deriving_landmarks

Don Li

06/06/2020

Purpose

Derive landmark locations and use this to group the data better.

Approaches:

- Use Google Places API or something like that to determine what locations people are going to
- Use statistics

I will use the second one because I don't know how to make an API work.

Data and stuff

K-means

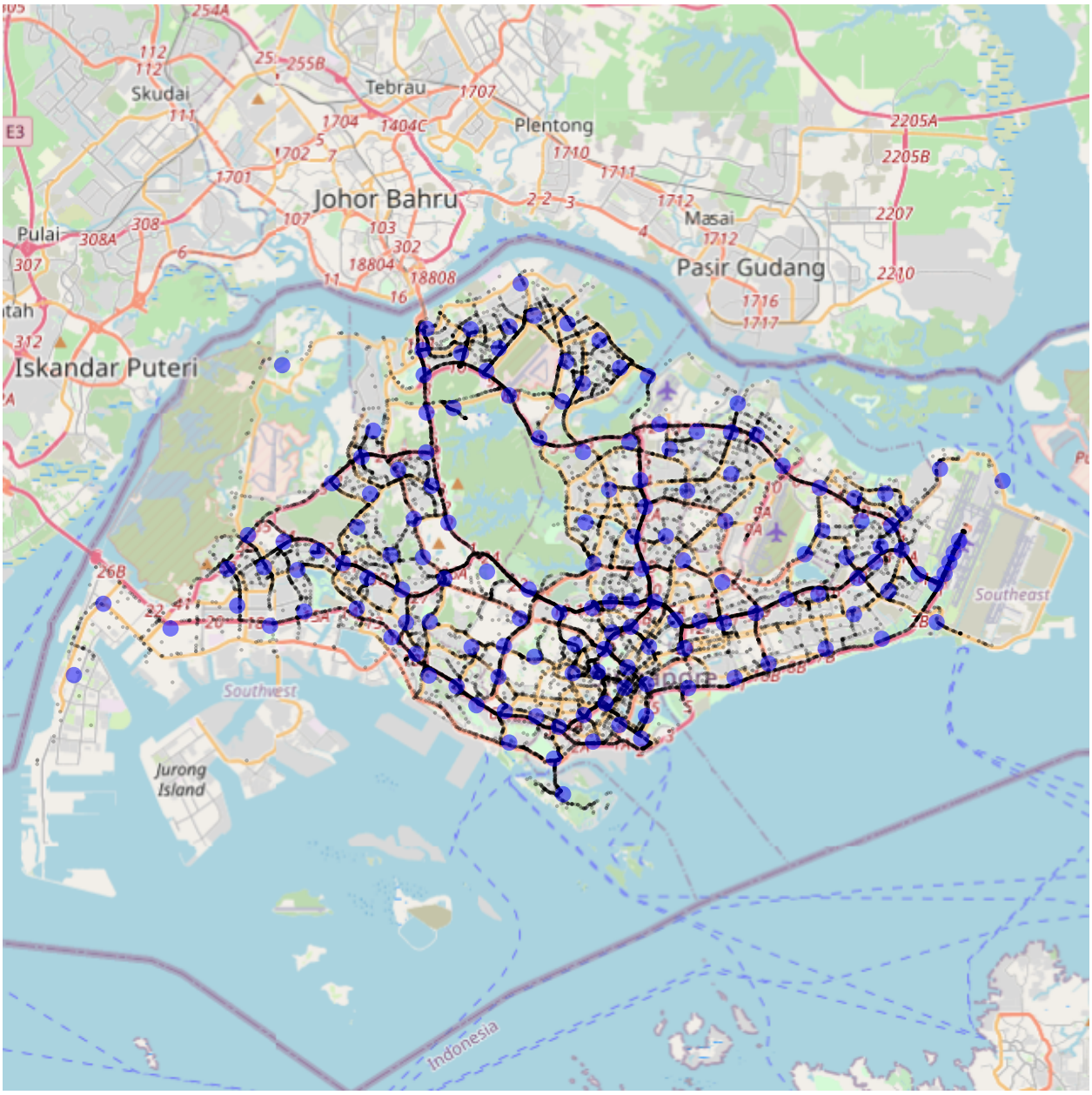
Use k-means clustering (Hartigan-Wong) on the start and end points.

```
load( "G:/azure_hackathon/dataset/trip_summary2_loopytrim.RData" )

predict_kmeans = function(object, newdata){
    centers = object$centers
    n_centers = nrow(centers)
    dist_mat = as.matrix(dist(rbind(centers, newdata)))
    dist_mat = dist_mat[-seq(n_centers), seq(n_centers)]
    max.col(-dist_mat)
}

start_end_data = trip_summary[ ,
    list(
        lat1 = start_y, lng1 = start_x,
        latN = end_x, lngN = end_y
    ), by = "trj_id" ]

kmeans_data = start_end_data[ , {
    cbind(
        lats = c( lat1, latN),
        lngs = c(lng1, lngN )
    ) } ]

kmeans_result = kmeans( kmeans_data, centers = 300, nstart = 1000, iter.max = 1e9 )

save( kmeans_result, start_end_data, kmeans_data,
    file = "G:/azure_hackathon/dataset/landmarks/landmarks_kmeans.RData" )
```

Put these points on a map

```
## sleptTotal= 0
```

```
## [1] "Caution: map type is OpenStreetMap. Until we find the correct projection algorithm, we treat la
## NULL
## NULL
```



K-means with outliers removed

I want to automatically detect high transit destinations. In order to do this, I will filter the points that have no neighbours within 500m.

```
min_dist_km = 0.5
isolated_pts = rep( F, nrow( kmeans_data ) )
for ( i in 1:length(isolated_pts) ){
    if ( i %% 100 == 0 ) print( i )
```

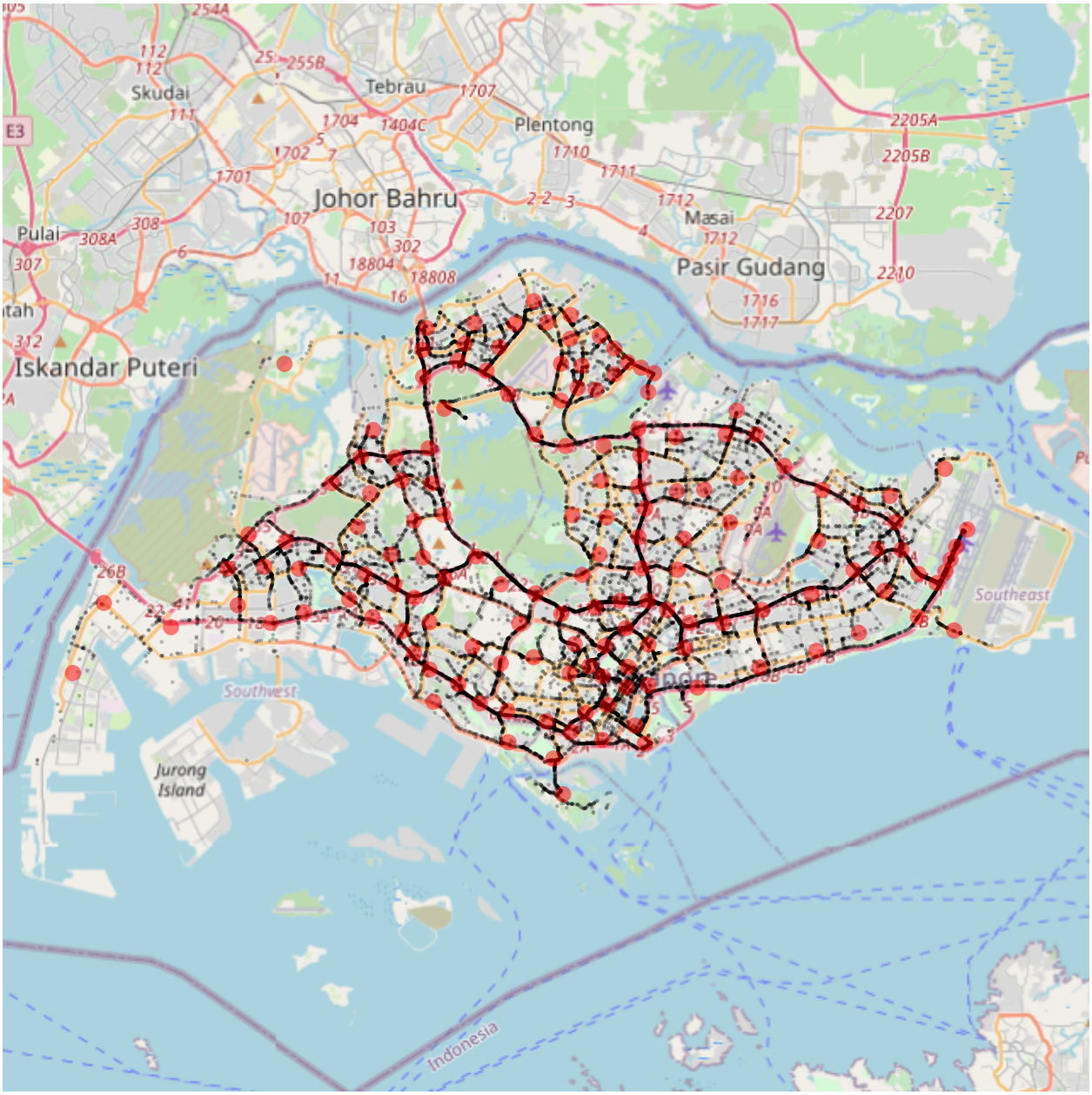
```
    all_haversines = haversine2( kmeans_data[i,,drop = F], kmeans_data )
    isolated = (sum(all_haversines < min_dist_km) - 1 ) == 0
    if ( isolated ){
        isolated_pts[i] = T
    }
}

new_keans_data = kmeans_data[ !isolated_pts, ]

set.seed(312)
kmeans_result2 = kmeans( new_keans_data, centers = 300,
    nstart = 1000, iter.max = 1e9 )
save( new_keans_data, kmeans_result2,
    file = "G:/azure_hackathon/dataset/landmarks/landmarks_kmeans_outliers_rm.RData" )
```

In my opinion, the clustering is nicer, especially around Jurong Island.

```
## [1] "Caution: map type is OpenStreetMap. Until we find the correct projection algorithm, we treat la
```

```
## NULL
```

```
## NULL
```

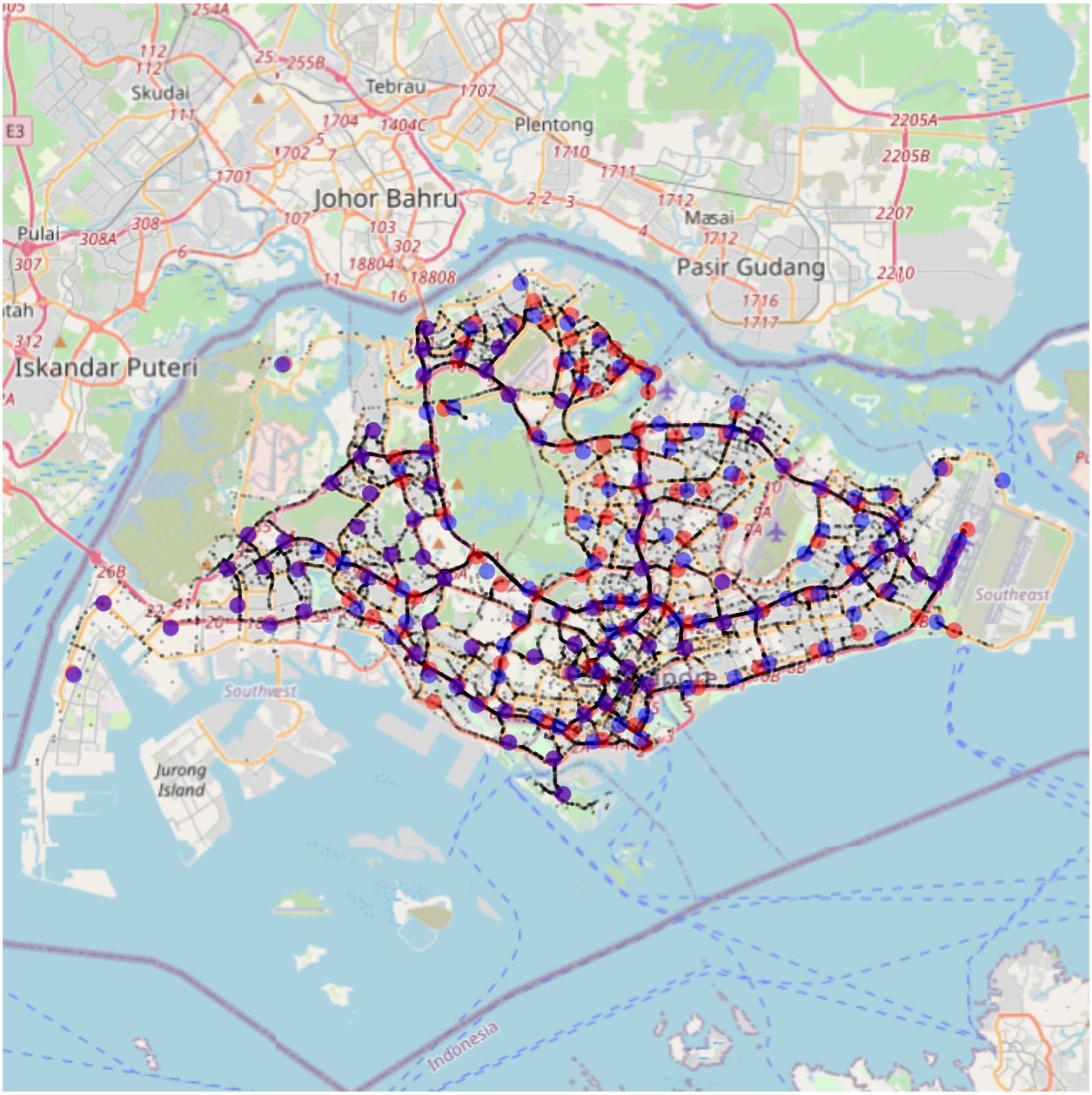


Overlay the two clusters to see how they differ. Picture below. Red is the clustering with the outliers removed. Blue is the clustering with the outliers in (first clustering). Expect some randomness due to the k-means clustering. But, the outliers removed tends to make a lot of random stuff.

```
## [1] "Caution: map type is OpenStreetMap. Until we find the correct projection algorithm, we treat la
```

```
## NULL
```

```
## NULL
```



Stacking clustering algorithms

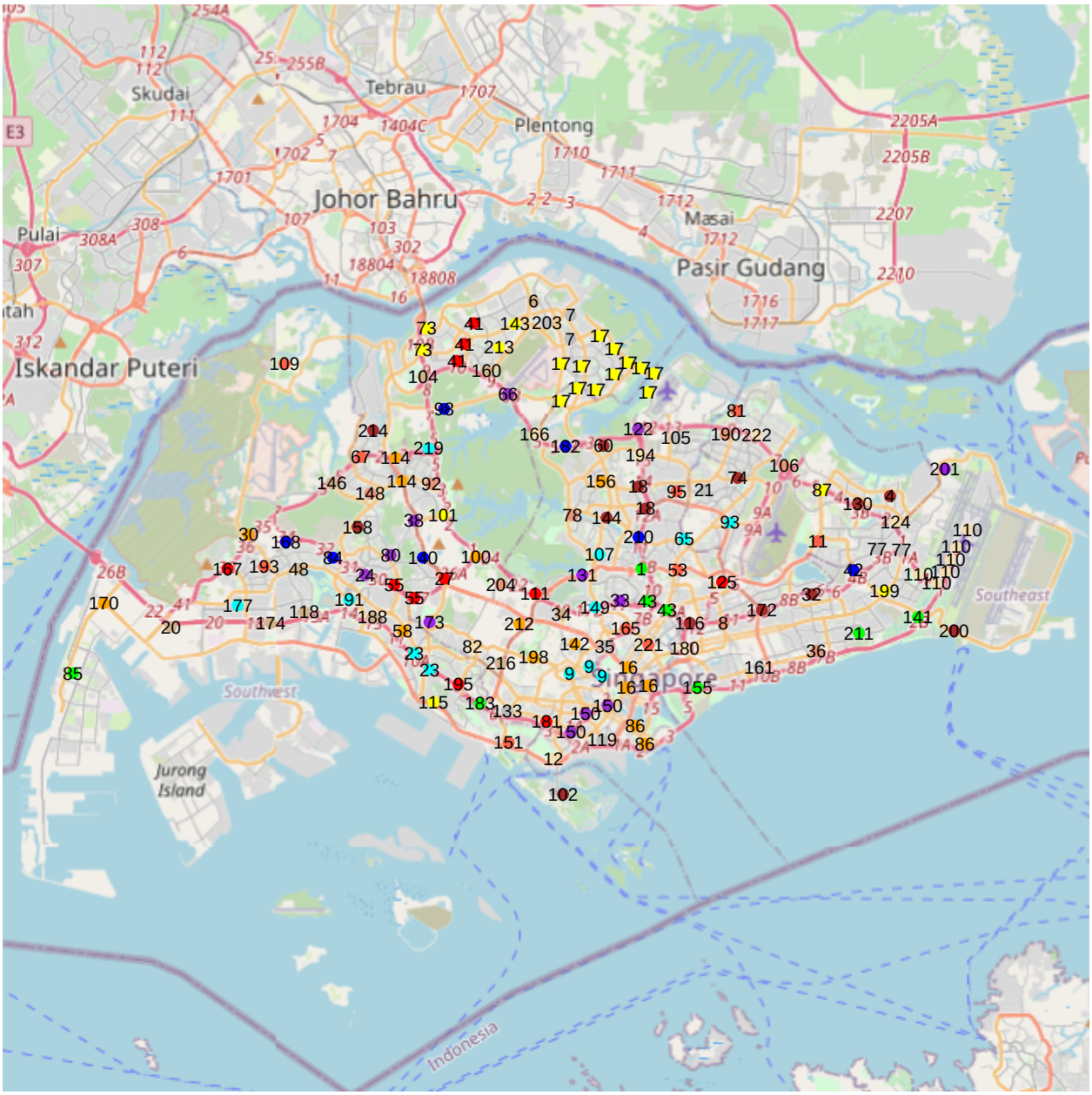
Using the results from K-means (outliers removed), we can link clusters together using DBSCAN. This is useful for points like the airport, where there is a whole collection of points in a row.

The results below show the linkage of the K-means centers using DBSCAN. In particular, look at the orange points at the airport (labeled "2"), also at some points in the city center. We can use this cluster information to classify trips based on start or end.

```
load(  "G:/azure_hackathon/dataset/landmarks/landmarks_kmeans_outliers_rm.RData" )

library( dbscan )
set.seed(42323)
```

```
dbscan_results = dbscan( kmeans_result2$centers, minPts = 1, eps = 0.01 )
k2_centers = data.table(
    cbind( kmeans_result2$centers, size = kmeans_result2$size,
        cluster = dbscan_results$cluster )
)
k2_centers[ , total_size := sum(size), by = "cluster" ]
big_clusters = k2_centers[ total_size > 500 ]

save( dbscan_results, big_clusters,
    file = "G:/azure_hackathon/dataset/landmarks/deriving_landmarks_dbscan.RData" )
```

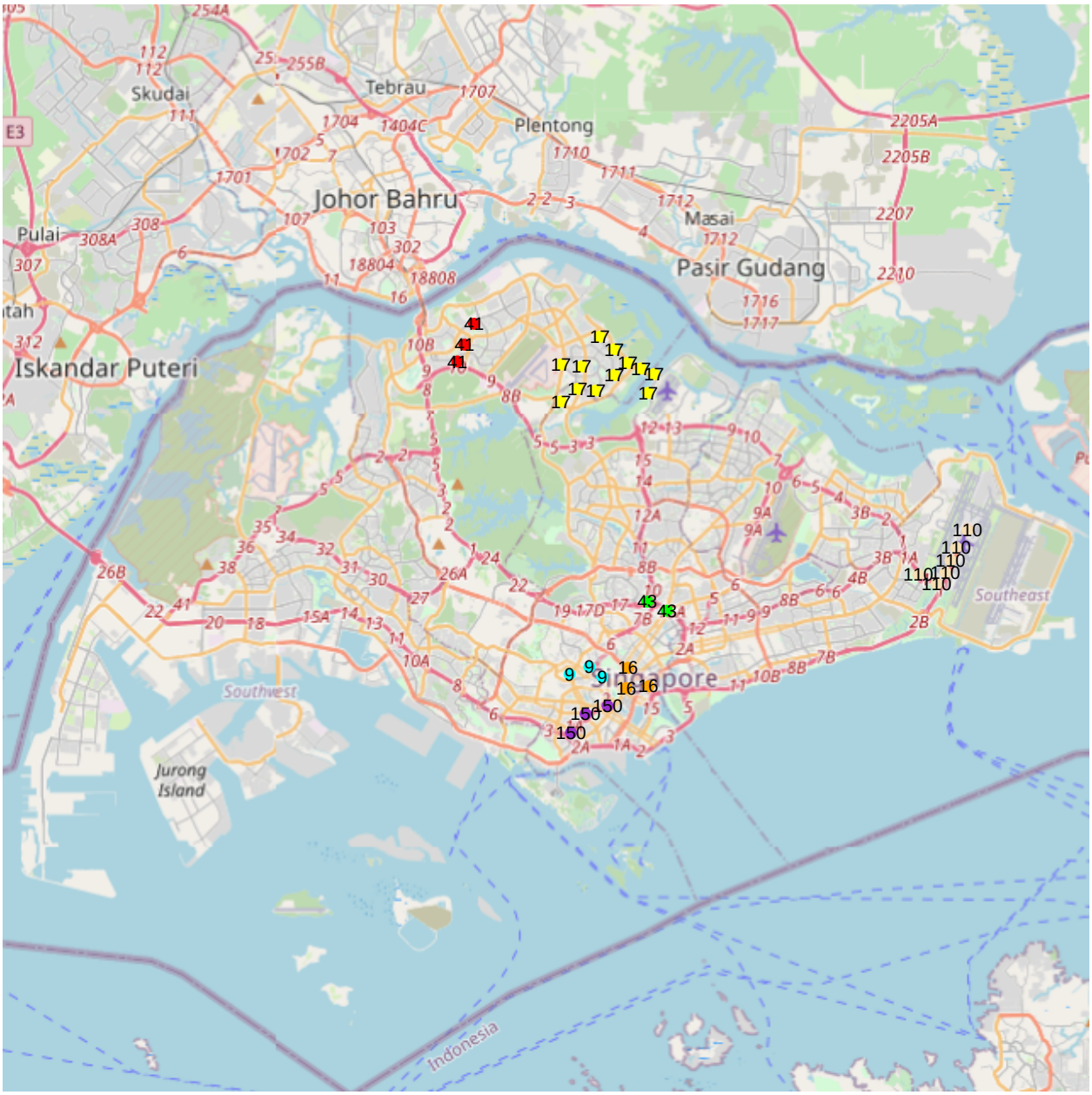
```
## [1] "Caution: map type is OpenStreetMap. Until we find the correct projection algorithm, we treat la
```



Count up the size of each cluster from K-means, summed over the cluster membership from DBSCAN and

plot only the clusters with more than 100 total.

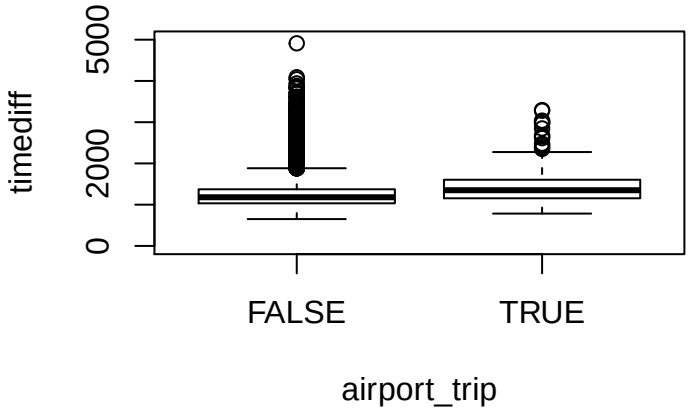
```
## [1] "Caution: map type is OpenStreetMap. Until we find the correct projection algorithm, we treat la
```



Proof of concept

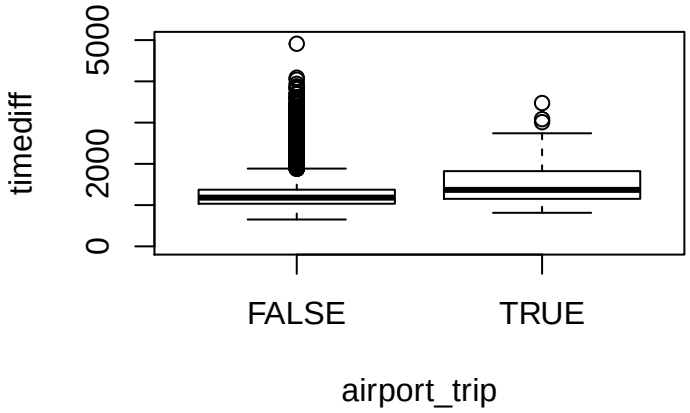
To show that this was not for nothing, we can look at trips to specific places.

We can see below that a trip to the airport does take longer than other trips.



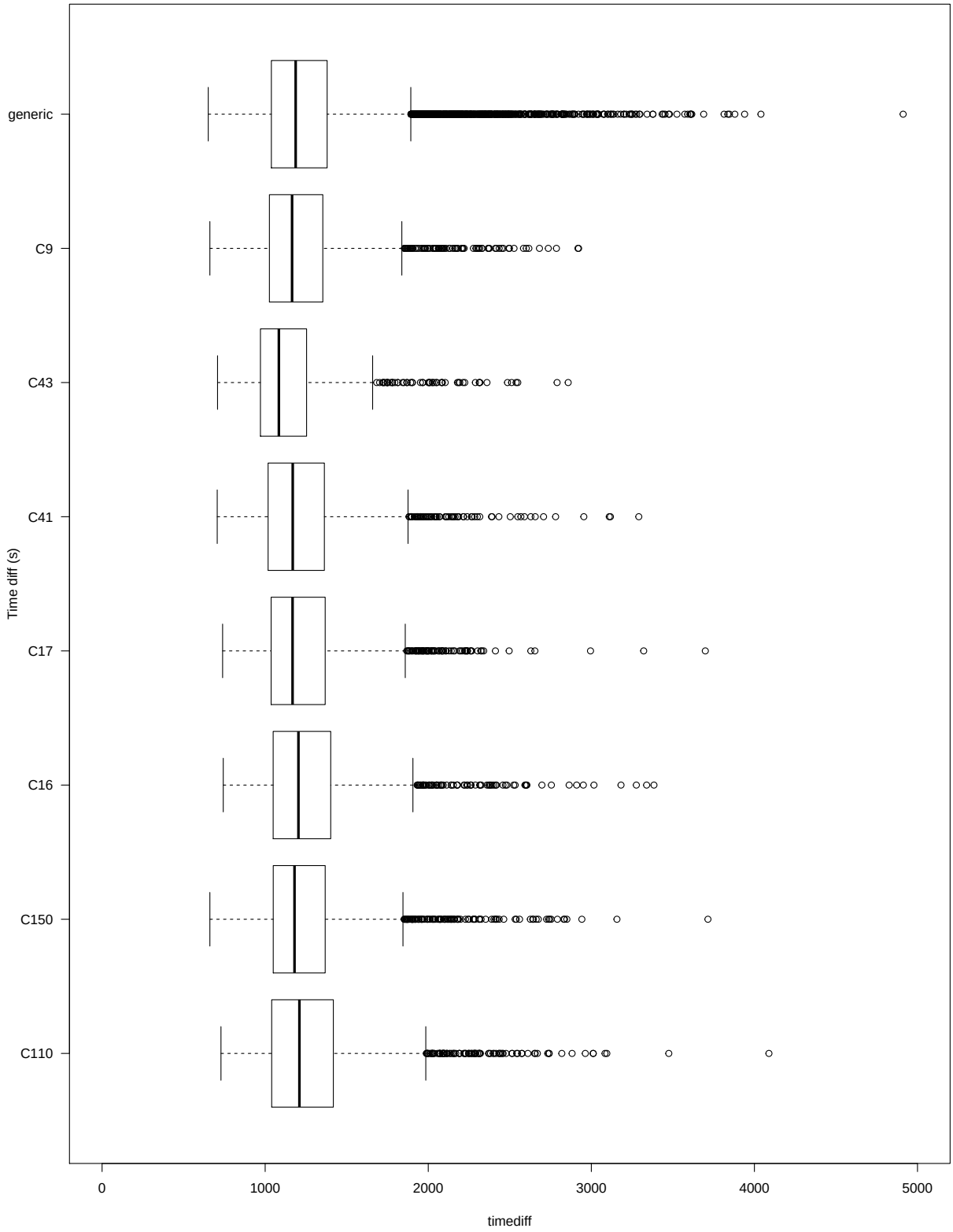

```
## NULL
```

Similarly, trips to the airport also take longer.

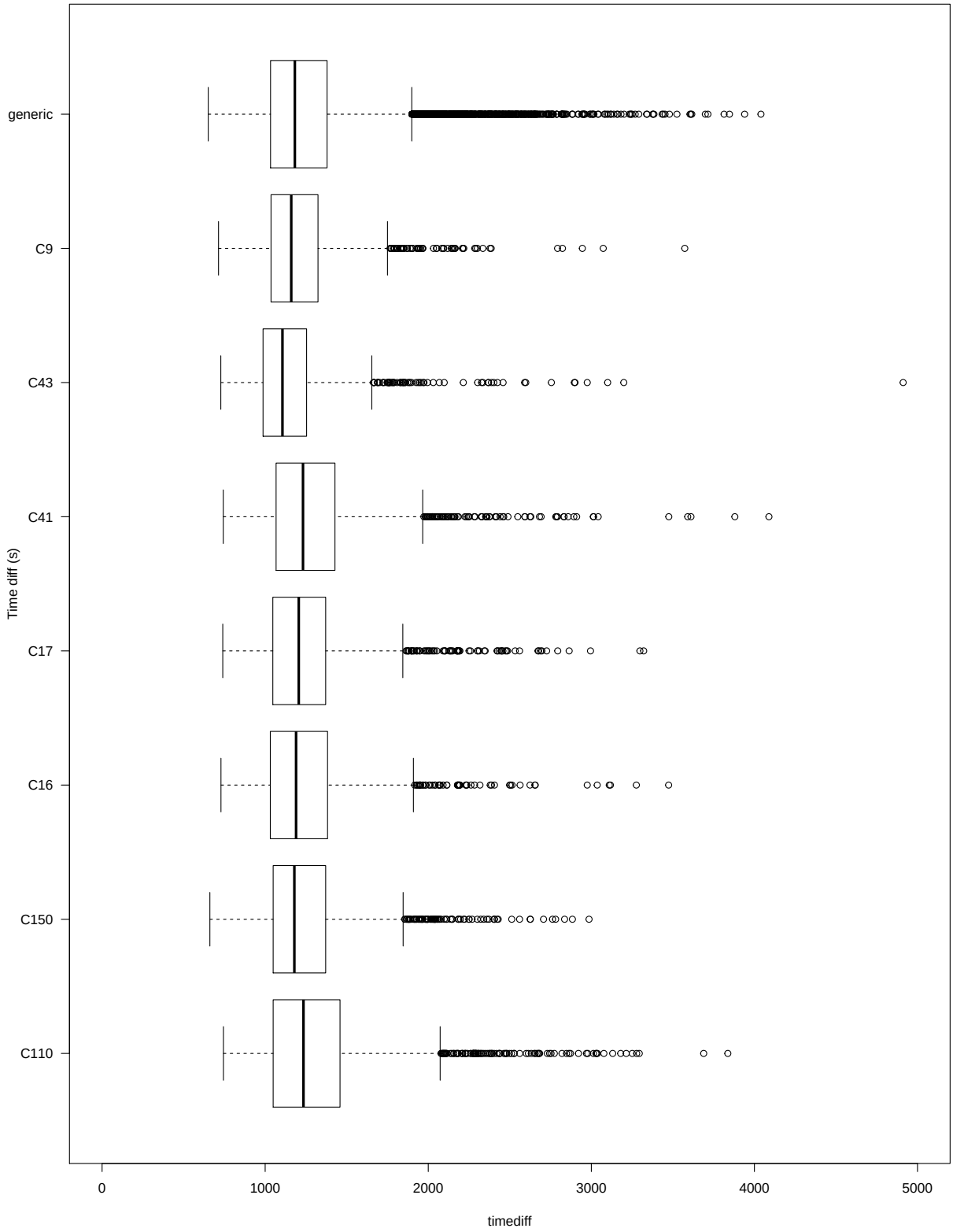



```
## NULL
```

We can do things more algorithmically:


generic
C9
C43
C41
C17
C16
C150
C110
Time diff (s)
0
1000
2000
3000
4000
5000
timediff

```
## NULL
```


generic
C9
C43
C41
C17
C16
C150
C110
Time diff (s)
0
1000
2000
3000
4000
5000
timediff

```
## NULL
```

```
##
## Call:
## lm(formula = tdiff2 ~ relevel(trip_Factor, ref = "generic") +
##     relevel(trip_Factor2, ref = "generic"), data = summary_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -614.0  -213.9   -66.5   123.0 15140.0
##
## Coefficients:
##                                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                                   5.023      3.283   1.530 0.125998
## relevel(trip_Factor, ref = "generic")C110    25.933      7.558   3.431 0.000602
## relevel(trip_Factor, ref = "generic")C150   -17.605      8.718  -2.019 0.043458
## relevel(trip_Factor, ref = "generic")C16     -8.028      9.566  -0.839 0.401348
## relevel(trip_Factor, ref = "generic")C17    -12.774      8.821  -1.448 0.147595
## relevel(trip_Factor, ref = "generic")C41    -21.074      8.193  -2.572 0.010110
## relevel(trip_Factor, ref = "generic")C43   -129.799     10.259 -12.653  < 2e-16
## relevel(trip_Factor, ref = "generic")C9     -42.999      9.284  -4.631 3.65e-06
## relevel(trip_Factor2, ref = "generic")C110   64.916      7.642   8.494  < 2e-16
## relevel(trip_Factor2, ref = "generic")C150  -15.730      8.706  -1.807 0.070810
## relevel(trip_Factor2, ref = "generic")C16   -10.252      9.947  -1.031 0.302698
## relevel(trip_Factor2, ref = "generic")C17     1.889      8.948   0.211 0.832819
## relevel(trip_Factor2, ref = "generic")C41    48.880      8.736   5.595 2.23e-08
## relevel(trip_Factor2, ref = "generic")C43  -113.518      8.143 -13.941  < 2e-16
## relevel(trip_Factor2, ref = "generic")C9    -51.838      9.231  -5.616 1.97e-08
##
## (Intercept)
## relevel(trip_Factor, ref = "generic")C110  ***
## relevel(trip_Factor, ref = "generic")C150  *
## relevel(trip_Factor, ref = "generic")C16
## relevel(trip_Factor, ref = "generic")C17
## relevel(trip_Factor, ref = "generic")C41   *
## relevel(trip_Factor, ref = "generic")C43   ***
## relevel(trip_Factor, ref = "generic")C9    ***
## relevel(trip_Factor2, ref = "generic")C110 ***
## relevel(trip_Factor2, ref = "generic")C150 .
## relevel(trip_Factor2, ref = "generic")C16
## relevel(trip_Factor2, ref = "generic")C17
## relevel(trip_Factor2, ref = "generic")C41  ***
## relevel(trip_Factor2, ref = "generic")C43  ***
## relevel(trip_Factor2, ref = "generic")C9   ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 340.7 on 27985 degrees of freedom
## Multiple R-squared:  0.01803,    Adjusted R-squared:  0.01754
## F-statistic: 36.71 on 14 and 27985 DF,  p-value: < 2.2e-16
```

Conclusion

We can see that classifying trips could be a useful covariate.