November 28th project update

This update explores data obtained through Google API directions requests. Three classes of requests were made. For each class of requests, a different mode of transportation is requested to make the trip. These modes were subway, driving, and walking. Directions were found for every single pair of NTAs. The directions were considered undirected- the significant characteristics of the trips are the same in each direction. This resulted in 1225 routes (50 choose 2). Each trip was given a departure time of 9:00am EST for Nov 21. The driving time used the "best guess" Google traffic model. The travel time for each transportation mode was saved to a csv file. For the subway trips, the number of subway lines required to make the trip are also recorded. Each route starts and ends in a piont in the NTA that is found using the sf 'point on surface' algorithm. This results in points that are roughly centered and guaranteed to be in the NTA.

Libraries

```
library(tidyverse)
library(tidygraph)
library(ggraph)
library(igraph)
library(tmap)
library(sf)
```

Files

```
## LODES data
nys_od <- readr::read_csv('data/ny_od_main_JT00_2019.csv')
## Areas for all nyc NTAs
nyc_nta_borders <- sf::st_read('data/nyc_2010_nta_borders.geojson')
```

```
## Reading layer `nyc_2010_nta_borders' from data source
##   `/home/miller/GeoI/fall-2022/gtech_705-spatial_anlysis/triboro-line/brooklyn-lodes/data/nyc_2010_n
##   using driver `GeoJSON'
## Simple feature collection with 195 features and 8 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -74.25559 ymin: 40.49614 xmax: -73.70001 ymax: 40.91554
## Geodetic CRS:  WGS 84
```

```
## Dataset to link census tracts with their NTA
nyc_nta_tract_equiv <- readxl::read_xlsx('data/nyc_2010_census_tract_nta_equiv.xlsx')

## The number of trains that are required for the trip
subway_lines <- readr::read_csv('data/nta-subway-lines.csv')

## The amount of time taken to travel by subway, car, or foot
subway_times <- readr::read_csv('data/nta-subway-times.csv')
driving_times <- readr::read_csv('data/nta-driving-times.csv')
walking_times <- readr::read_csv('data/nta-walking-times.csv')

## Infrastructure
nyc_boro_borders <- sf::st_read("./data/boro_boundaries.geojson")
```

```
## Reading layer `boro_boundaries' from data source
##   `/home/miller/GeoI/fall-2022/gtech_705-spatial_anlysis/triboro-line/brooklyn-lodes/data/boro_bound
##   using driver `GeoJSON'
## Simple feature collection with 5 features and 4 fields
```

```
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -74.25559 ymin: 40.49613 xmax: -73.70001 ymax: 40.91553
## Geodetic CRS:  WGS 84
```

```
major_roads <- sf::st_read("./data/DCM_ArterialsMajorStreets.geojson")
```

```
## Reading layer `DCM_ArterialsMajorStreets' from data source
##   `/home/miller/GeoI/fall-2022/gtech_705-spatial_anlysis/triboro-line/brooklyn-lodes/data/DCM_Arteria
##   using driver `GeoJSON'
## Simple feature collection with 725 features and 6 fields
## Geometry type: MULTILINESTRING
## Dimension:     XY
## Bounding box:  xmin: -74.25521 ymin: 40.49657 xmax: -73.70001 ymax: 40.91296
## Geodetic CRS:  WGS 84
```

```
subways <- sf::st_read("./data/subway_lines.geojson")
```

```
## Reading layer `subway_lines' from data source
##   `/home/miller/GeoI/fall-2022/gtech_705-spatial_anlysis/triboro-line/brooklyn-lodes/data/subway_line
##   using driver `GeoJSON'
## Simple feature collection with 742 features and 6 fields
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: -74.03088 ymin: 40.57559 xmax: -73.75541 ymax: 40.90312
## Geodetic CRS:  WGS 84
```

Data

```
bk_name <- "Brooklyn"
bk_county_code <- "047"
bk_parks <- "BK99"

bk_nta_border <- nyc_nta_borders %>%
  dplyr::filter(BoroName == bk_name)  %>%
  dplyr::filter(NTACode != bk_parks) %>%
  dplyr::select("NTACode", "Shape__Area")

bk_nta_centers <- bk_nta_border %>%
  dplyr::mutate(
    geometry = sf::st_point_on_surface(geometry)
  ) %>%
  dplyr::select(NTACode)
```

```
## Warning in st_point_on_surface.sfc(geometry): st_point_on_surface may not give
## correct results for longitude/latitude data
```

```
bk_nta_tract_equiv <- nyc_nta_tract_equiv %>%
  dplyr::filter(borough_name == bk_name) %>%
  dplyr::filter(nta_code != bk_parks) %>%
  dplyr::rename(tract = census_tract) %>%
  dplyr::select("tract", "nta_code")

od <- nys_od %>%
  dplyr::filter(
    stringr::str_sub(as.character(w_geocode), 3, 5) == bk_county_code &
```

```
    stringr::str_sub(as.character(h_geocode), 3, 5) == bk_county_code
  ) %>%
  dplyr::mutate(
   w_tract = stringr::str_sub(as.character(w_geocode), 6, 11)
  ) %>%
  dplyr::mutate(
   h_tract = stringr::str_sub(as.character(h_geocode), 6, 11)
  ) %>%
  dplyr::select("w_tract", "h_tract", "S000") %>%
  dplyr::left_join(bk_nta_tract_equiv, c("w_tract" = "tract")) %>%
  dplyr::rename(w_nta_code = nta_code) %>%
  dplyr::left_join(bk_nta_tract_equiv, c("h_tract" = "tract")) %>%
  dplyr::rename(h_nta_code = nta_code) %>%
  dplyr::select("h_nta_code", "w_nta_code", "S000") %>%
  dplyr::filter(w_nta_code != bk_parks & h_nta_code != bk_parks)

bk_boro_border <- nyc_boro_borders %>%
  dplyr::filter(boro_name == bk_name)

bk_subways <- subways %>%
  sf::st_intersection(bk_boro_border)
```

```
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

```
bk_major_roads <- major_roads %>%
  sf::st_intersection(bk_nta_border)
```

```
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

Exploratory data analysis

Distribution of job counts

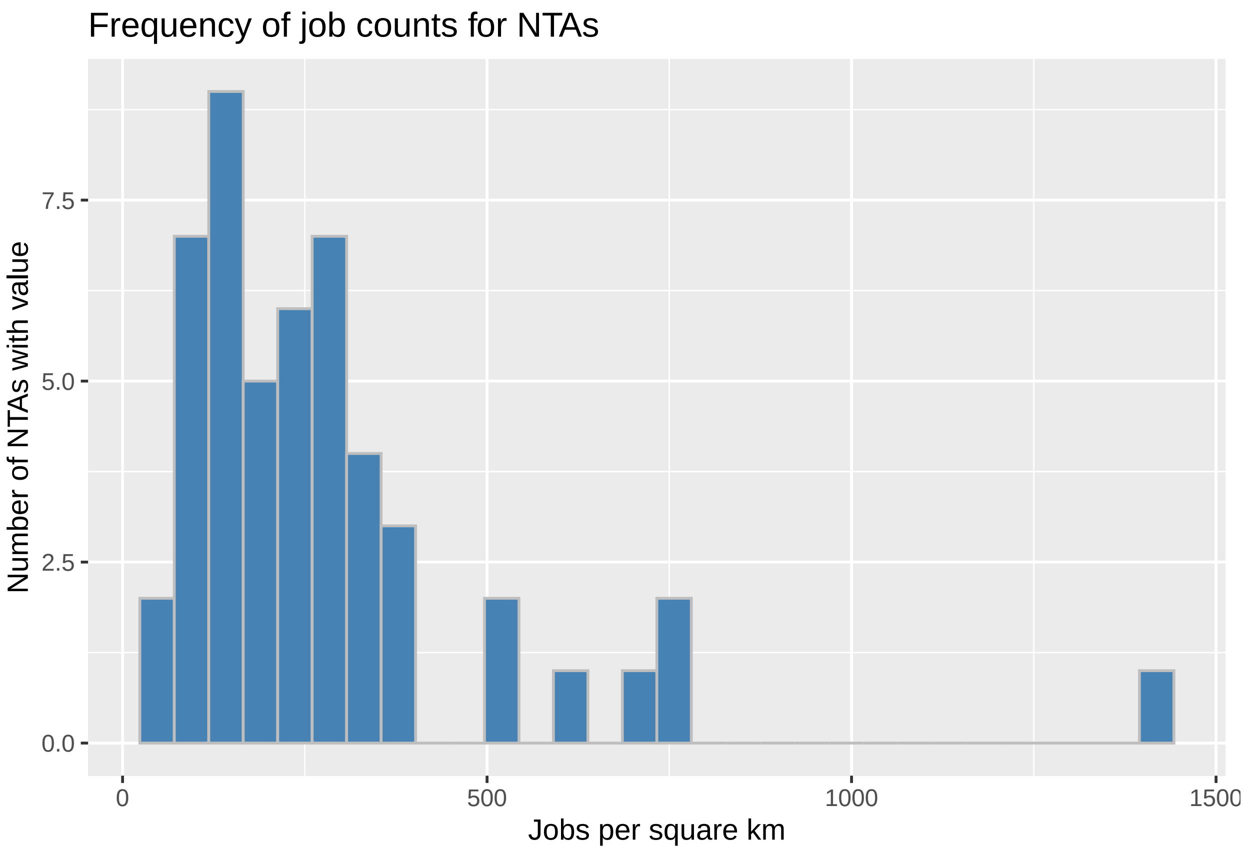
The job count of an NTA is equivalent to all of the trips that list the NTA as the destination.

Because each NTA has a different area, I felt it made sense to standardized the job count to a per square km area value, ie) Jobs/km2.

```
job_counts <- od %>%
  dplyr::group_by(w_nta_code) %>%
  dplyr::summarise(
    S000 = sum(S000),
  ) %>%
  unique() %>%
  dplyr::left_join(
    bk_nta_border, c("w_nta_code" = "NTACode")
  ) %>%
  mutate(
    S000_km2 = S000 / (Shape__Area / 1e6),
    log_S000_km2 = log(S000_km2)
  ) %>%
  sf::st_as_sf()
```

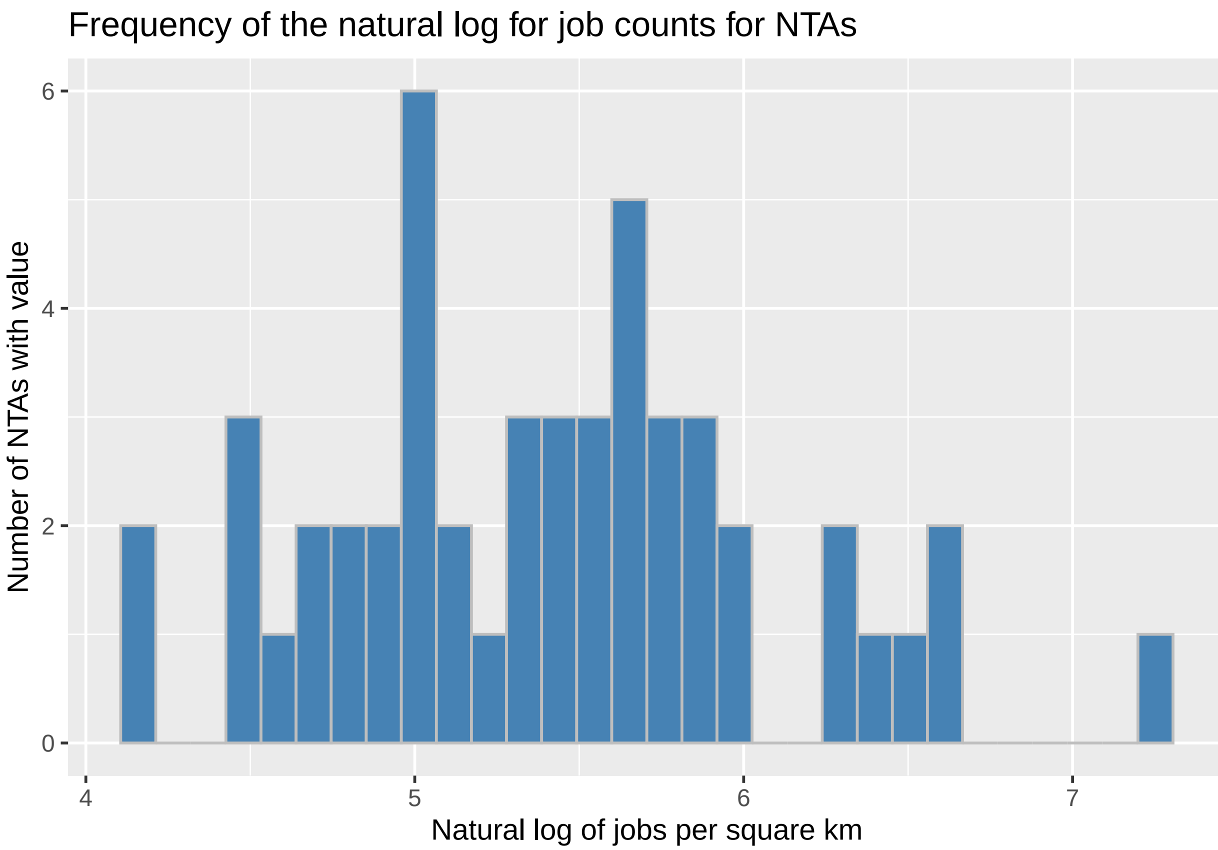
The original distribution of jobs/km2 is skewed to the right.

```r
ggplot(job_counts) +
  ggtitle("Frequency of job counts for NTAs") +
  xlab("Jobs per square km") +
  ylab("Number of NTAs with value") +
  geom_histogram(aes(x = S000_km2), fill = "steelblue", color = "grey", bins = "30")
```



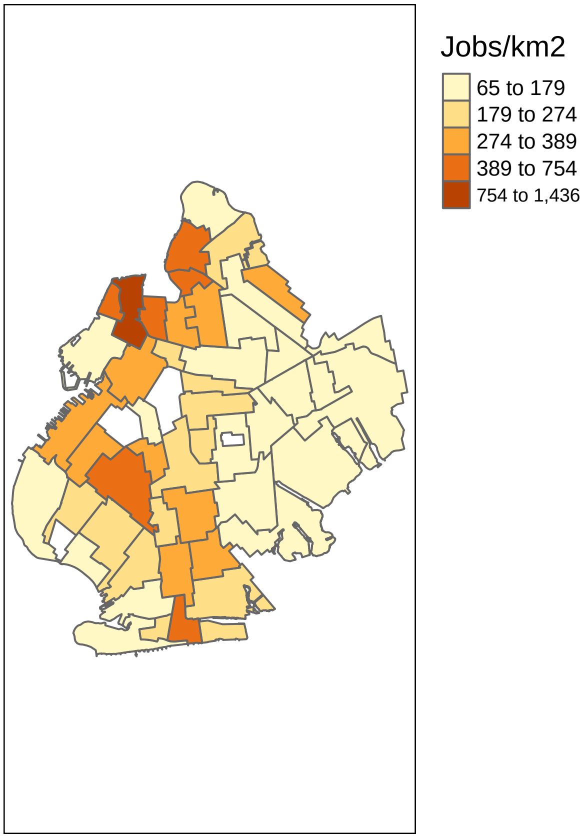
Applying a log transformation to the job count makes it more normal

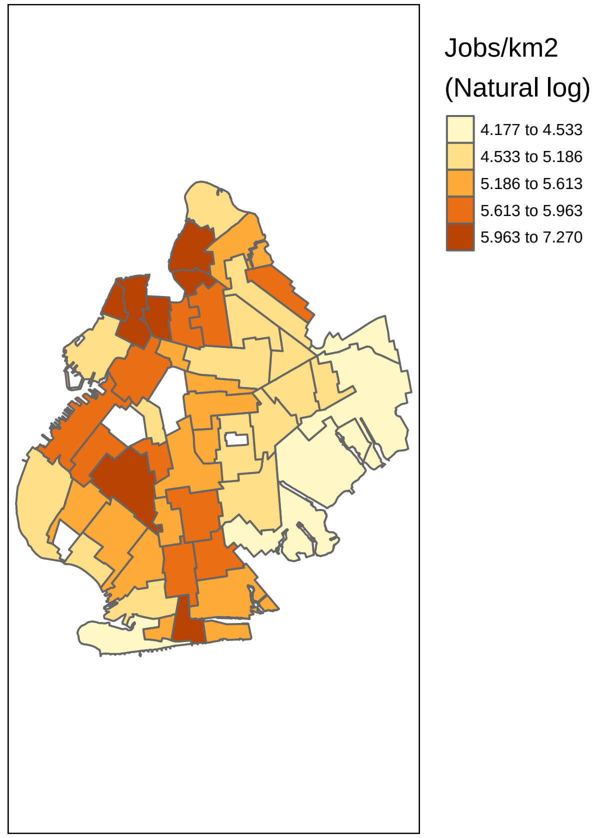
```r
ggplot(job_counts) +
  ggtitle("Frequency of the natural log for job counts for NTAs") +
  xlab("Natural log of jobs per square km") +
  ylab("Number of NTAs with value") +
  geom_histogram(aes(x = log_S000_km2), fill = "steelblue", color = "grey", bins = "30")
```



Maps of job count

```
map_original_jobs <- tmap::tm_shape(job_counts) +
  tmap::tm_polygons(
    col = "S000_km2",
    style = "jenks",
    title = "Jobs/km2"
  ) +
  tmap::tm_layout(
    legend.outside = TRUE
  )

map_log_jobs <- tmap::tm_shape(job_counts) +
  tmap::tm_polygons(
    col = "log_S000_km2",
    style = "jenks",
    title = "Jobs/km2\n(Natural log)"
  ) +
  tmap::tm_layout(
    legend.outside = TRUE
  )

tmap::tmap_arrange(map_original_jobs, map_log_jobs)
```





Distribution of commute counts

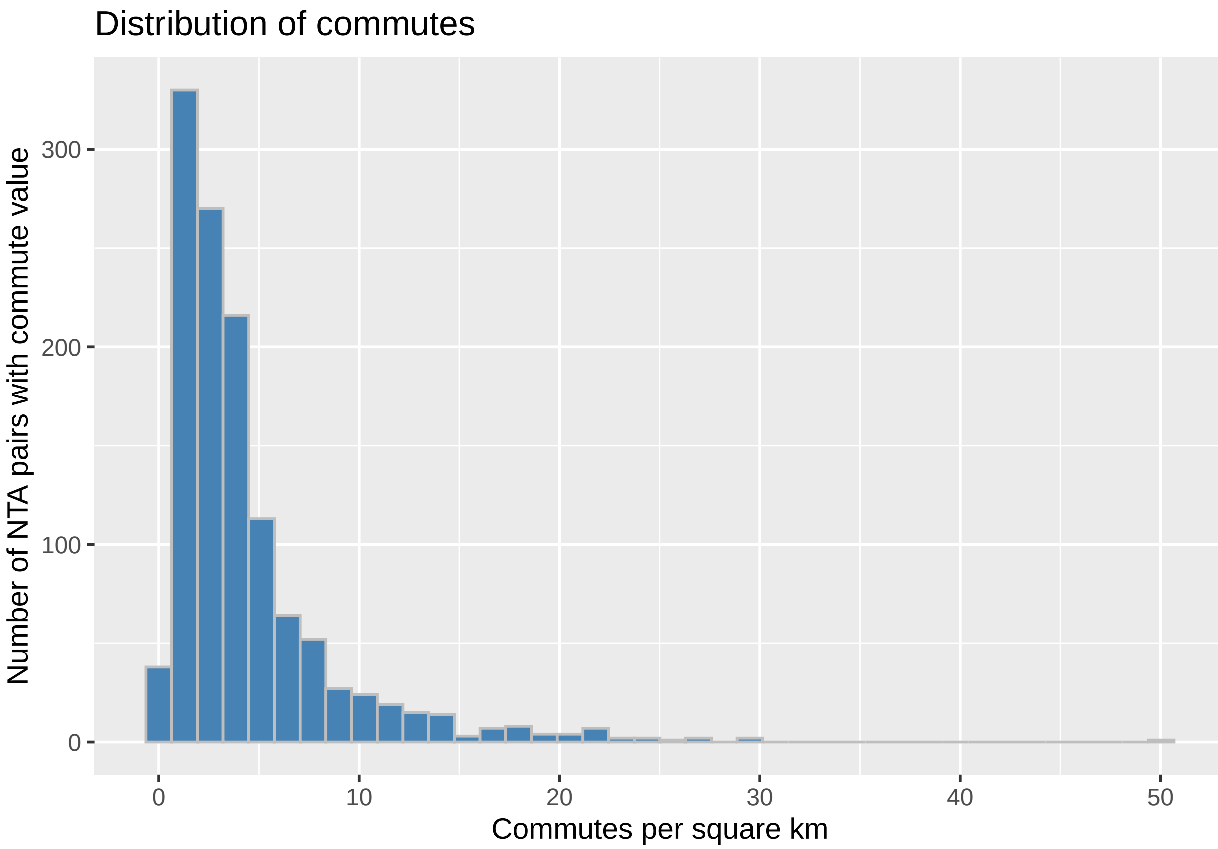
Commutes are the number of trips between two distinct NTAs. Like Job count, commute count is affected by the size of the NTAs involved in the commute. To standardize the commute count, the total number of commutes is divided by the total area of the two NTAs. The resulting unit is commutes/km2.

```
## Use the NTA commute data that is available in any of the infrastructure data
## (Subway times is an arbitrary choice among the suitable files)
commute_counts <- subway_times %>%
  dplyr::select(-seconds_in_transit) %>%
  dplyr::left_join(bk_nta_border, c("nta_one" = "NTACode")) %>%
  dplyr::rename(
    shape_area_one = Shape__Area,
    geometry_one = geometry,
  ) %>%
  dplyr::left_join(bk_nta_border, c("nta_two" = "NTACode")) %>%
  dplyr::rename(
    shape_area_two = Shape__Area,
    geometry_two = geometry
  ) %>%
  dplyr::mutate(
    S000_km2 = S000 / ((shape_area_one / 1e6) + (shape_area_two / 1e6)),
    log_S000_km2 = log(S000_km2)
  )
```

The original distribution of commute counts is skewed to the right

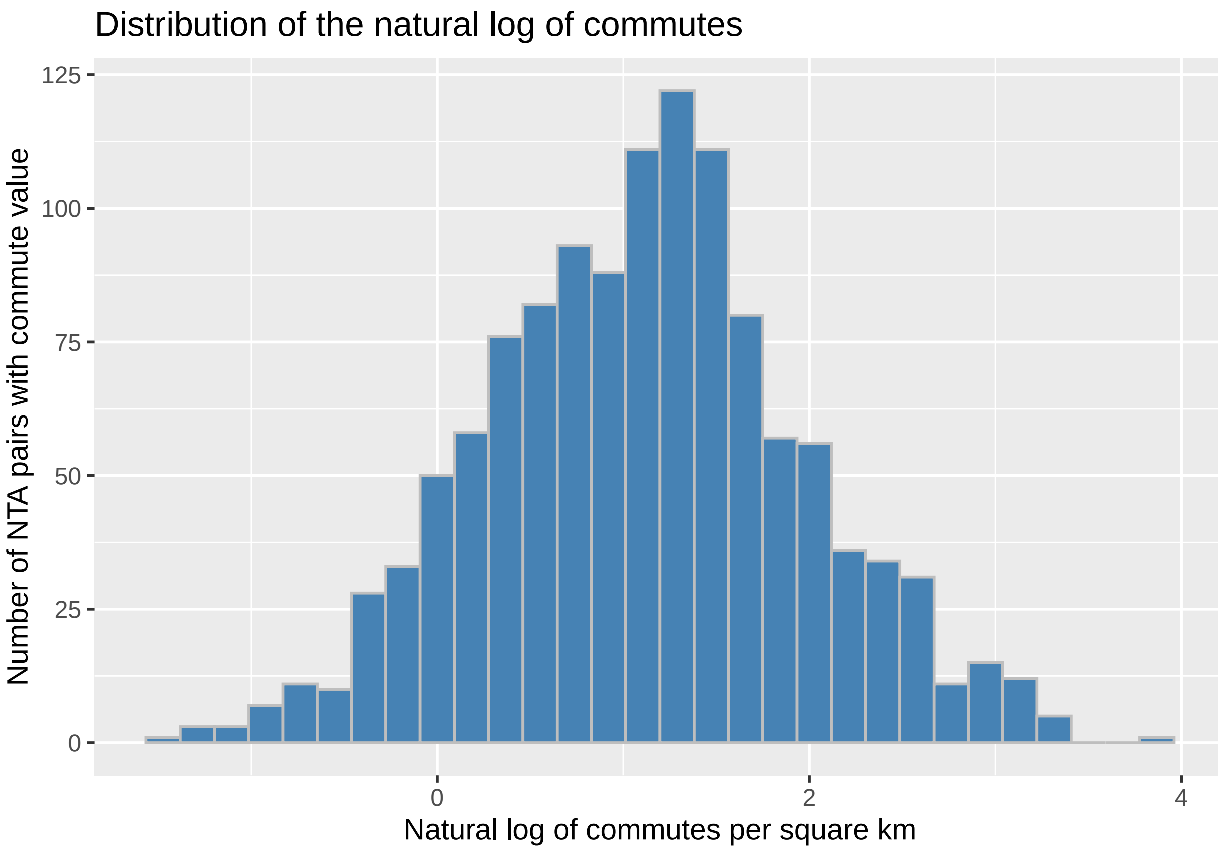
```
ggplot(commute_counts) +
  ggtitle("Distribution of commutes") +
```

```
  xlab("Commutes per square km") +
  ylab("Number of NTA pairs with commute value") +
  geom_histogram(aes(x = S000_km2), fill = "steelblue", color = "grey", bins = 40)
```



Using the natural log of commutes makes the data more normal

```
ggplot(data = commute_counts) +
  ggtitle("Distribution of the natural log of commutes") +
  xlab("Natural log of commutes per square km") +
  ylab("Number of NTA pairs with commute value") +
  geom_histogram(aes(x = log_S000_km2), bins = 30, fill = "steelblue", color = "grey")
```

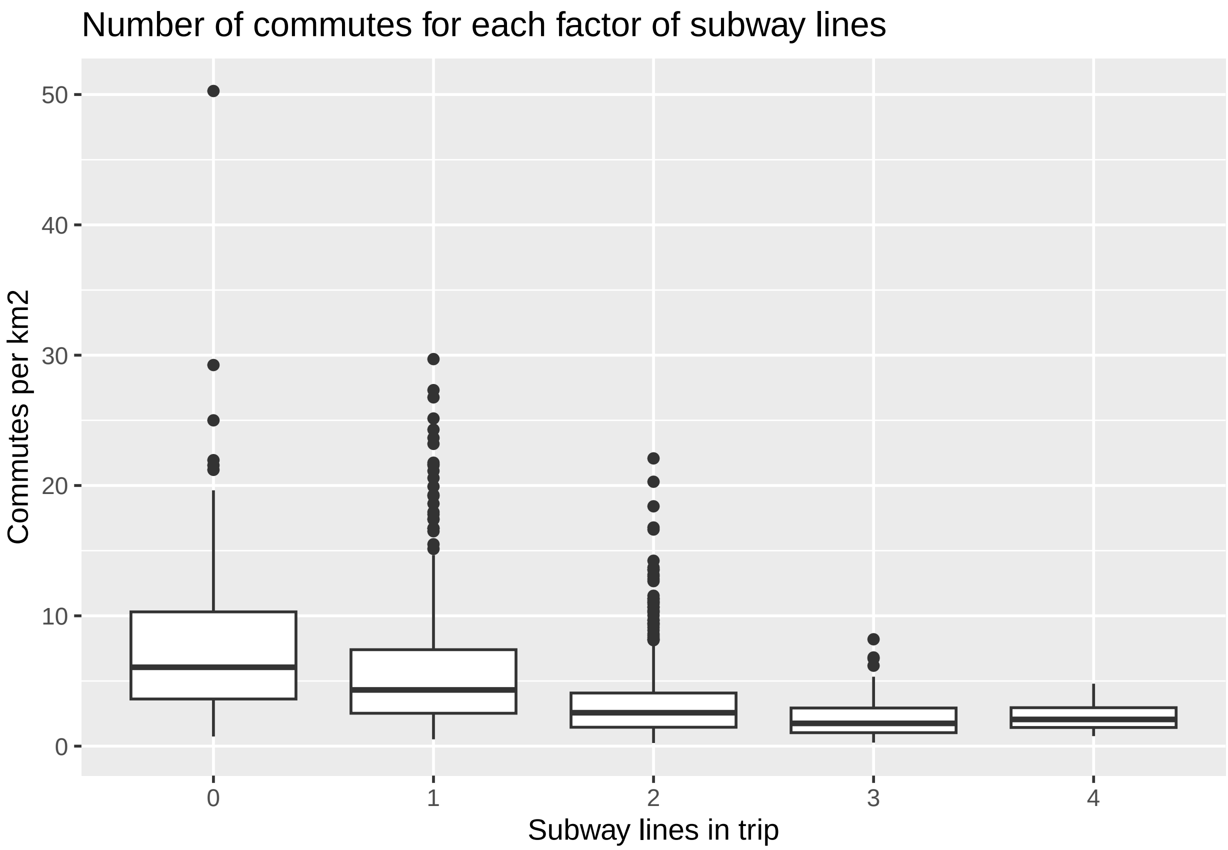


Number of subway lines and commute count

```
subway_lines <- subway_lines %>%
  dplyr::mutate(
    line_count_ordinal = as.character(line_count),
    S000_km2 = commute_counts$S000_km2,
    log_S000_km2 = commute_counts$log_S000_km2,
  )
```

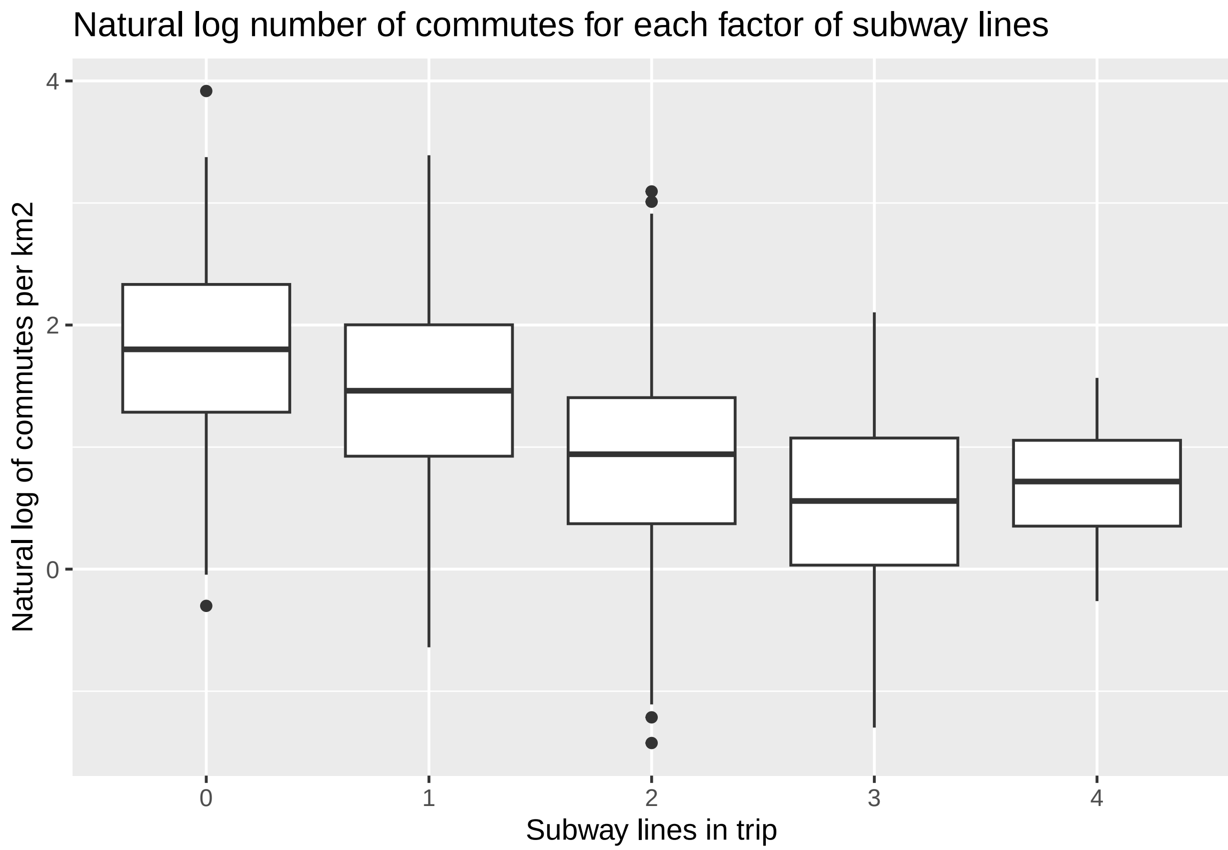
The number of subway lines that are involved in each trip. If zero subway lines are involved, then a subway route was not available or the available route was worse than walking.

```
ggplot(data = subway_lines, aes(x = line_count_ordinal, y = S000_km2)) +
  ggtitle("Number of commutes for each factor of subway lines") +
  xlab("Subway lines in trip") +
  ylab("Commutes per km2") +
  geom_boxplot()
```



The number of commutes decreases as the number of lines increases from 0 to 3. The number of commutes is stable between 3 and 4 subway lines.

```
ggplot(data = subway_lines, aes(x  = line_count_ordinal, y = log_S000_km2)) +
  ggtitle("Natural log number of commutes for each factor of subway lines") +
  xlab("Subway lines in trip") +
  ylab("Natural log of commutes per km2") +
  geom_boxplot()
```



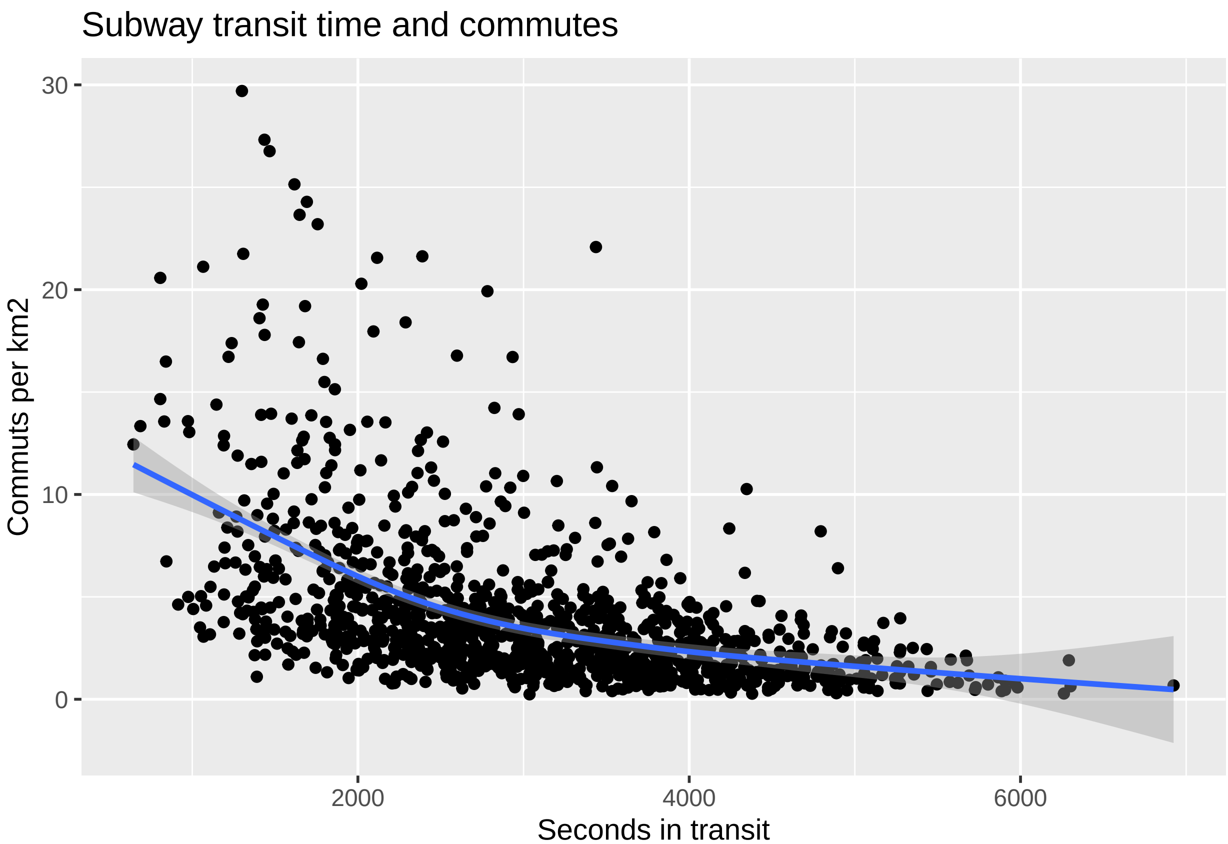
Subway Transit time and commute count

Not every route is possible using the subway system. For the linear regression, I am interested in the relationship between routes using the infrastructure and the number of trips made. Consequently, for the linear regression, I removed the trips where it's not possible or practical to use the subway. This removes 77 routes, leaving 1148 routes.

```
subway_times <- subway_times %>%
  dplyr::mutate(
    log_S000 = log(S000),
    S000_km2 = commute_counts$S000_km2,
    log_S000_km2 = commute_counts$log_S000_km2,
    i_seconds_in_transit = 1 / (seconds_in_transit^(1/8))
  )
subway_times_connected <- subway_times %>%
  dplyr::filter(
    subway_lines$line_count > 0
  )
```

There is a non-linear and negative monotonic relationship between the time of trip and the number of commutes that are made along this trip.

```
ggplot(data = subway_times_connected, aes(x = seconds_in_transit, y = S000_km2)) +
  geom_point() +
  ggtitle("Subway transit time and commutes") +
  xlab("Seconds in transit") +
  ylab("Commuts per km2") +
  stat_smooth()
```

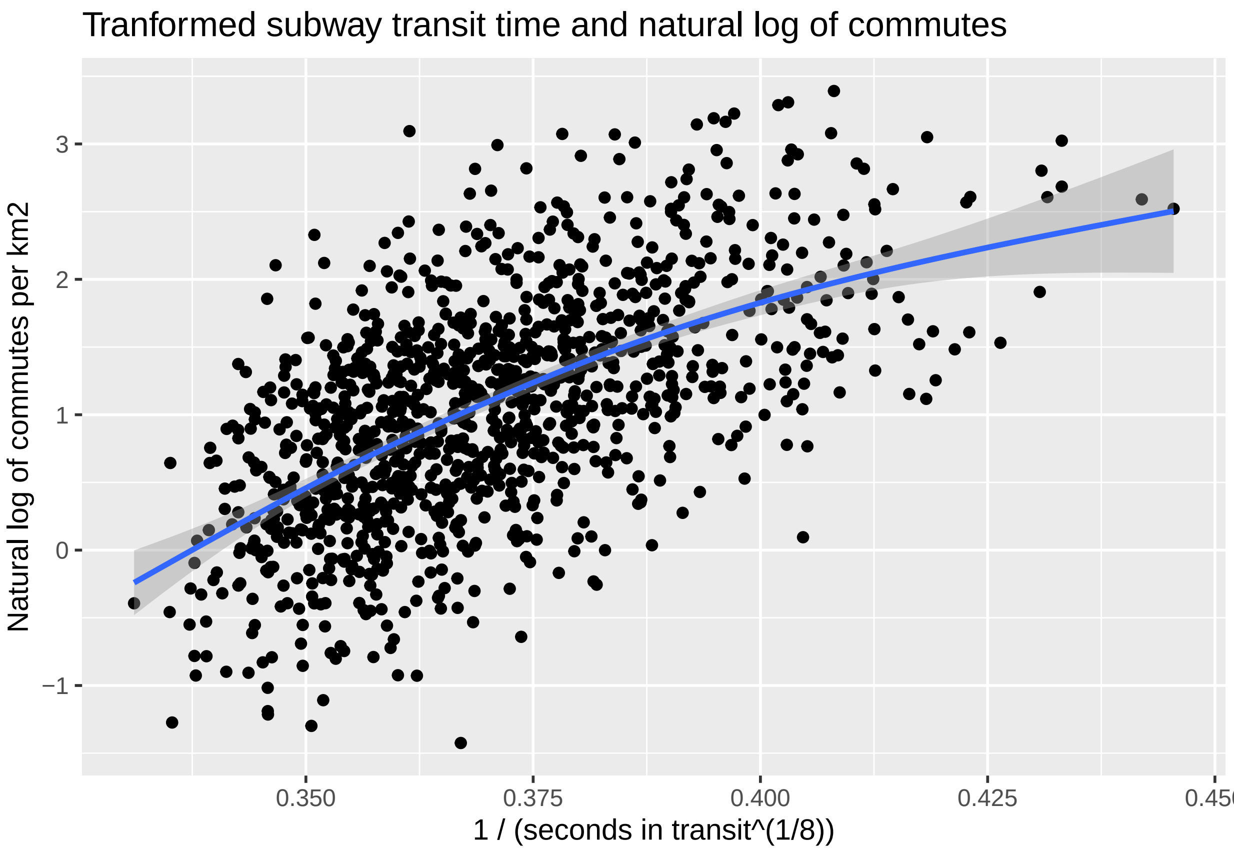


There appears to be heteroscedasticity. We know from the histogram of commute times that the original distribution is skewed to the right. To resolve these issues, we again use the natural log of commutes per km2.

We are also interested in creating a positive relationship between time and commutes. For this, we take the inverse of time. To remove as much non-linearity as practical, we also perform a power transformation, raising time to the power of 1/8.

The result is a data set in which a linear relationship is an acceptable model.

```
ggplot(data = subway_times_connected, aes(x = i_seconds_in_transit, y = log_S000_km2)) +
  ggtitle("Tranformed subway transit time and natural log of commutes") +
  xlab("1 / (seconds in transit^(1/8))") +
  ylab("Natural log of commutes per km2") +
  geom_point() +
  stat_smooth()
```



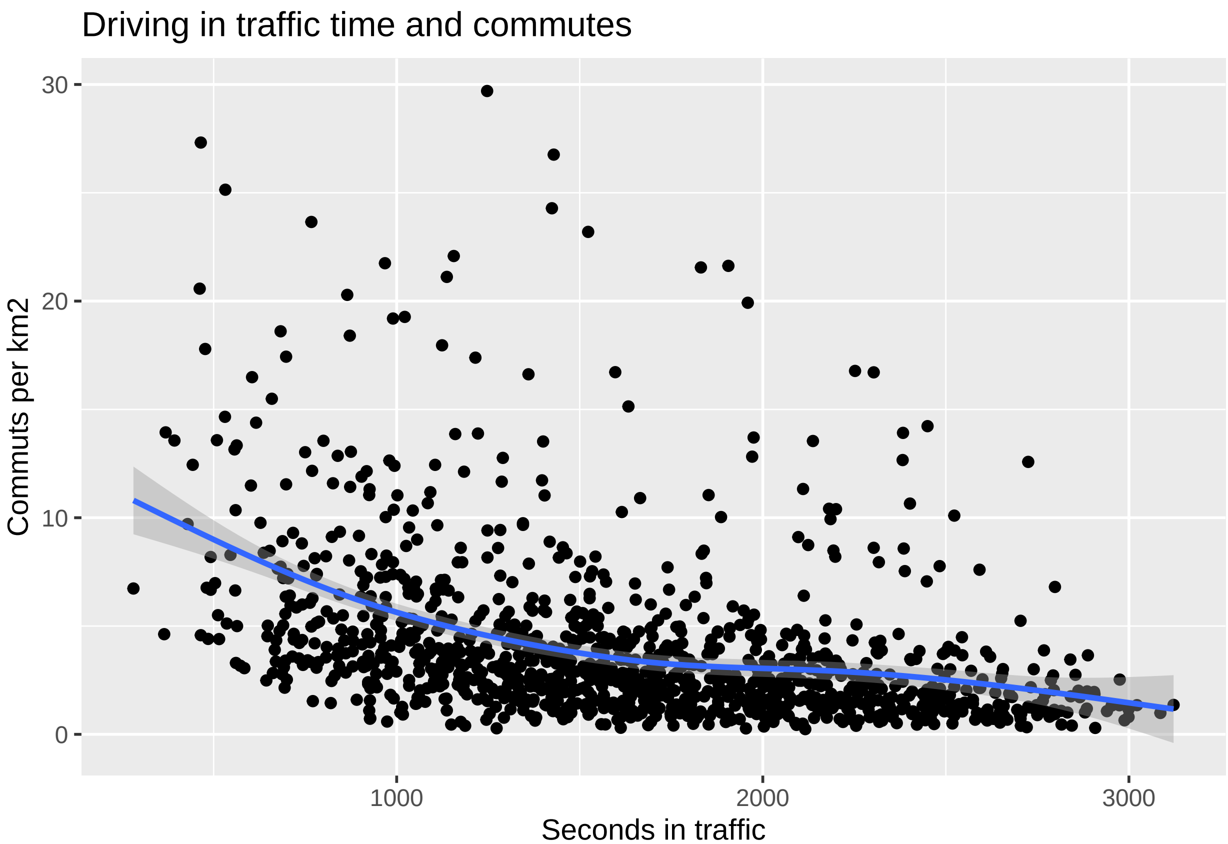
Driving in traffic time and commute count

To make driving times consistent with the subway system, we remove the same 77 data points that were not possible with the subway system.

```
driving_times <- driving_times %>%
  dplyr::mutate(
    log_S000 = log(S000),
    S000_km2 = commute_counts$S000_km2,
    log_S000_km2 = commute_counts$log_S000_km2,
    i_seconds_in_traffic = 1 / seconds_in_traffic^(1/8)
  )

driving_times_reduced <- driving_times %>%
  dplyr::filter(
    trip %in% subway_times_connected$trip
  )
```

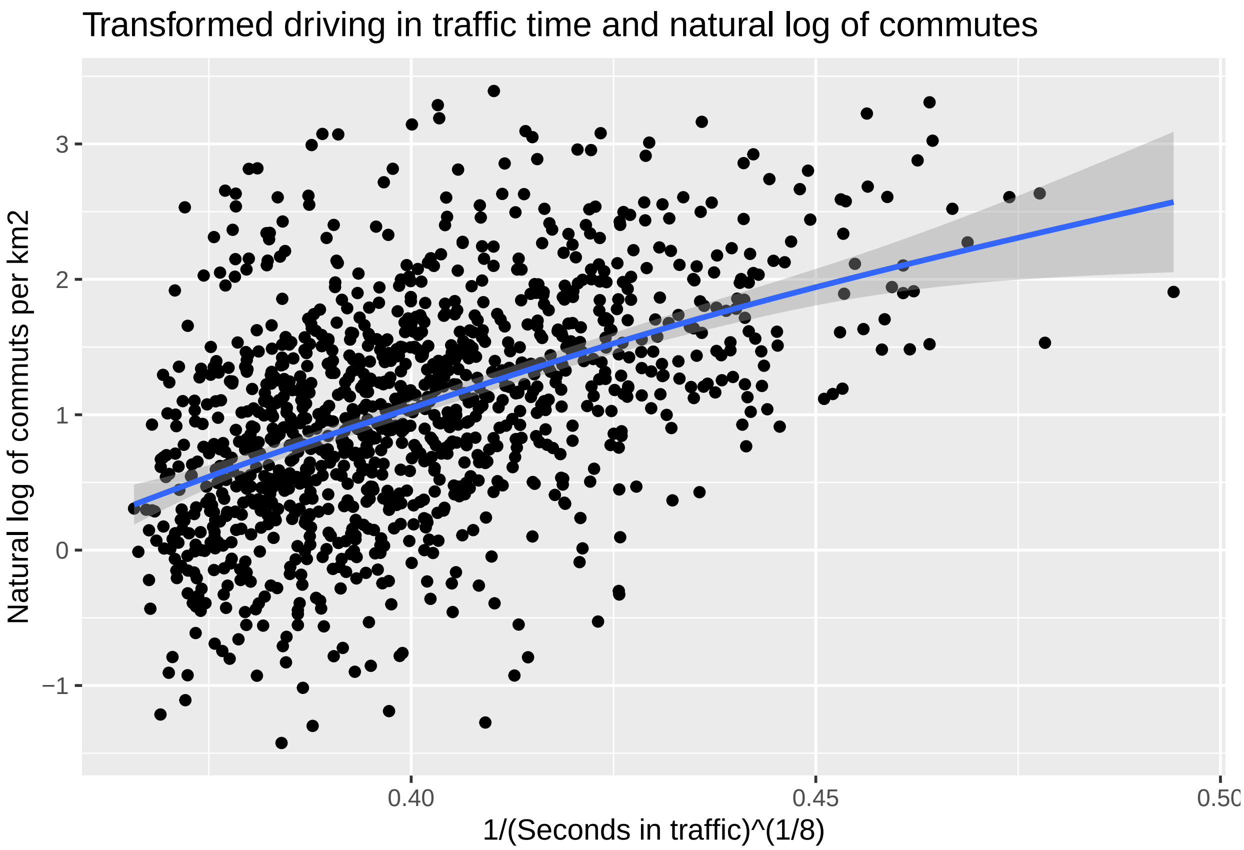
We see a similar pattern to subway time

```
ggplot(data = driving_times_reduced, aes(x = seconds_in_traffic, y = S000_km2)) +
  ggtitle("Driving in traffic time and commutes") +
  xlab("Seconds in traffic")+
  ylab("Commuts per km2") +
  geom_point() +
  stat_smooth()
```



The same transformations for subway trip times are applied to driving in traffic times.

```
ggplot(data = driving_times_reduced, aes(x = i_seconds_in_traffic, y = log_S000_km2)) +
  ggtitle("Transformed driving in traffic time and natural log of commutes") +
  xlab("1/(Seconds in traffic)^(1/8)")+
  ylab("Natural log of commuts per km2") +
  geom_point() +
  stat_smooth()
```

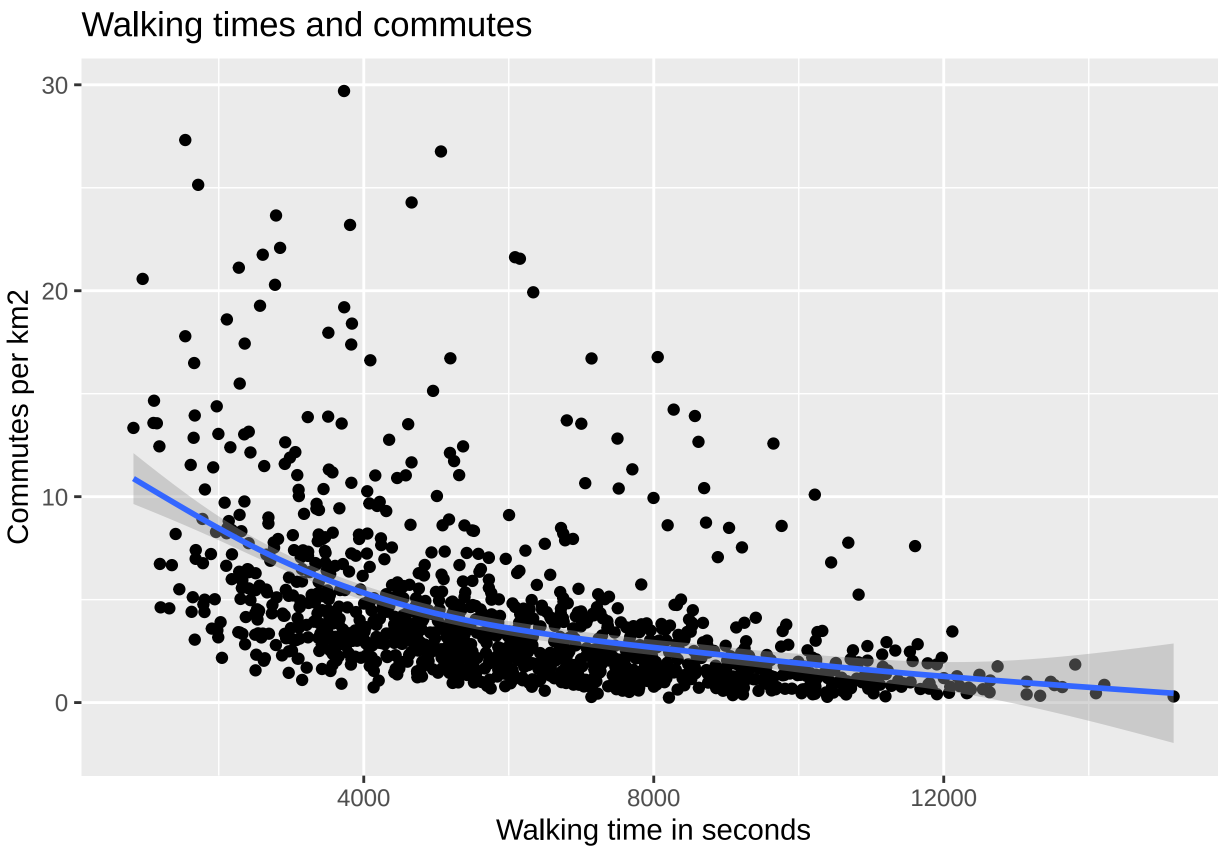


Walking time and commute count

We also remove the 77 points from the walking linear regression and apply the same transformations.

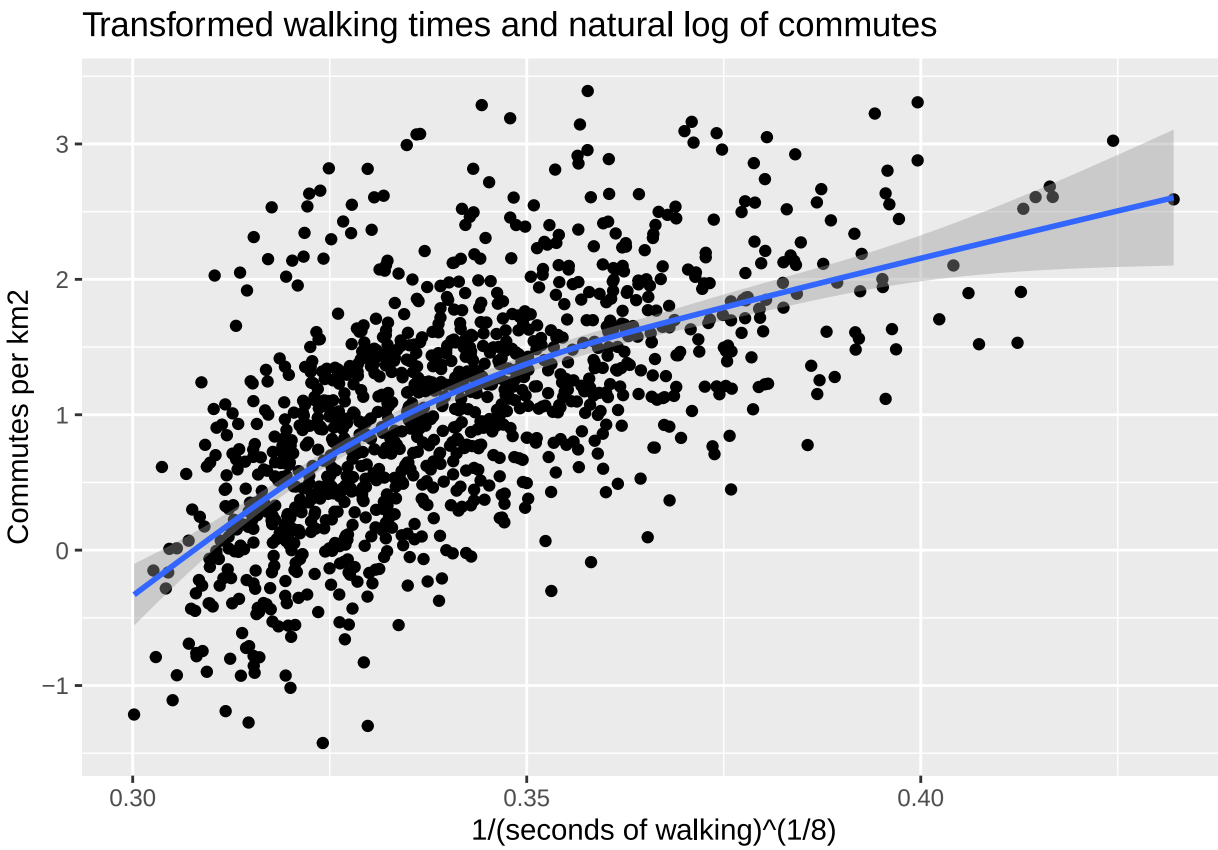
```
walking_times <- walking_times %>%
  dplyr::mutate(
    log_S000 = log(S000),
    S000_km2 = commute_counts$S000_km2,
    log_S000_km2 = commute_counts$log_S000_km2,
    i_seconds_of_walking = 1 / seconds_of_walking^(1/8)
  )

walking_times_reduced <- walking_times %>%
  dplyr::filter(
    trip %in% subway_times_connected$trip
  )
```

```
ggplot(data = walking_times_reduced, aes(x = seconds_of_walking, y = S000_km2)) +
  ggtitle("Walking times and commutes") +
  xlab("Walking time in seconds") +
  ylab("Commutes per km2") +
  geom_point() +
  stat_smooth()
```



Transformed

```
ggplot(data = walking_times_reduced, aes(x = i_seconds_of_walking, y = log_S000_km2)) +
  ggtitle("Transformed walking times and natural log of commutes") +
  xlab("1/(seconds of walking)^(1/8)") +
  ylab("Commutes per km2") +
  geom_point() +
  stat_smooth()
```



Regression of Subway, Driving, and walking

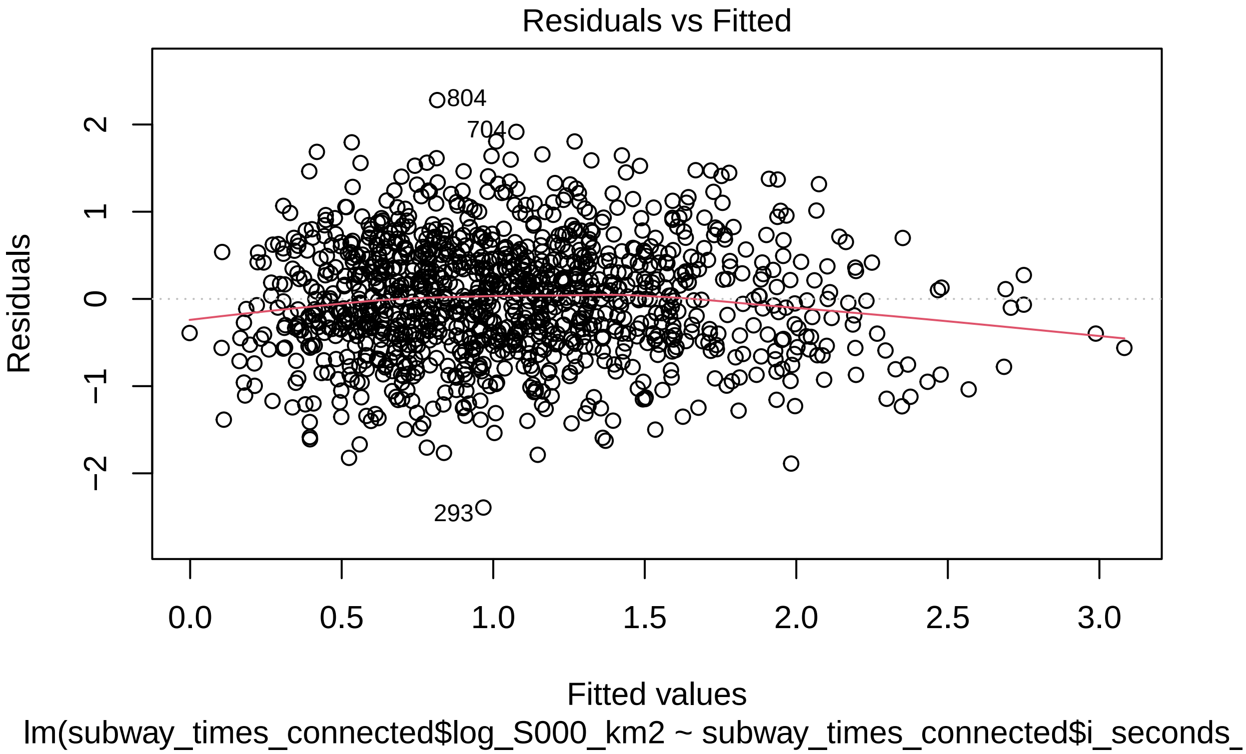
Subway model

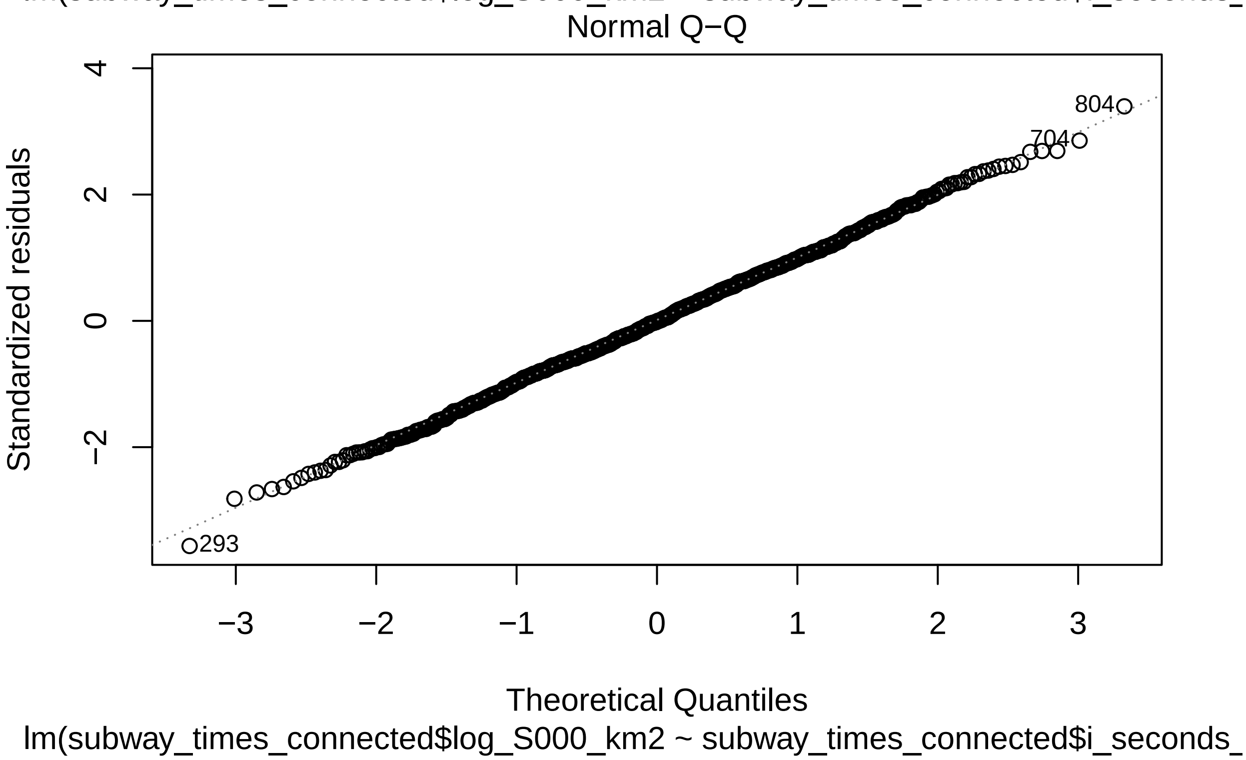
```
subway_connected_model <- lm(subway_times_connected$log_S000_km2 ~ subway_times_connected$i_seconds_in_
summary(subway_connected_model)
```

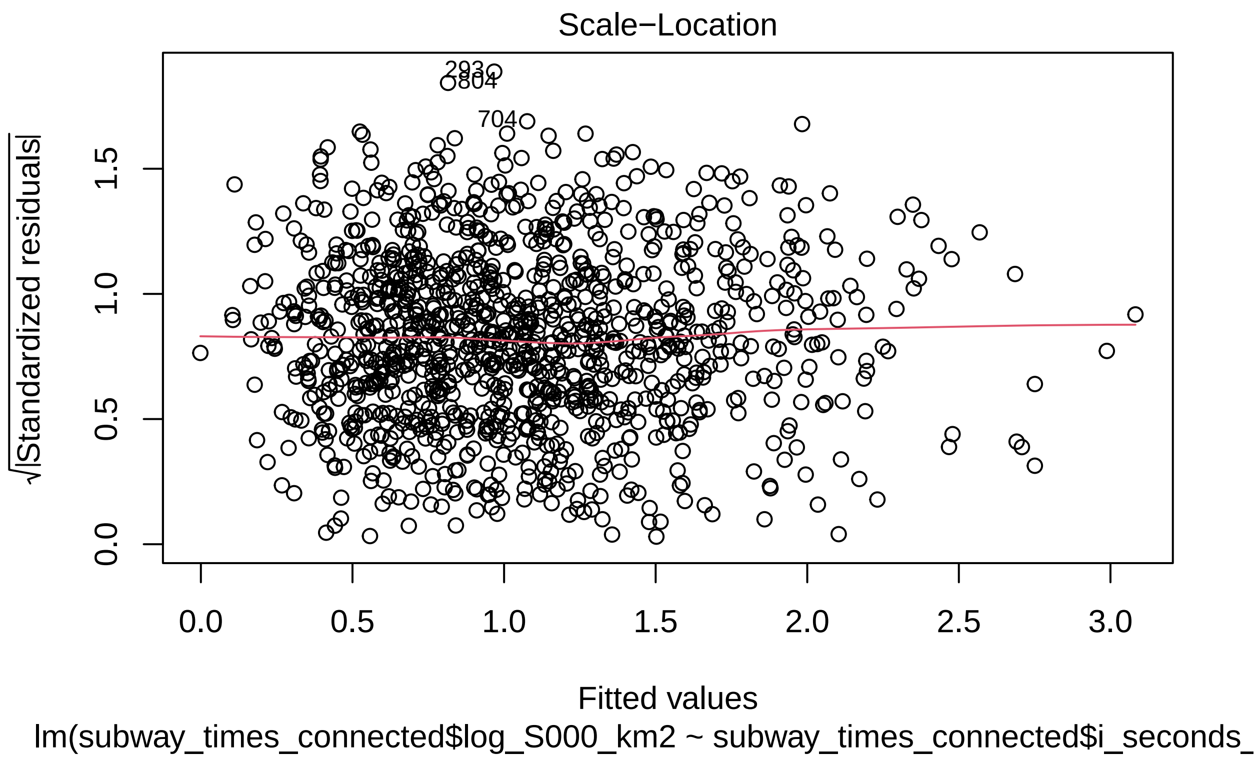
```
##
## Call:
## lm(formula = subway_times_connected$log_S000_km2 ~ subway_times_connected$i_seconds_in_transit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.39218 -0.43877 -0.00603  0.45819  2.27949
##
## Coefficients:
##                                             Estimate Std. Error t value
## (Intercept)                                  -8.9314     0.4023  -22.20
## subway_times_connected$i_seconds_in_transit  26.9699     1.0849   24.86
##                                             Pr(>|t|)
## (Intercept)                                   <2e-16 ***
## subway_times_connected$i_seconds_in_transit   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6714 on 1146 degrees of freedom
## Multiple R-squared:  0.3503, Adjusted R-squared:  0.3498
```

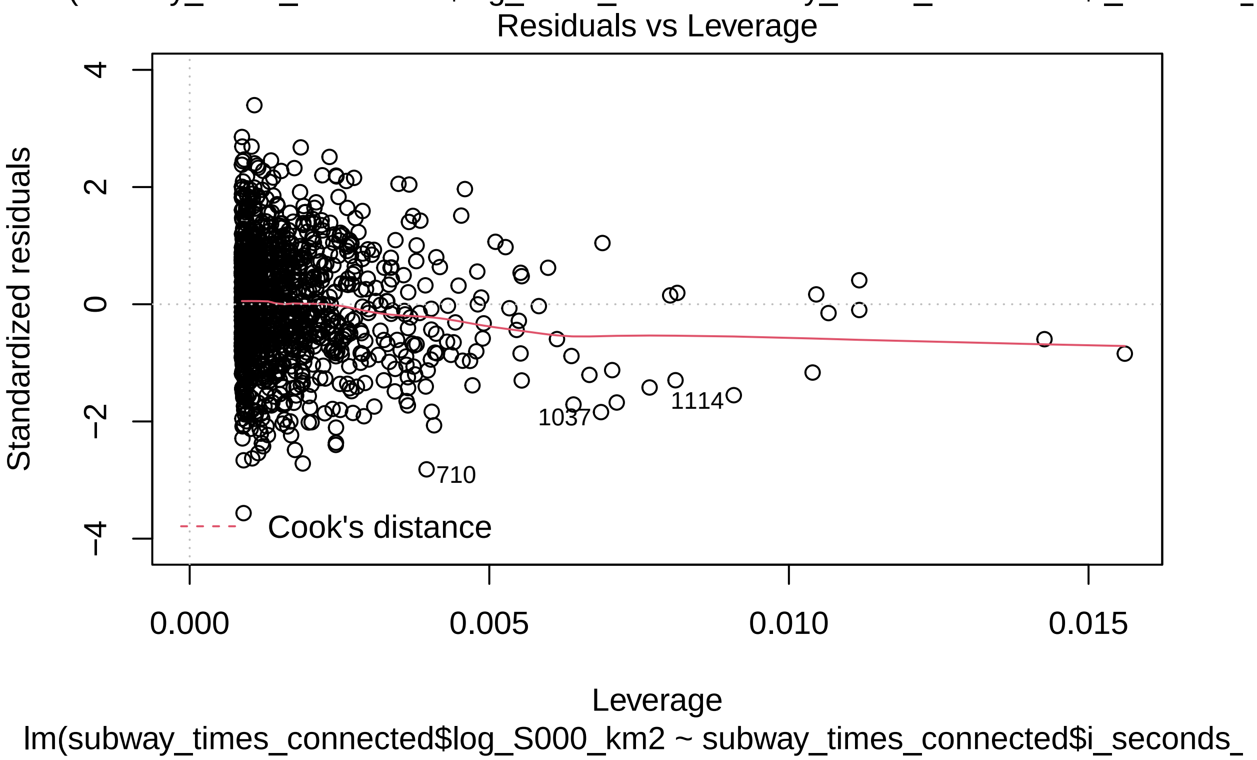
```
## F-statistic:   618 on 1 and 1146 DF,  p-value: < 2.2e-16
```

```
plot(subway_connected_model)
```









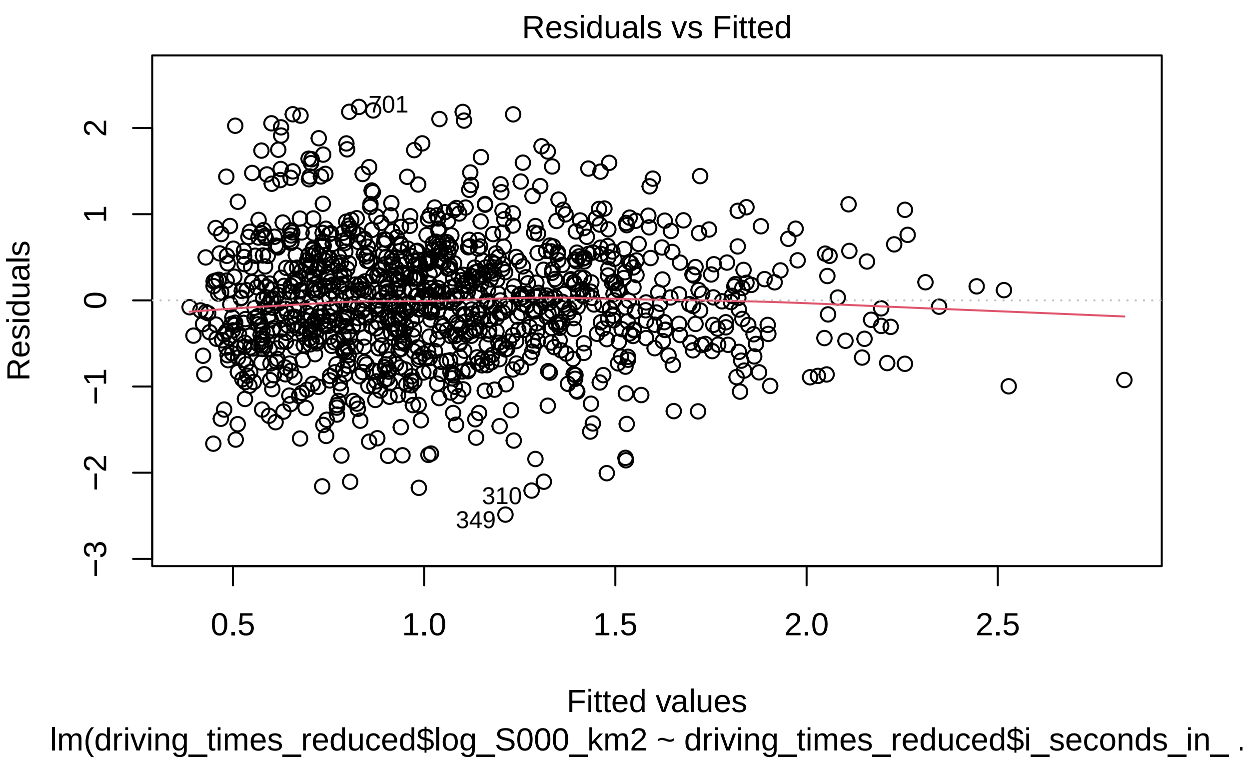
Driving

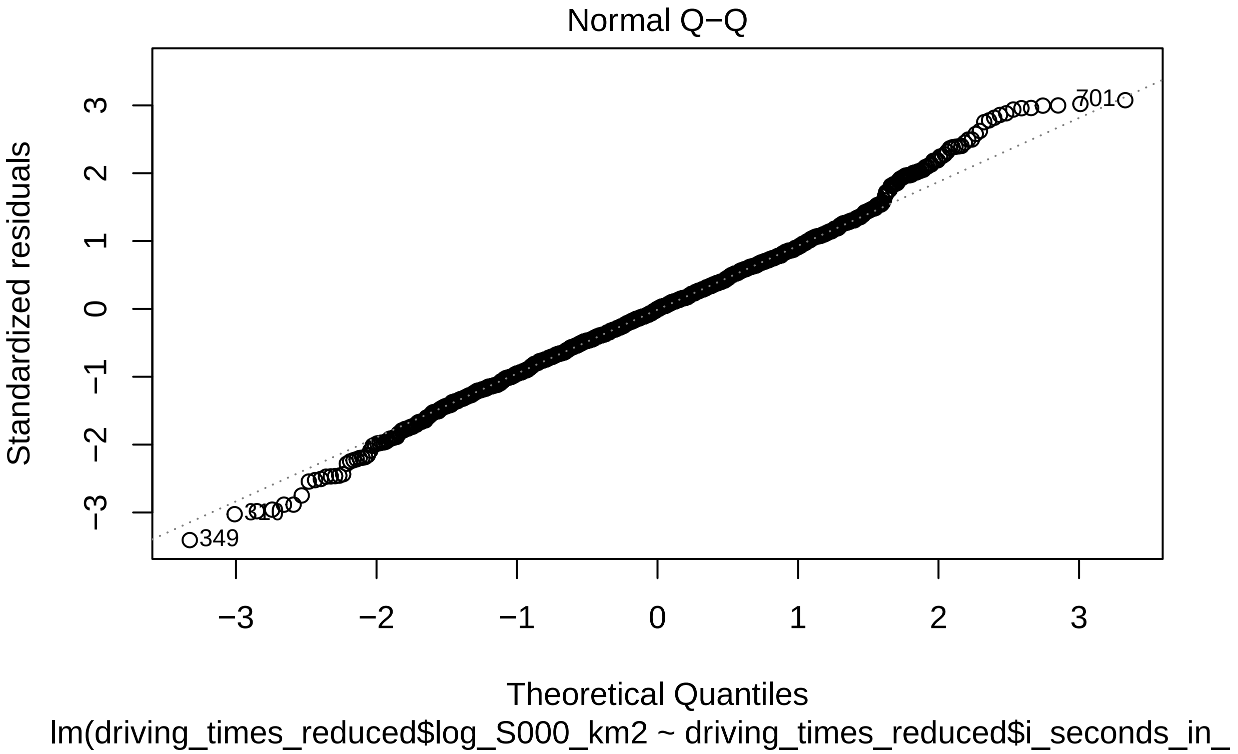
```
driving_model <- lm(driving_times_reduced$log_S000_km2 ~ driving_times_reduced$i_seconds_in_traffic)
summary(driving_model)
```

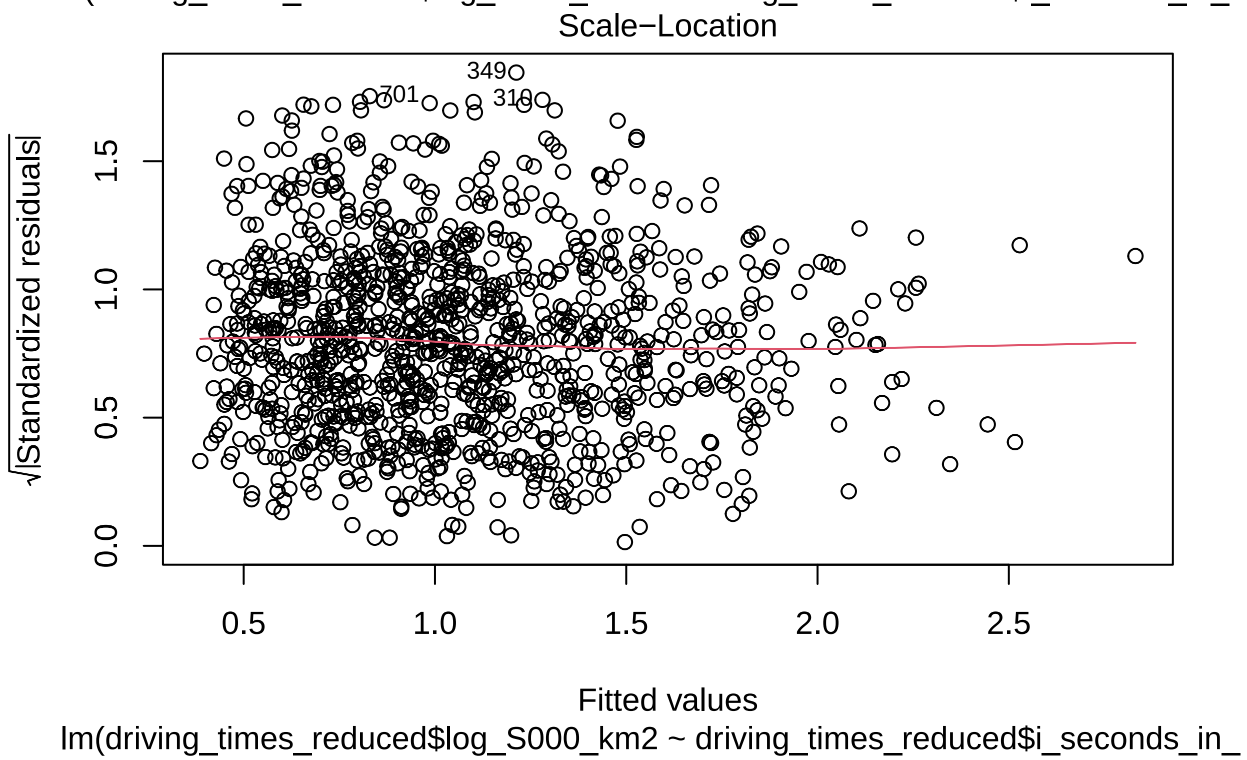
```
##
```

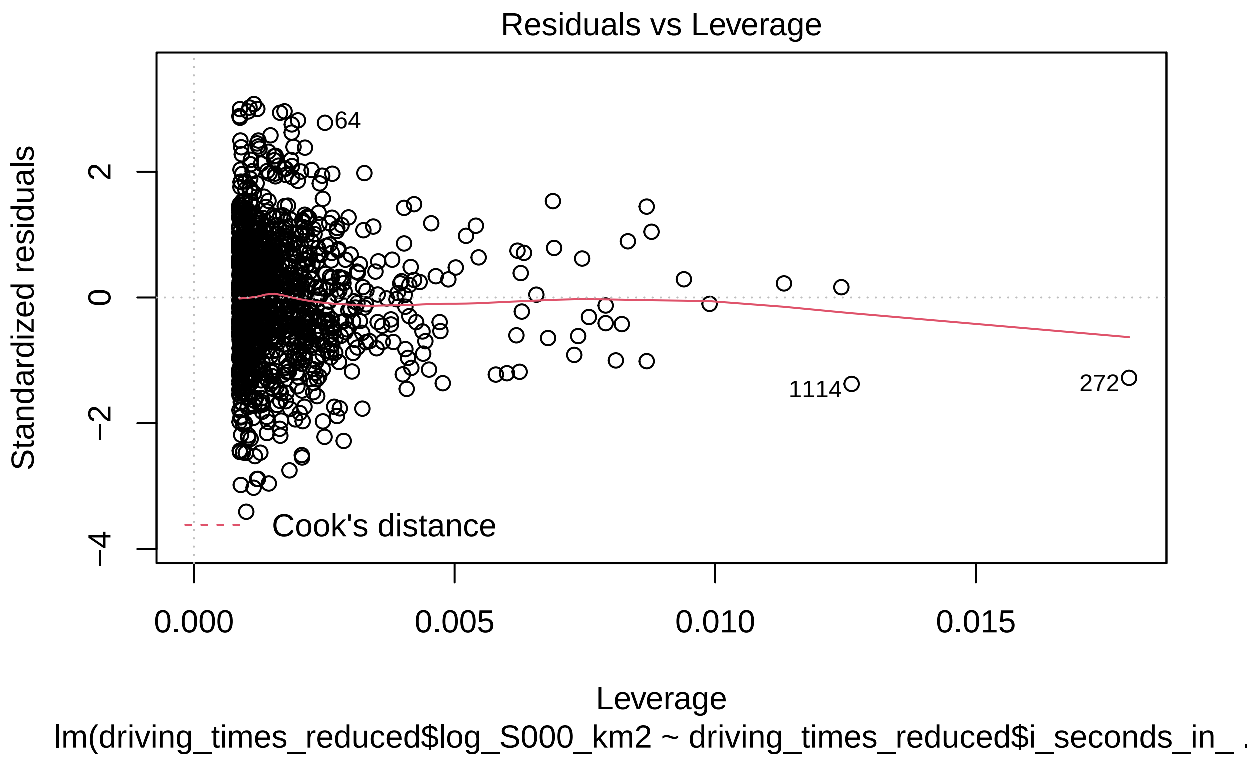
```
## Call:
## lm(formula = driving_times_reduced$log_S000_km2 ~ driving_times_reduced$i_seconds_in_traffic)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.48600 -0.47132 -0.00249  0.45555  2.24441
##
## Coefficients:
##                                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                                 -6.5733     0.4106  -16.01   <2e-16
## driving_times_reduced$i_seconds_in_traffic  19.0288     1.0226   18.61   <2e-16
##
## (Intercept)                                ***
## driving_times_reduced$i_seconds_in_traffic ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.73 on 1146 degrees of freedom
## Multiple R-squared:  0.232,  Adjusted R-squared:  0.2314
## F-statistic: 346.2 on 1 and 1146 DF,  p-value: < 2.2e-16
```

```
plot(driving_model)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(driving_times_reduced$log_S000_km2 ~ driving_times_reduced$i_seconds_in_ .
349
310
701


Scale−Location
√|Standardized residuals|
Fitted values
lm(driving_times_reduced$log_S000_km2 ~ driving_times_reduced$i_seconds_in_ .
349
701
310

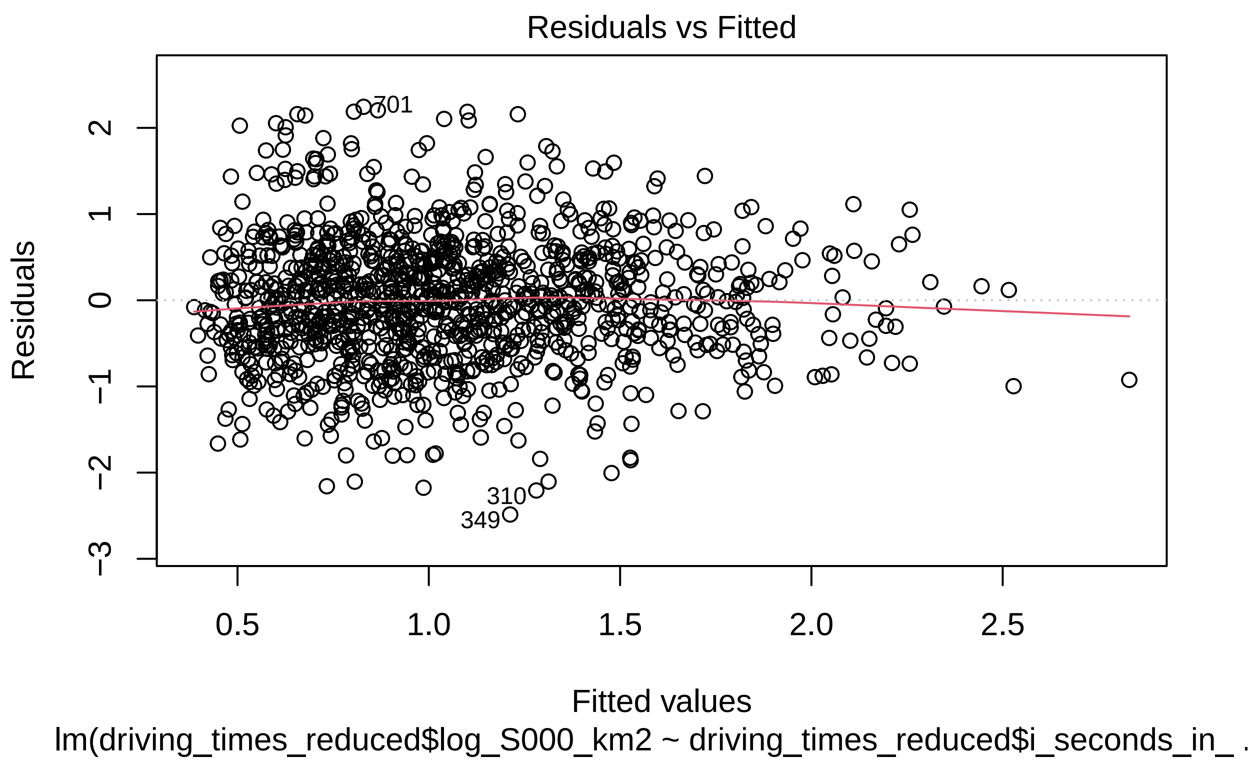


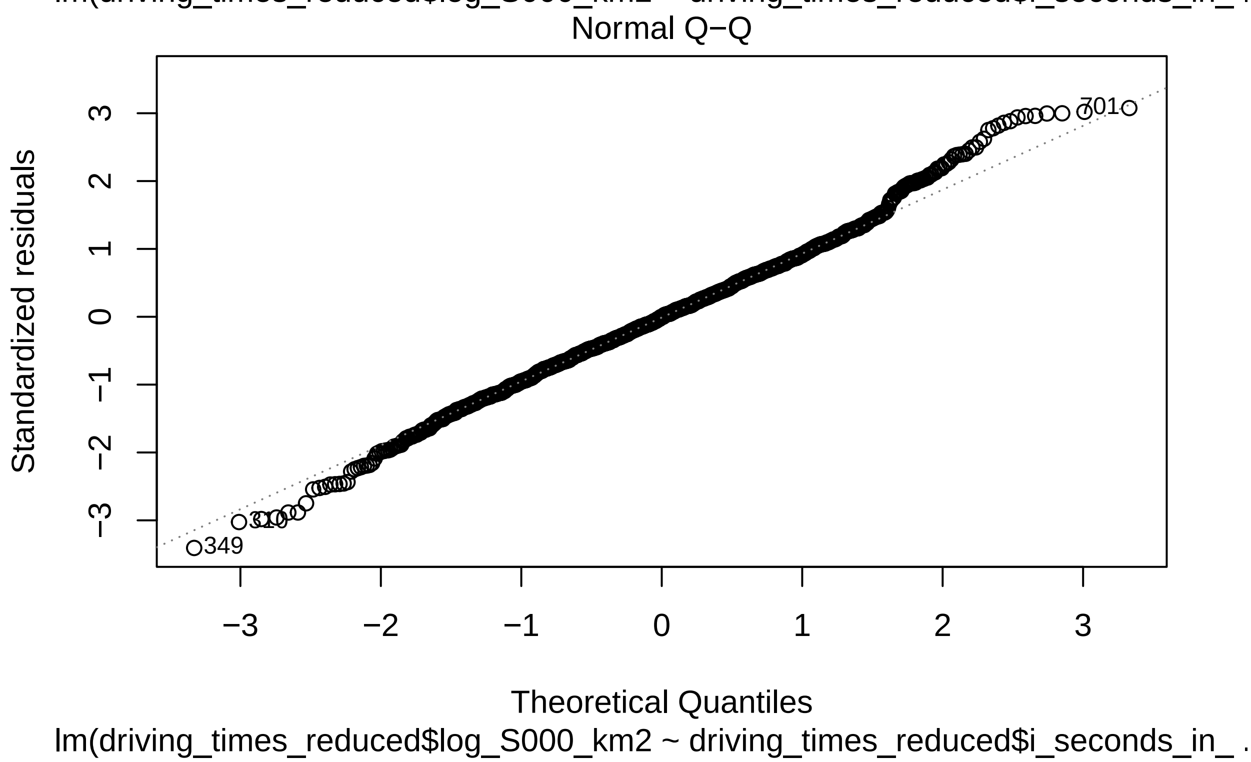
Walking

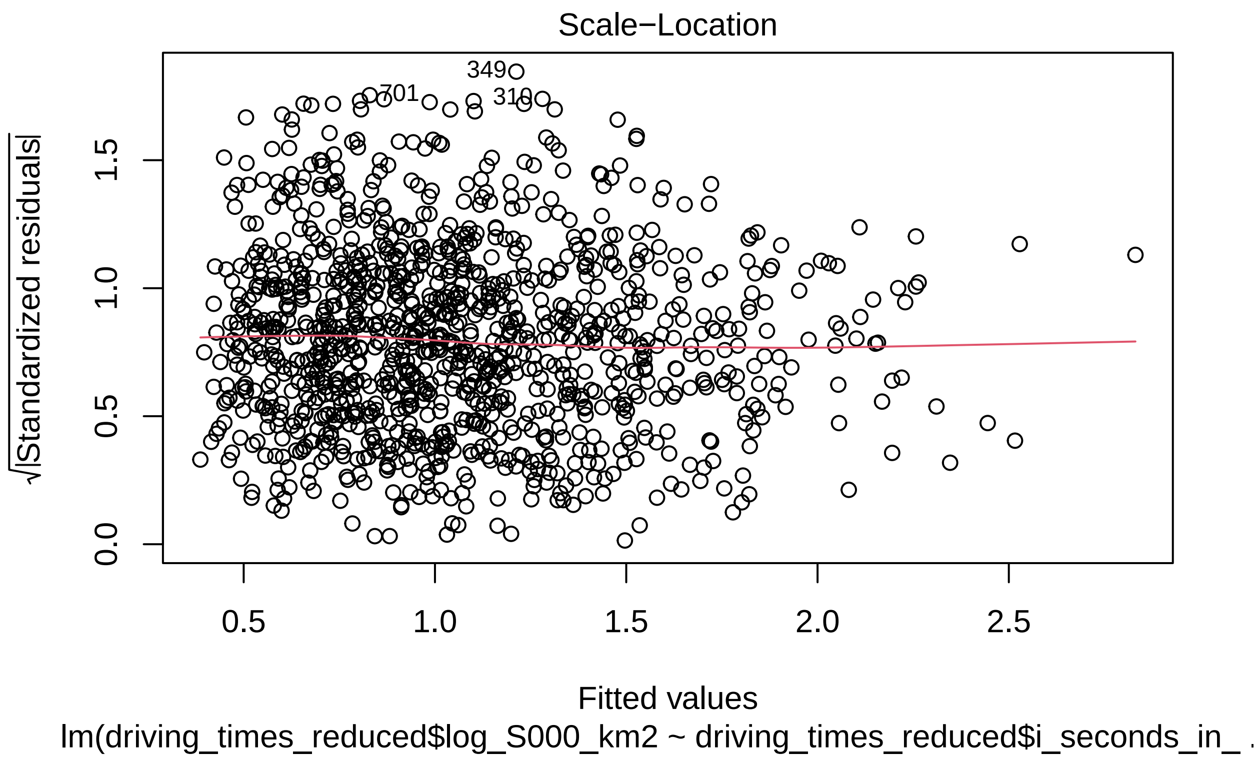
```
walking_model <- lm(walking_times_reduced$log_S000_km2 ~ walking_times_reduced$i_seconds_of_walking)
summary(walking_model)
```

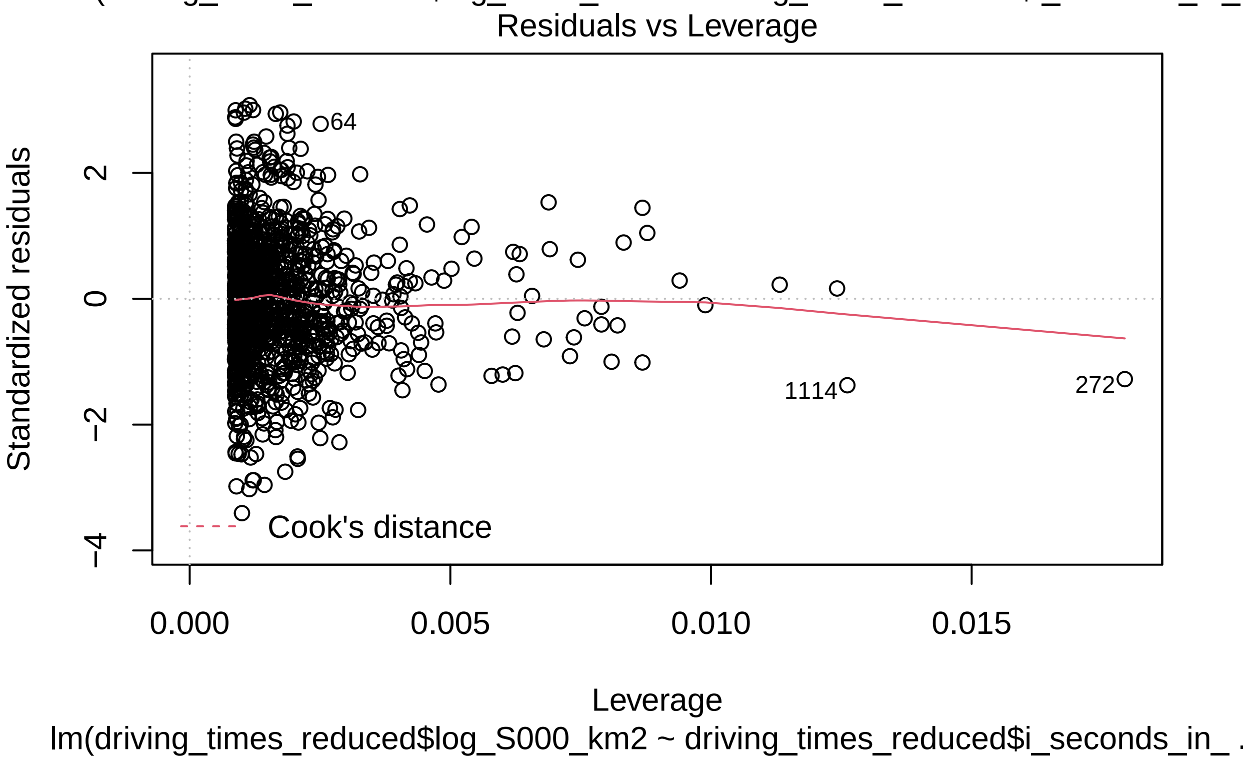
```
##
## Call:
## lm(formula = walking_times_reduced$log_S000_km2 ~ walking_times_reduced$i_seconds_of_walking)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09825 -0.41586 -0.02725  0.40889  2.14433
##
## Coefficients:
##                                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                                 -7.014      0.312  -22.48   <2e-16
## walking_times_reduced$i_seconds_of_walking  23.692      0.914   25.92   <2e-16
##
## (Intercept)                               ***
## walking_times_reduced$i_seconds_of_walking ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6614 on 1146 degrees of freedom
## Multiple R-squared:  0.3696, Adjusted R-squared:  0.3691
## F-statistic: 671.9 on 1 and 1146 DF,  p-value: < 2.2e-16
```

```
plot(driving_model)
```









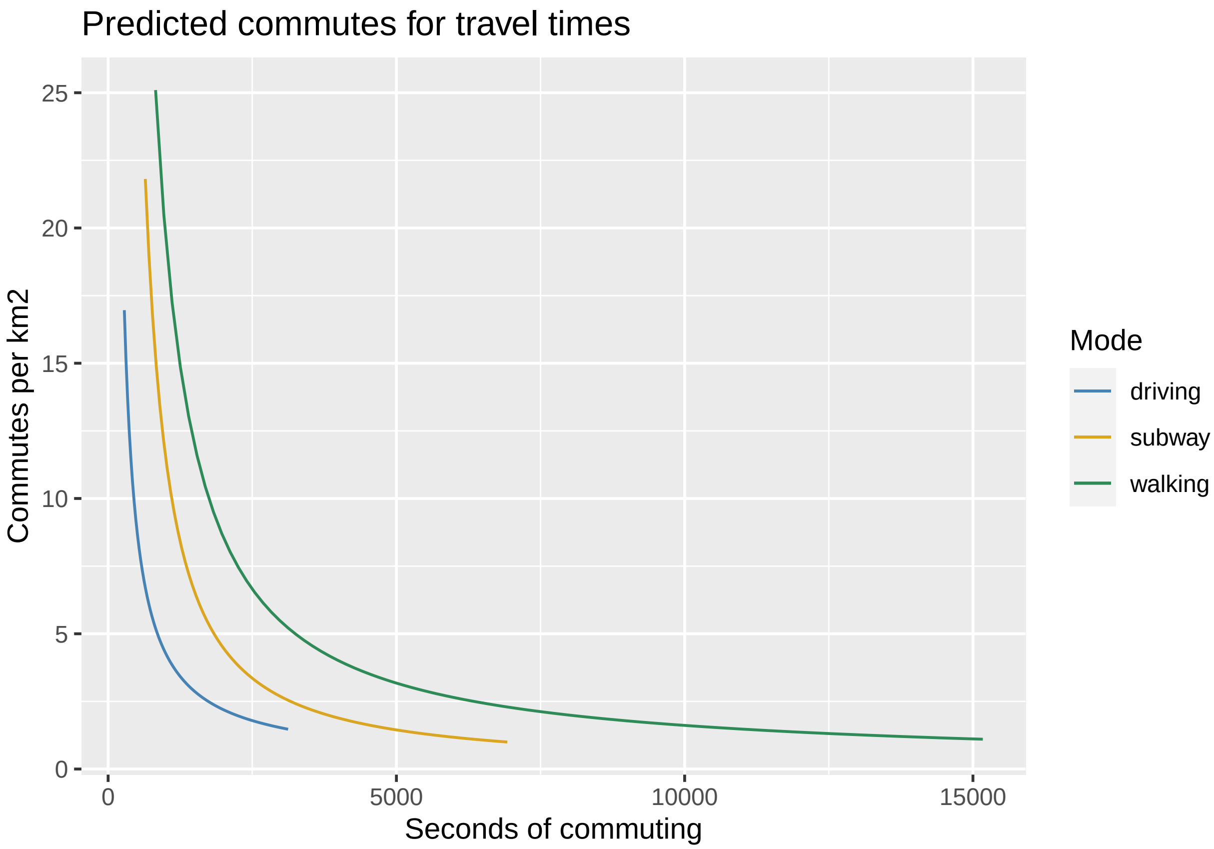
Multiple linear regression for all three factors

Equations plotted for all factors

```
subway_connected_eq <- function(t) exp(subway_connected_model$coefficients[[1]] + subway_connected_model
driving_eq <- function(t) exp(driving_model$coefficients[[1]] + driving_model$coefficients[[2]] / t^(1 /
```

```
walking_eq <- function(t) exp(walking_model$coefficients[[1]] + walking_model$coefficients[[2]] / t^(1 /
```

```
ggplot(
  dplyr::tibble(
    seconds = seq(from = 301, to = 15200, by = 14.9)
  ), aes(seconds)) +
  ggtitle("Predicted commutes for travel times") +
  xlab("Seconds of commuting") +
  ylab("Commutes per km2") +
  stat_function(fun = subway_connected_eq, aes(color = "subway"), xlim = c(min(subway_times_connected$se
  stat_function(fun = driving_eq, aes(color = "driving"), xlim = c(min(driving_times_reduced$seconds_in_
  stat_function(fun = walking_eq, aes(color = "walking"), xlim = c(min(walking_times_reduced$seconds_of_
  scale_color_manual("Mode", values=c("steelblue", "goldenrod", "seagreen"))
```




```
# The 12 minutes is the lowest round minute in each transportation mode's range
# 50 minutes is a reasonable high end of commute times, in each mode's range.

ggplot(
  dplyr::tibble(
    seconds = seq(from = 720, to = 3000, by = 14.9)
  ), aes(seconds)) +
  ggtitle(
    "Predicted commutes for travel times",
    subtitle = "Zoom to commutes between 12 and 50 minutes"
    ) +
  xlab("Seconds of commuting") +
  ylab("Commutes per km2") +
```

```
  stat_function(fun = subway_connected_eq, aes(color = "subway")) +
  stat_function(fun = driving_eq, aes(color = "driving")) +
  stat_function(fun = walking_eq, aes(color = "walking")) +
  scale_color_manual("Mode", values=c("steelblue", "goldenrod", "seagreen"))
```

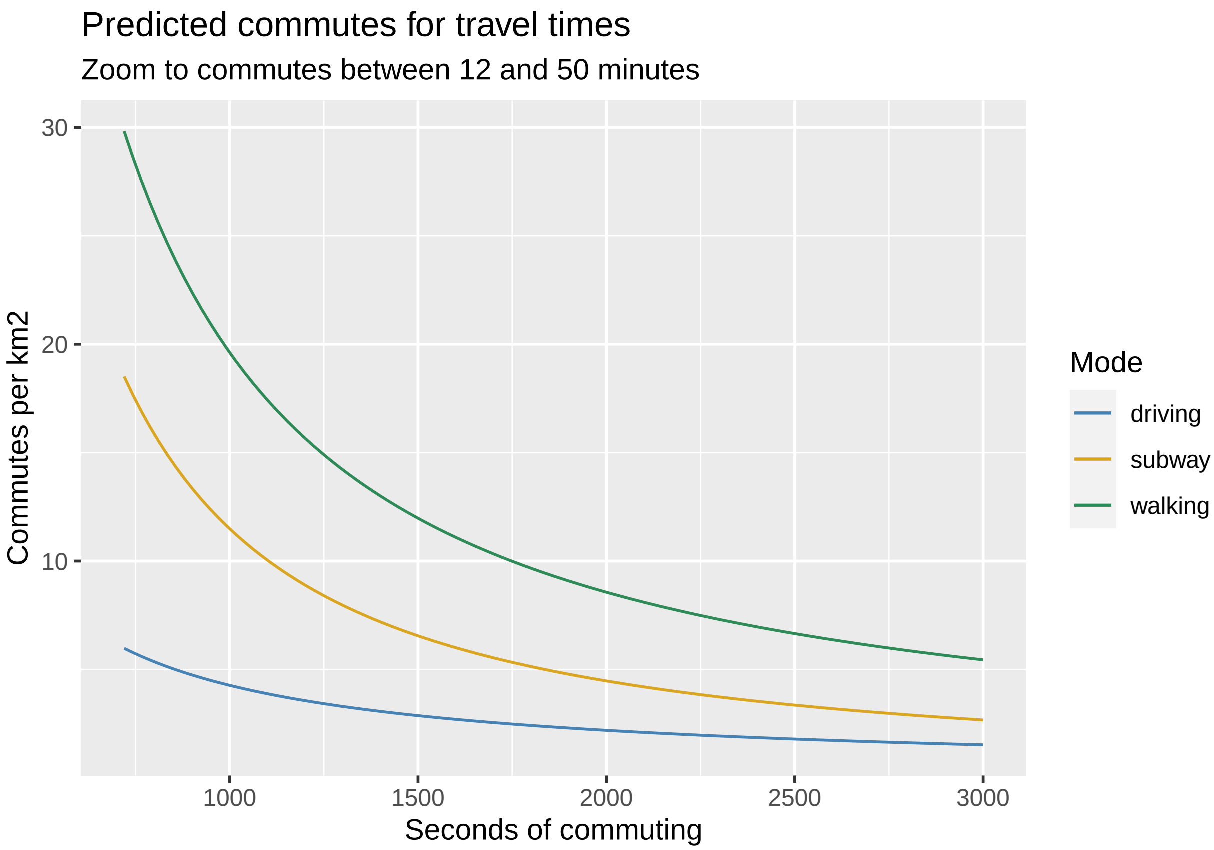


Table of values at 10, 25, 50

```
times_of_interest <- c(10, 25, 50)*60

## subway predictions
subway_predictions <- subway_connected_eq(times_of_interest)
subway_se <- summary(subway_connected_model)$sigma
subway_pred_low <- subway_predictions - (2*subway_se)
subway_pred_high <- subway_predictions + (2*subway_se)

## driving predictions
driving_prediction <- driving_eq(times_of_interest)
driving_se <- summary(driving_model)$sigma
driving_pred_low <- driving_prediction - (2*driving_se)
driving_pred_high <- driving_prediction + (2*driving_se)

## walking predictions
walking_predictions <- walking_eq(times_of_interest)
walking_se <- summary(walking_model)$sigma
walking_pred_low <- walking_predictions - (2*walking_se)
walking_pred_high <- walking_predictions + (2*walking_se)
```

```r
predictions <- dplyr::tibble(
  Minutes = c("Ten", "Twenty Five", "Fifty"),
  subway_predictions,
  subway_pred_low,
  subway_pred_high,
  driving_prediction,
  driving_pred_low,
  driving_pred_high,
  walking_predictions,
  walking_pred_low,
  walking_pred_high
)

print(predictions)
```

```
## # A tibble: 3 x 10
##   Minutes     subway_p~1 subwa~2 subwa~3 drivi~4 drivi~5 drivi~6 walki~7 walki~8
##   <chr>            <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Ten              24.3    23.0    25.7     7.24  5.78      8.70   37.9    36.6
## 2 Twenty Five       6.55    5.21    7.89    2.87  1.41      4.33   12.0    10.7
## 3 Fifty             2.67    1.33    4.01    1.52  0.0642    2.98    5.44    4.12
## # ... with 1 more variable: walking_pred_high <dbl>, and abbreviated variable
## #   names 1: subway_predictions, 2: subway_pred_low, 3: subway_pred_high,
## #   4: driving_prediction, 5: driving_pred_low, 6: driving_pred_high,
## #   7: walking_predictions, 8: walking_pred_low
```

Auto Correlation of Subway, Driving, and Walking

The data points removed for the linear regression are returned for the correlation. For driving and walking, they are returned without further modification. For the subway analysis, any trips that were made without using the subway are giving a value of 0. This reflects that the two NTAs involved in the trip are not actually neighbors through the subway definition.

In addition to the three infrastructure neighborhood definitions, a Queens contiguity analysis is run for reference.

The natural log of job counts per km2 is used as the value for each NTA.

Global Moran's I

```r
subway_times <- subway_times %>%
  dplyr::mutate(
    i_c_seconds_in_transit = ifelse(subway_lines$line_count > 0, i_seconds_in_transit, 0),
  )

subway_graph <- subway_times %>%
  dplyr::select(
    c(
      nta_one,
      nta_two,
      i_c_seconds_in_transit
    )
  ) %>%
  dplyr::rename(
    from = nta_one,
```

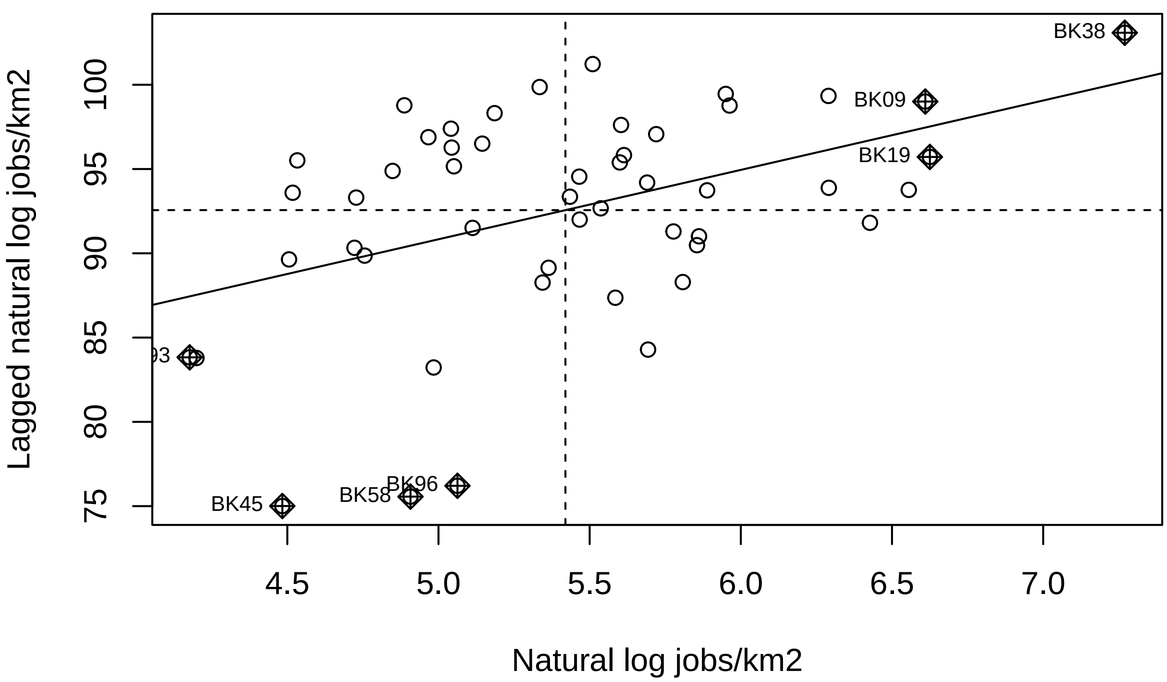
```
    to = nta_two,
    weight = i_c_seconds_in_transit,
  ) %>%
  igraph::graph.data.frame(
    directed = FALSE
  )

subway_weights <- subway_graph %>%
  igraph::as_adjacency_matrix(attr = "weight") %>%
  spdep::mat2listw()

driving_weights <- driving_times %>%
  dplyr::select(
    c(
      nta_one,
      nta_two,
      i_seconds_in_traffic
    )
  ) %>%
  dplyr::rename(
    from = nta_one,
    to = nta_two,
    weight = i_seconds_in_traffic
  ) %>%
  igraph::graph.data.frame(
    directed = FALSE
  ) %>%
  igraph::as_adjacency_matrix(attr = "weight") %>%
  spdep::mat2listw()

walking_weights <- walking_times %>%
  dplyr::select(
    c(
      nta_one,
      nta_two,
      i_seconds_of_walking
    )
  ) %>%
  dplyr::rename(
    from = nta_one,
    to = nta_two,
    weight = i_seconds_of_walking
  ) %>%
  igraph::graph.data.frame(
    directed = FALSE
  ) %>%
  igraph::as_adjacency_matrix(attr = "weight") %>%
  spdep::mat2listw()

queen_weights <- job_counts %>%
  spdep::poly2nb(c("w_nta_code")) %>%
  spdep::nb2listw(zero.policy = TRUE)
```

```
subway_global_morans <- spdep::moran.test(
  job_counts$log_S000_km2,
  subway_weights,
  zero.policy = TRUE,
)
print(subway_global_morans)
```

Subway

```
##
##  Moran I test under randomisation
##
## data:  job_counts$log_S000_km2
## weights: subway_weights
##
## Moran I statistic standard deviate = -3.5321, p-value = 0.9998
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##      -0.044946546      -0.020408163       0.000048265
```

```
spdep::moran.plot(
  job_counts$log_S000_km2,
  subway_weights,
  zero.policy = TRUE,
  xlab = "Natural log jobs/km2",
  ylab = "Lagged natural log jobs/km2"
)
```



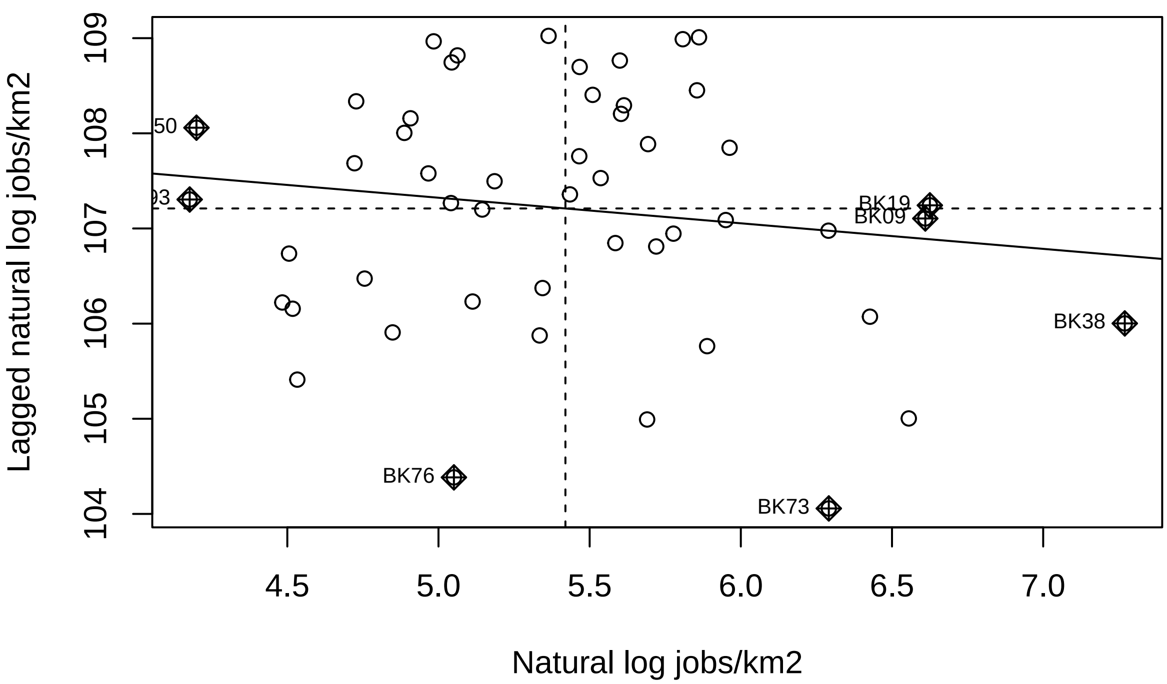
Driving

```
driving_global_morans <- spdep::moran.test(
  job_counts$log_S000_km2,
```

```r
  driving_weights,
  zero.policy = TRUE,
)
print(driving_global_morans)
```

```
##
##  Moran I test under randomisation
##
## data:  job_counts$log_S000_km2
## weights: driving_weights
##
## Moran I statistic standard deviate = 6.6184, p-value = 1.815e-11
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##     -9.846546e-03     -2.040816e-02      2.546533e-06
```

```r
spdep::moran.plot(
  job_counts$log_S000_km2,
  driving_weights,
  zero.policy = TRUE,
  xlab = "Natural log jobs/km2",
  ylab = "Lagged natural log jobs/km2"
)
```

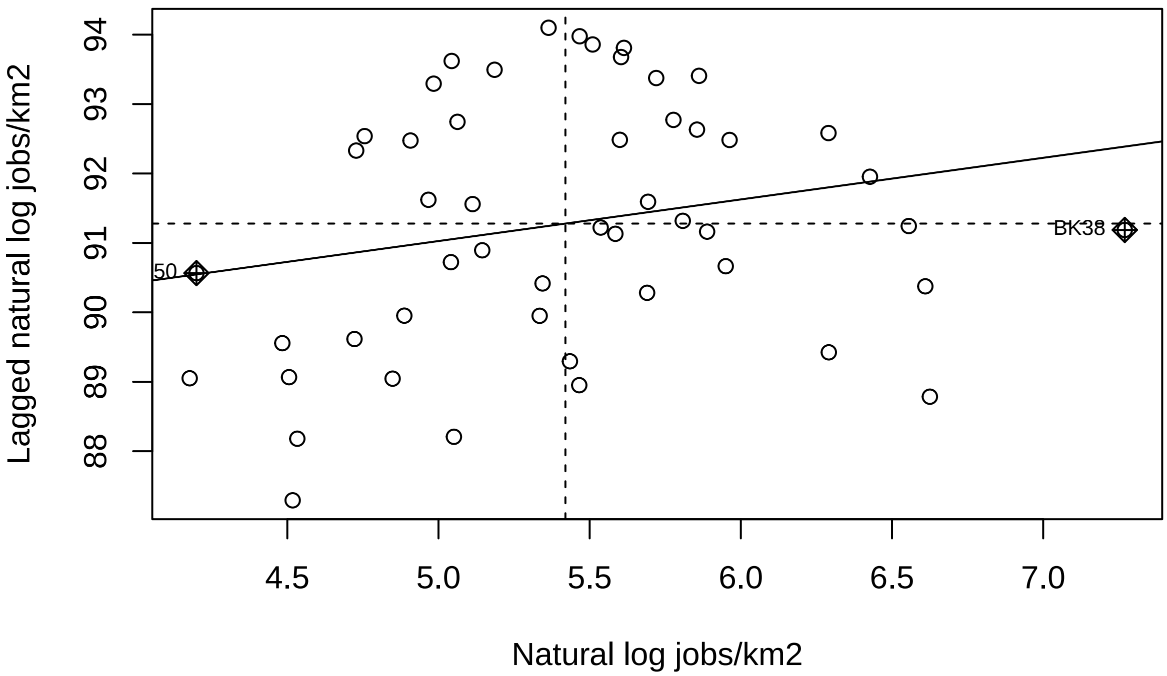



```r
walking_global_morans <- spdep::moran.test(
  job_counts$log_S000_km2,
  walking_weights,
  zero.policy = TRUE,
)
print(walking_global_morans)
```

Walking

```
##
##  Moran I test under randomisation
##
## data:  job_counts$log_S000_km2
## weights: walking_weights
##
## Moran I statistic standard deviate = 6.2078, p-value = 2.687e-10
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##     -8.981503e-03     -2.040816e-02      3.388162e-06
```

```
spdep::moran.plot(
  job_counts$log_S000_km2,
  walking_weights,
  zero.policy = TRUE,
  xlab = "Natural log jobs/km2",
  ylab = "Lagged natural log jobs/km2"
)
```



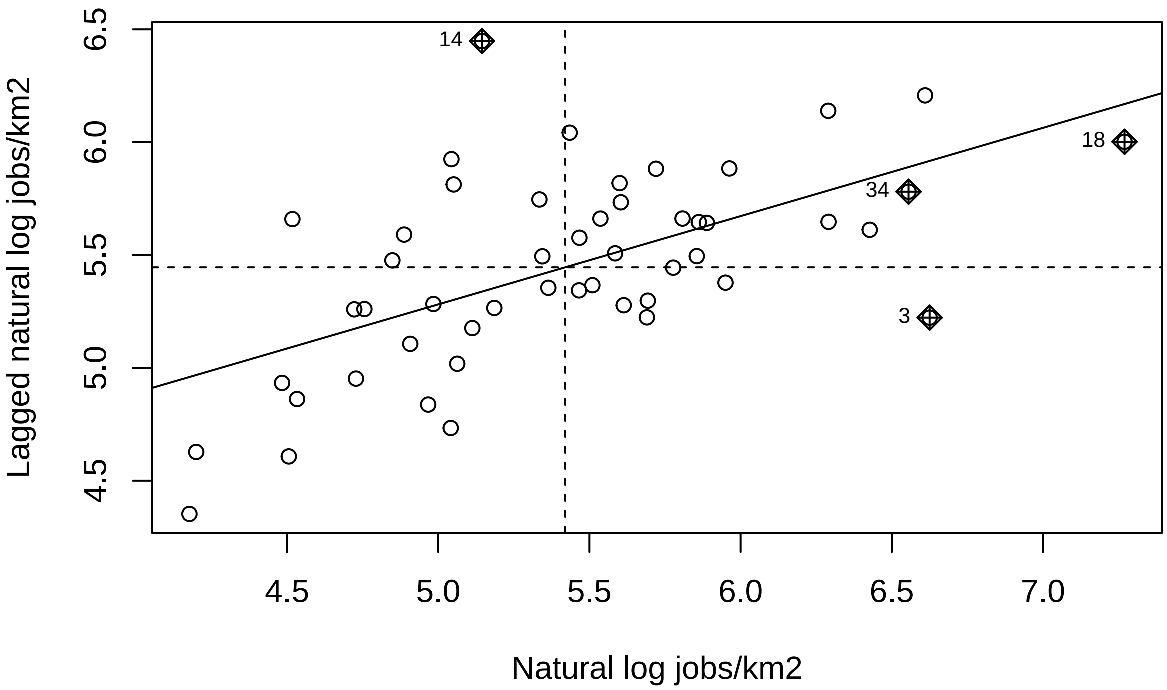
Queens

```
queen_global_morans <- spdep::moran.test(
  job_counts$log_S000_km2,
  queen_weights,
  zero.policy = TRUE,
)
print(queen_global_morans)
```

```
##
##  Moran I test under randomisation
##
## data:  job_counts$log_S000_km2
```

```
## weights: queen_weights
##
## Moran I statistic standard deviate = 4.4715, p-value = 3.884e-06
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##       0.391099443      -0.020408163       0.008469485
```

```
spdep::moran.plot(
  job_counts$log_S000_km2,
  queen_weights,
  zero.policy = TRUE,
  xlab = "Natural log jobs/km2",
  ylab = "Lagged natural log jobs/km2"
)
```



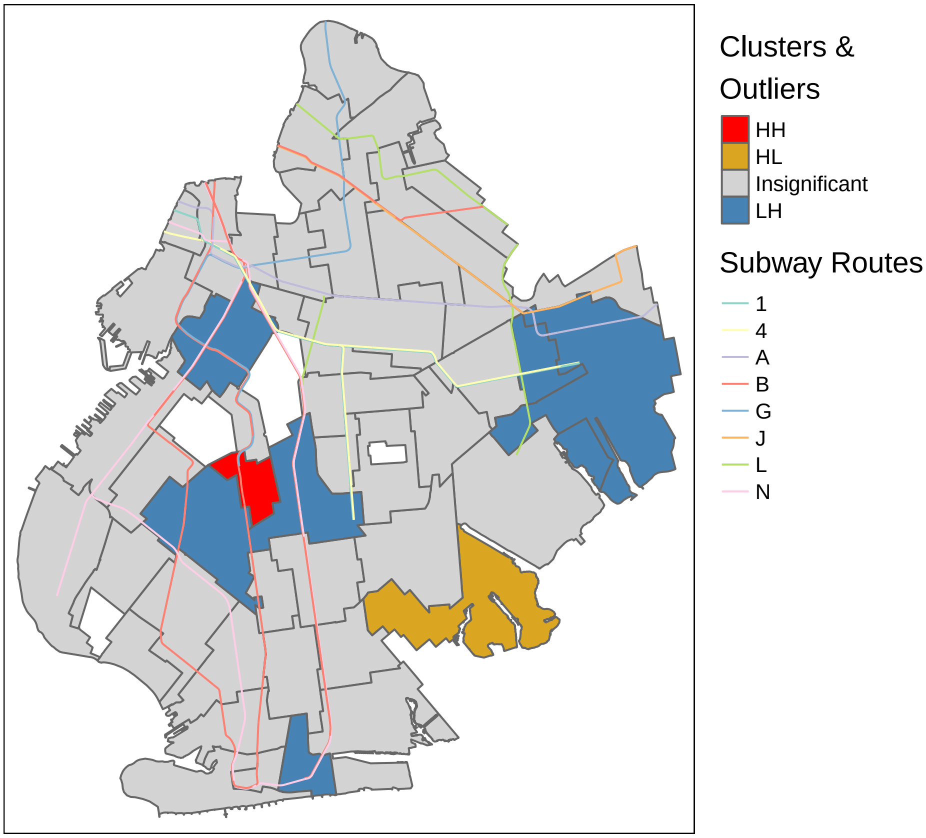
LISA

```
avg_jobs <- mean(job_counts$log_S000_km2)

classify_co_types <- function(mode_lisa, l_job_counts, avg_job_count) {
  mode_lisa %>%
    tibble::as_tibble() %>%
    magrittr::set_colnames(
      c("Ii","E.Ii","Var.Ii","Z.Ii","Pr(z > 0)")
    ) %>%
    dplyr::mutate(
      co_type = dplyr::case_when(
        `Pr(z > 0)` <= 0.05 &
          Ii >= 0 &
          l_job_counts >= avg_job_count ~ "HH",
        `Pr(z > 0)` <= 0.05 &
```

```r
          Ii >= 0 &
          l_job_counts <  avg_job_count ~ "LL",
        `Pr(z > 0)` <= 0.05 &
          Ii < 0 &
          l_job_counts >= avg_job_count ~ "HL",
        `Pr(z > 0)` <= 0.05 &
          Ii < 0 &
          l_job_counts < avg_job_count ~ "LH"
      )
    )
}

subway_lisa <- spdep::localmoran(
  job_counts$log_S000_km2,
  subway_weights,
  zero.policy = TRUE,
  na.action = na.omit
)

driving_lisa <- spdep::localmoran(
  job_counts$log_S000_km2,
  driving_weights,
  zero.policy = TRUE,
  na.action = na.omit
)

walking_lisa <- spdep::localmoran(
  job_counts$log_S000_km2,
  walking_weights,
  zero.policy = TRUE,
  na.action = na.omit
)

queen_lisa <- spdep::localmoran(
  job_counts$log_S000_km2,
  queen_weights,
  zero.policy = TRUE,
  na.action = na.omit
)

subway_classes <- classify_co_types(subway_lisa, job_counts$log_S000_km2, avg_jobs)
driving_classes <- classify_co_types(driving_lisa, job_counts$log_S000_km2, avg_jobs)
walking_classes <- classify_co_types(walking_lisa, job_counts$log_S000_km2, avg_jobs)
queen_classes <- classify_co_types(queen_lisa, job_counts$log_S000_km2, avg_jobs)

subway_bk_nta_border <- bk_nta_border %>%
  dplyr::mutate(
    co_type = ifelse(is.na(subway_classes$co_type), "Insignificant", subway_classes$co_type)
  )

driving_bk_nta_border <- bk_nta_border %>%
  dplyr::mutate(
    co_type = ifelse(is.na(driving_classes$co_type), "Insignificant",  driving_classes$co_type)
```

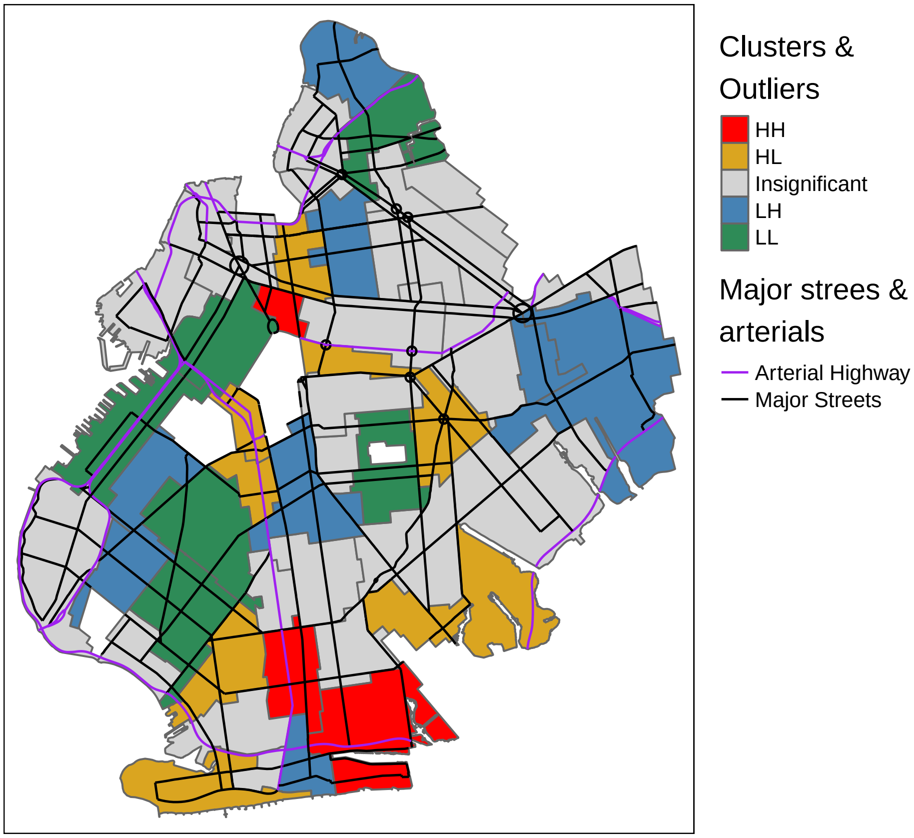
```
  )

walking_bk_nta_border <- bk_nta_border %>%
  dplyr::mutate(
    co_type = ifelse(is.na(walking_classes$co_type), "Insignificant", walking_classes$co_type)
  )

queen_bk_nta_border <- bk_nta_border %>%
  dplyr::mutate(
    co_type = ifelse(is.na(queen_classes$co_type), "Insignificant", queen_classes$co_type)
  )
```

```
tmap::tm_shape(subway_bk_nta_border) +
  tm_polygons(
    col = "co_type",
    palette = c("red", "goldenrod", "lightgrey", "steelblue"),
    title = "Clusters &\nOutliers"
  ) +
  tmap::tm_shape(bk_subways) +
  tmap::tm_lines(
    col = "rt_symbol",
    title.col = "Subway Routes"
  ) +
  tmap::tm_layout(
    legend.outside = TRUE
  )
```



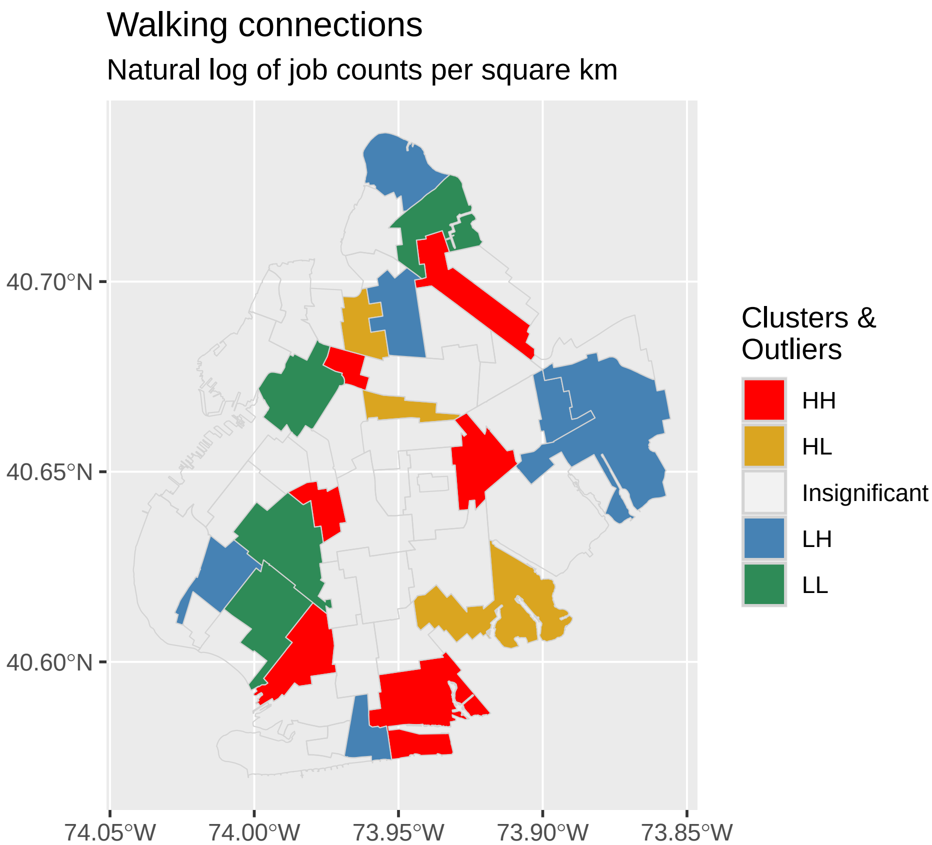
Subway #### Driving

```
tmap::tm_shape(driving_bk_nta_border) +
  tm_polygons(
    col = "co_type",
    palette = c("red", "goldenrod", "lightgrey", "steelblue", "seagreen"),
    title = "Clusters &\nOutliers"
  ) +
  tmap::tm_shape(bk_major_roads) +
  tmap::tm_lines(
    col = "route_type",
    lwd = 1.2,
    palette = c("purple", "black"),
    title.col = "Major strees &\narterials"
  ) +
  tmap::tm_layout(
    legend.outside = TRUE
  )
```



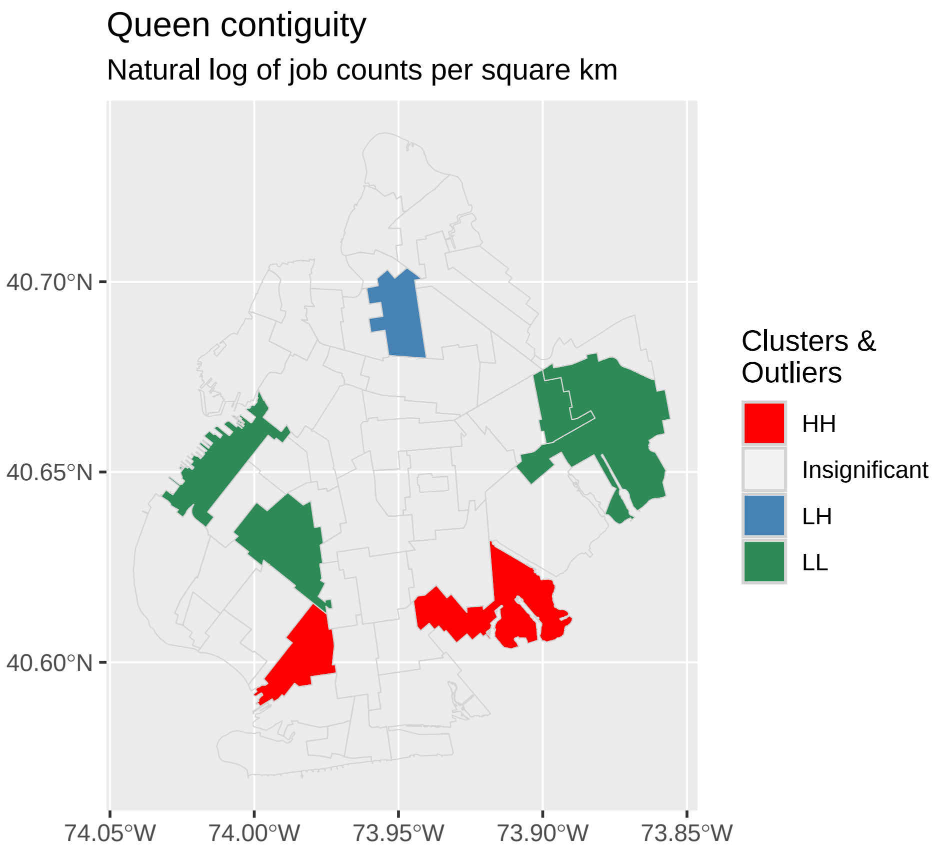
Walking

```
ggplot(walking_bk_nta_border) +
  geom_sf(aes(fill = co_type), col = 'lightgrey') +
  scale_fill_manual(
    values = c("red", "goldenrod", "NA", "steelblue", "seagreen"),
    name = "Clusters & \nOutliers"
  ) +
  labs(
    title = "Walking connections",
    subtitle = "Natural log of job counts per square km"
  )
```



Queen Contiguity

```
ggplot(queen_bk_nta_border) +
  geom_sf(aes(fill = co_type), col = 'lightgrey') +
  scale_fill_manual(
    values = c("red", "NA", "steelblue", "seagreen"),
    name = "Clusters & \nOutliers"
  ) +
  labs(
    title = "Queen contiguity",
    subtitle = "Natural log of job counts per square km"
  )
```



Network autocorrelation

```
## Utility function that builds the edges between trips
build_edges <- function(nodes) {
  edges_from <- vector()
  edges_to <- vector()
  nodes_count <- length(nodes)
  for(i in 1:(nodes_count - 1)) {
    offset <- i + 1
    from_node <- nodes[i]
    from_nta_one <- stringr::str_sub(from_node, 1, 4)
    from_nta_two <- stringr::str_sub(from_node, 5, 8)
    for(j in offset:nodes_count){
      to_node <- nodes[j]
      are_neighbors <- stringr::str_detect(to_node, from_nta_one) | stringr::str_detect(to_node, from_n
      if (are_neighbors) {
        edges_from <- append(edges_from, from_node)
        edges_to <- append(edges_to, to_node)
      }
    }
  }
  return (tibble::tibble(from = edges_from, to = edges_to))
}
```

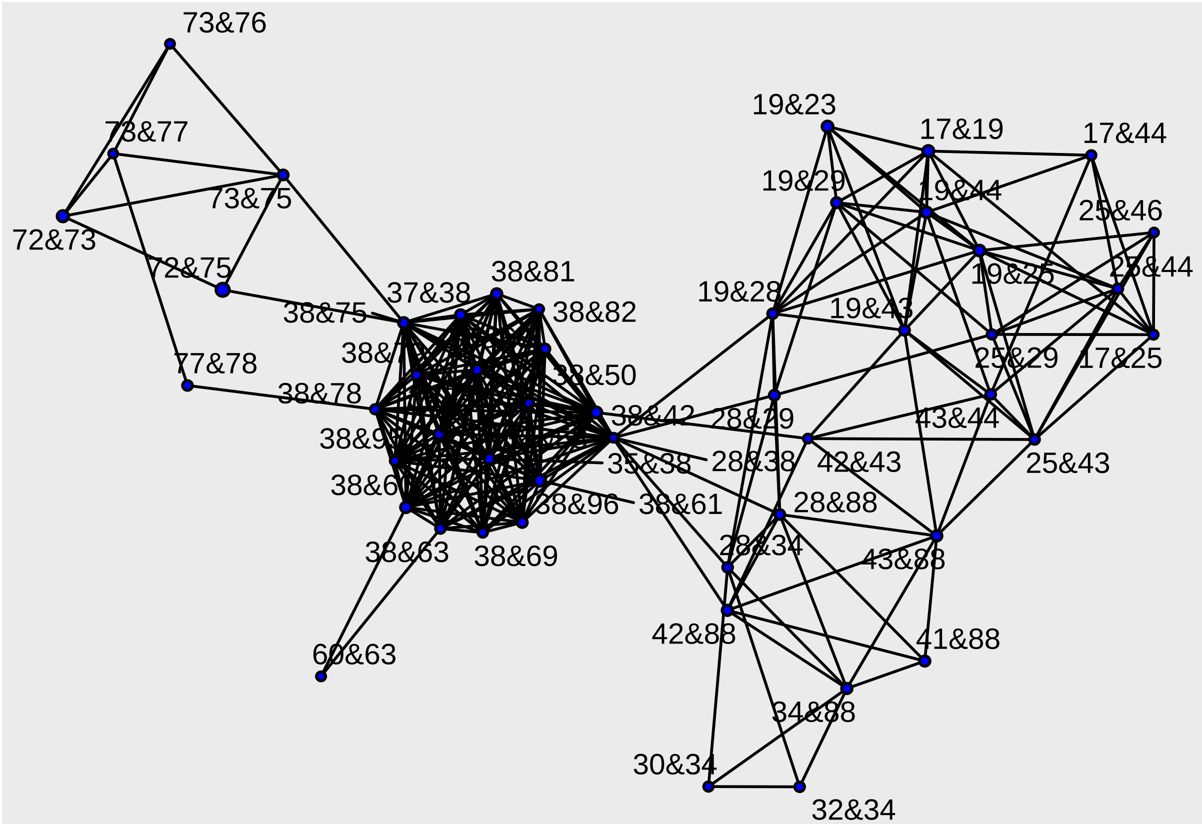
```
commute_nodes <- commute_counts %>%
  dplyr::select(
    c(trip, log_S000_km2)
```

```
  ) %>%
  dplyr::rename(
    name = trip
  )
commute_edges <- build_edges(commute_nodes$name)
commute_network <- tidygraph::tbl_graph(
  nodes = commute_nodes,
  edges = commute_edges,
  directed = FALSE
)
```

```
commute_nodes_top <- commute_nodes %>%
  dplyr::slice_max(
    order_by = log_S000_km2,
    n = 50
  )

commute_edges_top <- build_edges(commute_nodes_top$name)
commute_network_top <- tidygraph::tbl_graph(
  nodes = commute_nodes_top,
  edges = commute_edges_top,
  directed = FALSE
)

total_trips_top = sum(commute_nodes_top$log_S000_km2)
ggraph::ggraph(
  commute_network_top,
  layout = "stress"
) +
  geom_edge_link() +
  geom_node_circle(aes(r = commute_nodes_top$log_S000_km2 / 100), fill = "blue") +
  geom_node_text(aes(label = stringr::str_c(stringr::str_sub(name, 3,4), '&', stringr::str_sub(name, 7,8
```

Visualization of network's complement and most popular trips

```
## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` in the `default_aes` field and elsewhere instead.
```

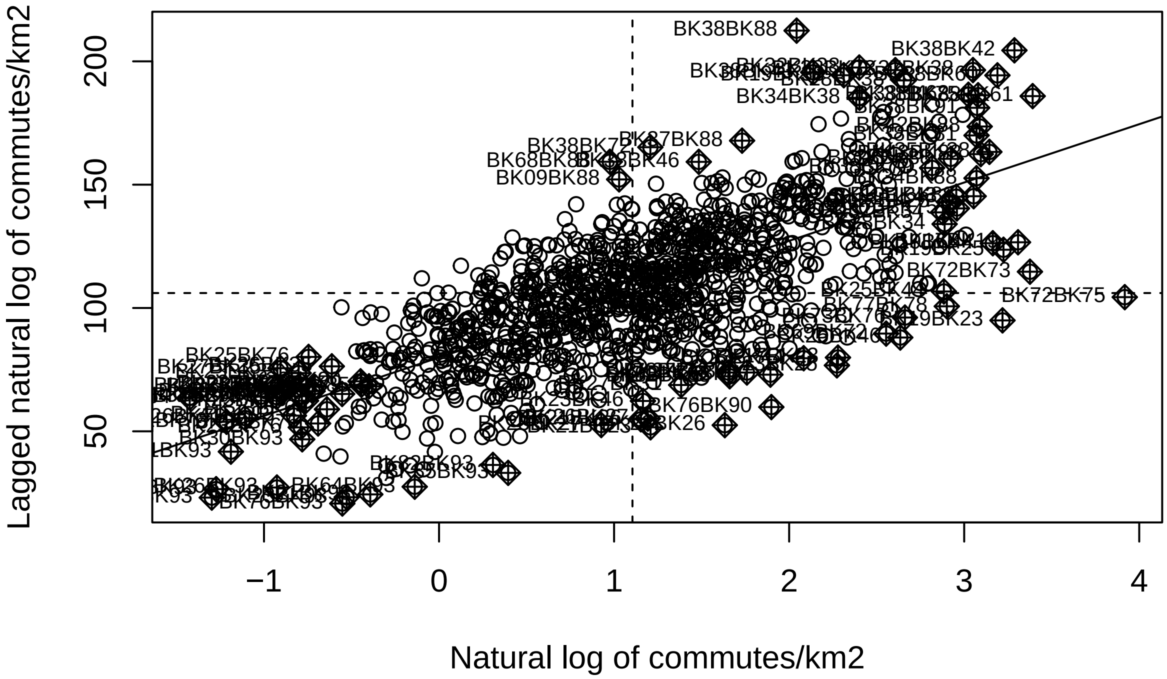



```
commute_weights <- commute_network %>%
  igraph::as_adj() %>%
  spdep::mat2listw()

commute_global_morans <- spdep::moran.test(
  commute_nodes$log_S000_km2,
  commute_weights,
  zero.policy = TRUE
)

print(commute_global_morans)
```

Global Moran's I

```
##
##  Moran I test under randomisation
##
## data:  commute_nodes$log_S000_km2
## weights: commute_weights
##
## Moran I statistic standard deviate = 62.413, p-value < 2.2e-16
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##      2.462778e-01     -8.169935e-04      1.567388e-05
```

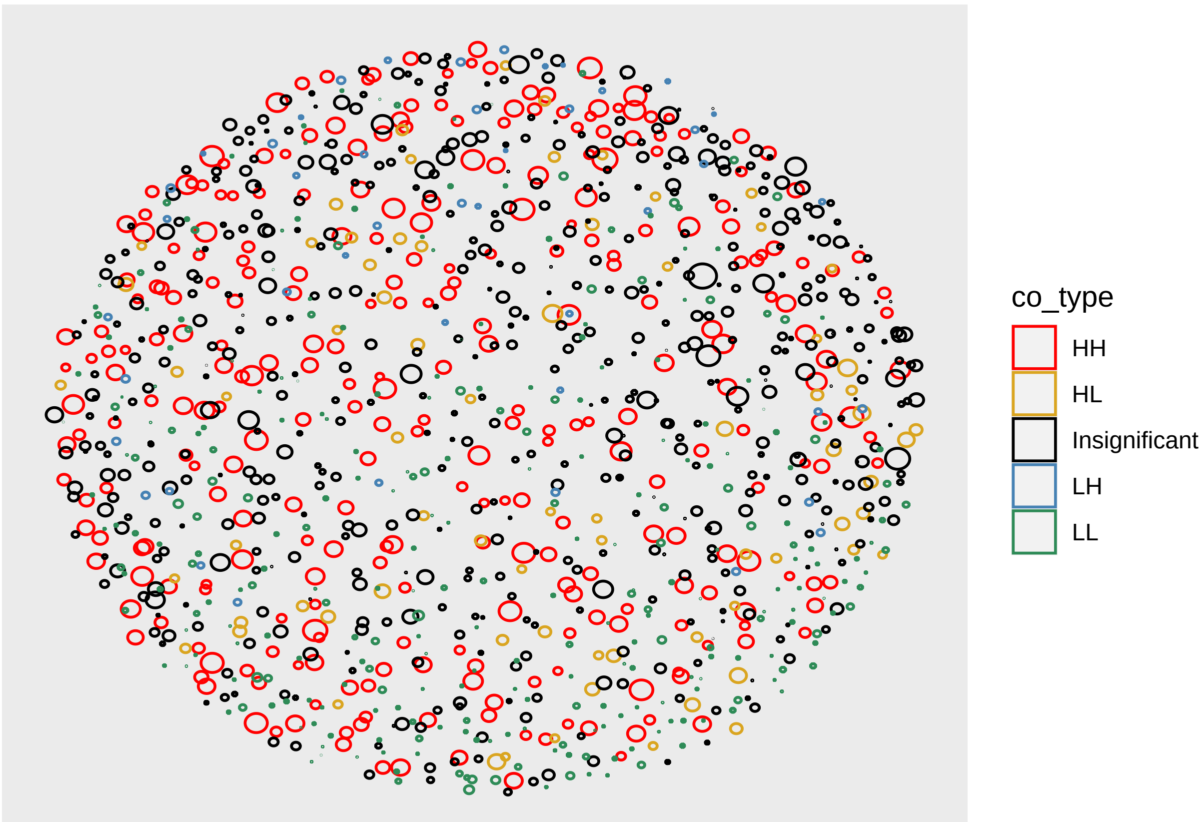
```
spdep::moran.plot(
  commute_nodes$log_S000_km2,
  commute_weights,
  zero.policy = TRUE,
  xlab = "Natural log of commutes/km2",
  ylab = "Lagged natural log of commutes/km2"
)
```




```
commute_lisa <- spdep::localmoran(
  commute_nodes$log_S000_km2,
  commute_weights,
  zero.policy = TRUE,
  na.action = na.omit
)

avg_commute_count <- mean(commute_nodes$log_S000_km2)

commute_classes <- classify_co_types(commute_lisa, commute_nodes$log_S000_km2, avg_commute_count)

commute_network_cluster <- commute_network %>%
  tidygraph::activate(nodes) %>%
  dplyr::mutate(
    co_type  = commute_classes$co_type %>% tidyr::replace_na("Insignificant")
  )

commute_network_cluster_sig <- commute_network_cluster %>%
  dplyr::filter(co_type != "Insignificant")
```

```
ggraph::ggraph(
  commute_network_cluster,
```

```
  layout = "stress"
) +
  ggraph::geom_node_circle(
    aes(r = log_S000_km2 / 75, color = co_type)
    ) +
  scale_color_manual(values = c("red", "goldenrod", "black", "steelblue", "seagreen"))
```



LISA

```
commute_count_lines <- commute_counts %>%
  dplyr::mutate(
    center_one = sf::st_point_on_surface(geometry_one),
    center_two = sf::st_point_on_surface(geometry_two)
  ) %>%
  dplyr::mutate(
    geometry = sf::st_cast(sf::st_union(center_one, center_two), "LINESTRING")
  ) %>%
  dplyr::select(
    c(trip, log_S000_km2, geometry)
  ) %>%
  sf::st_as_sf() %>%
  dplyr::mutate(
    co_type  = commute_classes$co_type %>% tidyr::replace_na("Insignificant")
  )
```

```
## Warning in st_point_on_surface.sfc(geometry_one): st_point_on_surface may not
## give correct results for longitude/latitude data

## Warning in st_point_on_surface.sfc(geometry_two): st_point_on_surface may not
## give correct results for longitude/latitude data
```

```r
tmap::tm_shape(bk_nta_border) +
  tmap::tm_polygons(
    col = "#eeeeee"
  ) + tmap::tm_shape(commute_count_lines) +
  tmap::tm_lines(
    col = "co_type",
    palette = c("red", "goldenrod", "white", "steelblue", "seagreen")
  )
```