Data Stormers Data Cleaning Document

**Introduction & Objective.**

This document provides a description of steps used in gathering and cleaning data. The objective of our project is to understand Tornado trends and analyzing the financial impact of tornadoes across the United States.

**Data Sources.**

***NOAA's Severe Weather Data Inventory (SWDI) API***

**Fetch and process data**

Key Components:

* Datasets: Currently, only the nx3tvs dataset is specified, which corresponds to NEXRAD Level-3 Tornado Vortex Signatures.
* Output Format: Data is requested in JSON format (outputFormat = "json").
* Date Range:The data is queried for the period from July 1, 2024, to July 31, 2024 (daterange = "20240701:20240731").

Process:

1. URL Construction: A base URL is created for each dataset using the format https://www.ncdc.noaa.gov/swdiws/{outputFormat}/{dataset}/{daterange}.
2. Data Fetching: The fetch\_api\_data() function is used to retrieve data from the constructed URL.
3. Data Handling:
   * If data is successfully fetched and contains results, it is written to a CSV file named swdiws\_{dataset}.csv.
   * If no data is found or the response format is invalid, an error message is printed.

***FEMA's OpenFEMA API.***

**Fetch and process data on tornado-related disaster declarations**

API Request:

* + Constructs the API request URL for the DisasterDeclarationsSummaries endpoint at https://www.fema.gov/api/open/v2/.
  + Filters the data to include only incidents where incidentType is 'Tornado'.

Data Retrieval and Saving:

* + Fetches data from the API.
  + Converts the data into a DataFrame.
  + Saves the DataFrame to a CSV file named DisasterDeclarationsSummaries.csv.

**Fetch and process data on housing assistance owners related to previously identified tornado disaster declarations**

API Request:

* + Constructs the API request URL for the DisasterDeclarationsSummaries endpoint at https://www.fema.gov/api/open/v2/.
  + Filters the data to include only incidents where incidentType is 'Tornado'.

Data Retrieval and Saving:

* + Fetches data from the API.
  + Converts the data into a DataFrame.
  + Saves the DataFrame to a CSV file named DisasterDeclarationsSummaries.csv

***United States Census Bureau API***

API Request and Parse to JSON data:

* Retrieve the Census API key from the configuration
* Check if the API key was found in the configuration
* Set up parameters for the API request
* Correct URL to fetch population data for all states (2020 Census data)

Data Retrieval and Saving:

* Fetch data from the API using the base URL and parameters
* Parse JSON data
* Convert the remaining JSON data into a DataFrame

**Data Cleaning Steps**

* Standardize column names to lowercase
  + swdiws\_tornado\_data\_df**.**columns **=** [col**.**lower() **for** col **in** swdiws\_tornado\_data\_df**.**columns]
* Dropped Cell and Cell ID columns
  + swdiws\_tornado\_data\_df**.**drop(columns**=**['cell\_type', 'cell\_id',], inplace**=True**)
* Renamed columns
  + max\_shear = change in wind speed and direction with height in the atmosphere
  + wsr\_id = weather stations
  + mxdv = maximum difference in velocity, particularly within areas of rotation
  + ztime = time of the event
  + azimuth = direction in which the tornado is moving
* Convert 'ztime' column to datetime
  + swdiws\_tornado\_data\_df['event\_time'] **=** pd**.**to\_datetime(swdiws\_tornado\_data\_df['event\_time'])
* Remove duplicates
  + swdiws\_tornado\_data\_df**.**drop\_duplicates(inplace**=True**)
* Display the cleaned dataframe
  + swdiws\_tornado\_data\_df**.**head()
* Convert declarationDate to datetime
  + disaster\_declarations\_df['declarationDate'] **=** pd**.**to\_datetime(disaster\_declarations\_df['declarationDate'])
* Group by year
  + yearly\_declarations **=** disaster\_declarations\_df**.**groupby(disaster\_declarations\_df['declarationDate']**.**dt**.**year)**.**size()
* Aggregate damage data by state
  + state\_damage **=** housing\_assistance\_df**.**groupby('state')['totalDamage']**.**sum()**.**sort\_values(ascending**=False**)
* Aggregate distribution of assistance amounts
  + assistance\_distribution **=** housing\_assistance\_df[['totalApprovedIhpAmount', 'repairReplaceAmount', 'rentalAmount', 'otherNeedsAmount']]**.**sum()
* Convert start and end dates to datetime
  + disaster\_declarations\_df['incidentBeginDate'] **=** pd**.**to\_datetime(disaster\_declarations\_df['incidentBeginDate'])
  + disaster\_declarations\_df['incidentEndDate'] **=** pd**.**to\_datetime(disaster\_declarations\_df['incidentEndDate'])
* Merge with housing data to include total damage
  + merged\_duration\_damage **=** pd**.**merge(disaster\_declarations\_df[['disasterNumber', 'durationDays']], housing\_assistance\_df[['disasterNumber', 'totalDamage']], on**=**'disasterNumber')
* Categorize the disaster duration into bins for better visualization
  + merged\_duration\_damage['durationCategory'] **=** pd**.**cut(merged\_duration\_damage['durationDays'],
  + bins**=**[0, 3, 7, 14, 30, 100],
  + labels**=**['0-3 days', '4-7 days', '8-14 days', '15-30 days', '>30 days'])
* Merge the binned data with the disaster data again
  + housing\_income\_df **=** pd**.**merge(housing\_assistance\_df, disaster\_declarations\_df, on**=**'disasterNumber', how**=**'inner')
* Merge the binned data with the disaster data again
  + housing\_income\_df **=** pd**.**merge(housing\_assistance\_df, disaster\_declarations\_df, on**=**'disasterNumber', how**=**'inner')
* Convert the necessary date columns to datetime format
  + housing\_income\_df['declarationDate'] **=** pd**.**to\_datetime(housing\_income\_df['declarationDate'])
  + housing\_income\_df['lastIAFilingDate'] **=** pd**.**to\_datetime(housing\_income\_df['lastIAFilingDate'])
* Calculate the time taken to provide assistance
  + housing\_income\_df['daysToAssistance'] **=** (housing\_income\_df['lastIAFilingDate'] **-** housing\_income\_df['declarationDate'])**.**dt**.**day

**Data Visualization Steps**

Heat Map

*# Check if the DataFrame is not empty*

**if** swdiws\_tornado\_data\_df**.**empty:

print("The DataFrame is empty. No data to plot.")

**else**:

*# Create a base map centered around the mean latitude and longitude*

base\_map **=** folium**.**Map(location**=**[swdiws\_tornado\_data\_df['latitude']**.**mean(), swdiws\_tornado\_data\_df['longitude']**.**mean()], zoom\_start**=**6)

*# Prepare data for the heatmap*

heat\_data **=** [[row['latitude'], row['longitude']] **for** index, row **in** swdiws\_tornado\_data\_df**.**iterrows()]

*# Add the heatmap to the base map*

HeatMap(heat\_data)**.**add\_to(base\_map)

*# Display the map inline*

display(base\_map)

**except** Exception **as** e:

print(f"An error occurred: {e}")

*# Aggregate data by state*

state\_damage **=** disaster\_declarations\_df**.**groupby('state')**.**size()**.**sort\_values(ascending**=False**)

*# Plot the bar chart*

plt**.**figure(figsize**=**(12, 6))

sns**.**barplot(x**=**state\_damage**.**index, y**=**state\_damage**.**values, color**=**'steelblue') *# Removed palette, added color*

plt**.**title('Number of Tornadoes by State')

plt**.**xlabel('State')

plt**.**ylabel('Number of Tornadoes')

plt**.**xticks(rotation**=**90)

plt**.**tight\_layout()

plt**.**show()

Bar Chart: Number of Tornadoes by State

*# Aggregate data by state*

state\_damage **=** disaster\_declarations\_df**.**groupby('state')**.**size()**.**sort\_values(ascending**=False**)

*# Plot the bar chart*

plt**.**figure(figsize**=**(12, 6))

sns**.**barplot(x**=**state\_damage**.**index, y**=**state\_damage**.**values, color**=**'steelblue') *# Removed palette, added color*

plt**.**title('Number of Tornadoes by State')

plt**.**xlabel('State')

plt**.**ylabel('Number of Tornadoes')

plt**.**xticks(rotation**=**90)

plt**.**tight\_layout()

plt**.**show()

Line Plot

*# Convert declarationDate to datetime*

disaster\_declarations\_df['declarationDate'] **=** pd**.**to\_datetime(disaster\_declarations\_df['declarationDate'])

*# Group by year*

yearly\_declarations **=** disaster\_declarations\_df**.**groupby(disaster\_declarations\_df['declarationDate']**.**dt**.**year)**.**size()

*# Plot*

plt**.**figure(figsize**=**(12, 6))

yearly\_declarations**.**plot(kind**=**'line', marker**=**'o', color**=**'b')

plt**.**xlabel('Year')

plt**.**ylabel('Number of Declarations')

plt**.**title('Disaster Declarations Over Time')

plt**.**grid(**True**)

plt**.**tight\_layout()

plt**.**show()

Bar Chart: Total Housing Damage by State

*# Aggregate damage data by state*

state\_damage **=** housing\_assistance\_df**.**groupby('state')['totalDamage']**.**sum()**.**sort\_values(ascending**=False**)

*# Plot the bar chart*

plt**.**figure(figsize**=**(12, 6))

sns**.**barplot(x**=**state\_damage**.**index, y**=**state\_damage**.**values, color**=**'steelblue') *# Removed palette, added color*

plt**.**title('Total Housing Damage by State')

plt**.**xlabel('State')

plt**.**ylabel('Total Damage ($)')

plt**.**xticks(rotation**=**90)

plt**.**tight\_layout()

plt**.**show()

Pie Chart

*# Aggregate distribution of assistance amounts*

assistance\_distribution **=** housing\_assistance\_df[['totalApprovedIhpAmount', 'repairReplaceAmount', 'rentalAmount', 'otherNeedsAmount']]**.**sum()

*# Plot the pie chart*

plt**.**figure(figsize**=**(10, 7))

assistance\_distribution**.**plot(kind**=**'pie', autopct**=**'%1.1f%%', colors**=**sns**.**color\_palette('viridis', len(assistance\_distribution)))

plt**.**ylabel('') *# Removes the y-label*

plt**.**title('Distribution of Housing Assistance Amounts')

plt**.**tight\_layout()

plt**.**show()

Bubble Chart

*# Calculate the linear regression*

slope, intercept, r\_value, p\_value, std\_err **=** stats**.**linregress(merged\_data['population'], merged\_data['severity'])

*# Print the regression results*

print(f"Slope: {slope}")

print(f"Intercept: {intercept}")

print(f"R-squared: {r\_value**\*\***2}")

print(f"P-value: {p\_value}")

print(f"Standard Error: {std\_err}")

*# Optionally, plot the regression line on the scatter plot (as shown earlier)*

plt**.**figure(figsize**=**(14, 8))

*# Plotting the bubbles*

sns**.**scatterplot(x**=**'population', y**=**'severity',

size**=**'severity',

sizes**=**(20, 2000), *# Adjust size scale for better visibility*

hue**=**'severity',

palette**=**'viridis',

data**=**merged\_data, alpha**=**0.6, legend**=False**)

*# Add the regression line*

plt**.**plot(merged\_data['population'], intercept **+** slope **\*** merged\_data['population'], color**=**'red', label**=**f'Regression Line (R² = {r\_value**\*\***2:.2f})')

*# Annotate each point with the state abbreviation*

**for** i **in** range(merged\_data**.**shape[0]):

plt**.**text(merged\_data['population']**.**iloc[i], merged\_data['severity']**.**iloc[i],

merged\_data['state']**.**iloc[i], fontsize**=**9, ha**=**'right')

plt**.**xlabel('Population')

plt**.**ylabel('Tornado Severity (Number of Declarations)')

plt**.**title('Bubble Chart: Population vs Tornado Severity by State')

plt**.**grid(**True**)

plt**.**legend()

plt**.**show()

Polar Chart

*# Create the polar chart with vibrant colors*

plt**.**figure(figsize**=**(8, 8))

ax **=** plt**.**subplot(111, polar**=True**)

*# Plot the data with vibrant colors*

ax**.**fill(angles, values, color**=**colors[0], alpha**=**0.4)

ax**.**plot(angles, values, color**=**colors[0], linewidth**=**2)

ax**.**set\_yticklabels([])

ax**.**set\_xticks(angles[:**-**1])

ax**.**set\_xticklabels(categories, color**=**'darkblue')

*# Adding data labels to each point with vibrant color*

**for** i **in** range(len(values) **-** 1):

ax**.**text(angles[i], values[i] **+** 5, f'{values[i]}', horizontalalignment**=**'center', size**=**12, color**=**colors[i])

*# Enhance grid lines for visibility with a new color*

ax**.**grid(**True**, which**=**'major', color**=**'lightblue', linestyle**=**'--', linewidth**=**1)

plt**.**title('Polar Chart: Vibrant Frequency of Tornado-Related Disasters by Month', size**=**15, color**=**'darkblue', y**=**1.1)

plt**.**tight\_layout()

Line Plot with Regression

*# Generate the line plot with regression for valid registrations vs. total approved housing assistance*

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

sns**.**regplot(x**=**valid\_registrations\_vs\_assistance['validRegistrations'],

y**=**valid\_registrations\_vs\_assistance['totalApprovedIhpAmount'],

scatter\_kws**=**{'s': 10}, line\_kws**=**{"color":"red"})

plt**.**title('Line Plot with Regression: Valid Registrations vs. Total Approved Housing Assistance')

plt**.**xlabel('Number of Valid Registrations')

plt**.**ylabel('Total Approved Housing Assistance (IHP Amount)')

plt**.**grid(**True**)

plt**.**tight\_layout()