Final Project - Waze Shiny Dashboard

Surya Hardiansyah and Astari Raihanah

2024-12-01

```
# Create the data directory
data_dir = os.path.join(os.getcwd(), "data")
os.makedirs(data_dir, exist_ok=True)

# API endpoints
carjacking_url = "https://data.cityofchicago.org/resource/t66u-yzsn.json"
neighborhood_url =
 ↪  "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

# File paths
carjacking_file = os.path.join(data_dir, "chicago_carjackings.csv")
neighborhood_file = os.path.join(data_dir, "chicago_neighborhoods.geojson")
processed_file = os.path.join(data_dir, "processed_carjackings.csv")
```

Data Wrangling

Step 1: Download all carjacking data using pagination

```
limit = 1000  # Maximum rows per request announced in the webpage
offset = 0  # Start at the beginning
all_data = []  # List to store all rows

while True:
    # Add pagination parameters to the API request
    params = {"$limit": limit, "$offset": offset}
    response = requests.get(carjacking_url, params=params)
    if response.status_code != 200:
        print(f"Failed to fetch carjacking data. Status code:
         ↪  {response.status_code}")
        break

    # Parse the JSON response
    chunk = response.json()

    # Stop the loop if no more data is returned
    if not chunk:
        break

    all_data.extend(chunk)  # Append the fetched rows
    offset += limit  # Increment the offset for the next request
    print(f"Fetched {len(chunk)} rows. Total rows so far: {len(all_data)}")

# Convert the collected data to a DataFrame
if all_data:
    carjacking_data = pd.DataFrame(all_data)
    carjacking_data.to_csv(carjacking_file, index=False)
    print(f"All carjacking data saved to {carjacking_file}")
else:
    print("No data was downloaded.")
```

```
Fetched 1000 rows. Total rows so far: 1000
Fetched 1000 rows. Total rows so far: 2000
Fetched 1000 rows. Total rows so far: 3000
Fetched 1000 rows. Total rows so far: 4000
Fetched 1000 rows. Total rows so far: 5000
Fetched 1000 rows. Total rows so far: 6000
Fetched 1000 rows. Total rows so far: 7000
Fetched 1000 rows. Total rows so far: 8000
Fetched 1000 rows. Total rows so far: 9000
Fetched 1000 rows. Total rows so far: 10000
Fetched 1000 rows. Total rows so far: 11000
Fetched 1000 rows. Total rows so far: 12000
Fetched 1000 rows. Total rows so far: 13000
Fetched 1000 rows. Total rows so far: 14000
Fetched 1000 rows. Total rows so far: 15000
Fetched 1000 rows. Total rows so far: 16000
Fetched 1000 rows. Total rows so far: 17000
Fetched 1000 rows. Total rows so far: 18000
Fetched 1000 rows. Total rows so far: 19000
Fetched 1000 rows. Total rows so far: 20000
Fetched 1000 rows. Total rows so far: 21000
Fetched 1000 rows. Total rows so far: 22000
Fetched 196 rows. Total rows so far: 22196
All carjacking data saved to D:\UCHICAGO\DATA ANALYSIS PYTHON
II\ppha30538-final-project\data\chicago_carjackings.csv
```

Step 2: Download neighborhood boundaries

```
response = requests.get(neighborhood_url)
if response.status_code == 200:
    with open(neighborhood_file, "wb") as f:
        f.write(response.content)
    print(f"Neighborhood boundaries saved to {neighborhood_file}")
else:
    print(f"Failed to download neighborhood boundaries. Status code:
     ↪  {response.status_code}")
```

```
Neighborhood boundaries saved to D:\UCHICAGO\DATA ANALYSIS PYTHON
II\ppha30538-final-project\data\chicago_neighborhoods.geojson
```

Step 3: Load datasets

```
carjackings = pd.read_csv(carjacking_file)
neighborhoods = gpd.read_file(neighborhood_file)

# Ensure carjackings have proper datetime parsing
if "date" in carjackings.columns:
    carjackings["date"] = pd.to_datetime(carjackings["date"],
 ↪  errors="coerce")
```

Step 4: Spatial join to associate carjackings with neighborhoods

```
# Convert carjacking data to GeoDataFrame
if "latitude" in carjackings.columns and "longitude" in carjackings.columns:
    carjackings = gpd.GeoDataFrame(
        carjackings,
        geometry=gpd.points_from_xy(carjackings.longitude,
 ↪  carjackings.latitude),
        crs=neighborhoods.crs
    )

# Perform spatial join
carjackings_with_neighborhoods = gpd.sjoin(carjackings, neighborhoods,
 ↪  how="left", predicate="intersects")
```

Step 5: Add new columns

```
if "date" in carjackings_with_neighborhoods.columns:
    carjackings_with_neighborhoods["month"] =
 ↪  carjackings_with_neighborhoods["date"].dt.month
    carjackings_with_neighborhoods["year"] =
 ↪  carjackings_with_neighborhoods["date"].dt.year
    carjackings_with_neighborhoods["time_of_day"] =
 ↪  carjackings_with_neighborhoods["date"].dt.hour.apply(
        lambda x: "Morning" if 6 <= x < 12 else "Afternoon" if 12 <= x < 18
         ↪  else "Evening"
    )
```

Step 5.1: Add binned latitude and longitude

```
# Round latitude and longitude to 2 decimal places
carjackings_with_neighborhoods["binned_latitude"] =
 ↪  carjackings_with_neighborhoods["latitude"].round(2)
carjackings_with_neighborhoods["binned_longitude"] =
 ↪  carjackings_with_neighborhoods["longitude"].round(2)
print("Binned latitude and longitude columns added.")
```

```
Binned latitude and longitude columns added.
```

Step 6: Save raw and processed datasets

```
carjackings_with_neighborhoods.to_csv(processed_file, index=False)
print(f"Processed carjackings data saved to {processed_file}")
```

```
Processed carjackings data saved to D:\UCHICAGO\DATA ANALYSIS PYTHON
II\ppha30538-final-project\data\processed_carjackings.csv
```

Static Plot 1: Choropleth Map of Carjacking Incidents by Neighborhood

Step 1: Load data

```
carjacking_file = "data/processed_carjackings.csv"
neighborhood_file = "data/chicago_neighborhoods.geojson"

carjackings = pd.read_csv(carjacking_file)
neighborhoods = gpd.read_file(neighborhood_file)
```

Step 2: Aggregate carjacking incidents by neighborhood

```
carjacking_counts =
 ↪  carjackings.groupby("pri_neigh").size().reset_index(name="count")
```

Step 3: Merge counts with neighborhood boundaries

```
neighborhoods = neighborhoods.merge(carjacking_counts, left_on="pri_neigh",
 ↪  right_on="pri_neigh", how="left")
neighborhoods["count"] = neighborhoods["count"].fillna(0)
```

Step 4: Prepare GeoJSON for Altair

```
geojson_data = json.loads(neighborhoods.to_json())
```

Step 5: Create Choropleth Map

```
choropleth =
 ↪  alt.Chart(alt.Data(values=geojson_data["features"])).mark_geoshape().encode(
    color=alt.Color("properties.count:Q", title="Carjacking Count"),
    tooltip=["properties.neighborhood:N", "properties.count:Q"]
).properties(
    title="Choropleth Map of Carjacking Incidents by Neighborhood",
    width=600,
    height=400
).project(
    type="equirectangular"
)
```

Step 6: Save to PNG

```
pictures_dir = os.path.join(os.getcwd(), "pictures")
os.makedirs(pictures_dir, exist_ok=True)
choropleth.save(os.path.join(pictures_dir, "choropleth_map.png"))

print("Choropleth map saved to pictures/choropleth_map.png")
```

```
Choropleth map saved to pictures/choropleth_map.png
```

Static Plot 2: Line Chart Showing Carjacking Trends Over Time

Step 1: Extract month and year

```
carjackings["date"] = pd.to_datetime(carjackings["date"], errors="coerce")
carjackings["year_month"] = carjackings["date"].dt.to_period("M")
```

Step 2: Aggregate counts by time

```
time_trends =
 ↪  carjackings.groupby("year_month").size().reset_index(name="count")
time_trends["year_month"] = time_trends["year_month"].astype(str)
```

Step 3: Create Line Chart

```
line_chart = alt.Chart(time_trends).mark_line(point=True).encode(
    x=alt.X("year_month:T", title="Time (Year-Month)"),
    y=alt.Y("count:Q", title="Number of Carjackings"),
    tooltip=["year_month:T", "count:Q"]
).properties(
    title="Carjacking Trends Over Time",
    width=800,
    height=400
)
```

Step 4: Save to PNG

```
line_chart.save(os.path.join(pictures_dir, "line_chart.png"))

print("Line chart saved to pictures/line_chart.png")
```

```
Line chart saved to pictures/line_chart.png
```