1. import pandas as pd
2. from pyspark.sql.types import StringType
3. from pyspark import SQLContext
4. import numpy as np # arrays and numerical processing
5. from sklearn.cluster import KMeans # cluster analysis by partitioning
6. from sklearn.metrics import silhouette\_score as silhouette\_score
7. from \_\_future\_\_ import division, print\_function
8. import plotly.plotly as py
9. import plotly.graph\_objs as go
10. from datetime import datetime
11. from pyspark.sql import functions as F
12. from pyspark.sql.functions import col,udf, unix\_timestamp
13. from pyspark.sql.types import DateType
14. from pyspark.sql.functions import from\_unixtime
15. from pyspark.sql.functions import col, unix\_timestamp, round
16. road\_df1 = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/Users/jyothi/Desktop/capstone/capstone/\*.csv")
17. road\_df2 = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/Users/jyothi/Desktop/capstone/capstone2/\*.csv", schema=custschema)
18. road\_df3 = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/Users/jyothi/Desktop/capstone/capstone3/\*.csv"
19. road\_df.printSchema()
20. from pyspark.sql.types import \*
21. custschema = StructType([
22. StructField("status", StringType(), True),
23. StructField("avgMeasuredTime", StringType(), True),
24. StructField("avgSpeed", StringType(), True),
25. StructField("extID", StringType(), True),
26. StructField("medianMeasuredTime", StringType(), True),
27. StructField("TIMESTAMP", StringType(), True),
28. StructField("vehicleCount", StringType(), True),
29. StructField("\_id", StringType(), True),
30. StructField("REPORT\_ID", StringType(), True)
31. ])
32. road\_columns = list(road\_df.columns)
33. meta\_df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/Users/jyothi/Desktop/capstone/meta/trafficMetaData.csv")
34. road\_df1 = road\_df1.unionAll(road\_df2)
35. road\_df1.count()
36. road\_df1 = road\_df1.unionAll(road\_df3)
37. road\_df1.count()
38. meta\_df = meta\_df.drop('extID')
39. meta\_df = meta\_df.drop('\_id')
40. metadata\_columns = list(meta\_df.columns)
41. merged\_df = road\_df1.join(meta\_df, (road\_df1.REPORT\_ID == meta\_df.REPORT\_ID) ).drop(meta\_df.REPORT\_ID)
42. merged\_df.printSchema()
43. type(merged\_df)
44. 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'}])
45. road\_merged\_df.printSchema()
46. rd\_df = road\_merged\_df.select([c for c in road\_merged\_df.columns if c not in
47. {'DURATION\_IN\_SEC','DISTANCE\_IN\_METERS','POINT\_2\_NAME'}])
48. format = "yyyy-MM-dd'T'HH:mm:ss"
49. 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'))
50. rd\_data\_selected = rd\_df.select([c for c in rd\_df.columns if c not in
51. {'TIMESTAMP','REPORT\_ID','avgMeasuredTime','vehicleCount'}])
52. 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') )
53. 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'))
54. temp = rd\_data\_selected.select('avgSpeed','date',F. weekofyear('date').alias('weekofyear') )
55. temp1.repartition(1).write.csv("/Users/jyothi/Downloads/timecsv2")
56. from pyspark.sql.functions import col, unix\_timestamp, round
57. from pyspark.sql.functions import hour, mean
58. 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"))
59. rd\_data\_set =rd\_data\_set.withColumn("mean", round(rd\_data\_set.mean, 3))
60. rd\_data\_set =rd\_data\_set.withColumn("POINT\_1\_LNG", round(rd\_data\_set.POINT\_1\_LNG, 9))
61. rd\_data\_set =rd\_data\_set.withColumn("POINT\_2\_LNG", round(rd\_data\_set.POINT\_2\_LNG, 9))
62. rd\_data\_set =rd\_data\_set.withColumn("POINT\_1\_LAT", round(rd\_data\_set.POINT\_1\_LAT, 9))
63. rd\_data\_set =rd\_data\_set.withColumn("POINT\_2\_LAT", round(rd\_data\_set.POINT\_1\_LAT, 9))
64. rd\_data\_set.registerTempTable("latdata")
65. rd\_data\_set.show()
66. library(readr)
67. library(dplyr)
68. library(data.table)
69. library(sqldf)
70. library(ggplot2)
71. library(ggmap)
72. library(maps)
73. data1 <- read\_csv("/Users/jyothi/Downloads/myc1.csv/road.csv")
74. tbl1<-subset(data1, data1$hr\_group == 1 & data1$mean < 30)
75. #tbl1<-subset(tbl, data1$mean > 50)
76. result4<-kmeans(tbl1,4)
77. tbl1$cluster <- result4$cluster
78. #View(tbl1)
79. tbl1$cluster <- as.factor(tbl1$cluster)
80. mapgilbert <- get\_map(location = c(lon = mean(tbl1$long1), lat = mean(tbl1$lat1)), zoom = 12, maptype = "satellite", scale = 2)
81. # plotting the map with some points on it
82. ggmap(mapgilbert) + geom\_point(data = tbl1, aes(x = as.numeric(long1), y = as.numeric(lat1),color = cluster, alpha = 0.1), size = 5, shape = 20) + guides(fill=FALSE, alpha=FALSE, size=FALSE)
83. tbl2<-subset(data1, data1$hr\_group == 2 & data1$mean < 30)
84. result4<-kmeans(tbl2,4)
85. tbl2$cluster <- result4$cluster
86. #View(tbl2)
87. tbl2$cluster <- as.factor(tbl2$cluster)
88. mapgilbert <- get\_map(location = c(lon = mean(tbl2$long1), lat = mean(tbl2$lat1)), zoom = 11, maptype = "satellite", scale = 2)
89. # plotting the map with some points on it
90. ggmap(mapgilbert) + geom\_point(data = tbl2, aes(x = as.numeric(long1), y = as.numeric(lat1),color = cluster, alpha = 0.1), size = 5, shape = 20) + guides(fill=FALSE, alpha=FALSE, size=FALSE)
91. tbl3<-subset(data1, data1$hr\_group == 3 & data1$mean < 30)
92. result4<-kmeans(tbl3,4)
93. tbl3$cluster <- result4$cluster
94. #View(tbl3)
95. tbl3$cluster <- as.factor(tbl3$cluster)
96. mapgilbert <- get\_map(location = c(lon = mean(tbl3$long1), lat = mean(tbl3$lat1)), zoom = 11, maptype = "satellite", scale = 2)
97. # plotting the map with some points on it
98. ggmap(mapgilbert) + geom\_point(data = tbl3, aes(x = as.numeric(long1), y = as.numeric(lat1),colour = cluster, alpha = 0.5), size = 5, shape = 20) + guides(fill=FALSE, alpha=FALSE, size=FALSE)
99. tbl4<-subset(data1, data1$hr\_group == 4 & data1$mean < 30)
100. result4<-kmeans(tbl4,4)
101. tbl4$cluster <- result4$cluster
102. View(tbl4)
103. tbl4$cluster <- as.factor(tbl4$cluster)
104. mapgilbert <- get\_map(location = c(lon = mean(tbl4$long1), lat = mean(tbl4$lat1)), zoom = 12, maptype = "satellite", scale = 2)
105. # plotting the map with some points on it
106. ggmap(mapgilbert) + geom\_point(data = tbl4, aes(x = as.numeric(long1), y = as.numeric(lat1),colour = cluster , alpha = 0.1), size=5, shape = 20) + guides(fill=FALSE, alpha=FALSE, size=FALSE)
107. tbl5<-subset(data1, data1$hr\_group == 5 & data1$mean < 30)
108. result4<-kmeans(tbl5,4)
109. tbl5$cluster <- result4$cluster
110. View(tbl5)
111. tbl5$cluster <- as.factor(tbl5$cluster)
112. mapgilbert <- get\_map(location = c(lon = mean(tbl5$long1), lat = mean(tbl5$lat1)), zoom = 12, maptype = "satellite", scale = 2)
113. # plotting the map with some points on it
114. 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)
115. f.registerTempTable("minutedata")
116. select TIMESTAMP , avgSpeed from minutedata
117. select (hour\*60 + min)/60 as time ,avg(avgSpeed) as avg\_speed from minutedata group by hour, min order by hour,min
118. import seaborn as sns
119. import matplotlib
120. matplotlib.use('Agg')
121. import matplotlib.pyplot as plt
122. %pyspark
123. def show(p):
124. img = StringIO.StringIO()
125. p.savefig(img, format='svg')
126. img.seek(0)
127. print( "%html " + img.read())
128. df = sqlContext.sql("select vehicleCount,avg(avgSpeed) as avgSpeed , hour from minutedata group by hour, vehicleCount")
129. data = df.toPandas()
130. value = "avgSpeed"
131. x = "vehicleCount"
132. grouping = ["hour"]
133. heatmap\_data = data.pivot\_table(values=value, index=x, columns=grouping)
134. 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.L0X8\_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.
   1. 
   2. 
2. Using SQL did aggregations and again converted back to Pyspark Dataframe objects. These operations done with sample dataset with 6 files.
   1. 
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.





1. Researched R support in creating clusters and Geolocation data in map
   1. library(readr)
   2. library(dplyr)
   3. library(data.table)
   4. library(sqldf)
   5. library(ggplot2)
   6. library(ggmap)
   7. library(maps)

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

1. 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.



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.

1. Grouped data to make a smaller footprint and stored aggregated data into csv format using spark partitions with single partition.
2. Divided data into 4 time zones
   1. 22- 6 (10PM - 6 AM) - block 1
   2. (7AM – 11 AM) - block 2
   3. (12 AM – 4 PM) - block 3
   4. (5 PM - 9PM) – block 4
3. 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.
4. Implemented K-Means to these blocks separately and plotted to find an patterns.
5. 

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:**

1. 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.
2. R has better mapping support compared to JavaScript and google map.
3. In order to create a User Interface still needs to do more research HTML, google map compatibility with Zeppelin.
4. More research necessary to make conclusion about traffic congestions.