## Introduction

This main aim of the project is to predict future traffic conditions based on historical and present data. Driving situations varies for different places and also different times. Driving conditions identified by some important features can be studied to make future predictions. These circumstances vary for different transportation systems like public or private. This analysis is helpful in many areas like intelligent transportation systems and safety. Mathematical modelling and visualization patterns might reveal some important information about the hazardous condition in the city. In this project we are going to apply K-means clustering algorithm to create similarity

groups to recognize patterns.

## K means Clustering

## K-Means is a popular method for clustering. Clustering methods are preferred when there is no response variable to be predicted but attribute divided into groups. The clusters hopefully will represent same characteristics. The data points assigned to the cluster should have a strong resemblance among them. This section describes K-means algorithm. First we will have to choose number of clusters K. One way to find out best value, by running K-means for number of K values and know which K has minimum squared errors. It is the sum of squared distance all points from the centroid. This method is called Elbow method. When we draw a graph between Squared errors to some clusters if there is no significant drop in errors even if we increase K, it represents best K as value. When we apply K-means algorithm it ran iteratively and calculates best cluster representation. In each iteration sum of squared distances from centroids will be calculated and minimized. By doing this, a new centroid determined, and another iteration starts. This process repeats until there is no change in centroid positions. Algorithm converge when it found a best cluster.

## PySpark

## We are going to use Python with the spark as our work environment. We develop and deploy the application using Zeppelin. Python is easy to learn and a highly productive language where we can do more data analytics tasks quickly. Spark is cluster computing framework designed to handle large datasets. Spark aims to cover wide range of workloads in one place. It reduces the burden of maintaining different tools. RDD is a core data structure in Spark which provides in-memory computation for many general usages and also provides fault tolerance by making RDD coarse-grained. Pyspark is spark framework for Python language. PySpark can be started just like Spark. When PySpark shell opened, spark context created. Objects communicate with the cluster by using Spark Context.Spark, a framework designed for in-memory cluster computing. It mainly supports iterative algorithms. Spark allocate nodes automatically, with only small changes by the user in parametrization part. We implemented the traffic estimation algorithm in Spark.

## Traffic Data

Aarhus city traffic data is a collection of datasets of vehicle traffic, observed between two points over a period of six months. There are 450 observation points. The data is available in CSV format and semantically annotated format(TTL) using the city pulse information model. which show information about the data streams, position of each of the two sensors in the dataset, distance in meters, type of road, etc.

**Meta Data:**

* POINT\_1\_STREET
* DURATION\_IN\_SEC
* POINT\_1\_NAME
* POINT\_1\_CITY
* POINT\_2\_NAME
* POINT\_2\_LNG
* NDT\_IN\_KMH
* POINT\_2\_POSTAL\_CODE
* POINT\_2\_COUNTRY
* POINT\_1\_STREET\_NUMBER
* ORGANISATION
* POINT\_1\_LAT
* POINT\_2\_LAT
* POINT\_1\_POSTAL\_CODE
* POINT\_2\_STREET\_NUMBER
* extID
* ROAD\_TYPE
* POINT\_1\_LNG
* REPORT\_ID
* POINT\_1\_COUNTRY
* DISTANCE\_IN\_METERS
* REPORT\_NAME
* RBA\_ID
* \_id

**Data:**

1. status,
2. AvgMeasuredTime,
3. avgSpeed exitID,
4. medianMeasuredTime,
5. TIMESTAMP,
6. vehicleCount,
7. \_id,
8. REPORT\_ID

## This information stored in two data sets. One is meta-data set another one is temporal data. Each record in the dataset is an observation of how many vehicles passed between two points and its average speed with in five-minute interval. Meta data consists of geographic details such as Latitude and longitude of each road and exit information. These two datasets combined with report\_ID. By running analytics on this data, we will be able to find out some patterns trends of traffic in past for different situations like different weather conditions, in major city events, peak times, holiday period.

## Feature Evaluation

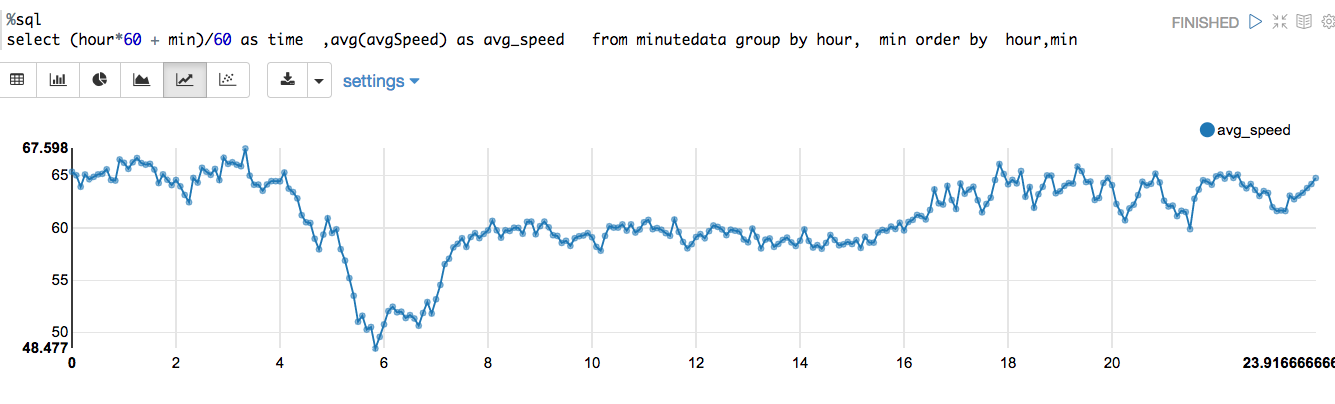
The model we choose for this data is Unsupervised learning. Since there is no response variable to predict and not sure clusters types, it is the best option at this point of time. Clustering divides data into groups. With the help of predicted groups, we will be able to learn, how the traffic might be in future.

Steps fallowed.:

**Pre Process data**.

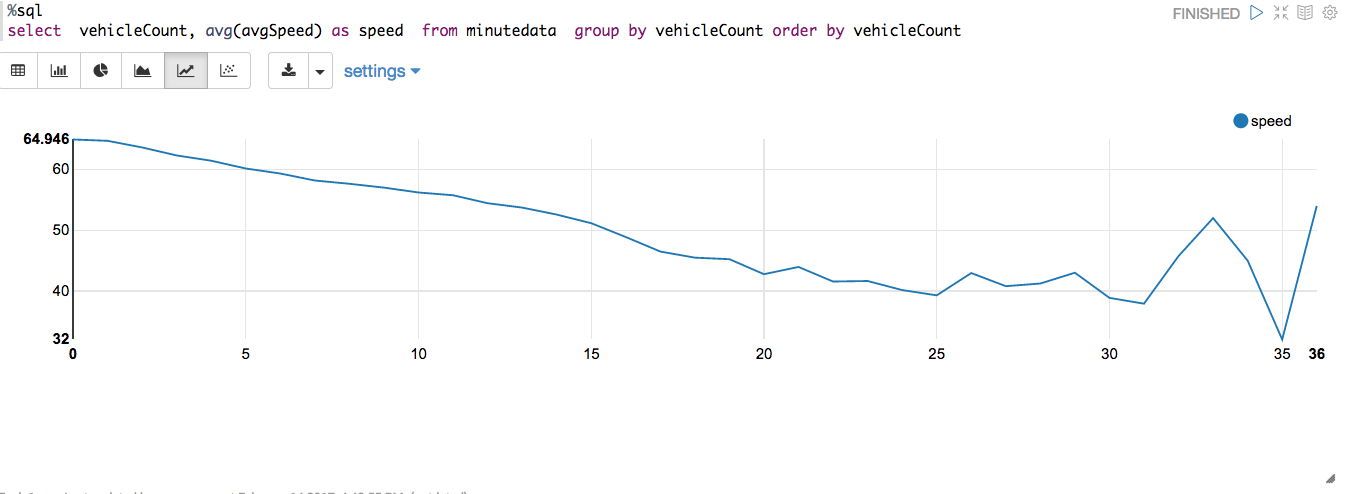
* + - Load Meta data and Time variant data
    - Remove NA’s
    - Merge both datasets by REPORT\_ID
    - Remove duplicate columns
    - Convert string to Time stamp.
    - Add column hourly which is extracted from timestamp.
    - Convert date set to make average speed by hourly.
    - Draws graphs to find patterns.

Both data sets need to be merged because Metadata contains geographical information. Road data contain time variant traffic information. In order to make clusters we need both information to be exists in one single dataset.



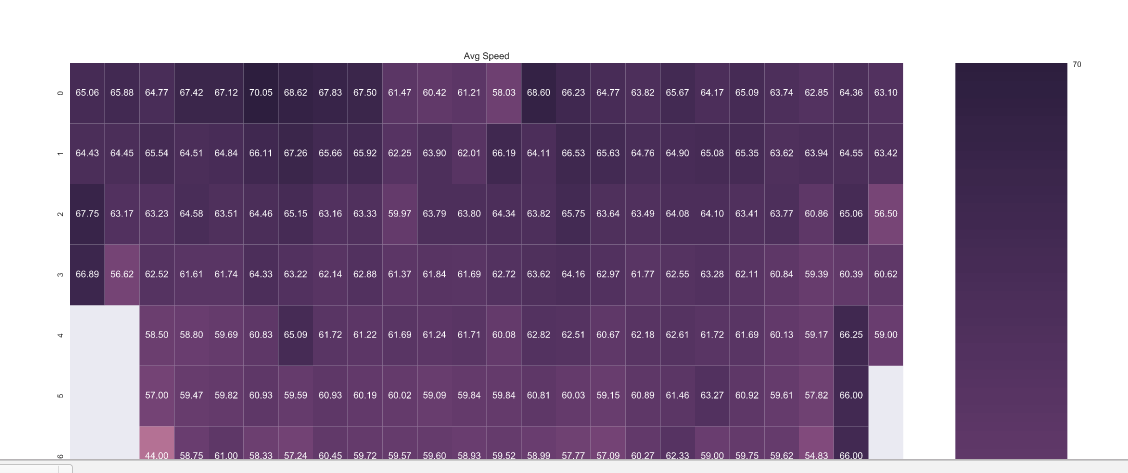
**Fig:1**

* Fig 1 shows time series graph for sample data. We can observe, difference in traffic speed for day and night.



**Fig:2**

* Fig 2 Shows there is a strong relationship between number of vehicles in the street with in five minutes time period and average speed. We can eliminate either one of the variable to build a model.



**Fig :3**

* Fig 3 is a heat map. X-axis represents hour of the day. Y-Axis represents Vehicle count. Each cell is for Average speed. By observing we identify clearly, in a day average speed, vehicles travelled in the street are more compared to night.
* Based on research, attributes which are unnecessary are dropped out. Once we preprocess the data, our final feature set looks as fallows.

*root*

*|-- hour: integer (nullable = true)*

*|-- POINT\_1\_LAT: double (nullable = true)*

*|-- POINT\_1\_LNG: double (nullable = true)*

*|-- POINT\_2\_LAT: double (nullable = true)*

*|-- POINT\_2\_LNG: double (nullable = true)*

*|-- mean: double (nullable = true)*

**Hour**: Hour of the day extracted from timestamp.

**Mean:** Average of average speed group by hourly.

**POINT\_1\_LAT**: Latitude of street starting point.

**POINT\_1\_LNG**: Longitude of street stating point

**POINT\_2\_LAT**: Latitude of street ending point

**POINT\_2\_LNG**: Longitude of street ending point

## 

**Fig:4** Formatted data set.

This data set much cleaner and concise. It contains much smaller foot print compared to original data sets. The below figure shows how the sample data for clustering looks like in Zeppelin.



**Fig :5**

**Fig5**: Explains process fallowed, to convert raw data into spark Data Frame object.

*import pandas as*

*import numpy as np*

*from \_\_future\_\_ import division, print\_function*

***#Loading Meta Data***

*meta\_df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/Users/jyothi/Desktop/capstone/meta/trafficMetaData.csv")*

Meta data loaded as PySpark data frame. data frames, are an abstraction for tables in Python. Data frames support operations similar to relational tables, and also expose them as functions in a procedural language First data loaded into spark as data frame objects. Spark Data Frame objects can be parallelized for distributed parallel processing. Data types converted from string to float and timestamp. Dataframe API similar to Pandas Dataframe.

*meta\_df = meta\_df.drop('extID')*

*meta\_df = meta\_df.drop('\_id')*

***#Loading Data***

*road\_df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/Users/jyothi/Desktop/capstone/\*.csv")*

*#****Merge Both DF***

*merged\_df = road\_df.join(meta\_df, (road\_df.REPORT\_ID == meta\_df.REPORT\_ID) ).drop(meta\_df.REPORT\_ID)*

Both data frames contain REPORT\_ID which will be used to merge both datasets.

*rd\_df = road\_merged\_df.select([c for c in road\_merged\_df.columns if c not in*

*{'DURATION\_IN\_SEC','DISTANCE\_IN\_METERS','POINT\_2\_NAME'}])*

***Converting String to TimeStamp***

*format = "yyyy-MM-dd'T'HH:mm:ss"*

*rd\_df = rd\_df.select('avgMeasuredTime','avgSpeed','TIMESTAMP','vehicleCount','REPORT\_ID','POINT\_2\_LNG','POINT\_1\_LAT','POINT\_2\_LAT','POINT\_1\_LNG', from\_unixtime(unix\_timestamp('TIMESTAMP',format)).cast("timestamp").alias('date'))*

When we load the data, timestamp column is in string format. We need to convert that to timestamp, for further data processing.

***Aggregate Hourly data***

*from pyspark.sql.functions import hour, mean*

*rd\_data\_sel = rd\_data\_selected.select('avgSpeed','POINT\_2\_LNG','POINT\_1\_LAT','POINT\_2\_LAT','POINT\_1\_LNG', F.hour('date').alias('hour') )*

By converting the data frame into timestamp we will be able to abstract data by hour, minute, or day of the week for all kinds of temporal data distributions.

***Round data***

*from pyspark.sql.functions import col, unix\_timestamp, round*

*#rd\_data\_sel.groupBy("hour").agg(mean("avgSpeed").alias("meanSpeed"),round((mean("vehicleCount").alias("meanveh")).show()),2)*

*rd\_data\_set = rd\_data\_sel.groupBy('hour','POINT\_1\_LAT' , 'POINT\_1\_LNG','POINT\_2\_LAT','POINT\_2\_LNG').agg(mean('avgSpeed').alias("mean"))*

*rd\_data\_set =rd\_data\_set.withColumn("mean", round(rd\_data\_set.mean, 3))*

*rd\_data\_set =rd\_data\_set.withColumn("POINT\_1\_LNG", round(rd\_data\_set.POINT\_1\_LNG, 3))*

*rd\_data\_set =rd\_data\_set.withColumn("POINT\_2\_LNG", round(rd\_data\_set.POINT\_2\_LNG, 3))*

*rd\_data\_set =rd\_data\_set.withColumn("POINT\_1\_LAT", round(rd\_data\_set.POINT\_1\_LAT, 3))*

*rd\_data\_set =rd\_data\_set.withColumn("POINT\_2\_LAT", round(rd\_data\_set.POINT\_1\_LAT, 3))*

Data needs to be rounded for a fewer decimal places otherwise clusters might not prove proper results.

***sklearn Kmeans***

*data\_for\_clustering = rd\_data\_set.toPandas()*

*#del data\_for\_clustering['REPORT\_ID']*

*#data\_for\_clustering = data\_for\_clustering.drop('TIMESTAMP')*

*data\_for\_clustering\_matrix = data\_for\_clustering.as\_matrix()*

*# investigate alternative numbers of clusters using silhouette score*

*silhouette\_value = []*

*k = range(2,20) # look at solutions between 2 and 20 clusters*

*for i in k:*

*clustering\_method = KMeans(n\_clusters = i , random\_state = 9999)*

*clustering\_method.fit(data\_for\_clustering\_matrix)*

*labels = clustering\_method.predict(data\_for\_clustering\_matrix)*

*silhouette\_average = silhouette\_score(data\_for\_clustering\_matrix, labels) silhouette\_value.append(silhouette\_average)*

Right now we are applying sklearn K-means on pandas Data Frame. Future implementation will be using Spark Mlib when we build model based on complete set of data.

K-Means is one of the fastest clustering algorithms. The Silhouette value calculated by using mean within cluster distance x and the mean nearest cluster distance y for each data point. The Silhouette value for a point is (y - x) / max(x, y). y is the distance between a data point and the nearest cluster that the point is not a part of. This function returns the mean Silhouette Coefficient over all samples. The least value is -1 and highest value is 1. 1 indicates the best value and near to 0 indicate overlapping clusters. When the points assigned to wrong groups, the values are Negative. In our case when K value 2 or 3 the Silhouette Coefficient value 0.68. It concludes that we might consider taking either two or three clusters. **

## Conclusions

Going further, research deeper these clusters might reveal patterns like for example, some road points are the higher influence on other road points. We consider predicting by carefully observing cluster by cluster. We have to do more research whether targeted road clustering captures the strong geographical influence between different road points. Another interesting thing to find out is that the number of clusters at peak hours. Based on these observations, we may be able to conclude that the groups identified by Algorithm can reflect the actual traffic situation. If this is not the case, we may have to check some other models like Expectation–maximization algorithm (EM) Method, Neural nets or hierarchical clustering. Sparks’s built in module for machine learning MLlib, is the best choice to implement these algorithms. We implemented the traffic estimation algorithm in Spark, a framework for in-memory cluster computing that was designed specifically to support iterative algorithm. The proposed framework using Spark enables the automatic allocation of these nodes at scale, with only change a few parametrizations on the part of the user, a capability that has a high potential impact in the field of traffic monitoring.