Exploring Streaming Data Using Summary Statistics

Nate Wilson, Pragya Garg

Department of Computer Science, University of San Francisco

May 21, 2019

1 Introduction

The ultimate motivation for this project came from two places, (1) a desire to combine two modern disciplines under the umbrella of data analysis: big data and data visualization, and (2) a dearth of infrastructure that is traditionally necessary in order to do any sort of analysis, let alone visualization, on sufficiently 'big' data.

Big data is becoming ever more important as larger and larger caches of higher and higher resolution data become available. Weather sensor networks that produce thousands of measurements per day at thousands of locations per day are constantly active and deadly important to understand given the global threat of climate change. Financial data including fiat currency, cryptocurrency, and stock trade data, among many other types, exists in super high volume streams coming from a large variety of sources. This economic data is far larger in size and equally important to understand so as to maintain a vital commercial and consequently social ecosystem, globally.

As data becomes larger and more complex, traditional methods of data analysis become more inefficient and ineffective. Interactive data visualization offers a promising alternative in that highly complex, intuition, expertise, and data driven analyses can be made in a recurring loop of observations, analyses, filtering and zooming. Although we did not ultimately implement all of these techniques, we recognize their importance and hope that future project leaders will do so; more details about this will be discussed in Section 3 Future Work.

With the goal in mind of applying modern data visualization tools and techniques to big data, we found an unfilled niche out of necessity: how does one do such analysis and visualization without massive storage and compute infrastructure. For example, consider a park ranger who runs a park with thousands of air and water quality sensors streaming hundreds of measurements per seconds. Assuming a meager 100 bytes per measurement, that park ranger will amass in the range of 100s of terabytes in just one year of data collection. While disk space in this day and age is cheap, and a storage server on that scale may even be affordable ($5,000-10,000) given the right funding sources, the resources and expertise needed to run valuable computations on that scale is not reasonable for such a user.

We were thus motivated to design a system that could ingest an arbitrarily large set of infinite data streams, while only storing a small, constant size data structure, that would still support visual interaction of various statistics at multiple resolutions.

2 Architecture and Design

2.1 Design

We simulate parallel data streams by passing pieces of a weather dataset to a custom streamer. The data gets passed across the network using a fast, custom

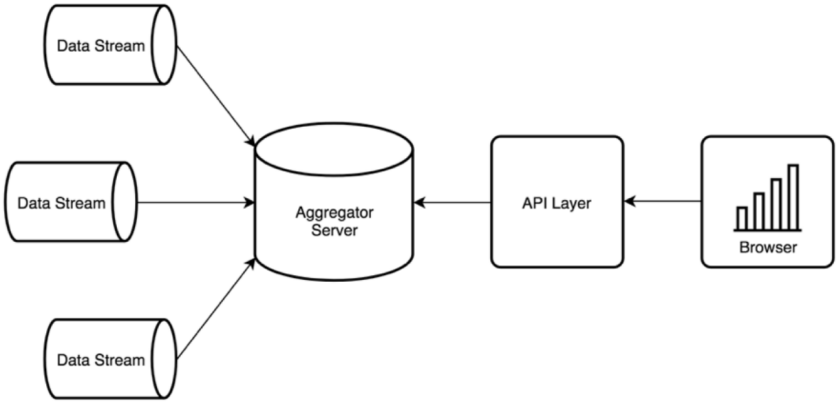

Figure 1: Architectural Design of the System

protocol, designed with the intention of hooking in a network of weather sensors.

The data streams are processed and aggregated by the Aggregator Server, which serves data via RPC to our Flask API server. [4]

Finally, our visualization interface makes REST call to our API layer to get the data needed to render the client-side interface.

2.2 Dataset

We decided to use the National Climatic Data Center (NCDC) dataset that contains measurements every 5 minutes from many locations within the United States. [1] We use the following data fields:

- UTC Date
- UTC Time
- Precipitation
- Solar Radiation
- Longitude
- Latitude
- Humidity
- Air Temperature
- Surface Temperature

2.3 Stream Simulators

We wanted to achieve two separate goals in our data stream simulations, so as to test two separate desired functionalities of our system. Those two functionalities were (1) to be able to handle an infinite, never ending data stream, and (2) to gracefully handle real data with all its natural blemishes including bad data and missing data. Because we did not have simple access to an infinite stream of real data, the simplest way for us to achieve this was by actually building two separate stream simulators, an infinite emitter, and a real data emitter.

The infinite emitter was made using a python generator which created data of various data types out of a set of simple functions according to a monotonically increasing input. We included linear, polynomial, and trigonometric functions in this generator, so as to test how our sample based visualizations would represent a variety of different types of functions.

The real data emitter was created by passing a large CSV of NCDC climate data, and emitting records to our Data Server one record at a time. The real data emitter was helpful in exposing a variety of flaws in our implementation, including how at first we did not deal with missing data; we fixed this by adding logic that first checked for a complete record before updating our data structures. This emitter was also useful because we could validate our statistics another way, according to intuition. For example, we were able to see a variety of commonly known correlations between features in our dataset, like a high negative correlation between Solar Radiation and Precipitation.

2.4 Data Server

The backend data server, referred to in our code as the Aggregator Server, serves as the main computational part of our system. Its job is to ingest the data, update the real time statistics and then throw the data away. The main exposed function call in this class is an update function that retrieve a data record, categorizes it into different resolution bins (currently by nth day of year and location), perform run-time computation algorithm to update aggregated statistics. Some of the important features of this server are as follows:

2.4.1 Parallel Stream Handling

The data server can process multiple streams in a parallel fashion. We wanted to deal with distributed streaming data simultaneously. This helps the system to access and process distributed data effectively.

2.4.2 Served Statistics

We are using 5 features from the dataset: Air Temperature, Precipitation, Solar Radiation, Surface Temperature and Relative Humidity. The server computes the statistics, minimum, maximum, mean, variance and correlation for each of these features present in the dataset. [3]

2.4.3 Stream Computation Algorithms Implemented

As our goal was to compute statistics on the data while maintaining just a constant size data structure, we were required to compute these values as effectively as possible. For the statistics like - minimum and maximum - there was absolutely no other storage required and everytime a new data record came, it was compared with the statistics already saved and would be updated when required whereas for statistics like - mean and variance - we had to save few other things like number of records, old mean value and old variance value to compute the new stats results. Also, for collecting correlation values among different features we saved and updated covariance matrix in the system.

2.5 Frontend

We designed a dashboard style interface to give fast, high level summary statistics of our streaming dataset. We use Altair, a python wrapper for Vega, a modern declarative data visualization language. [6] [5]

2.5.1 API Layer

For better understanding of the statistics results, we wanted to support a variety of visualization to the users. But because our data server just provides us with the discrete data and not visualization, we decided to make a REST API for serving our data. There could be a lot of different possible apps that could be made but for this project we just focussed on developing a web application to provide all the visualization required.

2.5.2 Visual Frontend

We wanted to provide visualization understanding to the users which was both simple to understand and also depict information in the most effective way. Since we wanted to show discrete comparisons of a statistic feature for a particular feature in the dataset, a simple bar chart was our way to go. And because we wanted to show the real-time analysis to users, all the bar charts were equipped with progressive updates.

Our dashboard is a composition of 3 separate web pages, all interconnected with each other in order to ease navigation. Here is what each web page provides:

2.5.3 Progressive Updates

This is the index page of our visualization and shows 15 bar charts - 3 statistics (minimum, maximum, mean) * 5 features (Air Temperature, Precipitation, Solar Radiation, Surface Temperature and Relative Humidity) - updating progressively.

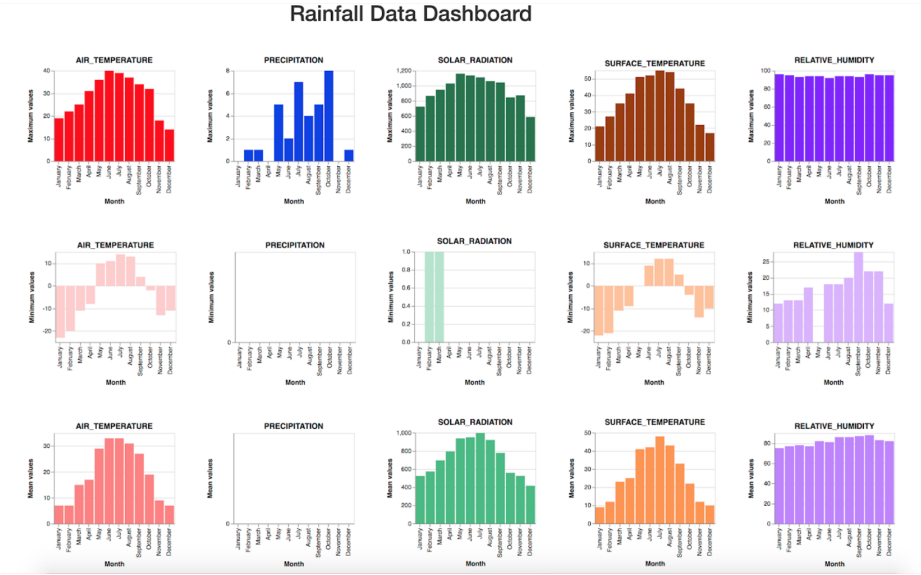

Figure 2: Five feature, three statistic, high level dashboard.

2.5.4 Correlation Matrix

This page presents a progressively updating heatMap that depicts the correlation among the various features in our dataset. The change in color intensities represents how features are related to each other.

2.5.5 Interactive Page

In this page, we tried to provide the users with the specific query selection in order to see the results in different resolutions. This helps the users to really dig deep down and get their queries answered. For this purpose, we provide 3 dropdowns and finally generate the result as requested.

The 3 dropdowns are as follows:

Resolution

- Monthly
- Daily

Features

- Air Temperature
- Precipitation
- Solar Radiation

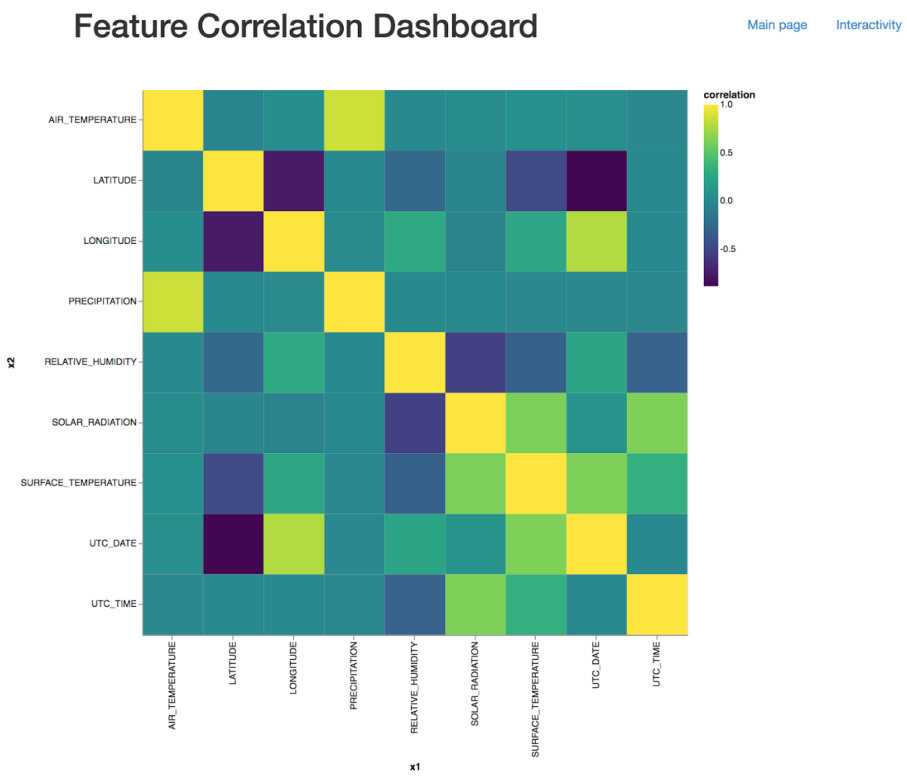

Figure 3: Correlation matrix for all the features.

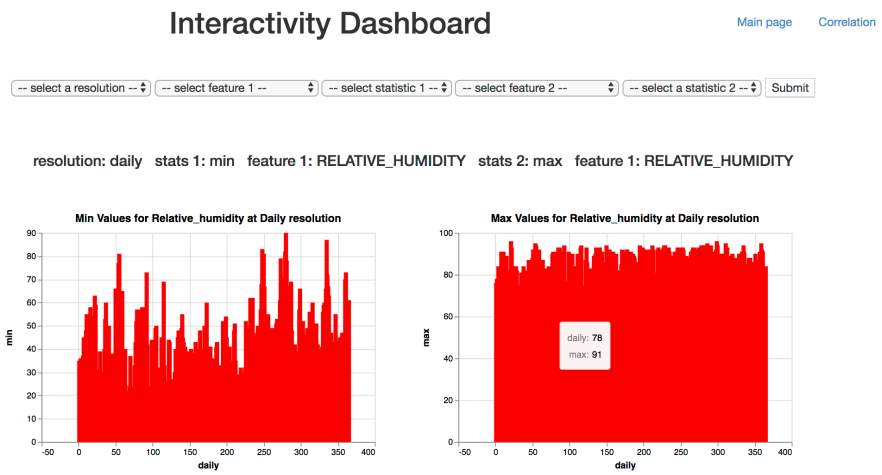

Figure 4: Interactive bar chart showing comparison of minimum and maximum relative humidity at daily resolution.

- Surface Temperature
- Relative Humidity

Statistics

- Minimum
- Maximum
- Mean

3 Future Work

We had a large variety of unrealized plans, which if implemented would serve to take this distributed application to a higher plane of efficacy. We will present them here as a list, with no particular attention to sequence.

A highly important piece of work that we wanted done was to actually hook our system into a real, live sensor network, to see if that would present any new failure modes. This would be relatively simple, as in theory we have already validated that we can handle both infinite and real data, and all it would require is to wrangle the incoming sensor stream into the data servers expected type, a one way byte stream over a raw socket. This would serve as a great next step to take on not only because of the further validation it would provide, but more importantly because of the milestone it would achieve, taking this system from

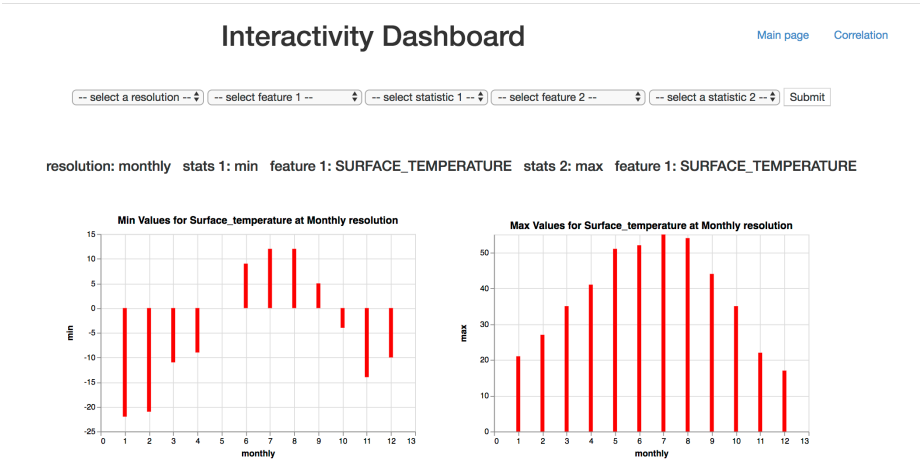

Figure 5: Interactive bar chart showing comparison of minimum and maximum surface temperature at monthly resolution.

a proof of concept, showing how it could do and help something, to an actual product, doing a job we set out to achieve, informing a user we set out to help.

Another valuable feature would be a websocket interface. While we currently have a reasonably justified raw bytes over socket interface, web sockets are the de-facto modern standard for high speed, one way data streams. Having this interface would open up a world of opportunities for big data analysis given the large variety of WebSocket API's that exist out in the wild.

One major concession we had to make in this project was deciding to using one specific dataset. Clearly though, this architecture would be much more valuable if it could handle a more general set of incoming data. For example, arbitrary growing column sets are typical in streaming datasets as sensor networks are upgraded. Logic that could sense such additions and dynamically update the internal data structures to accommodate such would be highly beneficial. Apart from changing and growing datasets, a more general input interface would more simply allow this system to be used with a variety of different datasets.

Another set of concessions we had to make was related to missing and bad data. Right now, our system has very simple, hard coded logic to deal with specific encodings of missing data as defined by our dataset. As one example, in our chosen NCDC Dataset, missing values are coded with the value -9999. We account for this by simply checking for that value, and throwing out the record if it is equal to that, but this is a non-general solution. A more general approach might implement a more sophisticated outlier detection system sensitive to distance from the as yet observed distribution or even more refined machine learning based anomaly detection scheme.

The final, and least important, though perhaps most interesting set addi-

tions to make include increasing the set of stream computations implemented and served. One obvious direction to go down at first is to implement linear regression on a data stream. Currently we have the correlation between different features in the stream, though a correlation, along with a slope and an intercept could be far more interesting. In addition to a simple two parameter fit of linear model, one could imagine the value in simultaneously computing and keeping a 3, and 4 parameter fit model, with the ultimate goal of overlaying those three lines/curves overtop of each other, which when visualized could provide some sort of aggregate understanding.

Although the above explained polynomial modeling of a stream could be highly interesting and informative, all sorts of interesting datasets worth visualizing have equally interesting non-linear relationships that can only be captured by more sophisticated machine learning models. Online machine learning is a growing field with lots of opportunities for new and useful insight. [2]

References

[1] NOAA national climatic data center. `https://www.ncdc.noaa.gov/cdo-web/search?datasetid=GHCND`, 2018.

[2] Online machine learning — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Online_machine_learning`, 2019.

[3] John D Cook. Accurately computing running variance. `https://www.johndcook.com/blog/standard_deviation`.

[4] Armin Ronacher. Flask (a python microframework). `http://flask.pocoo.org`, 2018.

[5] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2015.

[6] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive statistical visualizations for python. *The Journal of Open Source Software*, 3:1057, 2018.