```
import pandas as pd
from pyspark.sql.types import StringType
from pyspark import SQLContext
import numpy as np  # arrays and numerical processing
from sklearn.cluster import KMeans  # cluster analysis by partitioning
from sklearn.metrics import silhouette_score as silhouette_score
from __future__ import division, print_function
import plotly.plotly as py
import plotly.graph_objs as go
from datetime import datetime
from pyspark.sql import functions as F
from pyspark.sql.functions import col,udf, unix_timestamp
from pyspark.sql.types import DateType
from pyspark.sql.functions import from_unixtime
from pyspark.sql.functions import col, unix_timestamp, round
road_df1 = sqlContext.read.format("com.databricks.spark.csv").option("header",
    "true").load("/Users/jyothi/Desktop/capstone/capstone/*.csv")
road_df2 = sqlContext.read.format("com.databricks.spark.csv").option("header",
    "true").load("/Users/jyothi/Desktop/capstone/capstone2/*.csv", schema=custschema)
road_df3 = sqlContext.read.format("com.databricks.spark.csv").option("header",
    "true").load("/Users/jyothi/Desktop/capstone/capstone3/*.csv"
road_df.printSchema()
from pyspark.sql.types import *
custschema = StructType([
StructField("status", StringType(), True),
StructField("avgMeasuredTime", StringType(), True),
StructField("avgSpeed", StringType(), True),
StructField("extID", StringType(), True),
StructField("medianMeasuredTime", StringType(), True),
StructField("TIMESTAMP", StringType(), True),
StructField("vehicleCount", StringType(), True),
StructField("_id", StringType(), True),
StructField("REPORT_ID", StringType(), True)
])
road_columns = list(road_df.columns)

meta_df = sqlContext.read.format("com.databricks.spark.csv").option("header",
    "true").load("/Users/jyothi/Desktop/capstone/meta/trafficMetaData.csv")
road_df1 = road_df1.unionAll(road_df2)
road_df1.count()
road_df1 = road_df1.unionAll(road_df3)
road_df1.count()

meta_df = meta_df.drop('extID')
```

```
meta_df = meta_df.drop('_id')
metadata_columns = list(meta_df.columns)
merged_df = road_df1.join(meta_df, (road_df1.REPORT_ID == meta_df.REPORT_ID)
).drop(meta_df.REPORT_ID)
merged_df.printSchema()
type(merged_df)

road_merged_df = merged_df.select([c for c in merged_df.columns if c not
in{'DURATION_IN_SEC','DISTANCE_IN_METERS','POINT_2_NAME'}])
road_merged_df.printSchema()
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'}])
format = "yyyy-MM-dd'T'HH:mm:ss"
rd_df =
rd_df.select('avgMeasuredTime','avgSpeed','TIMESTAMP','vehicleCount','REPORT_ID','P
OINT_2_LNG','POINT_1_LAT','POINT_2_LAT','POINT_1_LNG',
from_unixtime(unix_timestamp('TIMESTAMP',format)).cast("timestamp").alias('date'))
rd_data_selected = rd_df.select([c for c in rd_df.columns if c not in
{'TIMESTAMP','REPORT_ID','avgMeasuredTime','vehicleCount'}])
rd_data_sel =
rd_data_selected.select('avgSpeed','POINT_2_LNG','POINT_1_LAT','POINT_2_LAT','POIN
T_1_LNG', F.hour('date').alias('hour') )
dayofyear(elevDF.date).alias('dt_dayofy'), hour(elevDF.date).alias('dt_hour'),
minute(elevDF.date).alias('dt_min'), weekofyear(elevDF.date).alias('dt_week_no'),
unix_timestamp(elevDF.date).alias('dt_int'))
temp = rd_data_selected.select('avgSpeed','date',F.
weekofyear('date').alias('weekofyear') )
temp1.repartition(1).write.csv("/Users/jyothi/Downloads/timecsv2")
from pyspark.sql.functions import col, unix_timestamp, round
from pyspark.sql.functions import hour, mean
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, 9))
rd_data_set =rd_data_set.withColumn("POINT_2_LNG",
round(rd_data_set.POINT_2_LNG, 9))
rd_data_set =rd_data_set.withColumn("POINT_1_LAT",
round(rd_data_set.POINT_1_LAT, 9))
rd_data_set =rd_data_set.withColumn("POINT_2_LAT",
round(rd_data_set.POINT_1_LAT, 9))
rd_data_set.registerTempTable("latdata")
rd_data_set.show()
library(readr)
```

```
library(dplyr)
library(data.table)
library(sqldf)
library(ggplot2)
library(ggmap)
library(maps)
data1 <- read_csv("/Users/jyothi/Downloads/myc1.csv/road.csv")
tbl1<-subset(data1, data1$hr_group == 1 & data1$mean < 30)
#tbl1<-subset(tbl, data1$mean > 50)
result4<-kmeans(tbl1,4)
tbl1$cluster <- result4$cluster
#View(tbl1)
tbl1$cluster <- as.factor(tbl1$cluster)
mapgilbert <- get_map(location = c(lon = mean(tbl1$long1), lat = mean(tbl1$lat1)),
    zoom = 12, maptype = "satellite", scale = 2)
# plotting the map with some points on it
ggmap(mapgilbert) + geom_point(data = tbl1, aes(x = as.numeric(long1), y =
    as.numeric(lat1),color = cluster, alpha = 0.1), size = 5, shape = 20) + guides(fill=FALSE,
    alpha=FALSE, size=FALSE)
tbl2<-subset(data1, data1$hr_group == 2 & data1$mean < 30)
result4<-kmeans(tbl2,4)
tbl2$cluster <- result4$cluster
#View(tbl2)
tbl2$cluster <- as.factor(tbl2$cluster)
mapgilbert <- get_map(location = c(lon = mean(tbl2$long1), lat = mean(tbl2$lat1)),
    zoom = 11, maptype = "satellite", scale = 2)
# plotting the map with some points on it
ggmap(mapgilbert) + geom_point(data = tbl2, aes(x = as.numeric(long1), y =
    as.numeric(lat1),color = cluster, alpha = 0.1), size = 5, shape = 20) + guides(fill=FALSE,
    alpha=FALSE, size=FALSE)
tbl3<-subset(data1, data1$hr_group == 3 & data1$mean < 30)
result4<-kmeans(tbl3,4)
tbl3$cluster <- result4$cluster
#View(tbl3)
tbl3$cluster <- as.factor(tbl3$cluster)
mapgilbert <- get_map(location = c(lon = mean(tbl3$long1), lat = mean(tbl3$lat1)),
    zoom = 11, maptype = "satellite", scale = 2)
# plotting the map with some points on it
ggmap(mapgilbert) + geom_point(data = tbl3, aes(x = as.numeric(long1), y =
    as.numeric(lat1),colour = cluster, alpha = 0.5), size = 5, shape = 20) + guides(fill=FALSE,
    alpha=FALSE, size=FALSE)
tbl4<-subset(data1, data1$hr_group == 4 & data1$mean < 30)
    result4<-kmeans(tbl4,4)
    tbl4$cluster <- result4$cluster
```

```
View(tbl4)
tbl4$cluster <- as.factor(tbl4$cluster)
mapgilbert <- get_map(location = c(lon = mean(tbl4$long1), lat = mean(tbl4$lat1)), zoom = 12, maptype = "satellite", scale = 2)
# plotting the map with some points on it
ggmap(mapgilbert) + geom_point(data = tbl4, aes(x = as.numeric(long1), y = as.numeric(lat1),colour = cluster , alpha = 0.1), size=5, shape = 20)  + guides(fill=FALSE, alpha=FALSE, size=FALSE)
tbl5<-subset(data1, data1$hr_group == 5 & data1$mean < 30)
result4<-kmeans(tbl5,4)
tbl5$cluster <- result4$cluster
View(tbl5)
tbl5$cluster <- as.factor(tbl5$cluster)
mapgilbert <- get_map(location = c(lon = mean(tbl5$long1), lat = mean(tbl5$lat1)), zoom = 12, maptype = "satellite", scale = 2)
# plotting the map with some points on it
ggmap(mapgilbert)  +  geom_point(data = tbl5, aes(x = as.numeric(long1), y = as.numeric(lat1),colour = cluster, alpha = 0.3),size=5,  shape = 20) + guides(fill=FALSE, alpha=FALSE, size=FALSE)

f.registerTempTable("minutedata")
select  TIMESTAMP , avgSpeed   from minutedata
select (hour*60 + min)/60 as time  ,avg(avgSpeed) as avg_speed   from minutedata group by hour,  min order by  hour,min
import seaborn as sns
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
%pyspark
def show(p):
img = StringIO.StringIO()
p.savefig(img, format='svg')
img.seek(0)
print( "%html " + img.read())
df = sqlContext.sql("select vehicleCount,avg(avgSpeed) as avgSpeed , hour from minutedata  group by hour, vehicleCount")
data = df.toPandas()
value = "avgSpeed"
x = "vehicleCount"
grouping = ["hour"]

heatmap_data = data.pivot_table(values=value, index=x, columns=grouping)
heatmap_data = heatmap_data[0:100]
```

```
135.        a4_dims = (len(heatmap_data.columns),50)
136.        fig, ax = plt.subplots(figsize=a4_dims)
137.        ax.set_title("Avg Speed")
138.        sns.heatmap(heatmap_data, ax=ax, annot=True, fmt=".02f")
139.        import plotly.plotly as py
140.        from plotly.graph_objs import *

141.        import numpy as np
142.        import requests
143.        import copy
144.        import googlemaps

145.        def plot_route_between_streets(address_start, address_end, zoom=3,
    endpt_size=6):
146.        start = ('56.21731711429131', '10.10711200000003')
147.        end = ('56.23490211108693', '10.12519614484404')
148.        start2 =('56.215086','10.139780')
149.        end2 = ('56.215086' ,   '10.105109' )

150.        directions = gmaps.directions(start, end)

151.        steps = []
152.        steps.append(start)  # add starting coordinate to trip

153.        for index in range(len(directions[0]['legs'][0]['steps'])):
154.        start_coords = directions[0]['legs'][0]['steps'][index]['start_location']
155.        steps.append((start_coords['lat'], start_coords['lng']))

156.        if index == len(directions[0]['legs'][0]['steps']) - 1:
157.        end_coords = directions[0]['legs'][0]['steps'][index]['end_location']
158.        steps.append((end_coords['lat'], end_coords['lng']))

159.        steps.append(end)  # add ending coordinate to trip
160.        directions = gmaps.directions(start2, end2)

161.        data = Data([
162.        Scattermapbox(
163.        lat=[item_x[0] for item_x in steps],
164.        lon=[item_y[1] for item_y in steps],
165.        mode='markers',
166.        marker=Marker(
167.        size=[endpt_size] + [4 for j in range(len(steps) - 2)] + [endpt_size]
168.        ),
169.        )
```

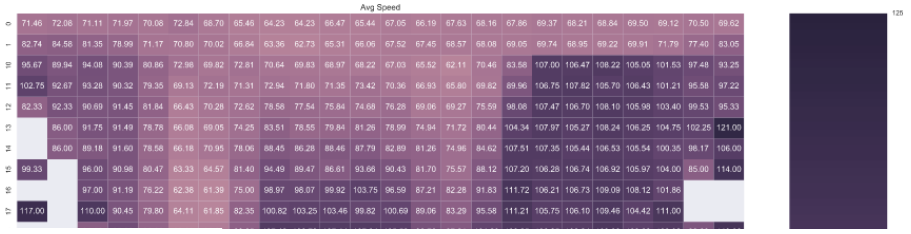
```
170.        ])
171.        layout = Layout(
172.        autosize=True,
173.        hovermode='closest',
174.        mapbox=dict(
175.        accesstoken=mapbox_access_token,
176.        bearing=0,
177.        style='streets',
178.        center=dict(
179.        lat=np.mean([float(step[0]) for step in steps]),
180.        lon=np.mean([float(step[1]) for step in steps]),
181.        ),
182.        pitch=0,
183.        zoom=zoom
184.        ),
185.        )

186.        fig = dict(data=data, layout=layout)
187.        return fig
188.        gmap_api_key = 'AIzaSyCpKt7PVTMeHKPOODBwS3fx3ZJneCNG_w0'

189.        gmaps = googlemaps.Client(gmap_api_key)
190.        address_start = '2410 Marine Avenue'
191.        address_end = '1 Rocket Rd'
192.        zoom=12.2
193.        endpt_size=20
194.        mapbox_access_token =
    'pk.eyJ1IjoicHpoYW8wOTE4IiwiYSI6ImNpdmI2aXY4MjAwdHUyb3AzMzE0bTlrNmwifQ.L0
    X8_PvnvigkDvgsVk-NlA'


195.        figure = plot_route_between_streets(address_start, address_end, zoom=12.2,
    endpt_size=20)

196.        import plotly
197.        from plotly.graph_objs import Scatter, Layout


198.        def plot(plot_dic, height=500, width=500, **kwargs):
199.        kwargs['output_type'] = 'div'
200.        plot_str = plotly.offline.plot(plot_dic, **kwargs)
201.        print('%%angular <div style="height: %ipx; width: %spx"> %s </div>' % (height,
    width,
202.        plot_str))
```

```
203.          print("""%html

<head>
  <script type="text/javascript">
  google.charts.load("current", {packages:["map"]});
  google.charts.setOnLoadCallback(drawChart3);
  function drawChart3() {
    var data = google.visualization.arrayToDataTable([
      ['Lat', 'Long', 'Name'],
      [56.225795, 10.116590, 'Street2'],
      [56.215086, 10.139780, 'Street3'],

    ]);

    var map = new google.visualization.Map(document.getElementById('m_div'));
    map.draw(data, {
      showTooltip: true,
      showInfoWindow: true
    });
  }

  </script>
</head>

<body>
  <div id="m_div" style="width: 400px; height: 300px"></div>
</body>

""")
```

Provide a perfomance analysis of your code (time for each operation, identify the bottlenecks, explain how you made your implementation faster).

1. Initially Started implementing with PySpark in Zepllien. Due to coding constrains shifted to Pandas. Created visualization and Heat-maps using Pandas Plotly and Seaborn.

a.



b.

```
%pyspark

def show(p):
  img = StringIO()
  p.savefig(img, format='svg')
  img.seek(0)
  print( "%html " + img.read())
#show(plt)
```

2. Using SQL did aggregations and again converted back to Pyspark Dataframe objects. These operations done with sample dataset with 6 files.

a.

```
%sql
select (hour*60 + min)/60 as time  ,avg(avgSpeed) as avg_speed   from minutedb group by hour,  min order by  hour,min
```

time	avg_speed
0	77.51182
0.08333	77.56653

3. Implemented Scikit learn Kmeans and created clusters. Tried plotting this data with Plotly and Google maps. To use google maps we need support HTML and Javascript in Zepplien. Researched about Java script and HTML support. Able to plot some static data in google maps. But could not find a way to map dynamic data using google maps.

```
%pyspark
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)
```

```
    random_state=9999, tol=0.0001, verbose=0)
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=8, n_init=10, n_jobs=1, precompute_distances='auto',
```

```
%pyspark
clustering_method = KMeans(n_clusters = 2, random_state = 9999)
clustering_method.fit(data_for_clustering_matrix)
labels = clustering_method.predict(data_for_clustering_matrix)
labels
```

```
        showInfoWindow: true
      });
    }

  </script>
</head>

<body>
  <div id="m_div" style="width: 400px; height: 300px"></di
</body>

""")
```

4. Researched R support in creating clusters and Geolocation data in map
    a. library(readr)
    b. library(dplyr)
    c. library(data.table)
    d. library(sqldf)
    e. library(ggplot2)
    f. library(ggmap)
    g. library(maps)

By using these api able to load data, implement K-means , Plot Cluster data in Map with sample data.

5. Researched to find a way to load entire data at one time, format and cleanup data. Found a solution to load data using Spark data bricks library.

```
%pyspark
road_df1 = road_df1.unionAll(road_df2)
road_df1.count()
```

```
19276612
```

   Merged data contains 10 Million rows. Loading data set is 10 sec and for count operation it took 17 sec. Count is aggregate function need to load entire dataframe into memory. Aggregated data using PySpark aggregate functions. Now this process much more efficient when compared to previous implementation. Each data aggregation took less than 2 minutes. Before we are not able to do with entire data.

6. Grouped data to make a smaller footprint and stored aggregated data into csv format using spark partitions with single partition.
7. Divided data into 4 time zones
    a. 22- 6 (10PM - 6 AM) - block 1
    b. (7AM – 11 AM) - block 2
    c. (12 AM – 4 PM) - block 3
    d. (5 PM - 9PM) – block 4

8. Considered only Mean traffic speed less than 30 miles/hour. The reason behind taking less mean speed is we are trying to research traffic congestions. So less speed mostly because of more traffic.
9. Implemented K-Means to these blocks separately and plotted to find an patterns.
10. 

Below map shown for time zone two. We can see one cluster in this zone (blue color) has more crowded than others. We can expect this problem due to traffic congestion.

Observations:

A. There is significant performance improvement with Spark compared to Pandas implementation. Loading 19 million records done in 10 sec. All aggregate functions are took only 2 to 3 secs time period.
B. R has better mapping support compared to JavaScript and google map.
C. In order to create a User Interface still needs to do more research HTML, google map compatibility with Zeppelin.
D. More research necessary to make conclusion about traffic congestions.