Widely cited global irrigation statistics lack empirical support

R code

Arnald Puy

Contents

```
#   PRELIMINARY FUNCTIONS ##########################################################

sensobol::load_packages(c("openxlsx", "data.table", "tidyverse", "bibliometrix",
                          "igraph", "ggraph", "cowplot", "tidygraph", "benchmarkme",
                          "parallel", "wesanderson", "scales", "countrycode",
                          "doParallel", "foreach", "sensobol", "sp", "raster","ncdf4",
                          "readxl", "fitdistrplus"))

# Create custom theme
theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent",
                                           color = NA),
          legend.key = element_rect(fill = "transparent",
                                    color = NA),
          strip.background = element_rect(fill = "white"),
          legend.margin = margin(0.5, 0.1, 0.1, 0.1),
          legend.box.margin = margin(0.2,-4,-7,-7),
          plot.margin = margin(3, 4, 0, 4),
          legend.text = element_text(size = 8),
          axis.title = element_text(size = 10),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.title = element_text(size = 7.8))
}
```

1 Network analysis

```
# CREATE VECTORS TO READ IN AND CLEAN THE DATASETS ##############################

tmp <- list()
names.files <- c("WORK", "NETWORK")
topics <- c("water", "food")
corpus <- c("abstract.corpus", "policy.corpus", "full.text.corpus")
cols_of_interest <- c("title", "author", "claim", "citation",
                      "document.type", "nature.claim")

# Paste all possible combinations of names ------------------------------------
```

```
combs <- expand.grid(corpus = corpus, topics = topics, approach = names.files)
all.files <- paste(paste(paste(combs$corpus, combs$topics, sep = "."), combs$approach, sep = "_
                   "xlsx", sep = ".")

# READ IN DATASETS AND TURN TO LOWERCAPS ########################################

tmp <- list()

for (i in 1:length(all.files)) {

  tmp[[i]] <- data.table(read.xlsx(all.files[i]))

  if (!str_detect(all.files[i], "NETWORK")) {

    tmp[[i]][, title:= tolower(title)]

     } else {

    tmp[[i]][, (cols_of_interest):= lapply(.SD, tolower), .SDcols = (cols_of_interest)]
  }
}

names(tmp) <- all.files

sub(".*\\.([^\\.]+)_.*", "\\1", all.files)
```

```
##  [1] "water" "water" "water" "food"  "food"  "food"  "water" "water" "water"
## [10] "food"  "food"  "food"
```

```
# CLEAN AND MERGE DATASETS ######################################################

# Work datasets ---------------------------------------------------------------

dataset.works <- all.files[str_detect(all.files, "_WORK")]
dataset.works.topics <- sub(".*\\.([^\\.]+)_.*", "\\1", dataset.works)

tmp.works <- tmp[dataset.works]
names(tmp.works) <- dataset.works.topics
lapply(tmp.works, function(dt) dt[, .(doi, title, claim.in.text)]) %>%
  rbindlist(., idcol = "topic") %>%
  .[, .N, .(topic, claim.in.text)]
```

```
##      topic claim.in.text     N
##     <char>        <char> <int>
##  1:  water             F  1377
##  2:  water          <NA>   159
##  3:  water             T  2674
##  4:  water     Paywalled     9
```

```
##  5:  water       Russian     1
##  6:  water        French     1
##  7:  water        Indian     1
##  8:  water      Ukranian     1
##  9:  water    Portuguese     1
## 10:  water             T     2
## 11:   food          <NA>   204
## 12:   food             T   649
## 13:   food             F  2875
```

```
# Network datasets ------------------------------------------------------------

dataset.networks <- all.files[str_detect(all.files, "NETWORK")]
dataset.networks.topics <- sub(".*\\.([^\\.]+)_.*", "\\1", dataset.networks)

tmp2 <- tmp[dataset.networks]
names(tmp2) <- dataset.networks.topics

network.dt <- rbindlist(tmp2, idcol = "topic") %>%
  .[, policy:= grepl("^policy", doi)]  %>%
  .[, document.type:= trimws(document.type)] %>%
  .[, document.type:= tolower(document.type)]

# Retrieve year ---------------------------------------------------------------

network.dt[, year:= as.integer(sub(".* (\\d{4})[a-z]?$", "\\1", author))]
```

```
## Warning in eval(jsub, SDenv, parent.frame()): NAs introduced by coercion
```

```
# move policy to author -------------------------------------------------------

network.dt[, author:= ifelse(policy == TRUE, doi, author)]

# CHECK NUMBER OF FAO AQUASTAT CITES ##########################################

aquastat.cites <- network.dt[citation %like% "fao aquastat"] %>%
  .[, .N, .(citation, topic)]

aquastat.cites
```

```
##              citation  topic     N
##                 <char> <char> <int>
##  1: fao aquastat 2006  water    31
##  2:      fao aquastat  water    48
##  3: fao aquastat 2010  water    11
##  4: fao aquastat 2020  water     3
##  5: fao aquastat 2011  water     3
##  6: fao aquastat 2012  water     9
##  7: fao aquastat 2021  water     4
```

```
##  8: fao aquastat 2017  water     2
##  9: fao aquastat 2015  water     9
## 10: fao aquastat 2019  water     4
## 11: fao aquastat 2016  water    22
## 12: fao aquastat 2014  water     5
## 13: fao aquastat 2023  water     4
## 14: fao aquastat 2018  water     5
## 15: fao aquastat 2004  water     6
## 16: fao aquastat 2005  water     3
## 17:     fao aquastat   water     1
## 18: fao aquastat 2003  water     2
## 19: fao aquastat 2013  water     4
## 20: fao aquastat 2008  water     1
## 21: fao aquastat 2022  water     1
## 22:      fao aquastat   food     8
## 23: fao aquastat 2014   food     2
## 24: fao aquastat 2012   food     9
## 25: fao aquastat 2019   food     1
## 26: fao aquastat 2016   food     6
## 27: fao aquastat 2018   food     1
## 28: fao aquastat 2020   food     1
## 29: fao aquastat 2015   food     1
## 30: fao aquastat 2022   food     2
## 31: fao aquastat 2021   food     1
##              citation  topic     N
```

```
oldest.aquastat.cite <- min(as.integer(sub(".* (\\d{4})[a-z]?$", "\\1",
                                           aquastat.cites$citation)),
    na.rm = TRUE)
```

```
## Warning: NAs introduced by coercion
```

```
# CHECK NUMBER OF FAOSTAT CITES ###################################################

faostat.cites <- network.dt[citation %like% "faostat"] %>%
  .[, .N, .(citation, topic)]

faostat.cites
```

```
##                   citation  topic     N
##                     <char> <char> <int>
##  1: faostat online service  water     1
##  2:           faostat 2011  water     3
##  3:           faostat 2019  water     2
##  4:           faostat 2008  water     2
##  5:           faostat 2020  water     3
##  6:           faostat 2012  water     1
##  7:           faostat 2021  water     2
##  8:         faostat online  water     1
```

```
##  9:                faostat  water     3
## 10:               faostat   water     5
## 11:           faostat 2023  water     1
## 12:           faostat 2012   food     2
## 13:           faostat 2011   food     1
## 14:           faostat 2019   food     2
## 15:           faostat 2002   food     2
## 16:           faostat 2016   food     2
## 17:           faostat 2003   food     1
## 18:           faostat 1999   food     1
## 19:                faostat   food     1
```

```
oldest.faostat.cite <- min(as.integer(sub(".* (\\d{4})[a-z]?$", "\\1",
                                          faostat.cites$citation)),
                           na.rm = TRUE)
```

```
## Warning: NAs introduced by coercion
```

```
# WRITE LOOKUP TABLE TO CHECK ALREADY RETRIEVED STUDIES ##########################

lookup.dt <- network.dt[, .(doi, title, author, topic)] %>%
  .[order(title)] %>%
  unique(.)

lookup.dt[, .(number.rows = nrow(.SD)), topic]
```

```
##     topic number.rows
##    <char>       <int>
## 1:   food         906
## 2:  water        3671
```

```
# Export lookup tables ------------------------------------------------------

write.xlsx(lookup.dt, "lookup.dt.xlsx")
write.xlsx(lookup.dt[topic == "water"], "lookup.water.dt.xlsx")
write.xlsx(lookup.dt[topic == "food"], "lookup.food.dt.xlsx")

# Remove the year from mentions to FAO Aquastat -------------------------------

pattern <- "\\b(?:19|20)\\d{2}\\b"  # Matches years between 1900 and 2099

for (col in c("citation", "author")) {
  matches <- grepl("^fao aquastat\\s+\\d+$", network.dt[[col]], ignore.case = TRUE)
  network.dt[matches, (col) := gsub("\\d+", "", network.dt[[col]][matches], perl = TRUE)]
  network.dt[, (col) := trimws(network.dt[[col]])]
}

# Remove the year from mentions to FAOSTAT ------------------------------------

for (col in c("citation", "author")) {
```

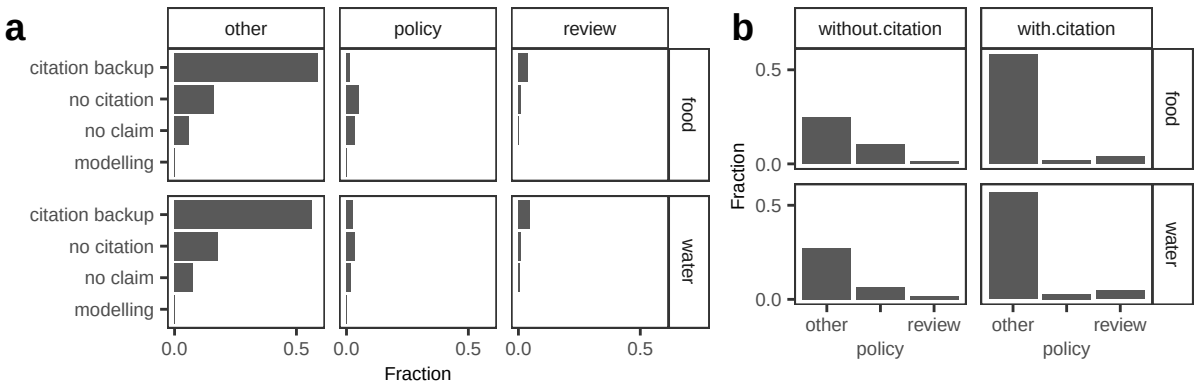
```
  matches <- grepl("^faostat\\s+\\d+$", network.dt[[col]], ignore.case = TRUE)
  network.dt[matches, (col) := gsub("\\d+", "", network.dt[[col]][matches], perl = TRUE)]
  network.dt[, (col) := trimws(network.dt[[col]])]
}

# Rename columns ------------------------------------------------------------

setnames(network.dt, c("author", "citation"), c("from", "to"))

# Rename category -----------------------------------------------------------

network.dt[, category:= ifelse(!classification == "F", "Uncertain", "Fact")]

# Create copy and remove duplicated -----------------------------------------

network.dt.claim <- copy(network.dt)
network.dt.claim <- unique(network.dt.claim, by = c("from", "to", "document.type",
                                                    "nature.claim"))
cols_to_change <- colnames(network.dt)
network.dt.claim[, (cols_to_change):= lapply(.SD, trimws), .SDcols = (cols_to_change)]
network.dt.claim[, (cols_to_change):= lapply(.SD, str_squish), .SDcols = (cols_to_change)]

fwrite(network.dt.claim, "network.dt.claim.csv")

# Convert all to lower caps -------------------------------------------------

network.dt <- network.dt[, .(from, to, year, document.type, nature.claim,
                             classification, category, topic)]
cols_to_change <- colnames(network.dt)
network.dt[, (cols_to_change):= lapply(.SD, trimws), .SDcols = (cols_to_change)]
network.dt[, (cols_to_change):= lapply(.SD, str_squish), .SDcols = (cols_to_change)]
```

```
# PLOT DESCRIPTIVE STATISTICS ###########################################################

total.rows <- network.dt[, .(number.rows = nrow(.SD)), topic]

# Check proportion of studies by nature of claim ----------------------------

network.dt.claim[, .N, .(nature.claim, topic)] %>%
  merge(., total.rows, by = "topic") %>%
  .[, fraction:= N / number.rows] %>%
  print()
```

```
## Key: <topic>
##       topic    nature.claim     N number.rows    fraction
##      <char>          <char> <int>       <int>       <num>
##  1:   food     no citation   195        1043 0.186960690
##  2:   food citation backup   620        1043 0.594439118
```

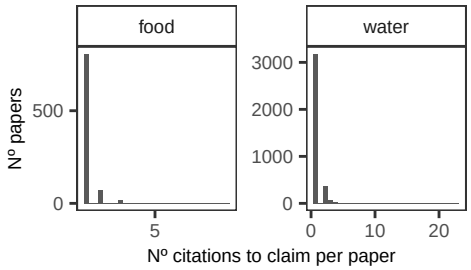
```
##  3:   food        no claim    93        1043 0.089165868
##  4:   food            <NA>    37        1043 0.035474593
##  5:   food       modelling     3        1043 0.002876318
##  6:  water citation backup  2751        4352 0.632123162
##  7:  water       modelling    20        4352 0.004595588
##  8:  water     no citation   973        4352 0.223575368
##  9:  water            <NA>   116        4352 0.026654412
## 10:  water        no claim   439        4352 0.100873162
```

```
# Count document type by nature of claim -------------------------------------

a <- network.dt[, .N, .(nature.claim, document.type, topic)] %>%
  merge(., total.rows, by = "topic") %>%
  .[, proportion:= N / number.rows] %>%
  na.omit() %>%
  ggplot(., aes(reorder(nature.claim, proportion), proportion)) +
  coord_flip() +
  geom_bar(stat = "identity") +
  facet_grid(topic~document.type) +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  labs(x = "", y = "Fraction") +
  theme_AP()

# Count how many documents make the claim and cite / do not cite,
# by document.type ------------------------------------------------------------

b <- network.dt[, .(without.citation = sum(is.na(to)),
                    with.citation = .N - sum(is.na(to))), .(document.type, topic)] %>%
  melt(., measure.vars = c("without.citation", "with.citation")) %>%
  merge(., total.rows, by = "topic") %>%
  .[, proportion:= value / number.rows] %>%
  ggplot(., aes(document.type, proportion)) +
  geom_bar(stat = "identity") +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  labs(x = "", y = "Fraction") +
  facet_grid(topic~variable) +
  theme_AP()

# merge ----------------------------------------------------------------------

plot.claim <- plot_grid(a, b, ncol = 2, rel_widths = c(0.6, 0.4), labels = "auto")
plot.claim
```




```
# PLOT DISTRIBUTION OF CITATION SUPPORTING THE CLAIM ############################

plot.supporting.claim <- network.dt[, .N, .(from, topic)] %>%
  .[order(-N)] %>%
  ggplot(., aes(N)) +
  geom_histogram() +
  facet_wrap(~topic, scale = "free") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  scale_y_continuous(breaks = breaks_pretty(n = 3)) +
  theme_AP() +
  labs(x = "Nº citations to claim per paper", y = "Nº papers")

plot.supporting.claim
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



1.1 Network metrics

```
# CALCULATE NETWORK METRICS ###################################################

# only complete cases ----------------------------------------------------------

network.dt.complete <- network.dt[complete.cases(network.dt$to), ]
split.networks <- split(network.dt.complete, network.dt.complete$topic)

# Export-----
write.xlsx(network.dt.complete, "network.dt.complete.xlsx")

# Transform to graph -----------------------------------------------------------
```

```r
citation_graph <- lapply(split.networks, function(dt)
  graph_from_data_frame(d = dt, directed = TRUE))
```

```
## Warning: In `d`, `NA` elements were replaced with string "NA".
## In `d`, `NA` elements were replaced with string "NA".
```

```r
# Calculate network metrics ------------------------------------------------

lapply(citation_graph, function(x) edge_density(x))
```

```
## $food
## [1] 0.001128925
##
## $water
## [1] 0.0003277328
```

```r
# Modularity:
# - c.1: Strong community structure, where nodes within groups are highly connected.
# - c. -1: Opposite of community structure, where nodes between groups are more connected.
# - c. 0: Indicates absence of community structure or anti-community structure in the network.
wtc <- lapply(citation_graph, function(x) cluster_walktrap(x))
lapply(wtc, function(x) modularity(x))
```

```
## $food
## [1] 0.9241608
##
## $water
## [1] 0.8739123
```

```r
network_metrics <- lapply(citation_graph, function(x)
  data.table(node = V(x)$name,

             # Degree of a node: The number of connections or
             # edges linked to that node.
             # It represents how well-connected or central a
             # node is within the graph.
             degree = degree(x, mode = "in"),

             degree.out = degree(x, mode = "out"),

             # Betweenness centrality of a node: Measures the
             # extent to which a node lies on the shortest
             # paths between all pairs of other nodes in the graph.
             # Nodes with high betweenness centrality act as
             # bridges or intermediaries, facilitating
             # communication and information flow between other nodes.
             betweenness = betweenness(x),

             # Closeness centrality of a node: Measures how
```

```
            # close a node is to all other nodes in the graph,
            # taking into account the length of the shortest paths.
            # Nodes with high closeness centrality are able to
            # efficiently communicate or interact with other
            # nodes in the graph.
            closeness = closeness(x),
            pagerank = page_rank(x)$vector)
)

# Define the max number of rows
max.number <- 3

degree.nodes <- lapply(network_metrics, function(dt) dt[order(-degree)][1:max.number])
degree.nodes.out <- lapply(network_metrics, function(dt) dt[order(-degree.out)][1:max.number])
betweenness.nodes <- lapply(network_metrics, function(dt) dt[order(-betweenness)][1:max.number]
pagerank.nodes <- lapply(network_metrics, function(dt) dt[order(-closeness)][1:max.number])

degree.nodes
```

```
## $food
##                 node degree degree.out betweenness closeness   pagerank
##               <char>  <num>      <num>       <num>     <num>      <num>
## 1:      fao aquastat     32          0           0       NaN 0.02938404
## 2:          fao 2002     18          0           0       NaN 0.01565168
## 3: morris et al 2003     16          0           0       NaN 0.01125169
##
## $water
##                  node degree degree.out betweenness closeness    pagerank
##                <char>  <num>      <num>       <num>     <num>       <num>
## 1:       fao aquastat    178          0      0.0000       NaN 0.063812081
## 2: siebert et al 2010     67          3    367.8333     0.125 0.008105668
## 3:           fao 2011     66          1    138.5000     1.000 0.017543929
```

```
degree.nodes.out
```

```
## $food
##                 node degree degree.out betweenness  closeness     pagerank
##               <char>  <num>      <num>       <num>      <num>        <num>
## 1: taguta et al 2022      0          9           0 0.04761905 0.0007552736
## 2:    pei et al 2017      1          6           4 0.20000000 0.0008836701
## 3: kadigi et al 2004      0          6           0 0.10000000 0.0007552736
##
## $water
##                 node degree degree.out betweenness  closeness     pagerank
##               <char>  <num>      <num>       <num>      <num>        <num>
## 1:        wada 2015      0         23           0 0.02702703 0.0001773598
## 2: wada et al 2014a      1          9          10 0.09090909 0.0002075110
## 3: zarei et al 2021      0          9           0 0.06666667 0.0001773598
```

```
betweenness.nodes
```

```
## $food
##                           node degree degree.out betweenness closeness
##                         <char>  <num>      <num>       <num>     <num>
## 1:                 postel 1999      9          3          30 0.3333333
## 2: vorosmarty and sahagian 2000      5          3          15 0.3333333
## 3:           siebert et al 2005     13          2          14 1.0000000
##       pagerank
##          <num>
## 1: 0.006436819
## 2: 0.003965186
## 3: 0.008303918
##
## $water
##                     node degree degree.out betweenness closeness    pagerank
##                   <char>  <num>      <num>       <num>     <num>       <num>
## 1:    siebert et al 2010     67          3    367.8333 0.1250000 0.008105668
## 2:             doll 2009     12          1    200.0000 0.3333333 0.004984458
## 3: boretti and rosa 2019     12          5    179.5000 0.0400000 0.001630897
```

```
pagerank.nodes
```

```
## $food
##                     node degree degree.out betweenness closeness     pagerank
##                   <char>  <num>      <num>       <num>     <num>        <num>
## 1: okorogbona et.al 2018      0          1           0         1 0.0007552736
## 2:   du preez et al 2018      0          1           0         1 0.0007552736
## 3:      meier et al 2018      3          1           3         1 0.0023602299
##
## $water
##                     node degree degree.out betweenness closeness     pagerank
##                   <char>  <num>      <num>       <num>     <num>        <num>
## 1: sharma and irmak 2012      0          1           0         1 0.0001773598
## 2:       world bank 2007      5          1          46         1 0.0052992873
## 3:   brajovic et al 2015      0          1           0         1 0.0001773598
```

1.2 Network plots

```
# ADD FEATURES TO NODES ############################################################

# Retrieve a vector with the node names ----------------------------------------

graph <- lapply(split.networks, function(nt)
  tidygraph::as_tbl_graph(nt, directed = TRUE))
```

```
## Warning: In `d`, `NA` elements were replaced with string "NA".
## In `d`, `NA` elements were replaced with string "NA".
```

```r
vec.names <- lapply(graph, function(graph)
  graph %>%
    activate(nodes) %>%
    pull() %>%
    data.table(name = .))


# Merge with info from the network.dt ----------------------------------------

tmp.network <- split(network.dt, network.dt$topic)

vec.nature.claim <- list()

for(i in names(tmp.network)) {

  vec.nature.claim[[i]] <- merge(merge(vec.names[[i]], unique(tmp.network[[i]][, .(from, year,
                                       by.x = "name", by.y = "from", all.x = TRUE),
                                 unique(tmp.network[[i]][, .(from, document.type, classificati
                                 by.x = "name", by.y = "from", all.x = TRUE)
}

# Merge with the correct order -----------------------------------------------

order_indices <- final.vec.nature.claim <- final.vec.document.type <-
  final.vec.year <- final.vec.classification <- final.vec.category <- list()

for (i in names(vec.names)) {

  order_indices[[i]] <- match(vec.names[[i]]$name, vec.nature.claim[[i]]$name)
  final.vec.nature.claim[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, nature.claim]
  final.vec.document.type[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, document.type]
  final.vec.year[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, year] %>%
    as.numeric()
  final.vec.classification[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, classification]
  final.vec.category[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, category]
}

# Attach to the graph --------------------------------------------------------

graph.final <- list()

for (i in names(graph)) {
```

```
  graph.final[[i]] <- graph[[i]] %>%
    activate(nodes) %>%
    mutate(nature.claim = final.vec.nature.claim[[i]],
           document.type = final.vec.document.type[[i]],
           year = final.vec.year[[i]],
           degree = network_metrics[[i]]$degree,
           classification = final.vec.classification[[i]],
           category = final.vec.category[[i]],
           degree.out = network_metrics[[i]]$degree.out,
           betweenness = network_metrics[[i]]$betweenness,
           pagerank = network_metrics[[i]]$pagerank)


}

for (i in names(graph.final)) {

  graph.final[[i]] <- graph.final[[i]] %>%
    activate(edges) %>%
    mutate(edge_color = .N()$nature.claim[to])
}
```

```
# EXPORT NODES AND EDGES ###############################################################

for (i in topics) {

  nodes <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    data.table()

  edges <- graph.final[[i]] %>%
    activate(edges) %>%
    data.frame() %>%
    data.table()

  write.xlsx(nodes, paste(i, ".nodes.xlsx", sep = ""))
  write.xlsx(edges, paste(i, ".edges.xlsx", sep = ""))

}
```

```
# NUMBER OF NODES ######################################################################

lapply(graph.final, function(graph) V(graph))
```

```
## $food
## + 768/768 vertices, named, from fd213ec:
##   [1] okorogbona et.al 2018
```

```
##   [2] du preez et al 2018
##   [3] niu et al 2023
##   [4] meier et al 2018
##   [5] lobell et al 2006
##   [6] rosa 2022
##   [7] rolle et al 2021
##   [8] mitchell et al 2018
##   [9] wang et al 2012
##  [10] hanjra and qureshi 2010
## + ... omitted several vertices
##
## $water
## + 2925/2925 vertices, named, from c2ae920:
##    [1] sharma and irmak 2012
##    [2] doreau et al 2012
##    [3] world water assessment programme 2009
##    [4] world bank 2007
##    [5] brajovic et al 2015
##    [6] rivers et al 2015
##    [7] kijne 2005
##    [8] hafeez and khalid awan 2022
##    [9] dunkelman et al 2017
##   [10] nordin et al 2013
## + ... omitted several vertices
```

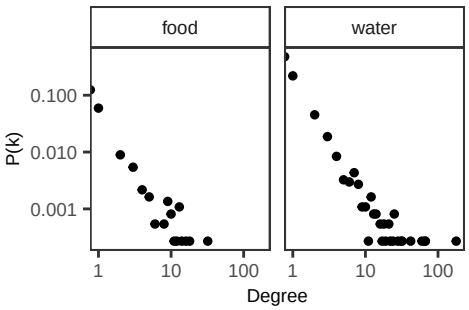
```
# NUMBER OF EDGES ##################################################################

lapply(graph.final, function(graph) ecount(graph))
```

```
## $food
## [1] 665
##
## $water
## [1] 2803
```

```
# SCALE-FREE PLOT ##################################################################

# Prepare data ----------------------------------------------------------------

dt.food <- graph.final[[1]] %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table() %>%
  .[, topic:= "food"]

dt.water <- graph.final[[2]] %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table() %>%
```

```
  .[, topic:= "water"]

tmp <- rbind(dt.food, dt.water)

# Calculate the degree distribution ----------------------------------------

degree_distribution <- tmp[, .(P_k = .N / nrow(tmp)), .(degree, topic)]

plot.degree.distribution <- degree_distribution %>%
  ggplot(., aes(degree, P_k)) +
  geom_point(size = 1) +
  scale_y_log10() +
  scale_x_log10() +
  facet_wrap(~topic) +
  labs(x = "Degree", y = "P(k)") +
  theme_AP()

plot.degree.distribution
```

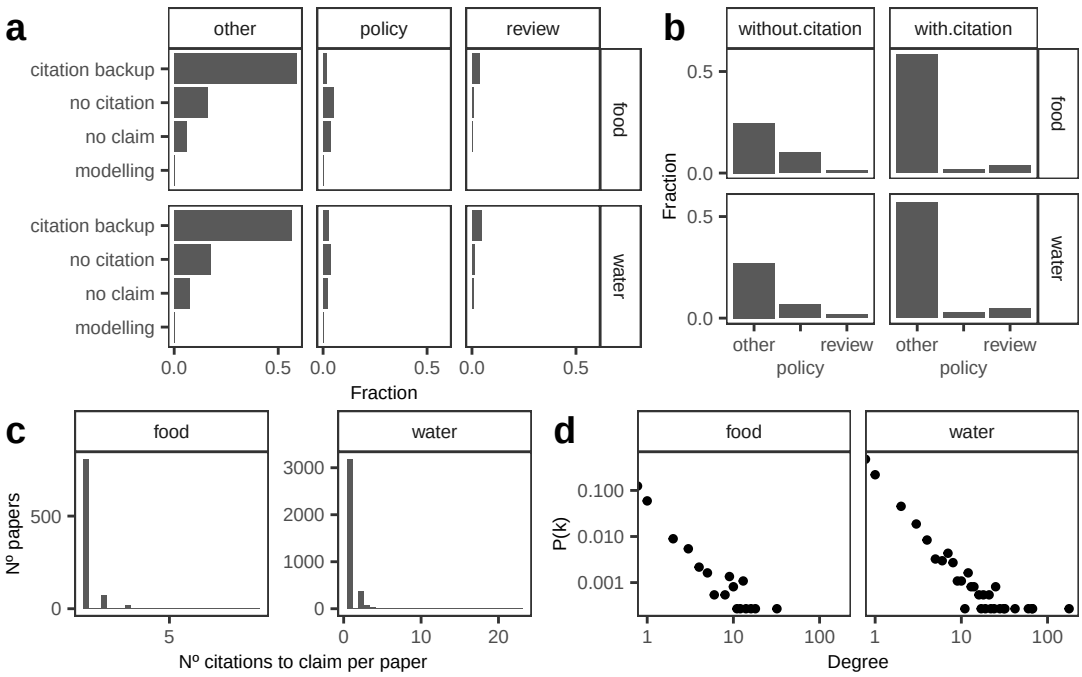
```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
```




```
# MERGE DESCRIPTIVE PLOTS ###################################################

bottom <- plot_grid(plot.supporting.claim, plot.degree.distribution, ncol = 2,
                    labels = c("c", "d"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
```

```
plot_grid(plot.claim, bottom, ncol = 1, rel_heights = c(0.6, 0.4))
```




```
# CALCULATE ALL POSSIBLE PATHS ###################################################

# Define function ------------------------------------------------------------

count_paths <- function(g) {

  # Extract the top 5 nodes with highest betweenness ---------------------------

  top_nodes <- g %>%
    activate(nodes) %>%
    top_n(5, betweenness) %>%
    pull(name)

  # Initialize counts ----------------------------------------------------------

  total_paths_count <- 0
  top_nodes_paths_count <- 0

  g <- as.igraph(g)

  results <- foreach(i = V(g),
                     .combine = "c",
                     .packages = "igraph") %:%

    foreach(j = V(g),
            .combine = "c") %dopar% {

              total_paths_pair <- 0
```

```
            top_nodes_paths_pair <- 0

            if (i != j) {

              # Calculate all possible paths -------------------------------

              paths <- all_simple_paths(g, from = i, to = j)
              total_paths_pair <- length(paths)

              # Check how many paths pass through the top betweenness nodes --

              for (path in paths) {

                if (any(names(path) %in% top_nodes)) {
                  top_nodes_paths_pair <- top_nodes_paths_pair + 1

                }
              }
            }

            # Return the count of all paths and paths through top nodes ------

            return(c(total_paths_pair, top_nodes_paths_pair))
          }

  # Aggregate results ---------------------------------------------------------

  total_paths_count <- sum(results[seq(1, length(results), by = 2)])
  top_nodes_paths_count <- sum(results[seq(2, length(results), by = 2)])

  return(c(total_paths_count, top_nodes_paths_count))

}

# Define parallel computing ----------------------------------------------------

cl <- makeCluster(floor(detectCores() * 0.75))
registerDoParallel(cl)

# Run the function -------------------------------------------------------------

results.counts <- lapply(graph.final, function(graph)
  count_paths(graph))

# Stop the cluster -------------------------------------------------------------

stopCluster(cl)
```

```
# SHOW TOTAL NUMBER OF PATHS AND PROPORTION OF PATHS PASSING
# THROUGH THE FIVE NODES WITH THE HIGHEST BETWEENNESS ############################

results.counts
```

```
## $food
## [1] 866 133
##
## $water
## [1] 5329 1375
```

```
lapply(results.counts, function(x) x[[2]] / x[[1]])
```

```
## $food
## [1] 0.1535797
##
## $water
## [1] 0.2580221
```

1.3 Citation paths ending in authorities

```
# CITATION PATHS TO AUTHORITIES ##################################################

# Define function ------------------------------------------------------------

citation_paths_to_authorities_fun <- function(graph) {

  name.authorities <- graph %>%
    activate(nodes) %>%
    data.frame() %>%
    data.table() %>%
    .[order(-degree)] %>%
    .[1:10] %>%
    .[, name]

  target_node_ids <- graph %>%
    activate(nodes) %>%
    filter(name %in% name.authorities) %>%
    pull(name)

  igraph_network <- as.igraph(graph)

  all_paths <- unlist(lapply(V(igraph_network), function(node) {

    all_simple_paths(igraph_network, from = node, to = name.authorities)

  }), recursive = FALSE)
```

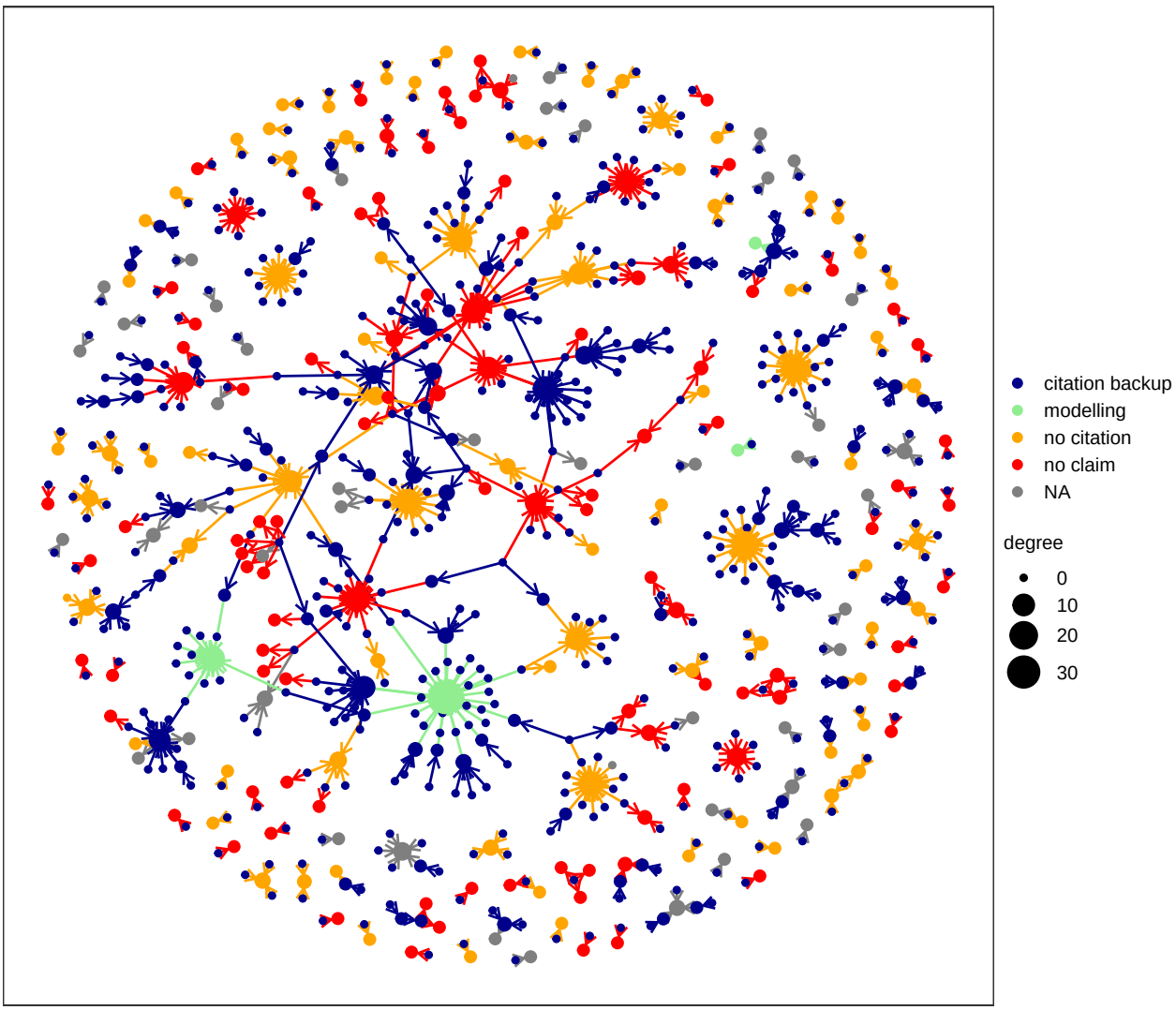
```
  num_paths_to_target <- length(all_paths)

  return(num_paths_to_target)
}

# Run function ------------------------------------------------------------

lapply(1:2, function(topic)
  citation_paths_to_authorities_fun(graph.final[[topic]]))
```

```
## [[1]]
## [1] 211
##
## [[2]]
## [1] 1363
```

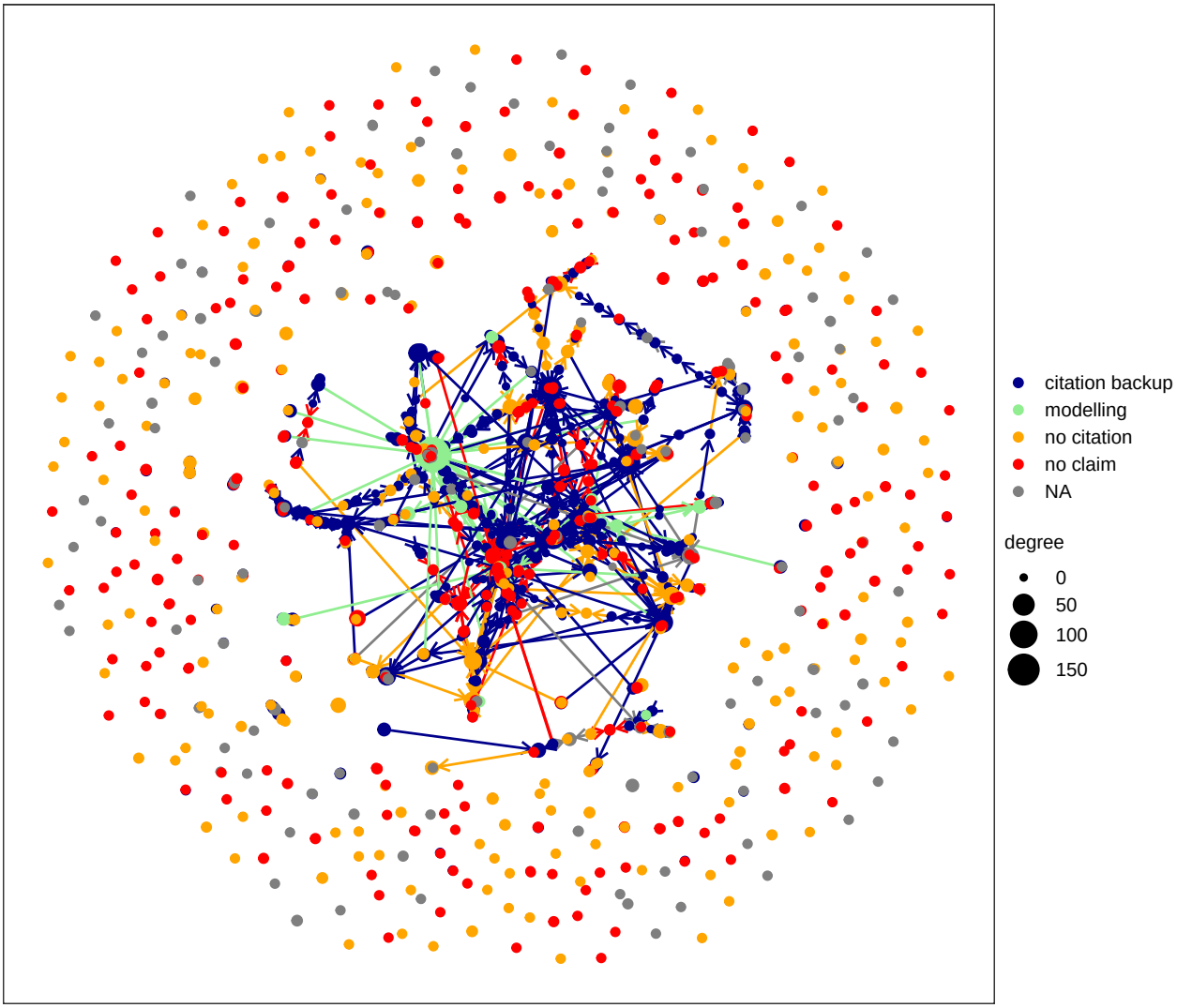
```
# PLOT NETWORK ##############################################################

seed <- 1234
selected_colors <- c("darkblue", "lightgreen", "orange", "red", "grey")

# by nature of claim ------------------------------------------------------

# Label the nodes with highest degree -------------------------------------

p1 <- p2 <- p3 <- p4 <- p5 <- list()

for(i in names(graph.final)) {

  set.seed(seed)

  p1[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm"),
                   aes(color = edge_color)) +
    scale_edge_color_manual(values = selected_colors, guide = "none") +
    geom_node_point(aes(color = nature.claim, size = degree)) +
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                       values = selected_colors) +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          legend.position = "right")
}
```

```
p1
```

```
## $food
```



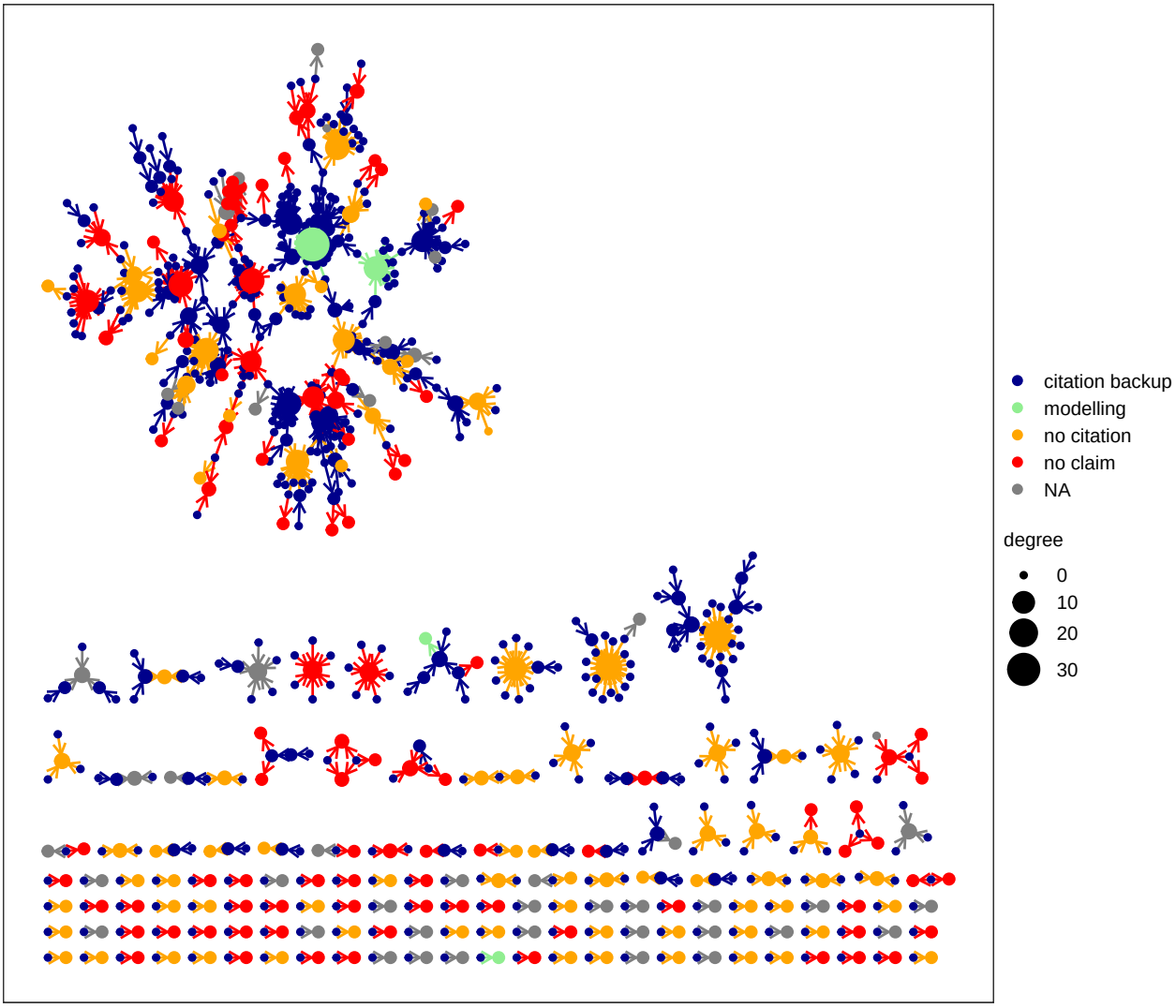

```
##
## $water
```



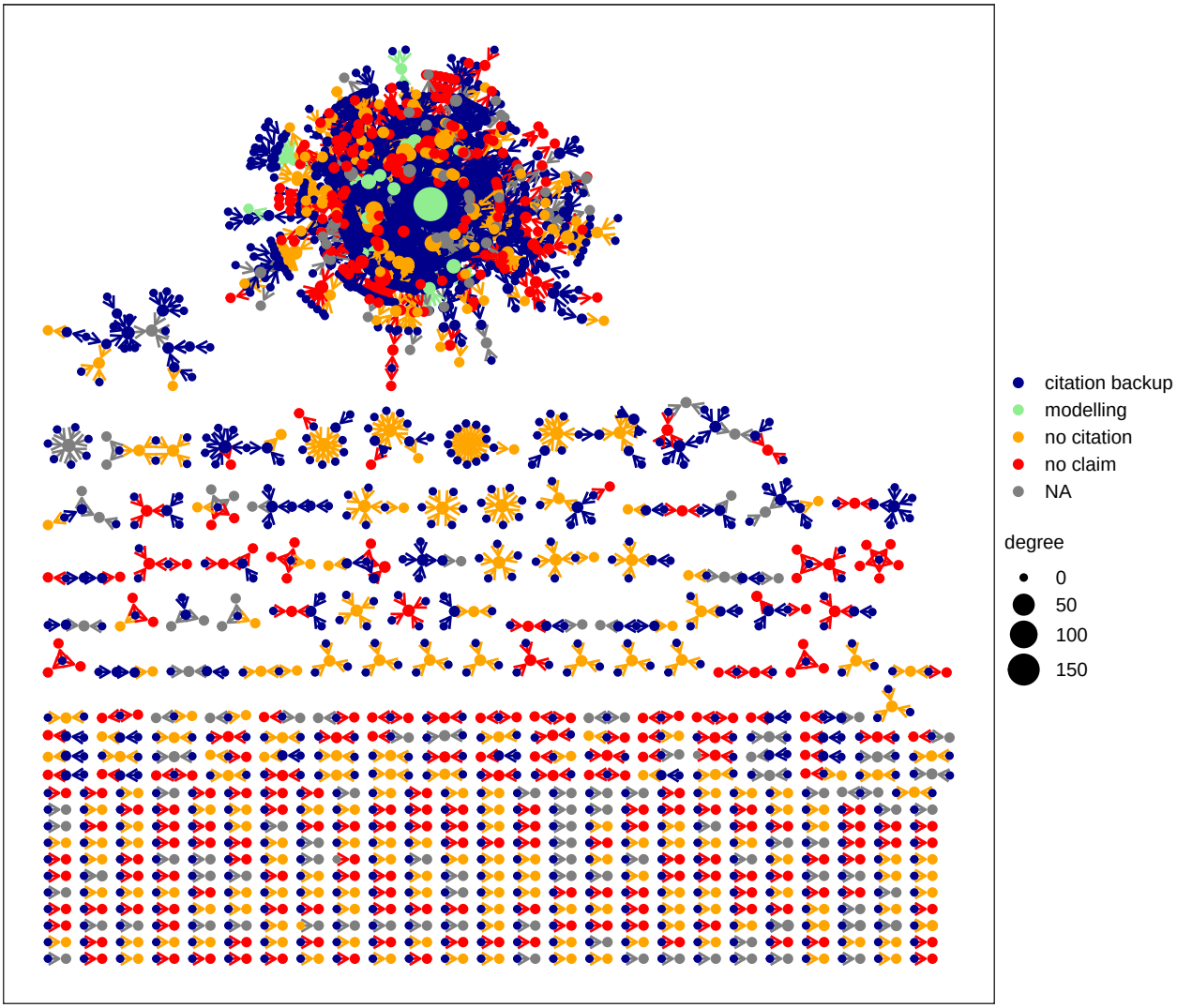

```
for(i in names(graph.final)) {

  set.seed(seed)

  p5[[i]] <- ggraph(graph.final[[i]], layout = "stress") +
    geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm"),
                   aes(color = edge_color)) +
    scale_edge_color_manual(values = selected_colors, guide = "none") +
    geom_node_point(aes(color = nature.claim, size = degree)) +
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                       values = selected_colors) +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
```

```
          axis.ticks.y = element_blank(),
          legend.position = "right")
}

p5
```

```
## $food
```




```
##
## $water
```

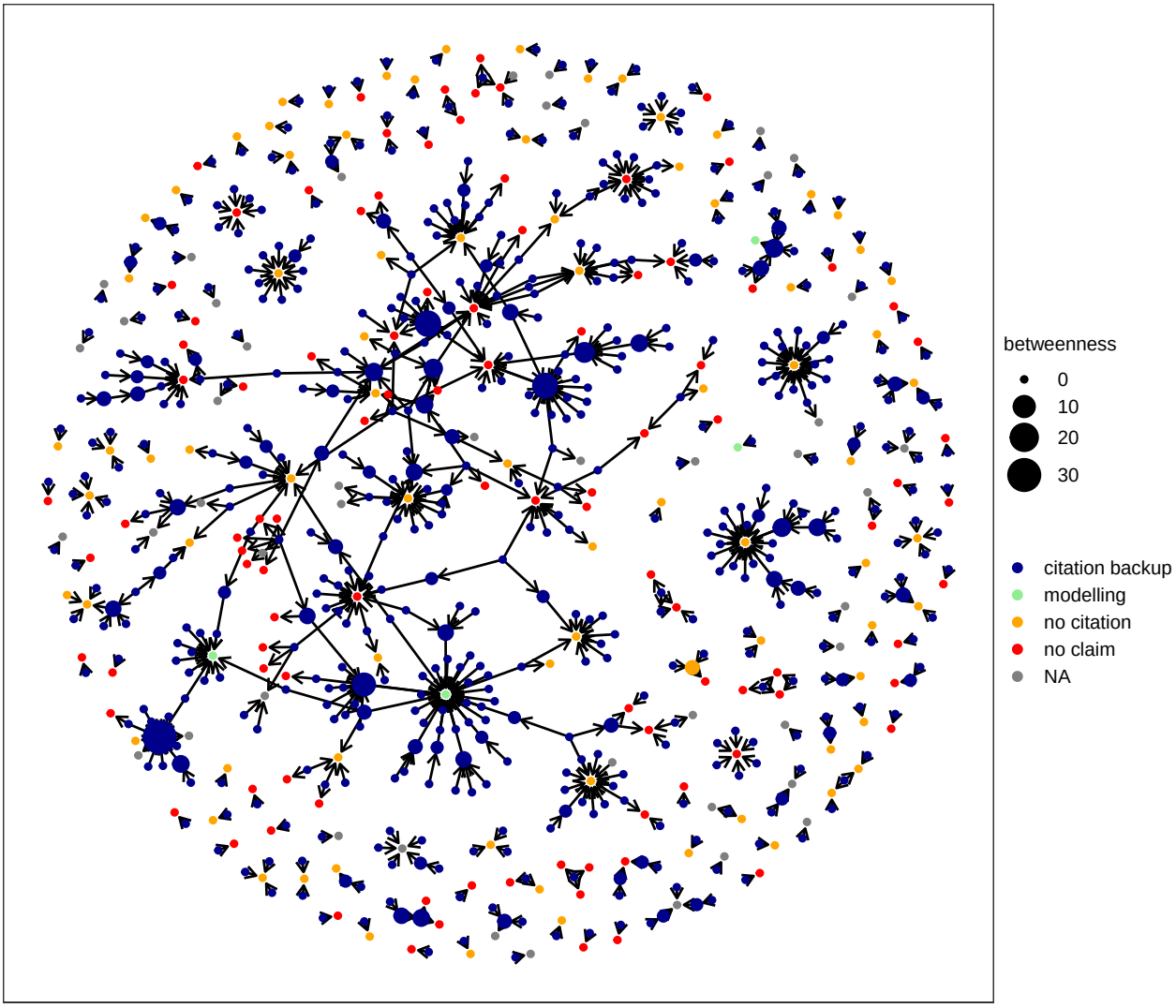



```
# Label the nodes with highest betweenness ------------------------------------

for (i in names(graph.final)) {

  set.seed(seed)

  p2[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = nature.claim, size = betweenness)) +
    geom_node_text(aes(label = ifelse(betweenness >= min(betweenness.nodes[[i]]$betweenness),
                                      name, NA)),
                   repel = TRUE, size = 2.2) +
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                       values = selected_colors) +
    theme_AP() +
```

```
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          legend.position = "right")
}

p2
```

```
## $food
```

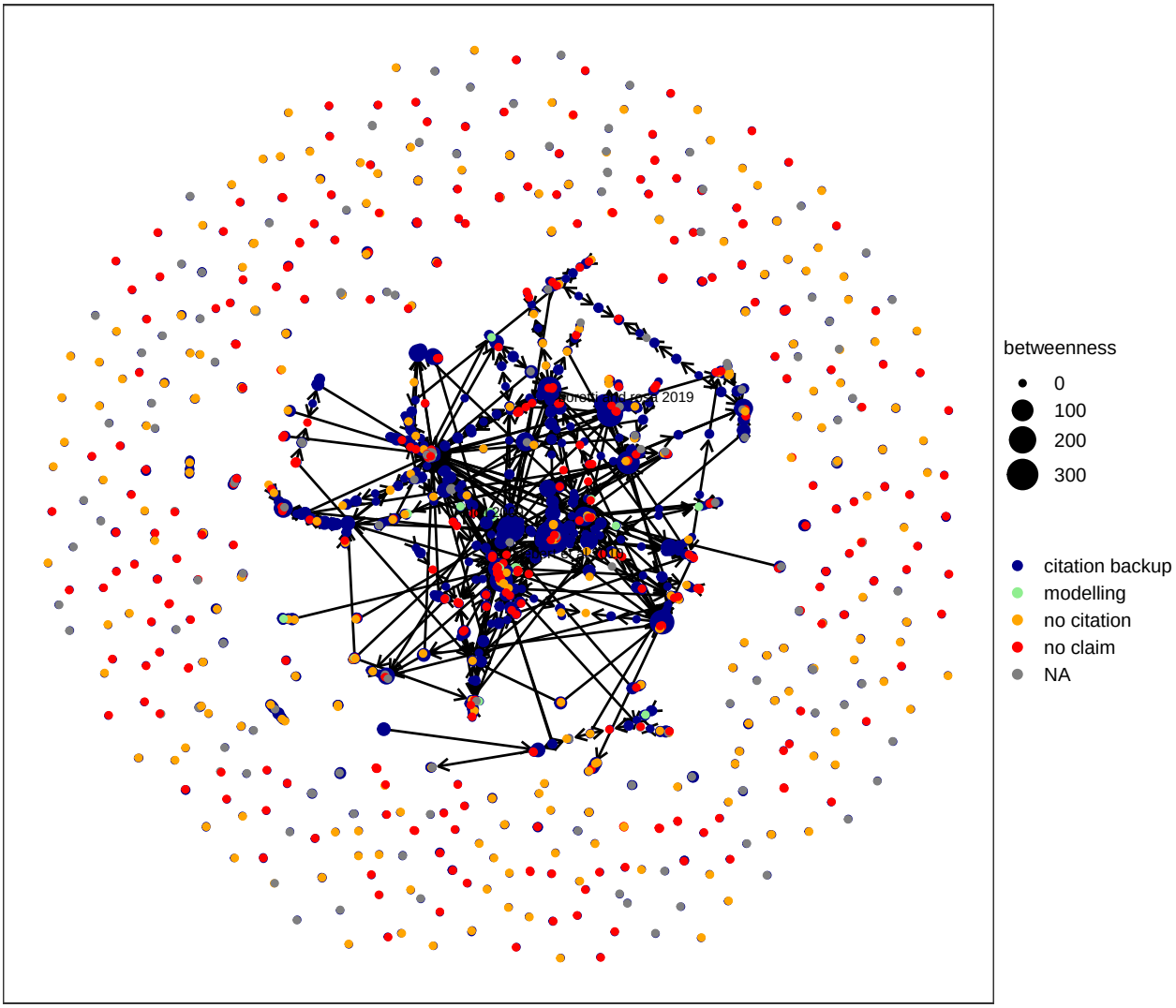
```
## Warning: Removed 768 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).
```




```
##
## $water
```

```
## Warning: Removed 2922 rows containing missing values or values outside the scale range
```

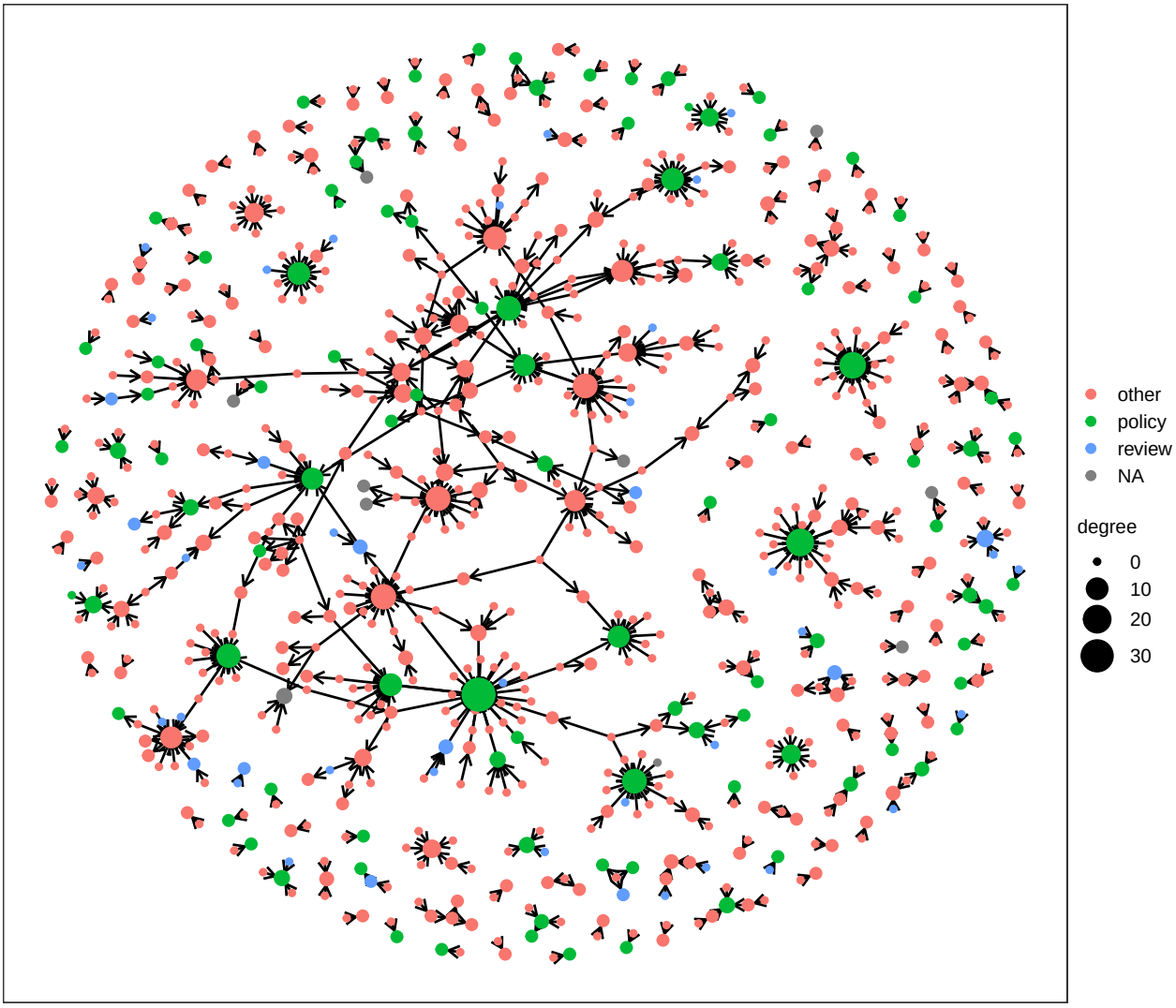
```
## (`geom_text_repel()`).
```



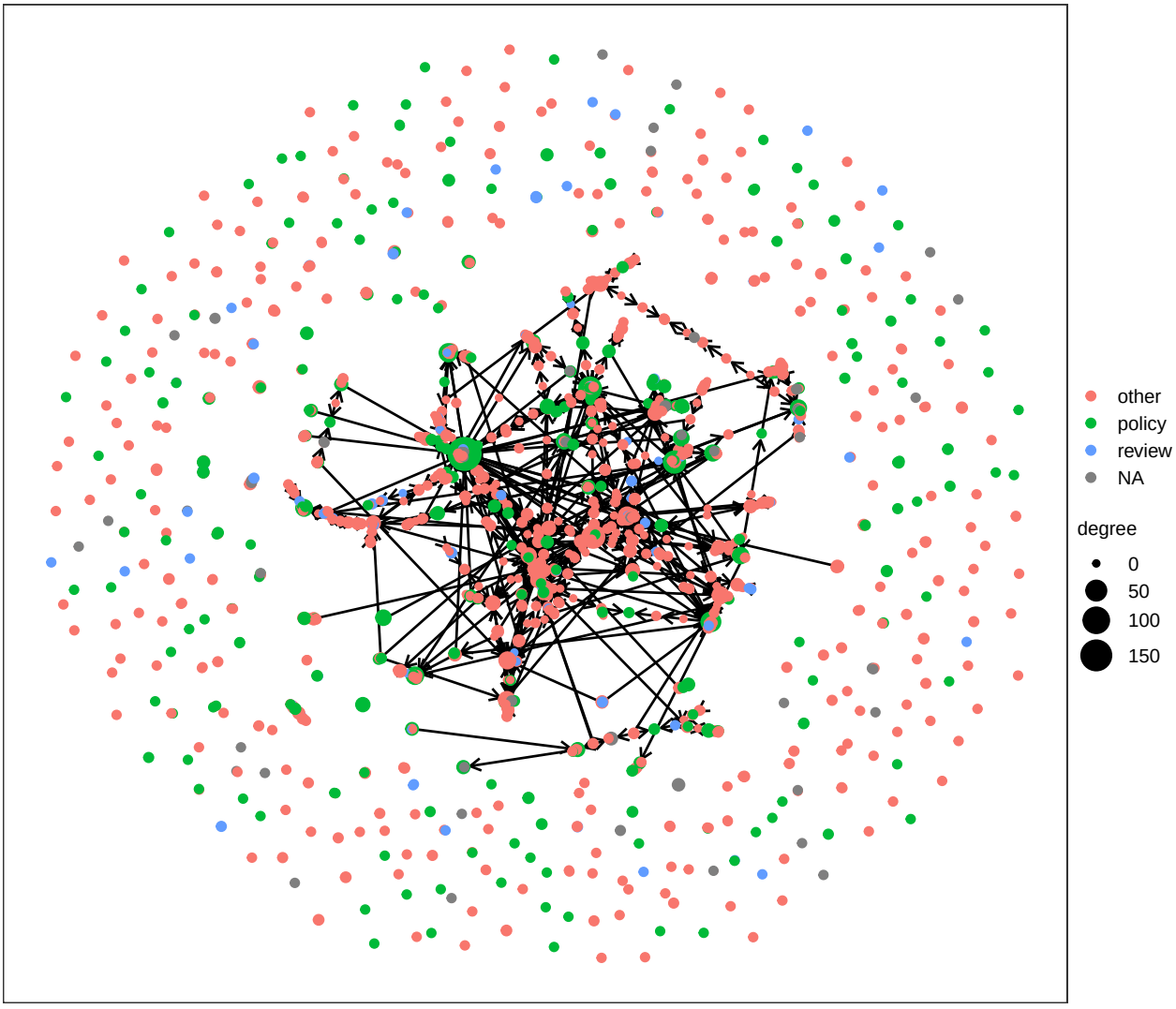

```
# by document.type-------------------------------------------------------------

for (i in names(graph.final)) {

  set.seed(seed)

  p3[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = document.type, size = degree)) +
    labs(x = "", y = "") +
    scale_color_discrete(name = "") +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
```

```
          axis.ticks.y = element_blank(),
          legend.position = "right")
}

p3
```

```
## $food
```



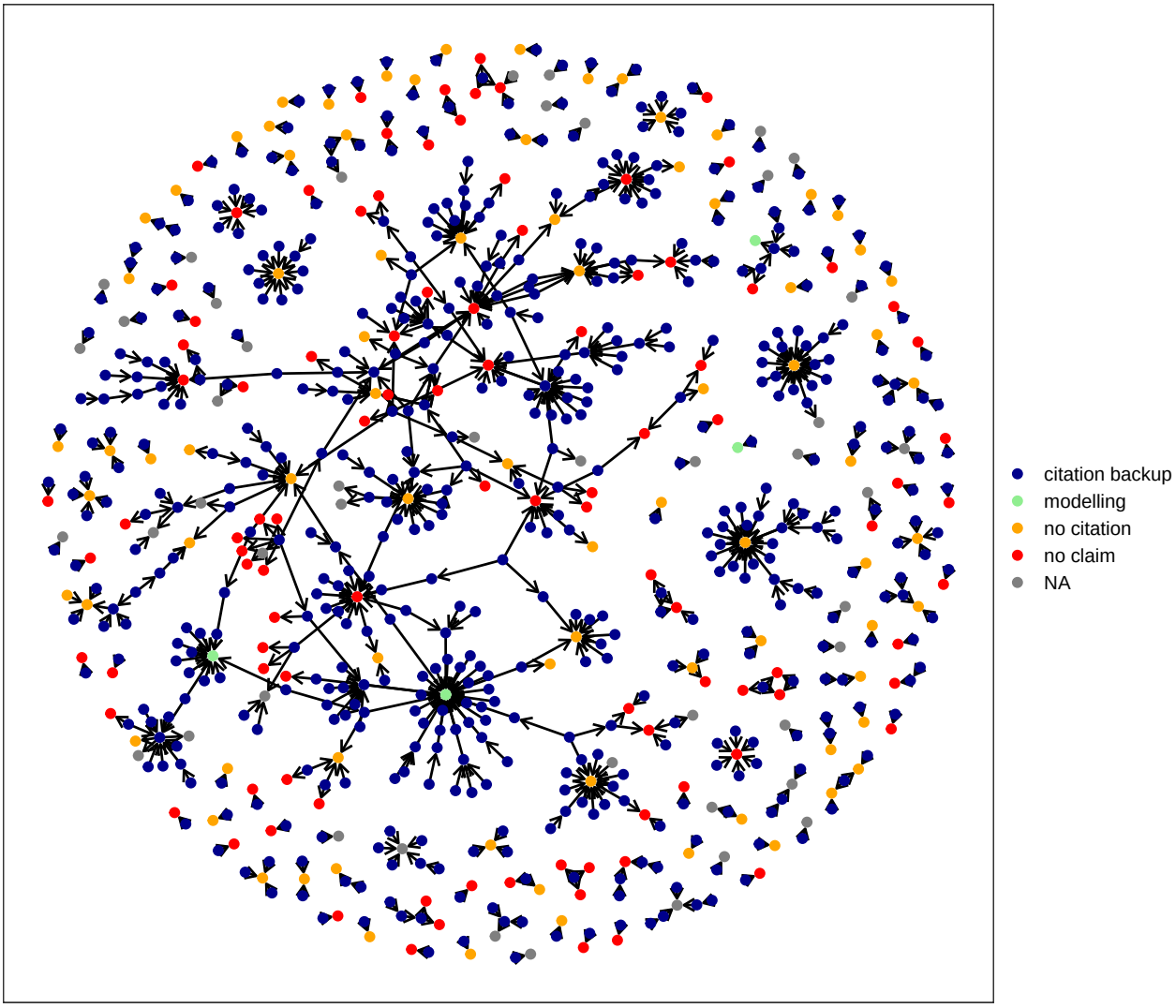

```
##
## $water
```



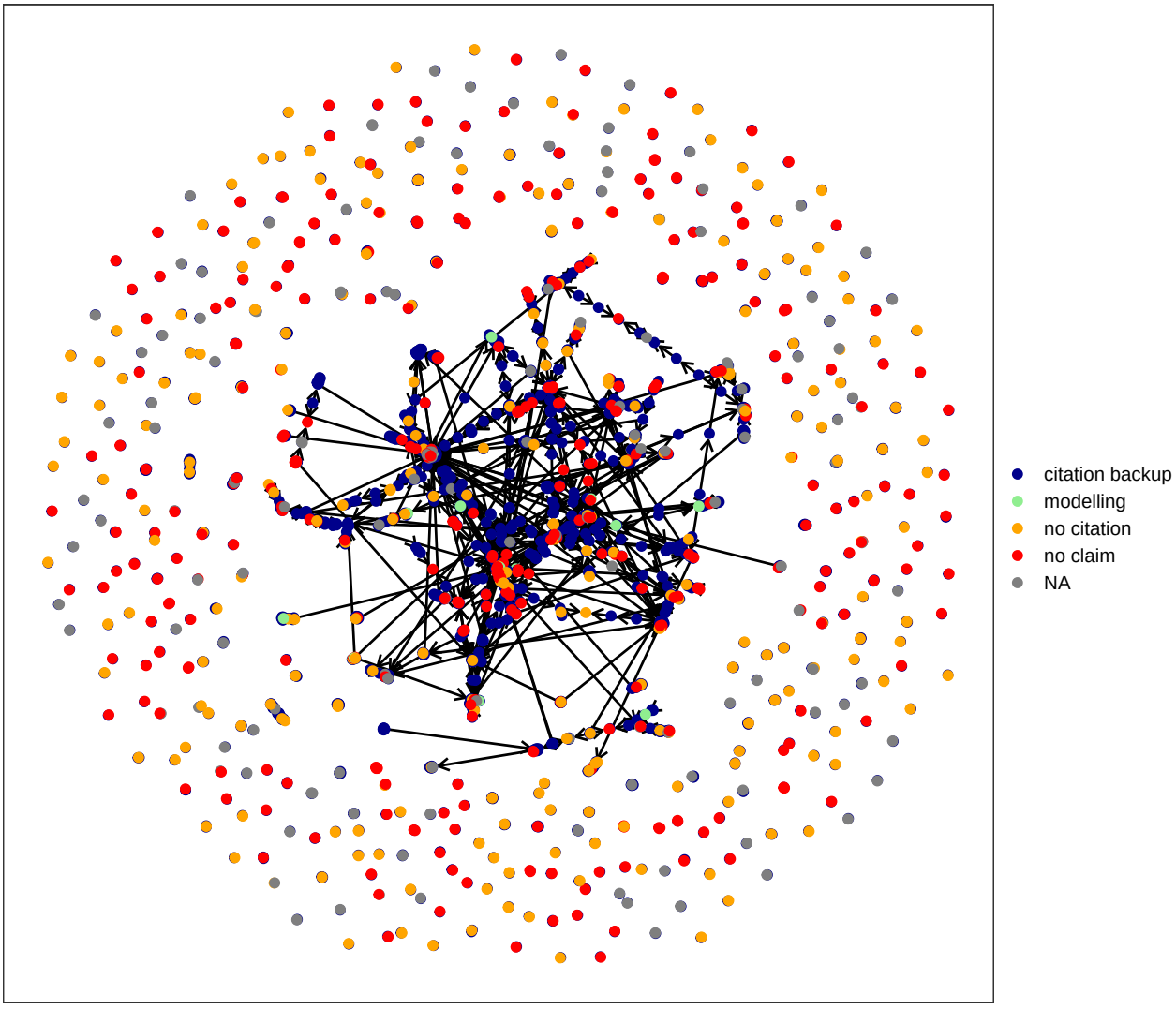

```
# Label nodes that are modelling exercises -----------------------------------

for (i in names(graph.final)) {

  set.seed(seed)

  p4[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = nature.claim)) +
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                       values = selected_colors) +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
```

```
          axis.ticks.y = element_blank(),
          legend.position = "right")
}

p4
```

```
## $food
```




```
##
## $water
```




```
# PLOT EVOLUTION NATURE CLAIM THROUGH TIME ######################################

out <- list()

for (i in names(graph.final)) {

    selected_colors <- c("darkblue", "lightgreen", "orange", "red", "grey")

  out[[i]] <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    data.table() %>%
    .[, .N, .(year, nature.claim)] %>%
    ggplot(., aes(year, N, fill = nature.claim)) +
    scale_fill_manual(values = selected_colors, name = "") +
    geom_area() +
    scale_x_continuous(breaks = breaks_pretty(n = 3)) +
```

```
    theme_AP() +
    ggtitle(names(graph.final[i])) +
    theme(legend.position = "none",
          plot.title = element_text(size = 8))

}

legend <- get_legend(out[[2]] + theme(legend.position = "right"))
```

```
## Warning: Removed 4 rows containing non-finite outside the scale range
## (`stat_align()`).
```

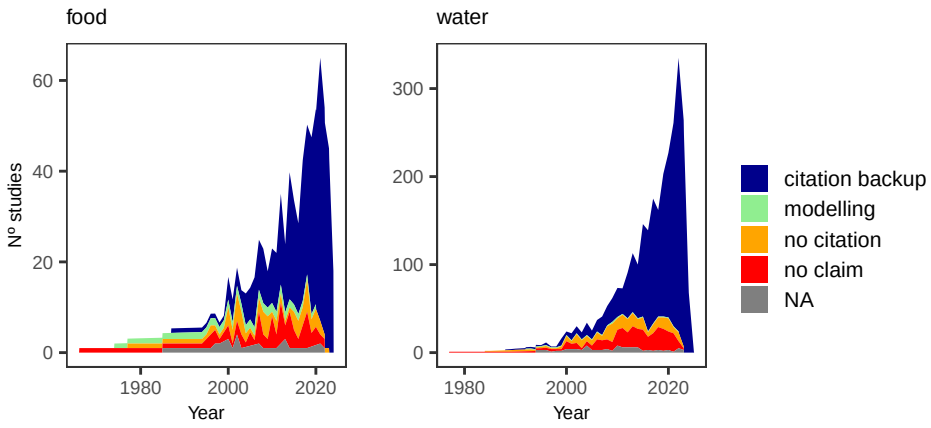
```
## Warning in get_plot_component(plot, "guide-box"): Multiple components found;
## returning the first one. To return all, use `return_all = TRUE`.
```

```
bottom <- plot_grid(out[[1]] + labs(x = "Year", y = "Nº studies"),
                    out[[2]] + labs(x = "Year", y = ""))
```

```
## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_align()`).
```

```
## Warning: Removed 4 rows containing non-finite outside the scale range
## (`stat_align()`).
```

```
plot.network.claims <- plot_grid(bottom, legend, rel_widths = c(0.75, 0.25),
                                 ncol = 2)
plot.network.claims
```



1.4 Uncertainties turned into facts

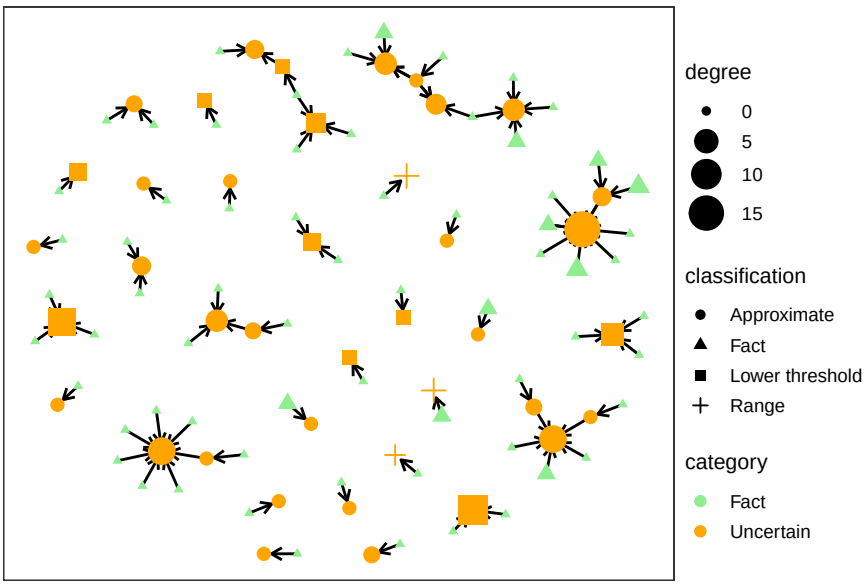
```
# COUNT PROPORTION OF NODES THAT STATE AS FACT A CLAIM UTTERED AS UNCERTAIN ####

uncertainty_plot_fun <- function(graph) {

# Extract name of all studies ----------------------------------------------

all.names <- graph %>%
```

```
    activate(nodes) %>%
    pull(name)

# Extract name of studies stating claim as fact -------------------------------

  f.names <- graph %>%
    activate(nodes) %>%
    data.frame() %>%
    filter(classification == "F") %>%
    pull(name)

  # Add names to edges ------------------------------------------------------

  add.names.edges <- graph %>%
    activate(edges) %>%
    mutate(from.name = all.names[from],
           to.name = all.names[to])

  # Calculate, for each study stating claim as fact, the studies it cites --------
  out.classes <- lapply(f.names, function(x) {

    out_nodes <- add.names.edges %>%
      activate(edges) %>%
      filter(from.name == x) %>%
      pull(to.name)

  })

  # unlist names of studies cited by studies uttering claim as fact --------------

  di <- sort(unlist(out.classes))

  # Extract only those that do not state claim as fact ----------------------------

  nodes.no.fact <- graph %>%
    activate(nodes) %>%
    data.frame() %>%
    data.table() %>%
    .[name %in% di] %>%
    .[!classification == "F"] %>%
    .$name

  name.edges <- add.names.edges  %>%
    activate(edges) %>%
    data.frame() %>%
    filter(from.name %in% f.names & to.name %in% nodes.no.fact) %>%
    .[, c("from.name", "to.name")] %>%
```

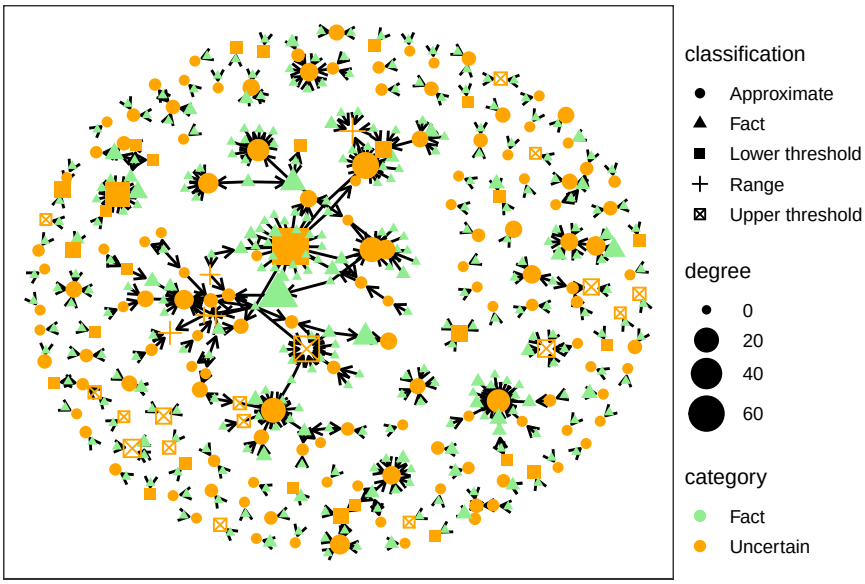
```
    c() %>%
    unlist() %>%
    unique(.)

  output <- add.names.edges  %>%
    activate(nodes) %>%
    filter(name %in% name.edges) %>%
    activate(edges) %>%
    filter(from.name %in% name.edges & to.name %in% name.edges)

  return(output)

}
```

```
# PLOT GRAPH UNCERTAINTIES TURNED INTO FACTS ###################################

out <- lapply(graph.final, function(x) uncertainty_plot_fun(x))

p7 <- list()

for (i in names(out)) {

  set.seed(seed)

  p7[[i]] <- ggraph(out[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1.6, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = category, size = degree, shape = classification)) +
    scale_color_manual(values = c("lightgreen", "orange")) +
    scale_shape_discrete(labels = c("Approximate", "Fact", "Lower threshold", "Range", "Upper t
    labs(x = "", y = "") +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          legend.position = "right",
          legend.text = element_text(size = 7.2))
}

p7
```

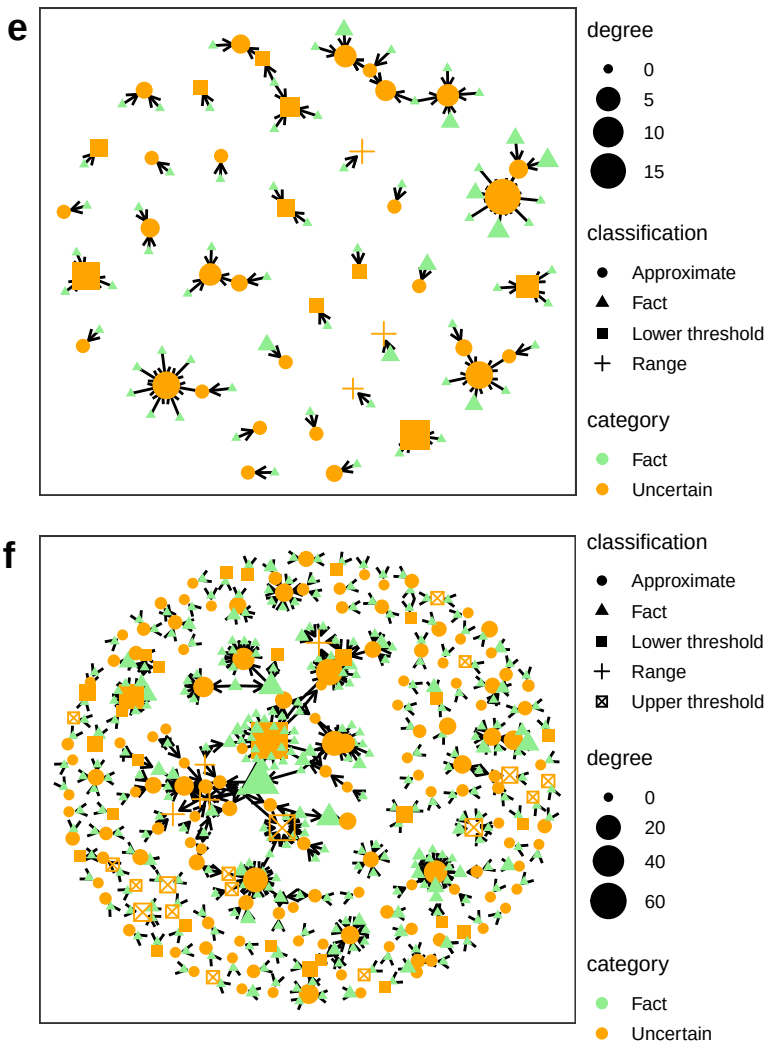
```
## $food
```




```
##
## $water
```




```
# MERGE PLOTS ########################################################################

plot.uncertainty.paths <- plot_grid(p7[[1]], p7[[2]], ncol = 1, labels = c("e", "f"))
plot.uncertainty.paths
```




```
# FUNCTION TO CALCULATE ALL PATHS BETWEEN PAIRS OF NODES ########################

calculate_paths <- function(graph) {

  # Convert to igraph -------------------------------------------------------

  igraph_graph <- as.igraph(graph)

  # Get all unique pairs of nodes -------------------------------------------

  node_pairs <- expand.grid(from = V(igraph_graph), to = V(igraph_graph))
  node_pairs <- node_pairs[node_pairs$from != node_pairs$to, ]

  # Function to calculate all simple paths between a pair of nodes----------

  calculate_paths <- function(from, to) {
    paths <- all_simple_paths(igraph_graph, from = from, to = to)
    lapply(paths, names)
```

```
  }

  # Apply the function to all node pairs and unnest the results-----------------

  all_paths <- node_pairs %>%
    rowwise() %>%
    mutate(paths = list(calculate_paths(from, to))) %>%
    unnest(cols = c(paths))

  out <- sum(sapply(all_paths$paths, function(x) length(x)))

  return(out)

}

# CALCULATE ALL PATHS / PATHS TURNING HYPOTHESIS INTO FACTS ####################

all.paths <- hypothesis.into.facts.paths <- list()

for (i in names(graph.final)) {

  all.paths[[i]] <- calculate_paths(graph.final[[i]])
  hypothesis.into.facts.paths[[i]] <- uncertainty_plot_fun(graph.final[[i]]) %>%
    calculate_paths(.)

}

# Print results: proportion of paths turning uncertainties into facts ------------------------

for (i in names(all.paths)) {
  print(hypothesis.into.facts.paths[[i]] / all.paths[[i]])
}
```

```
## [1] 0.09123344
## [1] 0.08385644
```

2 Proportion of paths ending up in no claim, no citation or modelling nodes

```
# DEFINE FUNCTION ##############################################################

proportion_paths <- function(graph) {

  # Turn into data.frame ------------------------------------------------------

  end_nodes <- graph %>%
```

```
  activate(nodes) %>%
  filter(degree.out == 0) %>%
  data.frame()

end_node_indices <- end_nodes$name

# Loop to store all paths to all end-nodes ---------------------------------
all_paths <- list()

for (v in igraph::V(as.igraph(graph))) {

  paths_from_v <- igraph::all_simple_paths(as.igraph(graph),
                                           from = v,
                                           to = end_node_indices)

  if (length(paths_from_v) > 0) {

    all_paths <- c(all_paths, paths_from_v)
  }
}

# Extract the label of the last node in each path --------------------------

end_labels <- sapply(all_paths, function(path) {

  last_node <- tail(path, 1)
  last_node_name <- V(as.igraph(graph))[last_node]$name

  graph %>%
    activate(nodes) %>%
    filter(name == last_node_name) %>%
    pull(nature.claim)
  })

# Proportion of paths ending in "no citation", "no claim" and "modelling" ----

no_citation_paths <- sum(end_labels == "no citation", na.rm = TRUE)
no_claim_paths <- sum(end_labels == "no claim", na.rm = TRUE)
modelling_paths <- sum(end_labels == "modelling", na.rm = TRUE)
na_paths <- sum(is.na(end_labels))
total_paths <- length(end_labels)

proportion_no_citation <- no_citation_paths / total_paths
proportion_no_claim <- no_claim_paths / total_paths
proportion_modelling <- modelling_paths / total_paths
proportion_na <- na_paths / total_paths
```

```
  # Wrap up for output ------------------------------------------------------

  output <- data.table("no citation" = proportion_no_citation,
                       "no claim" = proportion_no_claim,
                       "modelling" = proportion_modelling,
                       "NA" = proportion_na)
  return(output)

}

# RUN FUNCTION ##################################################################

out <- lapply(graph.final, function(graph) proportion_paths(graph))
out
```

```
## $food
##    no citation no claim modelling        NA
##          <num>    <num>     <num>     <num>
## 1:   0.3914286     0.38 0.1042857 0.1242857
##
## $water
##    no citation  no claim modelling       NA
##          <num>     <num>     <num>    <num>
## 1:   0.3043085 0.2717686 0.3097318 0.114191
```

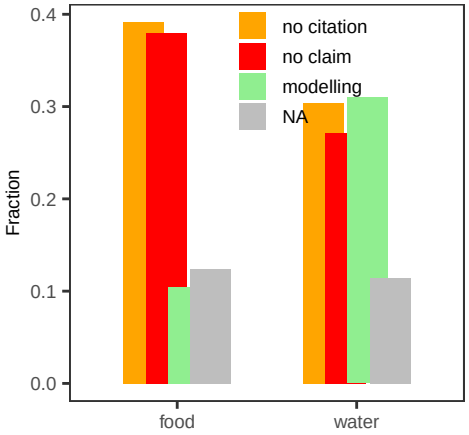
```
# SUM PROPORTION NO CLAIM AND NO CITATION ######################################

lapply(out, function(x) x[, `no citation` + `no claim`])
```

```
## $food
## [1] 0.7714286
##
## $water
## [1] 0.5760771
```

```
# PLOT PROPORTION OF PATHS ENDING IN MODELLING, NO CLAIM AND NO CITATION #######

plot.ending.modelling <- rbindlist(out, idcol = "belief") %>%
  melt(., measure.vars = colnames(.)[-1]) %>%
  ggplot(., aes(belief, value, fill = variable)) +
  geom_bar(stat = "identity",
           position = position_dodge(0.5)) +
  labs(x = "", y = "Fraction") +
  scale_fill_manual(values = c("orange","red", "lightgreen", "grey"),
                    name = "") +
  theme_AP() +
  theme(legend.position = c(0.6, 0.9),
        legend.text = element_text(size = 7))
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
plot.ending.modelling
```



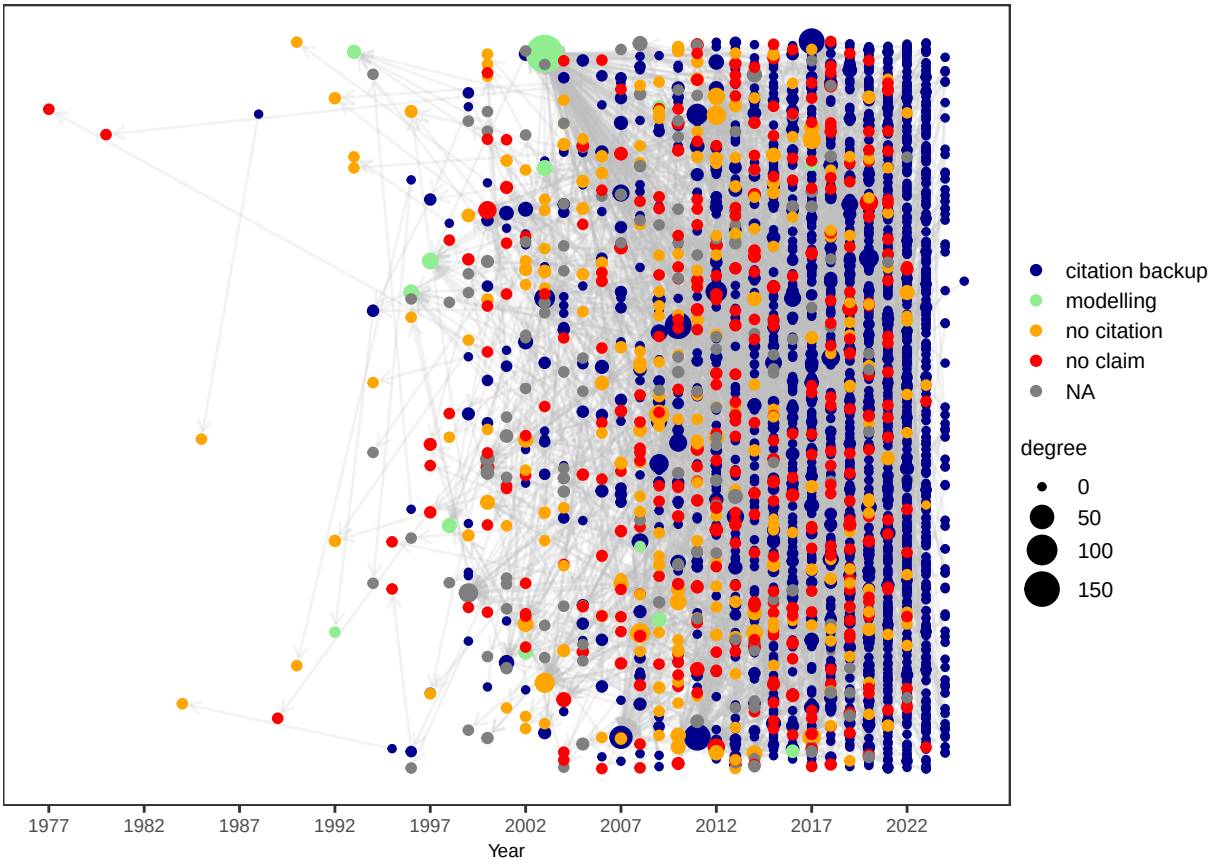
2.1 Network through time

```
# PLOT NETWORK THROUGH TIME ###################################

plot.years <- list()

for (i in c("water", "food")) {

  # Extract vector with names ------------------------------------------------

  location_aquastat <- graph.final[[i]] %>%
  activate(nodes) %>%
  data.frame() %>%
  pull(name) %>%
  grep("aquastat", .)

    location_faostat <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    pull(name) %>%
    grep("faostat", .)

  # Extract vector with years ------------------------------------------------

  v_years <- graph.final[[i]] %>%
  activate(nodes) %>%
```

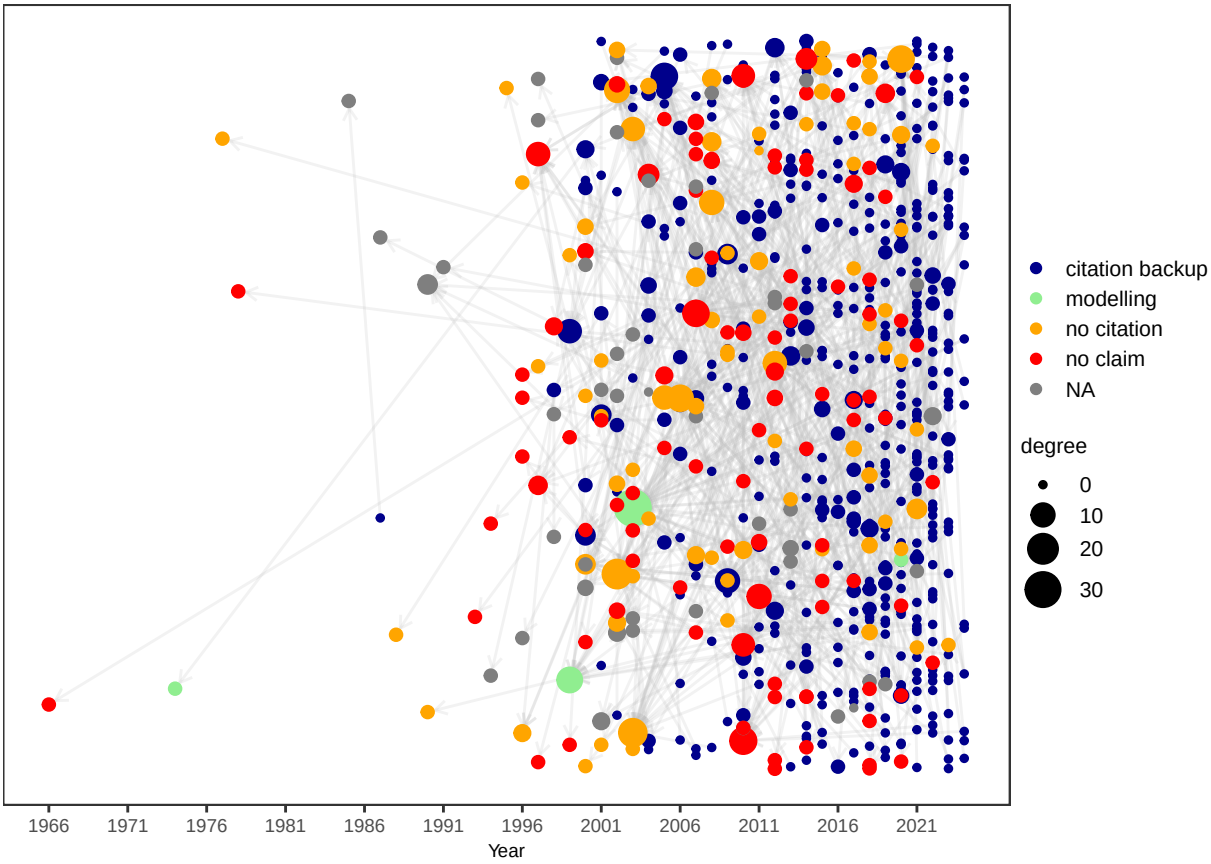
```
  data.frame() %>%
  pull(year)

  # Substitute fao aquastat/faostat without year with the oldest citations -----

  v_years[location_aquastat] <- oldest.aquastat.cite
  v_years[location_faostat] <- oldest.faostat.cite

# Find NA values ------------------------------------------------------------

  na_indices <- is.na(v_years)
  sum(na_indices)

  # Generate random values to replace NA ----------------------------------------

  random_values <- sample(2000:2020, sum(na_indices), replace = TRUE)

  # Replace NA with random values -----------------------------------------------

  v_years[na_indices] <- random_values

  # Define the coordinates------------------------------------------------------

  y_positions <- runif(length(v_years), min = -3, max = 3)  # Random y-axis position
  layout <- cbind(v_years, y_positions)  # Use actual years for x-axi
  layout_matrix <- as.matrix(layout)
  colnames(layout_matrix) <- c("x", "y")

  # PLOT NETWORK THROUGH TIME ###################################################

  # Set seed --------------------------------------------------------------------

  set.seed(seed)

  # Plot ------------------------------------------------------------------------

plot.years[[i]] <- ggraph(graph.final[[i]], layout = layout_matrix, algorithm = "nicely") +
  geom_edge_link(arrow = arrow(length = unit(1.8, "mm")),
                 end_cap = circle(1, "mm"),
                 color = "grey",
                 alpha = 0.2) +
  geom_node_point(aes(color = nature.claim, size = degree)) +
  scale_color_manual(name = "",
                     values = selected_colors) +
  scale_x_continuous(name = "Year",
                     limits = range(v_years),
                     breaks = seq(min(v_years),
```

```
                                  max(v_years), by = 5)) +
  labs(x = "Year", y = "") +
  theme_AP() +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank())

}

plot.years
```

```
## $water
```




```
##
## $food
```

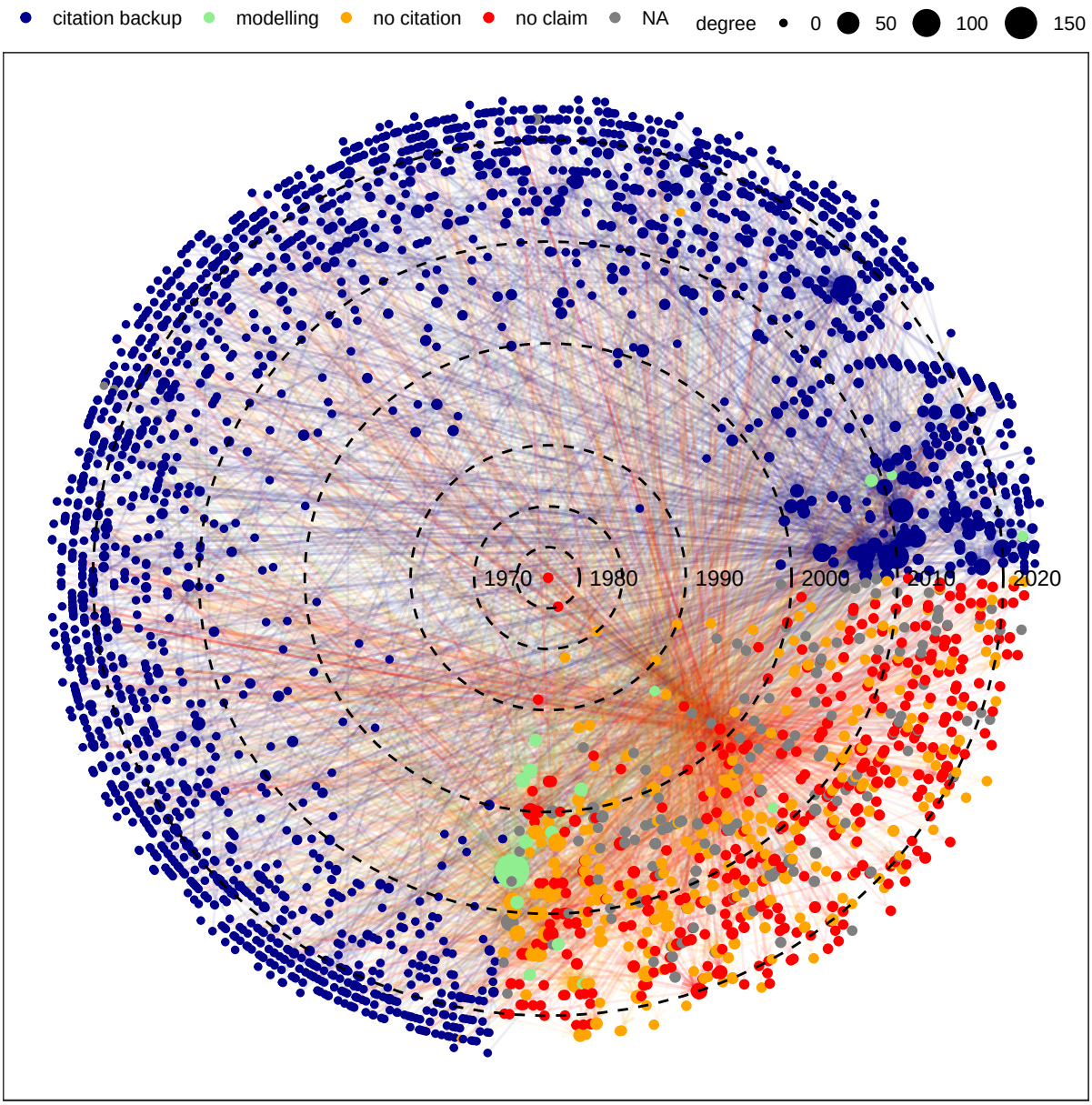



```
# ANOTHER VISUALIZATION FOR YEARS BASED ON POLAR COORDINATES ####################

plot.years <- list()

for (i in c("water", "food")) {

  # Extract vector with names ---------------------------------------------------

  location_aquastat <- graph.final[[i]] %>%
  activate(nodes) %>%
  data.frame() %>%
  pull(name) %>%
  grep("aquastat", .)

    location_faostat <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    pull(name) %>%
    grep("faostat", .)

  # Extract vector with years ---------------------------------------------------

  v_years <- graph.final[[i]] %>%
```

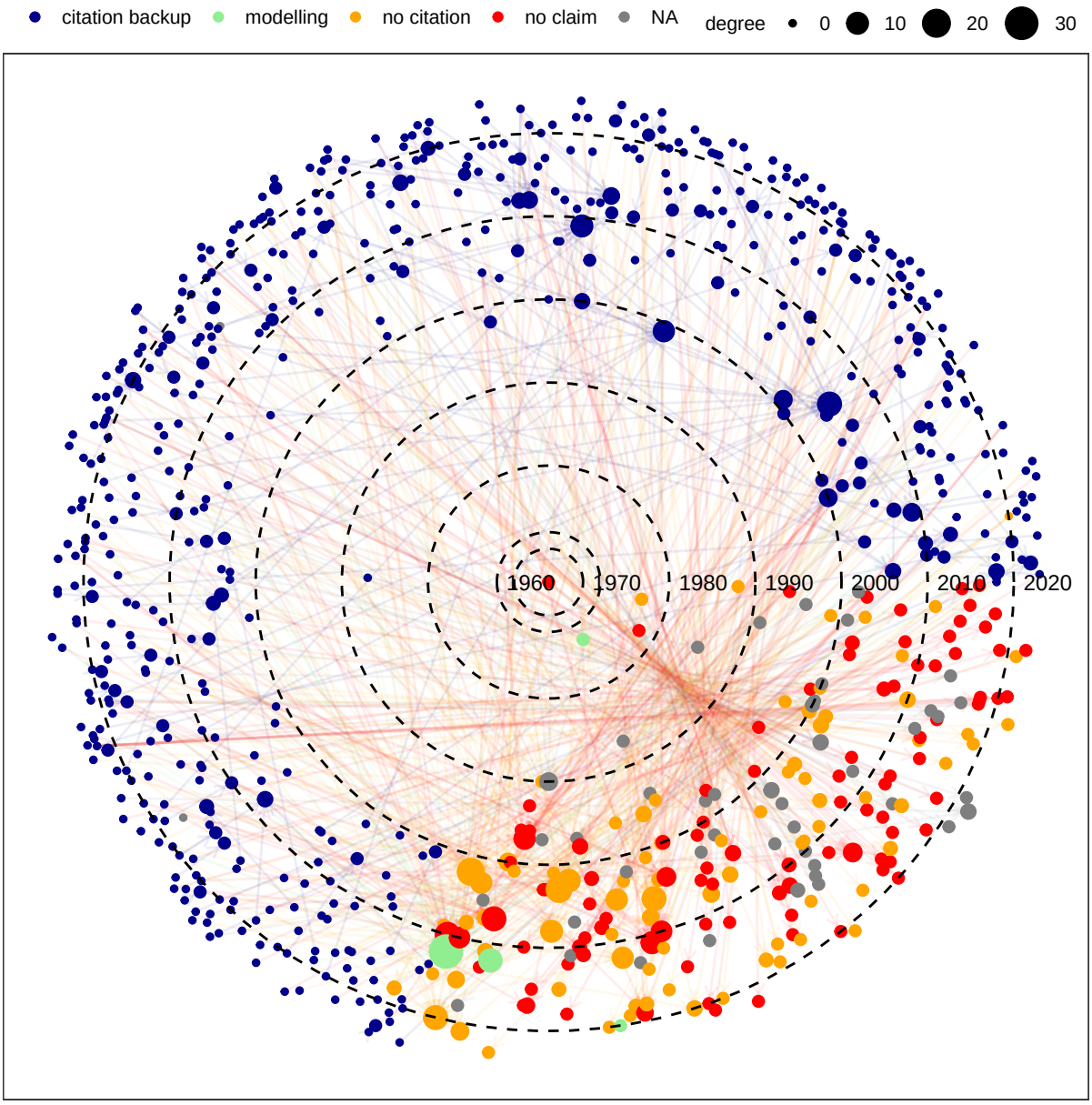
```r
activate(nodes) %>%
data.frame() %>%
pull(year)

# Substitute fao aquastat/faostat without year with the oldest citations -----

v_years[location_aquastat] <- oldest.aquastat.cite
v_years[location_faostat] <- oldest.faostat.cite


# Replace NA values in year with random samples from 2000 to 2020 ------------

g <- graph.final[[i]] %>%
  activate(nodes) %>%
  mutate(year = ifelse(is.na(year), sample(2000:2020, replace = TRUE), year)) %>%
  mutate(
    year_normalized = (year - min(year)) / (2024 - min(year)),  # Normalize relative to 2024
    radius = year_normalized
  )

# Assign the calculated positions ----------------------------------------------

g <- g %>%
  mutate(
    angle = seq(0, 2 * pi, length.out = n() + 1)[1:n()],
    x = radius * cos(angle),
    y = radius * sin(angle)
  )

# Determine the range of years -------------------------------------------------
min_year <- min(g %>% pull(year))

# Dynamically determine the start year for the concentric circles -----------
start_year <- floor(min_year / 10) * 10  # Round down to the nearest decade
end_year <- 2024  # Explicitly set the end year to 2024
year_intervals <- seq(start_year, end_year, by = 10)

# Create concentric circles every ten years ------------------------------------
circle_data <- lapply(year_intervals, function(yr) {
  r <- (yr - min_year) / (end_year - min_year)
  tibble(
    x = r * cos(seq(0, 2 * pi, length.out = 100)),
    y = r * sin(seq(0, 2 * pi, length.out = 100)),
    year = yr,
    label_x = r,  # Label position on the x-axis (angle = 0)
    label_y = 0   # Label position on the y-axis (angle = 0)
  )
```

```
  }) %>% bind_rows()

  # Remove duplicate labels ------------------------------------------------
  label_data <- circle_data %>%
    distinct(year, .keep_all = TRUE)  # Keep only unique year labels

  # Plot -------------------------------------------------------------------

  # Set seed ------------------------------

  set.seed(seed)

  plot.years[[i]] <- ggraph(g, layout = "manual", x = x, y = y) +
    # Add concentric circles
    geom_edge_link(arrow = arrow(length = unit(1.8, "mm")),
                   end_cap = circle(1, "mm"),
                   alpha = 0.07,
                   aes(color = edge_color)) +
    scale_edge_color_manual(values = selected_colors, guide = "none") +
    geom_node_point(aes(color = nature.claim, size = degree)) +
    scale_color_manual(name = "",
                       values = selected_colors) +
    geom_path(data = circle_data, aes(x = x, y = y, group = factor(year)),
              color = "black", linetype = "dashed") +
    # Add year labels
    geom_text(data = label_data, aes(x = label_x, y = label_y, label = year),
              hjust = -0.2, vjust = 0.5, size = 3) +
    labs(x = "", y = "") +
    theme_AP() +
    theme(axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          legend.position = "top")
}

plot.years
```

```
## $water
```


citation backup
modelling
no citation
no claim
NA
degree
0
50
100
150
1970
1980
1990
2000
2010
2020

```
##
## $food
```




```
# FUNCTION TO PLOT EVOLUTION OF NETWORK THROUGH TIME ############################

network_through_time_fun <- function(graph, Year, seed) {

  # Extract all names ---------------------------------------------------------

  all.names <- graph %>%
    activate(nodes) %>%
    pull(name)

  # Add names to edges --------------------------------------------------------

  add.names.edges <- graph %>%
    activate(edges) %>%
```

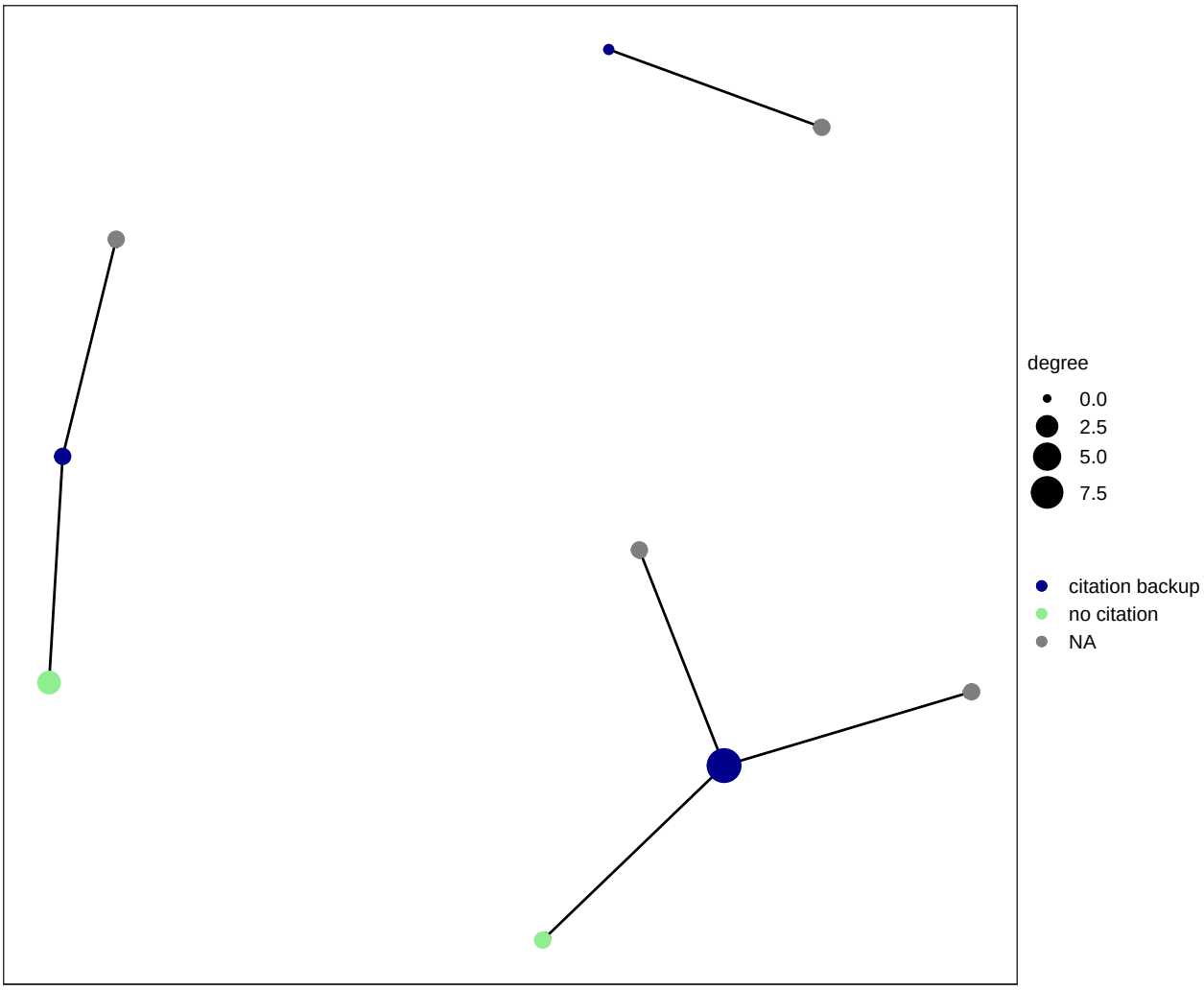
```
    mutate(from.name = all.names[from],
           to.name = all.names[to])

  # Extract nodes by year ----------------------------------------------------

  names.targeted <- add.names.edges %>%
    activate(edges) %>%
    filter(year < Year) %>%
    data.frame() %>%
    .[, c("from.name", "to.name")] %>%
    c() %>%
    unlist() %>%
    unique(.)

  name.nodes <- add.names.edges %>%
    activate(nodes) %>%
    filter(name %in% names.targeted) %>%
    activate(edges) %>%
    filter(from.name %in% names.targeted & to.name %in% names.targeted)

  set.seed(seed)

  # Plot ---------------------------------------------------------------------

  out <- ggraph(name.nodes, layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1, 'mm')),
                   end_cap = circle(0.3, "mm")) +
    geom_node_point(aes(color = nature.claim, size = degree)) +
    scale_color_manual(name = "",
                       values = selected_colors) +
    labs(x = "", y = "") +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          legend.position = "none")

  return(out)

}

# DEFINE YEARS OF INTEREST ###################################################

years.vector <- c(seq(2000, 2020, 10), 2024)

# RUN FUNCTION ###############################################################
```

```
plots.through.time <- list()

for (i in names(graph.final)) {

  plots.through.time[[i]] <- lapply(years.vector, function(year)
    network_through_time_fun(graph = graph.final[[i]], Year = year, seed = seed) +
      ggtitle(year))
}
```

```
da <- list()

for (i in names(plots.through.time)) {

  for (j in 1:length(plots.through.time[[i]])) {

    da[[i]][[j]] <- plots.through.time[[i]][[j]] +
      geom_node_point(aes(color = nature.claim)) +
      theme(axis.text.x = element_blank(),
            axis.ticks.x = element_blank(),
            axis.text.y = element_blank(),
            axis.ticks.y = element_blank(),
            legend.position = "right")
  }
}

da
```

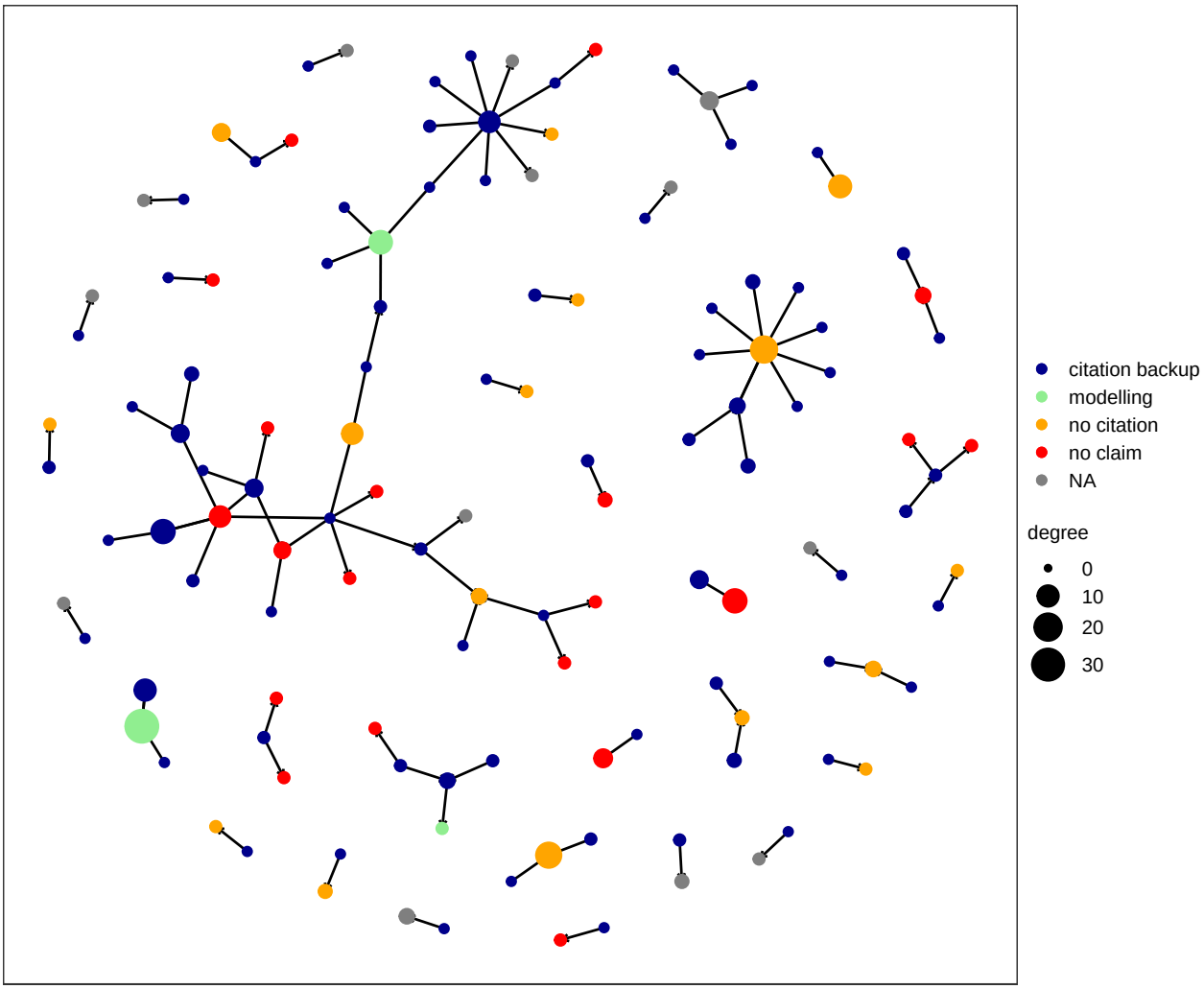
```
## $food
## $food[[1]]
```

2000



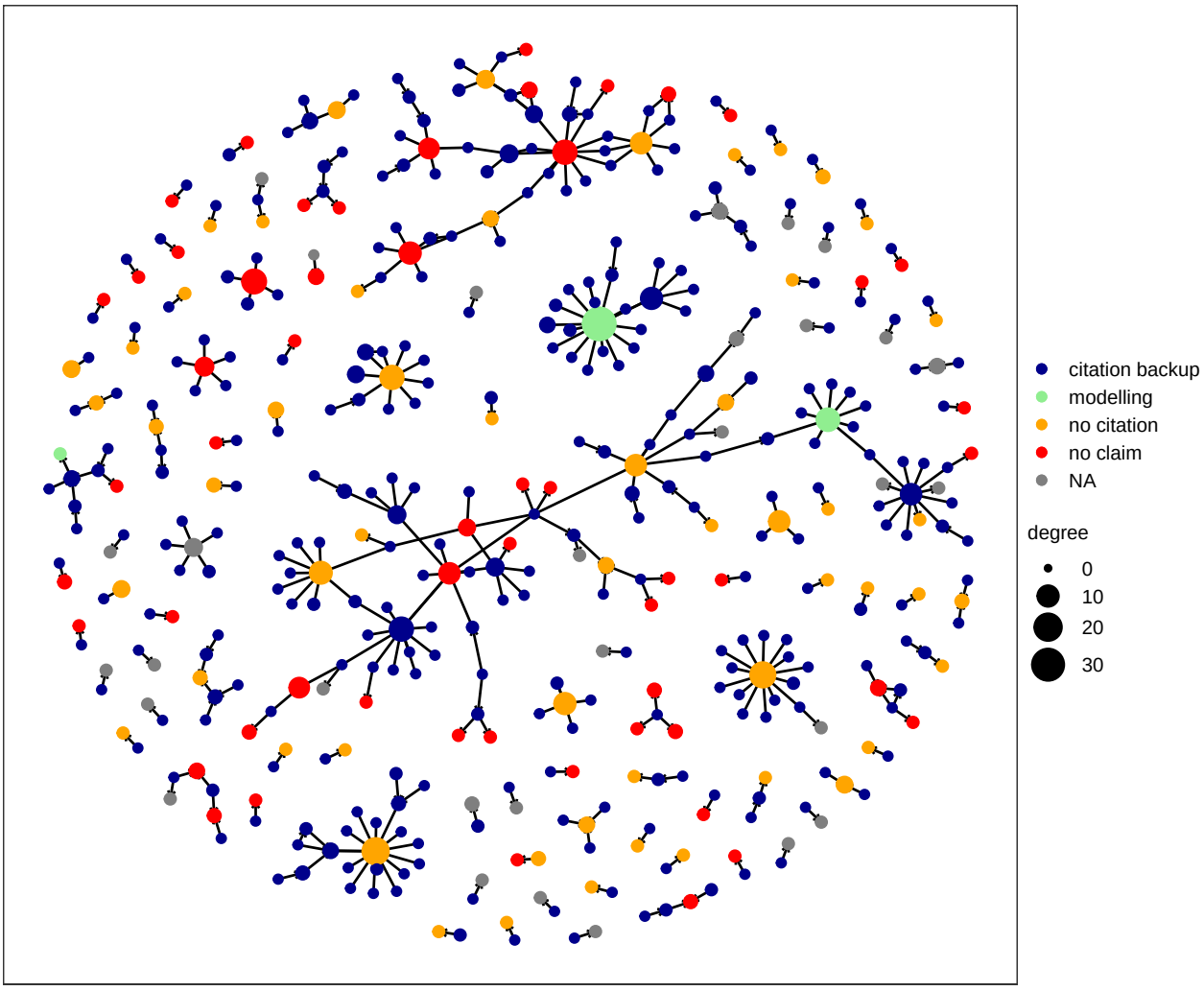

```
##
## $food[[2]]
```

2010


citation backup
modelling
no citation
no claim
NA
degree
0
10
20
30

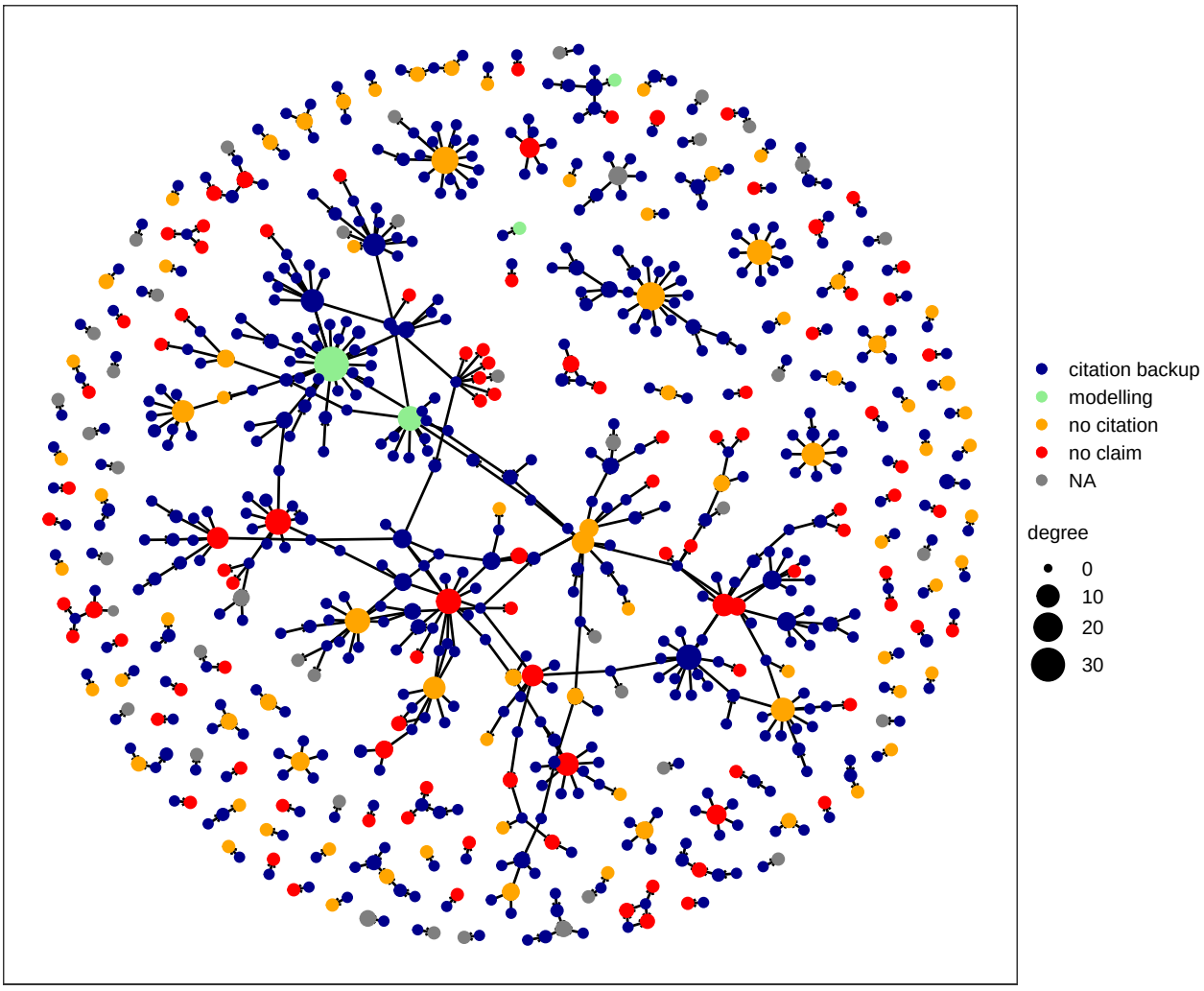
```
##
## $food[[3]]
```

2020


citation backup
modelling
no citation
no claim
NA
degree
0
10
20
30

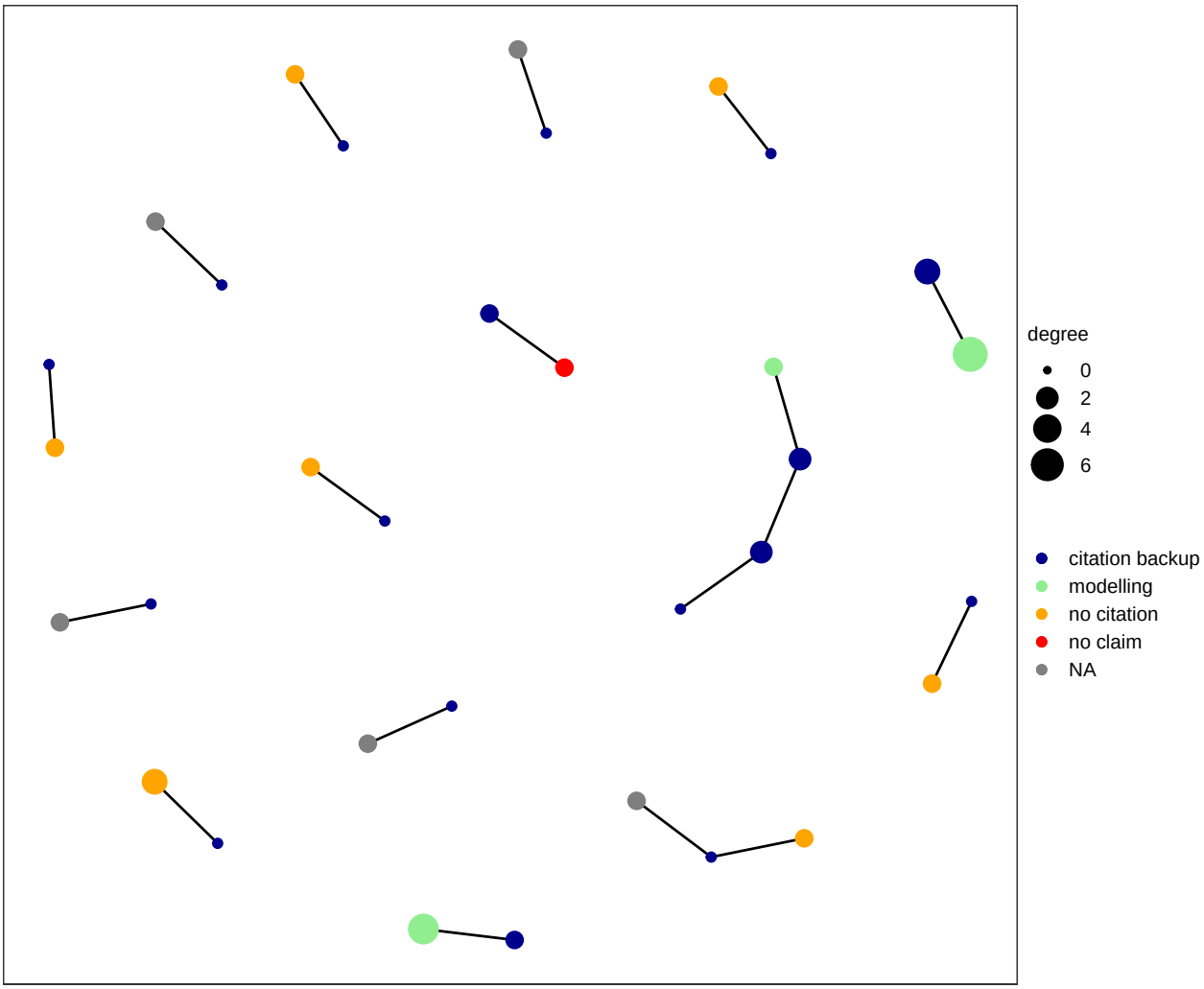
```
##
## $food[[4]]
```

2024



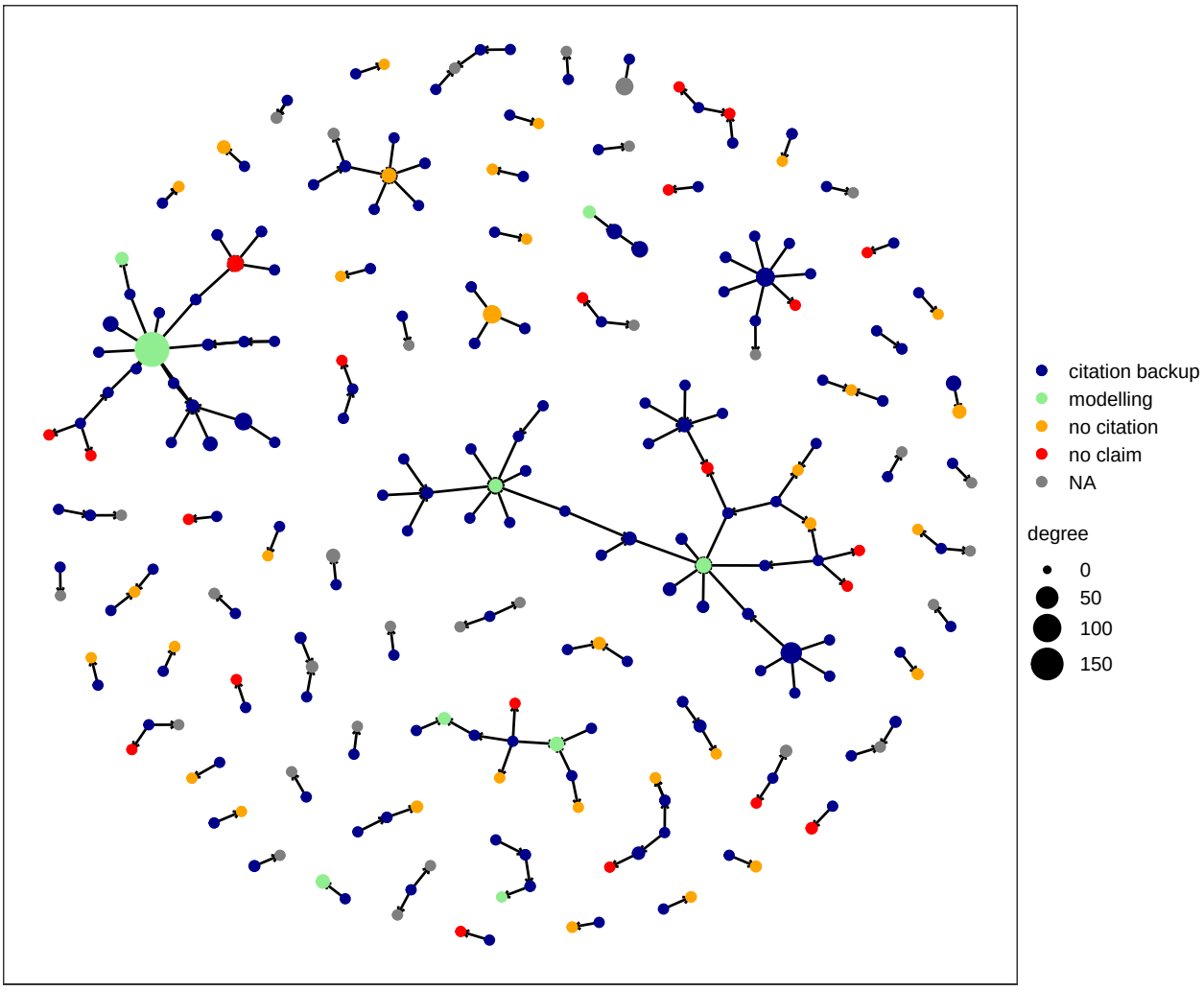

```
##
##
## $water
## $water[[1]]
```

2000


degree
0
2
4
6
citation backup
modelling
no citation
no claim
NA

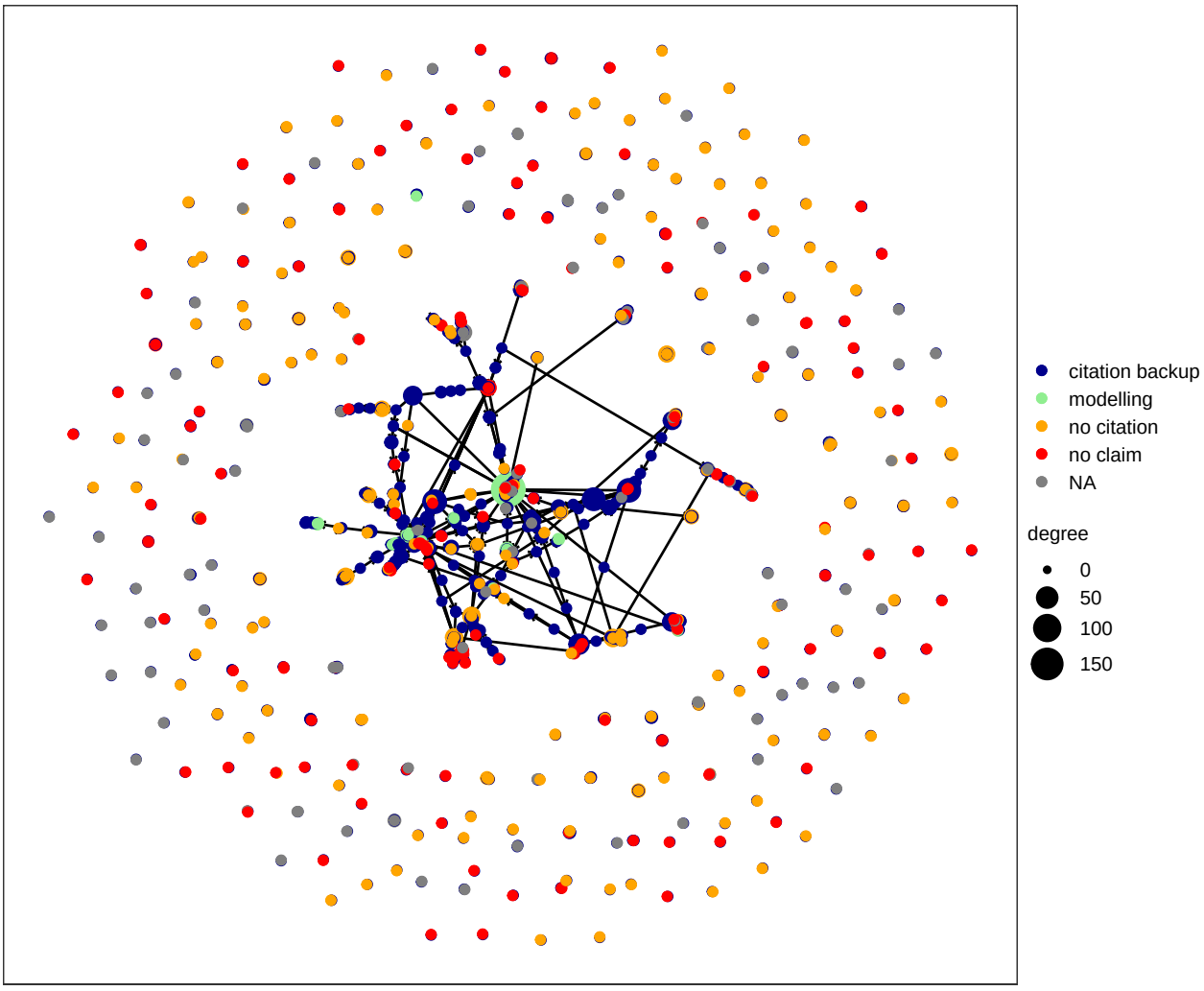
```
##
## $water[[2]]
```

2010


citation backup
modelling
no citation
no claim
NA
degree
0
50
100
150

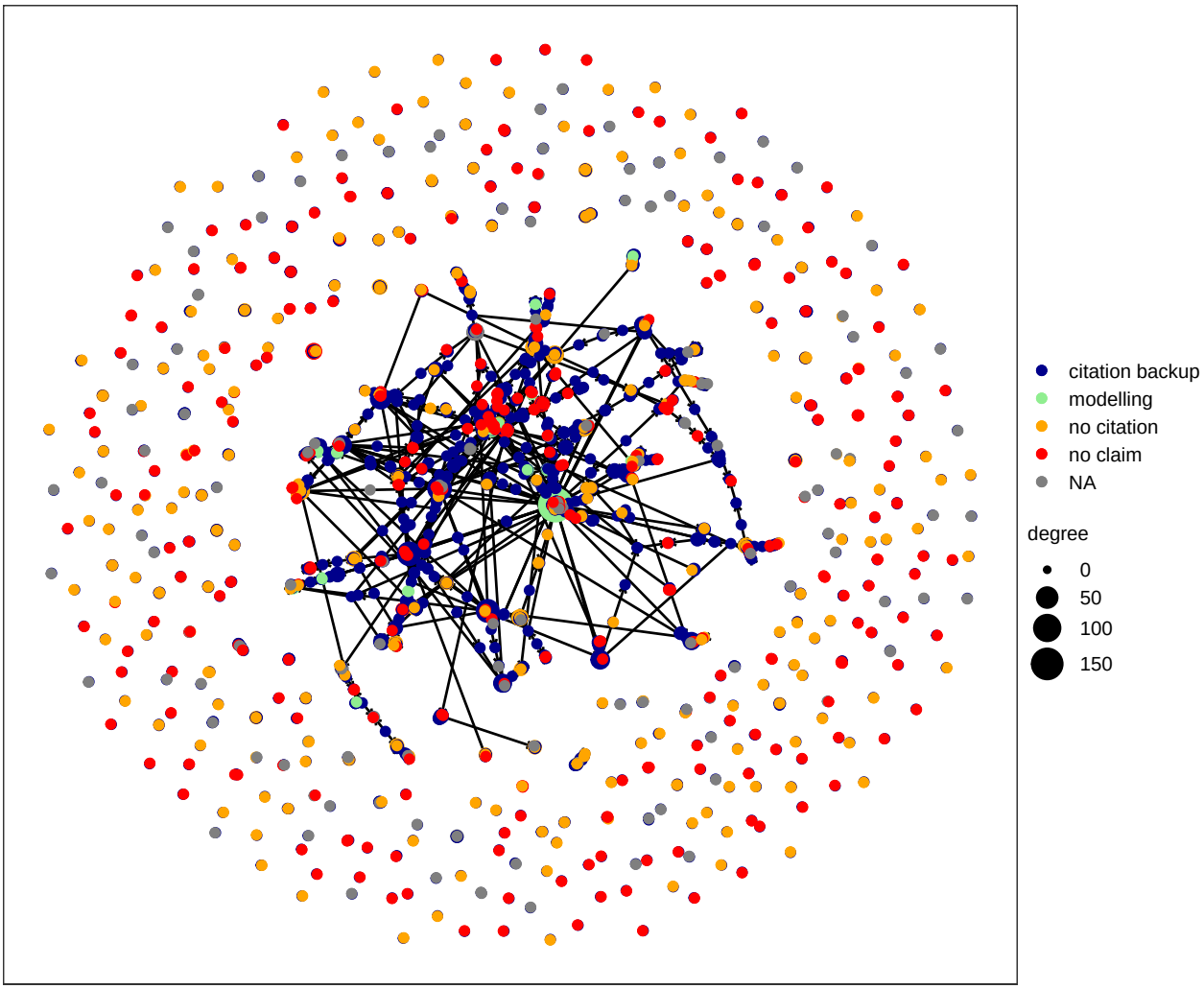
```
##
## $water[[3]]
```

2020


citation backup
modelling
no citation
no claim
NA
degree
0
50
100
150

```
##
## $water[[4]]
```

2024



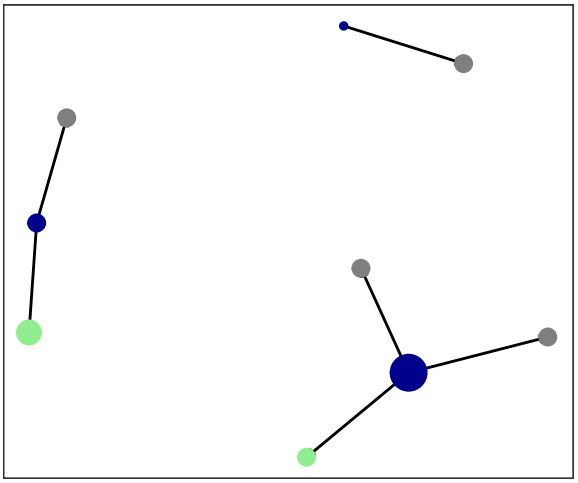

```
# PLOT ########################################################################

# Extract legend ----------------------------------------------------------

legend.plot <- list()

for (i in names(plots.through.time)) {

  legend.plot[[i]] <- get_legend(plots.through.time[[i]][[length(plots.through.time[[i]])]] +
                                   theme(legend.position = "top"))
}
```

```
## Warning in get_plot_component(plot, "guide-box"): Multiple components found;
## returning the first one. To return all, use `return_all = TRUE`.
## Warning in get_plot_component(plot, "guide-box"): Multiple components found;
## returning the first one. To return all, use `return_all = TRUE`.
```

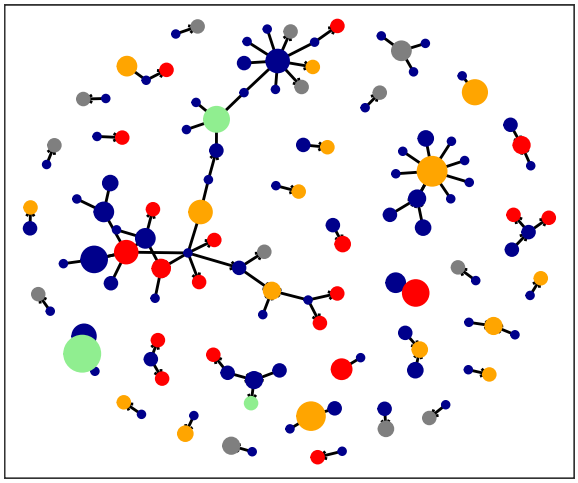
```
# Plot ------------------------------------------------------------------------

bottom <- out.plot <- list()

for (i in names(plots.through.time)) {

  bottom[[i]] <- do.call(plot_grid, c(plots.through.time[[i]],
                                      nrow = floor(length(years.vector) / 2)))
  out.plot[[i]] <- plot_grid(legend.plot[[i]],
                             bottom[[i]], ncol = 1, rel_heights = c(0.1, 0.9))

}

out.plot
```

```
## $food
```

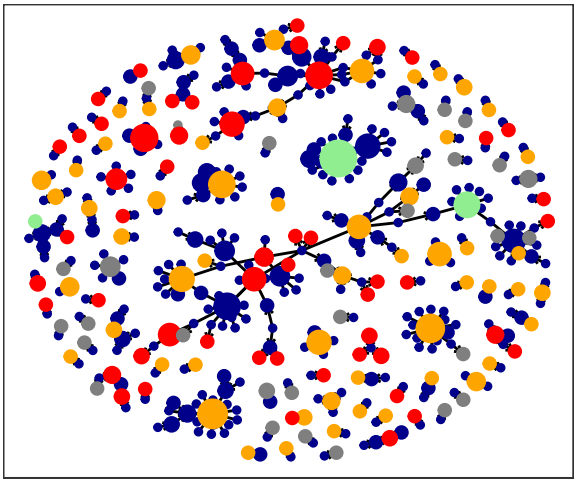
2000



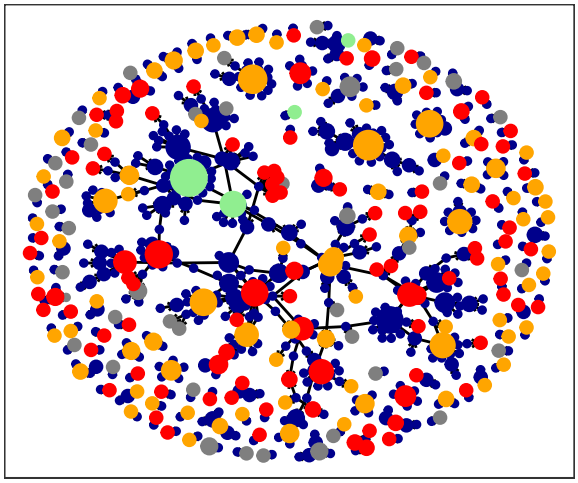
2010



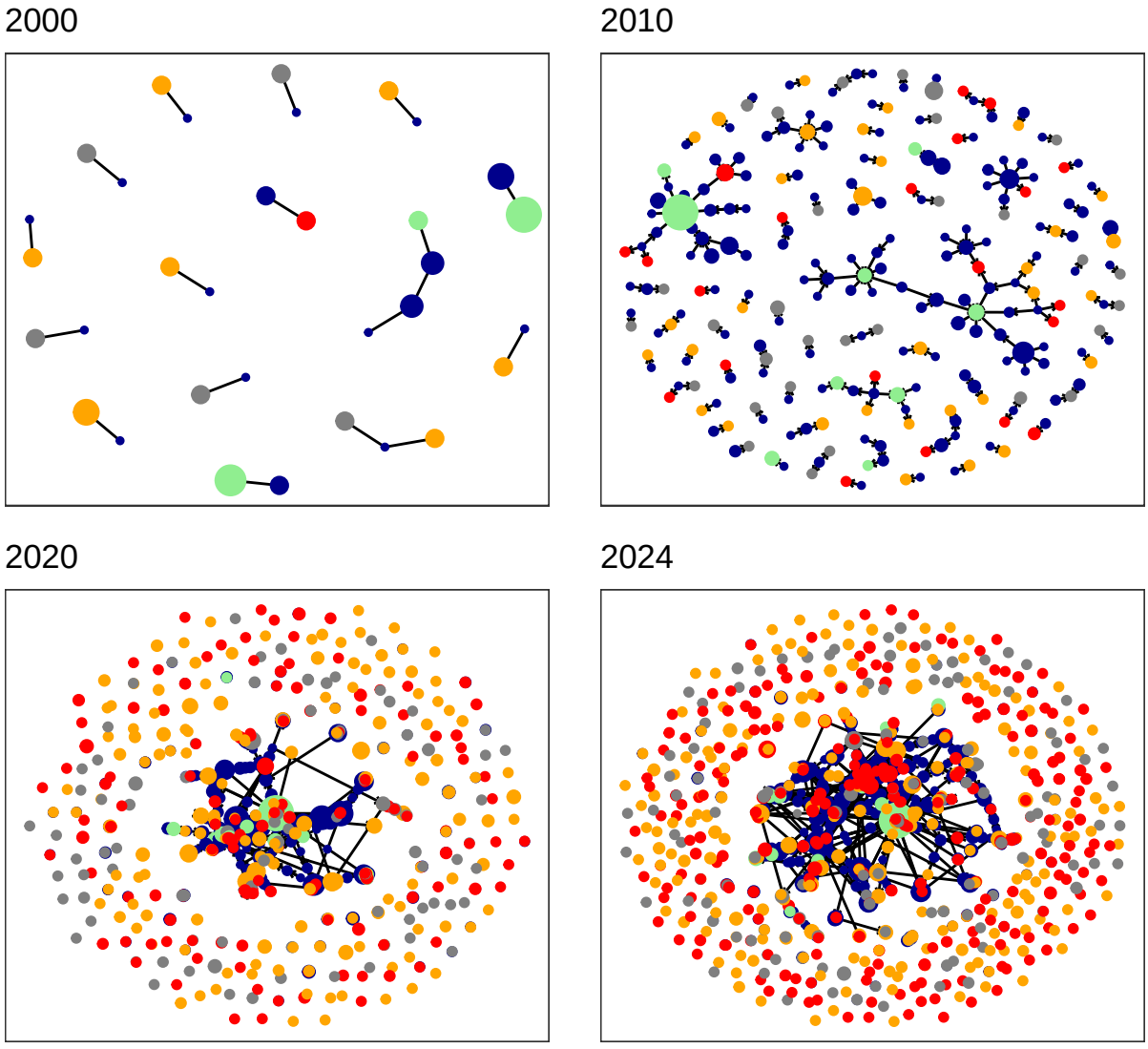
2020



2024



```
##
## $water
```



3 Analysis of paths

3.1 "no claim" or "no citation" paths

```
# COUNT THE NUMBER OF NODES WITH PATHS ULTIMATELY LEADING TO NODES
# THAT DO NOT MAKE THE CITATION ##################################################

# Function: loop through each node that do not make the claim to find all nodes
# connected to it ----------------------------------------------------------------

nodes_to_no_claim_node_fun <- function(g, terminal_nodes) {

  if (!is.igraph(g)) {
    g <- as.igraph(g)
```

```
  }

  all_predecessors <- vector("list", length(terminal_nodes))

  for (i in seq_along(terminal_nodes)) {

    terminal_node <- terminal_nodes[i]
    predecessors <- subcomponent(g, terminal_node, mode = "in")
    all_predecessors[[i]] <- predecessors
  }

  unique_predecessors <- unique(names(unlist(all_predecessors)))

  return(unique_predecessors)

}
```

```
# CALCULATE

# Extract name of all nodes ------------------------------------------------
all_nodes <- lapply(graph.final, function(graph)
  graph %>%
    activate(nodes) %>%
    pull(name))

# Extract name of nodes that do not make the claim -------------------------

no.claim_nodes <- lapply(graph.final, function(graph)
  graph %>%
    activate(nodes) %>%
    filter(degree.out == 0 & nature.claim == "no claim") %>%
    pull(., "name"))

# Extract name of nodes that do not make the claim and those that make
# the claim but do not cite anybody ----------------------------------------

no.claim.and.no.citation.nodes <- lapply(graph.final, function(graph)
  graph %>%
  activate(nodes) %>%
  filter(degree.out == 0 & nature.claim == "no claim" | nature.claim == "no citation" ) %>%
  pull(., "name"))

# Run the function ---------------------------------------------------------

tmp <- list()

for(i in names(graph.final)) {
```

```
  tmp[[i]] <- lapply(list(no.claim_nodes[[i]],
                          no.claim.and.no.citation.nodes[[i]]), function(x)
    sort(nodes_to_no_claim_node_fun(graph.final[[i]], terminal_nodes = x)))
}
```

```
## Warning: `is.igraph()` was deprecated in igraph 2.0.0.
## i Please use `is_igraph()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
for(i in names(graph.final)) {
  names(tmp[[i]]) <- c("path ending in no claim",
                               "path ending in no claim or no citation")
}

# Calculate proportions ---------------------------------------------------

out <- list()

for(i in names(tmp)) {
  out[[i]] <- lapply(tmp[[i]], function(x) length(x) / length(all_nodes[[i]]))
}

out
```

```
## $food
## $food$`path ending in no claim`
## [1] 0.3932292
##
## $food$`path ending in no claim or no citation`
## [1] 0.796875
##
##
## $water
## $water$`path ending in no claim`
## [1] 0.3582906
##
## $water$`path ending in no claim or no citation`
## [1] 0.6789744
```

3.2 Calculation of amplification

```
# CALCULATE AMPLIFICATION FUNCTION ############################################

# Prepare data ------------------------------------------------------------

vec.topics <- c("food", "water")
```

```r
graph <- modelling.papers <- list()

for (i in 1:length(vec.topics)) {

  graph[[i]] <- network.dt.complete[topic == vec.topics[[i]]] %>%
    graph_from_data_frame(.)

  modelling.papers[[i]] <- network.dt.complete[topic == vec.topics[[i]] &
                                                 nature.claim == "modelling"] %>%
    .[, from] %>%
    unique()
}
```

```
## Warning: In `d`, `NA` elements were replaced with string "NA".
## In `d`, `NA` elements were replaced with string "NA".
```

```r
names(graph) <- vec.topics
names(modelling.papers) <- vec.topics

# DEFINE AMPLIFICATION FUNCTIONS ##################################################

# Citation amplification function ----------------------------------------

citation_amplification_index <- function(graph, source_paper, modelling_papers) {

  # Get all simple paths from paper P ----------------------------------------

  all_paths <- all_simple_paths(graph, from = source_paper, mode = "out")

  # Filter out paths of length 1 that lead to modelling papers ----------------

  filtered_paths <- Filter(function(path) {

    if (length(path) > 2) {

      return(TRUE)

    } else {

      # If length is 2, check if terminal node is modelling paper -------------

      return(!(tail(path, n = 1) %in% modelling_papers))
    }
  }, all_paths)

  # Calculate the number of paths ----------------------------------------
```

```r
  citation_amplification_index <- length(filtered_paths)

  return(citation_amplification_index)
}

# Function to apply previous function to all papers ----------------------------

calculate_all_cai <- function(graph, modelling_papers) {

  papers <- V(graph)$name

  # Initialize an empty data.table to store results ----------------------------

  result_dt <- data.table(
    paper = character(),
    cai = numeric()
  )

  # Iterate over each paper ----------------------------------------------------

  for (paper in papers) {

    cai <- citation_amplification_index(graph, paper, modelling_papers)
    result_dt <- rbind(result_dt, data.table(paper = paper, cai = cai))
  }

  return(result_dt)
}

# CALCULATE AMPLIFICATION INDEX ################################################

# Calculate average amplification index of the networks ------------------------
# (e.g., the number paths initiated by the average paper
# leading to studies that do # not flow directly to "primary" data)

out <- list()

for(i in names(modelling.papers)) {

  out[[i]] <- calculate_all_cai(graph[[i]], modelling.papers[[i]])
}

# ARRANGE DATA #################################################################

tmp <- rbindlist(out, idcol = "topic")

# SUMMARY STATISTICS ###########################################################
```

```r
tmp[, mean(cai), topic]
```

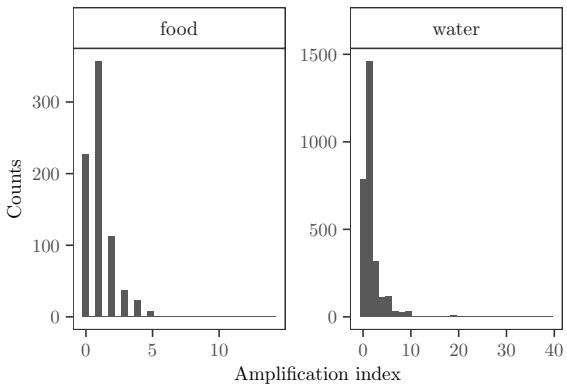
```
##     topic       V1
##    <char>    <num>
## 1:   food 1.127604
## 2:  water 1.821880
```

```r
# First 5 papers amplifying the most ----------------------------------------

tmp2 <- tmp[, .SD, topic] %>%
  .[order(-cai, topic)] %>%
  .[, head(.SD, 5), topic]

tmp2
```

```
##      topic               paper   cai
##     <char>              <char> <num>
##  1:  water           wada 2015    39
##  2:  water neysiani et al 2022    27
##  3:  water          habtu 2024    23
##  4:  water   abioye et al 2023    22
##  5:  water  derossi et al 2024    22
##  6:   food   taguta et al 2022    14
##  7:   food     guo et al 2023a    12
##  8:   food   kadigi et al 2004     8
##  9:   food   xie and lark 2021     7
## 10:   food      mao et al 2024     6
```

```r
# PLOT DISTRIBUTION OF AMPLIFICATION INDIXES ##################################

plot.amplification <- list()

# PLOT ########################################################################

plot.amplification <- ggplot(tmp, aes(cai)) +
  geom_histogram() +
  facet_wrap(~topic, scales = "free") +
  theme_AP() +
  labs(y = "Counts", x = "Amplification index")

plot.amplification
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

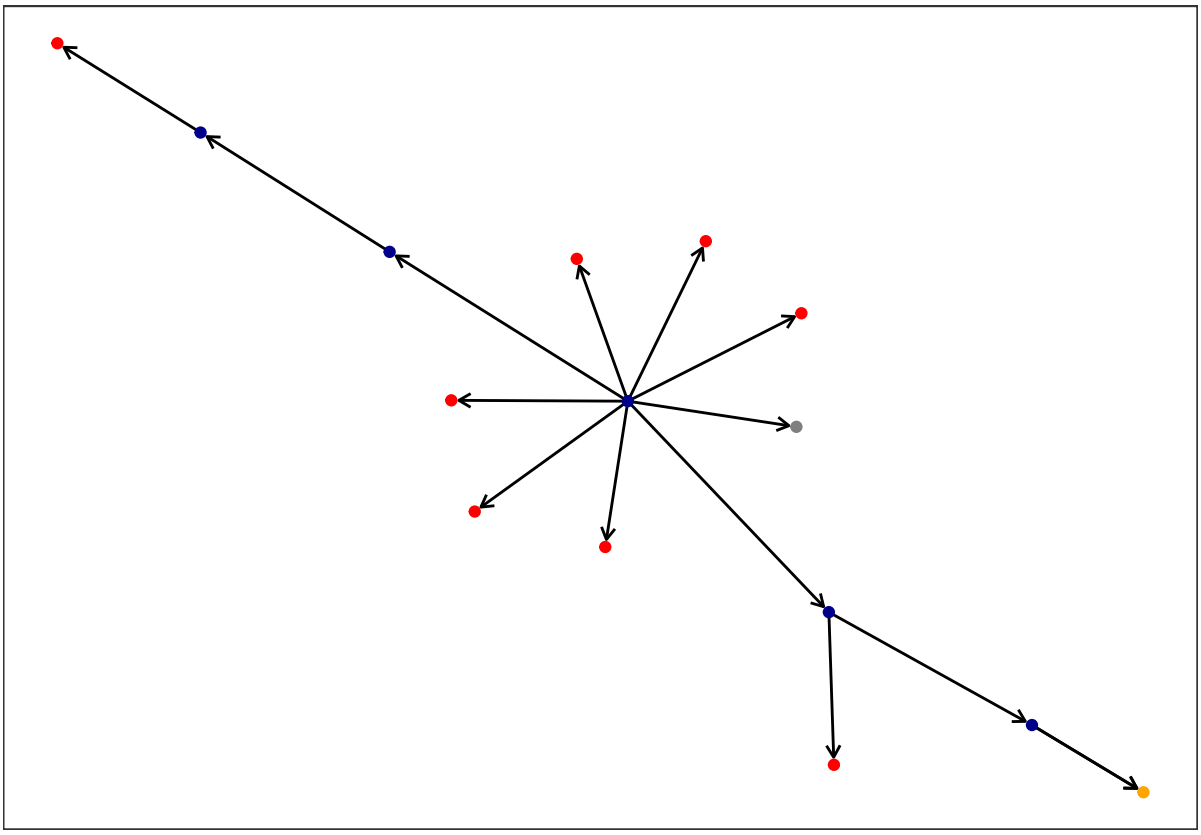



```
# PLOT THE NETWORK OF TOP AMPLIFYING PAPERS ###################################

# Define the starting nodes ---------------------------------------------------

vec.names.amplification <- tmp2[, slice_max(.SD, cai, n = 1), topic] %>%
  .[, paper]

# Reorder so the top node for food comes first --------------------------------

vec.names.amplification <- vec.names.amplification[order(vec.names.amplification)]

out <- list()

for (i in 1:length(vec.names.amplification)) {

  start_node <- vec.names.amplification[[i]]

  # Get all simple paths from the starting node to all reachable nodes
  paths <- all_simple_paths(graph[[i]], from = V(graph[[i]])[name == start_node])

  # Convert paths to a list of edge pairs for easy visualization
  edge_list <- lapply(paths, function(path) {
    edges <- as_edgelist(induced_subgraph(graph[[i]], path), names = TRUE)
    data.frame(from = edges[, 1], to = edges[, 2])
  })

  # Combine all path edges into a single data frame
  path_edges <- do.call(rbind, edge_list)

  #Get the unique nodes involved in the paths
  path_nodes <- unique(c(path_edges$from, path_edges$to))

  # Create the subgraph of all paths starting from the target node
  subgraph <- induced_subgraph(graph[[i]], vids = V(graph[[i]])[name %in% path_nodes])

  # Convert the subgraph to a tidygraph object
```

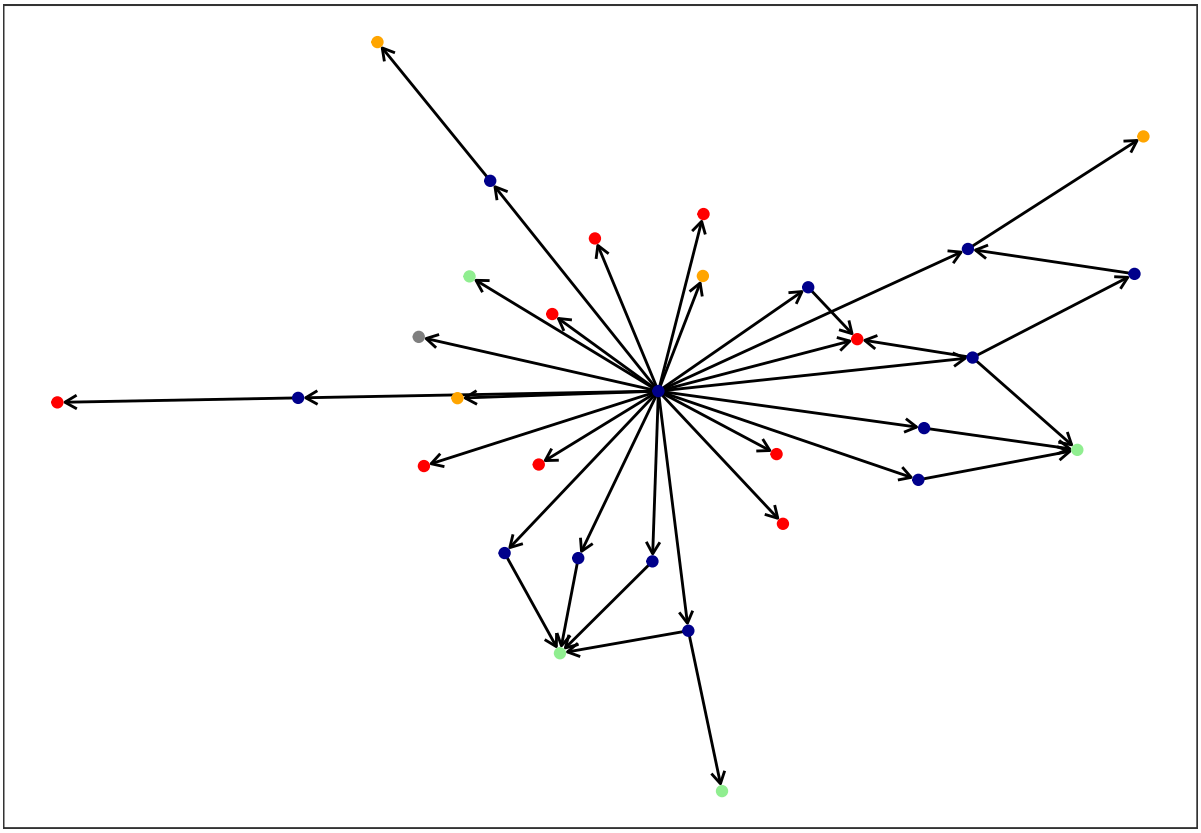
```r
subgraph_tbl <- as_tbl_graph(subgraph)

# Retrieve a vector with the node names ---------------------------------------

vec.names <- subgraph_tbl %>%
  activate(nodes) %>%
  pull() %>%
  data.table(name = .)

nature.claim.vec <- graph.final[[i]] %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table() %>%
  .[name %in% vec.names$name] %>%
  .[, nature.claim]

subgraph_tbl <- subgraph_tbl %>%
  activate(nodes) %>%
  mutate(nature.claim = nature.claim.vec)

# Plot the subgraph with ggraph

if (i == 1) {

  selected_colors <- c("darkblue", "orange", "red")

} else {

selected_colors <- c("darkblue", "lightgreen", "orange", "red")

}

set.seed(123)

out[[i]] <- ggraph(subgraph_tbl, layout = "igraph", algorithm = "nicely") +
  geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                 end_cap = circle(1, "mm")) +
  geom_node_point(size = 1.5, aes(color = nature.claim)) +
  scale_color_manual(name = "",
                     values = selected_colors) +
  theme_AP() +
  labs(x = "", y = "") +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "none",
```

```
          legend.text = element_text(size = 7.3))
}

out
```

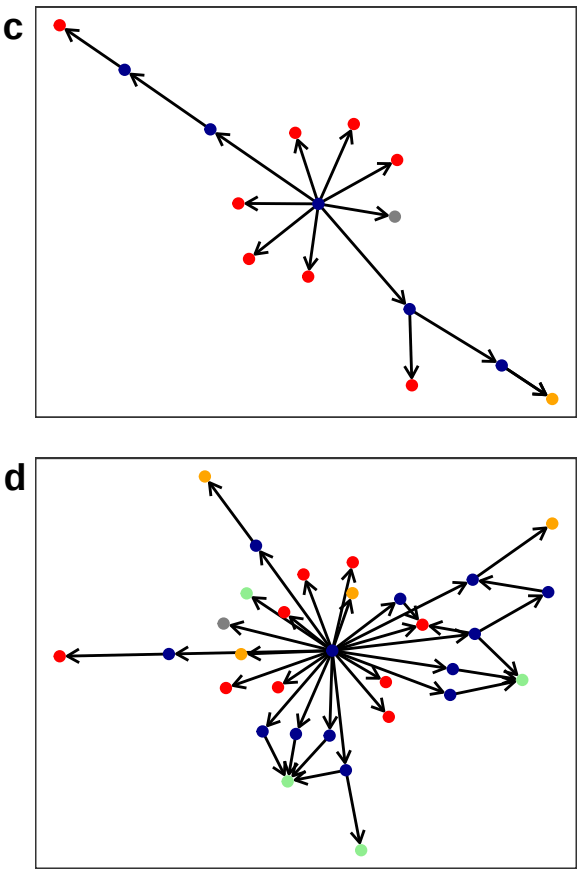
```
## [[1]]
```



```
##
## [[2]]
```



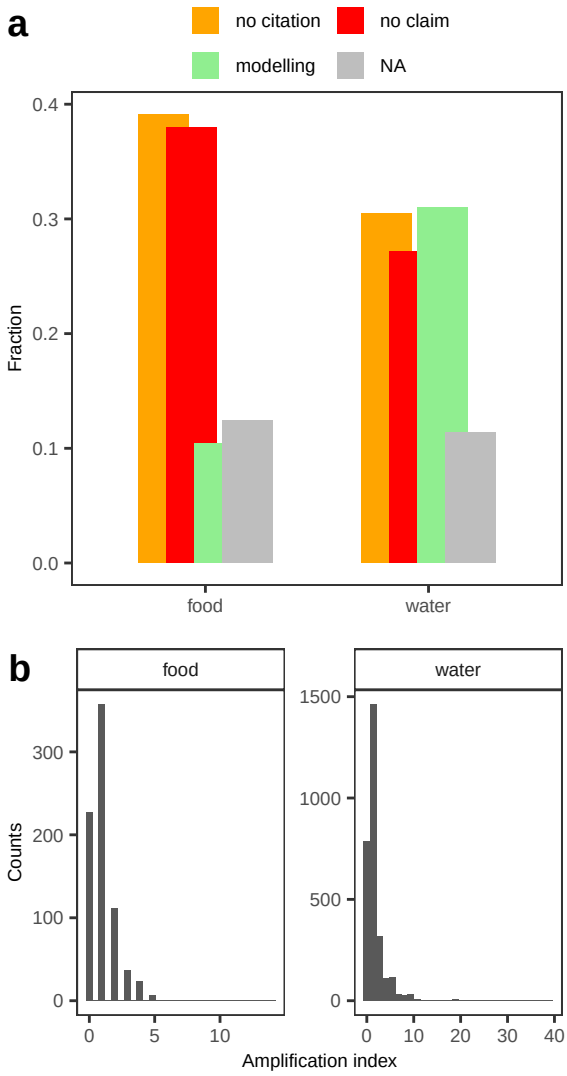
```
plot.ampl.merged <- plot_grid(out[[1]],
                              out[[2]], ncol = 1,
                              labels = c("c", "d"))

plot.ampl.merged
```



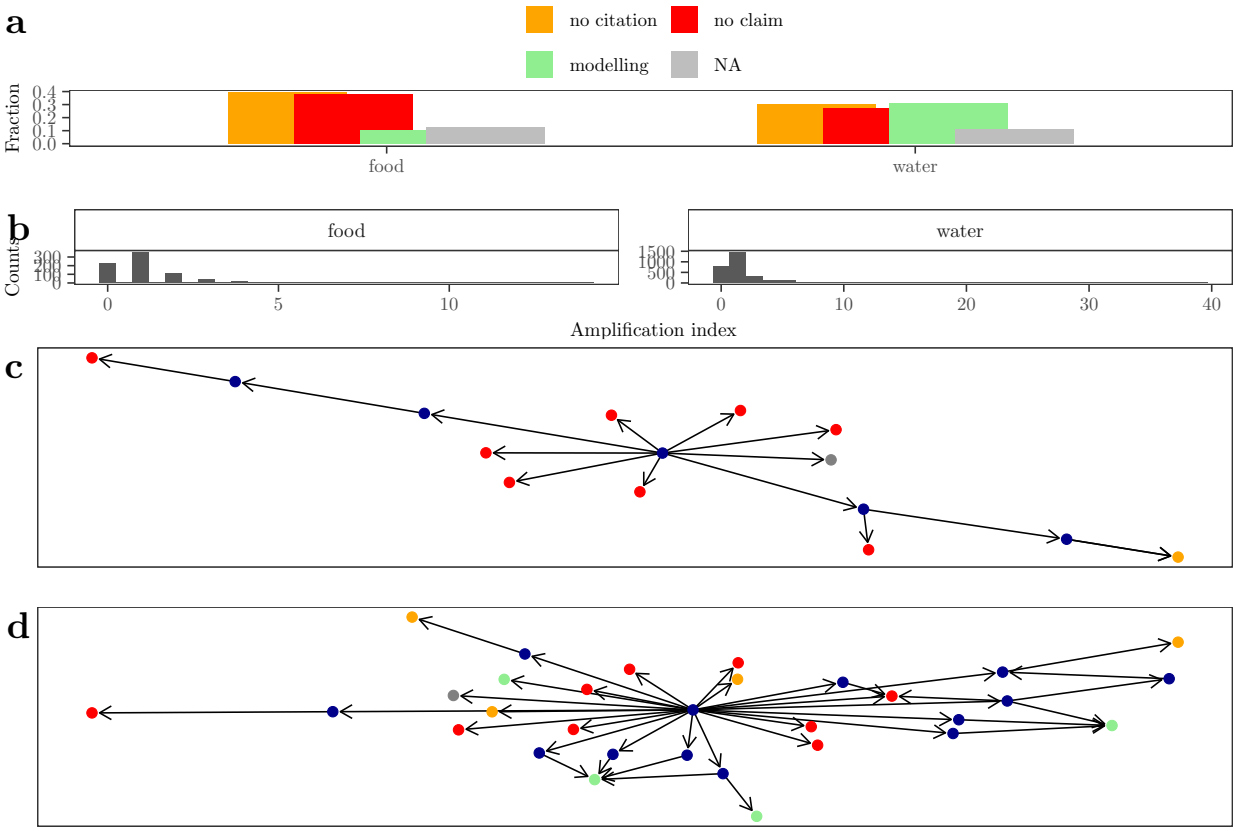

```
full.amplification.plot <- plot_grid(plot.ending.modelling +
                                       guides(fill = guide_legend(nrow = 2, byrow = TRUE)) +
                                       theme(legend.position = "top"),
                                     plot.amplification,
                                     rel_heights = c(0.6, 0.4),
                                     ncol = 1, labels = "auto")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

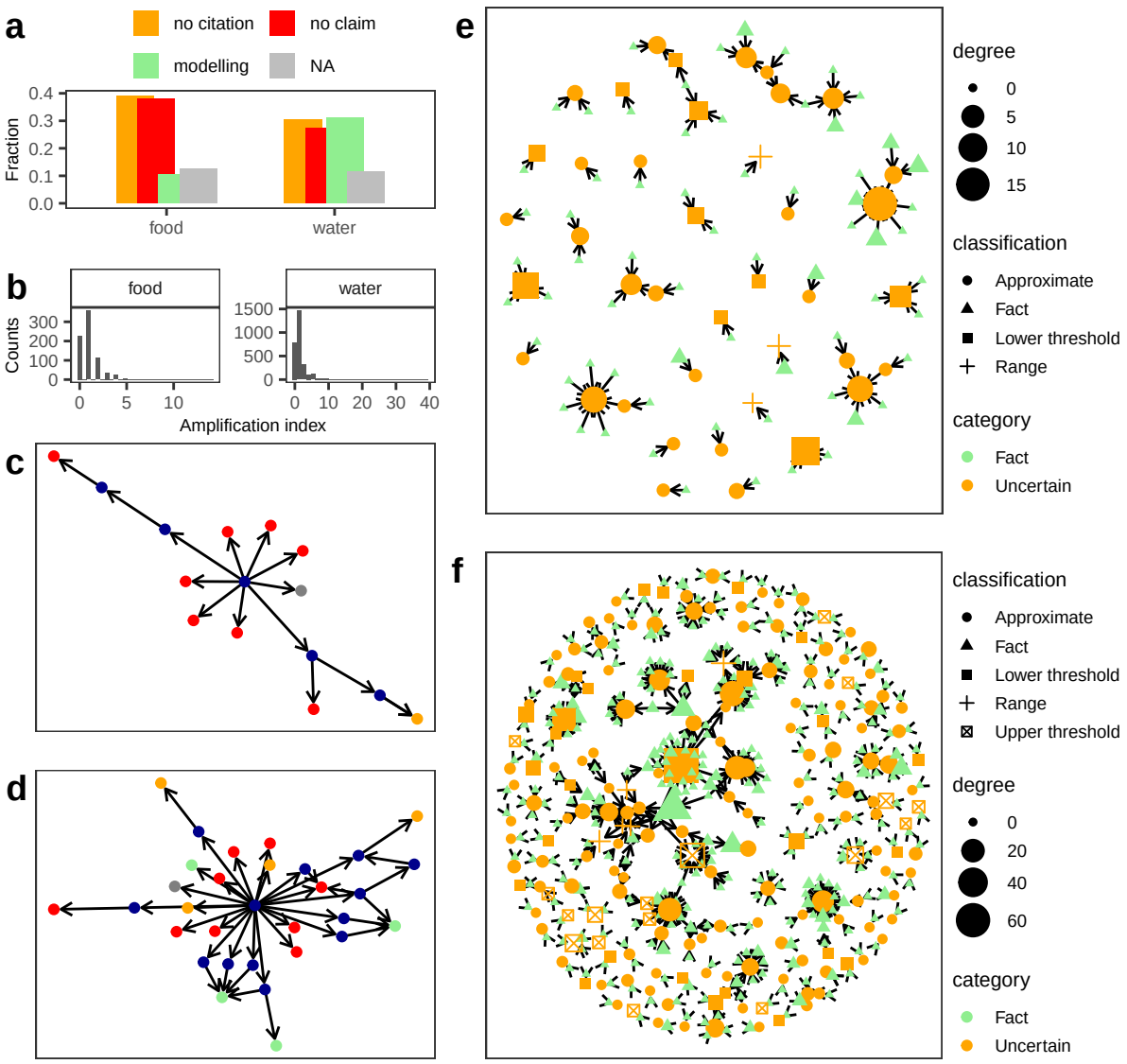
```
full.amplification.plot
```




```
full.amplification.plot <- plot_grid(full.amplification.plot,
                                     plot.ampl.merged, ncol = 1,
                                     rel_heights = c(0.4, 0.6),
                                     labels = c("", ""))

full.amplification.plot
```




```
plot_grid(full.amplification.plot, plot.uncertainty.paths, ncol = 2, rel_widths = c(0.4, 0.6))
```



4 Both networks

4.1 Overlap between networks

```
# CHECK FULL NETWORK AND OVERLAP BETWEEN WATER AND FOOD NETWORK ################

# Prepare data ----------------------------------------------------------------

dd <- tidygraph::as_tbl_graph(network.dt.complete, directed = TRUE)
```

```
## Warning: In `d`, `NA` elements were replaced with string "NA".
```

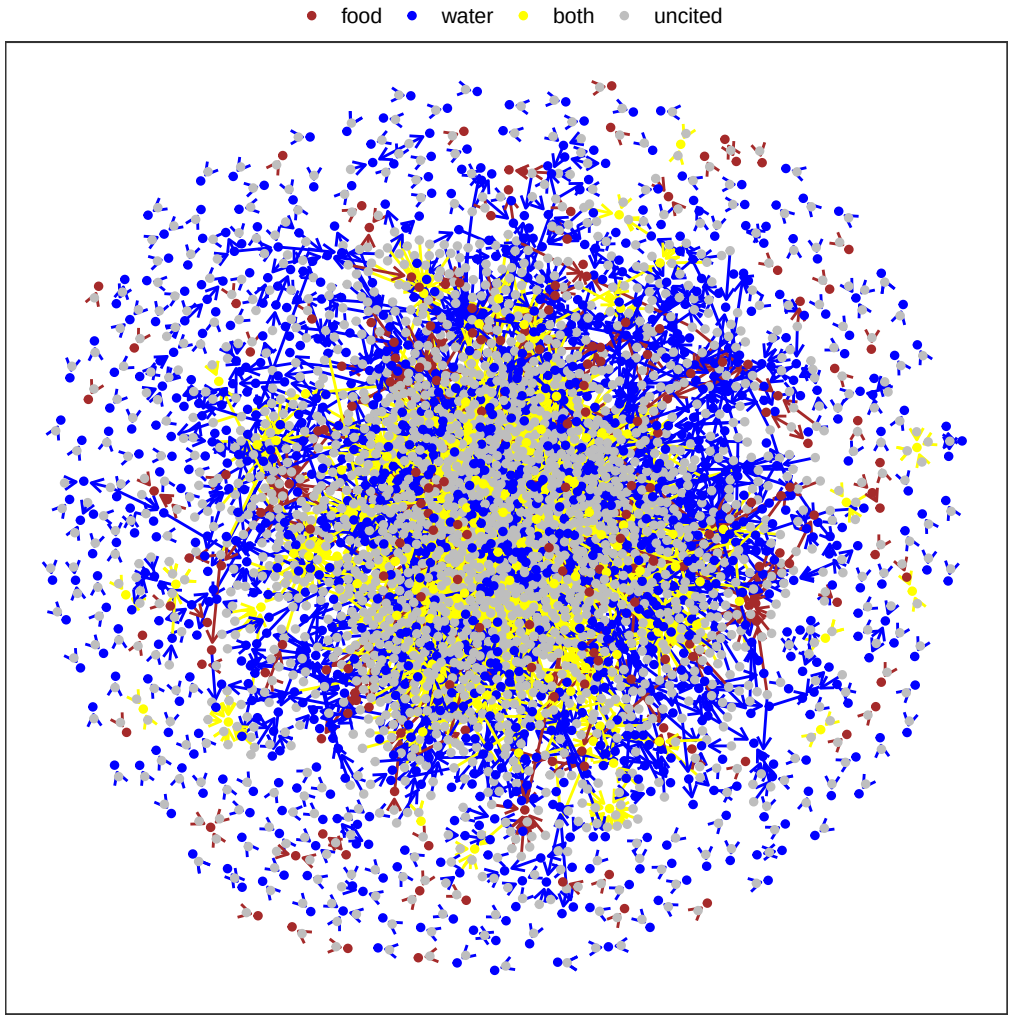
```
# Extract vector with names ---------------------------------------------------

all.names <- dd %>%
  activate(nodes) %>%
```

```
  pull(name)

# Retrieve names from water and food belief system ----------------------------

names.food <- network.dt.complete[topic == "food", to]
names.water <- network.dt.complete[topic == "water", to]

# Define intersections and differences ----------------------------------------

names.only.food <- setdiff(names.food, names.water)
names.only.water <- setdiff(names.water, names.food)
names.both <- intersect(names.water, names.food)

# New column defining whether nodes are in water, food or in both networks -----

final.graph <- dd %>%
  activate(nodes) %>%
  mutate(topic.final = ifelse(name %in% names.only.food, "food",
                              ifelse(name %in% names.only.water, "water",
                                     ifelse(name %in% names.both, "both", "uncited"))),
         topic.final = factor(topic.final, levels = c("food", "water", "both", "uncited")))

final.graph <- final.graph %>%
  activate(edges) %>%
  mutate(edge_color = .N()$topic.final[to])

# SOME STATS ###################################################################

dt.nodes <- final.graph %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table()

# Fraction of network overlap -------------------------------------------------

dt.nodes[, .N, topic.final] %>%
  .[, fraction:= N / nrow(dt.nodes)] %>%
  print
```

```
##    topic.final     N   fraction
##         <fctr> <int>      <num>
## 1:     uncited  2116 0.61050202
## 2:       water  1042 0.30063474
## 3:        both   125 0.03606463
## 4:        food   183 0.05279862
```

```
# PLOT MERGED NETWORK ##########################################################
```

```
selected.colors <- c("brown", "blue", "yellow", "grey")

ggraph(final.graph, layout = "graphopt") +
  geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                 end_cap = circle(1, "mm"),
                 aes(color = edge_color)) +
  geom_node_point(aes(color = topic.final), size = 1) +
  scale_edge_color_manual(values = selected.colors, guide = "none") +
  scale_color_manual(name = "",
                     values = selected.colors) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "top")
```



4.2 Shared network

```
# PLOT ONLY THE NETWORK OF NODES BEING CITED FOR BOTH BELIEFS ##################

# Prepare data ----------------------------------------------------------------

intersect.network <- network.dt.complete[to %in% names.both] %>%
  tidygraph::as_tbl_graph(., directed = TRUE)
```

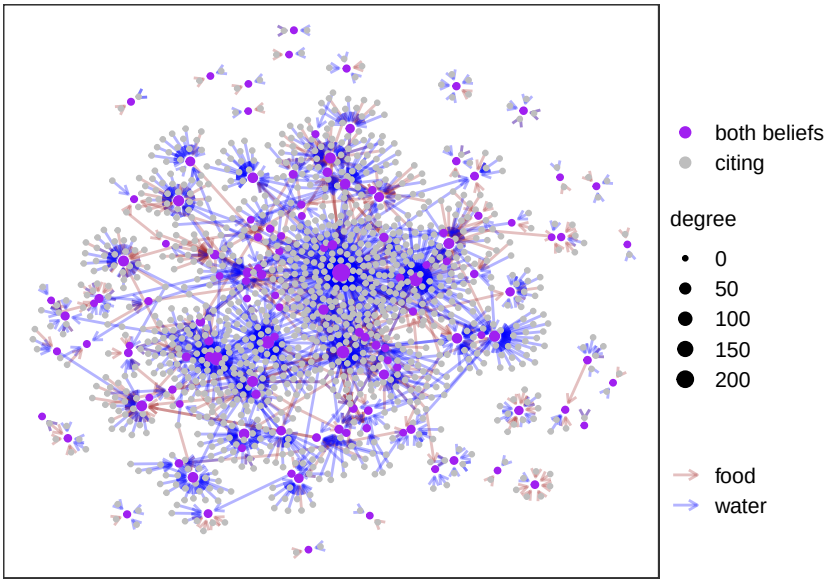
```
## Warning: In `d`, `NA` elements were replaced with string "NA".
```

```
intersect.graph <- network.dt.complete[to %in% names.both] %>%
  graph_from_data_frame(d = ., directed = TRUE)
```

```
## Warning: In `d`, `NA` elements were replaced with string "NA".
```

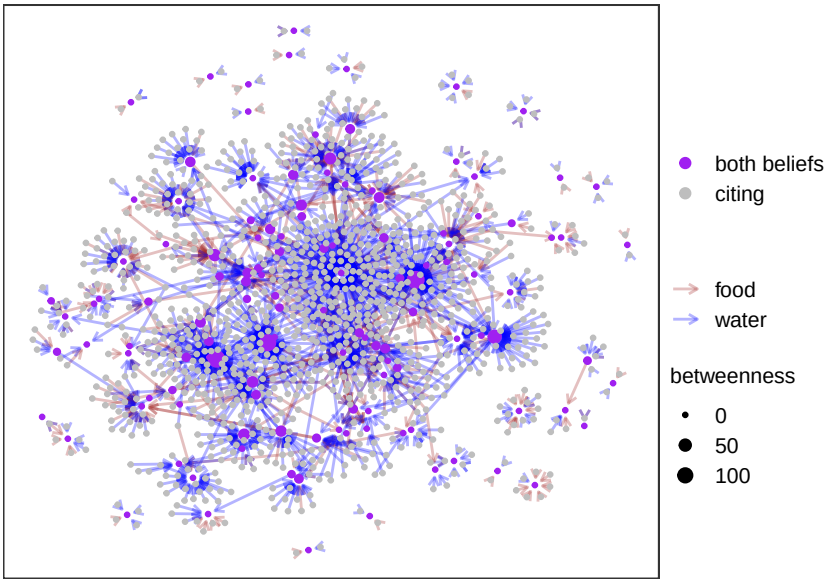
```
# Calculate metrics -----------------------------------------------------------

intersect.metrics <- data.table(node = V(intersect.graph)$name,
                                degree = degree(intersect.graph, mode = "in"),
                                degree.out = degree(intersect.graph, mode = "out"),
                                betweenness = betweenness(intersect.graph),
                                closeness = closeness(intersect.graph),
                                pagerank = page_rank(intersect.graph)$vector)

degree.nodes <- intersect.metrics[order(-degree)][1:3]
degree.out.nodes <- intersect.metrics[order(-degree.out)][1:3]
betweenness.nodes <- intersect.metrics[order(-betweenness)][1:3]

# Retrieve a vector with the node names ---------------------------------------

vec.names <- intersect.network %>%
  activate(nodes) %>%
  pull() %>%
  data.table(name = .)

order <- match(vec.names$name, intersect.metrics$node)
tmp <- intersect.metrics[order]

intersect.graph.final <- intersect.network %>%
  activate(nodes) %>%
  mutate(degree = tmp$degree,
         degree.out = tmp$degree.out,
         betweenness = tmp$betweenness)

intersect.graph.final <- intersect.graph.final %>%
  activate(nodes) %>%
  mutate(topic = ifelse(name %in% names.both, "both beliefs", "citing"))
```

```
# PLOT ###############################################################################

set.seed(12)

ggraph(intersect.graph.final, layout = "graphopt") +
  geom_edge_link(arrow = arrow(length = unit(1.3, 'mm')),
                 end_cap = circle(1, "mm"),
                 aes(color = topic),
                 alpha = 0.3) +
  geom_node_point(aes(color = topic, size = degree)) +
  scale_edge_color_manual(values = c("brown", "blue"),
                          name = "") +
  scale_size_continuous(range = c(0.5, 2.5)) +
  scale_color_manual(name = "",
                     values = c("purple", "grey")) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right")
```




```
set.seed(12)

ggraph(intersect.graph.final, layout = "graphopt") +
  geom_edge_link(arrow = arrow(length = unit(1.3, 'mm')),
                 end_cap = circle(1, "mm"),
                 aes(color = topic),
                 alpha = 0.3) +
  geom_node_point(aes(color = topic, size = betweenness)) +
```

```
  scale_edge_color_manual(values = c("brown", "blue"),
                          name = "") +
  scale_size_continuous(range = c(0.5, 2.5)) +
  scale_color_manual(name = "",
                     values = c("purple", "grey")) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right")
```



5 Study of Aquastat values

```
# STUDY OF AQUASTAT PERCENTAGES ###################################################

# Read in aquastat dataset ----------------------------------------------------

aquastat.dt <- read.xlsx("aquastat_dt.xlsx") %>%
  data.table() %>%
  .[Year == 2020] %>%
  setnames(., c("Value", "Area"), c("percentage", "country")) %>%
  .[, .(country, percentage)] %>%
  .[, data:= "aquastat 2020"] %>%
  .[, country:= countrycode(country, origin = "country.name", destination = "country.name")]
```

```
## Warning: Some values were not matched unambiguously: Australia and New Zealand
```

```
## Warning: Some strings were matched more than once, and therefore set to <NA> in the result:
```

```
aquastat.dt[, continent:= countrycode(country, origin = "country.name", destination = "continer

# Read in world resources institute dataset ----------------------------------

wri <- fread("world_resources_institut_guide_to_the_global_environment_1994.csv") %>%
  .[order(country)] %>%
  .[, data:= "wri 1994"] %>%
  .[, country:= countrycode(country, origin = "country.name", destination = "country.name")]
```

```
## Warning: Some values were not matched unambiguously: , Cote d'lvoire
```

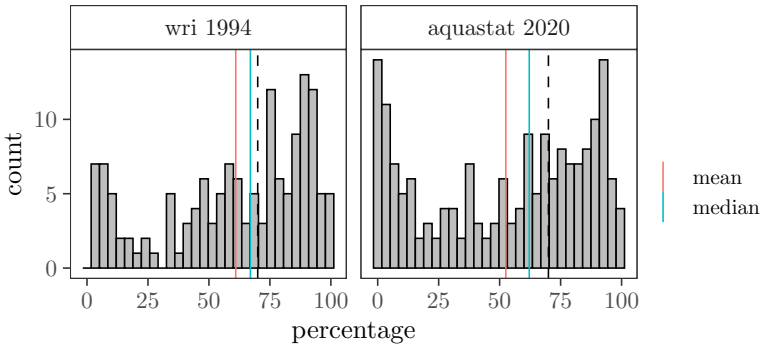
```
wri[, continent:= countrycode(country, origin = "country.name", destination = "continent")]
```

```
## Warning: Some values were not matched unambiguously: Czechoslovakia, Yugoslavia
```

```
# Compare distributions ------------------------------------------------------

dt.comparison <- rbind(aquastat.dt, wri) %>%
  .[, data:= factor(data, levels = c("wri 1994", "aquastat 2020"))]

dt.stats.comparison <- dt.comparison[, .(mean = mean(percentage, na.rm = TRUE),
                                         median = median(percentage, na.rm = TRUE)), data] %>%
  melt(., measure.vars = c("mean", "median"))


ggplot(dt.comparison, aes(percentage)) +
  geom_histogram(color = "black", fill = "grey") +
  facet_wrap(~data) +
  geom_vline(data = dt.stats.comparison, aes(xintercept = value, color = variable)) +
  scale_color_discrete(name = "") +
  geom_vline(xintercept = 70, lty = 2) +
  theme_AP()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```



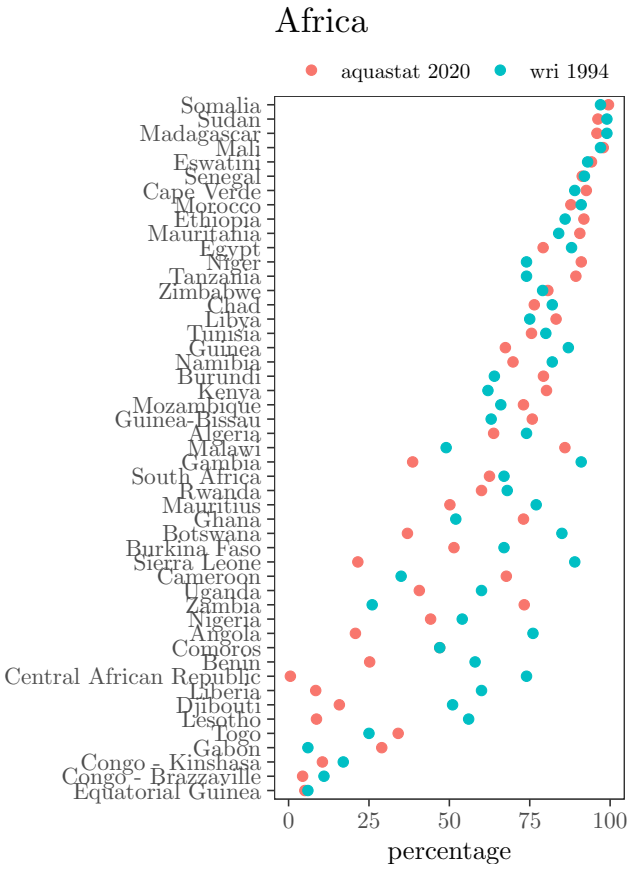

```
# At the country level -------------------------------------------------------
```

```
tmp <- aquastat.dt[wri, on = c("country", "continent")] %>%
  .[, .(country,continent, percentage, i.percentage)] %>%
  setnames(., c("percentage", "i.percentage"), c("aquastat 2020", "wri 1994")) %>%
  melt(., measure.vars = c("aquastat 2020", "wri 1994")) %>%
  .[, country:= ifelse(country == "Trinidad & Tobago", "Trinidad and Tobago", country)] %>%
  na.omit() %>%
  split(., .$continent)
```

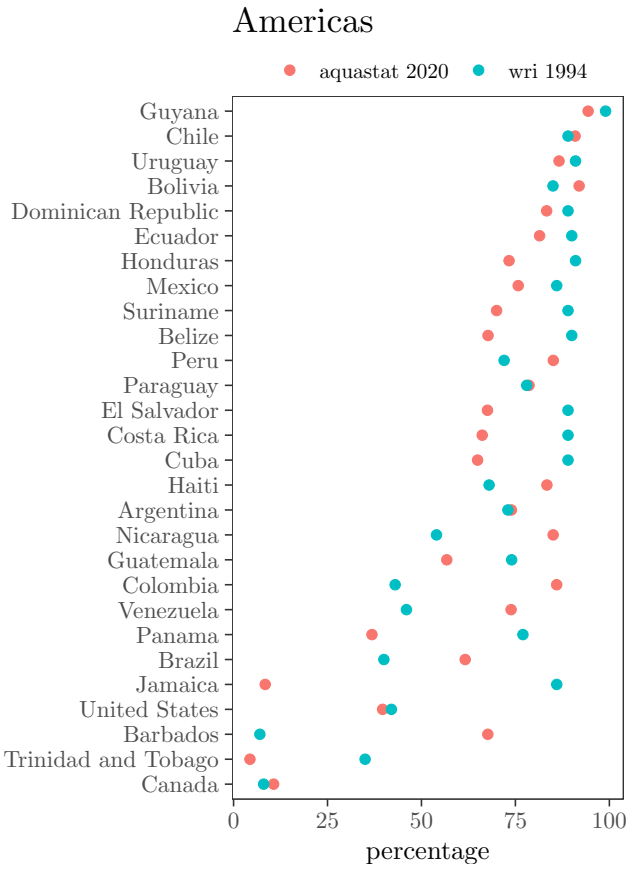
```
## Warning in melt.data.table(., measure.vars = c("aquastat 2020", "wri 1994")):
## 'measure.vars' [aquastat 2020, wri 1994] are not all of the same type. By order
## of hierarchy, the molten data value column will be of type 'double'. All
## measure variables not of type 'double' will be coerced too. Check DETAILS in
## ?melt.data.table for more on coercion.
```

```
out <- list()

for(i in names(tmp)) {

  out[[i]] <- ggplot(tmp[[i]], aes(reorder(country, value),
                                   value, color = variable)) +
    coord_flip() +
    scale_color_discrete(name = "") +
    geom_point() +
    theme_AP() +
    theme(legend.position = "top") +
    labs(x = "", y = "percentage") +
    ggtitle(names(tmp[i]))
}

out
```

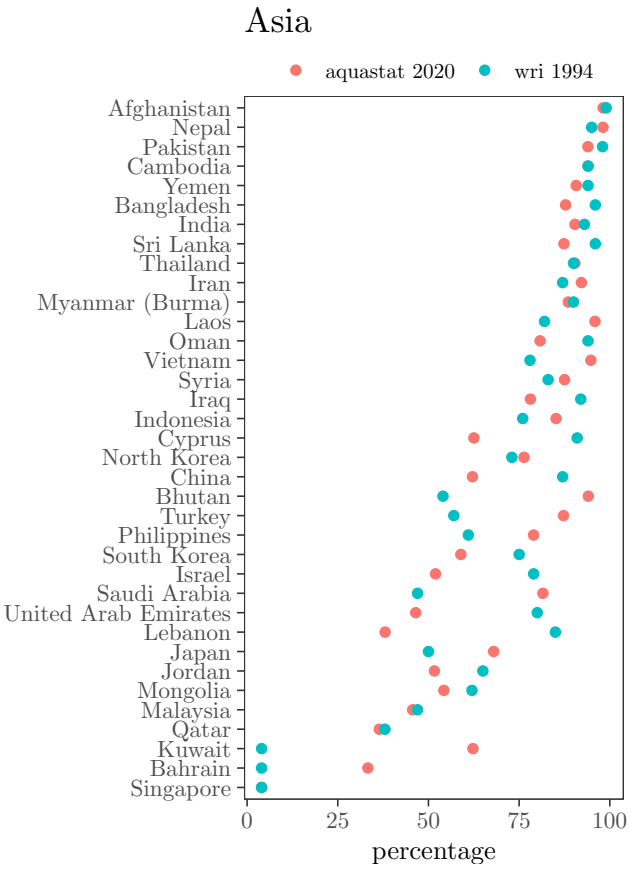
```
## $Africa
```


Africa
aquastat 2020
wri 1994
Somalia
Sudan
Madagascar
Mali
Eswatini
Senegal
Cape Verde
Morocco
Ethiopia
Mauritania
Egypt
Niger
Tanzania
Zimbabwe
Chad
Libya
Tunisia
Guinea
Namibia
Burundi
Kenya
Mozambique
Guinea-Bissau
Algeria
Malawi
Gambia
South Africa
Rwanda
Mauritius
Ghana
Botswana
Burkina Faso
Sierra Leone
Cameroon
Uganda
Zambia
Nigeria
Angola
Comoros
Benin
Central African Republic
Liberia
Djibouti
Lesotho
Togo
Gabon
Congo - Kinshasa
Congo - Brazzaville
Equatorial Guinea
0
25
50
75
100
percentage

```
##
## $Americas
```


Americas
aquastat 2020
wri 1994
Guyana
Chile
Uruguay
Bolivia
Dominican Republic
Ecuador
Honduras
Mexico
Suriname
Belize
Peru
Paraguay
El Salvador
Costa Rica
Cuba
Haiti
Argentina
Nicaragua
Guatemala
Colombia
Venezuela
Panama
Brazil
Jamaica
United States
Barbados
Trinidad and Tobago
Canada
0
25
50
75
100
percentage

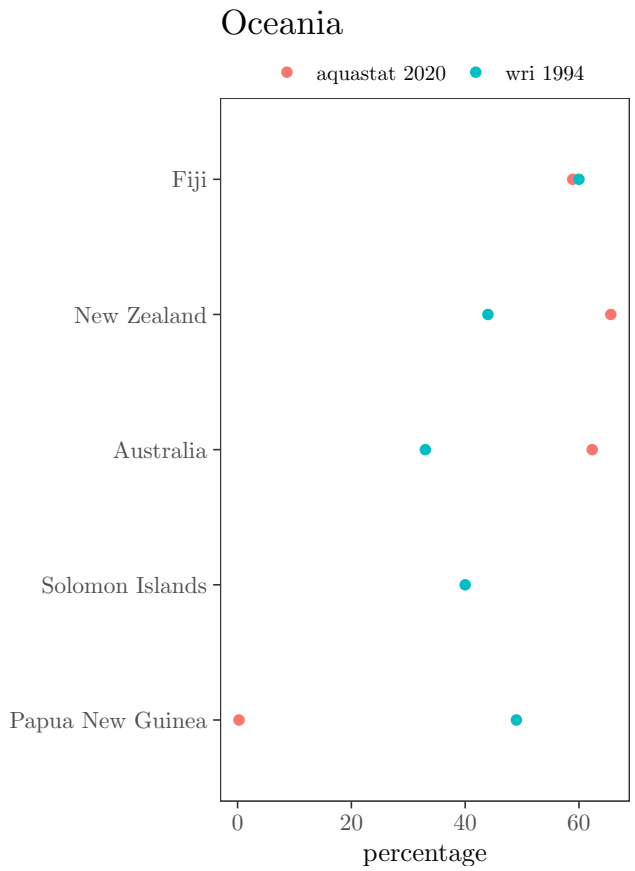
```
##
## $Asia
```


Asia
aquastat 2020
wri 1994
Afghanistan
Nepal
Pakistan
Cambodia
Yemen
Bangladesh
India
Sri Lanka
Thailand
Iran
Myanmar (Burma)
Laos
Oman
Vietnam
Syria
Iraq
Indonesia
Cyprus
North Korea
China
Bhutan
Turkey
Philippines
South Korea
Israel
Saudi Arabia
United Arab Emirates
Lebanon
Japan
Jordan
Mongolia
Malaysia
Qatar
Kuwait
Bahrain
Singapore
0
25
50
75
100
percentage

```
##
## $Europe
```


Europe
aquastat 2020
wri 1994
Albania
Greece
Spain
Italy
Portugal
Denmark
Romania
Bulgaria
Malta
Hungary
Norway
Poland
Netherlands
Finland
France
Germany
United Kingdom
Sweden
Switzerland
Luxembourg
Ireland
Austria
Iceland
Belgium
0
20
40
60
80
percentage

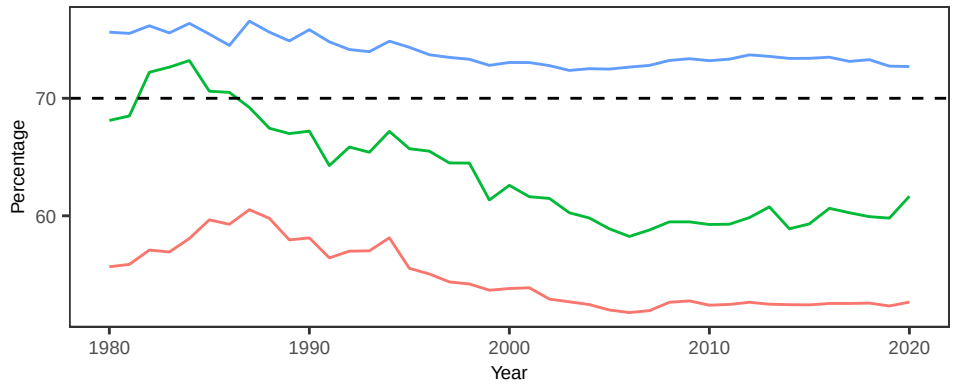
```
##
## $Oceania
```




```
# AQUASTAT ALL YEARS ##############################################################

# DATA FOR THE WATER BELIEF ########################################################

# Read in dataset ----------------------------------------------------------

aquastat.aww <- fread("aquastat.fraction.agriculture.withdrawals.csv")
aquastat.aww.stats <- aquastat.aww[, .(mean = mean(Value, na.rm = TRUE),
                                       median = median(Value, na.rm = TRUE)), Year] %>%
  melt(., measure.vars = c("mean", "median"))

# Weighted average --------------

aquastat.aww[, weights:= Value / sum(Value), Year]
weighted.average.dt <- aquastat.aww[, .(value = sum(Value * weights)), Year] %>%
  .[, variable:= "weighted \n average"]

# Plot ---------------------------------------------------------------------

a <- aquastat.aww.stats %>%
  rbind(weighted.average.dt) %>%
  ggplot(., aes(Year, value, group = variable, color = variable)) +
  geom_line() +
```

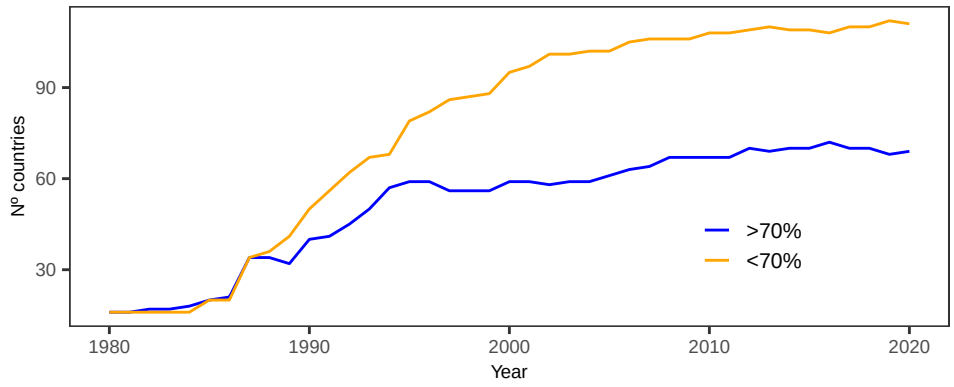
```
  scale_color_discrete(name = "") +
  geom_hline(yintercept = 70, lty = 2) +
  theme_AP() +
  labs(x = "Year", y = "Percentage") +
  theme(legend.position = "none")
```

a




```
b <- aquastat.aww[, .(above.70 = sum(Value > 70),
                      below.70 = sum(Value < 70)), Year] %>%
  melt(., measure.vars = c("above.70", "below.70")) %>%
  ggplot(., aes(Year, value, color = variable)) +
  geom_line() +
  theme_AP() +
  scale_color_manual(name = "", labels = c(">70%", "<70%"),
                     values = c("blue", "orange")) +

  labs(x = "Year", y = "Nº countries") +
  theme(legend.position = c(0.78, 0.34))
```

b




```
# Fraction of estimated and imputed values ------------------------------------

n.countries <- aquastat.aww[, .(total.countries = .N), Year]
fraction.estimate <- aquastat.aww[, .N, .(Symbol, Year)] %>%
```

```
  merge(., n.countries, by = "Year") %>%
  .[, fraction:= N / total.countries] %>%
  ggplot(., aes(Year, N, color = Symbol)) +
  geom_line() +
  labs(x = "Year", y = "Nº countries") +
  scale_color_discrete(name = "") +
  theme_AP() +
  theme(legend.position = c(0.85, 0.25),
        legend.text = element_text(size = 7))

dt.stats.year <- aquastat.aww.stats %>%
  rbind(weighted.average.dt) %>%
  .[Year == max(Year)]

water.histogram <- aquastat.aww[Year == max(Year)] %>%
  ggplot(., aes(Value)) +
  geom_histogram(color = "black", fill = "grey") +
  geom_vline(xintercept = 70, lty = 2) +
  geom_vline(data = dt.stats.year, aes(xintercept = value, color = variable)) +
  labs(x = "Percentage", y = "Nº countries") +
  theme_AP() +
  theme(legend.position = "none")

water.plots.aquastat <- plot_grid(a, water.histogram, b, fraction.estimate,
                                  ncol = 4, labels = c("b", ""))
```

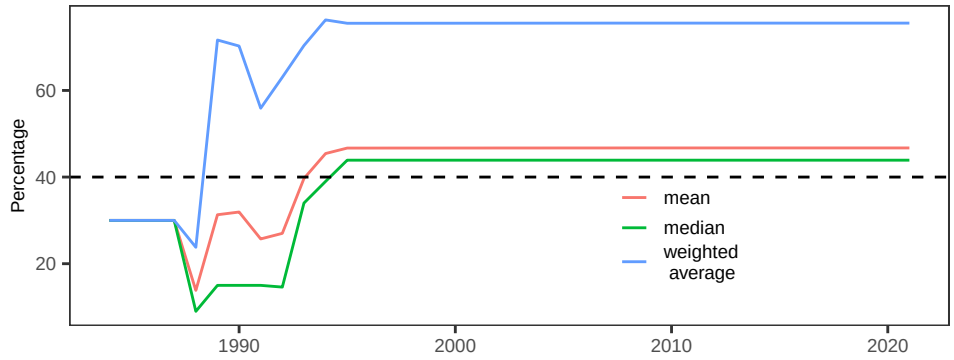
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
# DATA FOR THE FOOD BELIEF ############################################################

dt <- fread("aquastat.fraction.grain.irrigated.csv")

# Check the variable examined ---------------------------------------------------

unique(dt$Variable)
```

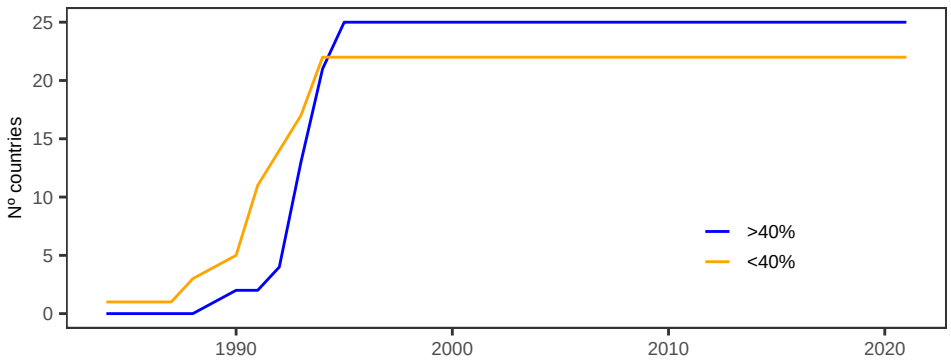
```
## [1] "% of total grain production irrigated"
```

```
# Calculate mean and median -----------------------------------------------------

aquastat.grain.stats <- dt[, .(mean = mean(Value),
                               median = median(Value),
                               N.countries = .N), Year] %>%
  melt(., measure.vars = c("mean", "median"))

# Calculated weighted average ---------------------------------------------------

dt[, weights:= Value / sum(Value), Year]
weighted.average.dt <- dt[, .(value = sum(Value * weights),
```

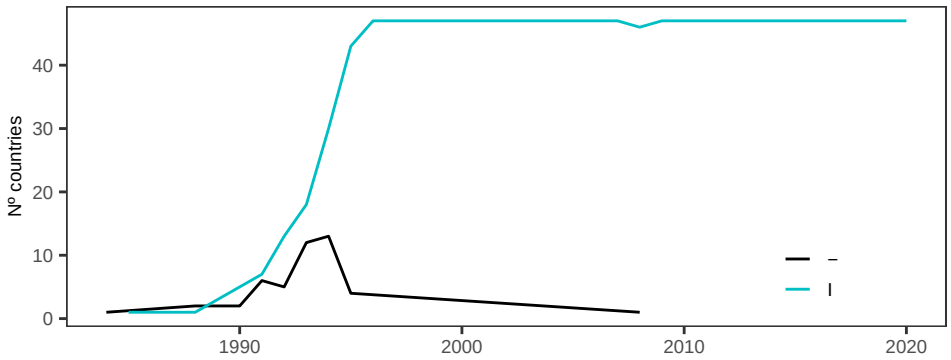
```
                                N.countries = .N), Year] %>%
  .[, variable:= "weighted \n average"]

# Plot ------------------------------------------------------------------------

a.crop <- aquastat.grain.stats %>%
  rbind(weighted.average.dt) %>%
  ggplot(., aes(Year, value, group = variable, color = variable)) +
  geom_line() +
  scale_color_discrete(name = "") +
  geom_hline(yintercept = 40, lty = 2) +
  theme_AP() +
  labs(x = "", y = "Percentage") +
  theme(legend.position = c(0.7, 0.38),
        legend.text = element_text(size = 7))

a.crop
```




```
b.crop <- dt[, .(above.40 = sum(Value > 40),
                 below.40 = sum(Value < 40)), Year] %>%
  melt(., measure.vars = c("above.40", "below.40")) %>%
  ggplot(., aes(Year, value, color = variable)) +
  geom_line() +
  theme_AP() +
  scale_color_manual(name = "", labels = c(">40%", "<40%"),
                     values = c("blue", "orange")) +

  labs(x = "", y = "Nº countries") +
  theme(legend.position = c(0.78, 0.34),
        legend.text = element_text(size = 7))

b.crop
```



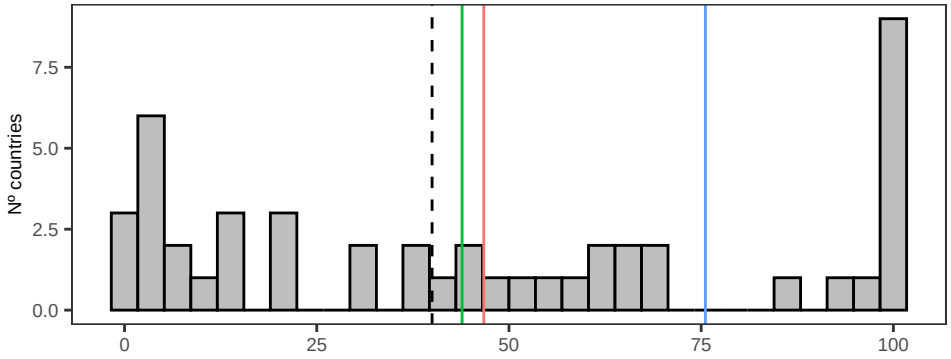

```
n.countries.crop <- dt[, .(total.countries = .N), Year]
fraction.estimate <- dt[, .N, .(Symbol, Year)] %>%
  merge(., n.countries, by = "Year") %>%
  .[, fraction:= N / total.countries] %>%
  ggplot(., aes(Year, N, color = Symbol)) +
  geom_line() +
  labs(x = "", y = "Nº countries") +
  scale_color_manual(name = "", values = c("black", "#00BFC4")) +
  theme_AP() +
  theme(legend.position = c(0.85, 0.25),
        legend.text = element_text(size = 7))

fraction.estimate
```




```
dt.stats.year.crops <- aquastat.grain.stats %>%
  rbind(weighted.average.dt) %>%
  .[Year == max(Year)]

crop.histogram <- dt[Year == max(Year)] %>%
  ggplot(., aes(Value)) +
  geom_histogram(color = "black", fill = "grey") +
  geom_vline(xintercept = 40, lty = 2) +
  geom_vline(data = dt.stats.year.crops, aes(xintercept = value, color = variable)) +
  labs(x = "", y = "Nº countries") +
  theme_AP() +
  theme(legend.position = "none")
```

```
crop.histogram
```

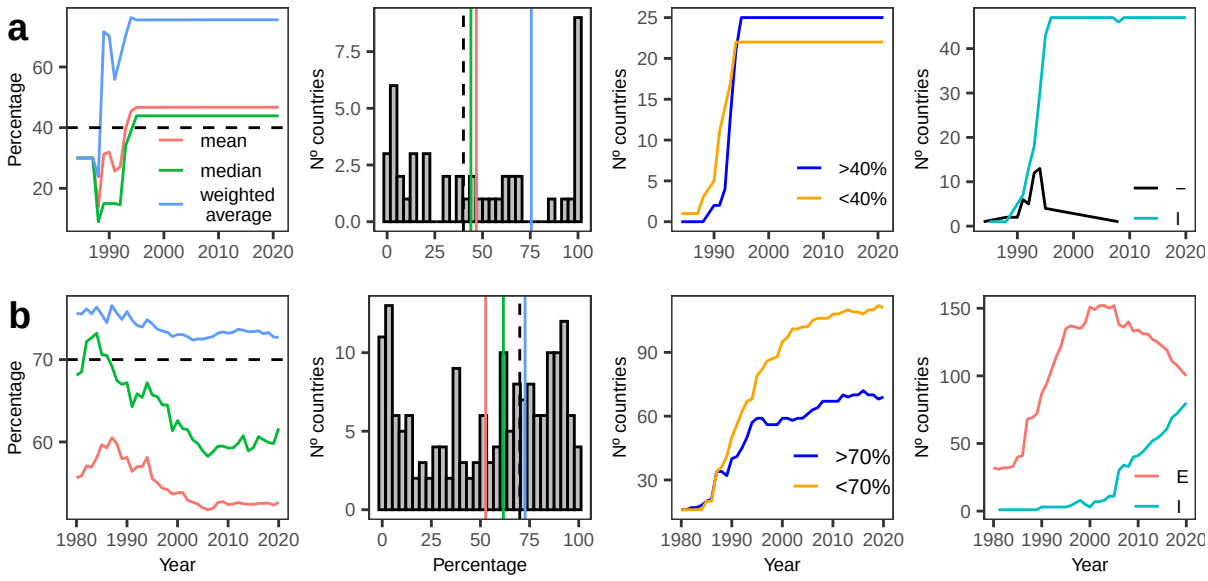
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
crop.plots.aquastat <- plot_grid(a.crop, crop.histogram, b.crop, fraction.estimate,
                                 labels = c("a", ""), ncol = 4)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
plot_grid(crop.plots.aquastat, water.plots.aquastat, ncol = 1)
```



6 Estimations compiled by Gleick 2000

```
# ANALYSIS OF PAST DATA NOT CITED ##############################################

# Read in data ----------------------------------------------------------------

sheets <- excel_sheets("Global_Projections_for_Water_Use.xlsx")
past.data <- lapply(sheets, function(sheet)
  data.table(read.xlsx("Global_Projections_for_Water_Use.xlsx", sheet = sheet)))
names(past.data) <- sheets

# Falkenmark and Lindh 1974 ---------------------------------------------------

past.data$`Falkenmark and Lindh 1974` %>%
  .[, fraction := withdrawal / .[Sector == "Total", withdrawal]] %>%
  .[, fraction2:= `withdrawal.(90%.industrial.reuse)` / .[Sector == "Total",
                                                          `withdrawal.(90%.industrial.reuse)`]]
  print()
```

```
## Index: <Sector>
##          Sector  year withdrawal withdrawal.(90%.industrial.reuse)   fraction
##          <char> <num>      <num>                              <num>      <num>
## 1:     Domestic  2015        890                                890 0.08210332
## 2:   Industrial  2015       4100                               1145 0.37822878
## 3: Agricultural  2015       5850                               5850 0.53966790
## 4:        Total    NA      10840                               7885 1.00000000
##    fraction2
##        <num>
## 1: 0.1128725
## 2: 0.1452124
## 3: 0.7419150
## 4: 1.0000000
```

```
# L'vovich 1974 ---------------------------------------------------------------

past.data$`L'vovich 1974` %>%
  .[sector == "Irrigated agriculture", fraction := withdrawals /
      .[sector == "TOTAL", withdrawals]] %>%
  print()
```

```
## Index: <sector>
##                         sector withdrawals consumption  fraction
##                         <char>       <num>       <num>     <num>
## 1: Residential/drinking water         920         180        NA
## 2:                  Livestock         150         100        NA
## 3:             Power industry        3100         270        NA
## 4:                 Industrial        3000         600        NA
## 5:      Irrigated agriculture        4400        4000 0.3445576
```

```
## 6:   Nonirrigated agriculture         700         700        NA
## 7:  Hydropower and navigation         500         500        NA
## 8: Fishery and sports fishing         175          85        NA
## 9:                       TOTAL       12770        6350        NA
```

```
# L'vovich 1974b ------------------------------------------------------------

past.data$`L'vovich 1974b` %>%
  .[sector == "Irrigated agriculture", fraction := withdrawals /
      .[sector == "TOTAL", withdrawals]] %>%
  print()
```

```
## Index: <sector>
##                         sector withdrawals consumption
##                         <char>       <num>       <num>
## 1:   Water supply (all types)        1500        1050
## 2:      Irrigated agriculture        3950        4000
## 3:   Nonirrigated agriculture         700         700
## 4:  Hydropower and navigation         500         500
## 5: Fishery and sports fishing         175          85
## 6:                       TOTAL        6825        6335
##    Discharge.of.Sewage.(km³/yr)  fraction
##                           <num>     <num>
## 1:                            0        NA
## 2:                          400 0.5787546
## 3:                            0        NA
## 4:                            0        NA
## 5:                           90        NA
## 6:                          490        NA
```

```
# Gleick 1997 ---------------------------------------------------------------

past.data$`gleick 1997`[, total:= agriculture + domestic + industrial] %>%
  .[, fraction.irrigation:= agriculture / total] %>%
  print()
```

```
##    agriculture domestic industrial reservoir total fraction.irrigation
##          <num>    <num>      <num>     <num> <num>               <num>
## 1:        2930     1000        340       225  4270           0.6861827
```

```
# Seckler et al 1998 --------------------------------------------------------

past.data$`seckler et al 1998`[, fraction.irrigation:= agricultural / total] %>%
  print()
```

```
##     year              Year/Scenario total agricultural domestic.industrial
##    <num>                     <char> <num>        <num>               <num>
## 1:  1990                   estimate  2907         2084                 823
## 2:  2025          business as usual  4569         3376                1193
## 3:  2025 high irrigation efficiency  3625         2432                1193
```

```
##    fraction.irrigation
##                  <num>
## 1:           0.7168903
## 2:           0.7388925
## 3:           0.6708966
```

```
# Falkenmark and Lindh 1974 -----------------------------------------------------

past.data$`Falkenmark and Lindh 1974` %>%
  .[, fraction := withdrawal / .[Sector == "Total", withdrawal]] %>%
  print()
```

```
## Index: <Sector>
##          Sector  year withdrawal withdrawal.(90%.industrial.reuse)   fraction
##          <char> <num>      <num>                              <num>      <num>
## 1:     Domestic  2015        890                                890 0.08210332
## 2:   Industrial  2015       4100                               1145 0.37822878
## 3: Agricultural  2015       5850                               5850 0.53966790
## 4:        Total    NA      10840                               7885 1.00000000
##    fraction2
##        <num>
## 1: 0.1128725
## 2: 0.1452124
## 3: 0.7419150
## 4: 1.0000000
```

7 Water belief

7.1 Uncertainty analysis

```
# DEFINE FUNCTIONS ###################################################################

# Transform columns to qunif with min and max reflecting uncertainty -----------

transform_columns_fun <- function(mat, distr) {

  for (i in 1:nrow(distr)) {

    country <- distr$Country[i]

    min_val <- distr$min[i]
    max_val <- distr$max[i]

    mat[[country]] <- qunif(mat[[country]], min = min_val, max = max_val)
  }

  return(mat)
}

## Function to transform longitude and latitude to country.
# It is borrowed from Andy:
# https://stackoverflow.com/questions/14334970/convert-latitude-and-longitude-coordinates-to-c

coords2country = function(points) {

  countriesSP <- rworldmap::getMap(resolution = 'low')
  pointsSP = sp::SpatialPoints(points, proj4string=CRS(proj4string(countriesSP)))
  indices = sp::over(pointsSP, countriesSP)
  indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  #indices$continent # returns the continent (6 continent model)
  #indices$REGION # returns the continent (7 continent model)

}

# Function to retrieve .nc data from Huang et al 2018 --------------------------

get_huang_fun <- function(nc_file, variable, year) {

  nc_data <- nc_open(nc_file)

  # Extract longitude, latitude, and water withdrawal data (withd_elec)
  lon <- ncvar_get(nc_data, "lon")
  lat <- ncvar_get(nc_data, "lat")
```

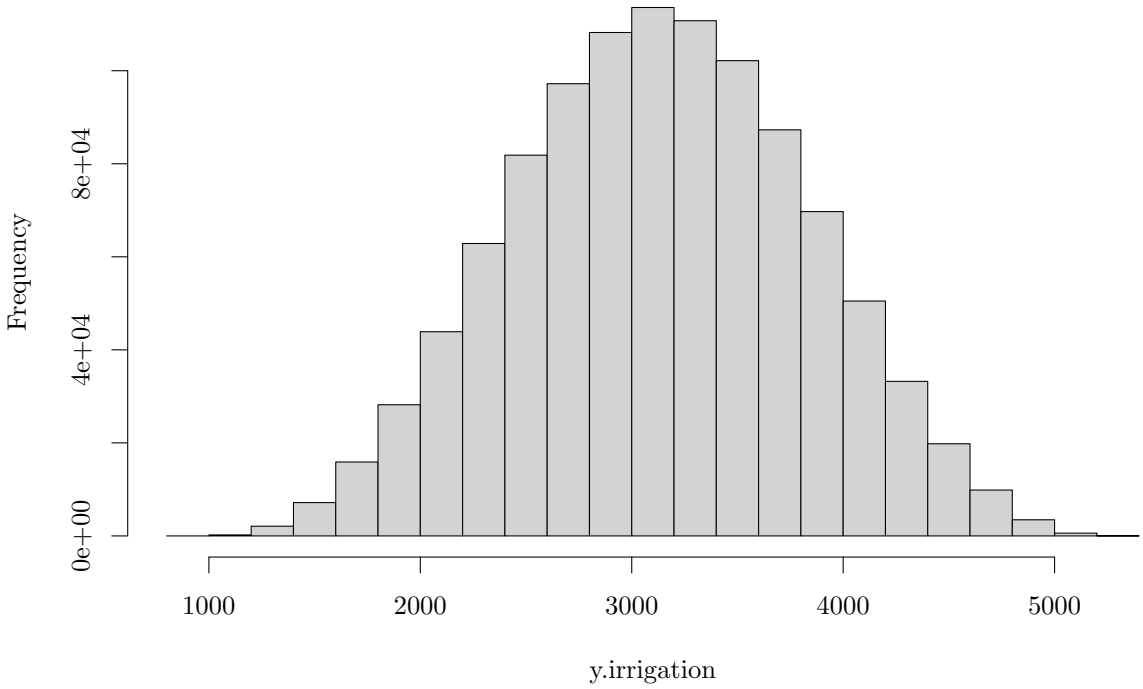
```r
withd_elec <- ncvar_get(nc_data, variable)
time <- ncvar_get(nc_data, "month")

# Close the NetCDF file after reading
nc_close(nc_data)

# Earth's radius in meters
earth_radius <- units::set_units(6371000, "m")

# Function to convert degrees to radians
deg_to_rad <- function(deg) {
  return(deg * pi / 180)
}

# Calculate the grid area (m²) for each grid cell
# Create a data.table with lon and lat values
dt <- data.table(lon = lon, lat = lat)

# Convert latitudes to radians for area calculation
dt[, lat_rad := deg_to_rad(lat)]

# Assume the grid cells have equal spacing (this should match your grid resolution)
# If you have grid cell dimensions, use them here. Otherwise, use a 0.5 degree grid as an ex
delta_lon <- deg_to_rad(0.5)  # Assuming 0.5-degree grid spacing in longitude
delta_lat <- deg_to_rad(0.5)  # Assuming 0.5-degree grid spacing in latitude

# Calculate the area of each grid cell (in square meters)
dt[, area_cell := as.numeric(earth_radius^2 * delta_lon * delta_lat * cos(lat_rad))]

# Generate the corresponding year and month for each time step
start_year <- 1971
start_month <- 1
n_months <- length(time)

# Generate year and month for each time step
years <- rep(start_year:(start_year + (n_months %/% 12)), each = 12, length.out = n_months)
months <- rep(1:12, length.out = n_months)

# Convert withdrawal from mm/month to km³/month
for (month_idx in 1:n_months) {
  dt[[paste0("year_", years[month_idx], "_month_", months[month_idx])]] <- as.numeric((withd
}

# Select only columns that contain 'year_2010'
columns_selected <- grep(year, colnames(dt), value = TRUE)

# Create a new data.table with only the relevant columns for 2010
```

```
  dt_2010 <- dt[, c("lon", "lat", columns_selected), with = FALSE]

  countries <- coords2country(dt_2010)

  dada <- cbind(countries, dt_2010) %>%
    melt(., measure.vars = columns_selected) %>%
    .[, sum(value, na.rm = TRUE), countries] %>%
    .[order(countries)] %>%
    .[, Country:= countrycode(countries, origin = "country.name", destination = "country.name")


  return(dada)

}

# Function to retrieve .nc4 files from ISIMIP ----------------------------------

get_isimip_fun <- function(nc_file, variable, year, start_year) {

  nc_data <- nc_open(nc_file)

  # Extract longitude, latitude, and water withdrawal data
  lon <- ncvar_get(nc_data, "lon")
  lat <- ncvar_get(nc_data, "lat")
  aindww <- ncvar_get(nc_data, variable)
  time <- ncvar_get(nc_data, "time")

  # Close the NetCDF file after reading
  nc_close(nc_data)

  # Create a grid of lon and lat coordinates
  lon_lat_grid <- expand.grid(lon = lon, lat = lat)

  # Earth's radius in meters
  earth_radius <- units::set_units(6371000, "m")

  # Function to convert degrees to radians
  deg_to_rad <- function(deg) {
    return(deg * pi / 180)
  }

  # Calculate the grid area (m²) for each grid cell
  lon_lat_grid$lat_rad <- deg_to_rad(lon_lat_grid$lat)

  # Assume the grid cells have equal spacing (match your grid resolution)
  delta_lon <- deg_to_rad(0.5)  # Assuming 0.5-degree grid spacing in longitude
  delta_lat <- deg_to_rad(0.5)  # Assuming 0.5-degree grid spacing in latitude
```

```r
  # Calculate the area of each grid cell (in square meters)
  lon_lat_grid$area_cell <- as.numeric(earth_radius^2 * delta_lon * delta_lat * cos(lon_lat_gri

  # Convert aindww from kg/m²/s to m³/month
  # 1 kg of water = 1 liter = 1e-3 m³
  # Multiply by the number of seconds in a month (30.44 days average)
  seconds_per_month <- 30.44 * 24 * 3600  # Average month duration in seconds

  # Generate the corresponding year and month for each time step
  start_year <- start_year
  n_months <- length(time)

  years <- rep(start_year:(start_year + (n_months %/% 12)), each = 12, length.out = n_months)
  months <- rep(1:12, length.out = n_months)

  # Convert withdrawal from kg/m²/s to km³/month for each grid cell
  results <- list()
  for (month_idx in 1:n_months) {
    aindww_month <- aindww[, , month_idx]  # Extract the 2D slice (lon x lat) for the given mo

    # Flatten the 2D data to match the grid
    aindww_month_flat <- as.numeric(aindww_month)

    # Calculate water withdrawal in km³/month for each grid cell
    lon_lat_grid[[paste0("year_", years[month_idx], "_month_", months[month_idx])]] <-
      aindww_month_flat * 1e-3 * lon_lat_grid$area_cell * seconds_per_month / 1e9  # Convert t
  }

  # Select only columns that contain the selected year
  columns_selected <- grep(year, colnames(lon_lat_grid), value = TRUE)

  # Create a new data table with only the relevant columns for the specified year
  dt_selected <- lon_lat_grid[, c("lon", "lat", columns_selected)] %>%
    na.omit()

  countries <- coords2country(dt_selected)

  result <- cbind(countries, dt_selected) %>%
    data.table() %>%
    melt(., measure.vars = columns_selected) %>%
    .[, sum(value, na.rm = TRUE), countries] %>%
    .[order(countries)] %>%
    .[, Country := countrycode(countries, origin = "country.name", destination = "country.name

  return(result)
}
```

```r
# DEFINE SETTINGS ######################################################################

N <- 2^20
type <- "QRN"
order <- "first"
matrices <- "A"
```

```r
# UNCERTAINTY IN IRRIGATION ############################################################

# Read dataset ---------------------------------------------------------------

gww <- fread("global_water_withdrawals.csv")
gww[, Country:= countrycode(Country, origin = "country.name", destination = "country.name")]

countries <- unique(gww$Country)

# Define sample matrix -------------------------------------------------------

params <- countries
mat <- data.table(sobol_matrices(matrices = matrices, N = N, params = params,
                                 order = order, type = type))


# Transform to appropriate distributions -------------------------------------

mat <- transform_columns_fun(mat = mat, distr = gww)

# Calculate output vector ----------------------------------------------------

y.irrigation <- rowSums(mat)
hist(y.irrigation)
```




```
# DATASETS WITH INDUSTRY AND DOMESTIC VALUES ###################################

# Aquastat dataset ------------------------------------------------------------

# https://data.apps.fao.org/aquastat/?lang=en&share=f-79eb812e-6cb9-406f-86f2-b7ee414c0ec6

aquastat.ind.dom <- fread("industrial_domestic_aquastat.csv") %>%
  .[, Country:= countrycode(Area, origin = "country.name", destination = "country.name")]
```

```
## Warning: Some values were not matched unambiguously: Australia and New Zealand, Central and
```

```
## Warning: Some strings were matched more than once, and therefore set to <NA> in the result:
```

```
aquastat.tmp <- aquastat.ind.dom[, .(Country, Variable, Value)] %>%
  .[Country %in% countries] %>%
  dcast(., Country ~ Variable, value.var = "Value") %>%
  setnames(., colnames(.)[2:3], c("Industry", "Domestic")) %>%
  .[order(Country)]

# Liu dataset -----------------------------------------------------------------

liu.dt <- data.table(read.xlsx("liu_dataset.xlsx")) %>%
  .[, Country:= countrycode(country, origin = "country.name", destination = "country.name")] %>
  .[Country %in% countries]

# Gleick dataset --------------------------------------------------------------
```

```
gleick.dt <- data.table(read.xlsx("gleick_table.xlsx"))
colnames_numeric <- colnames(gleick.dt)[-c(1,2)]
gleick.dt <- gleick.dt[, (colnames_numeric):= lapply(.SD, as.numeric), .SDcols = (colnames_nume
  .[, total.irrigation:= (agricultural / 100) * total.withdrawal] %>%
  .[, total.industry:= (industrial / 100) * total.withdrawal] %>%
  .[, total.domestic:= (domestic / 100) * total.withdrawal] %>%
  .[, Country:= countrycode(country, origin = "country.name", destination = "country.name")] %
  .[Country %in% countries]

# Folke dataset ------------------------------------------------------------

folke.dt <- data.table(read.xlsx("mmc4_country.xlsx")) %>%
  .[, .(`domestic.water.withdrawals.[mio.m3]`,
            `manufacturing.water.withdrawal.[mio.m3]`,
            `electricity.water.withdrawals.[mio.m3]`,
            Name, year)] %>%
  .[, folke.domestic:= `domestic.water.withdrawals.[mio.m3]` / 1000] %>%
  .[, folke.industrial:= (`manufacturing.water.withdrawal.[mio.m3]`+
      `electricity.water.withdrawals.[mio.m3]`) / 1000] %>%
  .[year == 2010] %>%
  .[, .(Name, year, folke.domestic, folke.industrial)] %>%
  .[, Country:= countrycode(Name, origin = "country.name", destination = "country.name")]
```

```
## Warning: Some values were not matched unambiguously: CENTRAL AFRICAN, EQUATORIAL GUIN, Korea
```

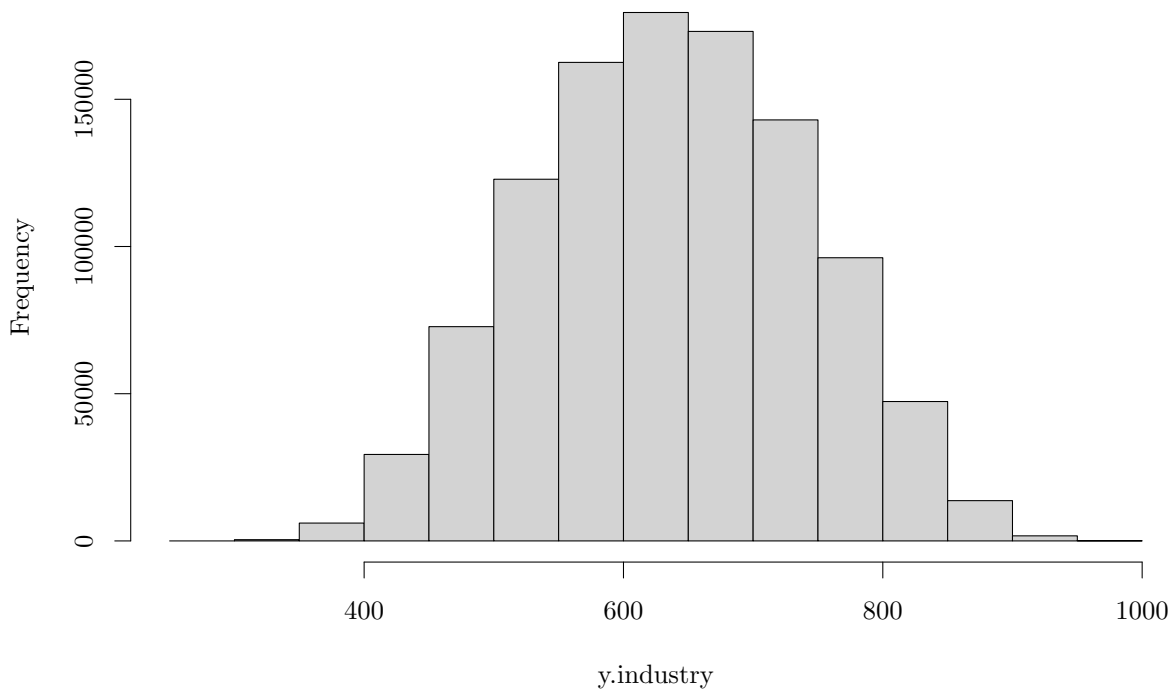
```
# INDUSTRY ###################################################################

# Khan et al 2023 dataset --------------------------------------------------

path.projections <- "./files/khan_et_al_2023/industry"
list.of.files <- list.files(path.projections, pattern = "\\.csv$")
combinations <- lapply(list.of.files, function(x) strsplit(x, "_")[[1]][1:4]) %>%
  do.call(rbind, .) %>%
  data.frame()
colnames(combinations) <- c("SSP", "RCP", "Climate", "Use")

# Create parallel cluster --------------------------------------------------

numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop -------------------------------------------------------------

result <- foreach(i = 1:length(list.of.files),
                  .combine = "rbind",
                  .packages = c("data.table", "countrycode", "tidyverse",
                                "sp", "rworldmap")) %dopar% {
```

```
                                    out <- fread(paste("./files/khan_et_al_2023/industry",
                                                       list.of.files[i], sep = "/"))

                                    out <- out[, `:=`(SSP = combinations[i, 1],
                                                      RCP = combinations[i, 2],
                                                      Climate = combinations[i, 3],
                                                      Use = combinations[i, 4])]

                                    Country <- coords2country(out[1:nrow(out), 2:3])

                                    df <- cbind(Country, out) %>%
                                      .[, Continent := countrycode(Country, origin = "country.nam
                                                                   destination = "continent")] %>
                                      .[, Country:= countrycode(Country, origin = "country.name"
                                                                destination = "country.name")] %>
                                      .[, Dataset:= list.of.files[i]]

                                    df
                                  }

# Stop the cluster after the computation ---------------------------------------

stopCluster(cl)

# Arrange Khan data ------------------------------------------------------------

khan.industry <- result[, sum(`2010`), .(Country, Dataset)] %>%
  .[Country %in% countries] %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[order(Country)]

# Huang et al 2018 dataset -----------------------------------------------------

path.projections <- "./files/huang_et_al_2018/industrial"
list.of.files <- list.files(path.projections, pattern = "\\.nc$")
file <- paste("./files/huang_et_al_2018/industrial",
              list.of.files, sep = "/")
variables <- sub("\\..*", "", list.of.files)

# Retrieve data ----------------------------------------------------------------

out <- list()

for (i in 1:length(file)) {

  out[[i]] <- get_huang_fun(nc_file = file[[i]], variable = variables[[i]], year = "year_2010")
}
```

```
## Warning: Some values were not matched unambiguously: Siachen Glacier

## Warning: Some values were not matched unambiguously: Siachen Glacier

## Warning: Some values were not matched unambiguously: Siachen Glacier
```

```
huang.industry <- data.table(Country = out[[1]]$Country,
                             industrial.huang = out[[1]]$V1 +
                               out[[2]]$V1 +
                               out[[3]]$V1) %>%
  .[Country %in% countries]

# ISIMIP: Create vector with list of files -------------------------------------

path <- "./files/isimip/industrial"
list.of.files <- list.files(path)
files.directory <- paste(path, list.of.files, sep = "/")
variable <- "aindww"
year <- "year_2010"
names.models <- lapply(list.of.files, function(x) sub("_.*", "", x))
start_year <- 1971

# Create parallel cluster ------------------------------------------------------

numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop -----------------------------------------------------------------

result <- foreach(i = 1:length(files.directory),
                  .packages = c("data.table", "countrycode", "tidyverse",
                                "sp", "rworldmap", "ncdf4")) %dopar% {

                  start_year <- ifelse(names.models[i] == "cwatm" | names.models[i] == "h08",
                                       ifelse(names.models[i] == "miroc-integ-land", 1901, 197

                      get_isimip_fun(nc_file = files.directory[i],
                                     variable = variable,
                                     year = year,
                                     start_year = start_year)
                                     }

# Stop the cluster after the computation ---------------------------------------

stopCluster(cl)
```

```r
# Arrange data ------------------------------------------------------------

names(result) <- list.of.files

isimip.industry <- rbindlist(result, idcol = "ID") %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[Country %in% countries]


# Merge all industry datasets ---------------------------------------------

industry.sample.matrix <- merge(khan.industry, aquastat.tmp[, .(Country, Industry)],
                                by = "Country") %>%
  merge(., liu.dt[, .(Country, ind)], by = "Country") %>%
  merge(., isimip.industry[, .(Country, min, max)], by = "Country") %>%
  merge(., huang.industry[, .(Country, industrial.huang)], by = "Country") %>%
  merge(., gleick.dt[, .(Country, total.industry)], by = "Country") %>%
  merge(., folke.dt[, .(Country, folke.industrial)], by = "Country") %>%
  melt(., measure.vars = colnames(.)[-1]) %>%
  .[, .(min = min(value), max = max(value)), Country]

# Define sample matrix ----------------------------------------------------

params <- industry.sample.matrix$Country

mat <- data.table(sobol_matrices(matrices = matrices, N = N, params = params,
                                 order = order, type = type))

# Transform to appropriate distributions ----------------------------------

mat <- transform_columns_fun(mat = mat, distr = industry.sample.matrix)

# Calculate output vector -------------------------------------------------

y.industry <- rowSums(mat, na.rm = TRUE)
hist(y.industry)
```




```r
max(y.industry)
```

```
## [1] 990.9168
```

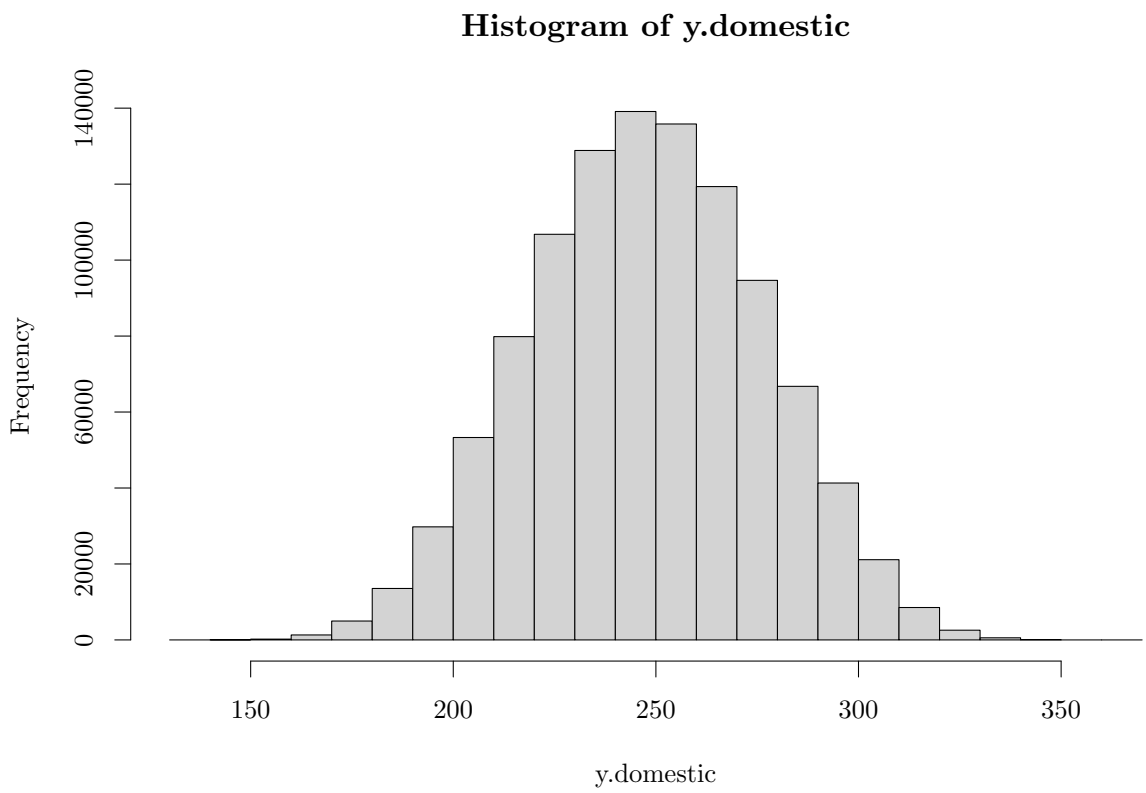
```r
# DOMESTIC ###################################################################

# Khan et al 2023 dataset ----------------------------------------------------

path.projections <- "./files/khan_et_al_2023/domestic"
list.of.files <- list.files(path.projections, pattern = "\\.csv$")
combinations <- lapply(list.of.files, function(x) strsplit(x, "_")[[1]][1:4]) %>%
  do.call(rbind, .) %>%
  data.frame()
colnames(combinations) <- c("SSP", "RCP", "Climate", "Use")

# Create parallel cluster ----------------------------------------------------

numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop ---------------------------------------------------------------

result <- foreach(i = 1:length(list.of.files),
                  .combine = "rbind",
                  .packages = c("data.table", "countrycode", "tidyverse",
```

```
                                "sp", "rworldmap")) %dopar% {

                                  out <- fread(paste("./files/khan_et_al_2023/domestic",
                                                     list.of.files[i], sep = "/"))

                                  out <- out[, `:=`(SSP = combinations[i, 1],
                                                    RCP = combinations[i, 2],
                                                    Climate = combinations[i, 3],
                                                    Use = combinations[i, 4])]

                                  Country <- coords2country(out[1:nrow(out), 2:3])

                                  df <- cbind(Country, out) %>%
                                    .[, Continent := countrycode(Country, origin = "country.nam
                                                                 destination = "continent")] %>
                                    .[, Country:= countrycode(Country, origin = "country.name"
                                                              destination = "country.name")] %>
                                    .[, Dataset := list.of.files[i]]

                                  df
                                }

# Stop the cluster after the computation -------------------------------------

stopCluster(cl)

# Arrange Khan data ----------------------------------------------------------

khan.domestic <- result[, sum(`2010`), .(Country, Dataset)] %>%
  .[Country %in% countries] %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[order(Country)]

# Huang et al 2018 dataset ---------------------------------------------------

path.projections <- "./files/huang_et_al_2018/domestic"
list.of.files <- list.files(path.projections, pattern = "\\.nc$")
file <- paste(path.projections,
              list.of.files, sep = "/")
variables <- sub("\\..*", "", list.of.files)

# Retrieve data --------------------------------------------------------------

huang.domestic <- get_huang_fun(nc_file = file, variable = variables, year = "year_2010") %>%
  .[Country %in% countries]
```

```
## Warning: Some values were not matched unambiguously: Siachen Glacier
```

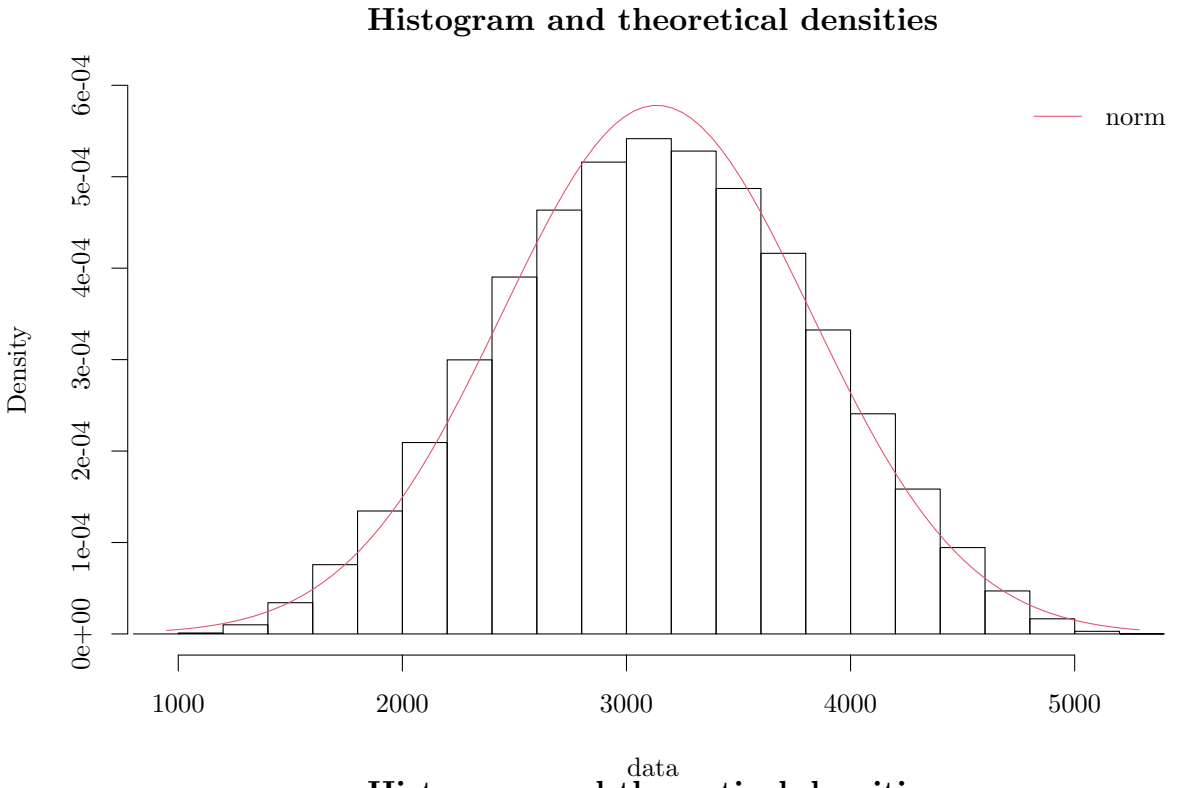
```
# ISIMIP: Create vector with list of files -----------------------------------

path <- "./files/isimip/domestic"
list.of.files <- list.files(path)
files.directory <- paste(path, list.of.files, sep = "/")
variable <- "adomww"
year <- "year_2010"
names.models <- lapply(list.of.files, function(x) sub("_.*", "", x))
start_year <- 1971

# Create parallel cluster ----------------------------------------------------

numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop ---------------------------------------------------------------

result <- foreach(i = 1:length(files.directory),
                  .packages = c("data.table", "countrycode", "tidyverse",
                                "sp", "rworldmap", "ncdf4")) %dopar% {

                                  start_year <- ifelse(names.models[i] == "miroc-integ-land" |
                                                         names.models[i] == "cwatm", 1901, 197

                                  get_isimip_fun(nc_file = files.directory[i],
                                                 variable = variable,
                                                 year = year,
                                                 start_year = start_year)
                                }

# Stop the cluster after the computation -------------------------------------

stopCluster(cl)

# Arrange data ---------------------------------------------------------------

names(result) <- list.of.files

isimip.domestic <- rbindlist(result, idcol = "ID") %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[Country %in% countries]


# Merge all domestic datasets ------------------------------------------------

domestic.sample.matrix <- merge(khan.domestic,
```

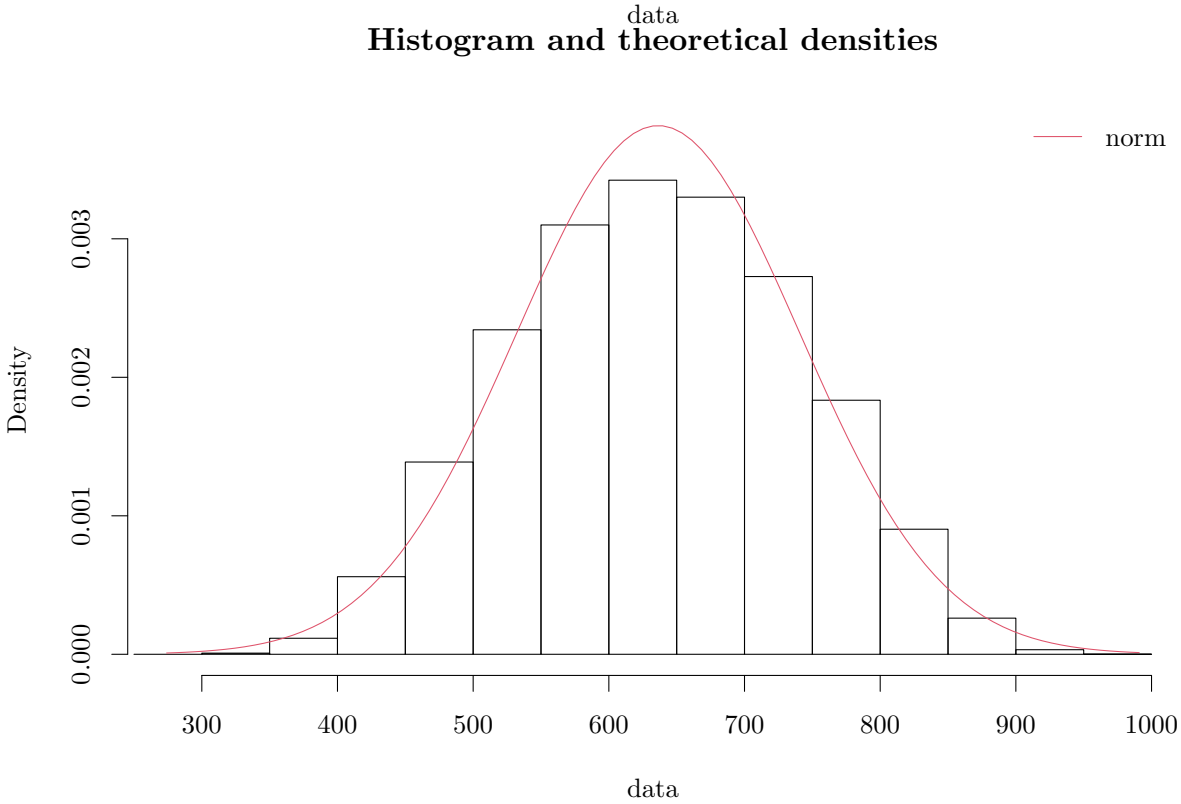
```
                                aquastat.tmp[, .(Country, Domestic)],
                                by = "Country") %>%
  merge(., huang.domestic[, .(Country, V1)], by = "Country") %>%
  merge(., liu.dt[, .(Country, muni)], by = "Country") %>%
  merge(., isimip.domestic[, .(Country, min, max)], by = "Country") %>%
  merge(., gleick.dt[, .(Country, total.domestic)], by = "Country") %>%
  merge(., folke.dt[, .(Country, folke.domestic)], by = "Country") %>%
  melt(., measure.vars = colnames(.)[-1]) %>%
  .[, .(min = min(value), max = max(value)), Country] %>%
  na.omit()

# Define sample matrix ----------------------------------------------------

params <- domestic.sample.matrix$Country

mat <- data.table(sobol_matrices(matrices = matrices, N = N, params = params,
                                 order = order, type = type))

# Transform to appropriate distributions ----------------------------------

mat <- transform_columns_fun(mat = mat, distr = domestic.sample.matrix)

# Calculate output vector -------------------------------------------------

y.domestic<- rowSums(mat, na.rm = TRUE)
hist(y.domestic)
```

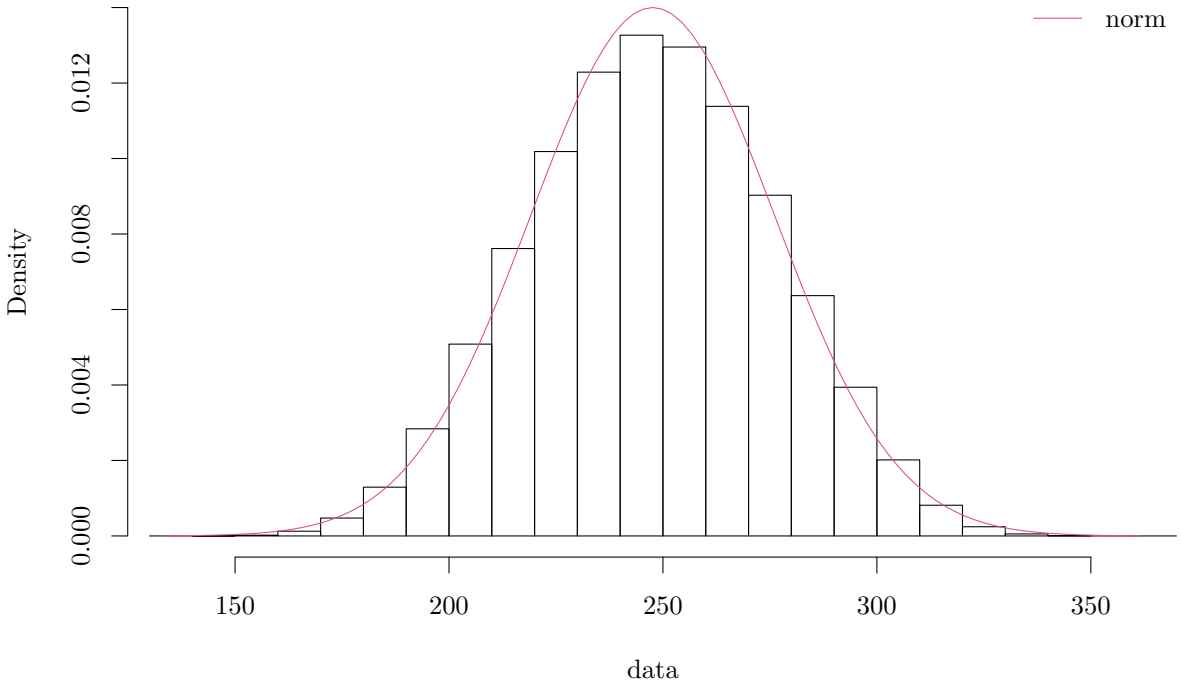



```
# CREATE WATER DT ###################################################################

water.dt <- data.table(irrigation = y.irrigation,
                       industry = y.industry,
                       domestic = y.domestic)

fwrite(water.dt, "water.dt.csv")
```

```
# UNCERTAINTY ANALYSIS ##############################################################

# Obtain normal distributions ---------------------------------------------------

selected_cols <- colnames(water.dt[, 1:3])
dist.estimates <- lapply(selected_cols, function(x) fitdist(water.dt[[x]], "norm"))
names(dist.estimates) <- selected_cols
lapply(dist.estimates, function(x) denscomp(x))
```


Histogram and theoretical densities
norm
Density
0e+00
1e-04
2e-04
3e-04
4e-04
5e-04
6e-04
1000
2000
3000
4000
5000
data


Histogram and theoretical densities
norm
Density
0.000
0.001
0.002
0.003
300
400
500
600
700
800
900
1000
data




```
## $irrigation
## NULL
##
## $industry
## NULL
##
## $domestic
## NULL
```

```
# Define function ----------------------------------------------------------

out <- lapply(selected_cols, function(x) {

  data.table(mean.value = dist.estimates[[x]]$estimate[["mean"]],
             sd.value = dist.estimates[[x]]$estimate[["sd"]])

})

names(out) <- selected_cols

for (i in names(out)) {

  out[[i]][, Fa.norm:= pnorm(min(water.dt[[i]]), mean = mean.value, sd = sd.value)]
  out[[i]][, Fb.norm:= pnorm(max(water.dt[[i]]), mean = mean.value, sd = sd.value)]
}
```

```r
# Define the sample matrix ----------------------------------------------------

params <- selected_cols
N <- 2^14
matrices <- c("A", "B", "AB")
order <- "second"
R <- 1000
first <- "jansen"
total <- "jansen"

mat <- sobol_matrices(matrices = matrices, params = params, order = order, N = N)

for (i in colnames(mat)) {

  mat[, i] <- qunif(mat[, i], out[[i]]$Fa.norm, out[[i]]$Fb.norm)
  mat[, i] <- qnorm(mat[, i], out[[i]]$mean.value, out[[i]]$sd.value)

}

# Run the computations --------------------------------------------------------

mat.dt <- data.table(mat)

mat.dt[, total:= rowSums(.SD)] %>%
  .[, percent.irrig:= (irrigation / total) * 100] %>%
  .[, percent.indus:= (industry / total) * 100] %>%
  .[, percent.domestic:= (domestic / total) * 100]


# STATISTICS ##################################################################

mat.dt[, .(min = min(percent.irrig), max = max(percent.irrig))]
```

```
##         min      max
##       <num>    <num>
## 1: 45.65791 90.08478
```

```r
mat.dt[, .(min = min(irrigation), max = max(irrigation))]
```

```
##         min      max
##       <num>    <num>
## 1: 963.4717 5274.615
```

```r
mat.dt[, .(min = min(total), max = max(total))]
```

```
##         min      max
##       <num>    <num>
## 1: 1725.261 6440.178
```

```r
quantile(mat.dt$percent.irrig, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 66.17393 84.76391
```

```r
quantile(mat.dt$irrigation, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 1791.329 4477.452
```

```r
quantile(mat.dt$total, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 2659.457 5379.444
```

```r
# Proportion of simulations around 70% ----------------------------------------

target.value <- 70  # target value
epsilon <- 5  # +-5%: tolerance around the target value

AB.mat <- mat.dt[1:(2 * N)]

# Calculate percentage of simulations within tolerance range
sum(AB.mat$percent.irrig >= (target.value - epsilon) &
      AB.mat$percent.irrig <= (target.value + epsilon)) /
  length(AB.mat$percent.irrig) * 100
```

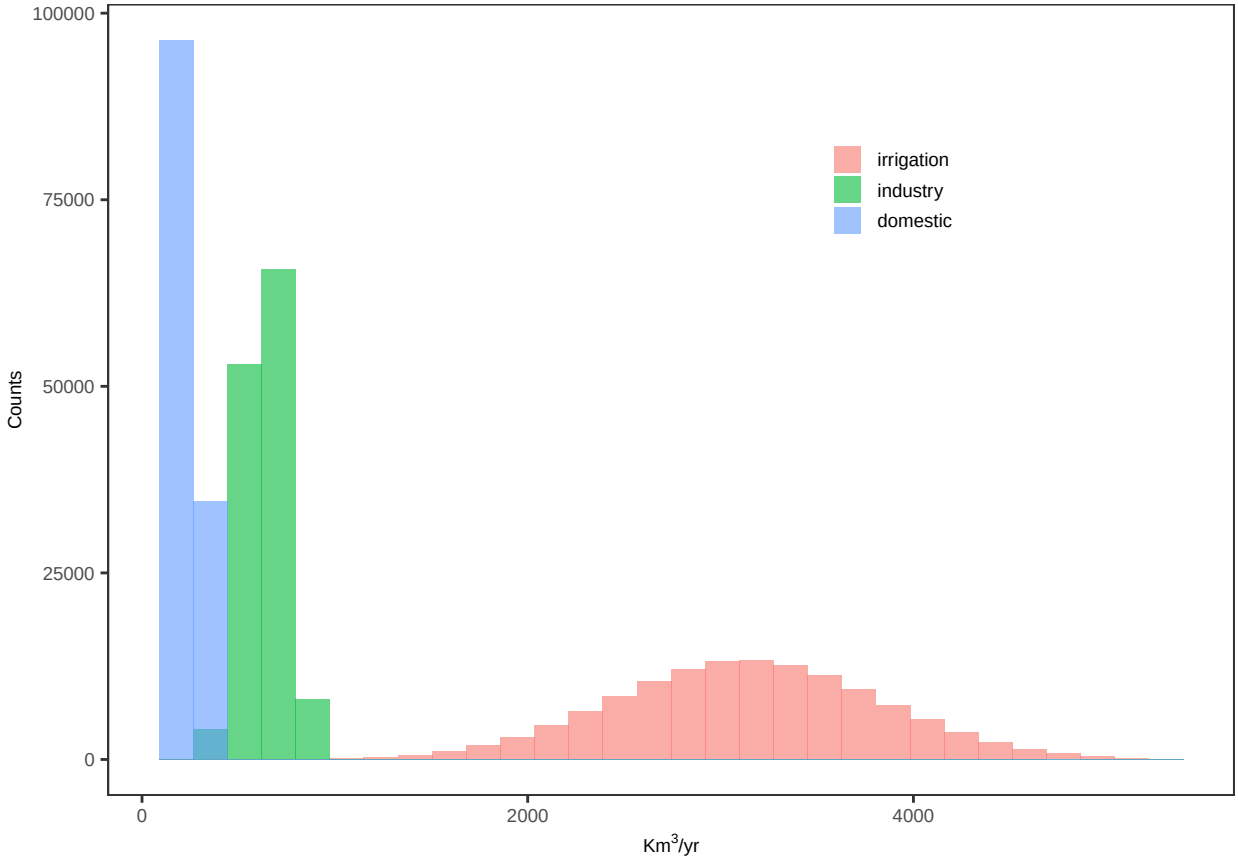
```
## [1] 24.42627
```

```r
# PLOT UNCERTAINTY ############################################################

vector.sectors <- c("irrigation", "industry", "domestic")

sectors.plot <- melt(mat.dt,
                     measure.vars = vector.sectors) %>%
  ggplot(., aes(value, fill = variable)) +
  geom_histogram(alpha = 0.6, position = "identity") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = bquote("Km"^3 * "/yr"), y = "Counts") +
  scale_fill_discrete(name = "") +
  theme_AP() +
  theme(legend.position = c(0.7, 0.8),
        legend.text = element_text(size = 7))
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

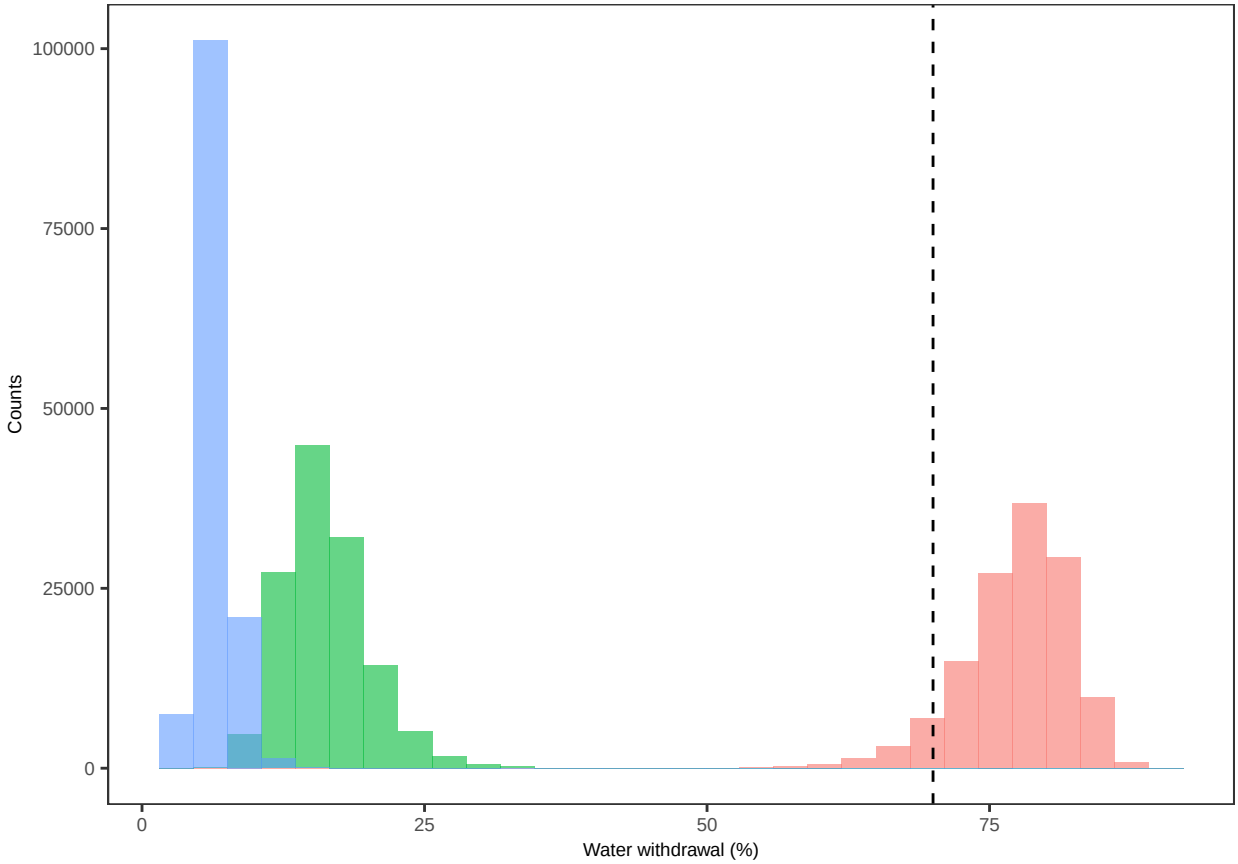
```
sectors.plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
total.plot <- mat.dt %>%
  ggplot(., aes(total)) +
  geom_histogram(color = "black", fill = "grey") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = bquote("Km"^3 * "/yr"), y = "Counts") +
  theme_AP()

percentages.plot <- mat.dt[, .(percent.irrig, percent.indus, percent.domestic)] %>%
  setnames(., colnames(.), vector.sectors) %>%
  melt(., measure.vars = vector.sectors) %>%
  ggplot(., aes(value, fill = variable)) +
  labs(x = "Water withdrawal (%)", y = "Counts") +
  scale_fill_discrete(name = "") +
  geom_histogram(alpha = 0.6, position = "identity") +
  geom_vline(xintercept = 70, lty = 2) +
  theme_AP() +
  theme(legend.position = "none")

percentages.plot
```

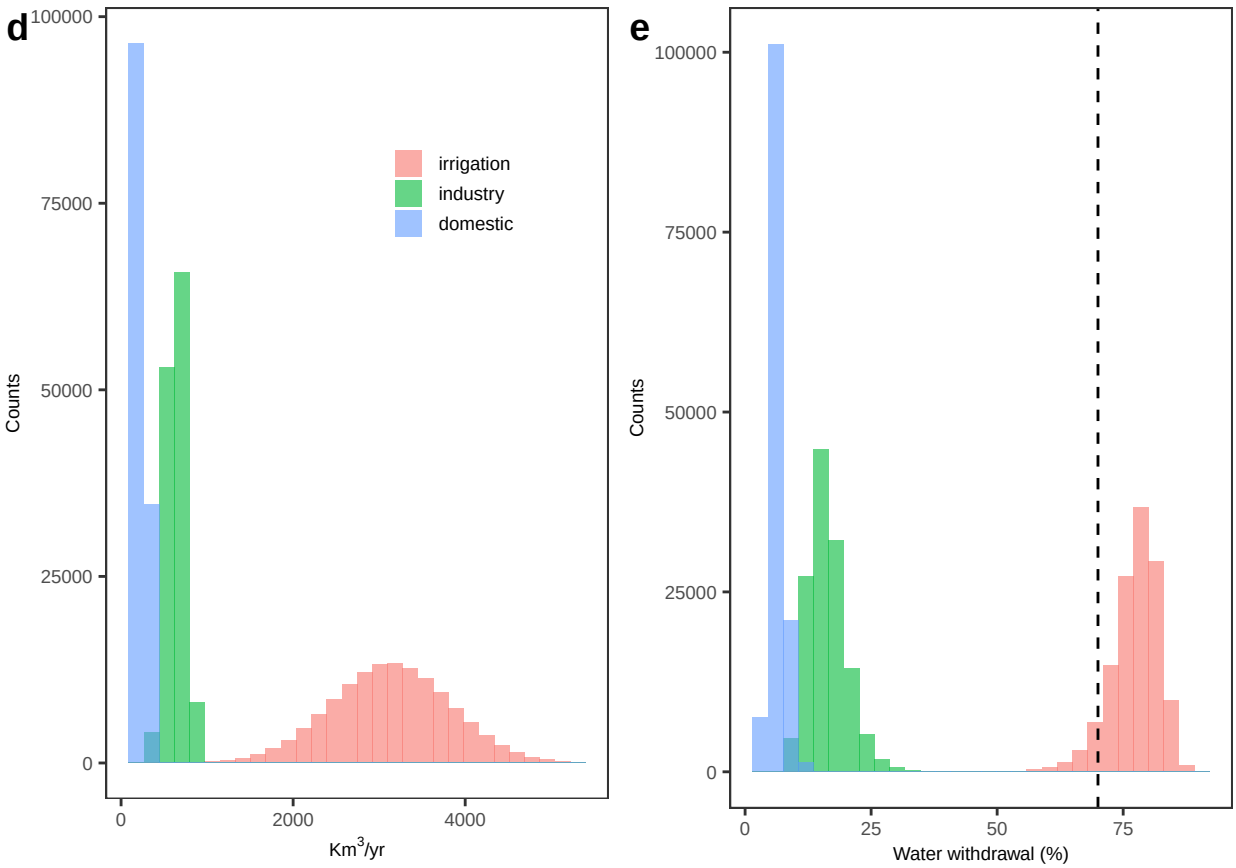
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
bottom <- plot_grid(sectors.plot, percentages.plot, ncol = 2, labels = c("d", "e"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
bottom
```



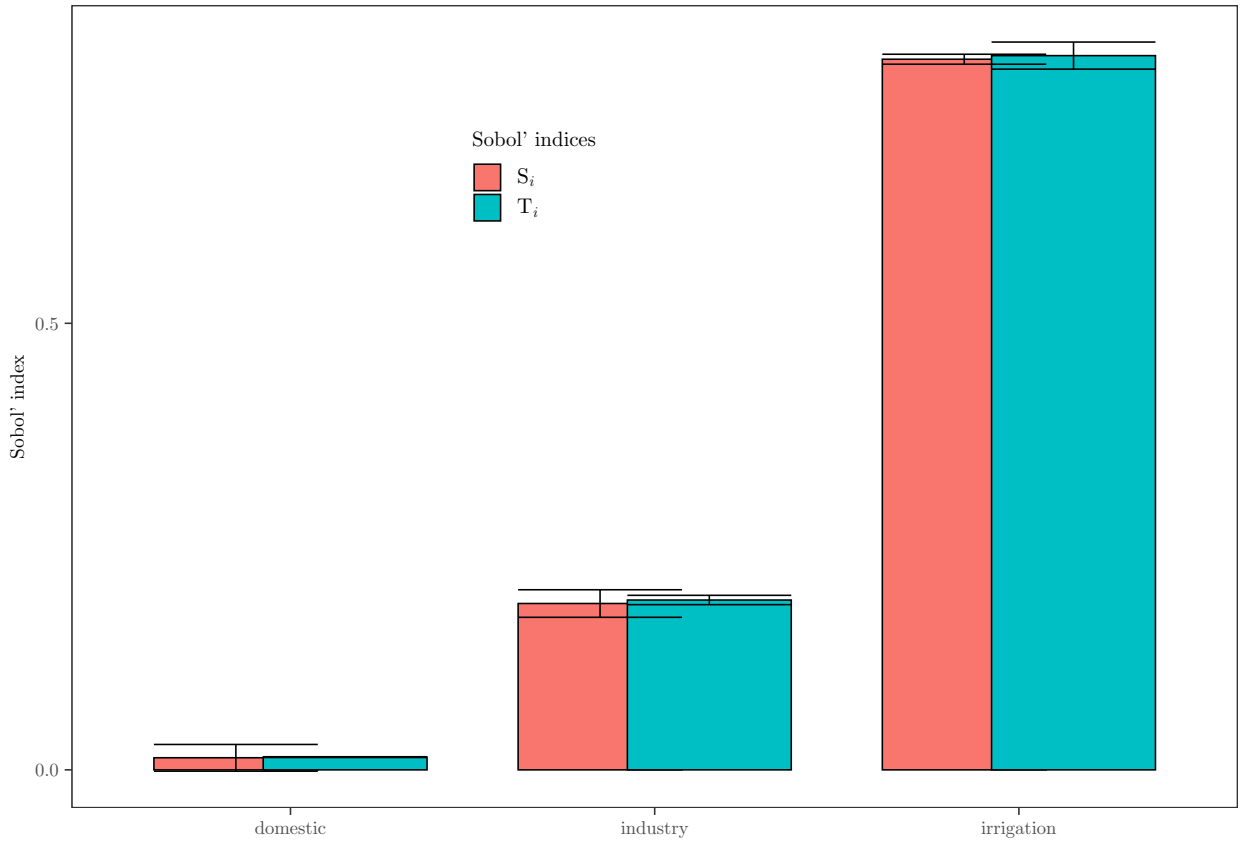
7.2 Sensitivity analysis

```
# SENSITIVITY ANALYSIS ############################################################

ind <- sobol_indices(matrices = matrices, Y = mat.dt$percent.irrig, N = N, order = order,
                     params = params, boot = TRUE, R = R, first = first, total = total)

print(ind)
```

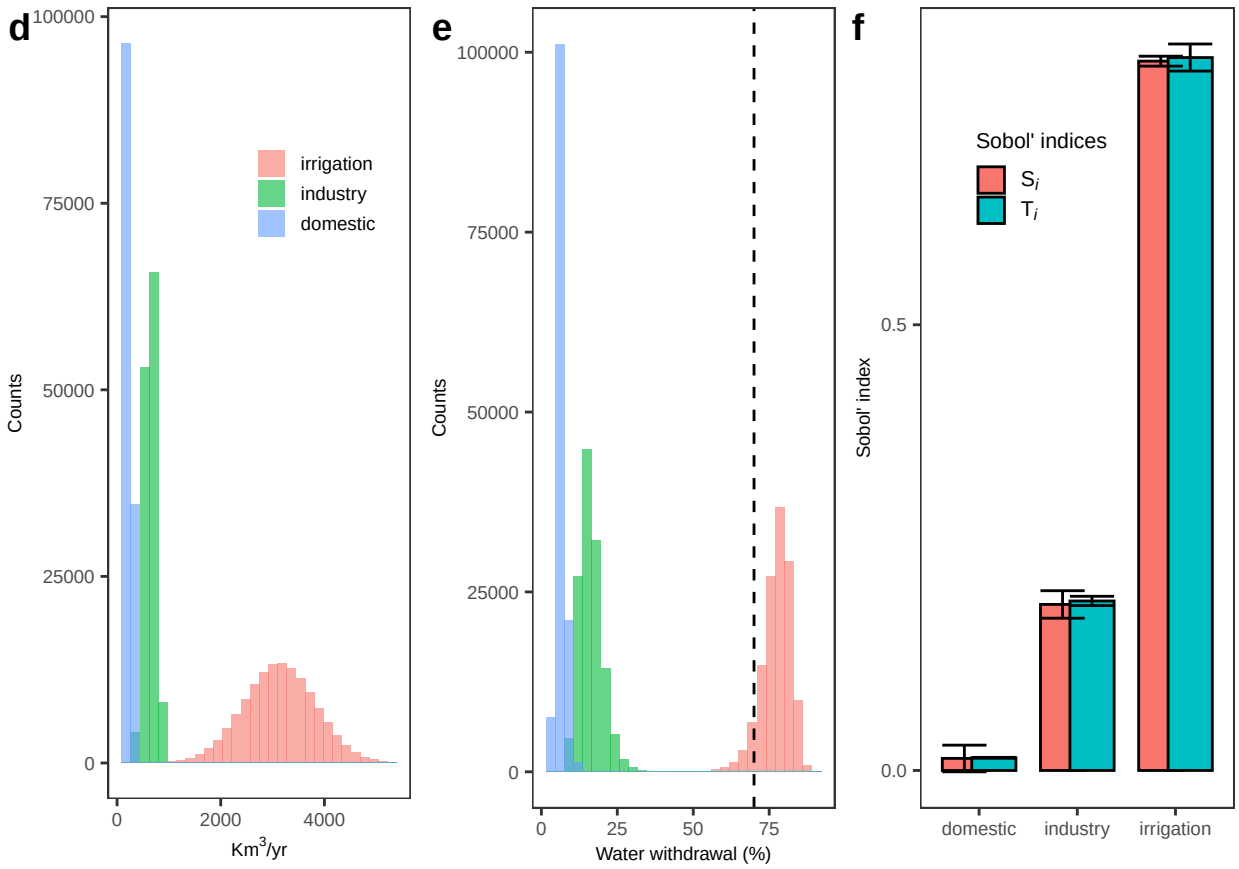
```
##
## First-order estimator: jansen | Total-order estimator: jansen
##
## Total number of model runs: 131072
##
## Sum of first order indices: 0.9955136
##        original          bias    std.error      low.ci    high.ci sensitivity
##           <num>         <num>        <num>       <num>      <num>      <char>
## 1: 0.7956889835  1.595281e-05 0.0028447592  0.79009741 0.80124866          Si
## 2: 0.1862781508  4.490900e-05 0.0078675151  0.17081320 0.20165329          Si
## 3: 0.0135464442  8.151473e-05 0.0076328032 -0.00149509 0.02842495          Si
## 4: 0.7996856961 -5.081727e-05 0.0077411366  0.78456416 0.81490886          Ti
## 5: 0.1901475009 -2.477803e-05 0.0026618057  0.18495524 0.19538932          Ti
## 6: 0.0142458889 -9.113272e-06 0.0002236826  0.01381659 0.01469341          Ti
```

```
## 7: 0.0037857804 -5.452485e-05 0.0082506647 -0.01233070 0.02001131         Sij
## 8: 0.0006527422 -1.056630e-04 0.0076692298 -0.01427301 0.01578982         Sij
## 9: 0.0006030811 -1.416324e-05 0.0077018649 -0.01447813 0.01571262         Sij
##             parameters
##                 <char>
## 1:          irrigation
## 2:            industry
## 3:            domestic
## 4:          irrigation
## 5:            industry
## 6:            domestic
## 7: irrigation.industry
## 8: irrigation.domestic
## 9:   industry.domestic
```

```
plot.sobol <- plot(ind) +
  theme_AP() +
  theme(legend.position = c(0.4, 0.8))

plot.sobol
```




```
water.ua.sa.plot <- plot_grid(bottom, plot.sobol, ncol = 2, labels = c("", "f"),
                              rel_widths = c(0.68, 0.32))
water.ua.sa.plot
```



8 Food belief

8.1 Uncertainty analysis

```
# AVERAGE YIELD OF IRRIGATED AND RAINFED WHEAT (TONS / HA) ######################

# Data from Dadrasi et al 2023 ------------------------------------------------

dadrasi.irrigated <- fread("irrigated_wheat_data.csv")[!HC27 == "Sum"]$`Irrigated wheat yield

dadrasi.rainfed <- fread("rainfed_wheat_data.csv")[!HC27 == "Sum"]$`Rainfed wheat yield (t/ha)

# Data from Mueller et al 2012 ------------------------------------------------

mueller.irrigated <- data.table(read.xlsx("mueller_et_al_2012.xlsx")) %>%
  .[, .(`yield.(t/ha).75%`, `yield.(t/ha).90%`, `yield.(t/ha).100%`)] %>%
  melt(., measure.vars = colnames(.)) %>%
  .[, value]

mueller.rainfed <- data.table(read.xlsx("mueller_et_al_2012.xlsx")) %>%
  .[, .(`yield.(t/ha)`, `yield.(t/ha).50%`)] %>%
  melt(., measure.vars = colnames(.)) %>%
```

```
  .[, value]

# Data from GAEZ ------------------------------------------------------------

gaez.irrigated <- data.table(read.xlsx("gaez_wheat_data.xlsx"))[, `Yield.(Irrigated).(tons/ha)`]

gaez.rainfed <- data.table(read.xlsx("gaez_wheat_data.xlsx"))[, `Yield.(tons/ha)`]

# Data from SPAM 2024 -------------------------------------------------------

spam.rainfed <- fread("spam2020V1r0_global_Y_TR.csv")[, .(ADM0_NAME, WHEA_R, unit)] %>%
  .[, Region:= countrycode(ADM0_NAME, origin = "country.name", destination = "region")] %>%
  .[, .(mean = mean(WHEA_R) / 1000), Region] %>%
  .[, mean]

spam.irrigated <- fread("spam2020V1r0_global_Y_TI.csv")[, .(ADM0_NAME, WHEA_I, unit)] %>%
  .[, Region:= countrycode(ADM0_NAME, origin = "country.name", destination = "region")] %>%
  .[, .(mean = mean(WHEA_I) / 1000), Region] %>%
  .[, mean]

# Data from Doll and Siebert 2010 -------------------------------------------

doll.rainfed <- 2.49
doll.irrigated <- 3.23

# Merge data and create irrigated and rainfed vector ------------------------

wheat.dt <- data.table(Y_grain_irr = c(dadrasi.irrigated, mueller.irrigated,
                                       gaez.irrigated, spam.irrigated, doll.irrigated),
                       Y_grain_non_irr = c(dadrasi.rainfed, mueller.rainfed,
                                           gaez.rainfed, spam.rainfed, doll.rainfed))
```

```
## Warning: Item 1 has 72 rows but longest item has 73; recycled with remainder.
```

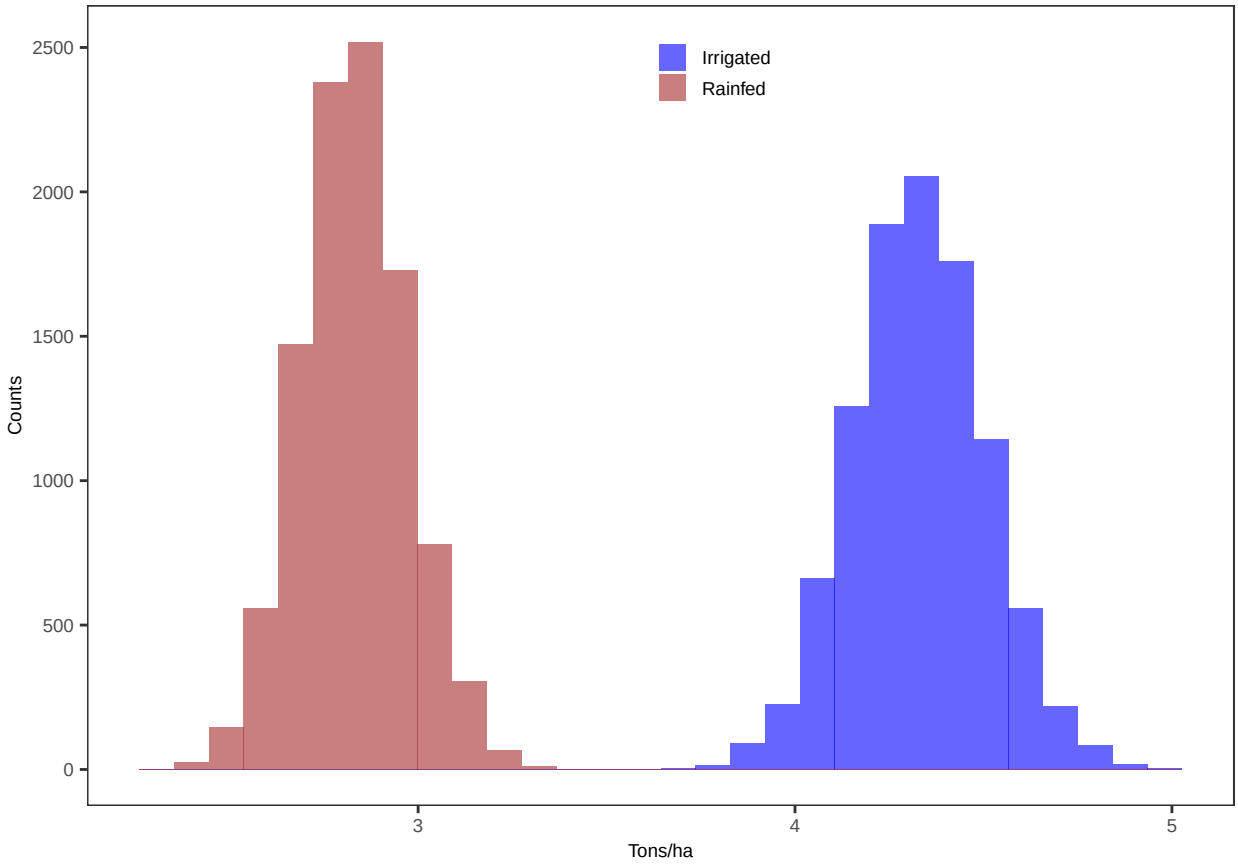
```
# Bootstrap wheat production ------------------------------------------------

bootstrap.wheat.dt <- replicate(10^4, {

  wheat.dt[sample(.N, replace = TRUE), .(Y_grain_irr = mean(Y_grain_irr),
                                         Y_grain_non_irr = mean(Y_grain_non_irr))]
}, simplify = FALSE) %>%
  rbindlist(.)

plot.bootstrap.wheat <- bootstrap.wheat.dt %>%
  melt(., measure.vars = colnames(.)) %>%
  ggplot(., aes(value, fill = variable)) +
  geom_histogram(alpha = 0.6, position = "identity") +
  scale_fill_manual(name = "", labels = c("Irrigated", "Rainfed"),
```

```
                    values = c("blue", "brown")) +
  labs(x = "Tons/ha", y = "Counts") +
  theme_AP() +
  theme(legend.position = c(0.55, 0.95),
        legend.text = element_text(size = 7))
```

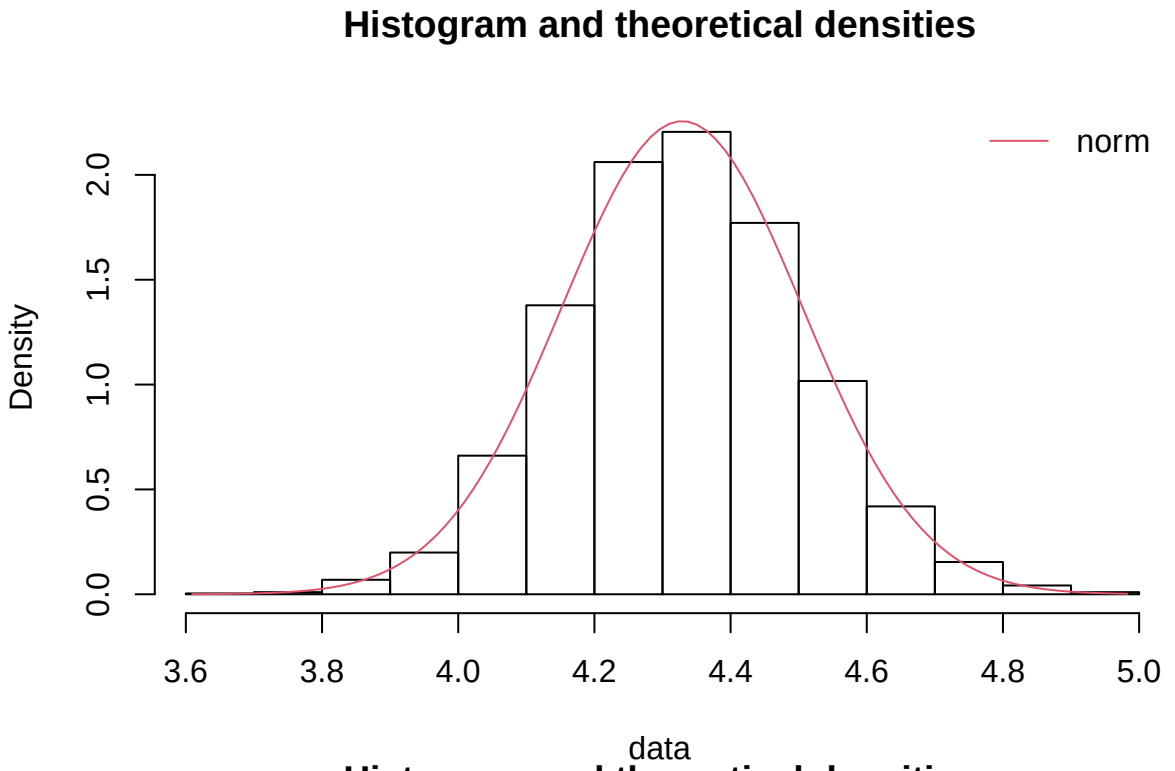
```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

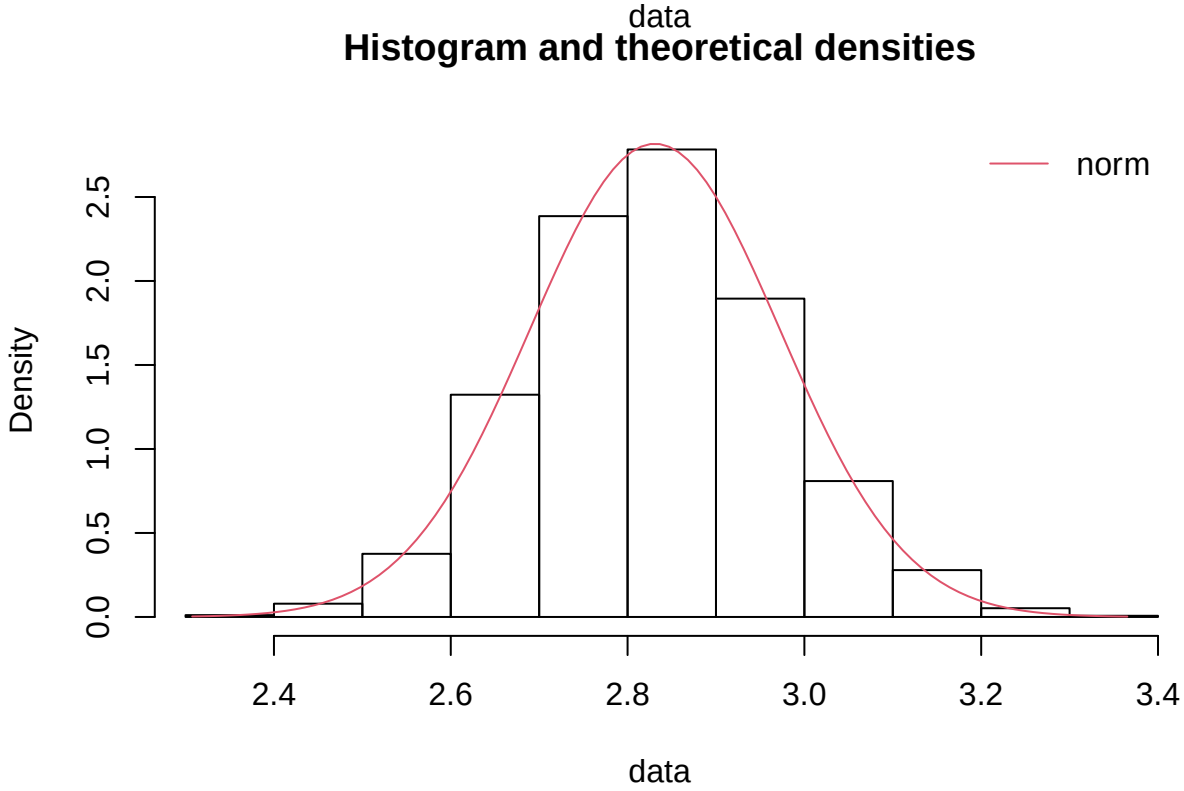
```
plot.bootstrap.wheat
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Describe with probability distribution ---------------------------------------

selected_cols <- colnames(wheat.dt)
dist.estimates <- lapply(selected_cols, function(x) fitdist(bootstrap.wheat.dt[[x]], "norm"))
names(dist.estimates) <- selected_cols
lapply(dist.estimates, function(x) denscomp(x))
```






```
## $Y_grain_irr
## NULL
##
## $Y_grain_non_irr
## NULL
```

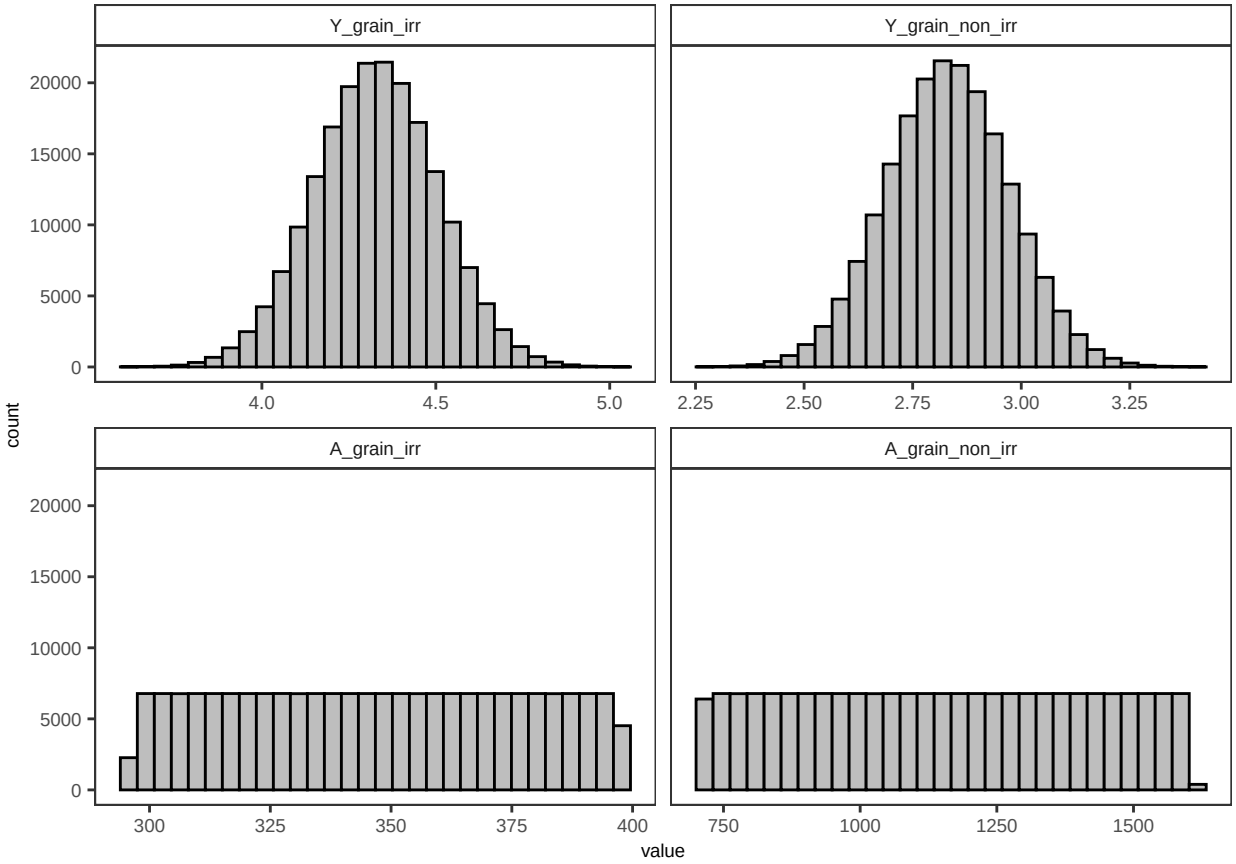
```r
max(bootstrap.wheat.dt$Y_grain_non_irr)
```

```
## [1] 3.365333
```

```r
# Define function ------------------------------------------------------------

out <- lapply(selected_cols, function(x) {

  data.table(mean.value = dist.estimates[[x]]$estimate[["mean"]],
             sd.value = dist.estimates[[x]]$estimate[["sd"]])

})

names(out) <- selected_cols

for (i in names(out)) {

  out[[i]][, Fa.norm:= pnorm(min(wheat.dt[[i]]), mean = mean.value, sd = sd.value)]
  out[[i]][, Fb.norm:= pnorm(max(wheat.dt[[i]]), mean = mean.value, sd = sd.value)]
}

# DATA ON THE EXTENSION OF IRRIGATED AREAS ######################################

meier <- fread("meier.csv")
col_maps <- colnames(meier)[-c(1:3)]
irrigated.area.dt <- melt(meier, measure.vars = col_maps) %>%
  .[, sum(value, na.rm = TRUE) / 10^6, variable]

range.irrigated.areas <- c(min(irrigated.area.dt$V1), max(irrigated.area.dt$V1))

# DATA ON THE EXTENSION OF TOTAL CROPLAND ######################################

tubiello.range <- c(1100, 1900)
potapov.range <- c(1181.5, 1306.9)

min.total.cropland <- min(c(tubiello.range, potapov.range))
max.total.cropland <- max(c(tubiello.range, potapov.range))

range.total.cropland <- c(min.total.cropland, max.total.cropland)

# DATA ON EXTENSION RAINFED AREAS ##############################################

rainfed.areas <- range.total.cropland - rev(range.irrigated.areas)

# DEFINE SAMPLE MATRIX -------------------------------------------------------

# Set parameters of calculation ----------------------------------------------
```

```r
N <- 2^15
matrices <- c("A", "B", "AB")
order = "first"
first <- "jansen"
total = "jansen"
R <- 10^3
params <- c("Y_grain_irr", "Y_grain_non_irr", "A_grain_irr", "A_grain_non_irr")

# create sample matrix and transform ------------------------------------------

mat <- sobol_matrices(matrices = matrices, N = N, order = order, params = params)

for (i in colnames(wheat.dt)) {

  mat[, i] <- qunif(mat[, i], out[[i]]$Fa.norm, out[[i]]$Fb.norm)
  mat[, i] <- qnorm(mat[, i], out[[i]]$mean.value, out[[i]]$sd.value)

}

mat[, "A_grain_irr"] <- qunif(mat[, "A_grain_irr"], min(irrigated.area.dt$V1), max(irrigated.ar
mat[, "A_grain_non_irr"] <- qunif(mat[, "A_grain_non_irr"], rainfed.areas[[1]], rainfed.areas[

# Plot uncertain distributions ------------------------------------------------

data.table(mat) %>%
  melt(., measure.vars = colnames(.)) %>%
  ggplot(., aes(value)) +
  geom_histogram(color = "black", fill = "grey") +
  facet_wrap(~variable, scales = "free_x") +
  theme_AP()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# RUN CALCULATIONS ##################################################################

production.irrigation <- mat[, "Y_grain_irr"] * mat[, "A_grain_irr"]
production.non.irrigation <- mat[, "Y_grain_non_irr"] * mat[, "A_grain_non_irr"]
Y <- (production.irrigation / (production.irrigation + production.non.irrigation)) * 100
```

```
# UNCERTAINTY ANALYSIS ##############################################################

final.dt <- cbind(mat, Y) %>%
  data.table()
AB.mat <- final.dt[1:(2 * N)]

AB.mat[, .(min = min(Y), max = max(Y))]
```

```
##         min      max
##       <num>    <num>
## 1: 18.47774 49.66922
```

```
quantile(Y, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 23.35906 43.74609
```

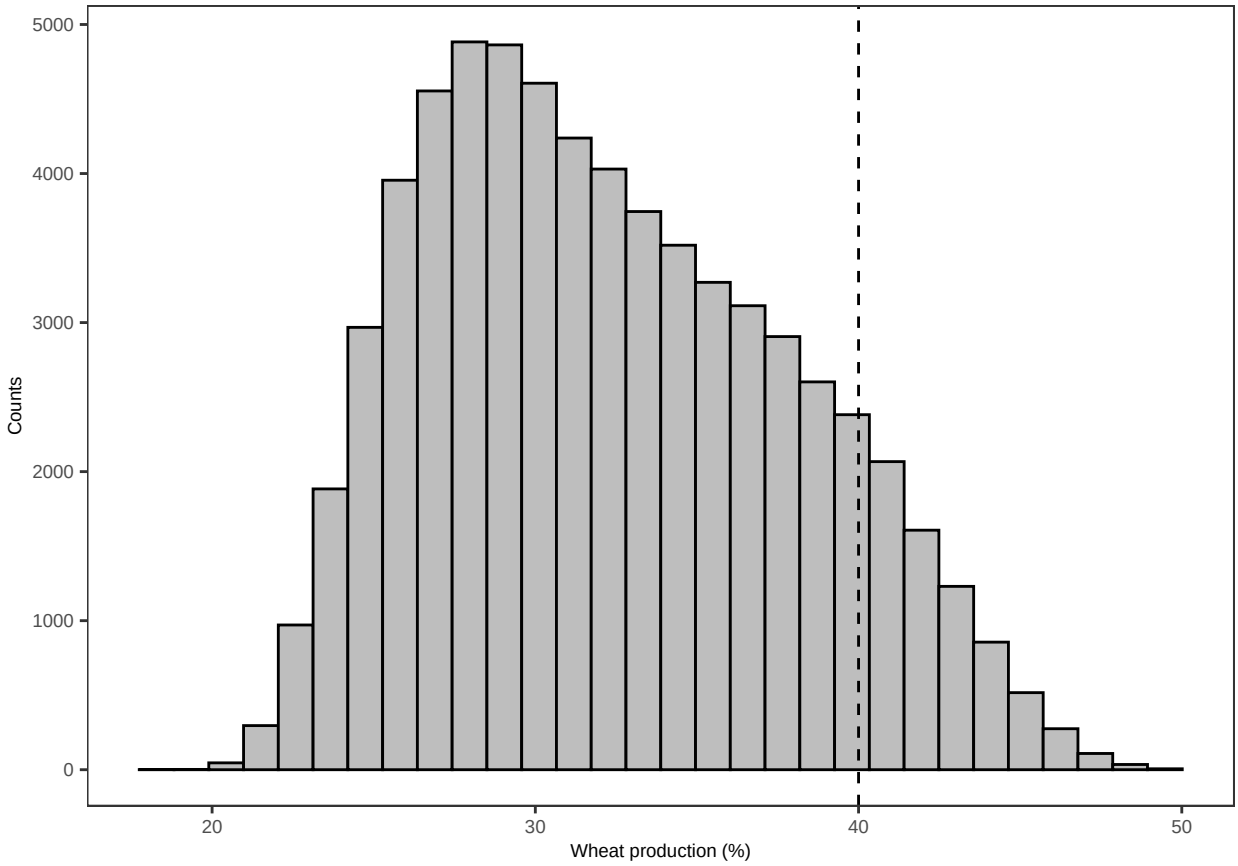
```
# Proportion of simulations around 70% ----------------------------------------
```

```
target.value <- 40  # target value
epsilon <- 5  # +-5%: tolerance around the target value

# Calculate percentage of simulations within tolerance range
sum(AB.mat$Y >= (target.value - epsilon) &
      AB.mat$Y <= (target.value + epsilon)) /
  length(AB.mat$Y) * 100
```

```
## [1] 30.66711
```

```
plot.uncertainty.food <- ggplot(AB.mat, aes(Y)) +
  geom_histogram(fill = "grey", color = "black") +
  geom_vline(xintercept = 40, lty = 2) +
  labs(y = "Counts", x = "Wheat production (%)") +
  theme_AP()

plot.uncertainty.food
```

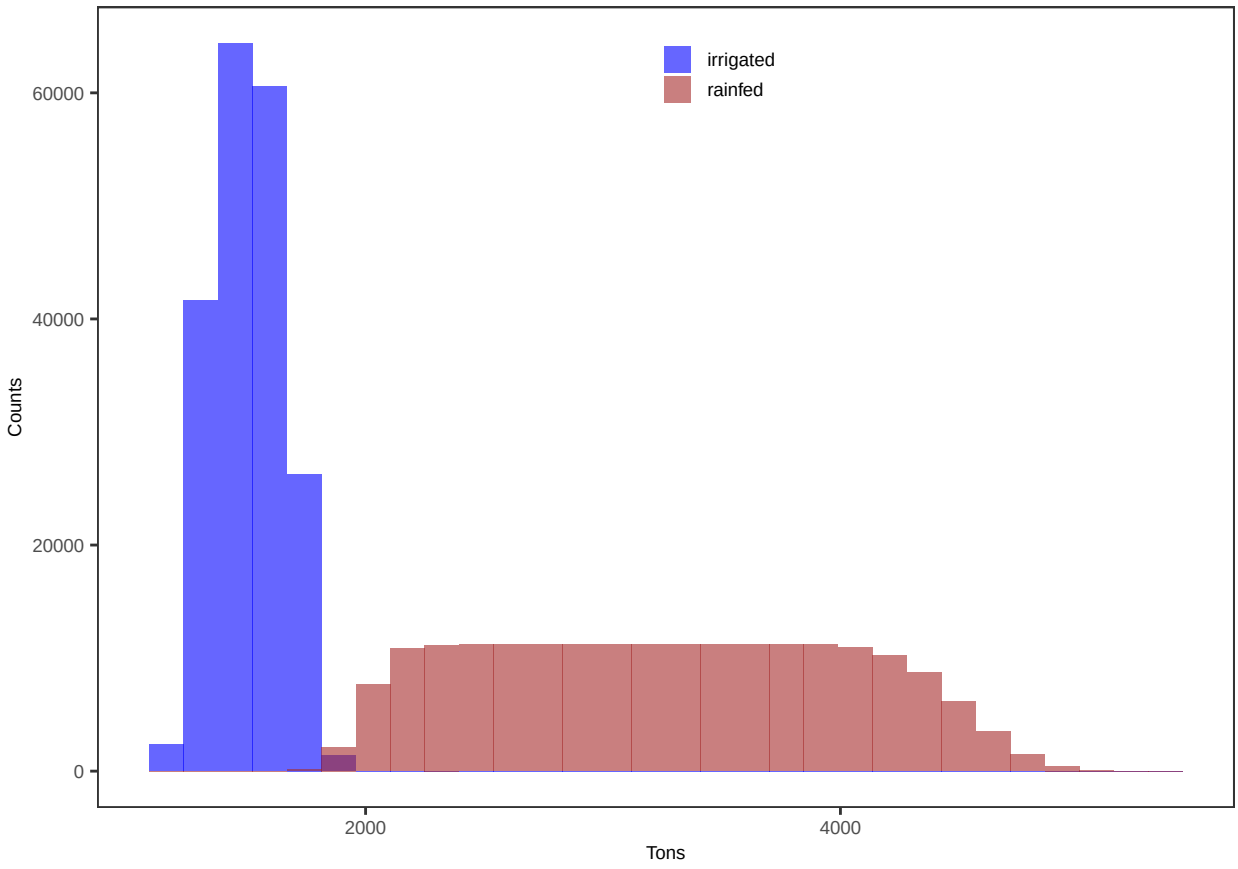
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
plot.uncertainty.production <- data.table(irrigated = production.irrigation,
           rainfed = production.non.irrigation) %>%
  melt(., measure.vars = colnames(.)) %>%
  ggplot(., aes(value, fill = variable)) +
```

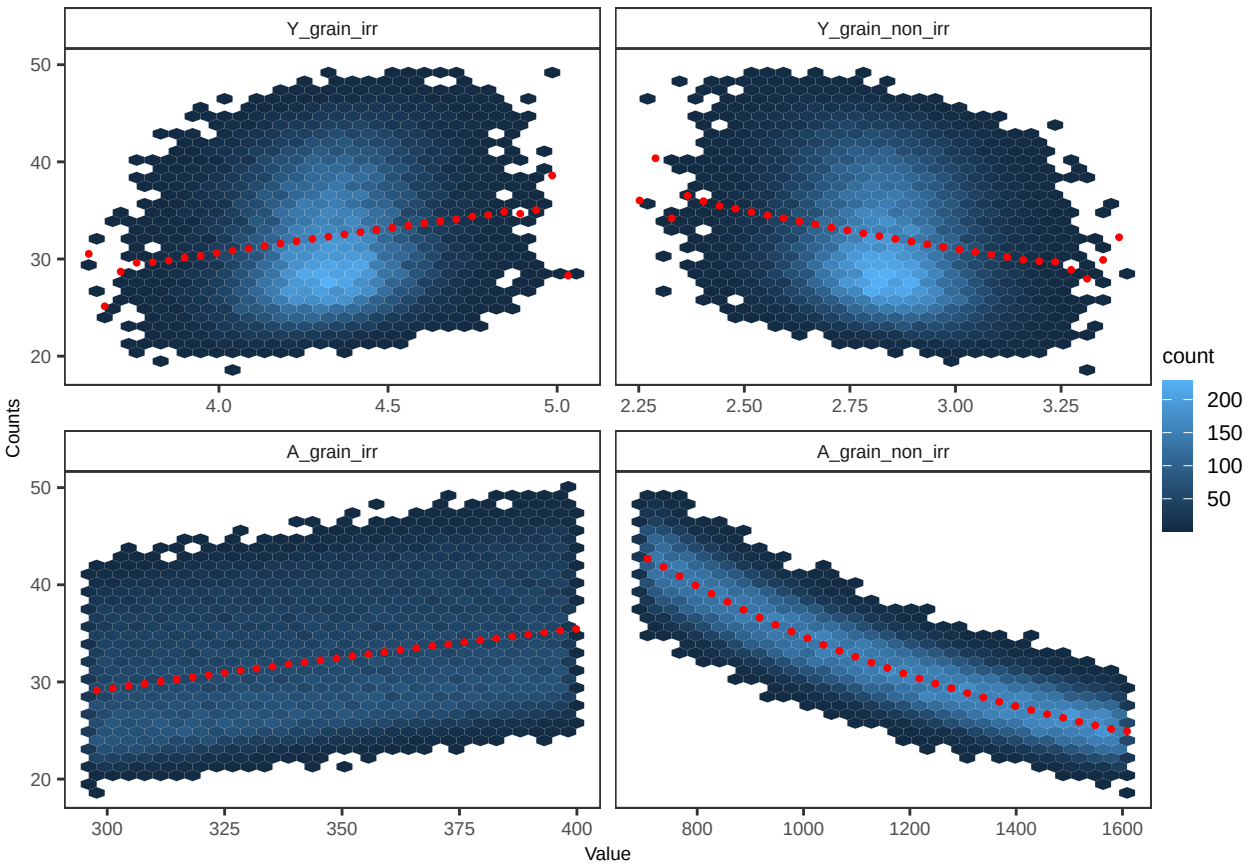
```
  geom_histogram(position = "identity", alpha = 0.6)  +
  scale_fill_manual(name = "", values = c("blue", "brown")) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "Tons", y = "Counts") +
  theme_AP() +
  theme(legend.position = c(0.55, 0.95),
        legend.text = element_text(size = 7))

plot.uncertainty.production
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



8.2 Sensitivity analysis

```
# SENSITIVITY ANALYSIS ###########################################################

# Scatterplot ------------------------------------------------------------------

plot.scatter <- plot_scatter(data = mat, N = N, Y = Y, params = params, method = "bin") +
  labs(x = "Value", y = "Counts") +
  theme_AP()

plot.scatter
```

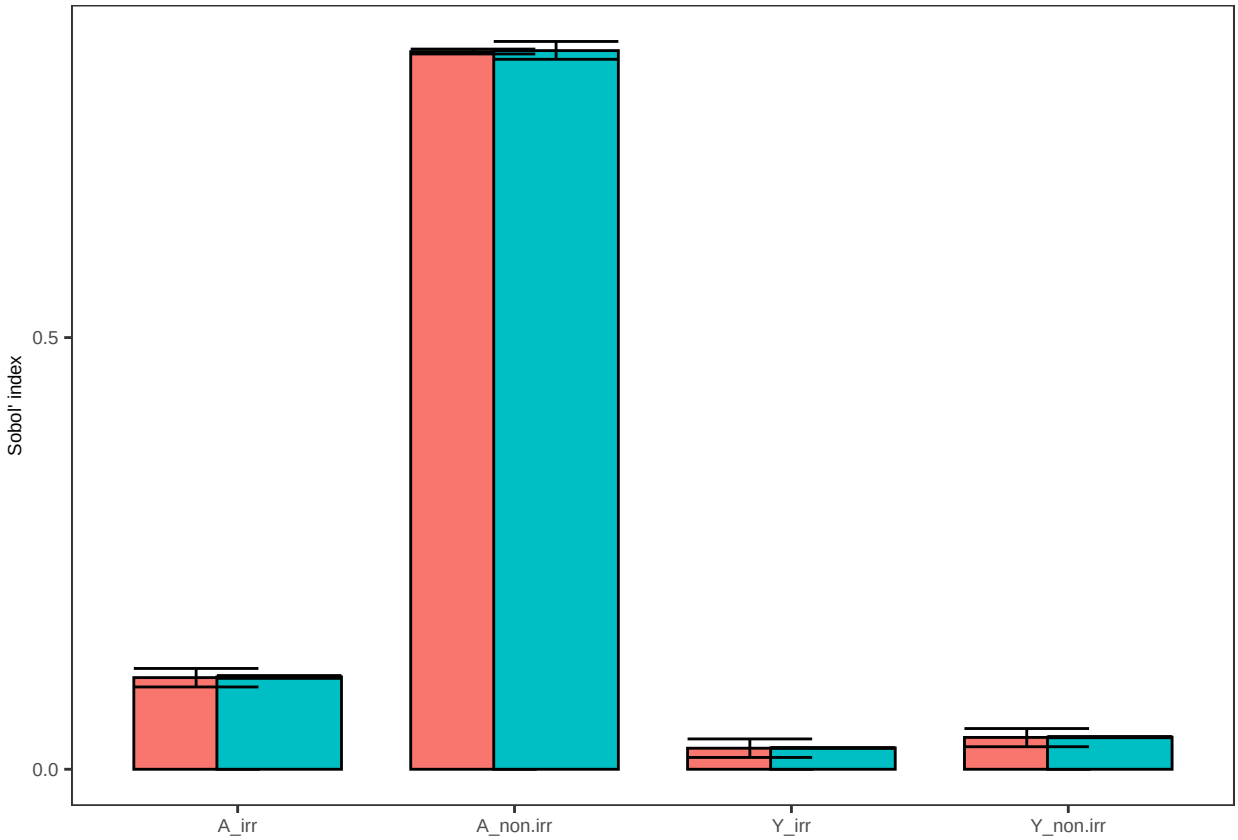



```
# Sobol' indices ----------------------------------------------------------

ind <- sobol_indices(matrices = matrices, Y = Y, N = N, params = params,
                     boot = TRUE, R = R, first = first, total = total)

ind
```

```
##
## First-order estimator: jansen | Total-order estimator: jansen
##
## Total number of model runs: 196608
##
## Sum of first order indices: 0.9985894
##      original          bias    std.error     low.ci    high.ci sensitivity
##         <num>         <num>        <num>      <num>      <num>      <char>
## 1: 0.02438511 -1.447274e-05 0.0054705753 0.01367745 0.03512171          Si
## 2: 0.03686929  2.326706e-04 0.0053775903 0.02609673 0.04717650          Si
## 3: 0.10615700  1.123836e-04 0.0054672009 0.09532910 0.11676013          Si
## 4: 0.83117806 -1.114025e-04 0.0014481353 0.82845117 0.83412775          Si
## 5: 0.02467180  1.054066e-05 0.0002197535 0.02423055 0.02509197          Ti
## 6: 0.03722756  1.836174e-06 0.0003513289 0.03653713 0.03791432          Ti
## 7: 0.10698070  5.673975e-05 0.0008294775 0.10529821 0.10854970          Ti
## 8: 0.83235721 -2.556394e-04 0.0052651204 0.82229340 0.84293230          Ti
##         parameters
```

```
##             <char>
## 1:     Y_grain_irr
## 2: Y_grain_non_irr
## 3:     A_grain_irr
## 4: A_grain_non_irr
## 5:     Y_grain_irr
## 6: Y_grain_non_irr
## 7:     A_grain_irr
## 8: A_grain_non_irr
```

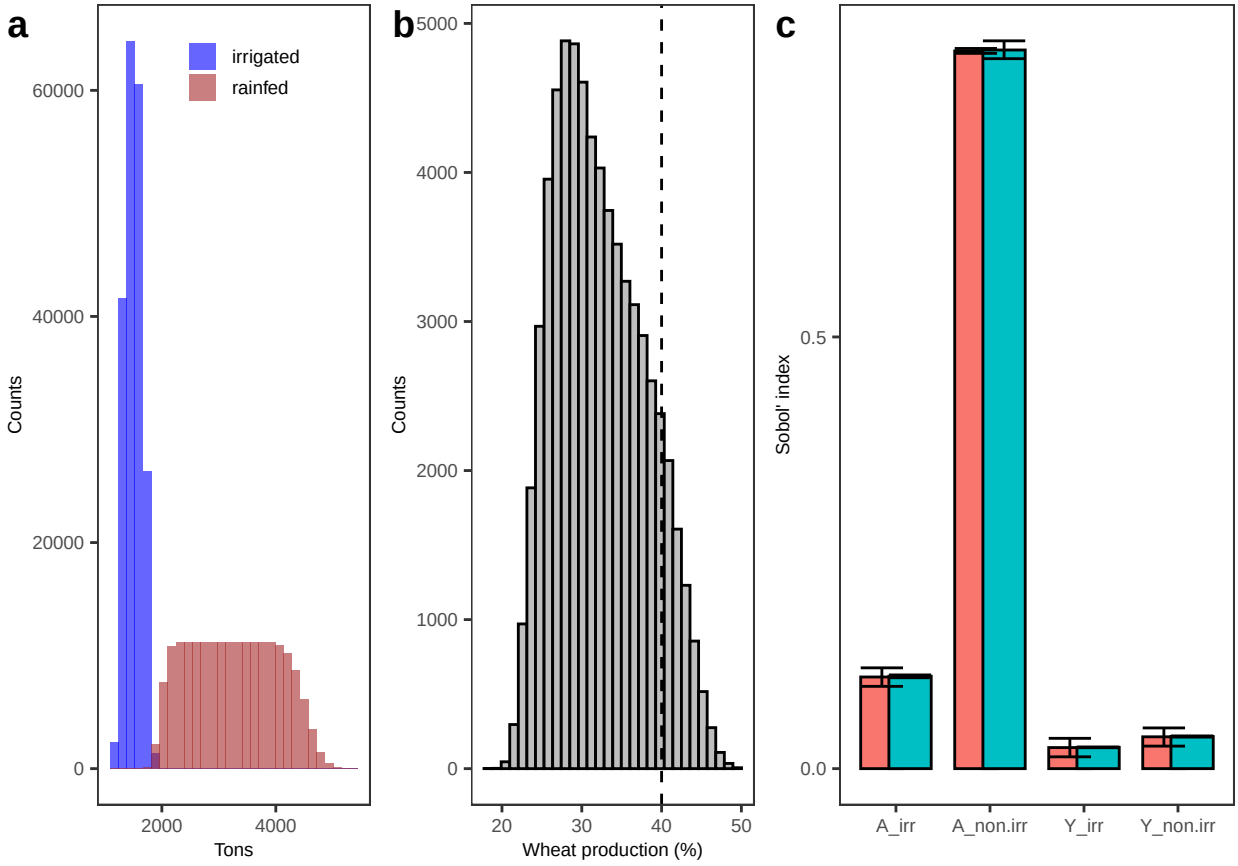
```
plot.indices.food <- plot(ind) +
  theme_AP() +
  theme(legend.position = "none") +
  scale_x_discrete(labels = c("A_irr", "A_non.irr", "Y_irr", "Y_non.irr"))

plot.indices.food
```



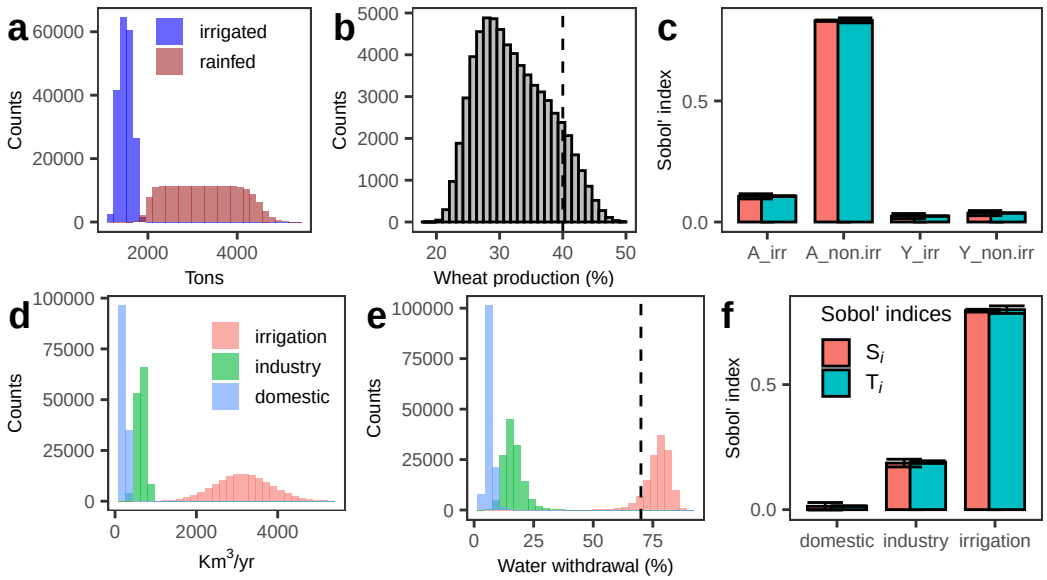

```
food.ua.sa.plot <- plot_grid(plot.uncertainty.production, plot.uncertainty.food,
                             plot.indices.food, ncol = 3, labels = c("a", "b", "c"),
                             rel_widths = c(0.32, 0.32, 0.4))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
food.ua.sa.plot
```




```
plot_grid(food.ua.sa.plot, water.ua.sa.plot, ncol = 1)
```



9 Extra: maize

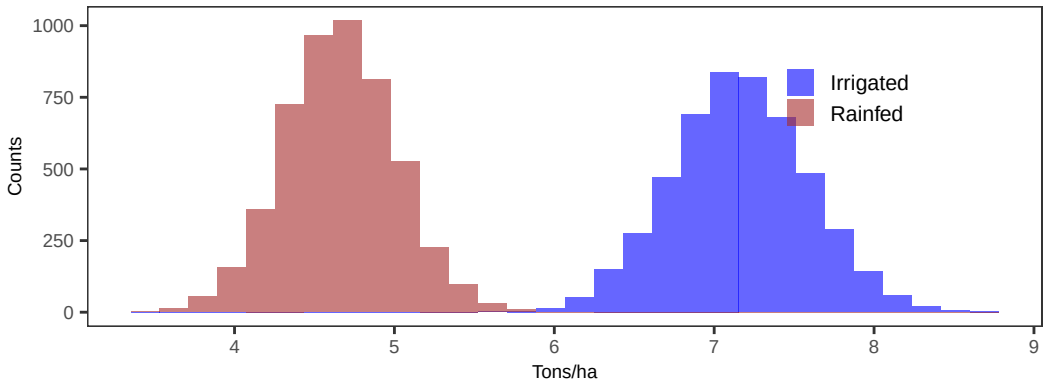
```
# AVERAGE YIELD OF IRRIGATED AND RAINFED MAIZE (TONS / HA) ####################

# Data from Mueller et al 2012 ------------------------------------------------

mueller.irrigated <- data.table(read.xlsx("mueller_et_al_2012_maize.xlsx")) %>%
  .[, .(`yield.(t/ha).75%`, `yield.(t/ha).90%`, `yield.(t/ha).100%`)] %>%
  melt(., measure.vars = colnames(.)) %>%
  .[, value]

mueller.rainfed <- data.table(read.xlsx("mueller_et_al_2012_maize.xlsx")) %>%
  .[, .(`yield.(t/ha)`, `yield.(t/ha).50%`)] %>%
  melt(., measure.vars = colnames(.)) %>%
  .[, value]

# Data from GAEZ --------------------------------------------------------------

gaez.irrigated <- data.table(read.xlsx("gaez_maize_data.xlsx"))[, `Yield.(Irrigated).(tons/ha)`

gaez.rainfed <- data.table(read.xlsx("gaez_maize_data.xlsx"))[, `Yield.(tons/ha)`]

# Data from SPAM 2024 ---------------------------------------------------------

spam.rainfed <- fread("spam2020V1r0_global_Y_TR.csv")[, .(ADM0_NAME, MAIZ_R, unit)] %>%
  .[, Region:= countrycode(ADM0_NAME, origin = "country.name", destination = "region")] %>%
  .[, .(mean = mean(MAIZ_R) / 1000), Region] %>%
  .[, mean]

spam.irrigated <- fread("spam2020V1r0_global_Y_TI.csv")[, .(ADM0_NAME, MAIZ_I, unit)] %>%
  .[, Region:= countrycode(ADM0_NAME, origin = "country.name", destination = "region")] %>%
  .[, .(mean = mean(MAIZ_I) / 1000), Region] %>%
  .[, mean]

# Data from Doll and Siebert 2010 ---------------------------------------------

doll.rainfed <- 4.11
doll.irrigated <- 5.68

# Merge data and create irrigated and rainfed vector --------------------------

maize.dt <- data.table(Y_grain_irr = c(mueller.irrigated, gaez.irrigated, spam.irrigated,
                                       doll.irrigated),
                       Y_grain_non_irr = c(mueller.rainfed, gaez.rainfed, spam.rainfed,
                                           doll.rainfed))
```

```
## Warning: Item 2 has 35 rows but longest item has 44; recycled with remainder.
```

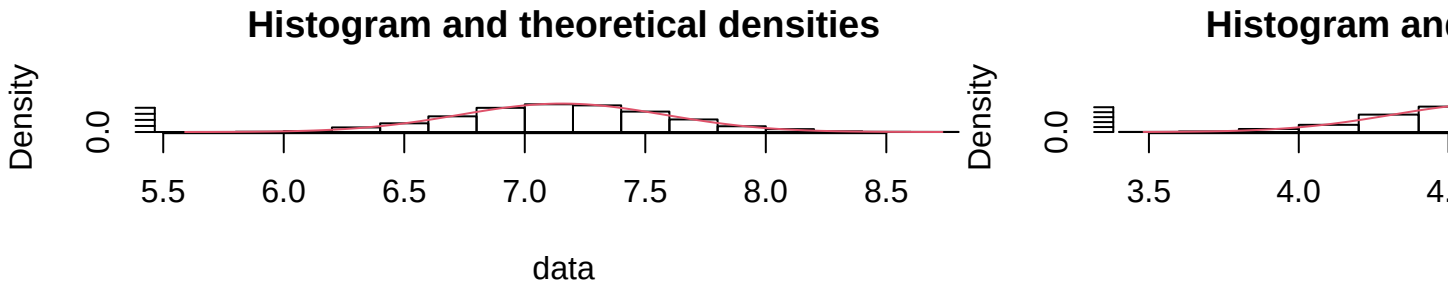
```
# Bootstrap wheat production ----------------------------------------------------

bootstrap.maize.dt <- replicate(5000, {

  maize.dt[sample(.N, replace = TRUE), .(Y_grain_irr = mean(Y_grain_irr),
                                         Y_grain_non_irr = mean(Y_grain_non_irr))]
}, simplify = FALSE) %>%
  rbindlist(.)

bootstrap.maize.dt %>%
  melt(., measure.vars = colnames(.)) %>%
  ggplot(., aes(value, fill = variable)) +
  geom_histogram(alpha = 0.6, position = "identity") +
  scale_fill_manual(name = "", values = c("blue", "brown"),
                    labels = c("Irrigated", "Rainfed")) +
  labs(x = "Tons/ha", y = "Counts") +
  theme_AP() +
  theme(legend.position = c(0.8, 0.8))
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Describe with probability distribution ----------------------------------------

selected_cols <- colnames(maize.dt)
dist.estimates <- lapply(selected_cols, function(x) fitdist(bootstrap.maize.dt[[x]], "norm"))
names(dist.estimates) <- selected_cols
lapply(dist.estimates, function(x) denscomp(x))
```

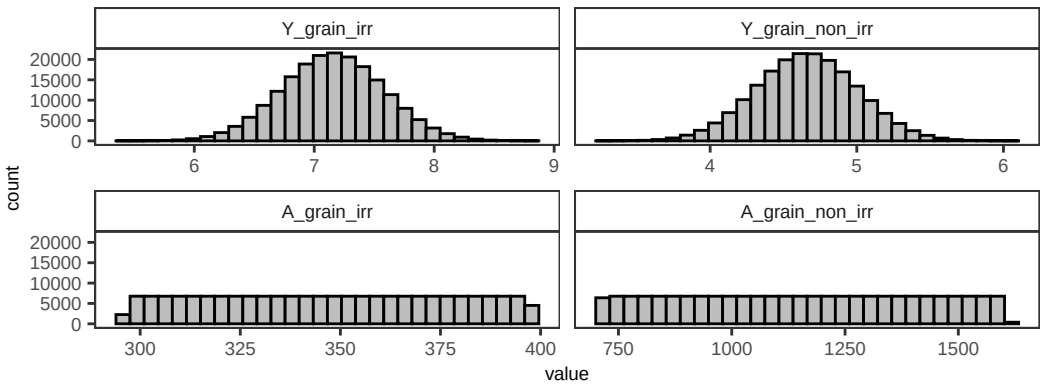



```
## $Y_grain_irr
## NULL
##
## $Y_grain_non_irr
## NULL
```

```
# Define function ------------------------------------------------------------

out <- lapply(selected_cols, function(x) {

  data.table(mean.value = dist.estimates[[x]]$estimate[["mean"]],
             sd.value = dist.estimates[[x]]$estimate[["sd"]])

})

names(out) <- selected_cols

for (i in names(out)) {

  out[[i]][, Fa.norm:= pnorm(min(maize.dt[[i]]), mean = mean.value, sd = sd.value)]
  out[[i]][, Fb.norm:= pnorm(max(maize.dt[[i]]), mean = mean.value, sd = sd.value)]
}

# DATA ON THE EXTENSION OF IRRIGATED AREAS ######################################

meier <- fread("meier.csv")
col_maps <- colnames(meier)[-c(1:3)]
irrigated.area.dt <- melt(meier, measure.vars = col_maps) %>%
  .[, sum(value, na.rm = TRUE) / 10^6, variable]

range.irrigated.areas <- c(min(irrigated.area.dt$V1), max(irrigated.area.dt$V1))

# DATA ON THE EXTENSION OF TOTAL CROPLAND ######################################

tubiello.range <- c(1100, 1900)
potapov.range <- c(1181.5, 1306.9)

min.total.cropland <- min(c(tubiello.range, potapov.range))
max.total.cropland <- max(c(tubiello.range, potapov.range))
```

```r
range.total.cropland <- c(min.total.cropland, max.total.cropland)

# DATA ON EXTENSION RAINFED AREAS ##############################################

rainfed.areas <- range.total.cropland - rev(range.irrigated.areas)

# DEFINE SAMPLE MATRIX -------------------------------------------------------

# Set parameters of calculation ----------------------------------------------

N <- 2^15
matrices <- c("A", "B", "AB")
order = "first"
first <- "jansen"
total = "jansen"
R <- 10^3
params <- c("Y_grain_irr", "Y_grain_non_irr", "A_grain_irr", "A_grain_non_irr")

# create sample matrix and transform -----------------------------------------

mat <- sobol_matrices(matrices = matrices, N = N, order = order, params = params)

for (i in colnames(wheat.dt)) {

  mat[, i] <- qunif(mat[, i], out[[i]]$Fa.norm, out[[i]]$Fb.norm)
  mat[, i] <- qnorm(mat[, i], out[[i]]$mean.value, out[[i]]$sd.value)

}

mat[, "A_grain_irr"] <- qunif(mat[, "A_grain_irr"], min(irrigated.area.dt$V1), max(irrigated.ar
mat[, "A_grain_non_irr"] <- qunif(mat[, "A_grain_non_irr"], rainfed.areas[[1]], rainfed.areas[

# Plot uncertain distributions -----------------------------------------------

data.table(mat) %>%
  melt(., measure.vars = colnames(.)) %>%
  ggplot(., aes(value)) +
  geom_histogram(color = "black", fill = "grey") +
  facet_wrap(~variable, scales = "free_x") +
  theme_AP()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# RUN CALCULATIONS ##################################################################

production.irrigation <- mat[, "Y_grain_irr"] * mat[, "A_grain_irr"]
production.non.irrigation <- mat[, "Y_grain_non_irr"] * mat[, "A_grain_non_irr"]
Y <- (production.irrigation / (production.irrigation + production.non.irrigation)) * 100

# UNCERTAINTY ANALYSIS ##############################################################

final.dt <- cbind(mat, Y) %>%
  data.table()
AB.mat <- final.dt[1:(2 * N)]

AB.mat[, .(min = min(Y), max = max(Y))]
```

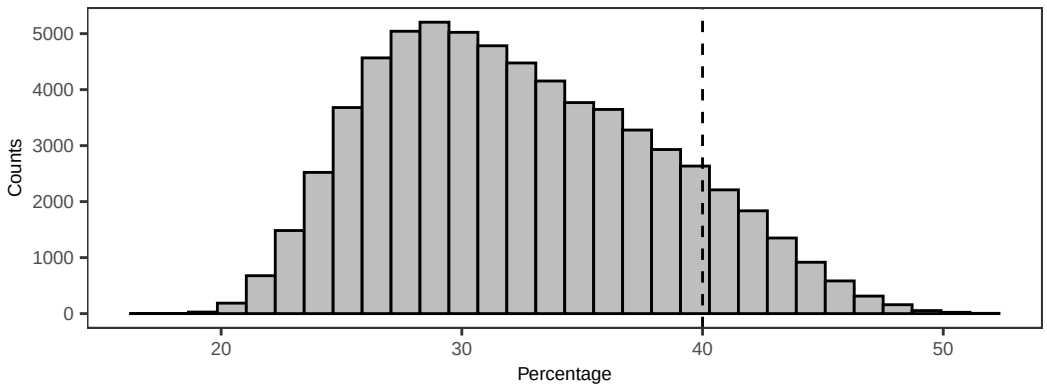
```
##         min      max
##       <num>    <num>
## 1: 17.07184 51.94844
```

```
quantile(Y, probs = c(0.025, 0.975))
```

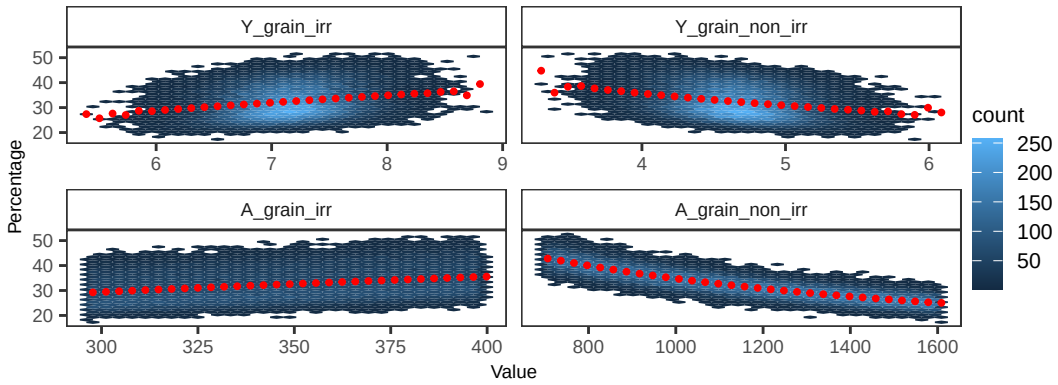
```
##     2.5%    97.5%
## 22.99705 44.38976
```

```
plot.uncertainty <- ggplot(AB.mat, aes(Y)) +
  geom_histogram(fill = "grey", color = "black") +
  geom_vline(xintercept = 40, lty = 2) +
  labs(y = "Counts", x = "Percentage") +
  theme_AP()

plot.uncertainty
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# SENSITIVITY ANALYSIS ###########################################################

# Scatterplot ------------------------------------------------------------------

plot.scatter <- plot_scatter(data = mat, N = N, Y = Y, params = params, method = "bin") +
  labs(x = "Value", y = "Percentage") +
  theme_AP()

plot.scatter
```




```
# Sobol' indices ---------------------------------------------------------------

ind <- sobol_indices(matrices = matrices, Y = Y, N = N, params = params,
                     boot = TRUE, R = R, first = first, total = total)

ind
```

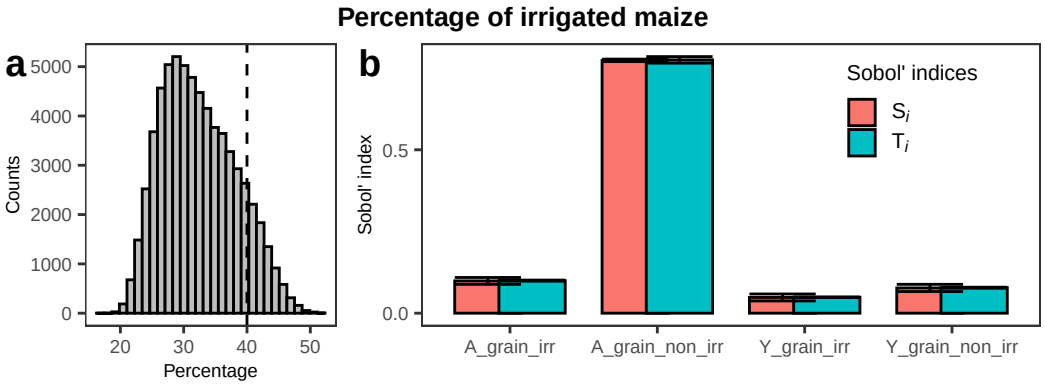
```
##
## First-order estimator: jansen | Total-order estimator: jansen
##
## Total number of model runs: 196608
##
## Sum of first order indices: 0.9981129
##      original          bias    std.error     low.ci    high.ci sensitivity
##         <num>         <num>        <num>      <num>       <num>      <char>
## 1: 0.04823467  1.217249e-04 0.0054288346 0.03747262 0.05875326          Si
```

```
## 2: 0.07732469  1.196263e-04 0.0056400784 0.06615071 0.08825941          Si
## 3: 0.09883057  6.980201e-05 0.0053384382 0.08829762 0.10922391          Si
## 4: 0.77372297 -2.794170e-05 0.0019271856 0.76997369 0.77752812          Si
## 5: 0.04874477  9.494739e-06 0.0004446789 0.04786372 0.04960683          Ti
## 6: 0.07800673  4.863816e-06 0.0007364469 0.07655845 0.07944527          Ti
## 7: 0.09966098 -1.528909e-05 0.0008066372 0.09809529 0.10125725          Ti
## 8: 0.77525985  9.042625e-06 0.0050807222 0.76529277 0.78520884          Ti
##         parameters
##             <char>
## 1:     Y_grain_irr
## 2: Y_grain_non_irr
## 3:     A_grain_irr
## 4: A_grain_non_irr
## 5:     Y_grain_irr
## 6: Y_grain_non_irr
## 7:     A_grain_irr
## 8: A_grain_non_irr
```

```
plot.indices <- plot(ind) +
  theme_AP() +
  theme(legend.position = c(0.8, 0.8))

# MERGE UNCERTAINTY AND SENSITIVITY PLOTS -------------------------------------

my_plot_list <- list(plot.uncertainty, plot.indices)
final.plot <- ggpubr::ggarrange(plotlist = my_plot_list, labels = "auto", ncol = 2, widths = c
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
ggpubr::annotate_figure(final.plot, top = ggpubr::text_grob("Percentage of irrigated maize",
                                                            face = "bold", size =10))
```

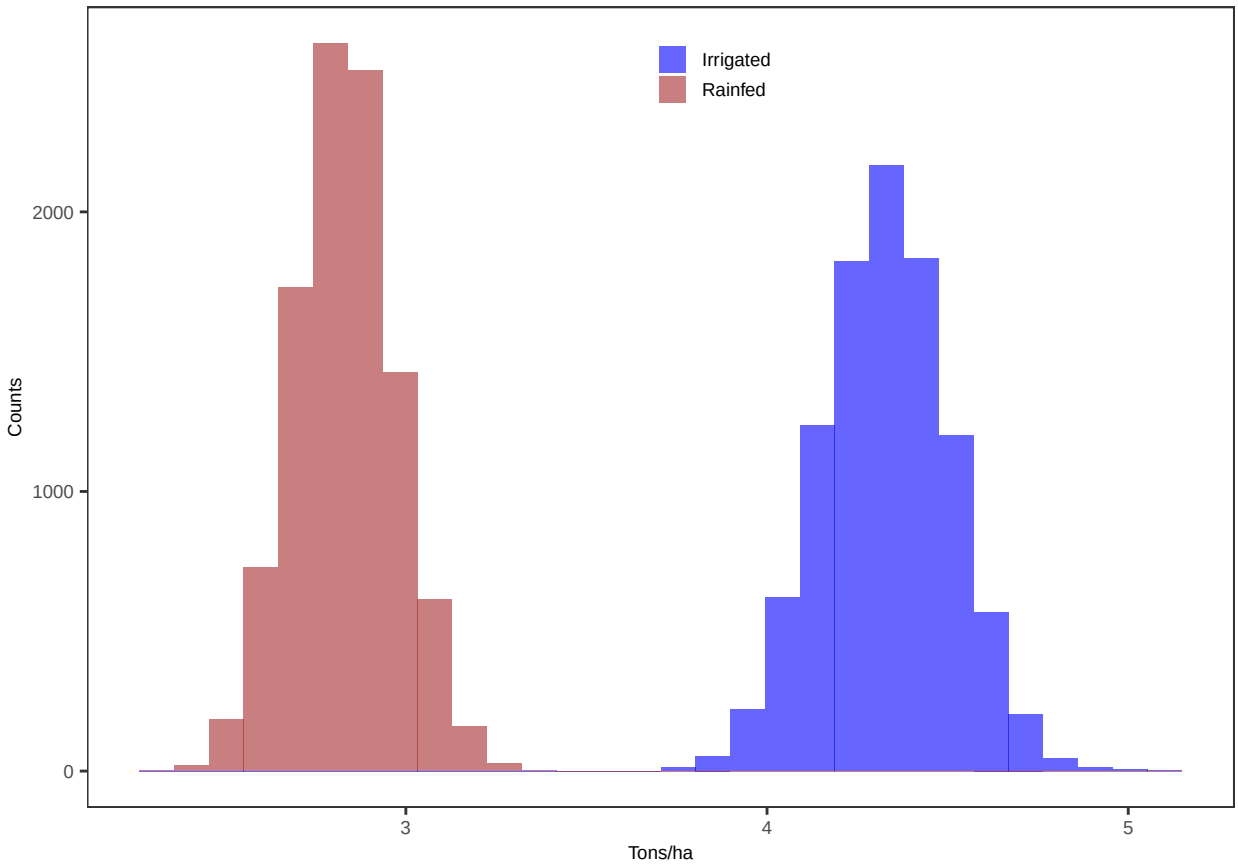


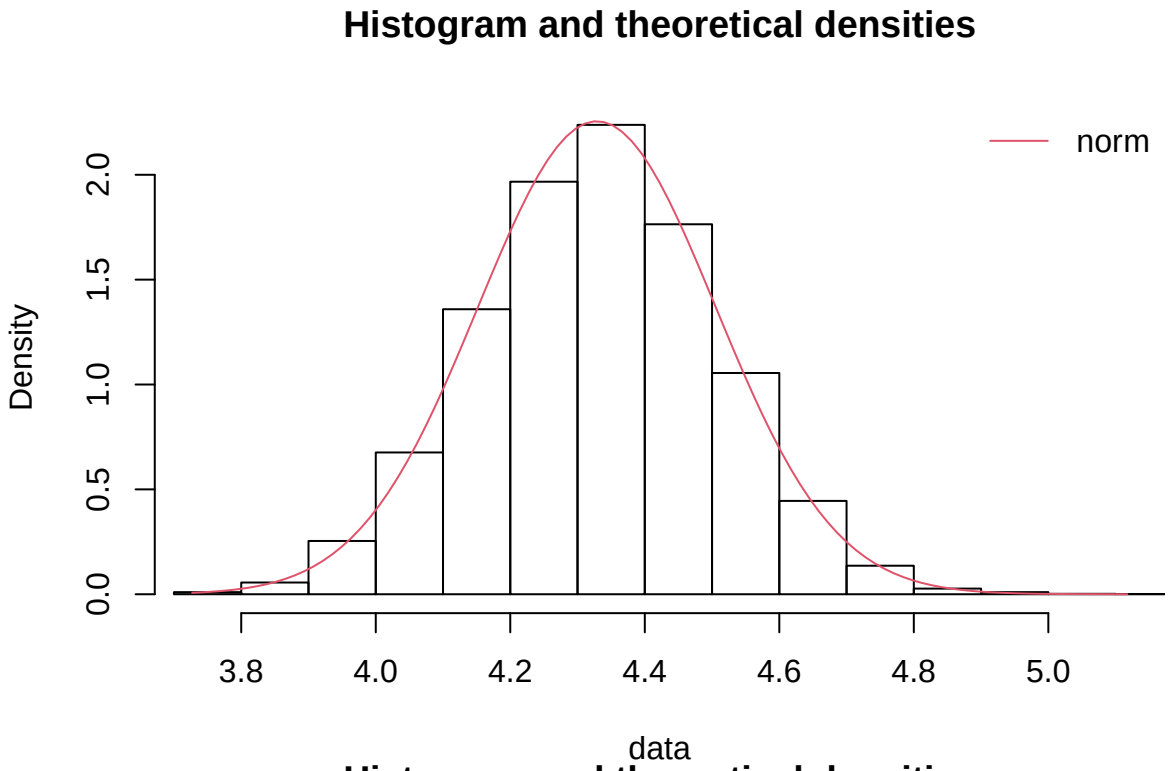
10 Extra: kilocalories

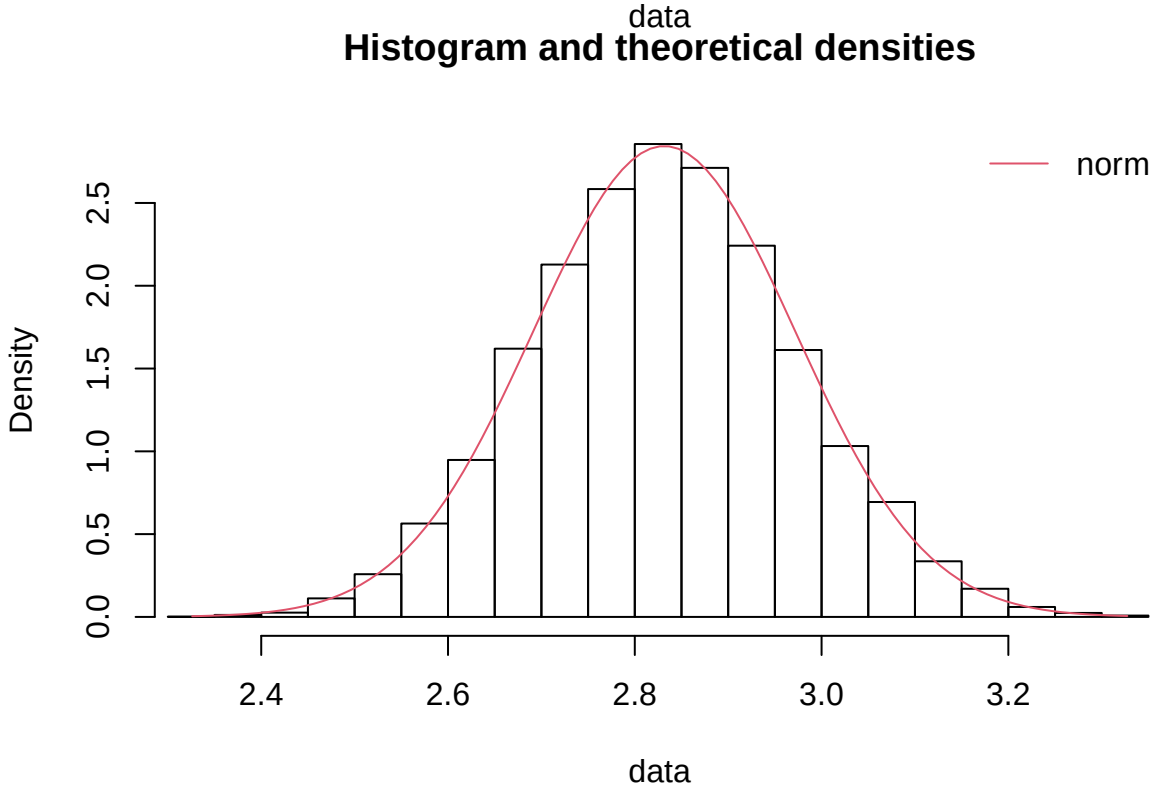
```
## Warning in as.data.table.list(x, keep.rownames = keep.rownames, check.names =
## check.names, : Item 1 has 72 rows but longest item has 73; recycled with
## remainder.
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```








```
## $Y_grain_irr
## $Y_grain_irr$hist
## $breaks
##  [1] 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0 5.1 5.2
##
```

```
## $counts
##  [1]   10   56  254  676 1359 1967 2238 1764 1055  445  136   27   10    2    1
##
## $density
##  [1] 0.010 0.056 0.254 0.676 1.359 1.967 2.238 1.764 1.055 0.445 0.136 0.027
## [13] 0.010 0.002 0.001
##
## $mids
##  [1] 3.75 3.85 3.95 4.05 4.15 4.25 4.35 4.45 4.55 4.65 4.75 4.85 4.95 5.05 5.15
##
## $xname
## [1] "mydata"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## $Y_grain_irr$densities
##                [,1]
##   [1,] 0.0069378817
##   [2,] 0.0090361001
##   [3,] 0.0116964049
##   [4,] 0.0150466897
##   [5,] 0.0192374165
##   [6,] 0.0244438576
##   [7,] 0.0308681040
##   [8,] 0.0387406925
##   [9,] 0.0483216790
##  [10,] 0.0599009715
##  [11,] 0.0737977173
##  [12,] 0.0903585400
##  [13,] 0.1099544206
##  [14,] 0.1329760404
##  [15,] 0.1598274286
##  [16,] 0.1909178088
##  [17,] 0.2266515968
##  [18,] 0.2674165849
##  [19,] 0.3135704335
##  [20,] 0.3654257002
##  [21,] 0.4232337401
##  [22,] 0.4871679286
##  [23,] 0.5573067598
##  [24,] 0.6336174725
##  [25,] 0.7159409305
##  [26,] 0.8039785317
##  [27,] 0.8972819377
```

```
##  [28,] 0.9952463925
##  [29,] 1.0971083280
##  [30,] 1.2019478490
##  [31,] 1.3086965291
##  [32,] 1.4161507599
##  [33,] 1.5229906672
##  [34,] 1.6278043602
##  [35,] 1.7291170214
##  [36,] 1.8254240906
##  [37,] 1.9152275601
##  [38,] 1.9970741978
##  [39,] 2.0695943581
##  [40,] 2.1315399469
##  [41,] 2.1818200759
##  [42,] 2.2195329881
##  [43,] 2.2439929488
##  [44,] 2.2547509824
##  [45,] 2.2516085798
##  [46,] 2.2346237904
##  [47,] 2.2041094390
##  [48,] 2.1606235424
##  [49,] 2.1049523375
##  [50,] 2.0380866410
##  [51,] 1.9611925339
##  [52,] 1.8755775828
##  [53,] 1.7826539613
##  [54,] 1.6838999224
##  [55,] 1.5808210797
##  [56,] 1.4749128967
##  [57,] 1.3676256610
##  [58,] 1.2603330410
##  [59,] 1.1543051040
##  [60,] 1.0506864301
##  [61,] 0.9504796953
##  [62,] 0.8545348465
##  [63,] 0.7635437453
##  [64,] 0.6780399509
##  [65,] 0.5984031304
##  [66,] 0.5248674528
##  [67,] 0.4575332304
##  [68,] 0.3963810234
##  [69,] 0.3412874199
##  [70,] 0.2920417343
##  [71,] 0.2483629278
##  [72,] 0.2099161419
##  [73,] 0.1763283380
##  [74,] 0.1472026422
##  [75,] 0.1221311115
```

```
##  [76,] 0.1007057405
##  [77,] 0.0825276271
##  [78,] 0.0672143034
##  [79,] 0.0544053062
##  [80,] 0.0437661182
##  [81,] 0.0349906502
##  [82,] 0.0278024590
##  [83,] 0.0219549079
##  [84,] 0.0172304747
##  [85,] 0.0134394061
##  [86,] 0.0104178986
##  [87,] 0.0080259674
##  [88,] 0.0061451412
##  [89,] 0.0046760974
##  [90,] 0.0035363273
##  [91,] 0.0026578996
##  [92,] 0.0019853720
##  [93,] 0.0014738811
##  [94,] 0.0010874272
##  [95,] 0.0007973613
##  [96,] 0.0005810685
##  [97,] 0.0004208397
##  [98,] 0.0003029168
##  [99,] 0.0002166941
## [100,] 0.0001540594
## [101,] 0.0001088545
##
##
## $Y_grain_non_irr
## $Y_grain_non_irr$hist
## $breaks
##  [1] 2.30 2.35 2.40 2.45 2.50 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 2.95 3.00
## [16] 3.05 3.10 3.15 3.20 3.25 3.30 3.35
##
## $counts
##  [1]    1    6   13   56  129  282  474  810 1064 1292 1428 1356 1121  806  516
## [16]  347  168   85   30   12    4
##
## $density
##  [1] 0.002 0.012 0.026 0.112 0.258 0.564 0.948 1.620 2.128 2.584 2.856 2.712
## [13] 2.242 1.612 1.032 0.694 0.336 0.170 0.060 0.024 0.008
##
## $mids
##  [1] 2.325 2.375 2.425 2.475 2.525 2.575 2.625 2.675 2.725 2.775 2.825 2.875
## [13] 2.925 2.975 3.025 3.075 3.125 3.175 3.225 3.275 3.325
##
## $xname
## [1] "mydata"
```

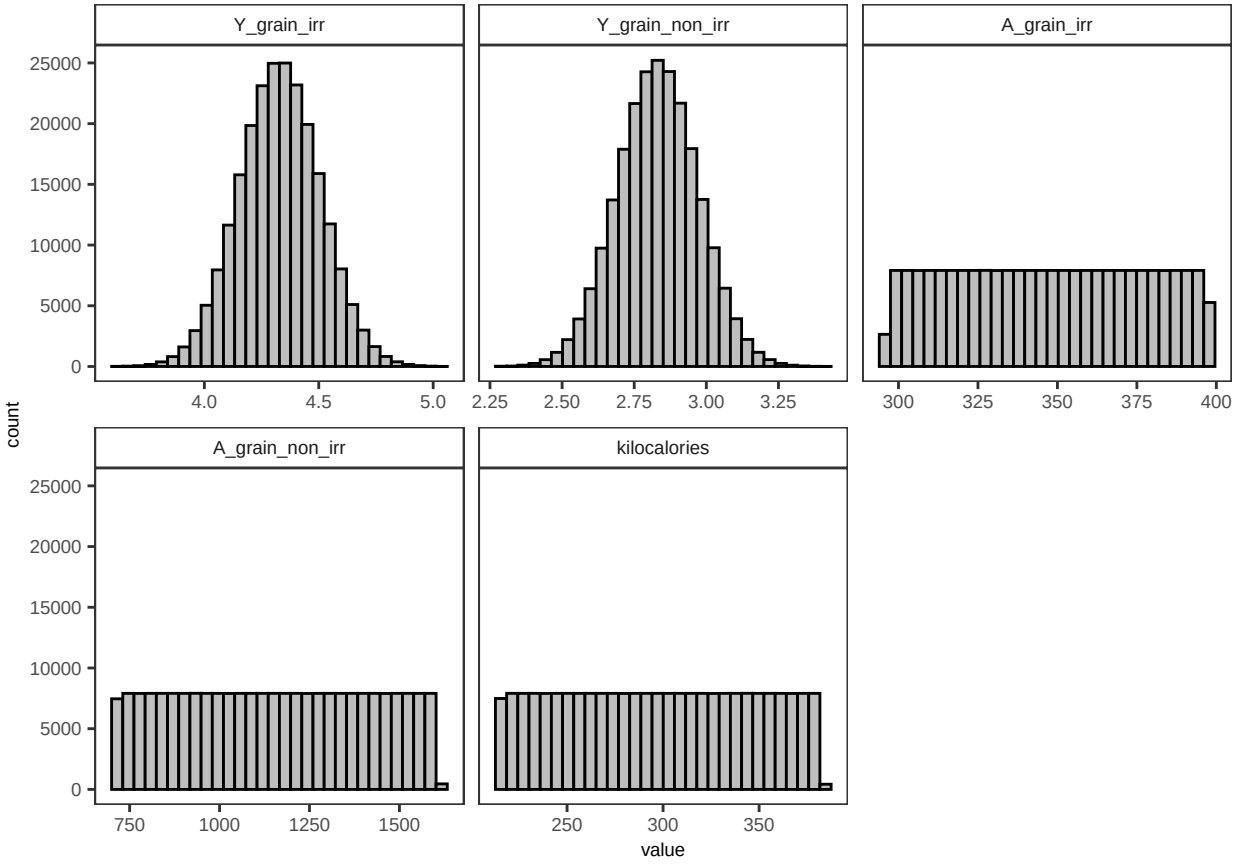
```
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## $Y_grain_non_irr$densities
##               [,1]
##   [1,] 0.004313090
##   [2,] 0.005563830
##   [3,] 0.007140795
##   [4,] 0.009118150
##   [5,] 0.011583885
##   [6,] 0.014641622
##   [7,] 0.018412449
##   [8,] 0.023036756
##   [9,] 0.028675995
##  [10,] 0.035514285
##  [11,] 0.043759774
##  [12,] 0.053645646
##  [13,] 0.065430660
##  [14,] 0.079399090
##  [15,] 0.095859943
##  [16,] 0.115145300
##  [17,] 0.137607680
##  [18,] 0.163616291
##  [19,] 0.193552071
##  [20,] 0.227801464
##  [21,] 0.266748884
##  [22,] 0.310767878
##  [23,] 0.360211060
##  [24,] 0.415398931
##  [25,] 0.476607767
##  [26,] 0.544056831
##  [27,] 0.617895212
##  [28,] 0.698188678
##  [29,] 0.784906969
##  [30,] 0.877911985
##  [31,] 0.976947400
##  [32,] 1.081630176
##  [33,] 1.191444506
##  [34,] 1.305738631
##  [35,] 1.423724956
##  [36,] 1.544483784
##  [37,] 1.666970898
##  [38,] 1.790029110
##  [39,] 1.912403718
```

```
##  [40,] 2.032761729
##  [41,] 2.149714482
##  [42,] 2.261843223
##  [43,] 2.367726997
##  [44,] 2.465972128
##  [45,] 2.555242436
##  [46,] 2.634289309
##  [47,] 2.701980670
##  [48,] 2.757327908
##  [49,] 2.799509886
##  [50,] 2.827893208
##  [51,] 2.842048039
##  [52,] 2.841758937
##  [53,] 2.827030310
##  [54,] 2.798086302
##  [55,] 2.755365120
##  [56,] 2.699507992
##  [57,] 2.631343160
##  [58,] 2.551865444
##  [59,] 2.462212111
##  [60,] 2.363635835
##  [61,] 2.257475671
##  [62,] 2.145126962
##  [63,] 2.028011134
##  [64,] 1.907546260
##  [65,] 1.785119249
##  [66,] 1.662060381
##  [67,] 1.539620808
##  [68,] 1.418953479
##  [69,] 1.301097829
##  [70,] 1.186968402
##  [71,] 1.077347415
##  [72,] 0.972881176
##  [73,] 0.874080108
##  [74,] 0.781322055
##  [75,] 0.694858446
##  [76,] 0.614822862
##  [77,] 0.541241497
##  [78,] 0.474045005
##  [79,] 0.413081241
##  [80,] 0.358128416
##  [81,] 0.308908246
##  [82,] 0.265098720
##  [83,] 0.226346180
##  [84,] 0.192276461
##  [85,] 0.162504907
##  [86,] 0.136645159
##  [87,] 0.114316635
```

```
##  [88,] 0.095150708
##  [89,] 0.078795611
##  [90,] 0.064920138
##  [91,] 0.053216249
##  [92,] 0.043400676
##  [93,] 0.035215685
##  [94,] 0.028429105
##  [95,] 0.022833772
##  [96,] 0.018246498
##  [97,] 0.014506706
##  [98,] 0.011474810
##  [99,] 0.009030455
## [100,] 0.007070679
## [101,] 0.005508078
```

```
## [1] 3.327403
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

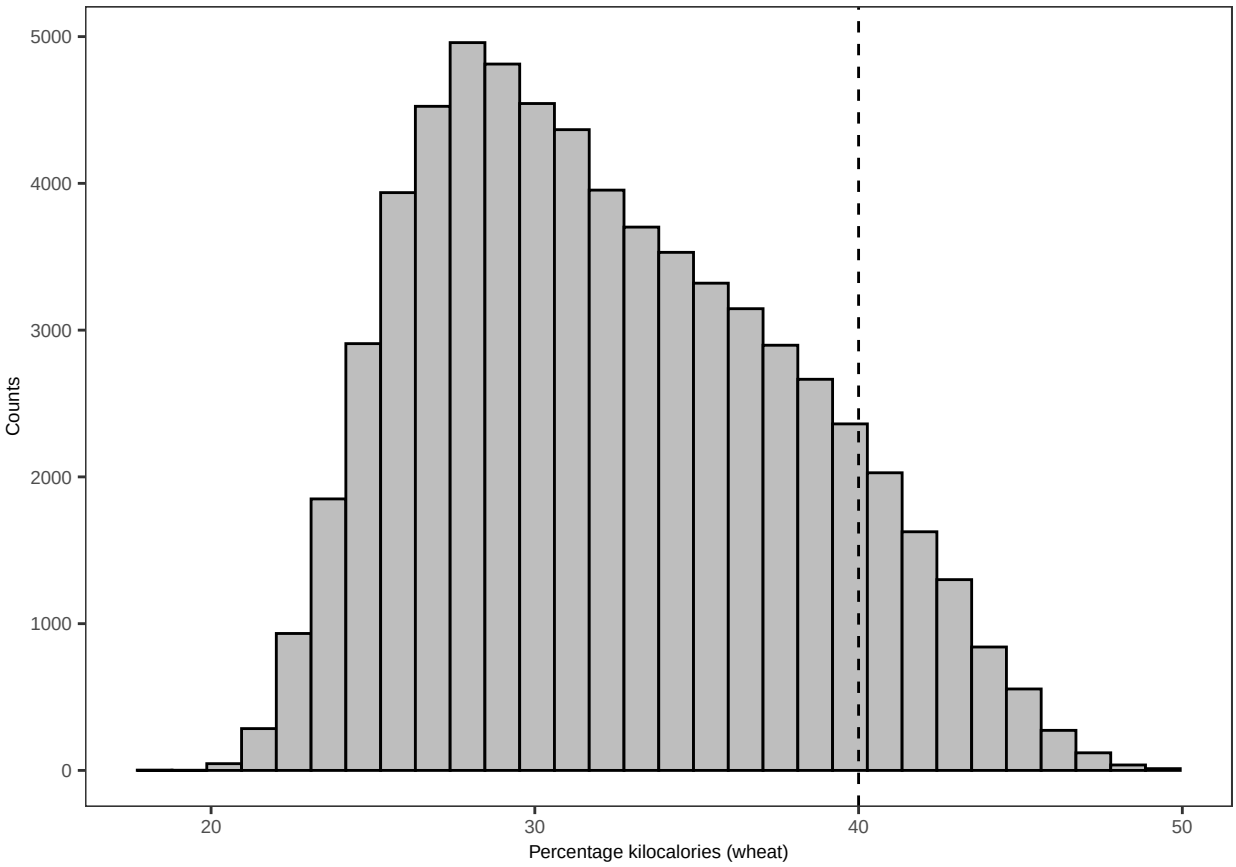



```
##         min      max
##       <num>    <num>
## 1: 18.49531 49.63822
```

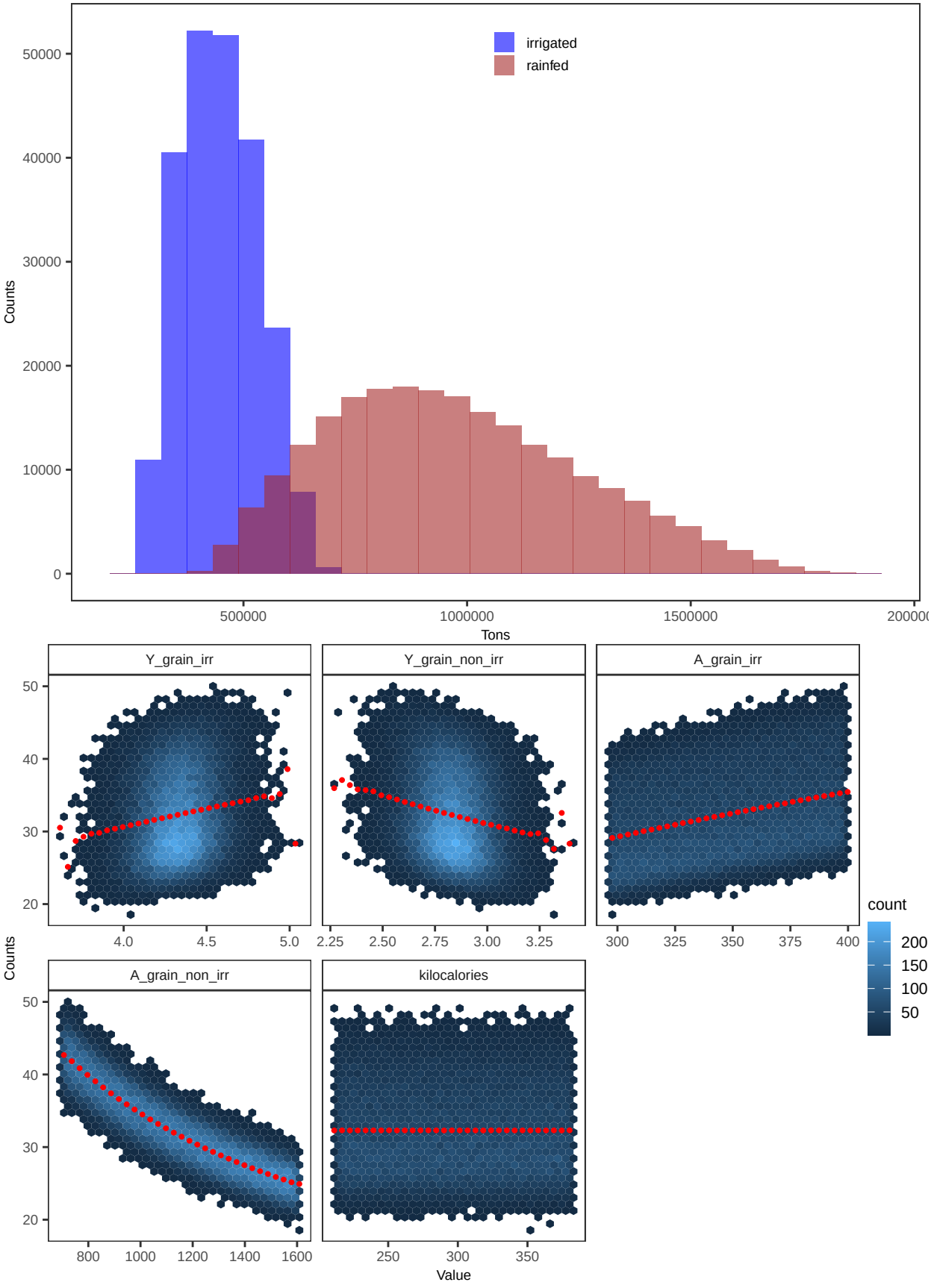
```
##     2.5%    97.5%
## 23.35638 43.72223
```

```
## [1] 30.7312
```

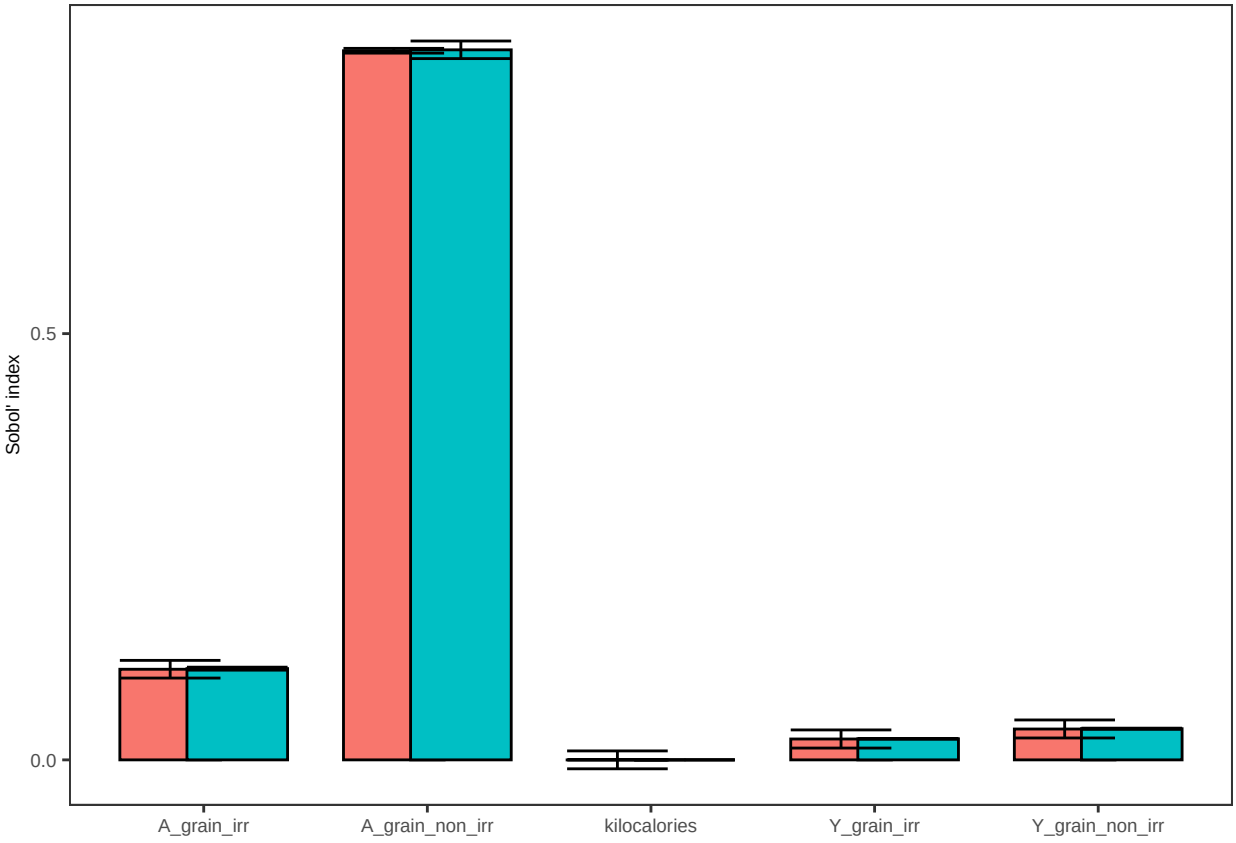
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

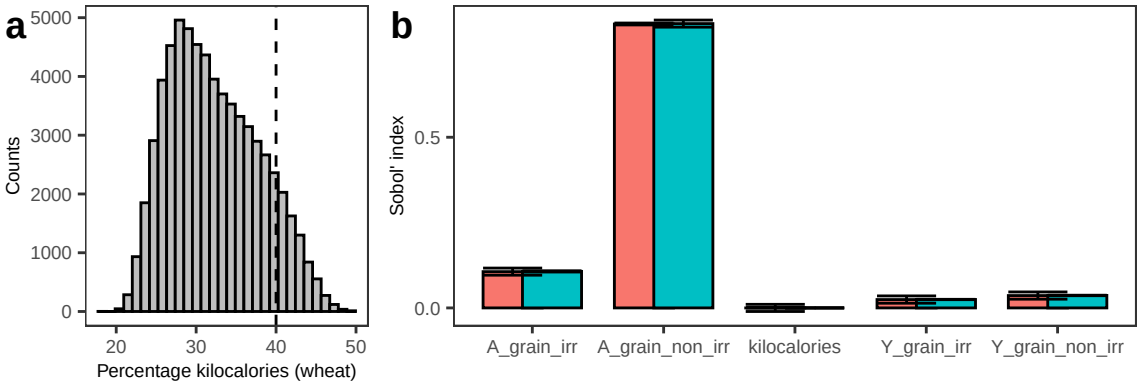

irrigated
rainfed
Counts
Tons
Y_grain_irr
Y_grain_non_irr
A_grain_irr
A_grain_non_irr
kilocalories
Value
count
200
150
100
50

```
##
## First-order estimator: jansen | Total-order estimator: jansen
##
## Total number of model runs: 229376
##
## Sum of first order indices: 0.9987019
##          original          bias    std.error        low.ci      high.ci
##             <num>         <num>        <num>         <num>        <num>
##  1:  2.448036e-02 -2.431507e-05 5.429353e-03  1.386334e-02 3.514601e-02
##  2:  3.620516e-02 -5.207601e-05 5.398008e-03  2.567734e-02 4.683714e-02
##  3:  1.062815e-01 -5.885805e-05 5.292944e-03  9.596635e-02 1.167143e-01
##  4:  8.317764e-01 -2.851686e-05 1.420961e-03  8.290199e-01 8.345900e-01
##  5: -4.153039e-05 -6.041131e-05 5.352747e-03 -1.047231e-02 1.051007e-02
##  6:  2.470173e-02  2.786068e-06 2.278015e-04  2.425246e-02 2.514543e-02
##  7:  3.653244e-02 -2.225513e-06 3.445128e-04  3.585943e-02 3.720990e-02
##  8:  1.070530e-01  3.496198e-05 8.855637e-04  1.052823e-01 1.087537e-01
##  9:  8.328792e-01 -1.880356e-05 5.266639e-03  8.225756e-01 8.432205e-01
## 10:  3.508372e-31 -1.663263e-34 3.330677e-33  3.444756e-31 3.575316e-31
##     sensitivity      parameters
##          <char>          <char>
##  1:          Si     Y_grain_irr
##  2:          Si Y_grain_non_irr
##  3:          Si     A_grain_irr
##  4:          Si A_grain_non_irr
##  5:          Si    kilocalories
##  6:          Ti     Y_grain_irr
##  7:          Ti Y_grain_non_irr
##  8:          Ti     A_grain_irr
##  9:          Ti A_grain_non_irr
## 10:          Ti    kilocalories
```



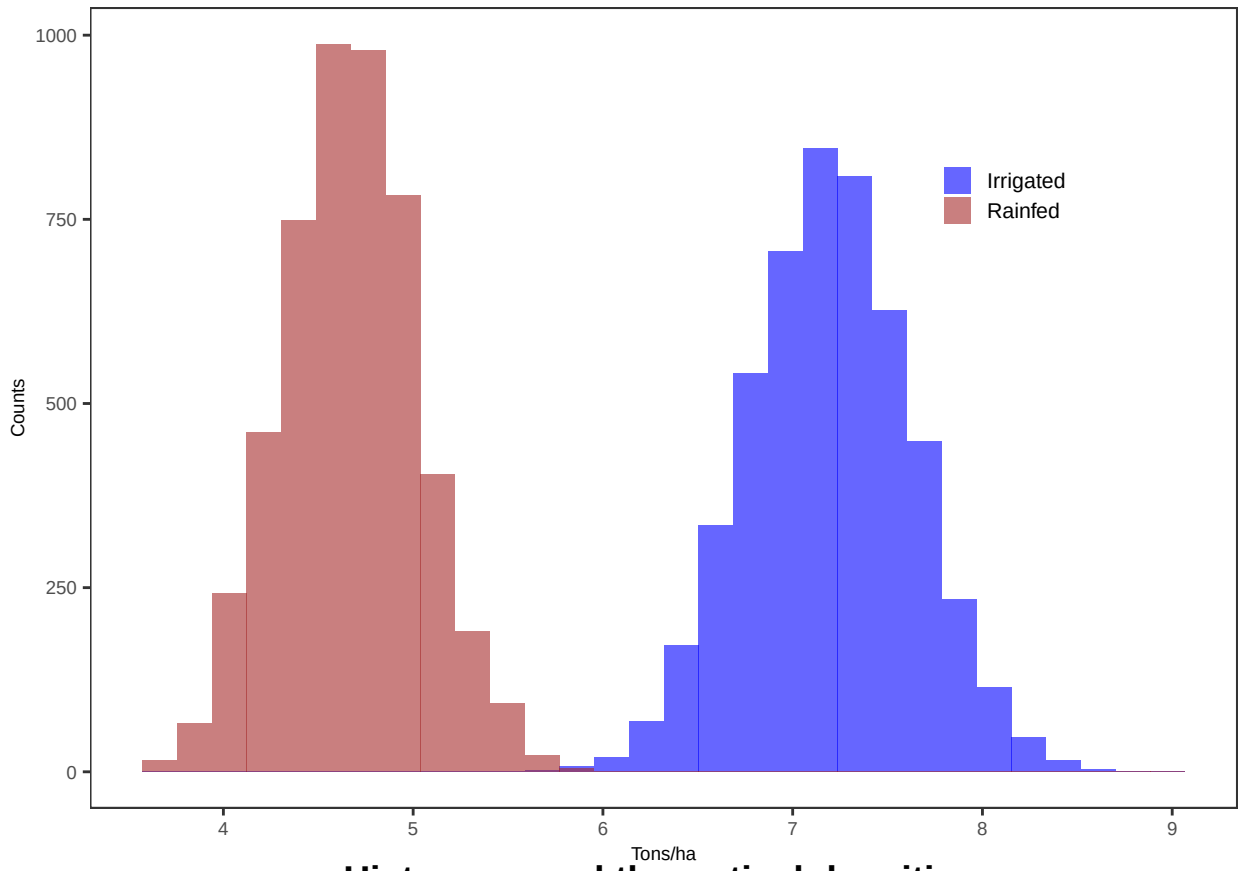

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
# MERGE UNCERTAINTY AND SENSITIVITY PLOTS -------------------------------------

food.ua.sa.plot
```

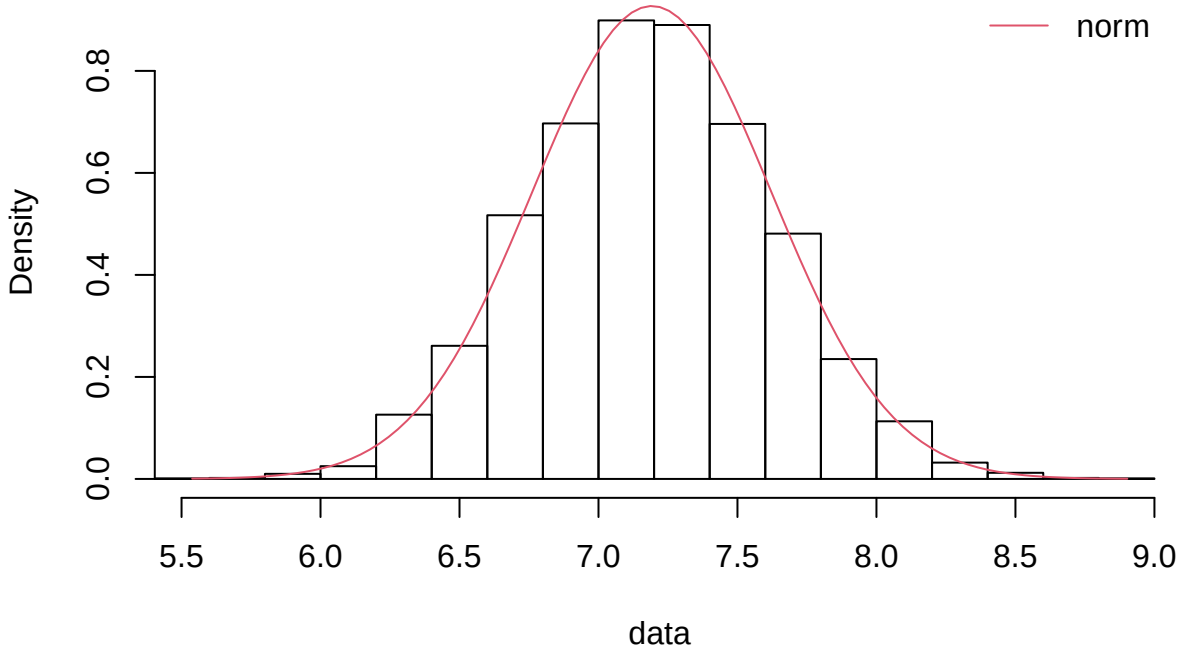


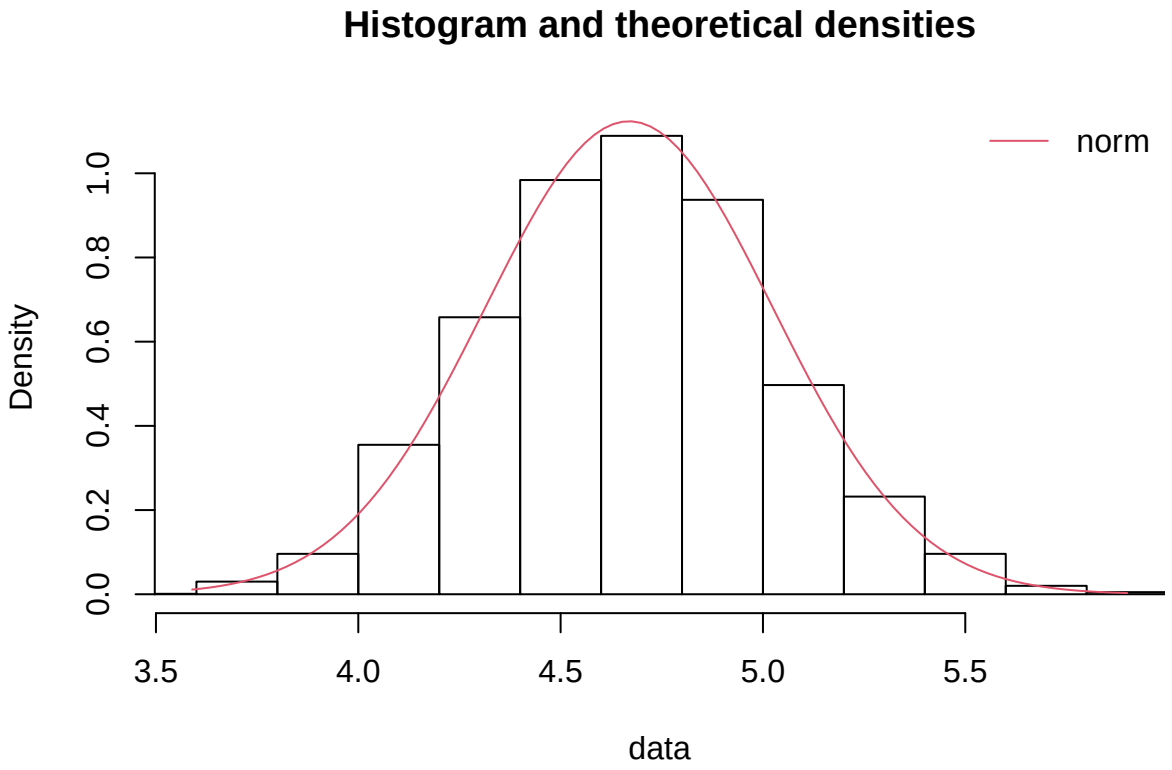

```
## Warning in as.data.table.list(x, keep.rownames = keep.rownames, check.names =
## check.names, : Item 2 has 34 rows but longest item has 43; recycled with
## remainder.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Histogram and theoretical densities






```
## $Y_grain_irr
## $Y_grain_irr$hist
## $breaks
##  [1] 5.4 5.6 5.8 6.0 6.2 6.4 6.6 6.8 7.0 7.2 7.4 7.6 7.8 8.0 8.2 8.4 8.6 8.8 9.0
##
## $counts
##  [1]   1   2  10  25 126 261 517 697 899 890 696 481 235 113  32  12   2   1
##
## $density
##  [1] 0.001 0.002 0.010 0.025 0.126 0.261 0.517 0.697 0.899 0.890 0.696 0.481
## [13] 0.235 0.113 0.032 0.012 0.002 0.001
##
## $mids
##  [1] 5.5 5.7 5.9 6.1 6.3 6.5 6.7 6.9 7.1 7.3 7.5 7.7 7.9 8.1 8.3 8.5 8.7 8.9
##
## $xname
## [1] "mydata"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## $Y_grain_irr$densities
##                [,1]
```

```
##   [1,] 0.0005753772
##   [2,] 0.0007747035
##   [3,] 0.0010367237
##   [4,] 0.0013789074
##   [5,] 0.0018228537
##   [6,] 0.0023950419
##   [7,] 0.0031276563
##   [8,] 0.0040594719
##   [9,] 0.0052367840
##  [10,] 0.0067143565
##  [11,] 0.0085563538
##  [12,] 0.0108372147
##  [13,] 0.0136424142
##  [14,] 0.0170690516
##  [15,] 0.0212261953
##  [16,] 0.0262349070
##  [17,] 0.0322278631
##  [18,] 0.0393484923
##  [19,] 0.0477495508
##  [20,] 0.0575910625
##  [21,] 0.0690375703
##  [22,] 0.0822546603
##  [23,] 0.0974047496
##  [24,] 0.1146421607
##  [25,] 0.1341075405
##  [26,] 0.1559217282
##  [27,] 0.1801792143
##  [28,] 0.2069413818
##  [29,] 0.2362297593
##  [30,] 0.2680195521
##  [31,] 0.3022337458
##  [32,] 0.3387380922
##  [33,] 0.3773372913
##  [34,] 0.4177726691
##  [35,] 0.4597216206
##  [36,] 0.5027990403
##  [37,] 0.5465608978
##  [38,] 0.5905100358
##  [39,] 0.6341041780
##  [40,] 0.6767660332
##  [41,] 0.7178952817
##  [42,] 0.7568821296
##  [43,] 0.7931220257
##  [44,] 0.8260310587
##  [45,] 0.8550614959
##  [46,] 0.8797168902
##  [47,] 0.8995661756
##  [48,] 0.9142561936
```

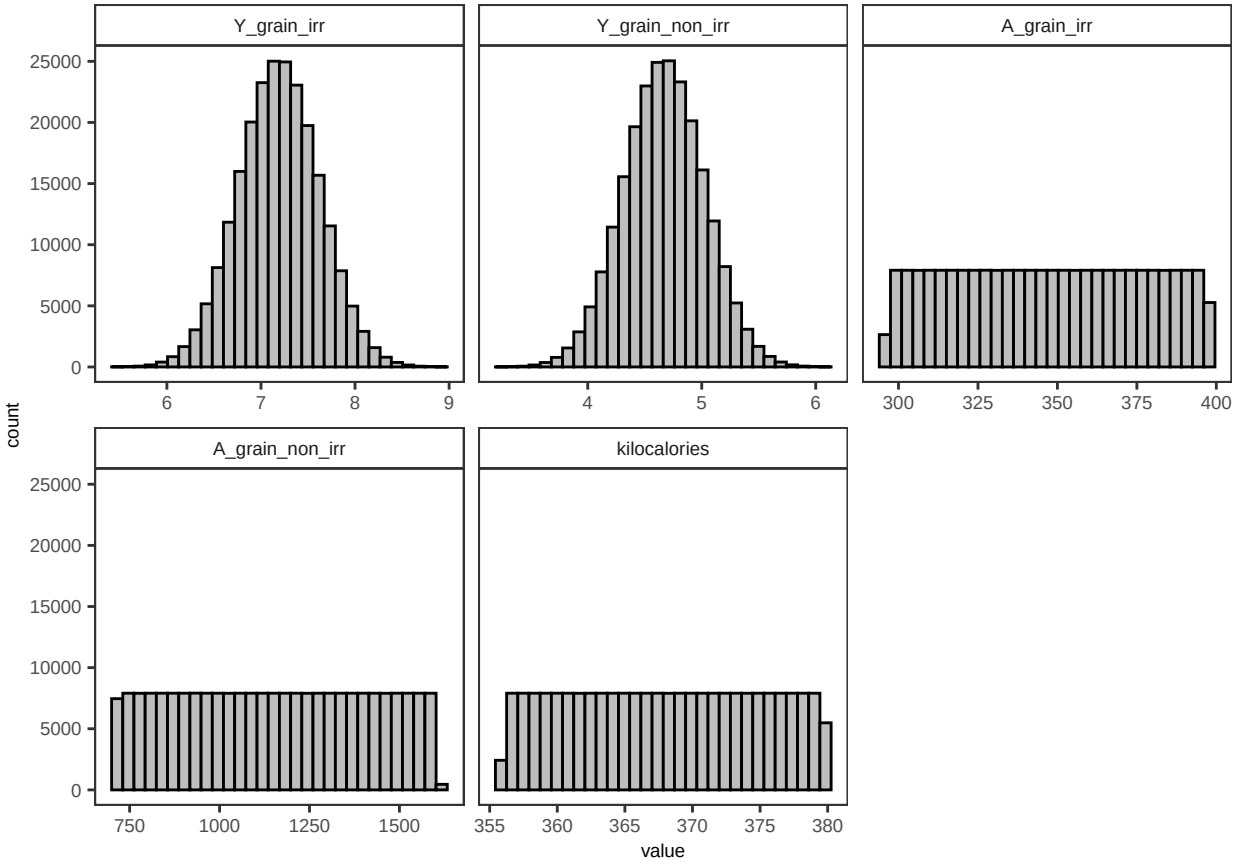
```
##  [49,] 0.9235221410
##  [50,] 0.9271955094
##  [51,] 0.9252091820
##  [52,] 0.9175994733
##  [53,] 0.9045050251
##  [54,] 0.8861626047
##  [55,] 0.8628999849
##  [56,] 0.8351262032
##  [57,] 0.8033196057
##  [58,] 0.7680141636
##  [59,] 0.7297846075
##  [60,] 0.6892309571
##  [61,] 0.6469630228
##  [62,] 0.6035854328
##  [63,] 0.5596836848
##  [64,] 0.5158116555
##  [65,] 0.4724809062
##  [66,] 0.4301520338
##  [67,] 0.3892282069
##  [68,] 0.3500509313
##  [69,] 0.3128979921
##  [70,] 0.2779834380
##  [71,] 0.2454594034
##  [72,] 0.2154195101
##  [73,] 0.1879035565
##  [74,] 0.1629031842
##  [75,] 0.1403682074
##  [76,] 0.1202133059
##  [77,] 0.1023248071
##  [78,] 0.0865673126
##  [79,] 0.0727899705
##  [80,] 0.0608322329
##  [81,] 0.0505289856
##  [82,] 0.0417149769
##  [83,] 0.0342285139
##  [84,] 0.0279144249
##  [85,] 0.0226263205
##  [86,] 0.0182282002
##  [87,] 0.0145954757
##  [88,] 0.0116154843
##  [89,] 0.0091875772
##  [90,] 0.0072228614
##  [91,] 0.0056436773
##  [92,] 0.0043828809
##  [93,] 0.0033829983
##  [94,] 0.0025953055
##  [95,] 0.0019788816
##  [96,] 0.0014996701
```

```
##  [97,] 0.0011295781
##  [98,] 0.0008456320
##  [99,] 0.0006292035
## [100,] 0.0004653133
## [101,] 0.0003420145
##
##
## $Y_grain_non_irr
## $Y_grain_non_irr$hist
## $breaks
##  [1] 3.4 3.6 3.8 4.0 4.2 4.4 4.6 4.8 5.0 5.2 5.4 5.6 5.8 6.0
##
## $counts
##  [1]    1   30   96  355  658  984 1089  937  497  232   96   20    5
##
## $density
##  [1] 0.001 0.030 0.096 0.355 0.658 0.984 1.089 0.937 0.497 0.232 0.096 0.020
## [13] 0.005
##
## $mids
##  [1] 3.5 3.7 3.9 4.1 4.3 4.5 4.7 4.9 5.1 5.3 5.5 5.7 5.9
##
## $xname
## [1] "mydata"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## $Y_grain_non_irr$densities
##               [,1]
##   [1,] 0.011052593
##   [2,] 0.013442428
##   [3,] 0.016279899
##   [4,] 0.019632977
##   [5,] 0.023576591
##   [6,] 0.028192678
##   [7,] 0.033570056
##   [8,] 0.039804141
##   [9,] 0.046996430
##  [10,] 0.055253772
##  [11,] 0.064687360
##  [12,] 0.075411462
##  [13,] 0.087541852
##  [14,] 0.101193947
##  [15,] 0.116480650
```

```
##  [16,] 0.133509899
##  [17,] 0.152381972
##  [18,] 0.173186545
##  [19,] 0.195999582
##  [20,] 0.220880095
##  [21,] 0.247866853
##  [22,] 0.276975122
##  [23,] 0.308193524
##  [24,] 0.341481118
##  [25,] 0.376764797
##  [26,] 0.413937124
##  [27,] 0.452854697
##  [28,] 0.493337140
##  [29,] 0.535166827
##  [30,] 0.578089387
##  [31,] 0.621815073
##  [32,] 0.666021014
##  [33,] 0.710354370
##  [34,] 0.754436372
##  [35,] 0.797867211
##  [36,] 0.840231685
##  [37,] 0.881105535
##  [38,] 0.920062314
##  [39,] 0.956680655
##  [40,] 0.990551774
##  [41,] 1.021287005
##  [42,] 1.048525191
##  [43,] 1.071939735
##  [44,] 1.091245104
##  [45,] 1.106202624
##  [46,] 1.116625389
##  [47,] 1.122382146
##  [48,] 1.123400050
##  [49,] 1.119666194
##  [50,] 1.111227883
##  [51,] 1.098191638
##  [52,] 1.080720955
##  [53,] 1.059032894
##  [54,] 1.033393594
##  [55,] 1.004112833
##  [56,] 0.971537813
##  [57,] 0.936046312
##  [58,] 0.898039423
##  [59,] 0.857934052
##  [60,] 0.816155378
##  [61,] 0.773129466
##  [62,] 0.729276203
##  [63,] 0.685002721
```

```
##  [64,] 0.640697439
##  [65,] 0.596724841
##  [66,] 0.553421064
##  [67,] 0.511090360
##  [68,] 0.470002461
##  [69,] 0.430390833
##  [70,] 0.392451804
##  [71,] 0.356344521
##  [72,] 0.322191648
##  [73,] 0.290080750
##  [74,] 0.260066240
##  [75,] 0.232171808
##  [76,] 0.206393221
##  [77,] 0.182701372
##  [78,] 0.161045513
##  [79,] 0.141356533
##  [80,] 0.123550236
##  [81,] 0.107530512
##  [82,] 0.093192354
##  [83,] 0.080424670
##  [84,] 0.069112839
##  [85,] 0.059140994
##  [86,] 0.050394012
##  [87,] 0.042759212
##  [88,] 0.036127748
##  [89,] 0.030395726
##  [90,] 0.025465054
##  [91,] 0.021244040
##  [92,] 0.017647779
##  [93,] 0.014598339
##  [94,] 0.012024783
##  [95,] 0.009863056
##  [96,] 0.008055754
##  [97,] 0.006551811
##  [98,] 0.005306118
##  [99,] 0.004279105
## [100,] 0.003436287
## [101,] 0.002747807
```

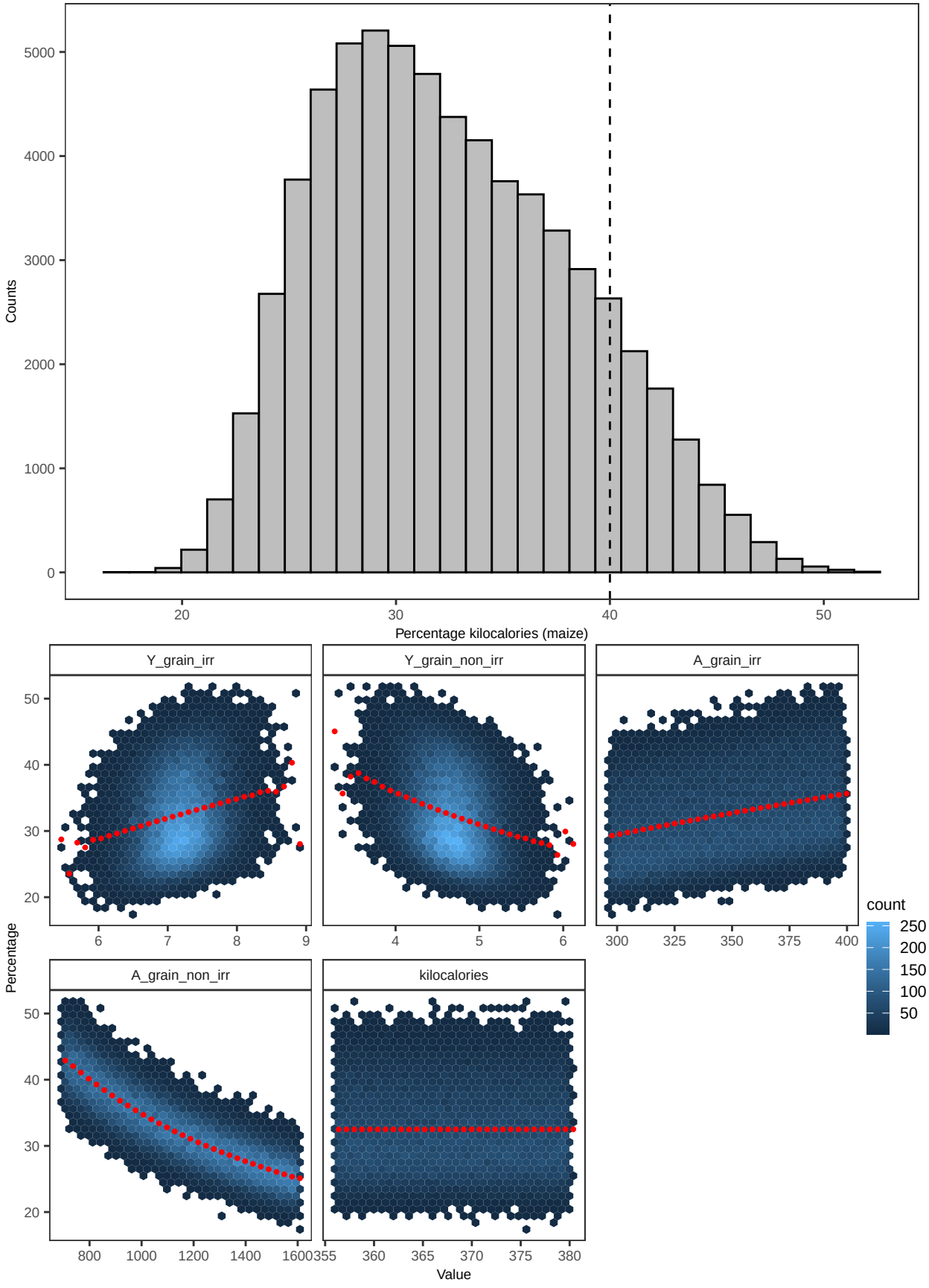
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
##         min     max
##       <num>   <num>
## 1: 17.04788 52.1339
```

```
##     2.5%    97.5%
## 23.01091 44.48328
```

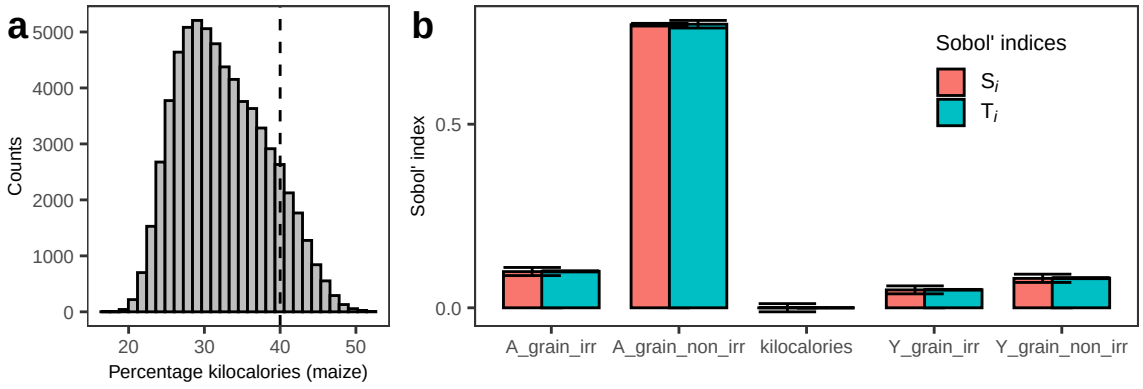
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


Counts
5000
4000
3000
2000
1000
0
20
30
40
50
Percentage kilocalories (maize)
Y_grain_irr
Y_grain_non_irr
A_grain_irr
A_grain_non_irr
kilocalories
Percentage
Value
count
250
200
150
100
50

```
##
## First-order estimator: jansen | Total-order estimator: jansen
##
## Total number of model runs: 229376
##
## Sum of first order indices: 0.9981829
##          original          bias    std.error        low.ci      high.ci
##             <num>         <num>        <num>         <num>        <num>
##  1:  4.878648e-02 -5.940199e-05 5.530045e-03  3.800719e-02 5.968457e-02
##  2:  8.007553e-02 -2.486986e-04 5.733058e-03  6.908764e-02 9.156081e-02
##  3:  9.851706e-02 -2.828815e-04 5.652851e-03  8.772056e-02 1.098793e-01
##  4:  7.708648e-01 -6.173398e-05 1.855171e-03  7.672905e-01 7.745626e-01
##  5: -6.101580e-05 -2.617949e-04 5.699555e-03 -1.097014e-02 1.137170e-02
##  6:  4.923524e-02  1.692489e-05 4.479384e-04  4.834037e-02 5.009626e-02
##  7:  8.076036e-02 -4.010463e-05 7.168479e-04  7.939547e-02 8.220546e-02
##  8:  9.929425e-02  1.295296e-05 7.890986e-04  9.773469e-02 1.008279e-01
##  9:  7.723314e-01  8.955537e-05 5.265786e-03  7.619210e-01 7.825625e-01
## 10:  3.145563e-31 -1.169291e-34 3.082284e-33  3.086321e-31 3.207144e-31
##     sensitivity      parameters
##          <char>          <char>
##  1:          Si     Y_grain_irr
##  2:          Si Y_grain_non_irr
##  3:          Si     A_grain_irr
##  4:          Si A_grain_non_irr
##  5:          Si    kilocalories
##  6:          Ti     Y_grain_irr
##  7:          Ti Y_grain_non_irr
##  8:          Ti     A_grain_irr
##  9:          Ti A_grain_non_irr
## 10:          Ti    kilocalories
```

```
# MERGE UNCERTAINTY AND SENSITIVITY PLOTS -------------------------------------

plot_grid(plot.uncertainty, plot.indices, ncol = 2, labels = "auto",
          rel_widths = c(0.35, 0.65))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



11 Session information

```
# SESSION INFORMATION ############################################################

sessionInfo()
```

```
## R version 4.3.3 (2024-02-29)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Sonoma 14.2.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/London
## tzcode source: internal
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] fitdistrplus_1.1-11 survival_3.5-8      MASS_7.3-60.0.1
##  [4] readxl_1.4.2        ncdf4_1.23          raster_3.6-26
##  [7] sp_1.6-0            sensobol_1.1.5      doParallel_1.0.17
## [10] iterators_1.0.14    foreach_1.5.2       countrycode_1.5.0
## [13] scales_1.3.0        wesanderson_0.3.6   benchmarkme_1.0.8
## [16] tidygraph_1.3.0     cowplot_1.1.3       ggraph_2.1.0
## [19] igraph_2.0.3        bibliometrix_4.0.1  lubridate_1.9.3
## [22] forcats_1.0.0       stringr_1.5.1       dplyr_1.1.4
## [25] purrr_1.0.2         readr_2.1.4         tidyr_1.3.1
## [28] tibble_3.2.1        ggplot2_3.5.1       tidyverse_2.0.0
## [31] data.table_1.14.99  openxlsx_4.2.5.2
##
## loaded via a namespace (and not attached):
##  [1] Rdpack_2.6             gridExtra_2.3          rlang_1.1.4
##  [4] magrittr_2.0.3         tidytext_0.4.1         compiler_4.3.3
##  [7] vctrs_0.6.5            pkgconfig_2.0.3        fastmap_1.1.1
## [10] ellipsis_0.3.2         utf8_1.2.4             promises_1.2.0.1
## [13] rmarkdown_2.21         tzdb_0.3.0             xfun_0.39
## [16] jsonlite_1.8.4         flashClust_1.01-2      SnowballC_0.7.1
## [19] later_1.3.0            tweenr_2.0.2           terra_1.7-78
## [22] cluster_2.1.6          R6_2.5.1               stringi_1.8.3
## [25] RColorBrewer_1.1-3     cellranger_1.1.0       estimability_1.4.1
## [28] Rcpp_1.0.12            knitr_1.42             filehash_2.4-5
```

```
## [31] splines_4.3.3          httpuv_1.6.9           rentrez_1.2.3
## [34] Matrix_1.6-5           timechange_0.2.0       tidyselect_1.2.0
## [37] rstudioapi_0.15.0      stringdist_0.9.10      pubmedR_0.0.3
## [40] yaml_2.3.7             viridis_0.6.4          codetools_0.2-19
## [43] lattice_0.22-5         plyr_1.8.8             shiny_1.7.4
## [46] withr_3.0.0            benchmarkmeData_1.0.4  coda_0.19-4
## [49] evaluate_0.20          polyclip_1.10-6        zip_2.3.0
## [52] pillar_1.9.0           janeaustenr_1.0.0      DT_0.27
## [55] plotly_4.10.1          generics_0.1.3         hms_1.1.3
## [58] munsell_0.5.1          xtable_1.8-4           leaps_3.1
## [61] glue_1.7.0             tikzDevice_0.12.4      emmeans_1.8.5
## [64] scatterplot3d_0.3-43   lazyeval_0.2.2         tools_4.3.3
## [67] tokenizers_0.3.0       mvtnorm_1.1-3          graphlayouts_1.1.1
## [70] XML_3.99-0.14          grid_4.3.3             rbibutils_2.2.16
## [73] rscopus_0.6.6          colorspace_2.1-0       dimensionsR_0.0.3
## [76] ggforce_0.4.1          bibliometrixData_0.3.0 cli_3.6.3
## [79] fansi_1.0.6            viridisLite_0.4.2      gtable_0.3.5
## [82] digest_0.6.34          ggrepel_0.9.5          FactoMineR_2.8
## [85] htmlwidgets_1.6.2      farver_2.1.2           htmltools_0.5.5
## [88] factoextra_1.0.7       lifecycle_1.0.4        httr_1.4.5
## [91] multcompView_0.1-9     mime_0.12
```

```
## Return the machine CPU ------------------------------------------------

cat("Machine:     "); print(get_cpu()$model_name)
```

```
## Machine:

## [1] "Apple M1 Max"
```

```
## Return number of true cores -------------------------------------------

cat("Num cores:   "); print(detectCores(logical = FALSE))
```

```
## Num cores:

## [1] 10
```

```
## Return number of threads ----------------------------------------------

cat("Num threads: "); print(detectCores(logical = FALSE))
```

```
## Num threads:

## [1] 10
```