Uber Trip Data Clustering Analysis

Ondřej Marvan (USOS ID: 477001)

2025-11-13

1 1. Main

The task is as following: Uber trip data: 1.1 Load the data [uber-data.csv] □ Discover and comment clusters of Uber data based on locations (longitude & latitude) □ Analyze the cluster centers by time □ Analyze the cluster centers by date □ Remember to choose the right algorithm, compute the optimal number of clusters and quality measures □ Develop adequate plots □ Apply the dataset for forecasting

1.1 1.1 Load Libraries

```
# install.packages("tidyverse")
# install.packages("lubridate")
# install.packages("cluster")
# install.packages("factoextra")
# install.packages("forecast")
install.packages("leaflet") # not installed yet on my machine



library(tidyverse) # For data loading, manipulation, and plotting
library(lubridate) # For easier date-time parsing
library(cluster) # For k-Means and silhouette analysis
library(factoextra) # For cluster visualization
library(forecast) # For time-series forecasting
library(leaflet) # For map of New York display
library(dbscan) #

# Set system language as English
Sys.setlocale("LC_ALL", "en_US.UTF-8")
```

```
## [1] "LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF-8;LC_
MONETARY=en_US.UTF-8;LC_MESSAGES=en_US.UTF-8;LC_PAPER=en_US.UTF-8;LC_NAME=C;LC_ADDRESS=
C;LC_TELEPHONE=C;LC_MEASUREMENT=en_US.UTF-8;LC_IDENTIFICATION=C"
```

```
Sys.setenv(LANGUAGE = 'en')

# Seed seed
set.seed(123)
```

1.2 1.2 Load data

```
# Load the data
uber_data_raw <- read_csv("uber-data.csv")

# Display the first few rows
print("Raw data:")
```

```
## [1] "Raw data:"
```

```
head(uber_data_raw)
```

```
## # A tibble: 6 × 4
##   `Date/Time`        Lat   Lon Base
##   <chr>            <dbl> <dbl> <chr>
## 1 9/1/2014 0:01:00  40.2 -74.0 B02512
## 2 9/1/2014 0:01:00  40.8 -74.0 B02512
## 3 9/1/2014 0:03:00  40.8 -74.0 B02512
## 4 9/1/2014 0:06:00  40.7 -74.0 B02512
## 5 9/1/2014 0:11:00  40.8 -73.9 B02512
## 6 9/1/2014 0:12:00  40.7 -74.0 B02512
```

```
# Display the structure
print("Structure of raw data:")
```

```
## [1] "Structure of raw data:"
```

```
glimpse(uber_data_raw)
```

```
## Rows: 1,028,136
## Columns: 4
## $ `Date/Time` <chr> "9/1/2014 0:01:00", "9/1/2014 0:01:00", "9/1/2014 0:03:00"…
## $ Lat         <dbl> 40.2201, 40.7500, 40.7559, 40.7450, 40.8145, 40.6735, 40.7…
## $ Lon         <dbl> -74.0021, -74.0027, -73.9864, -73.9889, -73.9444, -73.9918…
## $ Base        <chr> "B02512", "B02512", "B02512", "B02512", "B02512", "B02512"…
```

column names “Date/Time” (as a character -> parse), “Lat” (lattitude), “Lon” (longitude), and “Base” (not figured out yet). There are over 1 mil rows.

1.3 1.3 Summary statistics

```
summary(uber_data_raw)
```

```
##   Date/Time              Lat             Lon             Base          
##  Length:1028136     Min.   :39.99   Min.   :-74.77   Length:1028136    
##  Class :character   1st Qu.:40.72   1st Qu.:-74.00   Class :character  
##  Mode  :character   Median :40.74   Median :-73.98   Mode  :character  
##                     Mean   :40.74   Mean   :-73.97                     
##                     3rd Qu.:40.76   3rd Qu.:-73.96                     
##                     Max.   :41.35   Max.   :-72.72
```

Nothing extraordinary here.

Base ‘B02764’ handled the most trips, while ‘B02512’ handled the fewest. No clue what it refers to.

2 2. Data Preparation

2.1 2.1 Data Cleaning

```
uber_data <- uber_data_raw %>%
  # Columns rename
  rename(
    datetime = `Date/Time`,
    latitude = Lat,
    longitude = Lon,
    base = Base
  ) %>%
  # DateTime Parse
  mutate(
    datetime = mdy_hms(datetime), # Parse
    hour = hour(datetime),       # Extract hour
    date = as.Date(datetime)     # Extract date
  )

# Check for missing data
na_count <- sum(is.na(uber_data))
print(paste("Missing values:", na_count))
```

```
## [1] "Missing values: 0"
```

```
# Show the first few rows of the new, cleaned data
print("Cleaned data:")
```

```
## [1] "Cleaned data:"
```

```
head(uber_data)
```

```
## # A tibble: 6 × 6
##   datetime            latitude longitude base    hour date
##   <dttm>                 <dbl>     <dbl> <chr>  <int> <date>
## 1 2014-09-01 00:01:00     40.2     -74.0 B02512     0 2014-09-01
## 2 2014-09-01 00:01:00     40.8     -74.0 B02512     0 2014-09-01
## 3 2014-09-01 00:03:00     40.8     -74.0 B02512     0 2014-09-01
## 4 2014-09-01 00:06:00     40.7     -74.0 B02512     0 2014-09-01
## 5 2014-09-01 00:11:00     40.8     -73.9 B02512     0 2014-09-01
## 6 2014-09-01 00:12:00     40.7     -74.0 B02512     0 2014-09-01
```

2.2 1.4 Inspect “Base” Column

```
print("Base values")
```

```
## [1] "Base values"
```

```
table(uber_data$base)
```

```
##
## B02512 B02598 B02617 B02682 B02764
##  34370 240600 377695 197138 178333
```

2.3 2.2 Data scaling for clustering

```
# Latitude and Longitude columns only for the scale
location_data <- uber_data %>%
  select(latitude, longitude) %>%
  scale()

print("Scaled location data (first 6 rows):")
```

```
## [1] "Scaled location data (first 6 rows):"
```

```
head(location_data)
```

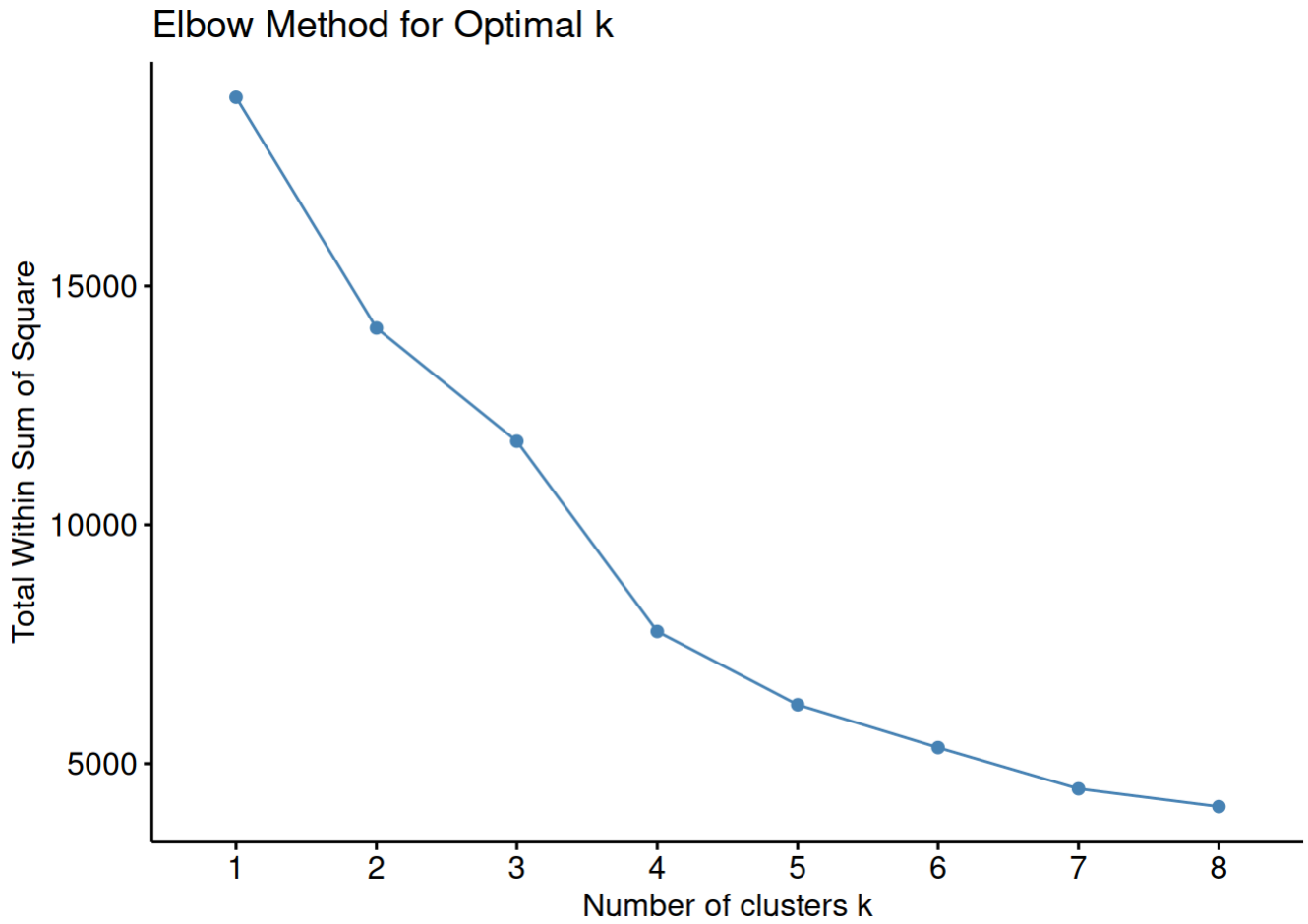
```
##         latitude  longitude
## [1,] -12.7146482 -0.5193102
## [2,]   0.2639973 -0.5295993
## [3,]   0.4085039 -0.2500787
## [4,]   0.1415342 -0.2929500
## [5,]   1.8437721  0.4701584
## [6,]  -1.6096890 -0.3426806
```

latitude and longitude standardization for clustering purposes.

3 3. Clustering Analysis (K-Means)

3.1 3.1 Find Optimal Number of Clusters (k)

```
# 10k sample from the scaled data for the elbow plot (20k were too much)
sample_data_elbow <- location_data[sample(nrow(location_data), 10000), ]

# Compute and plot the elbow method using the factoextra package
fviz_nbclust(sample_data_elbow, kmeans, method = "wss", k.max = 8) +
  ggtitle("Elbow Method for Optimal k")
```



10k sample is approx. 1 % of the whole dataset. The resulting plot shows “elbow” at k=4. Adding more clusters no longer leads to a significant reduction the sum of squares.

3.2 3.2 Perform K-Means Clustering

```
# This runs the algorithm 25 times and picks
# the best result.
k_fit <- kmeans(location_data, centers = 4, nstart = 25)

uber_data$cluster <- as.factor(k_fit$cluster)

print("K-means complete.")
```

```
## [1] "K-means complete."
```

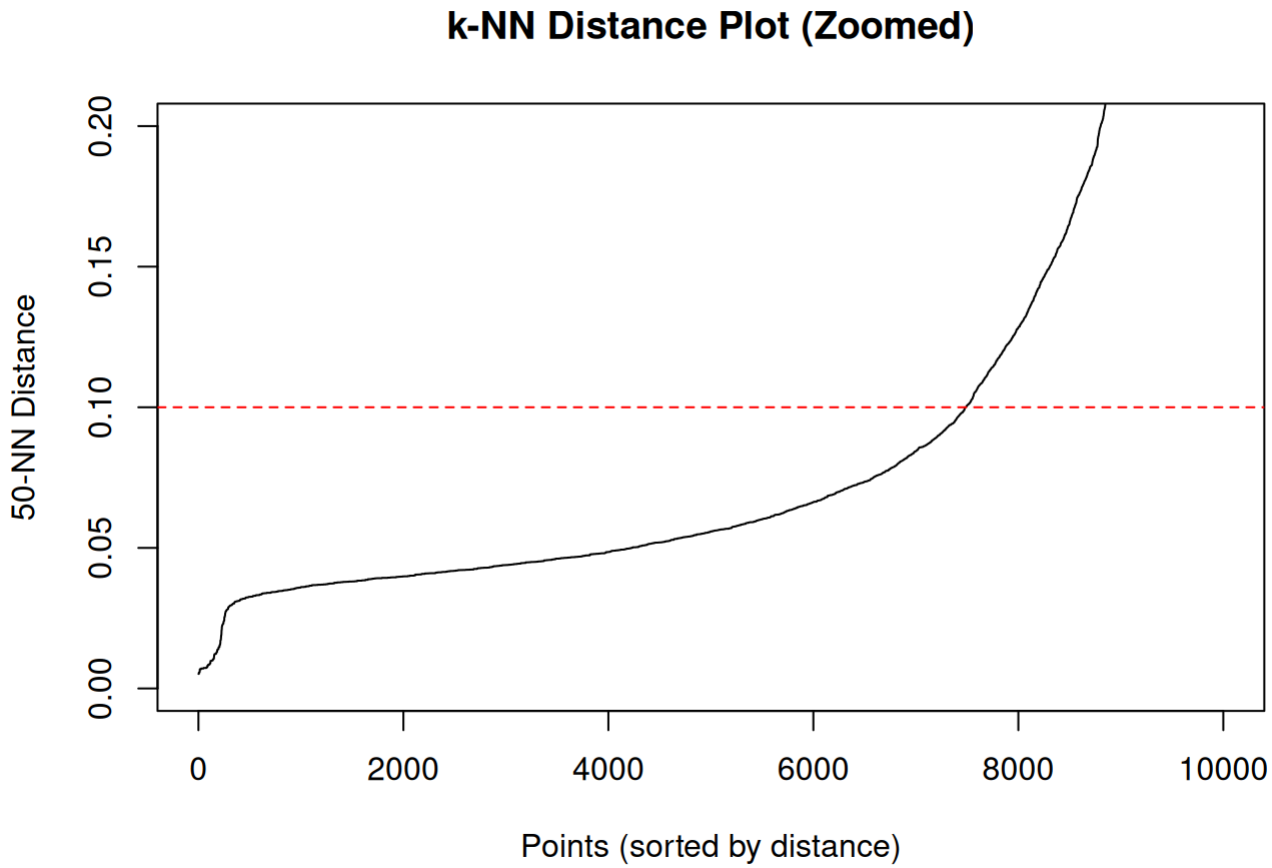
```
head(uber_data)
```

```
## # A tibble: 6 × 7
##   datetime            latitude longitude base    hour date       cluster
##   <dttm>                 <dbl>     <dbl> <chr>  <int> <date>     <fct>
## 1 2014-09-01 00:01:00     40.2     -74.0 B02512     0 2014-09-01 2
## 2 2014-09-01 00:01:00     40.8     -74.0 B02512     0 2014-09-01 1
## 3 2014-09-01 00:03:00     40.8     -74.0 B02512     0 2014-09-01 1
## 4 2014-09-01 00:06:00     40.7     -74.0 B02512     0 2014-09-01 1
## 5 2014-09-01 00:11:00     40.8     -73.9 B02512     0 2014-09-01 4
## 6 2014-09-01 00:12:00     40.7     -74.0 B02512     0 2014-09-01 3
```

4 4. Visualize and Analyze Clusters

4.1 4.1. Plot Clusters on Map

```
# Take a 10k sample to find optimal eps
sample_data_dbscan <- location_data[sample(nrow(location_data), 10000), ]

# Calculate the 50-nearest neighbor distances
knn_data <- kNN(sample_data_dbscan, k = 50)

# Get *only* the distances for the 50th neighbor and sort them
sorted_distances <- sort(knn_data$dist[, 50])

# Create the plot using the standard plot() function
plot(sorted_distances,
     type = 'l',
     ylab = "50-NN Distance",
     xlab = "Points (sorted by distance)",
     main = "k-NN Distance Plot (Zoomed)",
     ylim = c(0, 0.2)) # <-- This now works correctly

# Add our reference line
abline(h = 0.1, col = "red", lty = 2)
```



Each row in the dataset is a record of a single pickup (most probably), with the Date/Time being the time of the pickup and the Lat/Lon being the location.

4.2 4.2. Cluster Sizes

```
print("Cluster sizes:")
```

```
## [1] "Cluster sizes:"
```

```
table(uber_data$cluster)
```

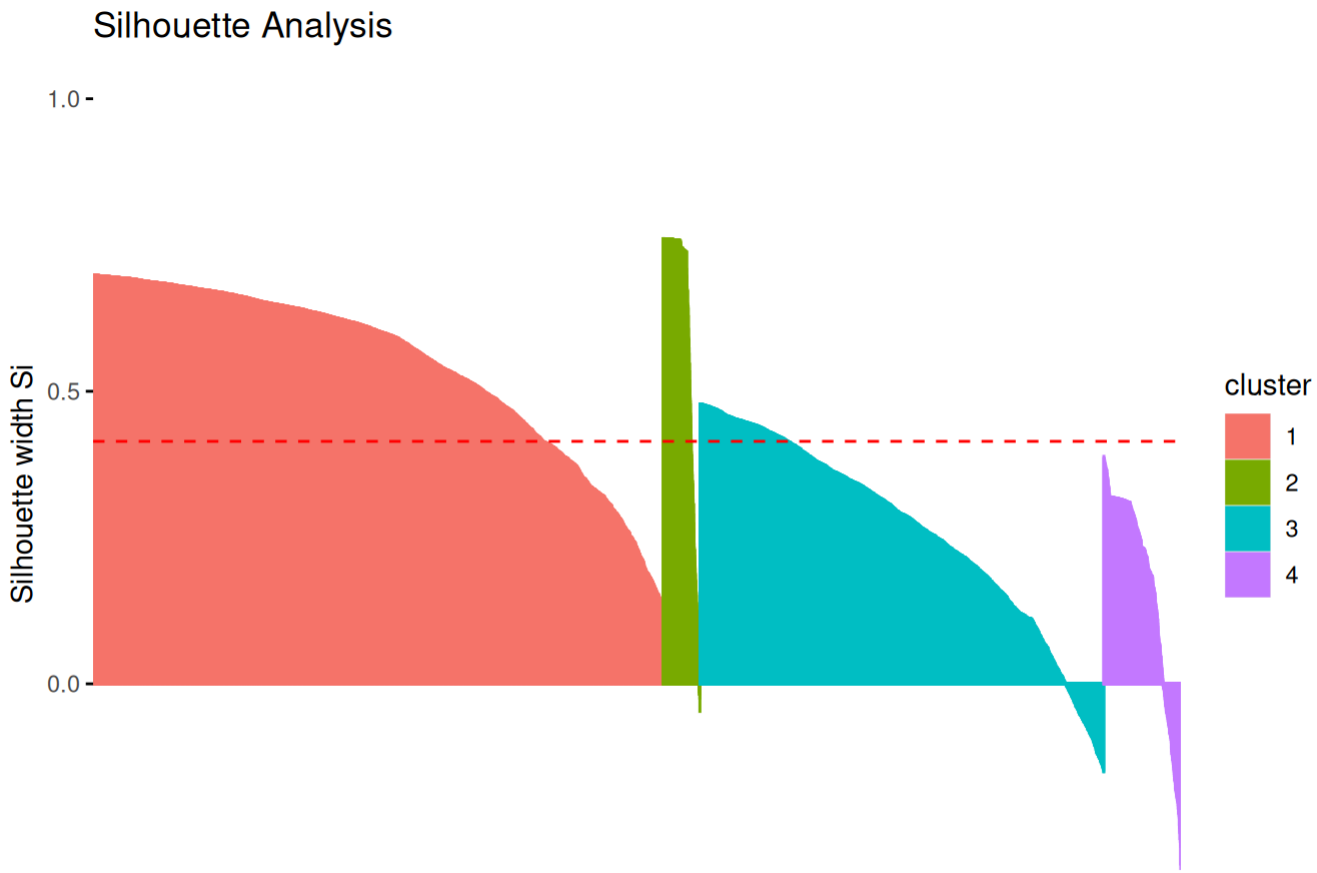
```
##
##      1      2      3      4
## 540075  35979 381501  70581
```

Cluster 1 (540,075 trips): high-density pickup area of Manhattan. Cluster 2 (381,501 trips): Brooklyn. Cluster 3 (35,979 trips): The smallest cluster, JFK Airport and Rhode Island (low density). Cluster 4 (70,581 trips): Represents the Bronx and Upper Manhattan area (lower density).

4.3 4.3. Cluster Quality (Silhouette Analysis)

```
sample_indices <- sample(nrow(location_data), 10000)
sample_data_sil <- location_data[sample_indices, ]

sample_clusters <- k_fit$cluster[sample_indices]

# Calculate and plot the silhouette
sil <- silhouette(sample_clusters, dist(sample_data_sil))
fviz_silhouette(sil) + ggtitle("Silhouette Analysis")
```

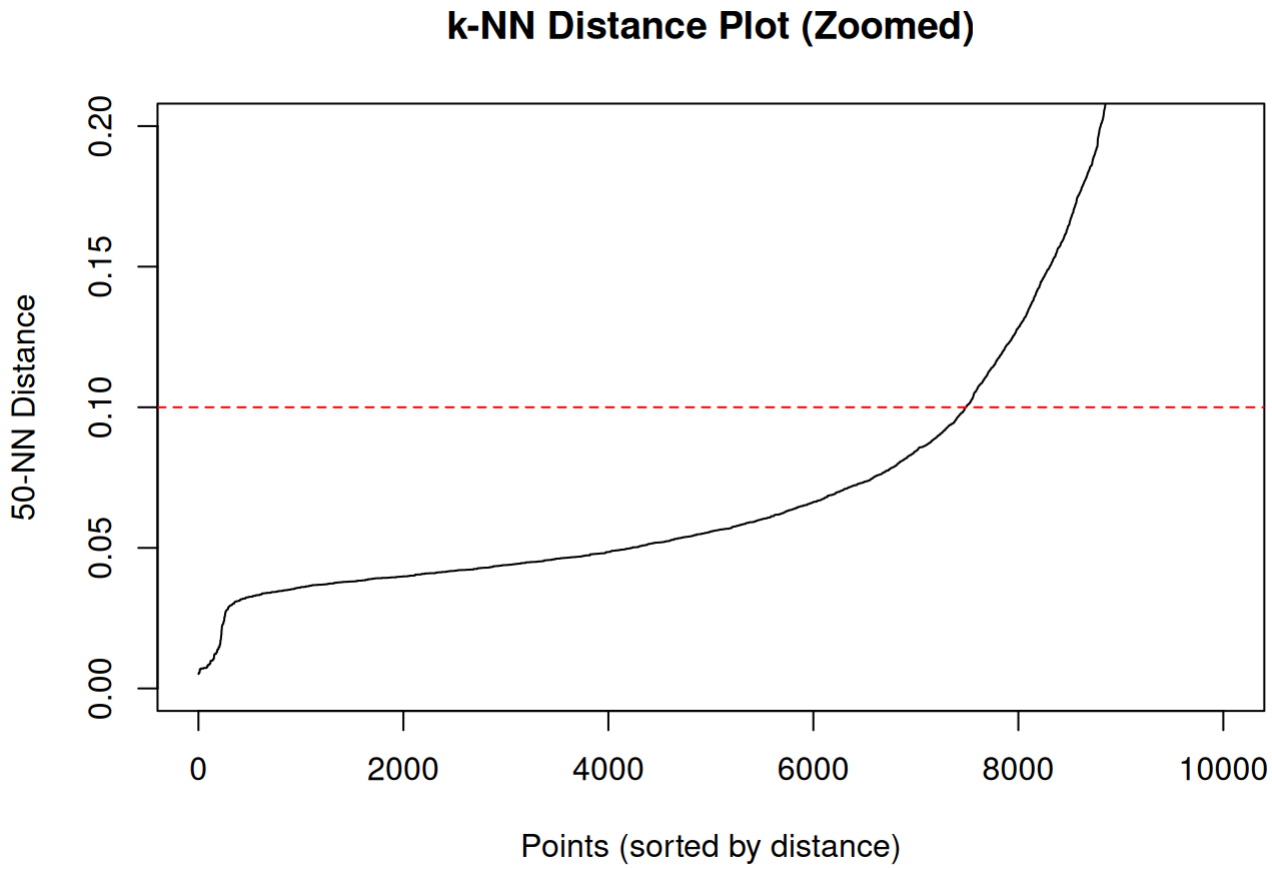
```
##   cluster size ave.sil.width
## 1       1 5241          0.54
## 2       2  338          0.60
## 3       3 3713          0.26
## 4       4  708          0.17
```



Avg. Silhoutte width is 0.4 that is quite fair score. Cluster 1 and 3 is realatively well definied while clusters 2 and 4 achieved poorer score and both with negative tails.

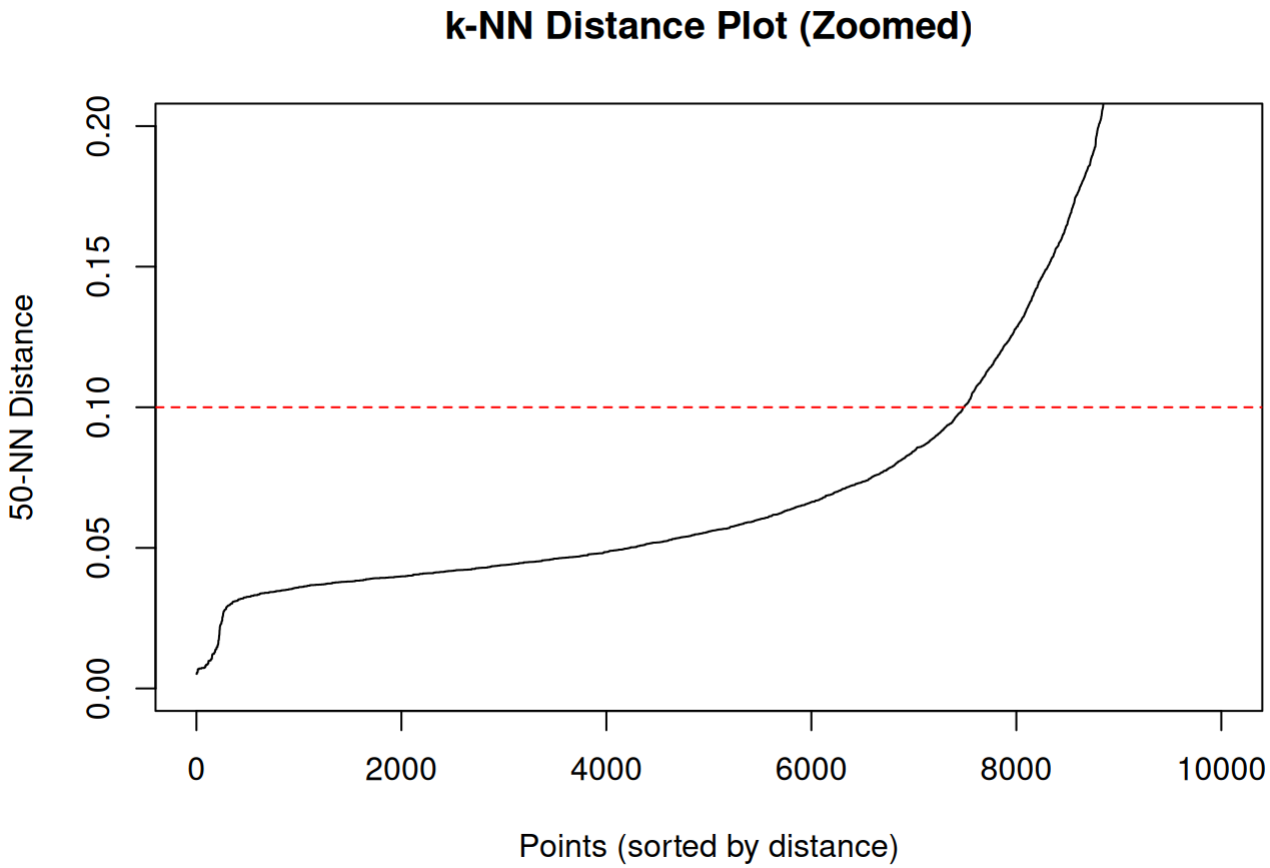
4.4 4.4 DBSCAN Find Optimal eps Value (Zoomed In)

```
# 10k sample to find optimal eps (if not already created)
# sample_data_dbscan <- location_data[sample(nrow(location_data), 10000), ]

# Calculate the 50-nearest neighbor distances
knn_data <- kNN(sample_data_dbscan, k = 50)

# Get *only* the distances for the 50th neighbor and sort them
sorted_distances <- sort(knn_data$dist[, 50])

# Create the plot using the standard plot() function
# This allows us to use ylim to zoom in
plot(sorted_distances,
     type = 'l',
     ylab = "50-NN Distance",
     xlab = "Points (sorted by distance)",
     main = "k-NN Distance Plot (Zoomed)",
     ylim = c(0, 0.2)) # <-- This now works correctly

# Add our reference line
abline(h = 0.1, col = "red", lty = 2)
```



Optimal EPS value isn't really visible, so it requires to zoom in the plot.

```
# Calculate the 50-nearest neighbor distances
# We use the same 10k sample from the previous chunk
knn_data <- kNN(sample_data_dbscan, k = 50)

# Get *only* the distances for the 50th neighbor and sort them
# This is what kNNdistplot does internally
sorted_distances <- sort(knn_data$dist[, 50])

# Create the plot using the standard plot() function
# This allows us to use ylim to zoom in
plot(sorted_distances,
     type = 'l',
     ylab = "50-NN Distance",
     xlab = "Points (sorted by distance)",
     main = "k-NN Distance Plot (Zoomed)",
     ylim = c(0, 0.2)) # <-- This now works correctly

# Add our reference line
abline(h = 0.1, col = "red", lty = 2)
```



```
# Take a larger 50k sample to run the algorithm
dbscan_sample_indices <- sample(nrow(location_data), 50000)
dbscan_sample_data <- location_data[dbscan_sample_indices, ]

# Run dbscan
# Cluster "0" is special: it represents all the "noise" points.
set.seed(123) # for reproducibility
db_fit <- dbscan(dbscan_sample_data, eps = 0.1, minPts = 50)

# Add cluster results back to the original (unscaled) data for plotting
dbscan_results <- uber_data[dbscan_sample_indices, ]
dbscan_results$dbscan_cluster <- as.factor(db_fit$cluster)

print("DBSCAN cluster counts (Cluster 0 = Noise):")
```

```
## [1] "DBSCAN cluster counts (Cluster 0 = Noise):"
```

```
table(dbscan_results$dbscan_cluster)
```

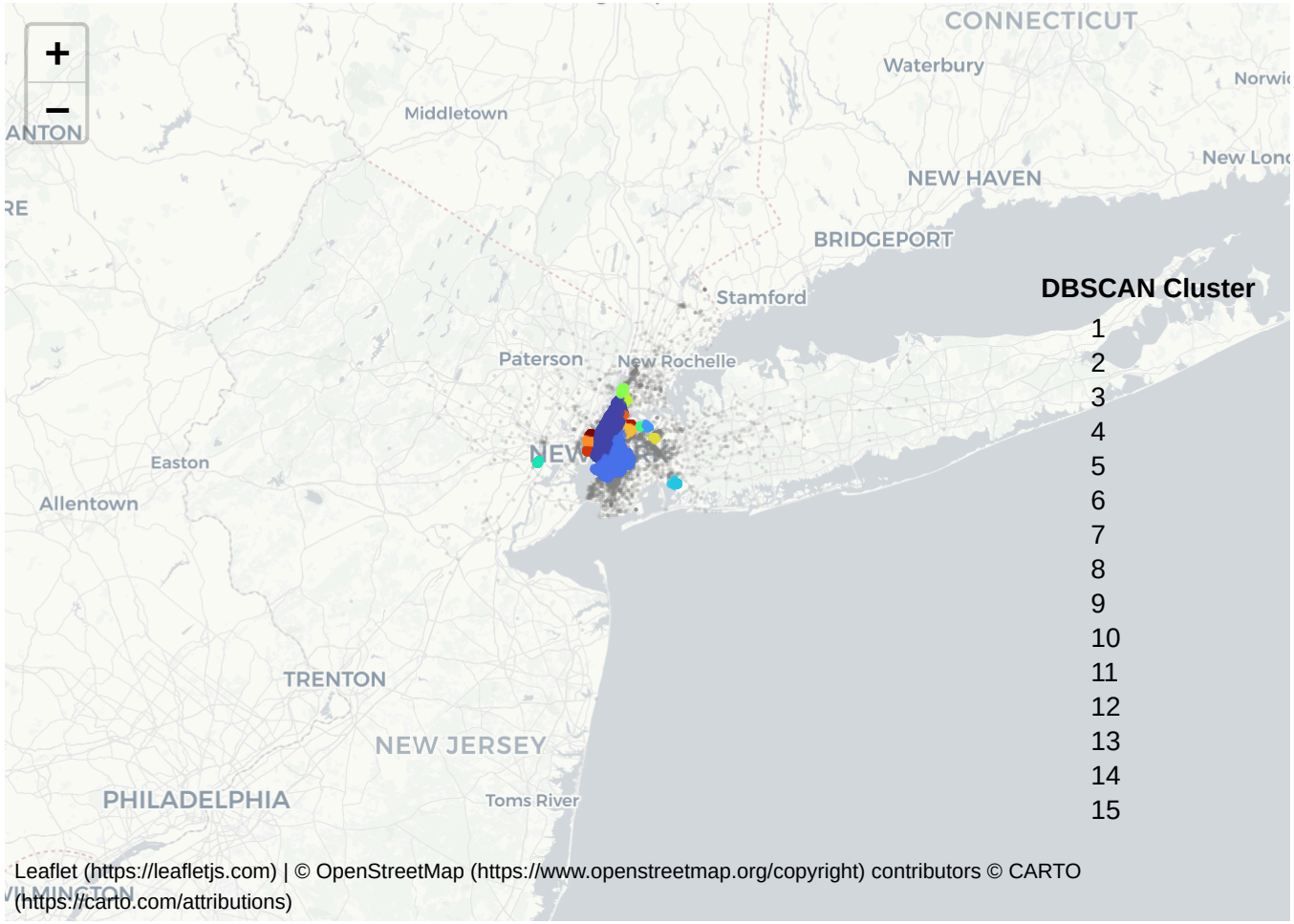
```
##
##     0     1     2     3     4     5     6     7     8     9    10    11    12
##  3579 36602  6363   968  1137   335   118   144    81    72   251   124    61
##    13    14    15
##    62    48    55
```

Those clusters represents seperate dense areas, e.g. JFK Airport and lower Brooklyn. Cluster 0 is noise, Cluster 1 high density area, most probably Manhattan, Clusters 2-14 are smaller dense areas.

4.4 DBSCAN Map

```
# Separate the noise from the clusters
noise_points <- filter(dbscan_results, dbscan_cluster == "0")
cluster_points <- filter(dbscan_results, dbscan_cluster != "0")

# Create a color palette
# 'viridis' again
pal_dbscan <- colorFactor(
  palette = "turbo",
  domain = cluster_points$dbscan_cluster
)

# 3. Leaflet map
leaflet() %>%
  addProviderTiles(providers$CartoDB.Positron) %>%

  # Noise points
  addCircleMarkers(
    data = noise_points,
    lng = ~longitude,
    lat = ~latitude,
    color = "grey",
    radius = 1,
    stroke = FALSE,
    fillOpacity = 0.2, # More transparent
    popup = "Noise"
  ) %>%

  # Real clusters on top
  addCircleMarkers(
    data = cluster_points,
    lng = ~longitude,
    lat = ~latitude,
    color = ~pal_dbscan(dbscan_cluster),
    radius = 2,
    stroke = FALSE,
    fillOpacity = 0.7,
    popup = ~paste("Cluster:", dbscan_cluster)
  ) %>%
```

```
# Legend for the real clusters
addLegend(
  "bottomright",
  pal = pal_dbscan,
  values = cluster_points$dbscan_cluster,
  title = "DBSCAN Cluster",
  opacity = 1
)
```

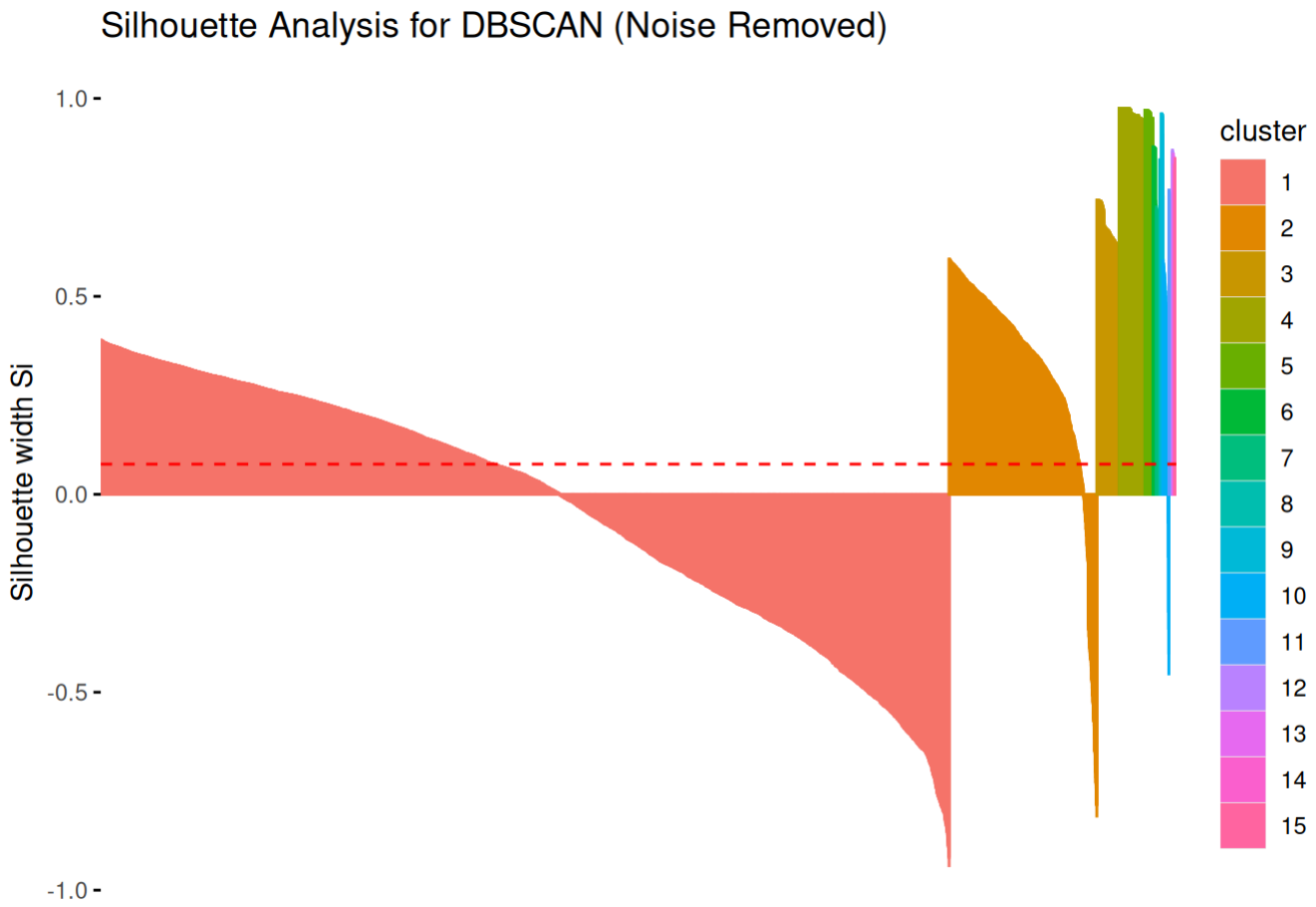


Much better result than for K-means.

4.5 Silhouette Analysis for DBSCAN

```
# ONLY the clustered points (no noise)
data_without_noise <- dbscan_sample_data[db_fit$cluster != 0, ]
labels_without_noise <- db_fit$cluster[db_fit$cluster != 0]

# Take a 10k point sample from the new dataset
set.seed(123) # for reproducibility
sample_indices_sil <- sample(nrow(data_without_noise), 10000)

sil_data <- data_without_noise[sample_indices_sil, ]
sil_labels <- labels_without_noise[sample_indices_sil]

# Calculate and plot the silhouette
# This will take a minute
sil_dbscan <- silhouette(sil_labels, dist(sil_data))
fviz_silhouette(sil_dbscan) + ggtitle("Silhouette Analysis for DBSCAN (Noise Removed)")
```

```
##    cluster size ave.sil.width
## 1        1 7888         -0.03
## 2        2 1373          0.32
## 3        3  209          0.67
## 4        4  237          0.96
## 5        5   77          0.96
## 6        6   30          0.82
## 7        7   31          0.60
## 8        8   16          0.79
## 9        9   16          0.93
## 10      10   54          0.33
## 11      11   32          0.65
## 12      12    9          0.83
## 13      13    6          0.81
## 14      14   12          0.77
## 15      15   10          0.78
```

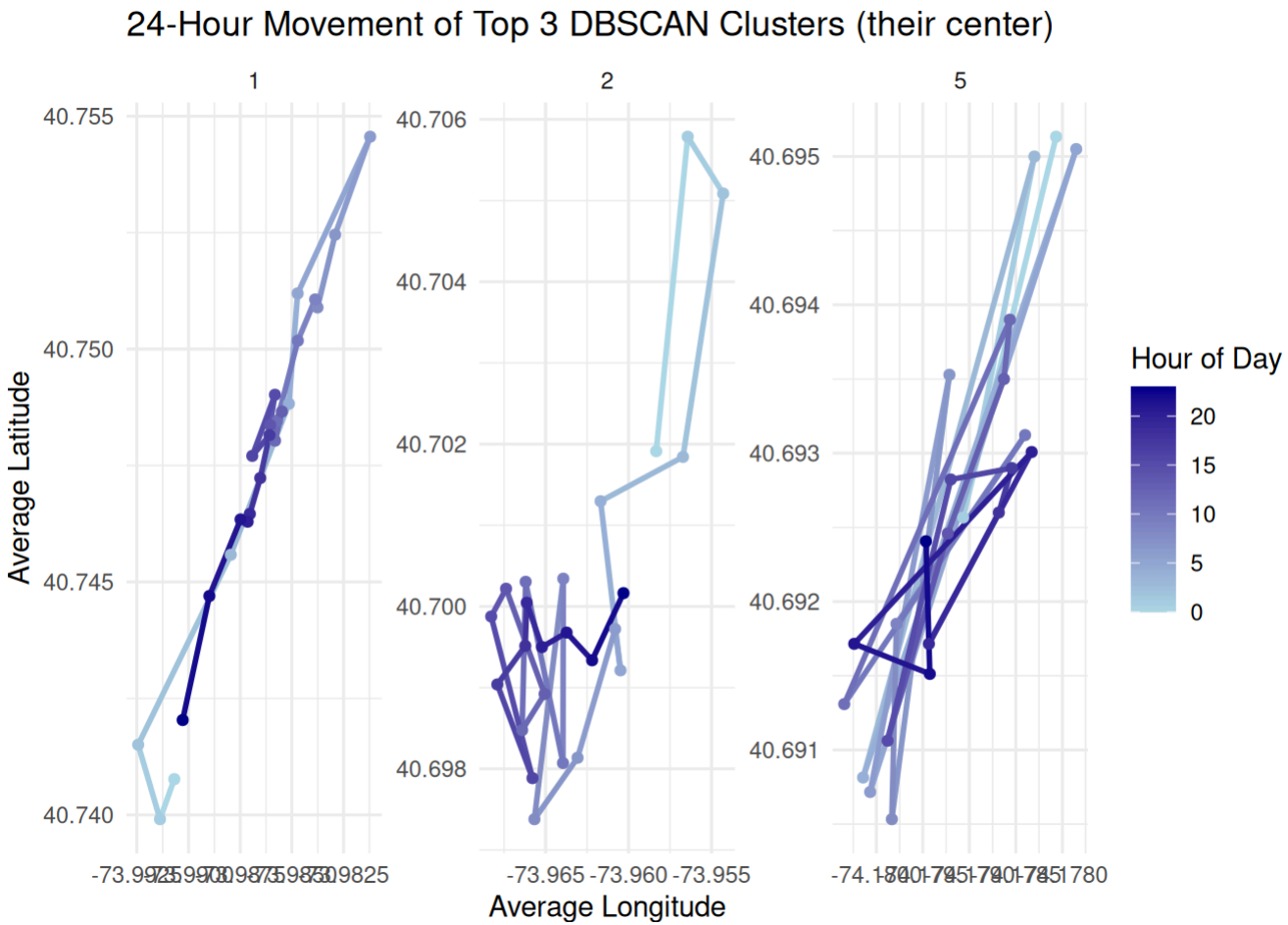


The results seems much better than with K-Means, only the Cluster 1 is ruining the avg. of approx. 0.73 a bit. Clusters 2 and 6 are extremely well-definied.

5 5. Time-Based Analysis

5.1 5.1. Analysis by Hour (DBSCAN, former K-Means)

```
# First, identify the top 3 clusters to analyze
top_3_dbscan_clusters <- c("1", "2", "5") # Those clusters will cover most of the data

# Avg. location per hour for these 3 clusters
dbscan_hourly <- dbscan_results %>%
  filter(dbscan_cluster %in% top_3_dbscan_clusters) %>% # Only top 3
  group_by(dbscan_cluster, hour) %>%
  summarize(
    avg_lat = mean(latitude),
    avg_lon = mean(longitude),
    .groups = 'drop'
  )

# Plot of 24-hour path
ggplot(dbscan_hourly, aes(x = avg_lon, y = avg_lat, color = hour)) +
  geom_path(linewidth = 1) +
  geom_point() +
  facet_wrap(~ dbscan_cluster, scales = "free") + # One map per cluster
  scale_color_gradient(low = "lightblue", high = "darkblue") +
  labs(
    title = "24-Hour Movement of Top 3 DBSCAN Clusters (their center)",
    x = "Average Longitude",
    y = "Average Latitude",
    color = "Hour of Day"
  ) +
  theme_minimal()
```



Interesting visualization proving there are some pickup points trends based on an hour.

6 6. Forecasting

6.1 6.1. Data Prep.

```
# Aggregate trips by day from the FULL dataset
daily_trips <- uber_data %>%
  count(date, name = "total_trips")

# Convert to a time series (`ts`) object
# to capture the weekly pattern (seasonality).
trips_ts <- ts(daily_trips$total_trips, frequency = 7)

print("Daily trip counts (first 6 days):")
```

```
## [1] "Daily trip counts (first 6 days):"
```

```
head(daily_trips)
```

```
## # A tibble: 6 × 2
##   date       total_trips
##   <date>           <int>
## 1 2014-09-01       19961
## 2 2014-09-02       28831
## 3 2014-09-03       32631
## 4 2014-09-04       38360
## 5 2014-09-05       42319
## 6 2014-09-06       40520
```

6.2 6.2. ARIMA Model Fit and Forecast

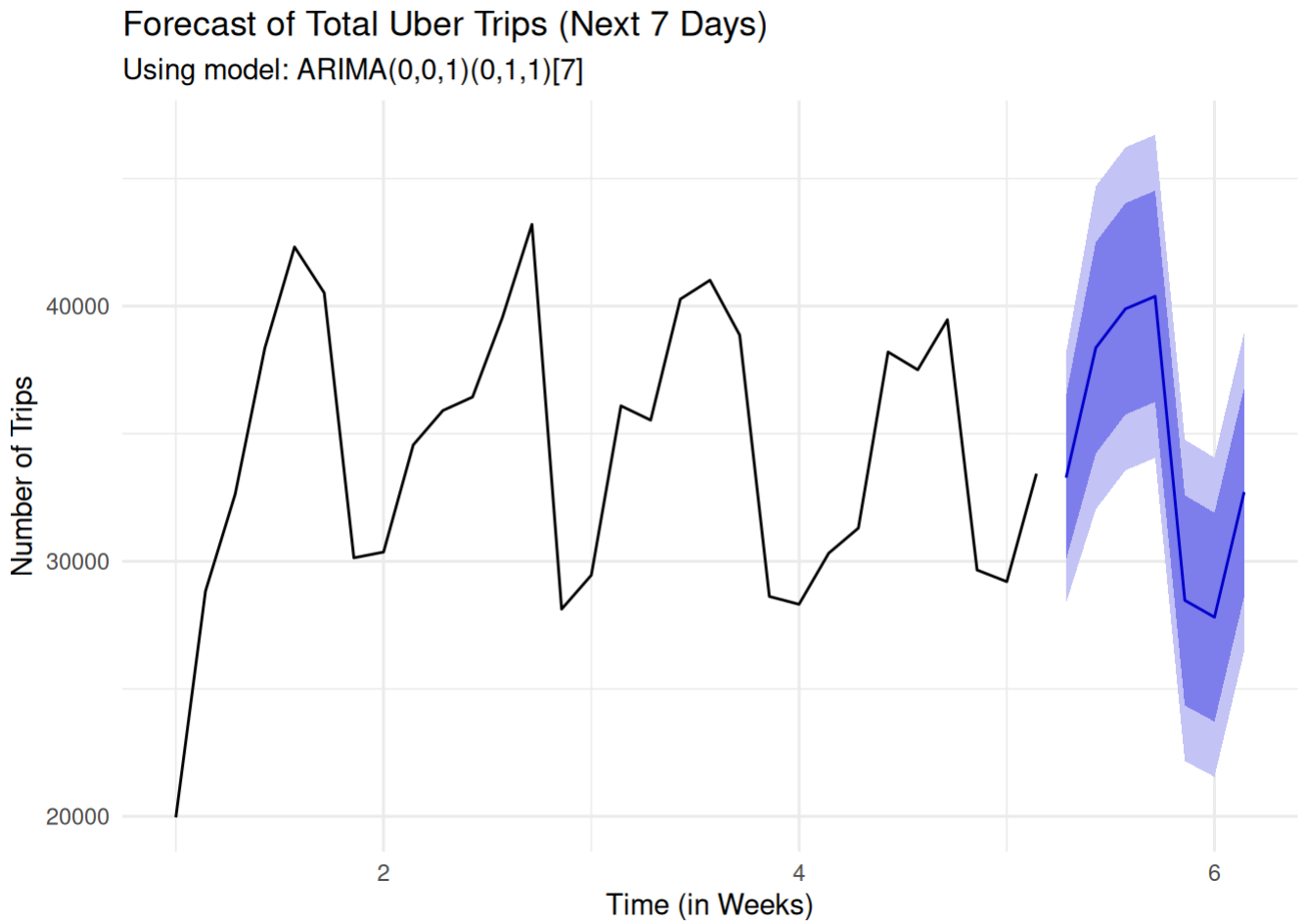
```
# 'auto.arima' finds the best seasonal ARIMA model automatically.
ts_fit <- auto.arima(trips_ts)

print("Model Summary:")
```

```
## [1] "Model Summary:"
```

```
print(summary(ts_fit))
```

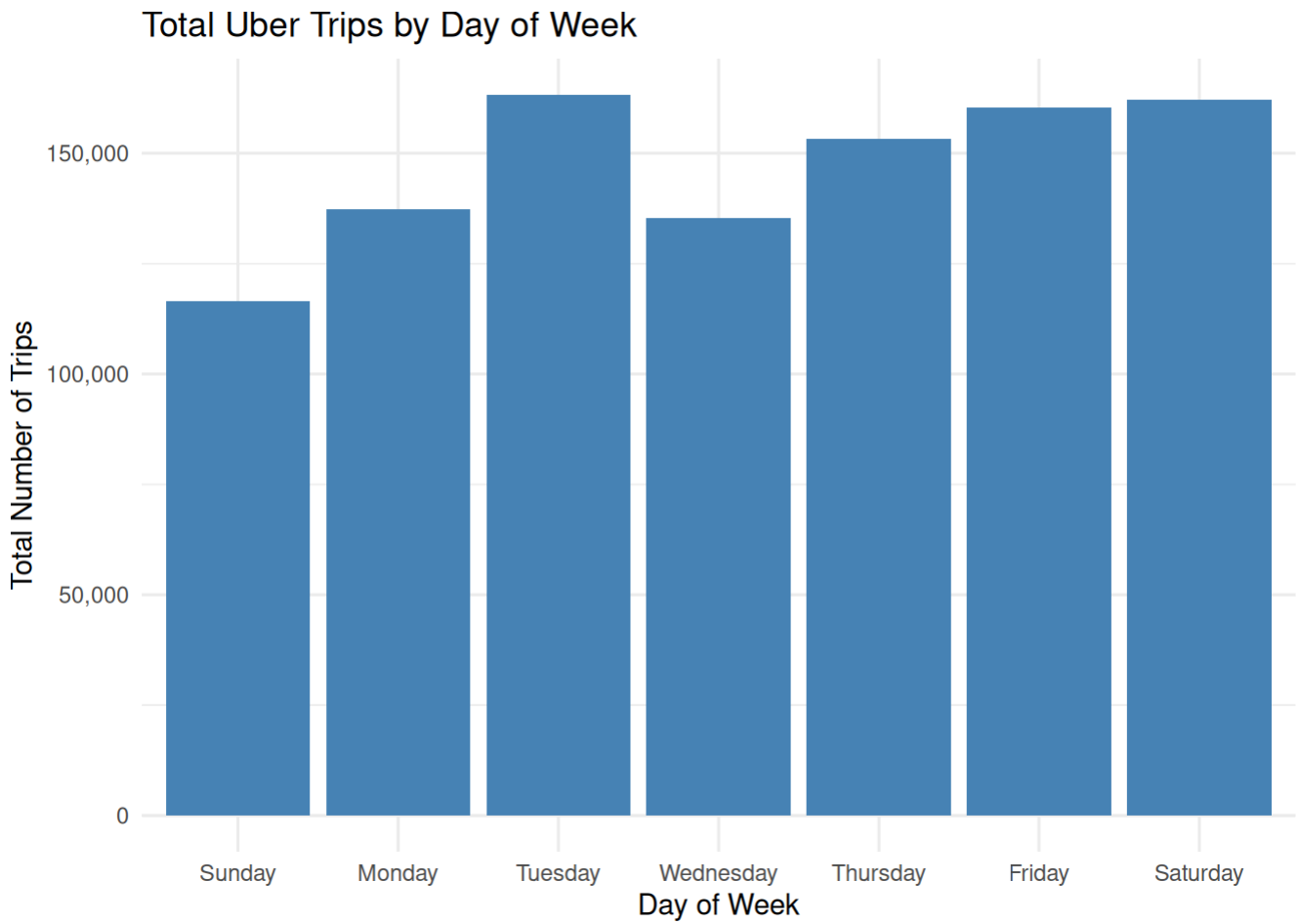
```
## Series: trips_ts
## ARIMA(0,0,1)(0,1,1)[7]
##
## Coefficients:
##          ma1     sma1
##       0.8188  -0.7727
## s.e.  0.1170   0.5898
##
## sigma^2 = 5900375:  log likelihood = -214.03
## AIC=434.07   AICc=435.33   BIC=437.47
##
## Training set error measures:
##                    ME     RMSE      MAE      MPE     MAPE      MASE       ACF1
## Training set 332.0289 2032.305 1367.907 1.096309 4.084306 0.4904574 -0.1829841
```

```
# 4. Create and plot the forecast
# 'h=7' forecasts 7 days into the future.
ts_forecast <- forecast(ts_fit, h = 7)

autoplot(ts_forecast) +
  labs(
    title = "Forecast of Total Uber Trips (Next 7 Days)",
    subtitle = paste("Using model:", ts_forecast$method),
    x = "Time (in Weeks)",
    y = "Number of Trips"
  ) +
  theme_minimal()
```



Blue and gray shaded areas represent the 80% and 95% confidence intervals. Seasonal pattern, but weekends or business daysas busier? Lets find out.

```
# We add 'abbr = FALSE' to get full names like "Monday"
uber_data <- uber_data %>%
  mutate(wday = wday(datetime, label = TRUE, abbr = FALSE))

# 2. Group by the new 'wday' column and count trips
weekday_counts <- uber_data %>%
  group_by(wday) %>%
  summarize(total_trips = n())

# 3. Plot the results
ggplot(weekday_counts, aes(x = wday, y = total_trips)) +
  geom_col(fill = "steelblue") +
  labs(
    title = "Total Uber Trips by Day of Week",
    x = "Day of Week",
    y = "Total Number of Trips"
  ) +
  # Format the y-axis to have commas (e.g., 100,000)
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()
```



Tuesday defined as the busiest day for Uber drivers in the largest clusters.