Case Study 1- Data Analysis of the NOAA storm events dataset

| **Summary** | In this codelab,we will be analysing storms to build up an understanding towards the Climate Risk in the United States and to create an analytical system that can be used to understand storms. |
| --- | --- |
| **URL** | your-first-pwapp |
| **Category** | Data Analysis |
| **Environment** | Google Colab, |
| **Status** | Not Published |
| **Author** | Kartik Kumar and Tanvi Tembhurne |

[Introduction](#_ok7k5uux6)

[Overview](#_97rh5z70mj9)

[Ingestion](#_5hnrehwfd5fx)

[Exploration](#_yhr61i6p0o9n)

1. [Initial EDA](#_pjyqie1jc216)

[1.1 Details](#_eff3hehni8rc)

[1.2 Locations](#_7d8xp7z7epq7)

[1.3 Fatalities](#_to5z1gwmoy96)

1. [Pandas Profiling](#_uidwuxftyn6o)

[2.1 Details](#_t4mv5behccd9)

[2.2 Locations](#_alhrv3ubyats)

[2.3 Fatalities](#_63i2v1ntcx1i)

1. [Great Expectations](#_l9p7ng6ui8kj)

[3.1 Report Configurations](#_5fz2n1soa9r6)

[3.2 Primary key(EVENT\_ID) null check](#_bd6m82ozfxel)

[3.3 String Type Expectation](#_n8gbugv6v9pd)

[3.4 Discrete Value Validation with count](#_5f8afuefpymb)

1. [GCS Bucket Upload](#_dp6jas4arjx9)
2. [Apache-Airflow](#_9rzpg2uzleoe)

5.1 [DAGs](#_te1kacauj5ju)

1. [Overviews](#_y3uv20ujrpkq)
2. [Exercise](#_28srmckjllu0)

7.1 [connect\_to\_gcs](#_59dxscnkj2es)

7.2 [connect\_to\_gcs\_read\_details](#_rb40pxaoebeh)

7.3 [connect\_to\_gcs\_read\_fatalities](#_c0s01fg6hjr7)

7.4 [clean\_details\_df](#_synifzuqi2cm)

7.5 [clean\_fatalities\_df](#_ceq20r6jhr2x)

7.6 [merge\_details\_fatalities](#_39200c23ek03)

7.7 [upload\_to\_gcs](#_wlgen35xk8ij)

1. [Pipeline](#_kmdj95kih228)
2. [Visualization](#_xoyq4iwc0f0z)

9.1 [Use Cases](#_oqlfqvio34n8)

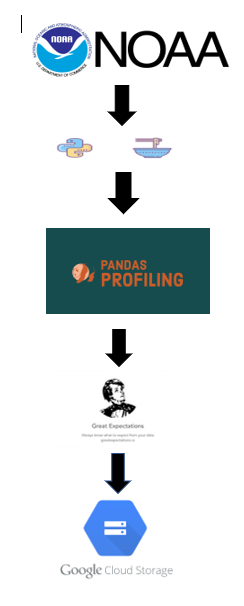
# Introduction

This case study is dedicated to analysing storms all across in the United States. So we chose our dataset from official website which does daily weather forecasts, severe storm warnings, and climate monitoring- NOAA



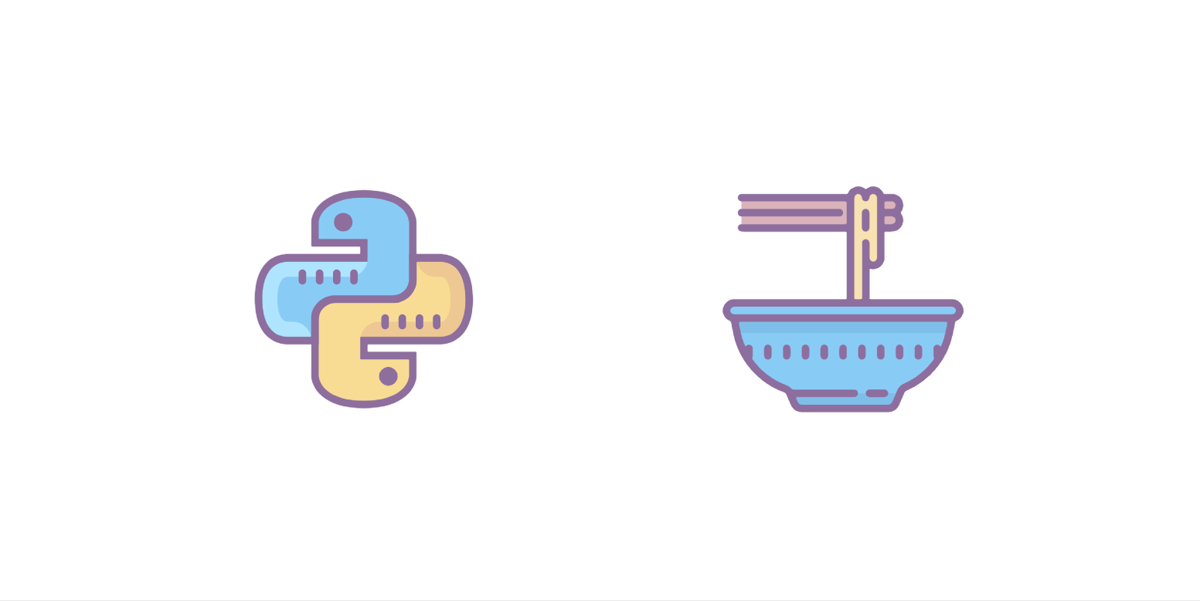
Here is the [link](https://www.ncdc.noaa.gov/stormevents/ftp.jsp) to their Database.

# Overview

****

# Ingestion

Pulling the HTML of the CSVs available on the website using BeautifulSoup, it is a python library that allows us to efficiently and easily pull out the information from HTML.



We were successfully able to retrieve URLs of the CSVs, currently we chose to work on the 2017-2021 year’s dataset.

After retrieval we unzipped the gzip files and combined them into master data frames of three categories:

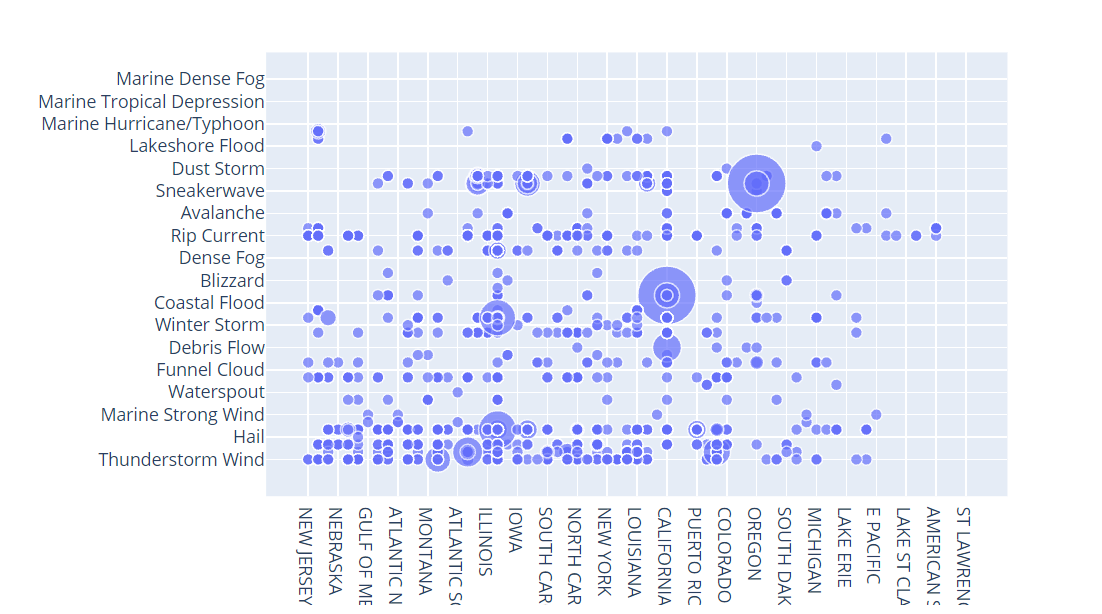
1. Details
2. Fatalities
3. Locations

# Exploration

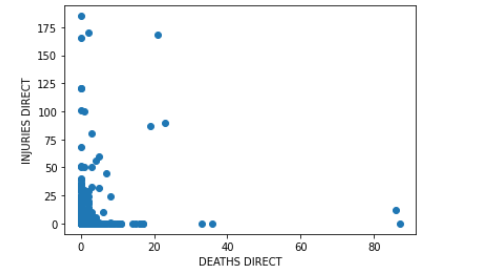
Exploratory data analysis is very important as well as time consuming. Let’s look at the initial EDA.

## Conclusions of Initial EDA:

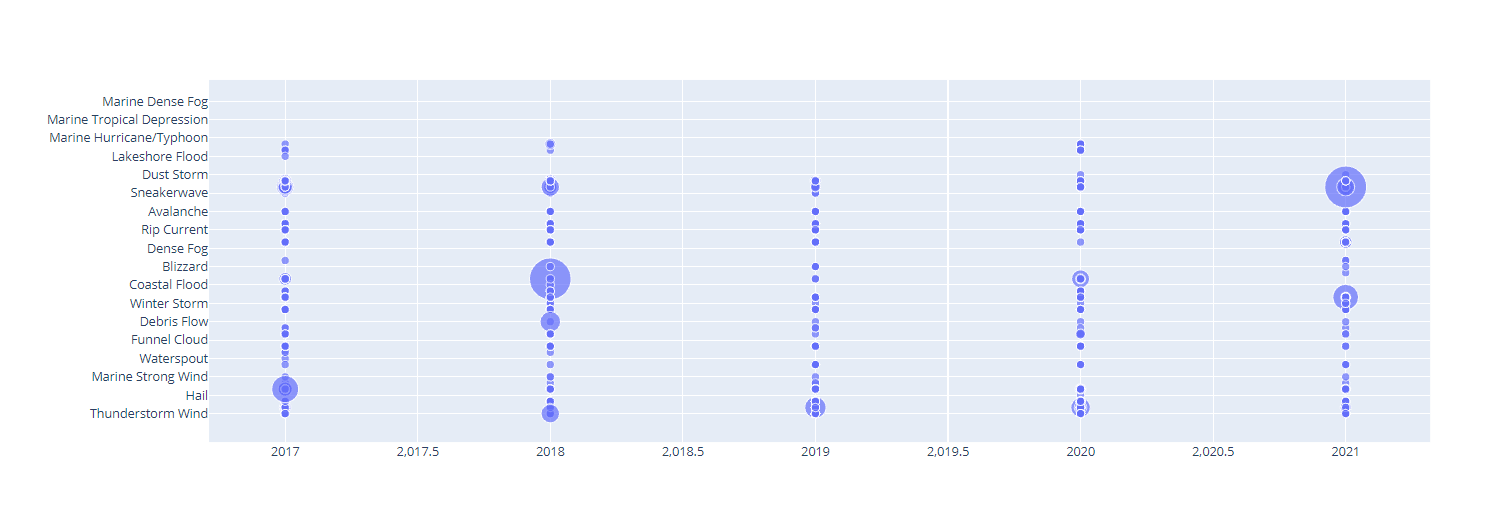
### Details



The above graph visualizes Event\_type with respect to the States in the US. Here we can see Puerto Rico faces a lot of Coastal Flood.

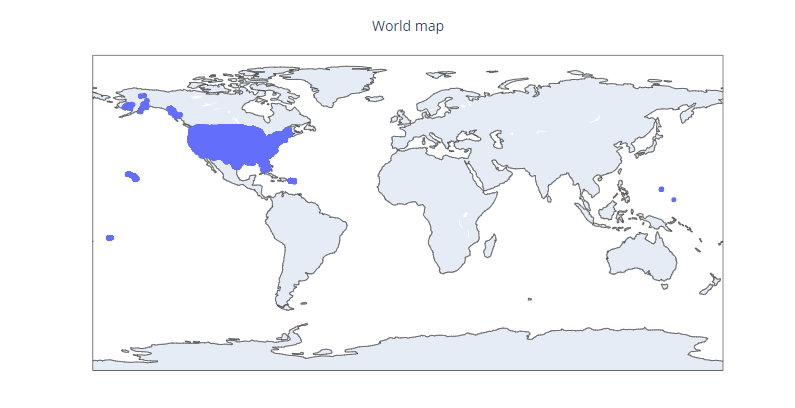


The above graph visualize correlation between Direct\_Deaths and Direct\_Injuries



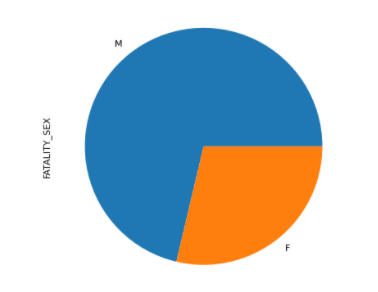
The above graph visualizes Event\_type with respect to Year. So here we can see in 2017 Hail events had maximum frequency.

### Locations



The above world map is plotted using latitude and longitude, So here we can see that storm event database has which areas are under observation.

### Fatalities



The above graph shows that the maximum count of fatality is male.

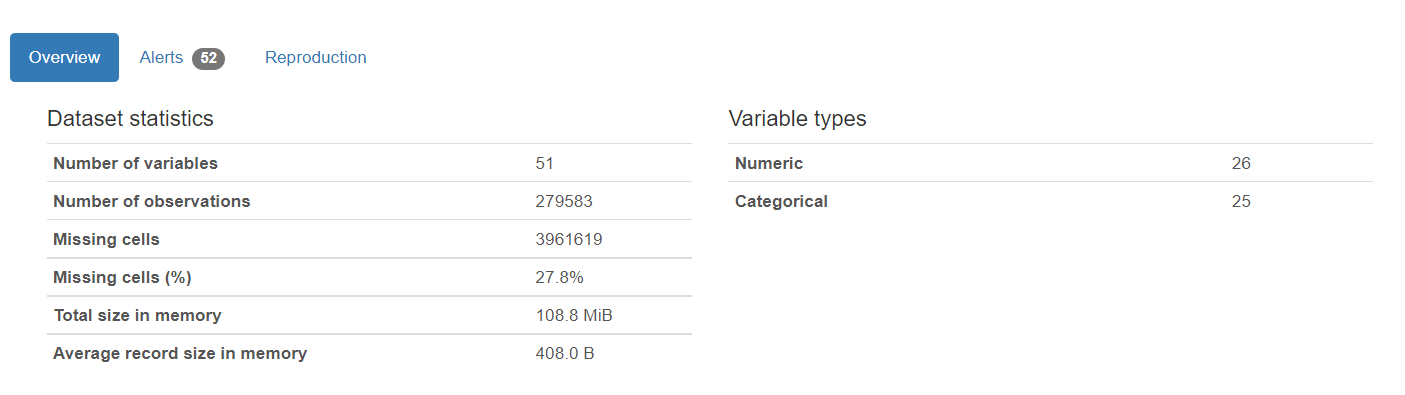
## Pandas Profiling

We used Pandas profiling-is an open source Python module with which we can quickly do an exploratory data analysis.



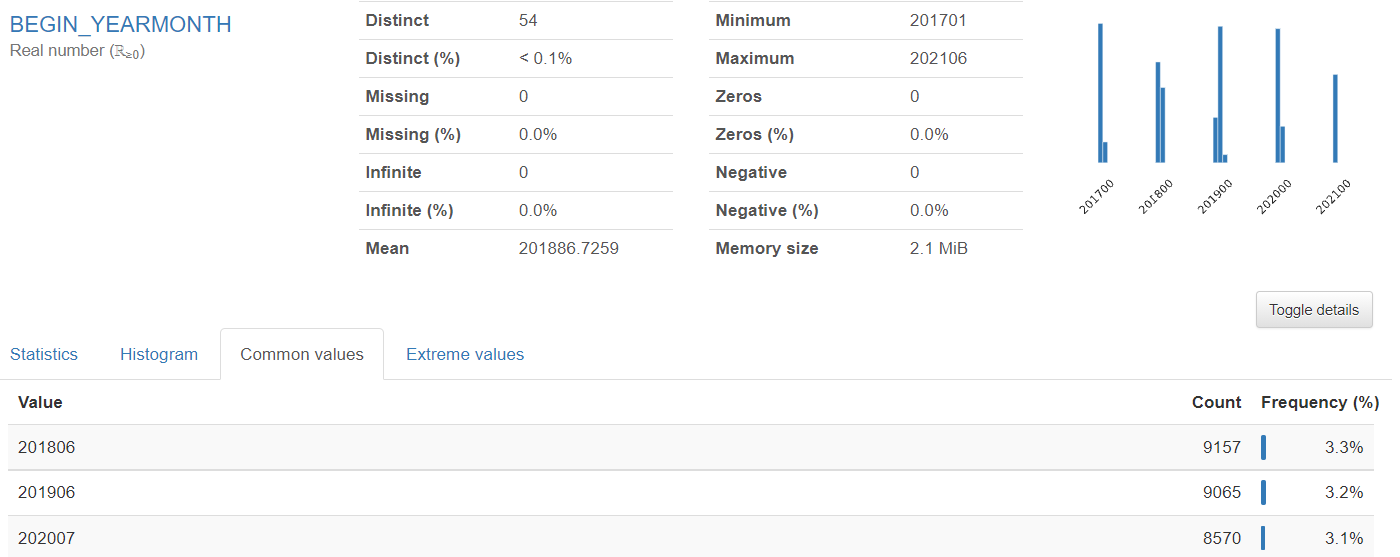
Conclusions of EDA

### Details:



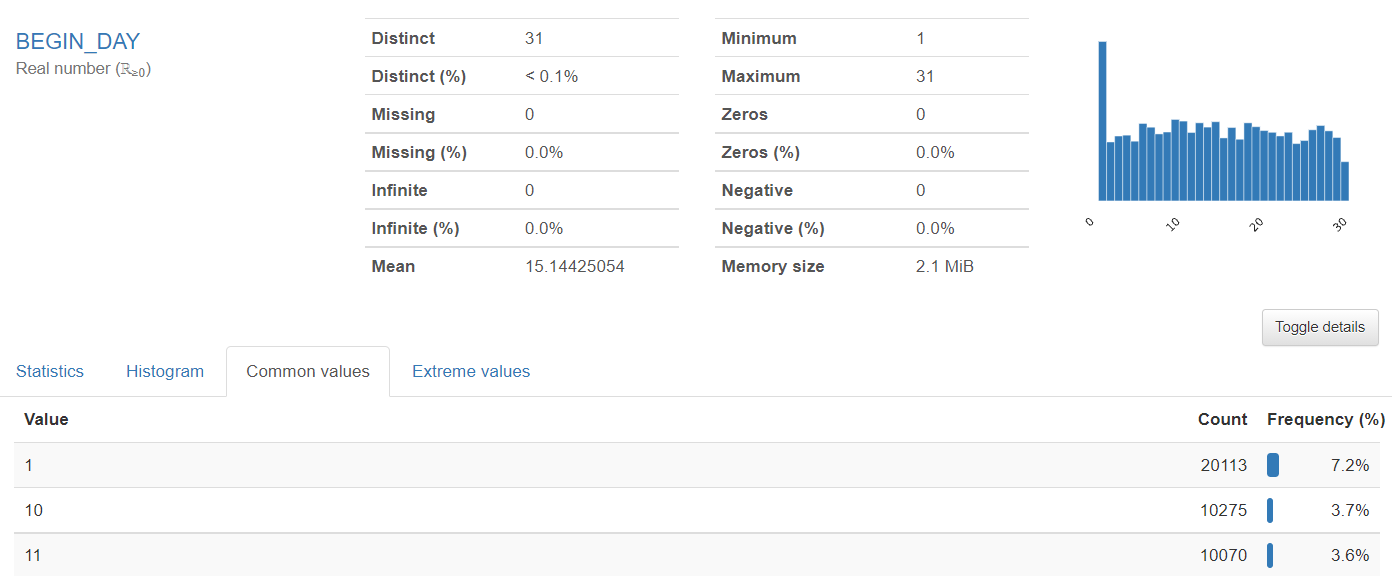
1. Begin\_YearMonth

Ex: 201212 (YYYYMM format) The year and month that the event began



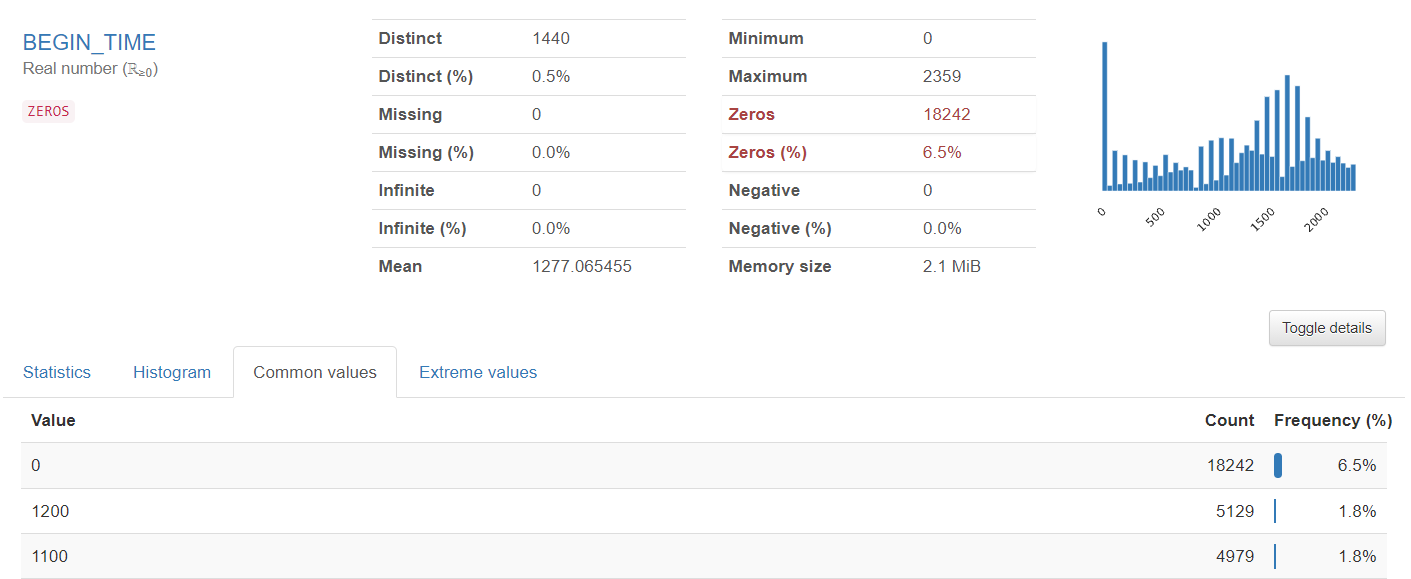
1. BEGIN\_DAY

Ex: 31 (DD format) The day of the month that the event began



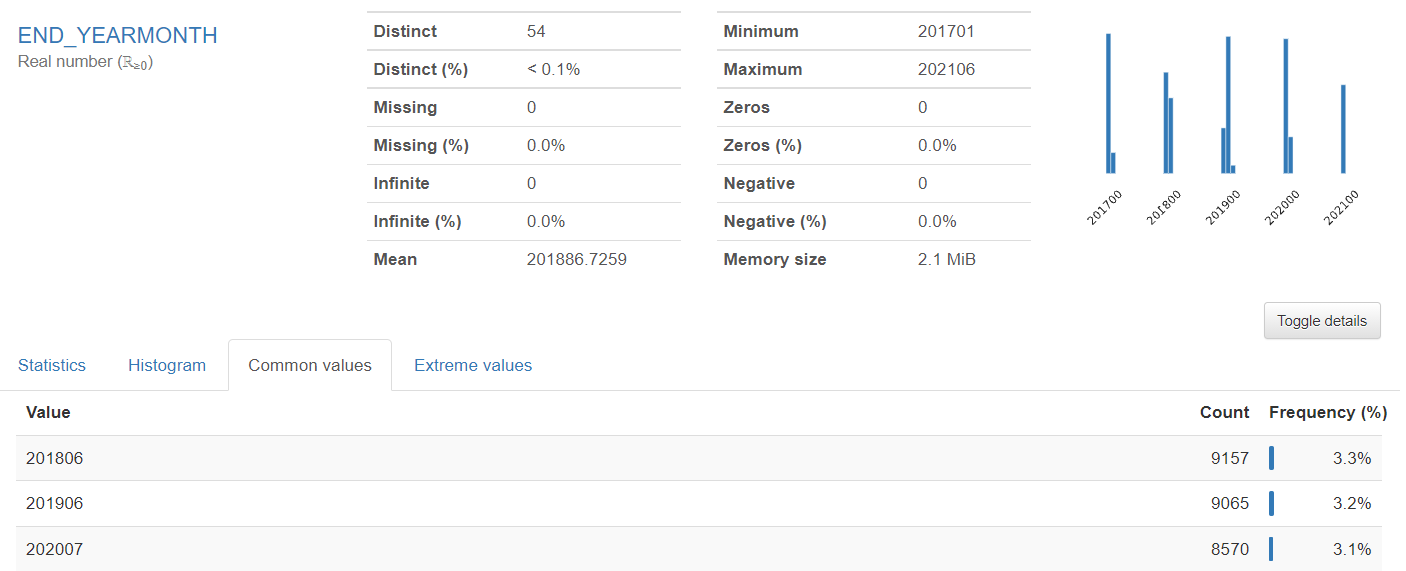
1. BEGIN\_TIME

Ex: 2359 (hhmm format) The time of day that the event began



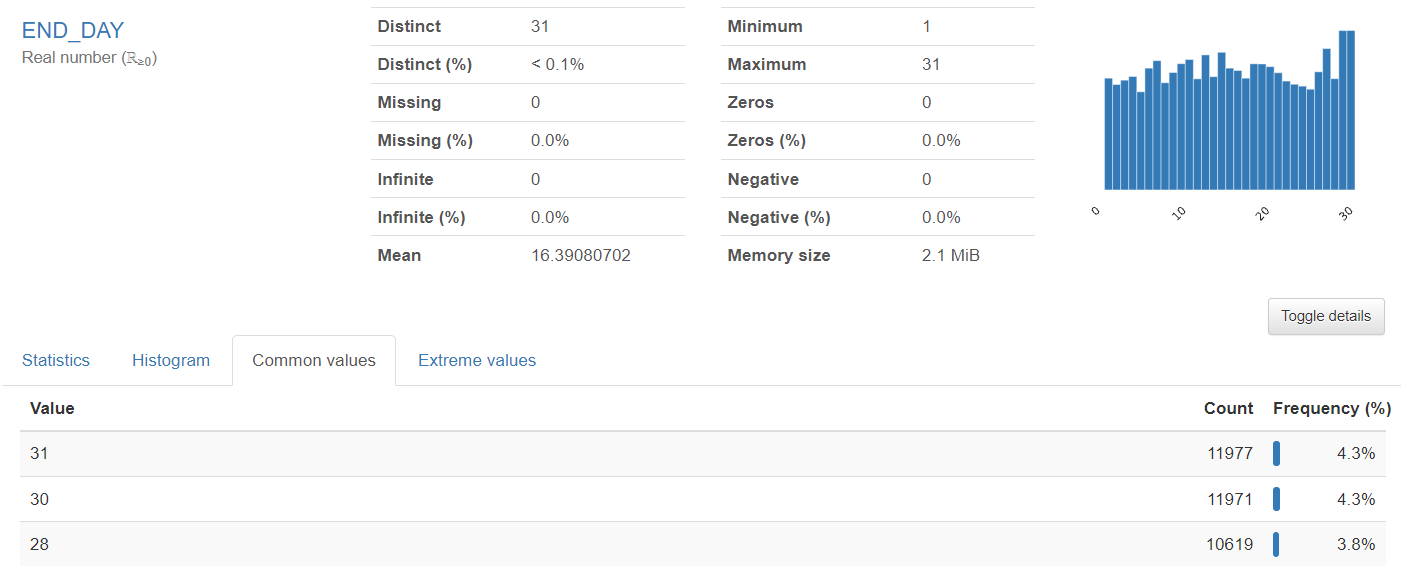
1. END\_YEARMONTH

Ex: Ex: 201301 (YYYYMM format) The year and month that the event ended



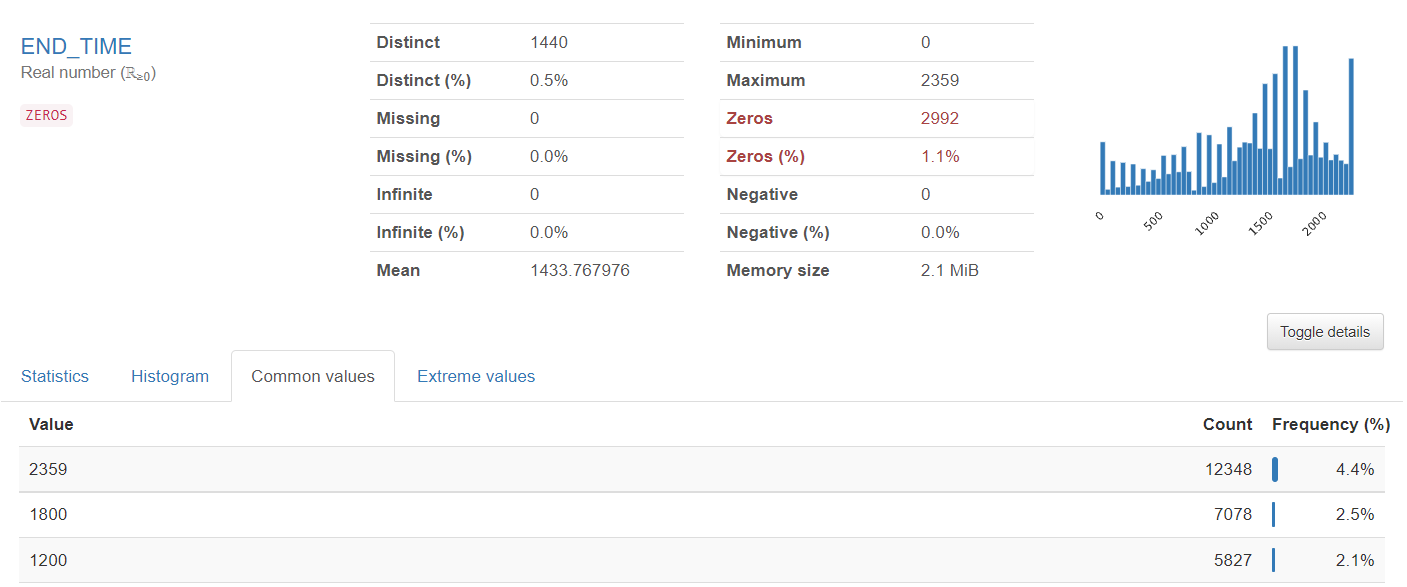
1. END\_DAY

Ex: 01 (DD format) The day of the month that the event ended



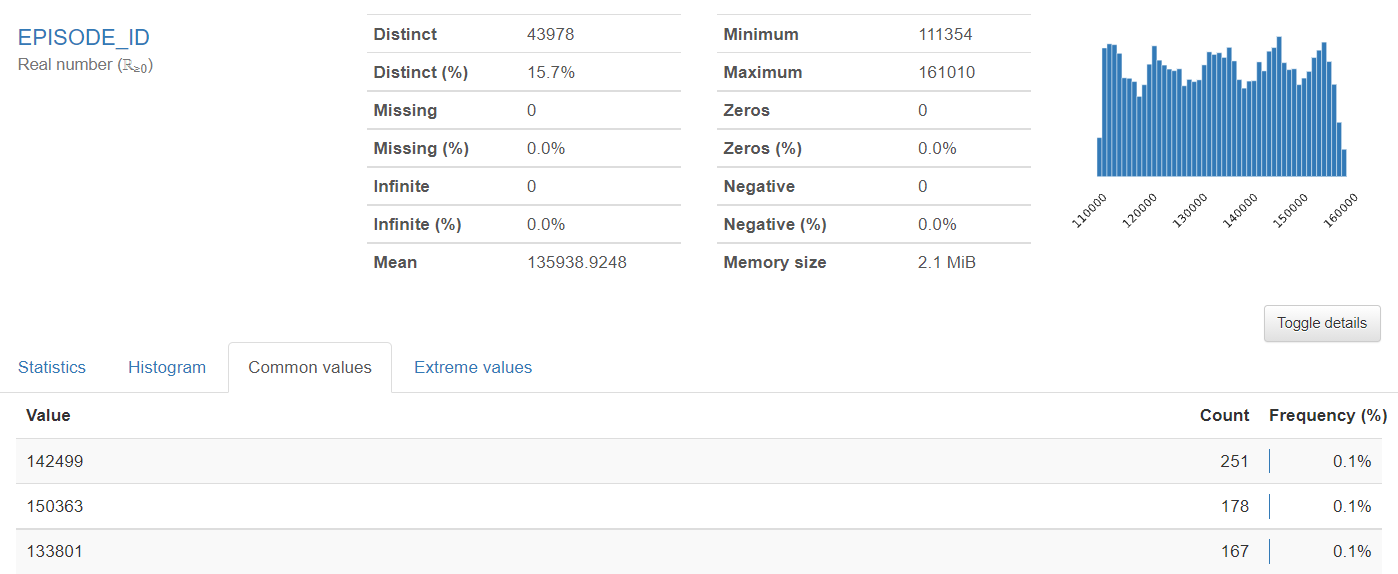
1. END\_TIME

Ex: 0001 (hhmm format) The time of day that the event ended



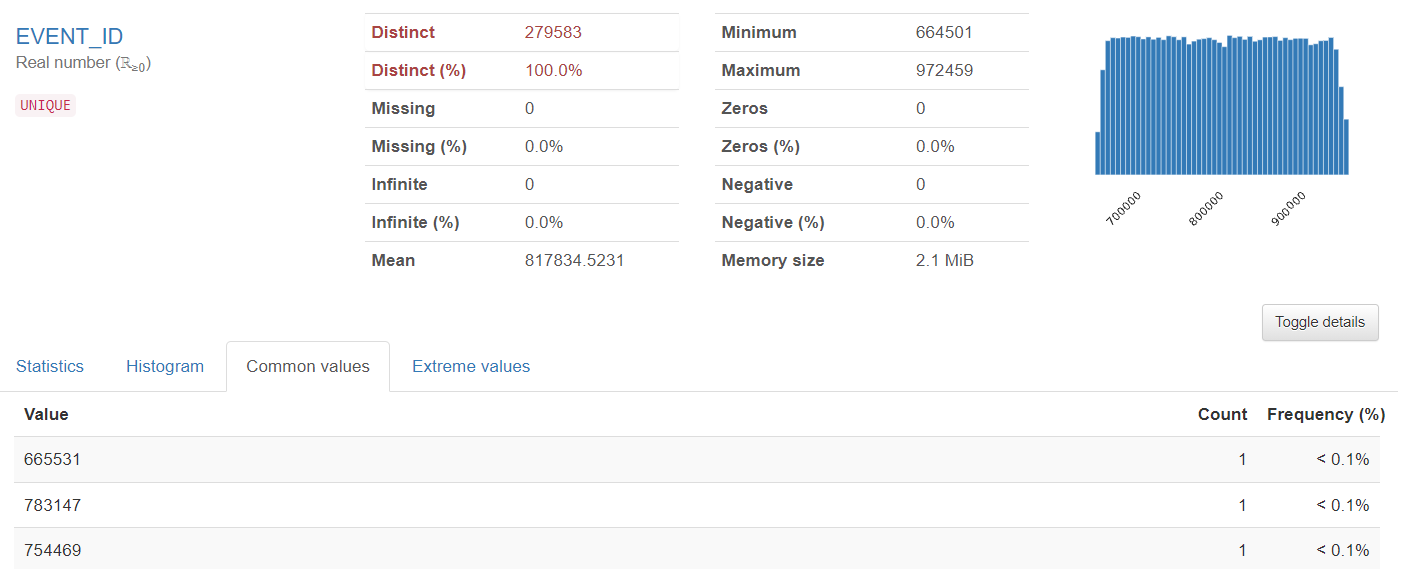
1. EPISODE\_ID

Ex: 61280, 62777, 63250 ID assigned by NWS to denote the storm episode; Episodes may contain multiple Events. The occurrence of storms and other significant weather phenomena having sufficient intensity to cause loss of life, injuries, significant property damage, and/or disruption to commerce.

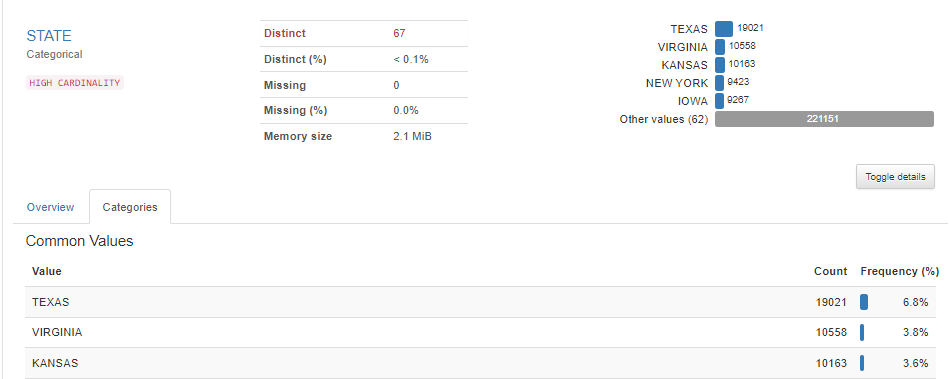


1. EVENT\_ID

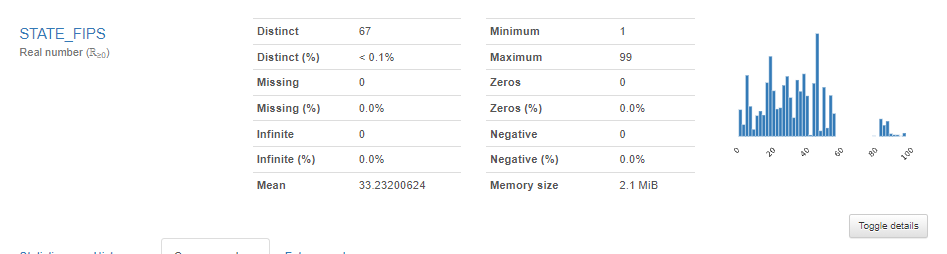
Ex: 383097, 374427, 364175 ID assigned by NWS for each individual storm event contained within a storm episode; links the record with the same event in the storm\_event\_details, storm\_event\_locations and storm\_event\_fatalities tables (Primary database key field).



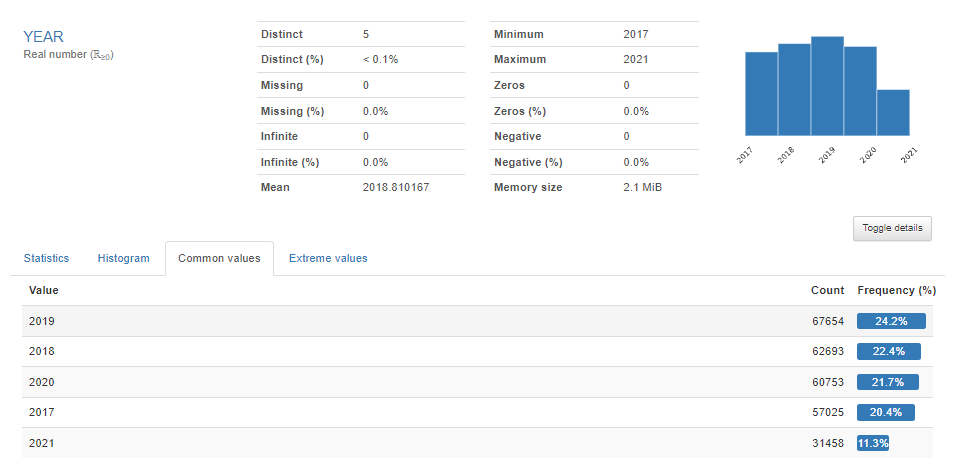
1. STATE



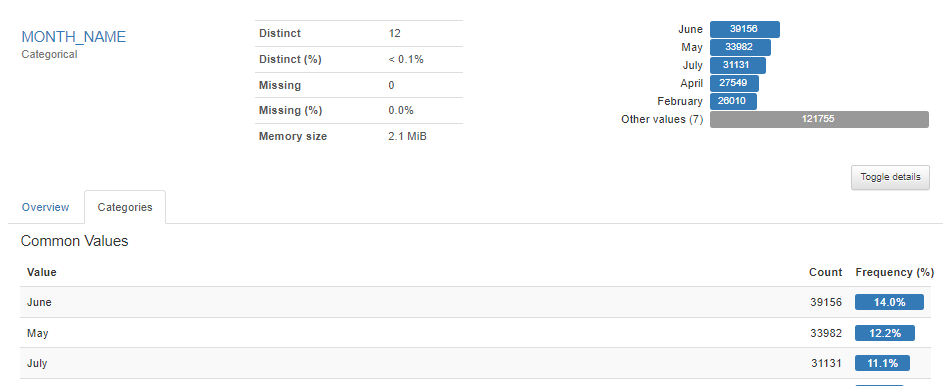
1. STATE\_FIPS



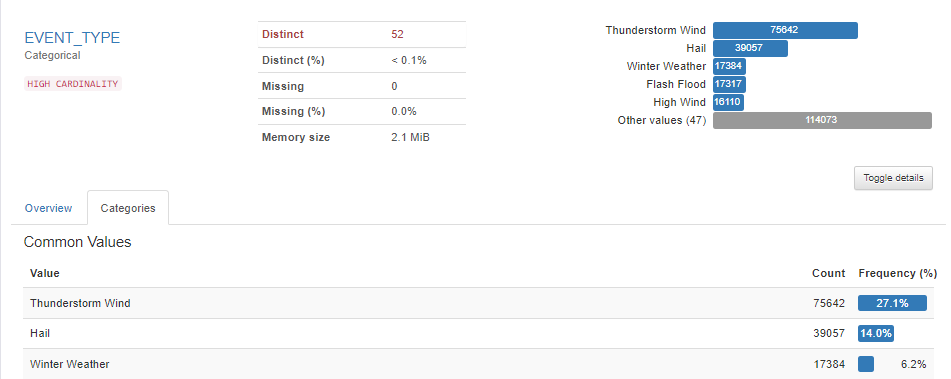
1. YEAR



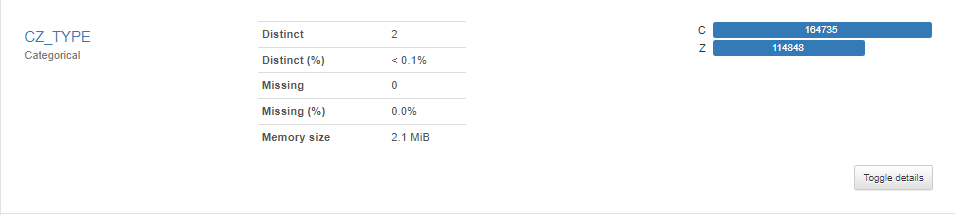
1. MONTH\_NAME



1. EVENT\_TYPE



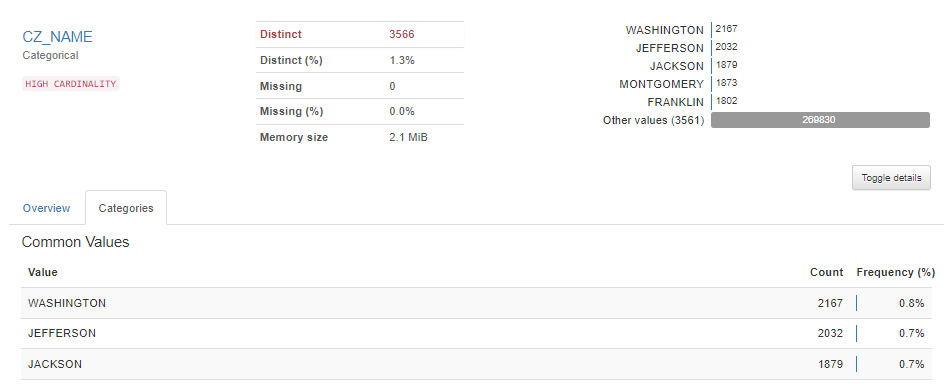
1. CZ\_TYPE



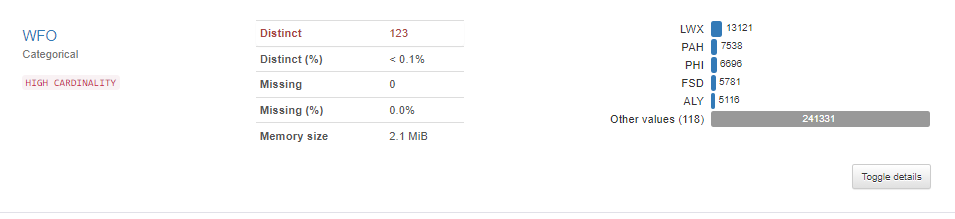
1. CZ\_FIPS



1. CZ\_NAME



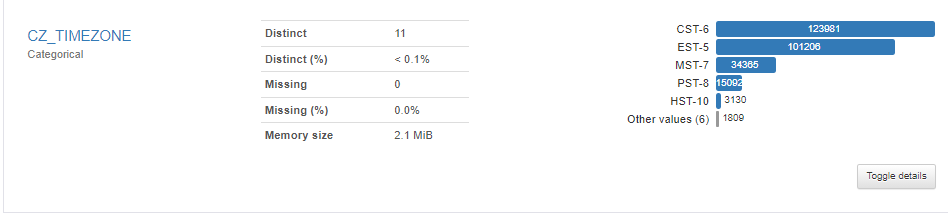
1. WFO



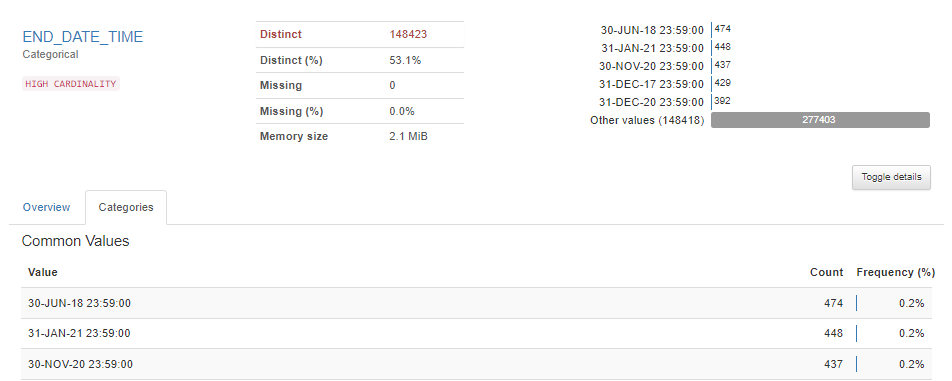
1. BEGIN\_DATE\_TIME



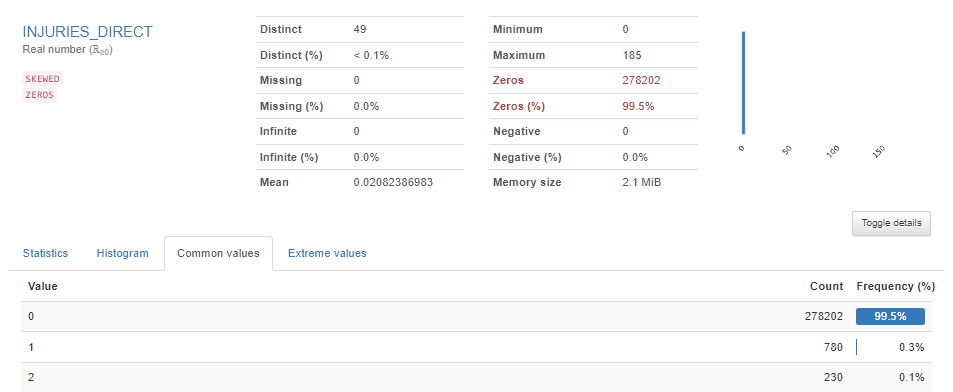
1. CZ\_TIMEZONE



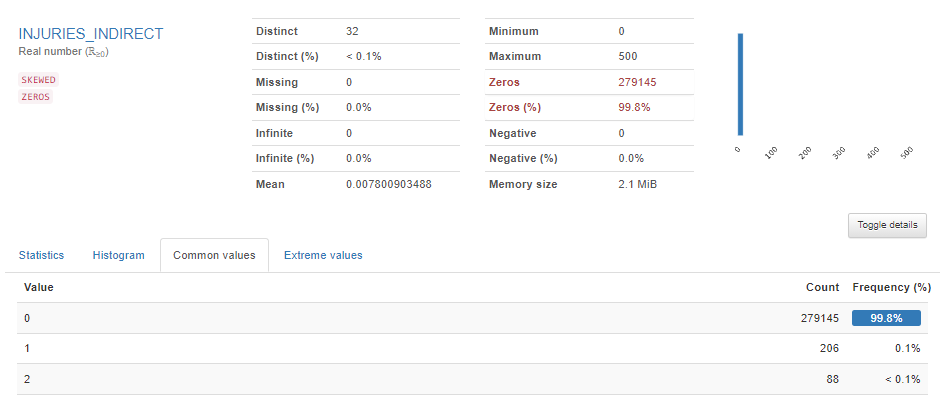
1. END\_DATE\_TIME



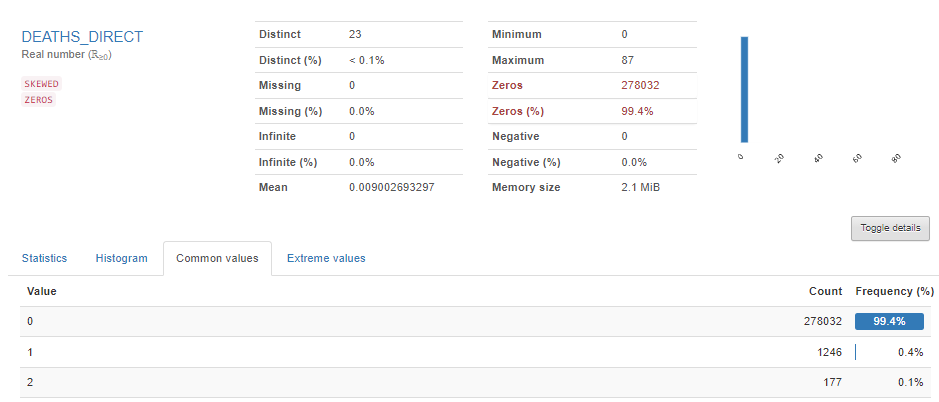
1. INJURIES\_DIRECT



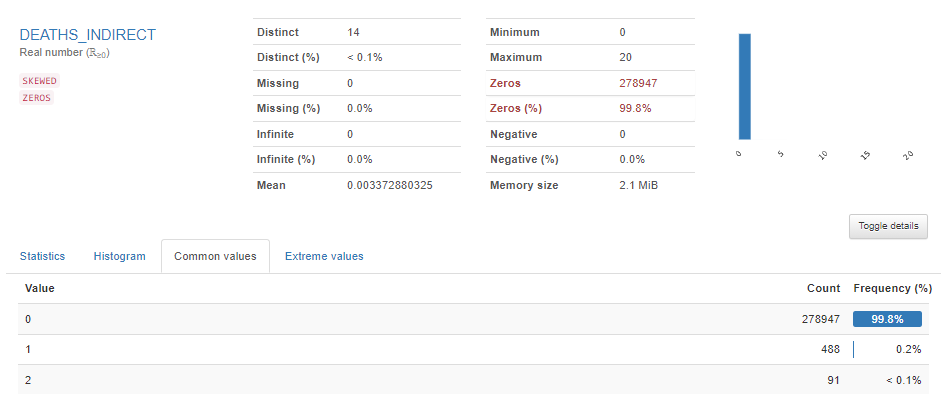
1. INDIRECT\_INJURIES



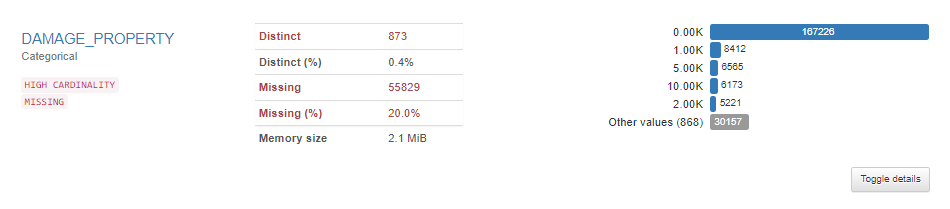
1. DEATHS\_DIRECT



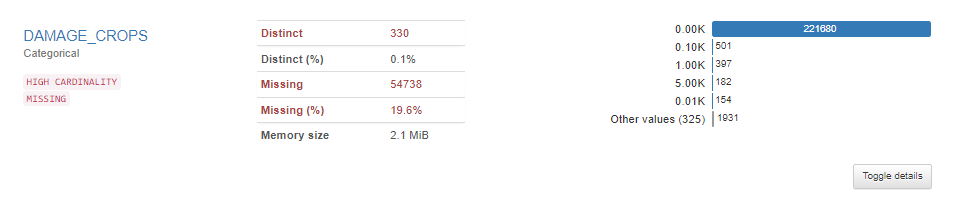
1. DEATHS\_INDIRECT



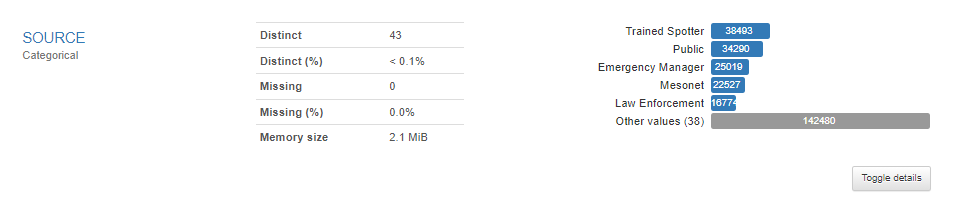
1. DAMAGE\_PROPERTY



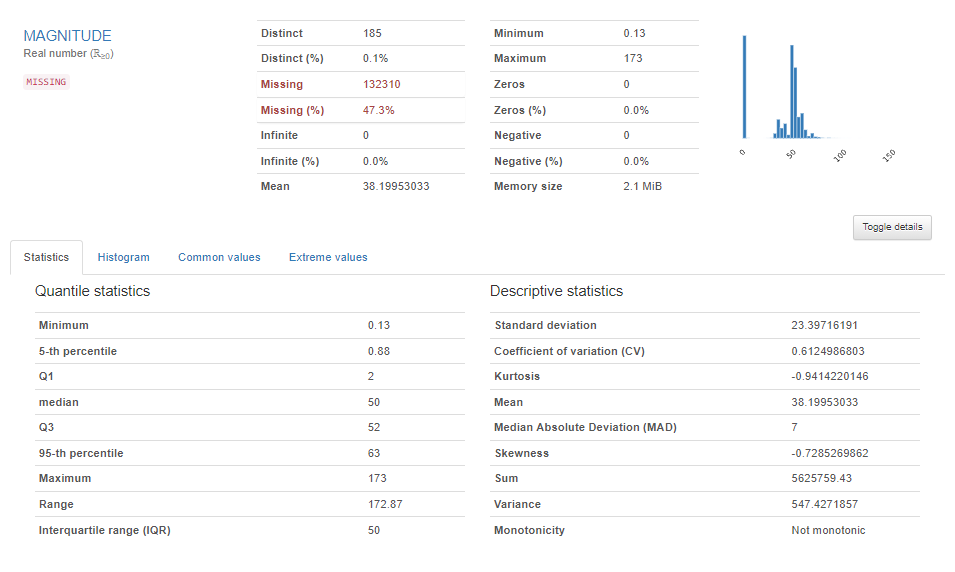
1. DAMAGE\_CROPS



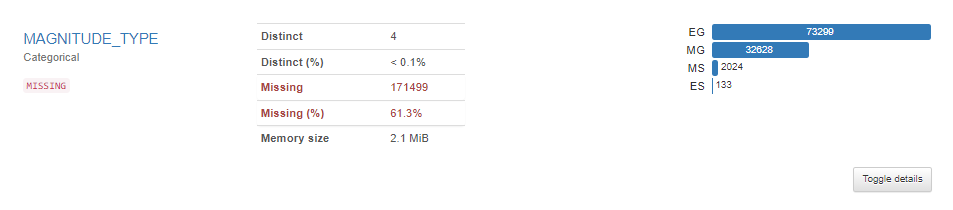
1. SOURCE



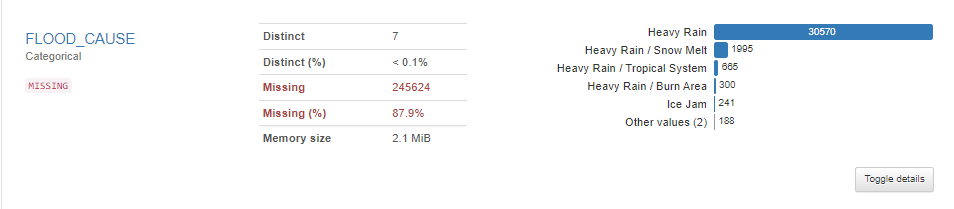
1. MAGNITUDE



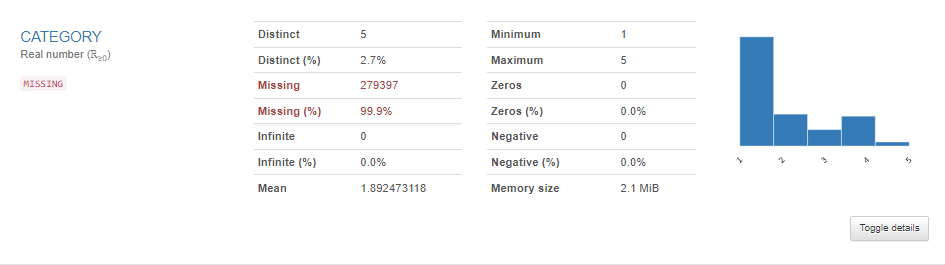
1. MAGNITUDE\_TYPE



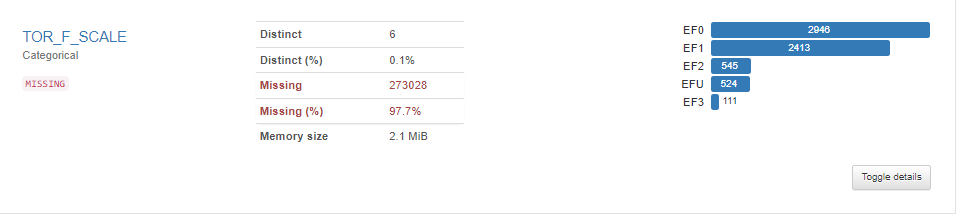
1. FLOOD\_CAUSE



1. CATEGORY



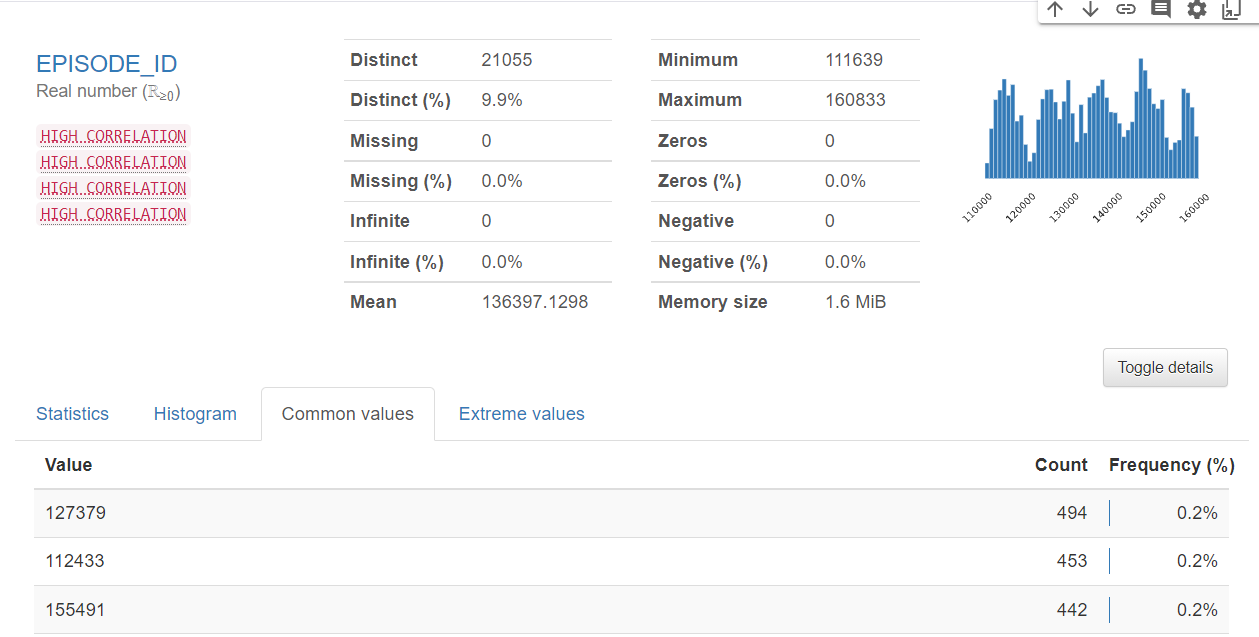
1. TOR\_F\_SCALE



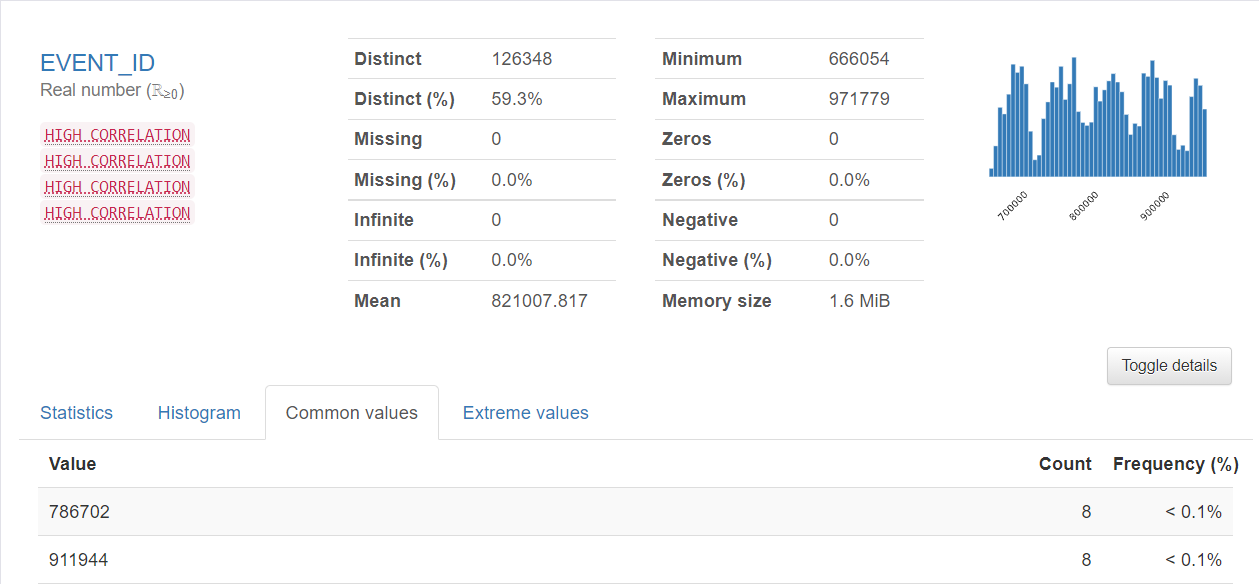
Similarly for the remaining 20 features.

### 2. Locations

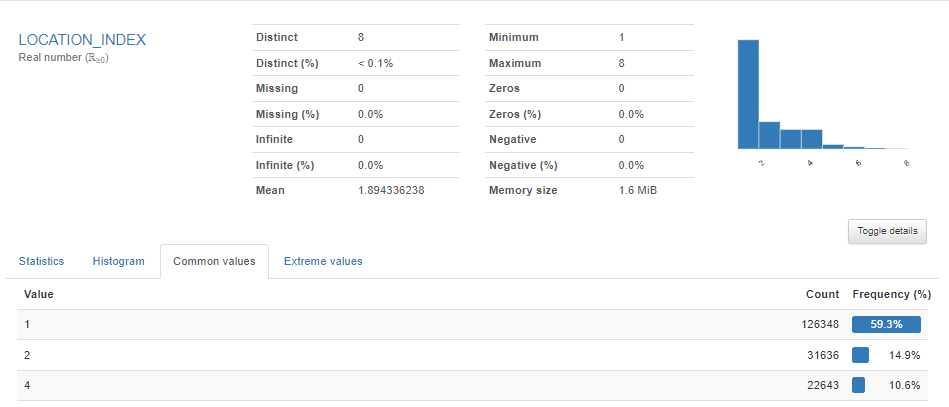
1. EPISODE\_ID



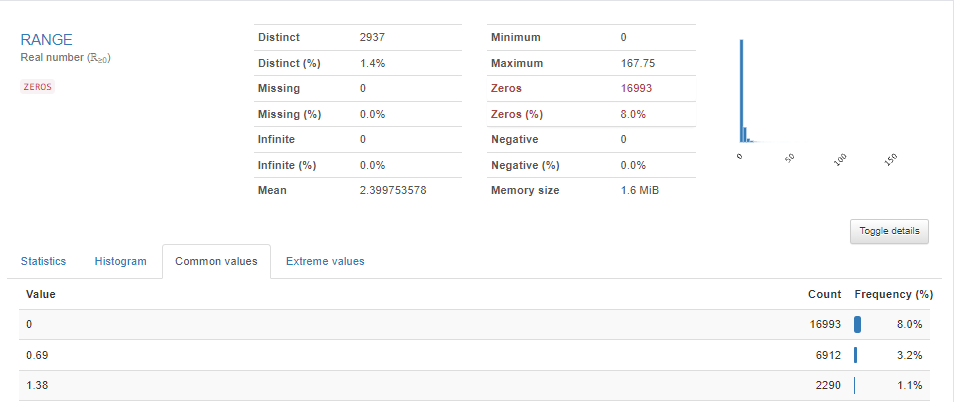
1. EVENT\_ID



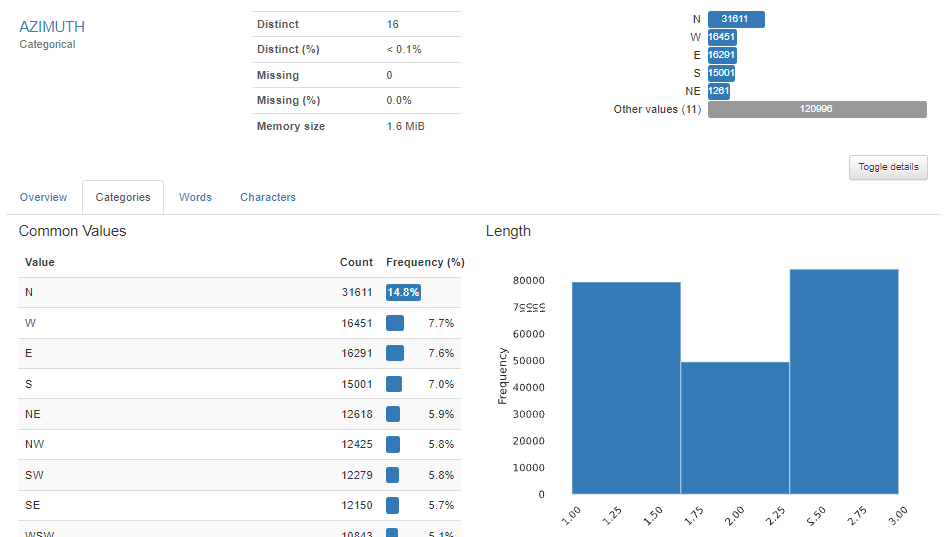
1. LOCATION\_INDEX



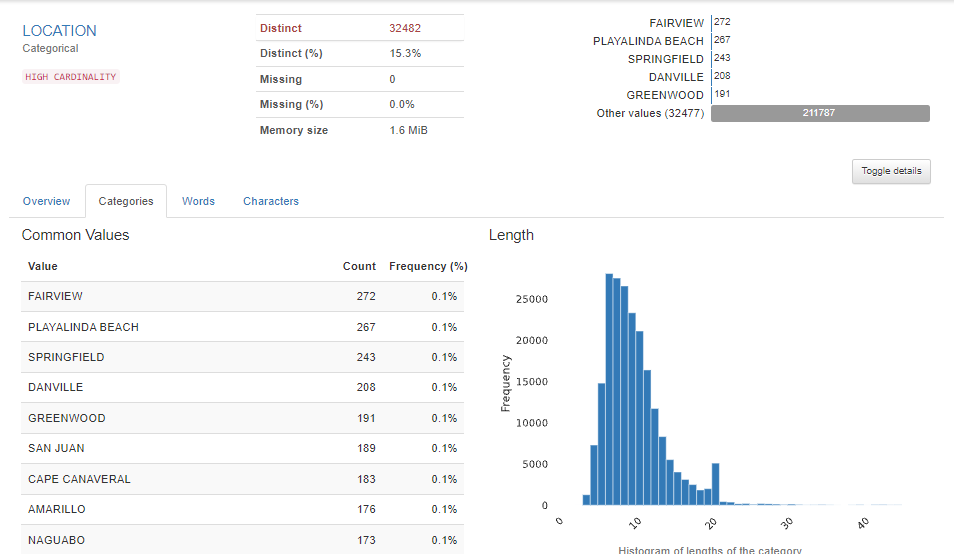
1. RANGE



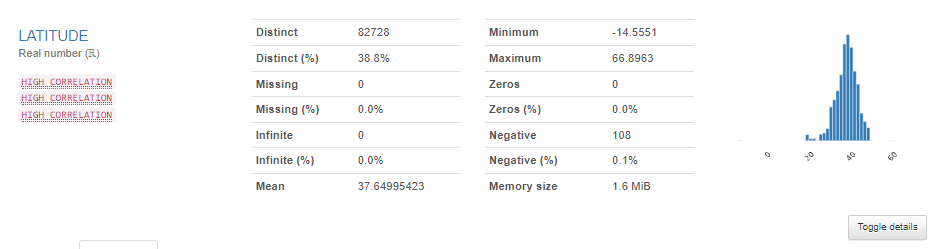
1. AZIMUTH



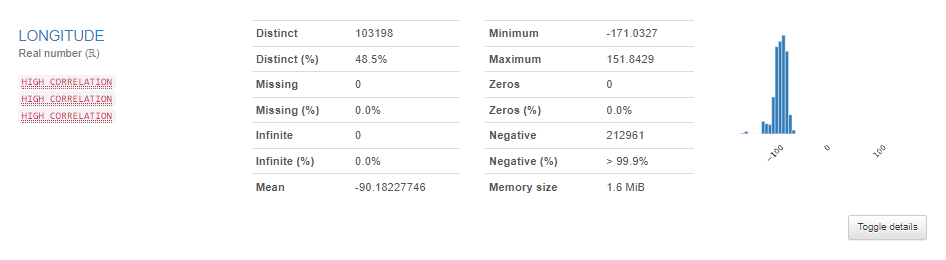
1. LOCATION



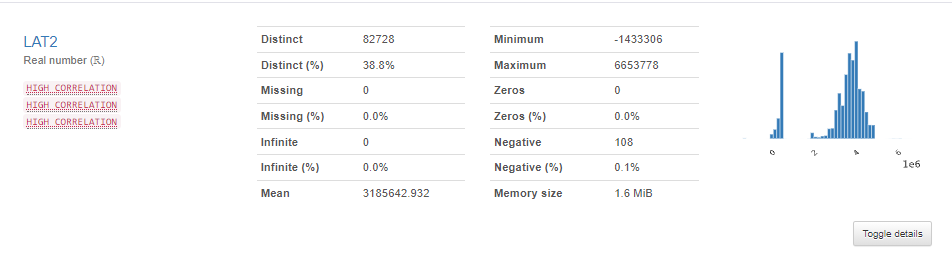
1. LATITUDE



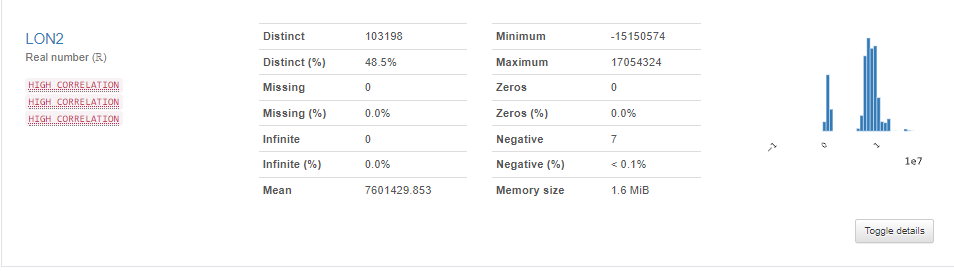
1. LONGITUDE



1. LAT2

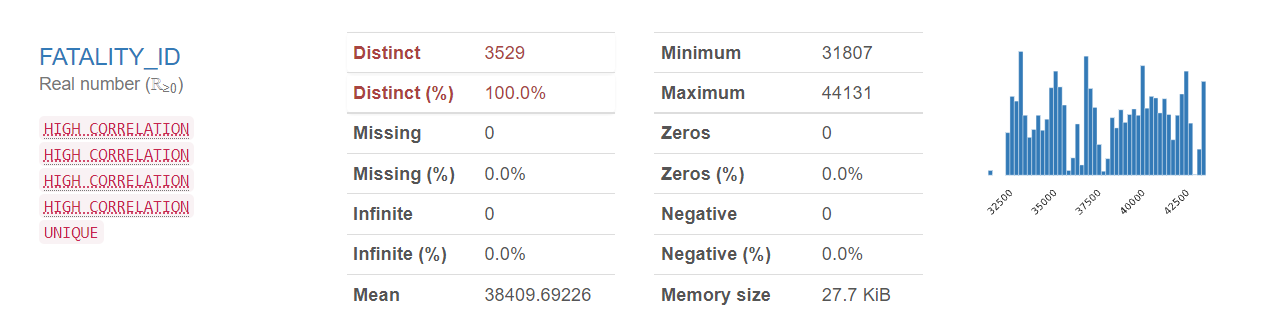


1. LON2

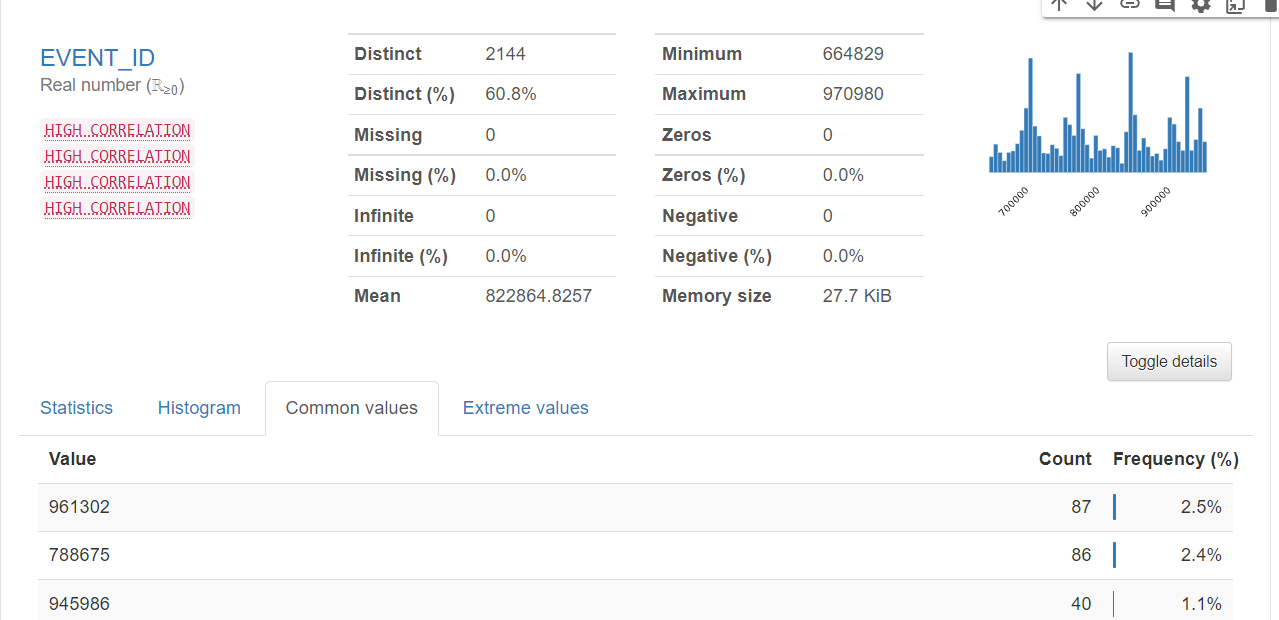


### 3. Fatalities

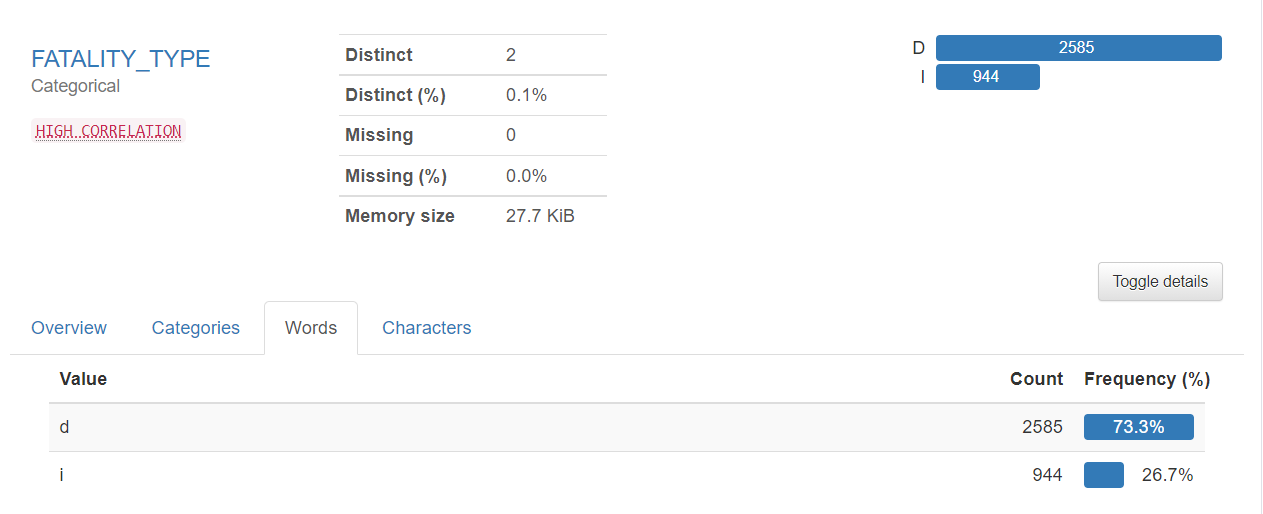
1. FATALITY\_ID



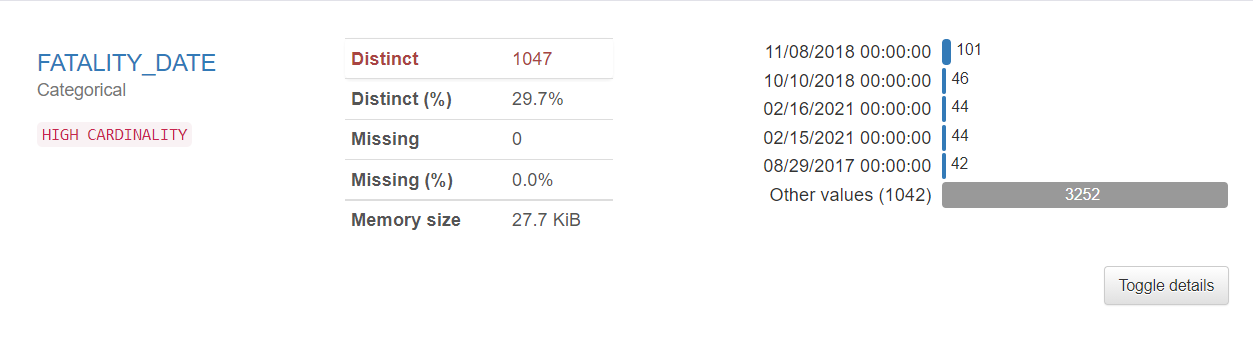
1. EVENT\_ID



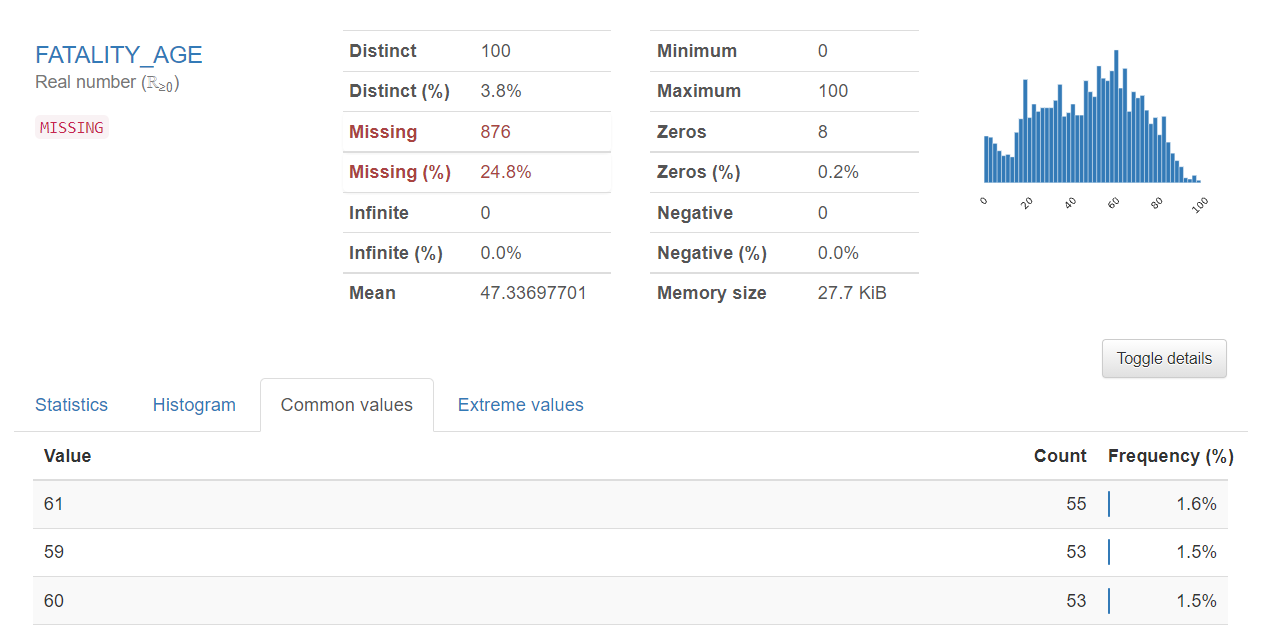
1. FATALITY\_TYPE



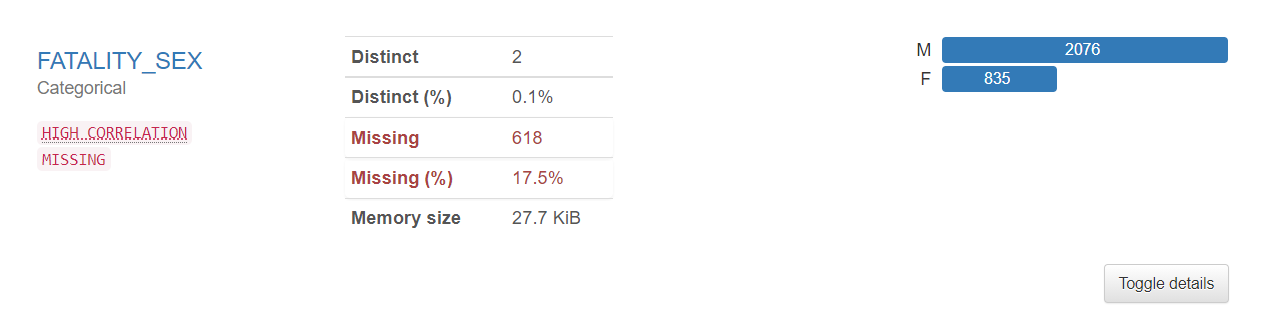
1. FATALITY\_DATE



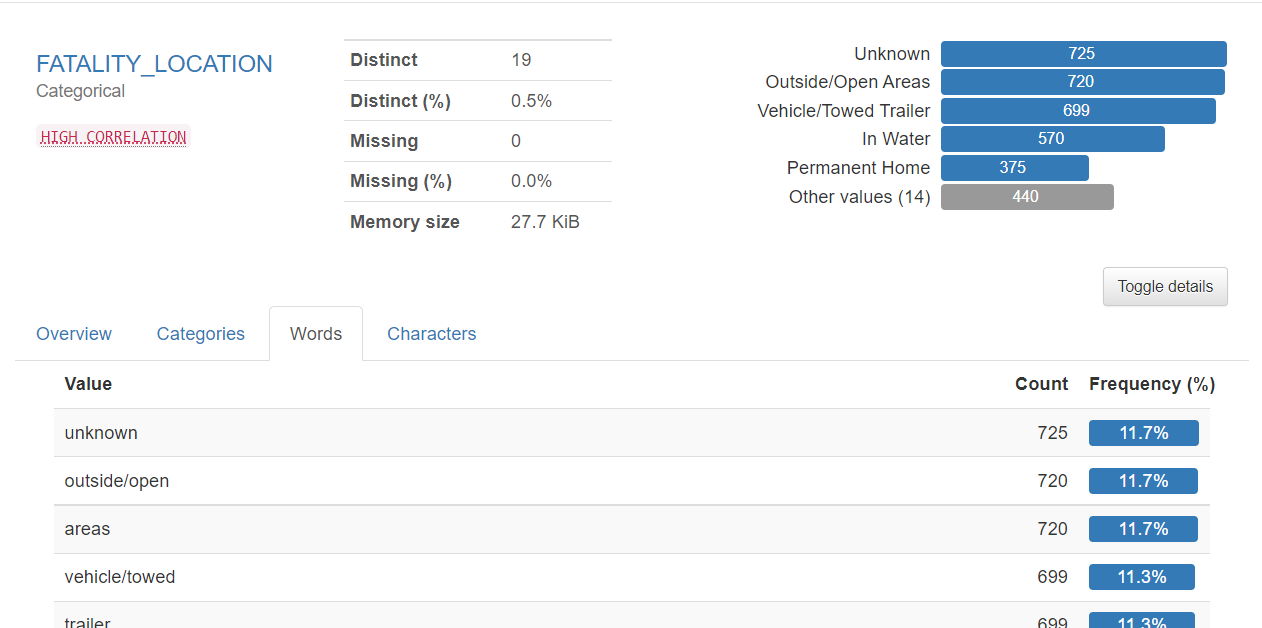
1. FATALITY\_AGE



1. FATALITY\_SEX



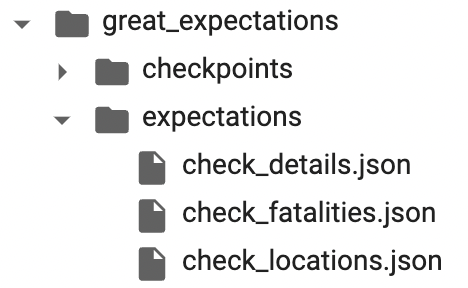
1. FATALITY\_LOCATION



## Great Expectations

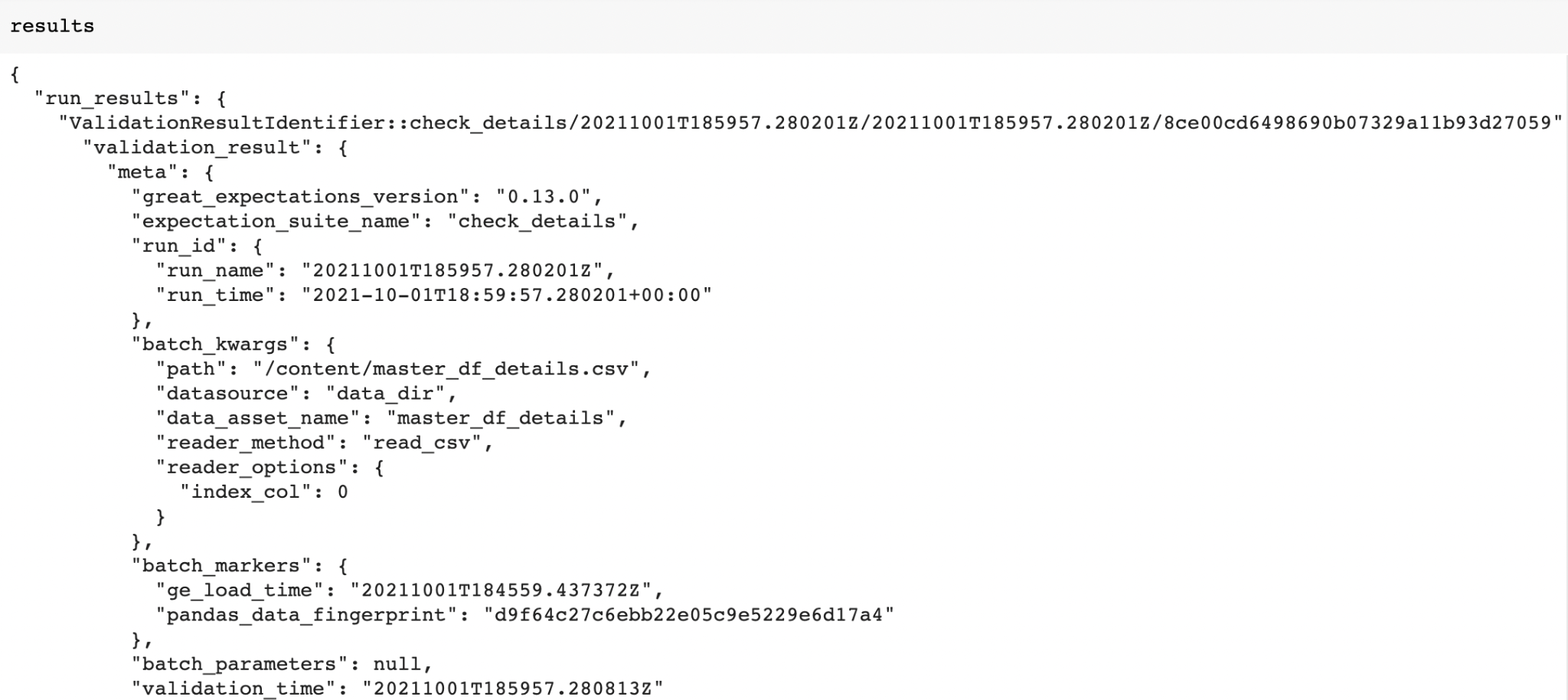
* Great Expectations is a Python library that allows you to validate the data
* Validations can be of multiple types - validating that there are no null values, validating data types of columns, validating the format of a date field, etc

We applied validations on all the 3 datasets and viewed results in json format



***Resulting JSON format***

Here’s how the results look:



### Report Configurations



### Primary key(EVENT\_ID) null check



### String Type Expectation

\* Note that the output of every expectation/validation returns the count of elements that match the expectation along with the end result - ‘success’:true/false



### Discrete Value Validation with count

## 

## 

## 

## Discrete Values in JSON



We have stored the unique values for all discrete columns in json format as shown above.

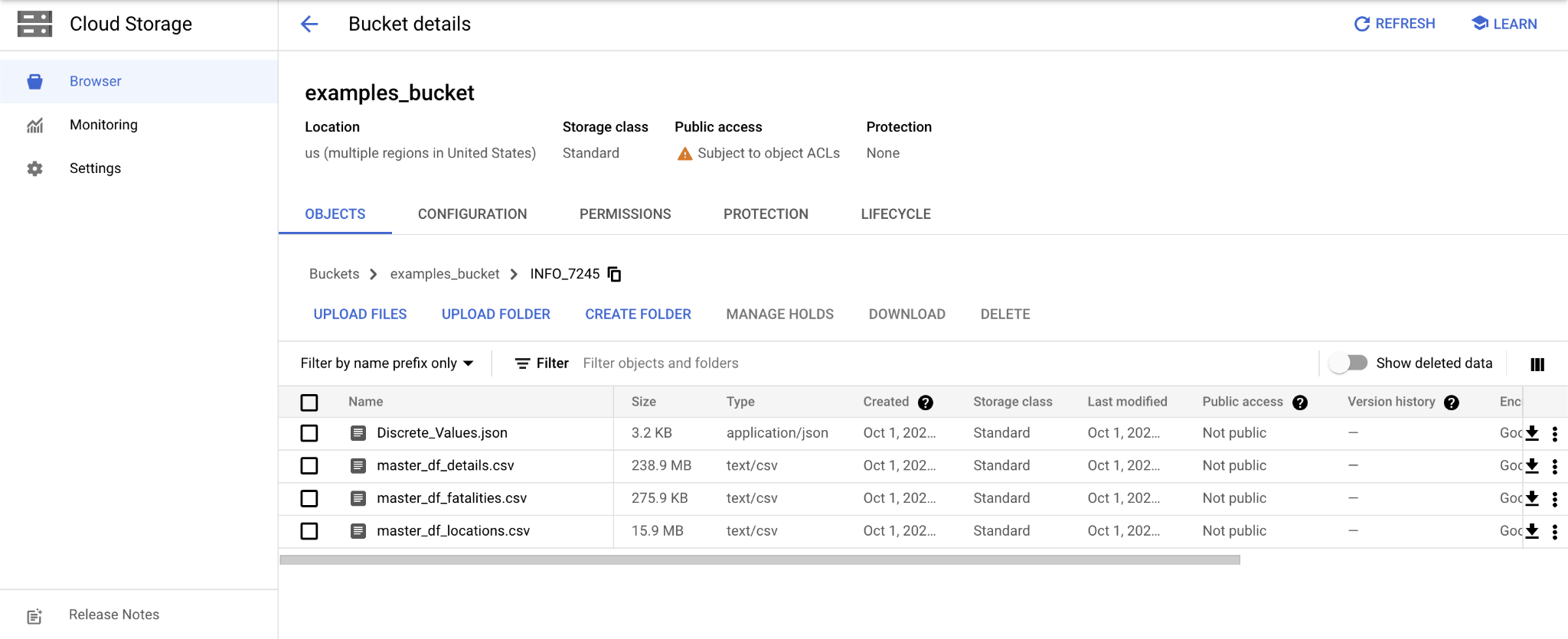
# GCS BUCKET UPLOAD

We then uploaded the files to GCS bucket. Here’s how:



Similarly, we uploaded the 3 merged data files

The bucket now has 4 files. Here’s how it looks on Google console:



# Apache- Airflow

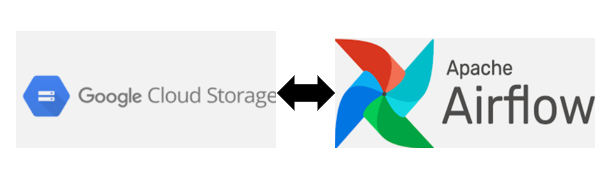
Apache Airflow is an open-source tool to programmatically author, schedule, and monitor workflows. It is one of the most robust platforms used by Data Engineers for orchestrating workflows or pipelines. You can easily visualize your data pipelines’ dependencies, progress, logs, code, trigger tasks, and success status.

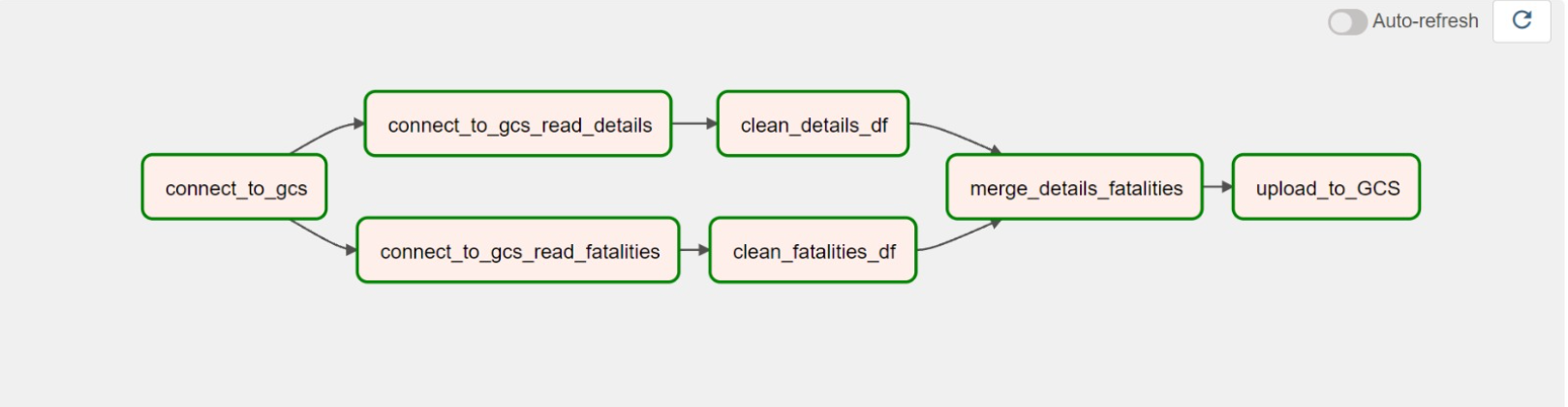
With Airflow, users can author workflows as Directed Acyclic Graphs (DAGs) of tasks. Airflow’s rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed. It connects with multiple data sources and can send an alert via email or Slack when a task completes or fails. Airflow is distributed, scalable, and flexible, making it well suited to handle the orchestration of complex business logic.

## DAGs

In Airflow, a DAG – or a Directed Acyclic Graph – is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies. A DAG is defined in a Python script, which represents the DAGs structure (tasks and their dependencies) as code.

# Overviews

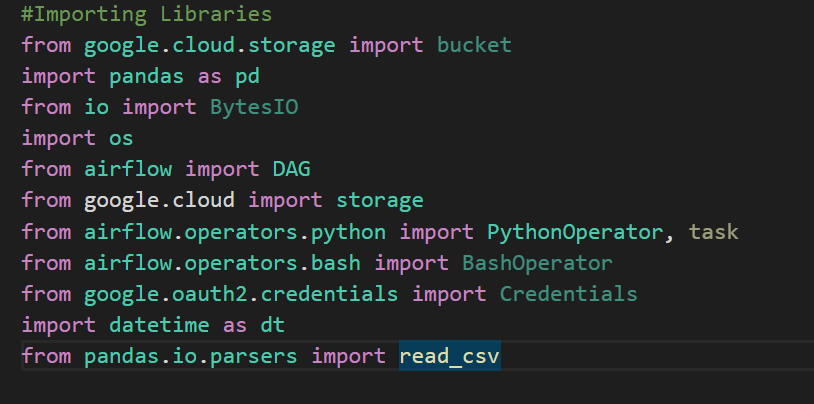




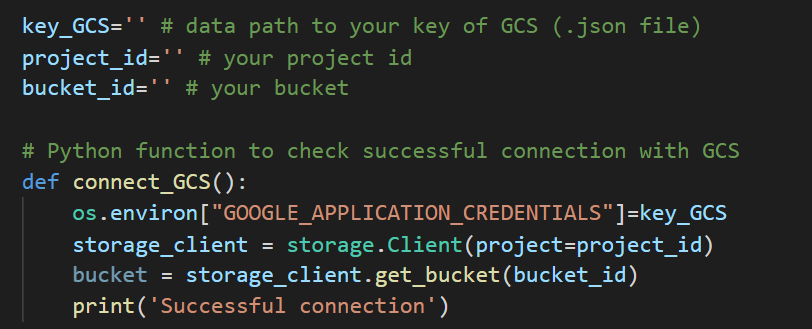
The above is series of DAGs we created using airflow

# Exercise:

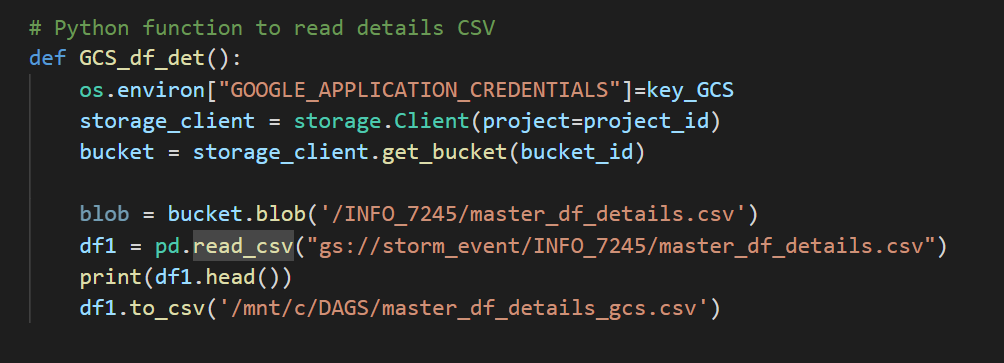
Imported the required dependencies



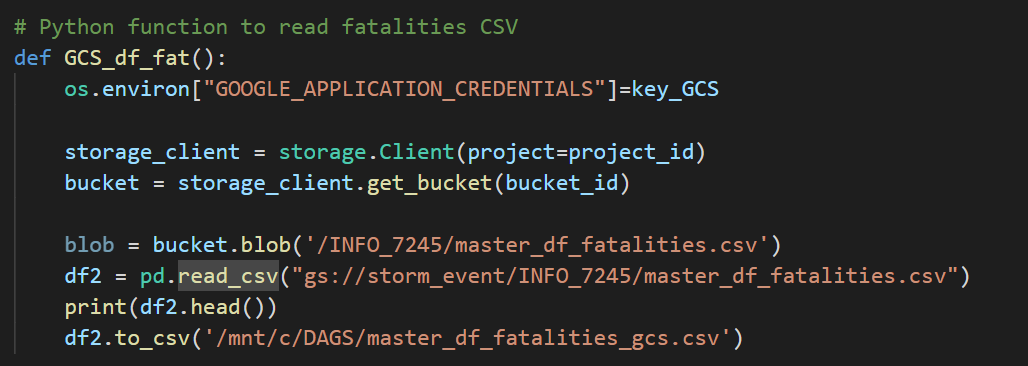
## Connect\_to\_gcs



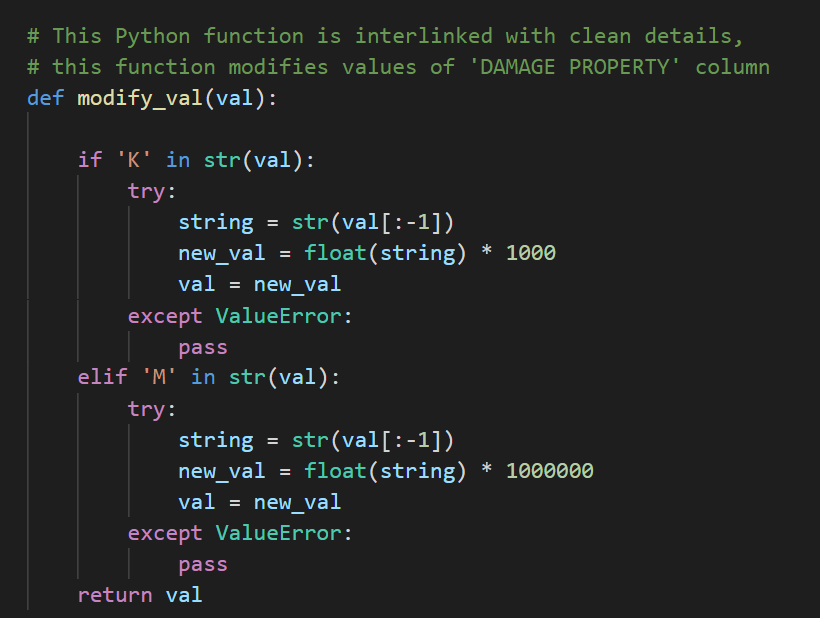
## connect \_to\_gcs\_read\_details



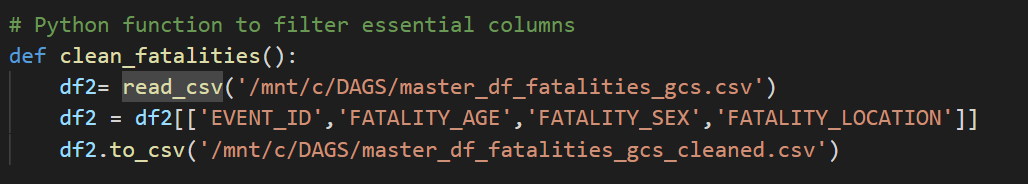
## Connect\_to\_gcs\_read\_fatalities



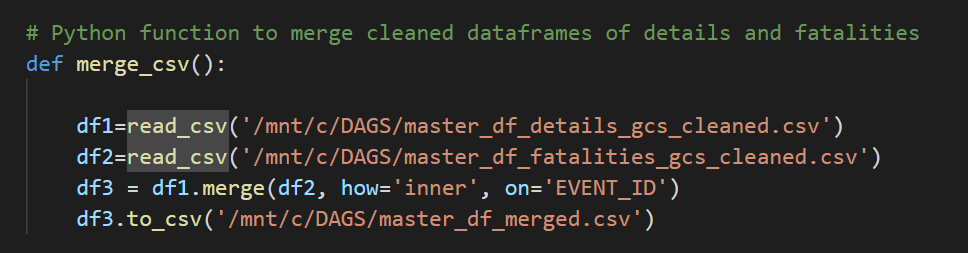
## Clean\_details\_df



## clean\_fatalites\_df



## Merge\_details\_fatalities

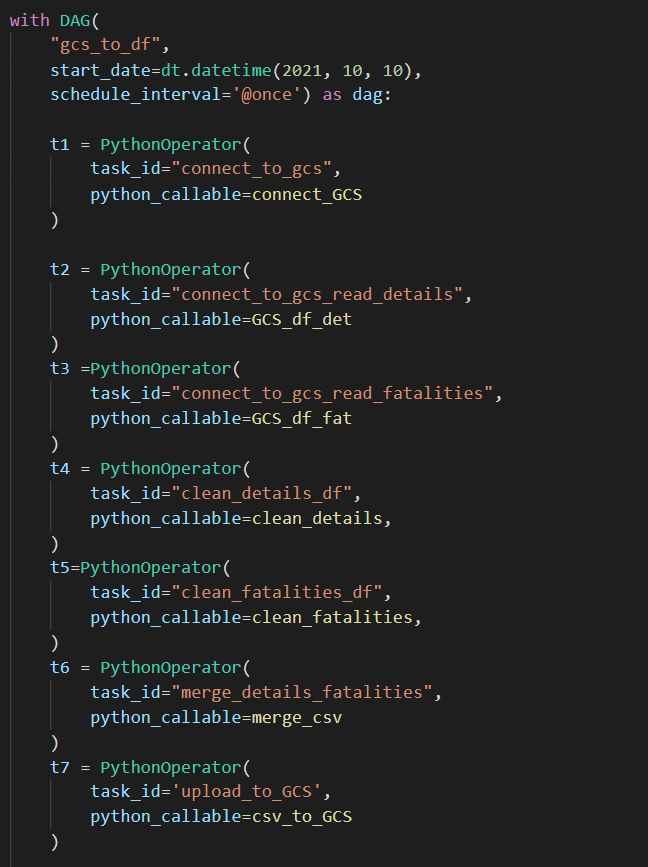


## upload \_to\_gcs

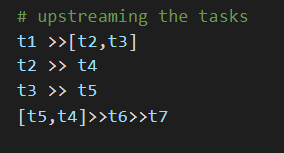


# Pipeline

Building pipeline using tasks in airflow

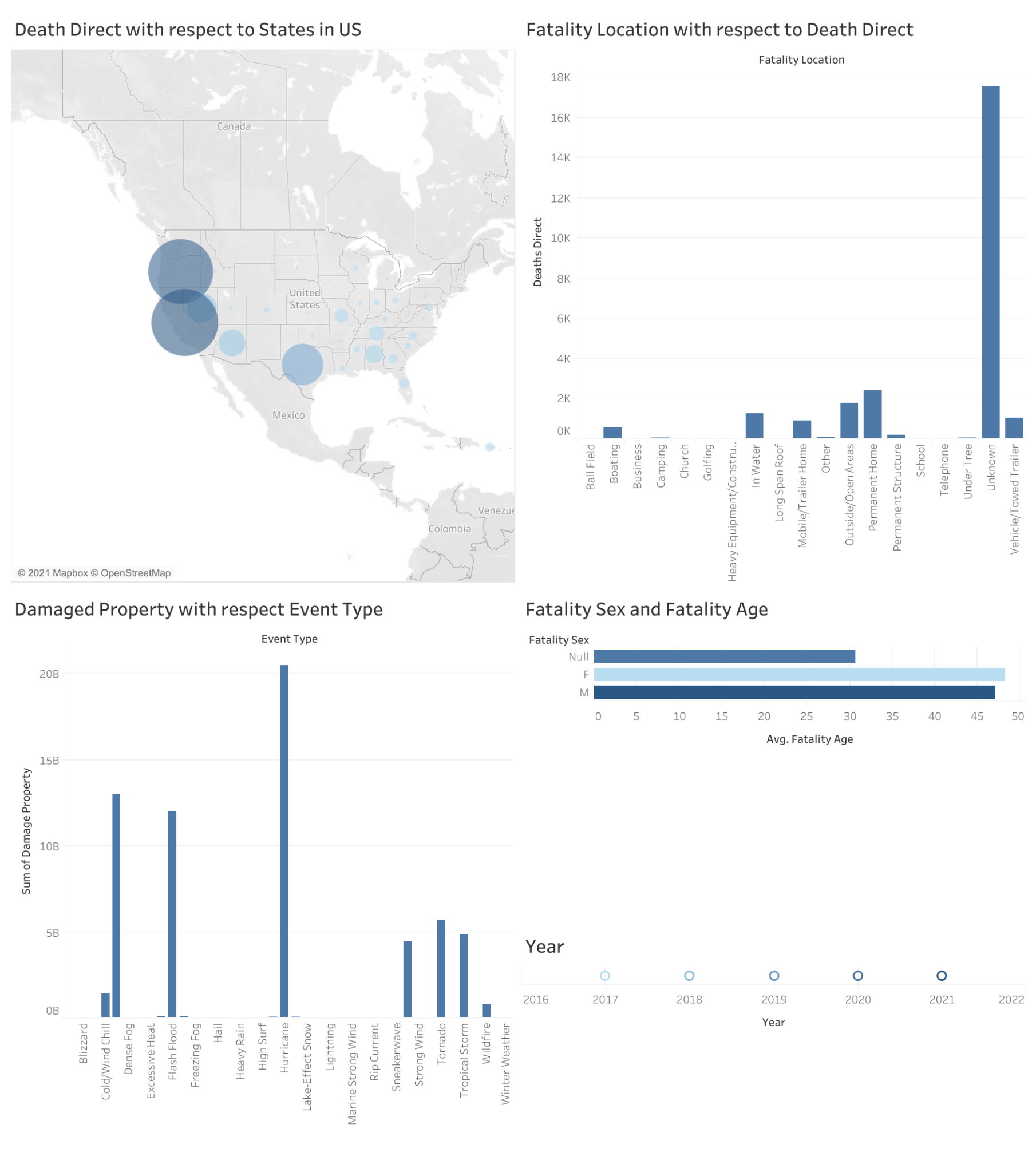


Upstreaming to making sequence of execution



# Visualization

Tableau:<https://public.tableau.com/views/Storm_Event-INFO7245/Dashboard1?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link>



## Use Cases

Where to base teams? Which states/regions have the most market potential?

What events do we want to cover in the insurance?

How age and sex play in factor of calculating insurance rate?

Which places to avoid during specific storm events?