Weatherlink Final Report

Alper Cetinkaya, Aditya Sanjay Gujral, Tharun Saravanan, and Matthew Winchester

George Washington University

Big Data and Analytics

Professor Roozbeh Haghnazar

December 15, 2023

Weatherlink

Combining demographic, meteorological, and traffic data for insights into traffic accidents and their causes.

The George Washington University
Alper Cetinkaya
Aditya Sanjay Gujral
Tharun Saravanan
Matthew Winchester

1. Introduction

In 2022, there were 1.7 million traffic accidents in the United States which resulted in 42,795 fatalities. The purpose of this project is to answer if adverse weather conditions and socioeconomic conditions of a given region influence traffic accidents, and to what degree these factors impact the frequency and fatality of accidents. We aim to use historical data for descriptive analysis and predictive forecasting to visualize traffic accident data with respect to weather and census information, and create a tool to predict future accidents. This could be used to help first responders prepare in areas where accidents are more likely, better allocate government resources, and raise awareness about public safety by illustrating accidents in an interactive and informative visualization.

The architectural framework comprises six parts: Data Sourcing, Data Extraction, Data Storage, and Data Analysis, Data Modeling, and Data Visualization. Our primary data sources comprise two large historical dataset of traffic accidents produced by the National Highway Traffic and Safety Authority, a U.S. Census Bureau dataset, and a set of daily historical weather data collected by National Oceanic and Atmospheric Administration.

2. Related Work

Before deciding on how to approach the problem space of analyzing the relationship between our three main sources of data, we looked for existing literature on the topic to guide our approach to the problem. There were several papers in the problem space of traffic accident analysis and prediction. We also looked for papers less specific to traffic accidents, about joining geospatial datasets. Ingesting all of the available weather, traffic, and census data will be no small task. Data from different years varies greatly in size, both in number of files and the size of each file. Processing this data into a useful form so we can visualize it easily will require batch processing these large datasets and storing the data in a way that we can reuse it many times to perform different kinds of analysis and queries. The process will need to be fault tolerant, both for the patch processing, and then storing of the data. If a single process fails while we are ingesting data, we want to avoid having to invalidate the entire dataset and starting over. We will need to homogenize the data, so that it is cleaned into a format that can be queried. Since weather, census information, and traffic are all separate geospatial data sources with many individual attributes, being able to effectively join disparate types of data was also a focus.

The paper “Data Integration: The Teenage Years” discusses how to effectively combine data from many sources, especially in large organizations and web search engines. This gave us background information on how to integrate our data. The Information Manifold project mentioned in the paper aims to offer a consistent query interface for many data sources and streamline access and integration. Additionally, it highlights the significance of exact source descriptions and presents the Local-as-View (LAV) method to facilitate this process. The paper examines efforts to enhance adaptive query processing in dynamic situations, lower manual labor, and semi-automate schema mapping.

The paper “A Framework for Scalable Real-Time Anomaly Detection over Voluminous, Geospatial Data Streams” focuses on providing a robust framework to detect anomalies in geospatial data and tries to support analytical activities by providing resources for analytical activities like visualization. According to the paper, an anomaly is a data point outside the norm, like inconsistent sensor readings or an irregular event. The paper uses individual anomaly detector implementations to classify events as anomalous. Observations comprise n-dimensional tuples with each dimension representing a feature of interest. Features include temperature, air pressure, humidity, etc. A feature may also have linear or non-linear relationships with each other; for instance, there may be a relationship between temperature and precipitation at certain geographic locations. These relationships are then used to classify data into anomalous or normal with a degree of irregularity

The techniques described in the paper will be useful for cleaning the geolocation data associated with our datasets. A model could be used to autonomously identify anomalies within the defined geographical extents. These anomalies might represent unusual patterns or events in either the weather data or first responders' response data that are potentially linked to weather conditions.

The paper, "A Survey on Spatio-temporal Data Analytics Systems," provides a comprehensive overview of the landscape of spatial and spatio-temporal data analytics, contributing to the solution of challenges in this domain. By surveying the existing ecosystem, it aids in the identification of efficient tools and systems required for capturing, managing, and analyzing the substantial volumes of spatio-temporal data generated by location-based services and diverse sources.

Another paper, focusing on "Crash Risk Modeling and Safer Routing," takes on the challenge of enhancing traffic safety by bridging the gap between statistical modeling of crash risk. It tackles the pressing issue of rising global traffic accidents by leveraging data-driven approaches. This integration can lead to practical strategies for optimizing routes and reducing crash risks in real-time, thus addressing the alarming increase in traffic-related deaths and economic losses. The authors introduce descriptive analytical methods, including data summarization, visualization, and dimension reduction, as tools to promote safer routing. It also offers code to assist practitioners and researchers in data collection and exploration.

One potential approach to the issue of non-linearity in the data is utilizing the cusp catastrophe approach. The cusp catastrophe model applies a high-order probability density function to analyze the data, giving it the capability to analyze both linear and non-linear relationships (Theofilatos, 473). In the analysis by Theofilatos, traffic flow was seen to have a “strong-non linear effect” on the number of crashes, while rainfall intensity had a “linear relationship” on the number of crashes (Theofilatos, 476-477) through the use of the model. Despite the fact that cusp models do not provide as robust of a conclusion as more traditional models, we can utilize cusp models as a strong tool to indicate the possibility of a catastrophe.

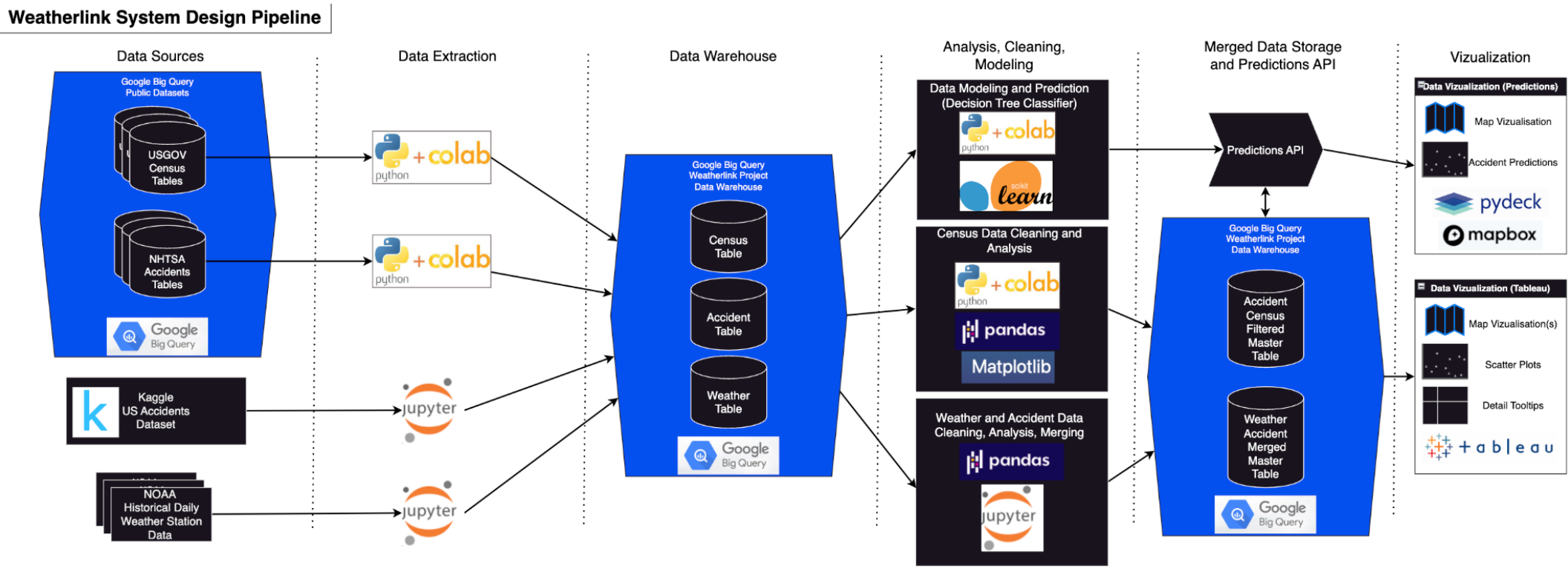
This could be supplemented by a more robust algorithm in analyzing the data in the form of a clustering algorithm. In an analysis by Gutierrez-Osorio regarding road accident forecasting algorithms, classification algorithms and decision trees were both found to have a high amount of interpretability, at the cost of precision and accuracy in trend analysis. Deep learning algorithms were susceptible to overfitting the presented data. Analyses utilizing Clustering Algorithms however, presented high performance and accuracy, making them a strong choice as a model for our data.

To familiarize ourselves with big data best practices, we also studied two fundamental papers in the field of big data and distributed computing. The first paper produced about the Map Reduce algorithm is a very good explanation of the core ideas behind distributed computation and how a cluster of computers can work together on the same problem. Hadoop Distributed File System is an example of a distributed file system. While we experimented with HDFS and A Scala spark job as part of our first assignments, we ultimately did not need these technologies for our project, though understanding them was helpful in our pipeline creation exercises.

In all, these papers helped us see how large data sets of geospatial data can be merged and analyzed, helping us prepare how we will be joining the data together for analysis, detecting and removing outliers, and modeling the data for accident forecasting.

3. Project Design

3.1 System Architecture Diagram



Full Resolution: https://drive.google.com/file/d/1_yLjOuVhJbpiYE75cIjVKvNFXWNI5U61/view?usp=sh

3.2 System Architecture Description

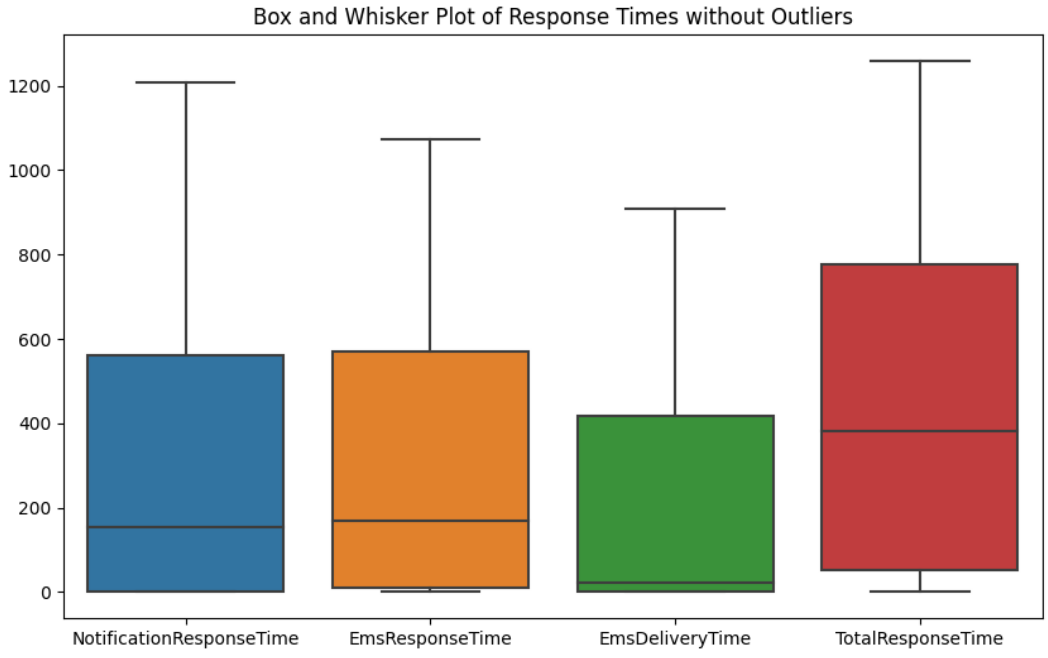
The Weatherlink Pipeline has six phases. Data is moved between phases and is extracted, cleaned, analyzed, modeled, and visualized. Each phase will be described with design decisions explained.

Data sources for the Weatherlink project come from three different locations. Two public datasets from Google Big Query were used for census data and accident data. Each of these data sources were split up between dozens of individual tables, one for each year. This data was selected due to its number of dimensions and size. The census data was of sufficient granularity (down to the zip code level if we wanted it) to join on the county level accident data. Weather data came from NOAA's historical archive of daily weather station readings. Several decades of weather data are accessible from the government which we used for the project. Finally, a Kaggle dataset of millions of traffic accidents was used to round out the traffic and accident data.

Data extraction for the Google Big Query public datasets was performed on a Google Colab notebook. Because this was a single extraction event and not a recurring event, we chose this approach because it allowed us to iterate on our extraction approach in a collaborative manner, where we could repeat the extraction process by running portions of the notebook with slight modifications as needed. Pandas dataframes load the tables from Google Big Query. Each year's table is loaded into a master dataframe. This master data frame has a year column appended to it, and then is stored in our data warehouse. The NOAA dataset was difficult to ingest. Each file from each year had slight variations, and NOAA's servers had to be queried for each file which had data from individual weather stations. A python script in a jupyter notebook loaded the data and cleaned it. This process specific for the NOAA data is detailed in the Big Data Challenges section. The Kaggle dataset was a large csv file with millions of rows. Python was used to load the data into a pandas dataframe and pushed to our data warehouse.

We used Google Big Query as our data warehouse due to the existing public datasets available. This meant with a single python client, we could pull public data and push to our team's google cloud project. Additionally, Google's built in tools were appealing to use with data stored in GBQ for modeling and visualization. Ultimately, we did not process data on Google Cloud due to cost constraints, and we did not use Google Looker for visualization due to feature preferences. However, storing the data in GBQ was a cost effective way to centralize the data for cleaning and analysis.

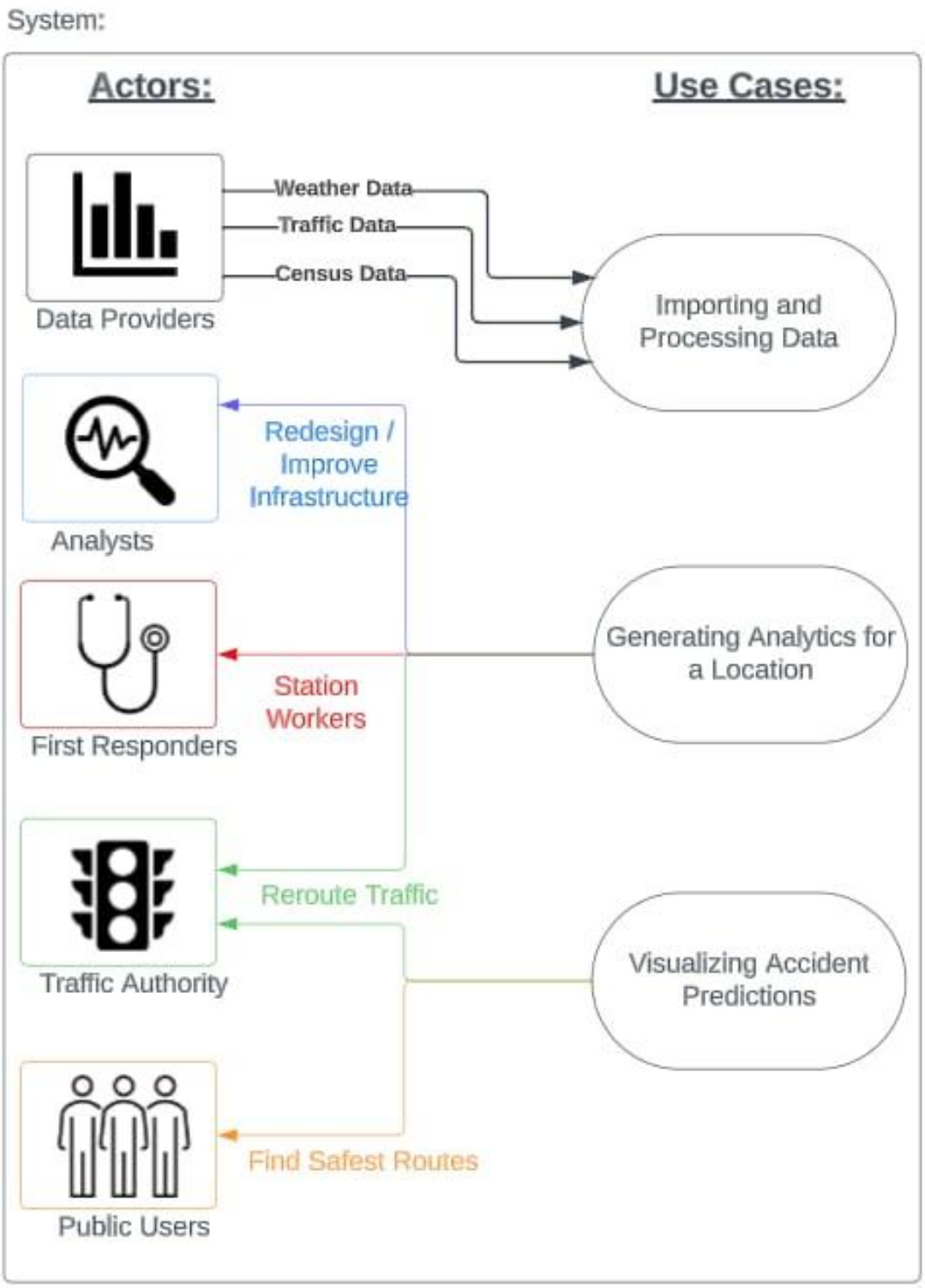
Several separate cleaning and analysis efforts were made on the data. To join the different types of data, we needed to find common dimensions. We used a combination of latitude and longitude, GEOID (which is a government defined FIPS code for a county, kind of like a zip code), and time. One of the big data challenges discussed later describes how we accomplished joining the data from a weather station to the latitude and longitude of a crash. To get an initial feel for the census data, we used Google Colab to load census data into a pandas dataframe, and generated scatter plots of features to feel the shape of the data. We had to remove nulls from the dataset, and used z-scores to create quartiles and define outliers that skewed the data. Here is an example of a box and whiskers plot used to analyze the data (showing response times):



Data modeling and prediction analyzed the many features of the data and. We looked at each column and looked at data types, common statistical markers like min, max, average, number of unique values, and number of null values. We reduced the features for analysis from several hundred possibilities down to about 20 that seemed useful. Many different iterations of a scikit-learn (a data modeling python library) Decision Tree learning model were used for classification and regression, with different combinations of these features, with the goal of predicting accidents. Comparing the accuracy of each, we were able to further eliminate features that did not correlate with the number of traffic accidents. The final test accuracy of the model is around 80%. Since this course was not a machine learning course, we did not seek to improve this metric. An API was created to query the model to generate predictions of crashes in the future.

We experimented with several different ways to visualize the data. Tableau, DeckGL and pydeck were used in our first attempts to map out latitude and longitude points of accidents and traffic reports. DeckGL and pydeck were responsive and could handle visualizing many points after a long load time. Tableau struggled to handle large data in the web hosted version, so we initially ruled it out until we were able to get a student license for the desktop version. Matplotlib was used to generate simple scatter plots for initial analysis. Ultimately, we decided to use Tableau, as the quality of visualization and tools available seemed to be the best. Tableau allowed us to create calculated fields and advanced visualizations that can react and filter in real time. Challenges with tableau are detailed in the next section. Tableau is used to view historical data. To view predictions, we used the API we built for our model and created a deckGL visualization to see accident predictions given data in the future.

3.3 Use Case Diagram



3.4 Data Sources

Dataset	Usage
Weather Based Datasets:	
NOAA's: Daily Weather Dataset https://www.ncei.noaa.gov/access/crn/qcdatasets.html	Freely accessible weather data across North America
Traffic and Accident Datasets:	
NHTSA Traffic Fatalities: https://console.cloud.google.com/marketplace/product/nhtsa-data/nhtsa-traffic-fatalities	Traffic Incident Data, including types of cars and roads, the maneuvers that preceded the accident, and the involvement of pedestrians and cyclists.
US Accidents (2016-2023): https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents	Comprehensive dataset including geospatial location alongside key weather timestamps of traffic accidents
Miscellaneous Linkage Data:	
Census Bureau Data: https://console.cloud.google.com/marketplace/product/united-states-census-bureau/acs?q=search&referrer=search&project=mattbigquerytest1&pli=1	Miscellaneous data that can be used to link traffic data to weather based datasets
Census County Boundary Data https://www2.census.gov/geo/tiger/TIGER2023/COUNTY/	Shapefile with county bounding data

3.5 Weather Data

The weather data was not available in a bulk downloadable format, but rather was provided by NOAA in the form of a table on Google Big Query and also was listed on a government webpage. The format on Google Big Query was in the form of multiple TAR files, which proved to be a challenge to work with. The Google Big Query weather data repository

contained a comprehensive set of data, but the datasets were inconsistent in how they were labeled, and the dataset was too large to feasibly use without some sort of high-performance computing solution. Therefore, the web repository was utilized instead. The web repository had more consistent data, including a precipitation score, average temperature, and geodetic coordinates recorded by multiple different weather stations spanning the U.S. However, this dataset did not allow for quick aggregation of the data into one weather "master" table. Data in the web repository was structured into year folders, with daily city data weather included in txt files within, mandating further processing into a table.

Index of /pub/data/uscrn/products/daily01/2007

Name	Last modified	Size	Description
Parent Directory		-	
CRND0103-2007-AK_Fairbanks_11_NE.txt	2022-11-18 13:21	77K	
CRND0103-2007-AK_Sitka_1_NE.txt	2022-11-18 13:21	77K	
CRND0103-2007-AK_St._Paul_4_NE.txt	2022-11-18 13:21	77K	
CRND0103-2007-AK_Utqiagvik_formerly_Barrow_4_ENE.txt	2022-11-18 13:21	77K	
CRND0103-2007-AL_Clanton_2_NE.txt	2022-11-18 13:21	62K	
CRND0103-2007-AL_Courtland_2_WSW.txt	2022-11-18 13:21	77K	
CRND0103-2007-AL_Cullman_3_ENE.txt	2022-11-18 13:21	77K	
CRND0103-2007-AL_Fairhope_3_NE.txt	2022-11-18 13:21	77K	
CRND0103-2007-AL_Gadsden_19_N.txt	2022-11-18 13:21	77K	

To concatenate this data, a jupyter notebook was first used to scrape the web repository. To begin the data extraction, the Jupyter notebook scanned each folder within the parent directory, and created local folders for each year from 2006 to 2023. After that, the notebook parsed each city txt file into a data frame, and then converted each data frame into a csv file within the respective year folder. The headers were not present within each txt file, so they

were manually appended for each generated csv. Based on the latitude and longitude of the weather station, the county FIPS code and county name were also appended to the dataset each csv file by querying a census bureau web api and extracting that information.

```
def fetch_county_info(latitude, longitude):
    params = urllib.parse.urlencode({'latitude': latitude, 'longitude': longitude, 'format': 'json'})
    url = f'https://geo.fcc.gov/api/census/block/find?{params}'
    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    return data['County']['FIPS'], data['County']['name']
```

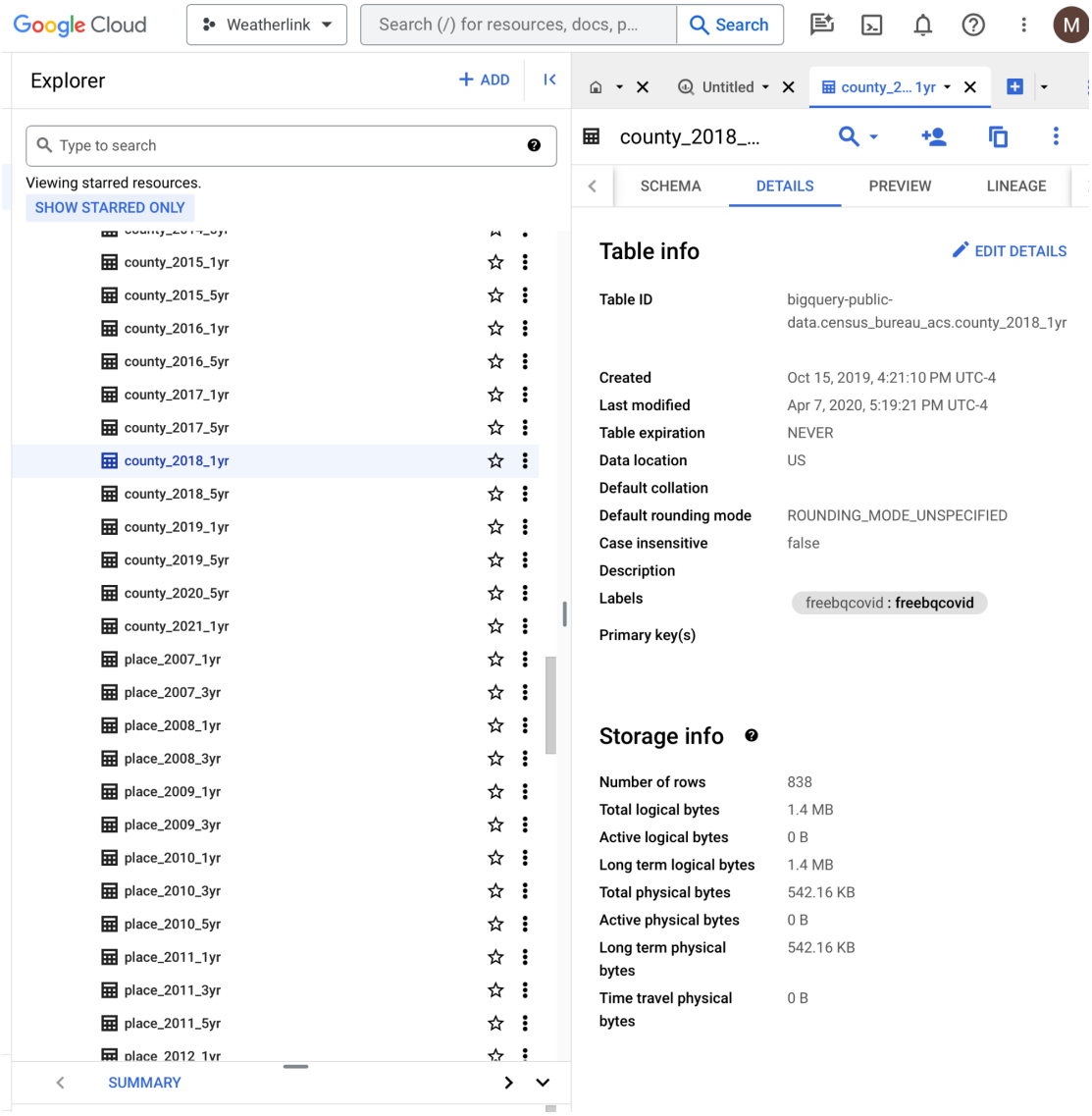
In the case that the weather station did not belong to an established county, a “None” value was appended for the county fields.

The weather parameter we were most interested in, precipitation, had to be cleaned during this process as well. When invalid precipitation recordings were found, it would be given a precipitation score of -9999.0 rather than a positive float number like with the other recordings, which would skew our analysis heavily. These were simplified to 0.0, or no recorded precipitation.

After extracting each individual city to the respective folder, the data was concatenated into one large weather master table. The year was appended to each row depending on the respective folder, and the resulting csv file was the weather master table.

3.6 Census Data

The Census data consisted of several tables spanning socioeconomic data gathered from surveys of United States citizens. This government provided data exists in several places on the internet, including in a public dataset in Google Big Query. Here is a screenshot of this data set showing many of the tables. Note the table selected is of data from 2018, and has 838 rows. This table has 252 columns, and all years have similar dimensions.



To join the various years' data together, a google colab notebook was created to probe the data and extract the columns common between each table into a 'master' dataframe, which was saved back to GBQ in our data warehouse. Here is a snippet of the extraction process (the whole notebook is in our github repository).

Analyzing some of the census data, I want to see what the number of columns in each year are to see if they differ.

```
[ ]  accident_client = bigquery.Client()

     for i in range(2007,2024):
       table_name = f"bigquery-public-data.census_bureau_acs.county_{i}_1yr"


       sql_query = (f"SELECT * FROM {table_name} LIMIT 1")

       try:
         df = accident_client.query(sql_query).to_dataframe()
         print (f"Table name {table_name} has {(df.shape[1])} cols")

       #print(df)
       except:
         print (f"Table name {table_name} was not found :( ")
```

```
Table name bigquery-public-data.census_bureau_acs.county_2007_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2008_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2009_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2010_1yr has 247 cols
Table name bigquery-public-data.census_bureau_acs.county_2011_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2012_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2013_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2014_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2015_1yr has 246 cols
Table name bigquery-public-data.census_bureau_acs.county_2016_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2017_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2018_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2019_1yr has 245 cols
Table name bigquery-public-data.census_bureau_acs.county_2020_1yr was not found :(
```

As you can see, not every year has the same number of columns. To see if we could ignore those columns that all did not have in common, we extracted the set of columns that were unique in each year and compared them.

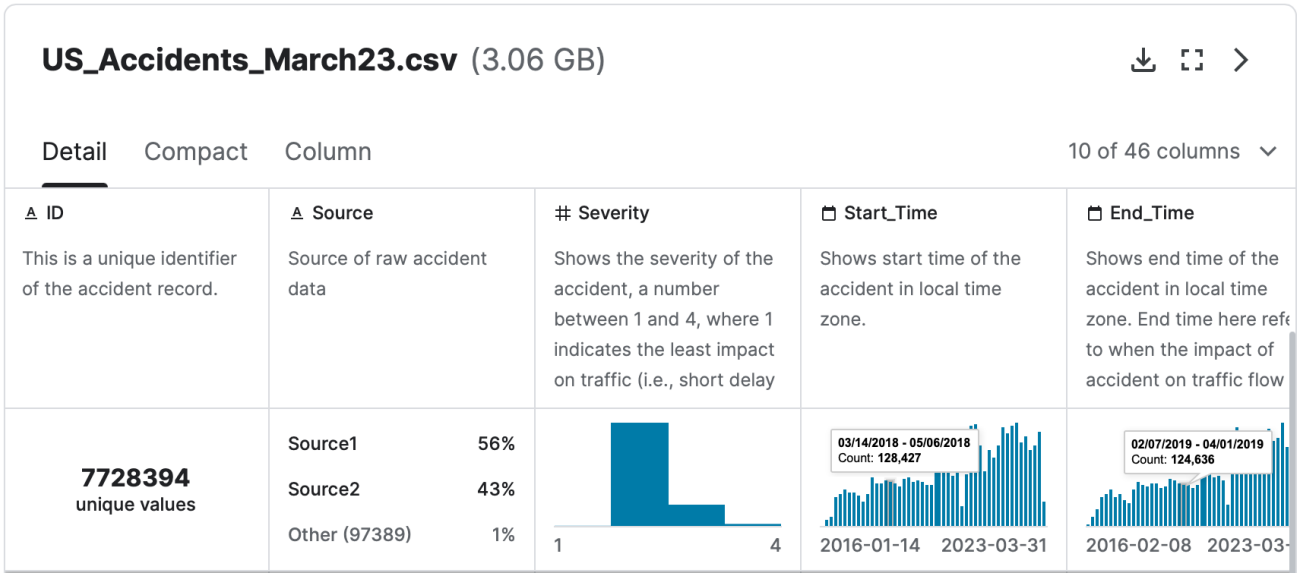
```
# Print unique columns for each table
for i in range(year_start, year_stop):
    table_name = f"bigquery-public-data.census_bureau_acs.county_{i}_1yr"
    try:
        print(f"Unique columns in {table_name}: {unique_columns_dict[table_name]}")
    except:
        print(f"Table {table_name} not found")

# Concatenate all DataFrames into a master DataFrame
master_df = pd.concat(dfs, ignore_index=True)

# Print the master DataFrame
print("\nMaster DataFrame:")
print(master_df)
```

```
Table name bigquery-public-data.census_bureau_acs.county_2018_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2019_1yr has 245 cols
Table bigquery-public-data.census_bureau_acs.county_2020_1yr not found
Table name bigquery-public-data.census_bureau_acs.county_2021_1yr has 245 cols
Table bigquery-public-data.census_bureau_acs.county_2022_1yr not found
Table bigquery-public-data.census_bureau_acs.county_2023_1yr not found
Table bigquery-public-data.census_bureau_acs.county_2024_1yr not found
Unique columns in bigquery-public-data.census_bureau_acs.county_2007_1yr: set()
Unique columns in bigquery-public-data.census_bureau_acs.county_2008_1yr: set()
Unique columns in bigquery-public-data.census_bureau_acs.county_2009_1yr: set()
Unique columns in bigquery-public-data.census_bureau_acs.county_2010_1yr: {'geoidsl', 'geoidsc'}
Unique columns in bigquery-public-data.census_bureau_acs.county_2011_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'unemployed_pop',
Unique columns in bigquery-public-data.census_bureau_acs.county_2012_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'unemployed_pop',
Unique columns in bigquery-public-data.census_bureau_acs.county_2013_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'unemployed_pop',
Unique columns in bigquery-public-data.census_bureau_acs.county_2014_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'unemployed_pop',
Unique columns in bigquery-public-data.census_bureau_acs.county_2015_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'unemployed_pop',
Unique columns in bigquery-public-data.census_bureau_acs.county_2016_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'pop_widowed', 'po
Unique columns in bigquery-public-data.census_bureau_acs.county_2017_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'pop_widowed', 'po
Unique columns in bigquery-public-data.census_bureau_acs.county_2018_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'pop_widowed', 'po
Unique columns in bigquery-public-data.census_bureau_acs.county_2019_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'male_60_to_61', '
Table bigquery-public-data.census_bureau_acs.county_2020_1yr not found
Unique columns in bigquery-public-data.census_bureau_acs.county_2021_1yr: {'civilian_labor_force', 'employed_pop', 'pop_in_labor_force', 'male_60_to_61', '
```

Since the vast majority of over 240 columns were the same between each year, we dropped the columns that were not in common and created a master table with the remaining columns.

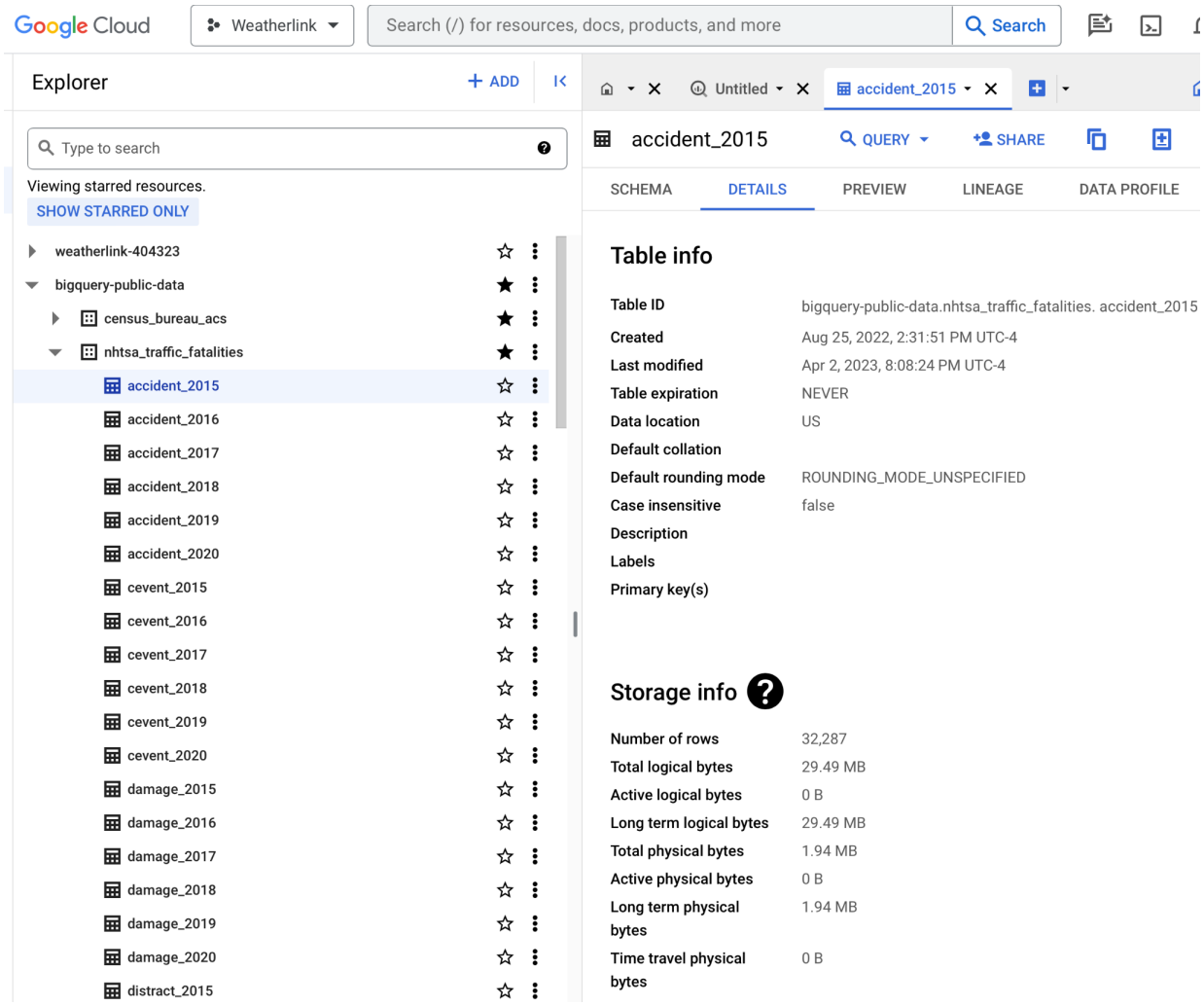
3.6 Accident Data

The accident data was split between two datasets. The first of the two datasets we used was a large csv file from a Kaggle dataset with nationwide accident reports.



With over 7 million rows of 46 columns detailing traffic accidents from 2016 to 2023, this is by far our largest data set. This data was loaded into a jupyter notebook with pandas and saved to our data warehouse in Google Big Query for processing.

The second dataset was a set of tables on Google Big Query public data. Like the Census data, there were several tables of government provided data on Google Big Query that we joined together into a master accident table. Here is a view of the available tables in the nhtsa (National Highway Traffic Safety Authority) public data on Google Big Query:



The currently selected table from 2015 has a record of 32,287 fatal accidents. Each table has many columns, between 82 and 92 in each table. Like the Census data, we joined together the tables into a single dataframe in a Google Colab notebook, and saved the result in an accident master table in our data warehouse for later cleaning and analysis. To be able to join the data with the census data, we had to format a GEOID (a 5 digit number unique to each county in the United States) to match that of the county FIPS code from the census data. We also appended the year to the data to match on census data for a specific year with each accident.

Here is a snippet of the extraction process appending these fields to the master data frame:

```
for i in range(year_start, year_stop):
    table_name = f"bigquery-public-data.nhtsa_traffic_fatalities. accident_{i}"


    try:
        # Fetch the schema (column information) for each table
        if table_name not in table_schemas:
            table = client.get_table(table_name)
            table_schemas[table_name] = set([field.name for field in table.schema])

        # If this is the first DataFrame, initialize the set with its columns
        if common_columns_set is None:
            common_columns_set = set(table_schemas[table_name])
        else:
            # Update the set to include only columns present in both DataFrames
            common_columns_set.intersection_update(table_schemas[table_name])

        # Update the set to include only columns not present in other DataFrames
        unique_columns_dict[table_name] = table_schemas[table_name].difference(common_columns_set)

        print(f"Table name {table_name} has {len(table_schemas[table_name])} cols")

        # Fetch the data and add it to a DataFrame
        query = f"SELECT * FROM `{table_name}`"
        df = client.query(query).to_dataframe()

        # Add the 'year' column to the DataFrame
        df['year'] = i

        # Calculate the FIPS code
        df['county'] = df['county'].astype(str)
        df['county'] = df['county'].str.zfill(3)
        df['state_number'] = df['state_number'].astype(str)
        df['state_number'] = df['state_number'].str.zfill(2)

        df['geoid'] = df['state_number'] + df['county']

        # Keep only the common columns
        df = df[list(common_columns_set) + ['year'] + ['geoid']]

        # Append the DataFrame to the list
        dfs.append(df)

    except Exception as e:
        print(e)
        print(f"Table {table_name} not found")
```

3.8 Data exploration and analysis

After creating our master data tables in Google Big Query, we joined census and accident data to understand the data's shape and clean the data. In Google Colab, we used google's big query

client to create a SQL statement and result joining the census and accident tables on GEOID and year of the accident.

```
[ ]
    dataset = 'weatherlink-master'

    # Define your SQL query
    query = """
    SELECT
      a.number_of_fatalities, a.geoid, a.year, a.timestamp_of_crash, a.hour_of_arrival_at_scene, a.minu
      c.income_per_capita, c.black_pop, c.total_pop, c.hispanic_pop, c.asian_pop, c.amerindian_pop
    FROM
      `weatherlink-404323.weatherlink_master.census_master` c
    JOIN
      `weatherlink-404323.weatherlink_master.accident_master` a
    ON
      c.YEAR = a.YEAR AND c.geo_id = a.GEOID
    """

    # Execute the query and store the results in a Pandas DataFrame
    df = client.query(query).to_dataframe()
```

```
[ ]
    print(df)
```

```
        number_of_fatalities  geoid  year         timestamp_of_crash  \
0                          1   4015  2015  2015-06-12 00:43:00+00:00
1                          1  21145  2015  2015-08-14 00:18:00+00:00
2                          1  23011  2015  2015-11-26 00:53:00+00:00
3                          1  37021  2015  2015-05-11 00:43:00+00:00
4                          1  44007  2015  2015-07-26 00:36:00+00:00
...                      ...    ...   ...                       ...
122001                     1  12086  2019  2019-05-15 09:13:00+00:00
122002                     1  49035  2019  2019-09-10 00:00:00+00:00
122003                     1  12057  2019  2019-03-27 20:30:00+00:00
122004                     1   4001  2019  2019-03-09 15:23:00+00:00
122005                     1   4013  2019  2019-10-13 15:00:00+00:00
```

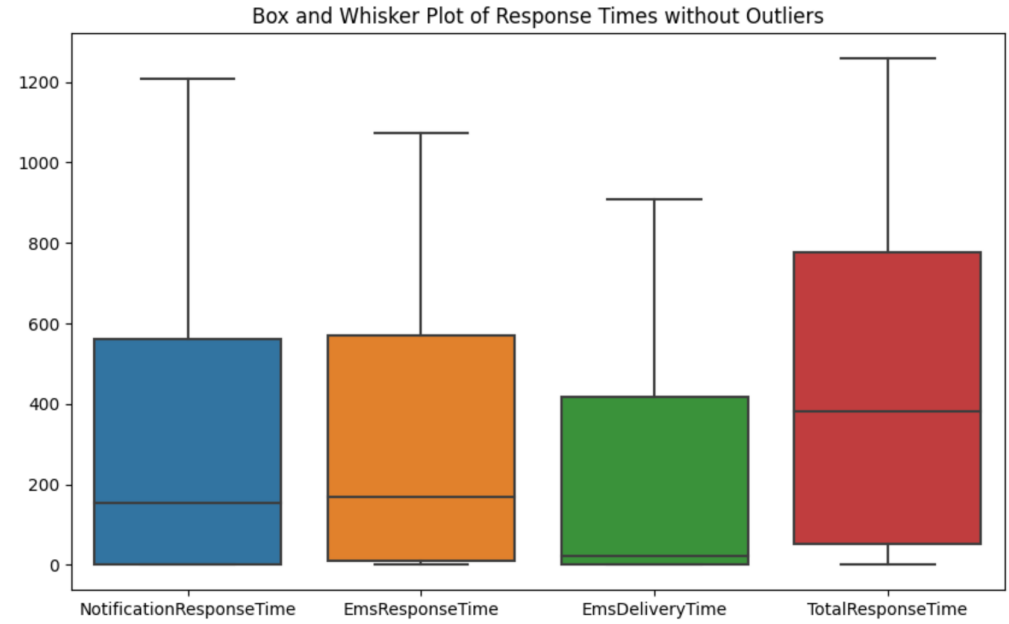
Here is the process we took for calculating response times for accidents:

1) Check the data for null values
2) Using the hour and minute of arrival vs house and minute of the crash, calculate the time delta (being sure to account for the case where an accident passed midnight and rolled over to the next day)
3) Calculate average, z scores, and quartiles for the data to remove outliers.
4) Calculate an average response time for each county
5) Append this average to the dataset for visualization against census factors

The entire process can be seen in the ‘Census Traffic Analysis’ jupyter notebook in our github.

Here is a snippet of the outlier detection and cleaning:

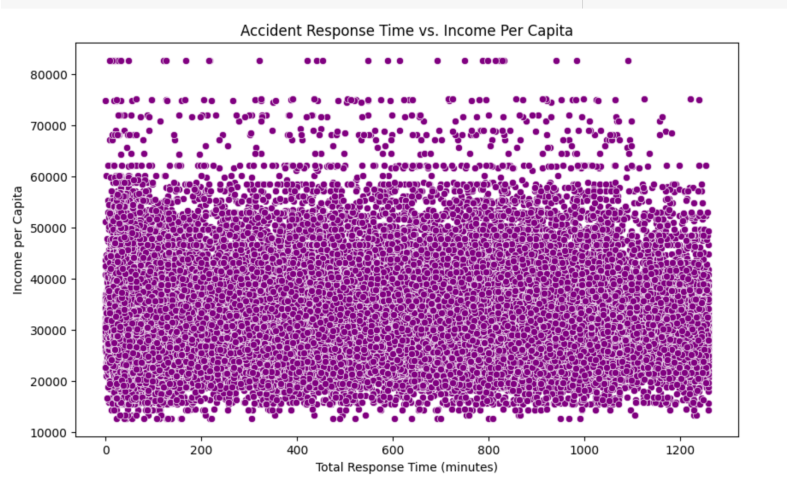
```
# Calculate z-scores for all rows
z_scores_all = stats.zscore(df[['NotificationResponseTime', 'EmsResponseTime', 'EmsDeliveryTime', 'TotalResponseTime']]

# Set a z-score threshold for outliers (e.g., 2 standard deviations)
z_threshold = 2

# Create mask for outliers
outliers_mask = (z_scores_all >= z_threshold) | (z_scores_all <= -z_threshold)

# Apply any() along the columns to create a 1D mask
any_outliers_mask = outliers_mask.any(axis=1)

# Filter the DataFrame based on the outliers mask
df_filtered = df.loc[~any_outliers_mask]

# Explicitly select the columns of interest after filtering
columns_of_interest = ['NotificationResponseTime', 'EmsResponseTime', 'EmsDeliveryTime', 'TotalResponseTime']

# Create a box and whisker plot with the filtered data
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_filtered[columns_of_interest])
plt.title('Box and Whisker Plot of Response Times without Outliers')
plt.show()

# Calculate the number of rows filtered out
num_filtered = df.shape[0] - df.loc[~any_outliers_mask].shape[0]

# Summary stats about the cleaning process
print(f"Using a z score of {z_threshold} you filtered out {num_filtered} values ")

# Recalculate averages for the cleaned data
average_notification_response_time_cleaned = df['NotificationResponseTime'].mean()
average_ems_response_time_cleaned = df['EmsResponseTime'].mean()
average_ems_delivery_time_cleaned = df['EmsDeliveryTime'].mean()
average_total_response_time_cleaned = df['TotalResponseTime'].mean()

print("\nAverages (UnCleaned Data):")
print(f"Notification Response Time: {average_notification_response_time_cleaned:.2f} minutes")
print(f"EMS Response Time: {average_ems_response_time_cleaned:.2f} minutes")
print(f"EMS Delivery Time: {average_ems_delivery_time_cleaned:.2f} minutes")
print(f"Total Response Time: {average_total_response_time_cleaned:.2f} minutes")

average_notification_response_time_cleaned = df_filtered['NotificationResponseTime'].mean()
average_ems_response_time_cleaned = df_filtered['EmsResponseTime'].mean()
average_ems_delivery_time_cleaned = df_filtered['EmsDeliveryTime'].mean()
average_total_response_time_cleaned = df_filtered['TotalResponseTime'].mean()

print("\nAverages (Cleaned Data):")
print(f"Notification Response Time: {average_notification_response_time_cleaned:.2f} minutes")
print(f"EMS Response Time: {average_ems_response_time_cleaned:.2f} minutes")
print(f"EMS Delivery Time: {average_ems_delivery_time_cleaned:.2f} minutes")
print(f"Total Response Time: {average_total_response_time_cleaned:.2f} minutes")
```

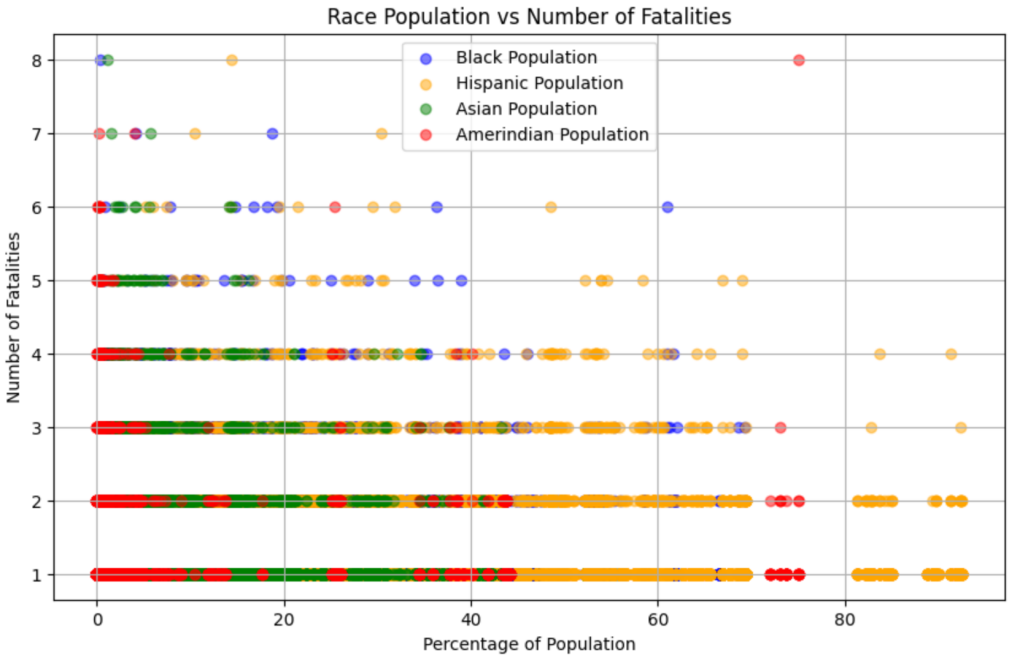



```
Using a z score of 1.5 you filtered out 32576 values

Averages (UnCleaned Data):
Notification Response Time: 465.15 minutes
EMS Response Time: 398.20 minutes
EMS Delivery Time: 298.65 minutes
Total Response Time: 569.26 minutes

Averages (Cleaned Data):
Notification Response Time: 299.08 minutes
EMS Response Time: 307.20 minutes
EMS Delivery Time: 204.60 minutes
Total Response Time: 440.25 minutes
```

Filtering out with a z score of 1.5 (meaning any result 1.5 standard deviations away from the mean were removed), the average response time for an accident was about 440 minutes. (From the time of the crash to when people were delivered to the hospital).

Next we compared these response times against a number of census factors.





(Thankfully, it appears race and income do not impact emergency response time.)

4. Big Data Challenges

One major challenge when working with the datasets was the issue of cleaning and processing the data for use in visualizations and the model.

While we were able to find sufficiently large datasets for census data, weather data, and traffic accident data, finding common columns between the different datasets to join them together proved to be a challenge. Concatenating the data tables into one large table was key, as Tableau would not be capable of quickly joining such large datasets together spontaneously when generating figures. We utilized python scripts to generate these concatenated merged data tables.

The python scripts cleaning the data utilized the pandas library to perform these joins. The csv and text based datasets were downloaded locally and then scanned into dataframes. Although we were able to find sufficiently large datasets for each of the target groups, they did not have matching columns by which to join each of them. The datasets needed to be joined based on the location and year. While trying to join on the date was fairly comparable across the different tables, the location fields were not. The census dataset used a GEOID field to indicate locations of census statistics, the weather dataset used the latitude and longitude of the corresponding weather station, and the accidents dataset listed the latitude and longitude of the accident itself. We utilized the Haversine formula to join the closest precipitation related weather data to each accident. With this merged dataset, we now had to join the latitude and longitude of each of the accidents with the GEOID in our census data. To do this, we utilized a census county data shapefile to link each census GEOID to an associated latitude and longitude. From this latitude and longitude, we then used the same Haversine formula to find the closest match alongside the census data year to link the data.

A challenge we faced with this approach was the scale of the data. As the base table we were utilizing had close to eight million rows of data, finding the distance between each data point using the Haversine formula was computationally expensive and would have taken several hundred hours to complete. To speed up this process, we utilized intermediate tables containing what was only necessary for the distance calculations and rows to serve as a composite key, then merged into the larger dataframe using the composite key rows as identifiers. This reduced completion time down to about half a day.

Precipitation data was particularly difficult to process. This semistructured data was in a very old text based tabular format, which had slight variations between tables. Amongst these variations was also slight changes in how the no precipitation was listed, depending on the recording station. Since these reports dating back to the 60s were probably done manually, the format prioritizes human readability over computational considerations. Null values for weather are sometimes listed as -9999 and other times listing 0 as the precipitation score. We standardized this aspect of the data by converting all such null instances of -9999 precipitation score to 0.

Modeling the data proved to have its own set of challenges. For prescriptive analysis, we needed to create several iterations of a Decision Tree with different combinations of columns as described in our system design. In the process of preparing data for training a Decision Tree Regressor, the main objective is to organize the input data effectively to facilitate optimal learning. This involves tasks such as managing missing values, encoding categorical variables, and dividing the dataset into features and the target variable. While Decision Trees can handle both categorical and numerical data with minimal preprocessing, it's essential to ensure the dataset is well-structured and free from inconsistencies. Additionally, a critical step involves splitting the dataset into training and test sets to accurately evaluate the model's performance. Addressing these considerations ensures that the decision tree regressor can be trained on a clean

and properly formatted dataset, improving its ability to discern patterns and make precise predictions during the learning process. Visualization of our data was a large challenge. We needed a way to view spatial data on a map, and we wanted it to be interactive so a user could pan around a map to look at various counties in the United States. Having done the work to normalize various location data (of accidents, weather stations, and counties), we could use either latitude and longitude or a GEOID to visualize location data. Initially, we used google looker to explore mapping locations by GEOID. This integration was easiest, since Google Big Query natively supports loading data into Google Looker. However, Looker was unsatisfactory in presenting the large dataset, and the team's unfamiliarity with the tool meant we wanted to focus elsewhere. We also used deckGL, which is a javascript based 3JS visualization tool, at the recommendation of our professor. DeckGL and its python implementation pydeck have support for pandas dataframes, and we used it to visualize weather and traffic information. Our final visualization of our accident predictions uses pydeck in a jupyter notebook. For descriptive analysis of historical data, our analysis in this project relied on the use of Tableau to visualize and create an interactive map. The Tableau app was connected to our data lake on Google Big Query, and queried data from there to produce visualizations. However, we noticed that directly trying to work with the data sources using a live connection was exceptionally slow due to the size of the datasets. The web based version of Tableau ran slower, but thankfully we were able to get student licenses for the desktop version of Tableau. We were able to generate visualizations faster by first creating a single filtered table with only the columns of interest for visualization, and then creating an extract of the table of interest, and then working with the locally extracted version. This sped up things considerably, and Tableau is able to visualize the entire dataset of millions of accidents in a reactive manner that can be actively filtered by the user to show information for individual counties or a range of dates.

5. Project Outcomes and Reflection

5.1 Design Decision Reflection

We utilized Google Big Query as a data warehouse for all of our processed datasets. As a data warehouse, we found Google Big Query to offer many advantages including speed, scalability, cost-effectiveness, and being conducive to collaboration. Querying for datasets that we stored on Google Big Query was a fast operation, allowing us to download for local analysis and work effectively. Furthermore, storing our data on Google Big Query offered the advantage of being scalable for an affordable cost, even when uploading datasets with gigabytes worth of data we found the pricing to be within our budget. Also, utilizing Google Big Query helped facilitate collaboration by allowing our team members to use our cleaned data sources after uploading. Our Tableau Visualizations connected to our Google Big Query server, which streamlined querying for the data in visualizations. Utilizing Google Big Query also allowed us to download our cleaned datasets locally through the use of python scripts, which was helpful when aggregating all of the data onto one machine to create the merged datasets. Google Big Query itself offered datasets on the platform as well, which we found the formatting on to be of higher quality than datasets shown elsewhere. Particularly, the government data hosted on Google Big Query was of higher quality in scope and processing when compared to the same dataset on Amazon Web Services' marketplace.

At the same time, Google Big Query offered drawbacks that we might consider as reasons to migrate to another platform in pursuing further research. While storing data on the platform was relatively inexpensive, the pricing for data processing itself far exceeded our budget for this project. This meant that we had to do our data analysis for our model, and data merging operations locally, contributing to them taking a long period of time. The visualization tool, Looker Studio, was also complex to understand and was slow for large datasets. This reason drove us to utilize Tableau with

extracted tables instead. Additionally, while the formatting was better for the public datasets on Google Big Query, other services like Amazon Web Services' marketplace offered much more data as a whole and could be more convenient to work with. Specifically, data on Google Big Query oftentimes split the data over multiple tables, requiring us to join the data manually. Also, some of the datasets on Google Big Query contained data in an unconventional format, such as in the case with NOAA's public dataset. NOAA's dataset hosted on Google Big Query contained a lot of data, but the data was in TAR archives and was inconsistent in formatting between different weather stations, making it difficult to work with.

Why did we use python to join and process our datasets, instead of some other platform or technology? We chose python as a means to join our datasets because of the number of tools that facilitated joining operations. In particular, libraries such as pandas, geopandas, and the os related libraries made developing scripts convenient to write and conceptualize compared to alternatives requiring a complex setup for the same result. The main barrier constraining our dataset merging was the issue of merging data that was close together in geographical location. In particular, our weather dataset listed weather readings by the location of the weather station, our census dataset listed census recording regions based on a GEOID, and our accident dataset was variable in location depending on where the accident occurred. This meant that we had to not only convert all of the datasets to a standard latitude and longitude indicating location, but we also needed to join the data based on the closest data in other tables rather than exactly matching on specific coordinates. Python was exceptionally helpful in this regard, as utilizing a census shapefile allowed us to convert the GEOID in our census dataset to a latitude and longitude using the geopandas library. Finding the closest station was also much easier to develop a script for, as it was simple to develop a method to determine the closest location in the dataframe after matching on the date. The caveat to this simplicity was the associated time cost it took to process each dataset, given that python is much slower than pipelines such as Apache Spark. The final version of the python notebook that joined all

three datasets together took about 24 hours to complete, so any updates to the underlying data would currently have a one day lag time. A potential improvement would be to redesign the code to use multithreading or a map reduce function with something like pyspark, which could split up the job among multiple nodes.

Considering the scope of the project, we did not feel the need to set up a recurring data extraction mechanism, but this is one area we would like to improve on. We could have used a technology like Airbyte to set up a recurring execution of our python scripts to refresh the data used in our project. This kind of technology would help us centralize the ETL pipelines we currently have spread across various notebooks. At times, it was difficult to remember which notebook was extracting what data set, and having a centralized approach would have helped. Overall, our use of notebooks allowed for iteration and collaboration, a final step would be to take the contents and centralize them somewhere they could be processed without us directly running them one by one.

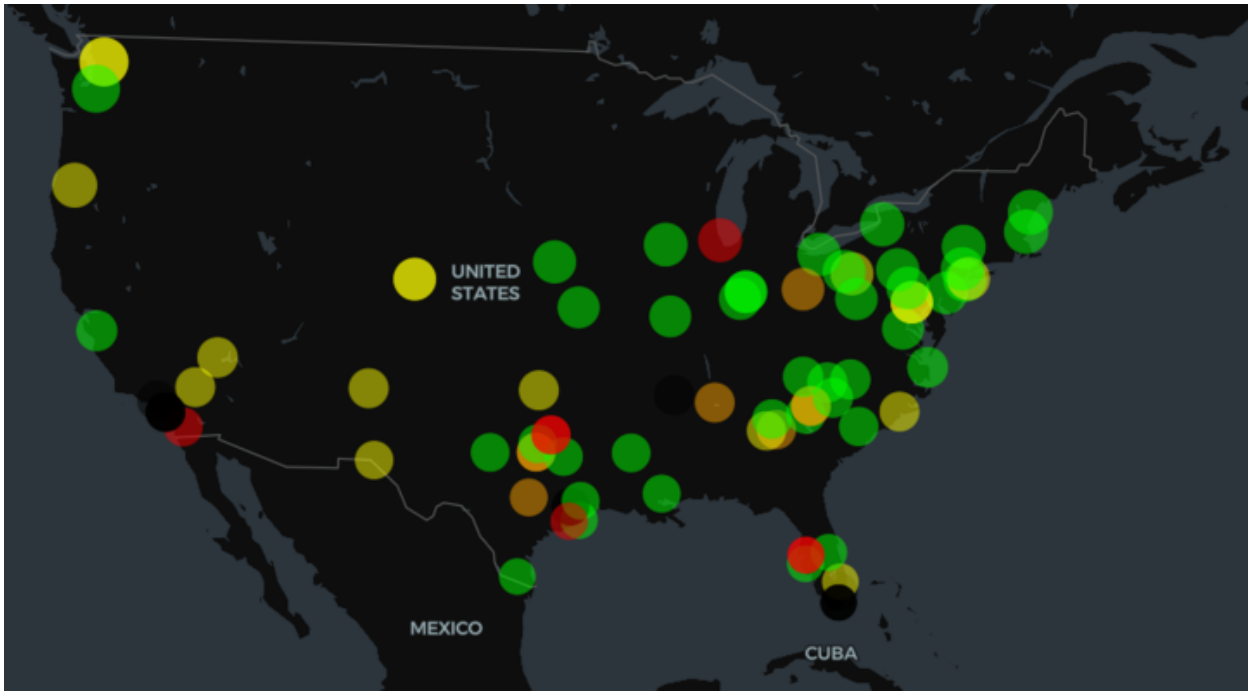
Our use of a decision tree for our learning and prediction model proved to be beneficial. Decision trees can handle both categorical and numerical features without the need for extensive pre-processing. This flexibility simplifies the data preparation process compared to some other models that require one-hot encoding or scaling.

Decision trees provide a clear and easy-to-understand representation of decision-making processes. The tree structure, with branches and nodes representing decisions and outcomes, is intuitive and interpretable.

Decision trees inherently rank features based on their importance in making decisions. Features that appear higher in the tree are more influential in predicting the target variable, providing insights into the underlying patterns in the data. Since we have a very high number of features in the dataset, this insight helped us to select useful features.

5.2 Outcomes

The API we developed to query the final model allows us to generate new predictions on the fly. This approach is flexible for development, but we were unable to get the visualization on the same level of end user functionality as the historical data in Tableau. Here is a generated visualization using the predictive model, rendered using pydeck:

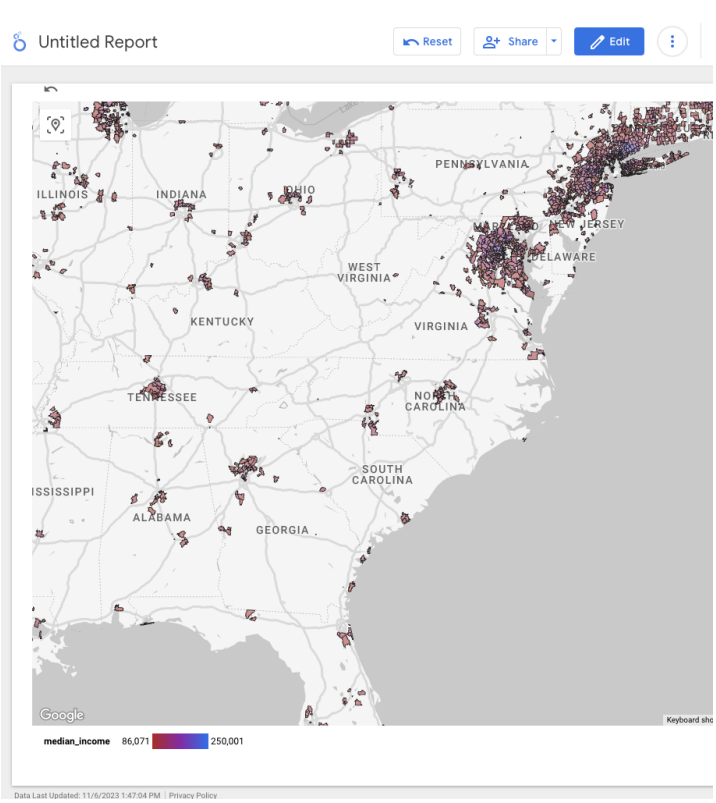


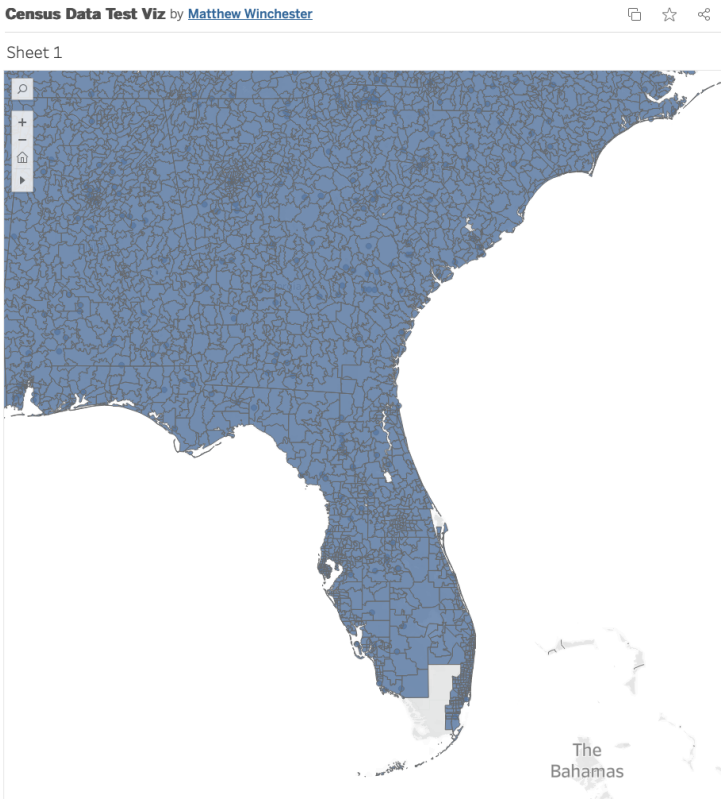
The darker colors indicate areas of higher fatality accidents, where green is a prediction of a single fatality. An improvement of this model would be to embed this in a web service so a user could interact with graphical elements to update predictions, rather than having to modify code in a notebook. The throughput of the API is fast enough that generating a new image takes about 5 seconds. Currently, the model is trained on data that was available from all data sets, the overlap of which was 2016 to 2020. Adding newer data would improve the model, since in recent years things have changed and accidents have increased as seen in our year by year breakdown in our Tableau visualization. To improve this model itself, further iterations of a decision tree could raise our prediction accuracy up from the 80% we landed on.

Querying the NOAA daily weather page was expensive in time. Unlike with other data sources, the extraction of the weather data itself from each NOAA page could not be downloaded in bulk. Furthermore, while the NOAA dataset contained information about precipitation, it lacked many other details about the weather data itself. If given funding, we might be able to expand our research and potentially draw new conclusions about accidents in relation to weather. We found that more detailed datasets regarding weather data aspects in relation to driving, such as visibility and cloudiness were only offered by paid historical data sources. Given such datasets, we could potentially pinpoint the source of accidents further.

Our final Tableau visualization has improved a lot from our first tests. Initially very slow, our current approach feels responsive and is much nicer to look at. Google Looker would have been an alternative, but seemed to be much less performant. When trying to visualize the same exact data, Google Looker could not compete in terms of density.

Here is an early iteration of testing Google Looker vs Tableau with the same census data:

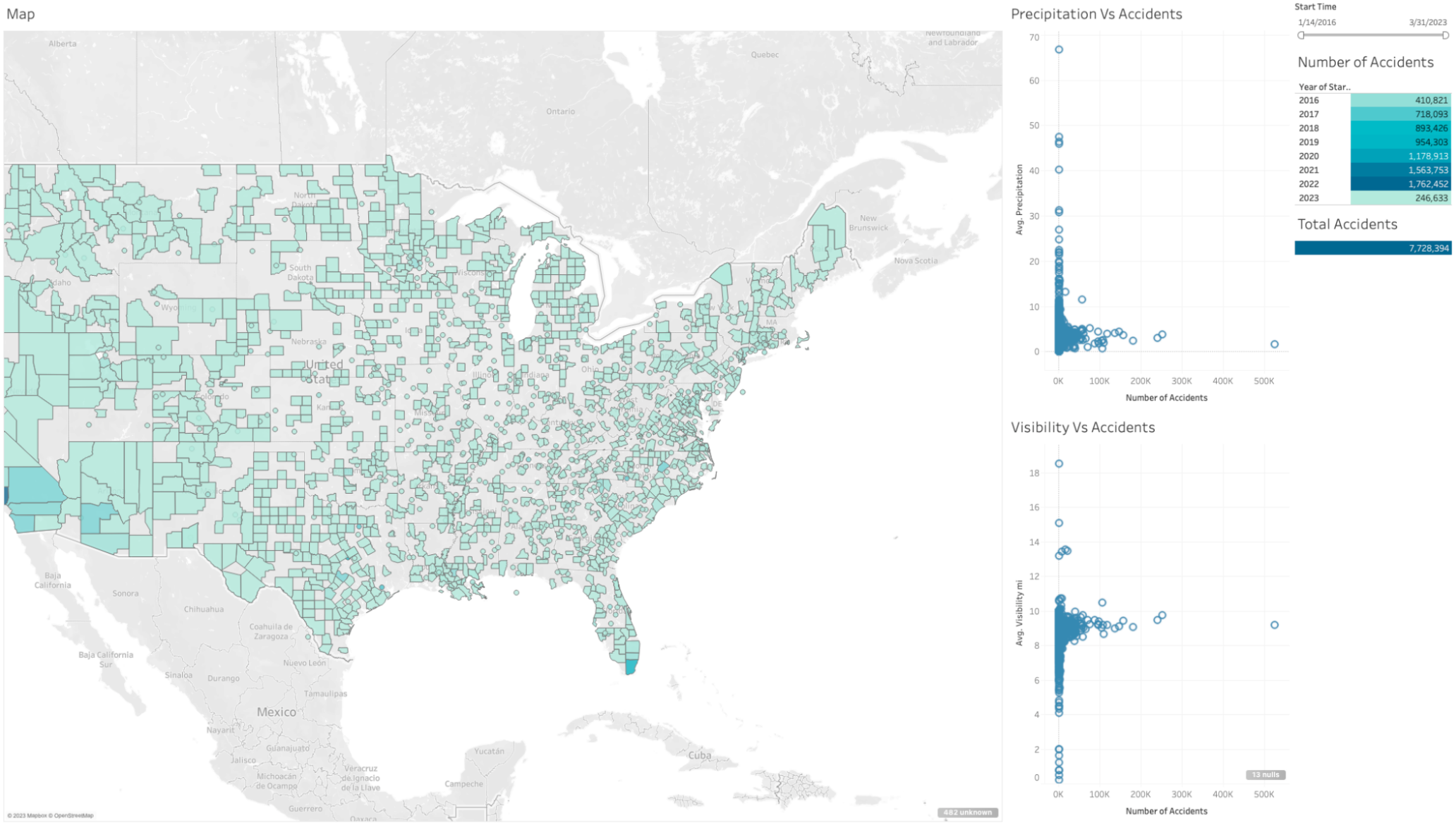




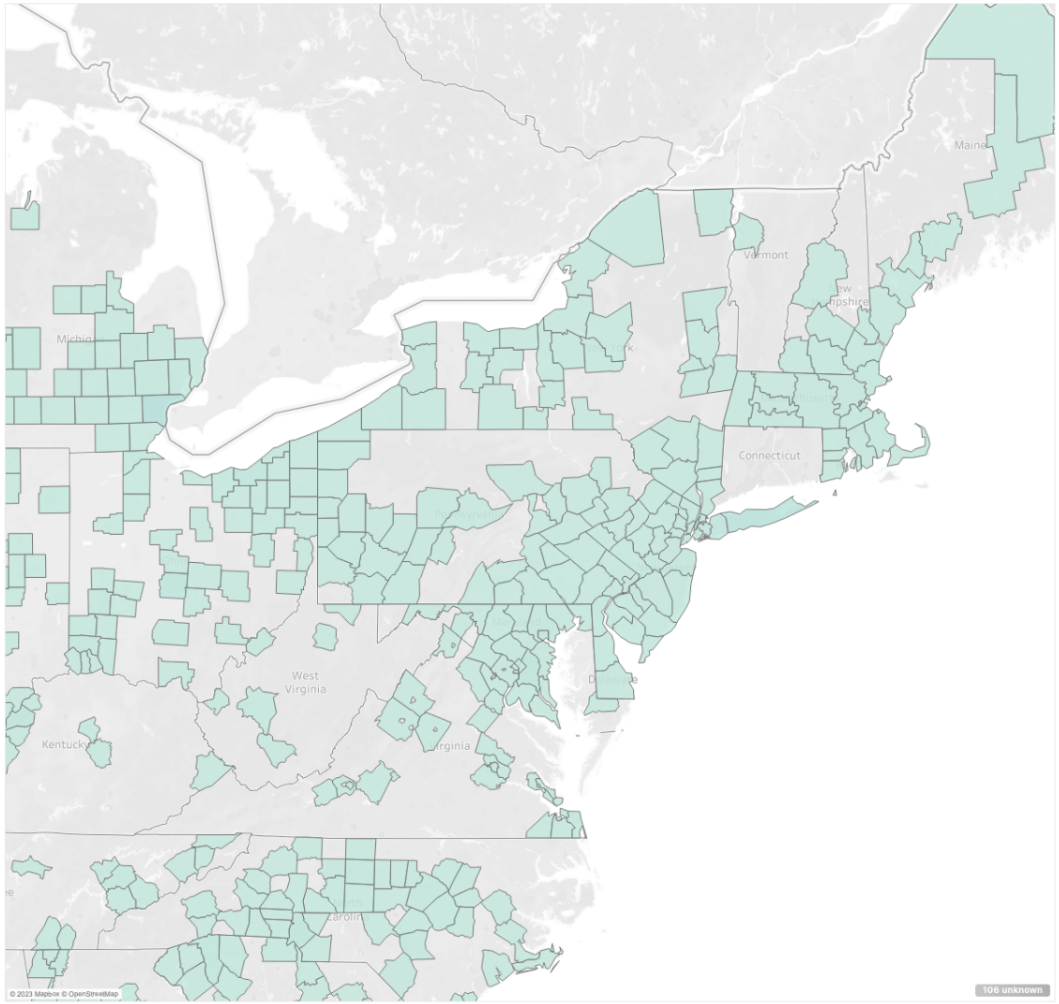
We utilized Tableau to perform descriptive analysis and present our historical data for visual impact. When working with smaller datasets, we could create a calculated field to join the related tables and plot the associated data. However, given that our base accident table contained close to eight million rows of data, our Tableau visualizations were unable to utilize tools such as a calculated field to join tables together. For this reason, we joined our tables in our data warehouse together into a larger merged table before then processing with Tableau. While this approach allowed us to generate visualizations much faster, adapting this framework to accommodate new weather information is difficult, due to the amount of preprocessing we required. Our throughput in generating updated visualizations was gated by our ability to preprocess the datasets.

What follows are several screenshots of iterations of our Tableau visualizations. As we learned the features of tableau and analyzed the data, we settled on a final set of graphs next to a large map. The user can modify the visualizations by clicking on a specific county and adjusting the date slider to see information about a certain period of time. The use of a color gradient on the county indicated which counties had a higher average accident count, which allows the high accident areas to stand out visually.

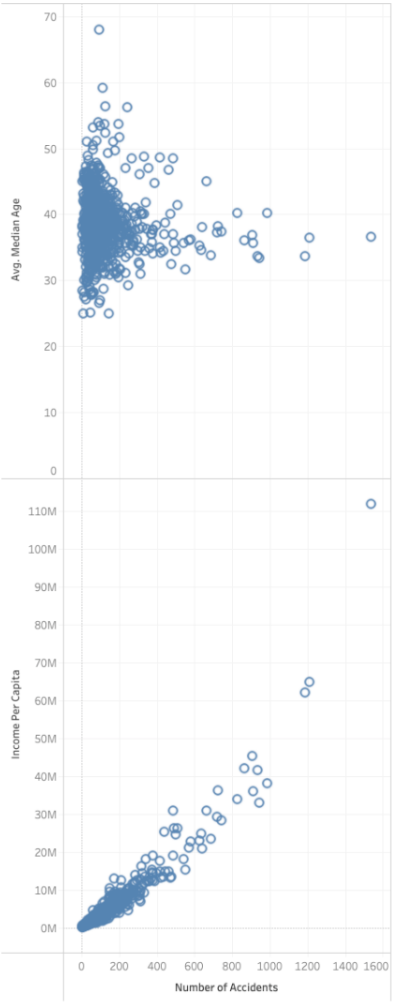
Screenshot of our Tableau visualization of historical merged datasets.



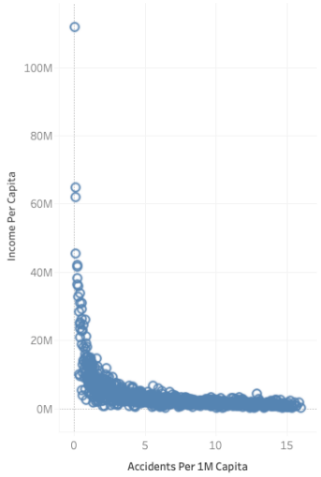
Census VS Accidents



Income Per Capita VS AVG Accidents



Accidents VS Income Per Capita



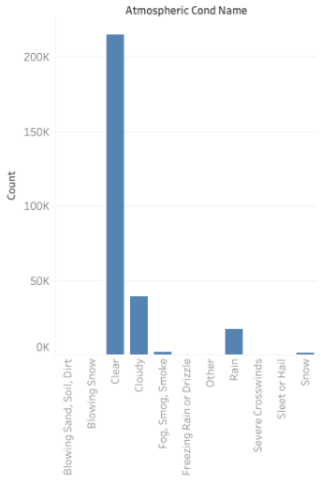
Timestamp Of Crash

(All)

Count

3 35,871

Conditions VS Accidents



Ultimately, average visibility and precipitation were meaningful indicators of accidents. Interestingly, areas with just a few inches of rain on average seem to have more accidents. We theorize this is due to the sudden onset of rain in areas where there is normally not much precipitation causing the most impact. From a census perspective, we were able to see that population percentages of different races did not impact accident response times. Income per capita however does seem to correlate with accidents, areas where there is a higher income per capita have a lower accident rate per 1M people. The average median age of each county also seems to peak with higher accidents occurring around the 37 year old mark. (Perhaps a midlife crisis purchase of a sports car?)

5.3 Further Research

To further investigate weather and its causes on traffic accidents, we would want to find data that would allow us to normalize the rate of accidents when it is sunny vs other weather conditions. As you can see from the bar graph, the vast majority of accidents occur when it is clear, but given more time we would want to normalize a rate of accidents per weather type to see if proportionally more accidents occur during other weather conditions.

Regarding accident response times, we found this data to be sparse and unreliable. Many of the data points were outliers or probably incorrect due to unrealistic times. Many response times were not reported at all, or a ‘99’ was placed in the hours column indicating a null value. Further research into response times would be merited provided a more robust data source.

To expand on the findings of this project, another avenue to expand the project is integration with APIs for the data. In particular, we considered potentially querying weather forecast data on a daily basis to determine the likelihood of an accident as well rather than purely historical datasets. Potentially integrating with a web application could also help guide users of our application to take routes less prone to accidents. As such to further this project, we considered integration with other APIs to drive predictions

5.4 Conclusion

This project presented us with the opportunity to learn big data principles by working in a problem space with many different data sources. Traffic accidents in the United States are a real problem, and their occurrences increase every year. Identifying potential causes for the increase in accidents is an area that needs more research, and our project identified some areas for further study. We were pleased to see that socioeconomic factors did not impact accident response times, but there do seem to be factors like median income and age that impact accidents. We would recommend further study into the effects of weather on traffic accidents, specifically instances where sudden changes in weather or variances in normal weather patterns occur, as data indicates a few inches of precipitation see the bulk of precipitation related accidents. Driver attentiveness in these situations would be another area of interest. The visualizations we have created are interesting tools for interacting and interpreting the data, and predictive analysis of future areas of high accident rates seems promising. Improving the accuracy of the model would be necessary for real world use.

We hope to take the skills we have learned and the spirit of the goals of this project into the future of our careers, to build better data analysis pipelines that serve the public good.

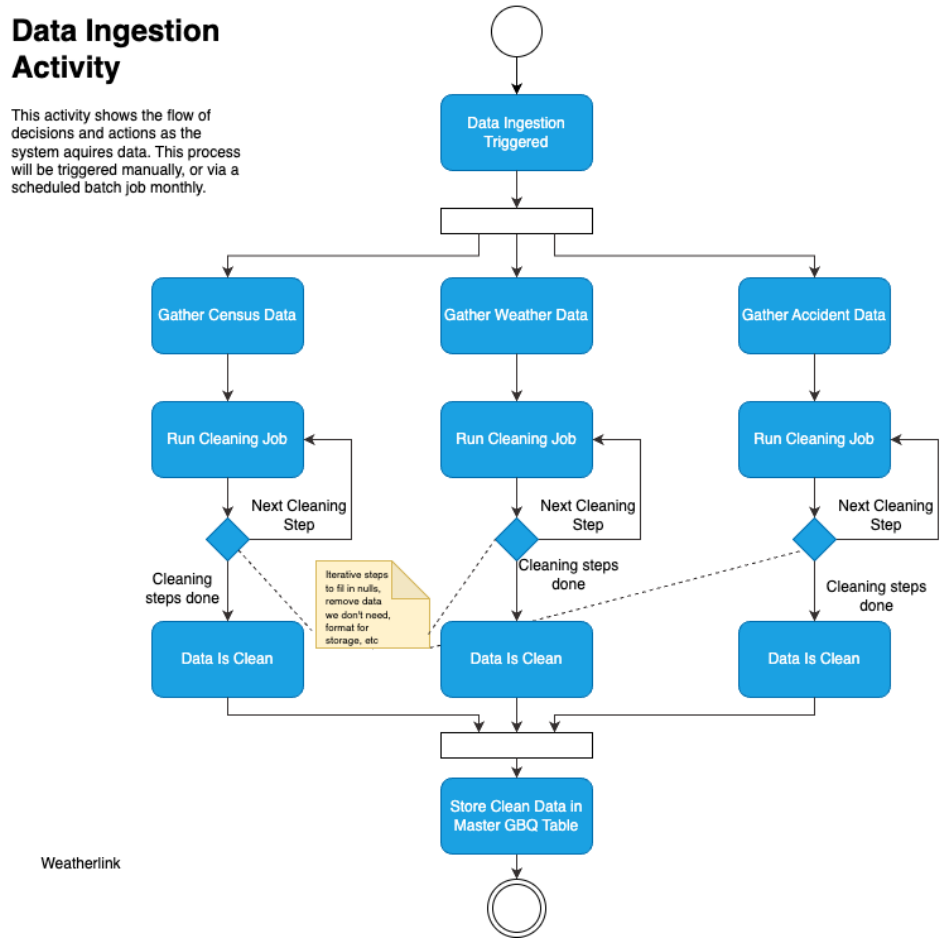
6. References

1. Gutierrez-Osorio, C., & Pedraza, C. (2020). Modern data sources and techniques for analysis and forecast of road accidents: A review. Journal of Traffic and Transportation Engineering (English Edition), 7(4), 432–446. doi:10.1016/j.jtte.2020.05.002
2. Theofilatos, A. (2019). Utilizing Real-time Traffic and Weather Data to Explore Crash Frequency on Urban Motorways: A Cusp Catastrophe Approach. Transportation Research Procedia, 41, 471–479. doi:10.1016/j.trpro.2019.09.078
3. Alam, M. M., Torgo, L., & Bifet, A. (2022). A survey on spatio-temporal data analytics systems. ACM Computing Surveys, 54(10s), 1-38.
4. Mehdizadeh, A., Cai, M., Hu, Q., Alamdar Yazdi, M. A., Mohabbati-Kalejahi, N., Vinel, A., ... & Megahed, F. M. (2020). A review of data analytic applications in road traffic safety. Part 1: Descriptive and predictive modelling. Sensors, 20(4), 110
5. Data Integration: The Teenage Years Alon Halevy Google Inc. halevy@google.com Anand RajaramanKosmix Corp .anand@kosmix.com Joann Ordille Avaya Labs joann@avaya.com
6. A Framework for Scalable Real-Time Anomaly Detection
over Voluminous, Geospatial Data Streams
Walid Budgaga, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara
Computer Science Department, Colorado State University, Fort Collins, CO, USA
7. Jeffrey Dean, Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. Google, Inc.
https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
8. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler. 2010. The Hadoop Distributed Filesystem. Apache Foundation. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pd

7. Appendix

7.1 Activity Diagrams

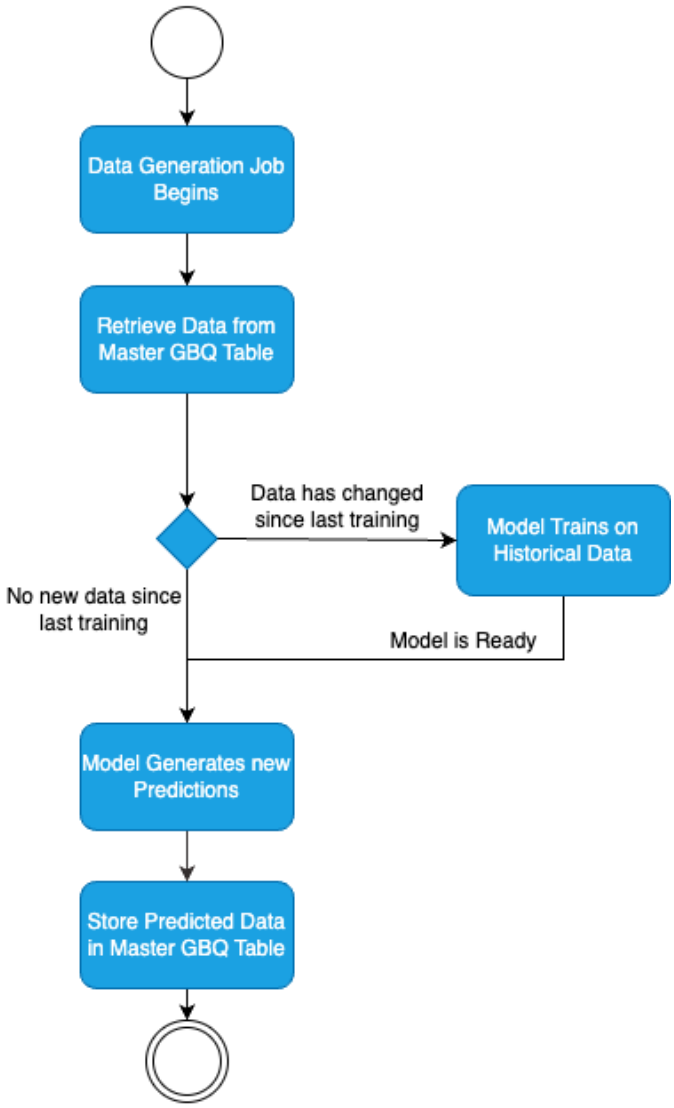
These diagrams were created for milestone 3 of our project and describe the flow of activity and information to the system. They are placed here because they do not reflect the final pipeline as it is now, but were still useful in thinking out how our system would interact with the data along different stages, and how a visualization of data would work. Ultimately our historical descriptive analysis visualization and predictive visualizations are two different things, and there was never a need for recurring batch data (though as mentioned in project outcomes above, that is an area of potential improvement for the project.) We include the diagrams here for insight.



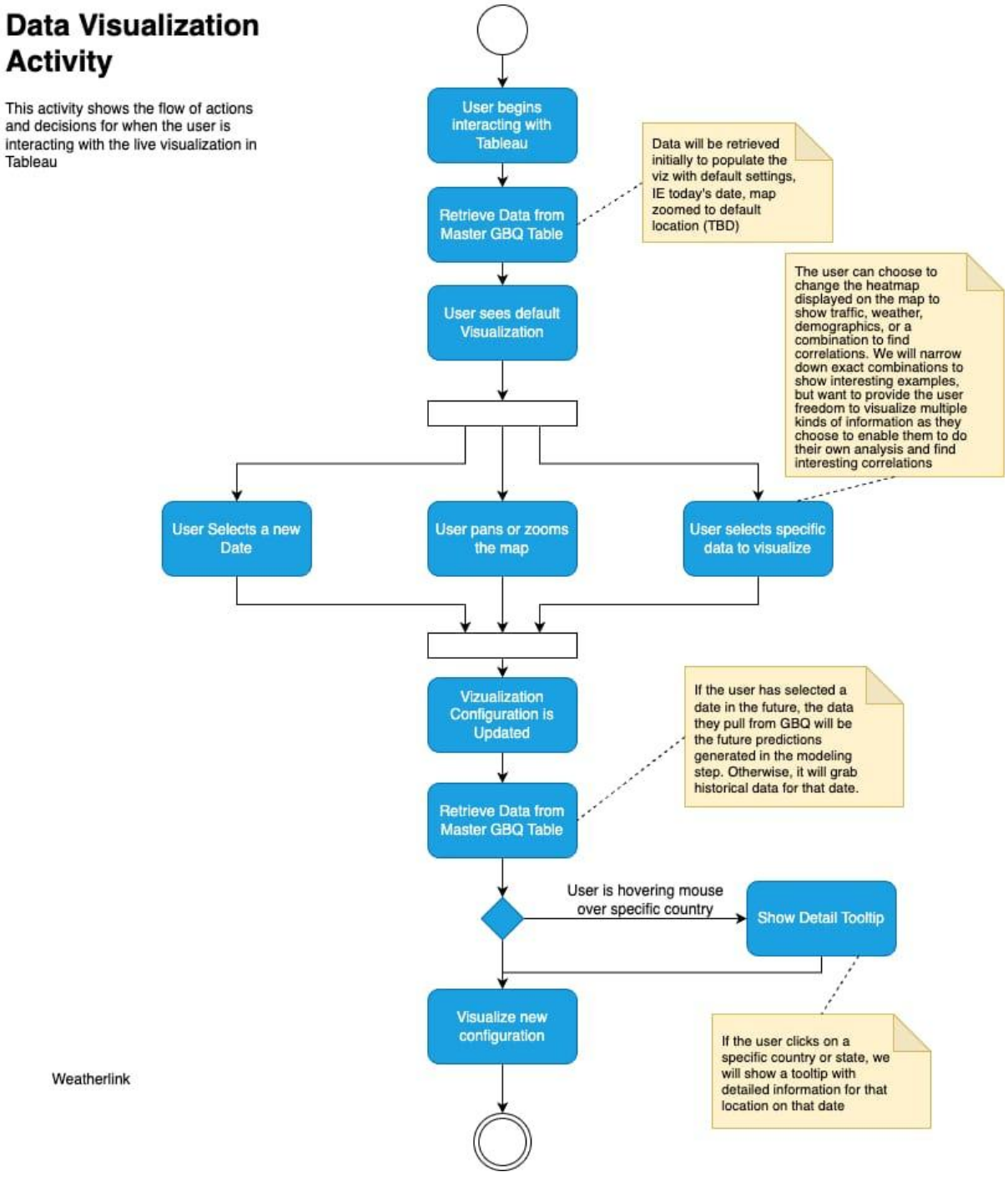
Data Model Generation Activity

This activity shows how the system will generate data to visualize for future dates, predicting what will happen based off of previous data from prior years. This will allow the user to visualize historical data, as well as see a probable future scenario.

This will likely run monthly, to generate more future predictions as historical data replaces previous predictions.



Weatherlink


Data Visualization Activity
This activity shows the flow of actions and decisions for when the user is interacting with the live visualization in Tableau
User begins interacting with Tableau
Retrieve Data from Master GBQ Table
Data will be retrieved initially to populate the viz with default settings, IE today's date, map zoomed to default location (TBD)
User sees default Visualization
The user can choose to change the heatmap displayed on the map to show traffic, weather, demographics, or a combination to find correlations. We will narrow down exact combinations to show interesting examples, but want to provide the user freedom to visualize multiple kinds of information as they choose to enable them to do their own analysis and find interesting correlations
User Selects a new Date
User pans or zooms the map
User selects specific data to visualize
Vizualization Configuration is Updated
If the user has selected a date in the future, the data they pull from GBQ will be the future predictions generated in the modeling step. Otherwise, it will grab historical data for that date.
Retrieve Data from Master GBQ Table
User is hovering mouse over specific country
Show Detail Tooltip
Visualize new configuration
If the user clicks on a specific country or state, we will show a tooltip with detailed information for that location on that date
Weatherlink

7.2 Authors

Alper Cetinkaya

alper.cetinkaya@gwu.edu

Aditya Sanjay Gujral

adityagujral@gwu.edu

Tharun Saravanan

tsaravanan@gwu.edu

Matthew Winchester

matthew.winchester@gwu.edu