Main Program for Final Project

Rebecca Thomson

2024-07-16

This is the main program for analyzing District Maps.
'Input_Analysis.Rmd' will analyze and check input data.

Libraries

These are all the libraries used for the project. We have tried to comment which part they were first called for use.

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

```
#library(Rtools)
library(redist) #Map Analysis
```

```
## Loading required package: redistmetrics
##
## Attaching package: 'redistmetrics'
##
## The following object is masked from 'package:dplyr':
##
##     tally
##
##
## Attaching package: 'redist'
##
## The following object is masked from 'package:stats':
##
##     filter
```

```
library(dplyr)
library(ggplot2)
library(alarmdata) #State graph/mapping data
library(ggredist) #Alarm data mapping
library(rlist)
```

Global Provided Information and Constraints

The global information and constraints are not changed during any algorithm or analysis, except addition of proposed Congressional Districts when the creation algorithm is completed. This information will be modified for each state analyzed.

These are the variables that are global: -Loading initial state information (selecting which state to analyze) -The list of Final District node choices. It is a list of lists of what nodes are in created districts, it shall be modified by our analysis and starts out empty.

These are the constraints that are global: -Required number of Districts -The range of population values allowed in each district. Historically, this is plus or minus 1% of the Total Population/Number of Districts.

```
#Selecting state and entering base information
#This_State<-read.csv(file='ks_2020_vtd.txt')
This_state_map<-alarm_50state_map('IL')

#Code in number of Districts
number_districts<- 17 #Hard-coded number of Congressional Districts, the minimum is two.

#Total Population
tot_pop<-sum(This_state_map$pop)

Population_Range<-c((tot_pop/number_districts)-0.01*(tot_pop/number_districts), (tot_pop/number_
districts)+0.01*(tot_pop/number_districts) )
```

Create list of Nodes Function

This function produces an original list of all nodes. It is a list of lists, where each list is: [[(original node index(integer), repeated n times [],[],[]…],
[raw population number of node(integer)], [list of adjacent nodes by original index number[],[],[]….], [0 (for later tracking in other functions, integer)]]

The purpose of this function is to: -Protect the original data from loss or corruption. -Isolate each node into its own list for ease of analysis. This will make creating lists of nodes within districts and lists of adjacent nodes easier. -The repeated naming of the node in [[1]] is because nodes need to be removed from the previous district’s adjacency list when a new district is made.

```
create_node_list<-function(){
all_nodes_adjlist<- This_state_map$adj
all_nodes_poplist<-This_state_map$pop
all_nodes_index<-1:(length(This_state_map$pop))
all_nodes_all_list<-list()
for (i in 1:(length(all_nodes_poplist))){
  temp_list<-list()
  temp_name<-1:number_districts
  for (k in 1:number_districts){temp_name[k]<-all_nodes_index[[i]]}
  temp_list[[length(temp_list)+1]]<-temp_name
  temp_list[[length(temp_list)+1]]<-all_nodes_poplist[i]
  check_adj<-all_nodes_adjlist[[i]]
  for (j in length(check_adj):1){
    if (as.numeric(check_adj[j])==as.numeric(all_nodes_index[[i]])){
      check_adj[j]<-NA
    }
  }
  temp_list[[length(temp_list)+1]]<-check_adj

  temp_list[length(temp_list)+1]<-c(0)

  all_nodes_all_list[[length(all_nodes_all_list)+1]]<-temp_list

}
return (all_nodes_all_list)
}
```

Created Global Information

```
#Iterate-able list of all nodes.
GLOBALNODESLIST<-create_node_list()
```

R function to Remove a node from the Adjacency list

This function takes the node to be removed from the adj.list. The function should be used when a node is added to the district being built. The function removes the node from the adjacency list, removes any mention of the node in other nodes' lists, and increases the count of adjacencies within those node's adjacency lists- because now the district has one more adjacency.

```
node.to.remove<-function(remove.this.node){

  #First, the node must be removed from other node's adjacency lists.  We do not need to keep tr
ack of which nodes are adjacent to our district sub-graph, because all nodes in the adjacency li
st are connected to the district.
  #Second, while we are removing the node from the adjacent node's adjacency lists, we need to u
pdate the Number of Connections each exterior and adjacent node has to the district sub-graph.

  for (i in (length(adj.list)):1){
    if(is.na(adj.list[i])){next}
    for (j in (length(adj.list[[i]][[3]])):1){
      if (is.na(adj.list[[i]][[3]][j])){next}
      if (as.numeric(adj.list[[i]][[3]][j])==as.numeric(remove.this.node)){
        adj.list[[i]][[3]][j]<<- NA #
        adj.list[[i]][[4]][[1]]<<-1+as.numeric(adj.list[[i]][[4]][[1]])

      }
    }
  }

}
```

R function to add a single node to Adjacency List

GLOBALNODELIST is the Global-level list that is a modifiable list of all the nodes that have not yet been used to build districts. It shall be modified with a different function. However, the indexing is the same as the node name at all times- any modification is just setting the node name to ‘na’ so that we can continue to use the list without excessive lookup times.

This function shall copy the node from the exterior node list and place it in the adjacency list. It shall return nothing, since adj.list is a global variable that is set to zero when each new district is created.

```
add.node<-function(new.adj.node){
  #building individual node list to bring into adjacency list.  Need to check that node has not
been placed in another district.
  if (!is.na(GLOBALNODESLIST[[new.adj.node]][[1]][[m]])){
  temp1<-list()
  temp1[[length(temp1)+1]]<-(as.numeric(GLOBALNODESLIST[[new.adj.node]][[1]][[m]]))

  temp1[[length(temp1)+1]]<-(as.numeric(GLOBALNODESLIST[[new.adj.node]][[2]]))
  temp1[[length(temp1)+1]]<-GLOBALNODESLIST[[new.adj.node]][[3]]
  #I have found that there are errors in the input data, that cause 'loops' of the same node bei
ng added over and over.  This should remove any node that is listed as adjacent to itself in its
adjacency list
  for (j in length(temp1[[3]]):1){
    if (is.na(temp1[[3]][j])){next}
    if (temp1[[3]][j]==as.numeric(new.adj.node)){
        temp1[[3]][j]<<- NA }}
  #Remember that these nodes are now connected to the district list, and should start with +1 ad
jacency.
  temp1[[length(temp1)+1]]<-(as.numeric(GLOBALNODESLIST[[new.adj.node]][[4]])+1)

  adj.list[[(length(adj.list)+1)]]<<-temp1 #Add node to adjacency
  #And remove the name of node from the exterior list, this run only
  exterior.node.removal(new.adj.node)
      }}
```

#Function to remove node from possible exterior nodes list. When a node is selected for the adjacency list, run this code to remove the node from the exterior node list. It does not remove this node from the list for the next run. The exterior node list is GLOBALNODESLIST, and it is a Global variable. Nothing is returned.

```
exterior.node.removal<-function(removed.node){
  #By leaving the Global list in place, except for the node #, we may minimize lookup time.
      GLOBALNODESLIST[[removed.node]][[1]][[m]]<<- NA #Remove node from
}
```

Function to add node to District list

This function shall take in the node to be added. It shall add the node to the district list. The district.temp list is a global variable'.

```
node.to.add<-function(add.this.node){
  #district.temp
  #Build the list of lists to add to district list
  for (i in (length(adj.list)):1){
    if(as.numeric(adj.list[[i]][[1]])!=0 && !is.na(adj.list[[i]][[1]])){
    if(as.numeric(adj.list[[i]][[1]])==(as.numeric(add.this.node))){
      #building individual node list to bring into district list


      temp1<-list()
      temp1[[length(temp1)+1]]<-(as.numeric(adj.list[[i]][[1]]))

      temp1[[length(temp1)+1]]<-(as.numeric(adj.list[[i]][[2]]))
      temp1[[length(temp1)+1]]<-adj.list[[i]][[3]]
      #Send new nodes to adj. list to be added
      adj.list.update(adj.list[[i]][[3]])

      temp1[[length(temp1)+1]]<-(as.numeric(adj.list[[i]][[4]]))

      district.temp[[length(district.temp)+1]]<<-temp1 #Add node to district
      temp.pop<<-temp.pop+adj.list[[i]][[2]]#Add population, so district will stop growing


      #Remove from adjacency list
      adj.list[[i]][[1]]<<-NA

    }}
  }
  node.to.remove(add.this.node)
  #Remove from exterior list in all cases
  for (x in m:number_districts){
  GLOBALNODESLIST[[add.this.node]][[1]][[x]]<<-NA}
}
```

Function to pick a new node

This function will pick a new node for the proposed district by looking at the adjacency list and picking the node with the largest percentage of adjacencies to the proposed district. In the case of a tie, it will use the queue structure inherit in the adjacency list.

```
pick.new.node<- function(){
  node.number=-99
 #Prioritize nodes with max adjacency with the temp district

  temp.max<-0
  #First, we will grab any node that has 100% of it's adjacency nodes already removed.
  for (i in 1:length(adj.list)){
    #No zeros or NA nodes
    if (is.na(adj.list[[i]][[1]])||as.numeric(adj.list[[i]][[1]]==0)){next}
    temp.length<-length(adj.list[[i]][[3]])
    temp.nacount<-sum(is.na(adj.list[[i]][[3]]))
    #If 75% or more of the adjacency to this node is all ready picked, pick this node first OR i
t only has three or more nodes it is connected to at 50% OR 1 nodes only
    if((temp.nacount/temp.length)>=0.75||(length(adj.list[[i]][[3]])<=3 && (temp.nacount/temp.le
ngth)>=0.5)||length(adj.list[[i]][[3]])==1){
      node.number<-adj.list[[i]][[1]]

      break
    }
    }
  #If the above is not met, we will continue to analyze nodes
  for (i in 1:length(adj.list)){
    #No zeros or NA nodes
    if (is.na(adj.list[[i]][[1]])||as.numeric(adj.list[[i]][[1]]==0)||node.number!=-99){next}
    temp.length<-length(adj.list[[i]][[3]])
    temp.nacount<-sum(is.na(adj.list[[i]][[3]]))

    #Pick the node that is the most connected already.
    if (adj.list[[i]][[4]]>=temp.max && !is.na(adj.list[[i]][[1]])){
      if(temp.pop+as.integer(adj.list[[i]][[2]])>Population_Range[2]){next}
      temp.max<-adj.list[[i]][[4]]
      node.number<-adj.list[[i]][[1]]
    }
  }

   if (node.number==-99){#if max not found, use queue from list.
     for (i in length(adj.list):1){
       #No zeros or NA nodes
       if (is.na(adj.list[[i]][[1]])||as.numeric(adj.list[[i]][[1]]==0)){next}
       if (!is.na(adj.list[[i]][[1]])){
       if(temp.pop+as.integer(adj.list[[i]][[2]])>Population_Range[2]){pass}
       node.number<-adj.list[[i]][[1]]
       break
         }
       }

  }
  return (node.number)
  }
```

Function to update the Adjacency List when node add to district

This function is given a list of the nodes that were adjacent to the node just added to the district. It will check if these nodes are within the adj.list, and if not, will add them.

```
adj.list.update<-function(list.new.adj.nodes){
  for (k in 1:(length(list.new.adj.nodes))){ #Loop through nodes to check
    if (is.na(list.new.adj.nodes[k])){next}
    It.is.here<-0

    for (i in 1: length(adj.list)){#for each node, check adj.list
      if(is.na(adj.list[[i]][[1]])){next}
      if(as.numeric(list.new.adj.nodes[k])==as.numeric(adj.list[[i]][[1]])){
        It.is.here=1
        #Update count
        adj.list[[i]][[4]][[1]]<<-as.numeric(adj.list[[i]][[4]][[1]])+1
        next
      }

    }
    #Node can neither be present or na in Global list to be added.
    if (It.is.here==0 && !is.na(GLOBALNODESLIST[[list.new.adj.nodes[k]]][[1]][[m]])){ #add the n
ew node in the adjacency list
      add.node(list.new.adj.nodes[k])
    }
  }

}
```

Creating a single District

```
make.district<-function(){
  temp.pop<<-0
  #Re-Create this district node list.  It is a global variable, for ease of use between function
s,
  #but needs to be set to empty for each 'run'
  #Re-Create this district adjacency list.  It is a global variable, for ease of use between fun
ctions,
  #but needs to be set to empty for each 'run'

temp_list<-list()
temp_list[[length(temp_list)+1]]<-c(0)
temp_list[[length(temp_list)+1]]<-c(0)
temp_list[[length(temp_list)+1]]<-c(0)
temp_list[length(temp_list)+1]<-c(0)
district.temp<<-list()
district.temp[[(length(district.temp)+1)]]<<-temp_list
adj.list<<-list()
adj.list[[(length(adj.list)+1)]]<<-temp_list

#Add node to district

  #Pick start Node, use very first valid node
  First.node<-1
  for (k in length(GLOBALNODESLIST):1){
    if (!is.na(GLOBALNODESLIST[[k]][[1]][[m]])){
      First.node<-as.numeric(GLOBALNODESLIST[[k]][[1]][[m]])
      break}

  }

  #Move  First Node to this district Node List


  temp1<-list()
  temp1[[length(temp1)+1]]<-(as.numeric(GLOBALNODESLIST[[First.node]][[1]][[m]]))

  temp1[[length(temp1)+1]]<-(as.numeric(GLOBALNODESLIST[[First.node]][[2]]))
  temp1[[length(temp1)+1]]<-GLOBALNODESLIST[[First.node]][[3]]

  temp1[[length(temp1)+1]]<-(as.numeric(GLOBALNODESLIST[[First.node]][[4]]))

  district.temp[[1]]<<-temp1 #Add node to district

  #Add nodes to Adjacency List

  First.adj<<-GLOBALNODESLIST[[First.node]][[3]]

  adj.list.update(First.adj)
```

```
  #Remove from exterior list in all cases
  for (x in m:number_districts){
  GLOBALNODESLIST[[First.node]][[1]][[x]]<<-NA}

  while (temp.pop<Population_Range[1]+1000){
    Next.node<-pick.new.node()
    #print(paste("Node no. ",Next.node," picked."))
    if(Next.node==-99){break}
    #Add node to district.temp, remove from adj.list
    node.to.add(Next.node)

  }
  #Add district.temp to overall list.
  tot.dist[[length(tot.dist)+1]]<<-district.temp
}
```

“Global Created Information (”Main”)

The created information here is the heart of the program. It will create lists of nodes and drive functions that will create districts.

```
#make some global variables
tot.dist<-list()
temp_list<-list()
temp_list[[length(temp_list)+1]]<-c(0)
temp_list[[length(temp_list)+1]]<-c(0)
temp_list[[length(temp_list)+1]]<-c(0)
temp_list[length(temp_list)+1]<-c(0)
district.temp<-list()
district.temp[[length(district.temp)+1]]<-temp_list
adj.list<-list()
temp.pop<-0
adj.list[[length(adj.list)+1]]<-temp_list
m<-1
pop.per.district<-1:number_districts
for (j in 1:number_districts){
  m<-j
  make.district()
  pop.per.district[j]<-temp.pop
}
```

Matching my numbers up to GEOID and creating a district map.

This section will take the tot.district list and assign districts to each GEOID on the original This_state_map. We will do this by making a list that is total nodes long, assigning each index space to it’s district #. (All node numbers in district 1 list shall be assigned a 1) This list will then be appended to This_state_map. Then we can compare our districts to the redist results.

```
brute_district<-replicate(length(This_state_map$pop),as.integer(number_districts))
for (j in 1:length(tot.dist)){
  for (i in 1:length(tot.dist[[j]])){
    brute_district[as.numeric(tot.dist[[j]][[i]][[1]])]<-as.integer(j)
  }
}
#Because the redist object won't let us analyze new columns of distracting without extensive dat
a-type manipulating, we will simply modify the cd_2020 district.
for (k in 1:length(This_state_map$pop)){
  This_state_map$cd_2020[[k]]<-as.integer(brute_district[k])

}
```

Now, we run the redist functions to make our group of comparison maps

```
#This
This_plans<-redist_smc(This_state_map,1000,compactness = 1,runs = 2 )
print(This_plans)
```

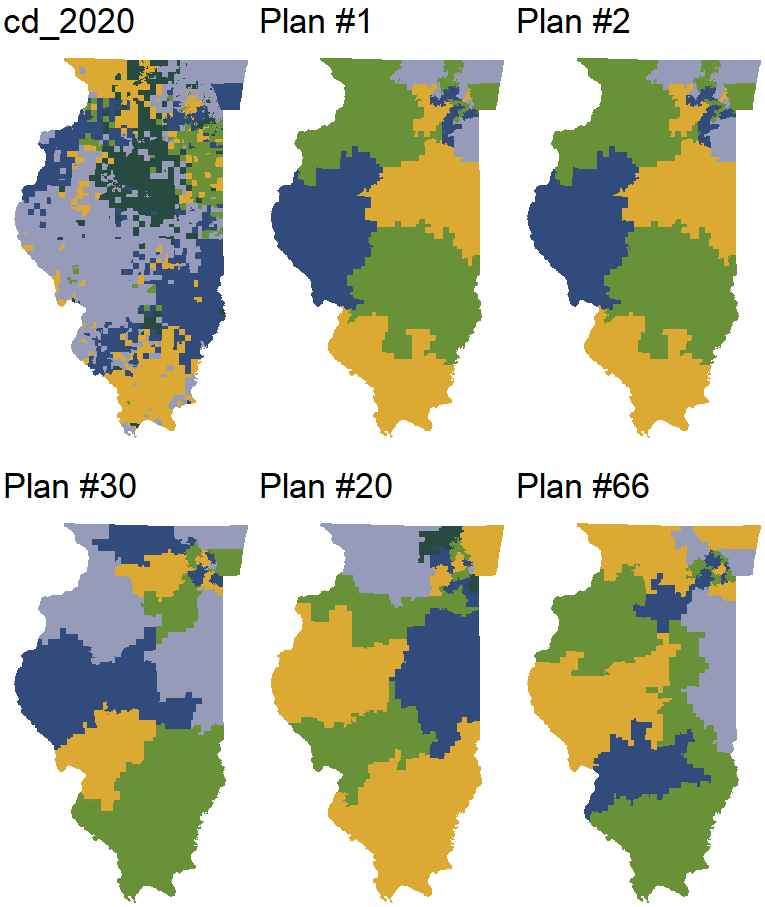
```
## A <redist_plans> containing 2,000 sampled plans and 1 reference plan
```

```
## Plans have 17 districts from a 10,084-unit map, and were drawn using Sequential
## Monte Carlo.
```

```
## With plans resampled from weights
## Plans matrix: int [1:10084, 1:2001] 1 2 2 2 2 2 2 2 1 2 ...
## # A tibble: 34,017 × 4
##    draw    district total_pop chain
##    <fct>      <int>     <dbl> <int>
##  1 cd_2020        1    747772    NA
##  2 cd_2020        2   7534371    NA
##  3 cd_2020        3    748749    NA
##  4 cd_2020        4    750264    NA
##  5 cd_2020        5    749691    NA
##  6 cd_2020        6    749781    NA
##  7 cd_2020        7    750421    NA
##  8 cd_2020        8    749512    NA
##  9 cd_2020        9      2346    NA
## 10 cd_2020       10     16260    NA
## # ℹ 34,007 more rows
```

Here is a map comparing some of the redist maps

```
redist.plot.plans(This_plans, draws=c("cd_2020","1","2","30","20","66"), shp=This_state_map)
```




```
#Matching district numbers

This_plans_today<-match_numbers(This_plans,This_state_map$cd_2020)

this_county_perims2020<- prep_perims(This_state_map, This_state_map$adj)
```

```
## Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.3.1; sf_use_s2() is TRUE
```

```
#we will be analyze the population deviation, the perimeter-based compactness measure, and the f
raction of minority voters and two-party Democratic vote share by district.

This_plans_today = This_plans_today %>%
  mutate(pop_dev = abs(total_pop / get_target(This_state_map)-1),
         comp=comp_polsby(pl(), This_state_map,perim_df = this_county_perims2020),
         pct_min = group_frac(This_state_map, vap-vap_white,vap),
         pct_dem= group_frac(This_state_map,adv_18,adv_18+arv_18))

print(This_plans_today)
```

```
## A <redist_plans> containing 2,000 sampled plans and 1 reference plan
```

```
## Plans have 17 districts from a 10,084-unit map, and were drawn using Sequential
## Monte Carlo.
```

```
## With plans resampled from weights
## Plans matrix: int [1:10084, 1:2001] 16 17 17 17 17 17 17 17 16 17 ...
## # A tibble: 34,017 × 9
##    draw    district total_pop chain pop_overlap pop_dev    comp pct_min pct_dem
##    <fct>   <ord>        <dbl> <int>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 cd_2020 1           750264    NA           1 0.00453 0.0101    0.176   0.457
##  2 cd_2020 2           748749    NA           1 0.00654 0.00897   0.199   0.468
##  3 cd_2020 3              898    NA           1 0.999   0.641     0.112   0.462
##  4 cd_2020 4             1699    NA           1 0.998   0.499     0.311   0.510
##  5 cd_2020 5             1235    NA           1 0.998   0.259     0.138   0.459
##  6 cd_2020 6           749691    NA           1 0.00529 0.0114    0.441   0.623
##  7 cd_2020 7           749512    NA           1 0.00553 0.0198    0.357   0.580
##  8 cd_2020 8           750421    NA           1 0.00432 0.0234    0.331   0.567
##  9 cd_2020 9             1707    NA           1 0.998   0.629     0.141   0.460
## 10 cd_2020 10            2740    NA           1 0.996   0.495     0.397   0.643
## # ℹ 34,007 more rows
```

```
summary(This_plans_today)
```

```
## SMC: 2,000 sampled plans of 17 districts on 10,084 units
```

```
## `adapt_k_thresh`=0.99 • `seq_alpha`=0.5
```

```
## `est_label_mult`=1 • `pop_temper`=0
```

```
## Plan diversity 80% range: 0.97 to 1.10
```

```
##
## R-hat values for summary statistics:
## pop_overlap     pop_dev        comp     pct_min     pct_dem
##      ❌1.05       1.011       1.006     ❌1.076       1.009
```

```
## ✖ WARNING: SMC runs have not converged.
```

```
## Sampling diagnostics for SMC run 1 of 2 (1,000 samples)
```

```
##          Eff. samples (%) Acc. rate Log wgt. sd  Max. unique Est. k
## Split 1       969 (96.9%)     23.1%        0.34   626 ( 99%)     11
## Split 2       958 (95.8%)     31.0%        0.40   630 (100%)      7
## Split 3       943 (94.3%)     37.2%        0.47   629 (100%)      5
## Split 4       912 (91.2%)     42.0%        0.55   614 ( 97%)      4
## Split 5       891 (89.1%)     30.1%        0.59   604 ( 96%)      6
## Split 6       874 (87.4%)     36.4%        0.62   607 ( 96%)      4
## Split 7       882 (88.2%)     40.7%        0.65   623 ( 99%)      3
## Split 8       867 (86.7%)     32.9%        0.74   607 ( 96%)      4
## Split 9       850 (85.0%)     35.1%        0.81   607 ( 96%)      3
## Split 10      834 (83.4%)     38.3%        0.84   590 ( 93%)      2
## Split 11      796 (79.6%)     30.0%        0.91   587 ( 93%)      3
## Split 12      697 (69.7%)     34.4%        0.98   584 ( 92%)      2
## Split 13      776 (77.6%)     30.7%        1.05   585 ( 93%)      2
## Split 14      832 (83.2%)     27.5%        0.96   570 ( 90%)      2
## Split 15      781 (78.1%)     21.8%        0.88   574 ( 91%)      2
## Split 16      854 (85.4%)      7.8%        0.79   485 ( 77%)      2
## Resample      503 (50.3%)       NA%        0.72   690 (109%)     NA
```

```
## Sampling diagnostics for SMC run 2 of 2 (1,000 samples)
```

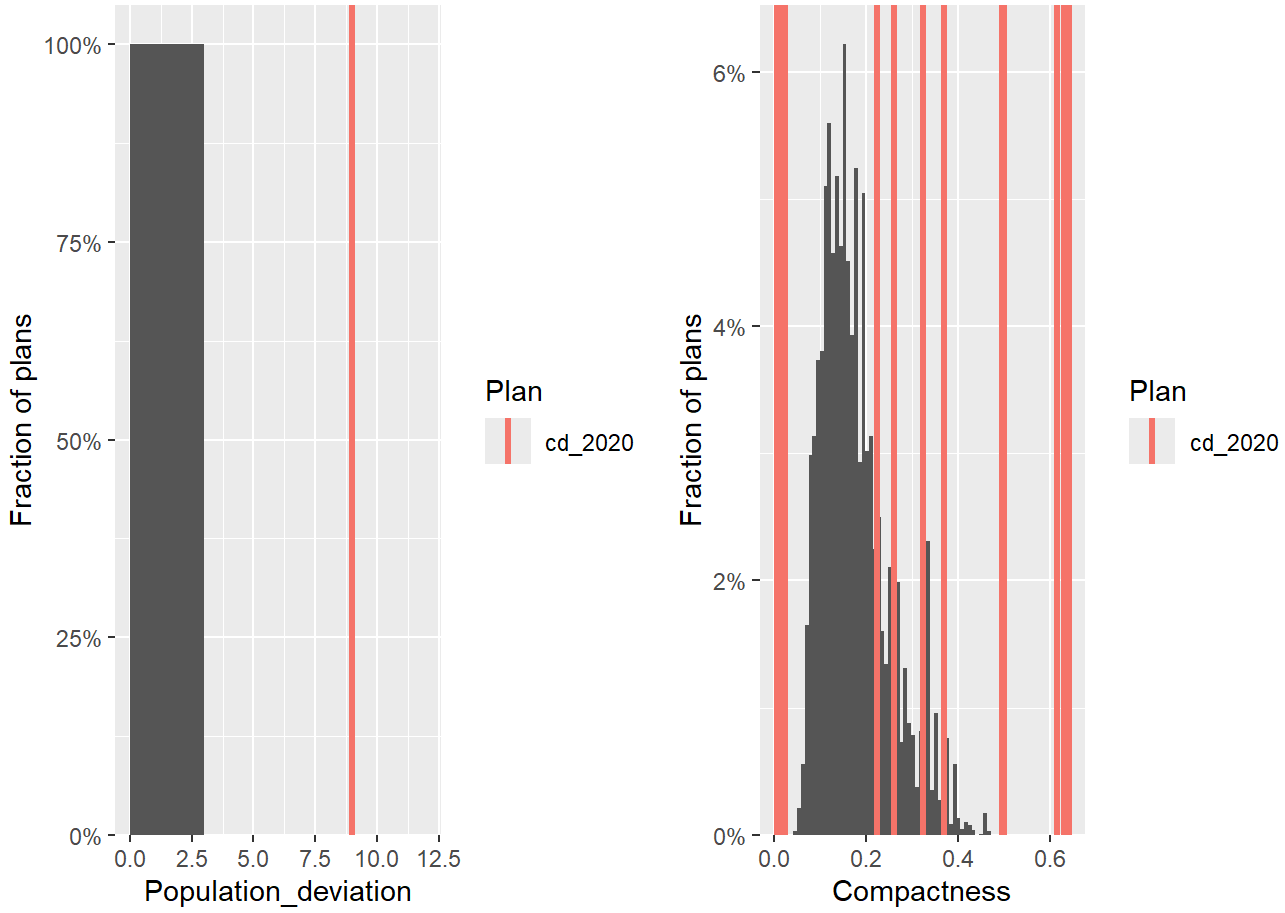
```
##          Eff. samples (%) Acc. rate Log wgt. sd  Max. unique Est. k
## Split 1       970 (97.0%)     16.2%        0.34   635 (100%)     15
## Split 2       957 (95.7%)     26.6%        0.40   622 ( 98%)      8
## Split 3       946 (94.6%)     26.6%        0.46   618 ( 98%)      8
## Split 4       865 (86.5%)     38.1%        0.62   612 ( 97%)      5
## Split 5       882 (88.2%)     47.0%        0.62   618 ( 98%)      3
## Split 6       838 (83.8%)     31.6%        0.64   606 ( 96%)      5
## Split 7       842 (84.2%)     29.0%        0.67   606 ( 96%)      5
## Split 8       792 (79.2%)     36.6%        0.71   609 ( 96%)      3
## Split 9       846 (84.6%)     36.2%        0.74   596 ( 94%)      3
## Split 10      866 (86.6%)     38.9%        0.77   611 ( 97%)      2
## Split 11      846 (84.6%)     36.7%        0.83   606 ( 96%)      2
## Split 12      831 (83.1%)     34.1%        0.87   586 ( 93%)      2
## Split 13      827 (82.7%)     25.8%        0.91   606 ( 96%)      3
## Split 14      856 (85.6%)     25.8%        0.88   584 ( 92%)      2
## Split 15      864 (86.4%)     13.2%        0.82   584 ( 92%)      4
## Split 16      852 (85.2%)      5.6%        0.77   511 ( 81%)      3
## Resample      405 (40.5%)       NA%        0.68   703 (111%)     NA
```

```
## • Watch out for low effective samples, very low acceptance rates (less than
## 1%), large std. devs. of the log weights (more than 3 or so), and low numbers
## of unique plans. R-hat values for summary statistics should be between 1 and
## 1.05.
```

```
## • SMC convergence: Increase the number of samples. If you are experiencing low
## plan diversity or bottlenecks as well, address those issues first.
```

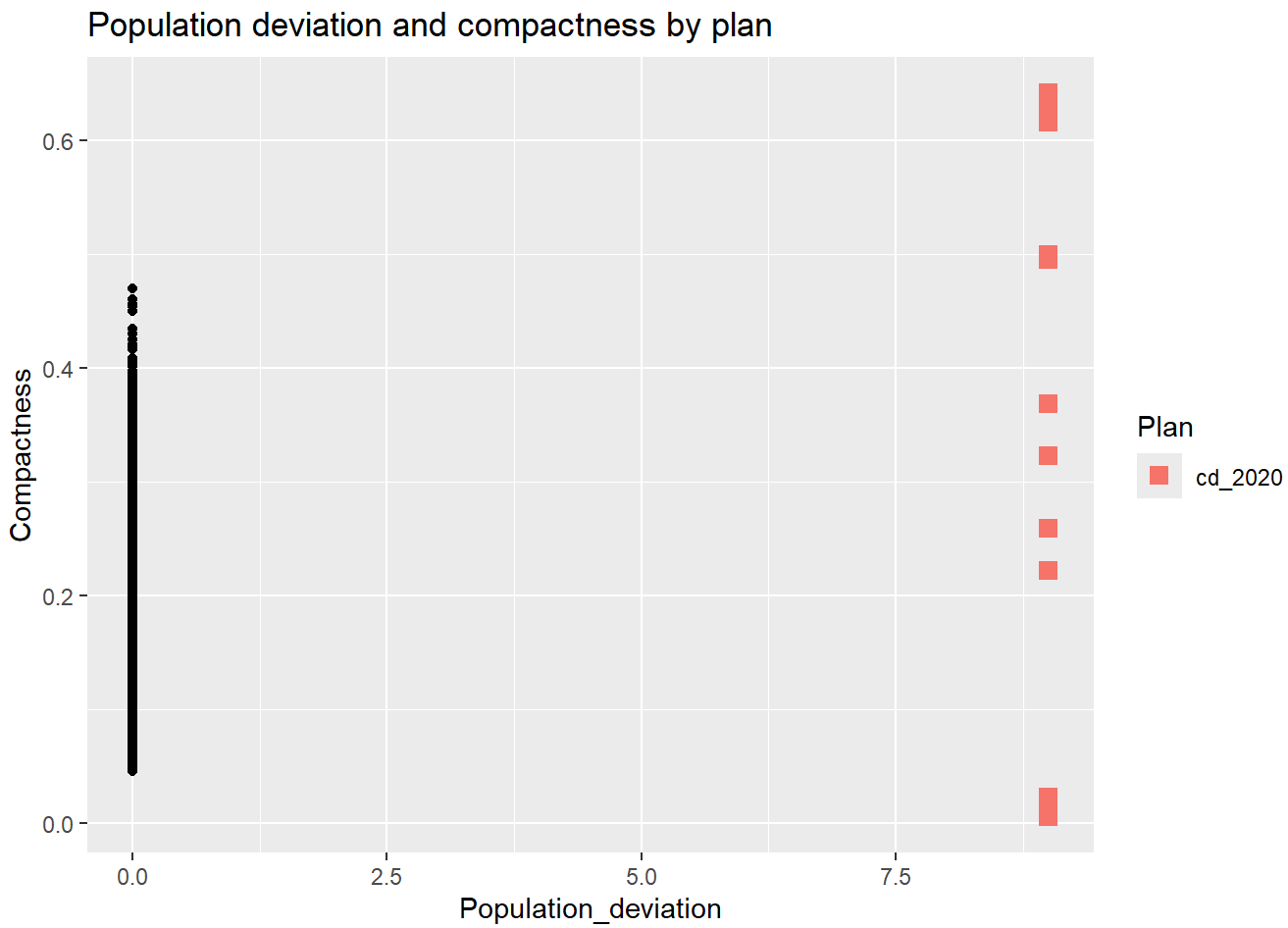
```
m_This_plans_today = This_plans_today %>%
  mutate(Compactness = comp_polsby(pl(),This_state_map),
         Population_deviation= plan_parity(This_state_map),
         Democratic_vote =group_frac(This_state_map,adv_18,(adv_18+arv_18)))

hist(m_This_plans_today,Population_deviation) + hist(m_This_plans_today,Compactness)#+
```

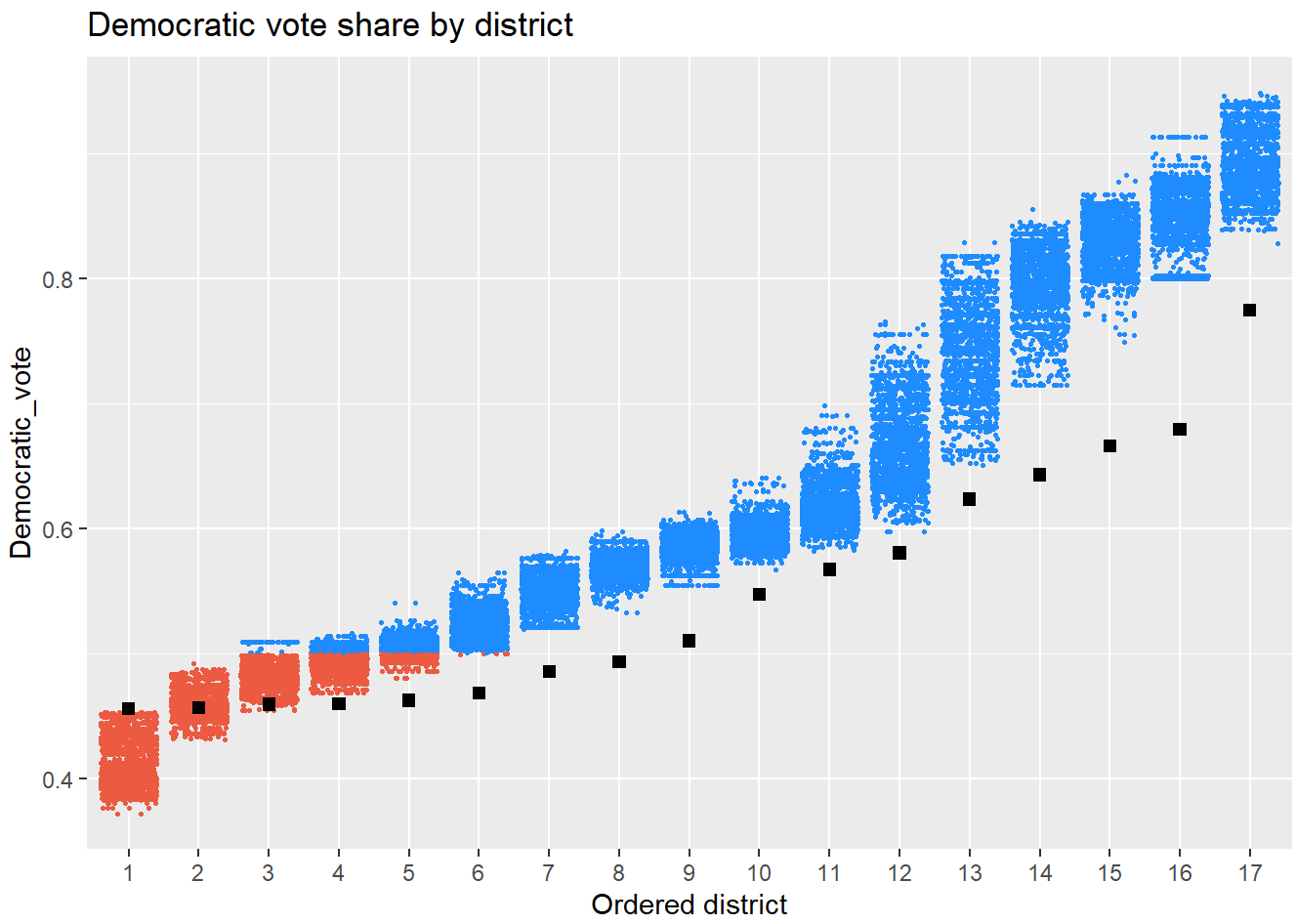



```
  #plot_layout(guides="collect")+
  #plot_annototion(title="Simulated plan charachteristics")
```

```
redist.plot.scatter(m_This_plans_today, Population_deviation,Compactness)+
  labs(title="Population deviation and compactness by plan")
```




```
plot(m_This_plans_today,Democratic_vote,size=0.5,color_thresh=0.5)+
  scale_color_manual(values=c("black","tomato2","dodgerblue"))+
  labs(title="Democratic vote share by district")
```


Democratic vote share by district
Democratic_vote
0.8
0.6
0.4
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Ordered district