Theses and Dissertations 1. Thesis and Dissertation Collection, all items

2018-12

SIMPLIFYING DATA ANALYSIS FOR SUBJECT MATTER EXPERTS

Vanzant, Timberon C.

Monterey, CA; Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

**SIMPLIFYING DATA ANALYSIS FOR
SUBJECT MATTER EXPERTS**

by

Timberon C. Vanzant

December 2018

Thesis Advisor:	Thomas W. Otani
Second Reader:	Arijit Das

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE SIMPLIFYING DATA ANALYSIS FOR SUBJECT MATTER EXPERTS		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Timberon C. Vanzant			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**	

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

In today's data-intensive world, the power to analyze huge amounts of data is critical to the success of any organization, including the military. Many data analysis tools have been developed in the past decade along with the high-performance machine learning algorithms. At present, many of these tools unfortunately are out of reach of the target audience—subject matter experts—because one must master some of the advanced computer science concepts to use these tools effectively.

This thesis proposes to build a prototype data analysis platform that will hide the underlying complexity of the tools from the subject matter experts. Using the platform, the end users can analyze data through a simple, menu-driven interface. The prototype will be built using the programming language Python and the open-source, distributed data processing engine Apache Spark 2.0. Different components of Spark 2.0 will be studied and evaluated to determine the best approach for building the prototype.

The effectiveness of the prototype will be examined using the ADSB (Automatic Dependent Surveillance - Broadcast) unfiltered flight data. The thesis concludes with the review of the prototype developed for ADSB and the recommendation on possible ways of extending the prototype.

14. SUBJECT TERMS data analysis, machine learning, Spark			**15. NUMBER OF PAGES** 71
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

SIMPLIFYING DATA ANALYSIS FOR SUBJECT MATTER EXPERTS

Timberon C. Vanzant
Lieutenant Commander, United States Navy
BA, University of Texas at Austin, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
December 2018**

Approved by: Thomas W. Otani
               Advisor

               Arijit Das
               Second Reader

               Peter J. Denning
               Chair, Department of Computer Science

iii

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In today's data-intensive world, the power to analyze huge amounts of data is critical to the success of any organization, including the military. Many data analysis tools have been developed in the past decade along with the high-performance machine learning algorithms. At present, many of these tools unfortunately are out of reach of the target audience—subject matter experts—because one must master some of the advanced computer science concepts to use these tools effectively.

This thesis proposes to build a prototype data analysis platform that will hide the underlying complexity of the tools from the subject matter experts. Using the platform, the end users can analyze data through a simple, menu-driven interface. The prototype will be built using the programming language Python and the open-source, distributed data processing engine Apache Spark 2.0. Different components of Spark 2.0 will be studied and evaluated to determine the best approach for building the prototype.

The effectiveness of the prototype will be examined using the ADSB (Automatic Dependent Surveillance - Broadcast) unfiltered flight data. The thesis concludes with the review of the prototype developed for ADSB and the recommendation on possible ways of extending the prototype.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ACARS	aircraft communications addressing and reporting system
ADSB	automatic dependent surveillance—broadcast
API	application programing interface
CSV	comma separated value
DAG	directed acyclic graph
IATA	International Air Transport Association
JSON	java script object notation
JDBC	java database connectivity
KML	keyhole markup language
MLAT	multilateration
ODBC	open database connectivity
ORC	optimized row columnar
SQL	structured query language
TIS-B	traffic information system broadcast
VRS	virtual radar server

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

THIS PAGE INTENTIONALLY LEFT BLANK

I.   INTRODUCTION

Throughout history, leaders in the military, government, business or other organizations have attempted to plan, collect, process, analyze, disseminate and act on information as quickly as possible. This process is very similar to the OODA loop or Observe, Orient, Decide, Act. The leader or organization that is able to analyze information more effectively and efficiently and make a well-informed decision quicker than their rivals is more likely to be successful in their endeavors and will have the initiative.

Information being produced, gathered and stored on a daily basis is growing in size and at an ever-increasing rate. "By the year 2020, about 1.7 megabytes of new information will be created every second for every human being on the planet" (Business First Magazine 2016, 70). This presents a problem in the "Orient" part of the OODA loop because discerning correct conclusions from data is becoming increasingly challenging and time consuming due to the size of the data. Big data analysis is the use of data analytic techniques and algorithms on very large datasets. Additionally, data analysis, especially in regard to big data, requires extensive time to learn and become proficient in. Understanding the basic and advanced data analytics, machine learning methodologies and computer programing in order to conduct analysis can be time consuming and challenging. This can be a heavy burden for domain subject matter experts.

An easy to use interface may be able to reduce this burden on subject matter experts. If data analysis methods and machine learning techniques can be implemented behind the scenes this would allow subject matter experts to conduct data analysis in a timely manner, and allow them to concentrate on their area of expertise rather than having to learn specifics of different programming languages, programming libraries and the various machine learning techniques.

A.   OBJECTIVE

The objective of this research is to build a prototype to study the feasibility of an easy-to-use interface for data analysis to include a machine learning algorithm. The

interface will be designed so that domain subject matter experts will be shielded from coding and from the underlying algorithms.

The prototype interface will be built using Python and Apache Spark 2.0. Python will be used for the interface because of the wide array of tools available as well as its ease for building a command line interface. Spark2 Data Frames and its library of data analysis and machine learning tools, will be used to examine if various analysis techniques can be abstracted away behind the scenes while still providing accurate and fast data analysis of big data.

The data to be studied with the prototype will be the automatic dependent surveillance—broadcast or ADSB data. The ADSB data was chosen because of the breadth of data types contained in the data and it allowed for the testing of the prototype.

B.   ORGANIZATION

The research is organized into five chapters. Chapter II provides background information and explains the various technologies and programming libraries used when implementing the prototype. Chapter III contains the methodology and specific approach for the implementation of the prototype. Chapter IV contains the review and analysis and challenges faced during this research and Chapter V contains the conclusions and future work in regard to research on this topic.

II.   BACKGROUND

A.   APACHE SPARK

Apache Spark was originally developed at UC Berkeley in 2009 and is defined by the Apache Software Foundation as "a lightning-fast unified analytics engine for large-scale data processing" (Apache Spark 2018).

Spark is an open-source program that enables distributed computing capabilities and machine learning capabilities to be brought to bear for the analysis of large data sets. Many large names in industry to include well-known companies such as Netflix, Yahoo, eBay and Hotels.com are deploying and using Spark to address the problems with big data. With this exposure, Spark "has quickly become the largest open source community in big data, with over 1000 contributors from 250+ organizations" (Databricks 2018).

New releases of Spark with added features occur constantly, with three updates occurring between June 08, 2018 and July 02, 2018. These releases are done by the Spark Software Foundation at https://spark.apache.org.

A.1   Benefits of using Spark

Speed is the primary benefit of Spark. In fact, Spark won the 2014 Gray Sort competition in the Daytona division (Xin 2014). This was a competition to sort 100TB of data. Spark was deployed on 206 EC2 machines and sorted the data in 23 minutes smashing the previous world record set by Hadoop MapReduce of 72 minutes using 2100 machines. In other words, Spark sorted the same amount of data in a third of the time using 10 percent of the computing power used previously (Xin 2014).

"Spark achieves this high performance using a state of the art DAG scheduler, a query optimizer and a physical execution engine" (Apache Spark 2018). Figure 1 shows the running time in seconds for Spark versus Hadoop when conducting Logistic regression on a sample data set.
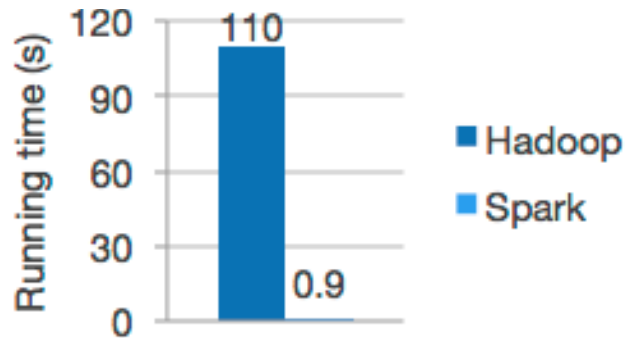
Figure 1.   Logistic regression bar graph comparison of Hadoop and Spark.

A second benefit of Spark is its ease of use. Spark can be utilized with a number of different programming languages. Spark currently supports some of the most well-known programming languages to include Java, Scala, R, Python and SQL. Additionally, Spark has a large number of application programming interfaces or APIs, for conducting operations on, transforming, and manipulating data.

Another benefit of Spark is the high-level libraries. These libraries include SQL, Data Frames, a machine learning library or MLlib, GraphX and even a library for streaming data. These libraries increase the productivity of a developer, allow for more complex programs while at the same time reducing the stress on the developer. The libraries and supported API's in the Spark Ecosystem are shown in Figure 2.
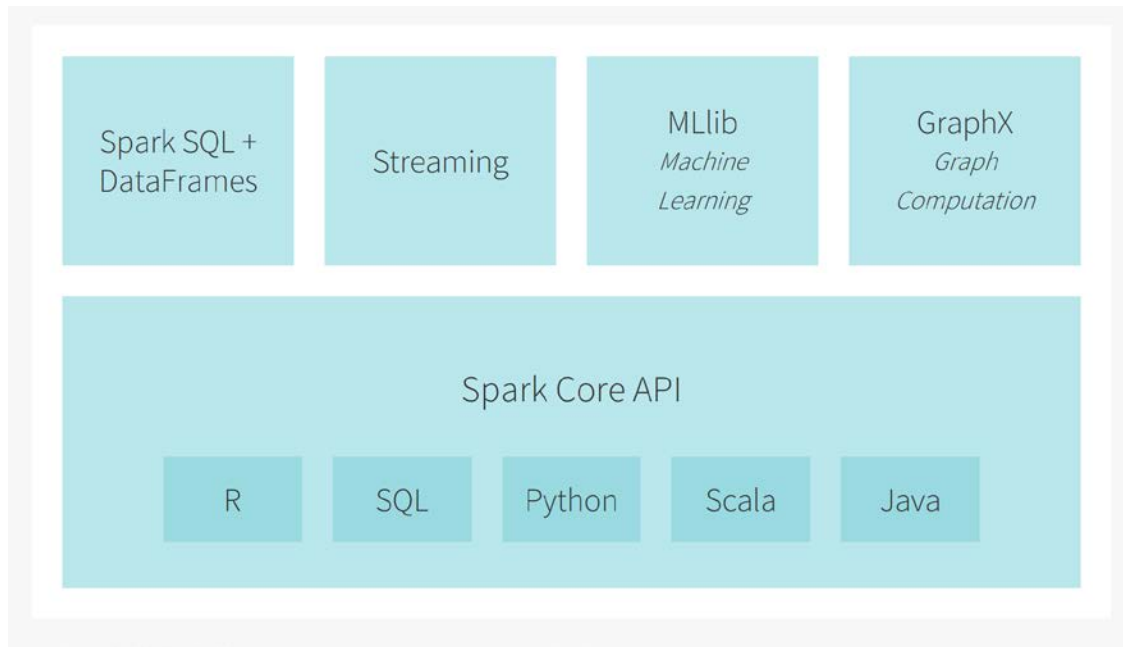
Figure 2.   Apache Spark ecosystem.

One last benefit of Spark is that it gives developers and users many options when it comes to the employment of Spark capabilities. Spark can be utilized just about anywhere. Spark can be run on a single laptop. It can also be employed with Hadoop, Apache Mesos and take advantage of many nodes in cloud architecture

A.2   Architecture

While Spark can be run on a standalone computer, a standalone computer does not contain enough power to perform timely analysis on big data. It takes a cluster, or a group of computers, running in parallel that can combine the resources of the various groups of computers to handle data analytics. This is where Spark provides an advantage.

Spark provides a framework for "managing and coordinating the execution of tasks on data across a cluster of computers" (Chambers and Zaharia 2018, 13). Spark consists of three main parts, the cluster manager, the driver process and the executors. A Spark cluster manager coordinates the resources for use between the driver process and the executor. All of this coordination is handled behind the scenes without programmer or user involvement

in the process. The driver process contains the user's code and is responsible for "maintaining information about the Spark application; responding to a user's program or input; and analyzing, distributing, and scheduling work across the executors. The driver process is absolutely essential—it's the heart of a Spark Application and maintains all relevant information during the lifetime of the application" (Chambers and Zaharia 2018, 14). The executors will execute the work assigned to them and report back the result of the computation to the cluster manager. Figure 3 shows cluster manager control of a Spark application.
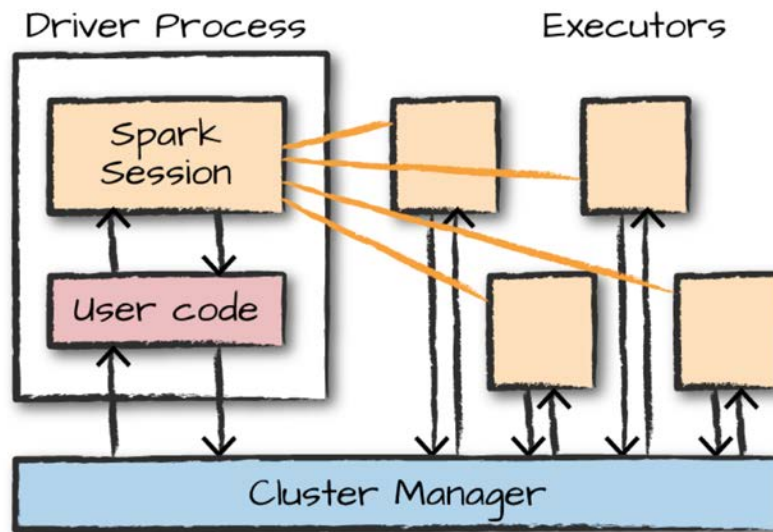
Figure 3.   The architecture of a Spark application.
Source: Chambers and Zaharia (2018).

A.3   Spark SQL

"Spark SQL is arguably one of the most important and powerful features of Spark" (Chambers and Zaharia 2018, 179). SQL or structured query language is a special programming language for accessing, managing and working with data in relational databases. Spark SQL allows a user to run SQL queries against tables and databases as well as be run with user defined functions. SQL can also be run with DataFrames and DataSets.

> The power of Spark SQL derives from several key facts: SQL analysts can
> now take advantage of Spark's computation abilities by plug in into the

Thrift Server or Spark's SQL interface, whereas data engineers and scientists can use Spark SQL where appropriate in any data flow. This unifying API allows for data to be extracted with SQL, manipulated as a DataFrame, passed into one of Spark's MLibs' large-scale machine learning algorithms, written out to another data source, and everything in between (Chambers and Zaharia 2018, 180).

A.4 DataFrames

A DataFrame is a data structure that work like a table, similar to a spreadsheet. There are well defined rows and columns. In contrast to a spreadsheet, there are restrictions on the types of data in the DataFrame. The data in a column must all be the same data type and the data across all the rows must be consistent for each column. A null value can be used in the case of the absence of data or if a value is unknown. Rows usually are instances of data whereas columns are the types of data associated with the row instance. Examples of columns are altitude, speed, etc. An advantage of DataFrames are that they allow operations to be easily be applied to data specific locations such as a specific row or column or even to a specific data cell at the intersection of a row/column. An example of a DataFrame from the ADSB data is shown in Figure 4.

```
+------+----+-----+----+----+-----+--------------------------+
|Icao  |Spd |Alt  |Lat |Long|Mil  |Cou                       |
+------+----+-----+----+----+-----+--------------------------+
|null  |null|null |null|null|null |null                      |
|464E35|null|31000|null|null|false|Finland                   |
|4CAB5E|null|38000|null|null|false|Ireland                   |
|53EE8B|null|-800 |null|null|false|Unknown or unassigned country|
|AB2458|null|300  |null|null|false|United States             |
+------+----+-----+----+----+-----+--------------------------+
```

Figure 4.   Example DataFrame printout

The schema or architecture of a DataFrame, can be manually defined by a developer or it can be obtained from some other source. A schema consists of the names of the columns in the DataFrame and the data type associated with each column. An example Schema is provided in Figure 5.

7

```
root
 |-- Icao: string (nullable = true)
 |-- Spd: double (nullable = true)
 |-- Alt: long (nullable = true)
 |-- Lat: double (nullable = true)
 |-- Long: double (nullable = true)
 |-- Mil: boolean (nullable = true)
 |-- Cou: string (nullable = true)
```

Figure 5.   Spark DataFrame schema example

A.5   Machine Learning

Machine learning is an area of computer science that attempts to get computers to learn and make predictions about patterns in new data based upon data that the computer was trained on or has seen previously seen. Machine Learning is classified as unsupervised or supervised machine learning.

Supervised learning consists of the computer receiving input variables and output variables and the machine attempts to discern the function or algorithm to create a model that maps the input to the output. In supervised learning the machine is provided a labeled training data set to "learn on" and then it will be tested using a new dataset to make a prediction on the new data. Supervised learning is usually either a classification or a regression problem. Some examples of supervised learning are Linear Regression, Support Vector Machines and Logistic Regression. The supervised learning model is shown in Figure 6.
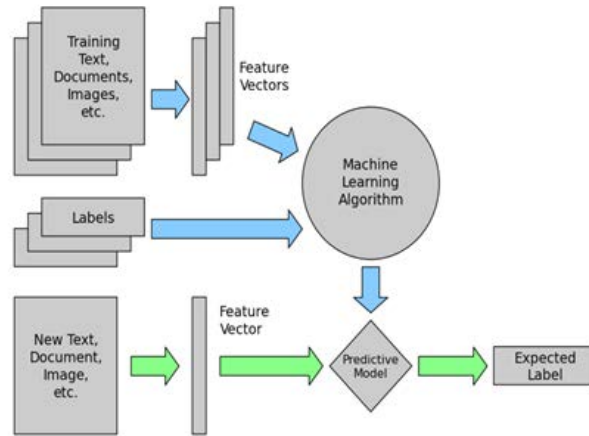
Figure 1. Supervised learning model

In unsupervised learning the machine tries to discern patterns in input data even though the data has not been labeled or classified. The machine must determine what is interesting in the data without a "teacher." Unsupervised learning usually falls along the lines of a clustering problem or an association problem. A common example of unsupervised learning is k-means clustering. The unsupervised learning model is shown in Figure 7.
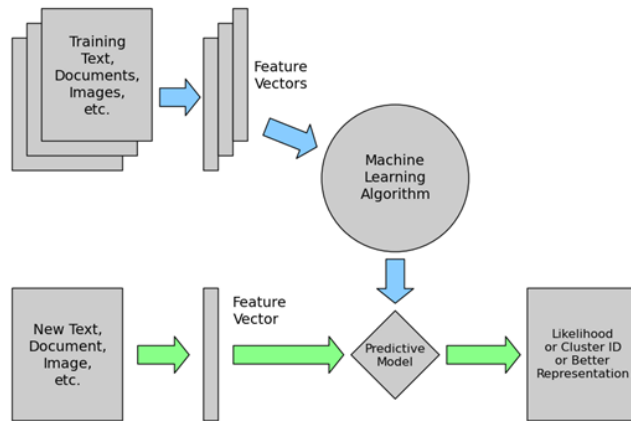
Figure 2.   Unsupervised learning model

9

B.   AUTOMATIC DEPENDENT SURVEILLANCE—BROADCAST (ADSB)

ADSB is an electronic surveillance technology at the heart of the Federal Aviation Administration's Next Generation Transport System. ADSB is designed to give a more accurate picture of the current air picture. ADSB will combine data from GPS satellites and aircraft avionics data such as speed and altitude and transmit it to a ground station that will re-transmit to air-traffic controllers as well as other pilots with properly equipped aircraft.

Historic ADSB data as well as a real-time stream of ADSB data can be found at https://www.adsbexchange.com/. The data at the ADSB site is in the java script object notation format or JSON.

The following is a list of the data fields from the ADSB JSON files. Next to each listing is a description of the identifier. When ingesting this data in with Spark, these data fields will correspond to the columns in the DataFrame. Information in the listing is from the data field descriptions page located at https://www.adsbexchange.com/datafields/

- Id—Unique identifier

- Rcvr—Receiver ID number

- HasSig—True if the aircraft has a signal level associated with it.

- Sig—Signal level for the last message received.

- Icao—Six-digit hexadecimal identifier broadcast by the aircraft

- Reg—Aircraft Registration number

- Fseen—Date and Time the receiver started seeing the aircraft on this flight

- Tsecs—Number of seconds that the aircraft has been tracked for

- Cmsgs—The count of messages received from the aircraft

- Alt—Altitude in feet at standard pressure

- Galt—The altitude adjusted for local air pressure, should be roughly the height above mean sea level

- InHG—Air pressure in inches of mercury that was used to calculate the AMSL altitude from the standard pressure altitude

- AltT—Type of altitude transmitted by aircraft

- Lat—Latitude of aircraft

- Long—Longitude of aircraft

- PosTime—Time that the position was last reported by aircraft. In epoch time.

- Mlat—True if latitude and longitude appear to have been calculated by an MLAT and not transmitted by aircraft

- TisB—True if last message received from aircraft was from TIS-B source

- Spd—Ground speed in knots

- SpdType—Type of speed that Spd represents

- Trak—Aircraft's track angle across the ground clockwise from zero degrees north

- Type—Aircraft model's ICAO type code

- Mdl—Aircraft Model

- Man—Manufacturer's name

- Year—Year aircraft was manufactured

- Cnum—Aircraft's construction or serial number

- Op—Name of aircraft operator

- OpIcao—ICAO code of the operator

- Sqk—Transponder code

- Vsi—Vertical speed in feet per minute

- VsiT—Vertical speed is barometric (0) or geometric (1)

- WTC—Wake Turbulence category

- Species—General Aircraft Type

- EngType—Type of Engine the aircraft uses. Five available classes

- EngMount—The placement of engines on the aircraft. This information is from a database based on Icao code.

- Engines—The number of engines on the aircraft.

- Mil—True if the aircraft appears to be operated by the military. Based on Icao code

- Cou—Country the aircraft is registered to

- From—The code and name of the departure airport

- To—The code and name of the destination airport

- Gnd—True if the Aircraft is on the ground

- Call—Callsign of the aircraft

- CallSus—True if the callsign may not be correct

- HasPic—True if aircraft has a picture associated with it in the VRS/ADSB exchange database

- Trt—Transponder type

- TT—Trail type

- Talt—Target altitude set by autopilot broadcast by aircraft

- Ttrk—The track or heading currently set on the aircraft's autopilot

- Sat—True if the data has been received via a SatCom ACARS feed

- PosStale—True if the last position is older than the display timeout value

III. METHODOLOGY

A. MAIN PROGRAM APPROACH

As mentioned in the Introduction, this program was developed using Python. The required python libraries for the program include numpy, matplotlib, and pandas. These libraries were crucial in implementing some of the features including the graphing or charting option. As of the work on this program, Spark did not have an inherent graphing or charting option. Figure 8 details the program scheme. This code has been developed to run on an individual laptop or on a cluster that contain the appropriate libraries listed above.
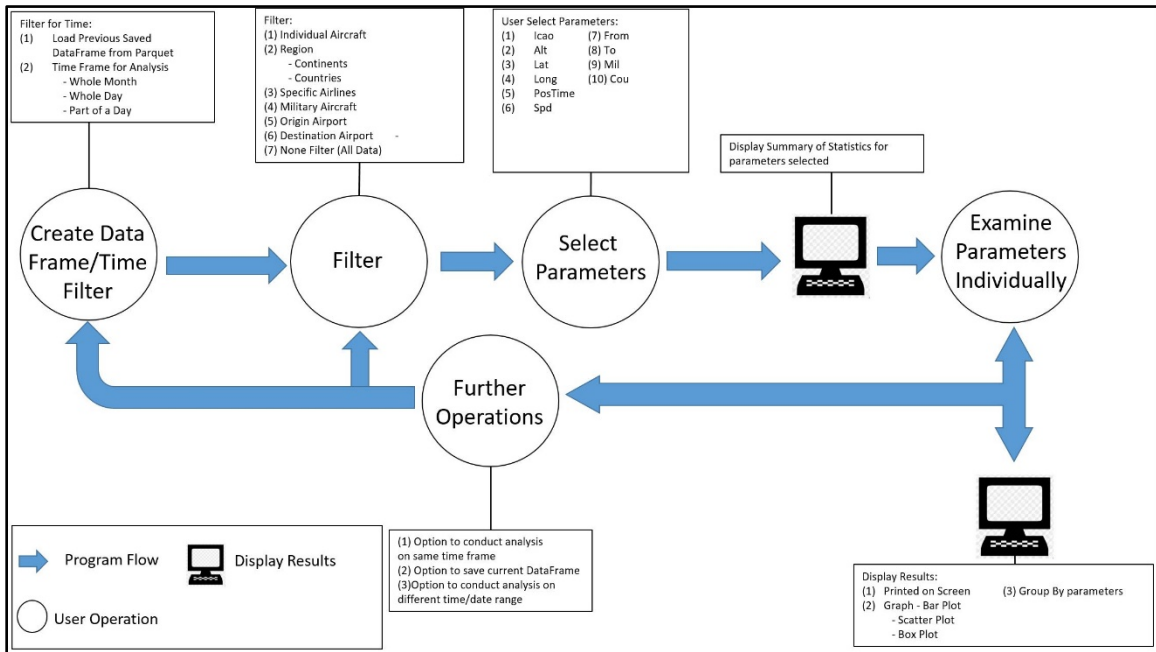
Figure 8.   Analysis program scheme

The main program control is maintained in the AnalysisControl file. A rough run through the program involves asking a user if they want to load a previously saved parquet file or provide a time filter with other options to filter. A time frame for the filter is required because of how the data is stored. The JSON files are stored and labeled according to date/time with each JSON file holding data by the minute. Other filter options consist of

specifying a specific region (countries or continents), a specific airline, military aircraft, a specific airport or no filter at all. After the filter, according to the options selected, the program will ask the user to select the parameters for analysis. The program will then give a basic analysis of each parameter as a group. The program will then proceed through each parameter one at a time to allow the user to conduct additional analysis on the parameter, allow for analysis by grouping by parameters, as well as provide the ability to display a chart or graph.

A.1   Loading a Previous Saved DataFrame in the Form of a Parquet

The option to load a previously saved parquet is the first option for a user to select in the program. This option is available so that a user can load a previously saved DataFrame that has already been analyzed. A parquet is "an open source column-oriented data store that provides a variety of storage optimizations, especially for analytics workloads" (Chambers and Zaharia 2018, 163). Specifically, a parquet is designed to save space and also allows for the reading in of single columns vice the whole file. Additionally, a parquet is quicker for the reading in of data than other file types such as CSV or JSON and a parquet can also save/load complex data types.

If it is desired to load a previously saved parquet, then a location must be provided to the program. The format for the location is in following format:

C:\Users\UserName\Desktop\Saved\saved.parquet.

Figure 9 shows a screenshot of the program for loading the parquet as well as the format required by the program.

```
Do you want to load a saved parquet file?(Y/N)
Y
Please provide the file location in the following format:
C:\Users\User1\Desktop\Saved\saved.parquet
```

Figure 9.   Program screenshot for loading a parquet file

After a location is provided, assuming it is valid, the program will progress to the main filter for the program. Note, the load parquet option allows the user to load a previously loaded dataset that has been already been worked on. Any filters, and parameters that were used previously will still be applied. After loading the parquet, the user will be given the list of filters to apply to the DataFrame.

If no parquet is loaded, the next option the user will be given will be to specify the time frame for analysis.

A.2   User Time Frame Input

The specification of a time frame for analysis is a part of the time filter. This filter is required because of how the ADSB JSON files are stored. As mentioned earlier, the JSON files are stored according to date and time with each file consisting of a minute worth of ADSB data. The following is an example name for a JSON file containing the ADSB data for midnight Zulu time on 15 November 2016: 2016–11-15-0000Z.json.

The time frame for analysis is provided by the user but it does have a few restrictions. The options for time frame analysis are a whole month, a whole day or specified part of a day. If a whole month is selected, the user will provide the year and the month. For a whole day, the user will provide the year, month, and day. For a part of a day, the user will provide the same information as a whole day but will also be asked to specify a beginning time and ending time for analysis.

Time frame analysis information is obtained via the *UserTimeInput* function in the ReadIn program file. This function takes the users' time inputs and builds the file location

or locations for Spark to read in the JSON file or files. The function returns the following information:

- analyze—a categorical variable for the time frame of analysis. Available values include a month, a day or part of day

- year—four-digit string variable pertaining to the year

- month—two-digit string for the month.

- day—two-digit string for the day.

- begin—four-digit string from 0000–2359 corresponding to time of day to begin analysis. Value will be None if analyzing a month or whole day.

- end—four-digit string from 0000–2359 corresponding to time of day to end analysis. None if analyzing a month or whole day. The value must be later than the begin time.

- filelocation—a string or list of strings for the file location or locations of the JSON files.

After the input for time frame of analysis is entered, the function *createDFfromtime*, shown in Figure 10, creates a spark session. A spark session is required for control of a spark application. The function then takes the outputs from the *UserTimeInput* function to create the initial DataFrame. The *spark.read.json* function is the spark function utilized to read in the JSON file or files. The *spark.read.json* function can take as input a string corresponding to a single file location or a list of strings corresponding to many file locations. It will then return a DataFrame with the data from the JSON files.

The code in Figure 10 also shows the spark session creation and the spark.read.json function call. If analyze is equal to one, which corresponds to analyzing a whole month or it equals two, which corresponds to analyzing a whole day, then the function uses the string corresponding to the time frame desired by the user for the file location to create the DataFrame. If analyze is equal to three, which corresponds to analysis on a part of a day, then the createDFfromtime function uses the function getTimeFiles in the ReadIn file to create a list of file locations. This list is then passed to the spark.read.json function for creating the DataFrame.

16

```
#Create the dataframe from the inputs from UserTimeInputs and the file locations.
#Calls getTimeFiles function and then creates the datframe
#Inputs:
#       analyze - A 1 digit indicator (1 for month, 2 for day, 3 for part of a day)
#       year - 4 digit (string)
#       month - 2 digit (string)
#       day - 2 digit (String)  - Returns None if analyzing a whole month
#       begin - 4 digit (String) - Returns None if analyzing a whole month or whole day
#       end - 4 digit (String) - Returns None if analyzing a whole month or whole day
#       filelocation - string or list of strings
#Returns:
# A dataframe corresponding to the timeframe selected by the user.
def createDFfromtime(analyze, year, month, day, begin, end, filelocation):

    if analyze == "1" or analyze == "2":

        spark = SparkSession.builder.appName("AppOne").getOrCreate()
        df = spark.read.json(filelocation)

    elif analyze == "3":

        listing = getTimeFiles(year, month, day, begin, end)

        spark = SparkSession.builder.appName("AppOne").getOrCreate()
        df = spark.read.json(listing)

    return df
```

Figure 10.   CreateDFfromtime function code

A.3   Apply Filter Function

The optional filters are next. The available optional filters are shown in Figure 11.

```
Select the ADSB data you want to analyze(Multiple Choices will be possible in some cases):
1:  Individual Aircraft
2:  Region Airspace
3:  Specific Airline
4:  Military Aircraft
5:  Origin Airport
6:  Destination Aiport
7:  No Filiter(All Data)
Please make a selection.  Press 0 to stop selecting options
```

Figure 11.   Filter options for program

These filters are selected with the *UserSpecificsRequest* function in the NewFilter file and are applied in the *ApplyFilter* function in the NewFilter file. In many cases, multiple filters can be selected by the user. The user will make selections and select zero

17

to finish selecting options. Once an option is selected, it is removed so the list of filtering options will shrink in size. Options that conflict with an already selected option are removed automatically. For example, if Military Aircraft is selected, the option for Specific Airline will be removed from the available options. The user selections will be stored in a list. This list of filters combined with the created DataFrame are the inputs into the *ApplyFilter* function which returns a DataFrame with the filters applied. After the filters have been selected the program will display the filters that were selected as a reminder to the user.

a.   *Individual Aircraft*

The individual aircraft filter in the *ApplyFilter* function is a filter that requests the user to input the ICAO for an individual aircraft to be analyzed. An ICAO is a six-digit hexadecimal string that is unique to each aircraft. An example of an ICAO number is A7FB98. In the function, the ICAO number is used to perform an SQL statement on the DataFrame. This statement will create a new DataFrame that will only contain information pertaining to the ICAO provided. Figure 12 is a small snippet of code, demonstrating the SQL statement, for the filtering of the data for a single ICAO. If the ICAO is an invalid ICAO or the ICAO is not in the DataFrame, the program will request another ICAO from the user. The program determines if there is any data for the ICAO in the DataFrame by using the Spark function *df.count* to count the number of rows in the DataFrame. If the returned count is zero, the DataFrame does not contain any data for the selected ICAO.

```
while True:
    print("Please provide the Icao of the aircraft you want to analyze: ")
    answer = input()
    df1 = df.where(col('Icao') == answer)
    count = df1.count()
    if count == 0:
        print("No data for that Icao.")

    else:
        df = df.where(col('Icao') == answer)
        break
```

Figure 12.   Individual aircraft filter demonstrating an SQL select statement

18

The individual aircraft filter is the only filter that will run without any other user input. The *getParameters* function from the DataEntries file is called by the program and it automatically selects the parameters "Spd," "Alt," "Lat," "Long," and "PosTime." The function is set up to add additional parameters for an individual aircraft in future updates.

Once the ICAO is provided to the program, the *ApplyFilter* function will return a DataFrame and the *analysisoperations* function is called on the DataFrame. This will output the basic statistics of count, mean, standard deviation, minimum and maximum as shown in Figure 13.

Unlike in the rest of the program the user is not given many graphing options. The graphing options are based on the available columns in the DataFrame. The program gives the user the option to display one of three graphs or all three graphs. The available graphs for an individual aircraft are shown in the bottom of Figure13.

```
Statistics for the parameters selected:
+-------+------------------+------------------+------------------+------------------+------------------+
|summary|              Spd|              Alt|              Lat|             Long|           PosTime|
+-------+------------------+------------------+------------------+------------------+------------------+
|  count|               60|               60|               60|               60|               60|
|   mean|461.98499999999996|35982.916666666664| 40.36458615000001|  -94.76351735|1.4791698057347E12|
| stddev| 8.864838656556552|20.321393641874778|0.9302885916255783|2.604580560598473|1038482.2947531358|
|    min|             447.6|            35950|         39.013103|      -98.581604|    1479168026051|
|    max|             479.0|            36025|          41.82416|      -90.249236|    1479171512459|
+-------+------------------+------------------+------------------+------------------+------------------+

Select the graphs to display
1: Plot of Spd vs Time
2: Plot of Alt vs Time
3: Plot of Spd vs Alt
4: All of the above
```

Figure 13.   Statistics of data and graphs to display

The charts or graphs available to the user are a plot of "Spd" vs "Time," a plot of "Alt" vs "Time," and a plot of "Spd" vs "Alt." A sample plot of time vs speed for a flight is provided in Figure 14. The plots are created using the *graphInd* function contained in the GraphIt file. Graphing will be discussed more in depth under Methodology section A.6
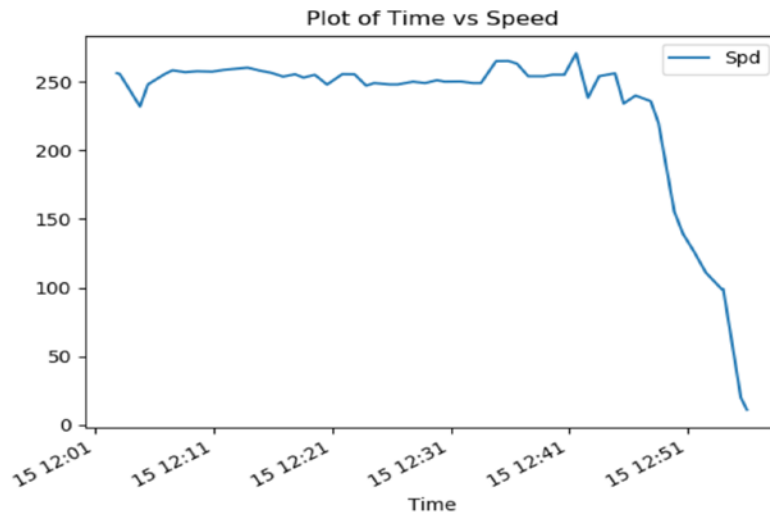
Figure 14.   Example graph for analyzing a single aircraft

The last item available to a user under individual aircraft analysis is the creation of a keyhole markup language, or KML, file. The KML file is automatically created by the program using the simplekml library available with Python programming language. The program will create a sequence of observations, with each observation consisting of a name and a coordinate for the aircraft. Each observation contained in the KML will match up with an observation in the DataFrame. The program will name each observation in the KML file by combining the ICAO with a modified position time value. The modified position time value is a position time value that has been converted to a YYYY-MM-DD HH:MM:SS format. This ensures a unique naming value for each coordinate. The coordinates for each observation in the KML are the values from the latitude and longitude columns in the DataFrame.

The created KML file can be loaded into Google Earth, for Geospatial Analysis, to analyze the individual aircraft's track. An example of an individual aircraft's track in Google Earth is shown in Figure 15. The ICAO, date and time of each observation can be seen for each observation.
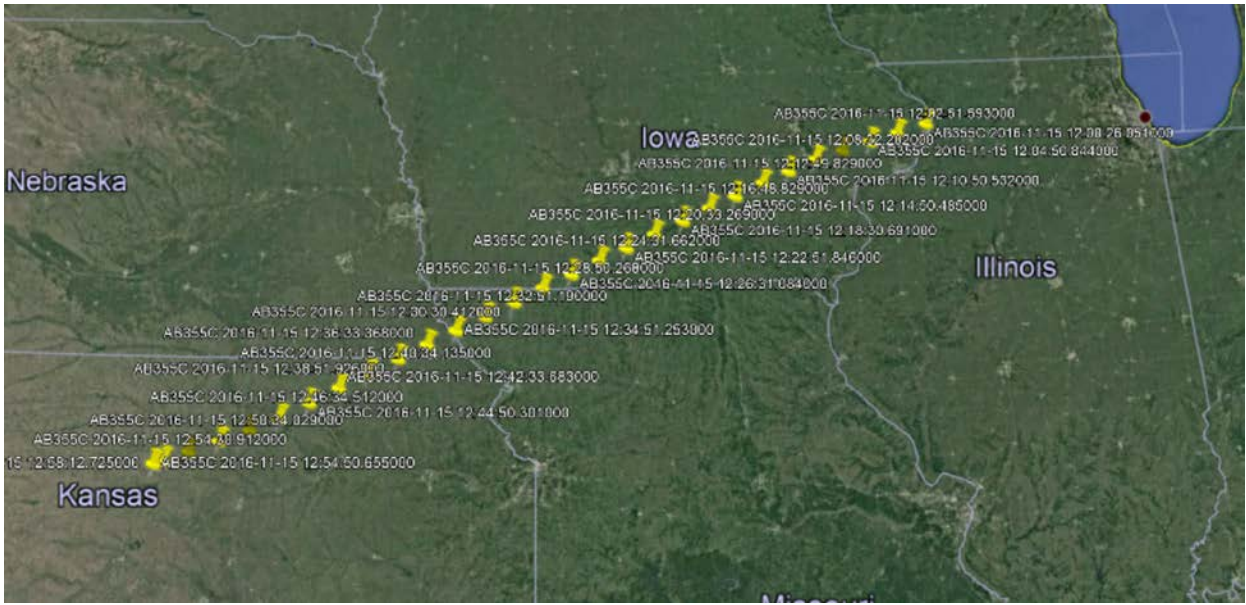
Figure15.   Google Earth screenshot for an individual aircraft's
track using a KML

b.   *Region Airspace Filter*

Analyzing a specific region airspace is a second part of the filter. This part of the filter is designed to filter ADSB data corresponding to a specific region. Regions currently available in the program include North America, South America, Europe, Africa, Australia, the entire world, or a specific country. Specific countries included in the program are the United States, France, Germany, United Kingdom, China, North Korea. The boundaries of these regions and countries are approximate and have been hardcoded.

The regional filter operates using the point in poly algorithm. The point in poly algorithm takes two inputs, a point and a polygon. A point corresponds to a coordinate that has a latitude and longitude. The format for the point is a tuple. The polygon is represented using a list of tuples. Each tuple in the list corresponds to a point on the boundary of the polygon. The general point in poly algorithm will return True if the point is contained inside the polygon or False if the point is not in the polygon. If the point lands on the edge of the polygon, then the program will return False. Parts of the *Inpolygon* function shown in Figure 16 are borrowed from W Randolph Franklin of Rensselaer Polytechnic Institute (Franklin 2017). Example code is available at his website at:

. His code was written in C, and has been translated to Python and modified in order to return a specific area.

For the program, the point is created using the latitude and longitude for each observation in the DataFrame. The polygon is a hardcoded list of coordinates for a region or country. A polygon's last entry in the polygon is a string corresponding to the name of the polygon.

```python
def Inpolygon(point,poly):

    name = poly[-1]
    poly=poly[0:-1]
    inside = False

    x=point[0]
    y=point[1]

    i = 0
    num = len(poly)
    j = num-1

    for i in range(num-1):

        poly1 = poly[i]
        poly2 = poly[j]

        if ((poly1[1] > y) != (poly2[1] > y)) and \
                (x < poly1[0] + (poly2[0] - poly1[0]) * (y - poly1[1]) /
                    (poly2[1] - poly1[1])):
            inside = not inside

        j = i

    if inside == True:
        return name
```

Source: W Randolph Franklin.
https://wrf.ecse.rpi.edu//Research/Short_Notes/pnpoly.html,
(accessed 02 November 2018)

Figure 16.   Python implementation of the Inpolygon function

The *Inpolygon* function is used inside the *assignCountry* or *assignContinent* functions. These functions loop through a list of the countries or continents. For each region the program will determine if the point is inside it. If it is inside, it will return the name of the region. The name is then added to the DataFrame on the row corresponding to the

22

latitude and longitude used. The functions are also named *countryudf* and *continentudf* for use in Spark. Multiple functions were required because the functions have to be defined as user-defined functions. When a function is designated as user-defined, it allows it to conduct operations on a DataFrame.

c.   *Specific Airline*

The next filter is one that will filter out specific airlines. The specific airlines that the user can select for analysis were hardcoded in the program. The names used were determined after analysis of data in the "Op" section of the JSON files. Examples of specific airlines that can be filtered are United Airlines, American Airlines, China Airlines, Virgin Australia, and Lufthansa. Additional airlines can be added very easily to the code to filter for them.

In the code, the "%" has been added at the end of the airline names. This is used to find the name in the "Op" column of the DataFrame. It was required because the names in the "Op" column were not consistent. For example, American Airlines in the "Op" column could have the names "American Airlines," "American Airlines carrier" or "American Airlines—Fort Worth." The "%" allows the program to capture all three of these that correspond to American Airlines.

In order to filter the rows that have a specific airline the program requests the user select an airline from a list. It then filters the DataFrame using the spark *df.filter* function so that the only rows included in the new DataFrame are those that have a specific airline in them.

d.   *Military Aircraft*

The Military Aircraft filter works in much the same way as the specific airline filter. The user selects a military or branch of the military that has been hard coded into the program. The available selections were based on quick examination of several ADSB JSON files. The military options provided include all the branches of the United States Armed forces, the armed forces of Spain and the Royal New Zealand military. More options can be added very easily. The program filters the rows of the DataFrame based on

the "Op" column in much the same way as the Specific Airline filter. The filter uses the *df.filter* spark function in the same way that the specific airline filter does but the filter for military aircraft also ensures that the column "Mil" is equal to True.

e. *Origin and Destination Filters*

The filter for selecting an origin or destination airport functions like the specific aircraft filter except in this case there is some work that occurs behind the scenes. The user is shown airport options for names of airports. The selection is then looked up in a dictionary, so that it can be converted to a corresponding airport IATA, or International Air Transport Association, code. The IATA code is used because of the format of the data in the "To" or "From" columns of the DataFrame. If the "To" or "From" column was not null, it contained the IATA code for the airport the aircraft. The *df.filter* spark function was used to filter the DataFrame for a specific origin or destination airport based on the IATA code for the airport selected. Other airports can easily be added in the future.

f. *No Filter*

Lastly there is an option for no filter. This allows the user to conduct analysis on all the data pulled in for the time frame selected earlier.

A.4   Selection of Parameters

After the application of filters has been completed, the user is given the options of which parameters to select. The parameters available correspond to the columns in the DataFrame. The available parameters for the program are shown in Figure 17. The user will select the parameters for analysis, which will then be added to a list of parameters. This is done until the users enters "0." The program will then create a new DataFrame that consists only of the columns corresponding to the parameters chosen by the user. More parameters are available for future use but have been commented out to simplify the program. Descriptions of the ADSB parameters are contained in Chapter II.

```
Which of the following parameters would you like to examine:
1: Icao    :   The ICAO of the Aircraft
2: Alt     :   The altitude in feet at standard pressure
3: Lat     :   Aircraft's latitude over the ground
4: Long    :   Aircrafts longitude over the ground
5: PosTime :   Time of the aircraft's last position
6: Spd     :   The ground speed in knots
7: From    :   The code and name of the departure airport
8: To      :   The code and name of the arrival airport
9: Mil     :   True if operated by the military
10: Cou    :   The country the aircraft is registered to
You will be entering values for the paramerters you want to examine.
Please select the first parameter to examine.  Enter 0 when done.
```

Figure 37.   Available parameters for the program

A.5   Analysis Operations

After the parameters have been selected the program will conduct analysis operations using the *analysisoperations* function in the DataEntries File. *Analysisoperations* begins by calling the Spark *describe* function which will provide a count, mean, standard deviation, min and max values for all the parameters in the DataFrame except for Boolean parameters. Boolean parameters do not get shown when using the Spark *describe* function.

The *analysisoperations* function will then go through each parameter one by one for individual analysis. Parameter types for the ADSB data are broken into three different types, with each parameter being either a string, a numeric or a Boolean. Examples of ADSB data that are string types are "Icao," "From," "To," and "Cou." Examples of Boolean parameters are "Bad," "Mil," and "Gnd." Examples of Numeric types are "Alt," "Lat," "Long," "PosTime," and "Spd." The *analysisoperations* function handles each data type differently because the information gleaned from each data type differs by its type.

For a string, the function will use the describe function. The describe function as mentioned earlier will provide a count, mean, standard deviation, the minimum and maximum value. The most useful part from the describe function on a string is the count. The count will not include rows that have null values. The function will then ask if the user wants to construct a graph. Graphing will be discussed later in section A.6. After asking

25

the user if they want a graph, the user will be asked if they want to group by the parameter. If yes, the program will ask what parameters, from the group of parameters the user had selected earlier, that they want to group by. The program will create a new temporary DataFrame of grouped parameters with a count for each grouping. The user will then be asked how they want to display the results. Available options for display are "On Screen," "Graph of values," or "Both." The display on screen option will look similar to Figure 18. Figure 18 shows a screenshot of "Icao" and "Spd" with a count given for each specific grouping.

```
|  Icao|  Spd|count|
+------+-----+-----+
|2B09DF| 83.0|    1|
|7809AD|349.1|    1|
|A258D4|308.0|    6|
|A34721|515.5|    2|
|A27FD6|385.8|    1|
|A7DB46|264.4|    6|
|A7BBA1|446.9|    1|
|A7BC58|324.2|    1|
|A50BBB| 34.0|    6|
|A51EB3|334.8|   21|
|A4DA13|415.3|    1|
|7C39F3|290.5|   11|
|AD9DAB|  8.0|    1|
|AA1C1F|582.0|   28|
|AC4A44|446.8|    2|
|89612C|296.0|    1|
|4005E2|304.0|    1|
|A2A87D|462.0|    2|
|ABDCFD|370.2|    1|
|4B187D|499.8|    2|
+------+-----+-----+
only showing top 20 rows
```

Figure 18.   Display on screen option screenshot after a groupby.
"Icao and "Spd"

Numeric parameters are handled slightly differently than string parameters. If the parameter being analyzed is "Spd" and the parameter "Alt" is in the DataFrame the program will ask if the analysis for "Spd" will be for aircraft that are in the air. In other

26

words, the user will have the option to examine statistics for speed for all aircraft in the DataFrame or only those in the air. To determine if an aircraft is in the air, the program will select the rows where "Alt" is greater than zero. The Spark describe function is then used to get the count, mean, standard deviation, minimum and maximum values. In addition to these values, the twenty-fifth, fiftieth, and seventy-fifth percentile values are calculated as well. All these values will be printed to the screen for the user. As was the case for the string variable, the user will be asked if they want a graph. Graphing is described in section A.5. Unlike for string parameters, there is no option to groupby the numeric datatype. This was due to the large number of different values that would be given for a group by with numeric parameters and there would not be much value added.

Boolean parameters are the simplest of the parameters. *Analysisoperations* will provide an overall count of the number of rows, a count of the true values, and a count of the false values for the Boolean parameter. One additional count that is also provided is the number of null values in the DataFrame for the Boolean parameter. This count shows how many rows had no value for the Boolean parameter in the DataFrame. The Spark describe function is not used on a Boolean parameter. It would only return the summary column. Figure 13 shows an example of the printout of the describe function. There would not be a column or any values relating to the Boolean parameter in the display. The counts of the values for Boolean parameters were accomplished using a group by SQL statement. The program will group by the Boolean parameter and count the number of true, false and null values for the parameter.

A.6   Plot of a Graph or Chart

The display of a graph or chart was accomplished using the pandas, numpy and matplotlib python libraries. In order to create a graph, the program converts the spark DataFrame to a pandas DataFrame. It then uses the matplotlib and pandas libraries to create and display the graph or graphs. The plotting of a graph or chart is accomplished in the GraphIt file. The main functions in this file that are utilized to provide a chart are the, *graphInd*, *graphIt, and graphGrouped*.

a. graphInd Function

The *graphInd* function is used to display graphs when analyzing an individual aircraft. The *graphInd* function takes in a DataFrame and a list of parameters. For an individual aircraft the parameters have been limited to "Icao," "Spd," "Alt," "Lat," "Long," and "PosTime." The function will give the user the option of displaying one of three graphs or all of them. The graphs available to view are line graphs of "Spd vs "Time," "Alt vs Time" and a scatter plot of "Spd" vs "Alt." Figure 19 shows an example of the three available graphs. When plotting time, the parameter "PosTime" is converted from epoch time to a spark TimeStamp containing the DD HH:MM. This conversion is required to allow matplotlib to display the date and time in that makes sense visually. If the epoch time was displayed it would be a large number that corresponds to the number of seconds since midnight January 1, 1970.
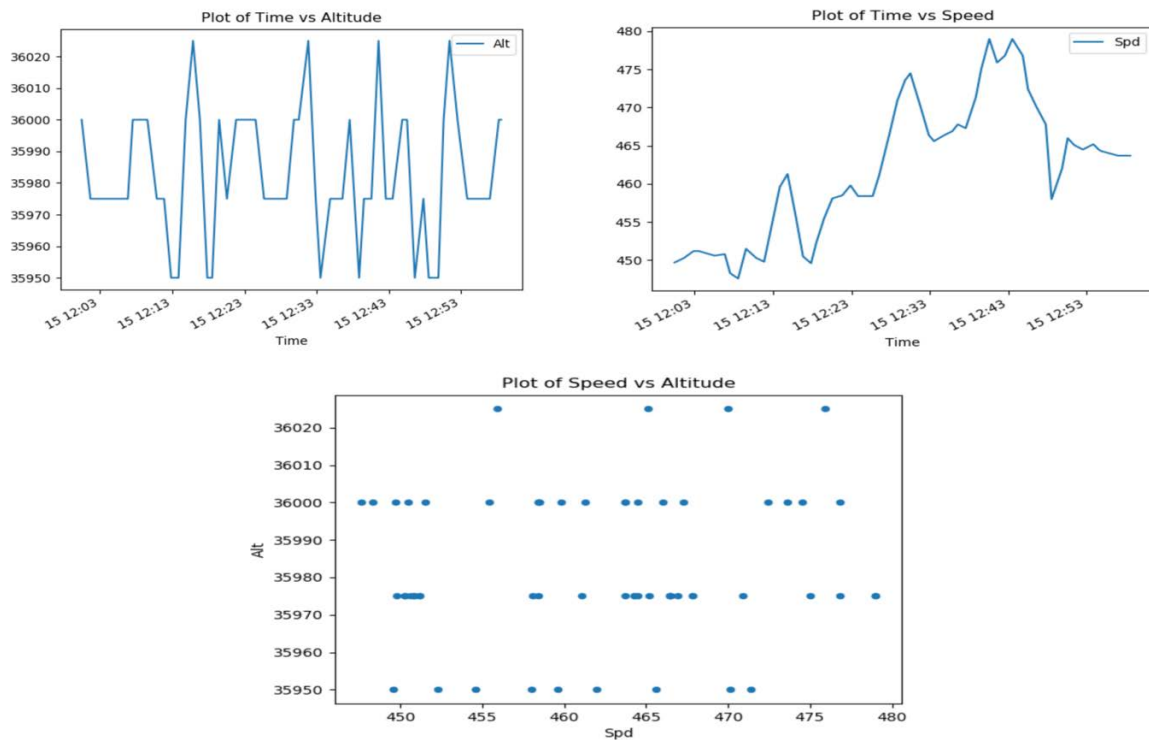

Figure 19.   Example graphs available from GraphInd

b. graphGrouped Function

GraphGrouped is a function in the GraphIt file that is designed to provide a bar graph/histogram for a DataFrame that has been grouped by a parameter or parameters and has a count column. With data sets as large as the ADSB data set there will be a large number of different groupings. Oftentimes, there will be too many to display on any single graph or histogram to be able to discern any sort of information from it. If there are over 120 different groupings in the DataFrame, the program will allow the user to specify a customized number to display and it will only display the largest counts based on the customized number. The number 120 was chosen because more than 120 groupings caused the graph to become unreadable. For grouped data plots, the user will also enter a title for the plot.

c. graphIt Function

The graphIt function is the main workhorse plotting function for the program. This function takes in a DataFrame and allows the user to select a type of graph they would like to display. When the program is going through each individual parameter during analysis operations, even though it asks if the user wants a plot using that parameter, the user will be allowed to select from any of the parameters that are a part of the DataFrame. This may cause some issues with the graphing function as it relies on certain parameters such as "Icao" to be available to conduct certain operations. For some of these problem cases, "Icao" has been kept in the DataFrame, even though not specifically asked for by the user. Available plotting/graphing options are a bar chart, a scatter plot and a boxplot.

(1) Bar Chart

If the bar chart/histogram option is chosen by the user, the program will give the user a list of the available parameters for the x-axis from which to make the plot. The program will then provide different options based on the type of the parameter chosen as well as based on the actual parameter chosen. If the parameter chosen is a Boolean, or a String, the only allowed plots are a histogram of values. The program will then further differentiate between the different types of string or Boolean parameters that are in the ADSB data.

If the parameter is "To" or "From," the program will ask if the user wants the count of the number of flights going to or from the airports in the DataFrame. If the user answers yes, then it will provide the counts of the number of flights going to or coming from different airports based upon the parameter "To" or "From." It accomplishes this by grouping first by "Icao" and then grouping by "To" or "From," whichever parameter was selected.

If the answer is no, then the program will group by "To" or "From." The program will provide a warning to the user that this information may not make much sense. This is because the grouping will be the number of flights going to or from an airport times the number of minutes that each flight was up. As was the case in the grouped data, if there will be more than 120 items plotted on the bar chart, the program will ask the user if they want to reduce the number of items to a customized number and display the values with the highest counts. The user will have an option to specify a title for the graph, but the x-axis label will be provided based upon the user's answer to the number of flights question.

If the parameter is "Mil," the program will provide a bar graph of the number of military aircraft vs the number of non-military aircraft. It does this by grouping by "Icao" and "Mil" and then by "Mil" again.

If the parameter is "Cou," the program will provide a bar graph for the number of flights for each country in the DataFrame. This is accomplished with a groupby "Cou" and "Icao" and then a second group by "Cou" again. This is so it counts only the flights for each country and not the number of flights combined with how long each flight was up.

If the parameter for the bar chart is "Icao," the bar chart will provide a bar chart for the number of minutes the flight was seen in during the time frame. This is a grouping by "Icao" which will be the number of instances that the specific "Icao" shows up in the files. Each instance will correspond to a single minute the aircraft was operational.

If the parameter is a numeric parameter such as, "Lat," "Long," "Spd," or Alt," then there are four different options for a bar chart. The available options are:

1. A histogram of all values (Group by parameter). This provides the number of instances of that parameter.

30

2.	A histogram of all values (Group by "Icao" and the parameter). This provides the number of instances of that parameter for each flight.

3.	A histogram using bins of all values (Group by parameter). This provides a histogram but with bins that provide for a range of values vice single values for each instance.

4.	A histogram using bins of all values (Group by "Icao" and the parameter). This provides a histogram of bins that provide for a range of values for each flight vice single values.

Option one will group by the parameter whereas option two will group by the parameter and "Icao." These different options will provide for the number of instances of a parameter or provide the number of instances for each flight.

(2)	Graphing with Bins

Options three and four will provide a graph with bins. The program will provide the user with the minimum and maximum value for the parameters. It will then ask the user to provide a minimum and maximum value for analysis. After the user provides a minimum and a maximum value, the system will ask the user to specify either a bin size or number of bins for graphing. If bin size is selected the user will specify the size of the bins desired. The program will compute the number of bins based on the minimum values, the maximum value and the bin size. The number of bars for the histogram will be the number of bins computed and the values for each bin will be based on the values from the DataFrame. If the user selects option for specifying the number of bins, then the program will compute the bin size using the number of bins desired, the minimum and the maximum values and provide a histogram. A histogram example from the prototype is shown in Figure 20.
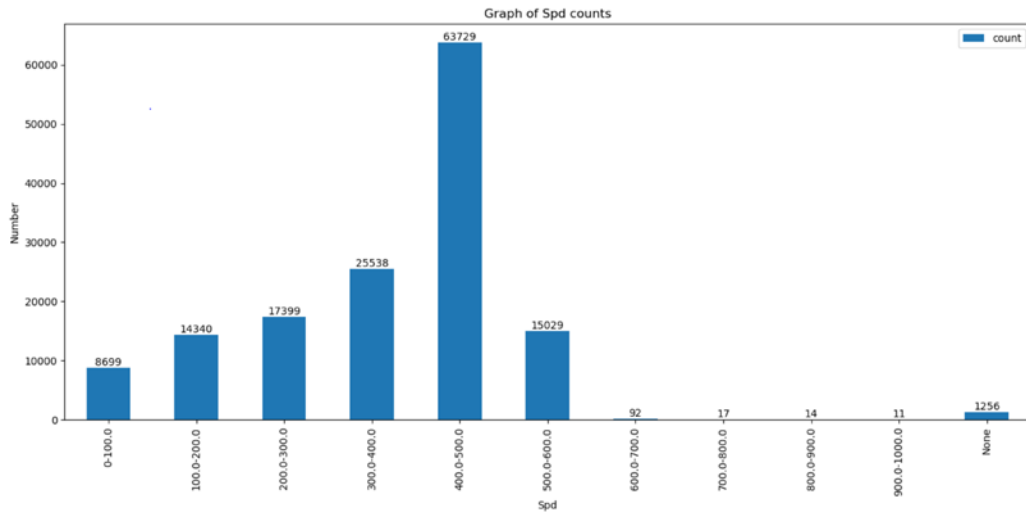
Figure 20.　Histogram example with Bins

To actually create the bins, the program creates a new column in the DataFrame, and it creates a list of tuples corresponding to the bin ranges for the histogram. Each tuple contains the minimum value for the bin, the maximum value for the bin and then a string corresponding to the range of the bin. The function *addSpdAlt* and the user defined function *makesspdaltcol* takes the values corresponding to the parameter being graphed and compares them to the maximum and minimum values for each bin. A string value, corresponding to the bin the value belongs to is then added to the new column in the DataFrame. The new column is then grouped and counted to provide the number of values that fall into each bin.

(3)　Scatter Plot

A scatter plot is available for parameters that have numeric values. For a scatter plot the user will provide a parameter for the x-axis and another for the y-axis of the plot. Only numeric parameters from the DataFrame can be used for a scatter plot. The pandas function *pandasd.df.plot(kind = 'scatter')* function is used to create the actual plot. An example scatter plot of "Spd" versus "Alt" is provided in Figure 21.
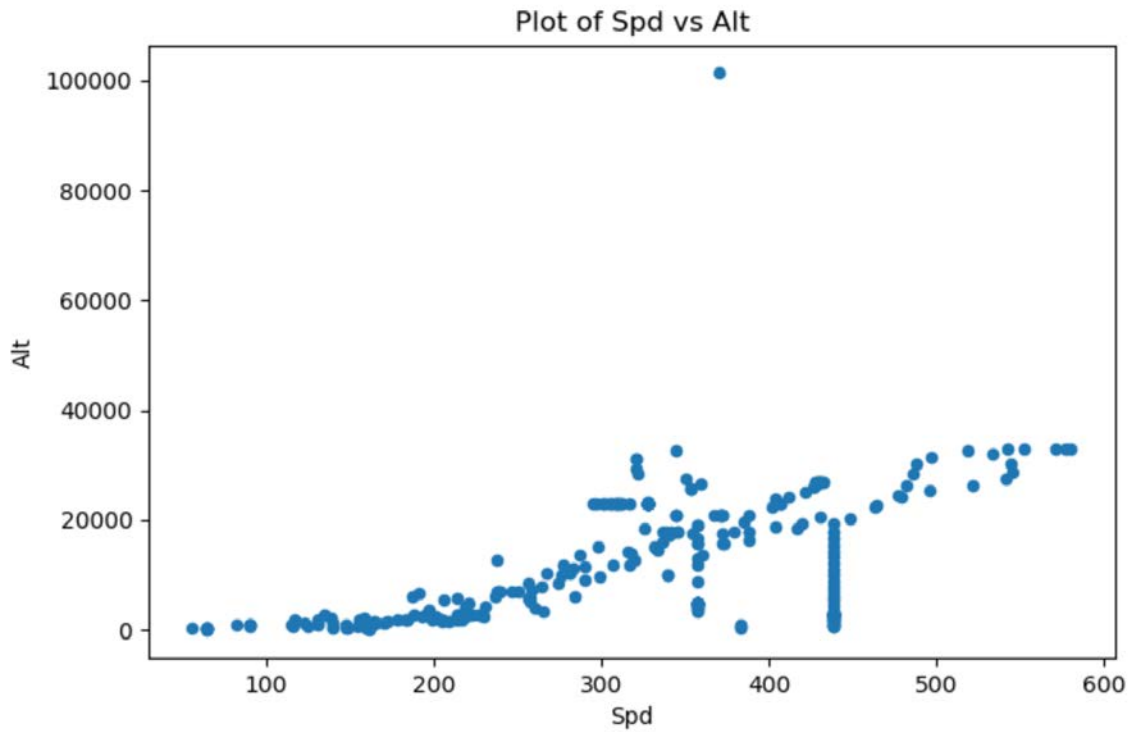
Figure 21.   Example Scatter plot of "Spd" vs "Alt" for US. Navy
Aircraft in North America

(4)   Box Plot

The box plot is the last plotting option available for visualizing numeric data. The *pandasdf.boxplot*function will return a boxplot for each individual parameter in the DataFrame. An example Boxplot for the parameter "Spd" is shown in Figure 22.
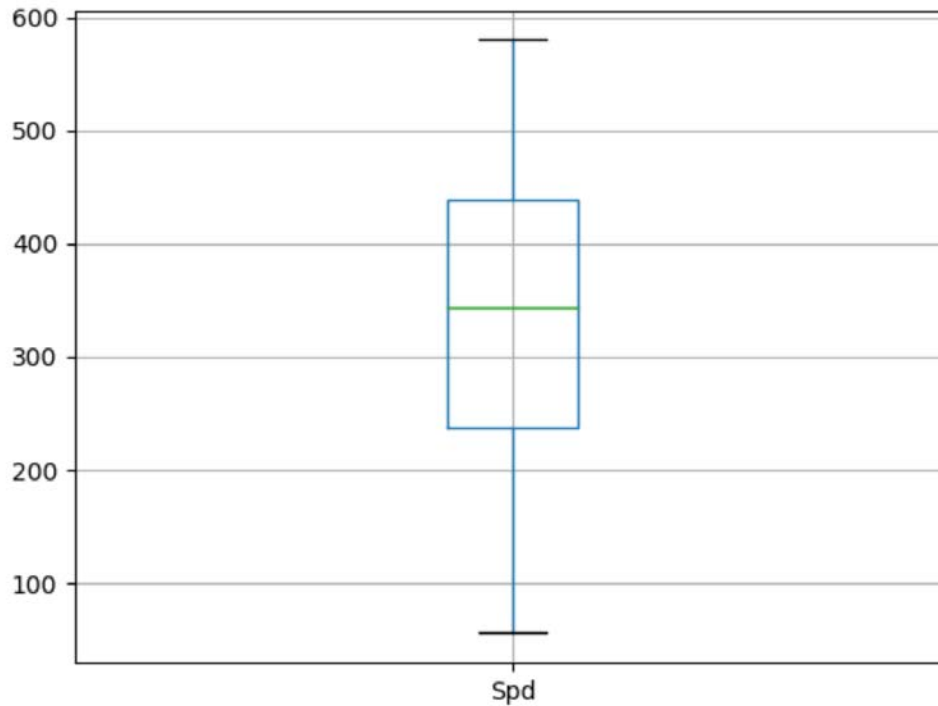
Figure 22.   Example box plot for the parameter "Spd"

A.7   Post Analysis

After the program has gone through all the parameters selected by the user, the program will ask a few questions. The first question is if the user wants to conduct any more analysis on this time frame. If the user wants to conduct more analysis on this time frame, the program will return to the filter stage and ask the user for the filters they want to apply to the time frame.

If the user is finished conducting analysis on the time frame, the program will ask if the user wants to save the DataFrame as a parquet. If the user wants to save as a parquet, then it is currently saved to a hardcoded location provided by the program. The program will then ask if the user is done conducting analysis. If the answer is yes, the program shuts down. If not, the program then returns to the very beginning and asks if the user wants to load a parquet.

B.   MACHINE LEARNING—LOGISTIC REGRESSION

A simple example of machine learning is provided in the LogisticRegression file. In this code, logistic regression is run on the following parameters: "Alt," "Lat," "Long," "Spd," and "Mil." The code is not meant to be fully operational. Rather, it is provided here as a simple illustration on how to run one of Spark's machine learning algorithms from its machine learning library.

The program starts by creating a Spark session and then a DataFrame with the parameters "Alt," "Lat," "Long," "Spd," and "Mil." A new column named "label" is created in the DataFrame for use in the logistic regression. The "label" column is created with the Spark *withColumn* function and is used for the logistic regression label of the data that the logistic regression is trying to predict. The 'label' must be in a specific data format where the values are either 1.0 or 0.0 or the Spark *LogisiticRegression* function will give an error. To do this a binary mapping was used to map the values from the "Mil" column which were in a true/false format to the 'label' column. The binary mapping performed the following mapping. "Yes" or True to are mapped to 1.0 and values that are "No" or False are mapped to 0.0.

The next important step in the logistic regression was to create a "features" column. In order to do this, the *VectorAssembler* transformer from the Spark machine learning library was used. This "transformer combines a list of columns into a single vector column. The *VectorAssemble*r accepts the following input column types: all numeric types, Boolean type, and vector type" (Apache Spark, MLib: Main Guide 2018). The output will be a vector consisting of the values from the input columns in the order listed as inputCols for the function. This vector is required by the *LogisticRegression* function. Example code is shown in Figure 23. The df1 is a DataFrame that is combined with the output from the *VectorAssembler* function.

```
df1 = df.na.drop()

assembler = VectorAssembler(
                inputCols=["Lat","Long", "Spd", "Alt", 'Species'],
                outputCol="features")

output = assembler.transform(df1)
```

Figure 23.   *VectorAssembler* function example

The Logistic Regression model is then created and fit to the data provided. The data available concerning the logistic regression are the following:

- Coefficients for each variable

- Intercept for the regression

- ROC or how the regression predicts.

- False Positive Rate

- True Positive Rate

These values can also be graphed using numpy and matplotlib. An example graph provided by the program showing the ROC Curve is shown in Figure 24.
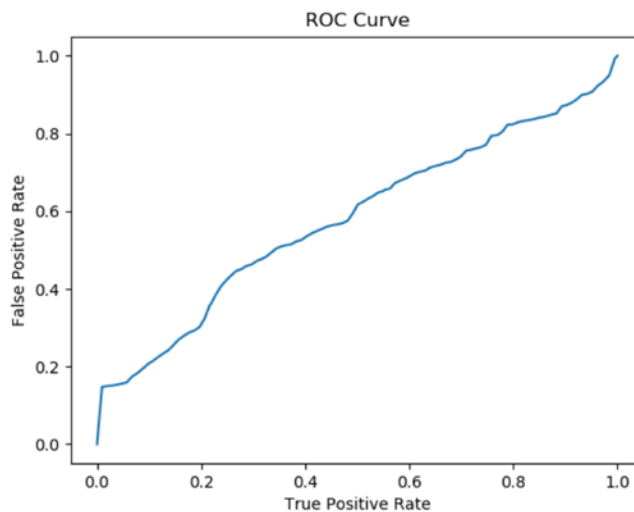

Figure 24.   ROC Curve example

IV.   REVIEW AND ANALYSIS

The prototype developed for this thesis succeeds in hiding the underlying Spark system from the end users. In this regard we achieved the primary objective of the thesis. However, for it to become a practical system, several aspects of the prototype need to be extended. We will address how the prototype can be extended in this chapter. In addition, we will describe some of the difficulties we encountered while developing the prototype.

A.   COMMAND LINE INTERFACE

The prototype was developed using a command line interface or CLI. This interface was chosen mainly because of time constraints. Other factors that played into the choice however were, the simplicity of a CLI for a user and also the ease with which a CLI can be updated in the future to provide additional features. A CLI was sufficient for hiding the underlying aspects of Spark from a data analyst.

The CLI used in this program is simple in that it gives the user a list of options to choose from or requests a specific input. The lists of options are hardcoded, in the case of the filters. Some other lists of options are based on the parameters in the DataFrame being analyzed, or the filter choices the user has made and these choices will determine the actions the user can take during analysis. This setup allows a user to concentrate on the data analysis and building the DataFrame rather than memorizing the commands required to make the program work.

While a CLI may take longer than a graphical user interface, or GUI, for a user to become familiar with, once the user has used the program several times, they will be able to easily and quickly transition through the program and probably in a more efficient manner than using a GUI. In addition, updates to the program will not change the format of the CLI prototype and will have limited impact on the navigation of the program unlike in a GUI. In a GUI making small changes can require significant structural changes and design changes to the program.

Additionally, adding new features for or requirements required by a subject matter expert, is very easy in the current format. The prototype was developed with this capability in mind. For example, the main filters of the program, shown in Figure 25 can be updated by adding additional numbers such as a filter number eight. Filter number eight could be type of filter function or this spot could where machine learning options could be implemented in the program.

```
Select the ADSB data you want to analyze(Multiple Choices will be possible in some cases):
1:  Individual Aircraft
2:  Region Airspace
3:  Specific Airline
4:  Military Aircraft
5:  Origin Airport
6:  Destination Aiport
7:  No Filiter(All Data)
Please make a selection.  Press 0 to stop selecting options
```

Figure 25.   Program filters

The CLI was sufficient for this prototype because it allowed for the hiding Spark from the user. CLI also had one other beneficial side effect in that Spark has a very useful function for the display of a DataFrame on the command line. The Spark function *df.show,* where df is a Spark DataFrame displays the DataFrame in a tabular format on the display. This command can be utilized to allow the user to customize the number of rows from the DataFrame to be displayed.

B.   MAKING IT A PRACTICAL SYSTEM

The prototype, while adequate to show that Spark functionality can be provided to a data analyst while hiding the underlying system, can be expanded to make it a more practical system. Many aspects of data analysis have been implemented in the program to include filtering, getting the mean, minimum, maximum, percentiles, counting, histograms, bar charts, scatterplots, and box plots. One recommendation is that even though the subject matter experts are not coding the program, they need to be deeply involved from the beginning to ensure that the desired filters, analysis tools, options and displays are built into the program.

In order to turn the prototype into a more practical system, the following implementations into the prototype could be performed.

B.1   Machine Learning Algorithms

The machine learning script included with the prototype serves as an example of logistic regression. This feature could be further implemented into the prototype. Additionally, other machine learning algorithms could be implemented into the program to include, linear regression, decision trees, random forest, multi class classification as well as clustering.

B.2   Graphical Users Interface

A graphical user interface or GUI could also make the prototype more practical in nature. GUIs tend to me more intuitive for users and it would allow users to pick up on functionality in the program much faster. Additionally, a GUI would allow the user to select various options at one time, similar to choosing items from a dinner menu, instead of having to flow through a program to get to a certain point. Another more practical aspect of a GUI is it could allow the user to maintain a better knowledge of the status of the DataFrame being operated on to include the parameters chosen, what filters have been applied and any other status the user would like to keep abreast of. A CLI just does not lend itself to this functionality. In a CLI the user would see pages of data and would have to scroll through it to find the information they were looking for.

B.3   Graphing

Scatter plots, box plots and histograms have been included in the functionality of this program to show that they can be used with Spark. They however, can be fine-tuned. In the prototype when graphing histograms, the user may be asked if they want to only look at a custom number of values for the histogram because the graphing of values over about 100-120 values results in a graph that is unreadable. The program as it is currently implemented will only graph the highest count values that the user chooses. Giving the user the choice of what they want to see from the histogram would make it more practical and functional for the user. For example, if an analyst wants to view the tail of the

39

histogram they would have the choice of viewing the values with the lowest counts from the histograms vice only the values with the highest counts. One last way to make the prototype more practical in regard graphing would be to graph the histogram over a series of graphs, where each graph would have a certain amount of values on it so that it remained readable. This way the user would still be able to view the data in a readable manner and also have all the data plotted.

B.4   Correlation

Correlation is a data analysis tool for determining the closeness of the relationship between two variables. Good correlation analysis will give an analyst a better understanding of the data they are analyzing. This would be beneficial when building models.

C.   LIMITATIONS/CHALLENGES OF SPARK

Python and Apache Spark 2.0 were a good combination for building the prototype. Python was selected because of the various Python tools and libraries available for making the backbone of the prototype. Spark 2.0 was chosen because of its DataFrame capability, its library of analysis tools, its machine learning library, and because of the speed with which it can process large amounts of data. However, they do provide some limitations and challenges. These challenges were the difficult parts of building parts of the prototype.

C.1   Spark Documentation

The Spark Documentation is extensive and examples are provided but they are not very easy to understand at times. For example, the documentation makes it seem very straightforward on how to implement machine learning techniques using the Spark machine learning libraries. Many books on the subject seem to show the same thing. The process however is not very straight forward. An example can be seen when performing logistic regression. When using the Spark command *fit,* which fits a logistic regression model to a dataset, the documentation indicates that the input is a dataset which is an instance of a Spark DataFrame. The only issue is that a normal DataFrame cannot be used as the input. It first must be manipulated. Spark requires that the DataFrame have a column

labeled "label and a column labeled "features." All the data in the "label" column must be a binary variable where the values are "1.0" or "0.0." Values of true and false will not work for this column. Each row in the "features" column contains a vector of all the data in that row pertaining to the features the user wants to use. The regression will not work without these two columns included or formatted correctly. As mentioned above the formatting requirement is not entirely clear, in the documentation or in the examples. Examples of how to convert a binary variable using a user defined function for the "label" column as well as creating the vector for the features column and adding them to the DataFrame using the Spark *VectorAssembler* function can be seen in the Logistic regression file.

C.2   Graphing with Spark

A data analyst when analyzing their data would want to be able to view their data in some sort of graph or chart. Ideally, the user would want to view the data with a boxplot, histogram, scatter plot or some other plot of the data. This would allow the user to visually examine the data for any issues or trends in the data. This is especially critical when analyzing data and looking at the underlying assumptions of the tests that are being conducted such as when conducting an ANOVA or a linear regression. If the underlying assumptions for the tests are not met, then false conclusions will be made in regard to the data.

The inability to display data in a graph format is a limitation of Spark. Spark lacks any native capability for creating a chart or graph for display of the data. A work around for this limitation, used in the prototype, is to use several other python libraries to create a chart or plot. The libraries required for this are Numpy, Pandas and Matplotlib. A generic way to create a chart is to take the Spark DataFrame and convert it to a Pandas DataFrame. Use Numpy to assist in arranging the data then using Matplotlib to provide a visual plot of the data from the Pandas DataFrame. A specific example of this is in the GraphIt program file.

41

The one drawback of this option is that converting the Spark DataFrame to a Pandas DataFrame can be costly and time consuming. Additionally, any work on the data should actually be done in Spark and not Pandas to take advantage of Spark's speed at data processing.

C.3   Loading Spark on a Windows Machine

The loading and installation of Spark on a machine running the Windows operating system can be quite challenging. It is not straight forward. Mr. Frank Kane of Sundog Software in his Udemy course, *"Taming Big Data with Apache Spark and Python—Hands On!"* provides a video walk though of step by step instructions on how to set up Spark on a windows machine. The instructions are also available on his website at *https://sundog-education.com/spark-python/* and are shown in the Figure 26 (Kane 2018). Instructions for MAC or Linux operating systems are also at the same website.

One thing that cannot be reiterated enough when installing and using Spark on a windows machine is to use JAVA 8. JAVA 9 and above are currently not compatible with Spark and will create massive headaches for the developer. This restriction or requirement is not provided during the Spark download and is not readily apparent on the Spark website.

1. Install a JDK (Java Development Kit) from http://www.oracle.com/technetwork/java/javase/downloads/index.html . **You must install the JDK into a path with no spaces**, for example c:\jdk. Be sure to change the default location for the installation! **DO NOT INSTALL JAVA 9 or 10 – INSTALL JAVA 8**. Spark is not compatible with Java 9 or newer.

2. Download a **pre-built** version of Apache Spark from https://spark.apache.org/downloads.html

3. If necessary, download and install WinRAR so you can extract the .tgz file you downloaded. http://www.rarlab.com/download.htm

4. Extract the Spark archive, and copy its **contents** into **C:\spark** after creating that directory. You should end up with directories like c:\spark\bin, c:\spark\conf, etc.

5. Download winutils.exe from https://sundog–s3.amazonaws.com/winutils.exe and move it into a **C:\winutils\bin** folder that you've created. (note, this is a 64-bit application. If you are on a 32-bit version of Windows, you'll need to search for a 32-bit build of winutils.exe for Hadoop.)

6. Create a **c:\tmp\hive** directory, and cd into c:\winutils\bin, and run **winutils.exe chmod 777 c:\tmp\hive**

7. Open the the **c:\spark\conf** folder, and make sure "File Name Extensions" is checked in the "view" tab of Windows Explorer. Rename the log4j.properties.template file to log4j.properties. Edit this file (using Wordpad or something similar) and change the error level from INFO to ERROR for log4j.rootCategory

8. Right-click your Windows menu, select Control Panel, System and Security, and then System. Click on "Advanced System Settings" and then the "Environment Variables" button.

9. Add the following new USER variables:

   1. SPARK_HOME c:\spark
   2. JAVA_HOME (the path you installed the JDK to in step 1, for example C:\JDK)
   3. HADOOP HOME c:\winutils

10. Add the following paths to your PATH user variable:

%SPARK_HOME%\bin

%JAVA_HOME%\bin

10. Close the environment variable screen and the control panels.

11. Install the latest **Enthought Canopy for Python 3.5** from https://store.enthought.com/downloads/#default Don't install a Python 2.7 version!

12. Test it out!

   1. Open up Canopy and select "Canopy Command Prompt" from the Tools menu.
   2. Enter **cd c:\spark** and then **dir** to get a directory listing.
   3. Look for a text file we can play with, like README.md or CHANGES.txt
   4. Enter **pyspark**
   5. At this point you should have a >>> prompt. If not, double check the steps above.
   6. Enter **rdd = sc.textFile("README.md")** (or whatever text file you've found) Enter **rdd.count()**
   7. You should get a count of the number of lines in that file! Congratulations, you just ran your first Spark program!
   8. Enter **quit()** to exit the spark shell, and close the console window
   9. You've got everything set up! Hooray!

Figure 26.　　Instructions on how to install Spark on a Windows machine

C.4   User Defined Functions

User defined functions while not a limitation in Spark do provide a challenge. User defined functions, are in fact, a powerful tool for custom manipulations and transformations of data. They allow for user customization and manipulation of columns both to and from the DataFrame. When writing a user defined function, the first thing to do is to write the actual function. Next, the function must be "registered with Spark so that it can be used on all of the worker machines" (Chambers and Zaharia 2018, 112). An example user defined function and its registration from the prototype is shown in Figure 27.

```python
def assignContinent(lat, long):

    NorthAmerica = [(11, -180), (90, -180), (90, -25), (11, -25), "North America"]
    SouthAmerica = [(-90, -121), (11, -121), (11, -25), (-90, -25), "South America"]
    Europe = [(36, -25), (90, -25), (90, 37), (36, 37), "Europe"]
    Africa = [(-90, -25), (36, -25), (36, 37), (15, 37), (15, 60), (-90, 60), "Africa"]
    Asia = [(15, 37), (90, 37), (90, 180), (0, 180), (0, 60), (15, 60), "Asia"]
    Australia = [(-90, 60), (0, 60), (0, 180), (-90, 180), "Australia"]

    world = [NorthAmerica, SouthAmerica, Europe, Africa, Asia, Australia]

    for area in world:

        location = Inpolygon((lat,long), area)

        if type(location) == str:
            return location

continentudf = udf(assignContinent)
```

Figure 27.   Example of a user defined function

The *def assignContinent()* function shown above is declared like a normal definition of a function in Python. The registration of this function is the line *continentudf = udf(assignContinent).* This function will take a latitude and longitude and return a location. Without the registration line, *continentudf = udf(assignContinent)*, Spark will be unable to use the function on the DataFrame as all the worker machines/nodes will not have it for use. The line *df = df.withColumn('Continent', continentudf(col('Lat'), col('Long')))* is an example of using the user defined function. Notice how the function used is the

registered function, not the actual declared function. For this example, a column "Continent" is added to the DataFrame df using the user defined function *continentudf* which is an object of the actual function. The inputs to the *continentudf* function are the columns 'Lat' and 'Long' from the DataFrame df.

An even more difficult and complicated use of user defined function occurs when a user defined function uses as inputs columns from a DataFrame and variables not in the DataFrame. The best way to show this is with an example as shown in Figure 28. The *addSpdAlt()* function is designed to return a string that corresponds to a certain interval in which a value is located. For example, if the value is 5, and the intervals were in groups of 10, then it would return the interval "1-10." The function takes a list of lists of intervals(xs), a maximum value(maxv), a minimum value(minv) and a value(from the DataFrame). This function returns None or string corresponding to the interval that the value was found in. The list of intervals, xs, has the following architecture: xs[i][0] is the minimum value in the interval, xs[i][1] is the maximum value in the interval and xs[i][2] is the string corresponding to the interval which is what would get returned.

```
# Used for making the bins
def addSpdAlt(xs, maxv, minv, value):
    for i in range(len(xs)):

        if value == None:
            return   # "None or Null"

        elif value > xs[i][0] and value <= xs[i][1]:
            return xs[i][2]


# User defined function for adding bins
def makespdaltcol(xs, maxv, minv):
    return udf(lambda c: addSpdAlt(xs, maxv, minv, c))
```

Figure 28.   A more complex user defined function from the prototype

The function call for the user defined function is the following: *df1 = df.withColumn(x + " Intervals," makespdaltcol(xs, maxv, minv)(x))*. The part that is not straight forward is why the (x) is not with the rest of the variables, as in a normal python function call, even though the function was declared that way. In this case the (x)

45

corresponds to the column that the variable value comes from during the function call. Normally in Python it would be included in the parentheses with the rest of the variables. If that is done, the function will not run. It must have its own parentheses.

C.5   File Input

Another small challenge Spark posed in the developing the prototype pertained to the input of the JSON files. While the typical file input is easy, the way the data files were stored presented a small challenge for reading in many files. Spark has the ability to read in a JSON file by its location or by a directory of JSON files. It has the capability to input a list of file locations. Spark will not, however, directly read in a directory that contains a directory of files. Examples of the various read ins are shown below:

Read in of a single file:

df = spark.read.json("C:\\2016_11\\2016-11-15\\2016-11-15-0002Z.json")

Read in of a directory where the directory 2016-11-15 is a directory of JSON files:

df = spark.read.json("C:\\2016_11\\2016-11-15\\")

Read in of a list(xs) of file locations

xs = ["C:\\2016_11\\2016-11-15\\2016-11-15-0000Z.json",

"C:\\2016_11\\2016-11-15\\2016-11-15-0001Z.json"]

df = spark.read.json(xs)

The examples above will all work. Shown below is an attempt at reading in all the JSON files that are in the sub-directories of the directory 2016_11. This will not work as the 2016_11 directory contains directories of JSON files and not the actual JSON files themselves. Spark will not automatically dig down into the sub-directories.

df = spark.read.json("C:\\2016_11\\")

The fix of this is to use a wildcard. Spark will now read JSON files from the sub-directories when a * wildcard is added to the end of the directory.

df = spark.read.json("C:\\2016_11*")

46

Using the command above, now all the JSON files contained in the directories under 2016_11 will be read in and added to the DataFrame df.

D. INVOLVEMENT OF SUBJECT MATTER EXPERTS

While the goal of the program is to shield subject matter experts form the programming and algorithms, subject matter experts must be involved in the development of the program. They should be involved early and often. Their involvement is key to ensuring that the correct characteristics, capabilities, displays and any other desired functions are contained in the program. They know the data the program is designed to analyze better than the programmer. This is very important when dealing with string data type variables.

In the ADSB data set, two parameters that were string data types were "Op' and "Cou." Knowing the different values and types of strings that are contained in these two columns was paramount in the development of the filters in the program that works on these two columns. In the "Op" column, which corresponds to the airplane operator, the same operator may be named several different ways. For example, in a quick study of the data of the "Op" column, the value for American Airlines, could be "American Airlines," "American Airlines—Fort Worth," "American Airlines—DFW." These all appear to be for the same airline but in actuality they may not be. There could be a desire to be able to have them separated out. If these were branches of American Airlines, then multiple filters may be needed to filter for the Airlines as whole, all American Airlines, or to filter by one of its sub branches, American Airlines—DFW. When collating the statistics, the programmer needs to know how these situations should be handled and a subject matter expert would provide this guidance. This is a simple but small illustration of the issue. If a situation like this is not understood or handled correctly then false numbers, analysis and conclusions could be gleaned. This could have a dramatic effect on a Commander's decision.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION AND FUTURE WORK

A. CONCLUSION

In this thesis, we developed a prototype that provides data analytics tools to subject matter experts. Unlike other currently available data analytics tools that require the users to know the programming languages such as R or Python and often the underlying data management system, our prototype removes such burden from the users.

We used Spark 2.0 as the underlying system for our prototype. The users of the prototype are completely shielded from the complexity of Spark. The details of the prototype and the development issues we encountered while constructing the prototype are described in Chapter III and Chapter IV, respectively. The key points we learned from using the Spark system is that it is very capable for the development of an analysis program. The APIs and libraries are very good for and fairly easy to use. There is however, a lack of an inherent graphing capability and specifics in the documentation leave a lot to be desired. Using the Python matplotlib and pandas library can cover the graphing shortfall but using matplotlib feels kind of clunky.

Spark is a fast data processing engine that is fairly easy to use. It has a wide range of capabilities including functions, processes, algorithms and data analysis functions, which can be accomplished through the use of a single line of code or a few lines of code. This is in contrast to the many lines of code it would take to accomplish the same task solely using Python or some other programming language. The ability to develop and utilize user defined functions, while not the most straight forward, is very powerful in regard to data transformation and is invaluable to data analysis. Additionally, the DataFrame abstract is very useful and eases the task of data analysis. As mentioned previously a built in graphing capability for use with Python would be very beneficial. In summary, spark has the capability to provide fast data analysis for users.

Spark's capabilities have the ability to assist users, watch standers, decision makers and commanders in understanding the current situation in regard to the data analysis and can aide in making decisions in a timelier manner.

The prototype developed can be extended and improved in several aspects. We describe the possible extensions and improvements in the next section.

B.   FUTURE WORK

The prototype for this research worked well but as with most software development, it could and should be handled in iterations. After building parts of it, there are ways that it could be improved upon to include program flow, user options, and additional capabilities. The main goal of this research was to show that an analysis program using Spark could be made that allowed for data analysis while hiding the underlying Spark system from the user. In that way it was a success but there are several ways that the research can be furthered.

The first recommendation for future work is to examine the development of the program using a graphical users interface or GUI. A GUI has a quicker learning curve than a command line interface, or CLI, and can be more intuitive for a user. The GUI would also give the user the advantage of knowing the current status of the program, what filters have been selected and what parameters have been selected. Printing out the DataFrame to the screen on a CLI makes the screen cluttered and the forces the user has to scroll back through quite a bit of information. A display of selections could be used in the GUI to keep the user appraised of what has been selected and the current status. Additionally, part of the DataFrame could be shown to the user as part of the GUI for easy feedback to the user on operations applied.

A second recommendation for future work would be to implement the prototype using Spark and R called SparkR or as a combination of Spark, R and Python. The R programming language is designed for data analysis and has a very good statistics libraries including very good-looking plotting functions. One of the weaknesses of Spark with Python was its lack of built in plotting functions. Matplotlib was used but was a little clunky. R can also be used with Python by installing the rpy2 Python library. In order to use the ryp2 library the R programming language must also be installed. This setup would allow for the running of R scripts using Python. Alternatively, another option is to use the

rPython package with R which would enable the calling of Python scripts from R. SparkR also supports the Spark machine learning library

A third recommendation is to improve the filters using an input from the user as the filter instead of using a hard-coded list of options. An example of this would be to setup the filter for specific airline by having the user provide an airline name and the program would filter for that airline and possibly all like it in the DataFrame. An input of American Airlines would filter the DataFrame for all rows that have American Airlines in the "Op" column.

Another recommendation is to develop more machine learning scripts for use with the prototype. This would allow other machine learning techniques to be brought to bear on data analysis. Some other machine learning algorithms that could be included in the program are linear models, support vector machines, decision trees, naïve Bayes as well as k-means clustering to name a few.

Finally, Spark includes an API for streaming data. The ADSB data can be obtained in real-time as well. This would allow for near real time analysis of the ADSB data, allow for the testing of new data against current models that had already been built and allow for prompt decisions to be made. One possible use of the streaming feature would be to identify and flag anomalous aircraft from the ADSB data for follow up by the user or potentially a military response.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Apache Spark. 2018. *Apache Spark.* August 1. Accessed August 07, 2018.
https://spark.apache.org/.

Apache Spark, MLib: Main Guide. 2018. Accessed October 26, 2018.
https://spark.apache.org/docs/latest/ml-features.html#vectorassembler.

Business First Magazine. 2016. *Data Takes A Quantum Leap.* August 08. Accessed
August 08, 2018. https://www.quantium.com/wp-
content/uploads/2016/08/BFM_Quantium.pdf.

Chambers, Bill, and Matei Zaharia. 2018. *Spark: The Definitive Guide.* Sebastpol:
O'Reilly.

Databricks. 2018. *databricks.com.* August 1. Accessed August 07, 2018.
https://databricks.com/spark/about.

Franklin, W. Randolph. 2017. *Point Inclusion in Polygon Test.* April 25. Accessed
November 2, 2018. https://wrf.ecse.rpi.edu/Research/Short_Notes/pnpoly.html.

Kane, Frank. 2018. *Sundog Education.* Accessed 10 28, 2018. https://sundog-
education.com/spark-python/.

Xin, Reynold. 2014. *Databricks.* 11 5. Accessed 11 01, 2018.
https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-
large-scale-sorting.html.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1.   Defense Technical Information Center
      Ft. Belvoir, Virginia

2.   Dudley Knox Library
      Naval Postgraduate School
      Monterey, California