VISUALIZING FLIGHT TEST DATA INTERACTIVELY WITH OPEN SOURCE TOOLS

Luke Starnes
Chief Engineer of Electronic Systems Laboratory
Georgia Tech Research Institute
Telephone: 404-407-6779 E-mail: luke.stanes@gtri.gatech.edu

1. BIOGRAPHY

Luke Starnes received a BS degree in Electrical Engineering (2002) and MS degrees in International Politics (2008) and Business (2013) from Georgia Tech. He works for the Georgia Tech Research Institute (GTRI) where he is the Chief Engineer of the Electronic Systems Laboratory. During his 15+ year career, Luke has been involved in defense technologies where he has primarily focused on avionics and mission system integration for fighter aircraft. The test and evaluation of these programs generates a large quantity of data. The team regularly processes terabytes of flight test data to evaluate the performance of their embedded avionics software. The employed data analysis process has been migrated to a Python-based tool chain. While their data exploration and analysis needs are atypical, they strive to leverage tools and best practices employed by the community. Luke serves as the co-organizer of the Atlanta Jupyter User Group.

2. INTRODUCTION

Much of the flight test community relies on proprietary tools for data analysis. These tools can be expensive and create a “vendor lock” problem. Configuring a tool or tool chain for a specific use case will always require a fair amount of up-front work. This can include adding the understanding of file and message formats, defining the relationships and levels of aggregation between the different data products, and creating visualizations specific to a project’s goals. If this configuration work is performed within a “walled garden” then the team can become locked in as the cost to exit the specific tool or tool chain becomes higher the longer the team uses the tooling. This is because, over time, more and more project specific information is defined within the walled-off space. Extrication can prove difficult.

The good news is that there is an alternative. Robust open source tools exist to store, transform, and visualize flight test data. This paper makes the case that open source tools are a superior choice for today’s flight test analysis problems. These tools utilize open interfaces, which has led to widespread compatibility across scores of tools and ensures seamless migration between tools (no “vendor lock”). The openness and the compatibility it facilitates has created a community of interoperable tools. This allows for the flexibility and agility to tailor tooling to a specific project’s needs rather than being forced to use a singular proprietary tool whether it suits the job at hand or not because that is where all the previous work has been done.

This paper includes a case study where Open Flight Data is stored, transformed, and visualized using open source tools. The data is publicly available flight data, primarily from Automatic Dependent Surveillance-Broadcast (ADS-B). This case study utilized a specific suite of open source tools although the arguments for their use are valid for other open source alternatives. This suite includes the Hierarchical Data Format 5 (HDF5), Pandas, Luigi, Jupyter, Bokeh, and Datashader. This suite of tools either are all Python based or work well with Python. Python is a popular scripting language used in scientific computing and is currently the fastest growing major programming language.[1] These tools are discussed in detail in the following sections.

The code that generated the analysis discussed in this paper can be found here: github.com/slstarnes/sfte2018-adsb

1.1. ADS-B Background

Historically air traffic control (ATC) has relied on radar to produce situational awareness about the commercial airspace. ATC radars exist around the world to scan the sky and locate, track aircraft. The figure below indicates the worldwide ATC radar coverage as of As of December 31, 2017.

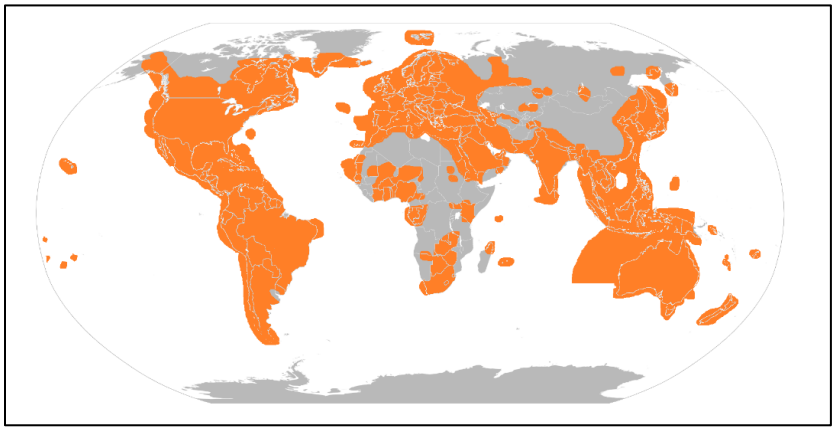

Figure 1: Map of the Active Radar Coverage of Air Traffic Control Worldwide as of 31 December 2017[2]

These ATC radars benefited from the military's Identification Friend or Foe (IFF) system as this provided a way for aircraft to provide identifying information to the radar.[3] The ATC radar-based system is antiquated, inefficient, and expensive. With the ubiquity of Global Positioning System (GPS), today's aircraft calculate their own positions. Rather than employing a network of radars to locate and track these aircraft, it would be prudent to have aircraft self-report their position. This is the concept behind the Automatic Dependent Surveillance-Broadcast system. The ADS-B system requires aircraft be outfitted with an ADS-B transmitter and that the transmitter be configured to continuously broadcast the aircraft's position, velocity vector, and identification along with additional information at a rate of at least once per second.

ADS-B is required for operation in the United States beginning January 1, 2020. At that point, all aircraft operating above 10,000', around airports, or off the Gulf of Mexico will be required to transmit their position per the ADS-B regulations. The primary ADS-B solution operates at 1090 MHz and is referred to as 1090ES. This is required above 18'000 feet. Below 18,000 feet an alternative solution, the Universal Access Transceiver (UAT) is sufficient. The UAT operates at 978 MHz. The 1090ES is required outside of the United States. The European mandate for ADS-B goes into effect on January 1, 2019.[4] Additional countries and regions will

be instituting mandates as well.[5] As of the beginning of 2018, 48,500 US-registered aircraft (including 1,500 US-based airliners) have been outfitted with ADS-B transmission capability.[6,7]

In addition to the transmission capability resident in the aircraft, the ADS-B system includes a network of ground stations. The figure below identifies the operational ADS-B ground stations in the US as of October 17, 2017.

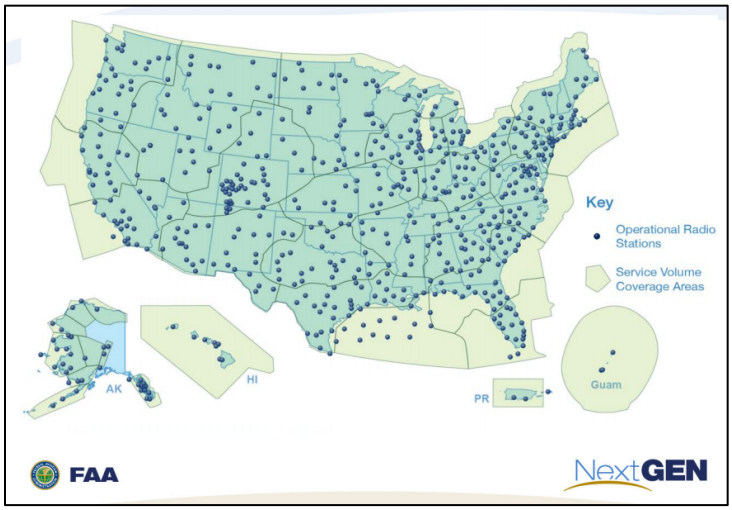

Figure 2: USA Operational ADS-B Radio Stations[8]

The air traffic picture is created based on the fusion of track reports from multiple ground stations. Given the physical limitations of radio frequency (RF) transmission distance, ground stations are required roughly every 100 NM.[9]

1.1. Open ADS-B Data Case Study Background

ADS-B transmissions are unencrypted and can be received and processed by any device operating in the RF band and within vicinity of the transmitting aircraft. This has led to a plethora of websites, which monitor this data and create interesting worldwide real-time visualizations. These websites, examples include flightradar24.com, flightaware.com, planefinder.net, and adsbexchange.com, rely on individual contributors to setup their own local ADS-B receiver and feed their data into the site's composite feed. Consumer-grade ADS-B receivers are inexpensive and easy to setup. An example visualization from one of these flight-tracking sites is shown below.

Figure 3: Example ADS-B Data Visualization from planefinder.net

The case study outlined in Section 4 discusses a data analysis task centered on publicly available data provided by ADSB-B Exchange (adsbexchange.com). ADSB-B Exchange provides access to their entire data catalog beginning on June 9, 2016. The data is made available via compressed JSON files. Each day is contained within a single zip file with a median size of 7.5 GB. Within each day's zip file are 1,440 JSON file, one file per minute of the day. This data was downloaded, stored, extracted, and then transformed to HDFS5. Once in HDFS5 the data was easily and efficiently accessible for follow-on analysis. Details are provided in Section 4.

3. OPEN SOURCE TOOLING

Open source tools provide the necessary capabilities, performance, and flexibility to meet the needs of flight test data analysis. There are multitudes of open source tools that are focused on different parts of the analytics chain. This paper will discuss a specific selection of tools that provide a robust capability for the flight test data analysis use case. In conjunction with each other, these tools form a highly capable tool chain.

3.1. Storage with HDF5

Hierarchical Data Format 5 (HDF5) is a file format for storing data.[10] It was developed by the National Center for Supercomputing Applications as a solution to managing large quantities of data in scientific research.[11] It was designed for efficient storage and rapid access. The data can be compressed and is indexed such that it can be queried quickly from disk, which is critical when storing large data sets. It also has the ability to store metadata alongside the data. This can be very useful in archiving data as you can search for specific flight test data based on metadata parameters (e.g. date, platform, system under test, etc.) as opposed to relying on a folder structure.

HDF5 is a cross-platform file format and can be written and accessed in many different ways. APIs exist in C, C++, Fortran, Java, and Python. For the purposes of this paper, Pandas (built on Python) was used as the interface to the HDF5 files as it seamlessly integrates with the rest of the data extract, load, and transform (ELT) stack.[12] Pandas will be discussed in the next section.

3.1.1. HDF5 Examples

- HDF5 provides the ability to select a subset of data from a file. This is especially useful when the file is too large to be read into working memory. The lines below would read from the *data* table meeting the criteria. This is using the Pandas API, so the returned data is in the format of a Pandas DataFrame.

```
h5.select('data', start=12000, stop=14000)
h5.select('data', start=12000, stop=14000, columns=['Lat', 'Long'])
```

- The lines below would select all rows from the *data* table that meet a certain conditional.

```
h5.select('data', where='To==KLAX & From==ZSPD')
h5.select('data', where='Lat<32.1 & Lat>31.9 & Long<-81.0 & Long>-81.3')
```

- Metadata can easily be added using the h5py package. This data can be quickly read in the future to assist with finding the right file.

```
h5 = h5py.File(path, 'r+')
h5.attrs['Date'] = '6/18/2018'
h5.attrs['Rows] = h5['data']['table'].len()
```

3.1.2. Storage Alternatives

Data can be stored as Comma Separated Values (CSV) files. CSVs are extremely flexible as any software package can read them, but they are inefficient, and they lack the ability to natively associate metadata with specific files, which makes organization difficult. Data can also be stored in a relational database (e.g. MySQL, PostgreSQL, and SQLite).

3.2. Data Transformation with Pandas

Pandas is a Python library that provides a robust data capability set for accessing, transforming, and analyzing tabular data.[13] Pandas was initially created to meet a quantitative financial analysis need. It has grown into a key component of any Python-based data analysis workflow. Pandas centers on the DataFrame, which is an in-memory tabular data structure, which can be manipulated and transformed in a myriad of ways. Pandas supports reading and writing across many different file formats, including CSV, JSON, Excel, and HDF5. It is estimated that Pandas has between 5 and 10 million users.[14]

3.2.1. Pandas Examples

Once data has been read into memory as a DataFrame (from HDF5 or another source), Pandas provides a rich API for interacting with and transforming the data.

- The lines below create a table showing the count of aircraft operated by a subset of airlines by aircraft type. In this example, *df* is a DataFrame containing a slice of ADS-B data.

```
airlines_filter = df['Op'].isin(('Southwest Airlines', 'American Airlines',
                                 'Delta Air Lines', 'United Airlines'))
table = df[airlines_filter].groupby(('Op','Type')).count().Icao.unstack().T[:10]
table['Total'] = table.sum(skipna=True, axis=1).map(int)
table.fillna('').sort_values('Total', ascending=False)
```

3.2.2. Data Transformation Alternatives

Within the Python ecosystem, Pandas stands alone with its level of capabilities. Numpy, another Python library is worth mentioning as it provides extensive transformation capability. Pandas is actually built on top of Numpy, so it inherits many of the capabilities while extending

them and adding a wealth of integrations to ease the data processing workflow. Outside of Python, R provides many similar capabilities in the realm of data manipulation and transformation. In addition, Excel can be used to do similar functions but much less efficiently or portable. Having data manipulation as code as opposed to Excel formulas facilitates a greater level of sharing across projects.

3.3. Pipeline with Luigi

It is typical for a data analysis project to consist of multiple, inter-dependent steps. There is a need to acquire munge, manipulate, merge, and aggregate data and each step in the process depends on some number of preceding steps. Typically, these actions are executed in batches when new data becomes available. Luigi is a Python-based pipelining tool developed by Spotify.[15] This open source tool allows for the definition of a pipeline process as a set of tasks and dependencies. Luigi manages the dependencies between tasks and determines when follow-on tasks can be initiated. This capability is well suited for the recursive nature of a standard data pipeline. The pipeline can be configured to auto-initiate based on certain conditions such as the presence of new data.

3.3.1. Pipeline Alternatives

There a variety of comparable, open source solutions. Two notable alternatives are Airflow, developed by Airbnb, and Pinball, developed by Pinterest. Luigi is the most popular of the three as a measure of PyPI downloads.[16]

3.4. Data Notebooks with Jupyter

Jupyter Notebooks are a great way to perform analysis and explore data. An open-source web application, Jupyter Notebooks allow “you to create and share documents that contain live code, equations, visualizations, and narrative text.”[17] They provide an interactive method for writing software. Building on the Read-Eval-Print Loop (REPL) concept employed by an interactive shell, Jupyter puts this capability in the browser and extends it by adding code completion and allowing for interspersing the code with Markdown for formatted text and visualizations. Jupyter Notebooks can be exported to PDF and HTML which allows for an entire report to be generated using Jupyter. Jupyter is based on IPython and IPython Notebooks. IPython was developed by Fernando Perez in 2011. He based the concept off of his experience with Maple and Mathmatica. The IPython Notebook was rebranded as Jupyter since there is nothing language specific about the notebook concept. In fact, Jupyter now supports more than 60 different programming languages.[18] The three most popular are Python, R, and Julia which is where the name came from – Ju(lia) + Py(thon) + (e)R. Jupyter continues to grow in popularity as shown in the plot below. The plot shows the number of notebooks stored on GitHub, currently over 2.5 Million.

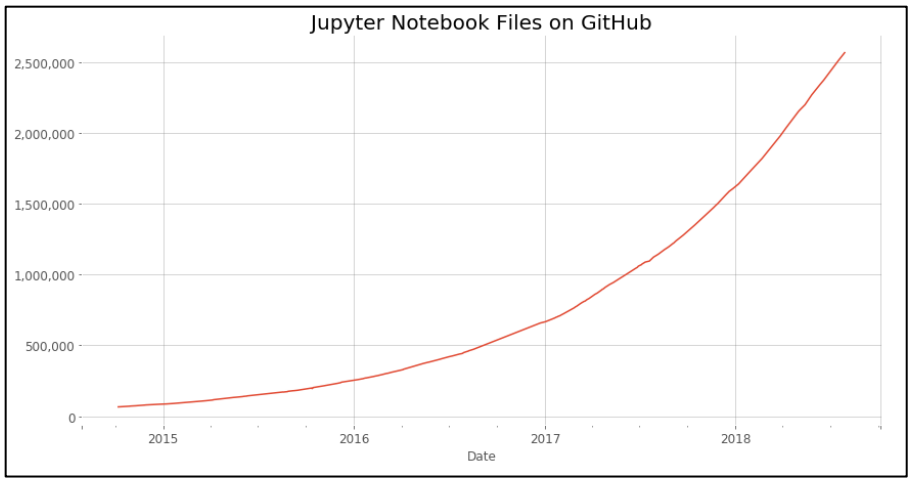

Figure 4: Jupyter Notebook Files on GitHub

3.4.1. Data Notebook Alternatives

The Apache Zeppelin Notebook is an open source offering similar to Jupyter, but is more geared to the Hadoop use case.[19] Zeppelin is not as extensible as Jupyter and does not have the same level of integration with the Python data analytics stack, but it is an impressive product and will continue to mature. The Apache Spark Notebook is another open source computational notebook with a focus on Big Data.[20] It is limited to Scala, a programming language similar to Java.

3.5. Data Visualization with Bokeh

The typical flight test analysis task is assessing a multitude of interacting variables. Through data exploration, the analyst can begin to understand the relationships between the various parameters of interest. Data exploration is the iterative characterization of data. Data visualization is a key component of data exploration. Through visualization the analyst is able gain a deeper understanding than what statistics alone can provide. This is well illustrated by the Anscombe's quartet, a dataset consisting of four sets of points which are all statistically similar, but visually varied.[21] The figure below shows the Anscombe's quartet.

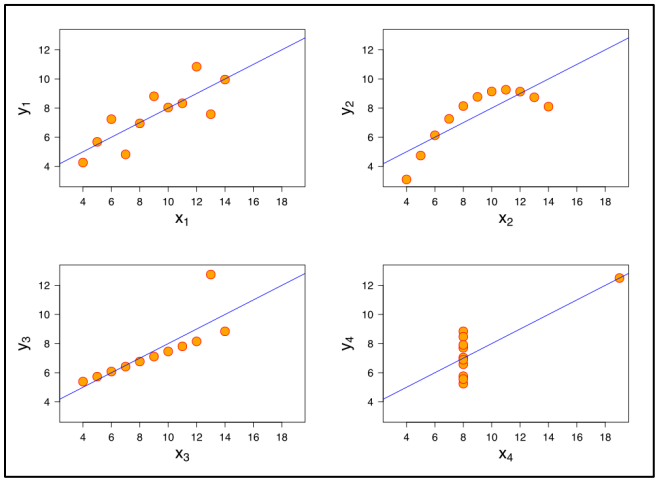

Figure 5: Anscombe's quartet[22]

An interactive visualization tool is highly recommended when exploring data. Static plots do not allow for the rapid poking and prodding of the data necessary. Through the process of exploration, the analyst will quickly think of questions to "ask" the data. If the queue builds up due to an inability to answer the questions faster than new questions arise, there is a risk of overlooking questions and forgoing valuable investigative paths.

Bokeh is an interactive visualization tool, built in JavaScript for execution in a web browser.[23] Bokeh is well integrated with Pandas and Jupyter. Plotting data from a Pandas DataFrame inside a Jupyter Notebook is seamless. This powerful tool allows the user to create robust, dynamic plots with extensive customization.

3.5.1. Data Visualization Alternatives

Matplotlib is the most popular plotting tool in the Python ecosystem, but it is focused on producing static plots.[24] Plotly is a browser-based interactive library that is comparable to Bokeh although it lacks the level of integration with Jupyter. Both tools are highly capable and selecting which one to use for a specific project is primarily driven by personal preference.

3.6. Massive Data Visualization with Datashader

As valuable as the browser-based visualization tools are, there is a drawback which can be problematic when analyzing flight test data. These tools suffer from performance issues as the number of points increases. They attempt to render all of the points in a defined plot area, regardless of how many there are. This can cause sluggishness and unresponsiveness in the browser. Datashader has solved this problem.[25]

Datashader facilitates the plotting of large datasets by handling issues of overplotting without the drawbacks of standard mitigation techniques. Datashader creates a 2-dimentional array equal in size to the desired resolution of the output. Each element in the array equates to a pixel in the output image Next, data points are aggregated into the appropriate element in the array. Finally, a transfer function is applied to transform the points within each element to determine the pixel color to represent those points.[26] Rather than plotting points, Datashader plots pixels based on this algorithmic approach. Not only does this method produce an information-rich visualization, it is quite fast.[27]

Datashader is also capable to being used interactively inside of Bokeh. As the user moves around within the data, Bokeh has Datashader generate the appropriate image which is then rendered inside the Bokeh frame.

3.6.1. Massive Data Visualization Alternatives

The author is unaware of any alternatives to Datashader. Without Datashader, the analyst is left employing sub-optimal approaches to plotting large data sets such as using transparency, color, and point size to adequately differentiate amongst overlapping data.

4. OPEN ADS-B DATA CASE STUDY

As stated in Section 1.1, ADSB-B Exchange provides access to their entire data catalog as compressed JSON files accessible via an API. Each day is contained within a single zip file and is comprised of 1,440 JSON file, one file per minute of the day. This data was downloaded, stored, extracted, and then transformed to HDFS5. This process was managed by Luigi. Luigi was configured to automatically execute the extraction and transformation process based on the presence of raw data for a specific day without the associated HDF5 file.

The compressed JSON files average 5.7 GB with a median size of 7.5 GB. The converted HDF5 files average 8.2 GB and contain roughly 30,000,000 rows. Using one representative day as an example, a day contains roughly 70,000 unique aircraft (based on the ICAO identifier) across 500 manufactures, registered in 190 unique countries.

Using Bokeh, it is possible to quickly explore the data through its feature-rich plotting framework. Below is one of many possible examples showing flight paths on a map background with a customizable tool tip. The color of the flight path indicates the ground speed (in knots) of the aircraft. Bokeh supports zooming and panning of the image which is critical to effective data exploration.

Figure 6: Bokeh Example – Flight Paths Colored by Ground Speed

Another Bokeh example is shown below. In this example, the aircraft ground track is plotted against the ground track programmed in the Flight Management System (FMS) or autopilot, grouped by aircraft manufacturer.

Figure 7: Bokeh Example – Aircraft Ground Track vs Programmed Track by Manufacturer

Bokeh is an exceptional visualization tool, but it is non-performant when the dataset is large. Datashader is built to solve the problem of visualizing large datasets. For illustrative purposes, 520,000,0000 points, across 16 days, were plotted using Datashader. The density of the plot is shown in bright yellow and clearly shows the high traffic areas and the major airports. The top plot is for the entire globe and one below is the continental Unite States.

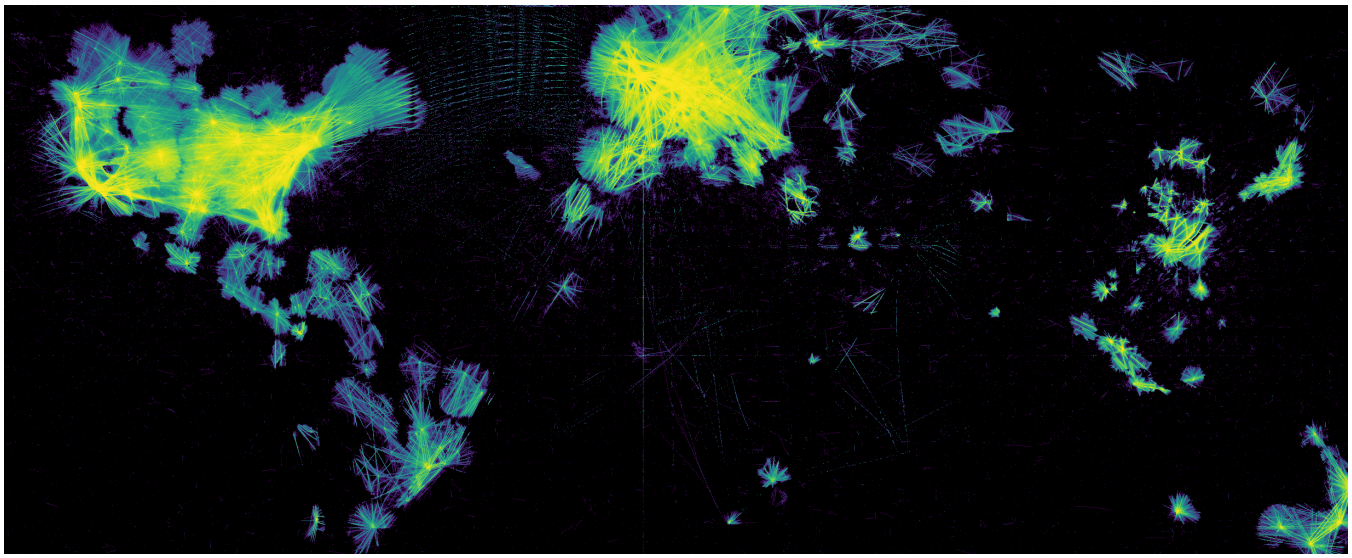

Figure 8: Datashader Example – 16 Days of Reports

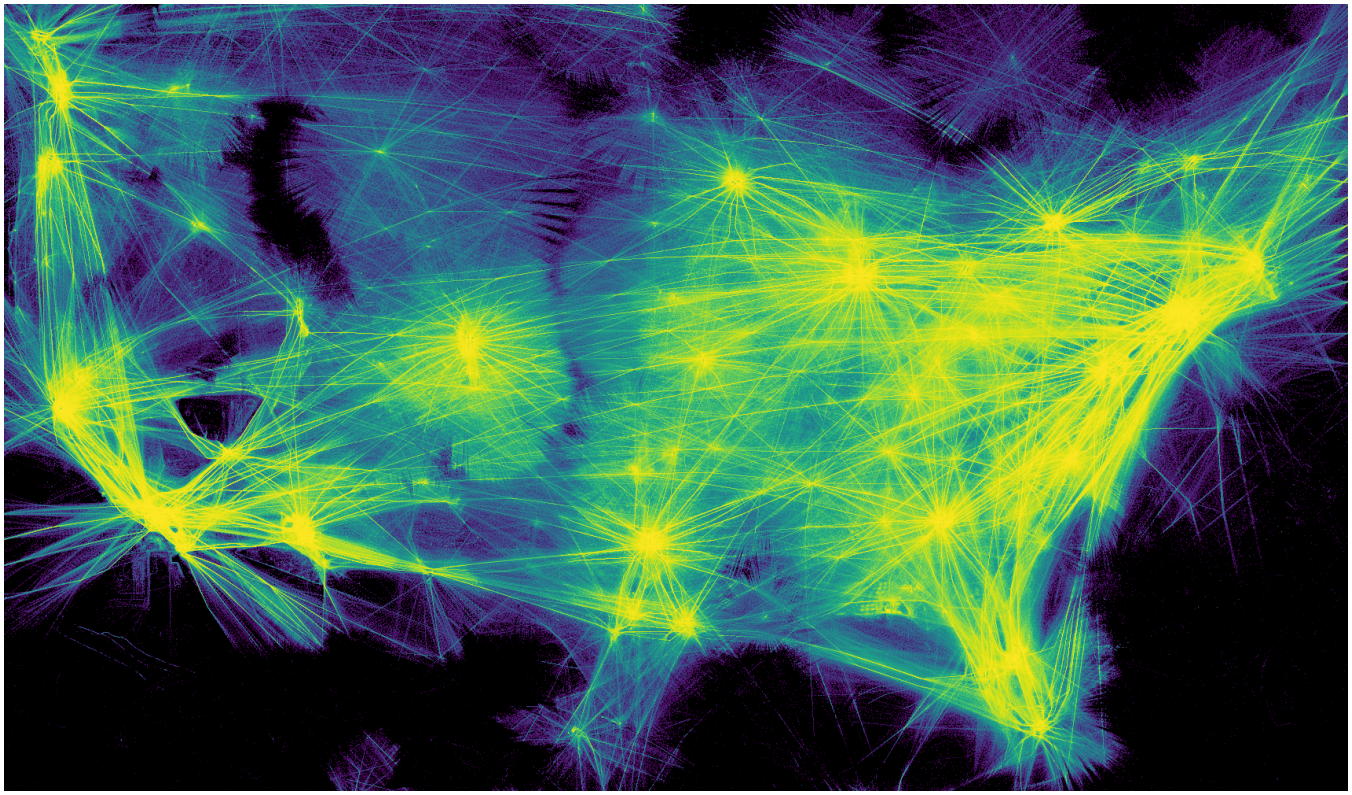

Figure 9: Datashader Example – 16 Days of Reports, Zoomed to United States

These images quickly show the areas of lacking data, those areas which are not serviced by an ADSB Exchange ground station, and the heavily used air corridors. Extracting this information from the data, without visualization, would be quite difficult.

5. CONCLUSION

This paper has made the case for using open source tools to process and visualize flight test data. The ecosystem of open source data analysis tools offers a robust alternative to proprietary tools. Any team seeking to define their toolset should focus on openness as a way to minimize the risk of becoming locked in to specific vendor solution. This paper specifically focused on a set of open source tools which can meet many of the data analysis need. This toolset includes: HDF5, Pandas, Luigi, Jupyter, Bokeh, and Datashader. Example usage of these tools is discussed through the lens of an analysis of a set of Automatic Dependent Surveillance-Broadcast data from ADSB Exchange.

6. FOOTNOTES

[1] David Robinson, "The Incredible Growth of Python," *The* Stack Overflow Blog, September 6, 2017, https://stackoverflow.blog/2017/09/06/incredible-growth-python.

[2] Fiver, der Hellseher (courtesy of Wikimedia Commons), February 10, 2108, https://commons.wikimedia.org/wiki/File:Map_of_the_worldwide_air_traffic_control_radar_coverag.png.

[3] "Air Traffic Control and Radar," Engineering and Technology History Wiki, last modified July 25, 2018, https://ethw.org/Air_Traffic_Control_and_Radar.

[4] "Air Space," Federal Aviation Administration, last modified April 16, 2018, https://www.faa.gov/nextgen/equipadsb/research/airspace.

[5] "Where is ADS-B Out Required?," Aircraft Owners and Pilots Association, https://www.aopa.org/go-fly/aircraft-and-ownership/ads-b/where-is-ads-b-out-required.

[6] "ADS-B: Bad Data, No Service: The FAA Is Suppressing ADS-B From Some Incorrectly Configured Aircraft, " Aircraft Owners and Pilots Association, March 1, 2018, https://www.aopa.org/news-and-media/all-news/2018/march/pilot/ads-b-bad-data-no-service.

[7] "ADS-B: In the Operation," Federal Aviation Administration, last modified May 24, 2018, https://www.faa.gov/nextgen/how_nextgen_works/new_technology/adsb/in_depth.

[8] "ADS-B ASE Processing", Manual Gonzalez, ASE Workshop, October 17, 2017, https://www.faa.gov/air_traffic/separation_standards/rvsm/documents/ASE/2.6_ADS-B_ASE_Processing.pdf.

[9] This based in part on the map shown in Figure 2 and the requirements discussed in RTCA DO-242A, Minimum Aviation System Performance Standards for ADS-B, Table 3-2(a) which lists a required range between 10 and 120 NM depending on equipage class.

[10] For more information see https://support.hdfgroup.org/HDF5.

[11] The HDF Group: About Us, https://www.hdfgroup.org/about-us.

[12] PyTables is another Python library and it is used by Pandas when interfacing with HDF5 files.

[13] For more information see https://pandas.pydata.org.

[14] "Towards Pandas 1.0," Marc Garcia, PyData London Meetup #47, August 14, 2018. https://youtu.be/hK6o_TDXXN8?t=2m20s.

[15] For more information see https://github.com/spotify/luigi.

[16] Based on PyPI package downloads as obtained using Big Query interface as described in https://packaging.python.org/guides/analyzing-pypi-package-downloads.

[17] For more information see http://jupyter.org. Quote is from front page.

[18] List of Jupyter kernels maintained by Project Jupyter, https://github.com/jupyter/jupyter/wiki/Jupyter-kernels.

[19] For more information see https://zeppelin.apache.org.

[20] For more information see http://spark-notebook.io.

[21] Francis J Anscombe, "Graphs in Statistical Analysis," *The American Statistician* Vol. 27, No. 1 (1973): 17–21.

[22] Schultz (courtesy of Wikimedia Commons), March 26, 2010, https://commons.wikimedia.org/wiki/File:Anscombe%27s_quartet_3.svg. Original data from Anscombe, Francis J. (1973).

[23] For more information see https://bokeh.pydata.org.

[24] 2018 Anaconda State of Data Science Report, June 2018, https://know.anaconda.com/rs/387-XNW-688/images/2018-06-Anaconda-State-of-data-science-report.pdf.

[25] For more information see http://datashader.org.

[26] "Big Data Visulization with Datashader," Dr. James Bednar, August 2016, https://www.cs.wcupa.edu/rburns/DataMining/papers/Whitepaper_Datashader_digital.pdf.

[27] Based on running an example notebook on Binder (https://www.mybinder.org) it took 50 seconds to read the 10.7M rows of data into memory. Once in memory, the aggregation and transformation into a 1500 x 1250 image took 70 ms and 16 ms respectively. This was based on running each action 10 times and taking the mean.