Clustering Analysis of Chicago Neighbourhoods Taxi Use

Jarome Leslie

July 06, 2020

Contents

Summary

With the goal of clustering Chicago's 77 community areas or neighbourhoods based on their taxi usage and demographics, this analysis employs the K-Means and Gaussian Mixture Model algorithms. Prior to implementing these algorithms, the data was first scaled to prevent features with the largest numeric values from overshadowing other columns with smaller values. Next, the dimension reduction technique of Principal Component Analysis was used to flatten the dataset into two dimensions.

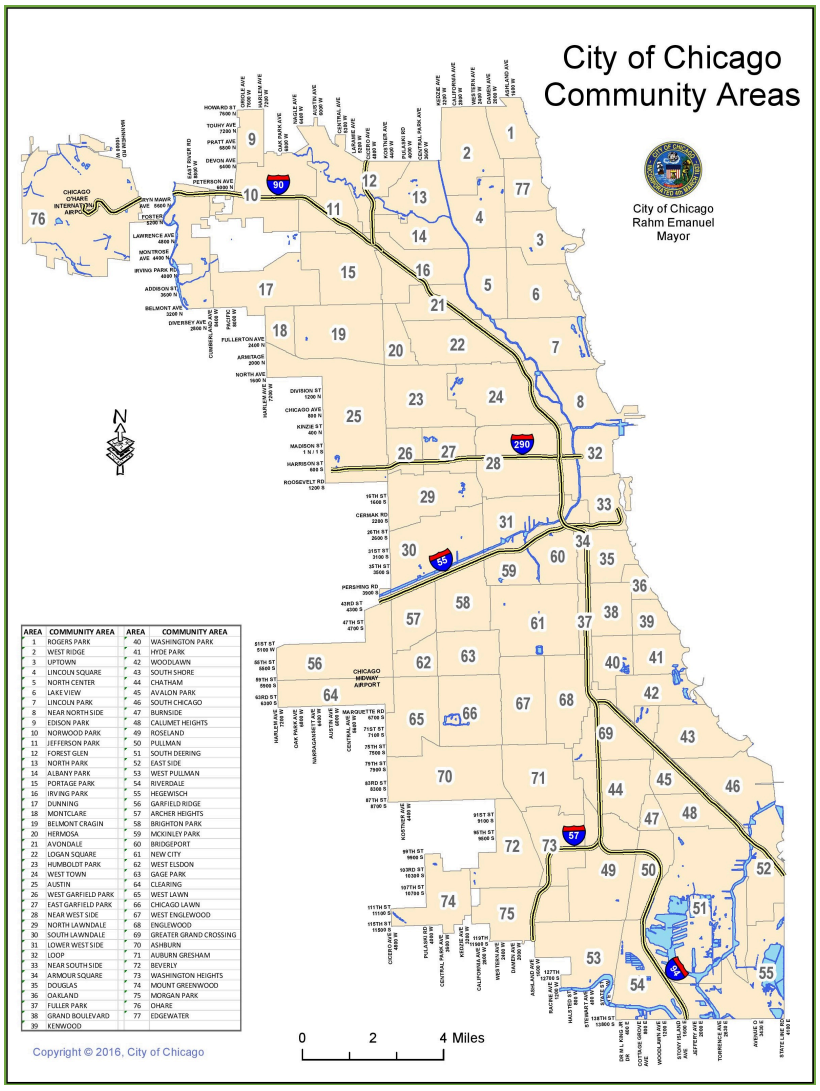
After testing multiple configurations, a K-Means algorithm with six groups was selected based on its superior metrics of silhouette score and inertia. Comparatively, a GMM clustering algorithm with four components was selected based on it having the best performing AIC and BIC scores. Upon examination of the clustering results, both approaches weighted the following features as most important: population employed, population in labor force, total commuters, population over 16, and total population. Going forward, iterating on removing some demographic data will help to increase the importance taxi ridership behavior in the creation of clusters.

This analysis was performed using the core python packages of `Pandas`, `NumPy`, and `Sklearn` with `altair` for graphics. The outcomes of both techniques were overlayed on a map of the city using `Kepler.gl`.

Analysis

Problem

As seen in Figure 1, the city of Chicago is divided into 77 community areas or neighbourhoods. Given access to publicly available data on community area taxi usage and demographics, the goal of this analysis is to cluster these neighbourhoods into logical groups.



AREA	COMMUNITY AREA	AREA	COMMUNITY AREA
1	ROGERS PARK	40	WASHINGTON PARK
2	WEST RIDGE	41	HYDE PARK
3	UPTOWN	42	WOODLAWN
4	LINCOLN SQUARE	43	SOUTH SHORE
5	NORTH CENTER	44	CHATHAM
6	LAKE VIEW	45	AVALON PARK
7	LINCOLN PARK	46	SOUTH CHICAGO
8	NEAR NORTH SIDE	47	BURNSIDE
9	EDISON PARK	48	CALUMET HEIGHTS
10	NORWOOD PARK	49	ROSELAND
11	JEFFERSON PARK	50	PULLMAN
12	FOREST GLEN	51	SOUTH DEERING
13	NORTH PARK	52	EAST SIDE
14	ALBANY PARK	53	WEST PULLMAN
15	PORTAGE PARK	54	RIVERDALE
16	IRVING PARK	55	HEGEWISCH
17	DUNNING	56	GARFIELD RIDGE
18	MONTCLARE	57	ARCHER HEIGHTS
19	BELMONT CRAGIN	58	BRIGHTON PARK
20	HERMOSA	59	MCKINLEY PARK
21	AVONDALE	60	BRIDGEPORT
22	LOGAN SQUARE	61	NEW CITY
23	HUMBOLDT PARK	62	WEST ELSDON
24	WEST TOWN	63	GAGE PARK
25	AUSTIN	64	CLEARING
26	WEST GARFIELD PARK	65	WEST LAWN
27	EAST GARFIELD PARK	66	CHICAGO LAWN
28	NEAR WEST SIDE	67	WEST ENGLEWOOD
29	NORTH LAWNDALE	68	ENGLEWOOD
30	SOUTH LAWNDALE	69	GREATER GRAND CROSSING
31	LOWER WEST SIDE	70	ASHBURN
32	LOOP	71	AUBURN GRESHAM
33	NEAR SOUTH SIDE	72	BEVERLY
34	ARMOUR SQUARE	73	WASHINGTON HEIGHTS
35	DOUGLAS	74	MOUNT GREENWOOD
36	OAKLAND	75	MORGAN PARK
37	FULLER PARK	76	OHARE
38	GRAND BOULEVARD	77	EDGEWATER
39	KENWOOD		

Figure 1: Map of Chicago Community Areas

Data Sources

The data used in this analysis were taken from two public data sets:

1. **Chicago Taxi Trips BigQuery Dataset** found here and queried for the inbound and outbound taxi trips for each city community area (CCA) in 2017; and
2. **Chicago City Demographics by CCA** based on the 2013-2017 American Community Survey downloaded from here with data dictionary provided here.

Data Exploration

Before any exploration can be done, the data was first read in using the following code. Figure 2 summarizes the features within the two datasets and highlights the field corresponding to the community area code which is used to merge them together. The second code chunk was executed to produce a merged dataset. Inbound trips represent taxi rides leaving from each community area while outbound trips represent taxi rides terminating in each community area.

```
# Read in datasets
inbound = pd.read_csv('data/CCA_inbound_trips.csv', index_col='Unnamed: 0')
outbound = pd.read_csv('data/CCA_outbound_trips.csv', index_col='Unnamed: 0')
demographics = pd.read_csv('data/CCAProfileSlim.csv')
```

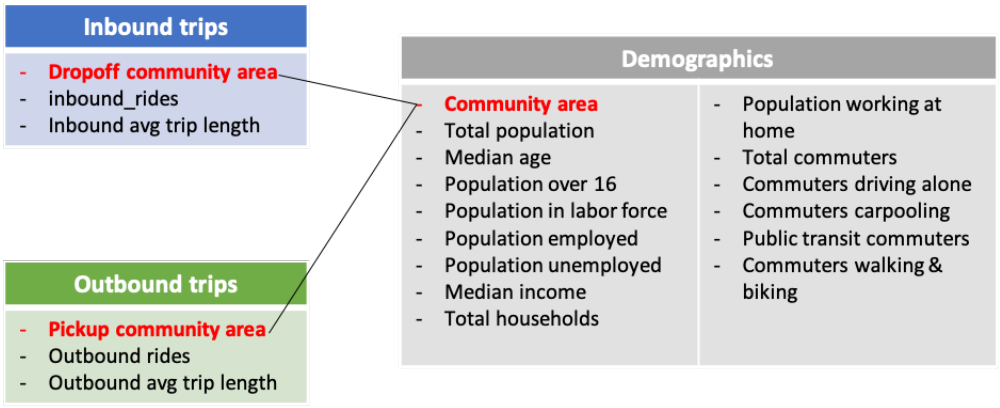


Figure 2: Schematic of dataset column names

```
# Merge inbound and outbound trip tables, dropping rides with no disclosed location
trips = inbound.merge(outbound, left_on='dropoff_community_area',
              right_on='pickup_community_area').dropna()

# Convert community area floats to integers
trips['Community_code'] = trips.apply(lambda x: int(x['dropoff_community_area']),
                                      axis=1)


# Merge trips and demographics
combined = trips.merge(demographics)
names_df = combined[['Community_code', 'GEOG']]

# Drop extraneous columns
vars_df = combined.drop(columns=['dropoff_community_area',
                                 'pickup_community_area',
                                 'Community_code',
                                 'GEOG'])
```

To get a better sense of our data, Figure 3 presents four scatter plots of the ridership columns where each data point is a Chicago community area. From these charts we observe that the majority of community areas had less than 1 million total inbound rides in 2017 with Near North Side, the Loop, Near West Side, Lake View and Lincoln being the exceptions. Four community areas saw outbound rides over 1 million including Near North Side, the Loop, Near West Side, and O'Hare.

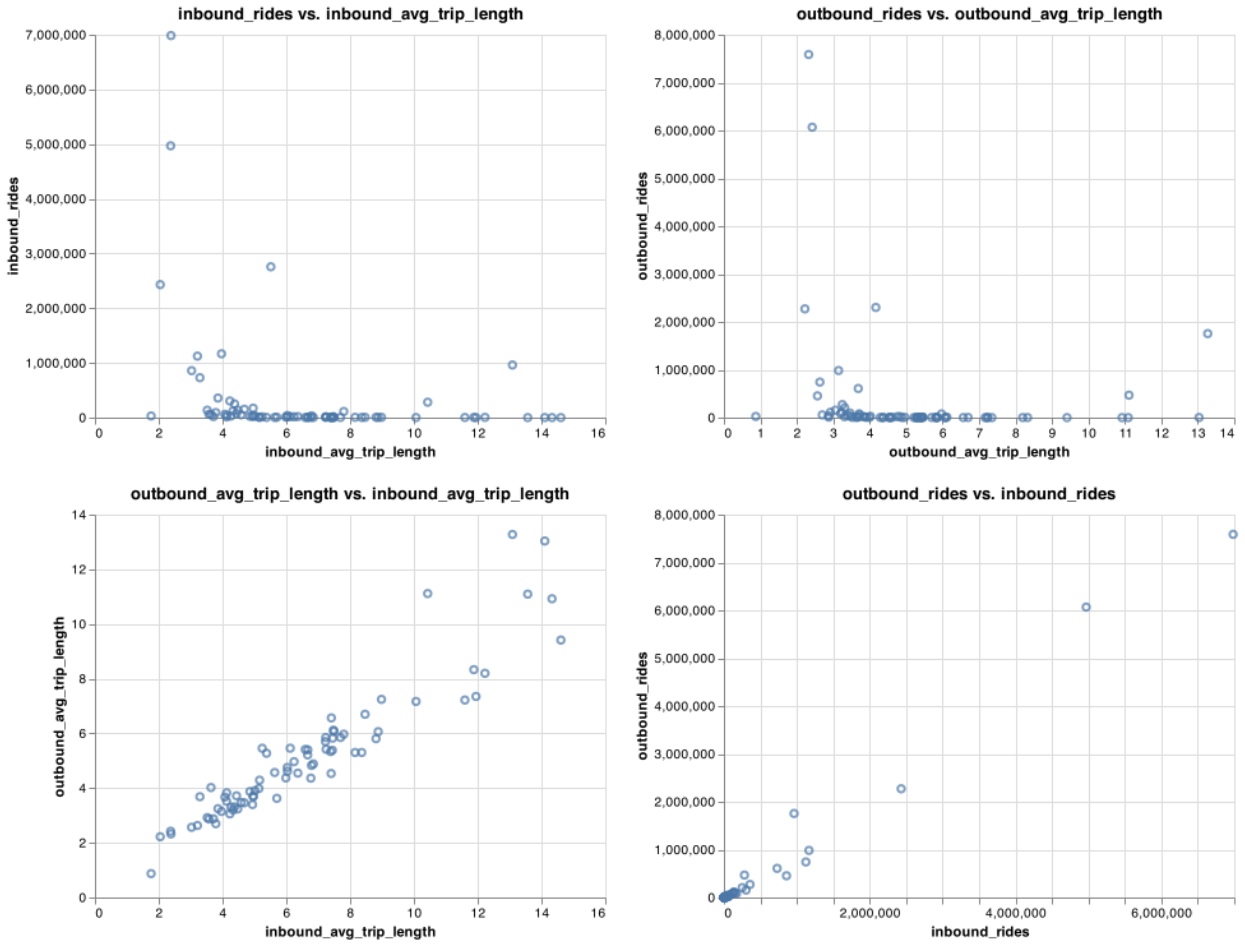


Figure 3: Scatter plots of ridership columns

While it is possible to perform and visualize clustering on any pair of variables, this becomes problematic as the number of features increases. The demographic data for each community area presents such a challenge and makes a case for the implementation of dimension reduction.

Scaling Data

Prior to implementing dimension reduction, the data must be scaled to prevent features with the largest numeric values from overshadowing other columns with smaller values. For example, the number of taxi rides is in the millions while the average trip length is measured in minutes. A standard scaler was used to convert the data in each column to Z-statistics.

```
# Scale data
scaler = StandardScaler()
scaled_vars_df = pd.DataFrame(scaler.fit_transform(vars_df),
                              columns = vars_df.columns)
```

Dimension Reduction

Principal Component Analysis was used to map the 18 features to 2 dimensions as shown in the code below. This reduction allows for the differences between each community area to be visualized on a single chart, as seen in Figure 4.

Table 1: Sample of Scaled Combined Dataset

X	inbound_rides	inbound_avg_trip_length	outbound_rides	outbound_avg_trip_length	TOT_POP	MED_AGE	POP_16OV	IN_LBFRC	EMP	UNEMP	WORK_AT_HOME
0	-0.1506617	-0.6905240	-0.1641290	-0.7845146	0.8647667	-0.3690088	0.9148035	0.9259441	0.9175782	0.4785181	0.8675256
1	-0.1515502	-1.0078340	-0.1613442	-0.9169364	1.7926199	-0.1145041	1.5803610	1.3183141	1.2720511	1.0893662	0.3100641
2	0.0687012	-0.8961850	-0.0195135	-0.7796655	0.9924545	0.1942477	1.2012901	1.2268527	1.2339673	0.4315298	1.4289832
3	-0.1343645	-0.6215325	-0.1779205	-0.6898292	0.2793151	-0.1487280	0.3548786	0.5482106	0.6231248	-0.5748034	0.9204744
4	-0.1150766	-0.5298912	-0.1904961	-0.5853802	0.0193776	-0.3411228	-0.0469096	0.2303290	0.3159530	-0.8434200	1.1402621
5	0.8707952	-0.8592051	0.6097369	-0.8231350	2.8565390	-0.9274223	3.1694750	3.8593740	3.9767922	0.3610473	4.1403638
6	0.8284565	-1.1098377	0.3960478	-1.0386696	1.4195574	-1.1131717	1.5745388	1.8425507	1.9486294	-0.3516089	2.2641763
7	6.6395714	-1.3854169	6.4487517	-1.1679246	2.3487266	-0.1856325	2.8035562	2.9023622	2.9813240	0.4080356	3.8956003
8	-0.2801615	0.2821415	-0.2592867	-0.2400100	-1.0414272	0.9201458	-1.0043600	-0.8465928	-0.7595414	-1.3164358	-0.4971561
9	-0.2476929	-0.1744774	-0.2430659	-0.1470862	0.0764007	1.6301482	0.1043653	0.0487805	0.1199604	-0.7517928	-0.0735653

WORK_AT_HOME	TOT_COMM	DROVE_AL	CARPOOL	TRANSIT	WALK_BIKE	COMM_OTHER	NO_VEH	ONE_VEH	TWO_VEH	THREEOM_VEH	MEDINC	TOT_HH
0.8675256	0.9182211	0.3862203	0.1251160	1.4229300	0.4925002	0.4735444	1.4668566	0.9143725	0.2143305	-0.5675161	-0.5056282	1.0074918
0.3100641	1.3251578	2.0807947	1.8747419	0.2753675	0.5561682	0.7445539	0.0503855	1.2788891	1.7704809	1.8419842	-0.0501477	1.1136373
1.4289832	1.2225908	0.4537396	0.1828262	1.9369386	0.6270799	0.9508447	1.9728471	1.4371090	0.4673831	-0.4606148	-0.0575389	1.5116347
0.9204744	0.5913858	0.5038001	-0.0351900	0.8166906	0.0979108	-0.0340929	0.0068134	0.6931010	0.6575410	-0.2496253	0.8770550	0.4516198
1.1402621	0.2648603	0.2175111	-0.2532062	0.4138909	0.0868547	-0.1756650	-0.4331893	0.2582426	0.7828389	-0.4029443	2.2590406	0.0861570
4.1403638	3.9454954	2.3264653	1.4075643	5.5947717	1.4696329	1.8953332	3.7570366	3.6657833	2.5664909	0.5788602	1.4864595	3.7579642
2.2641763	1.9145304	1.1469260	0.3779415	2.5112519	1.0369190	1.9924112	1.5232737	1.9470382	1.4378273	-0.1553833	2.0612977	1.7990720
3.8956003	2.8867473	1.1200252	0.2405364	1.9220398	6.2561730	7.0141774	4.8899113	4.0214333	0.9582559	-0.3002627	1.8080175	3.9887778
-0.4971561	-0.7690207	-0.6112484	-0.9814536	-0.7065344	-0.4609954	-0.4709441	-0.8349090	-0.8770789	-0.5359826	-0.2299329	2.0405197	-0.8655051
-0.0735653	0.1392855	1.0357597	-0.0150372	-0.4972417	-0.3809338	-0.3981355	-0.4407452	-0.2097180	1.5159542	1.3370159	0.9758892	0.1169642

```
# Initialize and fit a PCA model
pca1 = PCA(n_components=2)
Z = pca1.fit_transform(scaled_vars_df)
Z_df = pd.DataFrame(Z, columns=['V1','V2'])
```

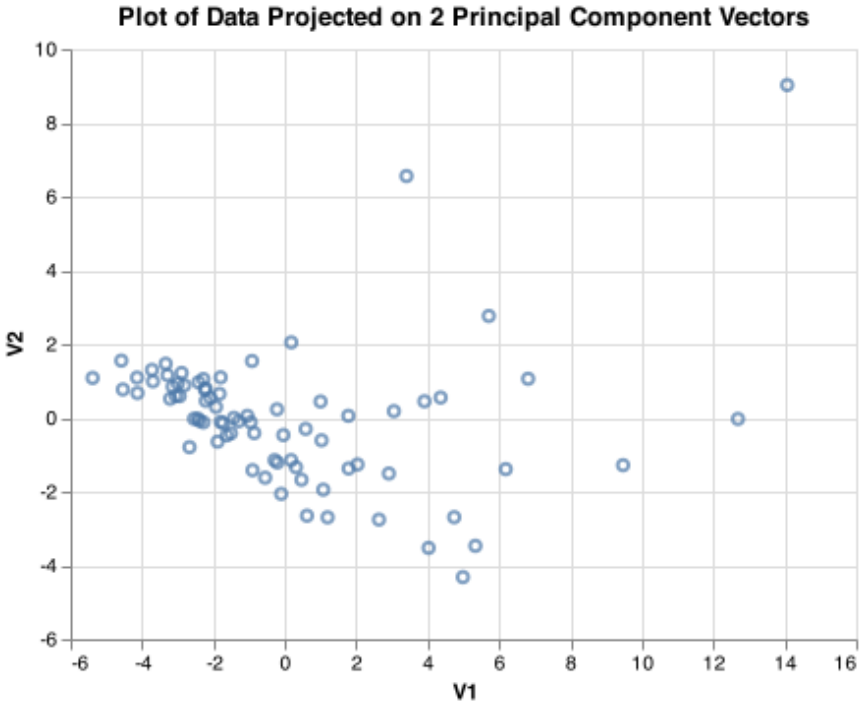


Figure 4: Scatter plot data projected on to Principal Component Vectors

Clustering

Given this transformed data, we may now embark on testing different clustering algorithms. K-means and Gaussian Mixture Models are two clustering algorithms used in this analysis. Their results are compared in the subsequent concluding section.

K-Means

The K-Means clustering algorithm creates spherical clusters of the input data and optimizes the location of the cluster centers and their radii. To obtain the best K-means result, several numbers of clusters were tested against two metrics: silhouette score and inertia. Silhouette score measures how similar an object to its own cluster compared to the next closest cluster. As such, the higher the silhouette score, the better. Inertia is the sum squared error for each cluster and corresponds to how densely packed together the points in a cluster are. Accordingly, the lower the intertia, the closer each data point in a cluster is to its center and the better the quality of results. In the code chunk and Figure 5 below, k represents the number of clusters tested in the algorithm.

```
# Initialize and fit a PCA model
results = {'k': [], 'inertia': [], 'silhouette': []}

#results = {'k': [], 'inertia': []}
for i in list(range(2,11,1)):
    km = KMeans(n_clusters = i).fit(Z_df)
    df_new = Z_df.copy()
    df_new['prediction'] = km.predict(Z_df)
    results['inertia'].append(km.inertia_)
    results['k'].append(i)
    results['silhouette'].append(silhouette_score(Z_df,df_new['prediction']))

k_test = pd.DataFrame(results)
```

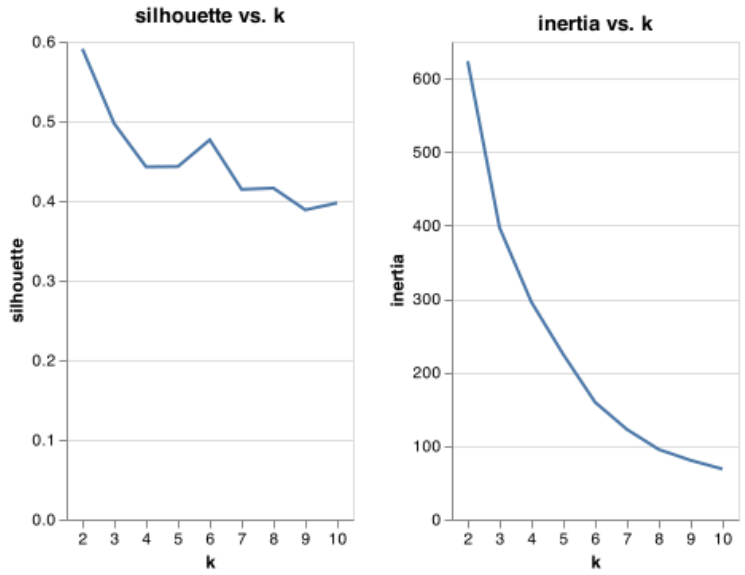


Figure 5: K-Means results for differing values of k

From the results above, we observe that the `k=2` and `k=6` maximize silhouette. After `k=6`, it appears that the inertia improvements wane. As such, `k=6` is taken as the best result. The following code chuck assigns each community area to a cluster and Figure 6 provides is visual representation.

```
# Reproduce K-Means with k=6
km1 = KMeans(n_clusters=6)
km1.fit(Z_df)
# Predict clusters for each community area
```

```
Z_df['cluster'] = km1.predict(Z_df)
```

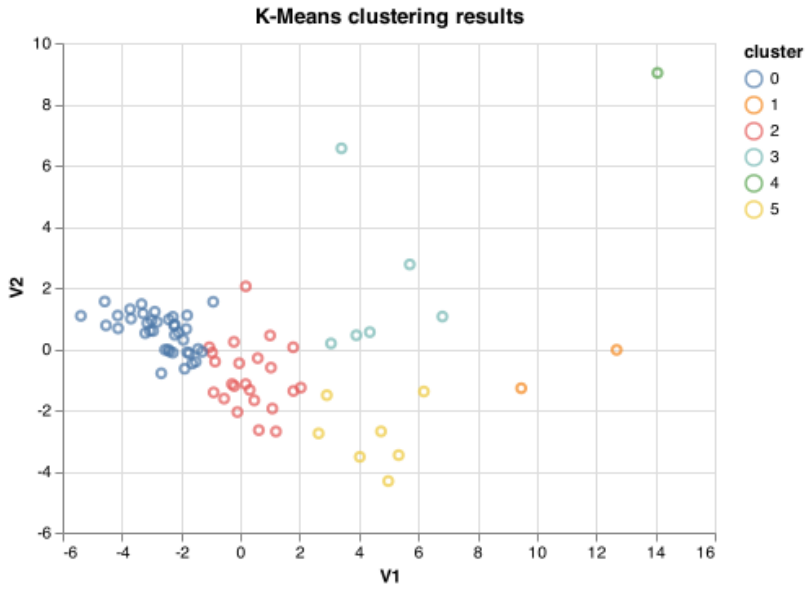


Figure 6: K-Means visualization for k = 6

Gaussian Mixture Models

Gaussian Mixture Models (GMMs) are another clustering algorithm which assumes the data points are generated from a discrete number of Gaussian distributions. In making this assumption, GMMs may be considered a generalization of K-Means to incorporate coveriances of the data in addition to the centers of clusters. The code chunk below initializes and fits a GMM model to the data for multiple values of k. The effectiveness of each model may be viewed looking at its Akaike Information criterion (AIC) and Bayesian Information criterion (BIC). Both of these metrics balance the goal to maximize the likelihood of the model while penalizing complexity under the following formulas:

$$AIC = 2k - 2ln(\hat{L})$$

$$BIC = kln(n) - 2ln(\hat{L})$$

As such, the lower the AIC and BIC scores, the better. The results of the algorithm are presented in Figure 7.

```
# Initialize and fit GMM over a range of k values
Z_df2 = pd.DataFrame(Z, columns=['V1','V2'])
results = {'k': [], 'AIC': [], 'BIC': []}

for i in list(range(2,11,1)):
    gm_new = GaussianMixture(n_init = 10, n_components = i).fit(Z_df2)
    results['AIC'].append(gm_new.aic(Z_df2))
    results['BIC'].append(gm_new.bic(Z_df2))
    results['k'].append(i)

gmm_test = pd.DataFrame(results)
```

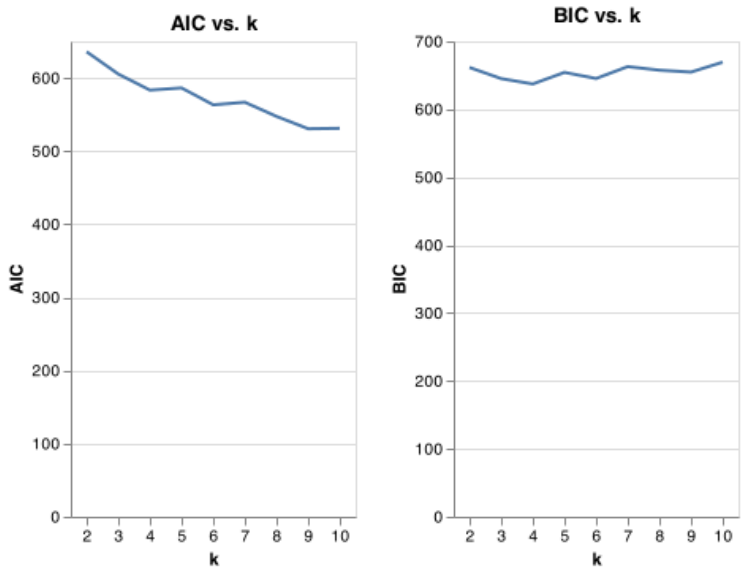


Figure 7: GMM results for differing values of k

While the AIC score decreased with increasing numbers of clusters, we observe that the BIC score is minimized at k=4. This picture is not abnormal as BIC penalizes more complex models more strictly. As such, k=4 is taken as the best result for this algorithm. Figure 8 presents a visualization of GMM clustering with k=4.

```
# Initialize and fit GMM with k = 4
Z_df3 = pd.DataFrame(Z, columns=['V1','V2'])
gm = GaussianMixture(n_init = 70, n_components = 4).fit(Z_df3)
# Predict clusters for each community area
Z_df3['cluster'] = gm.predict(Z_df3)
```

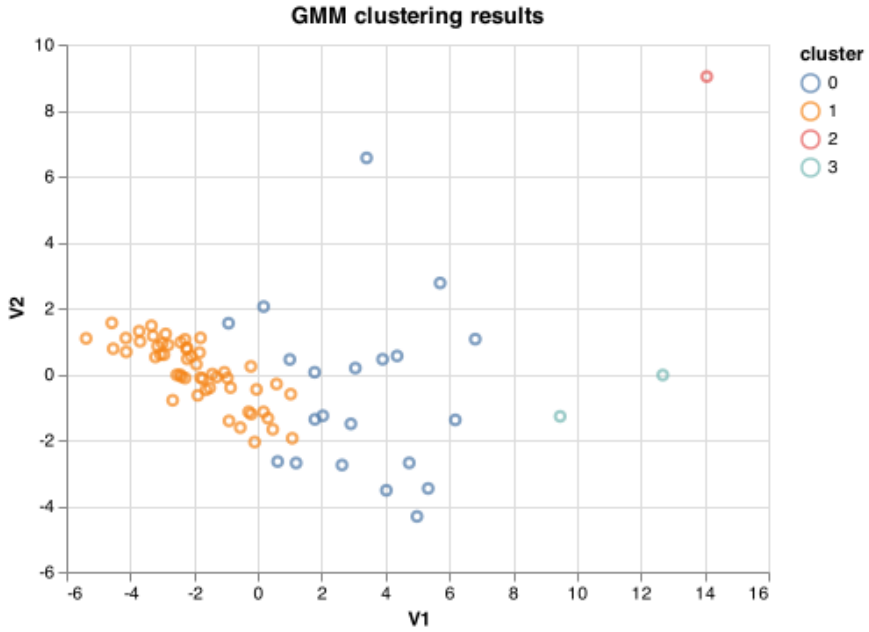


Figure 8: Gaussian Mixture Model Visualization with 4 Clusters

Discussion

Figures 9 and 10 overlay the clustering results from the K-Means and GMM clustering algorithms. Under K-Means, Near North Side was assigned a cluster of its own, while Lake View and West Town were assigned in a cluster together. A third cluster containing The Loop and other North West shoreline neighbourhoods was assigned while the remaining majority were split between the largest two clusters. Under GMM, Near North Side was again assigned to its own cluster, while Lake View and West Town were also assigned to a cluster together. The remaning neighbourhoods were split between two larger clusters.

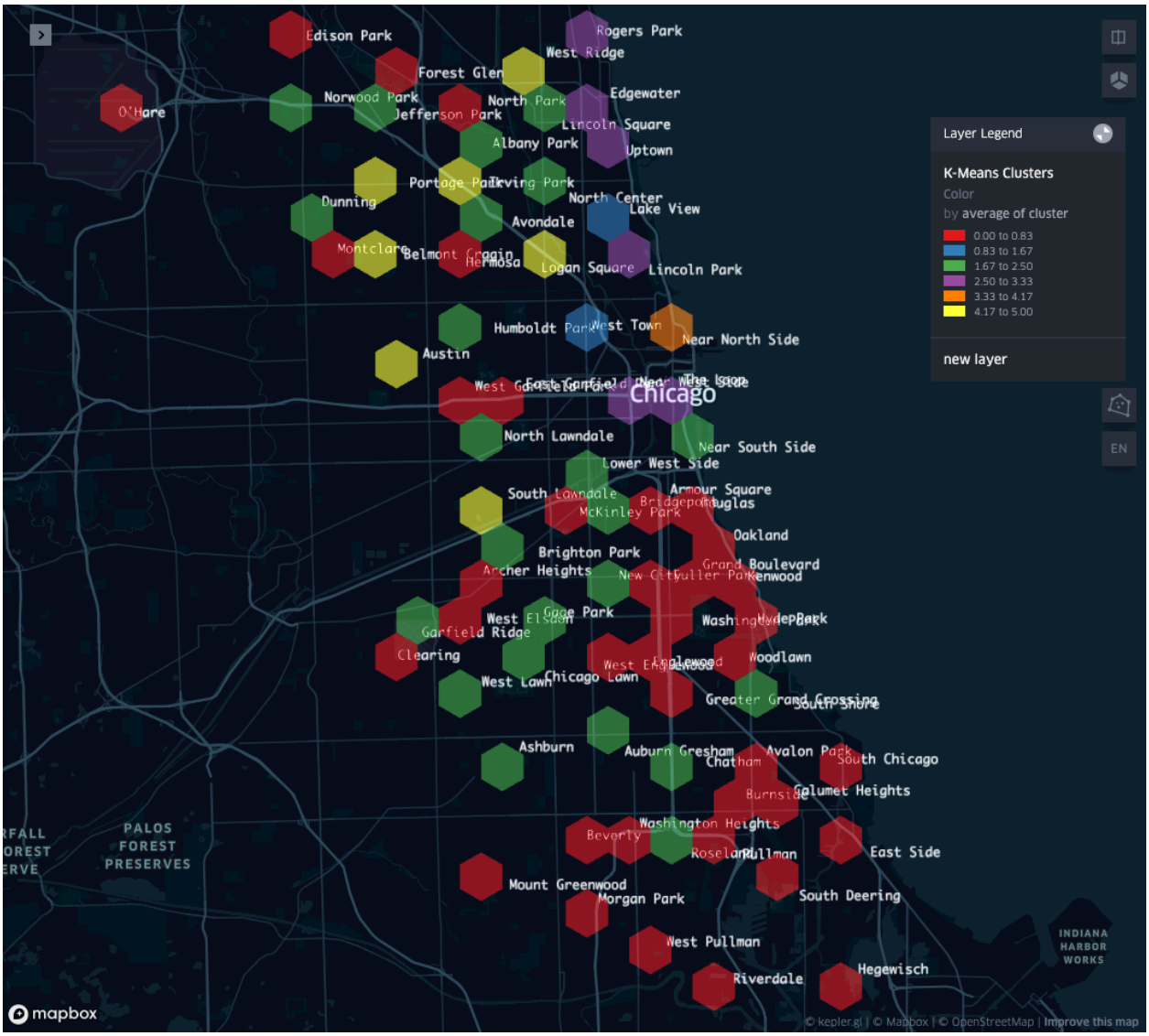


Figure 9: Overlay of K-Means clusters on City Map using Kepler.gl

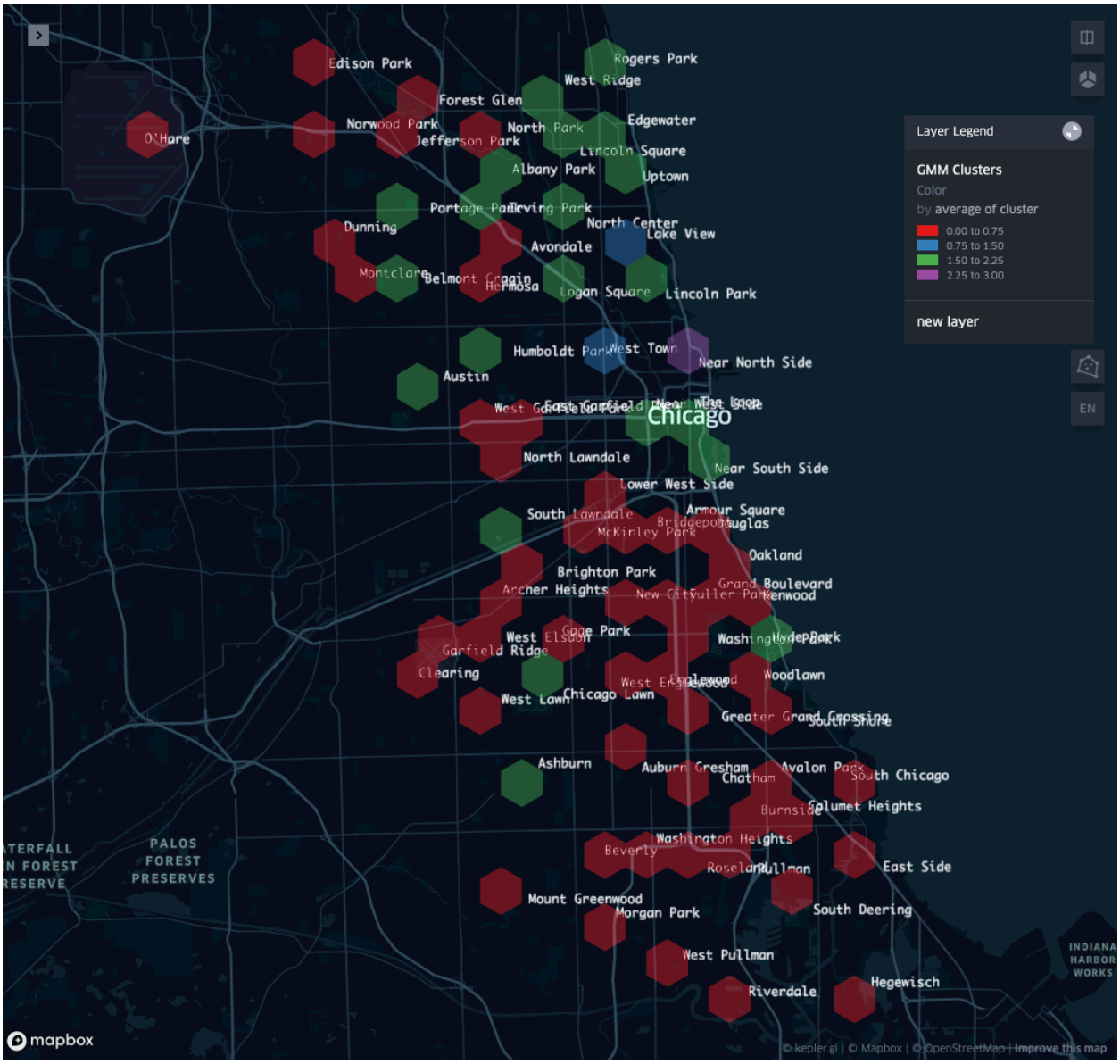


Figure 10: Overlay of K-Means clusters on City Map using Kepler.gl

Feature Importance

To answer the question of what features are driving the clustering results, the assigned clusters from K-Means and the GMM were used as the target for a Random Forest Classifier. In doing so, the original features are used as predictors and checked for their importance to the final cluster assignment. The code chunk below fits a classifier for each of the two methods and creates summary tables of feature importance.

```
# K-Means
# Set up predictors and targets for model
Z_df = pd.DataFrame(Z, columns=['V1','V2'])
km_scaled_vars_df = scaled_vars_df.copy()
km_Y = km1.predict(Z_df)

# Initialize and fit classifier
```

```
km_rf = RandomForestClassifier()
km_rf.fit(km_scaled_vars_df, km_Y)
```

```
# GMM
# Set up predictors and targets for model
Z_df3 = pd.DataFrame(Z, columns=['V1','V2'])
gm_scaled_vars_df = scaled_vars_df.copy()
gm_Y = gm.predict(Z_df3)

# Initialize and fit classifier
gm_rf = RandomForestClassifier()
gm_rf.fit(gm_scaled_vars_df, gm_Y)
```

Table 2: Top 5 features for clustering using K-Means

X	Feature	Importance
7	IN_LBFRC	0.1333159
8	EMP	0.1120256
6	POP_16OV	0.1021819
4	TOT_POP	0.0884568
11	TOT_COMM	0.0837074

Table 3: Top 5 features for clustering using a GMM

X	Feature	Importance
8	EMP	0.1458132
7	IN_LBFRC	0.1228078
10	WORK_AT_HOME	0.1050678
18	ONE_VEH	0.0860761
11	TOT_COMM	0.0814540

As seen in Table 2, the top 5 features for clustering using K-Means were population employed, population in labor force, total commuters, population over 16, and total population. Of these five features, population employed, population in labor force and total commuters were also among the most relevant for the GMM clusters. The other top features for the GMM clusters were population working at home and the number of households with one vehicle, as shown in Table 3. From these results, we note the demographic variables are overshadowing the taxi ridership features. Going forward, iterating on removing some demographic data will help to increase the importance taxi ridership behavior in the creation of clusters.

Appendix

Dependencies

This analysis was performed using the packages shown below.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.mixture import GaussianMixture
import altair as alt
from altair_saver import save
from sklearn.metrics import silhouette_samples,  silhouette_score
from sklearn.decomposition import PCA
```

Plotting functions

```
def alt_scatter(x_var,y_var,data):
    chart = alt.Chart(data).mark_point().encode(
    x=alt.X(x_var, axis = alt.Axis(labelAngle = 0)),
    y=y_var
).properties(title = y_var +' vs. ' +x_var)
    return chart


def alt_scatter_cluster(x_var,y_var, label, data):
    chart = alt.Chart(data).mark_point().encode(
    x=alt.X(x_var, axis = alt.Axis(labelAngle = 0)),
    y=y_var,
    color=label
).properties(title = y_var +' vs. ' +x_var + " by cluster")
    return chart

def alt_linechart(x_var,y_var,data):
    chart = alt.Chart(data).mark_line().encode(
    x=alt.X(x_var+':O', axis = alt.Axis(labelAngle = 0)),
    y=y_var
).properties(title = y_var +' vs. ' +x_var)
    return chart
```

SQL queries for Inbound and Outbound trips

```
## OUTBOUND QUERY
query2 = """SELECT

pickup_community_area,
COUNT(1) AS outbound_rides,
AVG(trip_miles) AS outbound_avg_trip_length

FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
```

```
WHERE
    EXTRACT(YEAR FROM trip_start_timestamp) = 2017

GROUP BY
    pickup_community_area

ORDER BY
    pickup_community_area

        """
outbound = chicago_taxi.query_to_pandas_safe(query2, max_gb_scanned=10)
```

```
## INBOUND QUERY
query3 = """SELECT

dropoff_community_area,
COUNT(1) AS inbound_rides,
AVG(trip_miles) AS inbound_avg_trip_length

FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`

WHERE
    EXTRACT(YEAR FROM trip_start_timestamp) = 2017

GROUP BY
    dropoff_community_area

ORDER BY
    dropoff_community_area

        """
inbound = chicago_taxi.query_to_pandas_safe(query3, max_gb_scanned=10)
```