CSC 495 Final Project ExploreBipartiteSplitNeighborhoods.Rmd

Kari Palmier

6/2/2018, Spring 2018

```
library(knitr)
setwd("C:\\DePaulCoursework\\Spring CSC 495\\Project\\")
read_chunk("ChicagoCrime_ExploreBipartiteSplitNeighborhoods.R")
knitr::opts_chunk$set(echo = TRUE)
```

Description

The file performs analysis on the bipartite neighborhood projections of the violent and non-violent Feb 2017 Chicago Crime datasets. The first step is to create the bipartite projections of both the violent and non-violent graph data. Next the projections were filtered so that edge weights below given thresholds were removed. After that any singleton nodes were removed. Finally the giant component of the final networks were extracted. Weighted degree, assortativity, transitivity, and modularity analyses were then performed on the giant components of the both networks.

Load the libraries and data

```
library("igraph")
library("ggplot2")
library("GGally")
# Must load other packages first
library("sand")
library("intergraph")
```

```
base_path = "C:\\DePaulCoursework\\Spring CSC 495\\Project\\"
source(paste(base_path, "mycugtest.R", sep=""))
source(paste(base_path, "myqaptest.R", sep=""))

dir.create(file.path(base_path, "R_Output"), showWarnings = FALSE)
base_out_path = paste(base_path, "R_Output\\", sep = "")
dir.create(file.path(base_out_path, "Neighborhoods"), showWarnings = FALSE)
output_path = paste(base_out_path, "Neighborhoods\\", sep = "")

path = paste(base_path, "Graph_Data\\", sep = "")
setwd(path)

violent_gr = read.graph("feb_violent.graphml", format = "graphml")

nonviolent_gr = read.graph("feb_nonviolent.graphml", format = "graphml")

save_to_file = FALSE
```

Summarize initial data

```
if (save_to_file){
  out_file_name = paste(output_path, "Graph_Summaries.txt", sep = '')
  outFile = file(out_file_name, open="wt")
  sink(file = outFile, append = TRUE)
}

# Summarize data
print("Original Violent Graph Summary:")
```

```
## [1] "Original Violent Graph Summary:"
```

```
print(summary(violent_gr))
```

```
## IGRAPH 95fd799 UN-B 91 6356 --
## + attr: name (v/c), Label (v/c), type (v/l), Prop.Occupied (v/n),
## | Prop.Rented (v/n), Prop.Vacant (v/n), Prop.Owned (v/n),
## | Median.Age (v/n), Total.Population (v/n), Average.Household.Size
## | (v/n), Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c)
## IGRAPH 95fd799 UN-B 91 6356 --
## + attr: name (v/c), Label (v/c), type (v/l), Prop.Occupied (v/n),
## | Prop.Rented (v/n), Prop.Vacant (v/n), Prop.Owned (v/n),
## | Median.Age (v/n), Total.Population (v/n), Average.Household.Size
## | (v/n), Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c)
## + edges from 95fd799 (vertex names):
##  [1] 25--85 68--87 35--87 27--79 30--87 54--87 29--87 67--79 15--87 71--82
## [11] 28--83 24--79 24--82 11--87 69--79 29--79 2 --87 6 --79 69--79 16--79
## [21] 14--79 3 --79 27--87 49--91 44--79 44--87 33--83 44--79 44--79 43--79
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("Original NonViolent Graph Summary:")
```

```
## [1] "Original NonViolent Graph Summary:"
```

```
print(summary(nonviolent_gr))
```

```
## IGRAPH 9602bbf UN-B 91 11445 --
## + attr: name (v/c), Label (v/c), type (v/l), Prop.Occupied (v/n),
## | Prop.Rented (v/n), Prop.Vacant (v/n), Prop.Owned (v/n),
## | Median.Age (v/n), Total.Population (v/n), Average.Household.Size
## | (v/n), Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c)
## IGRAPH 9602bbf UN-B 91 11445 --
## + attr: name (v/c), Label (v/c), type (v/l), Prop.Occupied (v/n),
## | Prop.Rented (v/n), Prop.Vacant (v/n), Prop.Owned (v/n),
## | Median.Age (v/n), Total.Population (v/n), Average.Household.Size
## | (v/n), Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c)
## + edges from 9602bbf (vertex names):
##  [1] 44--98  71--105 37--93  6 --98  51--102 60--98  25--102 32--93
##  [9] 12--93  6 --93  39--92  23--105 26--99  32--94  26--98  24--93
## [17] 2 --99  77--92  66--102 22--94  29--93  69--98  30--93  17--98
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

Create bipartite neighborhood projections

```
# Projections
print("Violent Projection Graph Summaries:")
```

```
## [1] "Violent Projection Graph Summaries:"
```

```
violent_comm = bipartite.projection(violent_gr, which = "TRUE")
print(summary(violent_comm))
```

```
## IGRAPH 962b90a UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c), weight (e/n)
## IGRAPH 962b90a UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c), weight (e/n)
## + edges from 962b90a (vertex names):
##  [1] 1--2  1--3  1--4  1--5  1--6  1--7  1--8  1--9  1--10 1--11 1--13
## [12] 1--14 1--15 1--16 1--19 1--20 1--21 1--22 1--23 1--24 1--25 1--26
## [23] 1--27 1--28 1--29 1--30 1--31 1--32 1--38 1--42 1--43 1--44 1--45
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("NonViolent Projection Graph Summaries:")
```

```
## [1] "NonViolent Projection Graph Summaries:"
```

```
nonviolent_comm = bipartite.projection(nonviolent_gr, which = "TRUE")
print(summary(nonviolent_comm))
```

```
## IGRAPH 963a0eb UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c), weight (e/n)
## IGRAPH 963a0eb UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), Crime.Type (v/c),
## | Crime.Desc (v/c), id (v/c), weight (e/n)
## + edges from 963a0eb (vertex names):
##  [1] 1--2  1--3  1--4  1--5  1--6  1--7  1--8  1--9  1--10 1--11 1--12
## [12] 1--13 1--14 1--15 1--16 1--17 1--18 1--19 1--20 1--21 1--22 1--23
## [23] 1--24 1--25 1--26 1--27 1--28 1--29 1--30 1--31 1--32 1--33 1--34
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

Remove unrelated node attributes

```
# Remove non-matching attributes
print("Violent Projections After Vertex Removal:")
```

```
## [1] "Violent Projections After Vertex Removal:"
```

```
violent_clean = delete_vertex_attr(violent_comm, "Crime.Type")
violent_clean = delete_vertex_attr(violent_clean, "Crime.Desc")
print(summary(violent_clean))
```

```
## IGRAPH 962b90a UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## IGRAPH 962b90a UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 962b90a (vertex names):
##  [1] 1--2  1--3  1--4  1--5  1--6  1--7  1--8  1--9  1--10 1--11 1--13
## [12] 1--14 1--15 1--16 1--19 1--20 1--21 1--22 1--23 1--24 1--25 1--26
## [23] 1--27 1--28 1--29 1--30 1--31 1--32 1--38 1--42 1--43 1--44 1--45
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("NonViolent Projections After Vertex Removal:")
```

```
## [1] "NonViolent Projections After Vertex Removal:"
```

```
nonviolent_clean = delete_vertex_attr(nonviolent_comm, "Crime.Type")
nonviolent_clean = delete_vertex_attr(nonviolent_clean, "Crime.Desc")
print(summary(nonviolent_clean))
```

```
## IGRAPH 963a0eb UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## IGRAPH 963a0eb UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 963a0eb (vertex names):
##  [1] 1--2  1--3  1--4  1--5  1--6  1--7  1--8  1--9  1--10 1--11 1--12
## [12] 1--13 1--14 1--15 1--16 1--17 1--18 1--19 1--20 1--21 1--22 1--23
## [23] 1--24 1--25 1--26 1--27 1--28 1--29 1--30 1--31 1--32 1--33 1--34
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

Calculate initial densities

```
# Calculate graph densities
violent_density = edge_density(violent_clean)
print(paste("Violent Graph Density:", violent_density))
```

```
## [1] "Violent Graph Density: 1"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
nonviolent_density = edge_density(nonviolent_clean)
print(paste("NonViolent Graph Density:", nonviolent_density))
```

```
## [1] "NonViolent Graph Density: 1"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

Calculate initial number of components

```
# See if there are any components
print("Violent Graph Components:")
```

```
## [1] "Violent Graph Components:"
```

```
violent_decomp = decompose(violent_clean)
print(violent_decomp)
```

```
## [[1]]
## IGRAPH 9651170 UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 9651170 (vertex names):
##  [1] 1--2  1--3  1--4  1--5  1--6  1--7  1--8  1--9  1--10 1--11 1--13
## [12] 1--14 1--15 1--16 1--19 1--20 1--21 1--22 1--23 1--24 1--25 1--26
## [23] 1--27 1--28 1--29 1--30 1--31 1--32 1--38 1--42 1--43 1--44 1--45
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("NonViolent Graph Components:")
```

```
## [1] "NonViolent Graph Components:"
```

```
nonviolent_decomp = decompose(nonviolent_clean)
print(nonviolent_decomp)
```

```
## [[1]]
## IGRAPH 96546c3 UNW- 77 2926 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 96546c3 (vertex names):
##  [1] 1--2  1--3  1--4  1--5  1--6  1--7  1--8  1--9  1--10 1--11 1--12
## [12] 1--13 1--14 1--15 1--16 1--17 1--18 1--19 1--20 1--21 1--22 1--23
## [23] 1--24 1--25 1--26 1--27 1--28 1--29 1--30 1--31 1--32 1--33 1--34
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

Summarize initial edge weights

```
print("Violent Edge Weight Summary:")
```

```
## [1] "Violent Edge Weight Summary:"
```

```
print(summary(E(violent_clean)$weight))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     6.0   398.2  1070.5  2344.6  2655.8 44381.0
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("NonViolent Edge Weight Summary:")
```

```
## [1] "NonViolent Edge Weight Summary:"
```

```
print(summary(E(nonviolent_clean)$weight))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     41.0    982.2   2272.0   4734.2   5127.0 138304.0
```

```
print("", quote=FALSE)
```

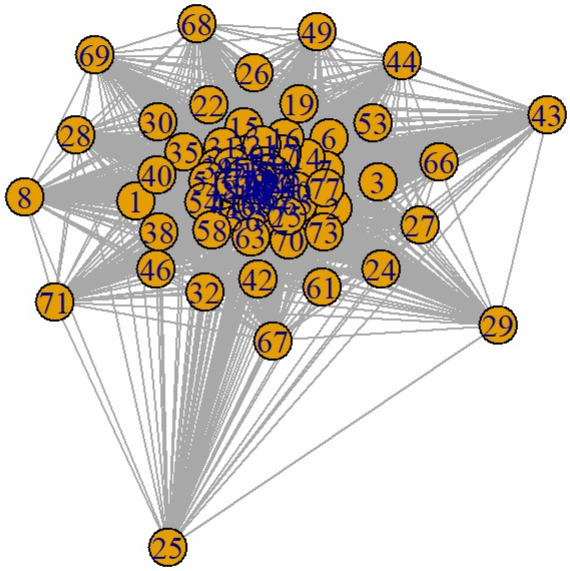
```
## [1]
```

```
print("", quote=FALSE)
```

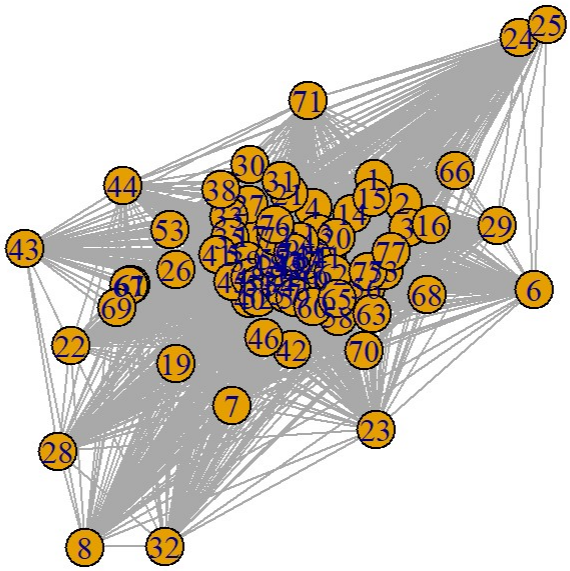
```
## [1]
```

Plot initial graphs

```
# Plot graphs
if (save_to_file){
  temp_jpg = paste(output_path, "Violent_Graph.jpg", sep = "")
  jpeg(file = temp_jpg)
}

plot(violent_clean, layout = layout_with_kk)
```



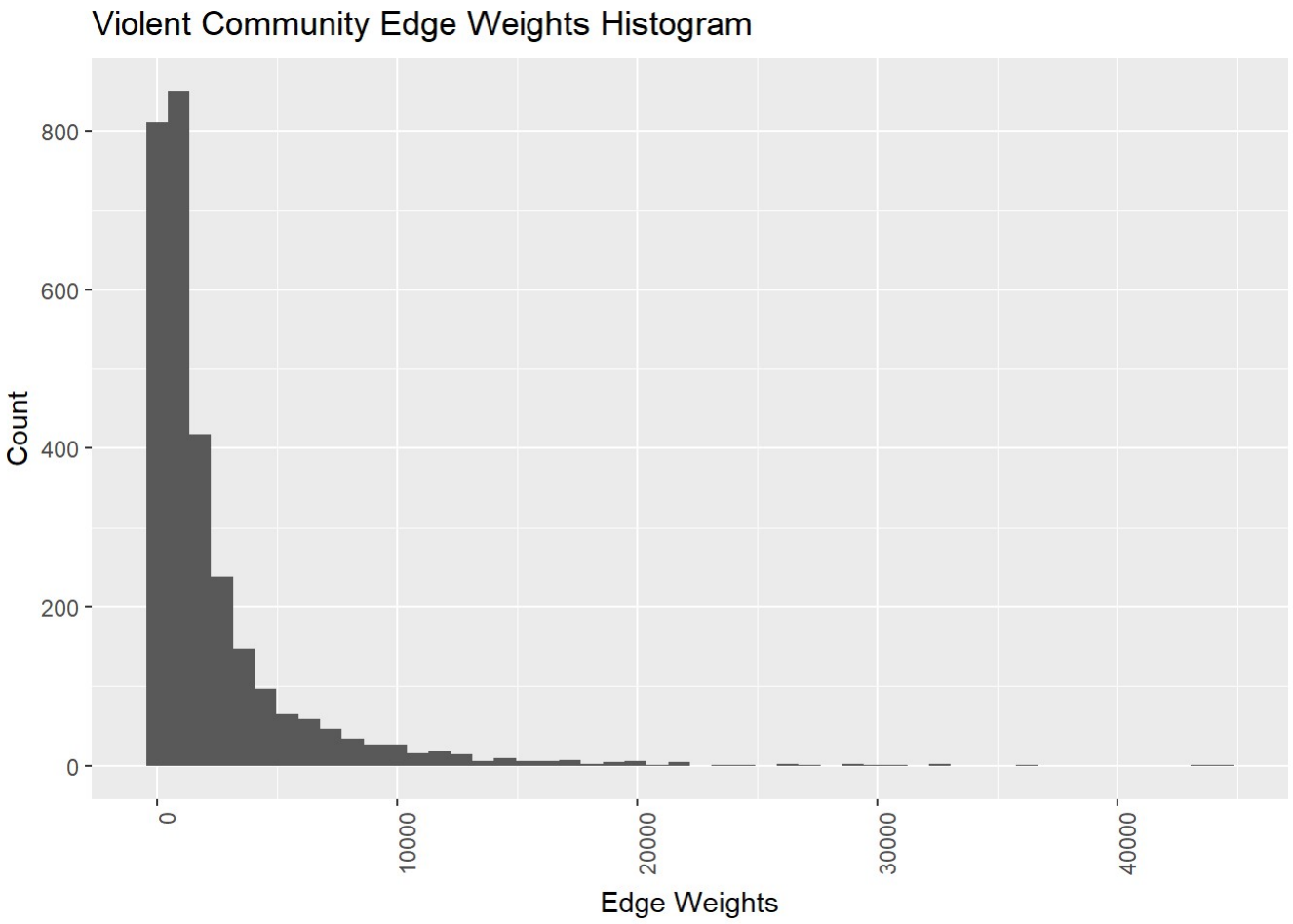
```
if (save_to_file){
  dev.off()
}

if (save_to_file){
  temp_jpg = paste(output_path, "Nonviolent_Graph.jpg", sep = "")
  jpeg(file = temp_jpg)
}

plot(nonviolent_clean, layout = layout_with_kk)
```



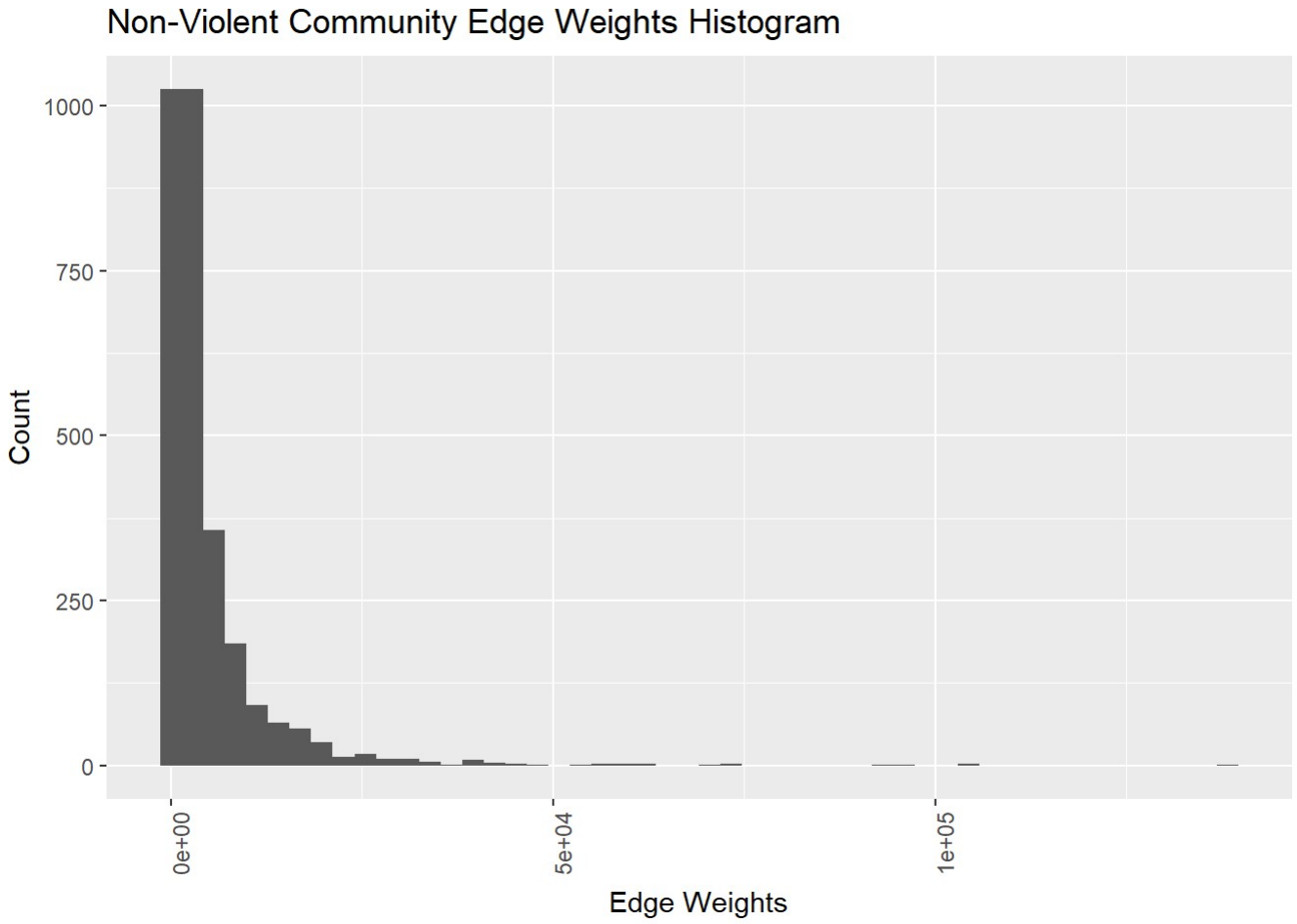

```
if (save_to_file){
  dev.off()
}

##################### Edge Weights ################################################
#############
```

Create Edge Weight plots to use for filtering

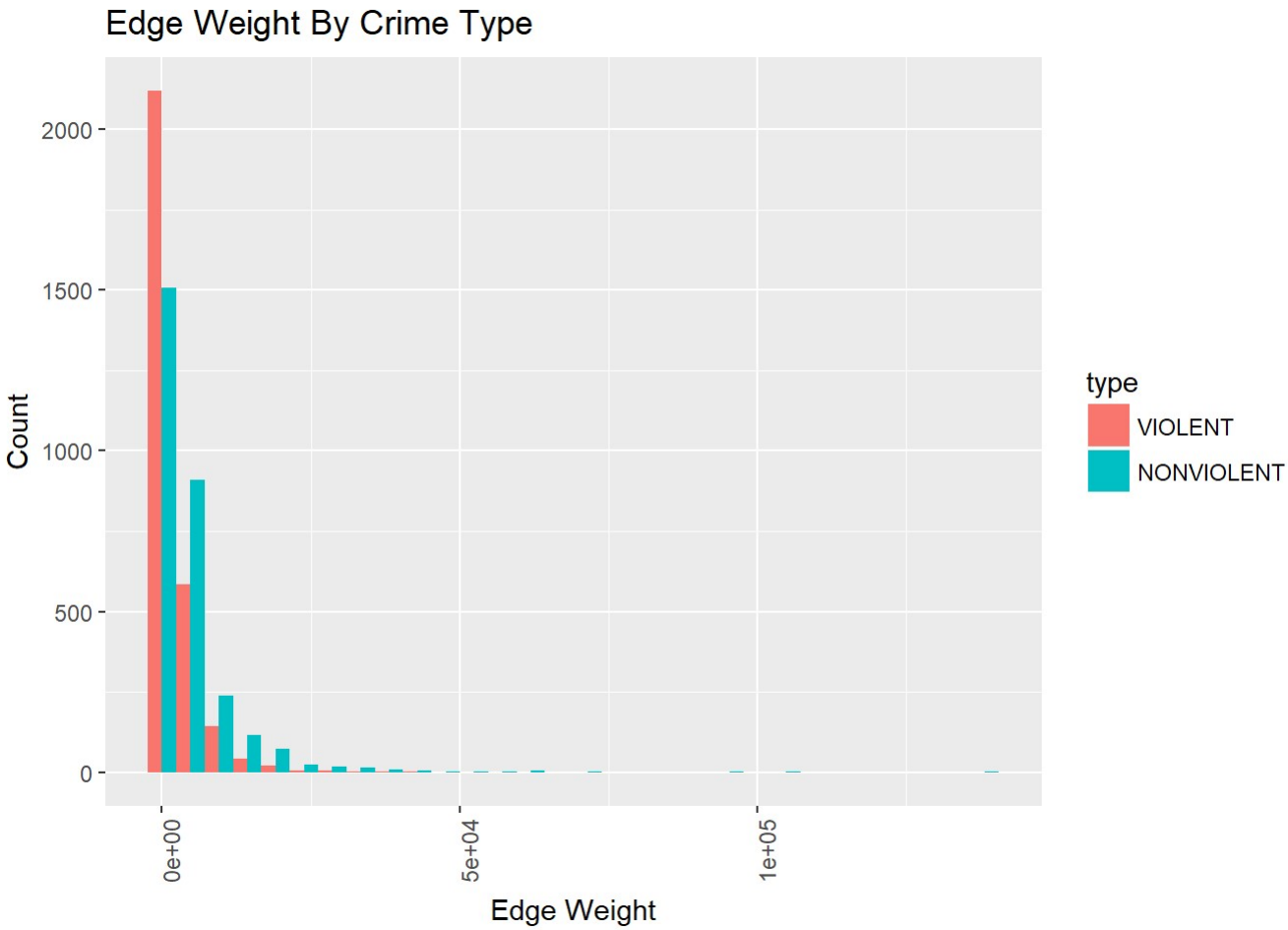
```
# Violent Edge weight histogram
if (save_to_file){
  temp_jpg = paste(output_path, "Violent_Hist_EdgeWeight_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_violentw = ggplot(data = data.frame(weights = E(violent_clean)$weight), aes(x=weigh
ts)) +
  geom_histogram(bins = 50) +
  ggtitle("Violent Community Edge Weights Histogram") +
  labs(x = "Edge Weights", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_violentw)
```



```
if (save_to_file){
  dev.off()
}

# Nonviolent Edge weight histogram
if (save_to_file){
  temp_jpg = paste(output_path, "Nonviolent_Hist_EdgeWeight_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_nonviolentw = ggplot(data = data.frame(weights = E(nonviolent_clean)$weight), aes(x
=weights)) +
  geom_histogram(bins = 50) +
  ggtitle("Non-Violent Community Edge Weights Histogram") +
  labs(x = "Edge Weights", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_nonviolentw)
```

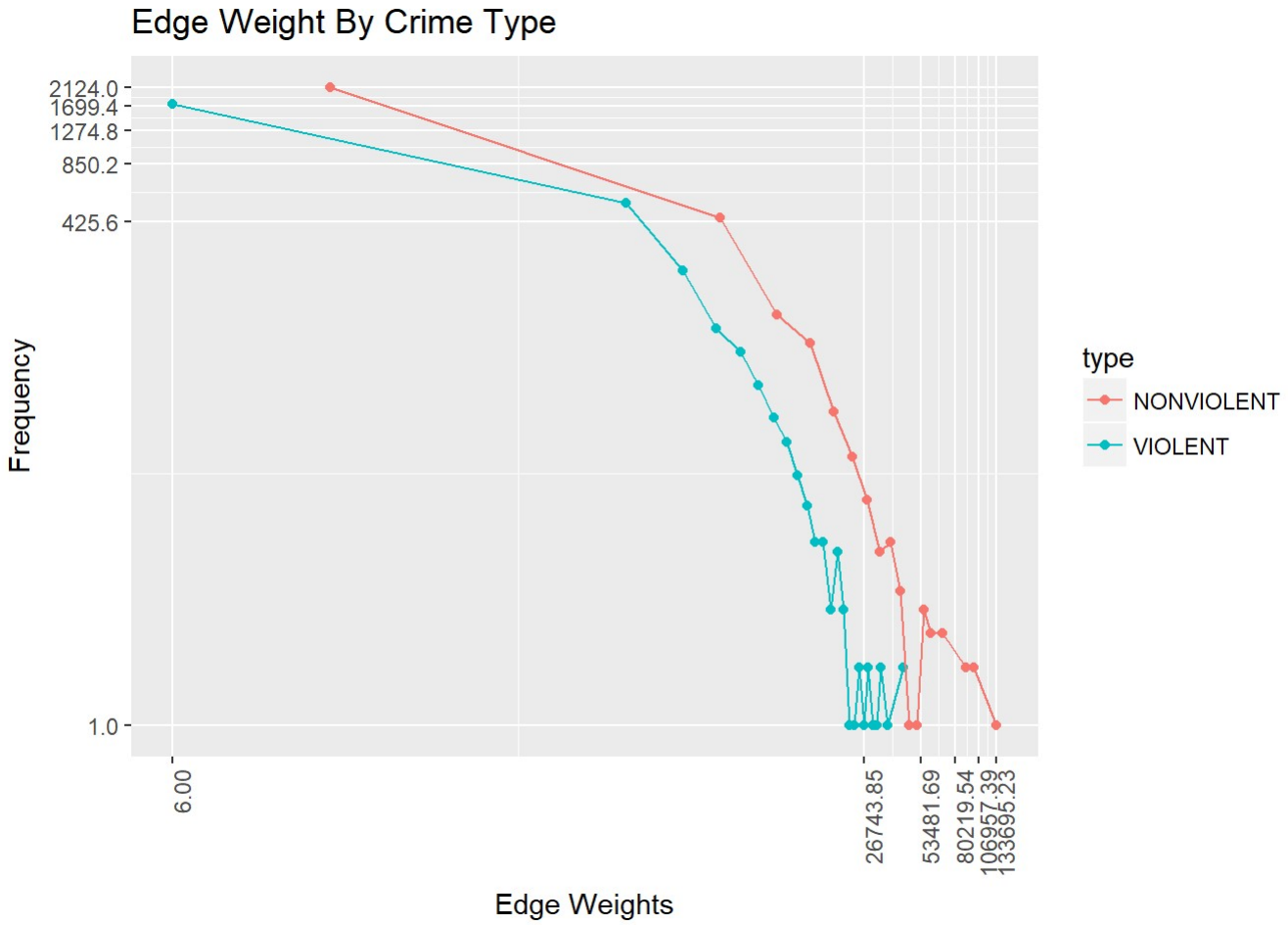


```
if (save_to_file){
  dev.off()
}

# Combined Edge Weight Distribution Histogram
weight_df = data.frame(type = rep("VIOLENT", length(E(violent_clean)$weight)),
                       weight = E(violent_clean)$weight)
weight_df = rbind(weight_df, data.frame(type = rep("NONVIOLENT", length(E(nonviolent_
clean)$weight)),
                                        weight = E(nonviolent_clean)$weight))

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Hist_EdgeW_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_ehist = ggplot(data = weight_df, aes(x = weight, fill = type)) +
  geom_histogram(position = "dodge", bins = 30) +
  ggtitle("Edge Weight By Crime Type") +
  labs(x = "Edge Weight", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_ehist)
```



```
if (save_to_file){
  dev.off()
}

# Combined Edge Weight Log-Log Distribution
v_num_edge_bins = 30
v_min_edge = min(E(violent_clean)$weight)
v_max_edge = max(E(violent_clean)$weight)
v_step_edge = (v_max_edge - v_min_edge) / v_num_edge_bins

v_edge_breaks = seq(v_min_edge, v_max_edge, v_step_edge)
edge_bins = cut(E(violent_clean)$weight, breaks = v_edge_breaks, labels=FALSE)
v_tab_edge = data.frame(tabulate(edge_bins))

v_tab_edge$edge <- v_edge_breaks[1:(length(v_edge_breaks)-1)]
v_tab_edge = v_tab_edge[v_tab_edge$tabulate.edge_bins.>0,]
v_tab_edge$type = rep("VIOLENT", length(v_tab_edge$edge))

nv_num_edge_bins = 30
nv_min_edge = min(E(nonviolent_clean)$weight)
nv_max_edge = max(E(nonviolent_clean)$weight)
nv_step_edge = (nv_max_edge - nv_min_edge) / nv_num_edge_bins

nv_edge_breaks = seq(nv_min_edge, nv_max_edge, nv_step_edge)
edge_bins = cut(E(nonviolent_clean)$weight, breaks = nv_edge_breaks, labels=FALSE)
nv_tab_edge = data.frame(tabulate(edge_bins))

nv_tab_edge$edge <- nv_edge_breaks[1:(length(nv_edge_breaks)-1)]
nv_tab_edge = nv_tab_edge[nv_tab_edge$tabulate.edge_bins.>0,]
nv_tab_edge$type = rep("NONVIOLENT", length(nv_tab_edge$edge))

log_weight_df = rbind(v_tab_edge,nv_tab_edge)

x_num_breaks = 5
x_min_edge = min(log_weight_df$edge)
x_max_edge = max(log_weight_df$edge)
x_step_edge = (x_max_edge - x_min_edge) / x_num_breaks

y_num_breaks = 5
y_min_edge = min(log_weight_df$tabulate.edge_bins.)
y_max_edge = max(log_weight_df$tabulate.edge_bins.)
y_step_edge = (y_max_edge - y_min_edge) / y_num_breaks

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Log_EdgeW_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_log = ggplot(log_weight_df, aes(x=edge, y=tabulate.edge_bins., color = typ
e)) +
  geom_point() + geom_line() +
  scale_x_log10(name="Edge Weights", breaks = seq(x_min_edge, x_max_edge, x_step_edge
)) +
```

```
  scale_y_log10("Frequency", breaks = seq(y_min_edge, y_max_edge, y_step_edge)) +
  ggtitle("Edge Weight By Crime Type") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_log)
```




```
if (save_to_file){
  dev.off()
}

################################## Edge Filtering ###################################
############################
```

Filter edges by percent of edges to eliminate

Initially filtered edges by values extracted from the histograms. This provided an initial idea of how many edges may need to be filtered. In order to get more precise filtering thresholds, a number of different of filtering percentages were performed (edges below the edge weight percentile value corresponding to the percentage were removed). Analyses of the assortativity, transitivity, and modularity of the filtered networks (as well as the Gephi plots of the networks) were performed to determine which percentages should be used in the final filtering steps.

```
# get violent edge filtering threshold from values of the histogram
# print("Violent Edge Weight Filtering:")
# ve_hist_data = ggplot_build(g_violentw)
# print(ve_hist_data$data[[1]])
# print("", quote=FALSE)
# ve_thresh_ndx = 4
# ve_thresh =  ve_hist_data$data[[1]]$xmax[ve_thresh_ndx]  # Found by using Gephi edg
e weight filtering to come up w/ approx.
# print(paste("Violent Edge Weight Filtering Threshold:", ve_thresh))
# print("", quote=FALSE)
# ve_perc_rem = sum(ve_hist_data$data[[1]]$count[1:ve_thresh_ndx]) / sum(ve_hist_data
$data[[1]]$count)
# print(paste("Violent Edge Weight Filtering % Edges Removed:", ve_perc_rem))
# print("", quote=FALSE)
# print("", quote=FALSE)

# get nonviolent edge filtering threshold from values of the histogram
# print("NonViolent Edge Weight Filtering:")
# nve_hist_data = ggplot_build(g_nonviolentw)
# print(nve_hist_data$data[[1]])
# print("", quote=FALSE)
# nve_thresh_ndx = 3
# nve_thresh =  nve_hist_data$data[[1]]$xmax[nve_thresh_ndx]  # Found by using Gephi
edge weight filtering to come up w/ approx.
# print(paste("NonViolent Edge Weight Filtering Threshold:", nve_thresh))
# print("", quote=FALSE)
# nve_perc_rem = sum(nve_hist_data$data[[1]]$count[1:nve_thresh_ndx]) / sum(nve_hist_
data$data[[1]]$count)
# print(paste("NonViolent Edge Weight Filtering % Edges Removed:", nve_perc_rem))
# print("", quote=FALSE)
# print("", quote=FALSE)

# get violent edge filtereing threshold from percent data left
ve_perc_rem = 0.95
v_edge_w = E(violent_clean)$weight
v_sorted_edge_w = sort(v_edge_w)
ve_thresh = quantile(v_sorted_edge_w, ve_perc_rem)
print(paste("Violent Edge Weight Filtering Threshold:", ve_thresh))
```

```
## [1] "Violent Edge Weight Filtering Threshold: 8950.25"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(paste("Violent Edge Weight Filtering % Edges Removed:", ve_perc_rem))
```

```
## [1] "Violent Edge Weight Filtering % Edges Removed: 0.95"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
# get nonviolent edge filtereing threshold from percent data left
nve_perc_rem = 0.95
nv_edge_w = E(nonviolent_clean)$weight
nv_sorted_edge_w = sort(nv_edge_w)
nve_thresh = quantile(nv_sorted_edge_w, nve_perc_rem)
print(paste("NonViolent Edge Weight Filtering Threshold:", nve_thresh))
```

```
## [1] "NonViolent Edge Weight Filtering Threshold: 17100"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(paste("NonViolent Edge Weight Filtering % Edges Removed:", nve_perc_rem))
```

```
## [1] "NonViolent Edge Weight Filtering % Edges Removed: 0.95"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
violent_filter = delete.edges(violent_clean, which(E(violent_clean)$weight < ve_thres
h))
print("Violent Graph After Edge Filtering:")
```

```
## [1] "Violent Graph After Edge Filtering:"
```

```
print(summary(violent_filter))
```

```
## IGRAPH 98e1601 UNW- 77 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## IGRAPH 98e1601 UNW- 77 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 98e1601 (vertex names):
##  [1] 1 --25 1 --29 1 --43 2 --25 3 --25 3 --29 3 --43 6 --25 8 --23 8 --24
## [11] 8 --25 8 --27 8 --28 8 --29 8 --43 8 --44 8 --46 8 --49 8 --61 8 --66
## [21] 8 --67 8 --68 8 --69 8 --71 15--25 19--25 19--29 22--25 23--25 23--28
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
nonviolent_filter = delete.edges(nonviolent_clean, which(E(nonviolent_clean)$weight <
nve_thresh))
print("NonViolent Graph After Edge Filtering:")
```

```
## [1] "NonViolent Graph After Edge Filtering:"
```

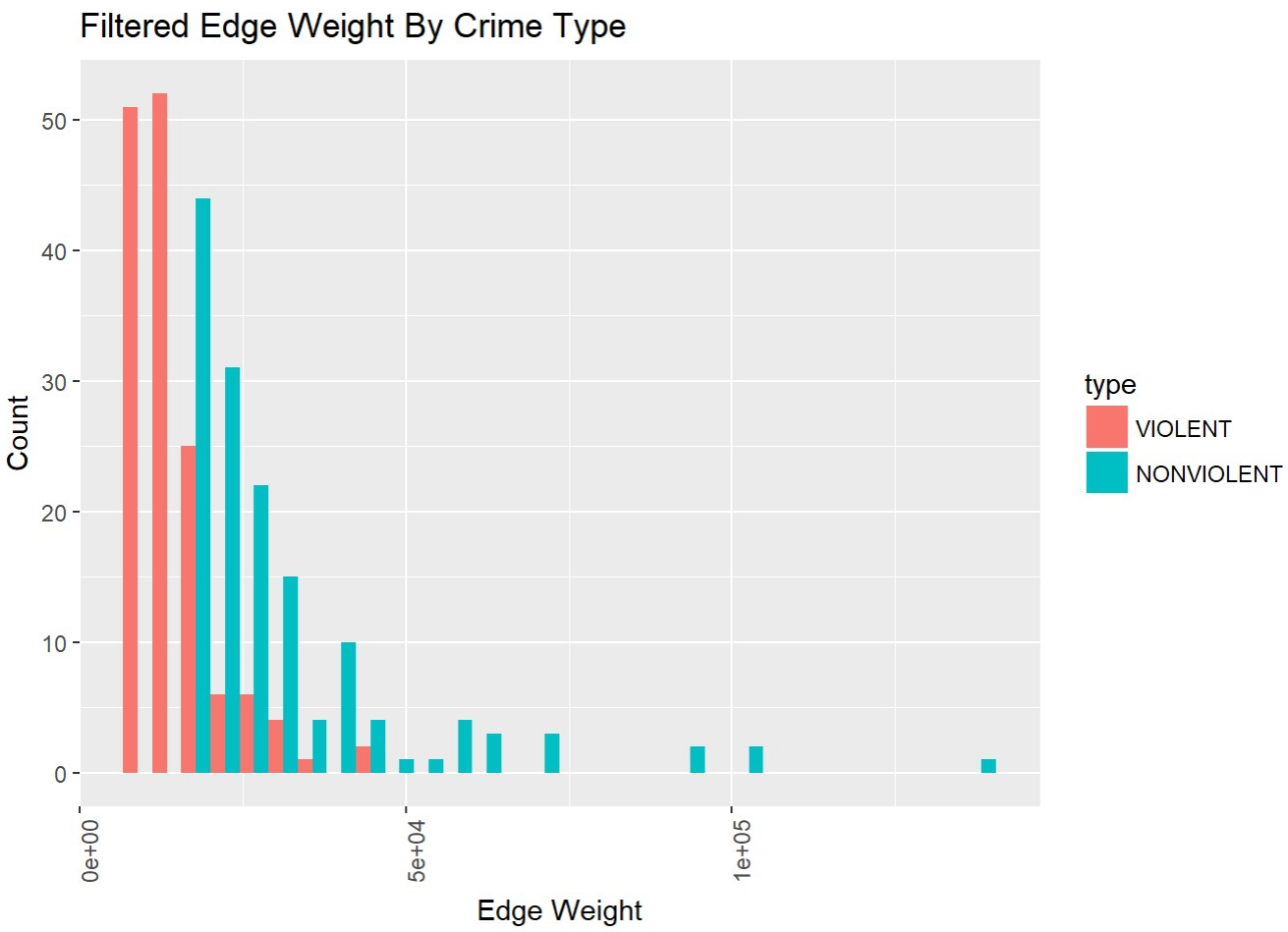
```
print(summary(nonviolent_filter))
```

```
## IGRAPH 98e3c26 UNW- 77 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## IGRAPH 98e3c26 UNW- 77 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 98e3c26 (vertex names):
##  [1] 1--8  1--24 1--25 1--28 1--32 2--8  2--32 3--8  3--24 3--28 3--32
## [12] 4--8  4--32 6--7  6--8  6--22 6--23 6--24 6--25 6--28 6--32 6--43
## [23] 6--44 6--71 7--8  7--22 7--23 7--24 7--25 7--28 7--32 7--43 7--44
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

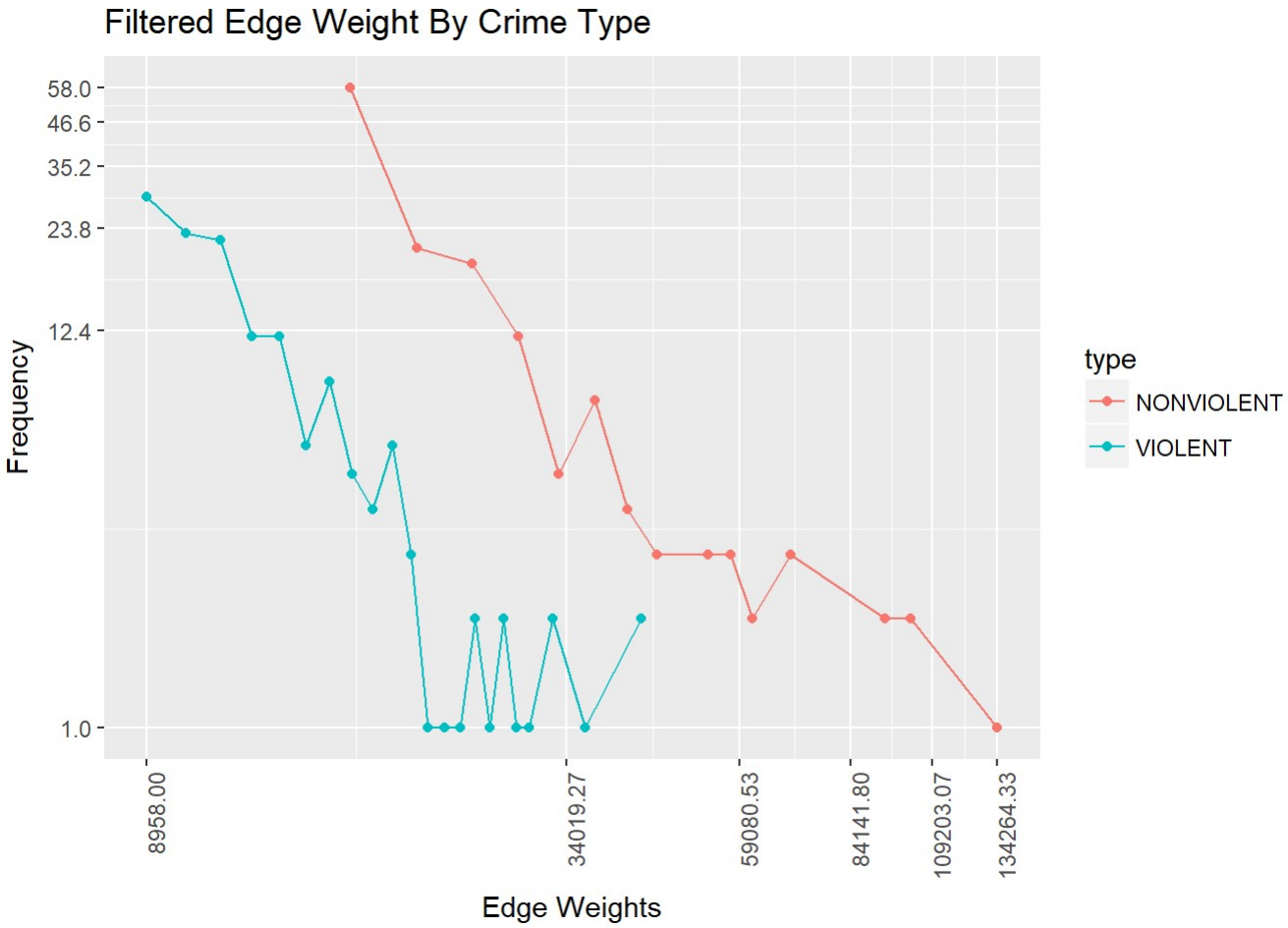
```
## [1]
```

```
# Combined Edge Weight Distribution Histogram
weight_df = data.frame(type = rep("VIOLENT", length(E(violent_filter)$weight)),
                       weight = E(violent_filter)$weight)
weight_df = rbind(weight_df, data.frame(type = rep("NONVIOLENT", length(E(nonviolent_
filter)$weight)),
                                        weight = E(nonviolent_filter)$weight))

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Hist_EdgeW_Filtered.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_ehist = ggplot(data = weight_df, aes(x = weight, fill = type)) +
  geom_histogram(position = "dodge", bins = 30) +
  ggtitle("Filtered Edge Weight By Crime Type") +
  labs(x = "Edge Weight", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_ehist)
```


Filtered Edge Weight By Crime Type
Count
50
40
30
20
10
0
0e+00
5e+04
1e+05
Edge Weight
type
VIOLENT
NONVIOLENT

```
if (save_to_file){
  dev.off()
}

# Combined Edge Weight Log-Log Distribution
v_num_edge_bins = 30
v_min_edge = min(E(violent_filter)$weight)
v_max_edge = max(E(violent_filter)$weight)
v_step_edge = (v_max_edge - v_min_edge) / v_num_edge_bins

v_edge_breaks = seq(v_min_edge, v_max_edge, v_step_edge)
edge_bins = cut(E(violent_filter)$weight, breaks = v_edge_breaks, labels=FALSE)
v_tab_edge = data.frame(tabulate(edge_bins))

v_tab_edge$edge <- v_edge_breaks[1:(length(v_edge_breaks)-1)]
v_tab_edge = v_tab_edge[v_tab_edge$tabulate.edge_bins.>0,]
v_tab_edge$type = rep("VIOLENT", length(v_tab_edge$edge))

nv_num_edge_bins = 30
nv_min_edge = min(E(nonviolent_filter)$weight)
nv_max_edge = max(E(nonviolent_filter)$weight)
nv_step_edge = (nv_max_edge - nv_min_edge) / nv_num_edge_bins

nv_edge_breaks = seq(nv_min_edge, nv_max_edge, nv_step_edge)
edge_bins = cut(E(nonviolent_filter)$weight, breaks = nv_edge_breaks, labels=FALSE)
nv_tab_edge = data.frame(tabulate(edge_bins))

nv_tab_edge$edge <- nv_edge_breaks[1:(length(nv_edge_breaks)-1)]
nv_tab_edge = nv_tab_edge[nv_tab_edge$tabulate.edge_bins.>0,]
nv_tab_edge$type = rep("NONVIOLENT", length(nv_tab_edge$edge))

log_weight_df = rbind(v_tab_edge,nv_tab_edge)

x_num_breaks = 5
x_min_edge = min(log_weight_df$edge)
x_max_edge = max(log_weight_df$edge)
x_step_edge = (x_max_edge - x_min_edge) / x_num_breaks

y_num_breaks = 5
y_min_edge = min(log_weight_df$tabulate.edge_bins.)
y_max_edge = max(log_weight_df$tabulate.edge_bins.)
y_step_edge = (y_max_edge - y_min_edge) / y_num_breaks

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Log_EdgeW_Filtered.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_log = ggplot(log_weight_df, aes(x=edge, y=tabulate.edge_bins., color = typ
e)) +
  geom_point() + geom_line() +
  scale_x_log10(name="Edge Weights", breaks = seq(x_min_edge, x_max_edge, x_step_edge
)) +
```

```
  scale_y_log10("Frequency", breaks = seq(y_min_edge, y_max_edge, y_step_edge)) +
  ggtitle("Filtered Edge Weight By Crime Type") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_log)
```




```
if (save_to_file){
  dev.off()
}

##################### Weighted Degree ##############################################
###############
```

Initial Weighted Degree plots

```
violent_wdeg = graph.strength(violent_clean)
nonviolent_wdeg = graph.strength(nonviolent_clean)

print("Violent Weighted Degree Summary:")
```

```
## [1] "Violent Weighted Degree Summary:"
```

```
print(summary(violent_wdeg))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9015   55348  116687  178193  261552  940526
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("NonViolent Weighted Degree Summary:")
```

```
## [1] "NonViolent Weighted Degree Summary:"
```

```
print(summary(nonviolent_wdeg))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   22468  144698  272209  359801  411553 1696784
```

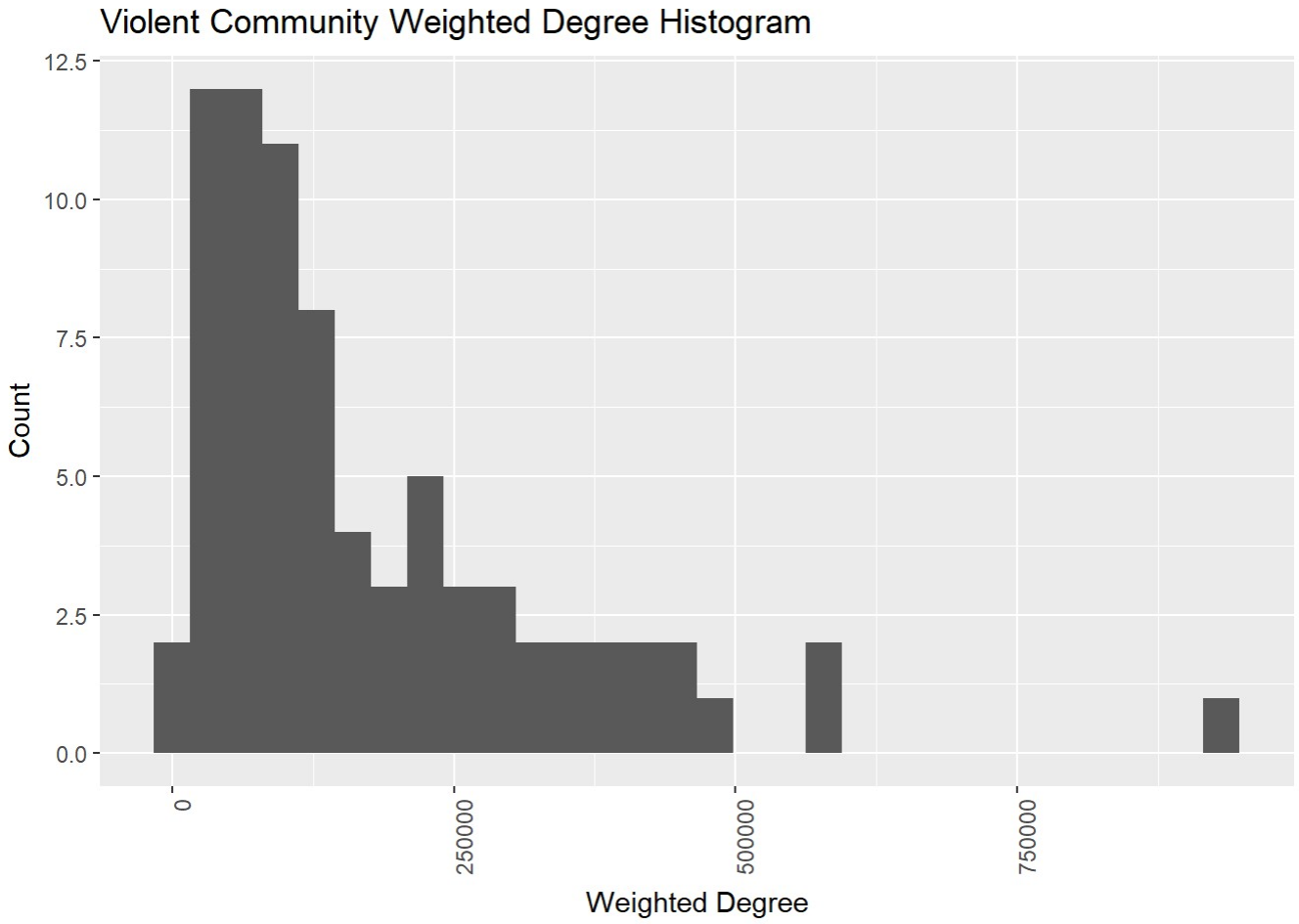
```
print("", quote=FALSE)
```

```
## [1]
```

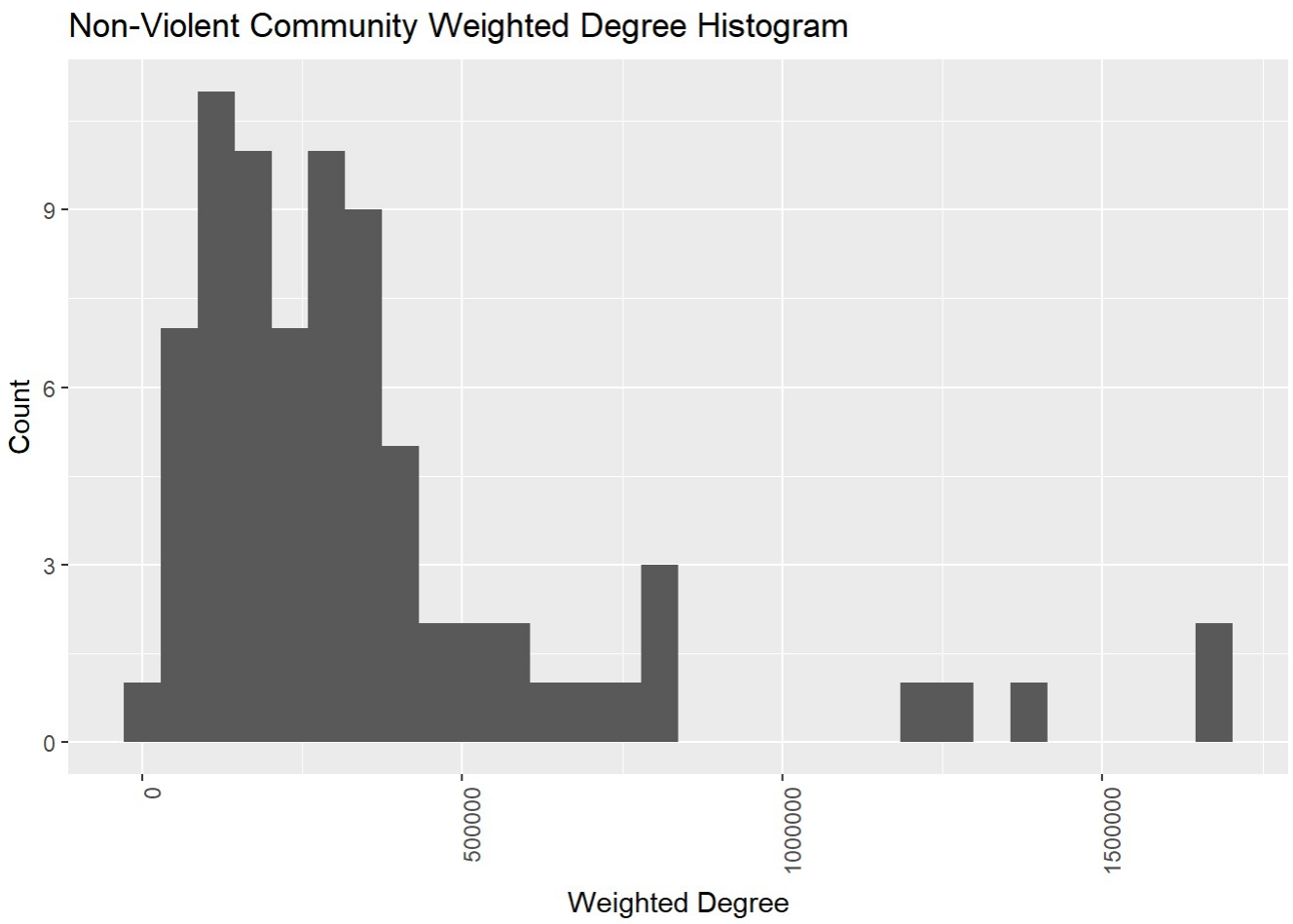
```
print("", quote=FALSE)
```

```
## [1]
```

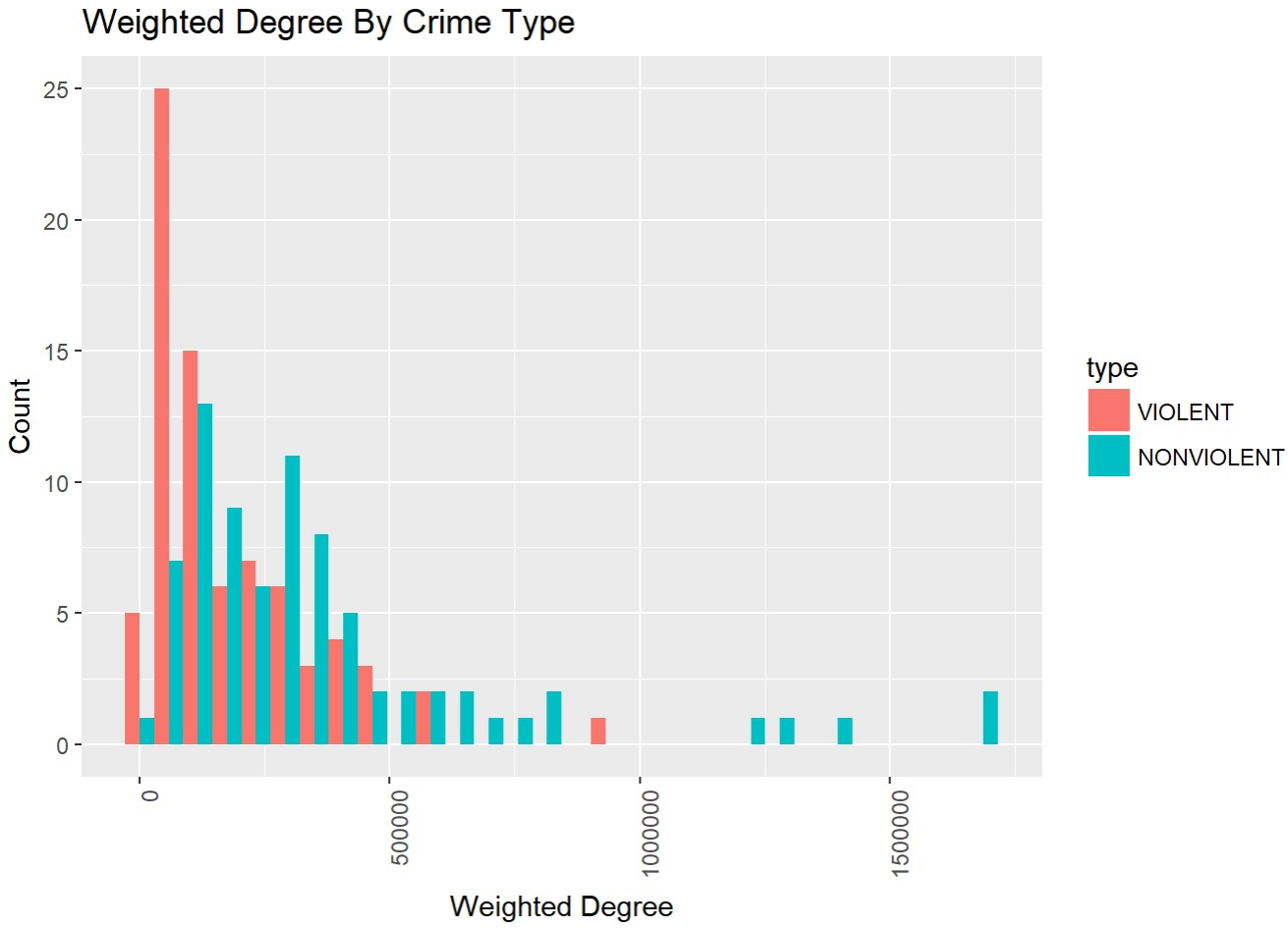
```
# Violent Degree Distribution
if (save_to_file){
  temp_jpg = paste(output_path, "Violent_Hist_WDegree_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_violent_wdeg = ggplot(data = data.frame(WDegree = violent_wdeg), aes(x=WDegree)) +
  geom_histogram(bins = 30) +
  ggtitle("Violent Community Weighted Degree Histogram") +
  labs(x = "Weighted Degree", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_violent_wdeg)
```




```
if (save_to_file){
  dev.off()
}

# Non-Violent Degree Distribution
if (save_to_file){
  temp_jpg = paste(output_path, "Nonviolent_Hist_WDegree_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_nonviolent_wdeg = ggplot(data = data.frame(WDegree = nonviolent_wdeg), aes(x=WDegre
e)) +
  geom_histogram(bins = 30) +
  ggtitle("Non-Violent Community Weighted Degree Histogram") +
  labs(x = "Weighted Degree", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_nonviolent_wdeg)
```

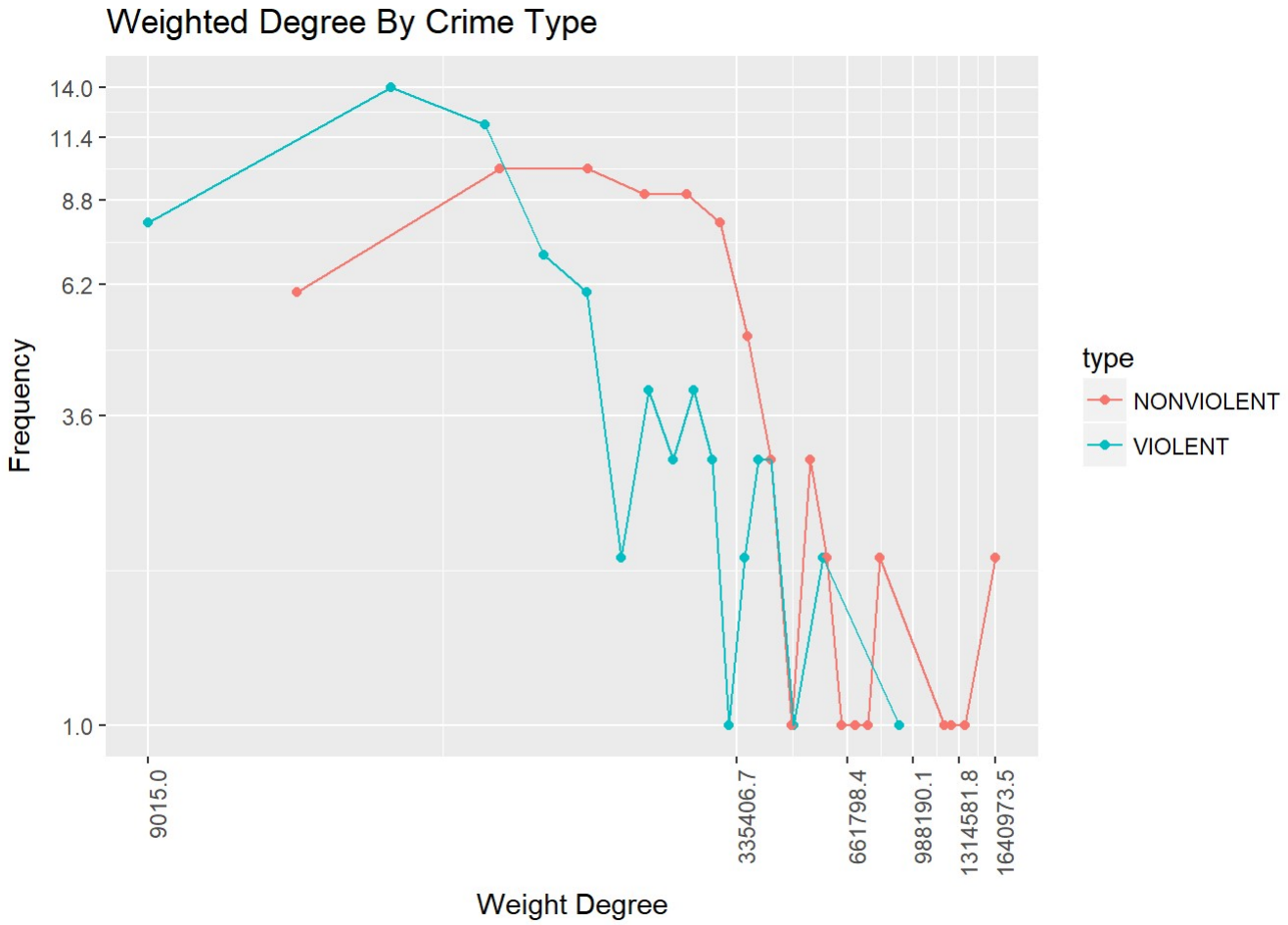



```
if (save_to_file){
  dev.off()
}

# Combined Weighted Degree Distribution Histogram
degree_df = data.frame(type = rep("VIOLENT", length(violent_wdeg)), degree = violent_
wdeg)
degree_df = rbind(degree_df, data.frame(type = rep("NONVIOLENT", length(nonviolent_wd
eg)),
                                        degree = nonviolent_wdeg))

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Hist_WDegree_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_hist = ggplot(data = degree_df, aes(x = degree, fill = type)) +
  geom_histogram(position = "dodge", bins = 30) +
  ggtitle("Weighted Degree By Crime Type") +
  labs(x = "Weighted Degree", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_hist)
```


Weighted Degree By Crime Type
Count
25
20
15
10
5
0
0
500000
1000000
1500000
Weighted Degree
type
VIOLENT
NONVIOLENT

```
if (save_to_file){
  dev.off()
}

# Combined Weighted Degree Log-Log Distribution
v_num_wdeg_bins = 30
v_min_wdeg = min(violent_wdeg)
v_max_wdeg = max(violent_wdeg)
v_step_wdeg = (v_max_wdeg - v_min_wdeg) / v_num_wdeg_bins

v_wdeg_breaks = seq(v_min_wdeg, v_max_wdeg, v_step_wdeg)
wdeg_bins = cut(violent_wdeg, breaks = v_wdeg_breaks, labels=FALSE)
v_tab_wdeg = data.frame(tabulate(wdeg_bins))

v_tab_wdeg$wdeg <- v_wdeg_breaks[1:(length(v_wdeg_breaks)-1)]
v_tab_wdeg = v_tab_wdeg[v_tab_wdeg$tabulate.wdeg_bins.>0,]
v_tab_wdeg$type = rep("VIOLENT", length(v_tab_wdeg$wdeg))

nv_num_wdeg_bins = 30
nv_min_wdeg = min(nonviolent_wdeg)
nv_max_wdeg = max(nonviolent_wdeg)
nv_step_wdeg = (nv_max_wdeg - nv_min_wdeg) / nv_num_wdeg_bins

nv_wdeg_breaks = seq(nv_min_wdeg, nv_max_wdeg, nv_step_wdeg)
wdeg_bins = cut(nonviolent_wdeg, breaks = nv_wdeg_breaks, labels=FALSE)
nv_tab_wdeg = data.frame(tabulate(wdeg_bins))

nv_tab_wdeg$wdeg <- nv_wdeg_breaks[1:(length(nv_wdeg_breaks)-1)]
nv_tab_wdeg = nv_tab_wdeg[nv_tab_wdeg$tabulate.wdeg_bins.>0,]
nv_tab_wdeg$type = rep("NONVIOLENT", length(nv_tab_wdeg$wdeg))

log_weight_df = rbind(v_tab_wdeg,nv_tab_wdeg)

x_num_breaks = 5
x_min_wdeg = min(log_weight_df$wdeg)
x_max_wdeg = max(log_weight_df$wdeg)
x_step_wdeg = (x_max_wdeg - x_min_wdeg) / x_num_breaks

y_num_breaks = 5
y_min_wdeg = min(log_weight_df$tabulate.wdeg_bins.)
y_max_wdeg = max(log_weight_df$tabulate.wdeg_bins.)
y_step_wdeg = (y_max_wdeg - y_min_wdeg) / y_num_breaks

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Log_WDeg_Init.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_log = ggplot(log_weight_df, aes(x=wdeg, y=tabulate.wdeg_bins., color = typ
e)) +
  geom_point() + geom_line() +
  scale_x_log10(name="Weight Degree", breaks = seq(x_min_wdeg, x_max_wdeg, x_step_wde
g)) +
```

```
  scale_y_log10("Frequency", breaks = seq(y_min_wdeg, y_max_wdeg, y_step_wdeg)) +
  ggtitle("Weighted Degree By Crime Type") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_log)
```




```
if (save_to_file){
  dev.off()
}

################################## Singleton Filtering ##############################
#########################
```

Singleton node removal

```
violent_filter = delete.vertices(violent_filter, which(degree(violent_filter) == 0))
print("Violent Graph After Singleton Filtering:")
```

```
## [1] "Violent Graph After Singleton Filtering:"
```

```
print(summary(violent_filter))
```

```
## IGRAPH 9a63112 UNW- 36 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## IGRAPH 9a63112 UNW- 36 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 9a63112 (vertex names):
##  [1] 1 --25 1 --29 1 --43 2 --25 3 --25 3 --29 3 --43 6 --25 8 --23 8 --24
## [11] 8 --25 8 --27 8 --28 8 --29 8 --43 8 --44 8 --46 8 --49 8 --61 8 --66
## [21] 8 --67 8 --68 8 --69 8 --71 15--25 19--25 19--29 22--25 23--25 23--28
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
nonviolent_filter = delete.vertices(nonviolent_filter, which(degree(nonviolent_filter
) == 0))
print("NonViolent Graph After Singleton Filtering:")
```

```
## [1] "NonViolent Graph After Singleton Filtering:"
```

```
print(summary(nonviolent_filter))
```

```
## IGRAPH 9a65738 UNW- 38 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## IGRAPH 9a65738 UNW- 38 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), weight
## | (e/n)
## + edges from 9a65738 (vertex names):
##  [1] 1--8  1--24 1--25 1--28 1--32 2--8  2--32 3--8  3--24 3--28 3--32
## [12] 4--8  4--32 6--7  6--8  6--22 6--23 6--24 6--25 6--28 6--32 6--43
## [23] 6--44 6--71 7--8  7--22 7--23 7--24 7--25 7--28 7--32 7--43 7--44
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
violent_filter_wdeg = graph.strength(violent_filter)
nonviolent_filter_wdeg = graph.strength(nonviolent_filter)

V(violent_filter)$wdegree = violent_filter_wdeg
V(nonviolent_filter)$wdegree = nonviolent_filter_wdeg


print("Filtered Violent Weighted Degree Summary:")
```

```
## [1] "Filtered Violent Weighted Degree Summary:"
```

```
print(summary(violent_filter_wdeg))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9535   14365   52938  120548  178986  745870
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("Filtered NonViolent Weighted Degree Summary:")
```

```
## [1] "Filtered NonViolent Weighted Degree Summary:"
```

```
print(summary(nonviolent_filter_wdeg))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17114   41082  108029  238031  248194 1319977
```

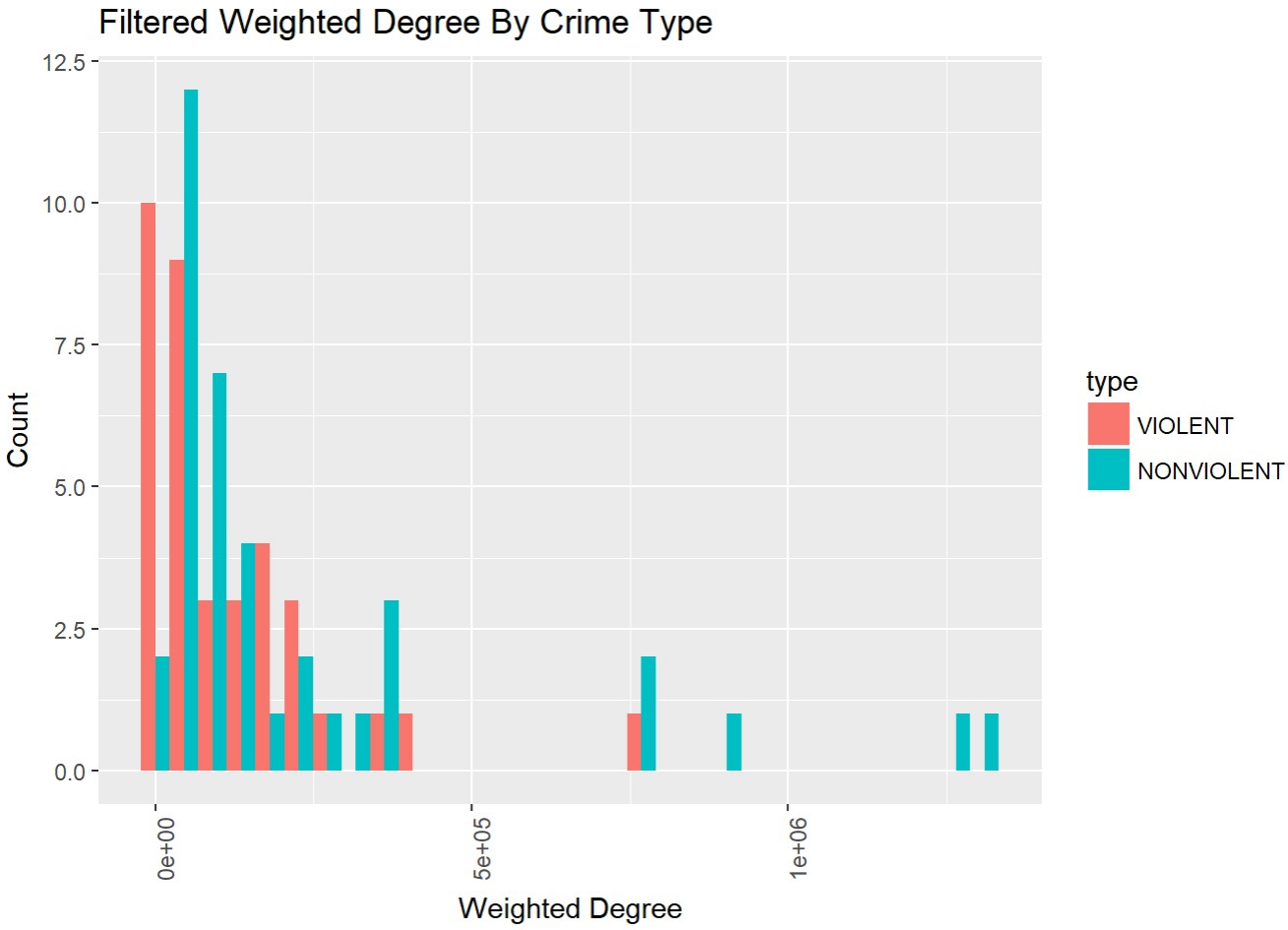
```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

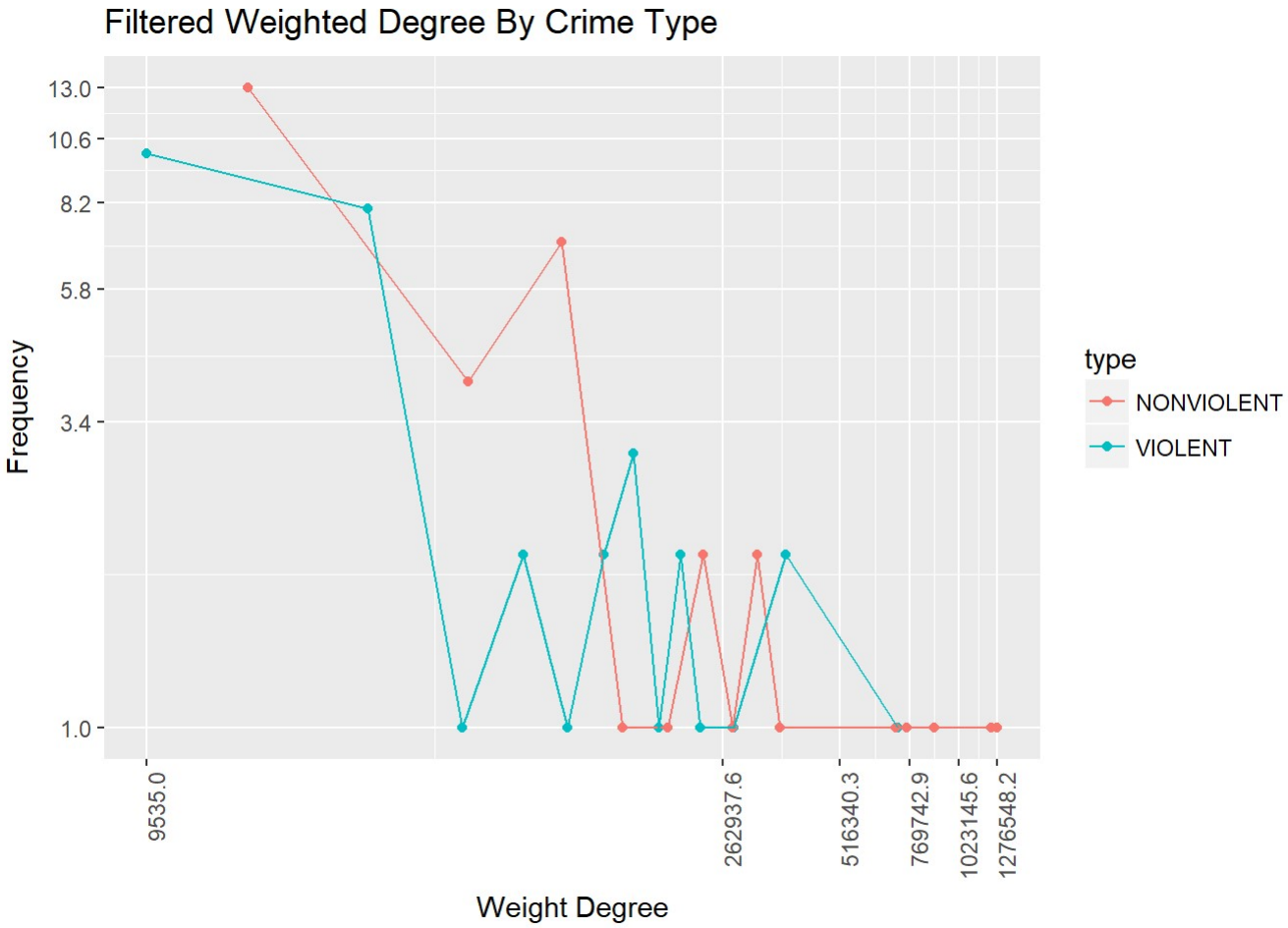
```
## [1]
```

```
# Combined Weighted Degree Distribution Histogram
degree_df = data.frame(type = rep("VIOLENT", length(violent_filter_wdeg)), degree = v
iolent_filter_wdeg)
degree_df = rbind(degree_df, data.frame(type = rep("NONVIOLENT", length(nonviolent_fi
lter_wdeg)),
                                        degree = nonviolent_filter_wdeg))

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Hist_WDegree_Filtered.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_hist = ggplot(data = degree_df, aes(x = degree, fill = type)) +
  geom_histogram(position = "dodge", bins = 30) +
  ggtitle("Filtered Weighted Degree By Crime Type") +
  labs(x = "Weighted Degree", y = "Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_hist)
```


Filtered Weighted Degree By Crime Type
12.5
10.0
7.5
5.0
2.5
0.0
Count
0e+00
5e+05
1e+06
Weighted Degree
type
VIOLENT
NONVIOLENT

```
if (save_to_file){
  dev.off()
}

# Combined Weighted Degree Log-Log Distribution
v_num_wdeg_bins = 30
v_min_wdeg = min(violent_filter_wdeg)
v_max_wdeg = max(violent_filter_wdeg)
v_step_wdeg = (v_max_wdeg - v_min_wdeg) / v_num_wdeg_bins

v_wdeg_breaks = seq(v_min_wdeg, v_max_wdeg, v_step_wdeg)
wdeg_bins = cut(violent_filter_wdeg, breaks = v_wdeg_breaks, labels=FALSE)
v_tab_wdeg = data.frame(tabulate(wdeg_bins))

v_tab_wdeg$wdeg <- v_wdeg_breaks[1:(length(v_wdeg_breaks)-1)]
v_tab_wdeg = v_tab_wdeg[v_tab_wdeg$tabulate.wdeg_bins.>0,]
v_tab_wdeg$type = rep("VIOLENT", length(v_tab_wdeg$wdeg))

nv_num_wdeg_bins = 30
nv_min_wdeg = min(nonviolent_filter_wdeg)
nv_max_wdeg = max(nonviolent_filter_wdeg)
nv_step_wdeg = (nv_max_wdeg - nv_min_wdeg) / nv_num_wdeg_bins

nv_wdeg_breaks = seq(nv_min_wdeg, nv_max_wdeg, nv_step_wdeg)
wdeg_bins = cut(nonviolent_filter_wdeg, breaks = nv_wdeg_breaks, labels=FALSE)
nv_tab_wdeg = data.frame(tabulate(wdeg_bins))

nv_tab_wdeg$wdeg <- nv_wdeg_breaks[1:(length(nv_wdeg_breaks)-1)]
nv_tab_wdeg = nv_tab_wdeg[nv_tab_wdeg$tabulate.wdeg_bins.>0,]
nv_tab_wdeg$type = rep("NONVIOLENT", length(nv_tab_wdeg$wdeg))

log_weight_df = rbind(v_tab_wdeg,nv_tab_wdeg)

x_num_breaks = 5
x_min_wdeg = min(log_weight_df$wdeg)
x_max_wdeg = max(log_weight_df$wdeg)
x_step_wdeg = (x_max_wdeg - x_min_wdeg) / x_num_breaks

y_num_breaks = 5
y_min_wdeg = min(log_weight_df$tabulate.wdeg_bins.)
y_max_wdeg = max(log_weight_df$tabulate.wdeg_bins.)
y_step_wdeg = (y_max_wdeg - y_min_wdeg) / y_num_breaks

if (save_to_file){
  temp_jpg = paste(output_path, "Combined_Log_WDeg_Filtered.jpg", sep = "")
  jpeg(file = temp_jpg)
}

g_combined_log = ggplot(log_weight_df, aes(x=wdeg, y=tabulate.wdeg_bins., color = typ
e)) +
  geom_point() + geom_line() +
  scale_x_log10(name="Weight Degree", breaks = seq(x_min_wdeg, x_max_wdeg, x_step_wde
g)) +
```

```
  scale_y_log10("Frequency", breaks = seq(y_min_wdeg, y_max_wdeg, y_step_wdeg)) +
  ggtitle("Filtered Weighted Degree By Crime Type") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(g_combined_log)
```




```
if (save_to_file){
  dev.off()
}
```

Decomposition and summary of filtered networks

```
# Calculate graph densities
violent_density = edge_density(violent_filter)
print(paste("Filtered Violent Graph Density:", violent_density))
```

```
## [1] "Filtered Violent Graph Density: 0.233333333333333"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
nonviolent_density = edge_density(nonviolent_filter)
print(paste("Filtered NonViolent Graph Density:", nonviolent_density))
```

```
## [1] "Filtered NonViolent Graph Density: 0.209103840682788"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
# See if there are any components
print("Filtered Violent Graph Components:")
```

```
## [1] "Filtered Violent Graph Components:"
```

```
violent_decomp = decompose(violent_filter)
v_comp_sizes = sapply(violent_decomp, vcount)
v_giant_ndx = which(v_comp_sizes == max(v_comp_sizes))
v_giant = violent_decomp[[v_giant_ndx]]
print(paste("Number of components:", length(v_comp_sizes)))
```

```
## [1] "Number of components: 1"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(paste("Number of nodes in Largest:", vcount(v_giant)))
```

```
## [1] "Number of nodes in Largest: 36"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(paste("Number of edges in Largest:", ecount(v_giant)))
```

```
## [1] "Number of edges in Largest: 147"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(violent_decomp)
```

```
## [[1]]
## IGRAPH 9ae4624 UNW- 36 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), wdegree
## | (v/n), weight (e/n)
## + edges from 9ae4624 (vertex names):
##  [1] 1 --25 1 --29 1 --43 2 --25 3 --25 3 --29 3 --43 6 --25 8 --23 8 --24
## [11] 8 --25 8 --27 8 --28 8 --29 8 --43 8 --44 8 --46 8 --49 8 --61 8 --66
## [21] 8 --67 8 --68 8 --69 8 --71 15--25 19--25 19--29 22--25 23--25 23--28
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("Filtered NonViolent Graph Components:")
```

```
## [1] "Filtered NonViolent Graph Components:"
```

```
nonviolent_decomp = decompose(nonviolent_filter)
nv_comp_sizes = sapply(nonviolent_decomp, vcount)
nv_giant_ndx = which(nv_comp_sizes == max(nv_comp_sizes))
nv_giant = nonviolent_decomp[[nv_giant_ndx]]
print(paste("Number of components:", length(nv_comp_sizes)))
```

```
## [1] "Number of components: 1"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(paste("Number of nodes in Largest:", vcount(nv_giant)))
```

```
## [1] "Number of nodes in Largest: 38"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(paste("Number of edges in Largest:", ecount(nv_giant)))
```

```
## [1] "Number of edges in Largest: 147"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print(nonviolent_decomp)
```

```
## [[1]]
## IGRAPH 9ae6c4a UNW- 38 147 --
## + attr: name (v/c), Label (v/c), Prop.Occupied (v/n), Prop.Rented
## | (v/n), Prop.Vacant (v/n), Prop.Owned (v/n), Median.Age (v/n),
## | Total.Population (v/n), Average.Household.Size (v/n),
## | Prop.African (v/n), Prop.White (v/n), Prop.Asian (v/n),
## | Prop.Hispanic (v/n), Community.Area (v/n), id (v/c), wdegree
## | (v/n), weight (e/n)
## + edges from 9ae6c4a (vertex names):
##  [1] 1--8  1--24 1--25 1--28 1--32 2--8  2--32 3--8  3--24 3--28 3--32
## [12] 4--8  4--32 6--7  6--8  6--22 6--23 6--24 6--25 6--28 6--32 6--43
## [23] 6--44 6--71 7--8  7--22 7--23 7--24 7--25 7--28 7--32 7--43 7--44
## + ... omitted several edges
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("Filtered Violent Edge Weight Summary:")
```

```
## [1] "Filtered Violent Edge Weight Summary:"
```

```
print(summary(E(violent_filter)$weight))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8958   10423   12384   14761   16615   44381
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
print("Filtered NonViolent Edge Weight Summary:")
```

```
## [1] "Filtered NonViolent Edge Weight Summary:"
```

```
print(summary(E(nonviolent_filter)$weight))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17114   19688   24488   30766   32861  138304
```

```
print("", quote=FALSE)
```

```
## [1]
```

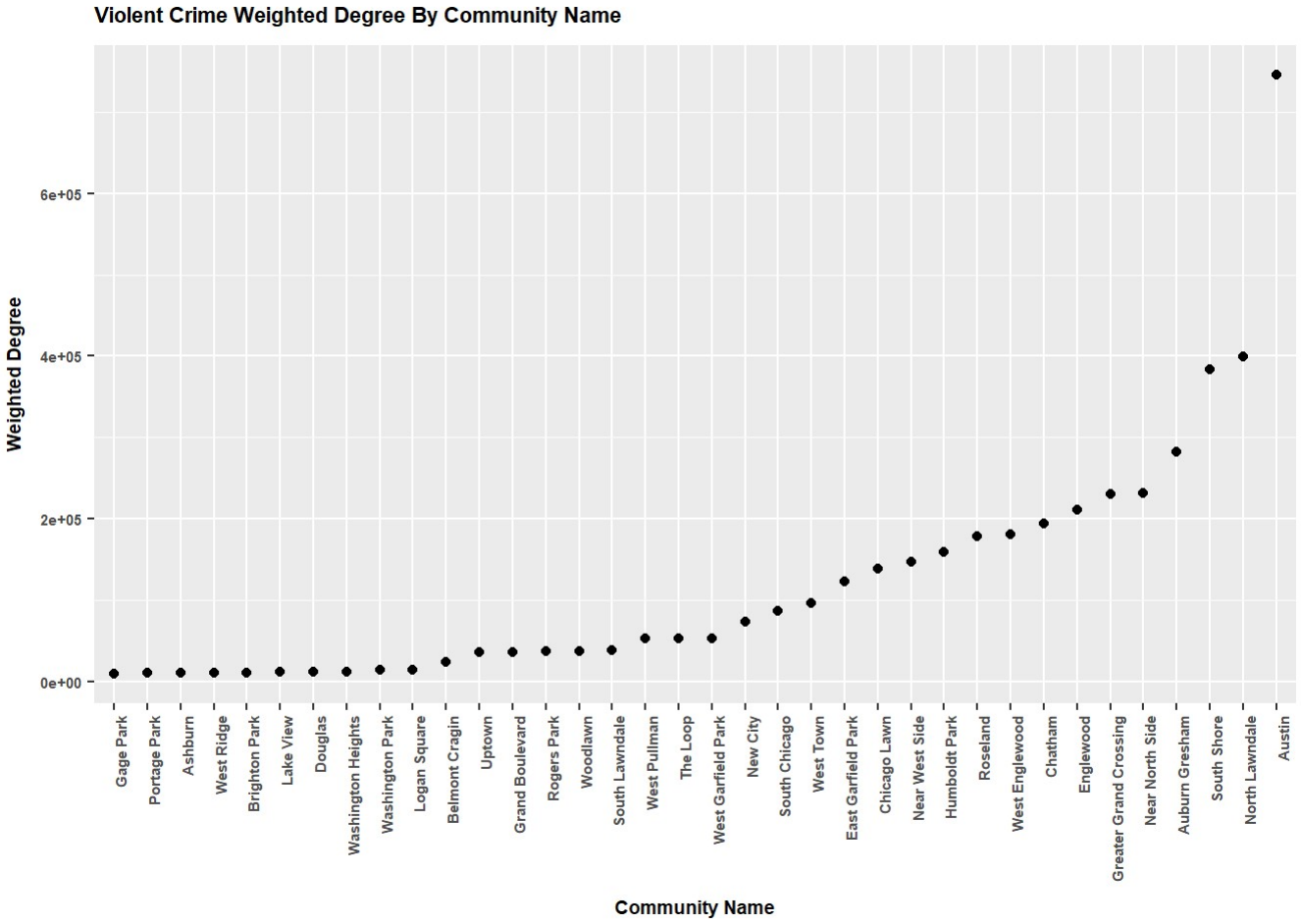
```
print("", quote=FALSE)
```

```
## [1]
```

```
violent_filter = v_giant

nonviolent_filter = nv_giant


if (save_to_file){
  sink()

  close(outFile)
  closeAllConnections()
}

###################### Assortativity ##################################################
#################
```

Violent crime projection assortativity analysis

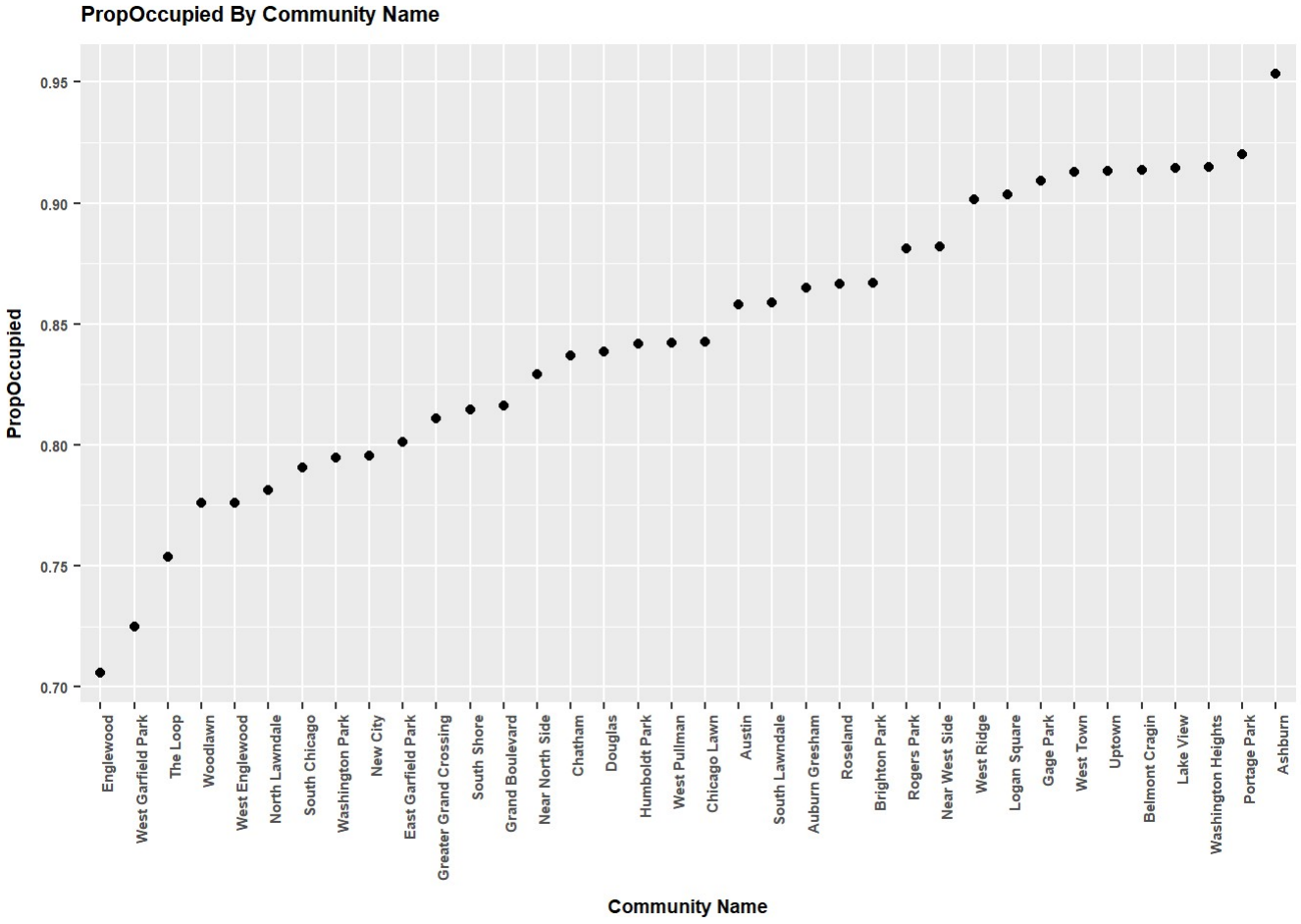
```
# Calculate assortativity of vertex attributes
dir.create(file.path(output_path, "Assortativity"), showWarnings = FALSE)
assort_path = paste(output_path, "Assortativity\\", sep = "")

dir.create(file.path(output_path, "Attributes"), showWarnings = FALSE)
attr_path = paste(output_path, "Attributes\\", sep = "")

if (save_to_file){
  tmp_scatter = paste(attr_path, "Violent_WeightedDegree_Vs_Community.jpg", sep = '')
  jpeg(file = tmp_scatter)
}

temp_df = data.frame(Community = V(violent_filter)$Label, WDegree = V(violent_filter)
$wdegree)
temp_df = temp_df[order(temp_df$WDegree),]
tmp_g = ggplot(data = temp_df, aes(x = reorder(Community, WDegree), y = WDegree)) +
  geom_point() +
  labs(x = "Community Name", y = "Weighted Degree") +
  ggtitle("Violent Crime Weighted Degree By Community Name") +
  theme(text = element_text(size=7, face="bold"), axis.text.x = element_text(angle =
90, hjust = 1))
print(tmp_g)
```



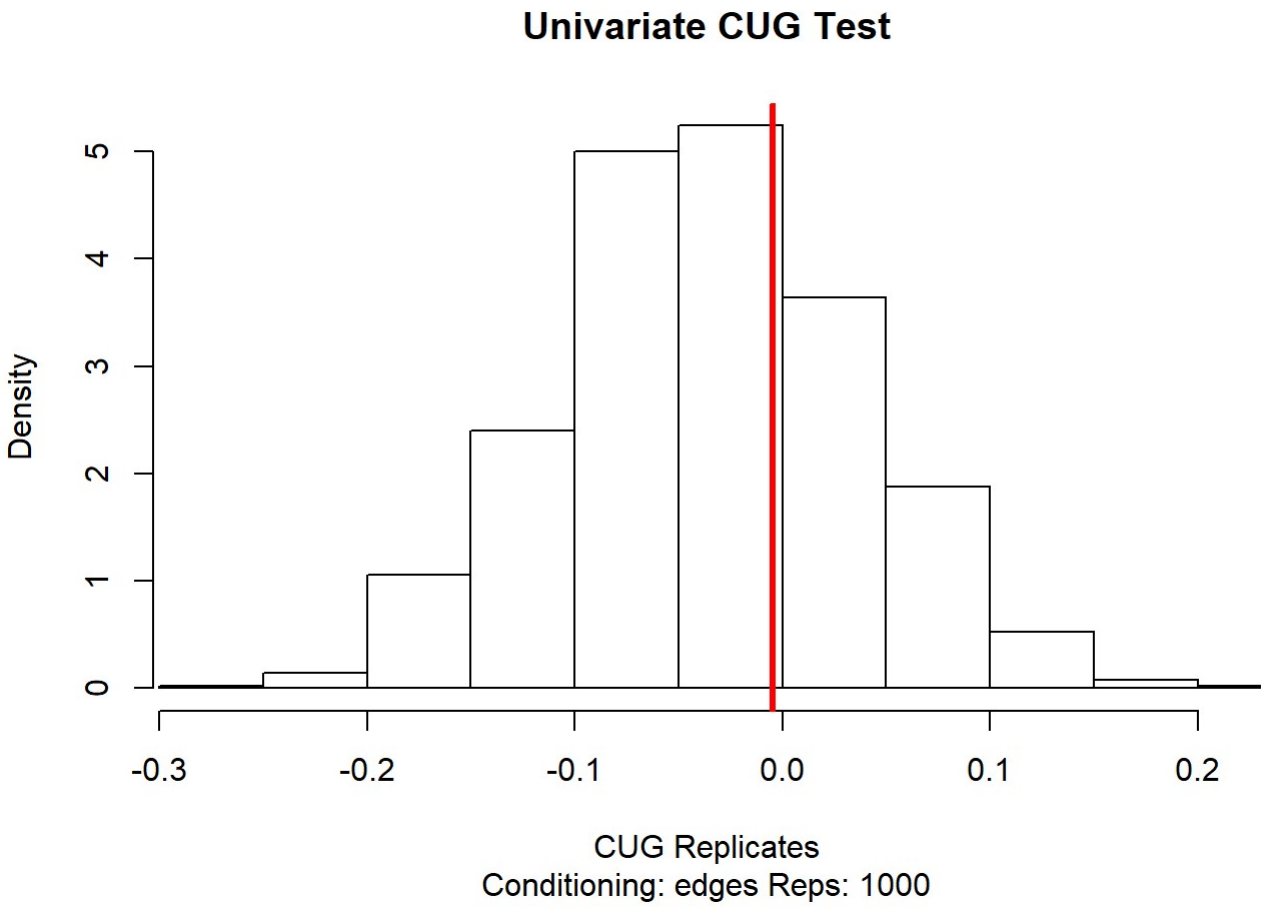
```
if (save_to_file){
  dev.off()
}

if (save_to_file){
  out_file_name = paste(assort_path, "Violent_Assortativity.txt", sep = '')
  outFile = file(out_file_name, open="wt")
  sink(file = outFile, append = TRUE)
}

attrs_skip = c("name", "Label", "id", "wdegree", "Community.Area")

violent_attr = list.vertex.attributes(violent_filter)

for ( i in 1:length(violent_attr)){
  temp_attr = violent_attr[i]

  if (!(temp_attr %in% attrs_skip)){

    print("-------------------------------------------------------------------------
----")
    print(paste("Testing Assortativity for attribute:", temp_attr))
    print("", quote=FALSE)

    temp_var = paste("V(violent_filter)$", temp_attr, sep = "")
    vertex_vals = eval(parse(text = temp_var))
    print("Vertex Values:")
    print(vertex_vals)
    print("", quote=FALSE)
    print("Vertex Value Summary:")
    print(summary(vertex_vals))
    print("", quote=FALSE)

    temp_attr_jpg =  sub("\\.", "", temp_attr)

    if (save_to_file){
      tmp_scatter = paste(attr_path, "Violent_", temp_attr_jpg, '_Vs_Community.jpg',
sep = '')
      jpeg(file = tmp_scatter)
    }

    temp_df = data.frame(Community = V(violent_filter)$Label, temp_attr_jpg = vertex_
vals)
    temp_df = temp_df[order(vertex_vals),]
    tmp_g = ggplot(data = temp_df, aes(x = reorder(Community, temp_attr_jpg), y = tem
p_attr_jpg)) +
      geom_point() +
      labs(x = "Community Name", y = temp_attr_jpg) +
      ggtitle(paste(temp_attr_jpg, "By Community Name")) +
      theme(text = element_text(size=7, face="bold"), axis.text.x = element_text(angl
e = 90, hjust = 1))
    print(tmp_g)
    if (save_to_file){
```

```
      dev.off()
    }

    temp_assort = assortativity(violent_filter, types1 = vertex_vals,
                                directed = FALSE)
    temp_cug = mycugtest(violent_filter, assortativity, cmode = "edges", types1 = ver
tex_vals,
                         directed = FALSE)
    temp_qap = myqaptest(violent_filter, assortativity, types1 = vertex_vals,
                         directed = FALSE)

    print(paste("Assortativity:", temp_assort))
    print("", quote=FALSE)

    print("Assortativity CUG Test Results:")
    print.cug.test(temp_cug)
    print("", quote=FALSE)

    if (save_to_file){
      tmp_cug = paste(assort_path, "Violent_", temp_attr_jpg, '_CUG.jpg', sep = '')
      jpeg(file = tmp_cug)
    }
    plot.cug.test(temp_cug)
    if (save_to_file){
      dev.off()
    }

    print("Assortativity QAP Test Results:")
    print(summary.qaptest(temp_qap))
    print("", quote=FALSE)

    if (save_to_file){
      tmp_qap = paste(assort_path, "Violent_", temp_attr_jpg, '_QAP.jpg', sep = '')
      jpeg(file = tmp_qap)
    }
    plot.qaptest(temp_qap)
    if (save_to_file){
      dev.off()
    }
  }

}
```

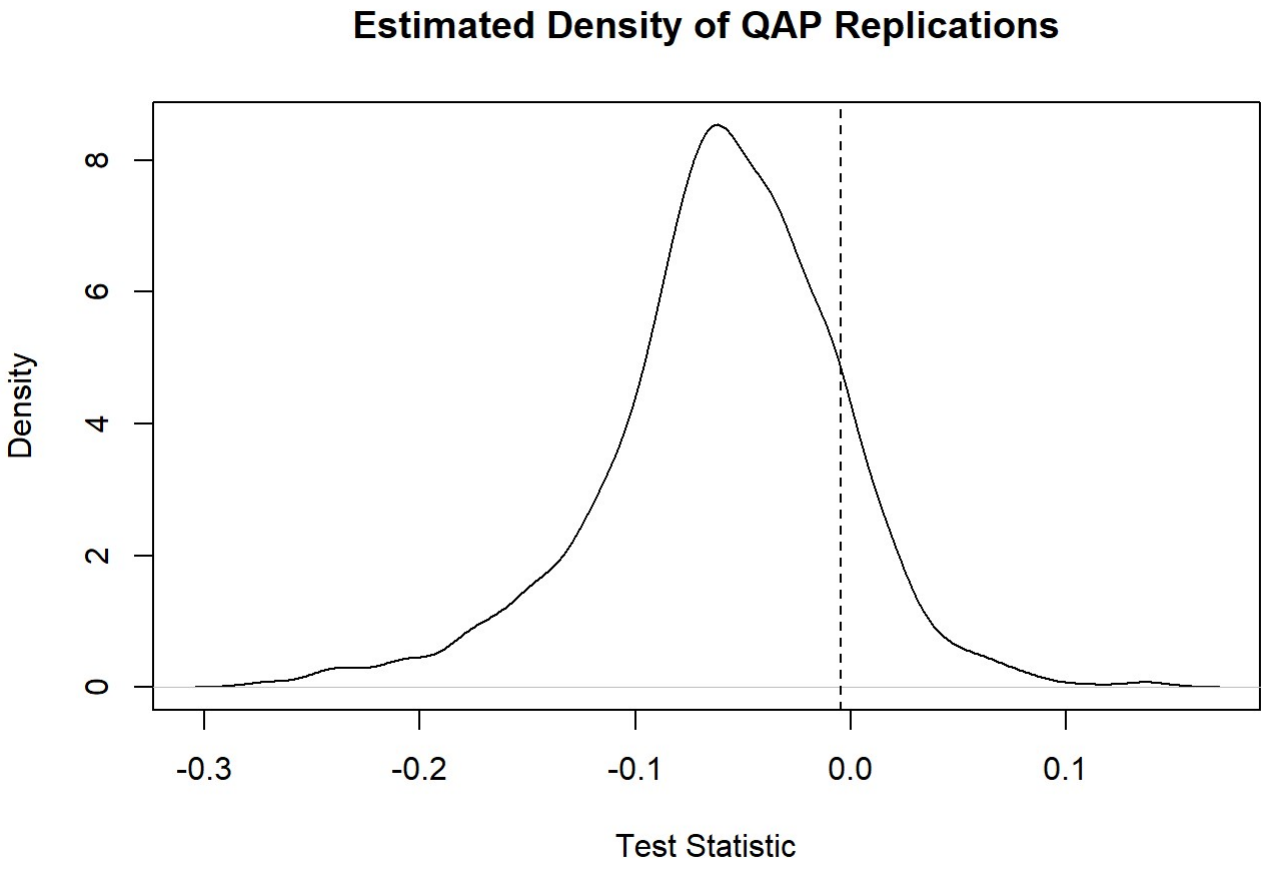
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Occupied"
## [1]
## [1] "Vertex Values:"
##  [1] 0.8812944 0.9012961 0.9132786 0.9142603 0.8292284 0.9201469 0.9135567
##  [8] 0.9035277 0.8416051 0.9126538 0.8581148 0.7249506 0.8010041 0.8820984
## [15] 0.7811182 0.8589634 0.7537092 0.8385105 0.8163131 0.7945005 0.7759617
## [22] 0.8145812 0.8370162 0.7904551 0.8666853 0.8420866 0.8667852 0.7952835
## [29] 0.9090381 0.8424216 0.7761138 0.7058824 0.8110459 0.9533691 0.8649698
## [36] 0.9148095
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.7059  0.7996  0.8423  0.8446  0.9019  0.9534
## [1]
```



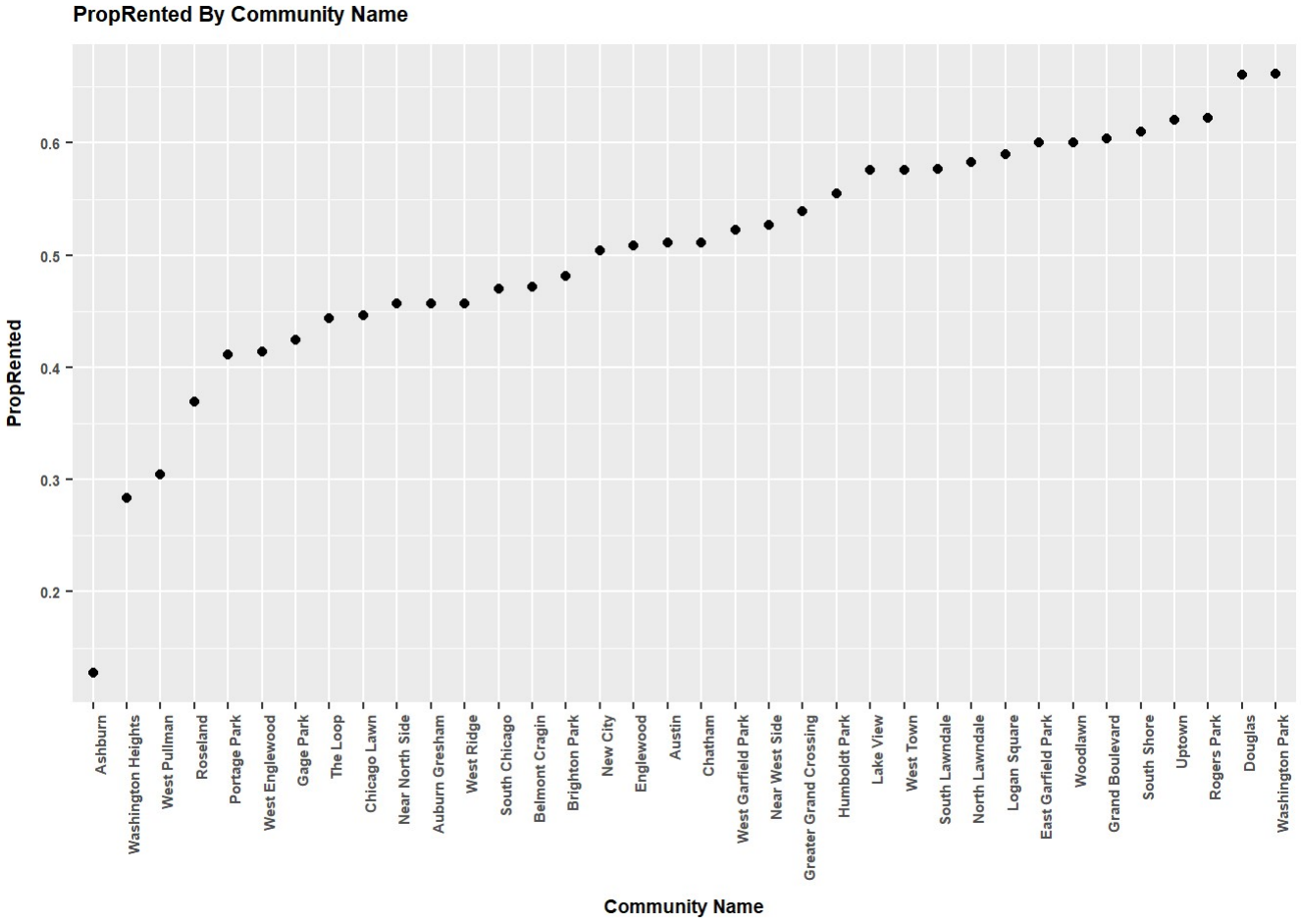
```
## [1] "Assortativity: -0.0046093819559339"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.004609382
## Pr(X>=Obs): 0.331
## Pr(X<=Obs): 0.669
##
## [1]
```



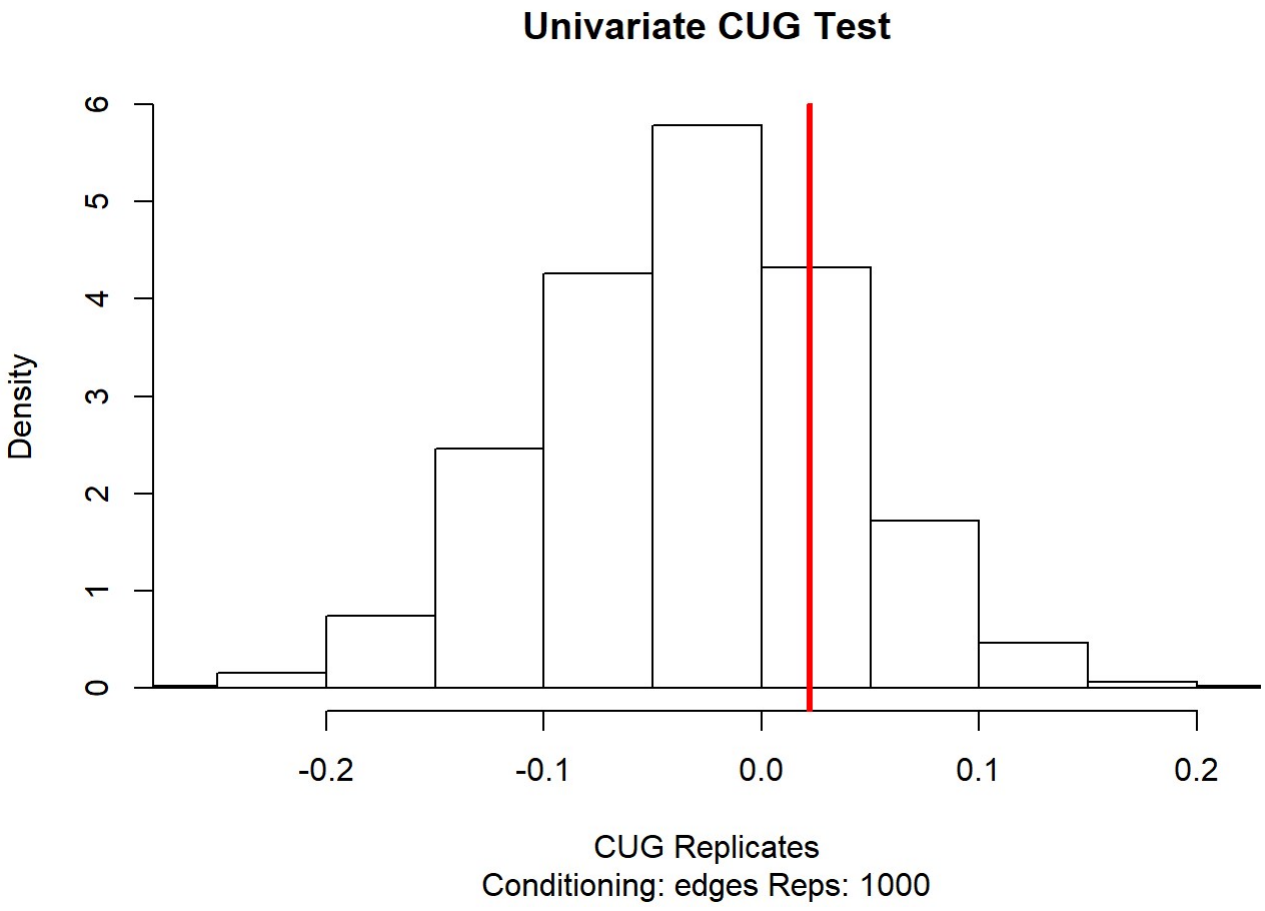
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.133
##  p(f(perm) <= f(d)): 0.867
##
## Test Diagnostics:
##  Test Value (f(d)): -0.004609382
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2729172
##      1stQ:    -0.08769985
##      Med:     -0.05689482
##      Mean:    -0.05981408
##      3rdQ:    -0.02460661
##      Max:     0.1393095
##
## [1]
```



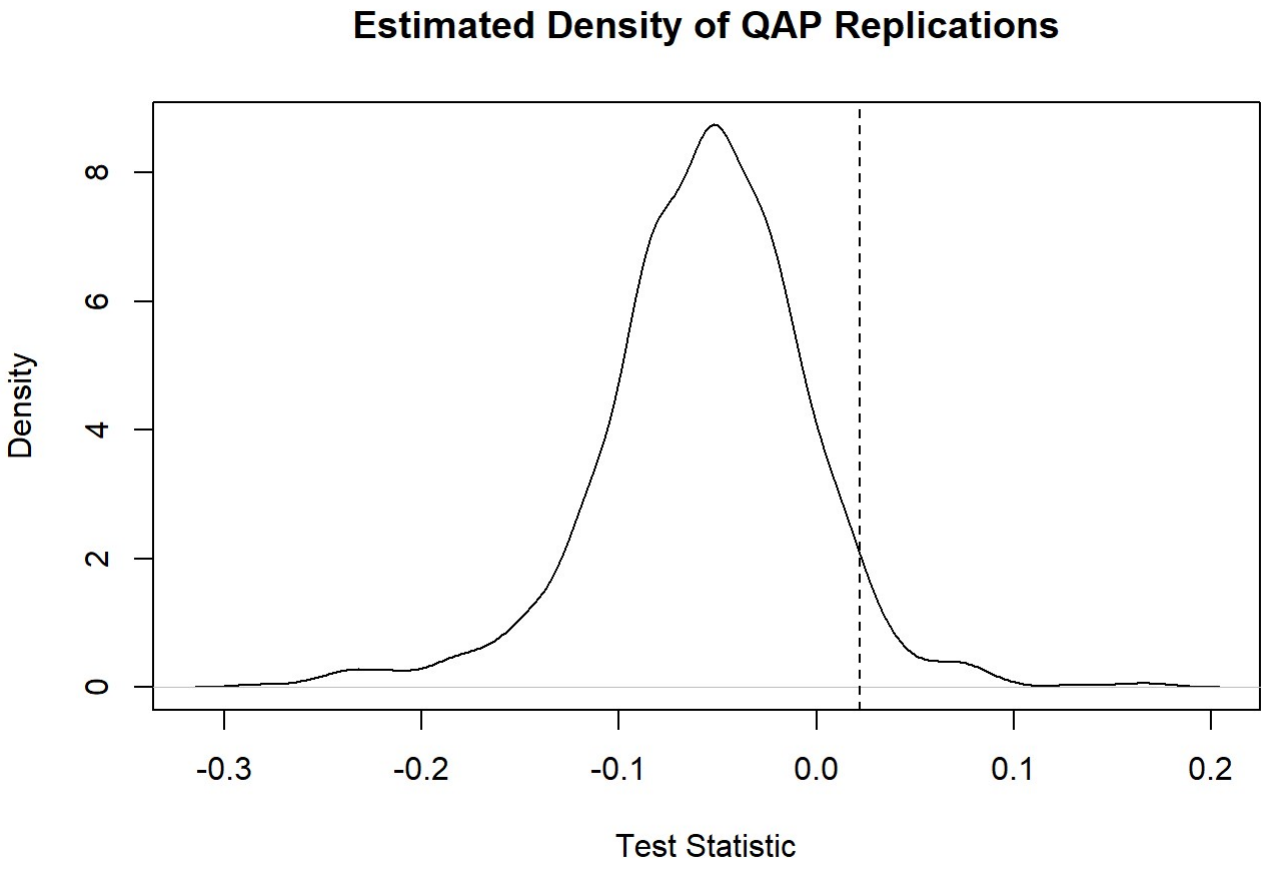
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Rented"
## [1]
## [1] "Vertex Values:"
##  [1] 0.6223698 0.4572562 0.6204788 0.5761394 0.4567699 0.4111689 0.4716087
##  [8] 0.5896078 0.5550468 0.5763413 0.5112786 0.5227048 0.6002967 0.5267165
## [15] 0.5826418 0.5766060 0.4438877 0.6610485 0.6039792 0.6615949 0.6002972
## [22] 0.6104102 0.5115371 0.4702436 0.3694545 0.3041164 0.4816501 0.5044531
## [29] 0.4243921 0.4465262 0.4137237 0.5086100 0.5393439 0.1280245 0.4569800
## [36] 0.2831645
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1280  0.4542  0.5114  0.5022  0.5844  0.6616
## [1]
```



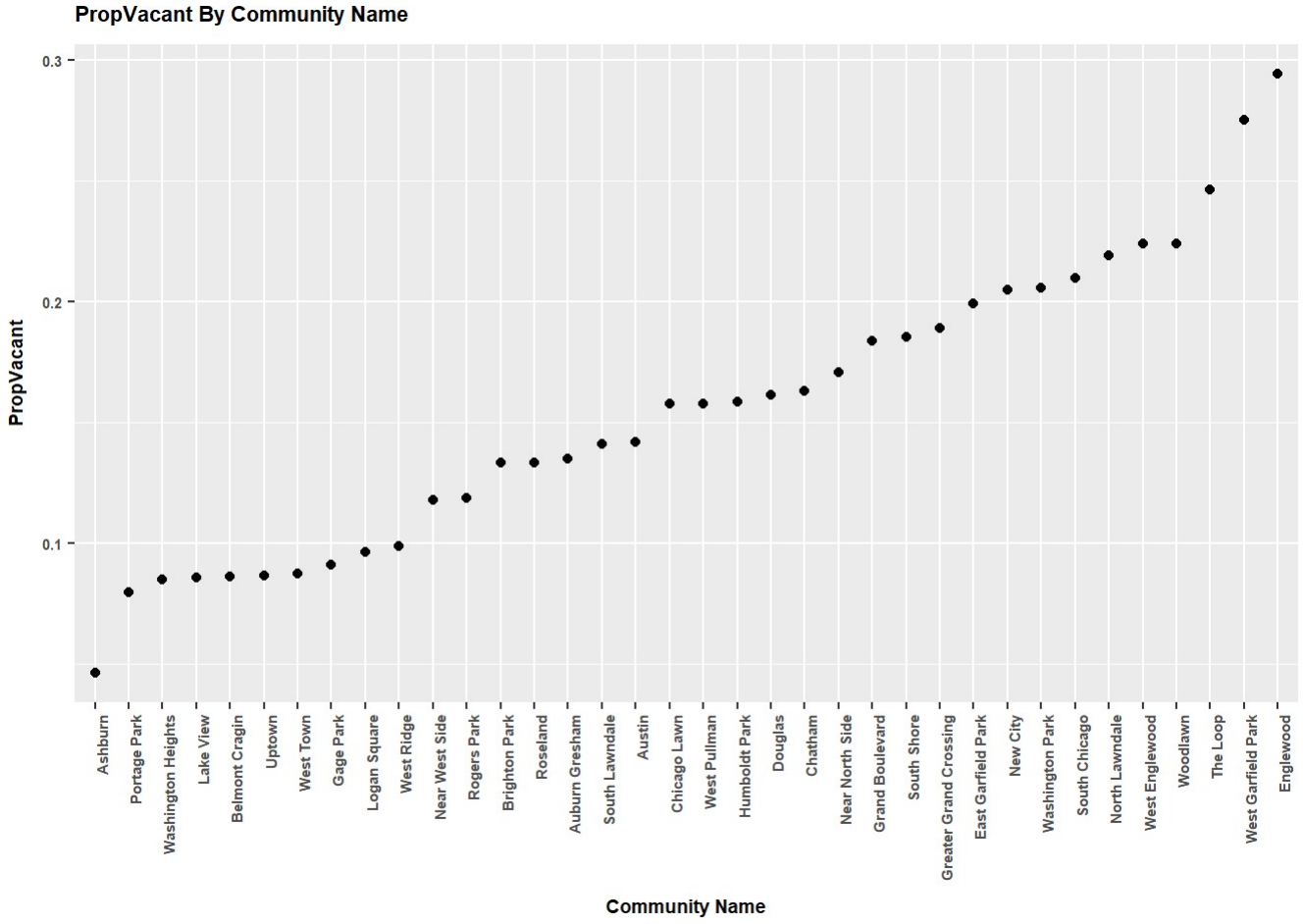
```
## [1] "Assortativity: 0.0222093849872616"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: 0.02220938
## Pr(X>=Obs): 0.219
## Pr(X<=Obs): 0.781
##
## [1]
```



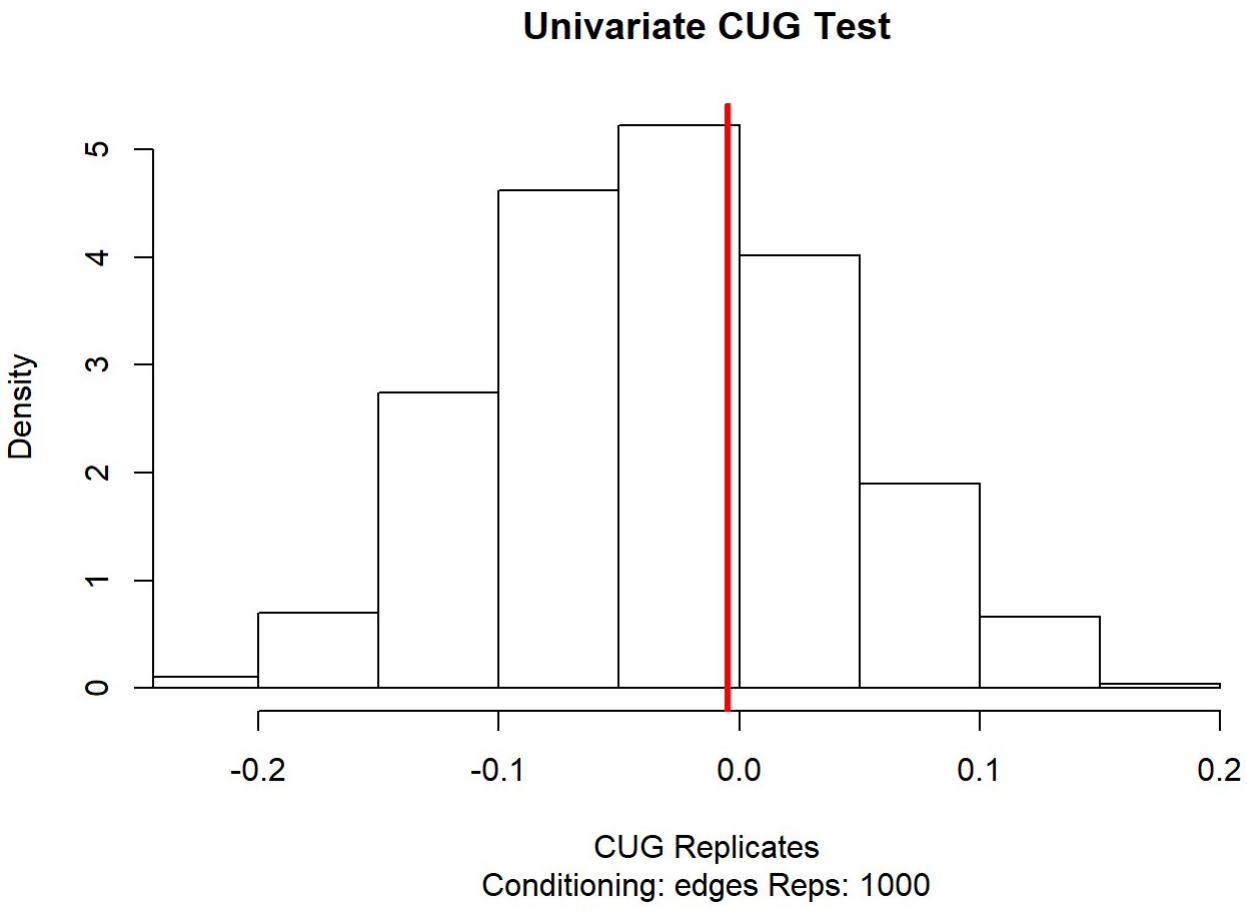
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.044
##  p(f(perm) <= f(d)): 0.956
##
## Test Diagnostics:
##  Test Value (f(d)): 0.02220938
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2841938
##      1stQ:    -0.08538981
##      Med:     -0.05374869
##      Mean:    -0.05670522
##      3rdQ:    -0.02416217
##      Max:     0.1729865
##
## [1]
```



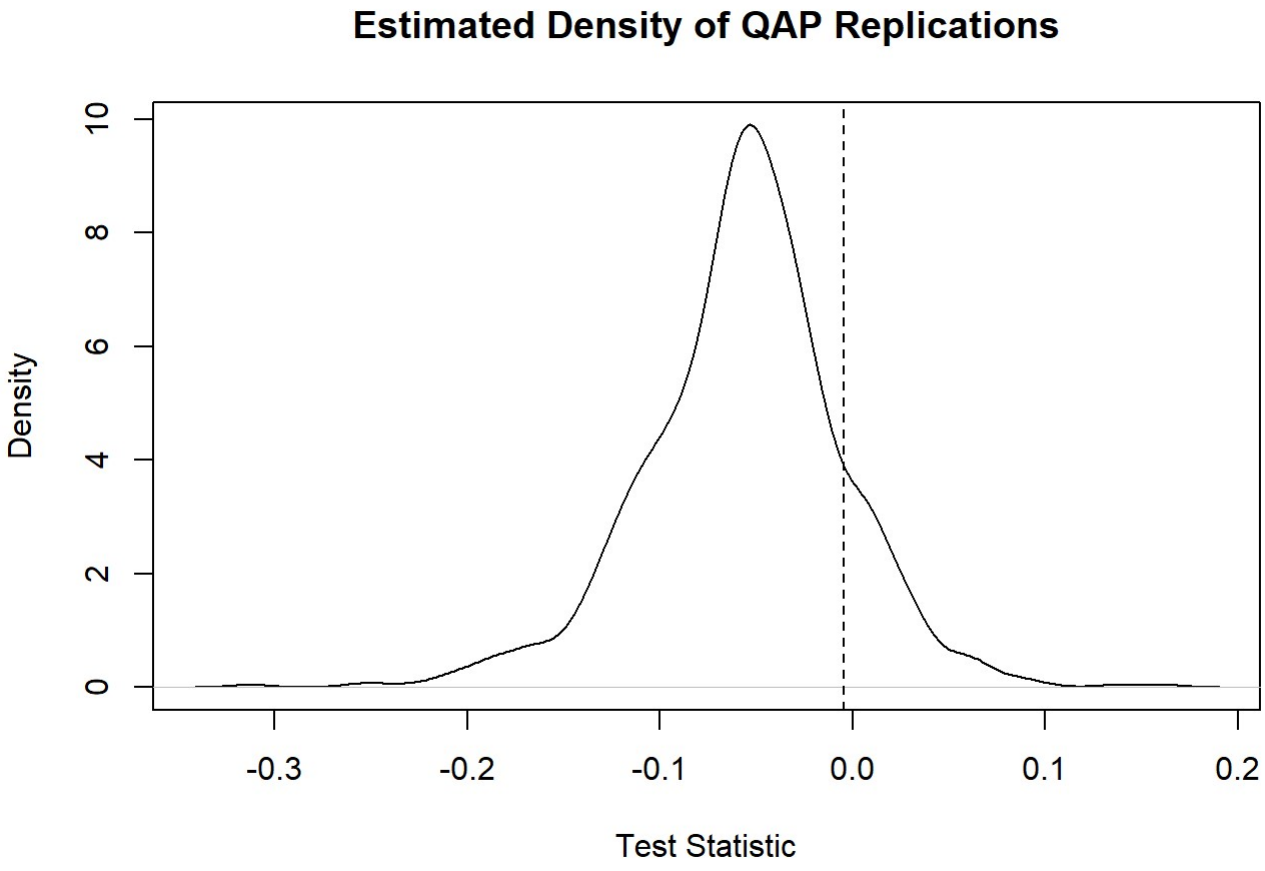
```
## [1] "-------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Vacant"
## [1]
## [1] "Vertex Values:"
##  [1] 0.11870556 0.09870385 0.08672137 0.08573972 0.17077160 0.07985313
##  [7] 0.08644326 0.09647233 0.15839488 0.08734616 0.14188517 0.27504936
## [13] 0.19899589 0.11790159 0.21888178 0.14103657 0.24629080 0.16148947
## [19] 0.18368695 0.20549954 0.22403830 0.18541882 0.16298376 0.20954490
## [25] 0.13331469 0.15791341 0.13321479 0.20471651 0.09096193 0.15757838
## [31] 0.22388616 0.29411765 0.18895405 0.04663093 0.13503022 0.08519055
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.04663 0.09815 0.15775 0.15537 0.20043 0.29412
## [1]
```



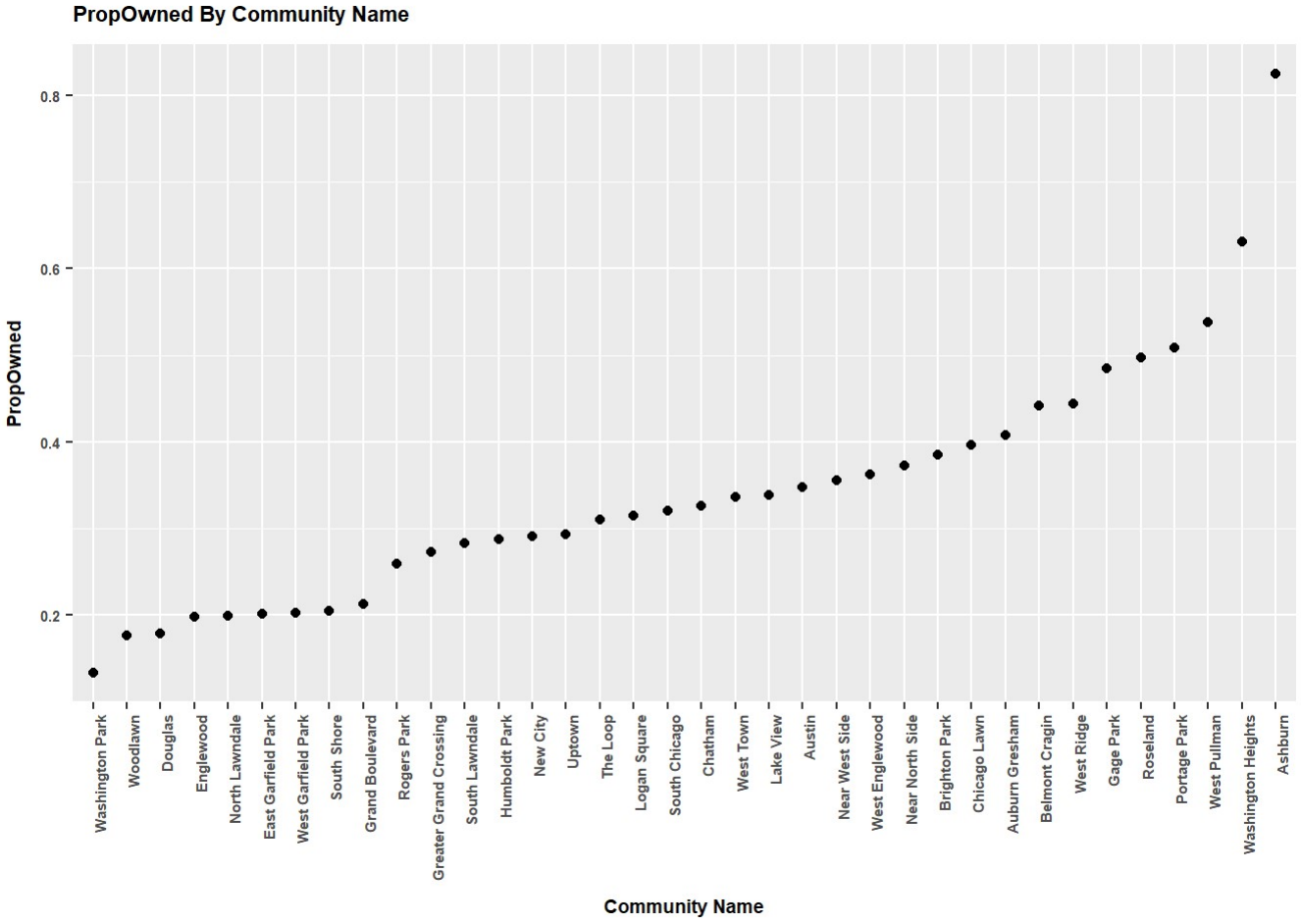
```
## [1] "Assortativity: -0.00460938195589999"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.004609382
## Pr(X>=Obs): 0.346
## Pr(X<=Obs): 0.654
##
## [1]
```



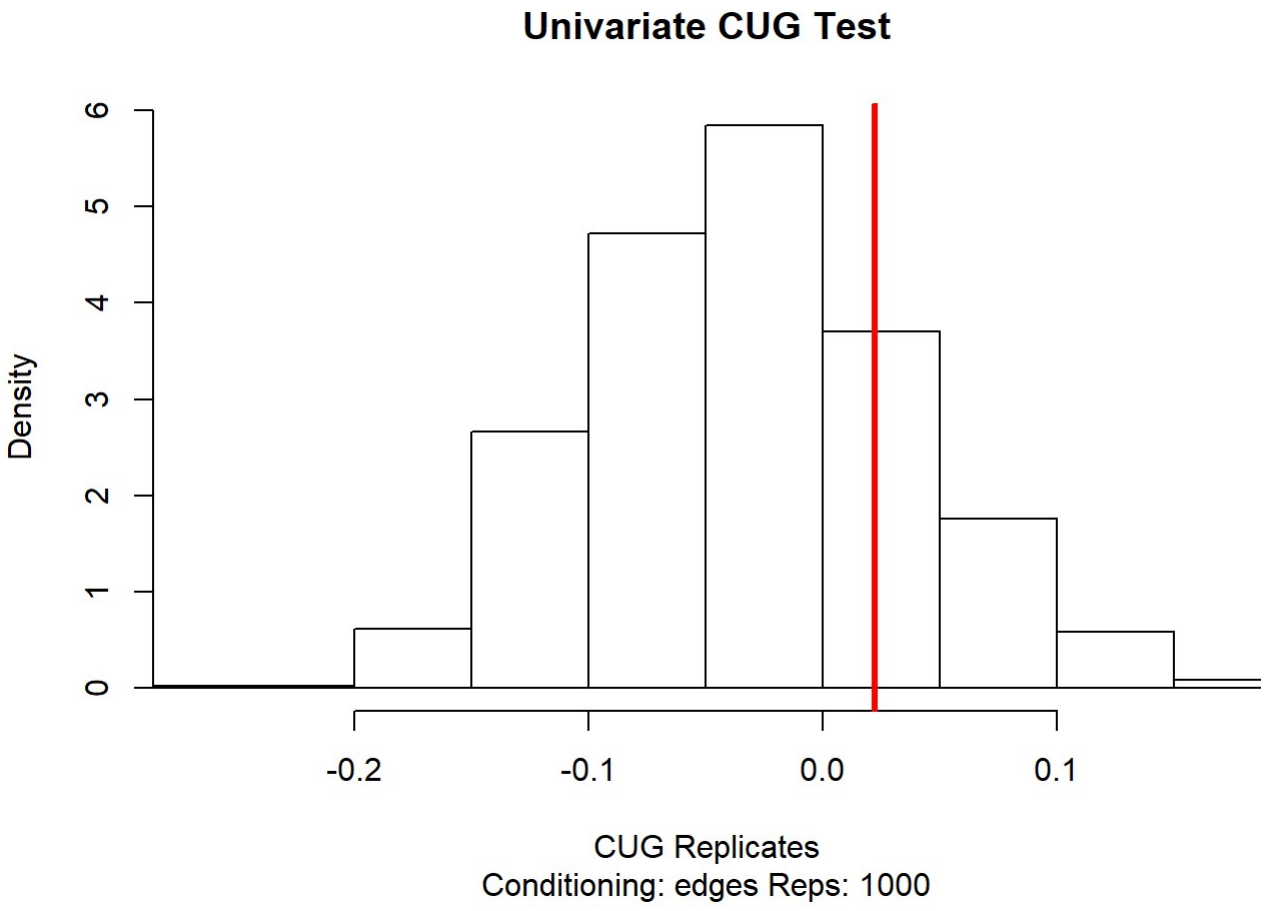
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.138
##  p(f(perm) <= f(d)): 0.862
##
## Test Diagnostics:
##  Test Value (f(d)): -0.004609382
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3122449
##      1stQ:    -0.08474909
##      Med:     -0.05402122
##      Mean:    -0.05631684
##      3rdQ:    -0.02666763
##      Max:     0.1607432
##
## [1]
```



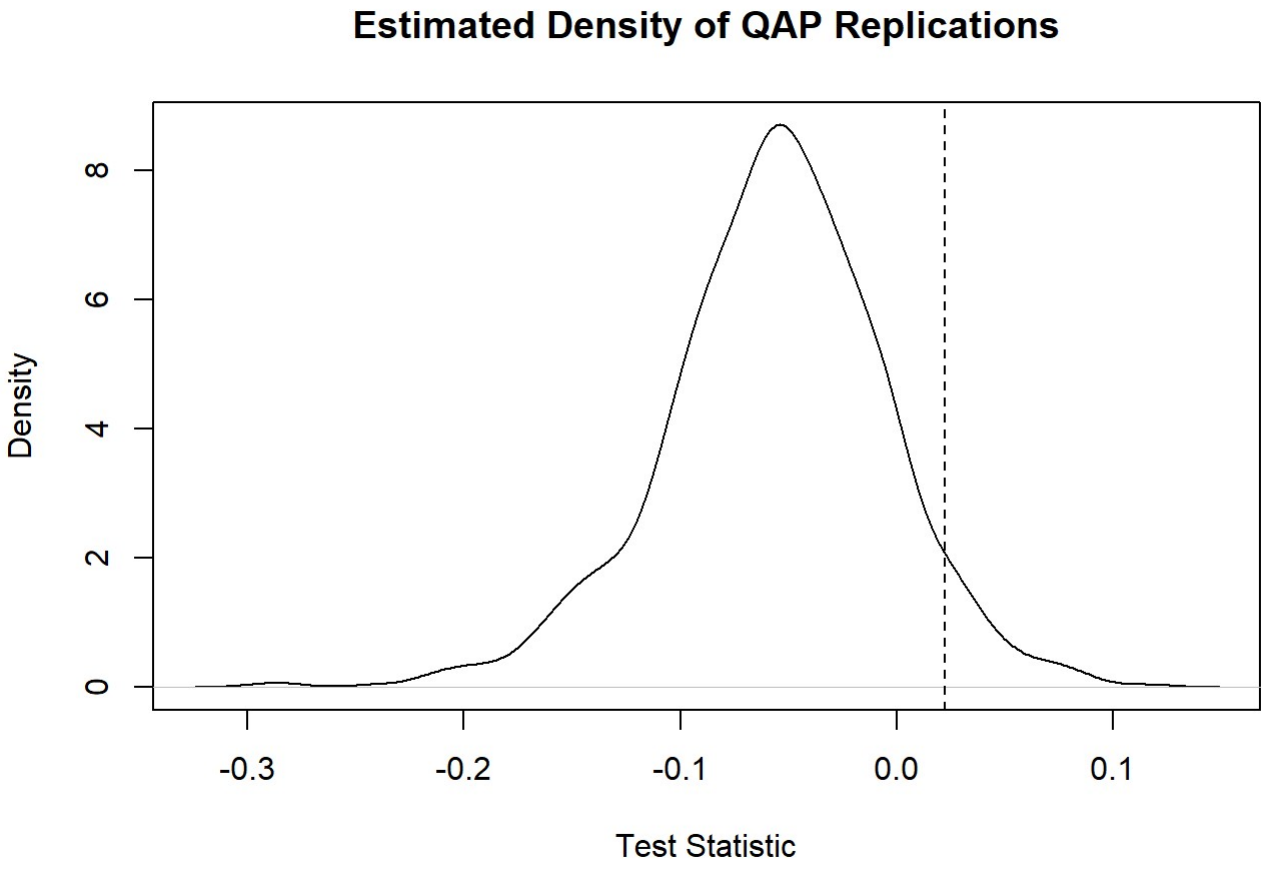
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Owned"
## [1]
## [1] "Vertex Values:"
##  [1] 0.2589247 0.4440399 0.2927999 0.3381209 0.3724585 0.5089779 0.4419480
##  [8] 0.3139198 0.2865583 0.3363126 0.3468362 0.2022458 0.2007074 0.3553819
## [15] 0.1984764 0.2823574 0.3098215 0.1774620 0.2123339 0.1329056 0.1756645
## [22] 0.2041710 0.3254792 0.3202115 0.4972308 0.5379702 0.3851351 0.2908304
## [29] 0.4846459 0.3958954 0.3623901 0.1972724 0.2717020 0.8253446 0.4079898
## [36] 0.6316450
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1329  0.2473  0.3228  0.3424  0.3989  0.8253
## [1]
```



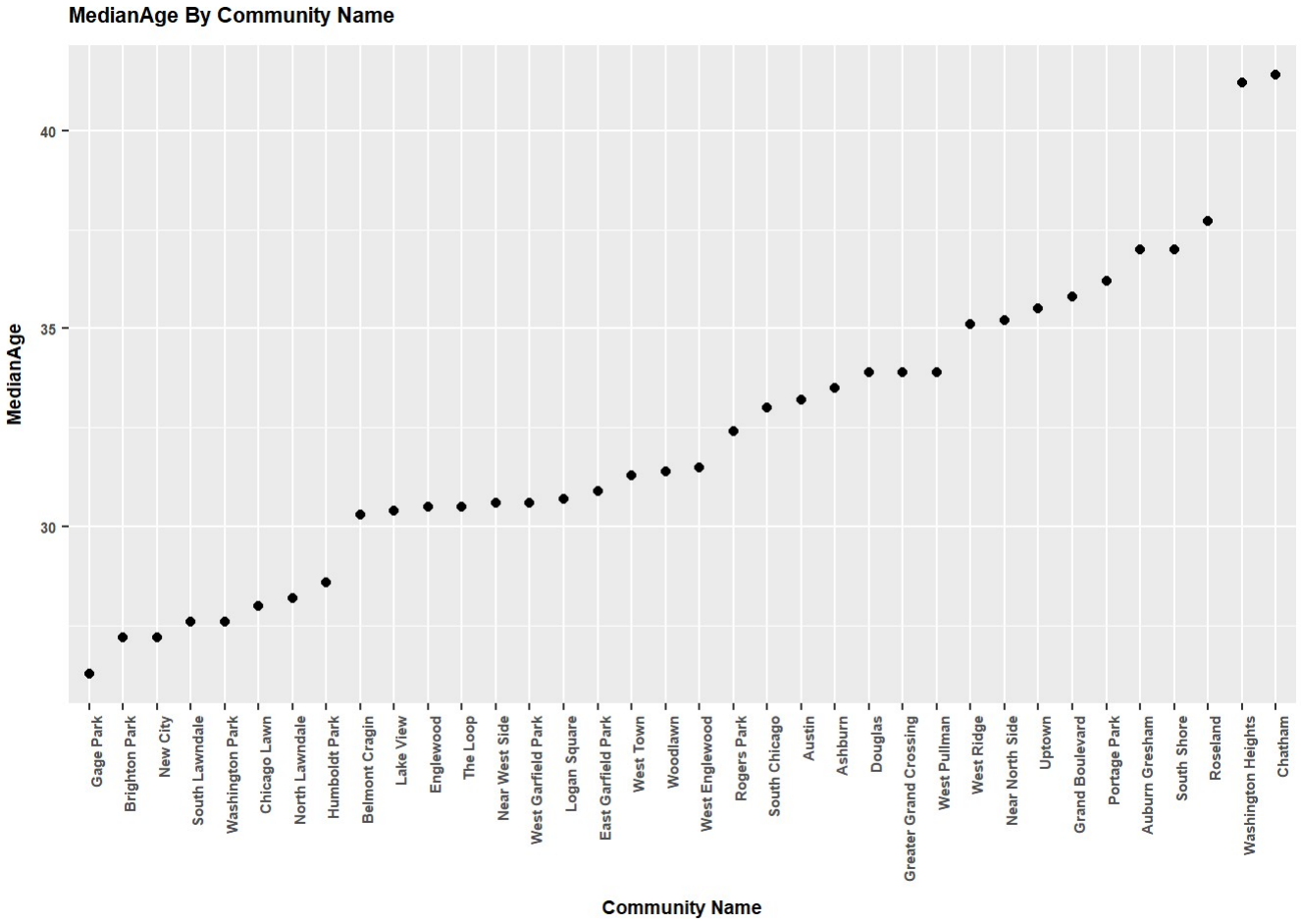
```
## [1] "Assortativity: 0.0221976069576776"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: 0.02219761
## Pr(X>=Obs): 0.215
## Pr(X<=Obs): 0.785
##
## [1]
```



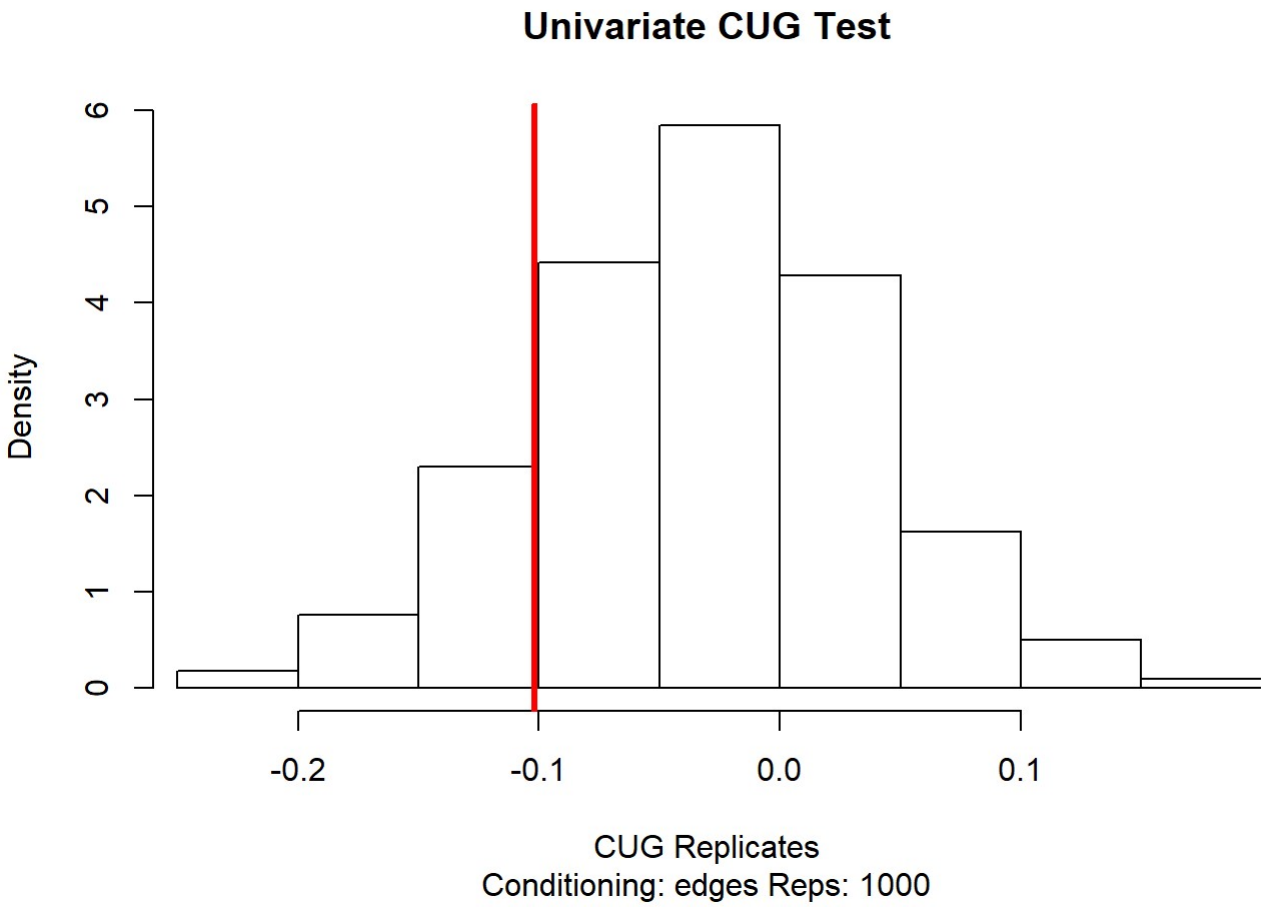
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.058
##  p(f(perm) <= f(d)): 0.942
##
## Test Diagnostics:
##  Test Value (f(d)): 0.02219761
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2924343
##      1stQ:    -0.08644056
##      Med:     -0.05467967
##      Mean:    -0.05639267
##      3rdQ:    -0.02331526
##      Max:     0.1168168
##
## [1]
```



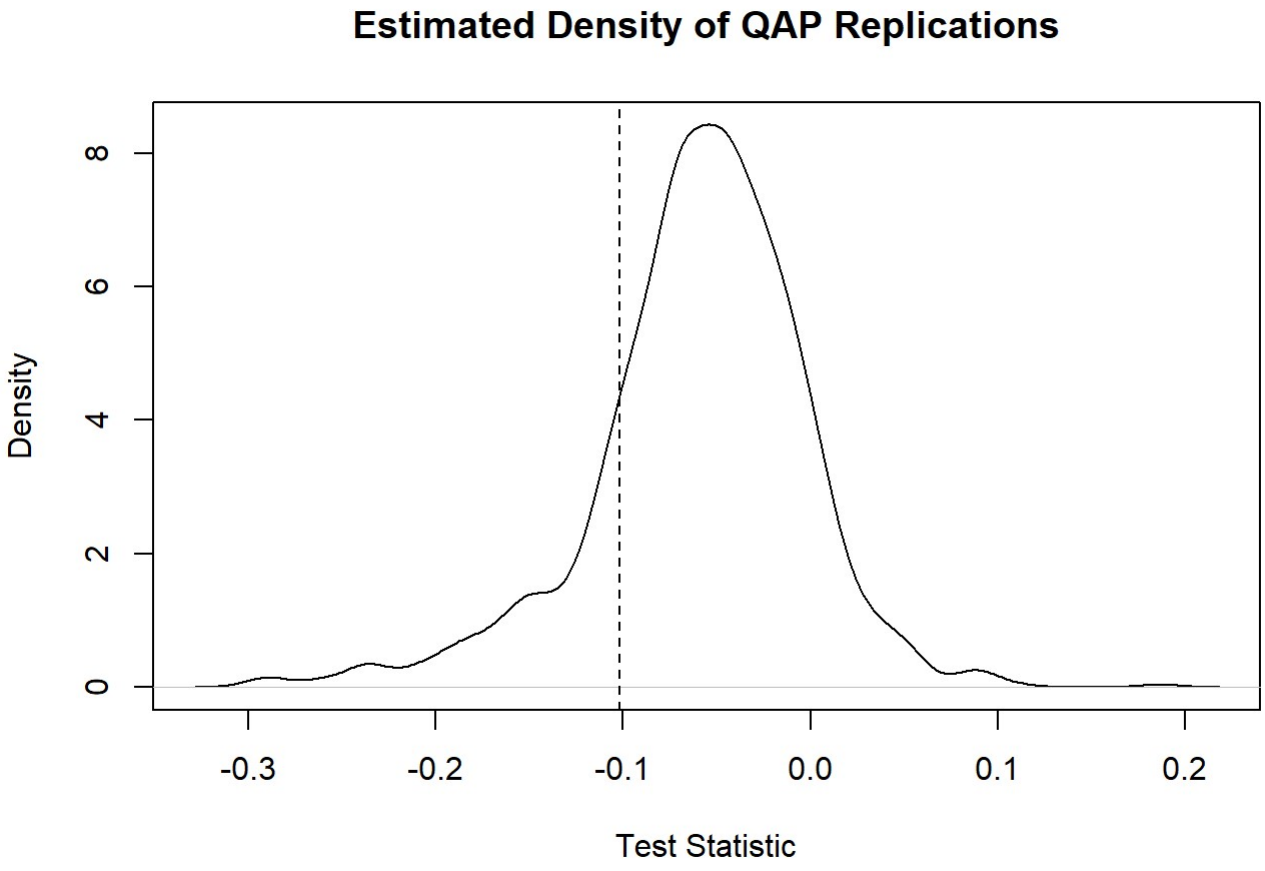
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Median.Age"
## [1]
## [1] "Vertex Values:"
##  [1] 32.4 35.1 35.5 30.4 35.2 36.2 30.3 30.7 28.6 31.3 33.2 30.6 30.9 30.6
## [15] 28.2 27.6 30.5 33.9 35.8 27.6 31.4 37.0 41.4 33.0 37.7 33.9 27.2 27.2
## [29] 26.3 28.0 31.5 30.5 33.9 33.5 37.0 41.2
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   26.30   30.38   31.45   32.37   35.12   41.40
## [1]
```



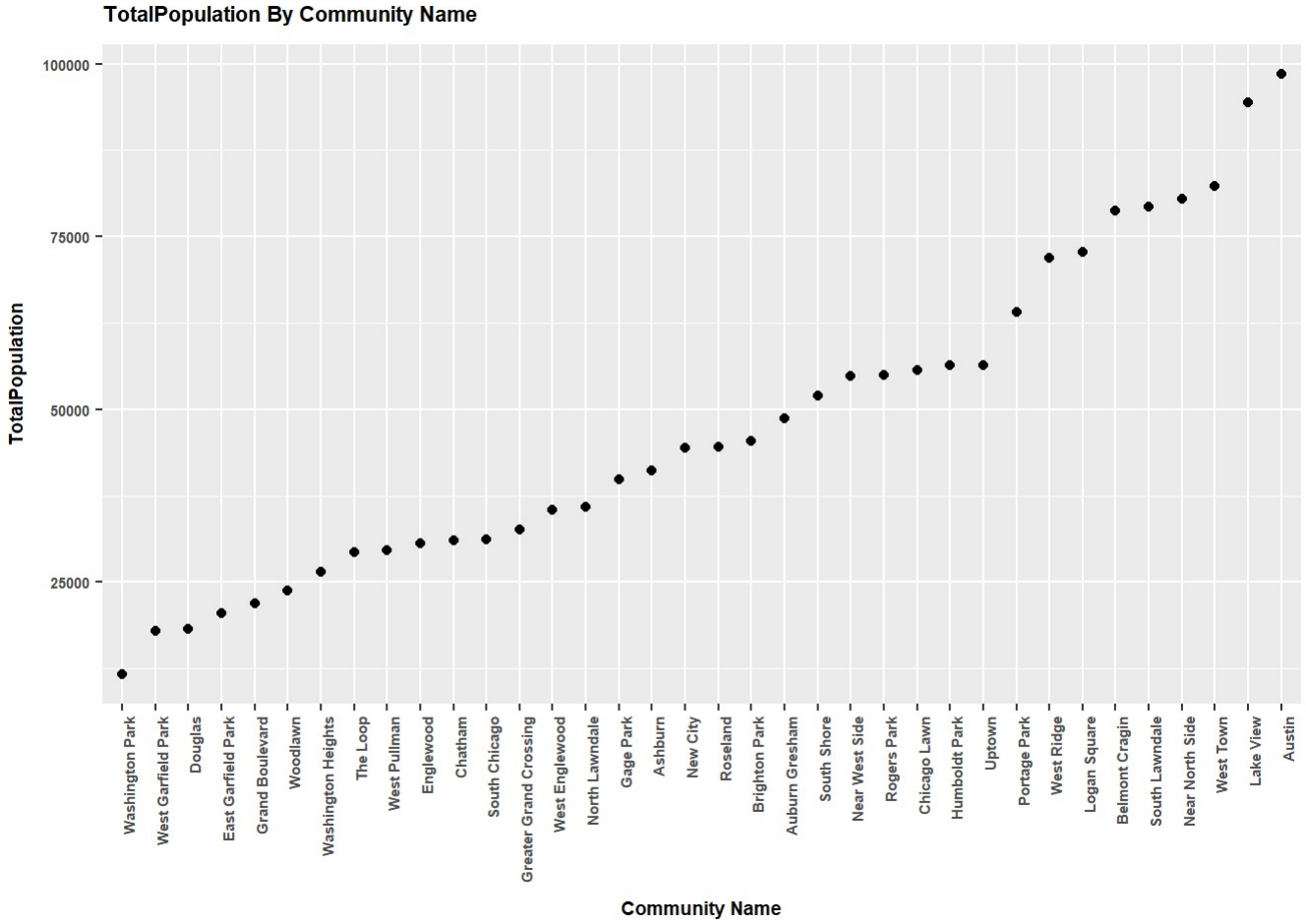
```
## [1] "Assortativity: -0.101919022138091"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.101919
## Pr(X>=Obs): 0.846
## Pr(X<=Obs): 0.154
##
## [1]
```



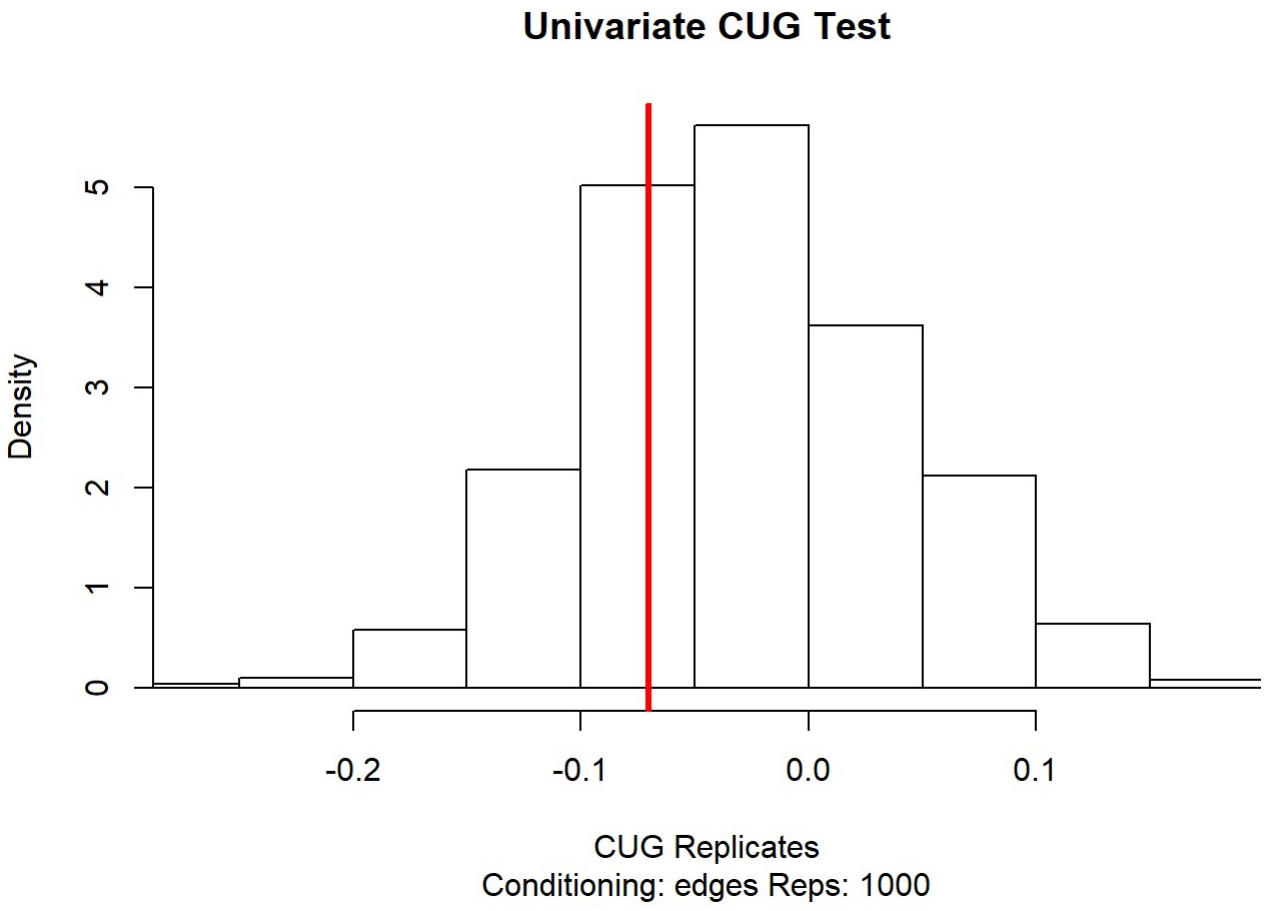
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.823
##  p(f(perm) <= f(d)): 0.177
##
## Test Diagnostics:
##  Test Value (f(d)): -0.101919
##  Replications: 1000
##  Distribution Summary:
##       Min:      -0.2970865
##       1stQ:     -0.08663659
##       Med:      -0.05420014
##       Mean:     -0.0594407
##       3rdQ:     -0.02422432
##       Max:      0.1863186
##
## [1]
```



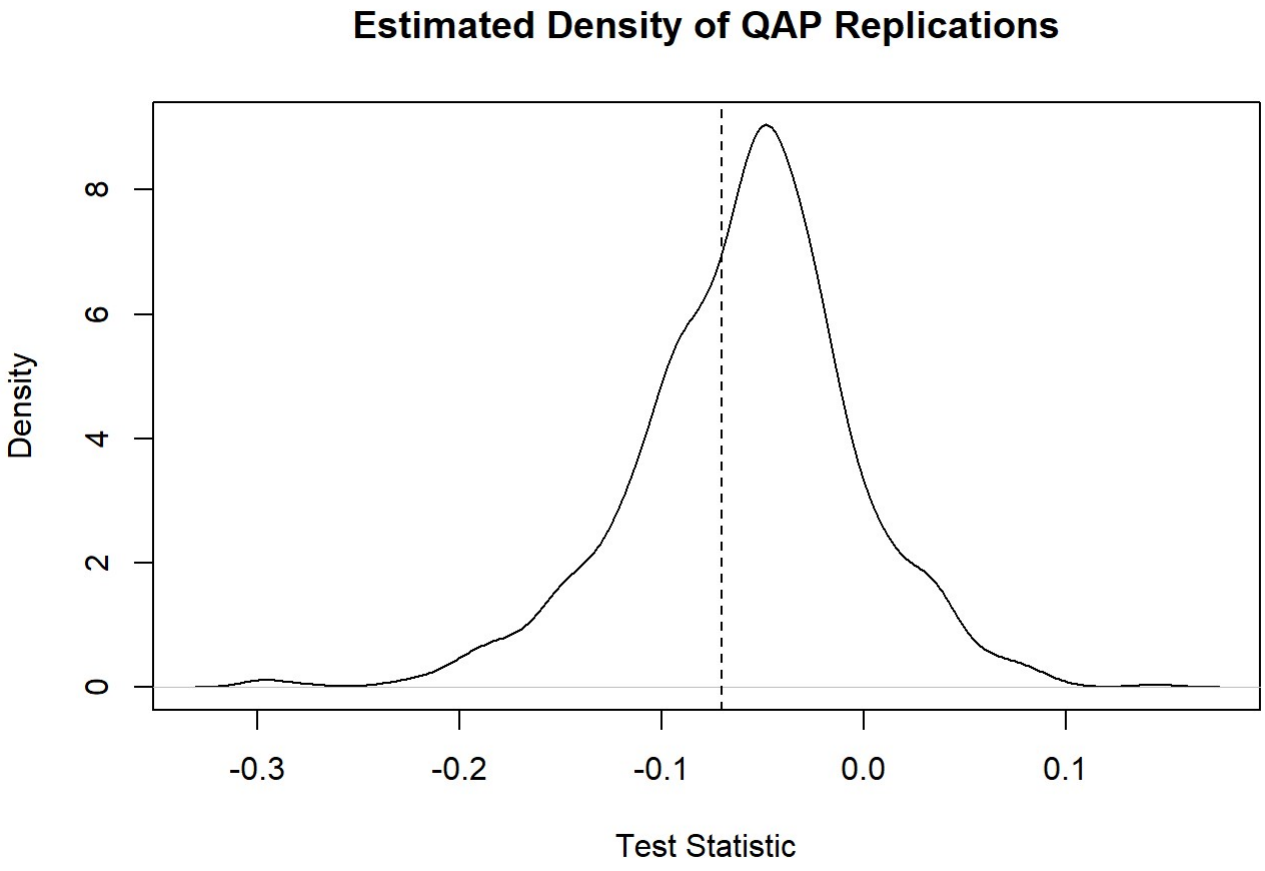
```
## [1] "-------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Total.Population"
## [1]
## [1] "Vertex Values:"
##  [1] 54991 71942 56362 94368 80484 64124 78743 72791 56323 82236 98514
## [12] 18001 20567 54881 35912 79288 29283 18238 21929 11717 23740 52010
## [23] 31028 31198 44619 29651 45368 44377 39894 55628 35505 30654 32602
## [34] 41081 48743 26493
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   11717   30403   44498   47591   58303   98514
## [1]
```



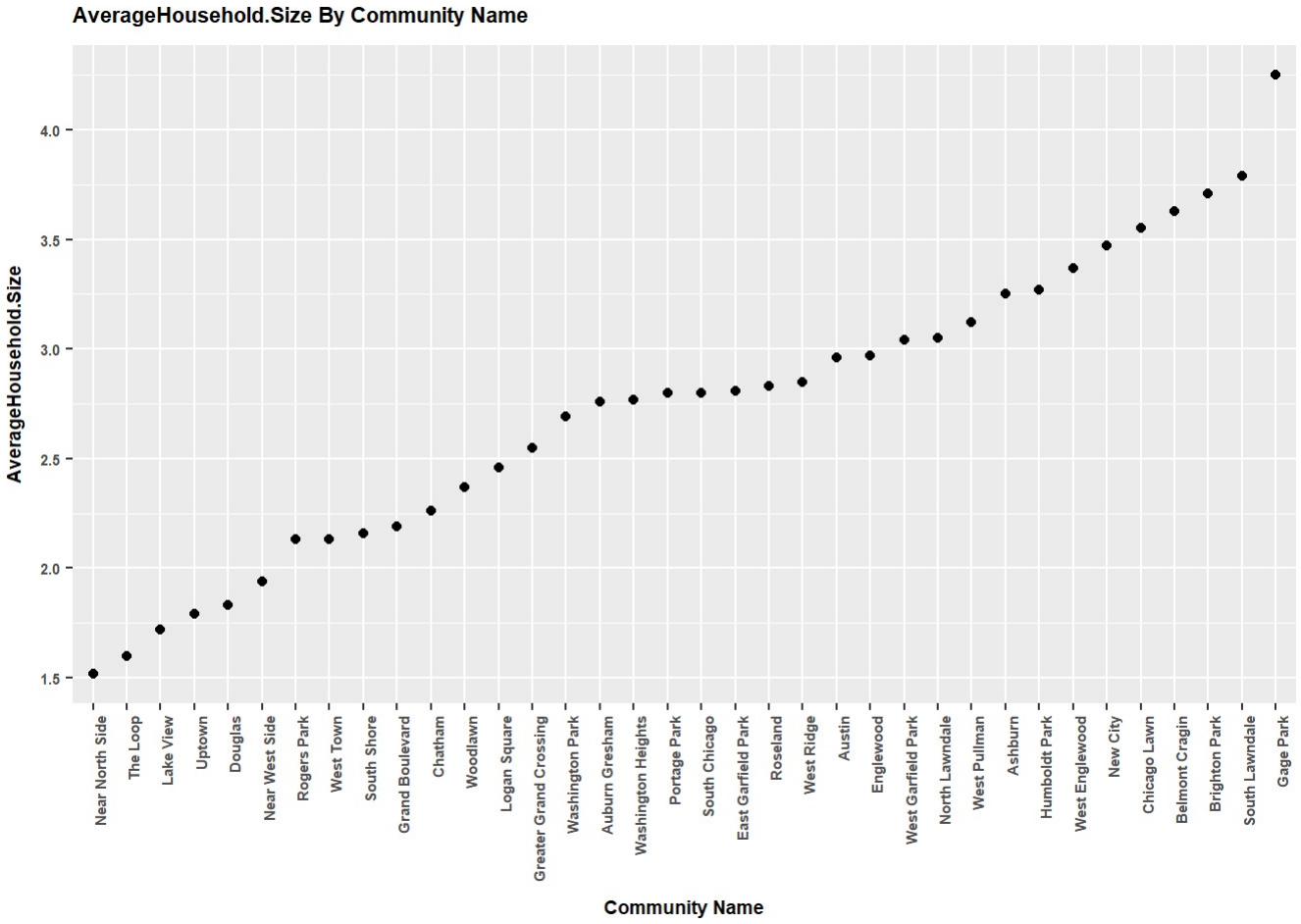
```
## [1] "Assortativity: -0.0704067175267584"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.07040672
## Pr(X>=Obs): 0.724
## Pr(X<=Obs): 0.276
##
## [1]
```



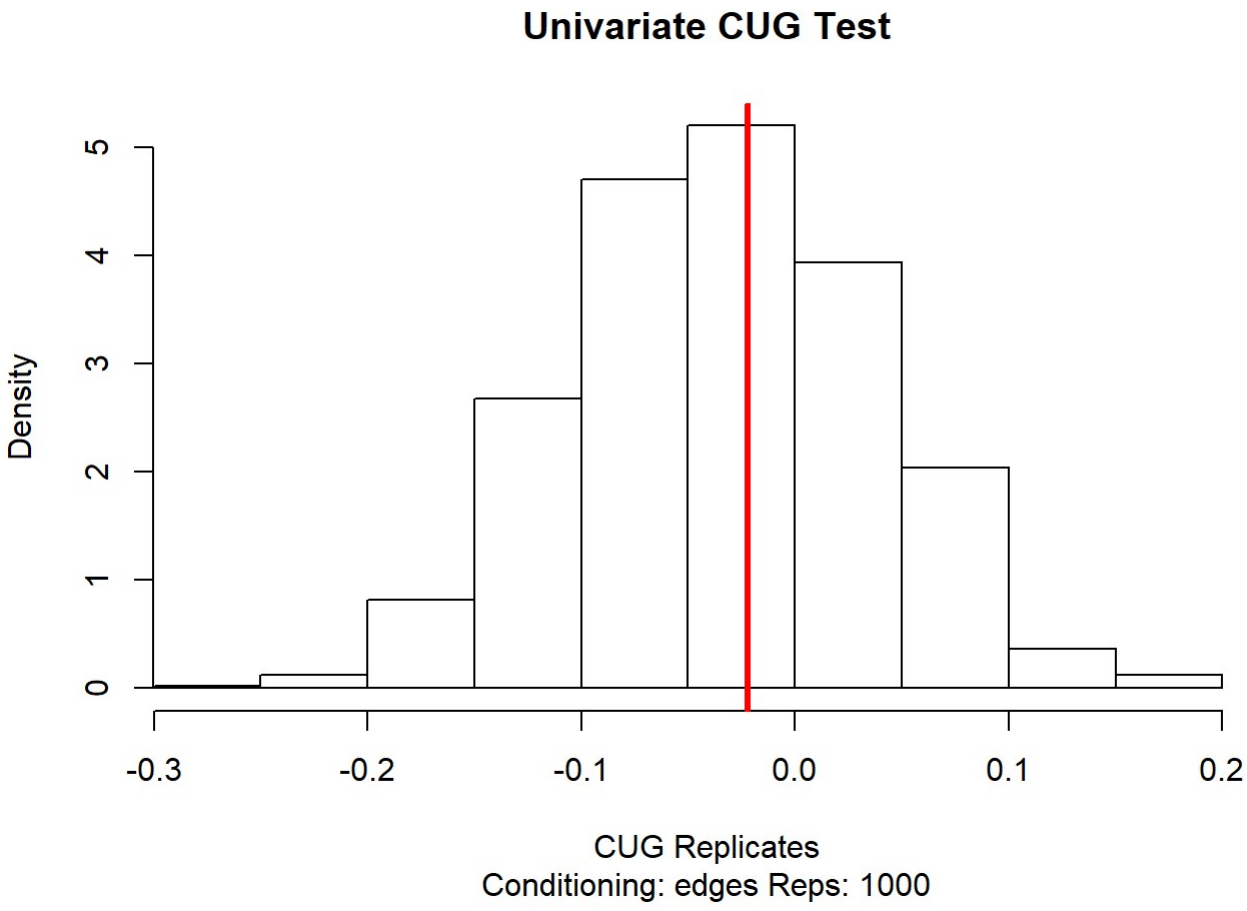
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.625
##  p(f(perm) <= f(d)): 0.375
##
## Test Diagnostics:
##  Test Value (f(d)): -0.07040672
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2992327
##      1stQ:    -0.09046521
##      Med:     -0.0544347
##      Mean:    -0.05890451
##      3rdQ:    -0.02709906
##      Max:     0.143481
##
## [1]
```



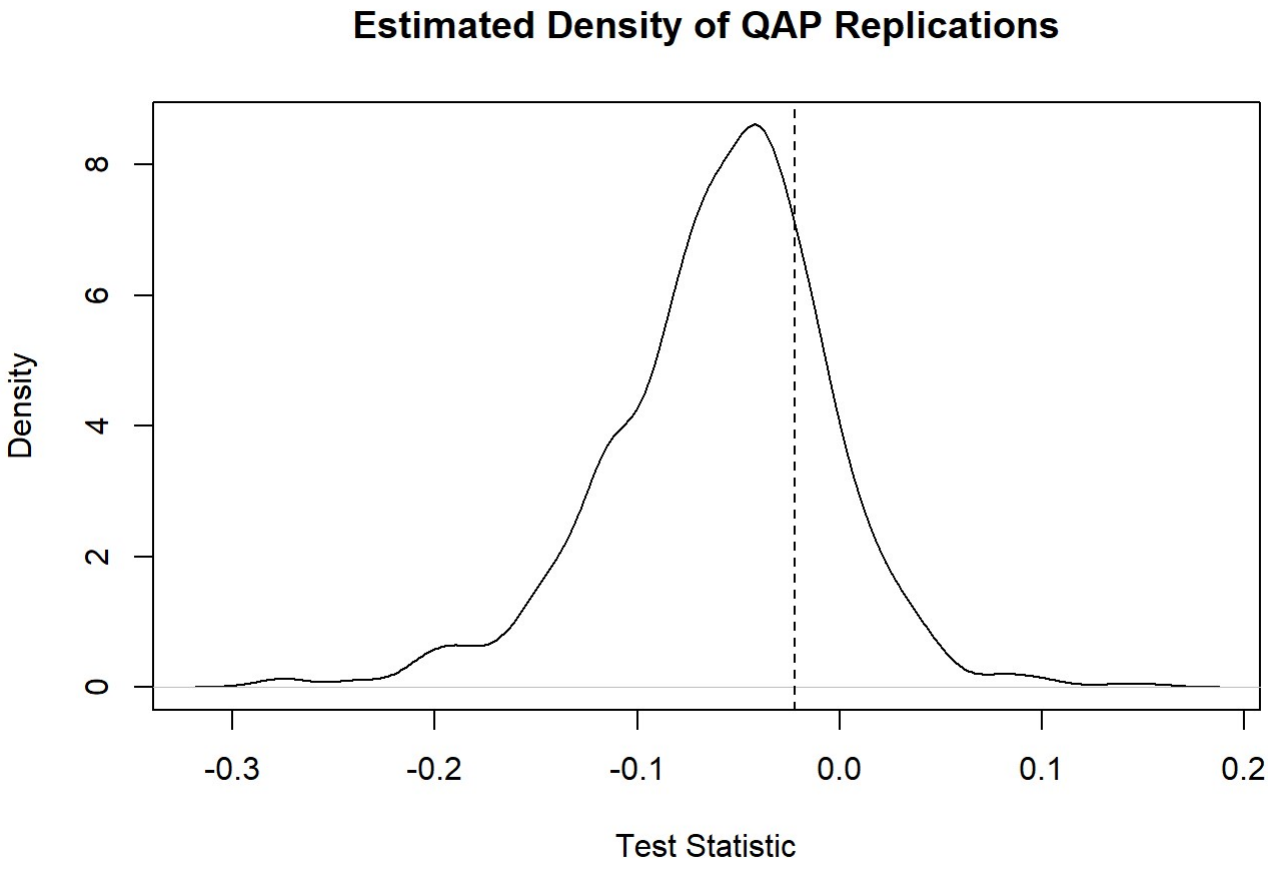
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Average.Household.Size"
## [1]
## [1] "Vertex Values:"
##  [1] 2.13 2.85 1.79 1.72 1.52 2.80 3.63 2.46 3.27 2.13 2.96 3.04 2.81 1.94
## [15] 3.05 3.79 1.60 1.83 2.19 2.69 2.37 2.16 2.26 2.80 2.83 3.12 3.71 3.47
## [29] 4.25 3.55 3.37 2.97 2.55 3.25 2.76 2.77
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.520   2.183   2.800   2.733   3.152   4.250
## [1]
```



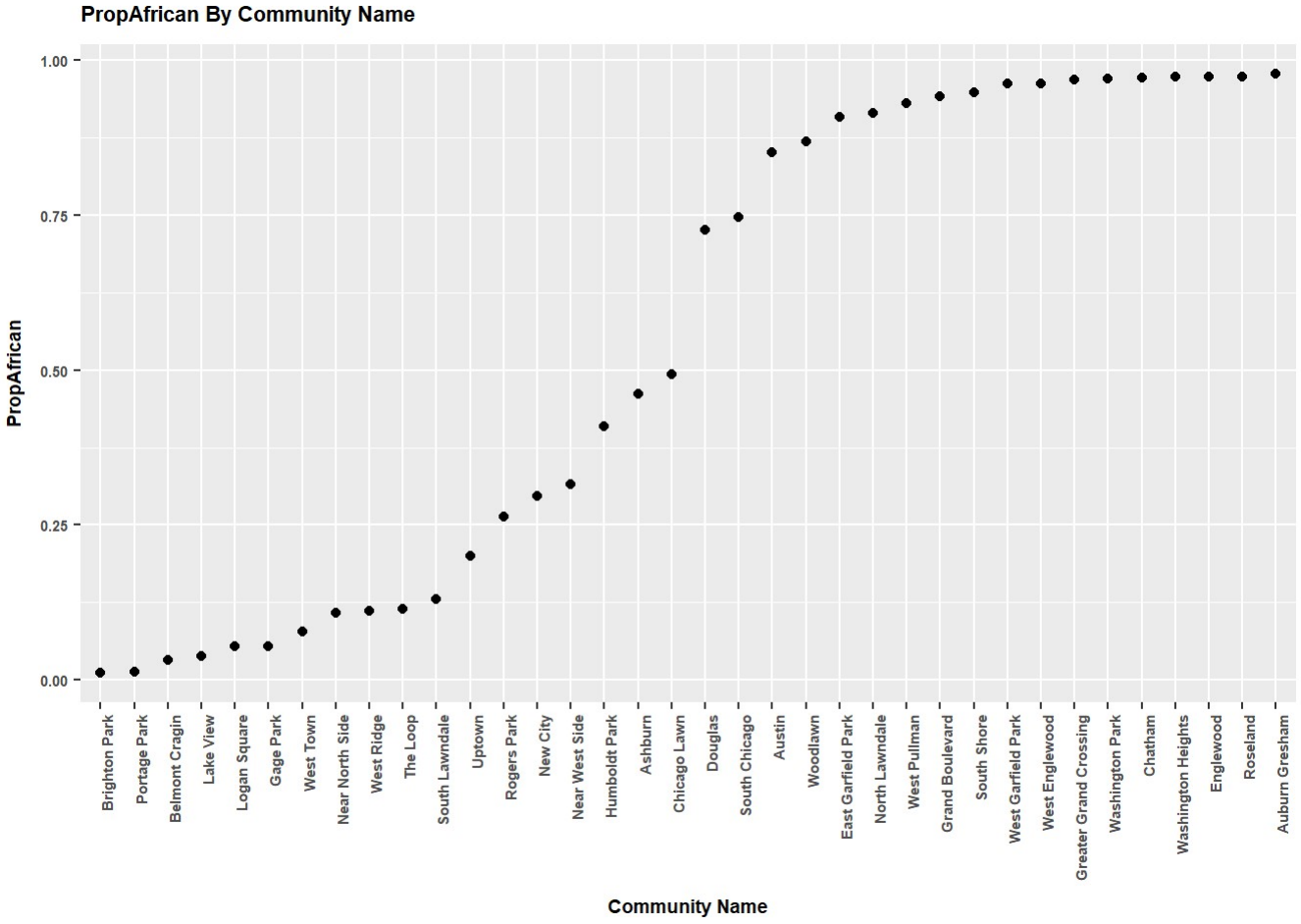
```
## [1] "Assortativity: -0.0219034227958298"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.02190342
## Pr(X>=Obs): 0.442
## Pr(X<=Obs): 0.558
##
## [1]
```



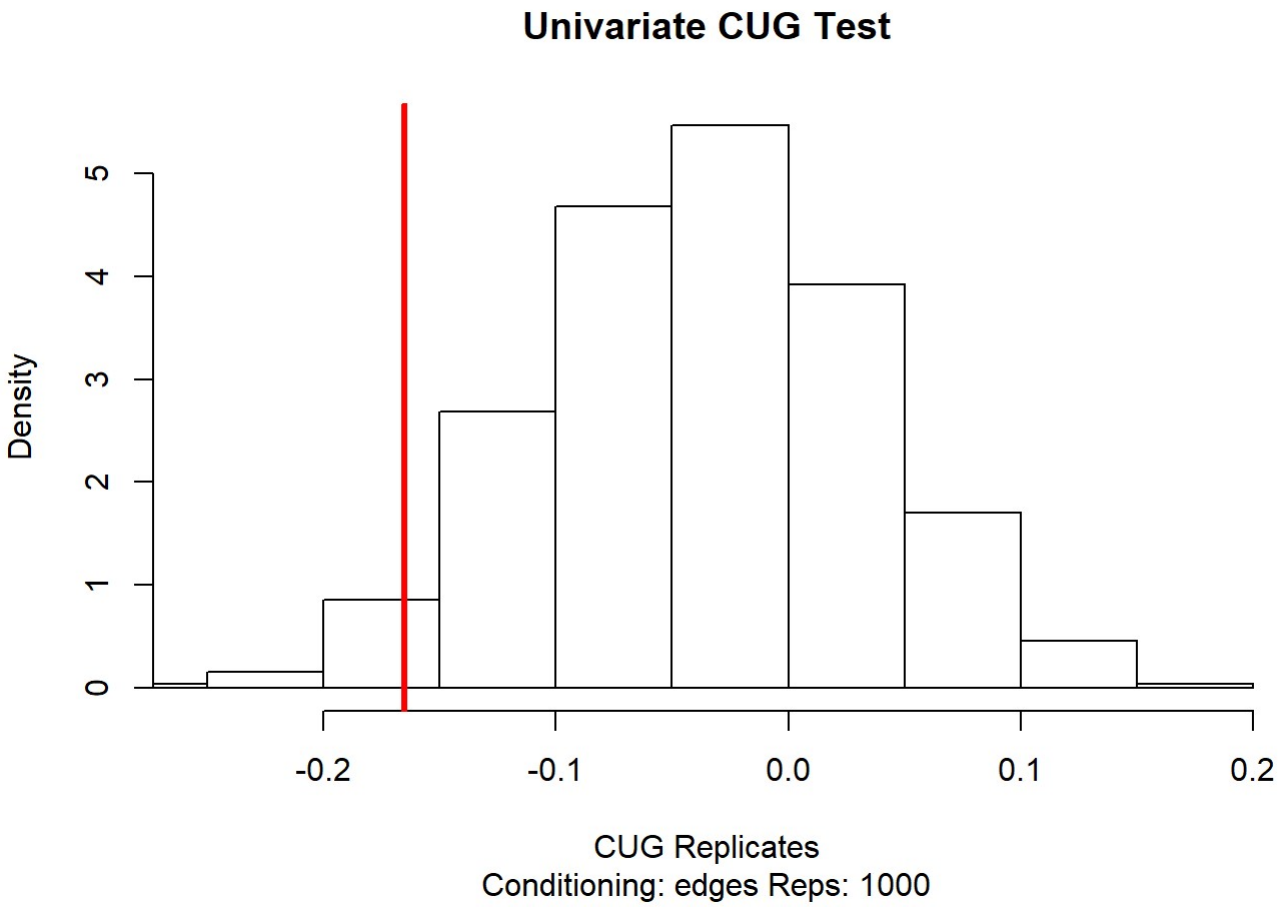
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.229
##  p(f(perm) <= f(d)): 0.771
##
## Test Diagnostics:
##  Test Value (f(d)): -0.02190342
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2858122
##      1stQ:    -0.08947557
##      Med:     -0.05324907
##      Mean:    -0.05881037
##      3rdQ:    -0.0241148
##      Max:     0.1543887
##
## [1]
```



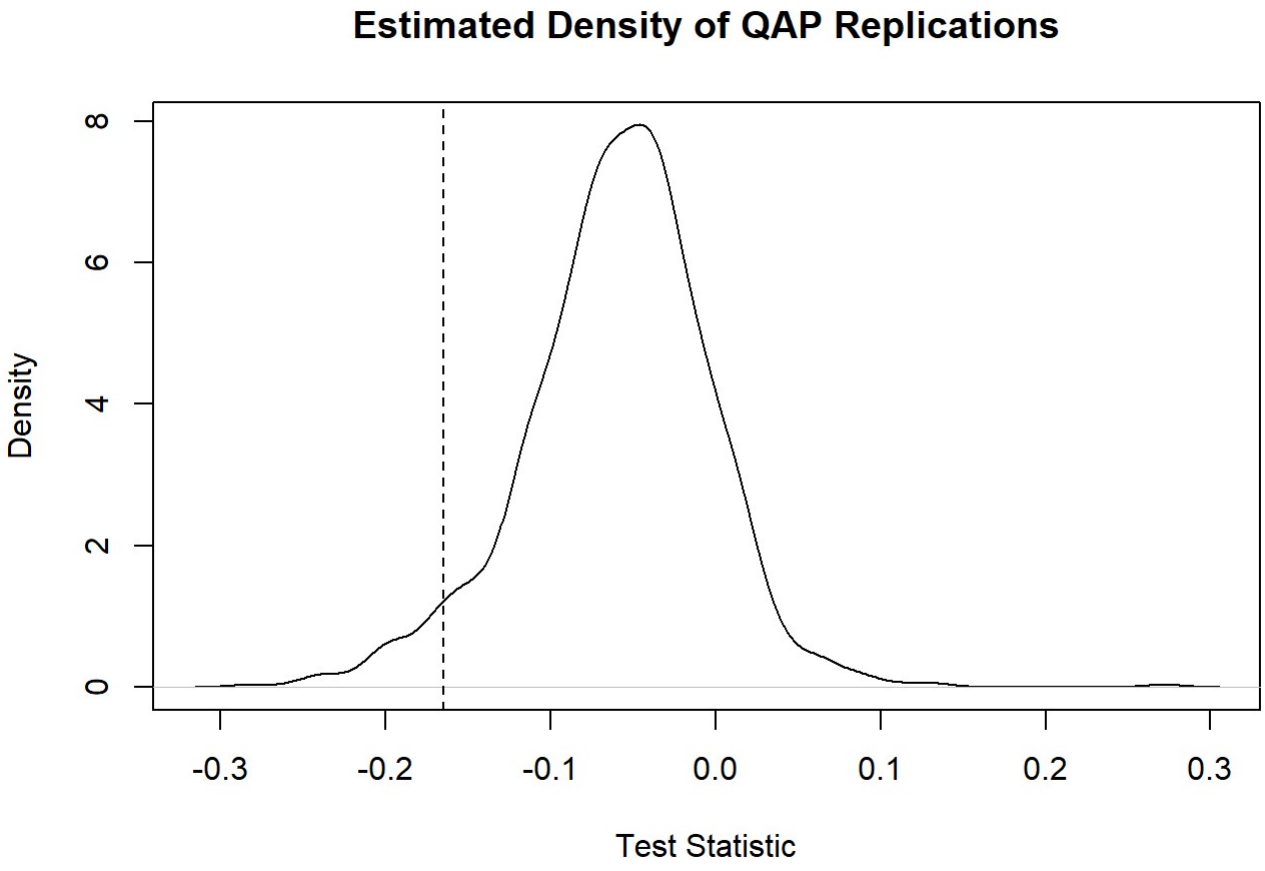
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.African"
## [1]
## [1] "Vertex Values:"
##  [1] 0.26297030 0.11140919 0.20004613 0.03868896 0.10849361 0.01334914
##  [7] 0.03165996 0.05357805 0.40889157 0.07787344 0.85101610 0.96189101
## [13] 0.90902903 0.31528216 0.91431833 0.13083947 0.11477649 0.72634061
## [19] 0.94144740 0.97038491 0.86794440 0.94862526 0.97202527 0.74671453
## [25] 0.97373316 0.93059256 0.01194675 0.29627960 0.05416855 0.49261163
## [31] 0.96262498 0.97367391 0.96895896 0.46191670 0.97780194 0.97357793
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01195 0.11393 0.60948 0.54876 0.95194 0.97780
## [1]
```



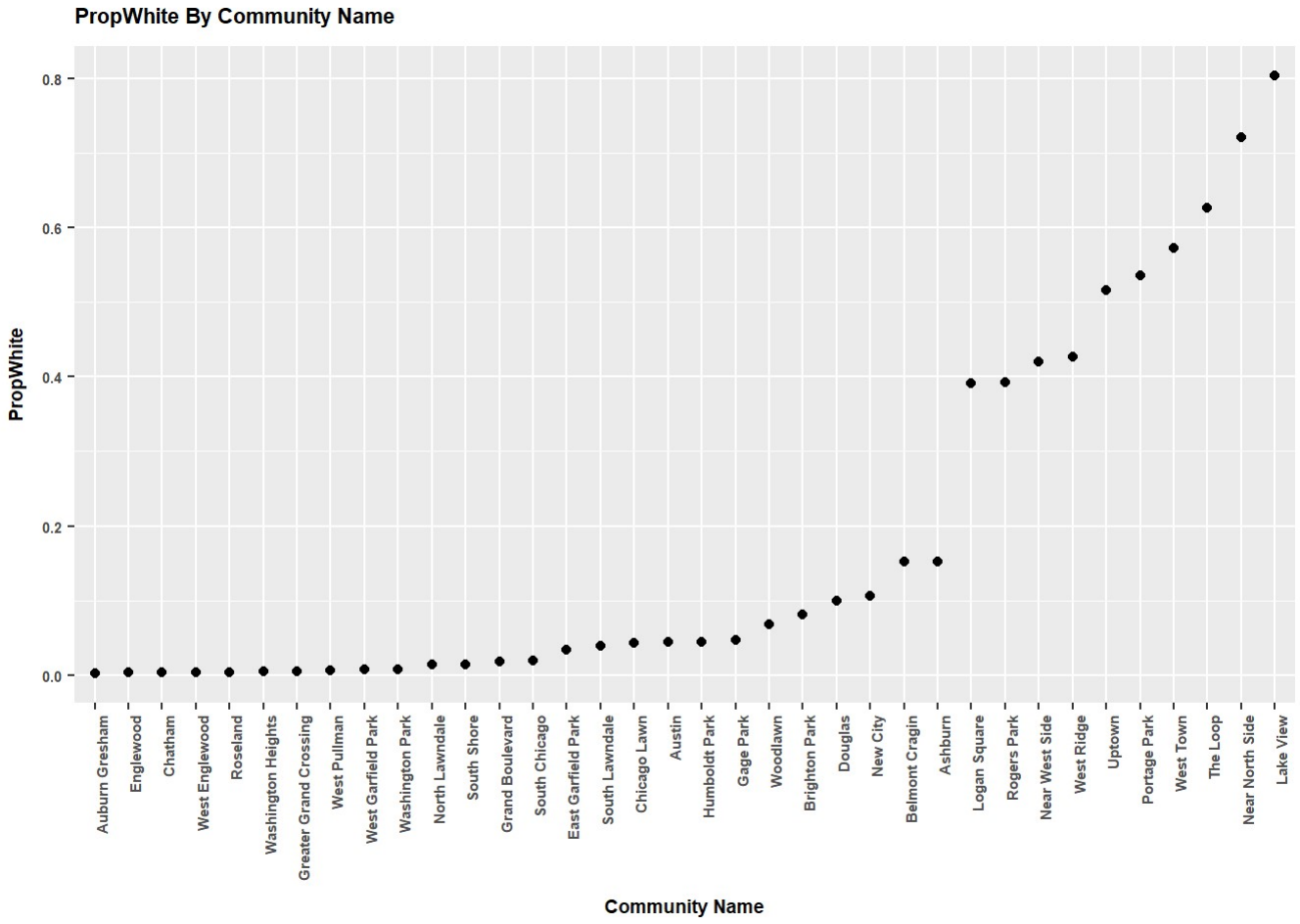
```
## [1] "Assortativity: -0.165140854863703"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1651409
## Pr(X>=Obs): 0.967
## Pr(X<=Obs): 0.033
##
## [1]
```



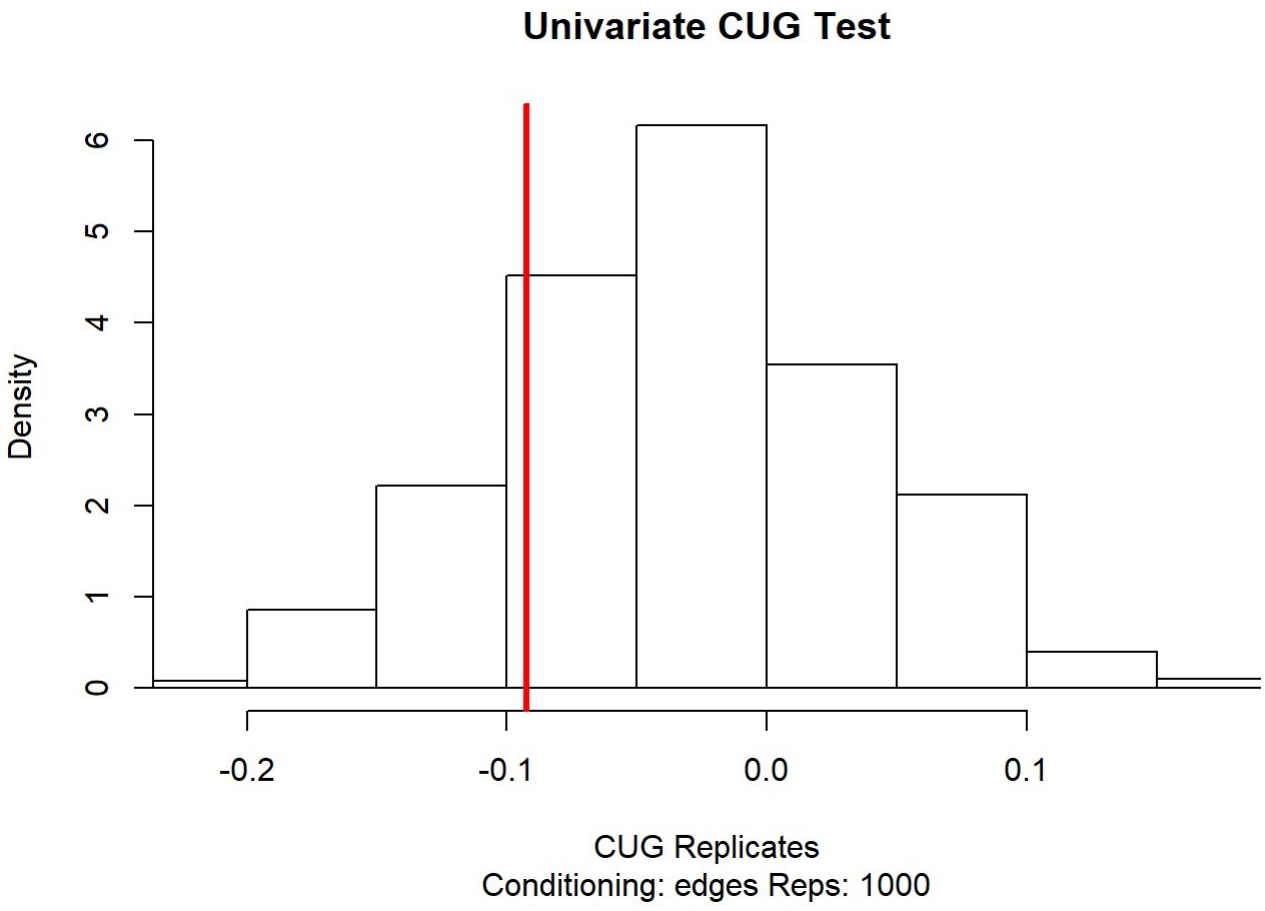
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.956
##  p(f(perm) <= f(d)): 0.044
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1651409
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2825157
##      1stQ:    -0.09123907
##      Med:     -0.05601504
##      Mean:    -0.0594397
##      3rdQ:    -0.02498376
##      Max:     0.271621
##
## [1]
```



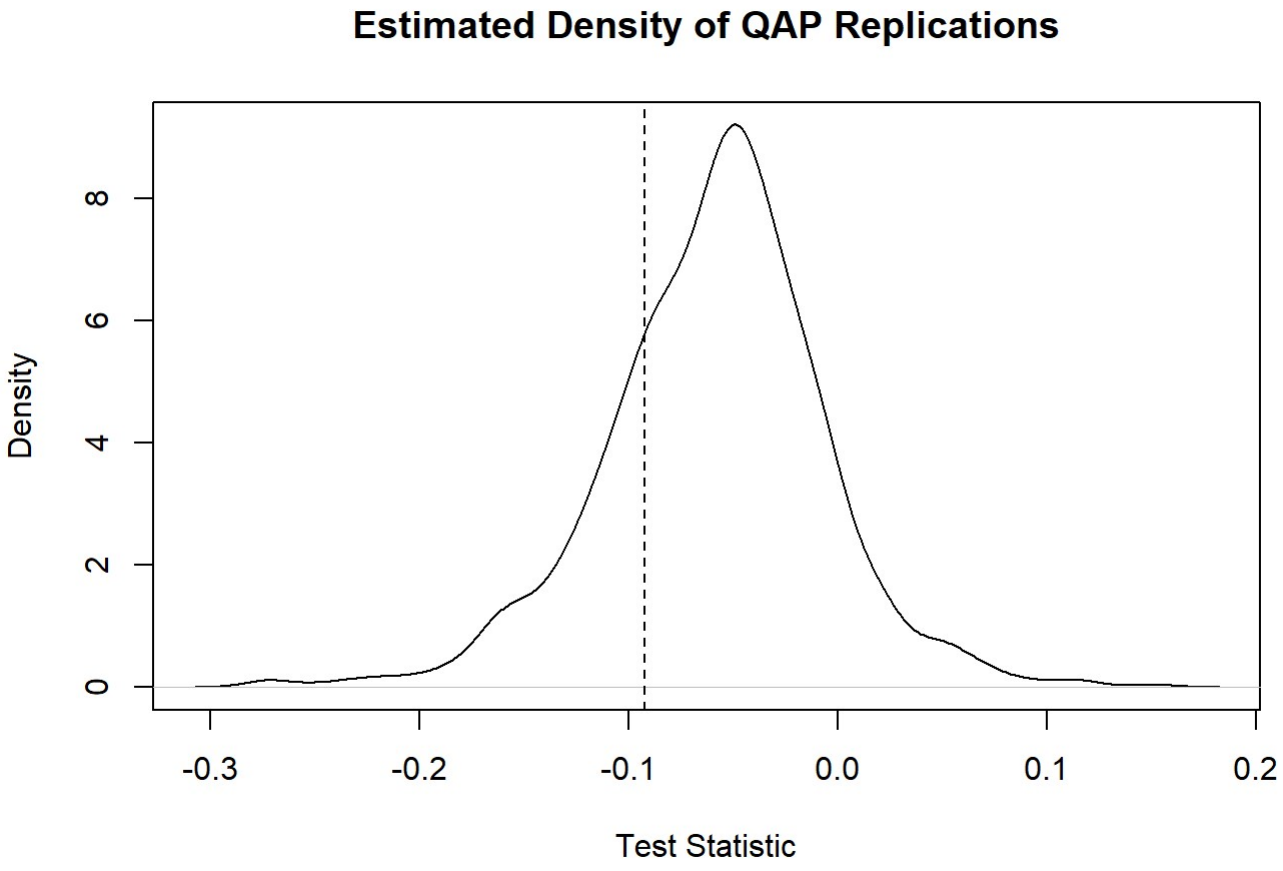
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.White"
## [1]
## [1] "Vertex Values:"
##  [1] 0.393118874 0.426816046 0.516269827 0.803715242 0.721050147
##  [6] 0.535353378 0.151873817 0.391724252 0.044422350 0.572146019
## [11] 0.044298272 0.007388478 0.033937862 0.419853866 0.013700156
## [16] 0.038543033 0.626950791 0.099243338 0.017830270 0.007425109
## [21] 0.068365628 0.013920400 0.003609643 0.019199949 0.004213452
## [26] 0.005598462 0.081158526 0.105843117 0.047425678 0.043485295
## [31] 0.003661456 0.003425328 0.005521134 0.152162800 0.002749113
## [36] 0.004567244
## [1]
## [1] "Vertex Value Summary:"
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.002749 0.007416 0.044360 0.178627 0.392073 0.803715
## [1]
```



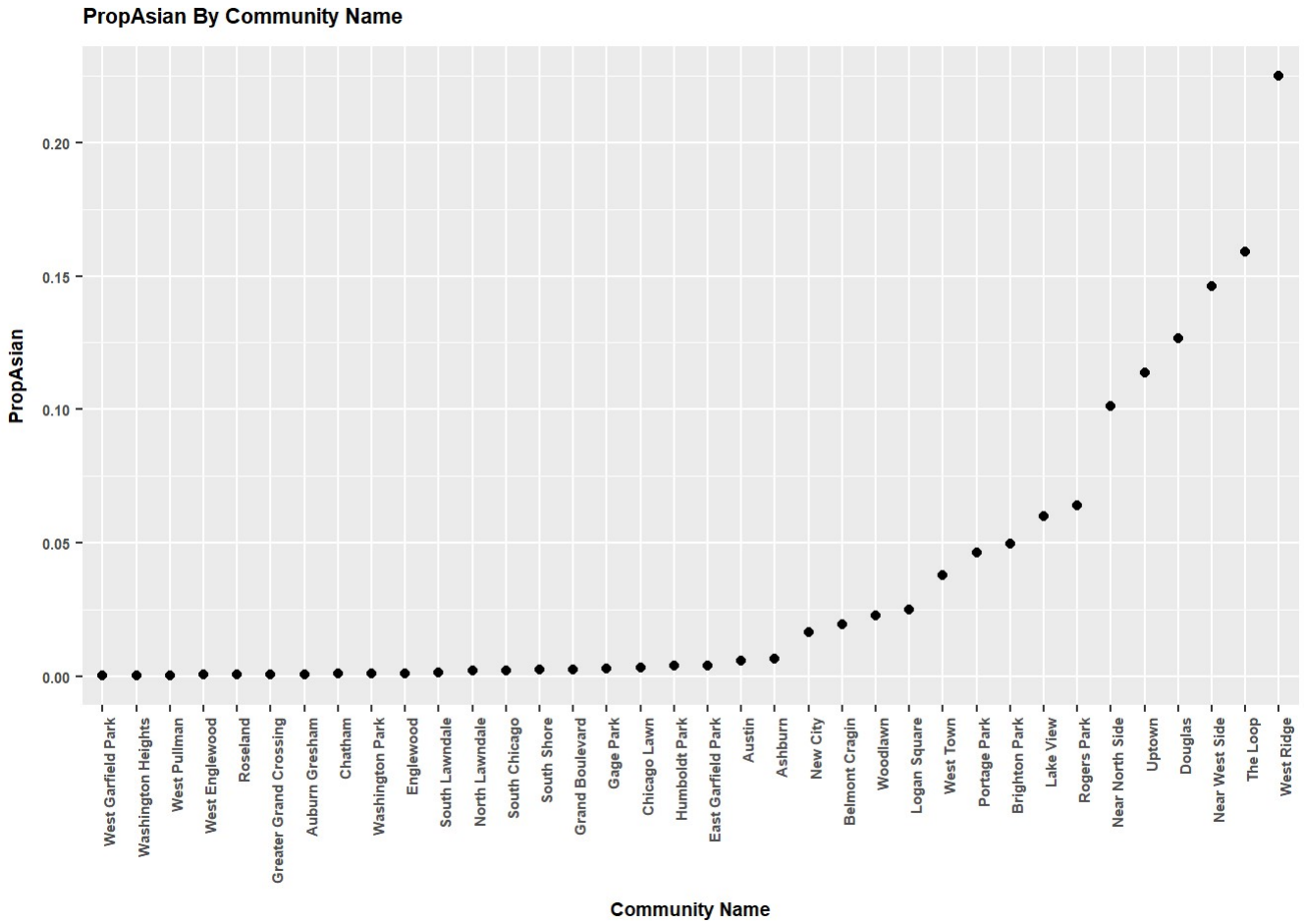
```
## [1] "Assortativity: -0.0923916566047209"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.09239166
## Pr(X>=Obs): 0.813
## Pr(X<=Obs): 0.187
##
## [1]
```



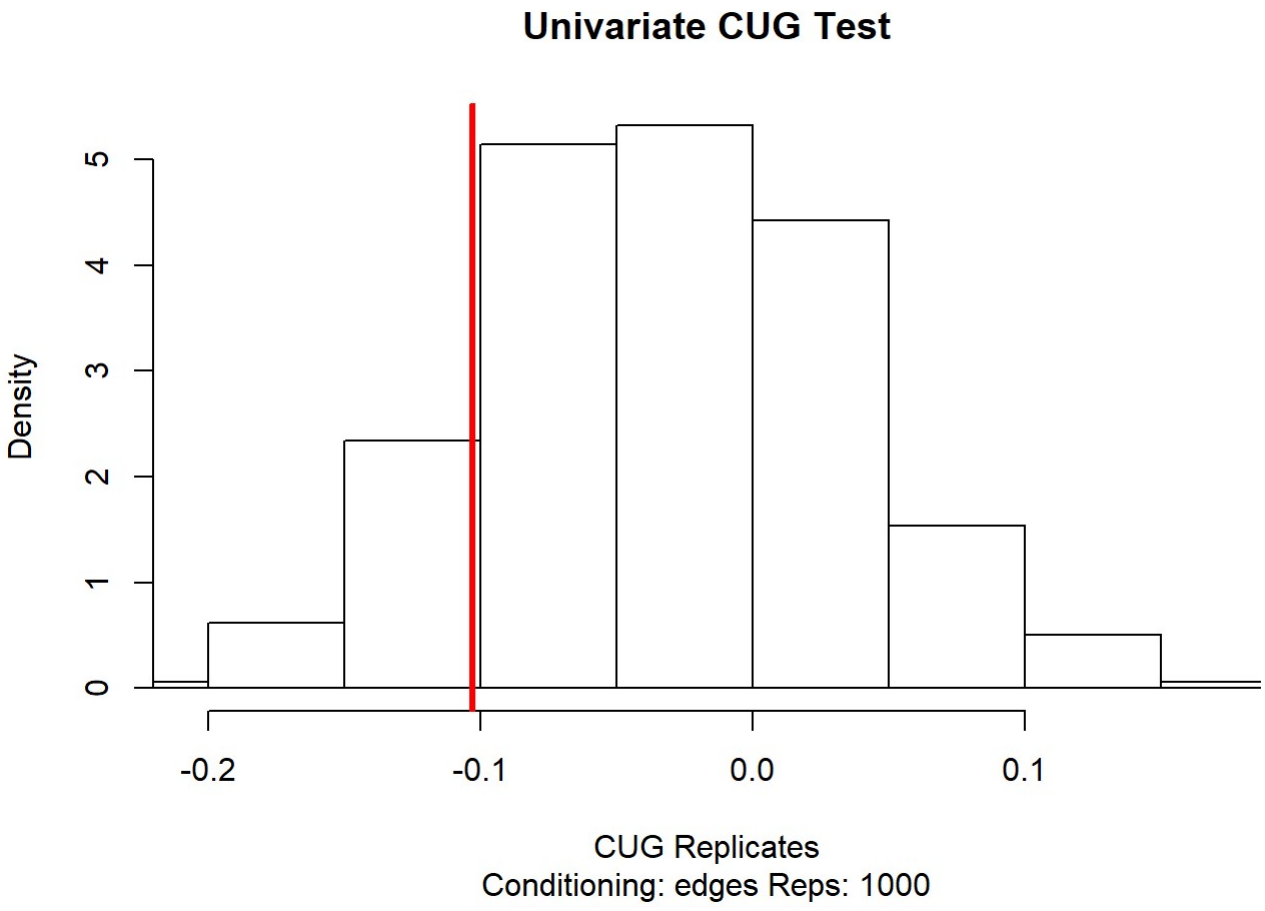
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.768
##  p(f(perm) <= f(d)): 0.232
##
## Test Diagnostics:
##  Test Value (f(d)): -0.09239166
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2765401
##      1stQ:    -0.08908627
##      Med:     -0.05561928
##      Mean:    -0.05893924
##      3rdQ:    -0.02773962
##      Max:     0.1510559
##
## [1]
```



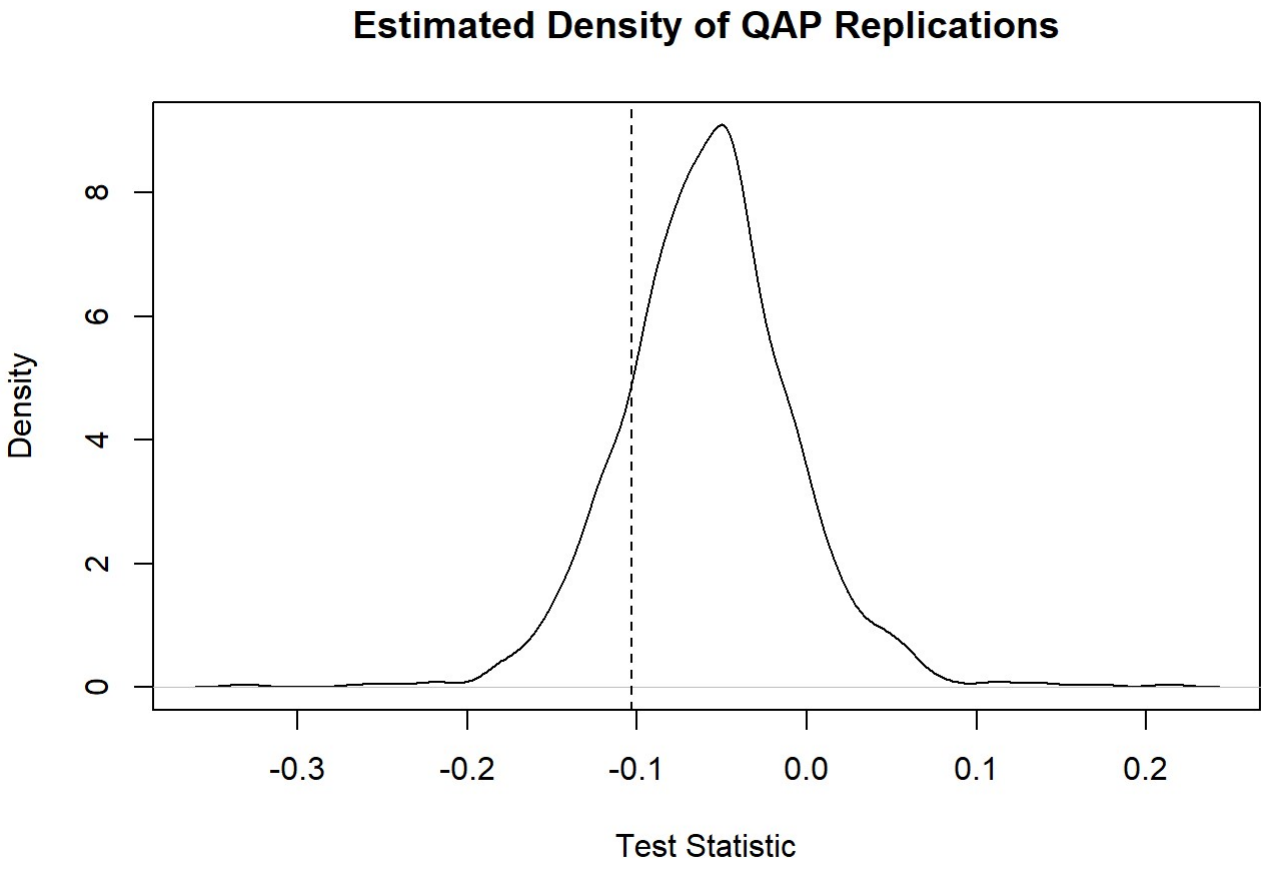
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Asian"
## [1]
## [1] "Vertex Values:"
##  [1] 0.0640650288 0.2249589947 0.1138000781 0.0599037809 0.1011381144
##  [6] 0.0463477013 0.0195191954 0.0251129947 0.0039415514 0.0377693468
## [11] 0.0058164322 0.0004444198 0.0040355910 0.1461708059 0.0020605926
## [16] 0.0014251841 0.1590684015 0.1268231166 0.0025992977 0.0010241529
## [21] 0.0229149115 0.0023841569 0.0008701818 0.0021475736 0.0006051234
## [26] 0.0004721595 0.0496385117 0.0163823602 0.0030330375 0.0034155461
## [31] 0.0005351359 0.0011091538 0.0006441323 0.0067184343 0.0006770203
## [36] 0.0004529498
## [1]
## [1] "Vertex Value Summary:"
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0004444 0.0010879 0.0039886 0.0349451 0.0471704 0.2249590
## [1]
```



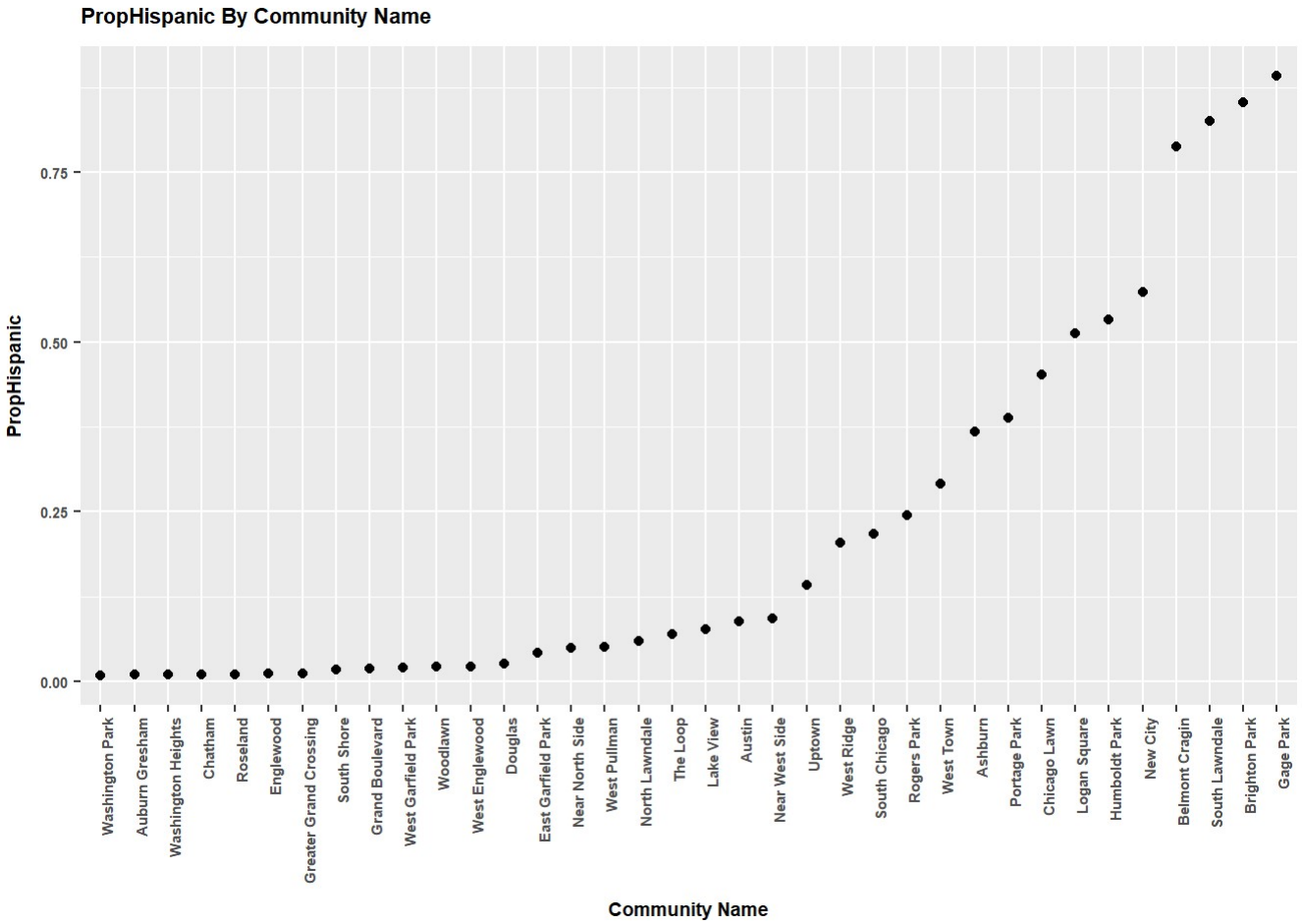
```
## [1] "Assortativity: -0.102935054152218"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1029351
## Pr(X>=Obs): 0.858
## Pr(X<=Obs): 0.142
##
## [1]
```



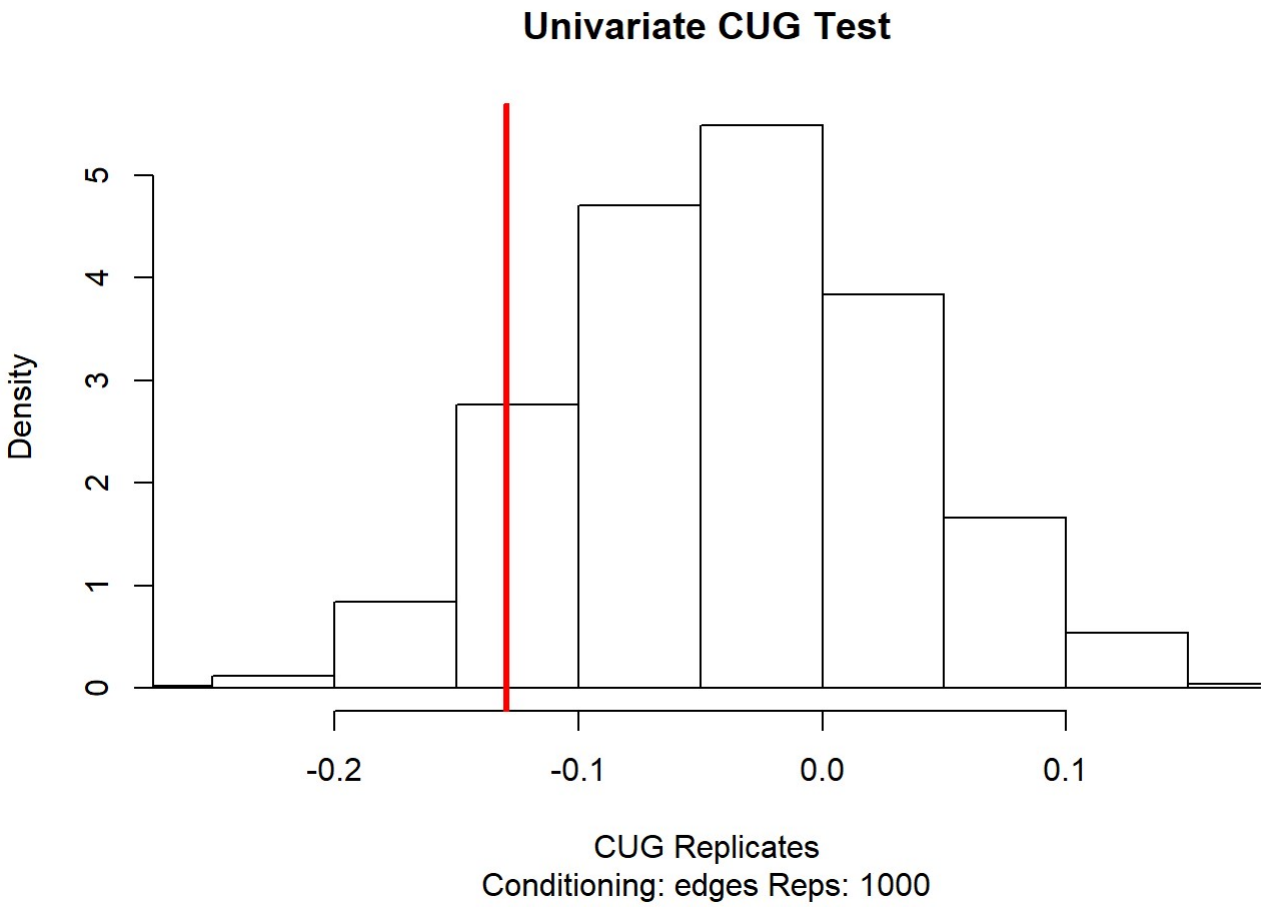
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.834
##  p(f(perm) <= f(d)): 0.166
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1029351
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3308154
##      1stQ:    -0.08887338
##      Med:     -0.05846928
##      Mean:    -0.05844598
##      3rdQ:    -0.0299322
##      Max:     0.2128573
##
## [1]
```



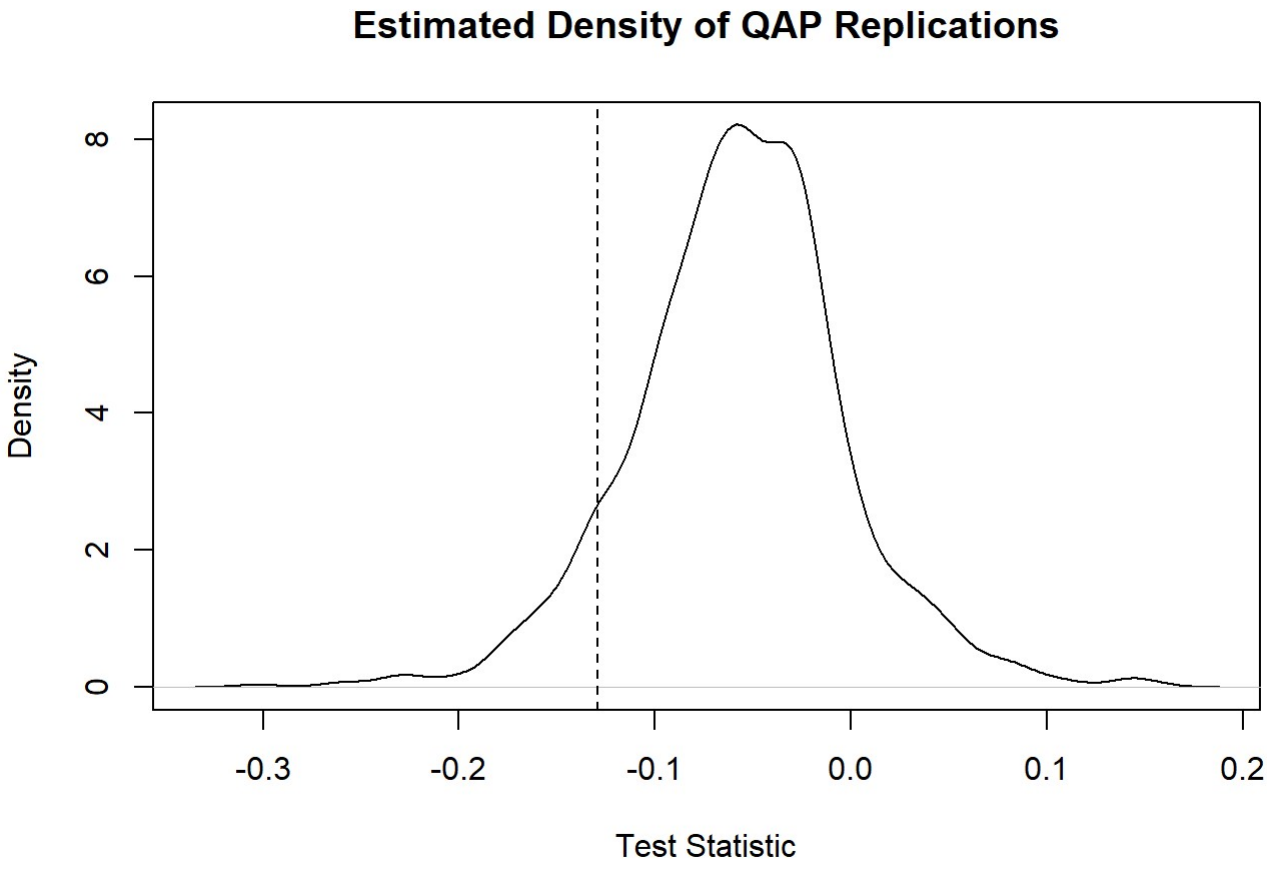
```
## [1] "-------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Hispanic"
## [1]
## [1] "Vertex Values:"
##  [1] 0.244276336 0.204345167 0.142099287 0.076275856 0.049376274
##  [6] 0.387701952 0.788654230 0.512357297 0.533458800 0.290639136
## [11] 0.088535640 0.019332259 0.041328342 0.091980831 0.059701493
## [16] 0.825559984 0.068640508 0.025496217 0.018012677 0.008875992
## [21] 0.021229992 0.017169775 0.010023205 0.217129303 0.010264685
## [26] 0.050892044 0.852869864 0.573067129 0.892089036 0.451948659
## [31] 0.021799747 0.010602205 0.011901110 0.368345464 0.009416737
## [36] 0.009738421
## [1]
## [1] "Vertex Value Summary:"
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.008876 0.019002 0.072458 0.222365 0.373185 0.892089
## [1]
```



```
## [1] "Assortativity: -0.129276065305737"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1292761
## Pr(X>=Obs): 0.921
## Pr(X<=Obs): 0.079
##
## [1]
```



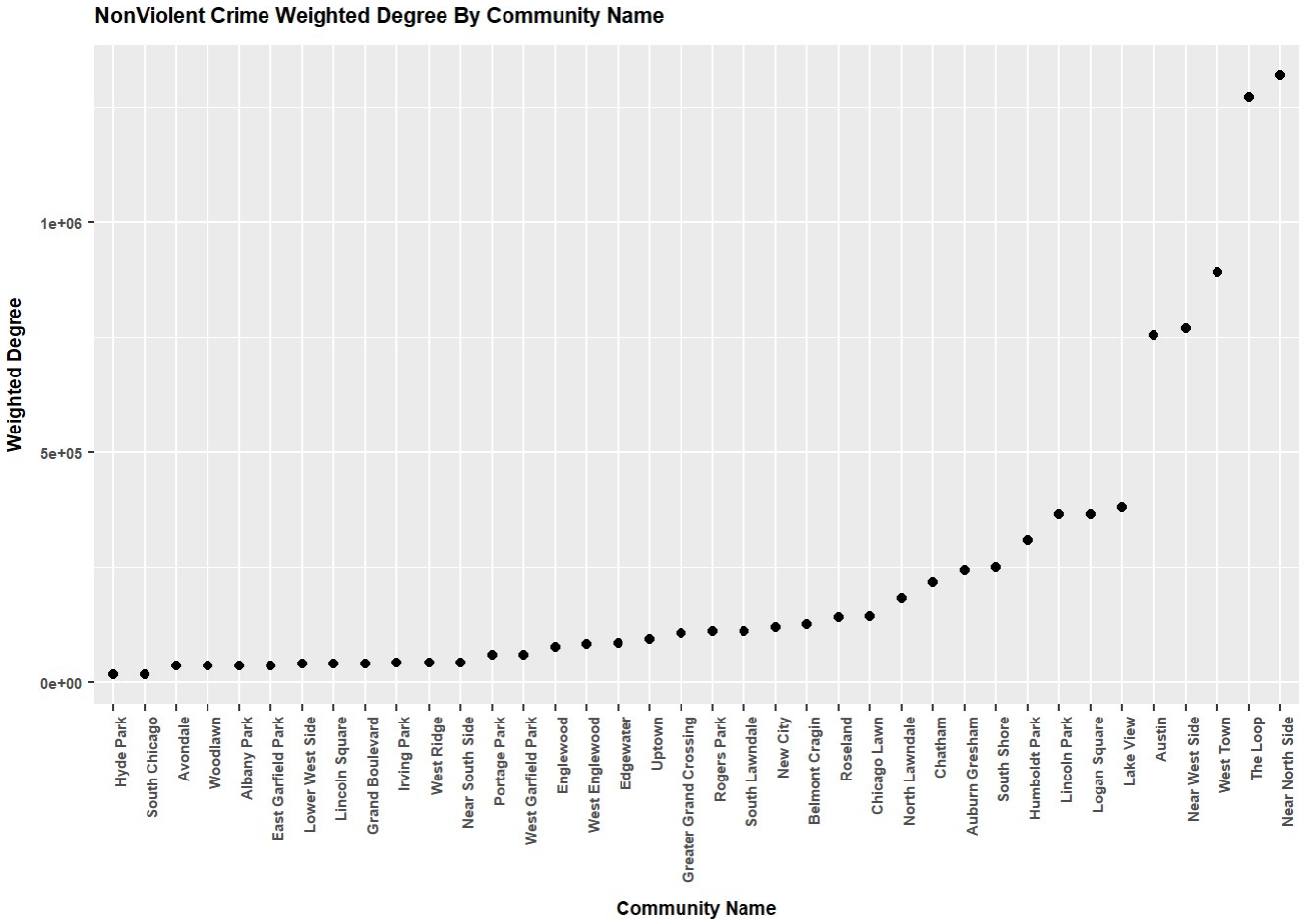
```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.912
##  p(f(perm) <= f(d)): 0.088
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1292761
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.303007
##      1stQ:    -0.08936458
##      Med:     -0.05594441
##      Mean:    -0.0570872
##      3rdQ:    -0.02596703
##      Max:     0.1558498
##
## [1]
```



```
if (save_to_file){
  sink()

  close(outFile)
  closeAllConnections()
}
```

Non-violent crime projection assortativity analysis

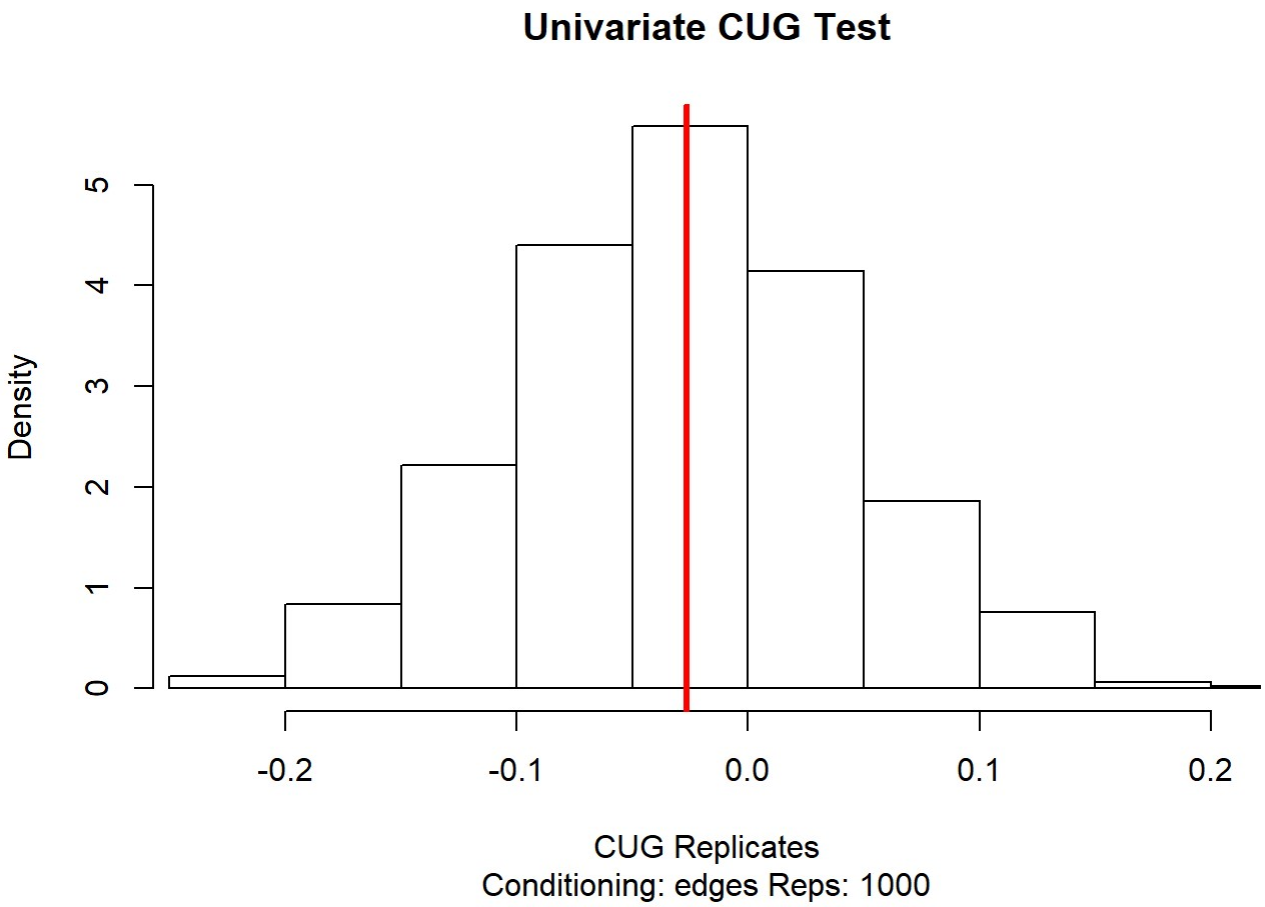
```
if (save_to_file){
  out_file_name = paste(assort_path, "NonViolent_Assortativity.txt", sep = '')
  outFile = file(out_file_name, open="wt")
  sink(file = outFile, append = TRUE)
}

if (save_to_file){
  tmp_scatter = paste(attr_path, "NonViolent_WeightedDegree_Vs_Community.jpg", sep =
'')
  jpeg(file = tmp_scatter)
}

temp_df = data.frame(Community = V(nonviolent_filter)$Label, WDegree = V(nonviolent_f
ilter)$wdegree)
temp_df = temp_df[order(temp_df$WDegree),]
tmp_g = ggplot(data = temp_df, aes(x = reorder(Community, WDegree), y = WDegree)) +
  geom_point() +
  labs(x = "Community Name", y = "Weighted Degree") +
  ggtitle("NonViolent Crime Weighted Degree By Community Name") +
  theme(text = element_text(size=7, face="bold"), axis.text.x = element_text(angle =
90, hjust = 1))
print(tmp_g)
```


NonViolent Crime Weighted Degree By Community Name
Weighted Degree
0e+00
5e+05
1e+06
Hyde Park
South Chicago
Avondale
Woodlawn
Albany Park
East Garfield Park
Lower West Side
Lincoln Square
Grand Boulevard
Irving Park
West Ridge
Near South Side
Portage Park
West Garfield Park
Englewood
West Englewood
Edgewater
Uptown
Greater Grand Crossing
Rogers Park
South Lawndale
New City
Belmont Cragin
Roseland
Chicago Lawn
North Lawndale
Chatham
Auburn Gresham
South Shore
Humboldt Park
Lincoln Park
Logan Square
Lake View
Austin
Near West Side
West Town
The Loop
Near North Side
Community Name

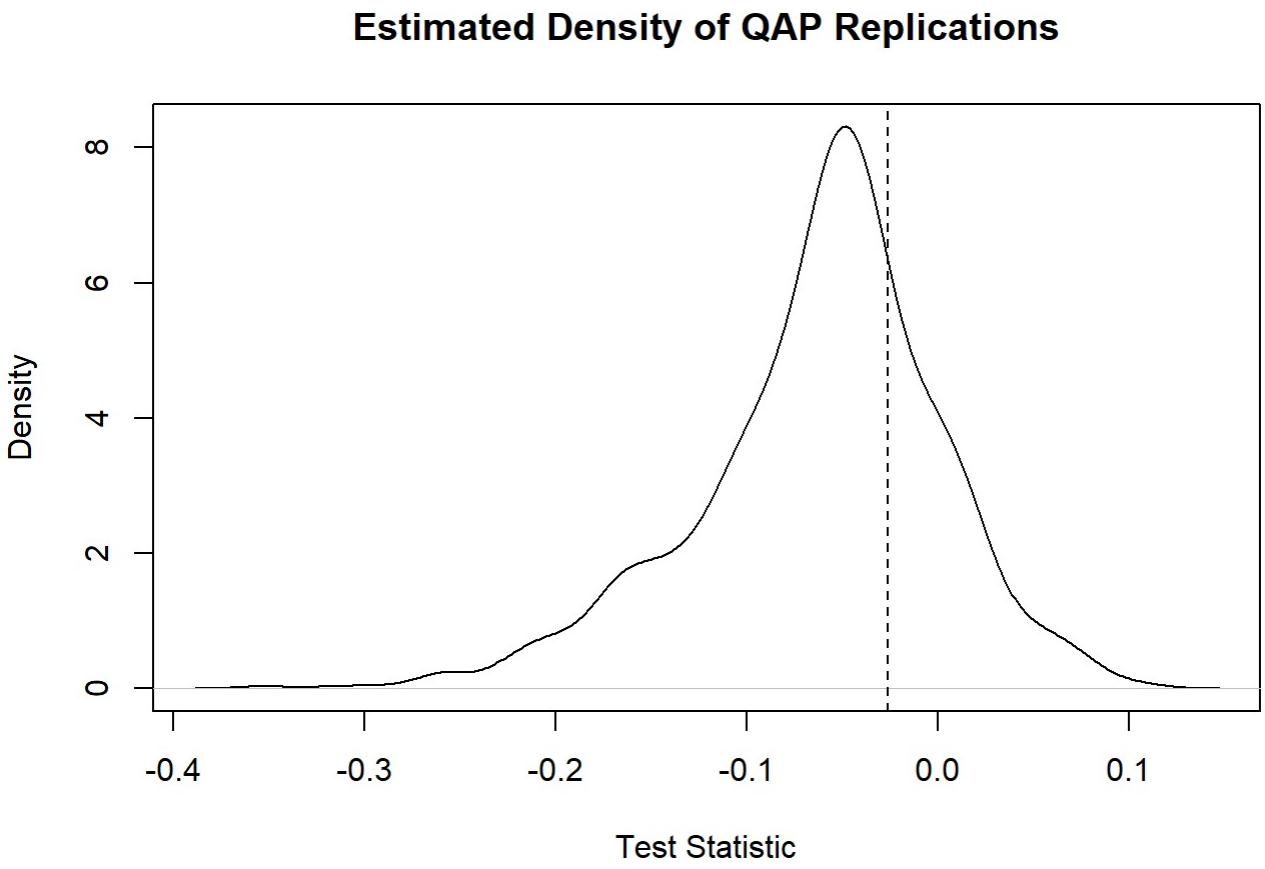
```
if (save_to_file){
  dev.off()
}

nonviolent_attr = list.vertex.attributes(nonviolent_filter)

for ( i in 1:length(nonviolent_attr)){
  temp_attr = nonviolent_attr[i]

  if (!(temp_attr %in% attrs_skip)){

    print("--------------------------------------------------------------------------
----")
    print(paste("Testing Assortativity for attribute:", temp_attr))
    print("", quote=FALSE)

    temp_var = paste("V(nonviolent_filter)$", temp_attr, sep = "")
    vertex_vals = eval(parse(text = temp_var))
    print("Vertex Values:")
    print(vertex_vals)
    print("", quote=FALSE)
    print("Vertex Value Summary:")
    print(summary(vertex_vals))
    print("", quote=FALSE)

    temp_assort = assortativity(nonviolent_filter, types1 = vertex_vals,
                                directed = FALSE)
    temp_cug = mycugtest(nonviolent_filter, assortativity, cmode = "edges", types1 =
vertex_vals,
                         directed = FALSE)
    temp_qap = myqaptest(nonviolent_filter, assortativity, types1 = vertex_vals,
                         directed = FALSE)



    print(paste("Assortativity:", temp_assort))
    print("", quote=FALSE)

    print("Assortativity CUG Test Results:")
    print.cug.test(temp_cug)
    print("", quote=FALSE)

    temp_attr_jpg =  gsub("\\.", "", temp_attr)
    if (save_to_file){
      tmp_cug = paste(assort_path, "NonViolent_", temp_attr_jpg, '_CUG.jpg', sep = ''
)
      jpeg(file = tmp_cug)
    }
    plot.cug.test(temp_cug)
    if (save_to_file){
      dev.off()
    }

    print("Assortativity QAP Test Results:")
```

```
    print(summary.qaptest(temp_qap))
    print("", quote=FALSE)

    if (save_to_file){
      tmp_qap = paste(assort_path, "NonViolent_", temp_attr_jpg, '_QAP.jpg', sep = ''
)
      jpeg(file = tmp_qap)
    }
    plot.qaptest(temp_qap)
    if (save_to_file){
      dev.off()
    }

  }

}
```

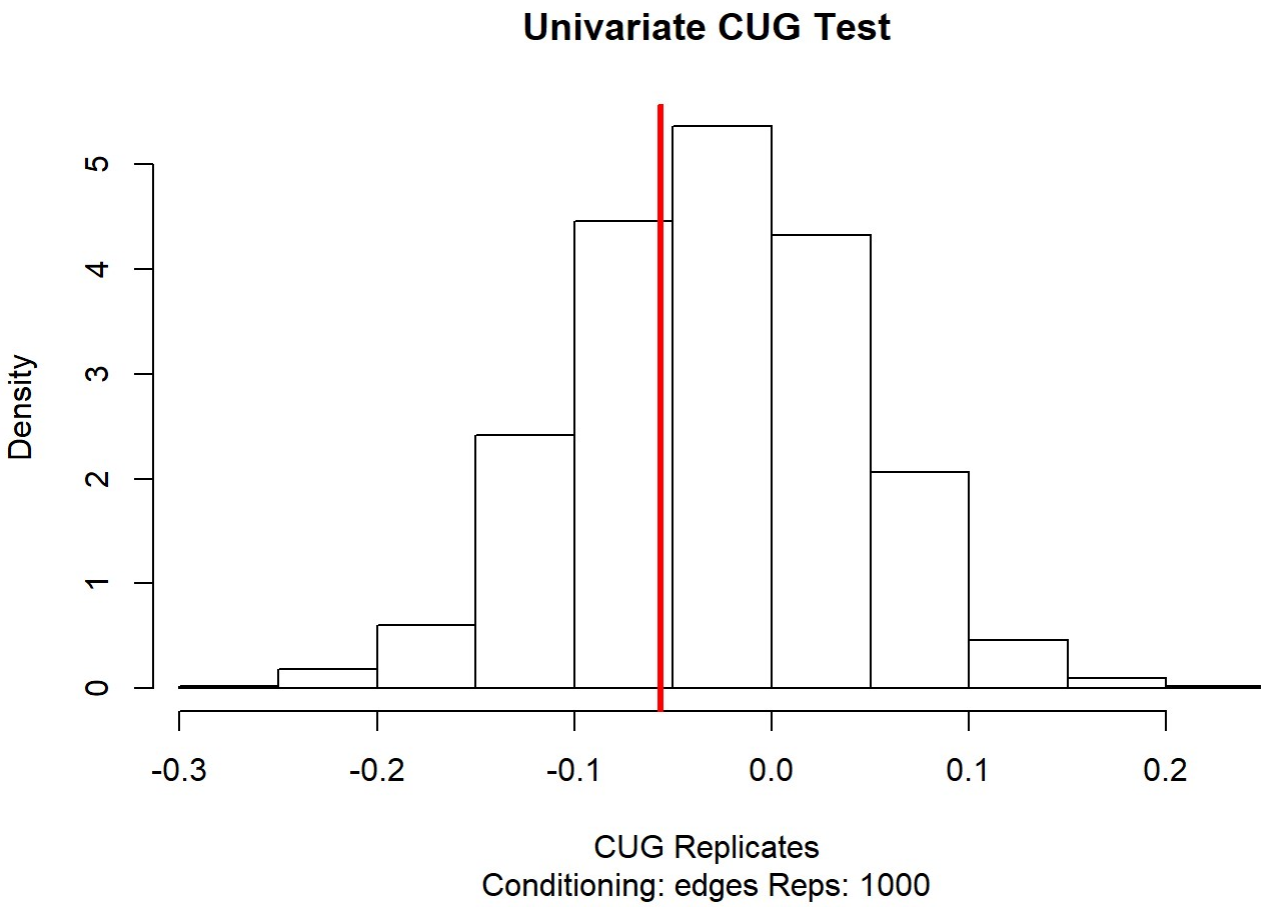
```
## [1] "----------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Occupied"
## [1]
## [1] "Vertex Values:"
##  [1] 0.8812944 0.9012961 0.9132786 0.9139958 0.9142603 0.9097222 0.8292284
##  [8] 0.9076855 0.9201469 0.9088314 0.9135567 0.8848686 0.9035277 0.8416051
## [15] 0.9126538 0.8581148 0.7249506 0.8010041 0.8820984 0.7811182 0.8589634
## [22] 0.8660516 0.7537092 0.8397798 0.8163131 0.8767700 0.7759617 0.8145812
## [29] 0.8370162 0.7904551 0.8666853 0.7952835 0.8424216 0.7761138 0.7058824
## [36] 0.8110459 0.8649698 0.8974429
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.7059  0.8119  0.8620  0.8498  0.9030  0.9201
## [1]
## [1] "Assortativity: -0.02647086155005"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.02647086
## Pr(X>=Obs): 0.482
## Pr(X<=Obs): 0.518
##
## [1]
```



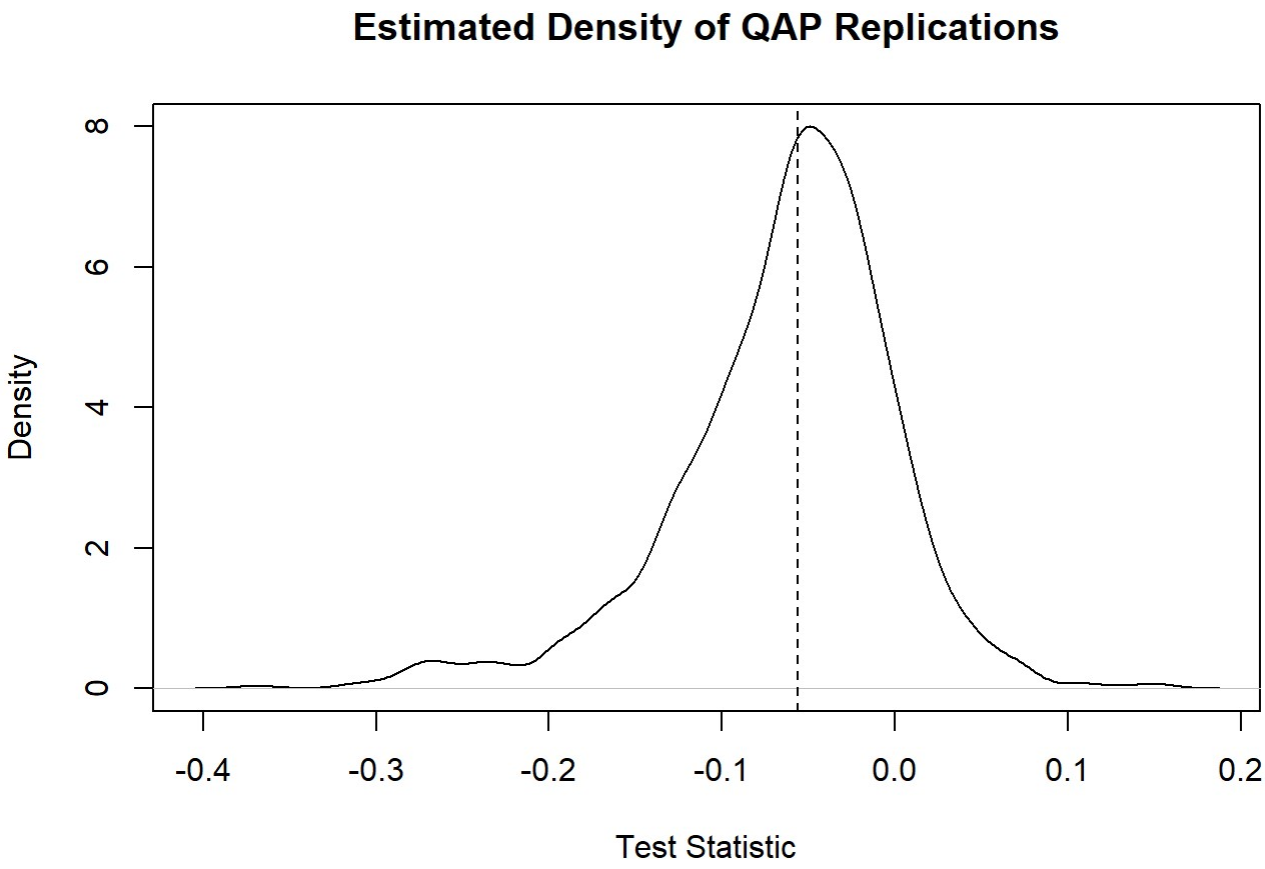

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.268
##  p(f(perm) <= f(d)): 0.732
##
## Test Diagnostics:
##  Test Value (f(d)): -0.02647086
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3526282
##      1stQ:    -0.09539917
##      Med:     -0.0536495
##      Mean:    -0.06215451
##      3rdQ:    -0.02316418
##      Max:     0.1103712
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
8
-0.4
-0.3
-0.2
-0.1
0.0
0.1
Test Statistic

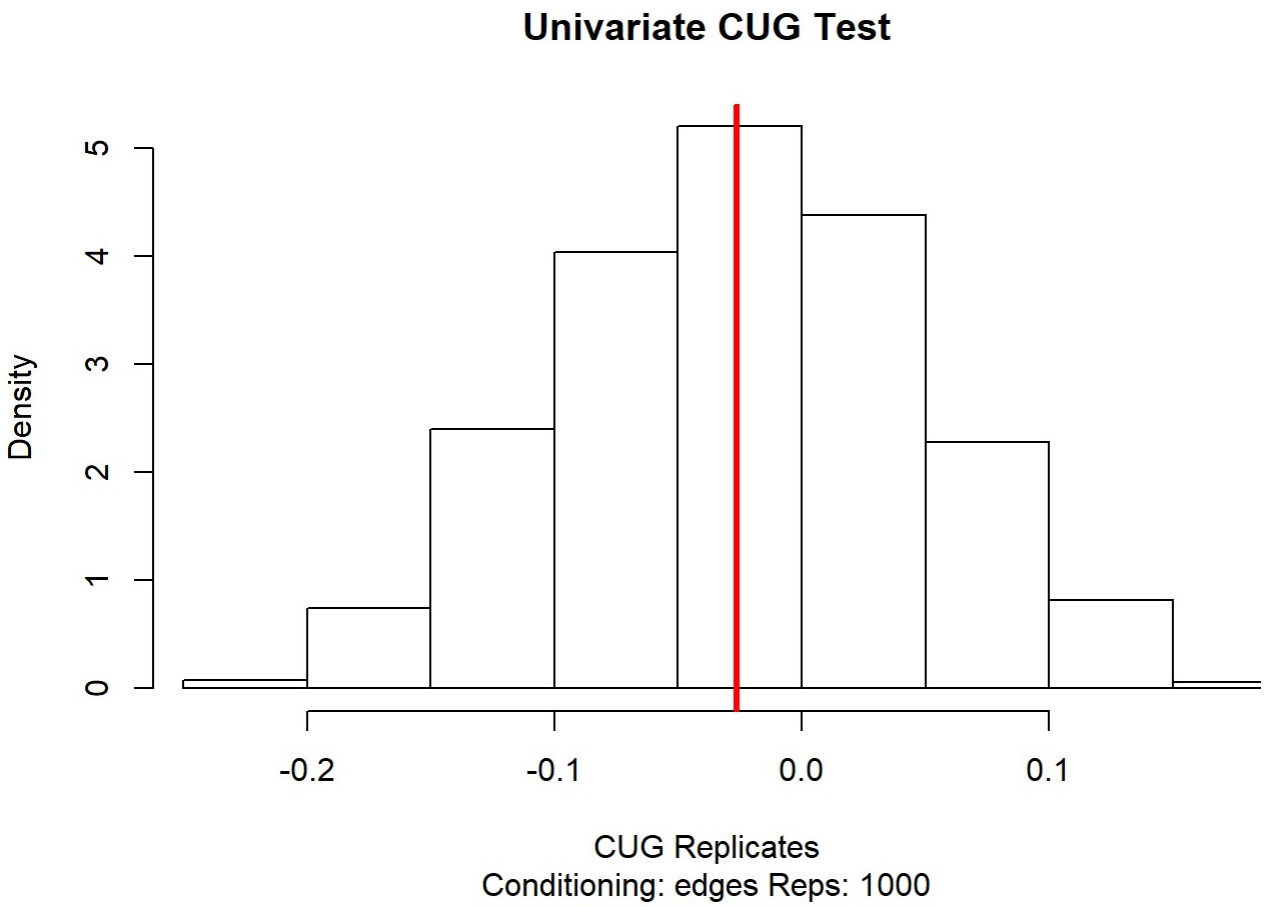
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Rented"
## [1]
## [1] "Vertex Values:"
##  [1] 0.6223698 0.4572562 0.6204788 0.5679695 0.5761394 0.5198568 0.4567699
##  [8] 0.5548326 0.4111689 0.4917484 0.4716087 0.5540126 0.5896078 0.5550468
## [15] 0.5763413 0.5112786 0.5227048 0.6002967 0.5267165 0.5826418 0.5766060
## [22] 0.6559394 0.4438877 0.4173810 0.6039792 0.5613732 0.6002972 0.6104102
## [29] 0.5115371 0.4702436 0.3694545 0.5044531 0.4465262 0.4137237 0.5086100
## [36] 0.5393439 0.4569800 0.5674542
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3695  0.4706  0.5330  0.5270  0.5765  0.6559
## [1]
## [1] "Assortativity: -0.0563947636064971"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.05639476
## Pr(X>=Obs): 0.646
## Pr(X<=Obs): 0.354
##
## [1]
```



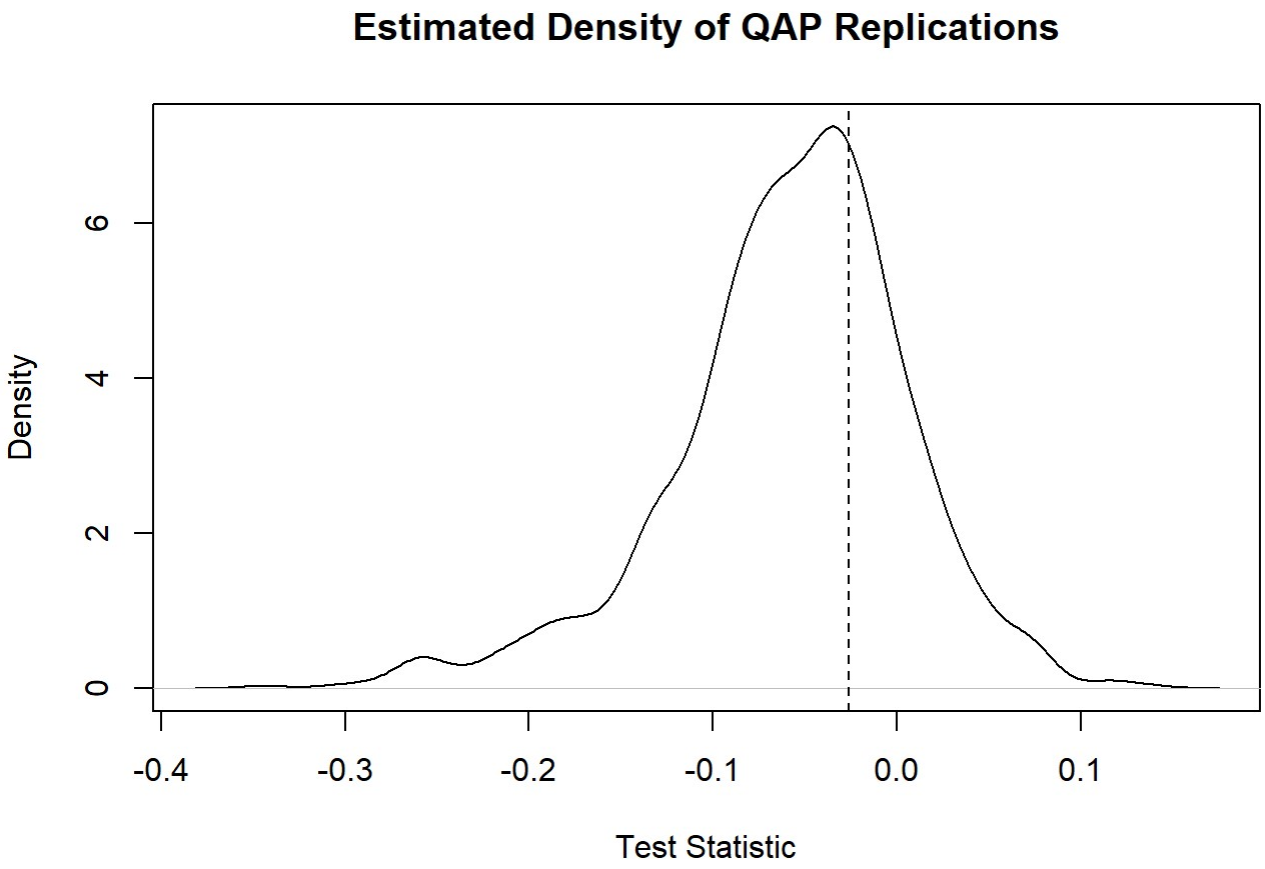

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.51
##  p(f(perm) <= f(d)): 0.49
##
## Test Diagnostics:
##  Test Value (f(d)): -0.05639476
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3690176
##      1stQ:    -0.09453412
##      Med:     -0.05491713
##      Mean:    -0.06338699
##      3rdQ:    -0.02330706
##      Max:     0.1511217
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
8
-0.4
-0.3
-0.2
-0.1
0.0
0.1
0.2
Test Statistic

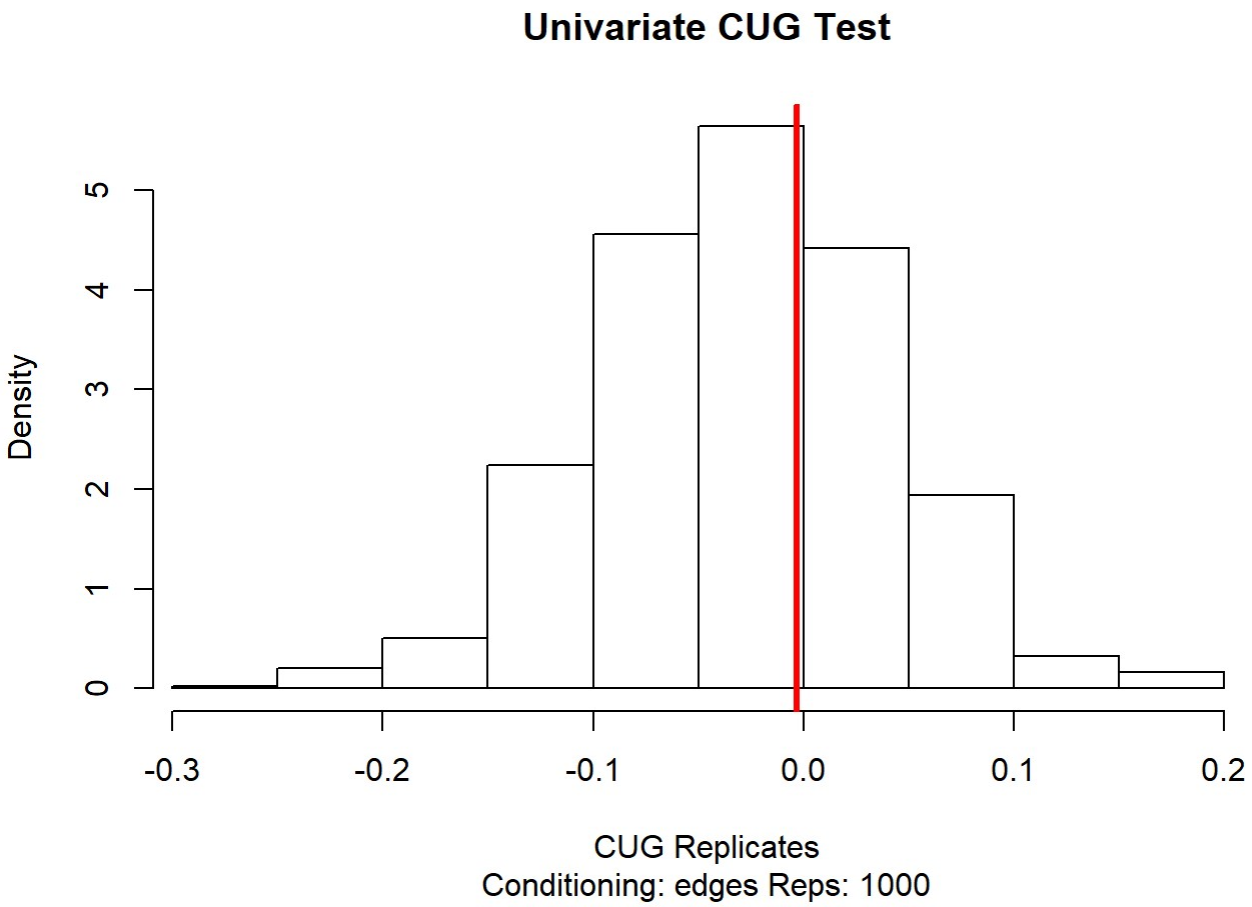
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Vacant"
## [1]
## [1] "Vertex Values:"
##  [1] 0.11870556 0.09870385 0.08672137 0.08600422 0.08573972 0.09027778
##  [7] 0.17077160 0.09231454 0.07985313 0.09116860 0.08644326 0.11513136
## [13] 0.09647233 0.15839488 0.08734616 0.14188517 0.27504936 0.19899589
## [19] 0.11790159 0.21888178 0.14103657 0.13394837 0.24629080 0.16022022
## [25] 0.18368695 0.12322995 0.22403830 0.18541882 0.16298376 0.20954490
## [31] 0.13331469 0.20471651 0.15757838 0.22388616 0.29411765 0.18895405
## [37] 0.13503022 0.10255709
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.07985 0.09703 0.13803 0.15019 0.18807 0.29412
## [1]
## [1] "Assortativity: -0.0264708615500305"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.02647086
## Pr(X>=Obs): 0.519
## Pr(X<=Obs): 0.481
##
## [1]
```



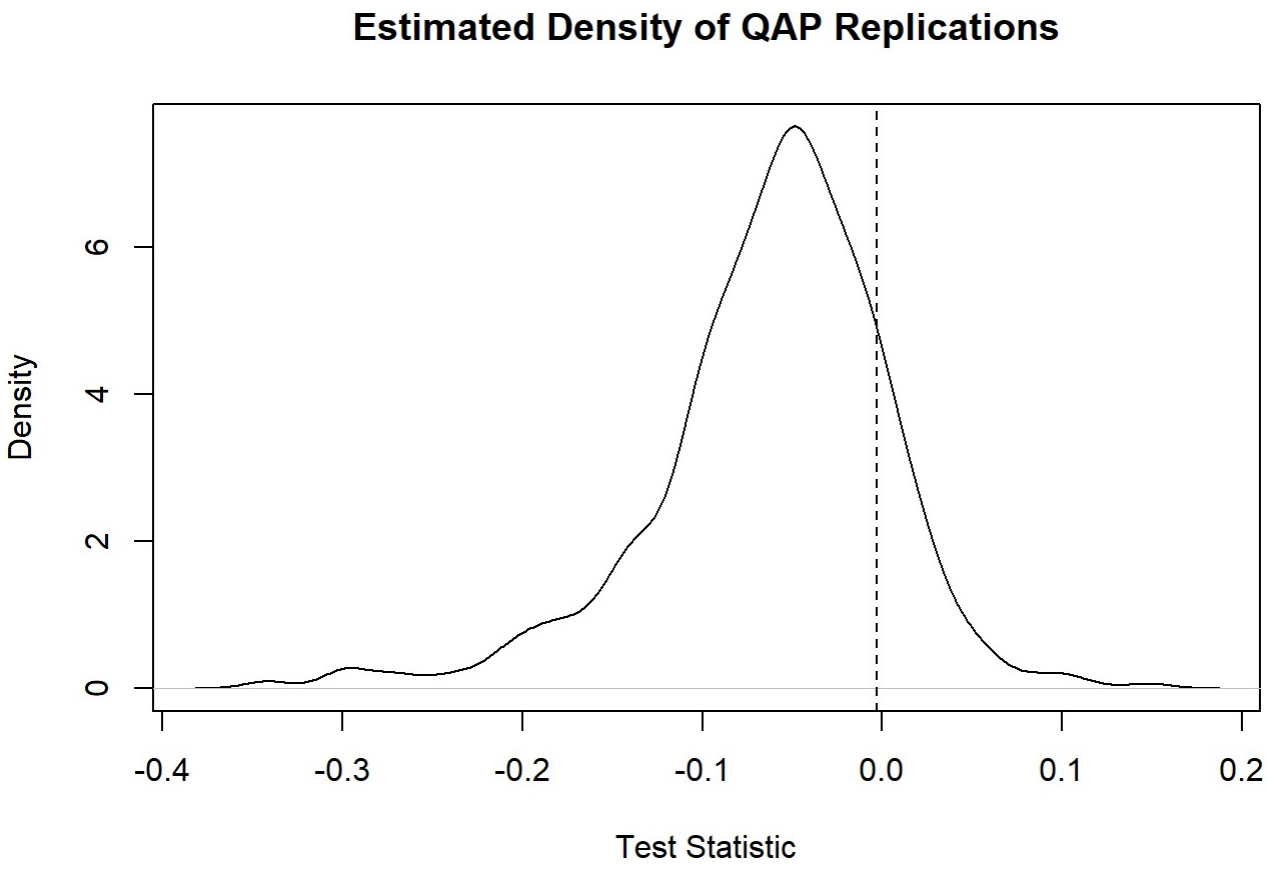

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.311
##  p(f(perm) <= f(d)): 0.689
##
## Test Diagnostics:
##  Test Value (f(d)): -0.02647086
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3445871
##      1stQ:    -0.09130096
##      Med:     -0.05253941
##      Mean:    -0.05859339
##      3rdQ:    -0.01760673
##      Max:     0.137544
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
-0.4
-0.3
-0.2
-0.1
0.0
0.1
Test Statistic

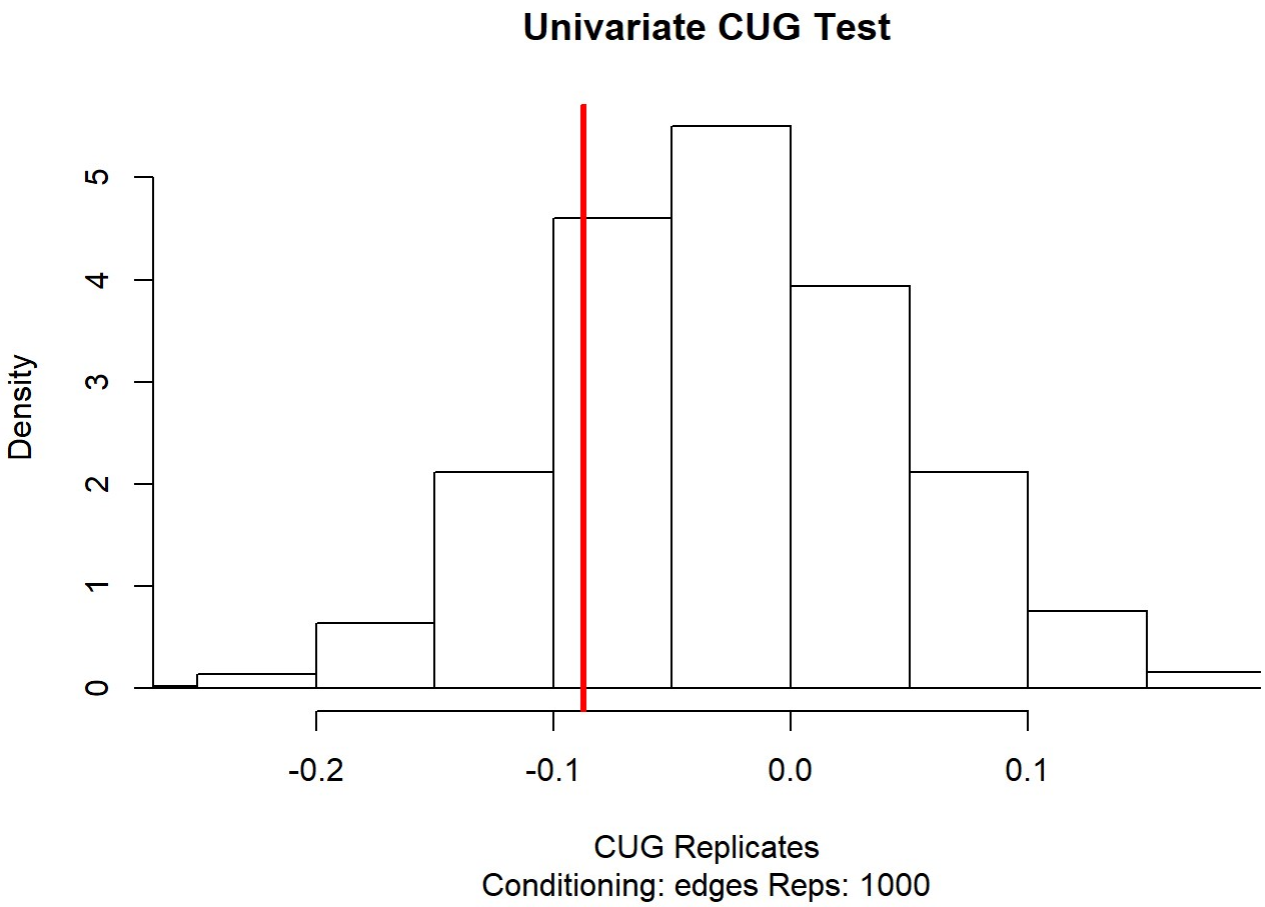
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Owned"
## [1]
## [1] "Vertex Values:"
##  [1] 0.2589247 0.4440399 0.2927999 0.3460263 0.3381209 0.3898655 0.3724585
##  [8] 0.3528529 0.5089779 0.4170830 0.4419480 0.3308560 0.3139198 0.2865583
## [15] 0.3363126 0.3468362 0.2022458 0.2007074 0.3553819 0.1984764 0.2823574
## [22] 0.2101123 0.3098215 0.4223988 0.2123339 0.3153968 0.1756645 0.2041710
## [29] 0.3254792 0.3202115 0.4972308 0.2908304 0.3958954 0.3623901 0.1972724
## [36] 0.2717020 0.4079898 0.3299888
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1757  0.2744  0.3277  0.3228  0.3699  0.5090
## [1]
## [1] "Assortativity: -0.00311298141018004"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.003112981
## Pr(X>=Obs): 0.36
## Pr(X<=Obs): 0.64
##
## [1]
```



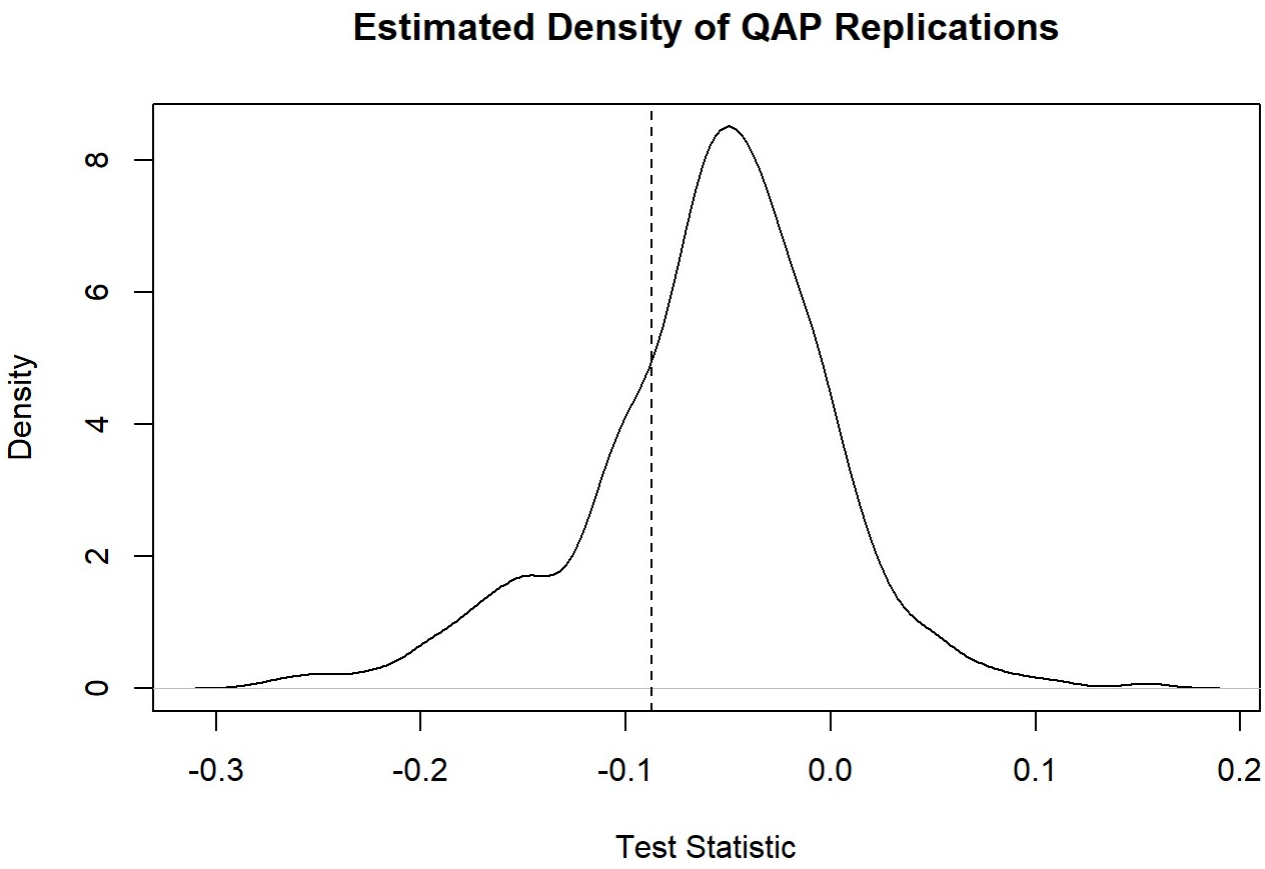

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.152
##  p(f(perm) <= f(d)): 0.848
##
## Test Diagnostics:
##  Test Value (f(d)): -0.003112981
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3446509
##      1stQ:    -0.09418767
##      Med:     -0.05369393
##      Mean:    -0.0619164
##      3rdQ:    -0.02029423
##      Max:     0.1495336
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
-0.4
-0.3
-0.2
-0.1
0.0
0.1
0.2
Test Statistic

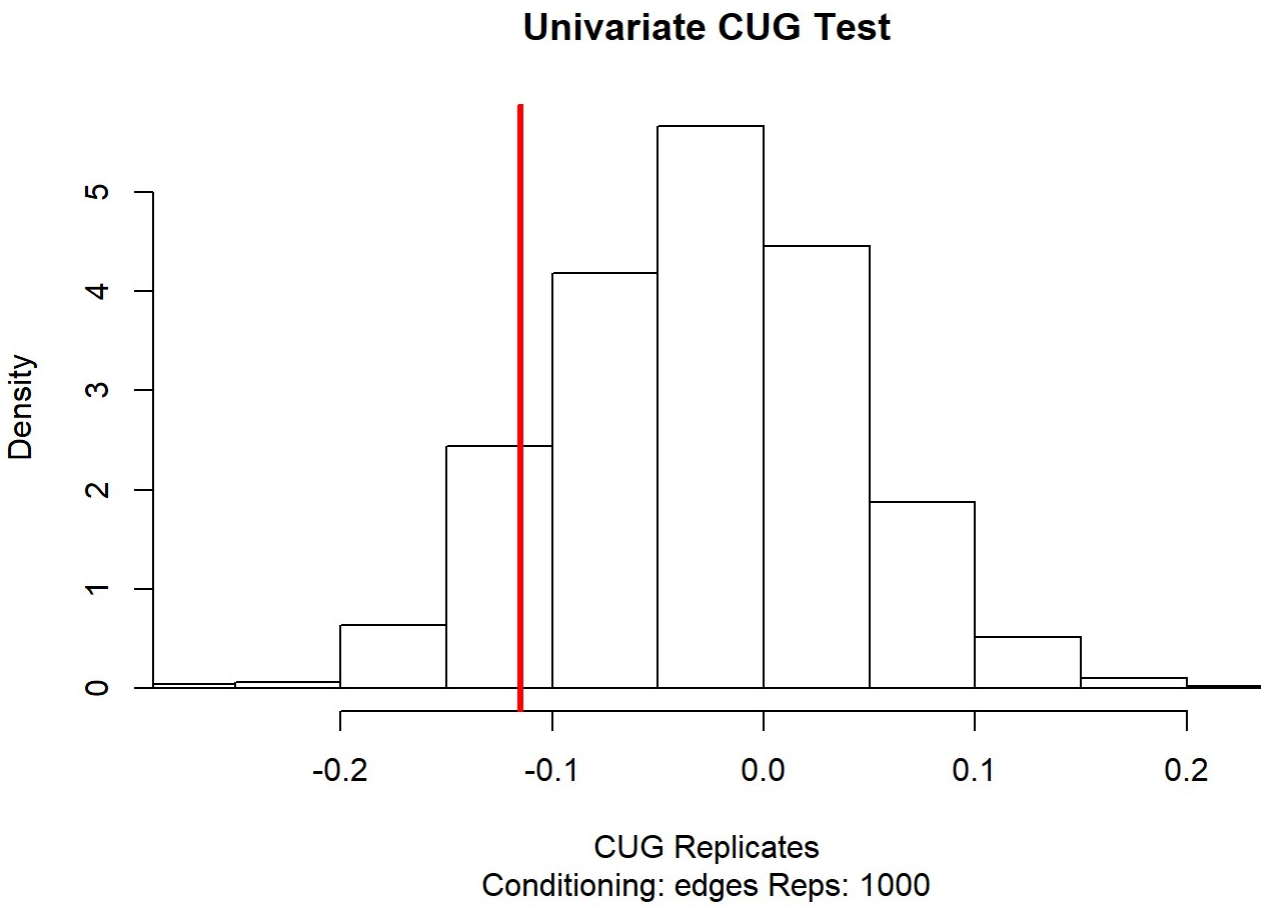
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Median.Age"
## [1]
## [1] "Vertex Values:"
##  [1] 32.4 35.1 35.5 33.5 30.4 28.9 35.2 31.1 36.2 33.3 30.3 31.2 30.7 28.6
## [15] 31.3 33.2 30.6 30.9 30.6 28.2 27.6 28.6 30.5 33.8 35.8 31.6 31.4 37.0
## [29] 41.4 33.0 37.7 27.2 28.0 31.5 30.5 33.9 37.0 36.6
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   27.20   30.50   31.45   32.38   34.80   41.40
## [1]
## [1] "Assortativity: -0.0873722429477236"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.08737224
## Pr(X>=Obs): 0.799
## Pr(X<=Obs): 0.201
##
## [1]
```



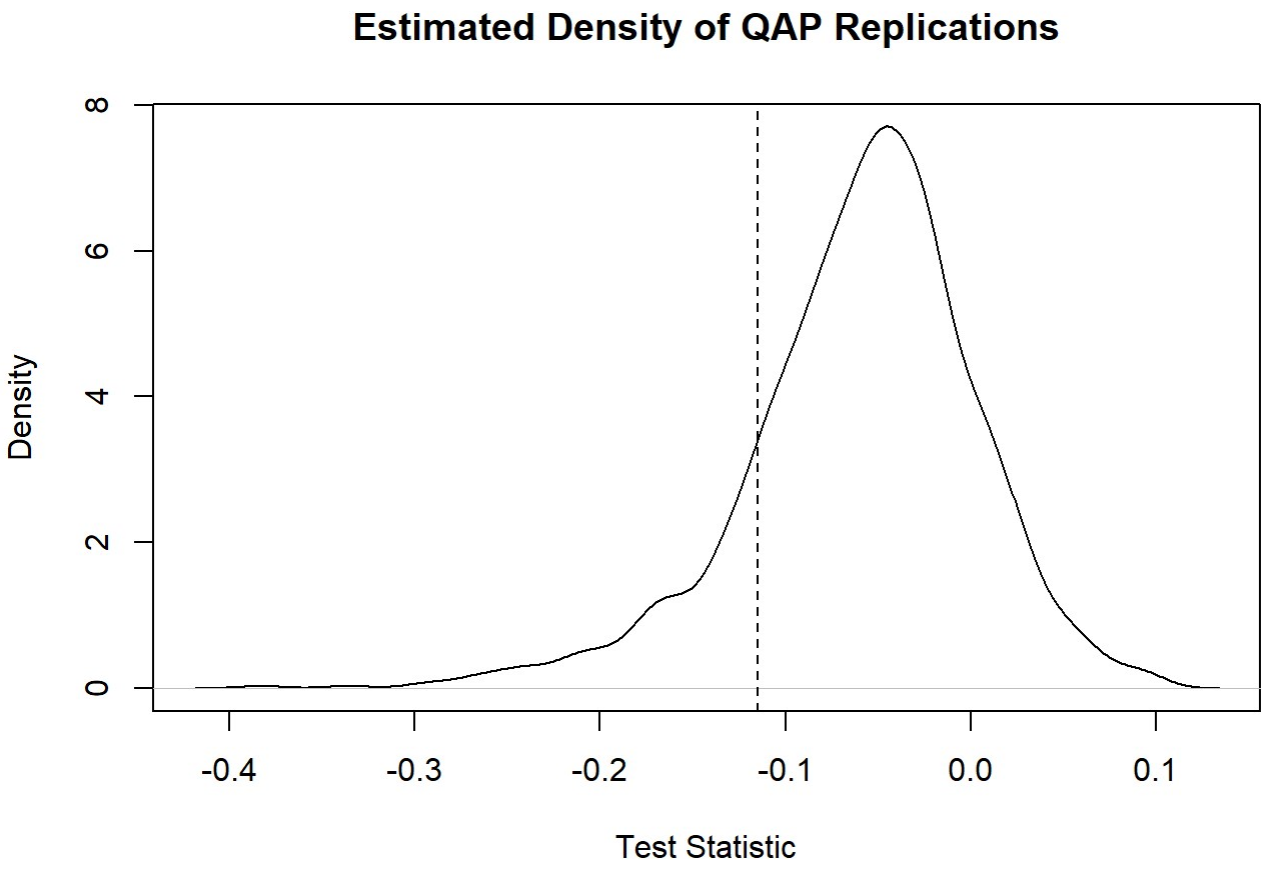

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.743
##  p(f(perm) <= f(d)): 0.257
##
## Test Diagnostics:
##  Test Value (f(d)): -0.08737224
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.2774502
##      1stQ:    -0.08862126
##      Med:     -0.05216745
##      Mean:    -0.05858892
##      3rdQ:    -0.02300809
##      Max:     0.1563561
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
8
-0.3
-0.2
-0.1
0.0
0.1
0.2
Test Statistic

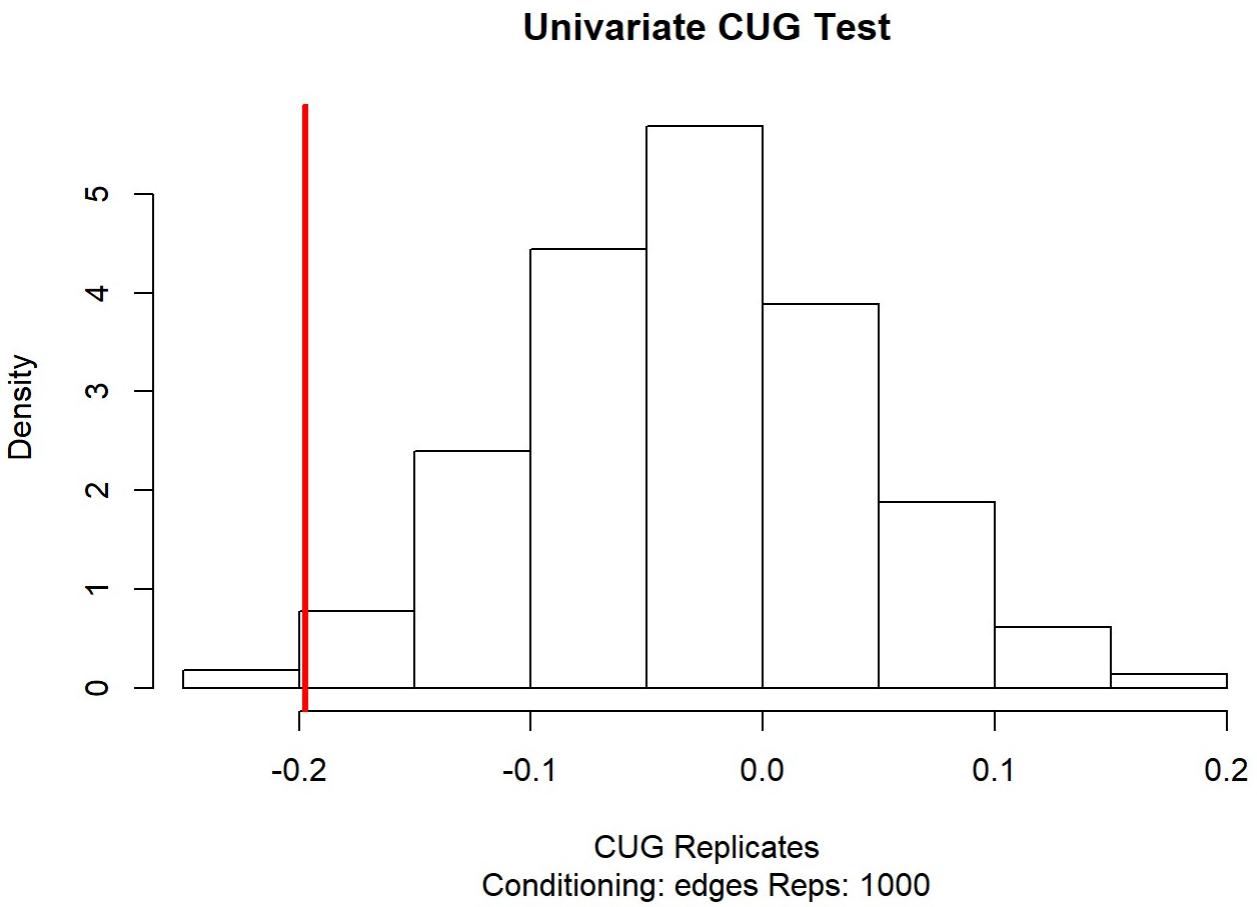
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Total.Population"
## [1]
## [1] "Vertex Values:"
##  [1] 54991 71942 56362 39493 94368 64116 80484 51542 64124 53359 78743
## [12] 39262 72791 56323 82236 98514 18001 20567 54881 35912 79288 35769
## [23] 29283 21390 21929 25681 23740 52010 31028 31198 44619 44377 55628
## [34] 35505 30654 32602 48743 56521
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18001   31549   50143   49684   62217   98514
## [1]
## [1] "Assortativity: -0.115123920769111"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1151239
## Pr(X>=Obs): 0.891
## Pr(X<=Obs): 0.109
##
## [1]
```



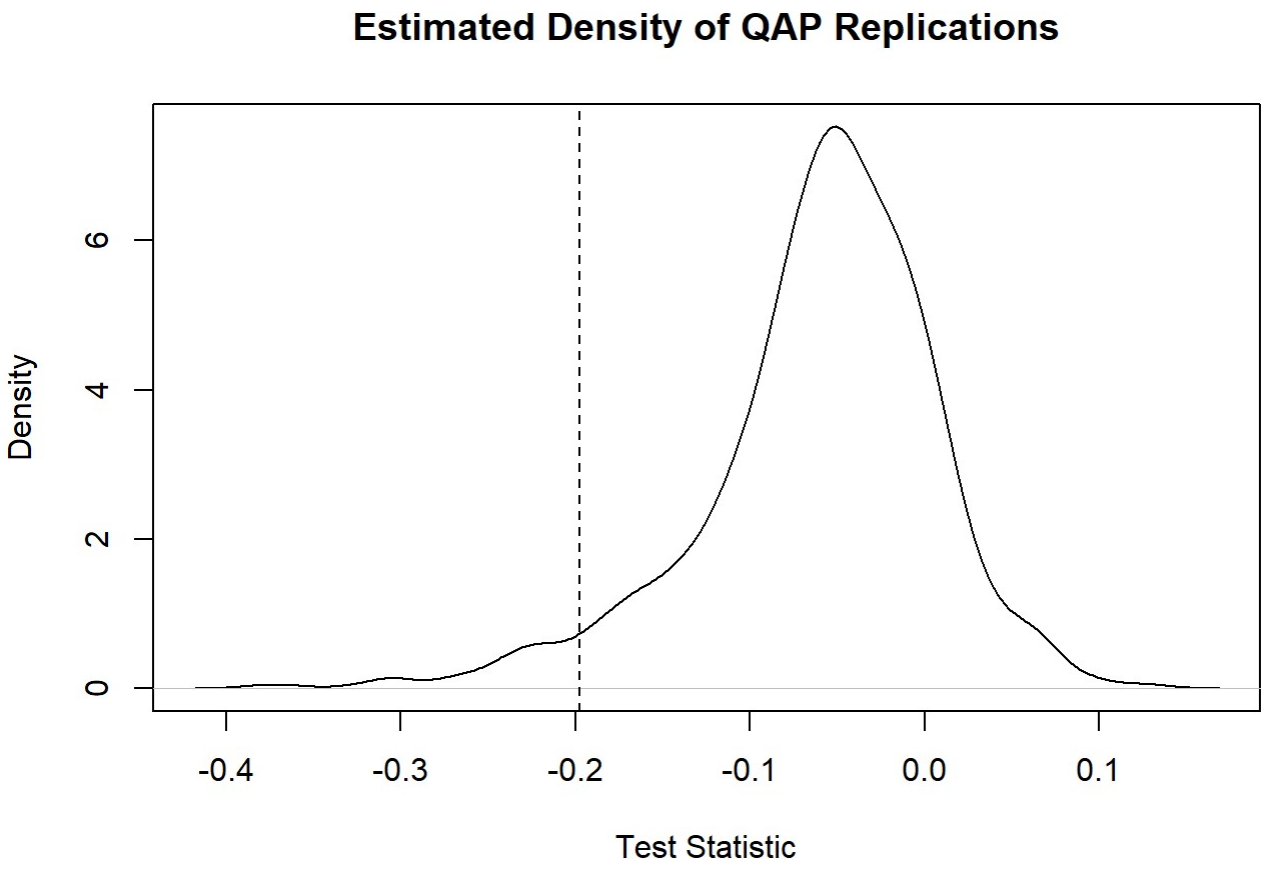

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.851
##  p(f(perm) <= f(d)): 0.149
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1151239
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3828664
##      1stQ:    -0.09212147
##      Med:     -0.0527475
##      Mean:    -0.05949181
##      3rdQ:    -0.02108359
##      Max:     0.0977861
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
8
-0.4
-0.3
-0.2
-0.1
0.0
0.1
Test Statistic

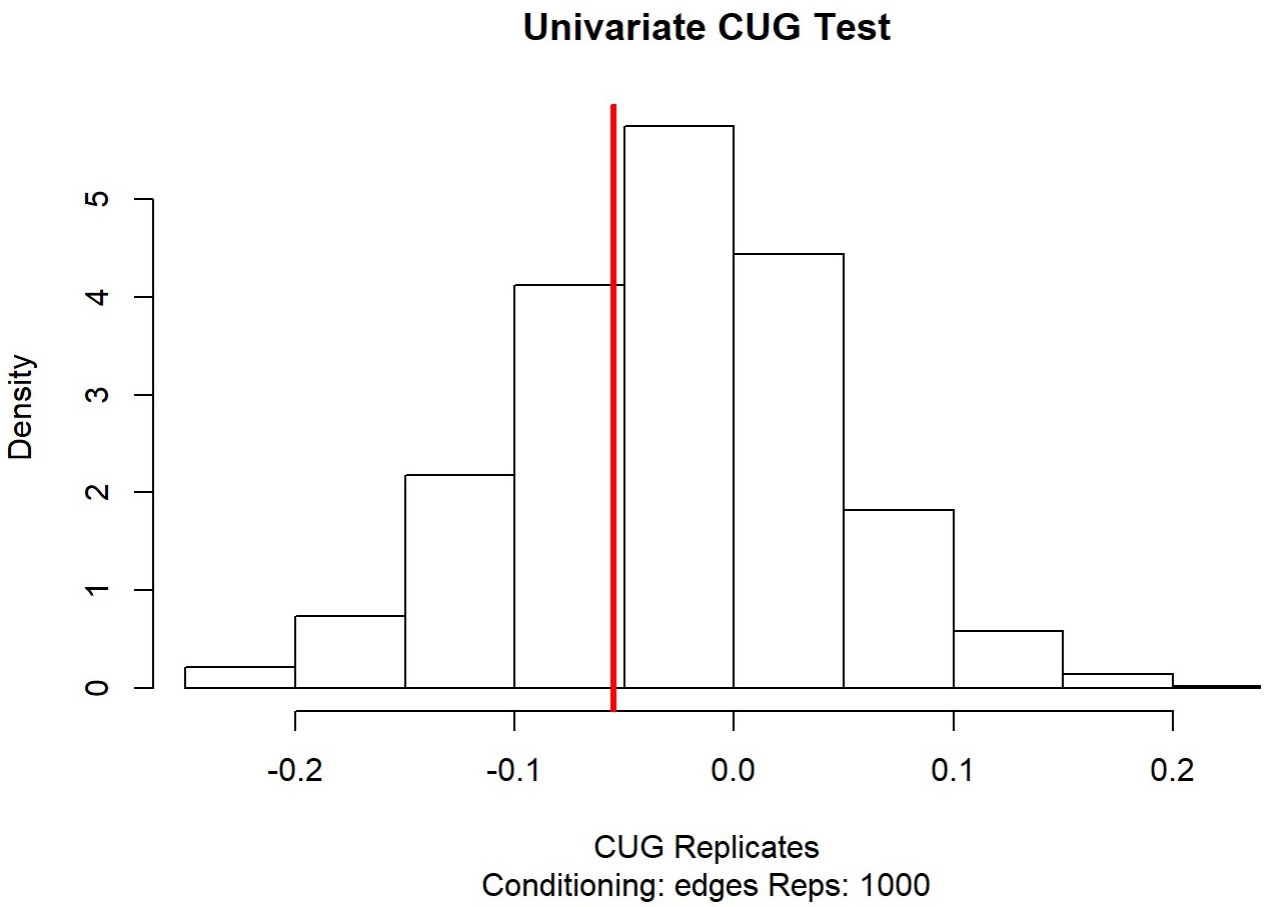
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Average.Household.Size"
## [1]
## [1] "Vertex Values:"
##  [1] 2.13 2.85 1.79 2.14 1.72 1.80 1.52 3.08 2.80 2.60 3.63 2.89 2.46 3.27
## [15] 2.13 2.96 3.04 2.81 1.94 3.05 3.79 2.92 1.60 1.75 2.19 1.81 2.37 2.16
## [29] 2.26 2.80 2.83 3.47 3.55 3.37 2.97 2.55 2.76 1.82
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.520   2.130   2.680   2.568   2.967   3.790
## [1]
## [1] "Assortativity: -0.197398802311909"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1973988
## Pr(X>=Obs): 0.989
## Pr(X<=Obs): 0.011
##
## [1]
```



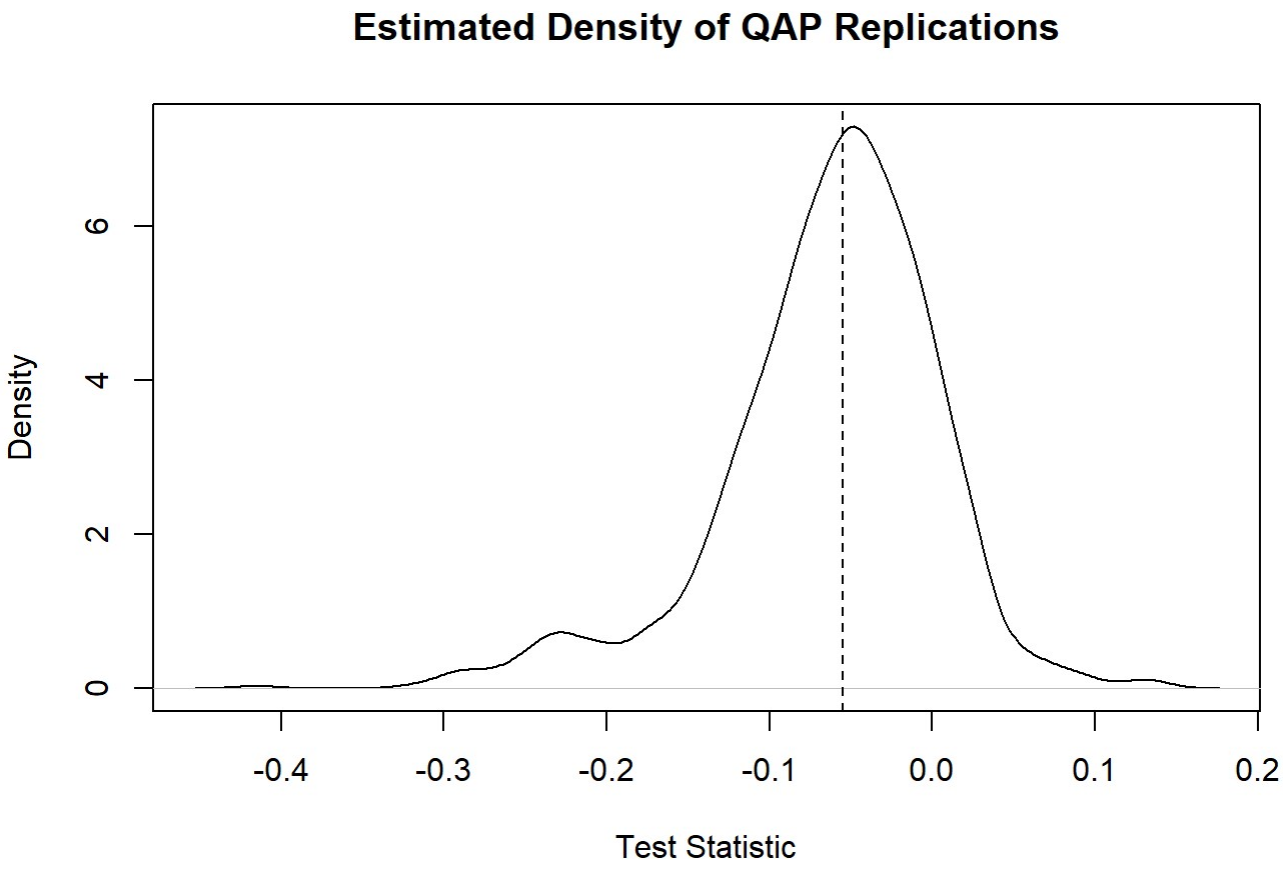

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.96
##  p(f(perm) <= f(d)): 0.04
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1973988
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3812624
##      1stQ:    -0.08971283
##      Med:     -0.05172346
##      Mean:    -0.05998133
##      3rdQ:    -0.01668918
##      Max:     0.1317905
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
-0.4
-0.3
-0.2
-0.1
0.0
0.1
Test Statistic

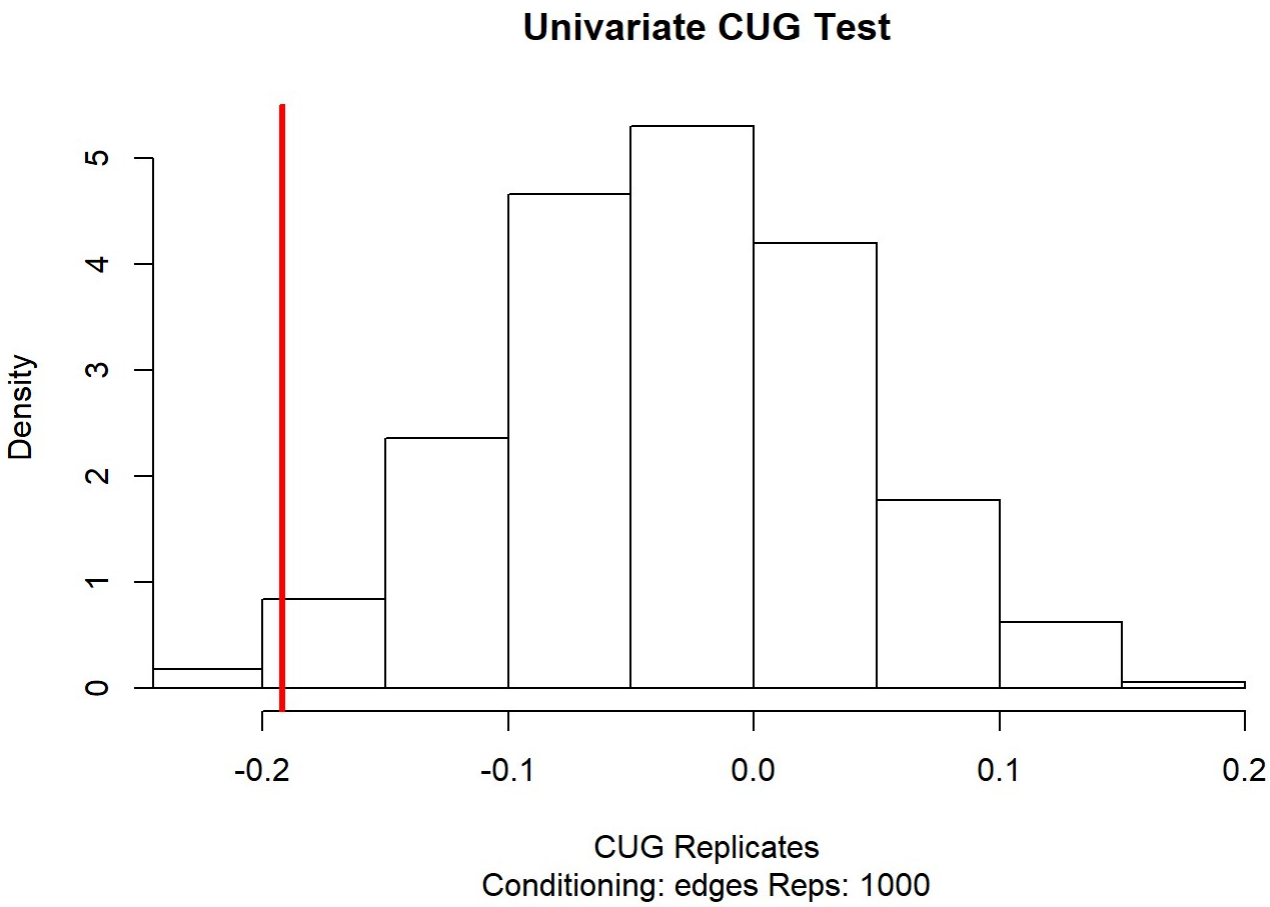
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.African"
## [1]
## [1] "Vertex Values:"
##  [1] 0.26297030 0.11140919 0.20004613 0.03767756 0.03868896 0.04293780
##  [7] 0.10849361 0.04027783 0.01334914 0.03253434 0.03165996 0.02524069
## [13] 0.05357805 0.40889157 0.07787344 0.85101610 0.96189101 0.90902903
## [19] 0.31528216 0.91431833 0.13083947 0.03097654 0.11477649 0.28120617
## [25] 0.94144740 0.30431058 0.86794440 0.94862526 0.97202527 0.74671453
## [31] 0.97373316 0.29627960 0.49261163 0.96262498 0.97367391 0.96895896
## [37] 0.97780194 0.14338034
## [1]
## [1] "Vertex Value Summary:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01335 0.05965 0.28874 0.43592 0.91300 0.97780
## [1]
## [1] "Assortativity: -0.0548089317331788"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.05480893
## Pr(X>=Obs): 0.662
## Pr(X<=Obs): 0.338
##
## [1]
```



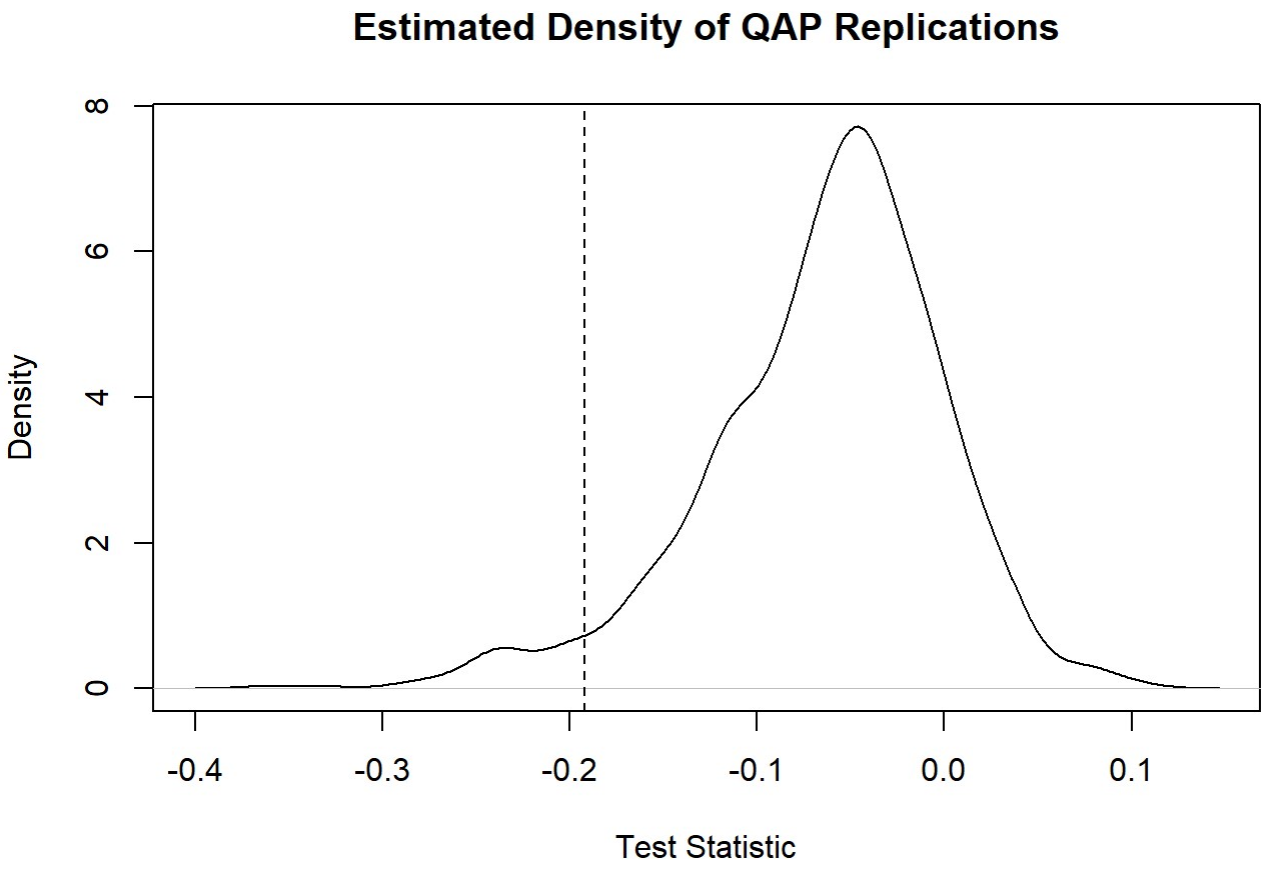

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.497
##  p(f(perm) <= f(d)): 0.503
##
## Test Diagnostics:
##  Test Value (f(d)): -0.05480893
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.4149466
##      1stQ:    -0.09620002
##      Med:     -0.05504076
##      Mean:    -0.06353608
##      3rdQ:    -0.02046273
##      Max:     0.137698
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
-0.4
-0.3
-0.2
-0.1
0.0
0.1
0.2
Test Statistic

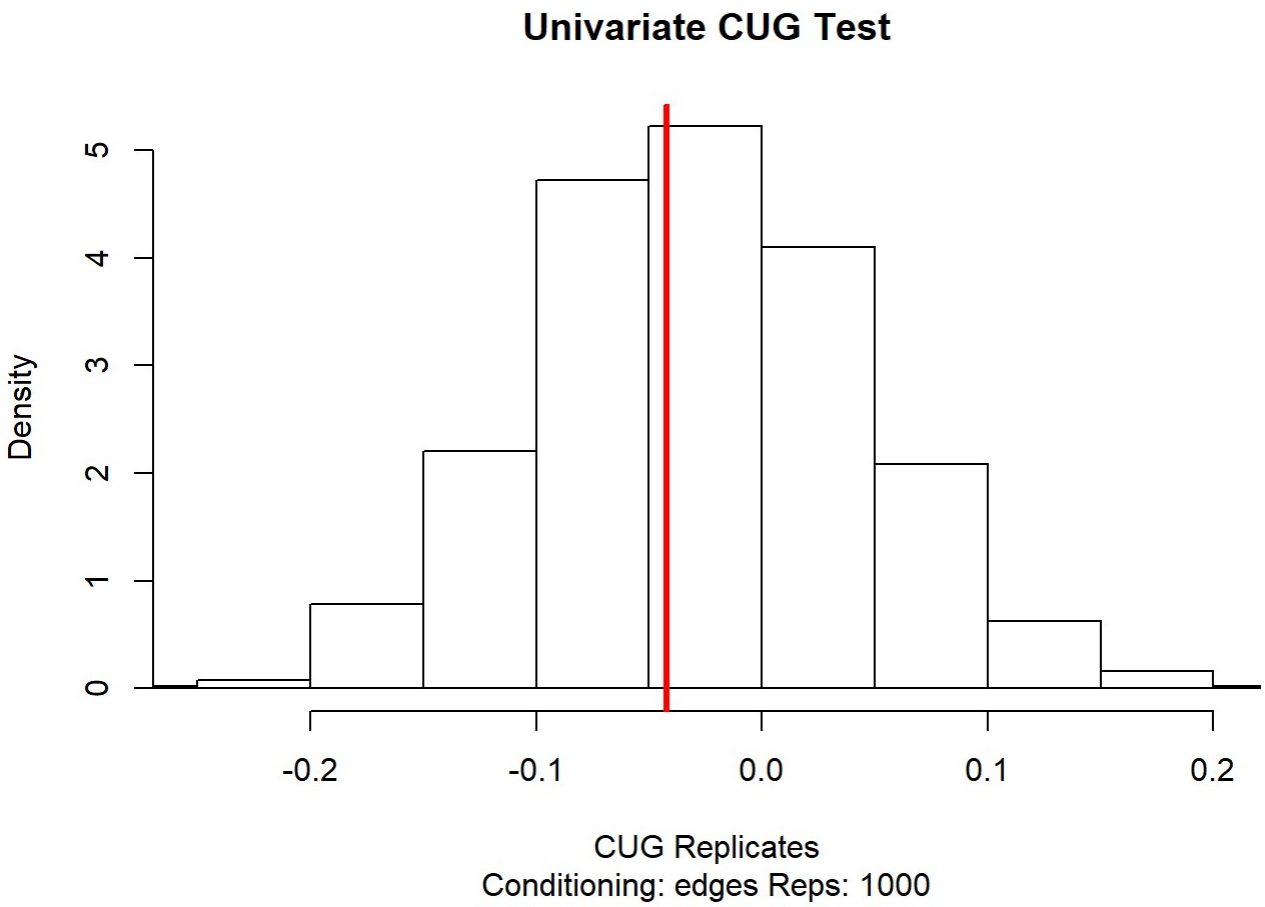
```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.White"
## [1]
## [1] "Vertex Values:"
##  [1] 0.393118874 0.426816046 0.516269827 0.630896615 0.803715242
##  [6] 0.828779088 0.721050147 0.292072485 0.535353378 0.416686969
## [11] 0.151873817 0.284397127 0.391724252 0.044422350 0.572146019
## [16] 0.044298272 0.007388478 0.033937862 0.419853866 0.013700156
## [21] 0.038543033 0.124325533 0.626950791 0.480645161 0.017830270
## [26] 0.466960009 0.068365628 0.013920400 0.003609643 0.019199949
## [31] 0.004213452 0.105843117 0.043485295 0.003661456 0.003425328
## [36] 0.005521134 0.002749113 0.546504839
## [1]
## [1] "Vertex Value Summary:"
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.002749 0.018173 0.138100 0.265901 0.477224 0.828779
## [1]
## [1] "Assortativity: -0.191861147123035"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1918611
## Pr(X>=Obs): 0.987
## Pr(X<=Obs): 0.013
##
## [1]
```



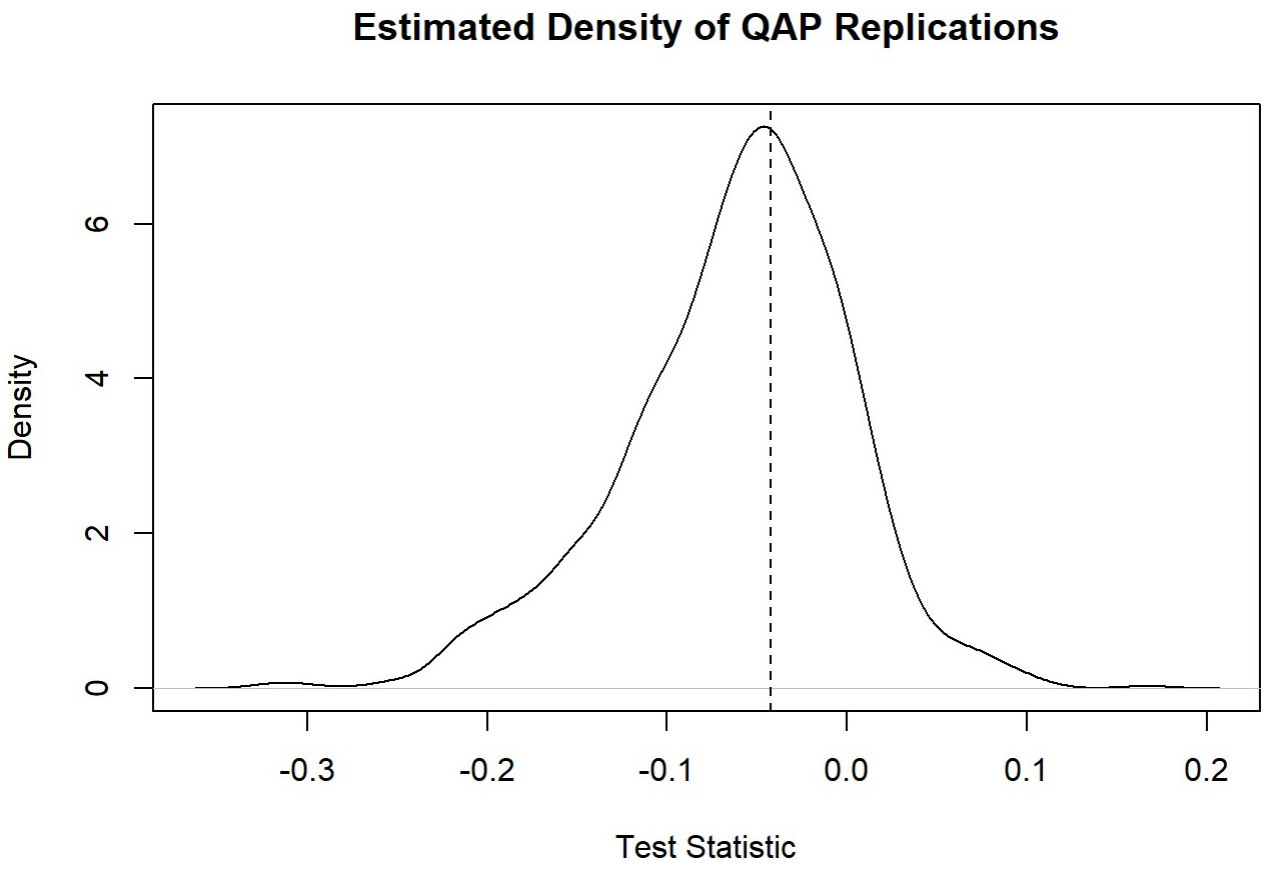

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.957
##  p(f(perm) <= f(d)): 0.043
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1918611
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3619222
##      1stQ:    -0.09867838
##      Med:     -0.05426579
##      Mean:    -0.0640744
##      3rdQ:    -0.02271524
##      Max:     0.1082034
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
8
-0.4
-0.3
-0.2
-0.1
0.0
0.1
Test Statistic

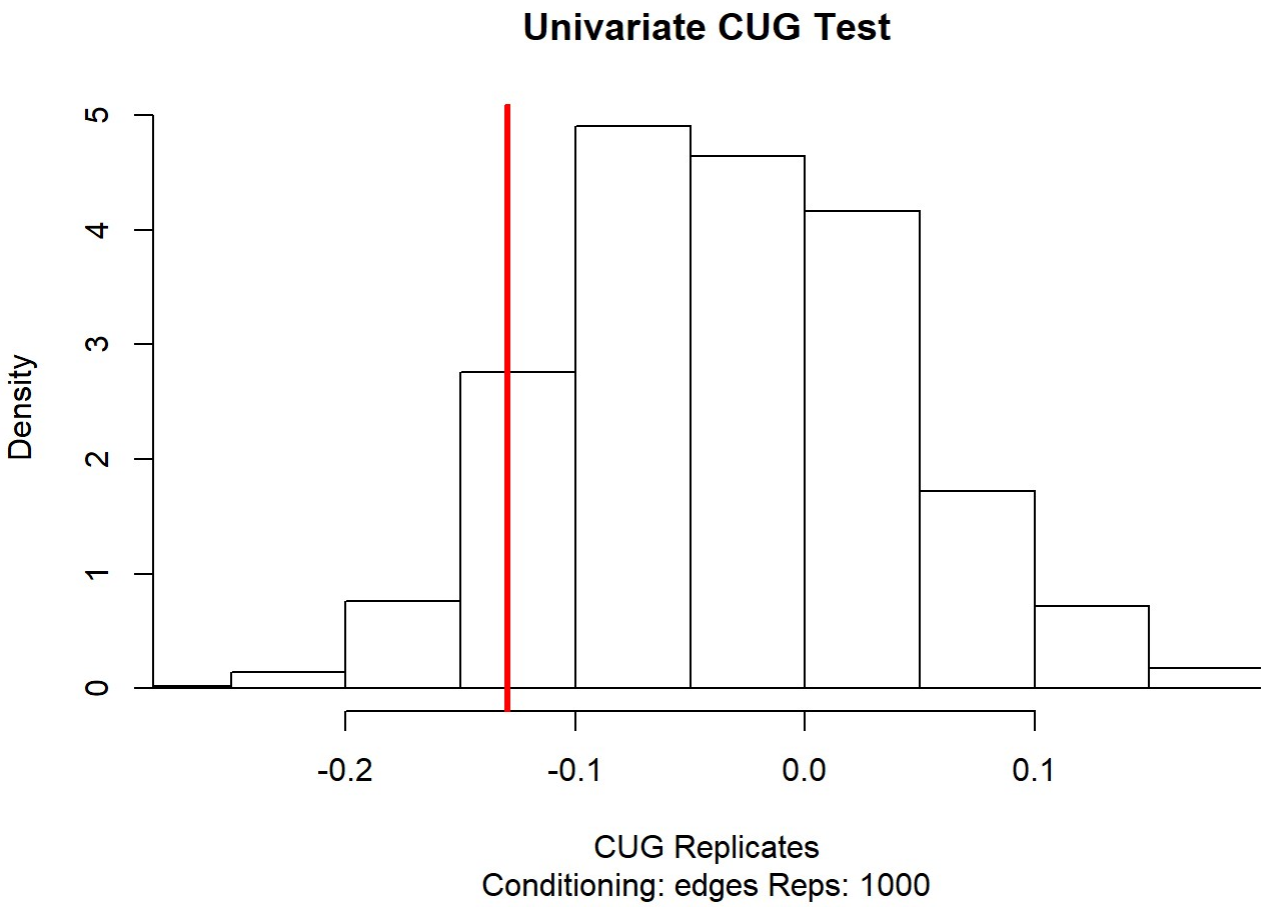
```
## [1] "--------------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Asian"
## [1]
## [1] "Vertex Values:"
##  [1] 0.0640650288 0.2249589947 0.1138000781 0.1113868280 0.0599037809
##  [6] 0.0514068251 0.1011381144 0.1441542819 0.0463477013 0.0700163047
## [11] 0.0195191954 0.0304110845 0.0251129947 0.0039415514 0.0377693468
## [16] 0.0058164322 0.0004444198 0.0040355910 0.1461708059 0.0020605926
## [21] 0.0014251841 0.0103721099 0.1590684015 0.1546049556 0.0025992977
## [26] 0.1240216502 0.0229149115 0.0023841569 0.0008701818 0.0021475736
## [31] 0.0006051234 0.0163823602 0.0034155461 0.0005351359 0.0011091538
## [36] 0.0006441323 0.0006770203 0.1164522921
## [1]
## [1] "Vertex Value Summary:"
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0004444 0.0022067 0.0212171 0.0495444 0.0933577 0.2249590
## [1]
## [1] "Assortativity: -0.0421017492302049"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.04210175
## Pr(X>=Obs): 0.569
## Pr(X<=Obs): 0.431
##
## [1]
```



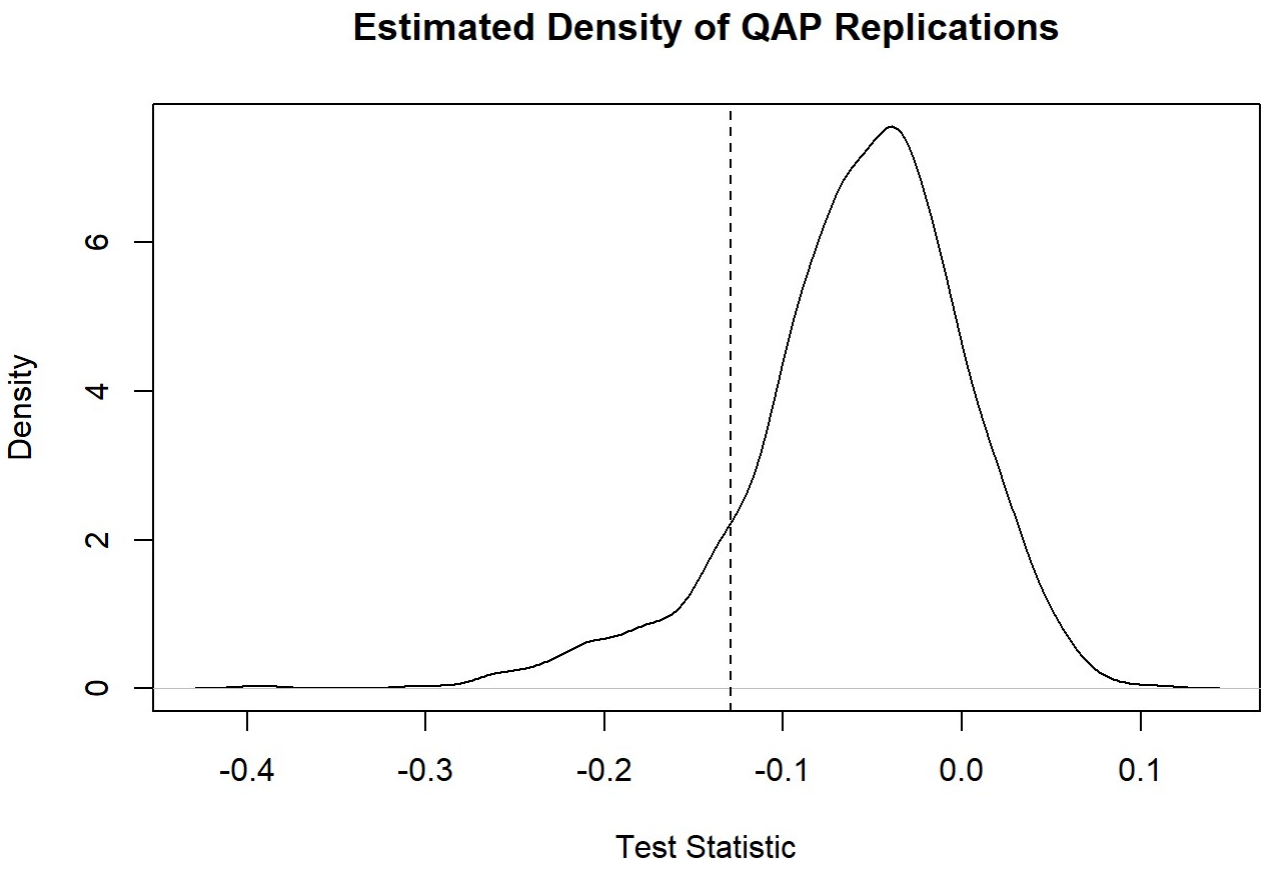

```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.407
##  p(f(perm) <= f(d)): 0.593
##
## Test Diagnostics:
##  Test Value (f(d)): -0.04210175
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.3225944
##      1stQ:    -0.09956357
##      Med:     -0.0546711
##      Mean:    -0.06168347
##      3rdQ:    -0.01982576
##      Max:     0.1664987
##
## [1]
```


Estimated Density of QAP Replications
Density
0
2
4
6
-0.3
-0.2
-0.1
0.0
0.1
0.2
Test Statistic

```
## [1] "-------------------------------------------------------------------------
-"
## [1] "Testing Assortativity for attribute: Prop.Hispanic"
## [1]
## [1] "Vertex Values:"
##  [1] 0.244276336 0.204345167 0.142099287 0.191476971 0.076275856
##  [6] 0.055695926 0.049376274 0.494489931 0.387701952 0.456005547
## [11] 0.788654230 0.644261627 0.512357297 0.533458800 0.290639136
## [16] 0.088535640 0.019332259 0.041328342 0.091980831 0.059701493
## [21] 0.825559984 0.824345103 0.068640508 0.056194483 0.018012677
## [26] 0.063315291 0.021229992 0.017169775 0.010023205 0.217129303
## [31] 0.010264685 0.573067129 0.451948659 0.021799747 0.010602205
## [36] 0.011901110 0.009416737 0.164859079
## [1]
## [1] "Vertex Value Summary:"
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.009417 0.026682 0.090258 0.230197 0.435887 0.825560
## [1]
## [1] "Assortativity: -0.129441754347806"
## [1]
## [1] "Assortativity CUG Test Results:"
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: -0.1294418
## Pr(X>=Obs): 0.909
## Pr(X<=Obs): 0.091
##
## [1]
```




```
## [1] "Assortativity QAP Test Results:"
##
## QAP Test Results
##
## Estimated p-values:
##  p(f(perm) >= f(d)): 0.888
##  p(f(perm) <= f(d)): 0.112
##
## Test Diagnostics:
##  Test Value (f(d)): -0.1294418
##  Replications: 1000
##  Distribution Summary:
##      Min:     -0.393291
##      1stQ:    -0.08965499
##      Med:     -0.05247651
##      Mean:    -0.05815564
##      3rdQ:    -0.01804077
##      Max:     0.1075637
##
## [1]
```




```
if (save_to_file){
  sink()

  close(outFile)
  closeAllConnections()
}


##################### Transitivity ################################################
################
```

Violent crime projection transitivity analysis

```
dir.create(file.path(output_path, "Transitivity"), showWarnings = FALSE)
trans_path = paste(output_path, "Transitivity\\", sep = "")

if (save_to_file){
  out_file_name = paste(trans_path, "Violent_Transitivity.txt", sep = '')
  outFile = file(out_file_name, open="wt")
  sink(file = outFile, append = TRUE)
}

v_local_trans = mean(transitivity(violent_filter, type = "local"))
print(paste("Mean Local Transitivity:", v_local_trans))
```

```
## [1] "Mean Local Transitivity: NaN"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
temp_cug = mycugtest(violent_filter, transitivity, cmode = "edges", type = "local")

print("Local Transitivity CUG Test Results:")
```

```
## [1] "Local Transitivity CUG Test Results:"
```

```
print.cug.test(temp_cug)
```

```
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: 1 NaN 1 NaN 0.7666667 NaN 1 NaN 0.9848485 1 0.1882353 1 1 1 0.3733
333 1 1 NaN 1 NaN 1 0.4021739 0.8901099 1 0.9487179 1 NaN 1 NaN 1 0.9487179 0.8285714
0.7666667 NaN 0.5906433 NaN
## Pr(X>=Obs): NA
## Pr(X<=Obs): NA
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
if (save_to_file){
  tmp_cug = paste(trans_path, "Violent_Local_Transitivity_CUG.jpg", sep = '')
  jpeg(file = tmp_cug)
  plot.cug.test(temp_cug)
  dev.off()
}

v_global_trans = transitivity(violent_filter, type = "global")
print(paste("Global Transitivity:", v_global_trans))
```

```
## [1] "Global Transitivity: 0.55659121171771"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
temp_cug = mycugtest(violent_filter, transitivity, cmode = "edges", type = "global")

print("Global Transitivity CUG Test Results:")
```

```
## [1] "Global Transitivity CUG Test Results:"
```

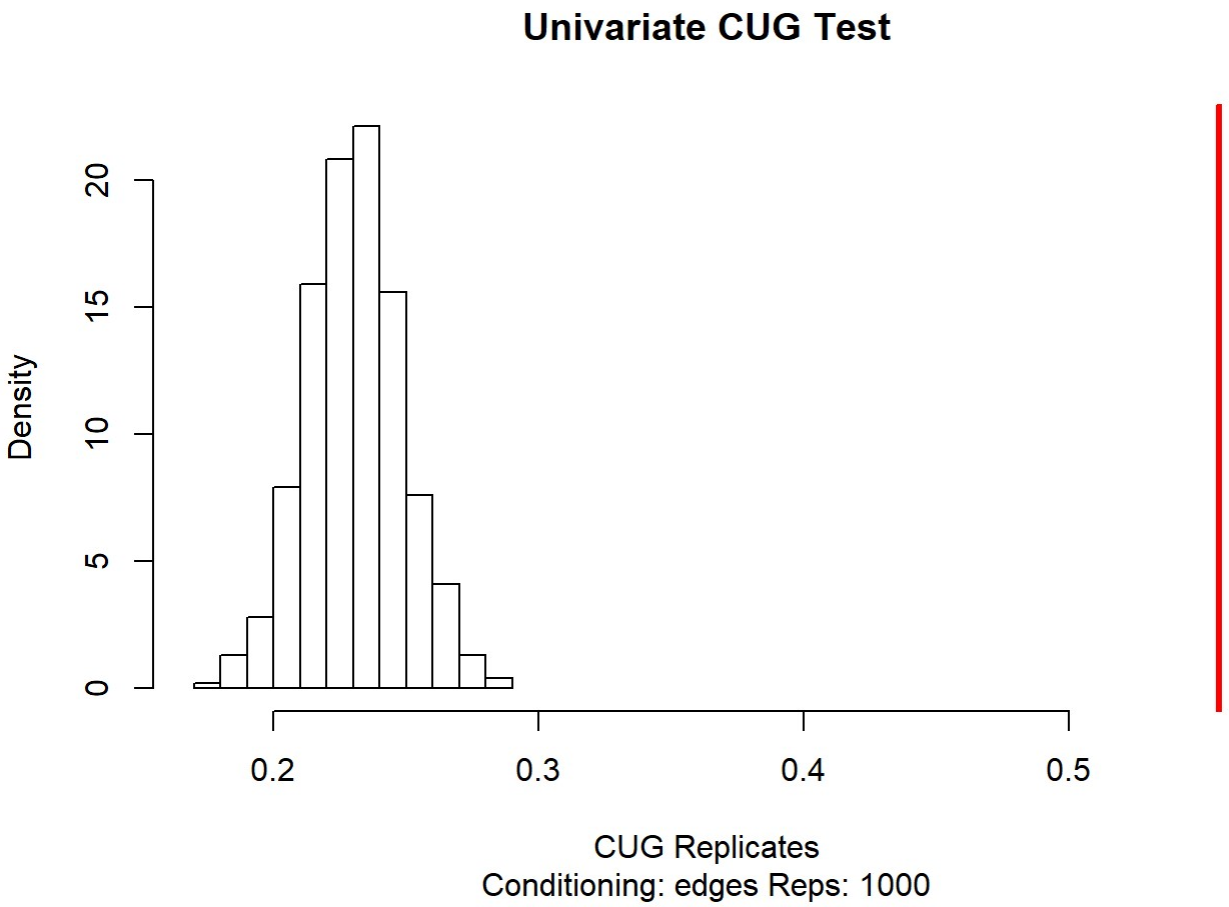
```
print.cug.test(temp_cug)
```

```
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: 0.5565912
## Pr(X>=Obs): 0
## Pr(X<=Obs): 1
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
if (save_to_file){
  tmp_cug = paste(trans_path, "Violent_Global_Transitivity_CUG.jpg", sep = '')
  jpeg(file = tmp_cug)
}
plot.cug.test(temp_cug)
```




```
if (save_to_file){
  dev.off()
}

if (save_to_file){
  sink()

  close(outFile)
  closeAllConnections()
}
```

Non-violent crime projection transitivity analysis

```
if (save_to_file){
  out_file_name = paste(trans_path, "NonViolent_Transitivity.txt", sep = '')
  outFile = file(out_file_name, open="wt")
  sink(file = outFile, append = TRUE)
}

v_local_trans = mean(transitivity(nonviolent_filter, type = "local"))
print(paste("Mean Local Transitivity:", v_local_trans))
```

```
## [1] "Mean Local Transitivity: NaN"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
temp_cug = mycugtest(nonviolent_filter, transitivity, cmode = "edges", type = "local"
)

print("Local Transitivity CUG Test Results:")
```

```
## [1] "Local Transitivity CUG Test Results:"
```

```
print.cug.test(temp_cug)
```

```
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: 1 1 1 1 0.9272727 0.9272727 0.1746032 1 1 1 1 1 0.9272727 0.890909
1 0.3550725 0.3675889 1 1 0.4285714 0.8095238 1 1 0.1907308 1 1 NaN 1 1 1 NaN 1 1 1 1
1 1 1 1
## Pr(X>=Obs): NA
## Pr(X<=Obs): NA
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
if (save_to_file){
  tmp_cug = paste(trans_path, "NonViolent_Local_Transitivity_CUG.jpg", sep = '')
  jpeg(file = tmp_cug)
  plot.cug.test(temp_cug)
  dev.off()
}

v_global_trans = transitivity(nonviolent_filter, type = "global")
print(paste("Global Transitivity:", v_global_trans))
```

```
## [1] "Global Transitivity: 0.389332213355733"
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
temp_cug = mycugtest(nonviolent_filter, transitivity, cmode = "edges", type = "global
")

print("Global Transitivity CUG Test Results:")
```

```
## [1] "Global Transitivity CUG Test Results:"
```

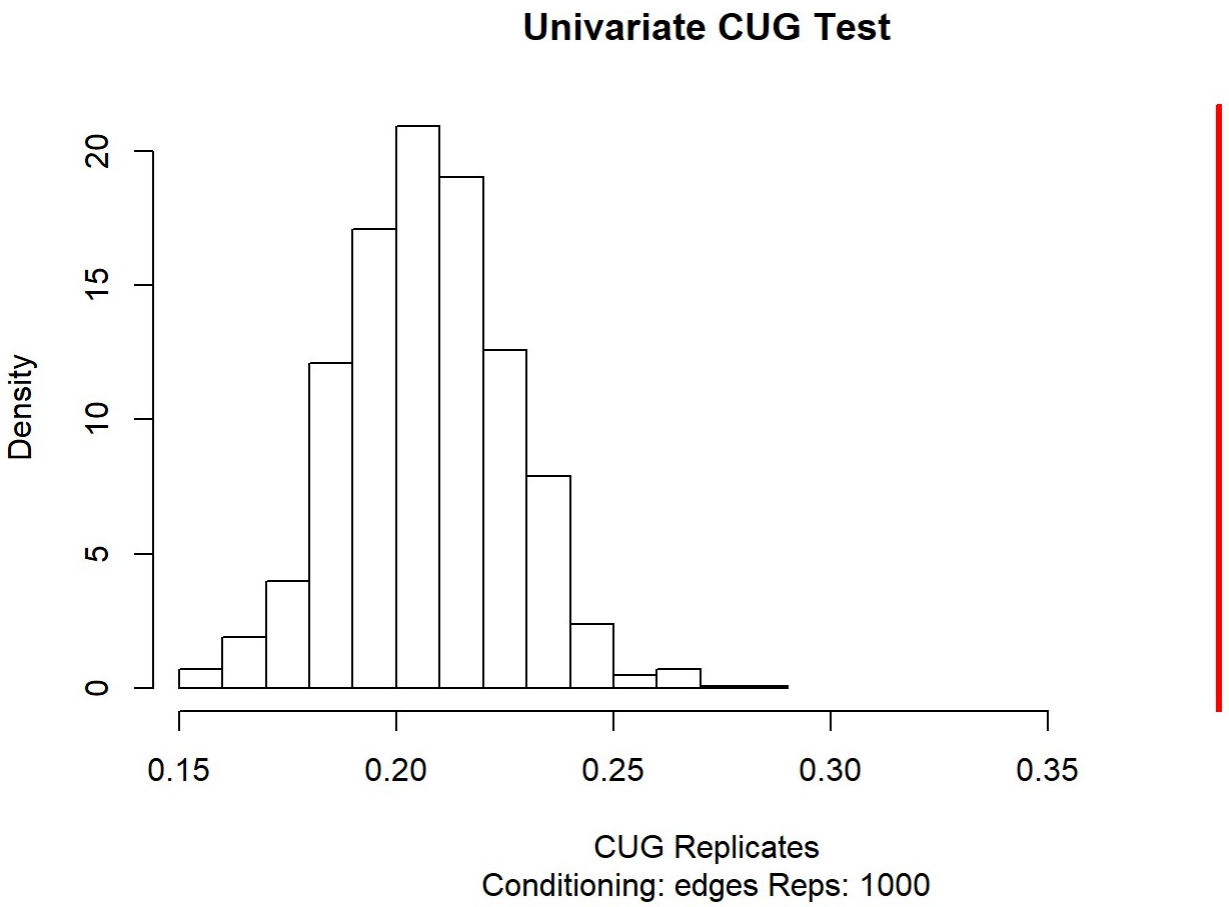
```
print.cug.test(temp_cug)
```

```
##
## Univariate Conditional Uniform Graph Test
##
## Conditioning Method: edges
## Graph Type:
## Diagonal Used: FALSE
## Replications: 1000
##
## Observed Value: 0.3893322
## Pr(X>=Obs): 0
## Pr(X<=Obs): 1
```

```
print("", quote=FALSE)
```

```
## [1]
```

```
if (save_to_file){
  tmp_cug = paste(trans_path, "NonViolent_Global_Transitivity_CUG.jpg", sep = '')
  jpeg(file = tmp_cug)
}
plot.cug.test(temp_cug)
```




```
if (save_to_file){
  dev.off()
}

if (save_to_file){
  sink()

  close(outFile)
  closeAllConnections()
}

###################### Community Detection ##########################################
###########

set.seed(20170423)
```

Violent crime projection modularity analysis

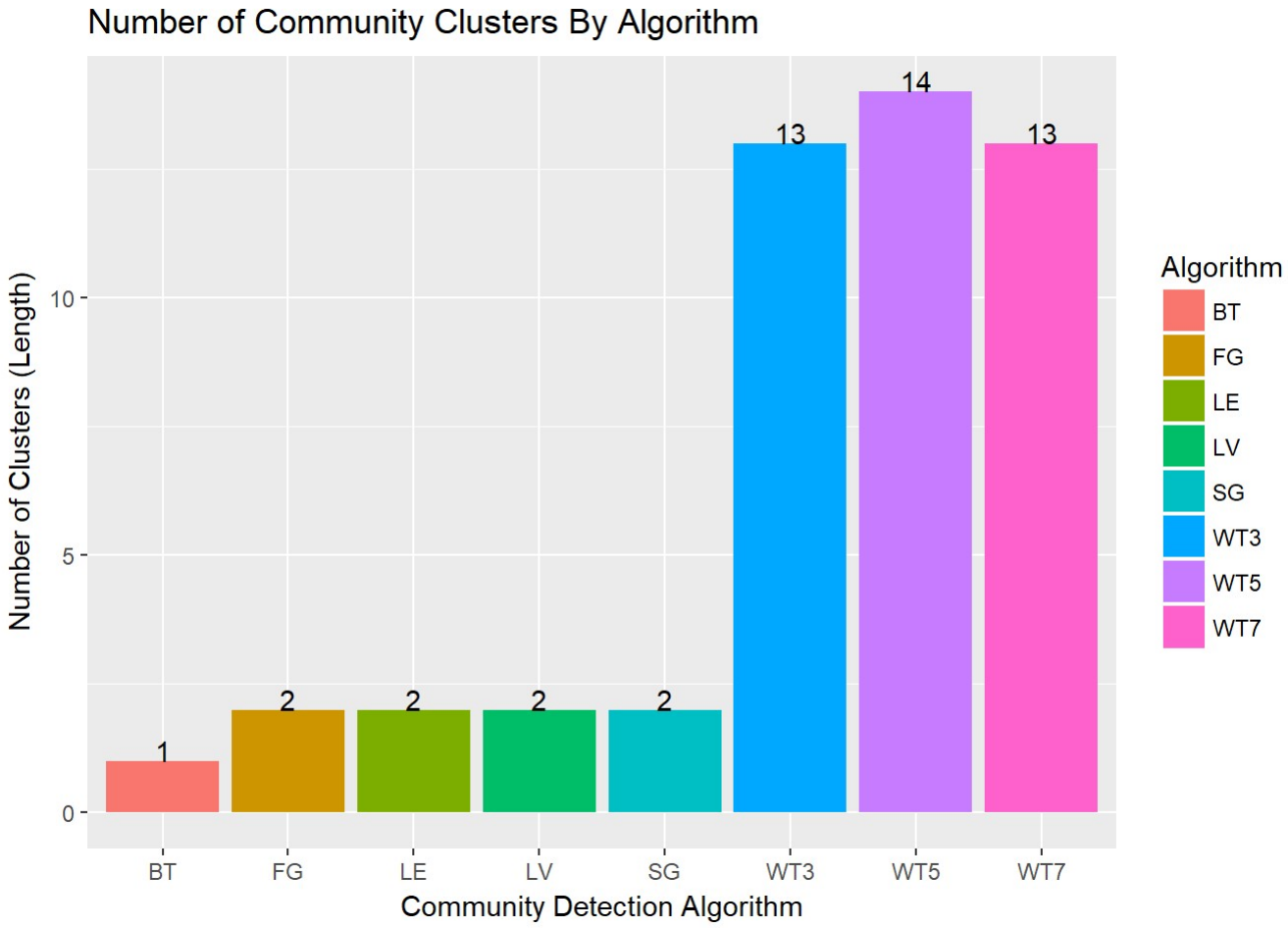
```
if (save_to_file){
  out_file_name = paste(output_path, "Graph_Modularities.txt", sep = '')
  outFile = file(out_file_name, open="wt")
  sink(file = outFile, append = TRUE)
}

# Violent community detection
comm.sg = cluster_spinglass(violent_filter)
V(violent_filter)$comm_sg = comm.sg$membership

comm.lv = cluster_louvain(violent_filter)
V(violent_filter)$comm_lv = comm.lv$membership

comm.le = cluster_leading_eigen(violent_filter)
V(violent_filter)$comm_le = comm.le$membership

comm.fg = cluster_fast_greedy(violent_filter)
V(violent_filter)$comm_fg = comm.fg$membership

comm.bt = cluster_edge_betweenness(violent_filter, weights = NULL)
V(violent_filter)$comm_bt = comm.bt$membership

comm.wt3 = cluster_walktrap(violent_filter, steps = 3)
V(violent_filter)$comm_wt5 = comm.wt3$membership

comm.wt5 = cluster_walktrap(violent_filter, steps = 5)
V(violent_filter)$comm_wt10 = comm.wt5$membership

comm.wt7 = cluster_walktrap(violent_filter, steps = 7)
V(violent_filter)$comm_wt10 = comm.wt7$membership

comm.mod = lapply(list(comm.sg, comm.lv, comm.le, comm.fg, comm.bt, comm.wt3, comm.wt
5, comm.wt7), modularity)
comm.len = lapply(list(comm.sg, comm.lv, comm.le, comm.fg, comm.bt, comm.wt3, comm.wt
5, comm.wt7), length)

print(comm.mod)
```

```
## [[1]]
## [1] 4.654424e-05
##
## [[2]]
## [1] 0.1074495
##
## [[3]]
## [1] 0.1074495
##
## [[4]]
## [1] 0.1074495
##
## [[5]]
## [1] 0
##
## [[6]]
## [1] 0.05773574
##
## [[7]]
## [1] 0.04976393
##
## [[8]]
## [1] 0.05375263
```

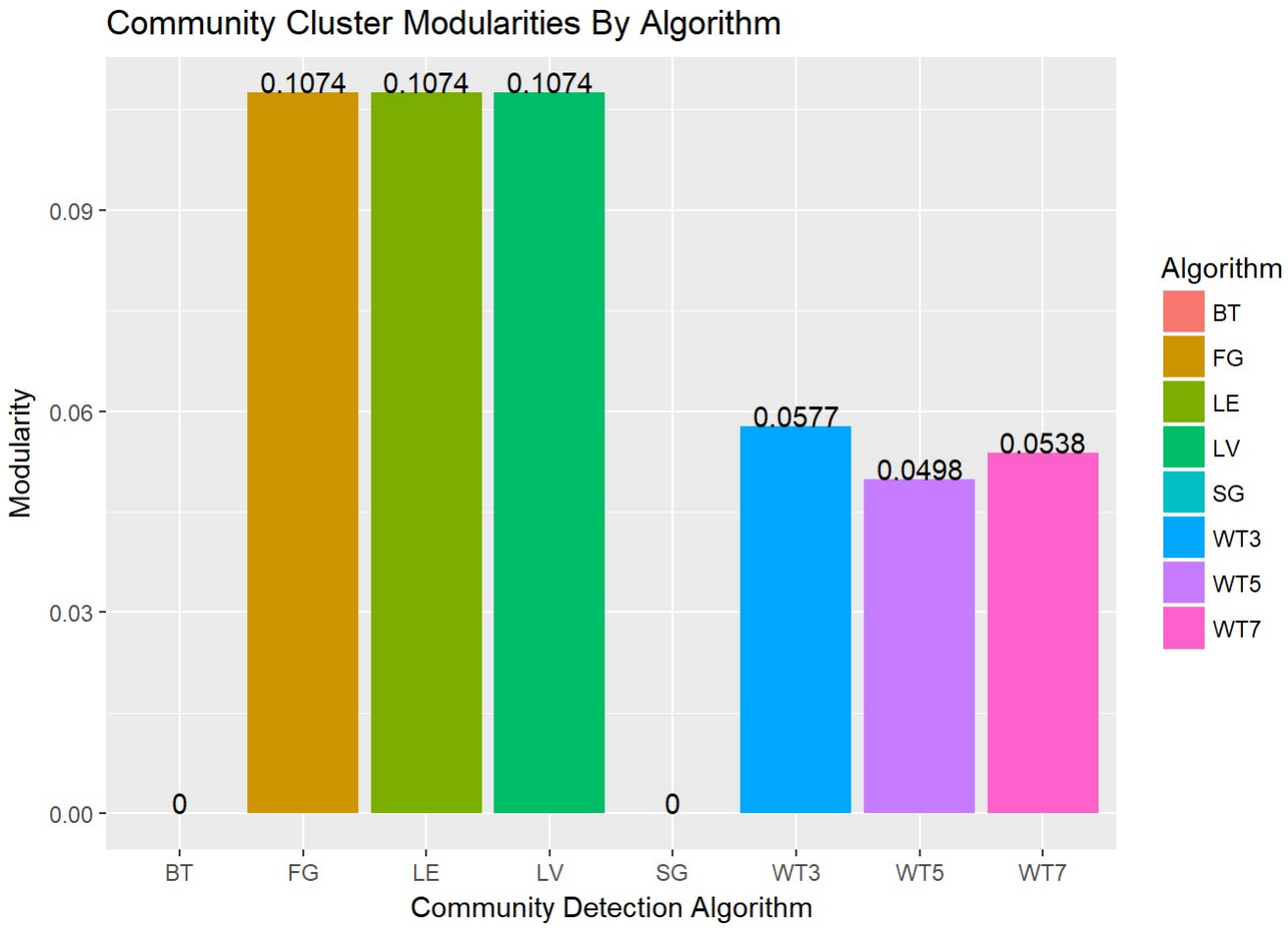
```
print(comm.len)
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 2
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 1
##
## [[6]]
## [1] 13
##
## [[7]]
## [1] 14
##
## [[8]]
## [1] 13
```

```
algorithms = c("SG", "LV", "LE", "FG", "BT", "WT3", "WT5", "WT7")
comm_df = data.frame(Modularity = as.vector(comm.mod, mode = "numeric"),
                     Length = as.vector(comm.len, mode = "numeric"), Algorithm = algo
rithms)

# Create bar chart of lengths
if (save_to_file){
  temp_jpg = paste(output_path, "Violent_Cluster_Sizes.jpg", sep = "")
  jpeg(file = temp_jpg)
}

p = ggplot(data = comm_df, aes(x = Algorithm, y = Length, fill = Algorithm))
p = p + geom_bar(stat = "identity")
p = p + geom_text(aes(label=round(Length, 4)), vjust=0)
p = p + ggtitle("Number of Community Clusters By Algorithm")
p = p + labs(x = "Community Detection Algorithm", y = "Number of Clusters (Length)")
print(p)
```



```
if (save_to_file){
  dev.off()
}

# Create bar chart of lengths
if (save_to_file){
  temp_jpg = paste(output_path, "Violent_Cluster_Modularities.jpg", sep = "")
  jpeg(file = temp_jpg)
}

p = ggplot(data = comm_df, aes(x = Algorithm, y = Modularity, fill = Algorithm))
p = p + geom_bar(stat = "identity")
p = p + geom_text(aes(label=round(Modularity, 4)), vjust=0)
p = p + ggtitle("Community Cluster Modularities By Algorithm")
p = p + labs(x = "Community Detection Algorithm", y = "Modularity")
print(p)
```




```
if (save_to_file){
  dev.off()
}
```

Non-violent crime projection modularity analysis

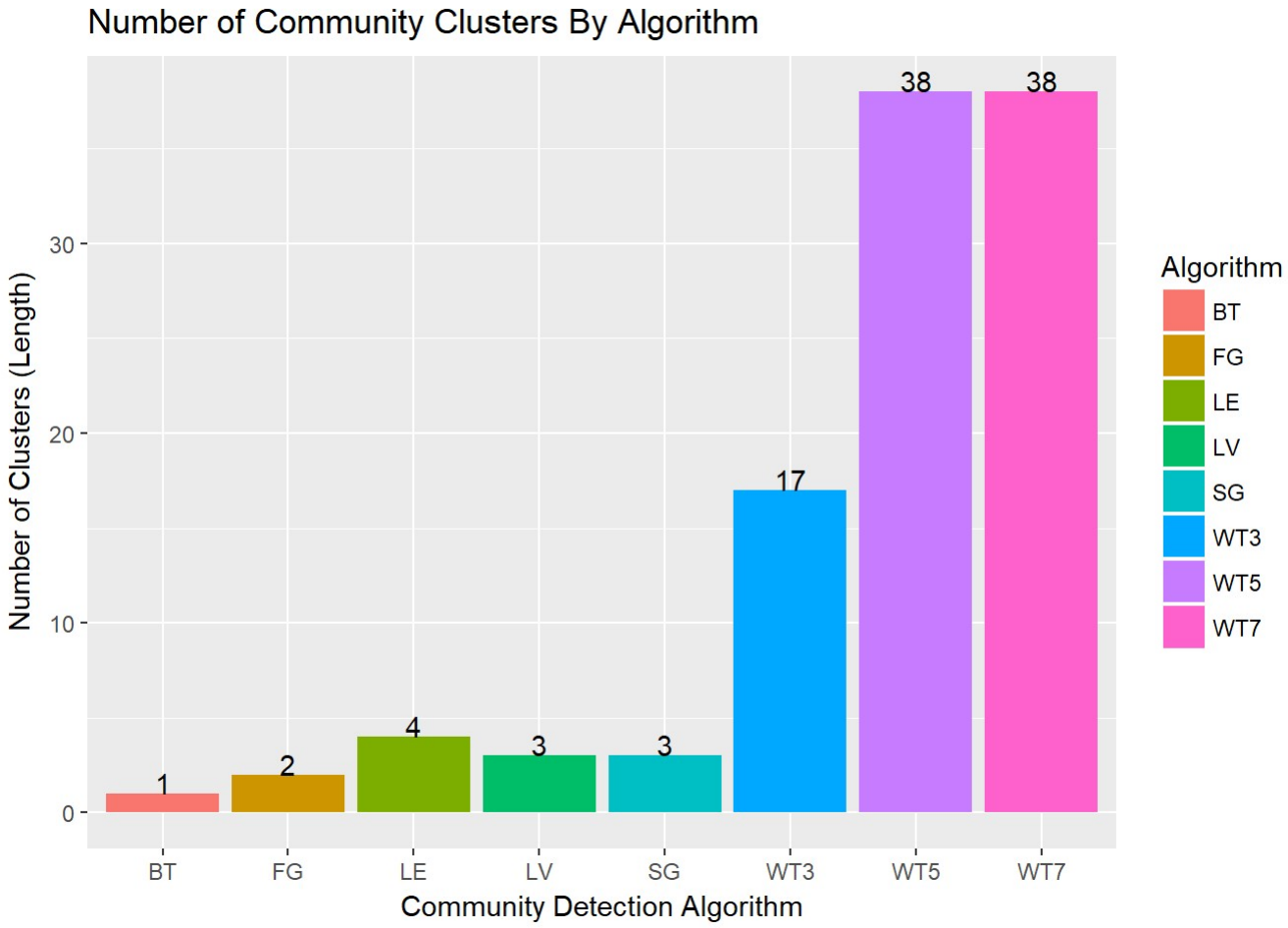
```
# NonViolent community detection
commNV.sg = cluster_spinglass(nonviolent_filter)
V(nonviolent_filter)$comm_sg = commNV.sg$membership

commNV.lv = cluster_louvain(nonviolent_filter)
V(nonviolent_filter)$comm_lv = commNV.lv$membership

commNV.le = cluster_leading_eigen(nonviolent_filter)
V(nonviolent_filter)$comm_le = commNV.le$membership

commNV.fg = cluster_fast_greedy(nonviolent_filter)
V(nonviolent_filter)$comm_fg = commNV.fg$membership

commNV.bt = cluster_edge_betweenness(nonviolent_filter, weights = NULL)
V(nonviolent_filter)$comm_bt = commNV.bt$membership

commNV.wt3 = cluster_walktrap(nonviolent_filter, steps = 3)
V(nonviolent_filter)$comm_wt5 = commNV.wt3$membership

commNV.wt5 = cluster_walktrap(nonviolent_filter, steps = 5)
V(nonviolent_filter)$comm_wt10 = commNV.wt5$membership

commNV.wt7 = cluster_walktrap(nonviolent_filter, steps = 7)
V(nonviolent_filter)$comm_wt10 = commNV.wt7$membership

commNV.mod = lapply(list(commNV.sg, commNV.lv, commNV.le, commNV.fg, commNV.bt, commN
V.wt3, commNV.wt5,
                         commNV.wt7), modularity)
commNV.len = lapply(list(commNV.sg, commNV.lv, commNV.le, commNV.fg, commNV.bt, commN
V.wt3, commNV.wt5,
                    commNV.wt7), length)

print(commNV.mod)
```

```
## [[1]]
## [1] 1.724639e-05
##
## [[2]]
## [1] 0.07712608
##
## [[3]]
## [1] 0.06716464
##
## [[4]]
## [1] 0.06936387
##
## [[5]]
## [1] 0
##
## [[6]]
## [1] 0.004362267
##
## [[7]]
## [1] 0
##
## [[8]]
## [1] 0
```

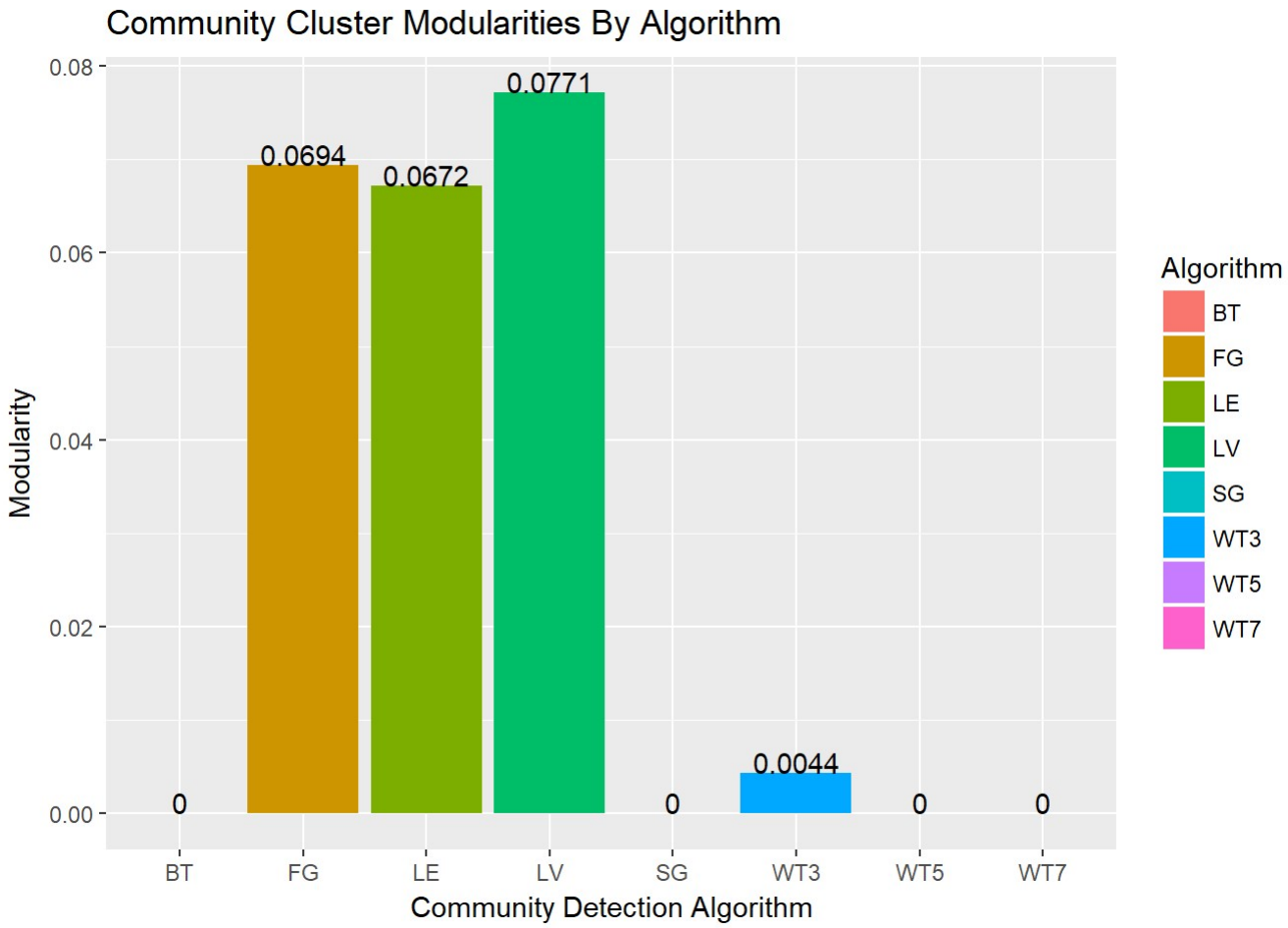
```
print(commNV.len)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 4
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 1
##
## [[6]]
## [1] 17
##
## [[7]]
## [1] 38
##
## [[8]]
## [1] 38
```

```
algorithms = c("SG", "LV", "LE", "FG", "BT", "WT3", "WT5", "WT7")
comm_df = data.frame(Modularity = as.vector(commNV.mod, mode = "numeric"),
                     Length = as.vector(commNV.len, mode = "numeric"), Algorithm = al
gorithms)

# Create bar chart of lengths
if (save_to_file){
  temp_jpg = paste(output_path, "NonViolent_Cluster_Sizes.jpg", sep = "")
  jpeg(file = temp_jpg)
}

p = ggplot(data = comm_df, aes(x = Algorithm, y = Length, fill = Algorithm))
p = p + geom_bar(stat = "identity")
p = p + geom_text(aes(label=round(Length, 4)), vjust=0)
p = p + ggtitle("Number of Community Clusters By Algorithm")
p = p + labs(x = "Community Detection Algorithm", y = "Number of Clusters (Length)")
print(p)
```



```
if (save_to_file){
  dev.off()
}

# Create bar chart of lengths
if (save_to_file){
  temp_jpg = paste(output_path, "NonViolent_Cluster_Modularities.jpg", sep = "")
  jpeg(file = temp_jpg)
}

p = ggplot(data = comm_df, aes(x = Algorithm, y = Modularity, fill = Algorithm))
p = p + geom_bar(stat = "identity")
p = p + geom_text(aes(label=round(Modularity, 4)), vjust=0)
p = p + ggtitle("Community Cluster Modularities By Algorithm")
p = p + labs(x = "Community Detection Algorithm", y = "Modularity")
print(p)
```



```
if (save_to_file){
  dev.off()
}

if (save_to_file){
  sink()

  close(outFile)
  closeAllConnections()
}
```

Save final networks in graphml format

```
# Save updated files
out_file_violent = paste(path, "feb_violent_neighborhoods_95_filtered.graphml", sep =
"")
write_graph(violent_filter, out_file_violent, format = "graphml")

out_file_nonviolent = paste(path, "feb_nonviolent_neighborhoods_95_filtered.graphml",
sep = "")
write_graph(nonviolent_filter, out_file_nonviolent, format = "graphml")
```