Milestone 2 - Location Data EDA and Pipeline Description

- **State Crash Data**
  - There was no missing data for the variables of interest in this dataset
  - We are not using any non-numerical features in this dataset
  - Raw Features:
    - Crash ID
    - Crash Year
    - Report State
    - Fatal Count
    - Fatalities
    - Injuries
    - Vehicles In Accident
    - City
    - City Code
    - State
    - Location
    - Crash Date
    - Year
  - Transformations (`yearly_crash_data_processing.ipynb`):
    - Taking a dataset that contains each individual crash event on each dat between 2023 and 2025 and group it by four columns:
      - Crash Year
      - Crash Month
      - State
      - County Code
    - For each unique combination of these four values, we calculate:
      - total number of crashes by counting each row as one crash)
      - total number of fatalities across all crashes by summing the "Fatalities" column after converting it to integer type
      - total number of injuries across all crashes by summing the "Injuries" column after converting it to integer type
      - total number of vehicles involved across all crashes by summing the "Vehicles In Accident" column after converting it to integer type

    - Derived Features:
        - Total number of crashes
        - Total number of fatalities across all crashes
        - Total number of injuries across all crashes
        - Total number of vehicles involved across all crashes
    - Examples:
        - We would go from having a list of all the crashes that happened on each day and all of the associated information from that crash to a dataframe where one row represents a summary of truck and bus crashes in a given month and year in a specific country and state (eg. total number of fatalities in Adams County, Iowa in June 2024)
    - Output: `state_crash_monthly_county_counts`
- **Precipitation Data**
    - Raw Features
        - Zip Code
        - Latitude
        - Longitude
        - Total precipitation (rain+melted snow) for each Date in mm
    - Used as input in aggregations, no direct transformations made to features
    - Not missing data within extracted fiels
    - We are not using any non-numerical features in this dataset
    - Output:
        - 2023_precipitation_data
        - 2024_precipitation_data
        - 2025_precipitation_data
- **Zip Code Shapefile**
    - Raw Features
        - geographic features as points, lines, or polygons composed of related files, including `.shp` (geometry), `.shx` (index), and `.dbf` (attributes
    - Transformations
        - Transform CRS for Accurate Distance Calculation
        - Check and transform the shapefile's Coordinate Reference System (CRS) to EPSG:5070 for accurate distance measurements. Convert centroids to EPSG:4326 for latitude/longitude.
        - We are not using any non-numerical features in this dataset
    - Derived Features
        - Zip Code

        - Latitude
        - Longitude
    - Create Target ZIP Code Lookup: Filter the shapefile for the target ZIP codes from the regional site data manually add any missing ZIP codes with known latitude and longitude.
- **Fusion Site Data**
    - Raw Features
        - ID
        - Brand
        - State
        - City
        - Latitude
        - Longitude
        - County
        - Zip
    - Transformations (`site_radius_by_zipcode.ipynb`)
        - Cross join zip code data with the target ZIP code DataFrame to generate all combinations of site and target ZIP codes.
        - Add Target Coordinates to DataFrame: For each target ZIP code, add its latitude and longitude as literal columns to the DataFrame for easy access in distance calculations.
        - Calculate Distance Using Haversine Formula
            - Apply the haversine_distance() UDF to compute the distance between each site ZIP code and the corresponding target ZIP code.
            - The UDF haversine_distance() computes the great-circle distance between two geographic points (latitude and longitude) using the Haversine formula.
        - Filter by Radius: Filter the DataFrame to only retain rows where the distance is within the specified radius (40 miles).
    - Derived Features(for each site we create an intermediate df for all the zip codes in a 40 mile radius)
        - Zip Code
        - Distance in miles
    - We are not using any non-numerical features in this dataset
    - We were missing information for one site in the original data that was manually added by specifying a new Row object and adding it to the Spark DataFrame
- **Aggregation Steps**
    - Map Crash Data to Sites (`map_external_data_to_sites.ipynb`)
        - Data Sources:

        - `state_crash_monthly_county_counts`: Contains crash statistics by county and month
        - `us_city_states`: Contains mappings between geographical entities including zip codes, counties, and states
    - County Data Preparation:
        - County names are cleaned (removing the last 7 characters)
        - County codes are standardized to 3-digit strings with leading zeros
        - County FIPS codes are split into state FIPS (first 2 digits) and county code (last 3 digits)
    - Joining Process:
        - Zip codes from site data are exploded to create a mapping between sites and individual zip codes
        - These zip codes are matched with county zip codes from the crash data
        - This creates a many-to-many relationship where each site is associated with multiple counties it serves
        - The crash data is then joined to this site-county mapping based on county code and time period
    - Aggregation:
        - Crash statistics (counts, fatalities, injuries, vehicles) are aggregated for each site and time period
        - We avoid double counting by collecting only distinct crash statistics for each county-month combination, establishing a clean baseline dataset. Then, we create a precise mapping between each site and the counties it serves, using the distinct operator to eliminate any duplicate relationships created by overlapping zip codes. When combining sites with crash data, the process employs inner joins that only connect matching records without creating duplicates. The aggregation explicitly groups statistics by site and time period to sum crash metrics across a site's service area without duplication.
        - The final result is stored in `aggregated_site_radius_crash_df`
- Map Precipitation Data to Sites (`map_external_data_to_sites.ipynb`)
    - Data Sources:
        - Three precipitation datasets covering 2023, 2024, and 2025
        - Data is in a wide format with columns representing precipitation on specific dates
    - Data Transformation:
        - The wide-format data is unpivoted to create a long-format dataset with dates and precipitation values

        - The transformation extracts date information from column names using regex
    - Joining Process:
        - Similar to crash data, site zip codes are exploded to create site-zip mappings
        - Precipitation data is joined to sites through matching zip codes (ZCTA5CE20)
        - Each site can have multiple zip codes, and thus multiple precipitation values for a given date
    - Aggregation:
        - For each site and date, precipitation statistics are calculated:
            - Mean, median, standard deviation, quartiles, min, max, and interquartile range
        - The final result is stored in `final_precipitation_stats`
        - The code also calculates what percentage of each site's zip codes have precipitation data to identify sites with incomplete coverage. There are only two sites that are missing a single zip code of precipitation data. Since we are aggregating across the entire radius, one zip code likely won't make a difference.
- Location Data Feature Engineering (`location_data_feature_engineering.ipynb`)
    - Statistical Aggregation by Site, Brand, Year, and Month
        - The code first aggregates crash data by site ID, brand, year, and month
        - For each grouping, it calculates sums of crashes, fatalities, injuries, and vehicles
    - Window-Based Statistical Calculations
        - Creates window specifications partitioned by site ID and brand
        - For each metric (crashes, fatalities, injuries, vehicles), calculates:
            - Mean, median, minimum, maximum, standard deviation
            - Interquartile range (IQR) using the difference between 75th and 25th percentiles
    - Date Standardization and Moving Averages
        - Creates a proper date column by combining crash year and month
        - For each metric and lookback period (1-6 months):
            - Uses lag functions to access previous months' values
            - Calculates moving averages that exclude the current month's data
            - Properly handles null values with coalesce and conditional expressions
    - Precipitation Data Processing
        - Adjusts dates to align with PRISM climate dataset measurement periods

        - Creates UTC and Eastern Time timestamps for precise temporal alignment
        - Calculates moving averages for precipitation metrics over 1-3 day lookback periods
    - Complete Calendar Creation
        - Generates a complete sequence of dates from minimum to maximum dates
        - Cross-joins with all site IDs to ensure every site has every possible date
        - This handles missing dates in either dataset for proper time series analysis
    - Data Integration and Joining
        - Joins site-date combinations with precipitation data (left join on local date)
        - Joins with monthly crash data based on site ID and first day of month
        - Uses window functions to distribute monthly crash data to daily records
        - Saves the engineered features to CSV file to be merged with Motive API data
    -