Faculty of Engineering and Applied Sciences

SOFE 4630U Cloud Computing

HighD Data Statistics

Deliverable: Cloud Computing Final Project

Date: April 6, 2023

Group Number: T2

Team Members:

Anson Tu: 100655482

Owen Musselman: 100657709

Andrew Mikaeel: 100525236

Abdul Bhutta: 100785884

Background
This project focuses on some of the derivation of values, and statistics using the highD dataset. The lane changes, traffic in each direction, and mean velocity of the vehicles on the highway are used to notify a user of the service if the highway is in a safe, uncongested state. This data is processed using buckets, dataflow, SQL, and front-end capabilities to deduce the highways safety.

Table 1 Component Requirements

Components Used	
Components	**Description of use**
HighD Dataset	Used the dataset used to analyze the safety of the highway.
Google Dataflow	Usage of dataflow in for batch processing the data used for calculations.
Buckets	Buckets were used for storing the highD dataset in the GCP environment.
MySQL	Used to store the data from the highD dataset so it is readily available for potential future analysis.
Front-end	Used to display the highway safety results.

Process Followed
Dataset: We decided to use the highD dataset ultimately as it is a simple representation of traffic, and is what people are most concerned about when it comes to driving safety. We were focused on certain values within the highD dataset, being the meanXVelocity, numFrames, class, driveDirection, and numLaneChanges. The usage of these data values will be explained in further detail later. **Buckets**: The usage of Buckets in this project was to hold the highD dataset 01_tracksMeta.csv (this was used due to time constraints on the project). With the data set in the bucket, it is available to be read using dataflow when needed. **Dataflow:** Dataflow was then used to do the actual processing of the data. The dataflow reads in the contents of the tracksMeta.csv, where the raw, unprocessed contents are written to an SQL database. Running in parallel to the first SQL write, we filter the data to remove any outlier records that are recorded for less than 50 frames. This data could potentially skew the results, so it was best to filter out vehicles that are caught at the edges of the frame at the start and end of the recording. Once filtered, the required data values are extracted for numLaneChanges, meanXVelocity, driveDirection. Where numLaneChanges is summed so that it can be used in a conditional to check if lane changes are excessive. MeanXVelocity for the data set is averaged, and like the numLaneChanges, is used in a conditional to check if the overall average meanXVelocity is slow, if so, the highway is congested. The driveDirection is summed to evaluate how many vehicles travel in each direction to check the safety and congestion of the highway. The findings of each of these are then combined and sent to a SQL database. **SQL Database**: The purpose of the SQL database is to store the raw, unprocessed data as it is read in from tracksMeta.csv. It is also used to store the resulting statistics. Both are on different tables. **Front-end**: The front-end extracts the resulting statistics from the database, where conditionals are performed on the data to determine the safety and congestion of the highway. These conditionals can be seen below for an in-depth description.

Table 2 Dataflow Jobs

Dataflow	Description
Read from csv	Input reader CSV
Convert to dictionary	A process to convert the CSV data to dictionary
Write Preprocessed Data to SQL	Write the current preprocessed data for future reference
Filter number of frames	Filter the out the number of frames in a car (<50)
Extract class	Extract all the vehicles in the class column
Counting Vehicles in both Directions	Count the vehicles in each direction
Set the key/value pair for each vehicle and directions	Convert the tuples into a dictionary
Sum up values for Car Direction	Total the vehicles
Write the Total Car and Truck direction in each lane to SQL	Write the car and truck values in each direction to the SQL database
Get meanXVelocity	Get the meanXVelocity column
Get mean value	Get the average speed of the whole column
Set the key/value pair for mean value	Convert into a dictionary to be stored in the database
Write the average speed of the highway	Write the average speed to SQL database
Get number of lane changes	Get the numLaneChanges column
Count nonzero values	Count all the values where it is not 0
Set the key/value pair value for each lane	Create a dictionary and keys
Sum up values for Lane Change	Add up the total lane changes that occurred
Write the results for each lane to SQL	Write the lane changes that occurred in each lane to SQL

Snapshots of the Components
Access SQL Database

```
abdul_bhutta18@cloudshell:~ (sofe4630u-finalproject)$ mysql -uusr -psofe4630u -h34.118.155.224
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 28963
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use Readings;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```

Accessing SQL database after writing preprocessed and processed data

Query output

```
| 1680830270.4149027 | Car   | 24.53         | 0              | 1                | 412       |
| 1680830270.4472568 | Truck | 22.11         | 0              | 2                | 458       |
| 1680830270.4789846 | Car   | 24.35         | 0              | 1                | 414       |
| 1680830270.5110307 | Truck | 22.18         | 0              | 2                | 456       |
| 1680830270.544457  | Car   | 36.23         | 0              | 2                | 280       |
| 1680830270.5761702 | Car   | 44.99         | 0              | 1                | 224       |
| 1680830270.6075587 | Car   | 42.92         | 0              | 1                | 236       |
| 1680830270.6415787 | Truck | 21.66         | 0              | 2                | 467       |
| 1680830270.673458  | Car   | 29.49         | 0              | 1                | 337       |
| 1680830270.7053819 | Car   | 32.82         | 0              | 2                | 310       |
| 1680830270.7372806 | Truck | 21.56         | 0              | 2                | 469       |
| 1680830270.7693474 | Car   | 35.43         | 0              | 1                | 286       |
| 1680830270.802718  | Truck | 21.31         | 0              | 2                | 475       |
| 1680830270.8336575 | Car   | 36.56         | 0              | 1                | 277       |
| 1680830270.8659925 | Car   | 21.53         | 0              | 2                | 472       |
| 1680830270.8982887 | Car   | 24.58         | 0              | 1                | 411       |
| 1680830270.9304326 | Car   | 36.49         | 0              | 1                | 278       |
| 1680830270.9622972 | Car   | 25.14         | 0              | 1                | 396       |
| 1680830270.9934678 | Car   | 27.69         | 1              | 1                | 366       |
| 1680830271.0249062 | Car   | 37.34         | 0              | 1                | 272       |
| 1680830271.055603  | Car   | 33.99         | 0              | 2                | 298       |
| 1680830271.087486  | Truck | 21.51         | 0              | 2                | 470       |
| 1680830271.118315  | Car   | 34.05         | 0              | 2                | 293       |
| 1680830271.1497374 | Car   | 34.08         | 0              | 2                | 292       |
| 1680830271.1811986 | Car   | 36.67         | 0              | 1                | 276       |
| 1680830271.2134454 | Car   | 28.46         | 0              | 1                | 355       |
| 1680830271.2455463 | Car   | 32.24         | 0              | 1                | 315       |
| 1680830271.2779083 | Car   | 29.39         | 1              | 2                | 345       |
| 1680830271.309696  | Truck | 23.83         | 0              | 2                | 424       |
| 1680830271.3441432 | Car   | 23.64         | 0              | 1                | 422       |
| 1680830271.3763251 | Truck | 23.29         | 0              | 2                | 435       |
| 1680830271.4084172 | Car   | 33.31         | 0              | 2                | 305       |
| 1680830271.4417539 | Car   | 32.15         | 1              | 1                | 315       |
| 1680830271.4739044 | Car   | 30.67         | 0              | 2                | 331       |
| 1680830271.5050848 | Truck | 23.49         | 0              | 2                | 429       |
| 1680830271.5375347 | Car   | 30.95         | 2              | 1                | 320       |
| 1680830271.5699732 | Car   | 35.05         | 0              | 1                | 289       |
| 1680830271.602905  | Car   | 30.08         | 1              | 1                | 330       |
| 1680830271.6348305 | Car   | 32.18         | 0              | 2                | 315       |
| 1680830271.6664338 | Car   | 30.20         | 0              | 2                | 298       |
| 1680830271.6985452 | Car   | 30.10         | 0              | 1                | 294       |
| 1680830271.7310777 | Car   | 37.42         | 0              | 2                | 271       |
| 1680830271.7655196 | Car   | 33.93         | 0              | 1                | 287       |
| 1680830271.798423  | Car   | 33.66         | 0              | 1                | 256       |
| 1680830271.8310986 | Car   | 36.96         | 0              | 2                | 245       |
| 1680830271.8642383 | Truck | 23.38         | 0              | 1                | 235       |
| 1680830271.896543  | Car   | 37.94         | 0              | 2                | 203       |
| 1680830271.9281898 | Car   | 33.03         | 0              | 1                | 183       |
| 1680830271.9603302 | Car   | 33.20         | 0              | 2                | 171       |
| 1680830271.9925911 | Car   | 32.21         | 0              | 1                | 164       |
| 1680830272.029425  | Car   | 37.75         | 0              | 2                | 156       |
| 1680830272.0632432 | Car   | 34.01         | 0              | 1                | 84        |
| 1680830272.0985687 | Car   | 29.96         | 0              | 1                | 76        |
| 1680830272.1305912 | Truck | 35.29         | 0              | 1                | 32        |
+--------------------+-------+---------------+----------------+------------------+-----------+
1047 rows in set (0.08 sec)
```

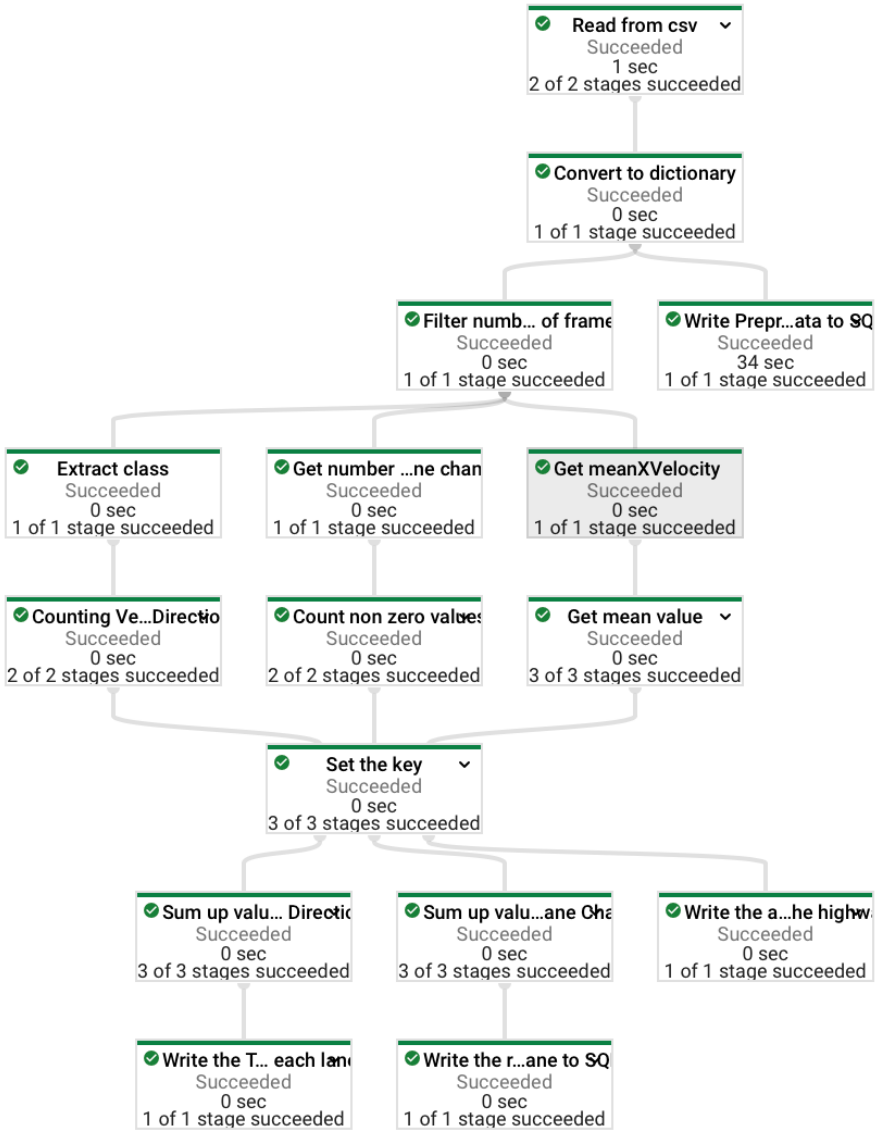
Dataflow before process has started.

Query output for processed data

```
mysql> select * from processedData;
```

```
mysql> select * from processedData;
+------+-----------------+-----------------+-----------------+------------+----------------+------------------+----------------+------------------+
| time | no_lane_changes | one_lane_change | two_lane_change | mean_value | car_direction1 | truck_direction1 | car_direction2 | truck_direction2 |
+------+-----------------+-----------------+-----------------+------------+----------------+------------------+----------------+------------------+
|    0 |             910 |             129 |               5 |    31.1495 |            504 |               88 |            357 |               95 |
+------+-----------------+-----------------+-----------------+------------+----------------+------------------+----------------+------------------+
1 row in set (0.03 sec)

mysql>
```



Above is the completed dataflow for the data processing. You can see that the data is read from csv, filtered, written to an SQL database before being filtered. Then we extract the data values that we want like numLaneChanges, numFrames, meanXVelocity, driveDirection. The "Set the Keys" creates a dictionary with the keys that are derived from the dataset values, and assigns this key to the corresponding database key, for easier database writing. We then count the lane changes, drive directions (based on car and truck) and then calculate the mean speed on the highway and write the results to an SQL table.

Front-end Highway Congestion Status

Germany Highway Traffic Status

Highway Status

Time Slot	No Lane Changes	One Total Lane Change	Two Total Lane Change	Average Speed of the Highway	Cars Direcion 1	Cars Direction 2	Trucks Direction 1	Trucks Direction 2
8:38 AM - 8:53 AM	910	129	5	31.1495	504	357	88	95

Congestion Status

Time Slot	Total Lane Changes	Average Highway Speed	Direction 1 Traffic Status	Direction 2 Traffic Status
8:38 AM - 8:53 AM	**134**	**31.1495**	**Traffic Congested**	**Traffic Congested**

Above shows the output of the congestion status of the highway being analyzed. We can see that the highway is currently considered congested at 8:38 AM – 8:53 AM, due to the substantial number of vehicles, lane changes, and mean velocity being low that were derived.

Encountered Difficulties

Some difficulties were encountered. The first difficulty was that we first planned to implement the project through Google Pub/Sub Stream processing. This showed not to be impossible with our current understanding of every cloud technology, with the time we had to work, so we had to change gears and pursue batch-processing. Another difficulty was writing to the SQL database, as there were issues with duplicate keys, and data getting overwritten. An additional difficulty that we encountered was that we had forgotten that batch processing could be run locally. We ended up debugging using dataflow runner which took ~5 minutes to begin the process and ended up wasting valuable time.

Future Implementation
For future implementation/changes that can be made to the safety/congestion tool would be to aggregate each tracksMeta.csv data set from highD, and perform the analysis on the continuous highway. We could also add an iterator to read from each tracksMeta.csv files instead of having to go through each one.

Conclusion
After this project's completion, we created a simple statistical analysis tool that works on the highD datasets. This was accomplished through Buckets, Dataflow (batch processing), SQL, and front-end implementation to analyze and evaluate road safety and congestion while the dataset was recorded. We gained more knowledge on how batch processing using Dataflows and Buckets work with csv data and how to create a pipeline to interpret how to extract and interact with this data to produce the desired safety and congestion results.

[The rest of this page has been intentionally left blank]

Appendix A – Code

```
import argparse
import json
import logging
import os
import apache_beam as beam
import tensorflow as tf
from apache_beam.options.pipeline_options import PipelineOptions
from apache_beam.options.pipeline_options import SetupOptions
from beam_nuggets.io import relational_db
import time

#Create a batch pipeline to read from the pubsub topic and write to the database
def run(argv=None, save_main_session=True):
  parser = argparse.ArgumentParser()
  known_args, pipeline_args = parser.parse_known_args(argv)

  pipeline_options = PipelineOptions(pipeline_args)
  pipeline_options.view_as(SetupOptions).save_main_session = save_main_session

  #read csv file from the bucket and extract the class column using batch pipeline
  with beam.Pipeline(options=pipeline_options) as p:
    output_config = relational_db.SourceConfiguration(
    drivername='mysql+pymysql',
    host='34.118.155.224',
    port=3306,
    username = 'usr',
    password = 'sofe4630u',
    database = 'Readings'
    )

    table_config = relational_db.TableConfiguration(
        name='processedData',
        create_if_missing=True,
        primary_key_columns=['time']
    )

    table_config2 = relational_db.TableConfiguration(
        name='highD',
        create_if_missing=True,
        primary_key_columns=['time']
    )

    def sumDict(values):
        sumDicts={'time': 0, 'no_lane_changes': 0, 'one_lane_change': 0, 'two_lane_change': 0}
        for v in values:
            for key in v.keys():
                sumDicts[key]+=v[key]
        return sumDicts

    def sumDict2(values):
        sumDicts={'time': 0, 'car_direction1': 0, 'car_direction2': 0, 'truck_direction1': 0, 'truck_direction2': 0}
        for v in values:
            for key in v.keys():
                sumDicts[key]+=v[key]
        return sumDicts

#id width   height  initialFrame    finalFrame  numFrames   class   drivingDirection    traveledDistance    minXVelocity
maxXVelocity    meanXVelocity   minDHW  minTHW  minTTC  numLaneChanges

    data = (p | 'Read from csv' >> beam.io.ReadFromText('gs://sofe4630u-finalproject-bucket/01_tracksMeta.csv',
skip_header_lines=1)
    | 'Convert to dictionary' >> beam.Map(lambda x: {"time": time.time(),'numFrames': x.split(',')[5], 'class': x.split(',')[6],
'drivingDirection': x.split(',')[7], 'meanXVelocity': x.split(',')[11], 'numLaneChanges': x.split(',')[15]})
    )

    #Write to text file in the bucket
    #data | 'Write to text' >> beam.io.WriteToText('gs://sofe4630u-finalproject-bucket/Output.txt')
    data | "Write Preprocessed Data to SQL" >> relational_db.Write(source_config=output_config, table_config=table_config2)

    #Filter the data to be less than 50 frames and write to text file in the bucket
    filteredData = (data | 'Filter number of frames' >> beam.Filter(lambda x: int(x['numFrames']) > 50)
    )

    #From filtered data dictionary, extract the class column and driving direction column and extract drivingDirection to
calculate the number of cars and trucks in each direction
    classData = (filteredData | 'Extract class' >> beam.Map(lambda x: (x['class'], x['drivingDirection']))
    )

    countDirData = (classData | 'Counting Vechicles in both Directions' >> beam.combiners.Count.PerElement()
```

```python
#set keys
    | 'Set the key/value pair for each vehicle and directions' >> beam.Map(lambda x: {
            'time': 0,
            'car_direction1': x[1] if x[0][0] == 'Car' and x[0][1]=='1' else 0,
            'car_direction2': x[1] if x[0][0] == 'Car' and x[0][1]=='2' else 0,
            'truck_direction1': x[1] if x[0][0] == 'Truck' and x[0][1]=='1' else 0,
            'truck_direction2': x[1] if x[0][0] == 'Truck' and x[0][1]=='2' else 0
    }) | 'Sum up values for Car Direction' >> beam.CombineGlobally(sumDict2)

    #Write the result to a text file in the bucket
    | "Write the Total Car and Truck direction in each lane" >> relational_db.Write(source_config=output_config,
table_config=table_config))
    #Write the countDirData to a text file in the bucket
    #countDirData | 'Write countdir to text' >> beam.io.WriteToText('gs://sofe4630u-finalproject-bucket/countdir.txt')

    #Get the meanxvelocity column from the filtered data and convert the values to float
    meanXVelocity = (filteredData | 'Get meanXVelocity' >> beam.Map(lambda x: float(x['meanXVelocity']))
        #Get the mean value of the meanXVelocity column
      | 'Get mean value' >> beam.combiners.Mean.Globally()

      | 'Set the key/value pair for mean value' >> beam.Map(lambda x: {'time':0,'mean_value': x})
      #write the mean value to the database
      | "Write the average speed of the highway" >> relational_db.Write(source_config=output_config, table_config=table_config)
    )

    #Get the numLaneChanges column and convert to int
    numLaneChanges = (filteredData | 'Get number of lane changes' >> beam.Map(lambda x: int(x['numLaneChanges']))

    #Calculate the number of non zero values in the numLaneChanges column and for each element in the PCollection put the value
in a dictionary with the key being the number of non zero values
    | 'Count non zero values' >> beam.combiners.Count.PerElement()

    #set keys
    | 'Set the key/value pair value for each lane' >> beam.Map(lambda x: {
            'time': 0,
            'no_lane_changes': x[1] if x[0] == 0 else 0,
            'one_lane_change': x[1] if x[0] == 1 else 0,
            'two_lane_change': x[1] if x[0] == 2 else 0,
    }))

    #sum the values for each key
    numLaneChanges= (numLaneChanges | 'Sum up values for Lane Change' >> beam.CombineGlobally(sumDict)#.without_defaults()

    #write to text file in the bucket
    #| 'Write to text' >> beam.io.WriteToText('gs://sofe4630u-finalproject-bucket/laneChanges.txt')

    #Write the result to a text file in the bucket
    | "Write the results for each lane to SQL" >> relational_db.Write(source_config=output_config, table_config=table_config)
    )

if __name__ == '__main__':
  logging.getLogger().setLevel(logging.INFO)
  run()
```

Appendix B – Links

GitHub:

https://github.com/abdulbhutta/HighD-Congestion

Demo Video:

https://drive.google.com/file/d/1uh8s-8wBkWs3eR8nSi_jj0UdhPIuwi0z/view?usp=share_link

Presentation Video:
https://drive.google.com/file/d/1CA0zvaalOviEfxvH_K6k1uU7R1x029BZ/view?usp=share_link

Link to Front-end:

https://sofe4630finalproject.azurewebsites.net/