# Social Network Analysis Home Assignment 1

*{Shupeyko Nikita}*

*due date - 08.05.2016 23:59*

## Contents

## Power law. Descriptive network analysis

```r
library(igraph)
```

### Problem 1

Recall from the lecture that probability density function (PDF) for power law distributed variable is:

$$p(x) = Cx^{-\alpha}$$

Take logarithm of both sides:

$$\log p(x) = \log C - \alpha \log x$$

Now you can use standard R functions like `lm()` to calculate $\alpha$ coefficient via linear regression. However you might find that it is a bad idea.

Alternatively, you can compute cumulative density function (CDF)

$$f(x) = Pr(x < X)$$

of power law distribution. Good things about CDF of the power law are:

- It still has form of power law
- On log-log plot it looks more like a line

**1. Derive the formula for CDF function of power law $F(x)$. It means you should calculate cumulative distribution function by integration of PDF function.**

Since $p(x) = Cx^{-\alpha}$, assume without loss of generality that $0 < x_{min} \le x \le x_{max} < +\infty$. Then,

$$F(x_{min} \le x \le x_{max}) = \int_{x_{min}}^{x_{max}} p(x)dx = C\int_{x_{min}}^{x_{max}} x^{-\alpha}dx = \frac{C}{1-\alpha}x^{1-\alpha}\Big|_{x_{min}}^{x_{min}} = \frac{C}{1-\alpha}\left(\frac{1}{x_{max}^{\alpha-1}} - \frac{1}{x_{min}^{\alpha-1}}\right).$$

Given the normalization condition $F(x_{min} \le x < +\infty) = 1 = \int_{x_{min}}^{+\infty} p(x)dx$, it is clear that

$$1 = \frac{C}{1-\alpha}\left(\frac{1}{x_{max}^{\alpha-1}} - \frac{1}{x_{min}^{\alpha-1}}\right)\Bigg|^{x_{max}\to+\infty} = \begin{cases} \frac{C}{\alpha-1}\frac{1}{x_{min}^{\alpha-1}} & \text{if } \alpha > 1, \\ undefined & otherwise. \end{cases}$$

$$C = (\alpha-1)x_{min}^{\alpha-1}.$$

Hence,

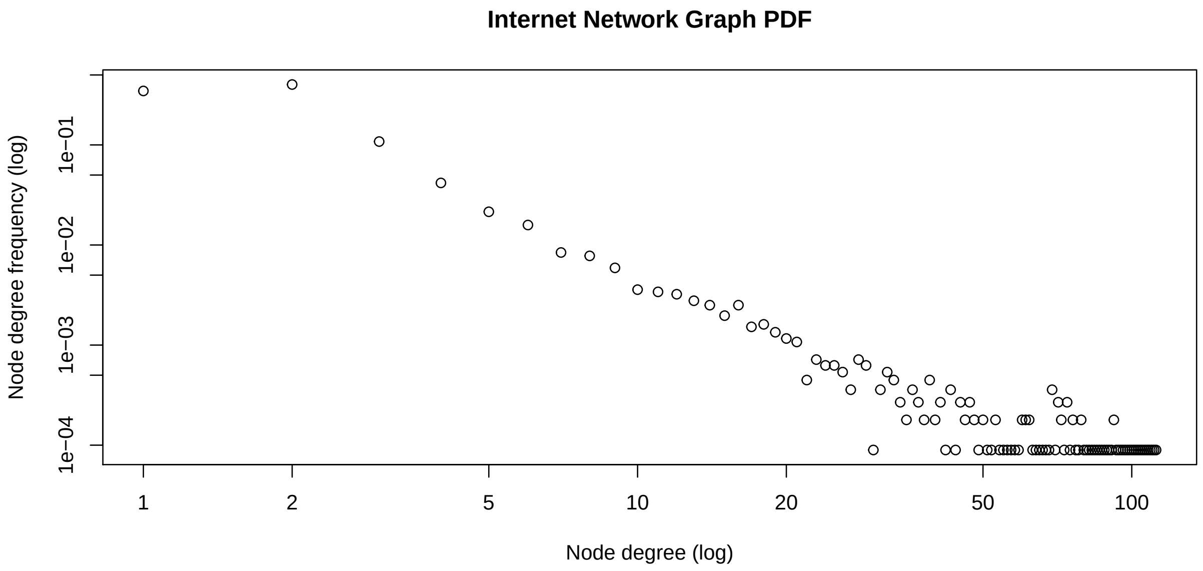
$$F(x_{min} \le x) = \frac{(\alpha-1)x_{min}^{\alpha-1}}{1-\alpha}\left(\frac{1}{x^{\alpha-1}} - \frac{1}{x_{min}^{\alpha-1}}\right) = \frac{\alpha-1}{\alpha-1}x_{min}^{\alpha-1}\left(\frac{1}{x_{min}^{\alpha-1}} - \frac{1}{x^{\alpha-1}}\right) = 1 - x_{min}^{\alpha-1}x^{1-\alpha} \text{ for } \alpha > 1.$$

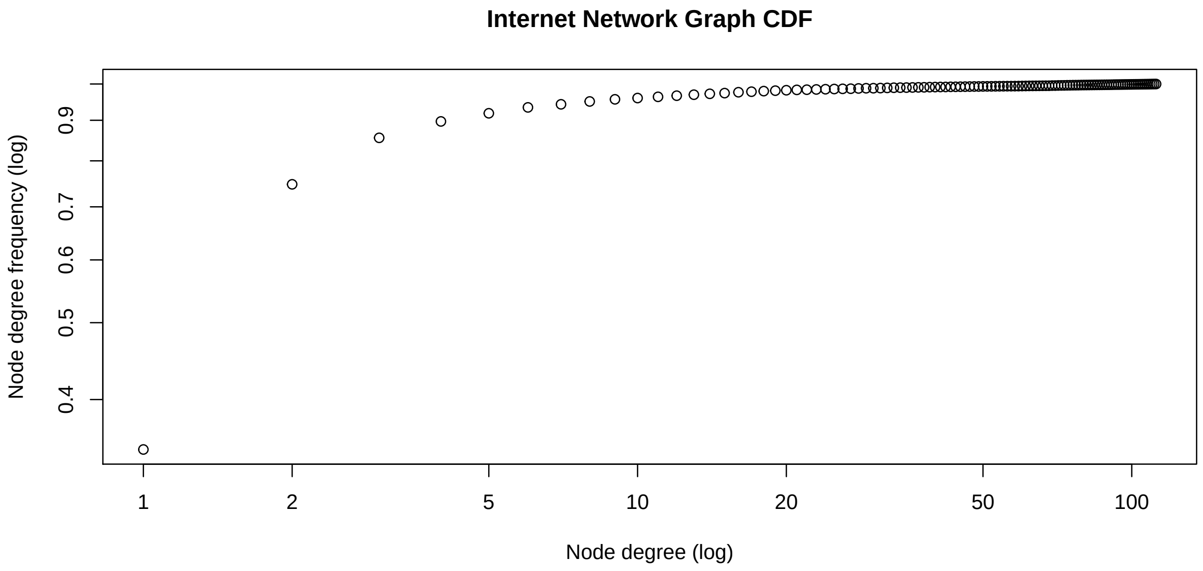
Also note that

$$p(x) = (\alpha-1)x_{min}^{\alpha-1}x^{-\alpha} \text{ for } \alpha > 1.$$

### 2. Download Internet Network and plot PDF and CDF of the degree distribution in log-log scale

```
dat = t(as.matrix(read.table("Internet_AS.dat")))
g = graph(dat)

g.deg = degree(g)

g.deg.rng = as.numeric(1:max(degree(g)))
g.deg.dist = degree.distribution(g, cumulative = F)[g.deg.rng]

g.deg.rng = g.deg.rng[which(g.deg.dist > 0)]
g.deg.dist = g.deg.dist[which(g.deg.dist > 0)]

g.cum.deg.rng = g.deg.rng + 1
g.cum.deg.dist = cumsum(g.deg.dist)

par(mfrow = c(2, 1))

plot(g.deg.dist, log="xy",
     main="Internet Network Graph PDF", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")

plot(g.cum.deg.dist, log="xy",
     main="Internet Network Graph CDF", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")
```

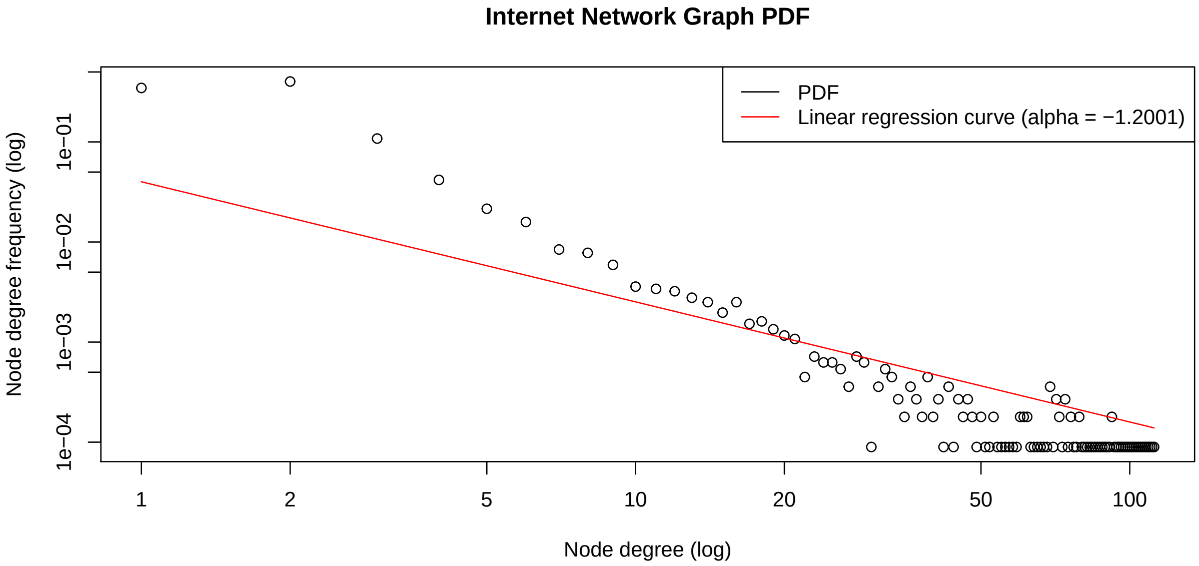


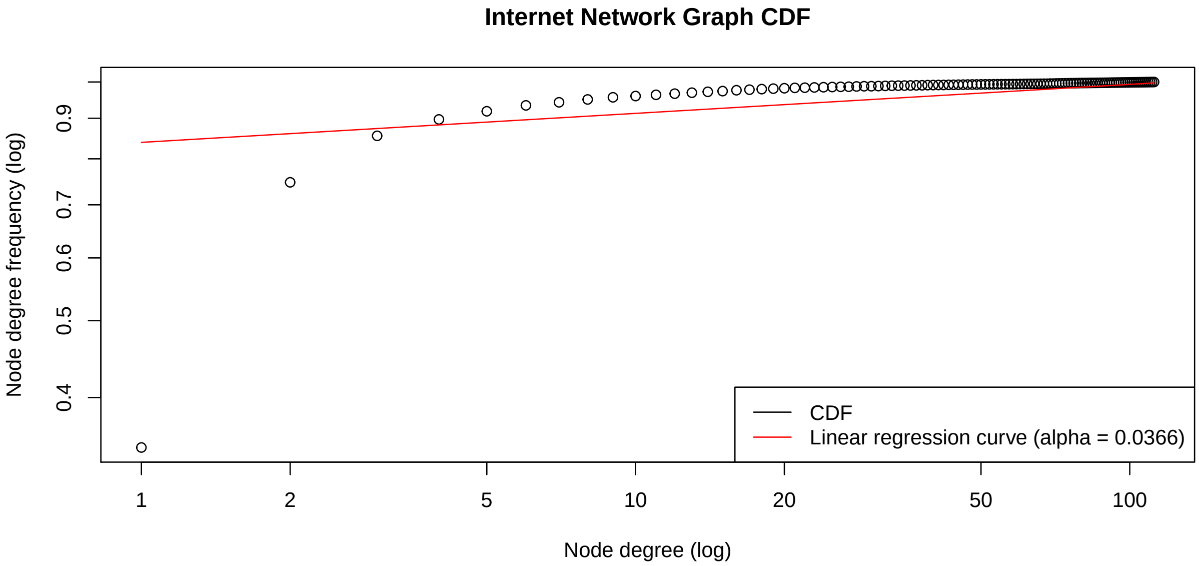


### 3. Fit linear regression model to PDF and CDF to estimate $\alpha$. Plot fitted models along with data

```
par(mfrow = c(2, 1))


plot(g.deg.dist, log="xy",
     main="Internet Network Graph PDF", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")

linreg = lm(log(g.deg.dist) ~ log(g.deg.rng))
linreg.coeffs = coef(linreg)
func_powerlaw <- function(x) exp(linreg.coeffs[[1]] + linreg.coeffs[[2]] * log(x))
```

```r
curve(func_powerlaw, col = "red", add = TRUE, n = length(g.deg.rng))

legend("topright",
       c("PDF", sprintf("Linear regression curve (alpha = %.4f)", linreg.coeffs[[2]])),
       lty = 1, col= c('black', 'red'))


plot(g.cum.deg.dist, log="xy",
     main="Internet Network Graph CDF", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")

linreg.cum = lm(log(g.cum.deg.dist) ~ log(g.cum.deg.rng))
linreg.cum.coeffs = coef(linreg.cum)
func_powerlaw.cum <- function(x) exp(linreg.cum.coeffs[[1]] + linreg.cum.coeffs[[2]] * log(x))
curve(func_powerlaw.cum, col = "red", add = TRUE, n = length(g.cum.deg.rng))

legend("bottomright",
       c("CDF", sprintf("Linear regression curve (alpha = %.4f)", linreg.cum.coeffs[[2]])),
       lty = 1, col= c('black', 'red'))
```





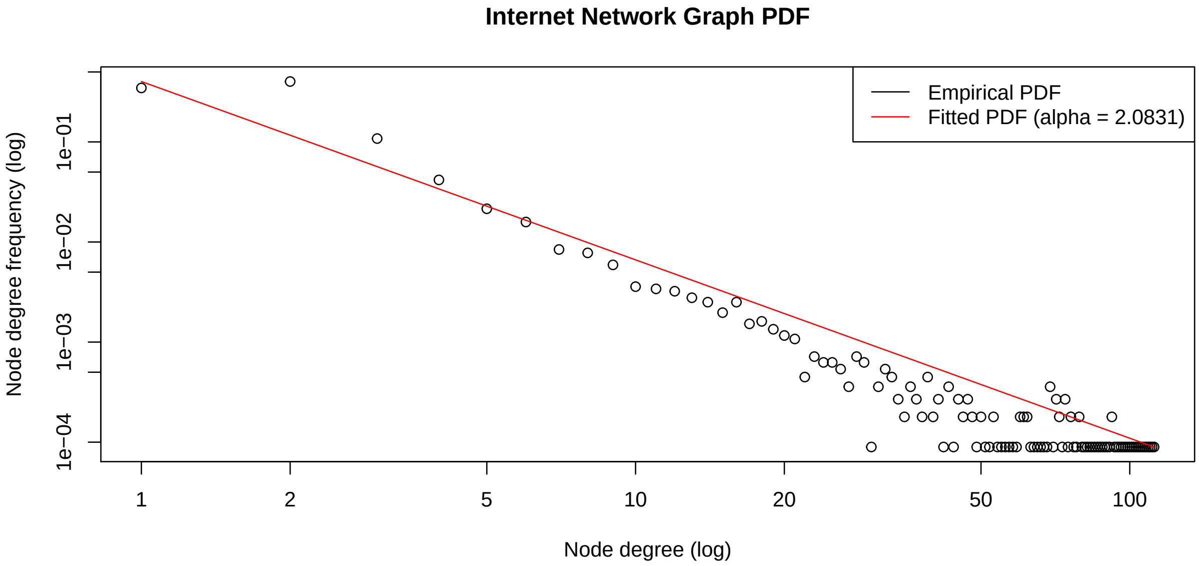
## Problem 2

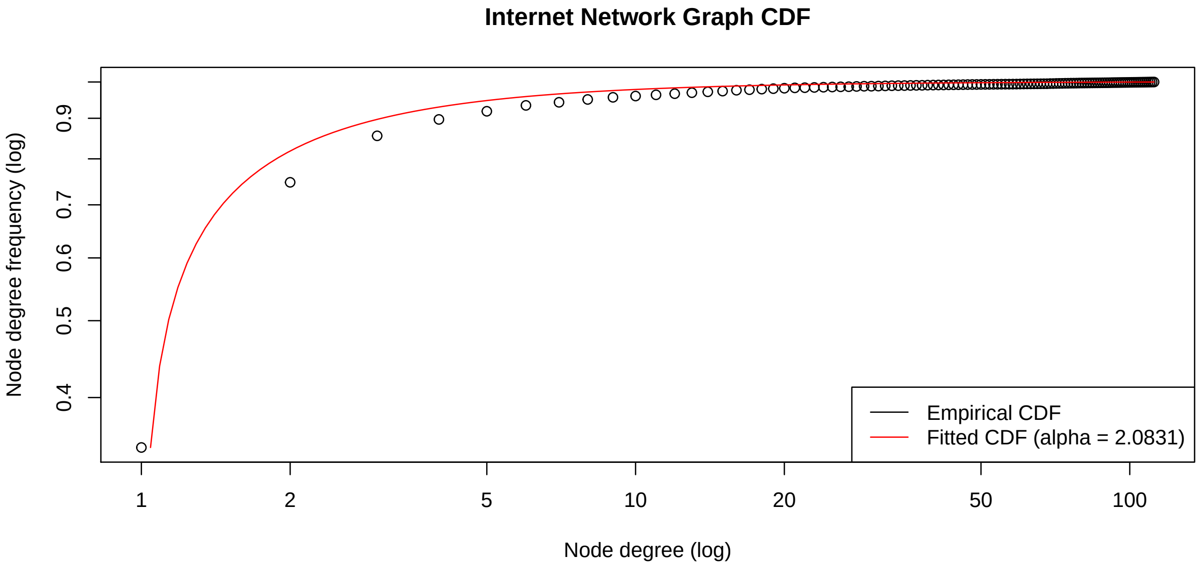
### 1. Using `power.law.fit` find `xmin` value and corresponding `alpha`

```
powerlaw.fit = power.law.fit(g.deg, xmin = NULL, implementation = "plfit")
alpha = powerlaw.fit$alpha
xmin = powerlaw.fit$xmin
```

### 2. Put fitted model along with empirical PDF (CDF)

```r
par(mfrow = c(2,1))


PDF.fit = function(x) return ((alpha-1) * xmin^(alpha-1) * x^(-alpha))

plot(g.deg.dist, log="xy",
     main="Internet Network Graph PDF", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")
par(new = TRUE)
curve(PDF.fit, from = xmin, to = max(g.deg), n = length(g.deg.rng), log="xy",
      col = "red", add = FALSE, main = "", xlab = "", ylab = "", axes = FALSE)
legend("topright", c("Empirical PDF", sprintf("Fitted PDF (alpha = %.4f)", alpha)),
       lty = 1, col= c('black', 'red'))


CDF.fit = function(x) return (1 - (xmin^(alpha-1)) * x^(1-alpha))

plot(g.cum.deg.dist, log="xy",
     main="Internet Network Graph CDF", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")
par(new = TRUE)
curve(CDF.fit, from = xmin, to = max(g.cum.deg.rng), n = length(g.cum.deg.rng), log="xy",
      col = "red", add = FALSE,  main = "", xlab = "", ylab = "", axes = FALSE)
legend("bottomright", c("Empirical CDF", sprintf("Fitted CDF (alpha = %.4f)", alpha)),
       lty = 1, col= c('black', 'red'))
```





## Problem 3.

For Wikipedia vote network (clear up comments in the begging of the file) derive the following characteristics:
1. The number of vertices and edges
2. The number of loops (edges that start and end at the same vertex)
3. The number of symmetrical edges
4. Degree distribution (without considering the direction)
5. The number of nodes with a degree greater than 1 and with a degree greater than 15
6. Find strongly connected components and their sizes.
7. Take subgraph of the original graph, which consists of the first 80 vertices and set color into red for those nodes in which the number of incoming edges is greater than the number of outgoing edges.Otherwise, set color in blue. For nodes with the same number of incoming and outgoing edges set color into green. Besides that, increase the size of vertices with a maximum value of transitivity (for example, you may set size into 10

for these nodes and 1 for others).
8.Take subgraph from the previous task and find maximal connected component. For this component highlight any way that corresponds to the diameter of the subgraph. How many such paths are in this graph?
9. Make average neighbor degree vs node degree scatter plot (one point on the plot per node) and aggregated plot, averaging over all nodes with the same degree (aggregated average vs degree, one value per degree). Explain your observations.
10. Make local clustering coefficient vs node degree scatter plot (one point on the plot per node) and aggregated, averaging over all nodes with the same degree (aggregated average vs degree, one value per degree). Explain your observations.

### 1. The number of vertices and edges.

```
wv = t(as.matrix(read.table("Wiki-vote.txt")))
wv_g = graph(wv)

paste0("Vertex count: ", vcount(wv_g))
```

```
## [1] "Vertex count: 8297"
```

```
paste0("Edge count: ", ecount(wv_g))
```

```
## [1] "Edge count: 103689"
```

### 2. The number of loops (edges that start and end at the same vertex)

```
paste0("Loop connection count: ", sum(is.loop(wv_g), na.rm = T))
```

```
## [1] "Loop connection count: 0"
```

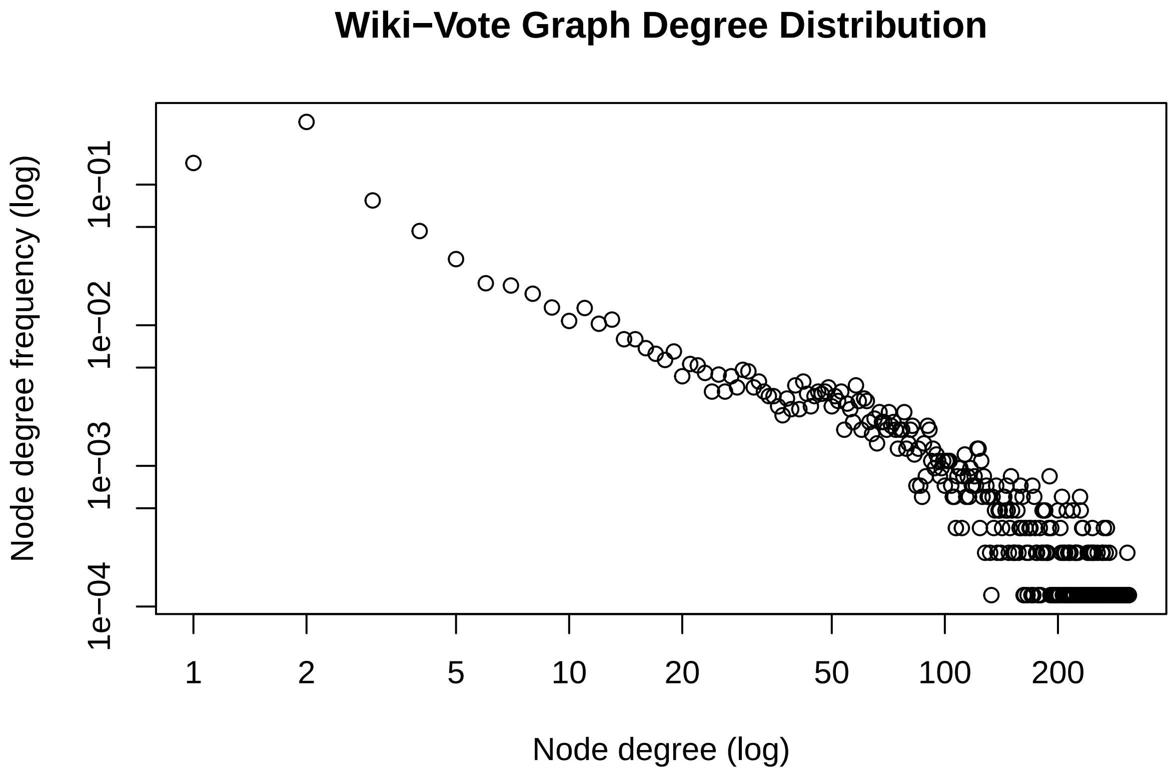
### 3. The number of symmetrical edges

```
paste0("Reciprocal connection count:", reciprocity(wv_g) * ecount(wv_g) / 2)
```

```
## [1] "Reciprocal connection count:2927"
```

### 4. Degree distribution

```
wv_g.deg.dist = degree.distribution(wv_g)
wv_g.deg.dist.no.gaps = degree.distribution(wv_g)[which(wv_g.deg.dist > 0)]
plot(wv_g.deg.dist.no.gaps, log = "xy",
     main = "Wiki-Vote Graph Degree Distribution", xlab="Node degree (log)",
     ylab="Node degree frequency (log)")
```



**5. The number of nodes with a degree greater than 1 and with a degree greater than 15**

```
paste0("Nodes degree > 1 count: ", sum(degree(wv_g) > 1, na.rm = T))
```

```
## [1] "Nodes degree > 1 count: 4800"
```

```
paste0("Nodes degree > 15 count: ", sum(degree(wv_g) > 15, na.rm = T))
```

```
## [1] "Nodes degree > 15 count: 2389"
```

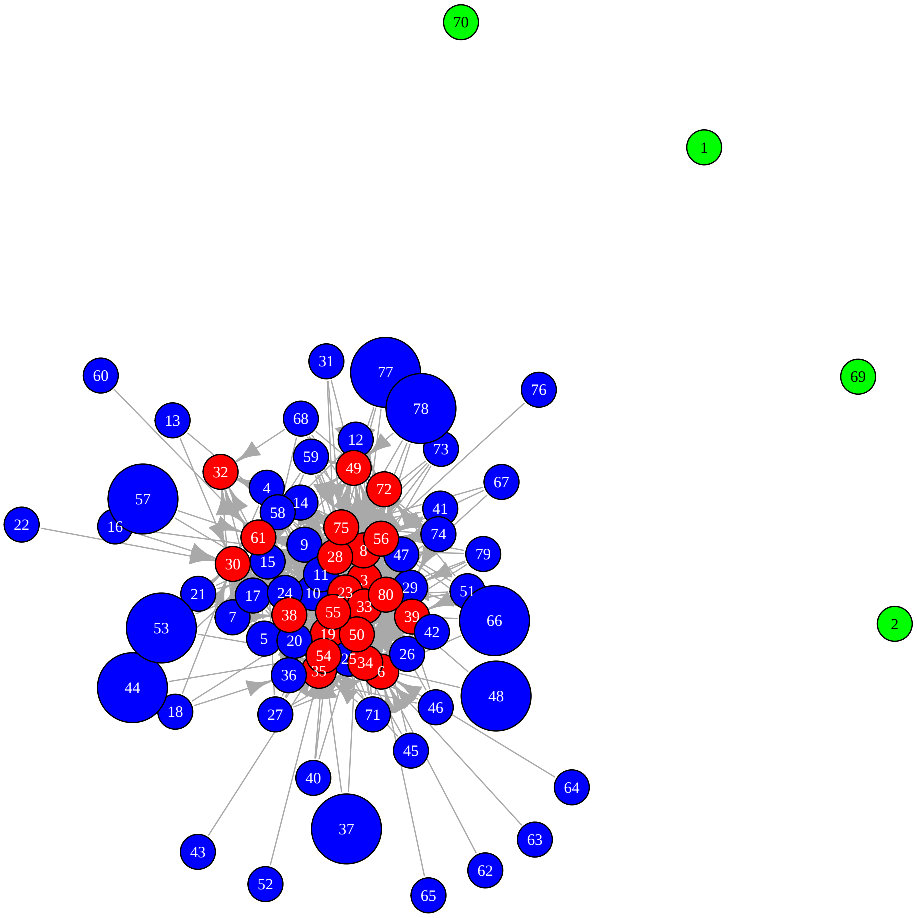
**6. Find strongly connected components and thier sizes.**

```
paste0("SCCs count: ", no.clusters(wv_g, mode = "strong"))
```

```
## [1] "SCCs count: 6998"
```

**7. Take subgraph of the original graph, which consists of the first 80 vertices and set color into red for those nodes in which the number of incoming edges is greater than the number of outgoing edges.Otherwise, set color in blue. For nodes with the same number of incoming and outgoing edges set color into green. Besides that, increase the size of vertices with a maximum value of transitivity (for example, you may set size into 10 for these nodes and 1 for others).**

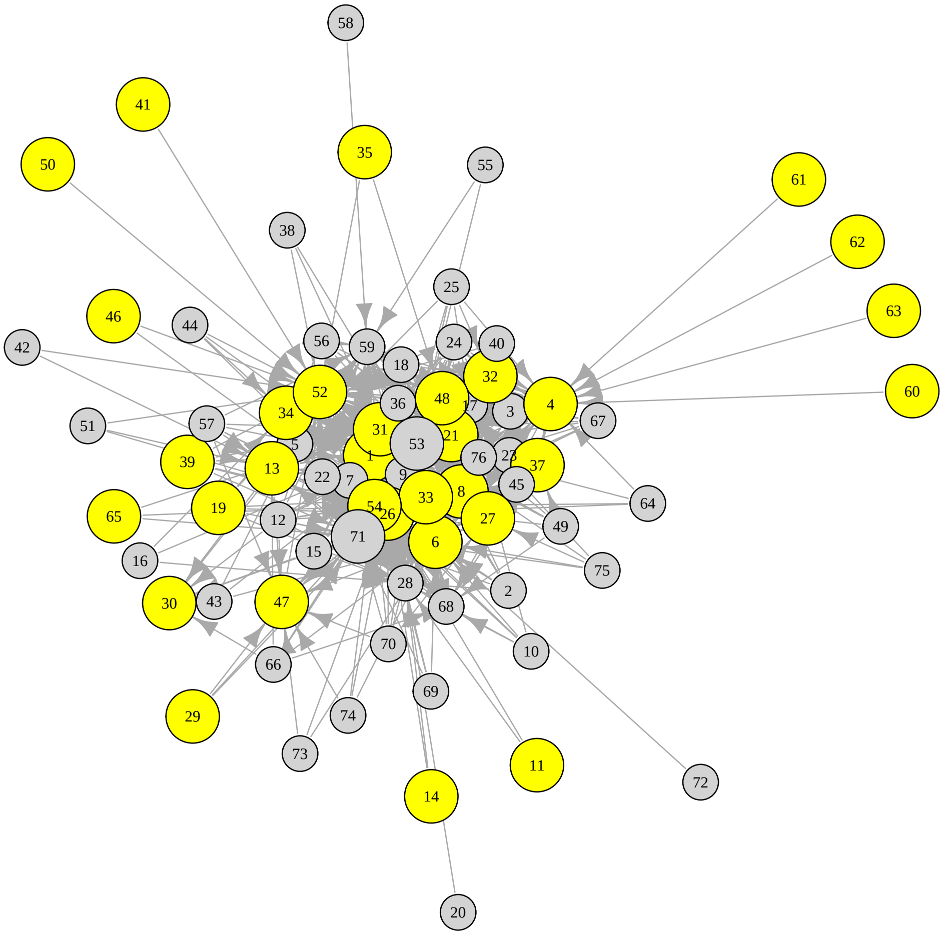
```
wv_g_subg = induced.subgraph(wv_g, V(wv_g)[1:80])

V(wv_g_subg)$label.cex = .75

for (v in V(wv_g_subg)) {
  ins.total = length(incident(wv_g_subg, v, mode = "in"))
  outs.total = length(incident(wv_g_subg, v, mode = "out"))
  if (ins.total > outs.total) {
    V(wv_g_subg)[v]$color = "red"
    V(wv_g_subg)[v]$label.color = "white"
  } else if (ins.total == outs.total) {
    V(wv_g_subg)[v]$color = "green"
  } else {
    V(wv_g_subg)[v]$color = "blue"
    V(wv_g_subg)[v]$label.color = "white"
  }
}

wv_g_subg.trans = transitivity(wv_g_subg, type = "local")
wv_g_subg.trans.max = max(wv_g_subg.trans, na.rm = T)
wv_g_subg.vert.size = rep(8, times = vcount(wv_g_subg))
for (v in V(wv_g_subg)[which(!is.na(wv_g_subg.trans))]) {
  if (wv_g_subg.trans[v] == wv_g_subg.trans.max) {
    wv_g_subg.vert.size[v] = 16
  }
}
plot(wv_g_subg,
     vertex.size = setNames(wv_g_subg.vert.size, V(wv_g_subg)))
```



**8.Take subgraph from the previous task and find maximal connected component. For this component highlight any way that corresponds to the diameter of the subgraph. How many such paths are in this graph?**

```
wv_g_subg.diam = diameter(wv_g_subg)

# revealing maximal connected (giant) component
func_giant.component = function(graph) {
  clusters = clusters(graph)
  induced.subgraph(graph, which(clusters$membership == which.max(clusters$csize)))
}
wv_g_subg.giant.subg = func_giant.component(wv_g_subg)
```

```r
# counting a number of giant component's shortest paths
# representing the containing graph's diameter
wv_g_subg.giant.subg.diam.count =
    path.length.hist(wv_g_subg.giant.subg)$res[wv_g_subg.diam]

# highlighting vertices (in yellow) comprised in those shortest paths
# on a giant component's plot
# (p.s. the performance is awful O(n^2))
wv_g_subg.giant.subg.vert.size = rep(8, times = vcount(wv_g_subg.giant.subg))
for (v1 in V(wv_g_subg.giant.subg)) {
  V(wv_g_subg.giant.subg)[v1]$color = "lightgray"
  V(wv_g_subg.giant.subg)[v1]$label.color = "black"
  for (v2 in V(wv_g_subg.giant.subg)) {
    v1.geo.v2 =
        get.shortest.paths(wv_g_subg.giant.subg, from = v1, to = v2)$vpath[[1]]
    if (length(v1.geo.v2) == wv_g_subg.diam) {
      for (v.geo in v1.geo.v2) {
        V(wv_g_subg.giant.subg)[v.geo]$color = "yellow"
        wv_g_subg.giant.subg.vert.size[v.geo] = 12
      }
    }
  }

}
plot(wv_g_subg.giant.subg,
     vertex.size = setNames(wv_g_subg.giant.subg.vert.size, V(wv_g_subg.giant.subg)))
```

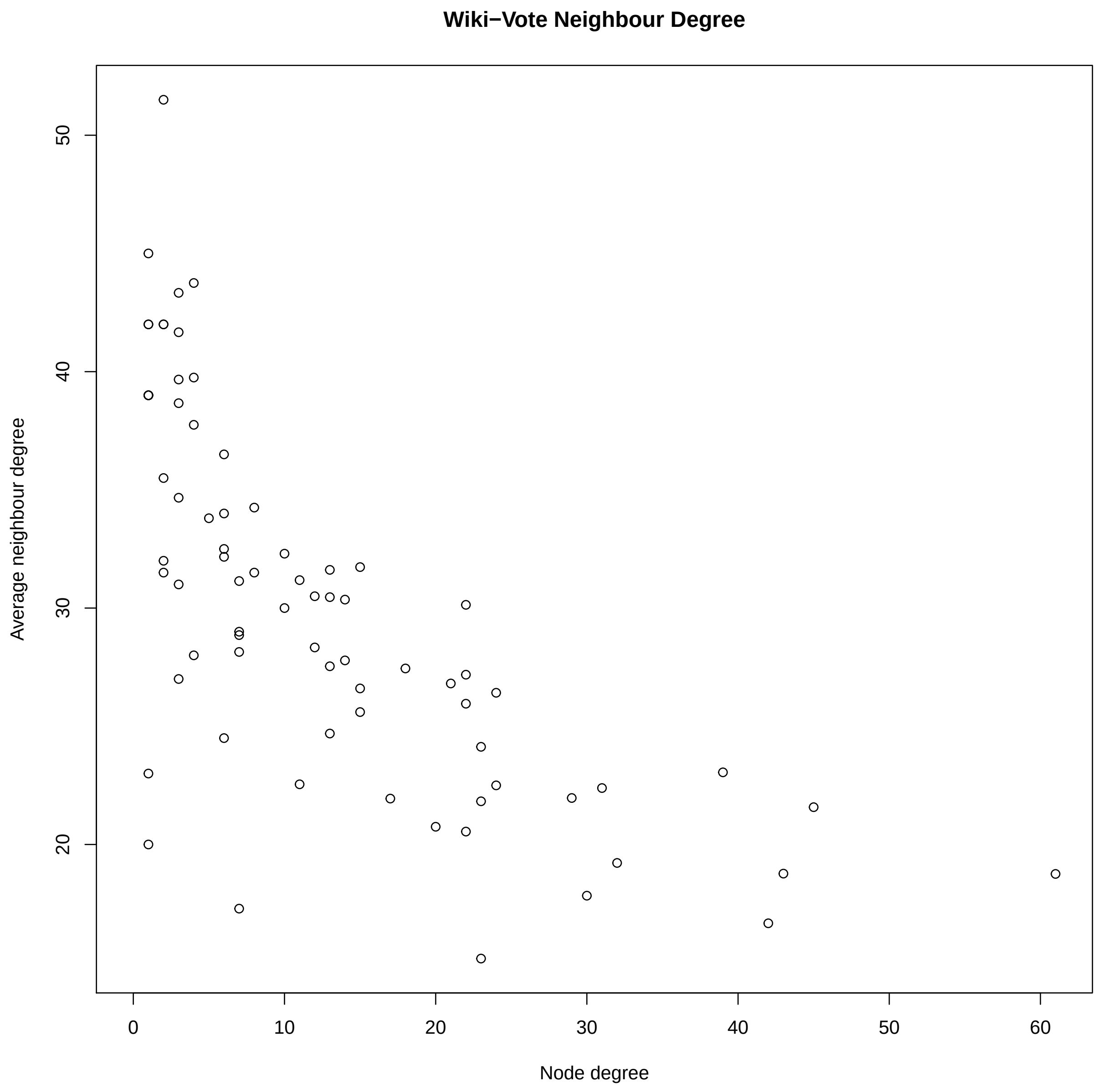


```
paste0("Diameter paths in giant component count: ", wv_g_subg.giant.subg.diam.count)
```

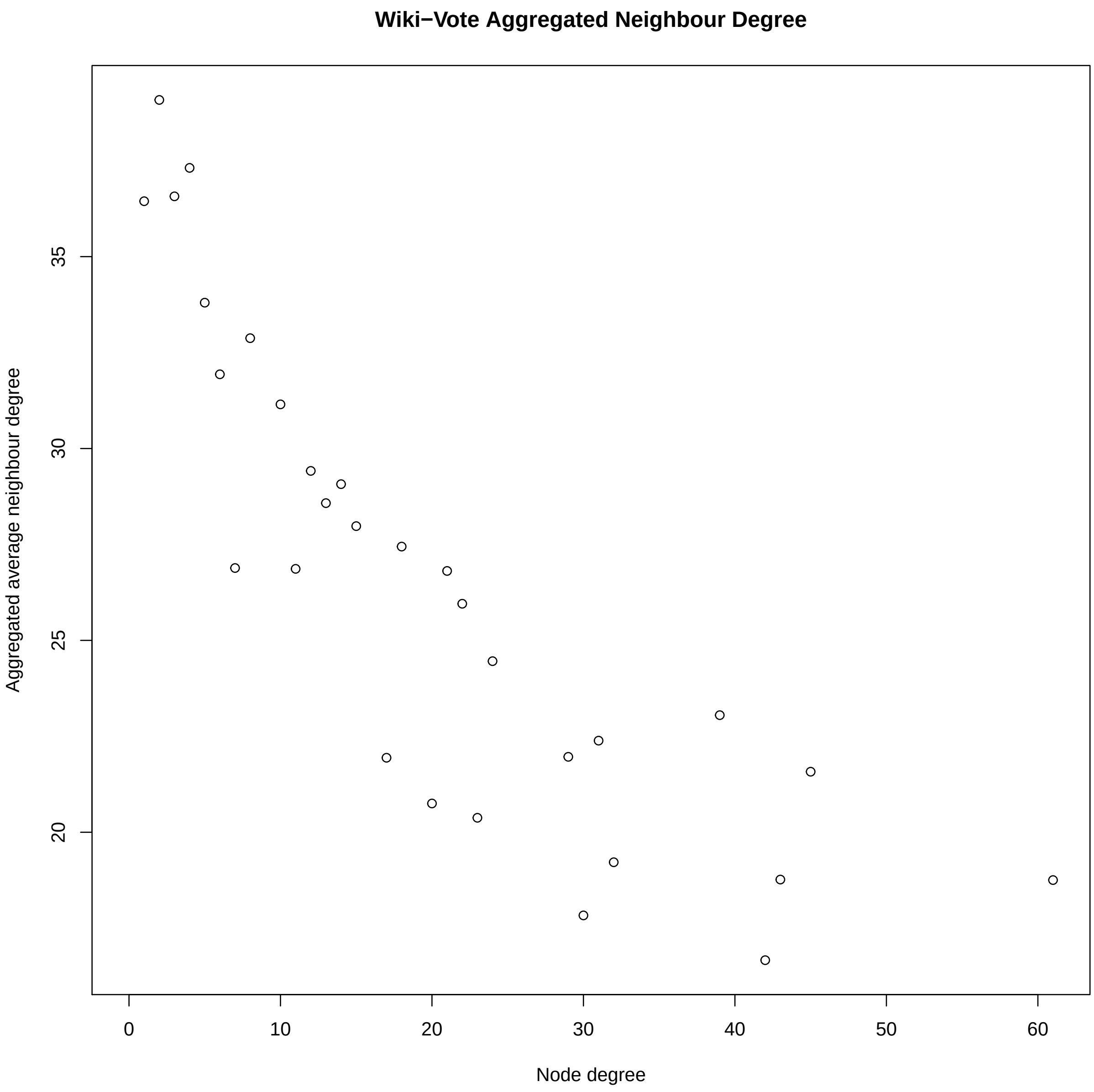
```
## [1] "Diameter paths in giant component count: 3"
```

**9. Make average neighbour degree vs node degree scatter plot (one point on the plot per node) and aggregated plot, averaging over all nodes with the same degree (aggregated average vs degree, one value per degree). Explain your observations.**

```
wv_g_subg.avg.nd = graph.knn(wv_g_subg)$knn
wv_g_subg.nd = degree(wv_g_subg)
plot(wv_g_subg.avg.nd ~ wv_g_subg.nd,
     main = "Wiki-Vote Neighbour Degree", xlab = "Node degree",
     ylab = "Average neighbour degree")
```



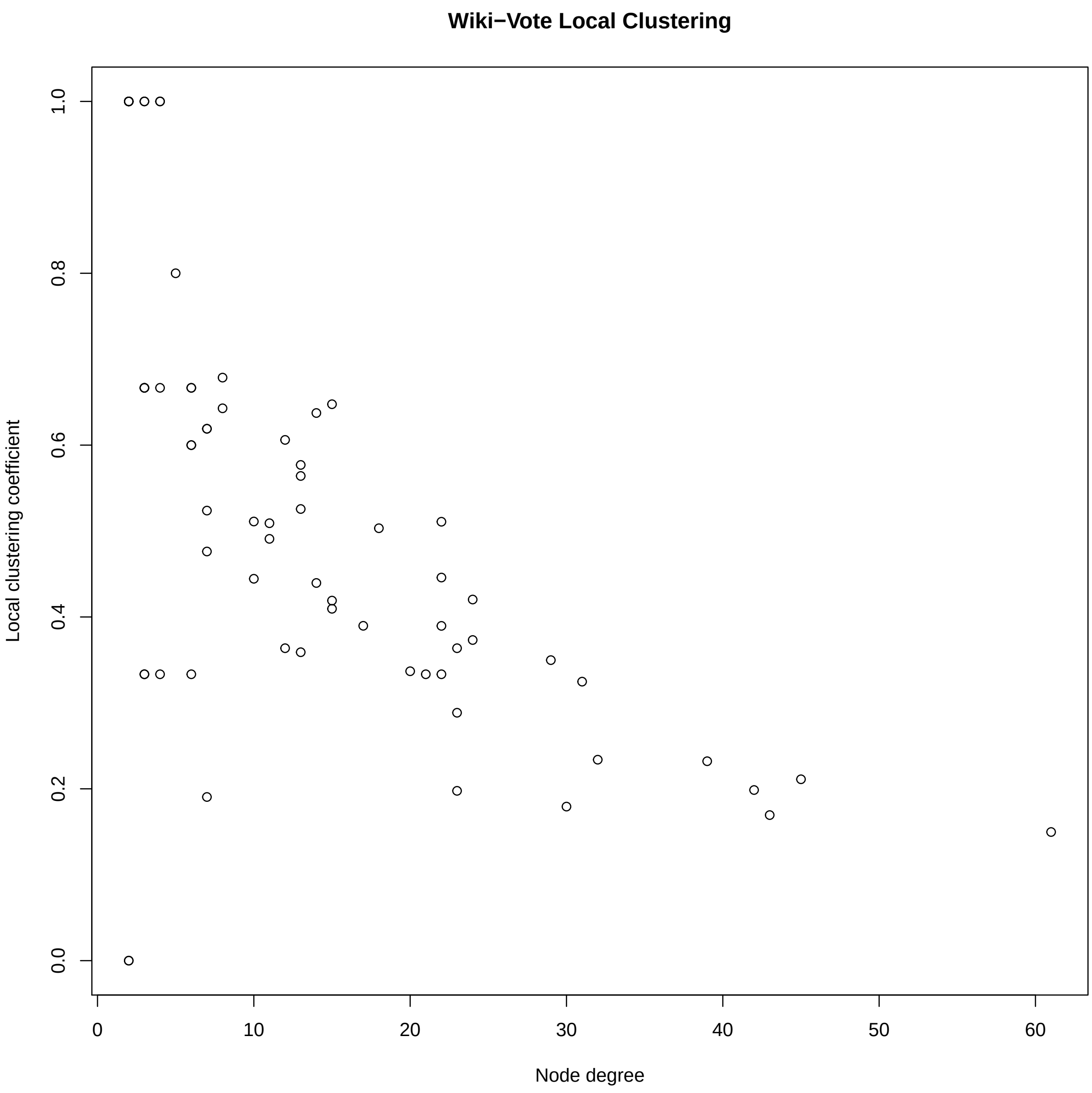
```
wv_g_subg.nd.uni = unique(wv_g_subg.nd)
wv_g_subg.aggr.avg.nd = c()
for (i in seq_along(wv_g_subg.nd.uni)) {
  ud = wv_g_subg.nd.uni[i]
  wv_g_subg.aggr.avg.nd[i] =
      sum(wv_g_subg.avg.nd[which(ud == wv_g_subg.nd)]) / length(which(ud == wv_g_subg.nd))
}
plot(wv_g_subg.aggr.avg.nd ~ wv_g_subg.nd.uni,
     main = "Wiki-Vote Aggregated Neighbour Degree", xlab = "Node degree",
     ylab = "Aggregated average neighbour degree")
```



```
# Observations explanation.
# From Wiki-Vote Neighbour Degree plot it is clear that the lower a node's degree is,
# the more neighbours it is expected to have, and vice-versa. This is inherent
# to the way a graph's giant component is formed: The overwhelming majority
# of its nodes are clustered so that within the giant component a node's neighbour
# degree is in average greater than outside.
```

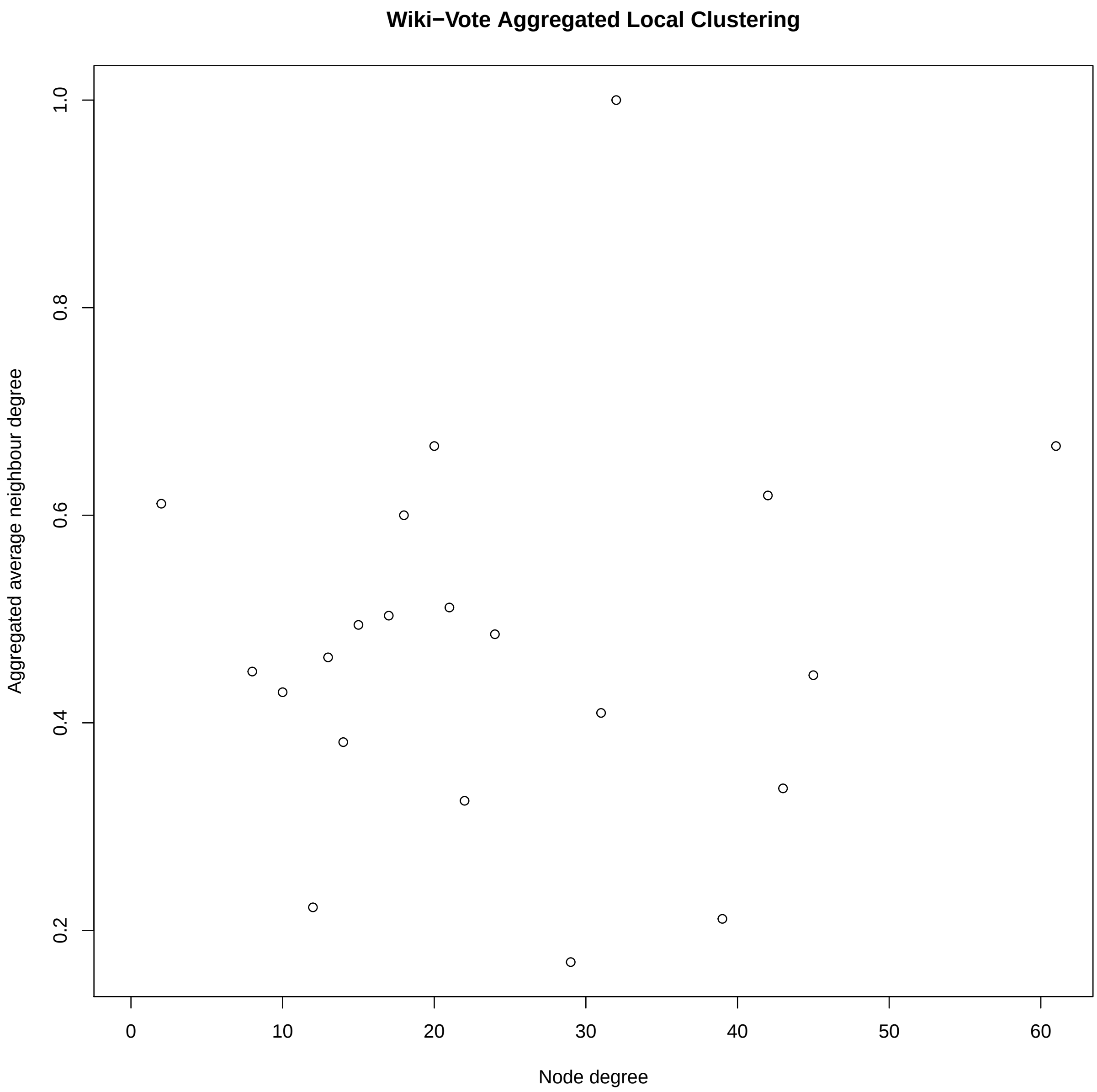
**10. Make local clustering coeff vs node degree scatter plot (one point on the plot per node) and aggregated, averaging over all nodes with the same degree (aggregated average vs degree, one value per degree). Explain your observations.**

```
wv_g_subg.cc.naomit = na.omit(transitivity(wv_g_subg, type = "local"))
wv_g_subg.nd.na = wv_g_subg.nd
for(i in seq_along(wv_g_subg.nd)) {
  if (i %in% na.action(wv_g_subg.cc.naomit)) {
    wv_g_subg.nd.na[i] = NA
  }
}
plot(wv_g_subg.cc.naomit ~ na.omit(wv_g_subg.nd.na),
     main = "Wiki-Vote Local Clustering", xlab = "Node degree",
     ylab = "Local clustering coefficient")
```



```
wv_g_subg.nd.uni = unique(wv_g_subg.nd)
wv_g_subg.aggr.avg.cc = c()
```

```
for (i in seq_along(wv_g_subg.nd.uni)) {
  ud = wv_g_subg.nd.uni[i]
  wv_g_subg.aggr.avg.cc[i] =
      sum(wv_g_subg.cc.naomit[which(ud == wv_g_subg.nd)]) / length(which(ud == wv_g_subg.nd))
}
plot(wv_g_subg.aggr.avg.cc ~ wv_g_subg.nd.uni,
     main = "Wiki-Vote Aggregated Local Clustering", xlab = "Node degree",
     ylab = "Aggregated average neighbour degree")
```



```
# Observations explanation.
# From Wiki-Vote Local Clustering plot it is clear that there is a negative correlation
# between local clustering coefficient and node degree.
```