Storage Warriors

NoSQL Proof of Concept

Project Documentation

Version 1.0

10/29/2017

TEAM MEMBERS:

Abhinay Agrawal

Ambuj Nayan

Nitilaksha Halakatti

Rahul R Sharma

Table of Contents

1. Introduction

The goal of this project was to build an application which could ingest, store, analyze, and extract meaningful insights from two different massive data stores. The first of these sources was NOAA (National Oceanic and Atmospheric Administration) and it provided us with hourly synoptic weather observations from station networks around the world. The second data source was UBTS (US Bureau of Transportation Services) and it provided us with flight history and delays.

During preliminary data analysis, we found that the volume of data was very huge. The respective sizes of weather and flight datasets were 750 GB and 225 GB approximately. The huge data volume and the NoSQL DB use case pushed us towards building a scalable and distributed system to store and process this data.

We decided to use Apache HBase which is a database associated with the Hadoop ecosystem. It is a distributed and scalable column family data store. The datasets in their raw form were not conducive for analysis and needed considerable amount of pre-processing. Custom python scripts were used to pre-process the data. After completion of pre-processing, we needed a scalable and distributed process which could perform bulk upload to HBase. Apache Spark was a good fit here because of its unique in-memory processing capabilities which allow it to process large-scale data at very high speeds.

Per our use case, this application had to provide ease of access to its users. Since most of the users in the current world are already familiar with SQL and SQL reduces the amount of code that needs to be written, we decided to integrate HBase with Apache Phoenix. Phoenix is an open source SQL skin for HBase and saves us the hassle of writing application code to query data using HBase APIs. Integrating Apache Phoenix with HBase, and using it to query huge volume of data from HBase had its fair share of problems which are described in greater detail later in this document. For visualization, we selected Apache Zeppelin. It is a web-based notebook application that enables data-driven, interactive data analytics, and collaboration with SQL.

One of the biggest challenges of this project was to find correlation between weather and flight data. These two datasets had different granularities and finding a spatiotemporal relationship between these two proved to be the trickiest problem. Since flight datasets did not have latitude and longitude information, we had to mine the internet to get the coordinates of all US airports. The next step was to find the weather station closest to each airport while ensuring that the station was close enough to have an impact on airport operations. This was achieved by a Java program which calculated the geographical distance between an airport and all the weather stations and picked the weather station closest to the airport. We also had to make sure that the observed weather phenomenon was in the same time window as the flight operation.

Once the relationship between weather and flight was established, we used machine learning to predict whether forecasted weather information will have an impact on flight schedules.

2. Infrastructure Setup

2.1 Hardware specification

Since this project involved downloading and processing huge data files, we selected a computer with the following specification:

- Physical storage: 1 TB (approximately 800 GB free)
- Memory: 8 GB
- Processor: Intel i7 quad core
- Operating system: Ubuntu 16.04

2.2 Hadoop ecosystem setup

Instead of using any cloud service provider, we decided to invest time in learning the setup and configuration of Hadoop ecosystem. Also, most of the cloud service providers had high tariff for our storage requirements. Following are the steps:

- Installation of Hadoop involved downloading the distribution from Apache and modifying the configuration files as mentioned here.
- We downloaded HBase from Apache and configured it as per the guidelines provided here.
- Upon the completion of installation and configuration, the services related to DFS, YARN, and HBASE were started.

2.3 Apache Phoenix setup

Downloaded the installation tar and copied the phoenix server jar into HBase lib directory. Restarted HBase and after successful restart, we started the query server.

2.4 Apache Zeppelin setup

Followed the steps at this site to download and install Zeppelin. Configured Zeppelin as a thin client and used a JDBC connection to connect it to Apache Phoenix.

3. Requirement Analysis

The main requirement of this project is to provide the users with an easy interface to query and analyze large datasets. These datasets have weather and flight information. Users should be able to run analytical queries on these datasets and should be provided with a way to visualize the results.

The application should also provide an automated mechanism for data ingestion. Raw datasets are noisy, and the application should remove the noise from datasets before persisting them on a physical medium.

The system should be able to scale because these datasets will witness exponential growth. Though the current requirement is to support historical analysis of weather and flight data, the system design should be flexible enough to support transactional analysis in future.

A nice to have feature is to help users find correlation between weather and flight data. This enables the users to predict flight delays based on forecasted weather information by using machine learning algorithm. Following are the main business questions that this application should be able to answer.

3.1.1 Business question 1

For a given weather station and year, report the highest observed temperatures (degree centigrade) by month. Provide a mechanism to visualize this information.

3.1.2 Business question 2

For a given weather station and year, report the average observed temperatures (degree centigrade) by month. Provide a mechanism to visualize this information.

3.1.3 Business question 3

For a given weather station and year, report the lowest observed temperatures (degree centigrade) by month. Provide a mechanism to visualize this information.

3.1.4 Business question 4

For a given weather station and month, show the variation in mean temperature (degree centigrade) over a period of 20 years. Provide a mechanism to visualize this information.

3.1.5 Business question 5

Find the relation between temperature and elevation (altitude). Ideally as elevation increases, temperature decreases, and the rate of decrease is 6.5 degree centigrade for each 1 km of altitude change. Validate if the observed data provides support for this trend along with providing a mechanism for visualizing this information.

3.1.6 Business question 6

Report the count of those weather stations which have witnessed Hurricane force winds. Beaufort wind force scale is an empirical measure that relates wind speed to observed conditions at sea or on land. Hurricane force winds are categorized as those which have wind speeds greater than equal to 118 km/hr.

3.1.7 Business question 7

Report the count of those weather stations which have very low visibility. Areas with visibility of less than 100 metres (330 ft.) are usually reported as very low visibility areas.

3.1.8 Business question 8

Find the maximum observed flight delay (in minutes) in the entire dataset.

3.1.9 Business question 9

Find the average observed flight delay (in minutes) in the entire dataset.

3.1.10 Business question 10

Report the number of instances where flights have been delayed by more than 15 minutes.

3.1.11 Business question 11

Find out the correlation between flight and weather data. Note that there is a spatiotemporal relation between weather and flight data.

3.1.12 Business question 12

Use the correlation between weather and flight data to predict if a flight will be delayed based on forecasted weather metrics.

4. Data Sources

4.1 Weather Data

The source of weather data is NOAA and it provides hourly weather observations from varied sources. The observation sources include the World Meteorological Organization, Automated Surface Observing System, Automated Weather Observing Stations, and US Climate Reference Network. Some of the more important metrics included in the weather data are station name, date, time, source, latitude, longitude, elevation, wind, ceiling height dimension, visibility observation, air temperature, dew point, sea level pressure.

Weather data is available from year 1901 up to the current year. The volume of the entire dataset is approximately 750 GB and it includes reports from weather stations all over the world. The data is in CSV format and allows the number of columns to vary by row. In each row, columns are organized in the following three sections:

- Control data section
- Mandatory data section
- Additional data section

Each of these sections have multiple columns associated with them. Following section describes the metadata associated with raw data files.

4.1.1 Metadata for raw data

Control Data Section

POS: 1-6

FIXED-WEATHER-STATION USAF MASTER STATION CATALOG identifier
The identifier that represents a FIXED-WEATHER-STATION.
DOM: A general domain comprised of the characters in the ASCII character set.
COMMENT: This field includes all surface reporting stations, including ships, buoys, etc.

POS: 7-14

GEOPHYSICAL-POINT-OBSERVATION date
The date of a GEOPHYSICAL-POINT-OBSERVATION.
MIN: 00000101 MAX: 99991231
DOM: A general domain comprised of integer values 0-9 in the format YYYYMMDD. YYYY can be any positive integer value; MM is restricted to values 01-12; and DD is restricted to values 01-31.

POS: 15-18

GEOPHYSICAL-POINT-OBSERVATION time
The time of a GEOPHYSICAL-POINT-OBSERVATION based on
Coordinated Universal Time Code (UTC).
MIN: 0000 MAX: 2359
DOM: A general domain comprised of integer values 0-9 in the format HHMM. HH is restricted to values 00-23; MM is restricted to values 00-59.

POS: 19-19

GEOPHYSICAL-POINT-OBSERVATION data source flag
The flag of a GEOPHYSICAL-POINT-OBSERVATION showing the source or combination of sources used in creating the observation.
MIN: 1 MAX: Z
DOM: A general domain comprised of values 1-9 and A-N.
9 = Missing

POS: 20-25

GEOPHYSICAL-POINT-OBSERVATION latitude coordinate
The latitude coordinates of a GEOPHYSICAL-POINT-OBSERVATION where southern hemisphere is negative.
MIN: -90000 MAX: +90000
UNITS: Angular Degrees
SCALING FACTOR: 1000
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-).
+99999 = Missing

POS: 26-32

GEOPHYSICAL-POINT-OBSERVATION longitude coordinate
The longitude coordinates of a GEOPHYSICAL-POINT-OBSERVATION where values west from 000000 to 179999 are signed negative.
MIN: -179999 MAX: +180000 UNITS: Angular Degrees
SCALING FACTOR: 1000
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-).
+999999 = Missing

POS: 33-37

GEOPHYSICAL-POINT-OBSERVATION elevation dimension
The elevation of a GEOPHYSICAL-POINT-OBSERVATION relative to Mean Sea Level (MSL).
MIN: -0400 MAX: +8850 UNITS: Meters
SCALING FACTOR: 1
DOM: A general domain comprised of the numeric characters (0-9), a minus sign (-), and a plus sign (+).
+9999 = Missing

POS: 38-42

GEOPHYSICAL-REPORT-TYPE code
The code that denotes the type of geophysical surface observation.
DOM: A specific domain comprised of the characters in the ASCII character set.
99999 = Missing

POS: 43-47

FIXED-WEATHER-STATION call letter identifier
The identifier that represents the call letters assigned to a FIXED-WEATHER-STATION.
DOM: A general domain comprised of the characters in the ASCII character set.
99999 = Missing.

POS: 48-51

METEOROLOGICAL-POINT-OBSERVATION quality control process name
The name of the quality control process applied to a weather observation.
DOM: A general domain comprised of the ASCII character set.

Mandatory data section

NOTE: For quality codes, use this

0 = Passed gross limits check
1 = Passed all quality control checks
2 = Suspect
3 = Erroneous
4 = Passed gross limits check, data originate from an NCEI data source
5 = Passed all quality control checks, data originate from an NCEI data source
6 = Suspect, data originate from an NCEI data source
7 = Erroneous, data originate from an NCEI data source
9 = Passed gross limits check if element is present

POS: 52-54

WIND-OBSERVATION direction angle
The angle, measured in a clockwise direction, between true north and the direction from which the wind is blowing.
MIN: 001 MAX: 360 UNITS: Angular Degrees
SCALING FACTOR: 1
DOM: A general domain comprised of the numeric characters (0-9).
999 = Missing. If type code (below) = V, then 999 indicates variable wind direction.

POS: 55-55

WIND-OBSERVATION direction quality code
The code that denotes a quality status of a reported WIND-OBSERVATION direction angle.
DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 56-56

WIND-OBSERVATION type code
The code that denotes the character of the WIND-OBSERVATION.
DOM: A specific domain comprised of the characters in the ASCII character set.
A: Abridged Beaufort
B: Beaufort
C: Calm
H: 5-Minute Average Speed
N: Normal
R: 60-Minute Average Speed
Q: Squall
T: 180 Minute Average Speed
V: Variable
9 = Missing
NOTE: If a value of 9 appears with a wind speed of 0000, this indicates calm winds.

POS: 57-60

WIND-OBSERVATION speed rate
The rate of horizontal travel of air past a fixed point.
MIN: 0000 MAX: 0900 UNITS: meters per second
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9).
9999 = Missing.

POS: 61-61

WIND-OBSERVATION speed quality code
The code that denotes a quality status of a reported WIND-OBSERVATION speed rate.
DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 62-66

SKY-CONDITION-OBSERVATION ceiling height dimension
The height above ground level (AGL) of the lowest cloud or obscuring phenomena layer aloft with 5/8 or more summation total sky cover, which may be predominantly opaque, or the vertical visibility into a surface-based obstruction. Unlimited = 22000.
MIN: 00000 MAX: 22000 UNITS: Meters
SCALING FACTOR: 1
DOM: A general domain comprised of the numeric characters (0-9).
99999 = Missing.

POS: 67-67

SKY-CONDTION-OBSERVATION ceiling quality code
The code that denotes a quality status of a reported ceiling height dimension.
DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 68-68

SKY-CONDITION-OBSERVATION ceiling determination code
The code that denotes the method used to determine the ceiling.
DOM: A specific domain comprised of the characters in the ASCII character set.
A: Aircraft
B: Balloon
C: Statistically derived
D: Persistent ceiling (pre-1950 data)
E: Estimated
M: Measured
P: Precipitation ceiling (pre-1950 data)
R: Radar
S: ASOS augmented
U: Unknown ceiling (pre-1950 data)
V: Variable ceiling (pre-1950 data)
W: Obscured
9: Missing

POS: 69-69

SKY-CONDITION-OBSERVATION CAVOK code

The code that represents whether the 'Ceiling and Visibility Okay' (CAVOK) condition has been reported.

DOM: A specific domain comprised of the characters in the ASCII character set.

N: No

Y: Yes

9: missing

POS: 70-75

VISIBILITY-OBSERVATION distance dimension

The horizontal distance at which an object can be seen and identified.

MIN: 000000 MAX: 160000 UNITS: Meters

DOM: A general domain comprised of the numeric characters (0-9).

Missing = 999999

NOTE: Values greater than 160000 are entered as 160000

POS: 85-85

VISIBILITY-OBSERVATION distance quality code

The code that denotes a quality status of a reported distance of a visibility observation.

DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 77-77

VISIBILITY-OBSERVATION variability code

The code that denotes whether or not the reported visibility is variable.

DOM: A specific domain comprised of the characters in the ASCII character set.

N: Not variable

V: Variable

9 = Missing

POS: 78-78

VISIBILITY-OBSERVATION quality variability code

The code that denotes a quality status of a reported VISIBILITY-OBSERVATION variability code.

DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 79-83

AIR-TEMPERATURE-OBSERVATION air temperature

The temperature of the air.

MIN: -0932 MAX: +0618 UNITS: Degrees Celsius

SCALING FACTOR: 10

DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-).

10

+9999 = Missing.

POS: 84-84

AIR-TEMPERATURE-OBSERVATION air temperature quality code

The code that denotes a quality status of an AIR-TEMPERATURE-OBSERVATION.

DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 85-89

AIR-TEMPERATURE-OBSERVATION dew point temperature
The temperature to which a given parcel of air must be cooled at constant pressure and water vapor content for saturation to occur.
MIN: -0982 MAX: +0368 UNITS: Degrees Celsius
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-)
+9999 = Missing.

POS: 90-90

AIR-TEMPERATURE-OBSERVATION dew point quality code
The code that denotes a quality status of the reported dew point temperature.
DOM: A specific domain comprised of the characters in the ASCII character set.

POS: 91-95

ATMOSPHERIC-PRESSURE-OBSERVATION sea level pressure
The air pressure relative to Mean Sea Level (MSL).
MIN: 08600 MAX: 10900 UNITS: Hectopascals
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9).
99999 = Missing.

POS: 96-96

ATMOSPHERIC-PRESSURE-OBSERVATION sea level pressure quality code
The code that denotes a quality status of the sea level pressure of an ATMOSPHERIC-PRESSURE-OBSERVATION.
DOM: A specific domain comprised of the characters in the ASCII character set.

Additional Data Section - Variable length data are provided after the mandatory data. These additional data contain information of significance and/or which are received with varying degrees of frequency. Identifiers are used to note when data are present in the record. If all data fields in a group are missing, the entire group is usually not reported. If no groups are reported the section will be omitted. The additional data section is variable in length with a minimum of 0 characters and a maximum of 637 (634 characters plus a 3-character section identifier) characters.
Note: Specific information (where applicable) pertaining to each variable group of data elements is provided in the data item definition.

4.1.2 Metadata for processed data

METADATA (for processed data, CSV format)

Control Data Section

GEOPHYSICAL-POINT-OBSERVATION date

The date of a GEOPHYSICAL-POINT-OBSERVATION.
MIN: 00000101 MAX: 99991231
DOM: A general domain comprised of integer values 0-9 in the format YYYYMMDD. YYYY can be any positive integer value; MM is restricted to values 01-12; and DD is restricted to values 01-31.

GEOPHYSICAL-POINT-OBSERVATION year

The year of a GEOPHYSICAL-POINT-OBSERVATION.
MIN: 0000 MAX: 9999
DOM: A general domain comprised of integer values 0-9 in the format YYYY. YYYY can be any positive integer value.

GEOPHYSICAL-POINT-OBSERVATION month

The month of a GEOPHYSICAL-POINT-OBSERVATION.
MIN: 01 MAX: 12
DOM: A general domain comprised of integer values 0-9 in the format MM. MM can be any positive integer value

GEOPHYSICAL-POINT-OBSERVATION day

The day of a GEOPHYSICAL-POINT-OBSERVATION.
MIN: 01 MAX: 31
DOM: A general domain comprised of integer values 0-9 in the format DD. DD can be any positive integer value

GEOPHYSICAL-POINT-OBSERVATION time

The time of a GEOPHYSICAL-POINT-OBSERVATION based on
Coordinated Universal Time Code (UTC).
MIN: 0000 MAX: 2359
DOM: A general domain comprised of integer values 0-9 in the format HHMM. HH is restricted to values 00-23; MM is restricted to values 00-59.

FIXED-WEATHER-STATION USAF MASTER STATION CATALOG identifier

The identifier that represents a FIXED-WEATHER-STATION.
DOM: A general domain comprised of the characters in the ASCII character set.
COMMENT: This field includes all surface reporting stations, including ships, buoys, etc.

GEOPHYSICAL-POINT-OBSERVATION latitude coordinate

The latitude coordinates of a GEOPHYSICAL-POINT-OBSERVATION where southern hemisphere is negative.
MIN: -90000 MAX: +90000
UNITS: Angular Degrees
SCALING FACTOR: 1000
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-).
+99999 = Missing

GEOPHYSICAL-POINT-OBSERVATION longitude coordinate

The longitude coordinates of a GEOPHYSICAL-POINT-OBSERVATION where values west from 000000 to 179999 are signed negative.
MIN: -179999 MAX: +180000 UNITS: Angular Degrees
SCALING FACTOR: 1000
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-).
+999999 = Missing

GEOPHYSICAL-POINT-OBSERVATION elevation dimension

The elevation of a GEOPHYSICAL-POINT-OBSERVATION relative to Mean Sea Level (MSL).
MIN: -0400 MAX: +8850 UNITS: Meters
SCALING FACTOR: 1
DOM: A general domain comprised of the numeric characters (0-9), a minus sign (-), and a plus sign (+).
+9999 = Missing

Mandatory data section

SKY-CONDITION-OBSERVATION ceiling height dimension

The height above ground level (AGL) of the lowest cloud or obscuring phenomena layer aloft with 5/8 or more summation total sky cover, which may be predominantly opaque, or the vertical visibility into a surface-based obstruction. Unlimited = 22000.
MIN: 00000 MAX: 22000 UNITS: Meters
SCALING FACTOR: 1
DOM: A general domain comprised of the numeric characters (0-9).
99999 = Missing.

VISIBILITY-OBSERVATION distance dimension

The horizontal distance at which an object can be seen and identified.
MIN: 000000 MAX: 160000 UNITS: Meters
DOM: A general domain comprised of the numeric characters (0-9).
Missing = 999999
NOTE: Values greater than 160000 are entered as 160000

AIR-TEMPERATURE-OBSERVATION air temperature

The temperature of the air.
MIN: -0932 MAX: +0618 UNITS: Degrees Celsius
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-).
10
+9999 = Missing.

AIR-TEMPERATURE-OBSERVATION dew point temperature

The temperature to which a given parcel of air must be cooled at constant pressure and water vapor
content in order for saturation to occur.
MIN: -0982 MAX: +0368 UNITS: Degrees Celsius
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9), a plus sign (+), and a minus sign (-)
+9999 = Missing.

ATMOSPHERIC-PRESSURE-OBSERVATION sea level pressure

The air pressure relative to Mean Sea Level (MSL).
MIN: 08600 MAX: 10900 UNITS: Hectopascals
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9).
99999 = Missing.

WIND-OBSERVATION speed rate

The rate of horizontal travel of air past a fixed point.
MIN: 0000 MAX: 0900 UNITS: meters per second
SCALING FACTOR: 10
DOM: A general domain comprised of the numeric characters (0-9).
9999 = Missing.

NOTE: We did not use additional data section because the data was either sparse or erroneous.

4.2 Flight Data

The source of flight data is US Bureau of Transportation Services. Some of the important metrics included in the flight data are origin airport name, destination airport name, delay information and departure time.

4.2.1 Metadata for raw data

#	Field Name	Field Description
1	Flight date	YYYYMMDD
2	Airline ID	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its Code, Name, or holding company/corporation. Character length = 5
3	Flight number	Character length = 4
4	Origin airport id	Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused. Character length = 5
5	Origin airport code	Character length = 3
6	Origin city name	Origin airport city name
7	Destination airport ID	Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused. Character length = 5
8	Destination airport code	Character length = 3
9	Destination city name	Destination airport city name
10	Departure time	Actual Departure Time (local time: hhmm)
11	Departure delay minutes	Difference in minutes between scheduled and actual departure time. Early departures set to 0
12	Departure delay 15	Departure Delay Indicator, 15 Minutes or More (1=Yes)
13	Arrival time	Actual Arrival Time (local time: hhmm)
14	Arrival delay minutes	Difference in minutes between scheduled and actual arrival time. Early arrivals set to 0.
15	Arrival delay 15	Arrival Delay Indicator, 15 Minutes or More (1=Yes)
16	Cancelled	Cancelled Flight Indicator (1=Yes)
17	Cancelled code	Specifies the Reason for Cancellation
18	Diverted	Diverted Flight Indicator (1=Yes)
19	Weather delay	Weather Delay, in Minutes

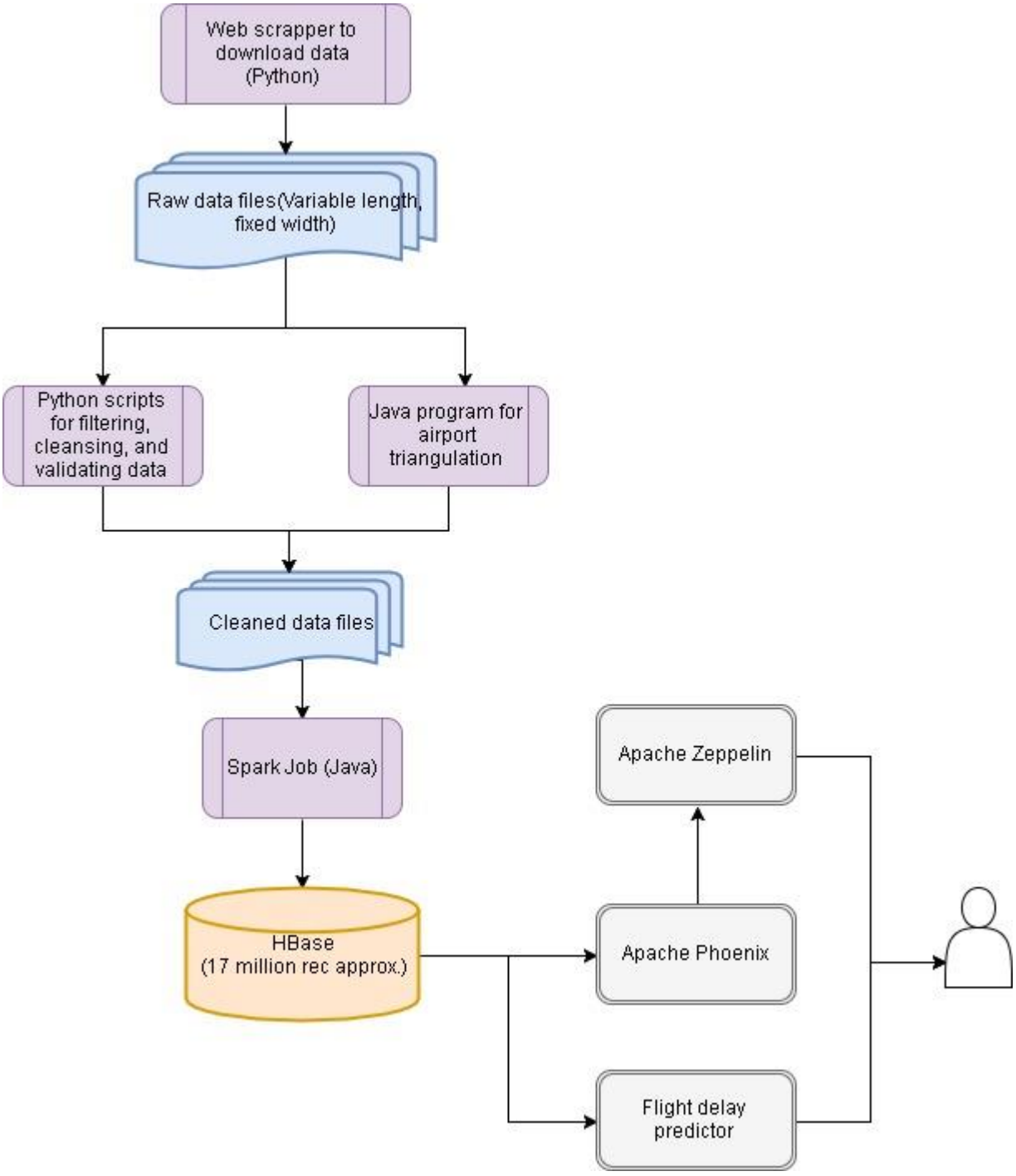
4.2.2 Metadata for processed data (CSV format)

#	Field Name	Field Description
1	Flight date	YYYYMMDD
2	Departure time	Actual Departure Time (local time: hhmm)
3	Airline ID	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its Code, Name, or holding company/corporation. Character length = 5
4	Flight number	Character length = 4
5	Origin airport code	Character length = 3
6	Origin city name	Origin airport city name
7	Latitude	Latitude info for airport
8	Longitude	Longitude info for airport
9	Departure delay minutes	Difference in minutes between scheduled and actual departure time. Early departures set to 0
10	Departure delay 15	Departure Delay Indicator, 15 Minutes or More (1=Yes)
11	Cancelled	Cancelled Flight Indicator (1=Yes)
12	Cancelled code	Specifies the Reason for Cancellation
13	Diverted	Diverted Flight Indicator (1=Yes)
14	Weather delay	Weather Delay, in Minutes
15	Year	Year of the flight date
16	Month	Month of the flight date
17	Day	Day of the flight date

5. High Level Design

This section explains the high-level design of the application.

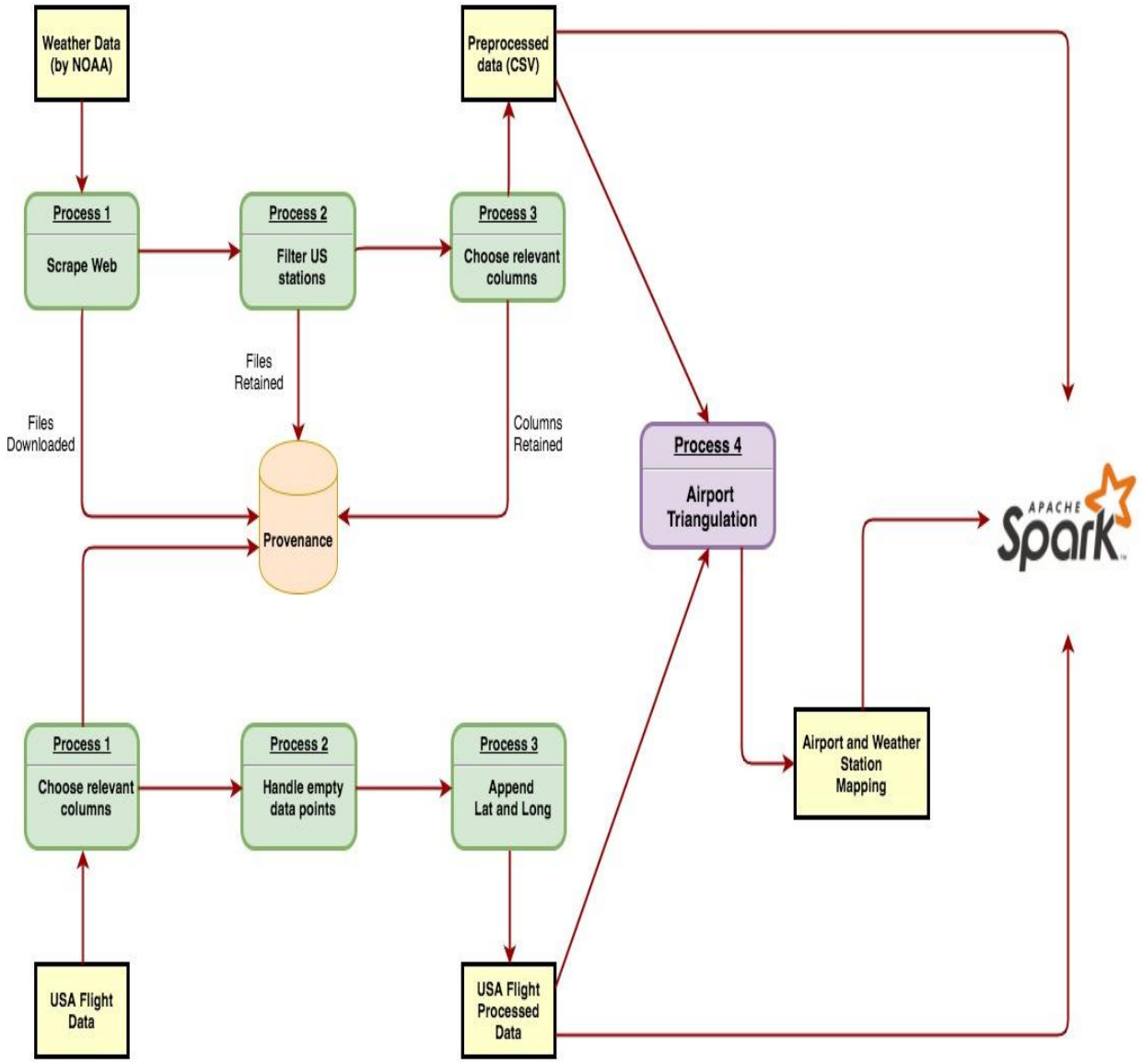
5.1 High Level Architecture



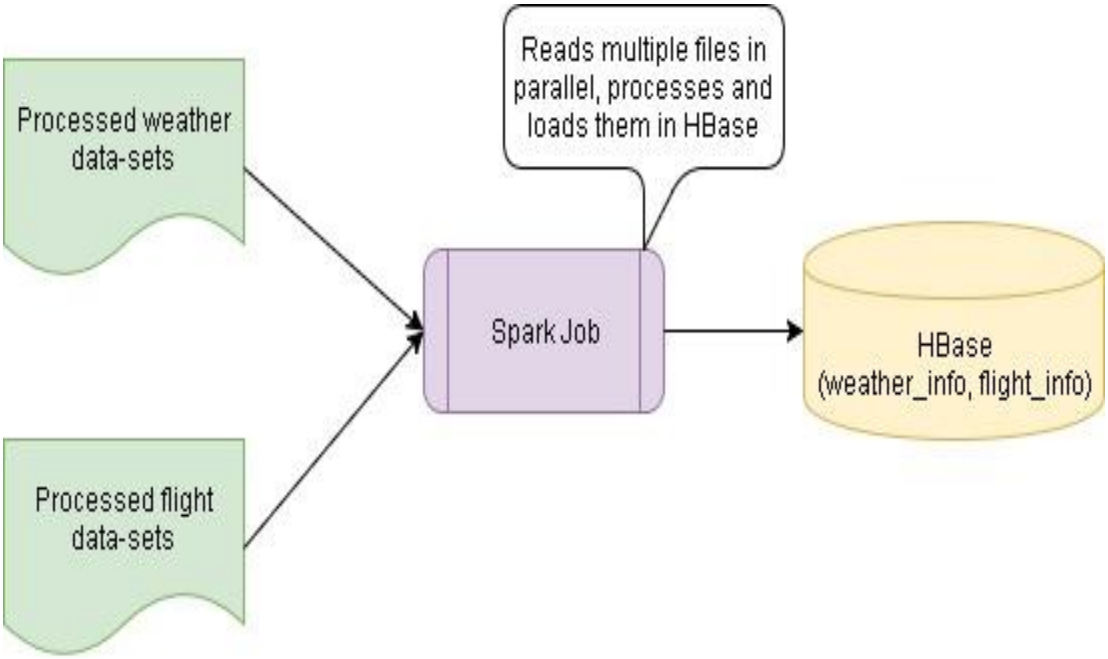
5.2 Technology Stack

#	Technology	Usage
1	**Python**	For writing web scraper, for data pre-processing
2	**bash**	For data pre-processing
3	**Java**	For airport triangulation, Apache Spark
4	**Apache Spark**	Massively parallel data ingestion pipeline
5	**HBase**	NoSQL database for persistent data storage
6	**Apache Phoenix**	Analytics for HBase
7	**Apache Zeppelin**	Web-based notebook for interactive-data analytics
8	**Hadoop Ecosystem**	HDFS as file system storage, YARN for resource management
9	**SQL**	For writing queries
10	**scikit-learn**	For machine learning
11	**Pandas**	For statistics

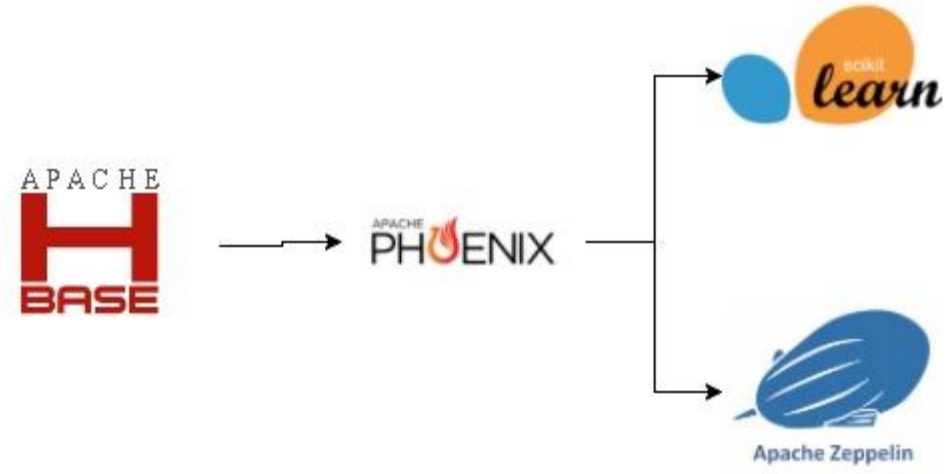
5.3 Data Pre-processing



5.4 Data ingestion pipeline using Apache Spark



5.5 Querying, Visualization, and Machine Learning



6. Low Level Design

This section explains the high-level design of the application.

6.1 GitHub Repository

All the code for this project can be found at the following repository:
https://github.com/rahulrsharma999/storage-warriors-hbase

Check the following directories for code:

- AirportTriangulation: Code to find the nearest weather station for each US airport.
- Machine-learning: Code for training the machine learning model to predict flight delays
- preprocess-abhinay: Code to download raw data and pre-process it.
- preprocessing-scripts: bash script for data pre-processing
- sparkInjectionJob/StorageWarriors: Spark job for data ingestion to HBase

6.2 Pre-processing Weather Data

- Input: weather dataset obtained from the NOAA which is organized as a set of many small files in different directories on the NOAA ftp.
- Each filename represents a station id and each year data is identified by the respective directory name.
- To preprocess the weather data study the sections below.

6.2.1 Process 1: Scrape Web

- Input: Starting and ending year of which data needs to be downloaded
- A custom python script is run to download all the data from the NOAA ftp site for all the years needed.
- Following is the list of steps performed by the python script to mine the data:
  - Specify the start and end year for which the weather data has to be gathered
  - Scrape the webpage to go through the HTML of the ftp website and create a list of links present on that webpage.
  - Code:

```
def scrape_webpage():
    try:
        soup = bs(urlopen(par_url))
        tar_files = soup.findAll('a')[5:]      #removing first 5 irrelevant links

        for i in range(0, len(tar_files)):
            # Make the list save the links and not just filenames
            file_links.append(par_url + tar_files[i].get('href'))
            #print file_links[i]

    except HTTPError, e:
        print "HTTP Error:", e.code, par_url
    except URLError, e:
        print "URL Error:", e.reason, par_url
```

- Download tar files for each year.

```
def download(year):
    # Downloading files now for the year provided
    try:
        file_url = file_links[year]
        f = urlopen(file_url)

        directory = os.getcwd() + "/zips/"
        if not os.path.exists(directory):
            os.makedirs(directory)

        with open(directory + os.path.basename(file_url), "wb") as local_file:
            local_file.write(f.read())

        file_name = file_url[-11:]

        return file_name

    except HTTPError, e:
        print "HTTP Error:", e.code, file_url
    except URLError, e:
        print "URL Error:", e.reason, file_url
```

6.2.2 Process 2: Filter US stations

- Extract only the files which belong to stations only in United States
- This involved a lot of mining tasks:
  - Search internet for the list of US stations
  - Using a tool to map the US weather stations to station IDs
  - Generating a file containing a list of IDs of weather station in the US
- Code:

```
with open("us-stations.txt") as f:
    for line in f:
        us_stations.append(line.rstrip('\n'))

for i in range(start_year, end_year+1):
    file_name = download(i - 1901)
    directory = untar(file_name)


    for filename in os.listdir(directory[:-1]):
        files_total = files_total + 1
        f = os.path.join(directory, filename)
        if filename in us_stations:
            clean_data(f)
            files_retained = files_retained + 1

        os.remove(f)
```

6.2.3 Process 3: Choose relevant columns

- Clean the data for every extracted file of a US weather station which involves keeping only the mandatory columns
- Out of a possible 29 mandatory and 676 additional columns (out of which each file had some 10-15 columns), we have kept only 15 columns namely: “STN_ID”, “DATE”, “TIME”, “LAT”, “LONG”, “ELEVATION”, “WND_SPD”, “CIG”, “VIS”, “TMP”, “DEW”, “SLP”.
- Code:

```
with open(dst_file,"wb") as result:
    wtr = csv.writer( result )
    wtr.writerow(("STN_ID", "DATE", "YEAR", "MONTH", "DAY", "TIME", "LAT", "LONG", "ELEVATION",
                  "WND_SPD", "CIG", "VIS", "TMP", "DEW", "SLP"))


    for r in rdr:
        wind_speed = r[10].split(",")[3]

        temp = r[1].split("T")
        date = temp[0].split("-")

        time_t = temp[1].split(":")
        time = str(time_t[0]) + str(time_t[1]) + "T"

        wtr.writerow( (r[0], date[0]+date[1]+date[2], date[0], date[1], date[2], time, r[3], r[4],
                       r[5], wind_speed, r[11].split(",")[0], r[12].split(",")[0], r[13].split(",")[0],
                       r[14].split(",")[0], r[15].split(",")[0]) )
```

Data Provenance is maintained by storing:

- Total files downloaded
- Total number of files retained
- Total mandatory columns
- Total number of mandatory columns retained

Finally, the output is a CSV file that contains the **preprocessed** weather data.

6.3 Pre-processing Flight Data

The data for flight delay was downloaded from website. The data did not have any information about latitude and longitude of the airports. Upon searching on internet, we found US airport geolocation data in JSON format, from which the latitude and longitude for every airport in USA was extracted.

Now, following is the data preprocessing pipeline which was developed in Python language.

6.3.1 Process 1: Choose relevant columns

- Inputs:
  - Monthly flight delay data
  - CSV file containing airport code and respective geo-location
- Reading each flight delay file and capturing columns relevant to our business questions.

6.3.2 Process 2: Handle empty data points

- For cancelled flights, rows in data have null/missing values. We are changing departure time and departure delay to 9999, departure delay more than 15 minutes to 1 and weather_delay to 0 if not cancelled due to weather.

Code:

```
port_code = r[4]
if port_code not in dict:
    continue
lat = dict[port_code][0]
lon = dict[port_code][1]

# if cancellation is not because of weather then just remove it
if r[16] != "" and r[16] != "B":
    continue

fl_date = r[0].split('-')
date = fl_date[0] + fl_date[1] + fl_date[2]

dep_time = r[9]
if dep_time == '':
    dep_time = "9999"

dep_delay = r[10]
if dep_delay == '':
    dep_delay = 9999

dep_delay15 = r[11]
if dep_delay15 == '':
    dep_delay15 = 1

w_delay = r[18]
if w_delay == '' and r[16] != "B":
    w_delay = 1
else:
    w_delay = 0
```

6.3.3 Process 3: Append latitude and longitude

- Add two columns for latitude and longitude information in the data for individual origin airport.
- Now, we have the processed flight delay data with columns mentioned in metadata (4.2). These csv files were used for airport triangulation.

6.4 Airport triangulation

For each airport, calculate the geometrical distance between the airport and all the weather stations in US. Select the weather station for which distance between airport and that weather station is minimum and the distance is under 50 KM.

Following are the steps of the algorithm used:

Inputs: List of US airport with geolocation info, list of weather stations with geolocation info

We computed the distance between all the US airports and the US stations using geometric distance formula.

We chose the station-airport pair with minimum distance found.

The mapping was saved in a new CSV file.

6.5 Spark Pipeline for data ingestion

Apache Spark is a fast and general engine for large-scale data processing. It can run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Apache Spark has an advanced DAG execution engine that supports acyclic data flow and in-memory computing. Spark powers a stack of libraries including SQL and Data Frames, MLlib for machine learning, GraphX, and Spark Streaming. We can combine these libraries seamlessly in the same application.

Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3. We can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos. It can access data in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source. Moreover, Spark offers over 80 high-level operators that make it easy to build parallel apps. Due to all these qualities, a hadoop ecosystem and HBase DB, Spark was the best framework for us to perform high-speed in-memory processing and loading of flight and weather data to HBase.

In our program, we configured Spark (using SparkConfiguration.java) to run with HBase (DBConfiguration.java) and defined the job configuration (in JobConfiguration.java) for the hadoop jobs to read the data from CSV files and load them into HBase. Our data ingestion application (HBaseIngestionApp.java), took as parameter the type of data to be loaded - WEATHER or FLIGHT. The processor files (FlightDataProcessor.java, WeatherDataProcessor.java) contain the processData() method that contains the logic to process each line of the CSV file. The configuration details about the HBase table such as the table name, column names, input CSV file details, row key and row values are provided in the .yml files (flight.yml and weather.yml) which is parsed by processData() to load the row field values in the required format into HBase.

The Spark program for our project is written in Java. The spark program was run on 4 processor cores of a single-node cluster. The deafult parallelism used for the Spark jobs was 8 (i.e., 8 threads per core). For performing serialization at high speed, KyroSerializer was used.

6.6 'Machine Learning

The input is the Flight-Weather correlation table which consists of rows from the weather table that match the time when the flight was delayed.

Important fields in the data are:

FLIGHT_ID: This field is important because multiple rows will match against a flight id due to the varying rate of reporting weather data by the weather stations. We have gone with a tolerance of 1 hour above or below the flight time to find the matching row in the weather table. We will only require one row per flight id which states the weather conditions when the flight was delayed.

WEATHER_DELAY: This field will be the "y" or target for our classification algorithm. The possible values for class are "0" or "1". A "0" indicates that the flight was not delayed due to weather and a "1" indicates that a flight was delayed.

WEATHER FIELDS: These are fields which determine the classification accuracy on the test data set as these will be used as features to train the model. The fields include: "Wind", "Visibility", "CIG", "Air Temperature", "Dew", "Sea Level Pressure". The output of this phase a CSV file containing only one row pertaining to a flight id.

Handling Missing Values before Training Model:

The weather data contains missing values in many columns and this was not removed during preprocessing stage of weather data because this is also a valuable information for performing analytics.

The following are the steps performed to handle missing values in python:

- Read the CSV file for containing the data preprocessed for machine learning
- Print the data statistics to figure out the max and min values in each column.
- The missing values legend for each column is as follows:
  - Wind Column: 9999
  - Cig Column: 99999
  - Visibility Column: 999999
  - Air Temperature Column: 9999
  - Dew Column: 9999
  - Sea Level Pressure Column: 99999
- For each column the missing values are replaced by NaN values in python
- The missing values in each column are then imputed to contain the mean of that columns data.
- Linear Discriminant analysis is performed to make sure that the missing data has been removed (This algorithm does not work if data is missing)

Machine Learning Model:

- Since the number of rows in the preprocessed data set is close to 150000 regular machine learning techniques using scikit learn implementations are slower to train this volume of data.
- The following algorithms are used and compared for performance:
- Logistic Regression with Stochastic Gradient Descent optimization solver is used because this leads to much faster convergence in case of a large dataset.
- Random Forests are used to predict the probability of flight delay based on training the model using a specific set of splits setting.
- The results are then compared to determine the better algorithm for this dataset.

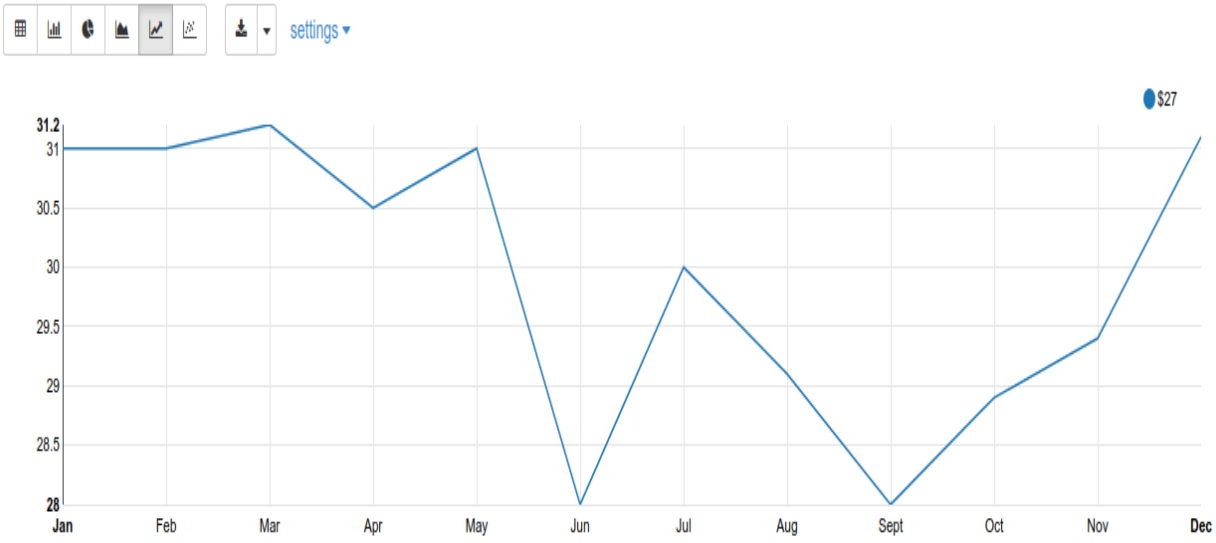
7. Queries and Results

7.1 Query description and results

7.1.1 Business question 1

For a given weather station and year, report the highest observed temperatures (degree centigrade) by month. Provide a mechanism to visualize this information.

```
%phoenix

/* QUESTION 1 : OLD SCHOOL APPROACH */
/* HIGHEST OBSERVED TEMPERATURE FOR EVERY MONTH IN 1990 FOR WEATHER STATION 6197609999 BY A SINGLE QUERY*/

select 'Jan', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199001_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Feb', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199002_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Mar', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199003_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Apr', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199004_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'May', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199005_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Jun', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199006_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Jul', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199007_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Aug', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199008_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Sept', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199009_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Oct', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199010_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Nov', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199011_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
union all
select 'Dec', max(TO_NUMBER("weather_main".TMP))/10 from "weather_info" where id like '199012_____6197609999' and TO_NUMBER("weather_main".TMP) != 9999
```



Took 2 min 9 sec. Last updated by anonymous at October 28 2017, 8:21:09 PM. (outdated)

```
%phoenix

/* QUESTION 1 : OPTIMIZED APPROACH */
/* OPTIMIZING THE MAX, MIN AND AVG TEMPERATURE CALCULATIONS BY USING A SUMMARY TABLE*/

CREATE SEQUENCE temp_summary.ts_seq START WITH 1 INCREMENT BY 1;

create table if not exists temp_summary(
rowid integer primary key,
stnid bigint,
month varchar,
max_temp double,
min_temp double,
avg_temp double
);
```

```
%phoenix

/*INSERTING ALL THE MIN, MAX, AVG VALUES INTO SUMMARY TABLE*/

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Jan',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199001______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Feb',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199002______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Mar',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199003______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Apr',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199004______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;
```

```
UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'May',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199005______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Jun',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199006______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Jul',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199007______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Aug',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199008______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;
```

```
UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Sept',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199009______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Oct',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199010______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Nov',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199011______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;

UPSERT INTO temp_summary
select NEXT VALUE for temp_summary.ts_seq, 6197609999, 'Dec',
max(TO_NUMBER("weather_main".TMP))/10,
min(TO_NUMBER("weather_main".TMP))/10,
avg(TO_NUMBER("weather_main".TMP))/10
from "weather_info"
where id like '199012______6197609999'
and TO_NUMBER("weather_main".TMP) != 9999;
```

```
%phoenix
select * from temp_summary
```

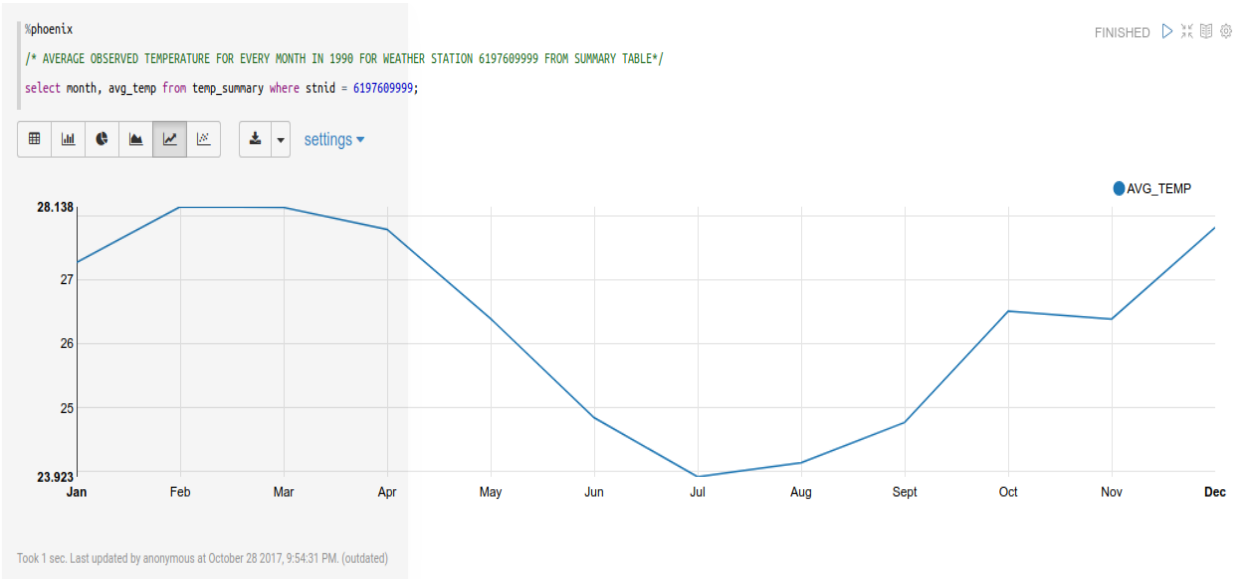
FINISHED

ROWID	STNID	MONTH	MAX_TEMP	MIN_TEMP	AVG_TEMP
3	6197609999	Jan	31.0	24.2	27.2714
4	6197609999	Feb	31.0	24.5	28.1382
5	6197609999	Mar	31.2	24.8	28.1269
6	6197609999	Apr	30.5	25.3	27.7844
7	6197609999	May	31.0	14.8	26.3918
8	6197609999	Jun	28.0	22.4	24.8464
9	6197609999	Jul	30.0	20.0	23.9229
10	6197609999	Aug	29.1	20.0	24.1427
11	6197609999	Sept	28.0	21.5	24.773

Took 1 sec. Last updated by anonymous at October 28 2017, 9:54:09 PM. (outdated)

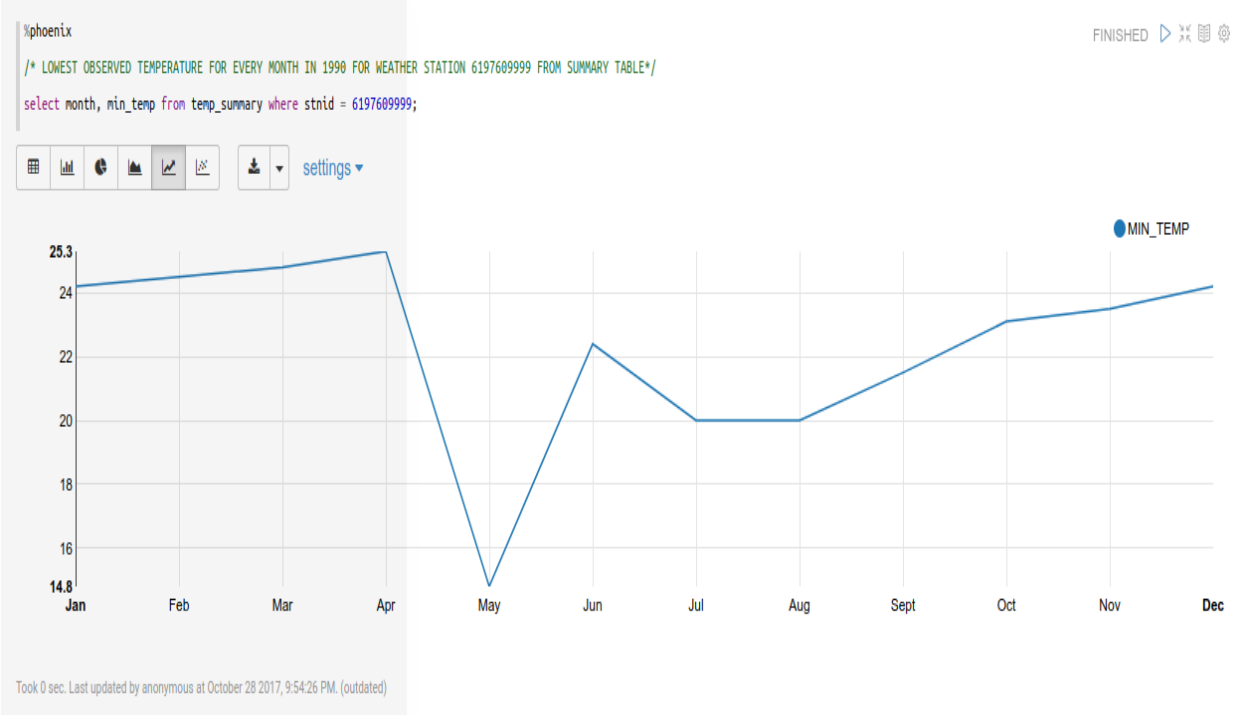
7.1.2 Business question 2

For a given weather station and year, report the average observed temperatures (degree centigrade) by month. Provide a mechanism to visualize this information.



7.1.3 Business question 3

For a given weather station and year, report the lowest observed temperatures (degree centigrade) by month. Provide a mechanism to visualize this information.



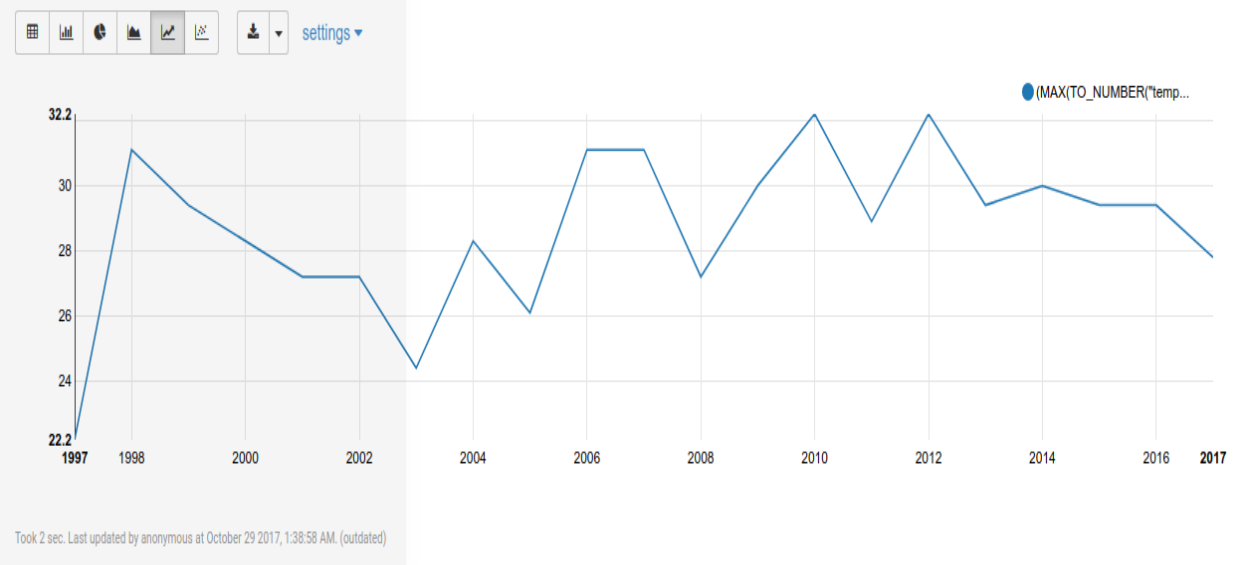
7.1.4 Business question 4

For a given weather station and month, show the variation in mean temperature (degree centigrade) over a period of 20 years. Provide a mechanism to visualize this information.

```
%phoenix

/* QUESTION 2 */
/* CALCULATING INCREASE IN MEAN TEMPERATURE IN THE LAST 20 YEARS FOR A WEATHER STATION FOR THE SAME MONTH*/

create view "temperature_austin" (
 id varchar primary key,
 "temp_main".LAT varchar,
 "temp_main".LONG varchar,
 "temp_main".TMP varchar,
 "temp_main".MONTH varchar,
 "temp_main".YEAR varchar
);

/* INCREASE IN MEAN TEMPERATURE IN THE LAST 20 YEARS (1997-2017) FOR THE MONTH OF MAY */

select "temp_main".YEAR,
max(TO_NUMBER("temp_main".TMP))/10
 from "temperature_austin"
where id like '____05_____7264501489'
and TO_NUMBER("temp_main".TMP) != 9999 and TO_NUMBER("temp_main".YEAR) > 1996
group by "temp_main".YEAR
```

```
Query executed successfully. Affected rows : 0
```

Took 0 sec. Last updated by anonymous at October 28 2017, 11:02:10 PM. (outdated)



Took 2 sec. Last updated by anonymous at October 29 2017, 1:38:58 AM. (outdated)

7.1.5 Business question 5

Find the relation between temperature and elevation (altitude). Ideally as elevation increases, temperature decreases, and the rate of decrease is 6.5 degree centigrade for each 1 km of altitude change. Validate if the observed data provides support for this trend along with providing a mechanism for visualizing this information.

```
%phoenix

/* QUESTION 3 */
/* CALCULATION FOR FINDING THE CHANGE IN TEMPERATURE OF AN AREA AS ITS ELEVATION INCREASES FROM THE SEA LEVEL */

create table elevation_vs_temp (
rowid integer primary key,
elevation_bkt integer,
temperature double
);

CREATE SEQUENCE elevation_vs_temp.et_seq START WITH 1 INCREMENT BY 1;
```

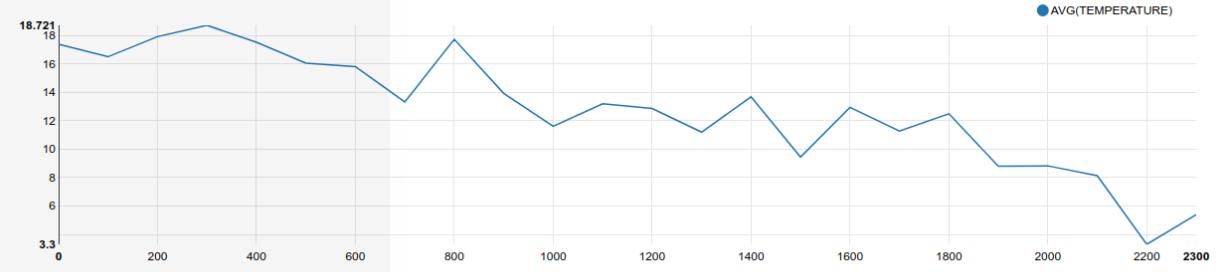
```
Query executed successfully. Affected rows : 0

Took 2 sec. Last updated by anonymous at October 29 2017, 2:23:24 AM. (outdated)
```

```
%phoenix

UPSERT INTO elevation_vs_temp
select NEXT VALUE for temp_summary.ts_seq,
 case when elv < -100 then -200
  when elv >= -100 and elv < 0 then -100
  when elv >= 0 and elv < 100 then 0
  when elv >= 100 and elv < 200 then 100
  when elv >= 200 and elv < 300 then 200
  when elv >= 300 and elv < 400 then 300
  when elv >= 400 and elv < 500 then 400
  when elv >= 500 and elv < 600 then 500
  when elv >= 600 and elv < 700 then 600
  when elv >= 700 and elv < 800 then 700
  when elv >= 800 and elv < 900 then 800
  when elv >= 900 and elv < 1000 then 900
  when elv >= 1000 and elv < 1100 then 1000
  when elv >= 1100 and elv < 1200 then 1100
  when elv >= 1200 and elv < 1300 then 1200
  when elv >= 1300 and elv < 1400 then 1300
  when elv >= 1400 and elv < 1500 then 1400
  when elv >= 1500 and elv < 1600 then 1500
  when elv >= 1600 and elv < 1700 then 1600
  when elv >= 1700 and elv < 1800 then 1700
  when elv >= 1800 and elv < 1900 then 1800
  when elv >= 1900 and elv < 2000 then 1900
  when elv >= 2000 and elv < 2100 then 2000
  when elv >= 2100 and elv < 2200 then 2100
  when elv >= 2200 and elv < 2300 then 2200
  when elv >= 2300 and elv < 2400 then 2300
  when elv >= 2400 and elv < 2500 then 2400
  when elv >=2500 then 2500 end as elvbkt, elvtmp
  from (
    select TO_NUMBER("weather_main".ELEVATION) as elv, (TO_NUMBER("weather_main".TMP))/10 as elvtmp
    from "weather_info"
    where id like '199006121200_________'
    and TO_NUMBER("weather_main".TMP) != 9999
    order by "weather_main".ELEVATION
    ) temp order by elvbkt

Query executed successfully. Affected rows : 967

Took 1 sec. Last updated by anonymous at October 29 2017, 2:25:12 AM.
```

```
%phoenix
/* ELEVATION VS TEMPERATURE CHANGE GRAPH FOR ALL STATIONS */
select elevation_bkt, avg(temperature) from elevation_vs_temp where elevation_bkt>=0 group by elevation_bkt
```

FINISHED

settings



Took 0 sec. Last updated by anonymous at October 29 2017, 2:39:03 AM. (outdated)

7.1.6 Business question 6

Report the count of those weather stations which have witnessed Hurricane force winds. Beaufort wind force scale is an empirical measure that relates wind speed to observed conditions at sea or on land. Hurricane force winds are categorized as those which have wind speeds greater than equal to 118 km/hr.

```
%phoenix

/* Question 4*/
/* CALCULATION FOR COUNT OF POSSIBLE HURRICANE LOCATIONS BY CREATING SECONDARY INDEXES */

create index idx_weather_visibility on "weather_info"(TO_NUMBER("weather_main".VIS) ASC)


/* FROM WIKIPEDIA: THERE WERE A TOTAL OF 8 STORMS IN THE YEAR 1990 */

select count(*) as Hurricane_Location_Count
from
    (select distinct SUBSTR(id, 13) as STATION_ID
    from "weather_info"
    where TO_NUMBER("weather_main".WND_SPD) > 327
    and TO_NUMBER("weather_main".WND_SPD) != 9999
) tmp
```

HURRICANE_LOCATION_COUNT
40

7.1.7 Business question 7

Report the count of those weather stations which have very low visibility. Areas with visibility of less than 100 metres (330 ft.) are usually reported as very low visibility areas.

```
%phoenix

/* Question 5*/
/* CALCULATION FOR COUNT OF POSSIBLE VERY LOW VISIBILITY LOCATIONS BY CREATING SECONDARY INDEXES */

create index idx_weather_windspeed on "weather_info"(TO_NUMBER("weather_main".WND_SPD) DESC);

select count(*) as Very_Low_Visibility_Location_Count
from (
    select distinct SUBSTR(id, 13) as STATION_ID
    from "weather_info"
    where TO_NUMBER("weather_main".VIS) < 100
    and TO_NUMBER("weather_main".VIS) != 9999
) tmp
```

VERY_LOW_VISIBILITY_LOCATION_COUNT
808

7.1.8 Business question 8

Find the maximum observed flight delay (in minutes) in the entire dataset.

7.1.9 Business question 9

Find the average observed flight delay (in minutes) in the entire dataset.

```
%phoenix
create view "flight_info" (
 id varchar primary key,
 "flight_main".CITY_NAME varchar,
 "flight_main".DAY varchar,
 "flight_main".DEP_DEL15 varchar,
 "flight_main".DEP_DELAY_TOTAL varchar,
 "flight_main".DIVERTED varchar,
 "flight_main".LAT varchar,
 "flight_main".LONG varchar,
 "flight_main".MONTH varchar,
 "flight_main".ORIGIN varchar,
 "flight_main".WEATHER_DELAY varchar,
 "flight_main".W_CANCELLED varchar,
 "flight_main".YEAR varchar
);

Query executed successfully. Affected rows : 0

Took 22 sec. Last updated by anonymous at October 29 2017, 8:53:51 AM.
```

```
%phoenix                                                                    FINISHED

/* QUESTION 6 */
/* CALCULATION TO DETERMINE THE MAXIMUM AND AVERAGE DEPARTURE DELAYS IN THE ENTIRE DATA */

create index idx_flight_delaytot on "flight_info"(TO_NUMBER("flight_main".DEP_DELAY_TOTAL) ASC)


select max(TO_NUMBER("flight_main".DEP_DELAY_TOTAL)) as MAX_DEPARTURE_DELAY_IN_MINUTES,
       avg(TO_NUMBER("flight_main".DEP_DELAY_TOTAL)) as AVG_DEPARTURE_DELAY_IN_MINUTES
from "flight_info"
where TO_NUMBER("flight_main".DEP_DELAY_TOTAL) != 9999;
```

MAX_DEPARTURE_DELAY_IN_MINUTES	AVG_DEPARTURE_DELAY_IN_MINUTES
1439	7.5114

7.1.10 Business question 10

Report the number of instances where flights have been delayed by more than 15 minutes.

```
%phoenix

/* QUESTION 7 */
/* CALCULATION TO DETERMINE THE COUNT OF FLIGHT DELAYS GREATER THAN 15 MINUTES IN THE ENTIRE DATA */

create index idx_flight_weatherdelay15 on "flight_info"(TO_NUMBER("flight_main".DEP_DEL15) ASC);


select count(*) as COUNT_OF_FLIGHT_DELAYS_GREATER_THAN_15_MINUTES
from "flight_info"
where TO_NUMBER("flight_main".DEP_DEL15) = 1;
```

COUNT_OF_FLIGHT_DELAYS_GREATER_THAN_15_MINUTES
731625

7.1.11 Business question 11

Find out the correlation between flight and weather data. Note that there is a spatiotemporal relation between weather and flight data.

```
%phoenix

/* QUESTION 8 */
/* OBTAIN THE SPATIO_TEMPORAL CORRELATION BETWEEN THE FLIGHT DATA AND THE
   WEATHER DATA OF THE NEAREST WEATHER STATION (WITHIN 50 KMS) FOR THAT FLIGHT
*/

create view "weather_EWR_info" (
 id varchar primary key,
 "weather_main".CIG varchar,
 "weather_main".DEW varchar,
 "weather_main".ELEVATION varchar,
 "weather_main".LAT varchar,
 "weather_main".LONG varchar,
 "weather_main".SLP varchar,
 "weather_main".TMP varchar,
 "weather_main".VIS varchar,
 "weather_main".WND_SPD varchar,
 "weather_main".YEAR varchar,
 "weather_main".MONTH varchar,
 "weather_main".DAY varchar,
 "weather_main".TIME varchar
)
```

```
Query executed successfully. Affected rows : 0
```

Took 11 sec. Last updated by anonymous at October 29 2017, 8:05:04 PM. (outdated)

```
%phoenix

create view "flight_EWR_info" (
  id varchar primary key,
  "flight_main".CITY_NAME varchar,
  "flight_main".DAY varchar,
  "flight_main".DEP_DEL15 varchar,
  "flight_main".DEP_DELAY_TOTAL varchar,
  "flight_main".DIVERTED varchar,
  "flight_main".LAT varchar,
  "flight_main".LONG varchar,
  "flight_main".MONTH varchar,
  "flight_main".ORIGIN varchar,
  "flight_main".WEATHER_DELAY varchar,
  "flight_main".W_CANCELLED varchar,
  "flight_main".YEAR varchar,
  "flight_main".TIME varchar
);
```

```
Query executed successfully. Affected rows : 0
```

Took 7 sec. Last updated by anonymous at October 29 2017, 8:24:04 PM.

```
%phoenix

create view "flight_EWR_info" (
 id varchar primary key,
 "flight_main".CITY_NAME varchar,
 "flight_main".DAY varchar,
 "flight_main".DEP_DEL15 varchar,
 "flight_main".DEP_DELAY_TOTAL varchar,
 "flight_main".DIVERTED varchar,
 "flight_main".LAT varchar,
 "flight_main".LONG varchar,
 "flight_main".MONTH varchar,
 "flight_main".ORIGIN varchar,
 "flight_main".WEATHER_DELAY varchar,
 "flight_main".W_CANCELLED varchar,
 "flight_main".YEAR varchar,
 "flight_main".TIME varchar
);
```

```
Query executed successfully. Affected rows : 0
```

Took 7 sec. Last updated by anonymous at October 29 2017, 8:24:04 PM.


```
%phoenix

create view "flight_EWR_info" (
 id varchar primary key,
 "flight_main".CITY_NAME varchar,
 "flight_main".DAY varchar,
 "flight_main".DEP_DEL15 varchar,
 "flight_main".DEP_DELAY_TOTAL varchar,
 "flight_main".DIVERTED varchar,
 "flight_main".LAT varchar,
 "flight_main".LONG varchar,
 "flight_main".MONTH varchar,
 "flight_main".ORIGIN varchar,
 "flight_main".WEATHER_DELAY varchar,
 "flight_main".W_CANCELLED varchar,
 "flight_main".YEAR varchar,
 "flight_main".TIME varchar
);
```

```
Query executed successfully. Affected rows : 0
```

Took 7 sec. Last updated by anonymous at October 29 2017, 8:24:04 PM.

```
%phoenix

UPSERT /*+ USE_SORT_MERGE_JOIN */ INTO flight_weather_delay_info
select
NEXT VALUE for temp_summary.ts_seq,
f.id,
f."flight_main".WEATHER_DELAY,
w."weather_main".CIG,
w."weather_main".DEW,
w."weather_main".SLP,
w."weather_main".TMP,
w."weather_main".VIS,
w."weather_main".WND_SPD
from "flight_EWR_info" f inner join "weather_EWR_info" w
on (
    ( TO_NUMBER("flight_main".YEAR)  = TO_NUMBER("weather_main".YEAR)  ) AND
    ( TO_NUMBER("flight_main".MONTH) = TO_NUMBER("weather_main".MONTH) ) AND
    ( TO_NUMBER("flight_main".DAY)   = TO_NUMBER("weather_main".DAY)   )
 )
 where TO_NUMBER("flight_main".YEAR) = 2015
 AND TO_NUMBER("weather_main".YEAR) = 2015
 AND TO_NUMBER("weather_main".TIME)
     between
         (TO_NUMBER("flight_main".TIME)-100)
     AND (TO_NUMBER("flight_main".TIME)+100)
```

```
Query executed successfully. Affected rows : 293354
```

Took 36 sec. Last updated by anonymous at October 29 2017, 10:07:42 PM. (outdated)

```
%phoenix
select * from flight_weather_delay_info limit 100
```

FINISHED

ROWKEY	FLIGHT_ID	WEATHER_DELAY	CIG	DEW	SLP	TMP	VIS	WND_SPD
1001	20150101055219805012O	0	22000	-117	10214	-44	16093	46
1002	20150101055219805012O	0	99999	-117	10214	-44	16000	46
1003	20150101055219805012O	0	7620	-117	10212	-44	16093	41
1004	20150101055219805012O	0	99999	9999	99999	9999	999999	9999
1005	20150101060219977016B	0	22000	-117	10214	-44	16093	46
1006	20150101060219977016B	0	99999	-117	10214	-44	16000	46
1007	20150101060219977016B	0	7620	-117	10212	-44	16093	41
1008	20150101065919977004I	0	99999	-117	10214	-44	16000	46
1009	20150101065919977004I	0	7620	-117	10212	-44	16093	41

Took 2 sec. Last updated by anonymous at October 29 2017, 10:16:50 PM.

7.1.12 Business question 12

Use the correlation between weather and flight data to predict if a flight will be delayed based on forecasted weather metrics.

The correlation data obtained from question 11 is used as input for the machine learning algorithm to predict the flight delays. The correlation data for 2015 was used as the training data to train the ML model.

8. Challenges Faced:

- Being the data size too large (750GB), saving the data on local machine or on any free cloud service was not possible. Hence, we had to take a decision to download limited data.
- Weather data was organized in a good way but few of the information need to answer the business questions were not available directly. For example, the country name of the weather station as not mentioned. Due to lack of domain knowledge, choosing the right set of columns to answer the BQ's took some time.
- Flight delay data did not contain the geolocation information which we had to search online and extract, connect the relevant info to our data.
- Flight delay data has a lot of missing values which we had to take care of. Also, during querying of the tables from HBase, we had to keep in mind that data has missing values.
- Choosing the right tool to use for querying and visualization took a lot of time and effort. In the end, we decided to go with Apache Phoenix and Apache Zeppelin to do this work. These tools also had a lot of shortcomings like,
    - They do provide SQL interface to NoSQL databases but not all commands are supported.
    - Support for joins are limited.
    - There are issues when the already created tables in HBase are imported into Phoenix. It is unable to read the metadata info for the tables from HBase and hence assumes every field to be a byte.

9. Future Work:

- The primary requirement is to scale our project to handle the needs of big data.
    - This can be done by distributing our database over a set of servers in a cluster
    - Setting up Hadoop cluster for operation in distributed environment
- Using a custom solution to visualize the HBase data to answer queries.
- The Apache Phoenix solution used in this project is not scalable due to its SQL structure and strict constraint to maintain the table in a format to query the data.
- This will involve the development of a custom tool having the following elements:
- JavaScript Front End Framework (like AngularJS) for user query
    - REST API's to interact with HBase
    - D3 charts to visualize the data returned from the queries
    - Due to the NoSQL nature of the database it very time consuming to answer some queries as it requires a full table scan in HBase due to the field not being a row key. This required the use of secondary index on the fields that are frequently queried. The usage of a secondary index on a field is non-trivial in HBase since it requires a large amount of coding to be performed by the developer including but not handling the update of the secondary index
- The prediction of a flight delay at an airport given the weather conditions can be leveraged in a sample application which needs this kind of analytics
- From a software engineering perspective, the following aspects need to be added to promote code maintainability:
    - Unit Testing
    - Integration Testing
    - System Testing
    - Performance Testing