areas), and line maps for routes.
- **Built-in Mapbox and OpenStreetMap Integration:** Allows seamless integration with Mapbox or OpenStreetMap, providing high-quality basemaps for geographic data.
- **Customization:** Options for adjusting map projections, markers, and color scales to suit the data being displayed.

#3.1 Scatter Maps:

Scatter Maps are used to display specific data points on a map. Each point is positioned based on its latitude and longitude values, and often additional attributes like the point's size or color are used to convey more information.

Example: This example shows bus stops in four cities on a map, and when you hover over a point, the city name appears.

```python
import plotly.express as px

# Example data: Bus stops
locations = [(51.5074, -0.1278), (52.5200, 13.4050), (48.8566, 2.3522), (40.7128, -74.0060)]
cities = ["London", "Berlin", "Paris", "New York"]

# Scatter map
fig = px.scatter_geo(locations, lat=[loc[0] for loc in locations], lon=[loc[1] for loc in locations],
                     text=cities, title="Bus Stops Locations on the Map")
fig.show()
```

Explanation:

scatter_geo: This function is used to plot data points on a map.

lat and *lon*: These parameters specify the latitude and longitude coordinates for each point on the map.

text: Adds additional information to each point, such as the city name, that appears when you hover over the point.

#3.2 Choropleth Maps:

Choropleth Maps colorize geographical regions to show the distribution of values across areas. These maps are typically used to visualize density or distribution data.

Example: The map color-codes regions based on the traffic congestion index. Cities with more traffic congestion will appear in darker shades, helping the viewer quickly assess the areas with the most intense traffic issues.

Explanation:

locations: Specifies the geographical regions (here, countries represented by their ISO country codes).

color: The congestion index, which reflects the level of traffic congestion. Higher values indicate more congestion.

color_continuous_scale: Uses the "Reds" color scale, where lighter shades represent lower congestion and darker reds represent higher congestion levels.

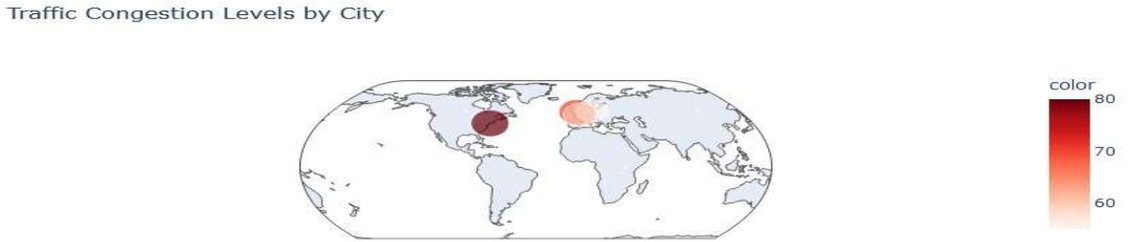
hover_name: Displays the city names when you hover over each region on the map.

```python
import plotly.express as px

# Example data: Traffic congestion levels (index values for different cities)
cities = ["London", "Berlin", "Paris", "New York"]
congestion_index = [70, 55, 60, 80]  # Higher value indicates more congestion
locations = [
    (51.5074, -0.1278),  # London
    (52.5200, 13.4050),  # Berlin
    (48.8566, 2.3522),   # Paris
    (40.7128, -74.0060)  # New York
]

# Scatter map for traffic congestion
fig = px.scatter_geo(
    lat=[loc[0] for loc in locations],  # Latitude values
    lon=[loc[1] for loc in locations],  # Longitude values
    color=congestion_index,  # Congestion index for color
    hover_name=cities,  # City names for hover info
    color_continuous_scale="Reds",  # Color scale
    size=[index / 10 for index in congestion_index],  # Adjust size for better visualization
    title="Traffic Congestion Levels by City"
)

fig.update_geos(projection_type="natural earth")
fig.show()
```



#3.3 Density Heatmaps:

Density Heatmaps visualize the concentration of data points on a map. In these maps, areas with higher concentrations of points are shown in brighter colors, which is particularly useful for visualizing traffic density, incident hotspots, or high-frequency locations.

Example: This heatmap shows areas of higher traffic density, with more congested areas appearing in brighter colors.

Explanation:

density_mapbox: This function is used to create a density heatmap, where data points are color-coded based on their intensity (e.g., traffic density).

z: Represents the intensity or density of each point. Higher values might indicate more traffic or more incidents.

radius: Controls the size of the influence of each point. Larger values spread the impact of each point over a wider area.

mapbox_style: Specifies the style of the map (e.g., "carto-positron" provides a light gray background; «open-street-map", which is a free map style and doesn't require a Mapbox token.).

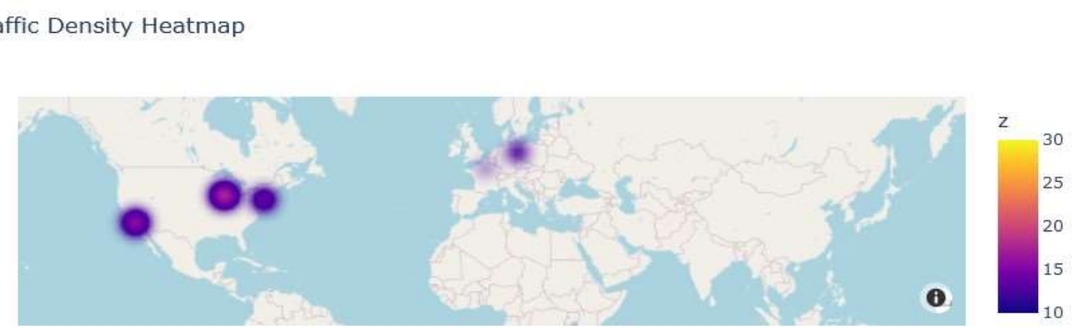
```python
import plotly.express as px

# Example data: Traffic density (locations and intensity)
locations = [(51.5074, -0.1278),  # London
             (52.5200, 13.4050),  # Berlin
             (48.8566, 2.3522),   # Paris
             (40.7128, -74.0060), # New York
             (34.0522, -118.2437),# Los Angeles
             (41.8781, -87.6298)] # Chicago

intensity = [10, 15, 12, 20, 25, 30]  # Traffic density values

# Heatmap generation using Open Street Map
fig = px.density_mapbox(lat=[loc[0] for loc in locations],
                        lon=[loc[1] for loc in locations],
                        z=intensity,
                        radius=20,
                        title="Traffic Density Heatmap",
                        mapbox_style="open-street-map")  # Free, tokenless map style

fig.show()
```



- **THANK YOU**
- **DANKE**