

INLA-Mortality-7.2

Jeffrey Wu

2025-10-07

Steps of my analysis

Load all relevant packages:

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidyverse)

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## vforcats    1.0.0      vreadr     2.1.5
## vggplot2    3.5.2      vstringr   1.5.1
## vlubridate  1.9.4      vtibble    3.3.0
## vpurrr      1.1.0      vtidyr    1.3.1

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(lubridate)
library(stringr)
library(zoo)

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

library(ggplot2)
library(urbnmapr)
library(devtools)

## Loading required package: usethis
```

```

library(readxl)
library(spdep)

## Loading required package: spData
## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')`
## Loading required package: sf
## Linking to GEOS 3.13.1, GDAL 3.11.0, PROJ 9.6.0; sf_use_s2() is TRUE
library(sp)
library(huge)
library(INLA)

## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyverse':
##       expand, pack, unpack
##
## This is INLA_25.06.07 built 2025-06-11 18:54:45 UTC.
##   - See www.r-inla.org/contact-us for how to get help.
##   - List available models/likelihoods/etc with inla.list.models()
##   - Use inla.doc(<NAME>) to access documentation

library(HMMpa)
library(invgamma)
library(brinla)
library(reshape2)

##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyverse':
##       smiths

library(patchwork)
library(jsonlite)

##
## Attaching package: 'jsonlite'
##
## The following object is masked from 'package:purrr':
##       flatten

library(geosphere)
library(RAQSAPI)

## Use the function
## RAQSAPI::aqs_credentials(username, key)
## before using other RAQSAPI functions
## See ?RAQSAPI::aqs_credentials for more information

```

```

library(con2aqi)
library(pscl)

## Classes and Methods for R originally developed in the
## Political Science Computational Laboratory
## Department of Political Science
## Stanford University (2002-2015),
## by and under the direction of Simon Jackman.
## hurdle and zeroinfl functions by Achim Zeileis.

```

```

library(reshape2)
library(corrplot)

```

```

## corrplot 0.95 loaded

```

```

library(superheat)
library(shapes)
library(scales)

```

```

##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##      discard
##
## The following object is masked from 'package:readr':
##
##      col_factor
library(kableExtra)

```

```

##
## Attaching package: 'kableExtra'
##
## The following object is masked from 'package:dplyr':
##
##      group_rows

```

Loading and (quickly cleaning) all necessary datasets:

```

###SoA data
soa.data = read_xlsx("SoA.data.1019.xlsx")

###county_fips are unique identifier for counties
soa.data$county_fips = as.character(soa.data$county_fips) ##change it to character

```

#IMPORTANT

*# This shape file contains the coordinates for county boundaries
##counties is from urbanmap*

```

CA.counties = urbnmpr::counties %>% filter(state_abbv == "CA")

```

*###IF WE WANT TO BOIL DOWN TIME SERIES AND KEEP ALL DATA, SWITCH to CA_newdata below
soa_joint <- left_join(CA.counties, soa.data, by = "county_fips")*

Warning in left_join(CA.counties, soa.data, by = "county_fips"): Detected an unexpected many-to-many

```

## i Row 1 of `x` matches multiple rows in `y`.
## i Row 1 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.

#Use with soa.data (full data)
CA_data = soa_joint %>% select(long, lat, county_name.y, Year, Score, Total_Pop,
                                EDUC_Lessthan9, EDUC_college, White_Collar,
                                Unemployment_Rate, Adj_HH_income, Income_Disparity,
                                Individuals_Below_Poverty, Median_Home_Value,
                                Median_Gross_Rent, Housing_No_Telephone,
                                Housing_Incomplete_Plumbing)

colnames(CA_data)[3] = "County"

CA_newdata = soa.data[1:58,]
CA_newdata = CA_newdata[, -c(4, 7, 8)]


###Cal-ViDa data
mortality = read.csv("respmortality1419.csv") #data from CalViDa-respmortality1423 for 2014-2019 bc we
mortality = filter(mortality, Cause_of_Death %in% c("Chronic lower respiratory diseases", "Influenza and
mortality = mortality[, -c(1, 4, 9)]
Population = rep(100000, nrow(mortality))
mortality = cbind(mortality, Population)

```

Quick descriptions of SoA data variables:

Score: social deprivation index (SDI) score calculated from the following 11 subindices
EDUC_Lessthan9: % of population older than 24 with less than 9 years of education
EDUC_college: % of population older than 24 with at least four years of college education
White_Collar: % of population older than 15 employed in a white collar occupation
Unemployment_Rate: unemployment rate for population older than 15
Adj_HH_income: median household income adjusted for local housing costs
Income_Disparity: an income disparity ratio
Individuals_Below_Poverty: % of population below the federal poverty line
Median_Home_Value: median home value for owned, occupied units
Median_Gross_Rent: median gross rent for rented units
Housing_No_Telephone: % of households without a telephone
Housing_Incomplete_Plumbing: % of households with incomplete plumbing

Heatmap of population by county

```

#Initializing map and station locations
ca_map <- map_data("county", region = "california")

#Match population dataset with ca_map
#2010-2019 population data for CA
USpops = read.csv("CA_census_pops1019.csv")
CApops = USpops %>% filter(STNAME == "California") %>% select(CTYNAME, POPESTIMATE2019)
CApops = CApops[-1,]

CApops$CTYNAME = unique(ca_map$subregion)
colnames(CApops) = c("subregion", "pop")

merged_data <- merge(ca_map, CApops, by = "subregion", all.x = TRUE)

```

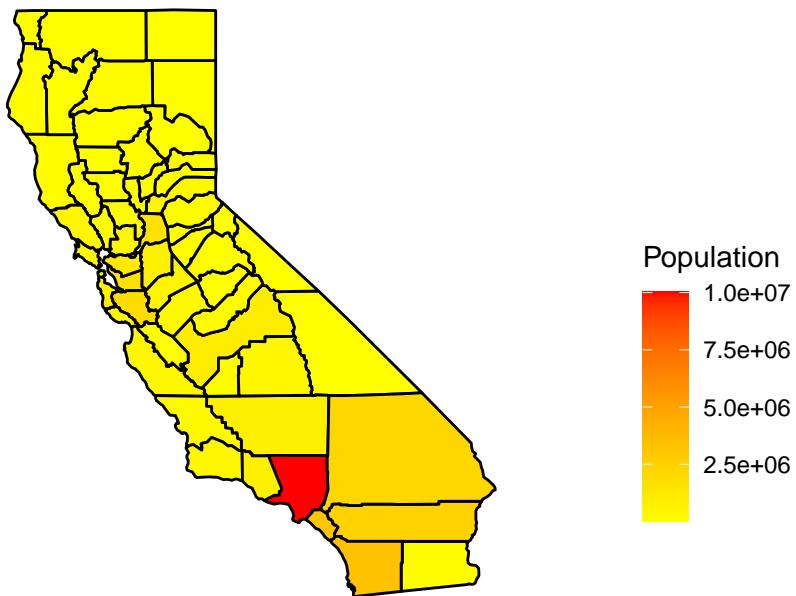
```

#Plot
gg_pop <- ggplot() +
  geom_polygon(data = merged_data, aes(x = long, y = lat, group = group, fill = pop),
               color = "black") +
  coord_fixed(ratio = 1.3, xlim = c(-125, -112), ylim = c(30, 42)) +
  theme_void() +
  labs(title = "Heatmap of County Populations for 2019", fill = expression("Population")) +
  scale_fill_gradient(low = "yellow", high = "red")

print(gg_pop)

```

Heatmap of County Populations for 2019



California state with county labels for reference:



SKATER clustering

The code below structures the dataframe to be fed into the spatial data frame (SPDF) object. Dimensions are 58 rows by 10 columns (each column is its own year)

```
###Setting up SPDF for CA counties
CA_sf = st_read(getwd(),"CA_Counties_TIGER2016")

## Reading layer `CA_Counties_TIGER2016` from data source
##   `C:\Users\jeffr\Desktop\Spatiotemporal + Causal Inference\Wildfire Paper 1 Code'
##   using driver `ESRI Shapefile'
## Simple feature collection with 58 features and 17 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -13857270 ymin: 3832931 xmax: -12705030 ymax: 5162404
## Projected CRS: WGS 84 / Pseudo-Mercator

CA_spdf = as_Spatial(CA_sf)

# score_scaled = scale(CA_data$Score)

c_index = unique(CA_data$County)
y_index = unique(CA_data$Year)
SDI_df = matrix(nrow=10,ncol=58)
```

```

track1 = 1
track2 = 1

for (i in c_index){
  for (j in y_index){
    scores = CA_data %>% filter(County == i) %>% filter(Year == j) %>% select(Score) %>% unique()
    SDI_df[track1,track2] = scores$Score
    track1 = track1 + 1
  }
  track1 = 1
  track2 = track2 + 1
}

score_scaled = scale(SDI_df) #NEED TO SCALE THE DATA BEFORE FEEDING IT INTO SKATER

#covariates_scale = data.frame(apply(CA_data[,4:16],2,scale))
covariates_scale = data.frame(t(score_scaled))

CA_spdf@data = covariates_scale

```

Using the SPDF from above, we follow the steps of SKATER tutorial (<https://www.dshkol.com/post/spatially-constrained-clustering-and-regionalization/>) to generate three separate clustering results: (1) Unconstrained/default (2) Clusters have minimum population constraint based on the total population / # of clusters (3) Clusters are comprised of a minimum number of counties (8 for smaller number of clusters, 4 for bigger numbers)

```

#Identify neighborhood list for counties
CA_nb = poly2nb(CA_spdf)

#summary(CA_nb)

# plot(CA_spdf, main = "With queen")
# plot(CA_nb, coords = coordinates(CA_spdf), col="blue", add = TRUE)

#Calculate edge costs (dissimilarity matrix) based on Euclidean distance
costs <- nbcosts(CA_nb, data = covariates_scale)

###Get adjacency matrix using nb2mat() (SEPARATE STEP FOR INLA)
adj = nb2mat(CA_nb,style = "B")

#Style means the coding scheme style used to create the weighting matrix
# B: basic binary coding scheme
# W: row standardized coding scheme
# C: globally standardized coding scheme
# U: values of C / number of neighbors
# S: variance stabilizing coding scheme

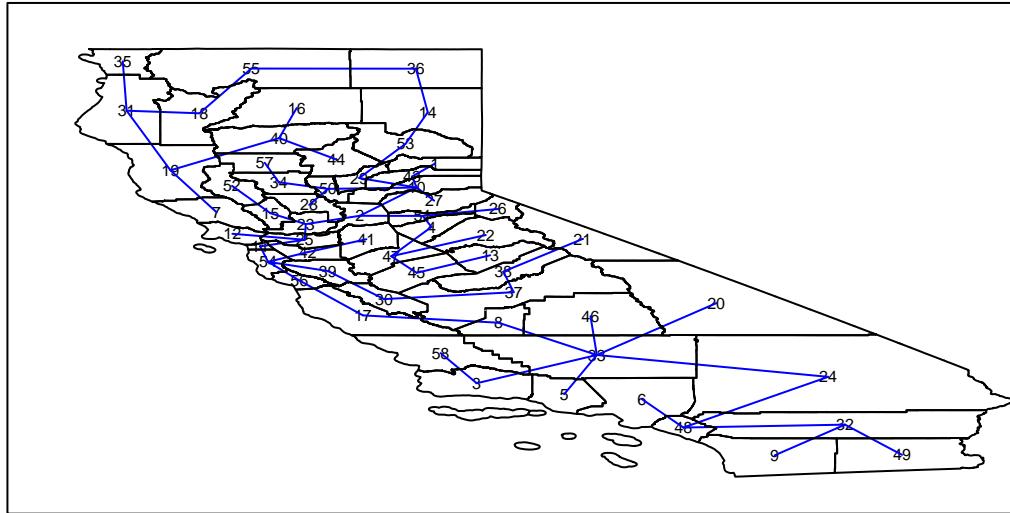
#Transform edge costs to spatial weights
ct_w <- nb2listw(CA_nb,costs,style="B")

#Create minimum spanning tree
ct_mst <- mstree(ct_w)

plot(ct_mst,coordinates(CA_spdf),col="blue", cex.lab=0.5)

```

```
plot(CA_spdf, add=TRUE)
```



```

#Run SKATER algorithm to get 7 contiguous clusters (cluster idx is in order of CA_sf)
clus7 <- skater(edges = ct_mst[,1:2], data = covariates_scale, ncuts = 6)

#Determine an appropriate minimum population threshold based on???
pops_summary = summary(unique(CA_data$Total_Pop))
pops_summary

##      Min.    1st Qu.     Median      Mean    3rd Qu.       Max.
## 12700     63275    219705   710796   750235 10105722

#Idea 1: Use median * (how many counties should be in a cluster at minimum)
min_pop = as.numeric(pops_summary[3] * 4)

#Idea 2: If we assume CA population is 39M, divide total pop by # clusters
min_pop2 = 39000000 / 7

#Add a min population constraint
clus7_min <- skater(edges = ct_mst[,1:2],
                      data = covariates_scale,
                      crit = min_pop2,
                      vec.crit = CA_data$Total_Pop,
                      ncuts = 6)

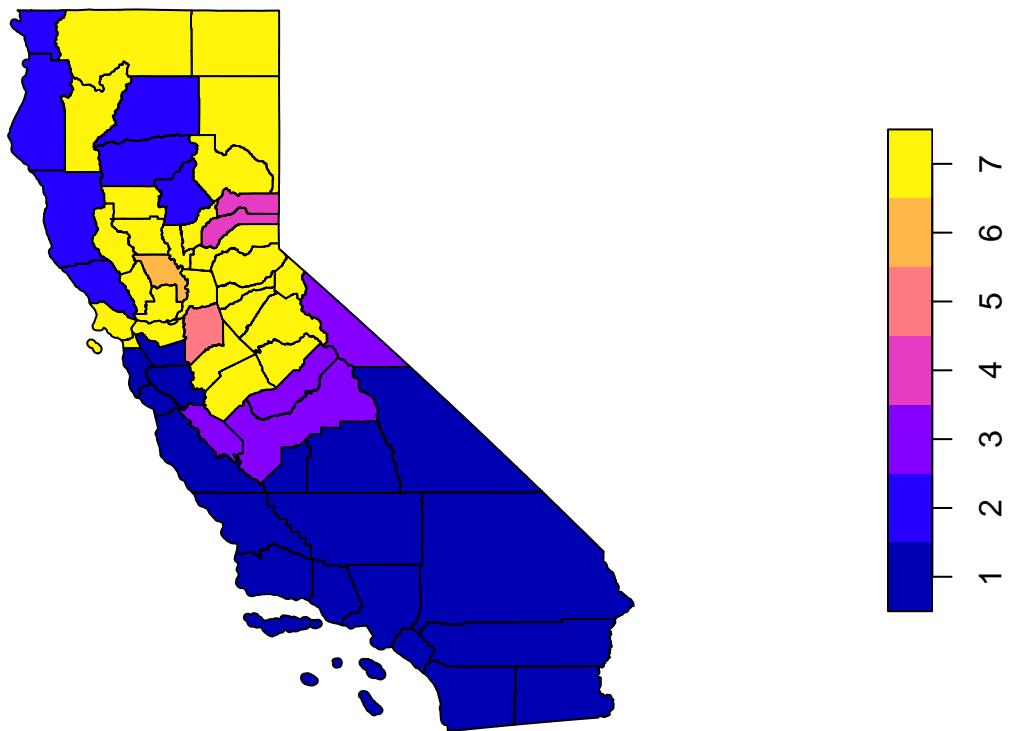
#Add a minimum number of areas in each cluster constraint
clus7_minarea = skater(edges = ct_mst[,1:2], data = covariates_scale, ncuts = 6, 4)

```

```
CA_data_cluster = (CA_sf %>% mutate(clus = clus7_min$groups))

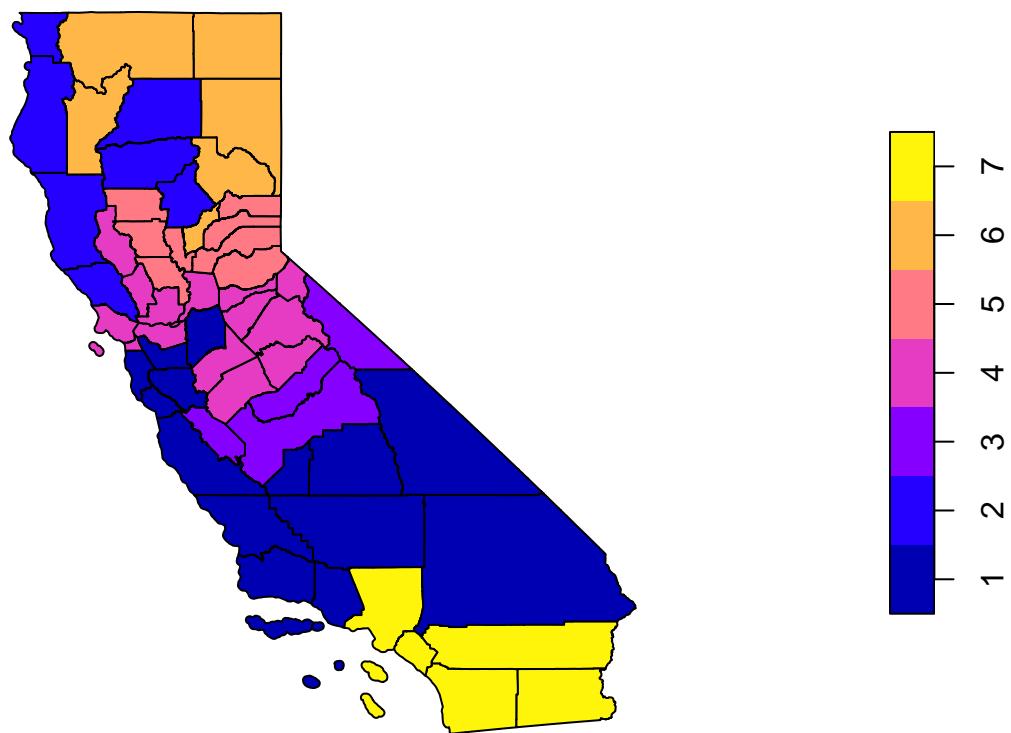
#Plot clustered CA
plot((CA_sf %>% mutate(clus = clus7$groups))['clus'], main = "7 cluster example")
```

7 cluster example



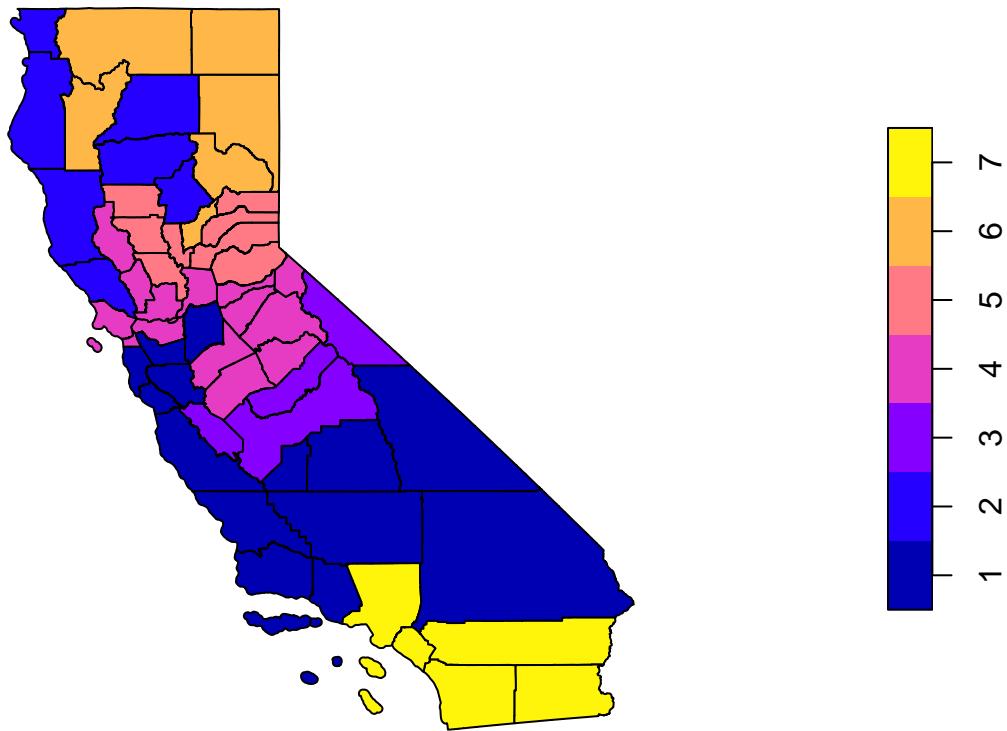
```
plot((CA_sf %>% mutate(clus = clus7_min$groups))['clus'], main = "7 cluster example with population con...")
```

7 cluster example with population constraint



```
plot((CA_sf %>% mutate(clus = clus7_minarea$groups))['clus'], main = "7 cluster example with minimum nu")
```

7 cluster example with minimum number of areas constraint



```
#plot(CA_sf,col=c("red","green","blue","purple","yellow")[clus7_min$groups],max.plot=17)
```

For reference, here are the cluster labels for each county:

```
clusterlabels = data.frame(CA_data_cluster$NAME,clus7_min$groups)
names(clusterlabels) = c("counties","Cluster")

o = order(clusterlabels$counties)
clusterlabels = clusterlabels[o,]
rownames(clusterlabels) = NULL

clusterlabels

##          counties Cluster
## 1          Alameda      1
## 2          Alpine       4
## 3          Amador       4
## 4          Butte        2
## 5          Calaveras     4
## 6          Colusa       5
## 7 Contra Costa       4
## 8          Del Norte     2
## 9          El Dorado     5
## 10         Fresno        3
## 11         Glenn        5
## 12 Humboldt        2
## 13 Imperial        7
```

```

## 14          Inyo      1
## 15          Kern      1
## 16          Kings     1
## 17          Lake      4
## 18          Lassen    6
## 19          Los Angeles 7
## 20          Madera    3
## 21          Marin     4
## 22          Mariposa   4
## 23          Mendocino  2
## 24          Merced    4
## 25          Modoc     6
## 26          Mono      3
## 27          Monterey   1
## 28          Napa      4
## 29          Nevada    5
## 30          Orange    7
## 31          Placer    5
## 32          Plumas    6
## 33          Riverside  7
## 34          Sacramento 4
## 35          San Benito 3
## 36          San Bernardino 1
## 37          San Diego   7
## 38          San Francisco 4
## 39          San Joaquin  1
## 40          San Luis Obispo 1
## 41          San Mateo   1
## 42          Santa Barbara 1
## 43          Santa Clara  1
## 44          Santa Cruz   1
## 45          Shasta     2
## 46          Sierra     5
## 47          Siskiyou   6
## 48          Solano     4
## 49          Sonoma     2
## 50          Stanislaus 4
## 51          Sutter     5
## 52          Tehama     2
## 53          Trinity    6
## 54          Tulare     1
## 55          Tuolumne   4
## 56          Ventura    1
## 57          Yolo       5
## 58          Yuba       6

counties = clusterlabels$counties
num_clus = max(clus7_min$groups)

```

Compare with KNN clustering:

```

knn_dataset = cbind(covariates_scale,clusterlabels$Cluster)
# knn7 = knn(train = knn_dataset[,1:5],test=knn_dataset[,6:10],cl=knn_dataset$`clusterlabels$Cluster`,k=7)

knn7 = knn.cv(train = knn_dataset[,1:10],cl=knn_dataset$`clusterlabels$Cluster`,k=7)

```

```

knn_clusterlabels = cbind(clusterlabels,knn7)

for (i in 1:num_clus){
  print(knn_clusterlabels %>% filter(Cluster==i))
}

plot((CA_sf %>% mutate(clus = knn_clusterlabels$knn7))['clus'], main = "7 cluster example with population")

```

HUGE graph estimation

This code chunk takes the cluster grouping from SKATER and aggregates the full dataframe from the SPDF (58x10) into a 10x7 matrix (10 time points x 7 clusters) to be fed into the graph estimation package HUGE. The data from each county in a given cluster is aggregated based on a population weighted mean.

HUGE uses glasso to estimate a graph structure based on the aggregated feature data which recall, is the SDI score (socioeconomic status) from the SoA. We use a grid of lambda values under 1 in order to ensure that some edges will be present in the estimates produced by HUGE. This decision is supported by the fact that partial correlations calculated via regression appear to be statistically significant. Based on simulation results, we believe that EBIC is a suitable criterion for choosing the best estimated graph in the huge.select() step.

The objective function for EBIC is as follows: $EBIC_\gamma(s) = -2\log L_n\{\hat{\theta}(s)\} + \nu(s)\log n + 2\gamma\log\tau(S_j)$ where $0 \leq \gamma \leq 1$ and S_j is model space of size $\tau(S_j)$

```

#Aggregate feature vectors into one vector for each SKATER cluster
CA_cluster = data.frame(CA_sf$NAMELSAD,clus7_min$groups)
names(CA_cluster) = c("County","Cluster")
year = 2010:2019

CA_cluster = left_join(CA_cluster,CA_data,by = "County")

#Get weighted avg value for Score for each cluster for each year
#Create new data matrix of aggregated feature vectors
cluster_features = matrix(NA,nrow = 10,ncol = 7)
pops = matrix(NA,nrow = 10,ncol = 7)

for (i in 1:num_clus){
  cluster = CA_cluster %>% filter(Cluster == i)

  for(j in 1:10){
    #Obtain a weighted mean based on population
    vec = cluster %>% filter(Year == year[j]) %>% select(Score,Total_Pop) %>% unique()
    cluster.pop = sum(vec$Total_Pop)
    pops[j,i] = cluster.pop
    cluster.popweights = vec$Total_Pop/cluster.pop
    cluster_features[j,i] = weighted.mean(vec$Score,cluster.popweights)
  }
}

cluster_pops = apply(pops,2,mean)

#Graph learning w HUGE
out.glasso = huge(cluster_features,lambda = seq(0.95,0.05,by=-0.05),method="glasso")

## Conducting the graphical lasso (glasso) wtih lossless screening....in progress: 5%Conducting the graph
## Conducting the graphical lasso (glasso)...done.

```

```

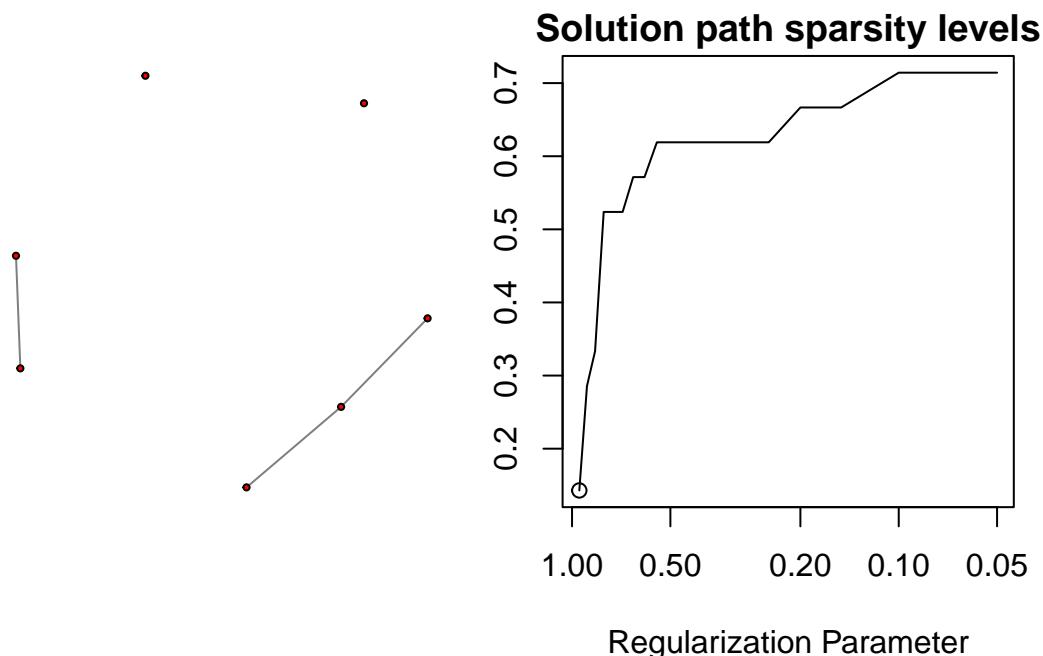
glasso.stars = huge.select(out.glasso,criterion = "stars",stars.thresh = 0.1)

## Conducting Subsampling....in progress:5% Conducting Subsampling....in progress:10% Conducting Subsampl
glasso.ric = huge.select(out.glasso,criterion = "ric")

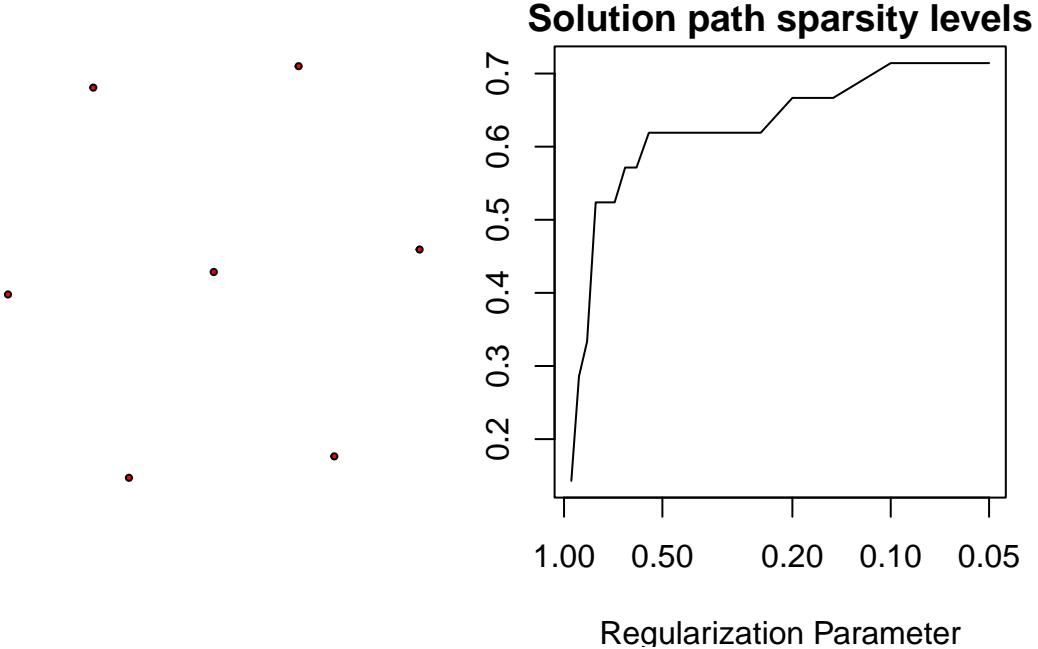
## Conducting rotation information criterion (ric) selection....done
## Computing the optimal graph....done
glasso.ebic = huge.select(out.glasso,criterion = "ebic")

## Conducting extended Bayesian information criterion (ebic) selection....done
plot(glasso.stars)

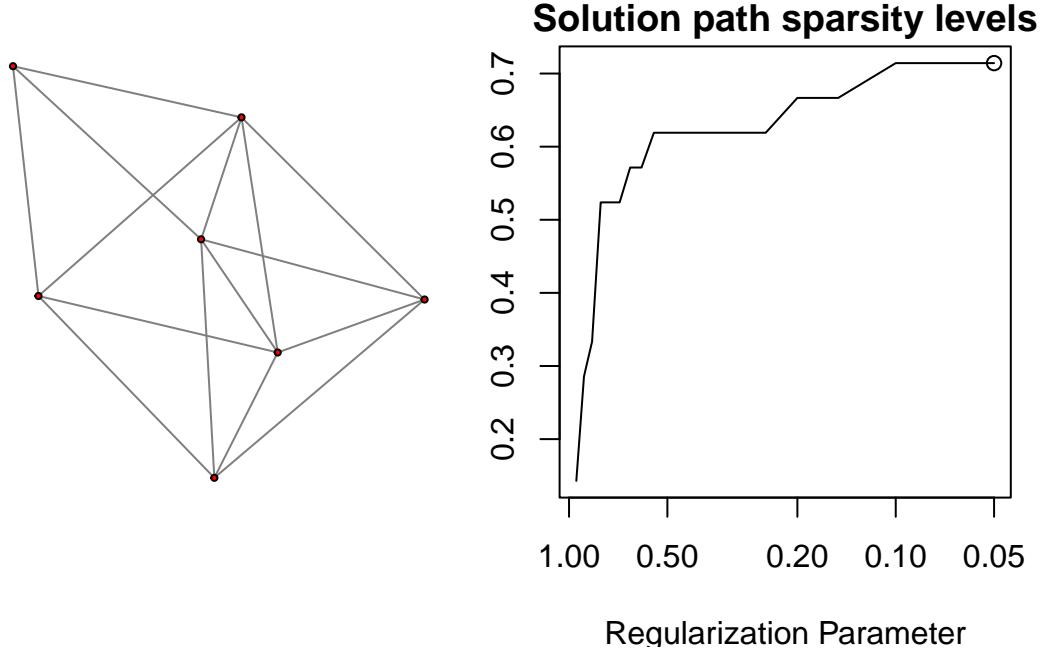
```



```
plot(glasso.ric)
```



```
plot(glasso.ebic)
```



```

huge.est = glasso.ebic$refit
huge.est

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]     0     0     1     1     1     0     1
## [2,]     0     0     1     1     1     1     1
## [3,]     1     1     0     0     1     1     0
## [4,]     1     1     0     0     0     0     1
## [5,]     1     1     1     0     0     1     1
## [6,]     0     1     1     0     1     0     1
## [7,]     1     1     0     1     1     1     0

#Identify which clusters/nodes are the most connected on the graph i.e. has the most association with t
degree_connectivity = data.frame(colSums(huge.est))
colnames(degree_connectivity) = "node_connections"
degree_connectivity = cbind(c(1:num_clus),degree_connectivity)

```

Plotting HUGE graph

```

install.packages("igraph")
library(igraph)
# Convert the adjacency matrix to a graph object
g <- graph_from_adjacency_matrix(huge.est, mode = "undirected")

# Assign custom labels to vertices
V(g)$name <- c(1,2,3,4,5,6,7)

```

```

# Assign colors to vertices
V(g)$color <- c("red", "cyan", "green", "yellow", "purple", "orange", "pink")

# Plot the graph with labeled vertices
plot(g, vertex.label = V(g)$name, vertex.color = V(g)$color, vertex.size = 20)

```

Transforming estimated adjacency matrix to graph filter H

The code below takes the adjacency matrix estimated in the previous step and transforms it into a graph filter H. The steps are explained in Antonian et al 2019 (Gareth Peters' paper). The cutoff transformation is applied to the eigenvalues of the graph Laplacian.

```

A = as.matrix(huge.est)
p = nrow(A)

#obtain graph Laplacian L
D = diag(p)
for (i in 1:p){
  d = sum(A[,i])
  D[i,i] = d
}

L = D - A

#eigendecomposition of L
Ldecomp = eigen(L)
U = as.matrix(Ldecomp$vectors)
Lambdas = Ldecomp$values

#test
#U %*% (diag(p)*Lambdas) %*% t(U)

#Function implementing cutoff transform for eigenvalues
cutoff.transform = function(lambdas,q){
  transformed = c()
  cutoff = quantile(lambdas,q)
  for (i in lambdas){
    if(i <= cutoff){
      transformed = c(transformed,1)
    }
    else{
      transformed = c(transformed,0)
    }
  }

  return(transformed)
}

#quantile(Lambdas,2/3)
transformed.L = cutoff.transform(Lambdas,2/3)
eta.L = diag(p)*transformed.L

#obtain graph filter

```

```

H = U %*% eta.L %*% t(U)
H

##          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,]  0.80121219 -0.28070511  0.139925150  0.086945750  0.15087817 -0.061815959
## [2,] -0.28070511  0.58851806  0.163559807  0.150878170  0.29315326 -0.068010981
## [3,]  0.13992515  0.16355981  0.824846850  0.002116461  0.07426411  0.086945750
## [4,]  0.08694575  0.15087817  0.002116461  0.909676139 -0.21504407 -0.008836559
## [5,]  0.15087817  0.29315326  0.074264113 -0.215044073  0.46065322 -0.055329344
## [6,] -0.06181596 -0.06801098  0.086945750 -0.008836559 -0.05532934  0.956168924
## [7,]  0.16355981  0.15260679 -0.291658130  0.074264113  0.29142465  0.150878170
##          [,7]
## [1,]  0.16355981
## [2,]  0.15260679
## [3,] -0.29165813
## [4,]  0.07426411
## [5,]  0.29142465
## [6,]  0.15087817
## [7,]  0.45892461
gfilter_weight = norm((1/7)*H^2,type = "F")
# gfilter_weight = norm(H^2,type = "F")

```

Create a function to generate heatmap plots of matrices

```

matrix_heatmap= function(matrix,title = "",gradient_zones = c(0,0.5,0.999)){
  r = nrow(matrix)
  df = as_tibble(cbind(expand_grid(rev(seq_len(r)),seq_len(r)),c(matrix))) %>% setNames(c("row","col",""))
  df$value[df$value == 1] = 0.999

  x_min = min(df$row) - 0.5
  x_max = max(df$row) + 0.5
  y_min = min(df$col) - 0.5
  y_max = max(df$col) + 0.5

  plot = ggplot(df,mapping = aes(x=row,y=col,fill=value)) + geom_tile() +
    geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max),
              fill = NA, color = "black", inherit.aes = FALSE) +
    scale_fill_gradientn(colors = c("white","lightblue","darkblue"),
                          values = rescale(gradient_zones),
                          limits = c(0, 0.99),
                          oob = squish) + ggtitle(title) + theme_void() +
    theme(plot.margin = margin(t = 10, r = 30, b = 10, l = 10),legend.key.size = unit(1,"cm"))

  return(plot)
}

matrix_heatmap2 = function(
  matrix,
  title = "",
  legend_title = "Covariance",
  xlab = "Entry j",
  ylab = "Entry i",

```

```

gradient_zones = c(0, 0.5, 0.999),
save_path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/Paper1-im
prefix = "",
filename = "heatmap.png", # base filename
text_scale = 2){
r = nrow(matrix)
c = ncol(matrix)

df = as_tibble(cbind(expand_grid(rev(seq_len(r)), seq_len(c)), c(matrix))) %>%
  setNames(c("row", "col", "value"))

df$value[df$value == 1] = 0.999

x_min = min(df$row) - 0.5
x_max = max(df$row) + 0.5
y_min = min(df$col) - 0.5
y_max = max(df$col) + 0.5

plot = ggplot(df, mapping = aes(x = row, y = col, fill = value)) +
  geom_tile() +
  geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max),
            fill = NA, color = "black", inherit.aes = FALSE) +
  scale_fill_gradientn(
    colors = c("white", "lightblue", "darkblue"),
    values = rescale(gradient_zones),
    limits = c(0, 0.99),
    oob = squish,
    name = legend_title
  ) +
  scale_x_continuous(
    breaks = c(1,c),
    labels = paste(c(1,c)), # reversed because we used rev() earlier
    name = xlab
  ) +
  scale_y_continuous(
    breaks = c(r,1),
    labels = paste(c(1,r)),
    name = ylab
  ) +
  ggtitle(title) +
  theme_minimal(base_size = 11 * text_scale) +
  theme(
    plot.margin = margin(t = 10, r = 30, b = 10, l = 10),
    legend.key.size = unit(1, "cm"),
    axis.text.x = element_text(hjust = 0.5), # center x labels
    panel.grid = element_blank() # remove grid lines
  )

ggsave(
  filename = paste0(prefix, filename),
  plot = plot,
  path = save_path,
  width = 8,

```

```

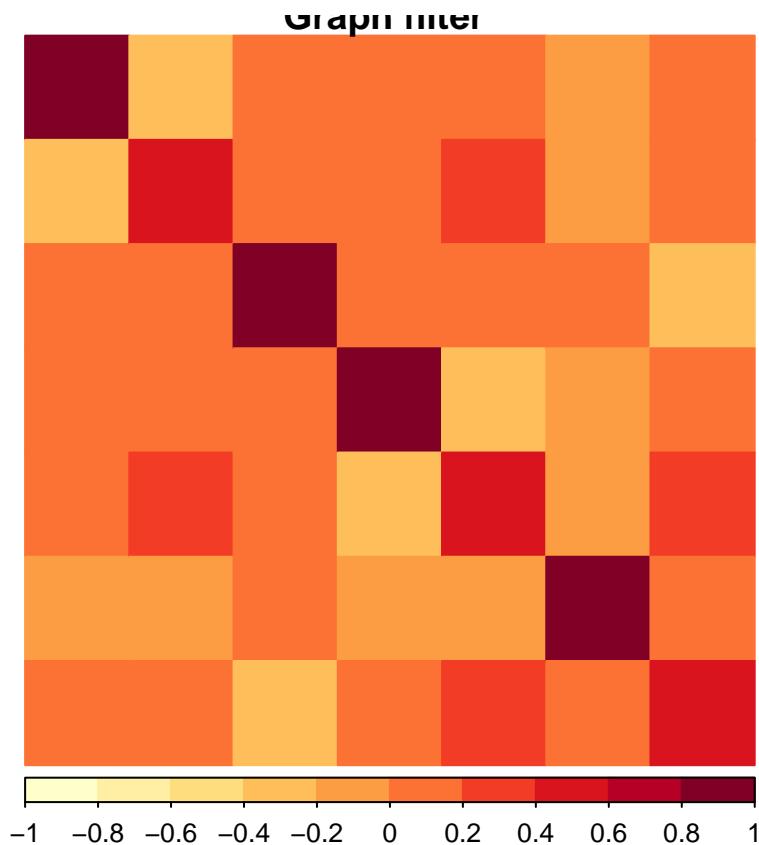
        height = 6,
        dpi = 300
    )

    return(plot)
}

```

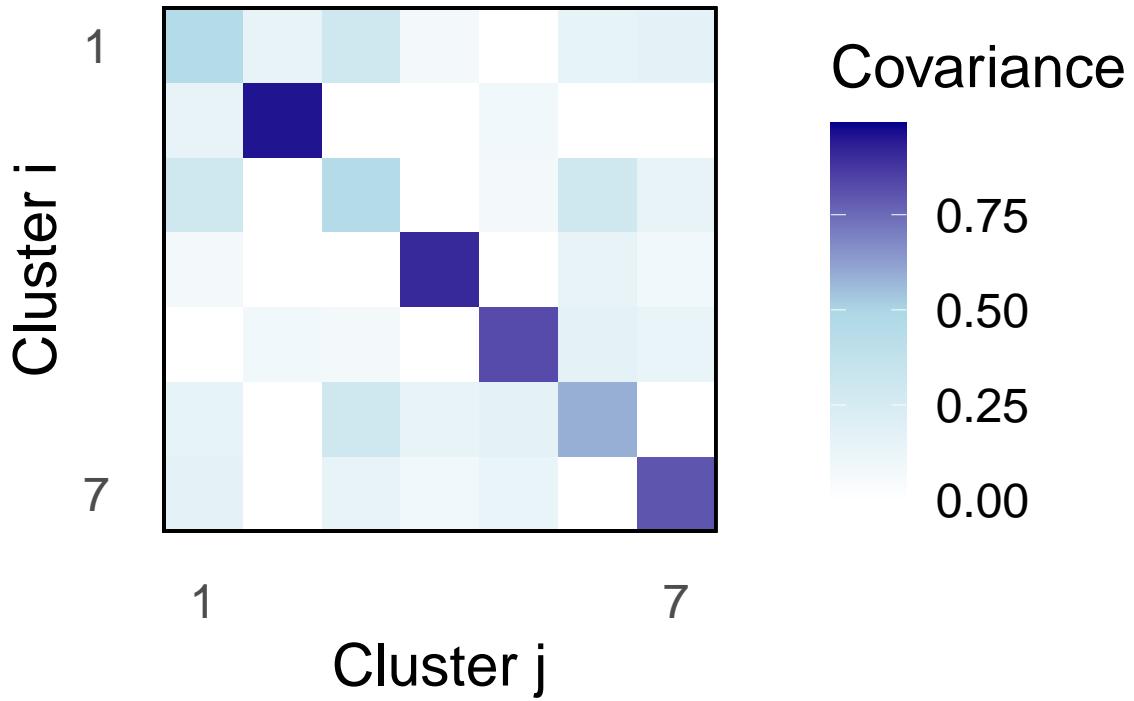
Heatmap plot of graph filter

```
#Heatmap of resulting H
corrplot(H, order = 'original', cl.pos = 'b', tl.pos = 'n', method = "color", col = COL1('YlOrRd',10),
         title = "Graph filter")
```



```
matrix_heatmap2(H,title = "",legend_title = "Covariance",prefix = "gfilter-7.2-",xlab = "Cluster j",ylab = "Cluster i")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have length 1, but the data has 49 rows.
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
## All aesthetics have length 1, but the data has 49 rows.
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



Downloading EPA data

```

library(tidyverse)
library(plyr)

## -----
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## -----
## 
## Attaching package: 'plyr'
## The following object is masked from 'package:purrr':
## 
##     compact
## The following objects are masked from 'package:dplyr':
## 
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
library(dplyr)
library(jsonlite)
library(lubridate)

```

```

library(ggplot2)
library(maps)

## 
## Attaching package: 'maps'
## The following object is masked from 'package:plyr':
## 
##     ozone
## The following object is masked from 'package:purrr':
## 
##     map
library(mapdata)
library(geosphere)
library(urbnmapr)
library(RAQSAPI)
library(con2aqi)

aqs_credentials("jeffreywu@ucsb.edu", "copperheron86")

```

Get county and pollutant reference codes from EPA

```

#Get county codes
counties_url = "https://aqs.epa.gov/data/api/list/countiesByState?email=jeffreywu@ucsb.edu&key=copperheron86"

countycodes = fromJSON(counties_url)
countycodes = countycodes[[2]]
california_counties = countycodes$code

#Get parameter codes
parameters_url = "https://aqs.epa.gov/data/api/list/parametersByClass?email=jeffreywu@ucsb.edu&key=copperheron86"

parametercodes = fromJSON(parameters_url)
parametercodes = parametercodes[[2]]
parametercodes = parametercodes[-7,]

pollutants = data.frame(parametercodes$code)
labels = c("lead", "co", "so2", "no2", "o3", "pm10", "pm25")
pollutants = cbind(pollutants, labels)

```

Identifying a set of monitoring stations that are well distributed across CA

The goal here is to query all of the EPA measurements for all 7 pollutants and AQI from 2014-2019. The first step is to query all stations in California that collect measurements for at least one of the 7 pollutants.

FUNCTION THAT QUERIES STATION LOCATIONS FOR A GIVEN POLLUTANT

```

query_aqs_station_data <- function(param,year){
  start_date <- paste0(year, "0101")
  end_date <- paste0(year, "1231")

```

```

url <- paste0("https://aqs.epa.gov/data/api/monitors/byState?email=jeffreywu@ucsb.edu&key=copperheron8")

myData <- fromJSON(url)
station_data = myData[[2]]

return(station_data)
}

test = query_aqs_station_data(pollutants$parametercodes.code[7], 2014)

FOR EACH POLLUTANT, GRAB ALL MONITORING STATIONS FOR EACH YEAR

stations_url = "https://aqs.epa.gov/data/api/monitors/byState?email=jeffreywu@ucsb.edu&key=copperheron8"

stations = fromJSON(stations_url)
stations = stations[[2]]

station_data2014_pm2.5 = stations %>% select(latitude, longitude, site_number, local_site_name, county_code)

#Get monitoring station locations for each pollutant for each year (takes approx 3 min)
all_pollutants_station_data <- list()
for (year in 2014:2019){
  year_data <- lapply(pollutants$parametercodes.code,
                      query_aqs_station_data, year = year)
  all_pollutants_station_data[[as.character(year)]] <- year_data
}

# 1 - Lead, 2 - Carbon monoxide (CO), 3 - Sulfure dioxide (SO2), 4 - Nitrogen dioxide (NO2)
# 5 - Ozone (O3), 6 - Total PM10, 7 - PM2.5

all_pollutants_station_2014data = all_pollutants_station_data[[1]]
all_pollutants_station_2015data = all_pollutants_station_data[[2]]
all_pollutants_station_2016data = all_pollutants_station_data[[3]]
all_pollutants_station_2017data = all_pollutants_station_data[[4]]
all_pollutants_station_2018data = all_pollutants_station_data[[5]]
all_pollutants_station_2019data = all_pollutants_station_data[[6]]

```

Identify subset of stations that has best spatial coverage wrt CA state

Starting off with a list of all the stations measuring each parameter/pollutant, we want to identify a subset of stations that measure each pollutant for each county. This overall set of stations (subset for each county combined together) should have a good spatial coverage of the state.

This is difficult because there is not a station in every county measuring each pollutant. So in order to identify a good set of stations to query data from, I first looked up the two largest cities in each county, based on population.

Lat/long for 2 biggest cities (based on population) in each county

```

citylats = c(37.8044, 37.5485, 38.7743, 38.8071, 38.3527, 38.3488, 39.7285, 39.7596,
           38.1231, 38.0678, 39.2143, 39.1546, 37.9780, 38.0049, 41.7558, 41.7548,
           38.9399, 38.6688, 36.7378, 36.8252, 39.7474, 39.5243, 40.8021, 40.8665,

```

```

32.7920,32.6789,37.3614,37.3855,35.3733,35.7688,36.3275,36.3008,
38.9582,38.8080,40.4163,40.2840,34.0522,33.7701,36.9613,37.1230,
37.9735,38.1074,37.4849,37.4320,39.4457,39.4096,37.3022,37.0583,
41.4871,41.4099,37.6485,38.5149,36.6777,36.6149,38.2975,38.1749,
39.3280,39.2191,33.8366,33.7455,38.7521,38.7907,39.9341,40.3063,
33.9806,33.9425,38.5816,38.4088,36.8525,36.8125,34.1083,34.0922,
32.7157,32.6401,37.7749,37.9780,37.9577,37.7396,35.2828,35.6369,
37.6879,37.5630,34.9530,34.4208,37.3387,37.3688,36.9741,36.9102,
40.5865,40.4482,39.6763,39.5595,41.7354,41.3099,38.1041,38.2492,
38.4404,38.2324,37.6393,37.4946,39.1404,39.1165,40.1785,39.9277,
40.7310,40.4156,36.3301,36.2077,38.0297,37.9829,34.1975,34.1706,
38.5449,38.6785,39.1277,39.0954)

citylongs = c(122.2712,121.9886,119.8219,119.7960,120.9327,120.7741,121.8375,121.6219,
120.8509,120.5385,122.0094,122.1494,122.0311,121.8058,124.2026,124.1580,
119.9772,120.9872,119.7871,119.7029,122.1964,122.1936,124.1637,124.0828,
115.5631,115.4989,118.3997,118.4105,119.0187,119.2471,119.6457,119.7829,
122.6264,122.5583,120.6530,120.5394,118.2437,118.1937,120.0607,120.2602,
122.5311,122.5697,119.9663,120.0985,123.8053,123.3556,120.4830,120.8499,
120.5425,120.6791,118.9721,119.4768,121.6555,121.8221,122.2869,122.2608,
120.1833,121.0611,117.9143,117.8677,121.2880,121.2358,120.8980,121.2319,
117.3755,117.2297,121.4944,121.3716,121.4016,121.3658,117.2898,117.4350,
117.1611,117.0842,122.4194,122.0311,121.2908,121.4260,120.6596,120.6545,
122.4702,122.3255,120.4357,119.6982,121.8853,122.0363,122.0308,121.7569,
122.3917,122.2978,120.2410,120.8277,122.6345,122.3106,122.2566,122.0405,
122.7141,122.6367,120.9970,120.8460,121.6169,121.6380,122.2358,122.1792,
122.9420,123.2100,119.2966,119.3473,119.9741,120.3822,119.1771,118.8376,
121.7405,121.7733,121.5508,121.5522)
citylongs = -1*citylongs

```

Alameda: Oakland (429082) and Fremont

Alpine: Alpine Village (225) and Mesa Vista

Amador: Ione (8363) and Jackson

Butte: Chico (94776) and Paradise

Calaveras: Rancho Calaveras (5324) and Angels Camp

Colusa: Colusa (5911) and Williams

Contra Costa: Concord (129688) and Antioch

Del Norte: Crescent City (6805) and Bertsch-Oceanview

El Dorado: South Lake Tahoe (22036) and Cameron Park

Fresno: Fresno (530093) and Clovis

Glenn: Orland (7644) and Willows

Humboldt: Eureka (26998) and Arcata

Imperial: El Centro (44120) and Calexico

Inyo: Bishop (3746) and Dixon Lane-Meadow Creek

Kern: Bakersfield (383579) and Delano

Kings: Hanford (56910) and Lemoore

Lake: Clearlake (15384) and Hidden Valley Lake
Lassen: Susanville (15165) and Janesville
Los Angeles: Los Angeles (3990000) and Long Beach
Madera: Madera (65706) and Chowchilla
Marin: San Rafael (58704) and Novato
Mariposa: Mariposa (1526) and Catheys Valley
Mendocino: Fort Bragg (7359) and Willits
Merced: Merced (83316) and Los Banos
Modoc: Alturas (2509) and California Pines
Mono: Mammoth Lakes (8127) and Walker
Monterey: Salinas (156259) and Seaside
Napa: Napa (79263) and American Canyon
Nevada: Truckee (16561) and Grass Valley
Orange: Anaheim (352005) and Santa Ana
Placer: Roseville (139117) and Rocklin
Plumas: East Quincy (2489) and Chester
Riverside: Riverside (330063) and Moreno Valley
Sacramento: Sacramento (508529) and Elk Grove
San Benito: Hollister (39749) and Ridgemark
San Bernardino: San Bernardino (215941) and Fontana
San Diego: San Diego (1426000) and Chula Vista
San Francisco: San Francisco (810000) and Concord
San Joaquin: Stockton (311178) and Tracy
San Luis Obispo: San Luis Obispo (47446) and Paso Robles
San Mateo: Daly City (107008) and San Mateo
Santa Barbara: Santa Maria (107408) and Santa Barbara
Santa Clara: San Jose (1030000) and Sunnyvale
Santa Cruz: Santa Cruz (64725) and Watsonville
Shasta: Redding (91772) and Anderson
Sierra: Loyalton (700) and Downieville
Siskiyou: Yreka (7556) and Mount Shasta
Solano: Vallejo (121913) and Fairfield
Sonoma: Santa Rosa (177586) and Petaluma
Stanislaus: Modesto (215030) and Turlock
Sutter: Yuba City and South Yuba City
Tehama: Red Bluff (14283) and Corning

Trinity: Weaverville (3667) and Post Mountain

Tulare: Visalia (133800) and Tulare

Tuolumne: Phoenix Lake-Cedar Ridge (5108) and Sonora

Ventura: Oxnard (209877) and Thousand Oaks

Yolo: Davis (69289) and Woodland

Yuba: Linda (17773) and Olivehurst

Then, I created the function below to choose a group of stations that are within a certain distance (Haversine distance from the latitude and longitude) of each city that I identified in the previous step. If there are less than 5 stations associated to a given city, the distance threshold (which starts at 100km) is increased by 50km.

FUNCTION THAT SELECTS SET OF STATIONS CLOSEST TO A GIVEN LAT/LONG

```
# Function to filter stations based on spatial coverage
subset_stations_by_spatial_coverage <- function(station_data, reference_lat, reference_lon, max_distance_km) {
  # Calculate distances between stations and reference location
  distances <- distHaversine(
    cbind(station_data$longitude, station_data$latitude),
    c(reference_lon, reference_lat)
  )
  distances <- distances/1000

  # idx = which(distances == min(distances))
  # #Identify station within min distance to centroid of county
  # station_data_subset <- station_data[idx, ]

  # Subset stations within the specified max_distance_km
  idx = which(distances <= max_distance_km)
  station_data_subset <- station_data[idx, ]
  station_data_subset <- cbind(station_data_subset,distances[idx])

  while (nrow(station_data_subset) < 5){
    max_distance_km = max_distance_km + 50
    station_data_subset = subset_stations_by_spatial_coverage(station_data,
                                                              reference_lat, reference_lon, max_distance_km)
  }

  return(station_data_subset)
}

# # Construct the subset of stations based on spatial coverage criteria (test)
# reference_lat = citylats[3]
# reference_lon = citylongs[3]
# max_distance_km = 100
#
# subset_stations <- subset_stations_by_spatial_coverage(station_data2014_pm2.5, reference_lat, reference_lon)
#
# # Print the subset of stations
# print(subset_stations)

# Obtain centroid lat/longs for each county
CA.counties2 = read.csv("counties.ca.data.csv")
```

```

ca.coordinates = data.frame(CA.counties2$county, CA.counties2$lat, CA.counties2$lng)
colnames(ca.coordinates) = c("county", "lat", "long")

ca.coordinates = ca.coordinates[order(ca.coordinates$county),]
row.names(ca.coordinates) = NULL

```

IMPORTANT FUNCTION:

Given a dataset containing station locations/codes for a given pollutant and year, the function below selects 5-20 stations that are closest to the lat/longs for the two biggest cities in each county and puts the station information (code, lat, long, etc) into a dataframe.

```

#Function that finds best monitoring station for each county for a specific pollutant for a specific year
# 1 - Lead, 2 - Carbon monoxide (CO), 3 - Sulfure dioxide (SO2), 4 - Nitrogen dioxide (NO2)
# 5 - Ozone (O3), 6 - Total PM10, 7 - PM2.5

best_stations = function(stationdata,pollutant){

  subset_list = list()

  #Load lat/longs for 58x2 cities into dataframe
  CA.coords = data.frame(rep(countycodes$value_represented,each = 2),citylats,citylongs)
  colnames(CA.coords) = c("County", "Lat", "Long")

  #Find closest station for each county centroid using subset_stations_by_spatial_coverage function
  for (i in 1:nrow(CA.coords)){
    reference_lat = CA.coords$Lat[i]
    reference_lon = CA.coords$Long[i]
    max_distance_km = 100

    subset_stations <- subset_stations_by_spatial_coverage(stationdata[[pollutant]], reference_lat, reference_lon, max_distance_km)
    subset_list[[i]] = subset_stations
  }

  #Combine pairs of city lists together
  subset_list2 = list()
  sequence = seq(2,116,2)
  for(i in sequence){
    combine = rbind(subset_list[[i]],subset_list[[i-1]])
    subset_list2[[i-1]] = combine
  }
  subset_list2 = subset_list2[!sapply(subset_list2,is.null)]

  #Create a county label vector
  repnames = c()
  for(i in 1:58){
    repnames = c(repnames,nrow(subset_list2[[i]]))
  }
  countylabels = rep(countycodes$value_represented,times = repnames)

  #Format the list into dataframe
  beststations = as.data.frame(do.call(rbind, subset_list2))
  beststations = cbind(countylabels,beststations$county_name,
                       beststations$`distances[idx]`,beststations)
  colnames(beststations)[c(1,2,3)] = c("measuring_county", "station_county", "distance_apart")
}

```

```

rownames(beststations) = NULL

return(beststations)
}

# #test cases
# pm2.5_stations_2014 = best_stations(all_pollutants_station_2014data,7)
# CO_stations_2016 = best_stations(all_pollutants_station_2016data,2)

```

CREATING BEST STATION LIST/DATAFRAME FOR EACH POLLUTANT, EACH ENTRY IS A YEAR

```
#Generate list for best stations for each pollutant for each year
```

```
Lead_stations = list()
```

```
Lead_stations[[1]] = best_stations(all_pollutants_station_2014data,1)
Lead_stations[[2]] = best_stations(all_pollutants_station_2015data,1)
Lead_stations[[3]] = best_stations(all_pollutants_station_2016data,1)
Lead_stations[[4]] = best_stations(all_pollutants_station_2017data,1)
Lead_stations[[5]] = best_stations(all_pollutants_station_2018data,1)
Lead_stations[[6]] = best_stations(all_pollutants_station_2019data,1)
```

```
CO_stations = list()
```

```
CO_stations[[1]] = best_stations(all_pollutants_station_2014data,2)
CO_stations[[2]] = best_stations(all_pollutants_station_2015data,2)
CO_stations[[3]] = best_stations(all_pollutants_station_2016data,2)
CO_stations[[4]] = best_stations(all_pollutants_station_2017data,2)
CO_stations[[5]] = best_stations(all_pollutants_station_2018data,2)
CO_stations[[6]] = best_stations(all_pollutants_station_2019data,2)
```

```
S02_stations = list()
```

```
S02_stations[[1]] = best_stations(all_pollutants_station_2014data,3)
S02_stations[[2]] = best_stations(all_pollutants_station_2015data,3)
S02_stations[[3]] = best_stations(all_pollutants_station_2016data,3)
S02_stations[[4]] = best_stations(all_pollutants_station_2017data,3)
S02_stations[[5]] = best_stations(all_pollutants_station_2018data,3)
S02_stations[[6]] = best_stations(all_pollutants_station_2019data,3)
```

```
N02_stations = list()
```

```
N02_stations[[1]] = best_stations(all_pollutants_station_2014data,4)
N02_stations[[2]] = best_stations(all_pollutants_station_2015data,4)
N02_stations[[3]] = best_stations(all_pollutants_station_2016data,4)
N02_stations[[4]] = best_stations(all_pollutants_station_2017data,4)
N02_stations[[5]] = best_stations(all_pollutants_station_2018data,4)
N02_stations[[6]] = best_stations(all_pollutants_station_2019data,4)
```

```

03_stations = list()

03_stations[[1]] = best_stations(all_pollutants_station_2014data,5)
03_stations[[2]] = best_stations(all_pollutants_station_2015data,5)
03_stations[[3]] = best_stations(all_pollutants_station_2016data,5)
03_stations[[4]] = best_stations(all_pollutants_station_2017data,5)
03_stations[[5]] = best_stations(all_pollutants_station_2018data,5)
03_stations[[6]] = best_stations(all_pollutants_station_2019data,5)

PM10_stations = list()

PM10_stations[[1]] = best_stations(all_pollutants_station_2014data,6)
PM10_stations[[2]] = best_stations(all_pollutants_station_2015data,6)
PM10_stations[[3]] = best_stations(all_pollutants_station_2016data,6)
PM10_stations[[4]] = best_stations(all_pollutants_station_2017data,6)
PM10_stations[[5]] = best_stations(all_pollutants_station_2018data,6)
PM10_stations[[6]] = best_stations(all_pollutants_station_2019data,6)

# Lead.PM10_stations = list()
#
# Lead.PM10_stations[[1]] = best_stations(all_pollutants_station_2014data,7)
# Lead.PM10_stations[[2]] = best_stations(all_pollutants_station_2015data,7)
# Lead.PM10_stations[[3]] = best_stations(all_pollutants_station_2016data,7)
# Lead.PM10_stations[[4]] = best_stations(all_pollutants_station_2017data,7)
# Lead.PM10_stations[[5]] = best_stations(all_pollutants_station_2018data,7)
# Lead.PM10_stations[[6]] = best_stations(all_pollutants_station_2019data,7)

PM2.5_stations = list()

PM2.5_stations[[1]] = best_stations(all_pollutants_station_2014data,7)
PM2.5_stations[[2]] = best_stations(all_pollutants_station_2015data,7)
PM2.5_stations[[3]] = best_stations(all_pollutants_station_2016data,7)
PM2.5_stations[[4]] = best_stations(all_pollutants_station_2017data,7)
PM2.5_stations[[5]] = best_stations(all_pollutants_station_2018data,7)
PM2.5_stations[[6]] = best_stations(all_pollutants_station_2019data,7)

```

Heatmap of population with station locations marked (2015)

Do the stations provide a good spatial coverage of California? To me, the coverage is reasonable especially because most of the northern and eastern counties are where most of the sparsely populated counties are located. There are probably not that many EPA stations there as a result.

```

pollutants1_2015 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 C
pollutants2_2015 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 C

stationlats = c(unique(pollutants1_2015$latitude),unique(pollutants2_2015$latitude))

```

```

stationlongs = c(unique(pollutants1_2015$longitude),unique(pollutants2_2015$longitude))

station_points = data.frame(stationlats,stationlongs)

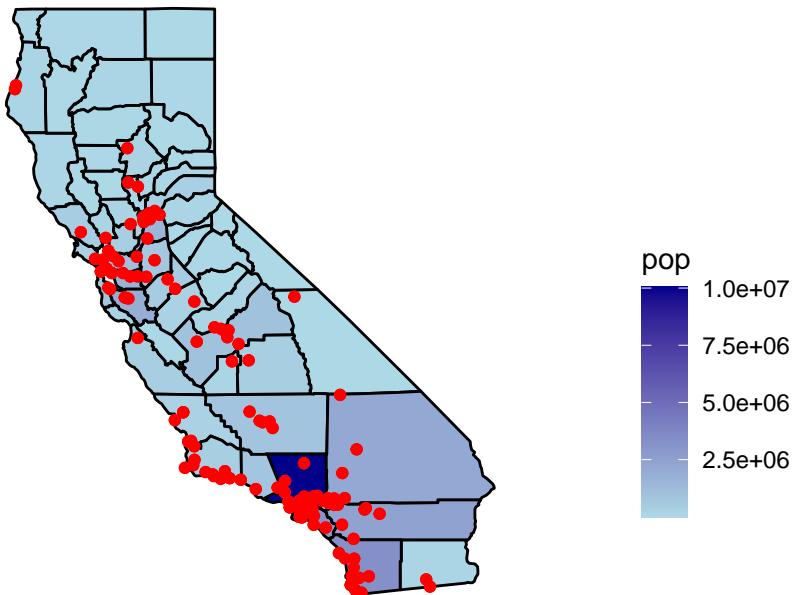
#Plot
gg_pop_stations <- ggplot() +
  geom_polygon(data = merged_data, aes(x = long, y = lat, group = group, fill = pop),
    color = "black") +
  coord_fixed(ratio = 1.3, xlim = c(-125, -112), ylim = c(30, 42)) +
  theme_void() +
  labs(title = "Heatmap of County Populations with Station Locations for 2015") +
  scale_fill_gradient(low = "lightblue", high = "darkblue")

# Add points
gg_pop_stations <- gg_pop_stations +
  geom_point(data = station_points, aes(x = stationlongs, y = stationlats),
    color = "red", size = 1.5)

print(gg_pop_stations)

```

Heatmap of County Populations with Station Locations for 2015



Downloading and aggregating air quality data using direct API calls

Given a set of 5-20 monitoring stations for each county, we loop through its station codes (for each year 2014-2019) and query using the EPA's AQs function. This function only allows you to query a maximum of 4 parameters at once for a single year, so two calls to the function have to be made for each year (4 and 3).

(BELOW SHOWS IT BEING DONE FOR 2019)

END GOAL FINAL FORM: ONE BIG DATAFRAME (ALL POLLUTANTS ALL YEARS TOGETHER, USE FILTER TO SEPARATE)

```
# 1 - Lead, 2 - Carbon monoxide (CO), 3 - Sulfure dioxide (SO2), 4 - Nitrogen dioxide (NO2)
# 5 - Ozone (O3), 6 - Total PM10, 7 - PM2.5

stations2019x = rbind(Lead_stations[[6]], CO_stations[[6]],
                      SO2_stations[[6]], NO2_stations[[6]])
stations2019y = rbind(O3_stations[[6]], PM10_stations[[6]], PM2.5_stations[[6]])

sitenums2019x = stations2019x %>% select(county_code, site_number) %>% unique() #198 stations
sitenums2019y = stations2019y %>% select(county_code, site_number) %>% unique() #178 stations

#Trying EPA R Package query (took 15 + 20 min!) gives us a dataframe
ccodes = sitenums2019y$county_code
snums = sitenums2019y$site_number
str1 = "2019-01-01"
str2 = "2019-12-31"

pollutants1_2019 = aqs_dailysummary_by_site(parameter = c("14129", "42101", "42401", "42602"), bdate = as.Date(str1))
pollutants2_2019 = aqs_dailysummary_by_site(parameter = c("44201", "81102", "88101"), bdate = as.Date(str2))

###SAVE LIST LOCALLY
saveRDS(pollutants2_2019, file = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper/2019_wildfire.RData")
```

After querying the raw data for each pollutant and each year, we want to go through it and remove any data from stations that have “bad” data. The standards that I set are that for a given year, a station should have at least 240 of the 365 days included for a year of data. Additionally, I applied a Hampel filter to identify outliers and if there are more than 14 consecutive outliers in the data i.e., days in a row with measurements that are abnormal, I considered the data from that station to not be suitable for inclusion in the final dataset.

NOTE: the daily values reported in the raw dataset are actually daily averages from periodic measurements made by the station throughout the day

After data from so called “bad” stations were removed, another function is applied which aggregates the daily observations into a monthly median. The raw data is in the form of a single dataframe, which is fed into the important function raw_transform(). This function uses several functions to create a list of dataframes, one for each county, which represent the monthly median measurements for a certain pollutant in a given county in a given year.

QUALITY CHECK FUNCTION FOR STATION DATA: WANT TO ADDRESS OUTLIERS, MISSINGNESS

```
#Given a dataset like CO2016 (list of 1000ish stations), check for 2/3 missing data and for strings of NAs
station_quality_check = function(station_data){
  l = length(station_data)
  badindex = c()
  consecutive_outliers = list()

  for (i in 1:l){
    aqi = station_data[[i]]$aqi
    pollutant_level = station_data[[i]]$arithmetic_mean
```

```

#check for outliers in AQI
median.aqi = median(na.omit(aqi))
mad.aqi = mad(na.omit(aqi))

min.aqi = median.aqi-(3*mad.aqi)
max.aqi = median.aqi+(3*mad.aqi)

outliers.aqi = which(aqi < min.aqi | aqi > max.aqi)

result.aqi = rle(diff(outliers.aqi))

#check for outliers in pollutant measure
median.pollutant = median(na.omit(pollutant_level))
mad.pollutant = mad(na.omit(pollutant_level))

min.pollutant = median.pollutant-(3*mad.pollutant)
max.pollutant = median.pollutant+(3*mad.pollutant)

outliers.pollutant = which(pollutant_level < min.pollutant | pollutant_level > max.pollutant)

result.pollutants = rle(diff(outliers.pollutant))

if (nrow(station_data[[i]]) < 240){
  badindex = c(badindex,i)
}

else if (any(result.aqi$lengths >= 14 & result.aqi$values == 1) == TRUE){
  badindex = c(badindex,i)
}

else if (any(result.pollutants$lengths >= 14 & result.pollutants$values == 1) == TRUE){
  badindex = c(badindex,i)
}

consecutive_outliers[[i]] = c("AQI",outliers.aqi,"POLLUTANTS",outliers.pollutant)
}

bad_list = list(badindex,consecutive_outliers)

return(bad_list)
}

#Test on CO2016 and CO2017
# station_quality_check(CO2016) #returns 61 "bad stations" out of 1230
#
# removeidx = station_quality_check(S022017)[[1]] #returns 715 out of 1056 "bad stations"
#
# test = S022017[-removeidx]

```

FUNCTION THAT AGGREGATES DAILY DATA INTO MONTHLY MEDIAN

```

monthly_agg = function(pollutantdata){
  #Aggregating all the station data at once
  date = ymd(pollutantdata$date_local)
  df2 <- pollutantdata                                     # Duplicate data
  df2$year_month <- floor_date(date,"month")   # Create year-month column
  df3 = df2 %>% select(county,site_number,arithmetic_mean,aqi,year_month) %>% as.data.frame()

  df3$arithmetic_mean = as.numeric(df3$arithmetic_mean)
  df3$aqi[which(df3$aqi == "NULL")] = NA
  df3$aqi = as.numeric(df3$aqi)

  df.agg = df3 %>% group_by(year_month) %>% dplyr::summarize(arithmetic_mean = median(na.omit(arithmetic_mean)))
  return(df.agg)
}

###NEW FUNCTION (6/20/25) TO GET MONTHLY TOTALS OF PM2.5
monthly_agg2 = function(pollutantdata){
  #Aggregating all the station data at once
  date = ymd(pollutantdata$date_local)
  df2 <- pollutantdata                                     # Duplicate data
  df2$year_month <- floor_date(date,"month")   # Create year-month column
  df3 = df2 %>% select(county,site_number,arithmetic_mean,aqi,year_month) %>% as.data.frame()

  df3$arithmetic_mean = as.numeric(df3$arithmetic_mean)
  df3$aqi[which(df3$aqi == "NULL")] = NA
  df3$aqi = as.numeric(df3$aqi)

  df.agg = df3 %>% group_by(year_month,site_number) %>% dplyr::summarize(monthly_total = sum(na.omit(arithmetic_mean)))
  df.agg2 = df.agg %>% group_by(year_month) %>% dplyr::summarize(avg_monthly_total = median(monthly_total))

  return(df.agg2)
}

```

IMPORTANT FUNCTION: TRANSFORMING RAW DATA TO FINAL FORM

```

# Group 1: 14129 - Lead, 421012 - Carbon monoxide (CO), 42401 - Sulfur dioxide (SO2), 42602 - Nitrogen
# Group 2: 44201 - Ozone (O3), 81102 - Total PM10, 88101 - PM2.5

raw_transform = function(rawdata,reference_list,standard){

  ###SEPARATE DF INTO A LIST OF DFs

  matched_list = list()

  if(missing(standard)){
    for (i in 1:nrow(reference_list)){
      data = rawdata %>% filter(county_code == reference_list$county_code[i], site_number == reference_list$site_number[i])
      matched_list[[i]] = data
    }
  } else {
    for (i in 1:nrow(reference_list)){
      data = rawdata %>% filter(county_code == reference_list$county_code[i], site_number == reference_list$site_number[i])
      matched_list[[i]] = data
    }
  }
}

```

```

    data = rawdata %>% filter(county_code == reference_list$county_code[i], site_number == reference_list$site_number[i])
    matched_list[[i]] = data
}
}

names(matched_list) = reference_list$measuring_county

####STATION QUALITY CHECK
removeidx = station_quality_check(matched_list)[[1]]
good_matched_list = matched_list[-removeidx]

#Convert list back into one big dataframe
temp = as.data.frame(do.call(rbind, good_matched_list)) #TOO MANY ROWS RIGHT?
good_df = unique.data.frame(temp)

####MAKE A LIST OF COMBINED STATION DATA FOR EACH COUNTY
mid_list = list()

for (i in unique(reference_list$measuring_county)){
  df_new = data.frame(good_df[1,])
  subset = reference_list %>% filter(measuring_county == i) %>% select(county_code,site_number)
  for (j in 1:nrow(subset)){
    pull = good_df %>% filter(county_code == reference_list$county_code[j], site_number == reference_list$site_number[j])
    df_new = rbind(df_new,pull)
  }
  df_new = df_new[-1,]
  mid_list[[i]] = df_new
}

####AGGREGATE DAILY DATA TO MONTHLY FOR EACH COUNTY
final_list = lapply(mid_list,monthly_agg2)

return(final_list)
}

```

Assembling daily averages for one pollutant, one year from mid_list:

```

Lead_daily19 = data.frame()

for (i in 1:58){
  data = mid_list[[i]]

  county_df = data19 %>% dplyr::group_by(date_local) %>%
    dplyr::summarise(daily_avg = mean(arithmetic_mean,na.rm = TRUE),
                     daily_avg_aqi = mean(aqi,na.rm = TRUE))

  county = rep(counties[i],nrow(county_df))

```

```

county_df = cbind(county, county_df)

Lead_daily19 = rbind(Lead_daily19, county_df)
}

```

When assembling final datasets, note that certain pollutant standards are used bc they have values for AQI... the ones I used were:

Lead: Lead 3-Month 2009 ?? Has all NAs for AQI

CO: CO 8-hour 1971

SO2: SO2 1-hour 2010

NO2: NO2 1-hour 2010

O3: Ozone 8-hour 2015 ; sample duration should be 8 HR

PM10: PM10 24-hour 2006

PM2.5: PM25 24-hour 2012

APPLY RAW TRANSFORM FUNCTION TO ALL POLLUTANTS FOR ALL YEARS (BELOW SHOWS IT BEING DONE FOR 2019)

```

#Load raw data
pollutants1_2019 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 C...
pollutants2_2019 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 C...

Lead2019 = pollutants1_2019 %>% filter(parameter_code == "14129")
CO2019 = pollutants1_2019 %>% filter(parameter_code == "42101")
S022019 = pollutants1_2019 %>% filter(parameter_code == "42401")
N022019 = pollutants1_2019 %>% filter(parameter_code == "42602")
O32019 = pollutants2_2019 %>% filter(parameter_code == "44201")
PM102019 = pollutants2_2019 %>% filter(parameter_code == "81102")
PM2.52019 = pollutants2_2019 %>% filter(parameter_code == "88101")

Lead2019_final = raw_transform(rawdata = Lead2019,reference_list = Lead_stations[[6]],standard = "Lead 3-Month 2009")
CO2019_final = raw_transform(rawdata = CO2019,reference_list = CO_stations[[6]],standard = "CO 8-hour 1971")
S022019_final = raw_transform(rawdata = S022019,reference_list = S02_stations[[6]],standard = "SO2 1-hour 2010")
N022019_final = raw_transform(rawdata = N022019,reference_list = N02_stations[[6]],standard = "NO2 1-hour 2010")
O32019_final = raw_transform(rawdata = O32019,reference_list = O3_stations[[6]],standard = "Ozone 8-hour 2015")
PM102019_final = raw_transform(rawdata = PM102019,reference_list = PM10_stations[[6]],standard = "PM10 24-hour 2006")
PM2.52019_final = raw_transform(rawdata = PM2.52019,reference_list = PM2.5_stations[[6]],standard = "PM2.5 24-hour 2012")

NEED TOTAL PM2.5 PER MONTH FOR PAPER 2
pollutants2_2014 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 C...
PM2.52014 = pollutants2_2014 %>% filter(parameter_code == "88101")

PM2.52014_final = raw_transform(rawdata = PM2.52014,reference_list = PM2.5_stations[[1]],standard = "PM2.5 24-hour 2012")
p1 = as.data.frame(do.call(rbind, PM2.52019_final))

```

```

p1 = cbind(p1,rep(pollutants$parametercodes.code[7],nrow(p1))) #maybe change parameter codes to 1-7?
colnames(p1) = c("Year-Month","Value","AQI","Pollutant")

p2 = as.data.frame(do.call(rbind, PM2.52018_final))
p2 = cbind(p2,rep(pollutants$parametercodes.code[7],nrow(p2))) #maybe change parameter codes to 1-7?
colnames(p2) = c("Year-Month","Value","AQI","Pollutant")

p3 = as.data.frame(do.call(rbind, PM2.52017_final))
p3 = cbind(p3,rep(pollutants$parametercodes.code[7],nrow(p3))) #maybe change parameter codes to 1-7?
colnames(p3) = c("Year-Month","Value","AQI","Pollutant")

p4 = as.data.frame(do.call(rbind, PM2.52016_final))
p4 = cbind(p4,rep(pollutants$parametercodes.code[7],nrow(p4))) #maybe change parameter codes to 1-7?
colnames(p4) = c("Year-Month","Value","AQI","Pollutant")

p5 = as.data.frame(do.call(rbind, PM2.52015_final))
p5 = cbind(p5,rep(pollutants$parametercodes.code[7],nrow(p5))) #maybe change parameter codes to 1-7?
colnames(p5) = c("Year-Month","Value","AQI","Pollutant")

p6 = as.data.frame(do.call(rbind, PM2.52014_final))
p6 = cbind(p6,rep(pollutants$parametercodes.code[7],nrow(p6))) #maybe change parameter codes to 1-7?
colnames(p6) = c("Year-Month","Value","AQI","Pollutant")

total_pm25_1419 = rbind(p6,p5,p4,p3,p2,p1)
c = rep(counties,each=12)
total_pm25_1419$county = rep(c,6)

saveRDS(total_pm25_1419,"total_pm25_1419_df.rds")

```

COMBINING EACH POLLUTANTS DATASET INTO A SINGLE DATAFRAME FOR THE YEAR (BELOW SHOWS IT BEING DONE FOR 2019)

```

###Combine final data into one dataframe for 2014
test1 = as.data.frame(do.call(rbind, Lead2019_final))
test1 = cbind(test1,rep(pollutants$parametercodes.code[1],nrow(test1))) #maybe change parameter codes to 1-7?
colnames(test1) = c("Year-Month","Value","AQI","Pollutant")

test2 = as.data.frame(do.call(rbind, CO2019_final))
test2 = cbind(test2,rep(pollutants$parametercodes.code[2],nrow(test2))) #maybe change parameter codes to 1-7?
colnames(test2) = c("Year-Month","Value","AQI","Pollutant")

test3 = as.data.frame(do.call(rbind, SO22019_final))
test3 = cbind(test3,rep(pollutants$parametercodes.code[3],nrow(test3))) #maybe change parameter codes to 1-7?
colnames(test3) = c("Year-Month","Value","AQI","Pollutant")

test4 = as.data.frame(do.call(rbind, NO22019_final))
test4 = cbind(test4,rep(pollutants$parametercodes.code[4],nrow(test4))) #maybe change parameter codes to 1-7?
colnames(test4) = c("Year-Month","Value","AQI","Pollutant")

test5 = as.data.frame(do.call(rbind, O32019_final))
test5 = cbind(test5,rep(pollutants$parametercodes.code[5],nrow(test5))) #maybe change parameter codes to 1-7?
colnames(test5) = c("Year-Month","Value","AQI","Pollutant")

test6 = as.data.frame(do.call(rbind, PM102019_final))

```

```

test6 = cbind(test6,rep(pollutants$parametercodes.code[6],nrow(test6))) #maybe change parameter codes to something else
colnames(test6) = c("Year-Month","Value","AQI","Pollutant")

test7 = as.data.frame(do.call(rbind, PM2.52019_final))
test7 = cbind(test7,rep(pollutants$parametercodes.code[7],nrow(test7))) #maybe change parameter codes to something else
colnames(test7) = c("Year-Month","Value","AQI","Pollutant")

#Combine each pollutant dataset into one big dataset for the year
final_data19 = rbind(test1,test2,test3,test4,test5,test6,test7)

###SAVE FINAL DATASET LOCALLY
saveRDS(final_data19,file = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Data.RDS")

```

HAVE TO CLEAN DATA BEFORE FINALIZING:

Having combined the monthly medians for every county for a single year for each pollutant into a single dataframe, one final cleaning step must be performed before putting each years' data together. The AQI value that is in each row corresponds to the AQI standardized measurement for that specific pollutant. Each pollutant has a different standardizing equation, but once they are all standardized as they are in the dataset and they can be compared against each other. The reported AQI measurement for a given day is just the maximum of the AQI values corresponding to each of the 7 pollutants. So the maximum AQI value (among 6 values bc Lead observations never have AQI values) was found for each month and that value was set as the actual AQI value for that month in all corresponding rows.

FIND MAX AQI (AMONG THE 7 POLLUTANTS) FOR EACH MONTH -> SET AS ACTUAL AQI FOR THAT MONTH

(BELOW SHOWS IT BEING DONE FOR 2019)

```

#Do for each year
months = c("01","02","03","04","05","06","07","08","09","10","11","12")

###Do for each year
for (i in 1:58){
  idx1 = which(stringr::str_starts(rownames(final_data19), counties[i]))
  subset1 = final_data19[idx1,]
  subset1$`Year-Month` = as.Date(subset1$`Year-Month`)

  for (j in months){
    #Filter by county and date
    date = paste0("2019-",j,"-01")
    date = as.Date(date)
    subset2 = subset1 %>% filter(`Year-Month` == as.Date(date))

    trueAQI = max(na.omit(subset2$AQI))

    idx2 = which(subset1$`Year-Month` == date)
    subset1$AQI[idx2] = trueAQI
  }

  final_data19[idx1,] = subset1
}

###SAVE FINAL DATASET LOCALLY
saveRDS(final_data19,file = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Data.RDS")

```

COMBINING EACH YEARS DATASET INTO ONE BIG TIDY DATAFRAME FOR AIR QUALITY COVARIATES

FILL IN YOUR OWN FILE DIRECTORIES HERE! START WORKING W ACTUAL EPA DATA FROM HERE ON

```
final_data14 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/1.RData")
final_data15 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/2.RData")
final_data16 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/3.RData")
final_data17 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/4.RData")
final_data18 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/5.RData")
final_data19 = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/6.RData")

final_EPA_data = rbind(final_data14,final_data15,final_data16,final_data17,
                      final_data18,final_data19)
saveRDS(final_EPA_data,file = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Data/epa.RData")

final_EPA_data = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Data/epa.RData")
head(final_EPA_data)

##           Year-Month   Value  AQI Pollutant
## Alameda.1 2014-01-01  0.0500  60    14129
## Alameda.2 2014-02-01  0.0520  28    14129
## Alameda.3 2014-03-01  0.0655  34    14129
## Alameda.4 2014-04-01  0.0360  40    14129
## Alameda.5 2014-05-01  0.0300  38    14129
## Alameda.6 2014-06-01  0.0230  37    14129
```

Adding cluster labels to the EPA data and aggregating based on clusters

Once the EPA data is in the correct format (one big dataframe) at the county level, we now need to aggregate it to the cluster level according to the SKATER cluster labels from before. A population weighted mean was used to aggregate here as well

```
Cluster = rep(1,length(final_EPA_data))
final_EPA_agg_data = cbind(final_EPA_data,Cluster)

for (i in 1:58){
  idx = which(stringr::str_starts(rownames(final_EPA_agg_data), counties[i]))
  final_EPA_agg_data$Cluster[idx] = clusterlabels$Cluster[i]
}

Time = c(rep(c(1:12),58),rep(13:24,58),rep(25:36,58),rep(37:48,58),rep(49:60,58),rep(61:72,58))
Time = rep(Time,7)
final_EPA_agg_data = cbind(Time,final_EPA_agg_data)
```

AGGREGATE CLUSTER DATA AND COMBINE INTO ONE DATAFRAME

```
countypops = CA_data %>% filter(Year > 2013) %>% select(Total_Pop,County,Year) %>% unique()
countypops = cbind(countypops,Cluster = rep(clusterlabels$Cluster,each=6))
countypops$County = rep(counties,each=6)

temp_EPA_agg_data = data.frame(final_EPA_agg_data[1,-2])
```

```

num_clus = length(unique(clusterlabels$Cluster))

for (k in 1:num_clus){

  EPA_clus_k = data.frame(final_EPA_agg_data[1,-2])

  for (i in pollutants$parametercodes.code){
    pollutant_data = final_EPA_agg_data %>% filter(Pollutant == i)

    cluster_data = pollutant_data %>% filter(Cluster == k)
    cluster_data$value = scale(cluster_data$value)
    cluster_data$aqi = scale(cluster_data$aqi)
    year = 2014

    for(j in 1:72){
      cluster_data_j = cluster_data %>% filter(Time == j)
      cluster_counties = countypops %>% filter(Cluster == k,Year == year)

      pops = countypops %>% filter(Year == year,Cluster == k) %>% select(Total_Pop)
      cluster.pop = sum(pops)
      cluster.popweights = pops/cluster.pop

      value_wmean = weighted.mean(cluster_data_j$value,cluster.popweights$Total_Pop)
      aqi_wmean = weighted.mean(cluster_data_j$aqi,cluster.popweights$Total_Pop)
      insert = data.frame(Time = j,value_wmean,aqi_wmean,
                           Pollutant = i,Cluster = k)
      colnames(insert) = colnames(EPA_clus_k)

      EPA_clus_k = rbind(EPA_clus_k,insert)

      if ((j>12) & (j<25)){
        year = 2015
      }

      else if ((j>24) & (j<37)){
        year = 2016
      }

      else if ((j>36) & (j<49)){
        year = 2017
      }

      else if ((j>48) & (j<61)){
        year = 2018
      }

      else if ((j>60) & (j<73)){
        year = 2019
      }

      else{
        year = 2014
      }
    }
  }
}

```

```

}

EPA_clus_k = EPA_clus_k[-1,]
trueAQI = EPA_clus_k$AQI[1:72]
trueAQI = rep(trueAQI,7)
EPA_clus_k$AQI = trueAQI
rownames(EPA_clus_k) = NULL

temp_EPA_agg_data = rbind(temp_EPA_agg_data,EPA_clus_k)

}

final_EPA_agg_data = temp_EPA_agg_data[-1,]

saveRDS(final_EPA_agg_data,file = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1.RDS")

final_EPA_agg_data = readRDS("C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1.RDS")

head(final_EPA_agg_data)

##   Time      Value        AQI Pollutant Cluster
## 1    1  3.489028  2.68675349    14129      1
## 2    2  3.709848 -1.23791773    14129      1
## 3    3  5.200381 -0.46590289    14129      1
## 4    4  1.943290  0.26531624    14129      1
## 5    5  1.280831  0.02551449    14129      1
## 6    6  0.507962 -0.23586863    14129      1

```

Plot time series of each EPA variable

```

library(astsa)

##
## Attaching package: 'astsa'

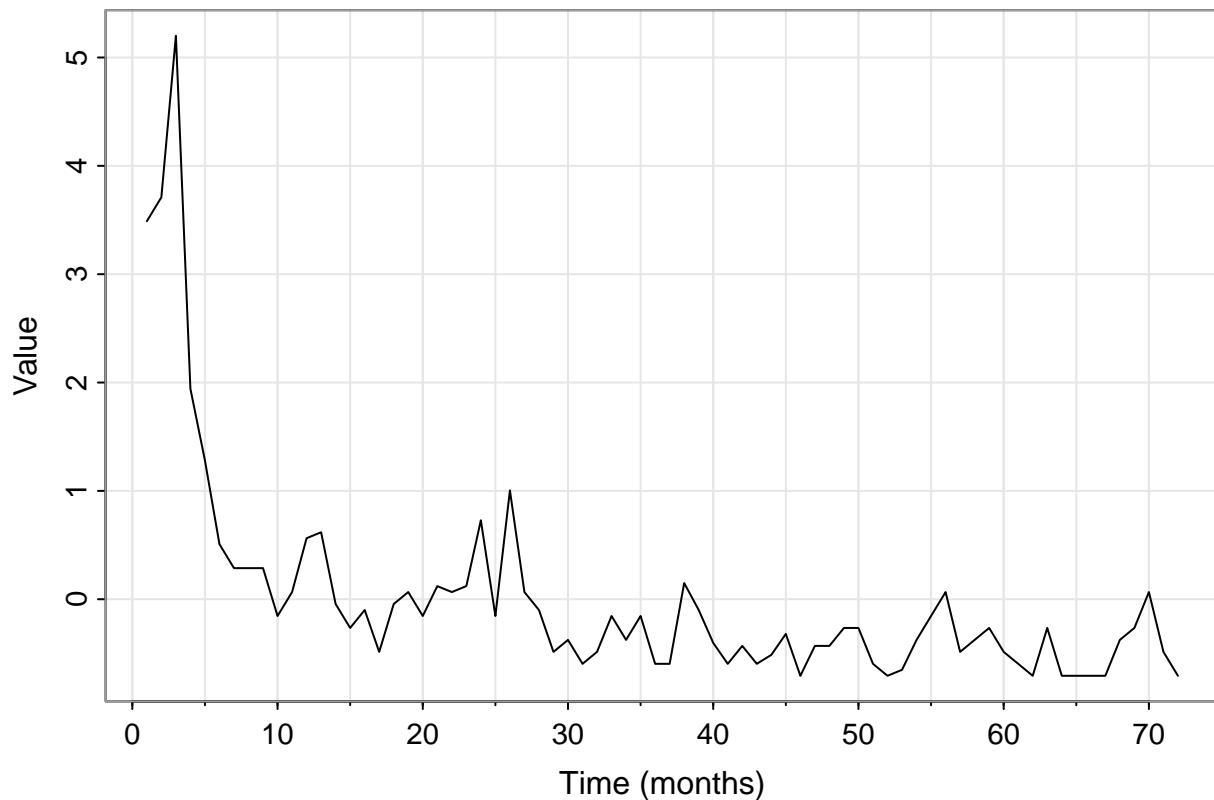
## The following object is masked from 'package:maps':
## 
##     unemp

for (i in pollutants$parametercodes.code){
  for (j in 1:num_clus){
    EPA_clus_ts = final_EPA_agg_data %>% filter(Pollutant == i) %>% filter(Cluster == j)

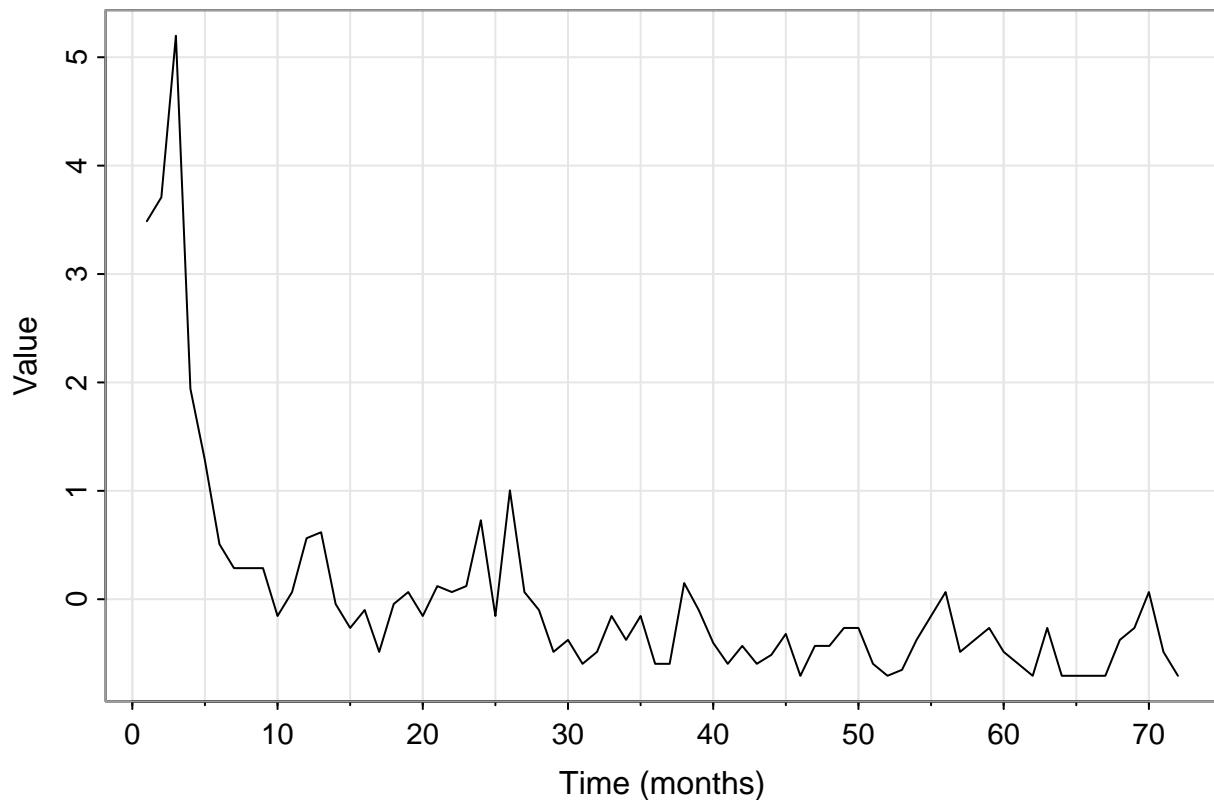
    title = sprintf("Pollutant %s - Cluster %1.0f",i,j)
    tsplot(EPA_clus_ts$value,main = title,xlab = "Time (months)",ylab = "Value")
  }
}

```

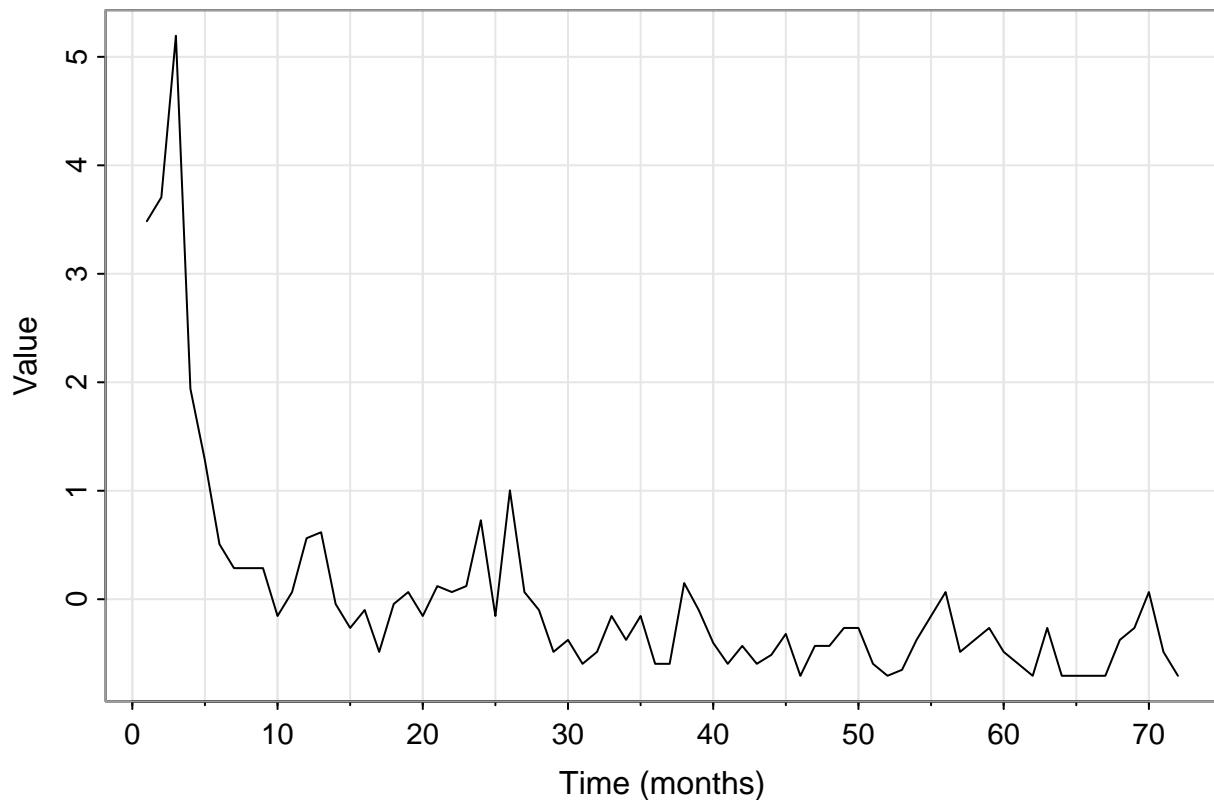
Pollutant 14129 – Cluster 1



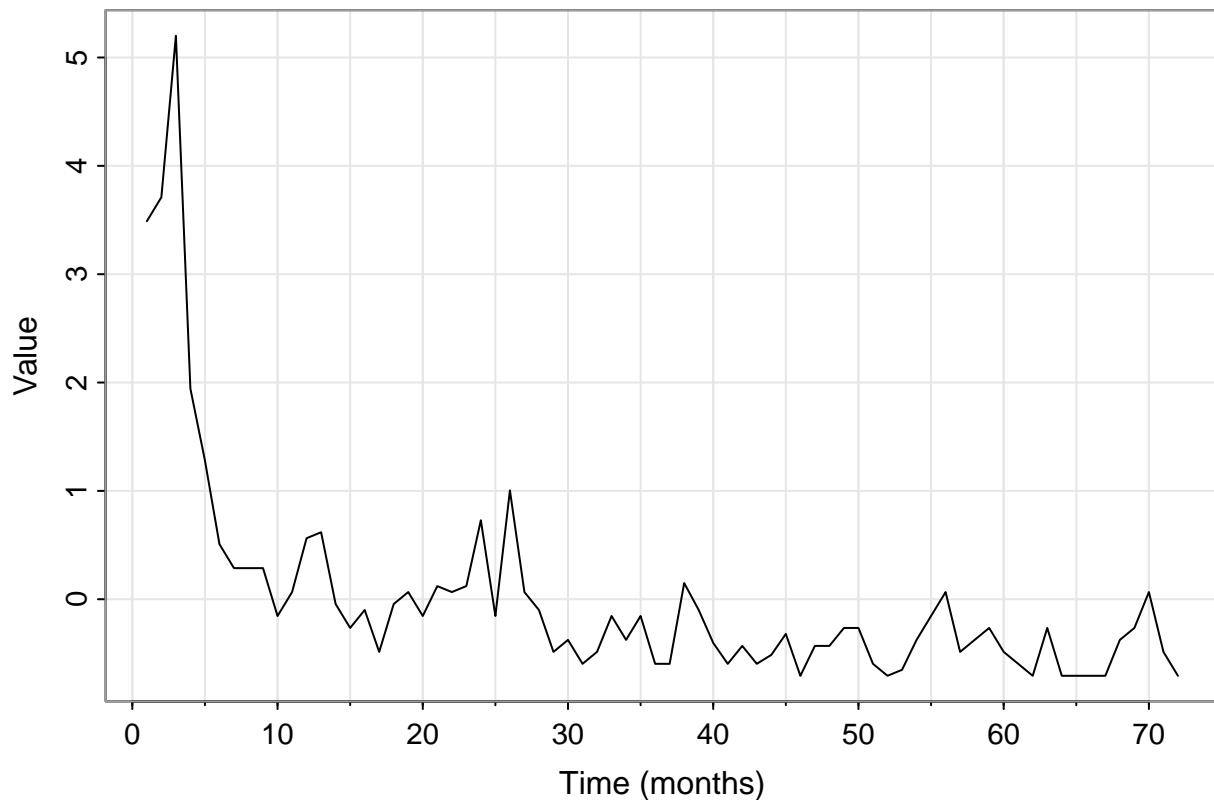
Pollutant 14129 – Cluster 2



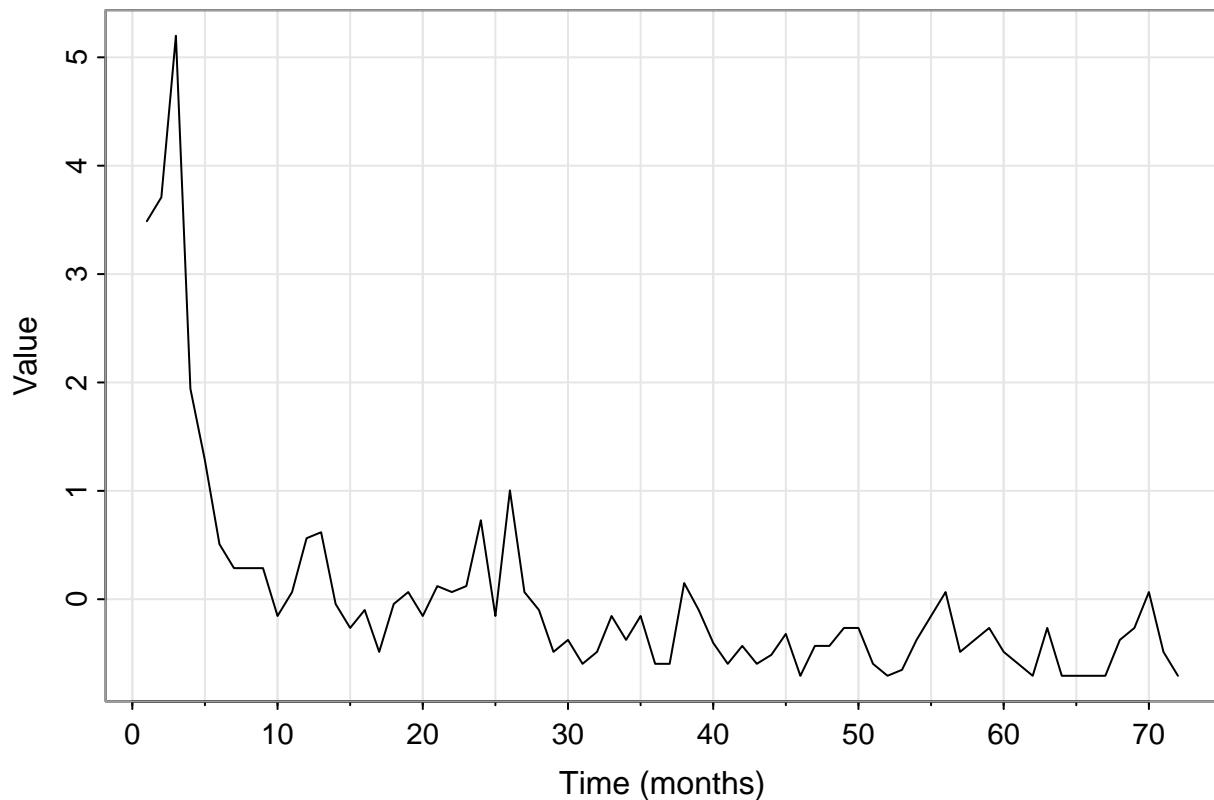
Pollutant 14129 – Cluster 3



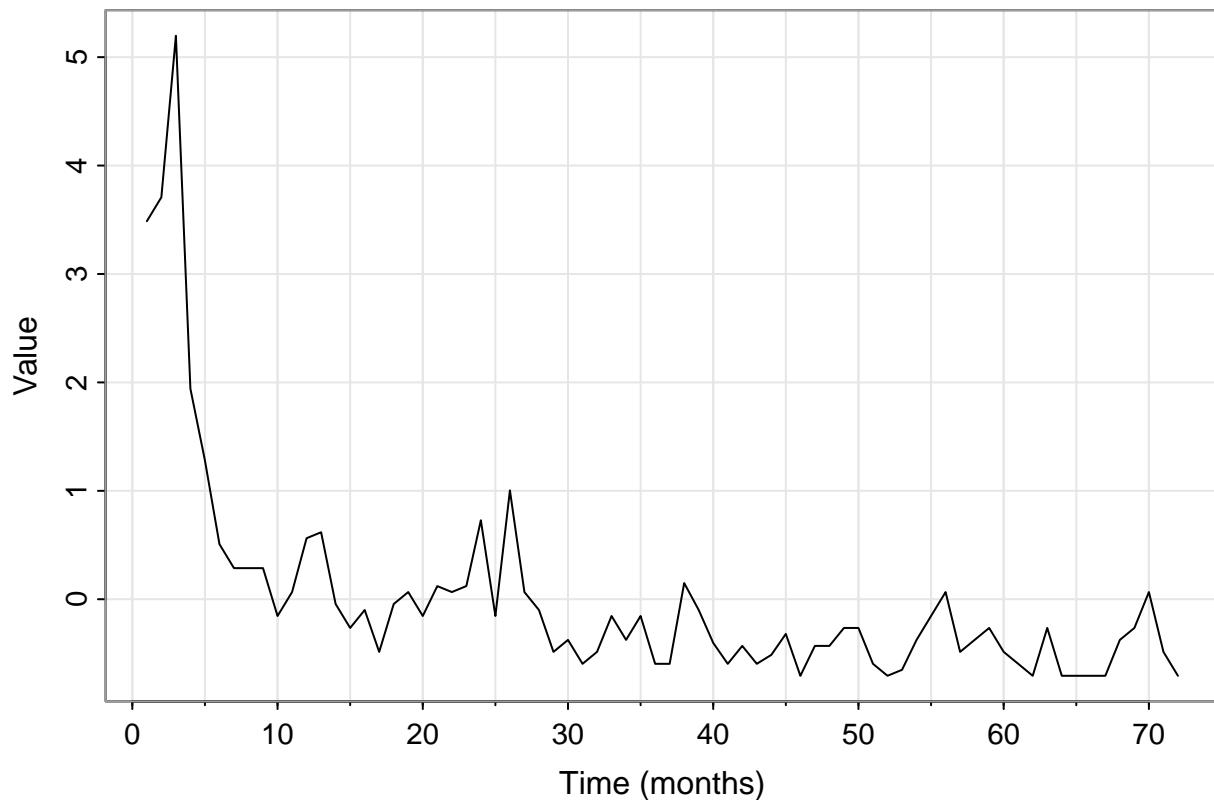
Pollutant 14129 – Cluster 4



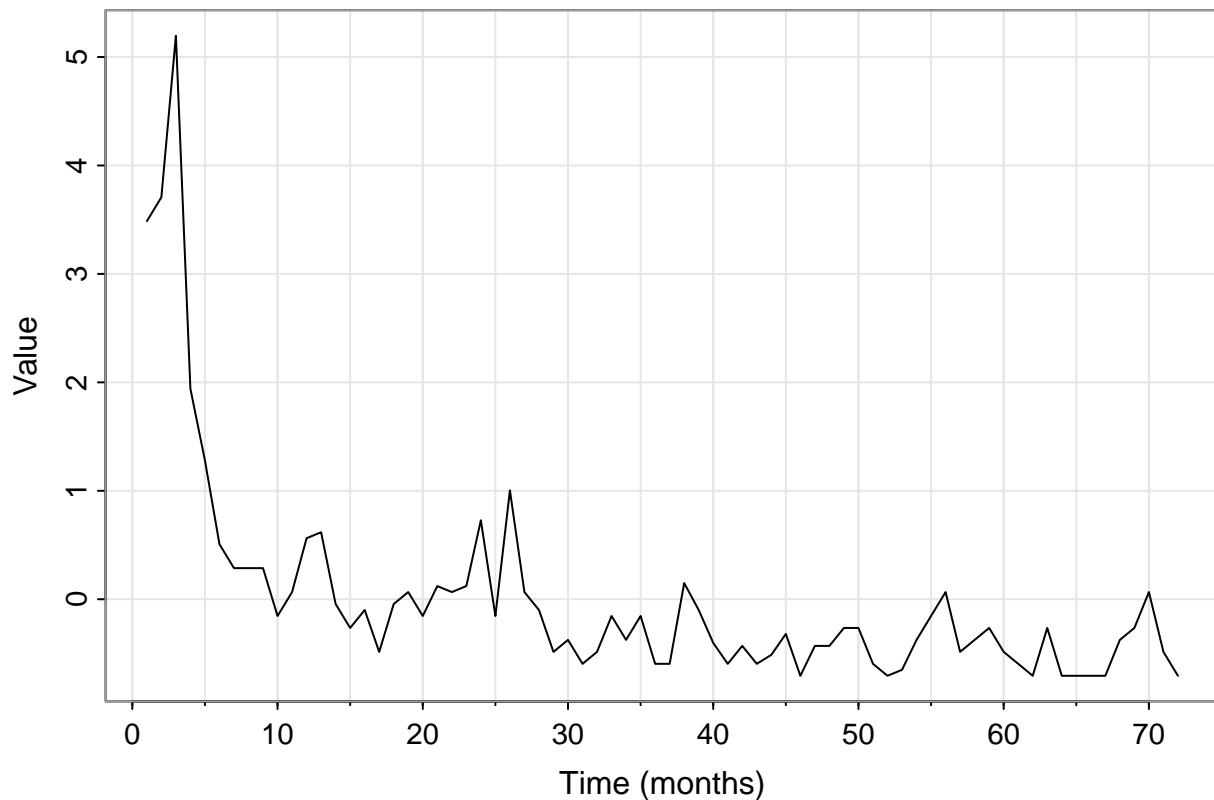
Pollutant 14129 – Cluster 5



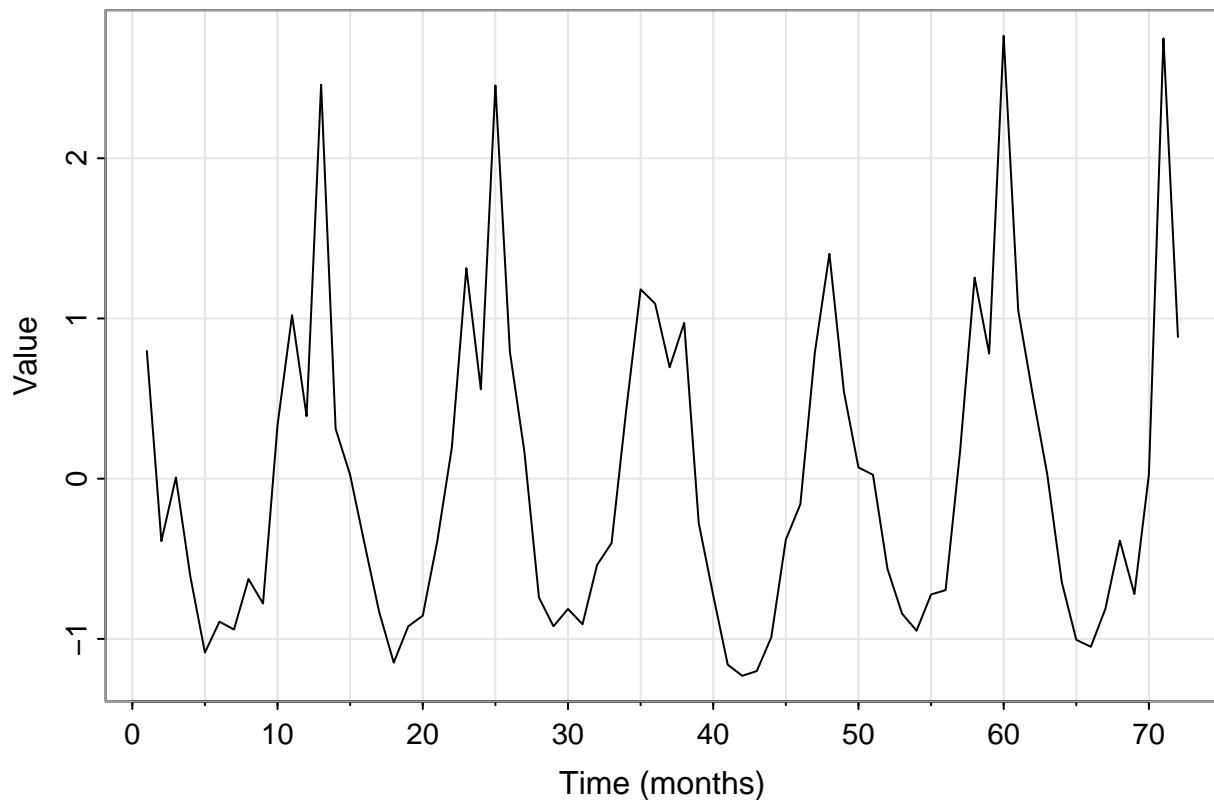
Pollutant 14129 – Cluster 6



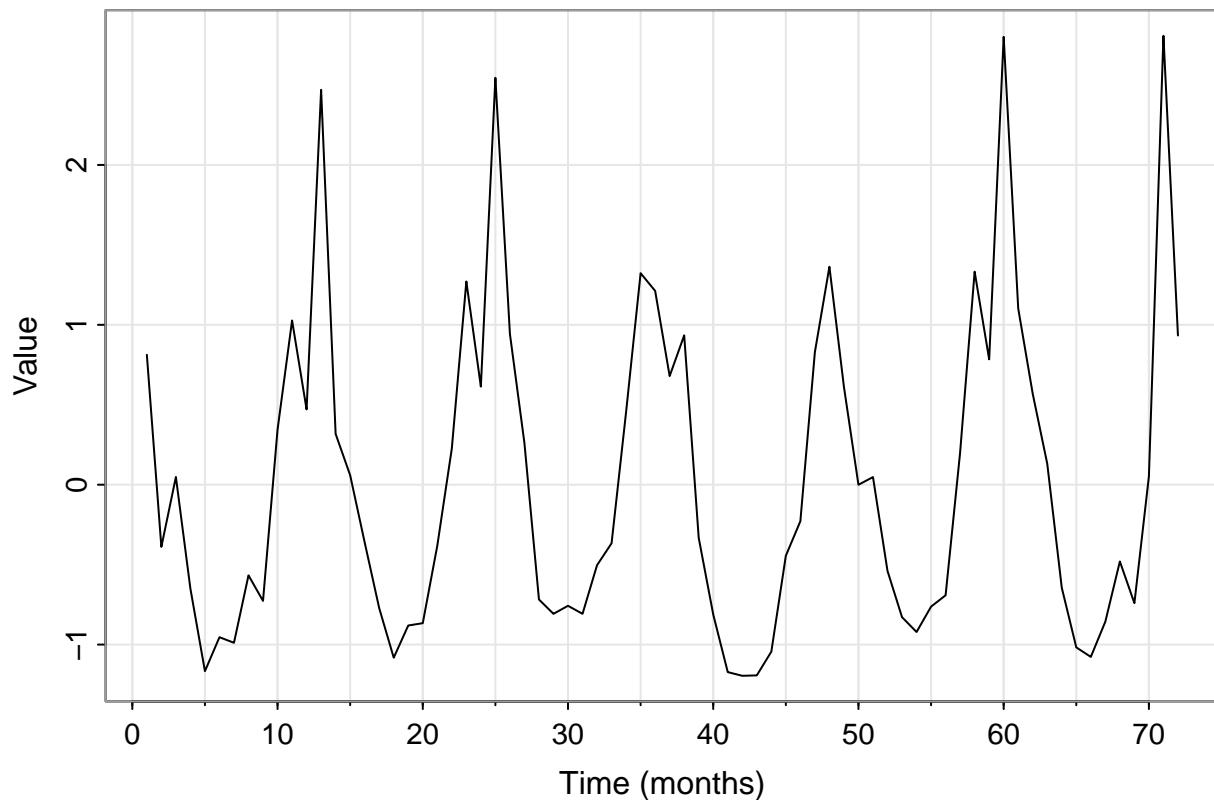
Pollutant 14129 – Cluster 7



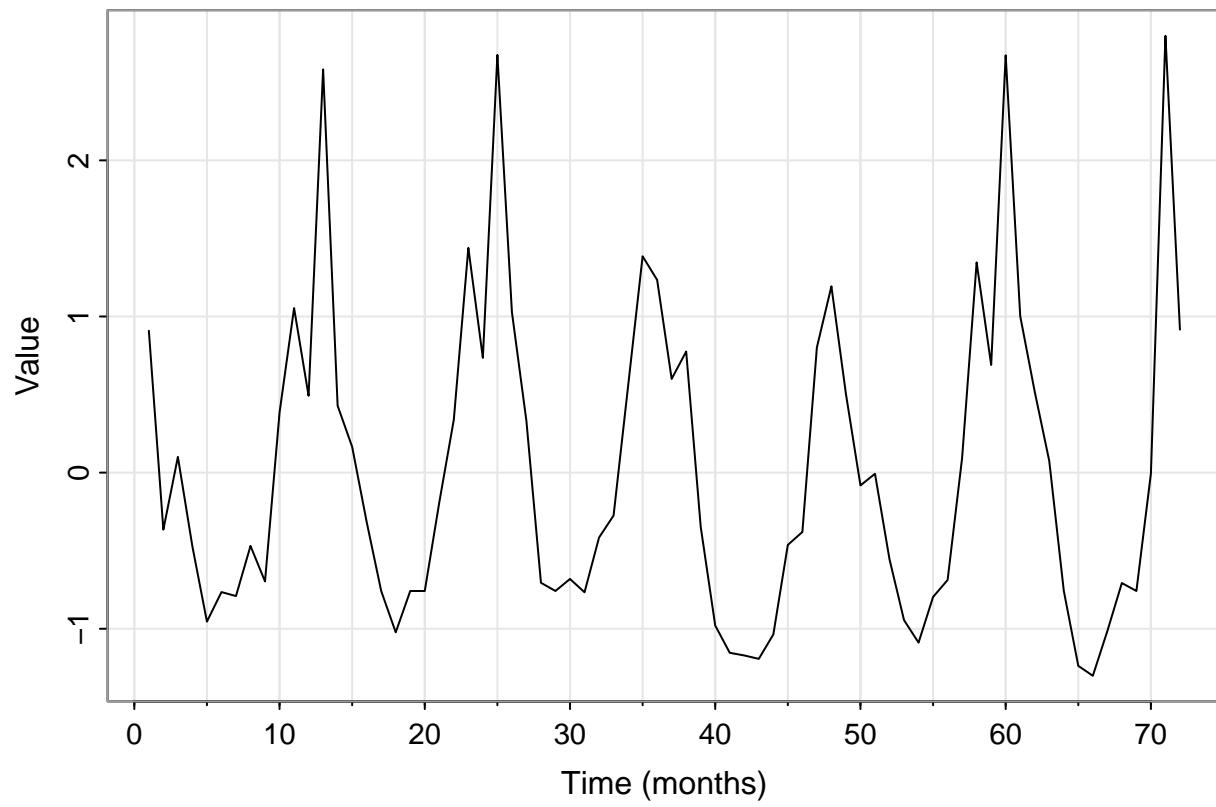
Pollutant 42101 – Cluster 1



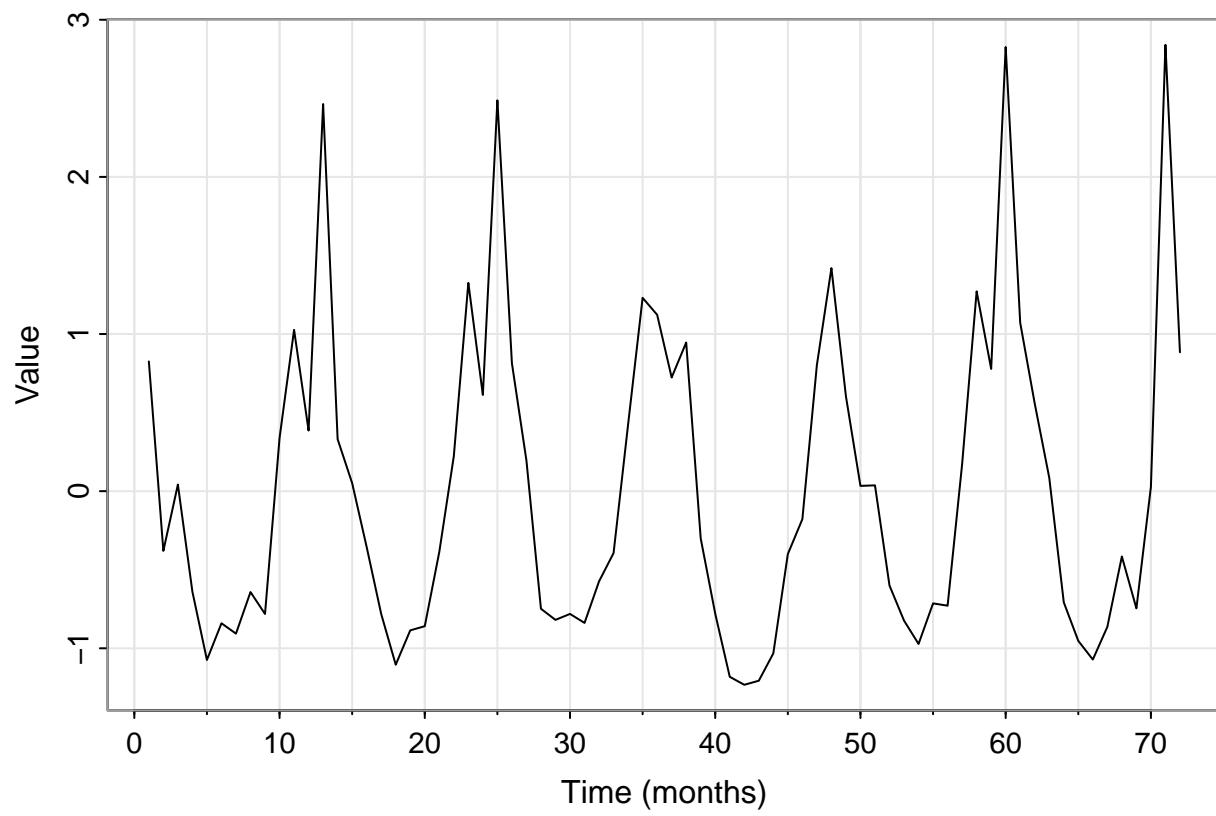
Pollutant 42101 – Cluster 2



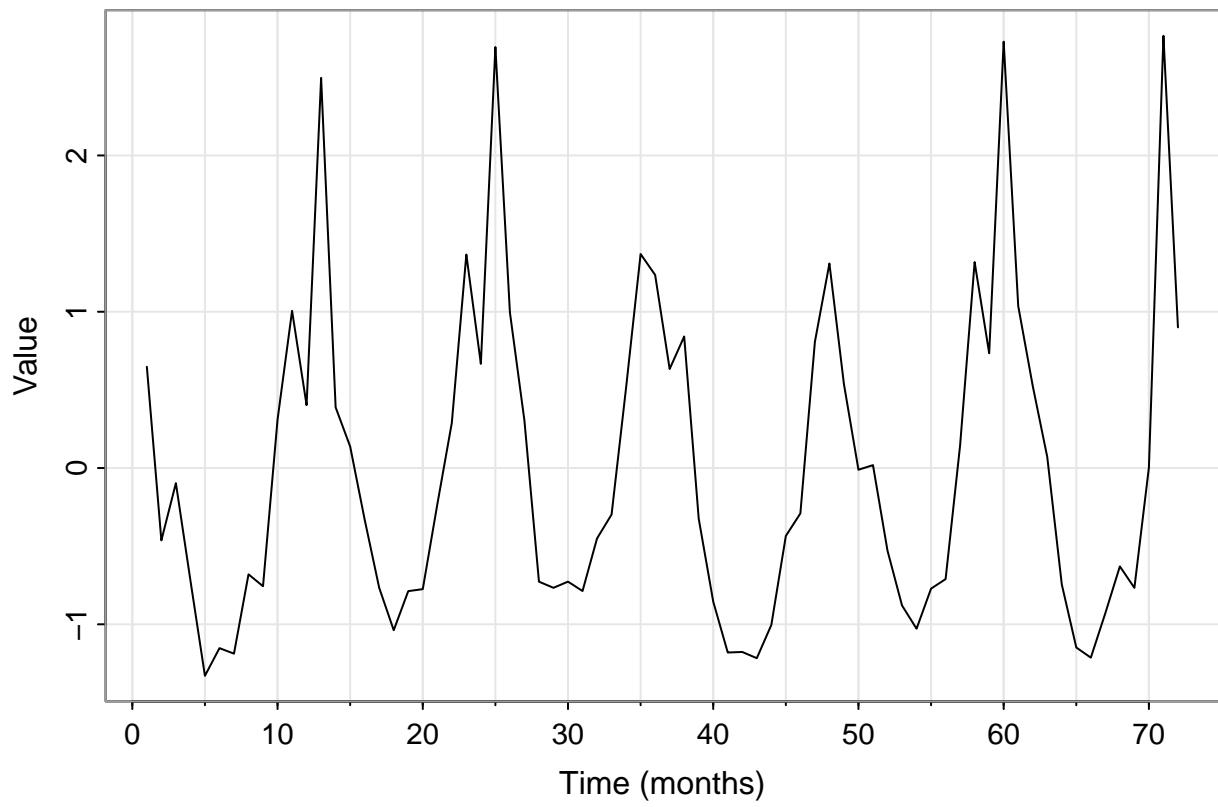
Pollutant 42101 – Cluster 3



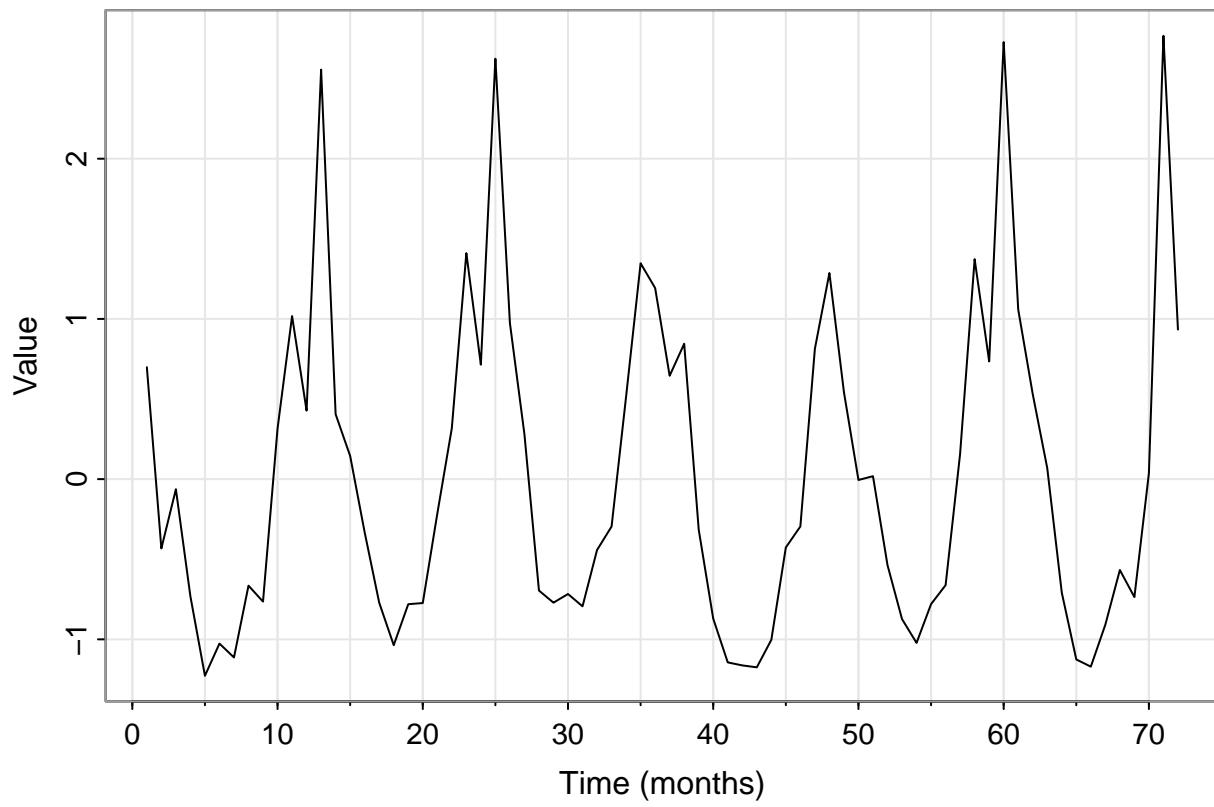
Pollutant 42101 – Cluster 4



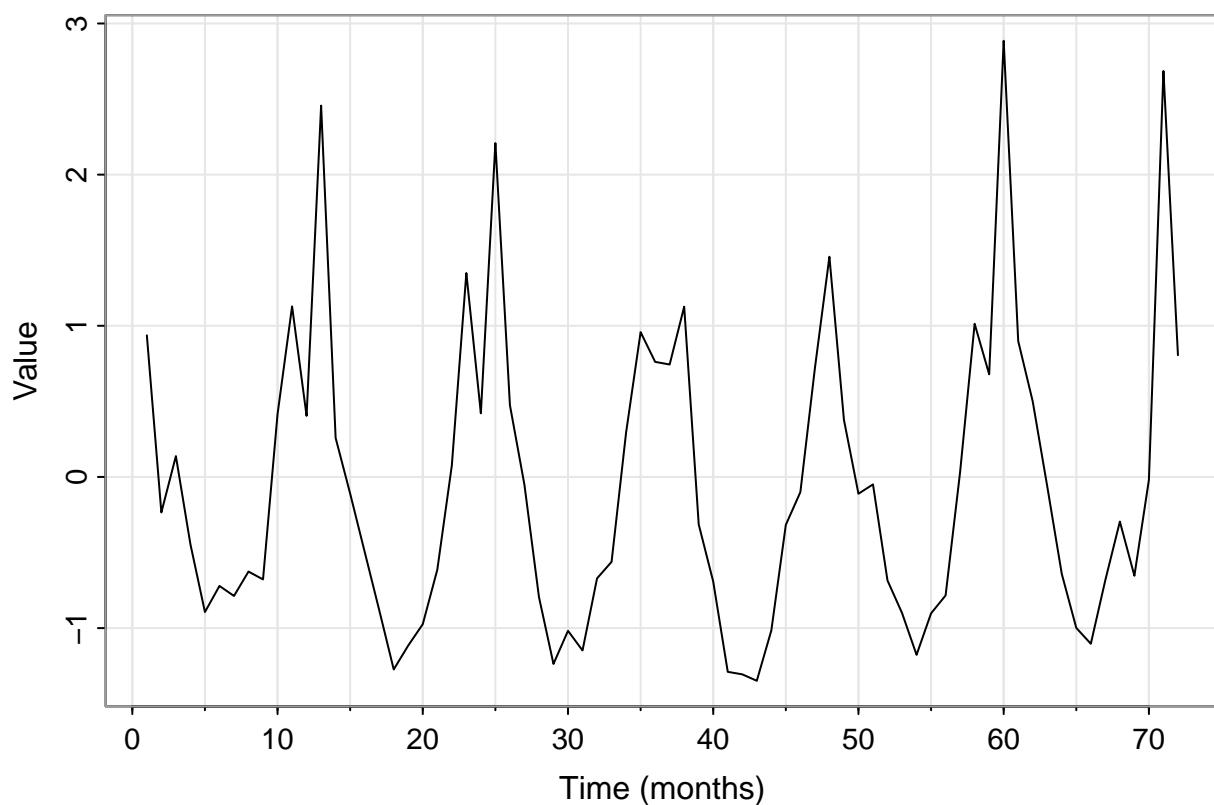
Pollutant 42101 – Cluster 5



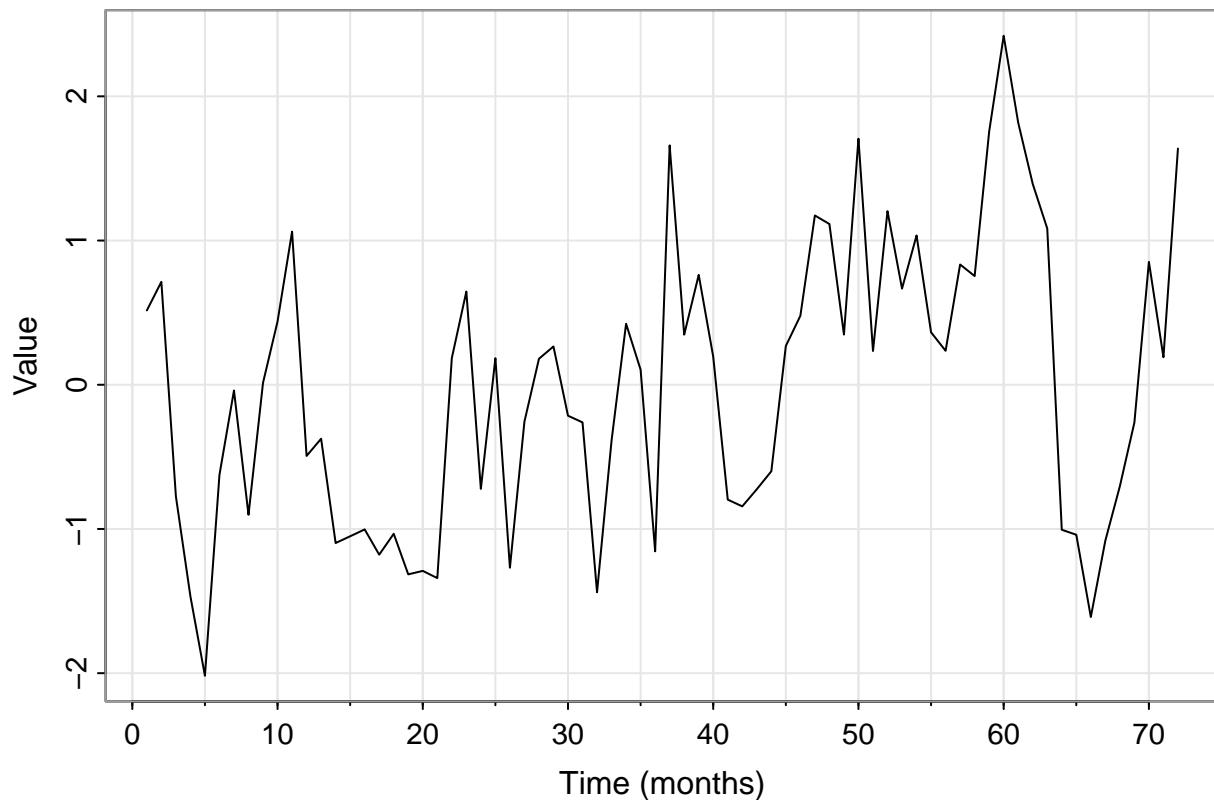
Pollutant 42101 – Cluster 6



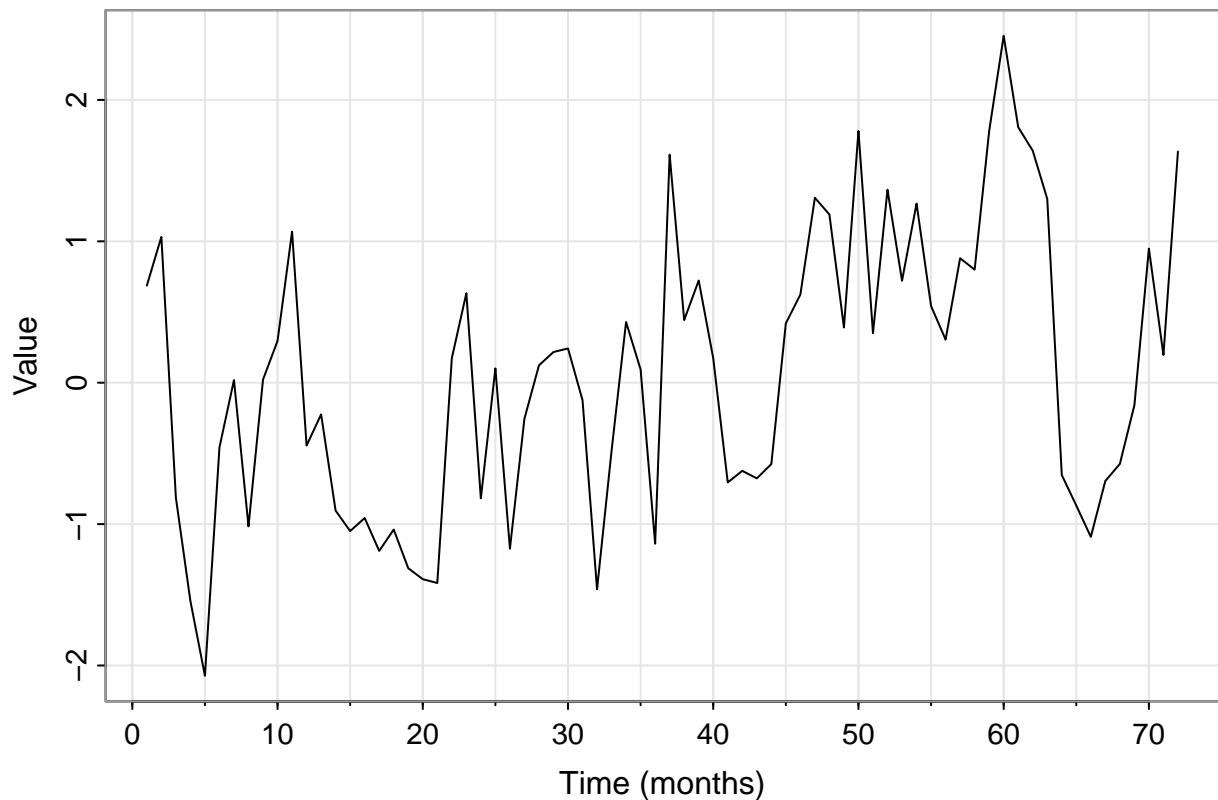
Pollutant 42101 – Cluster 7



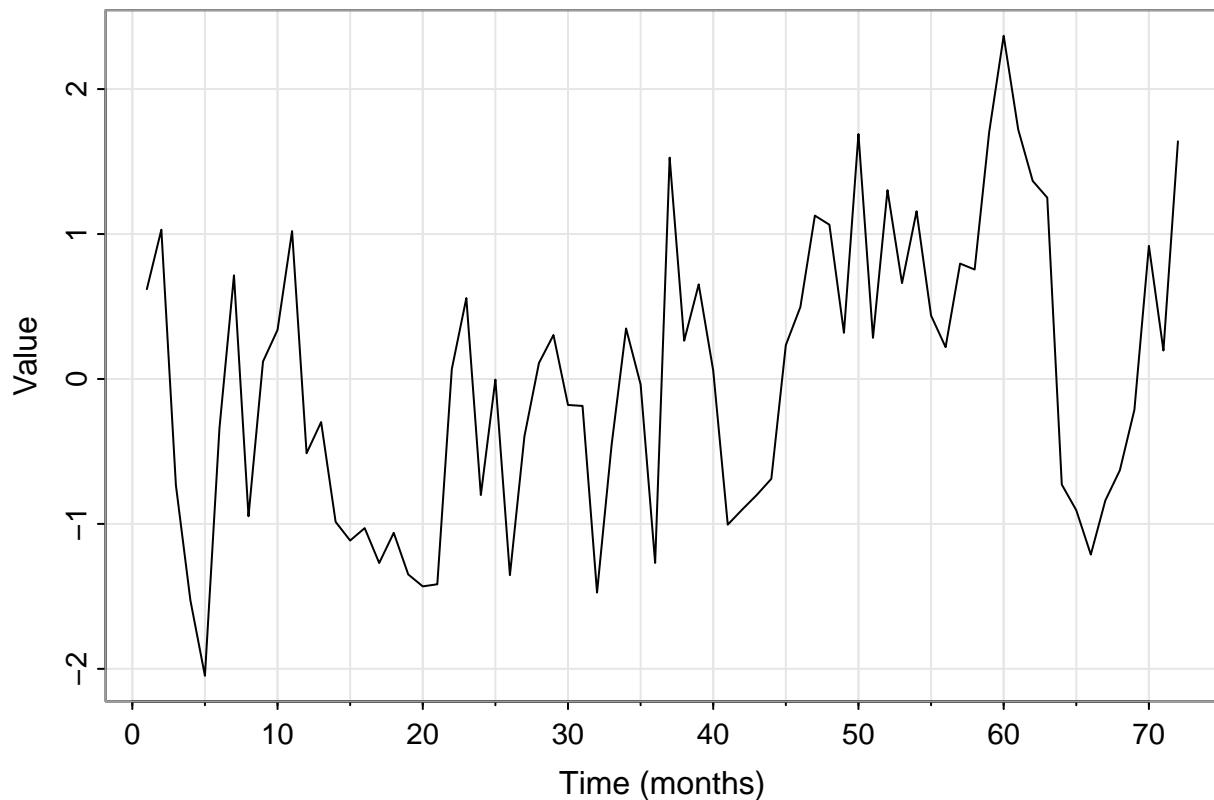
Pollutant 42401 – Cluster 1



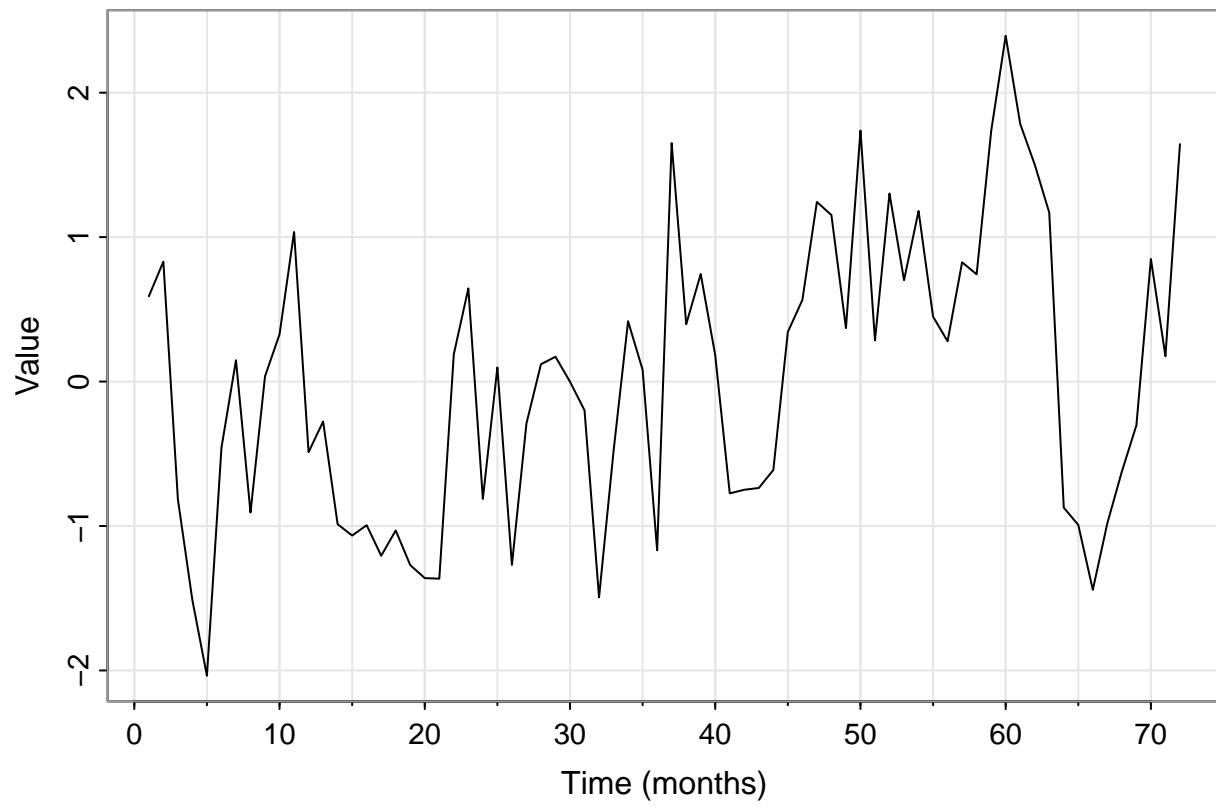
Pollutant 42401 – Cluster 2



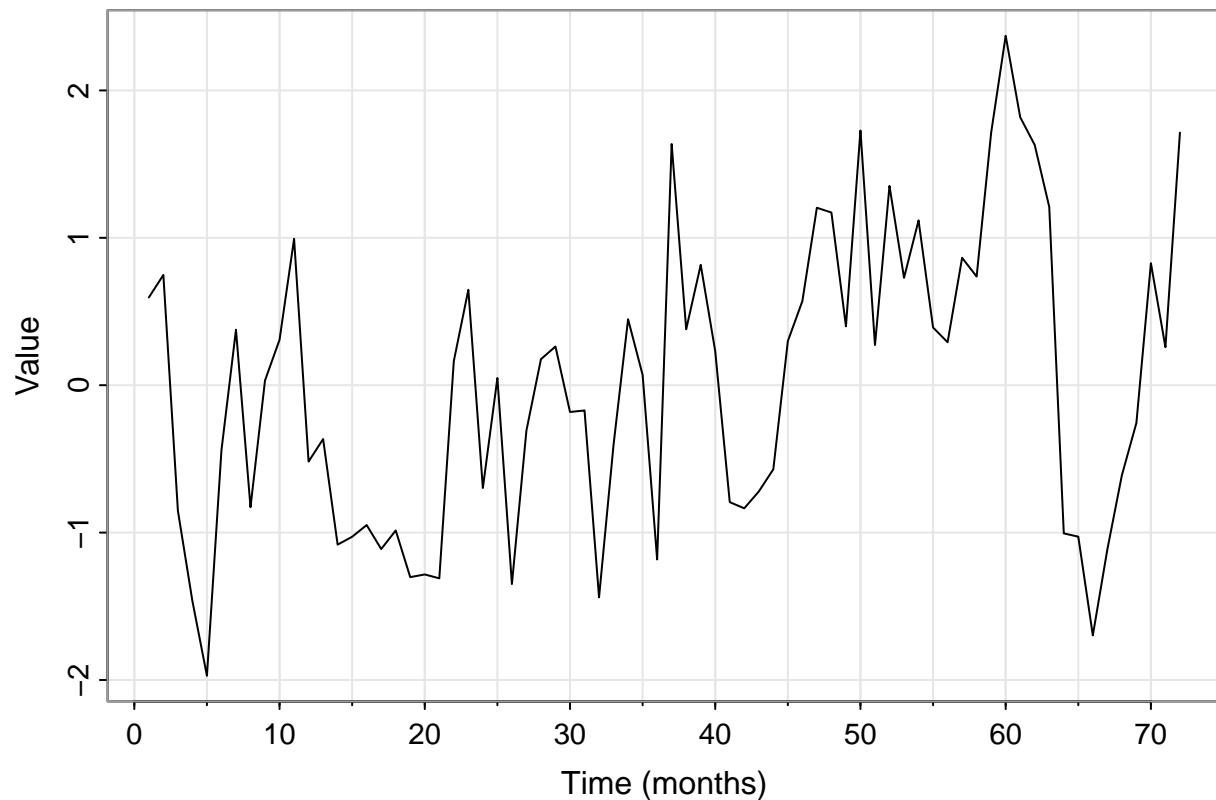
Pollutant 42401 – Cluster 3



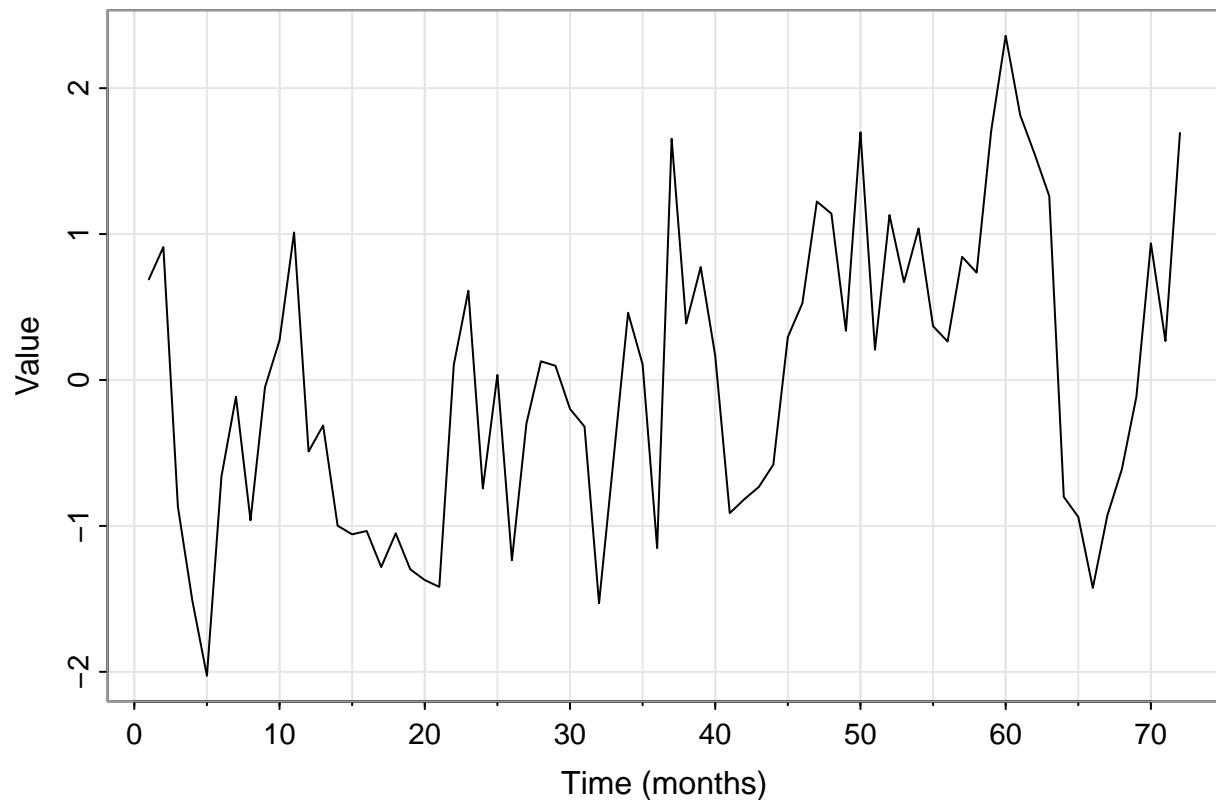
Pollutant 42401 – Cluster 4



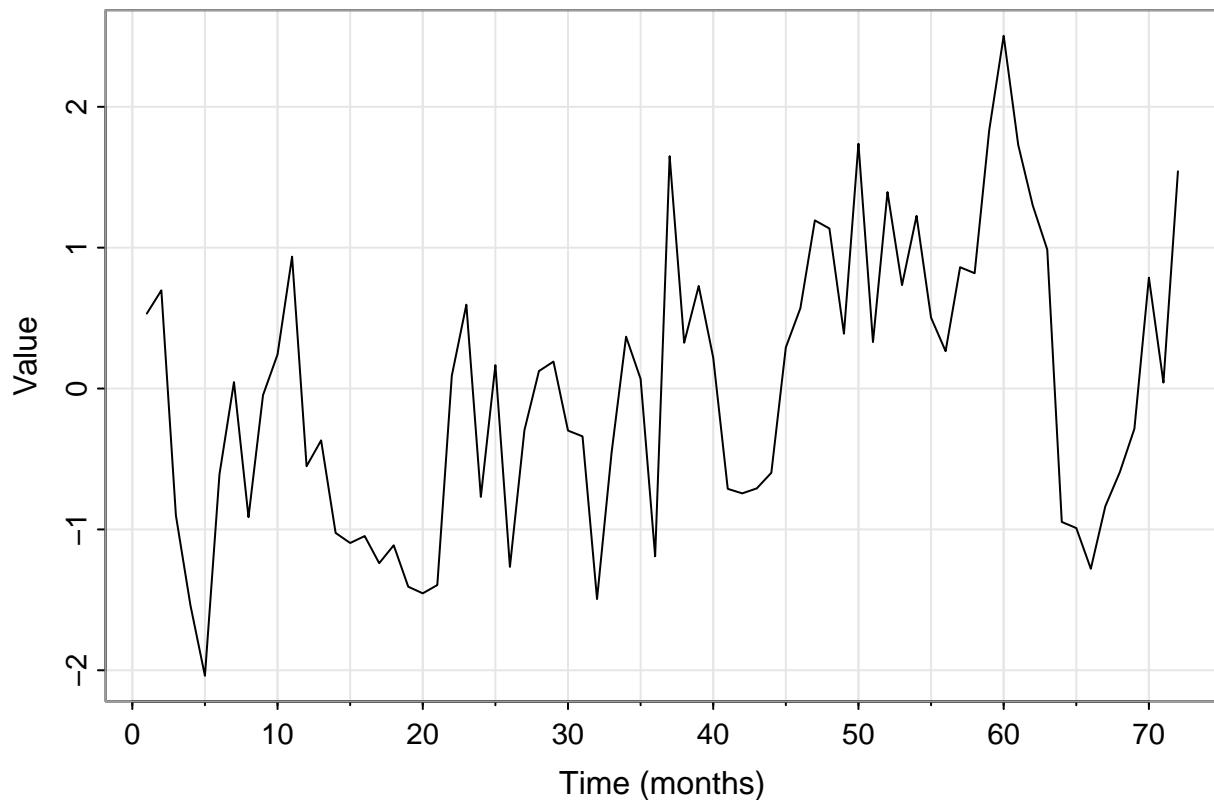
Pollutant 42401 – Cluster 5



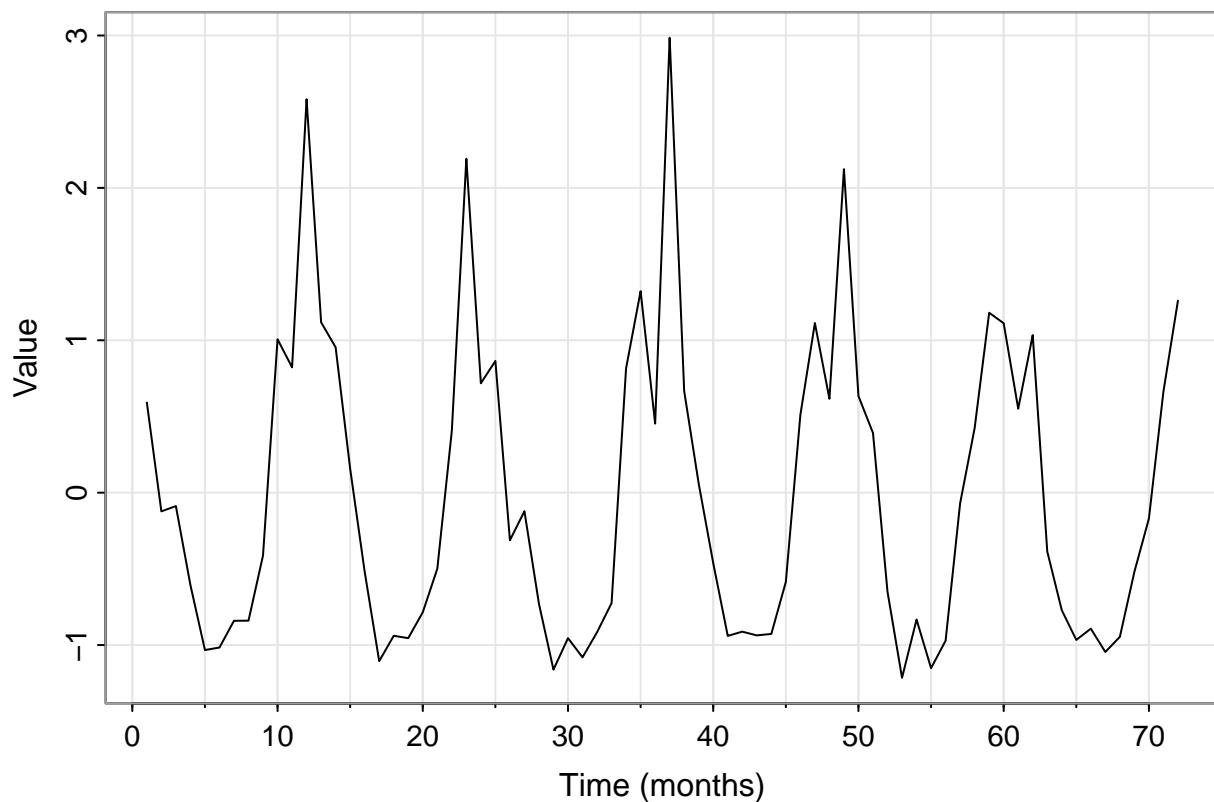
Pollutant 42401 – Cluster 6



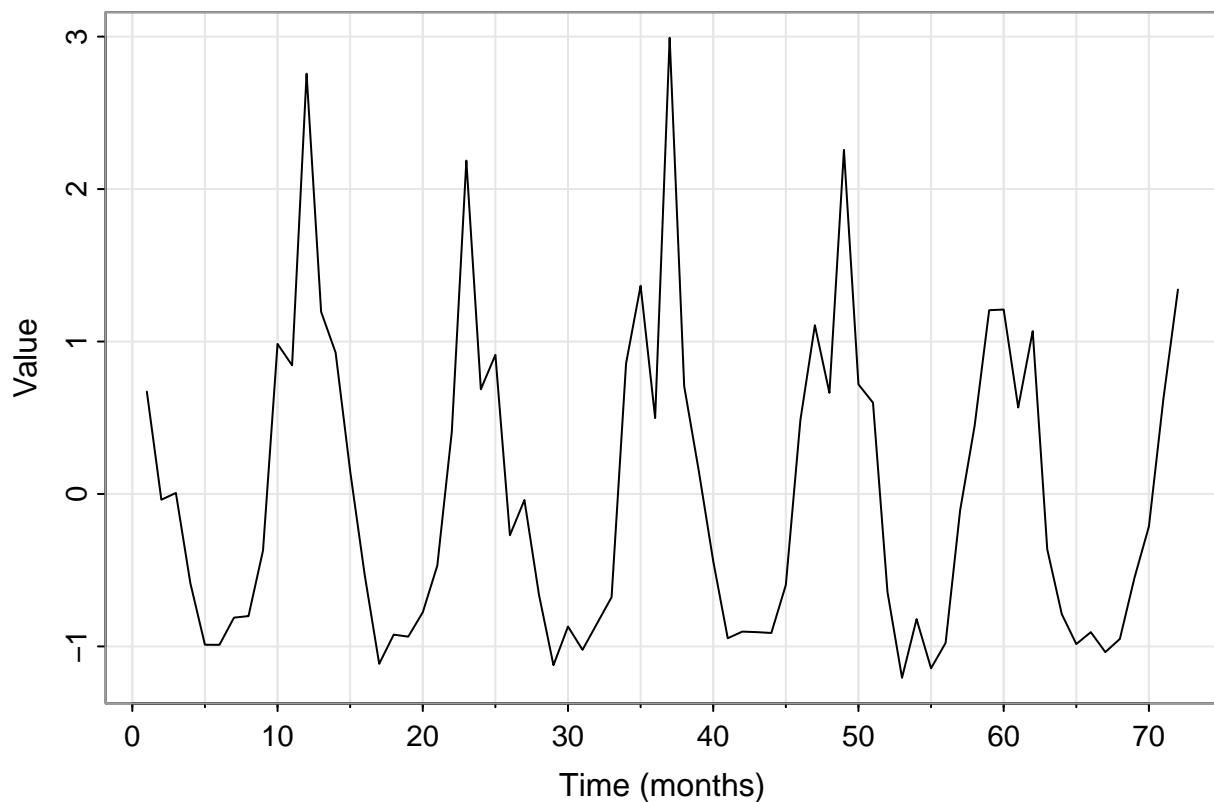
Pollutant 42401 – Cluster 7



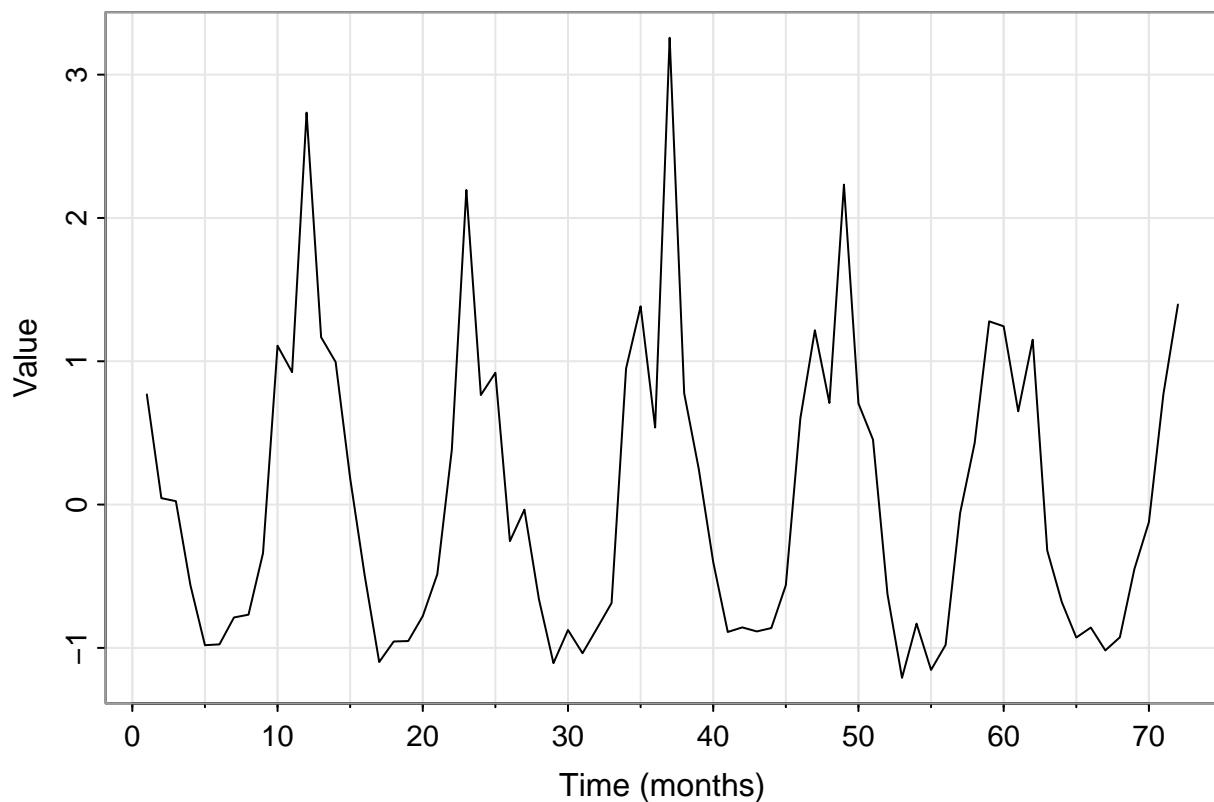
Pollutant 42602 – Cluster 1



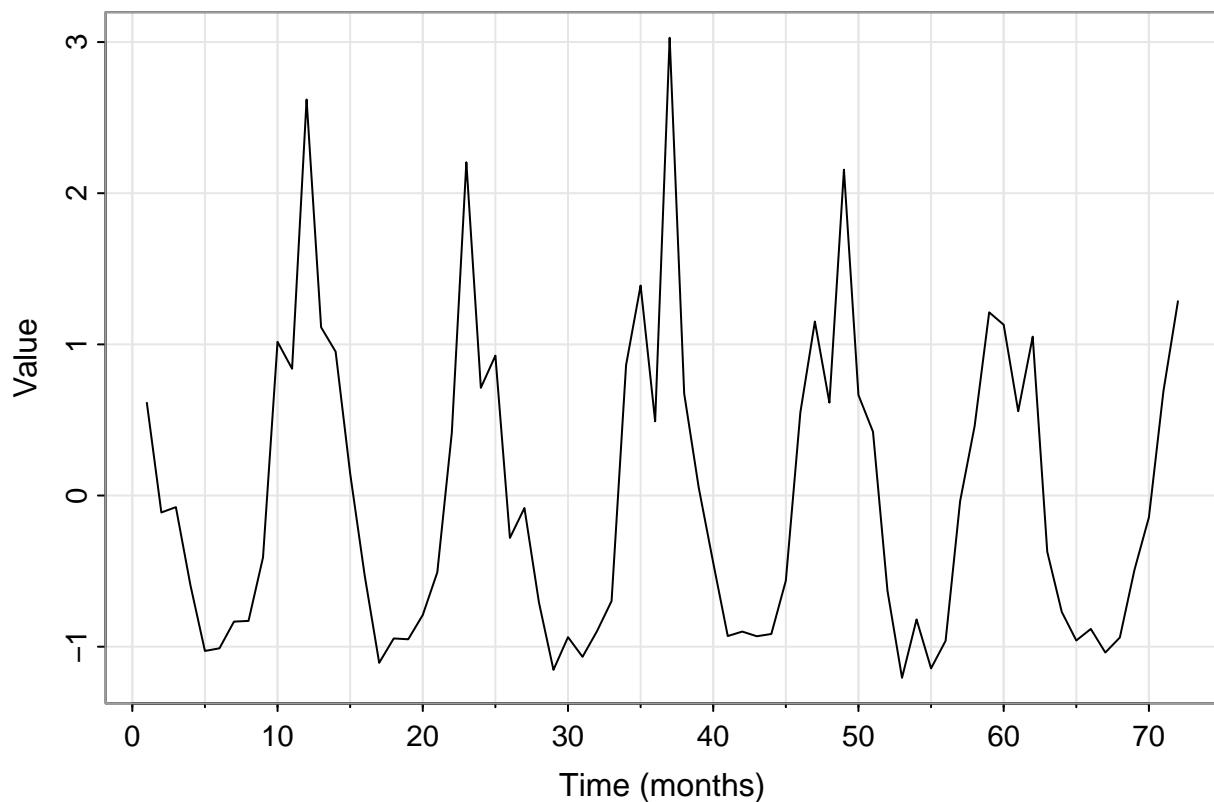
Pollutant 42602 – Cluster 2



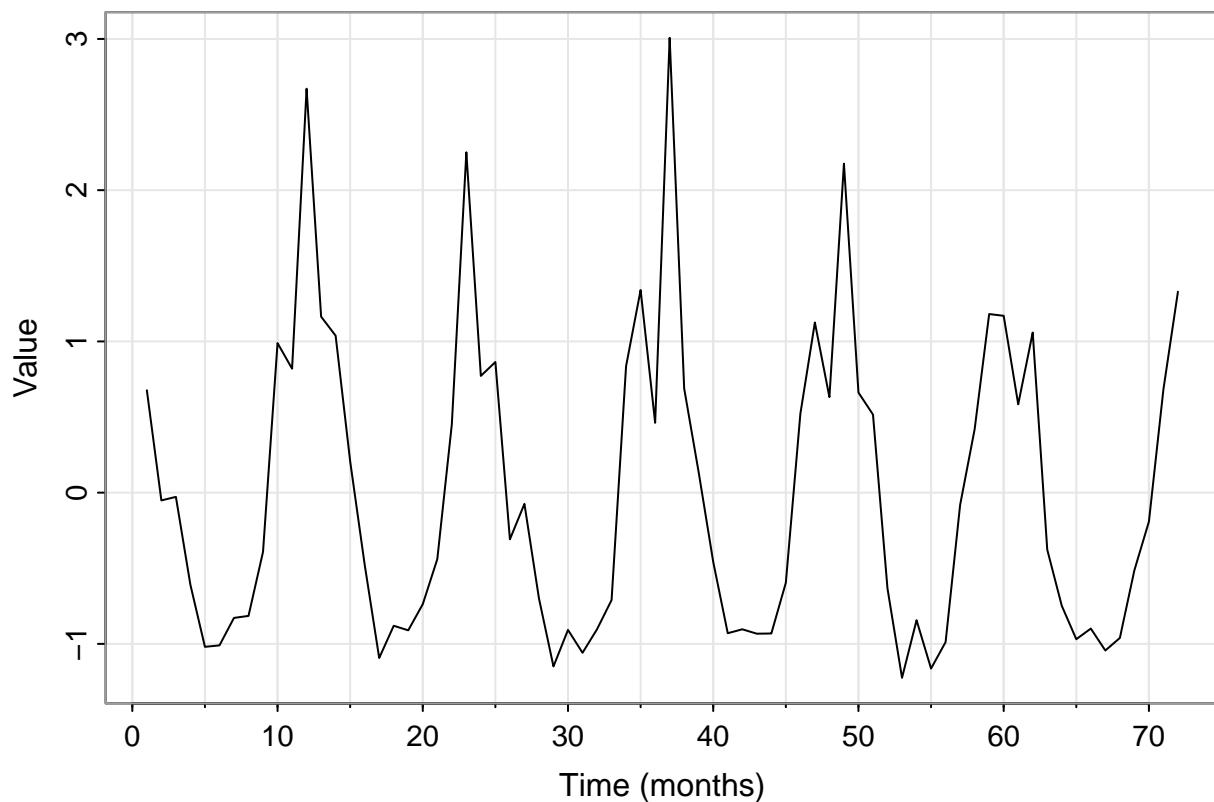
Pollutant 42602 – Cluster 3



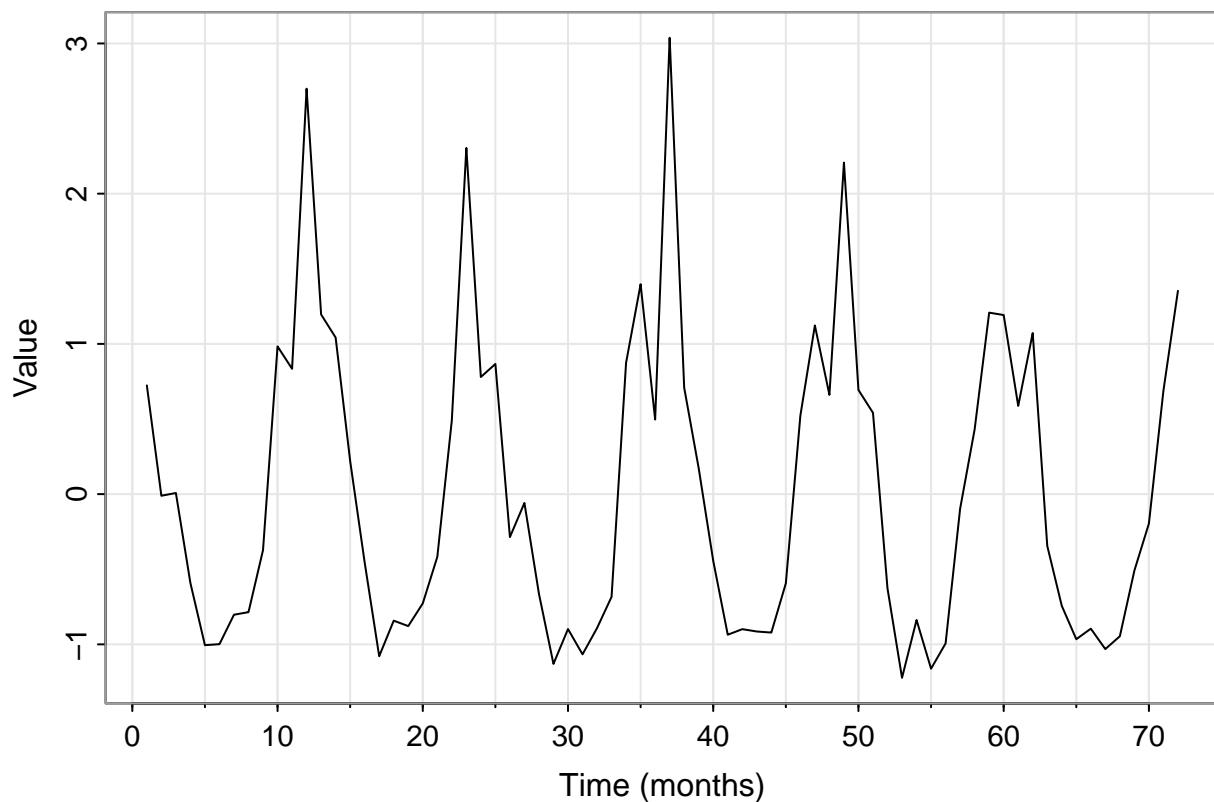
Pollutant 42602 – Cluster 4



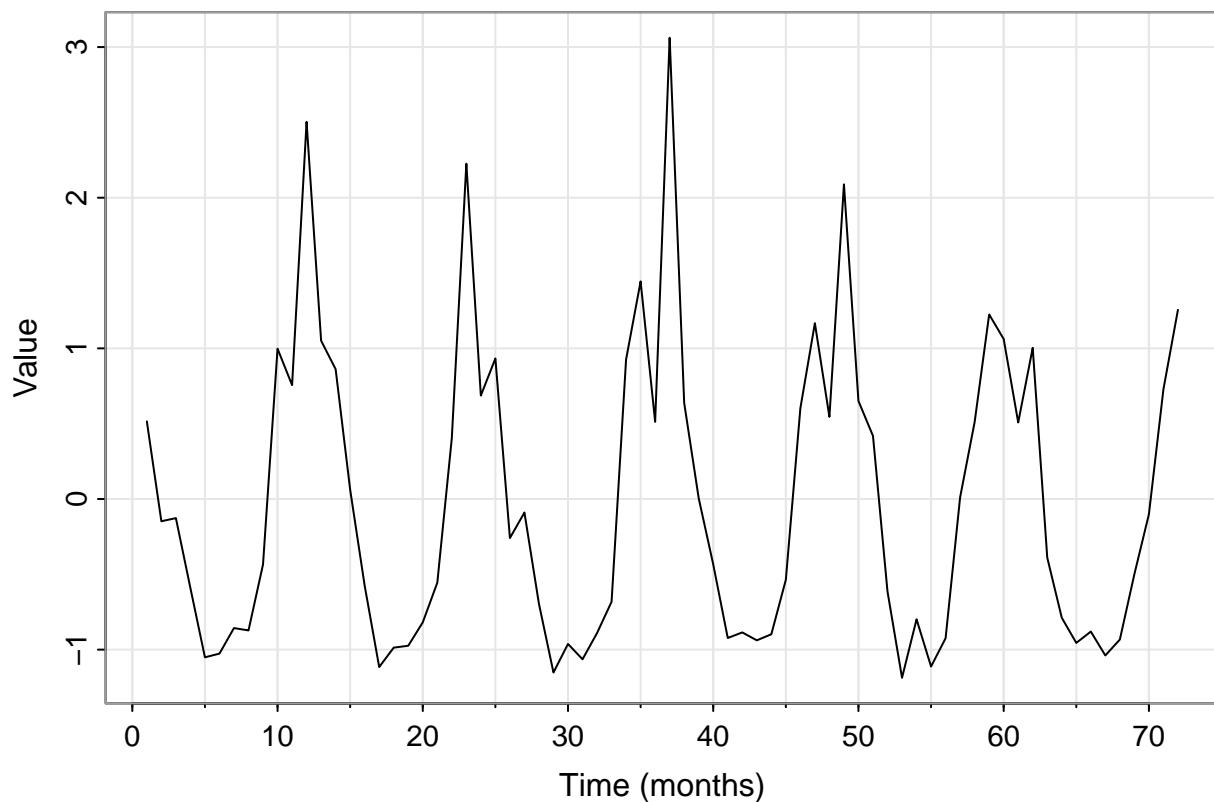
Pollutant 42602 – Cluster 5



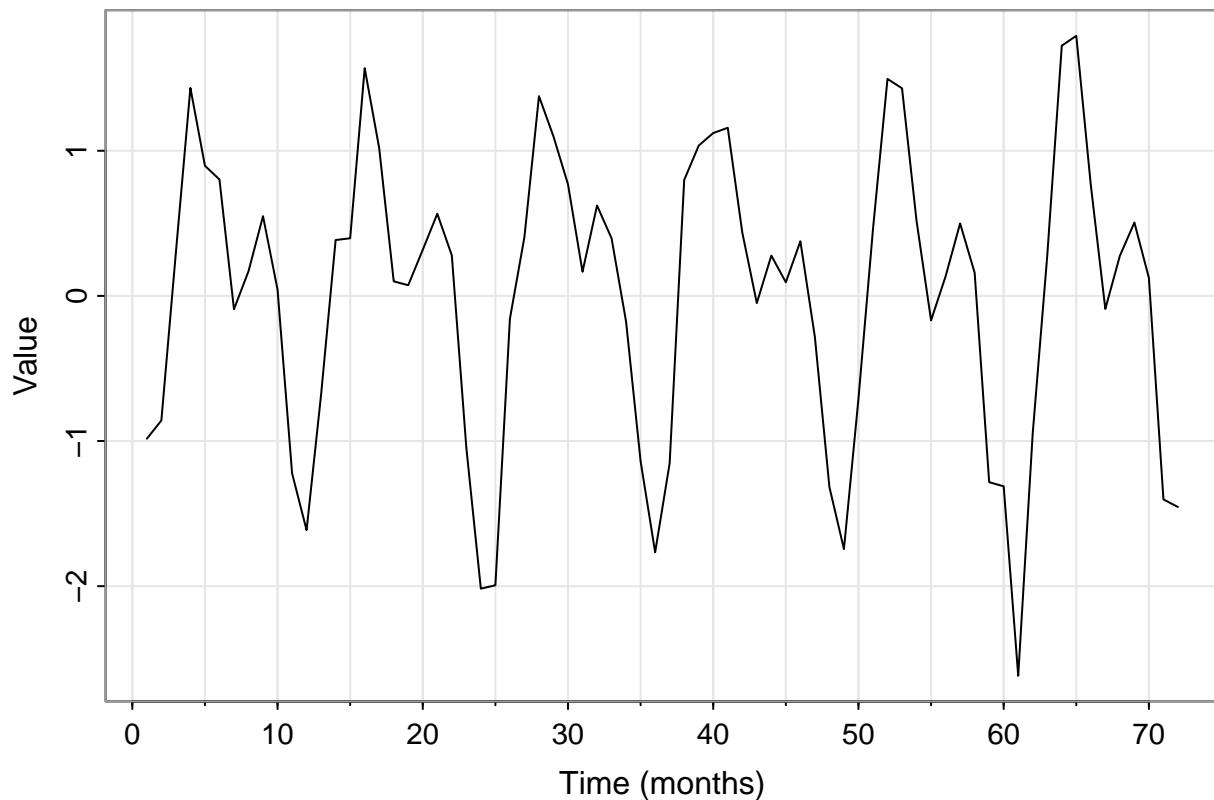
Pollutant 42602 – Cluster 6



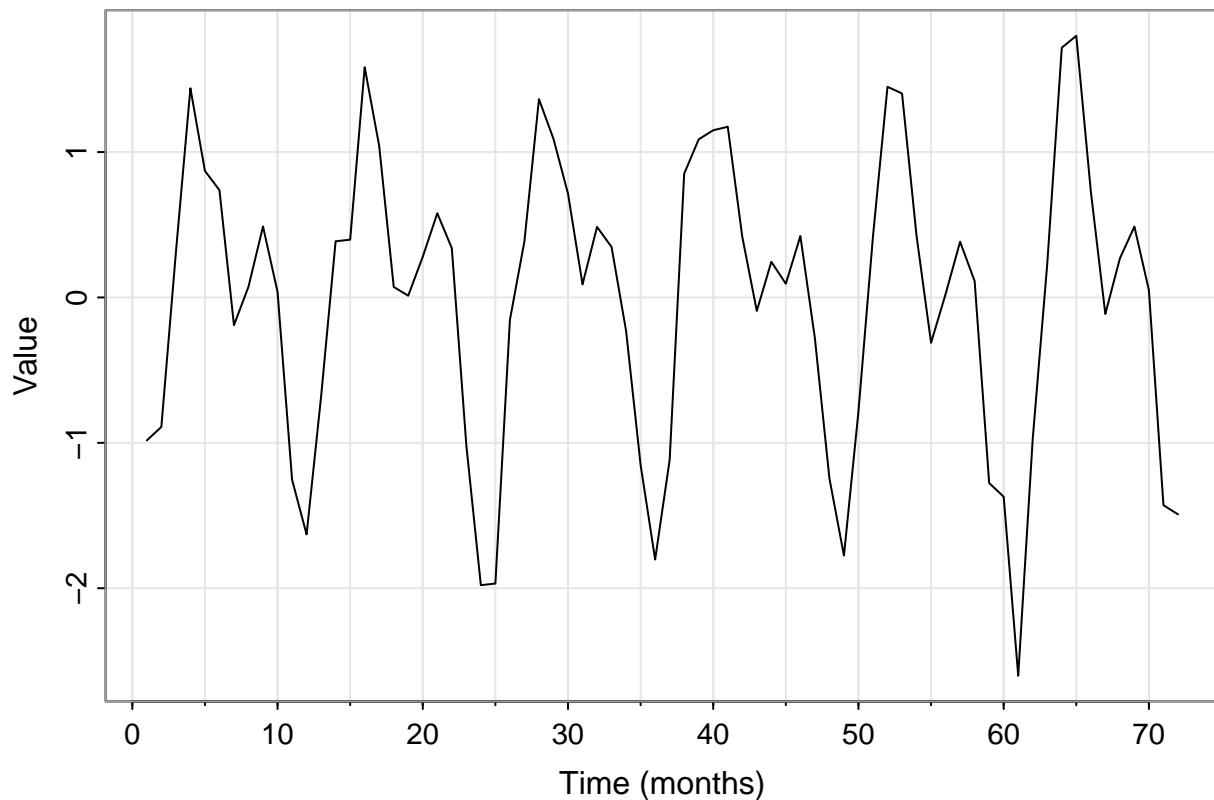
Pollutant 42602 – Cluster 7



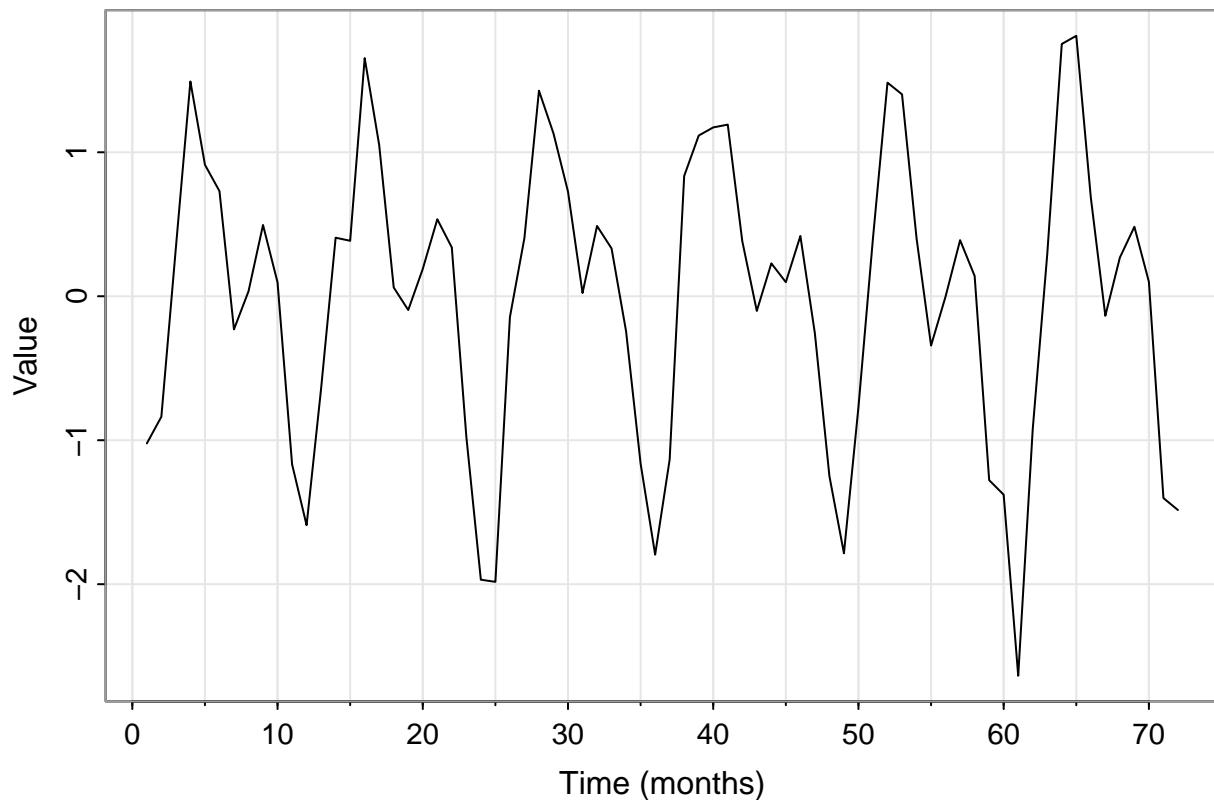
Pollutant 44201 – Cluster 1



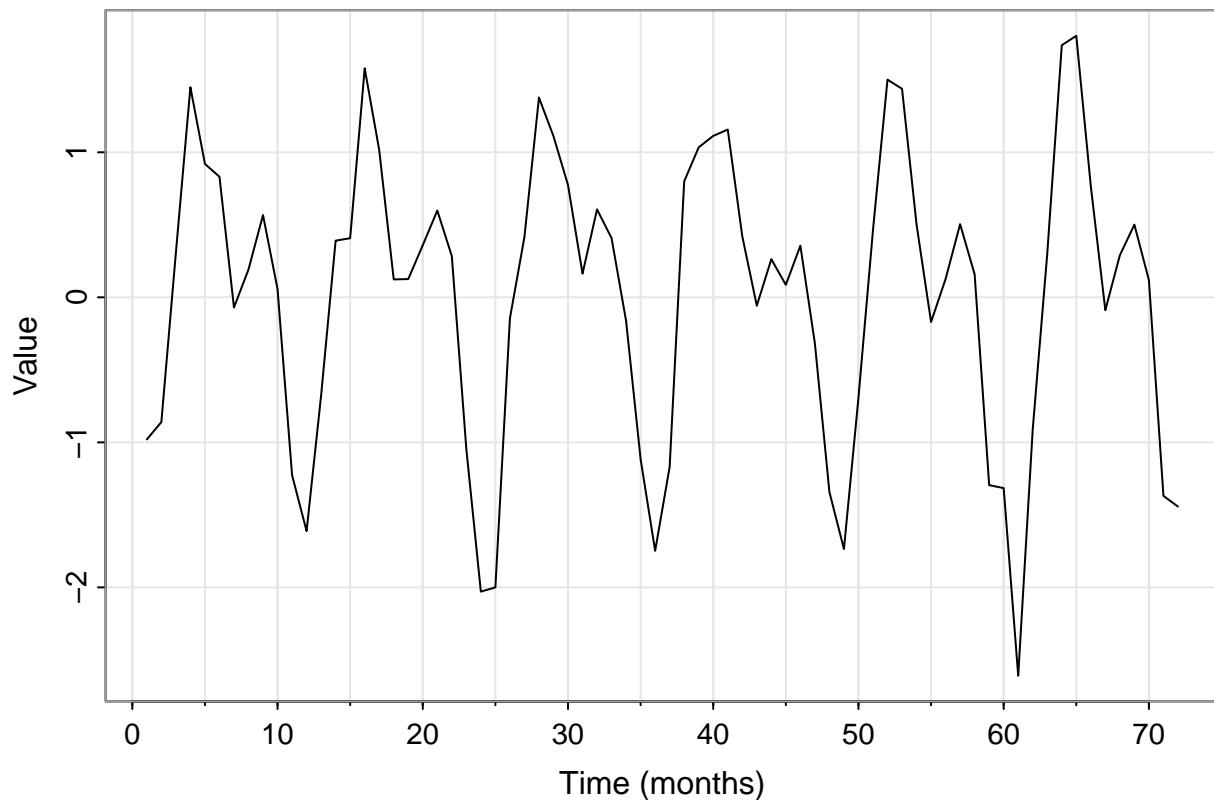
Pollutant 44201 – Cluster 2



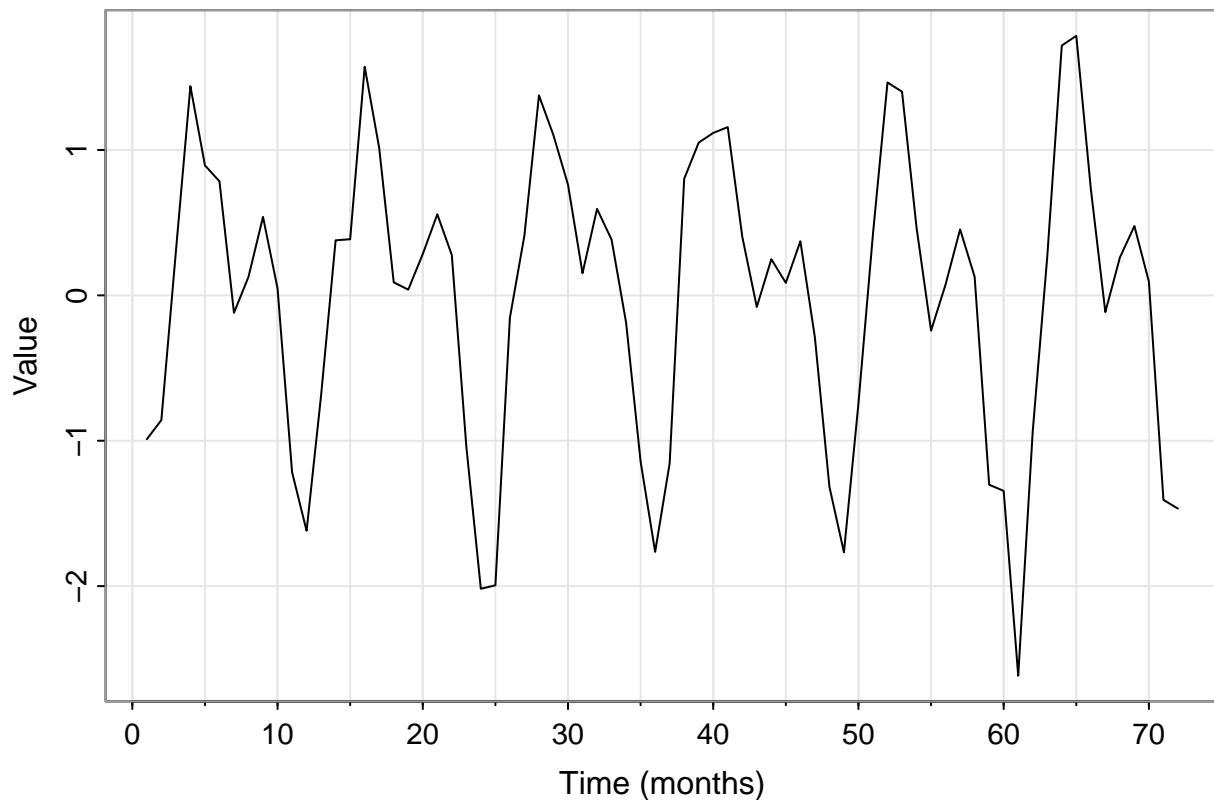
Pollutant 44201 – Cluster 3



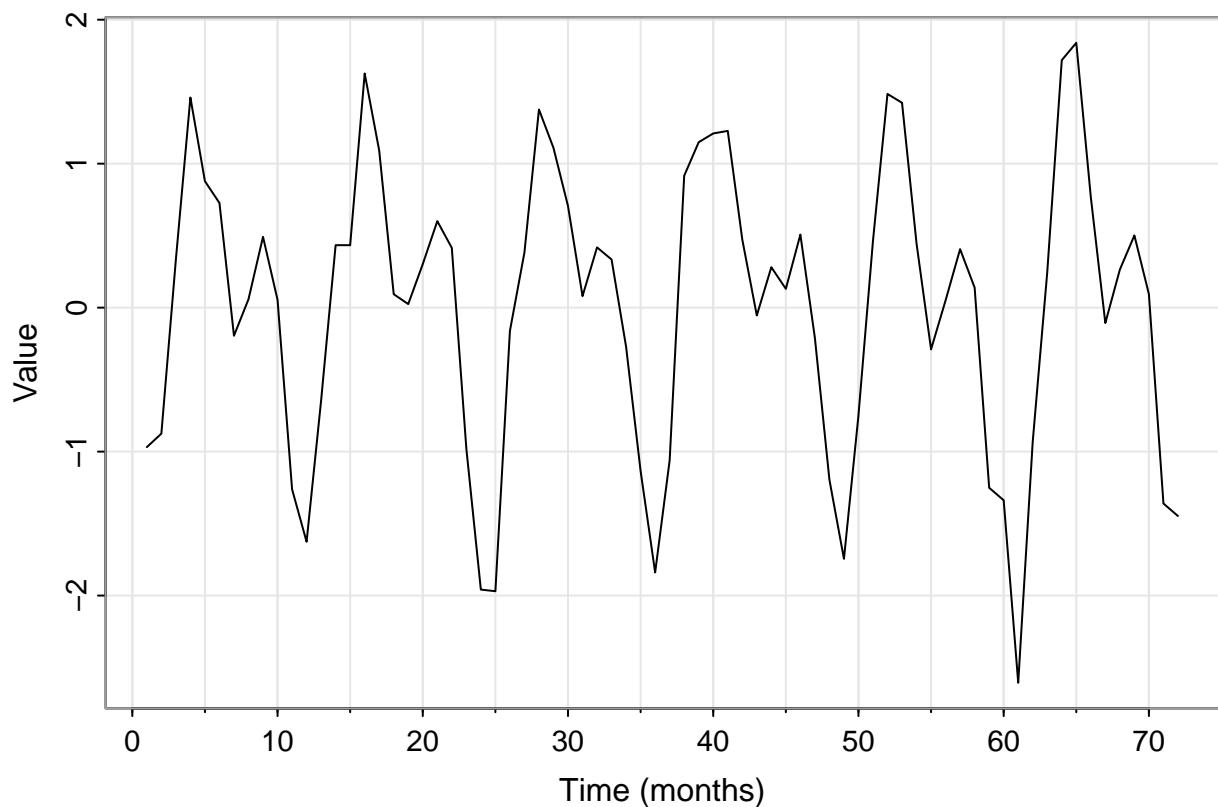
Pollutant 44201 – Cluster 4



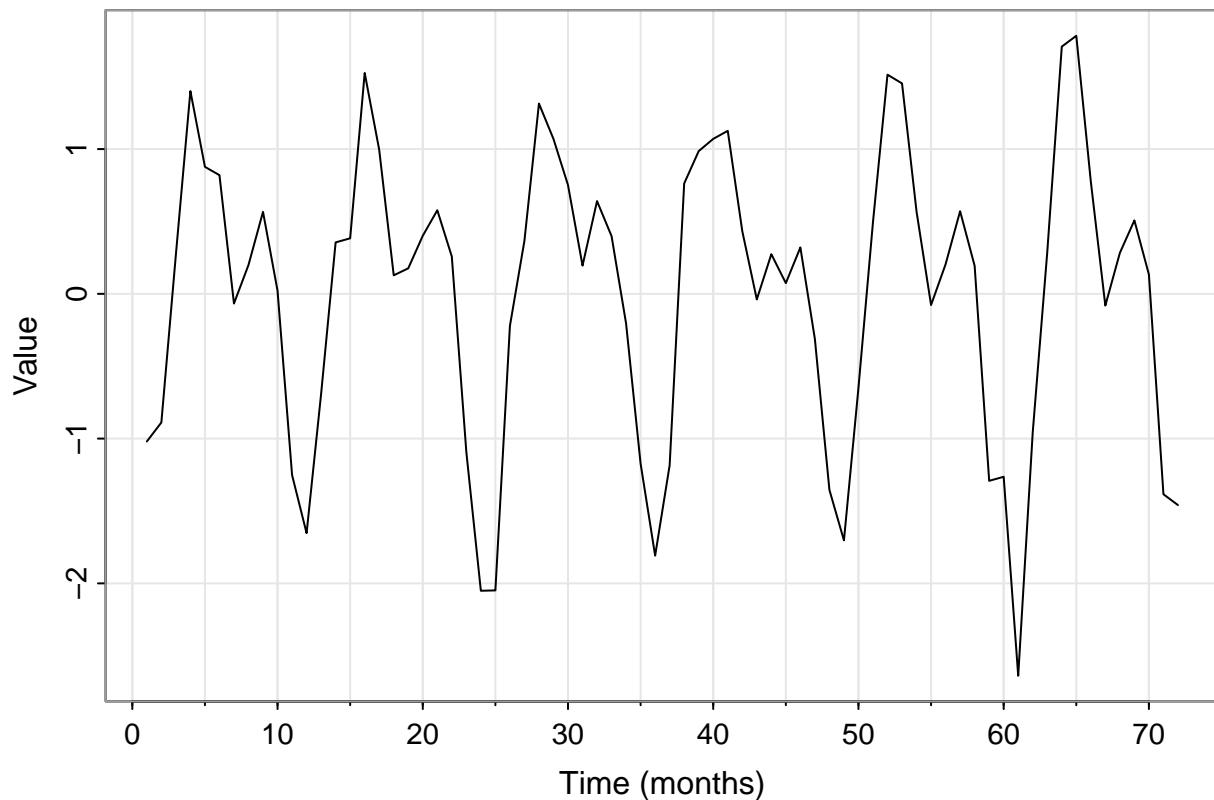
Pollutant 44201 – Cluster 5



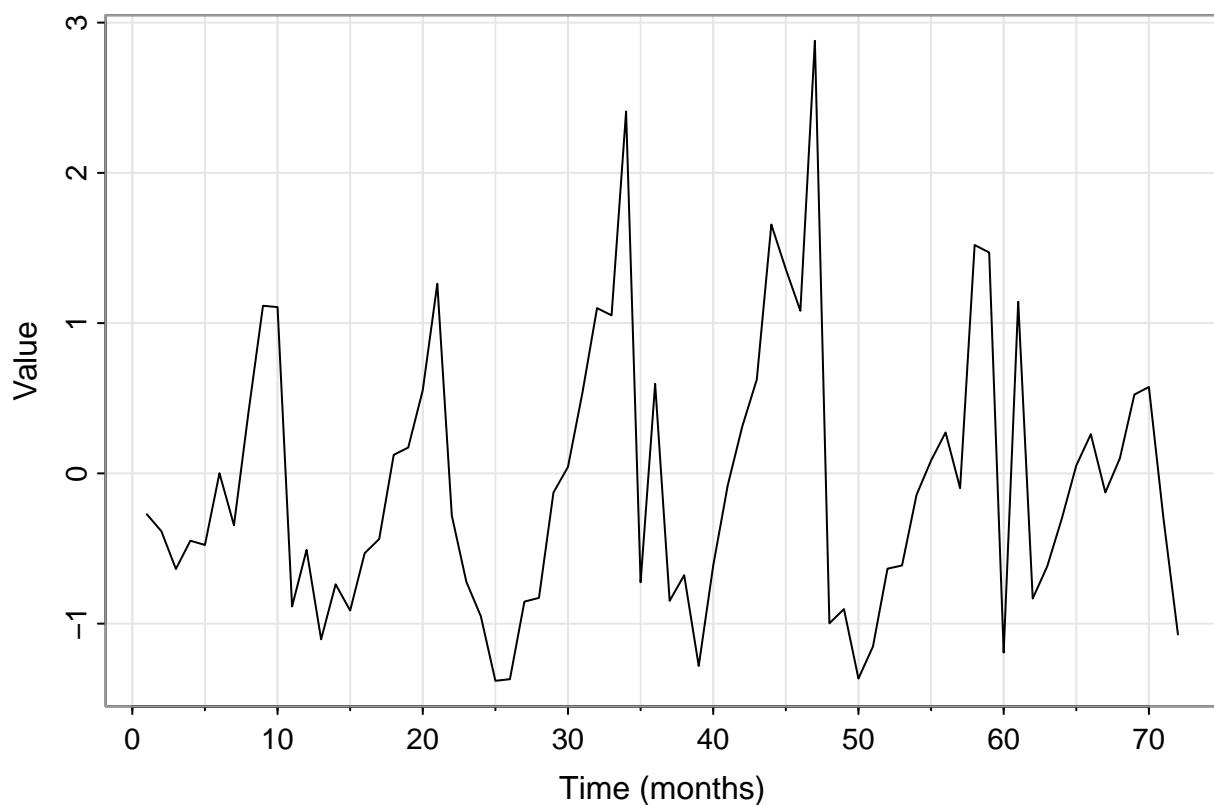
Pollutant 44201 – Cluster 6



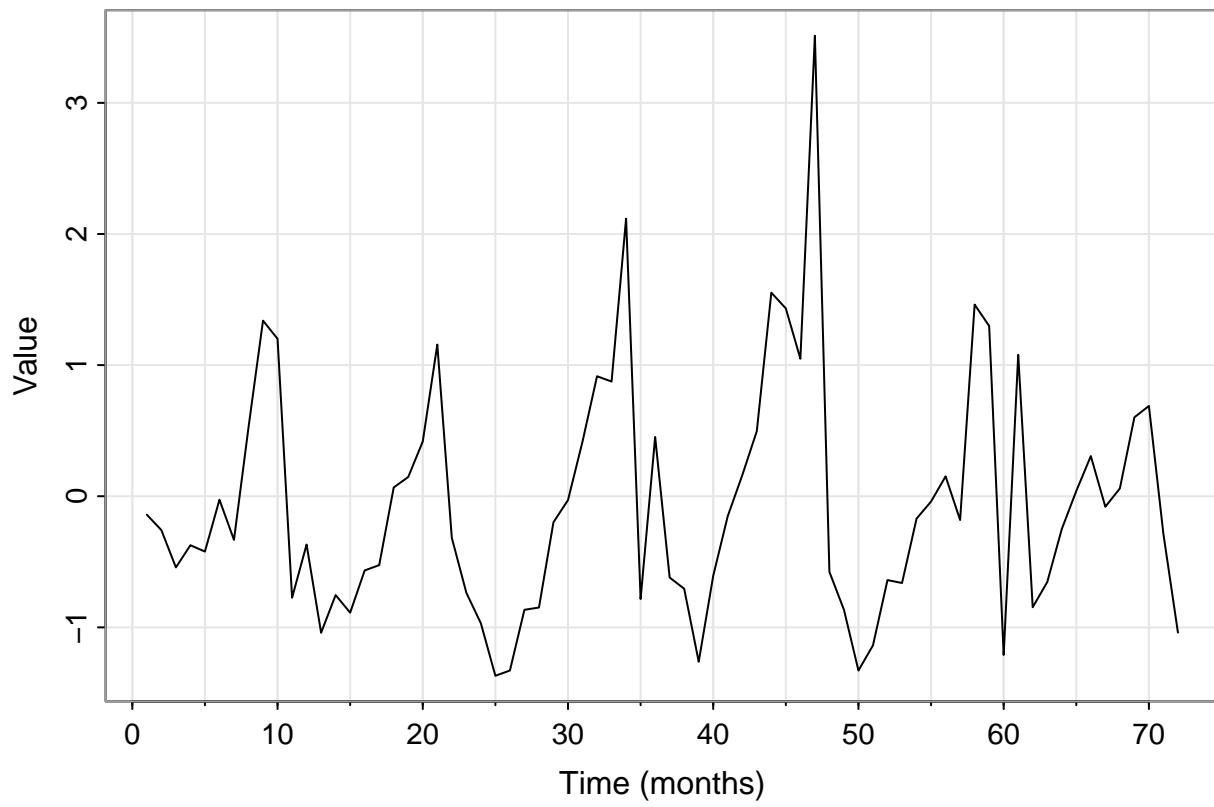
Pollutant 44201 – Cluster 7



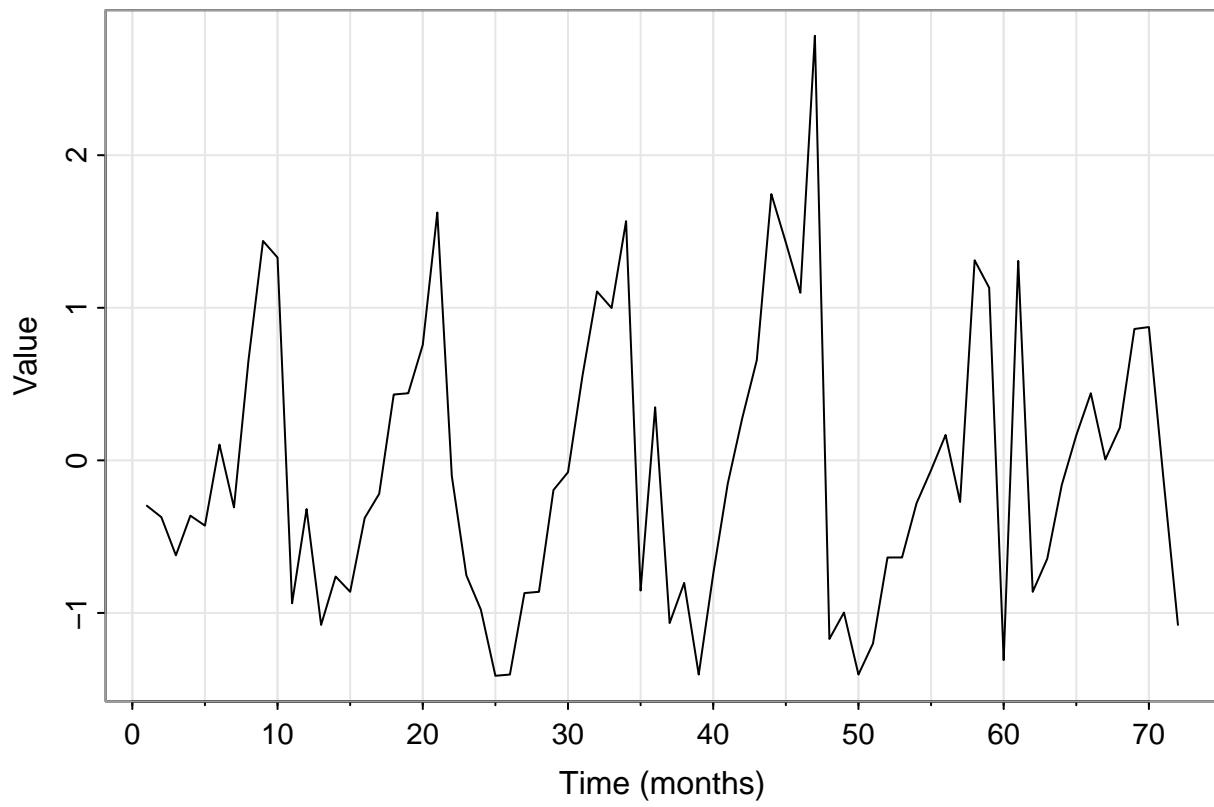
Pollutant 81102 – Cluster 1



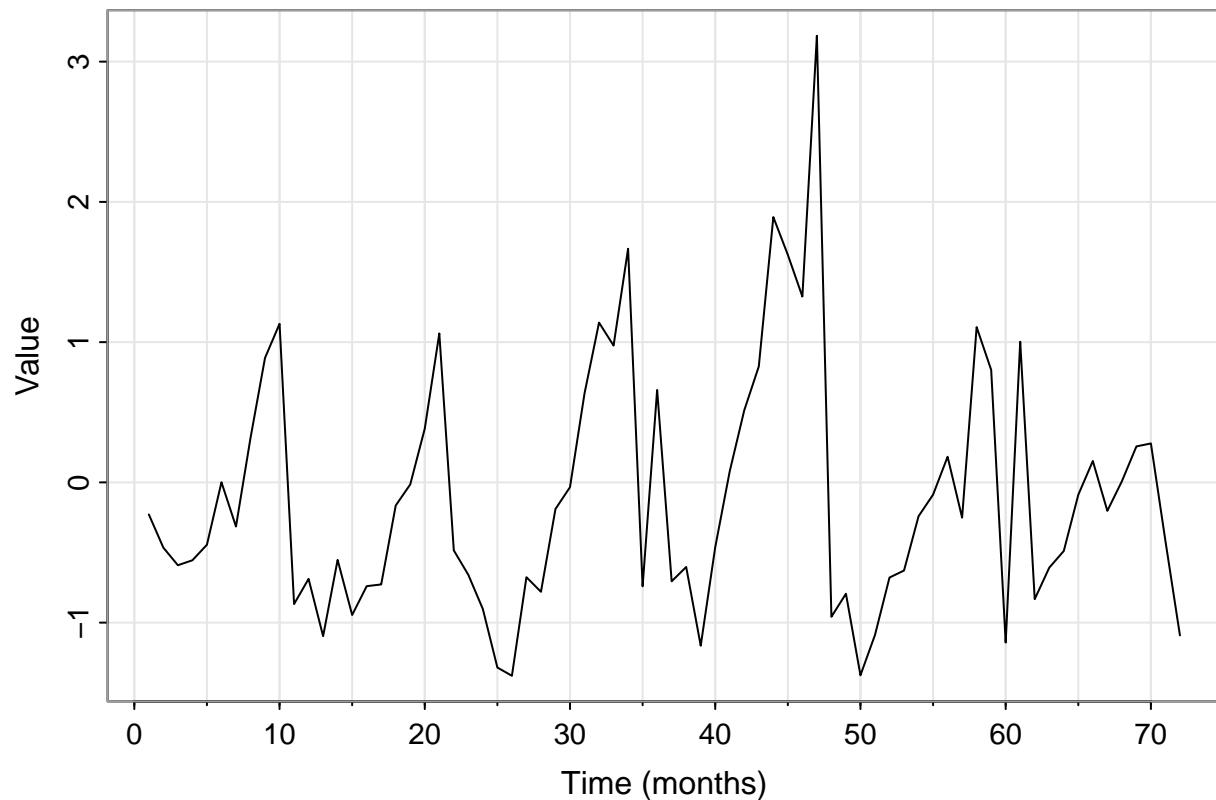
Pollutant 81102 – Cluster 2



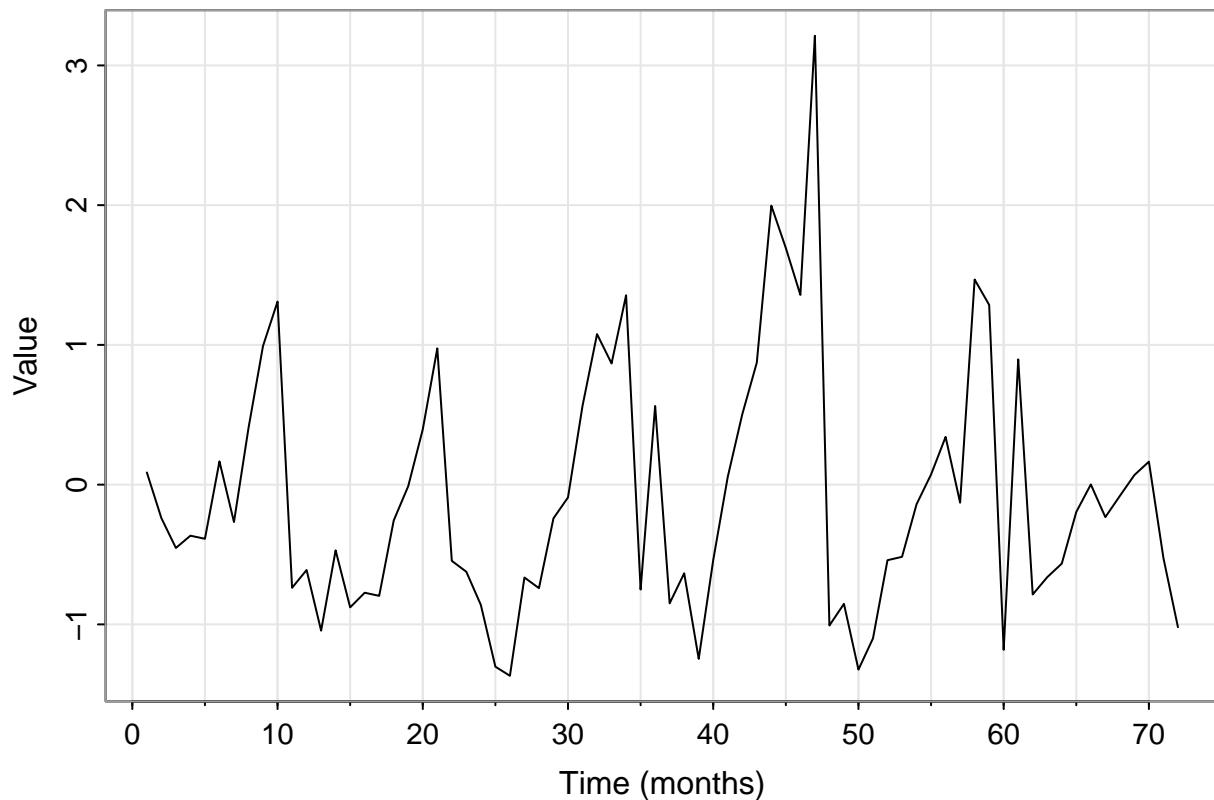
Pollutant 81102 – Cluster 3



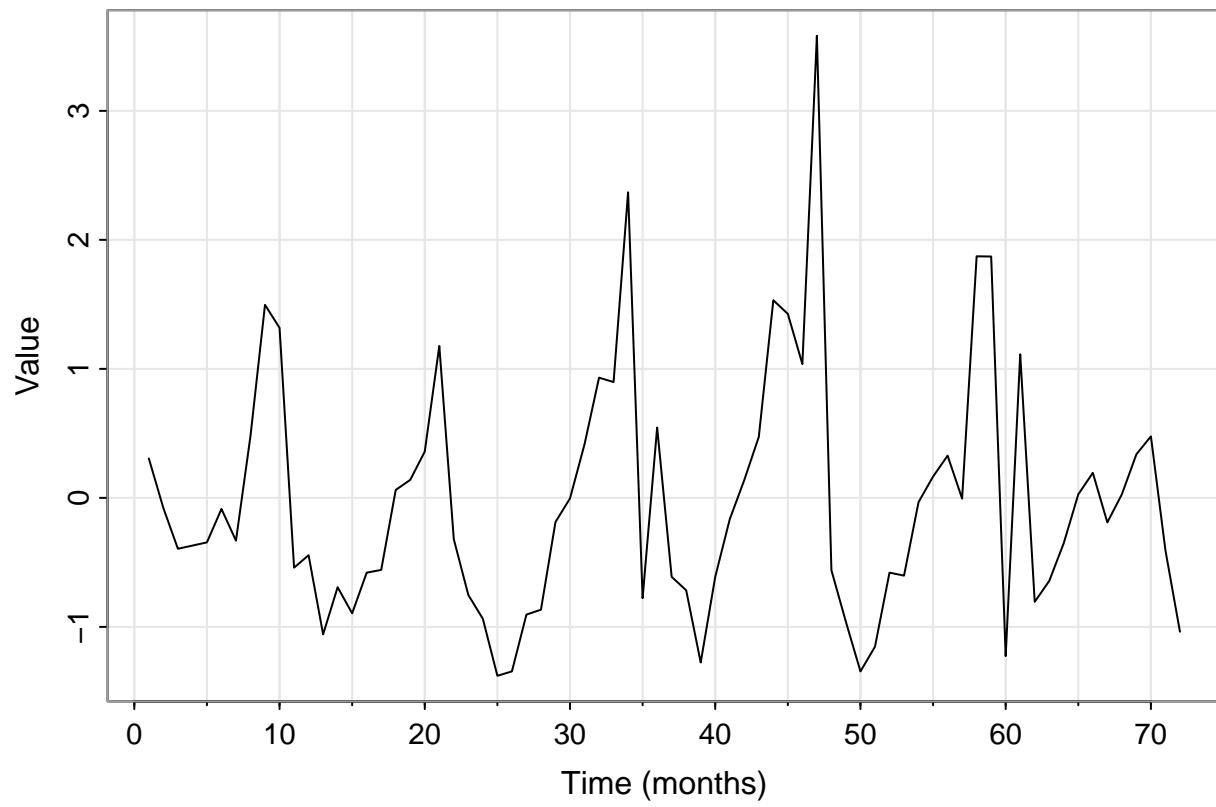
Pollutant 81102 – Cluster 4



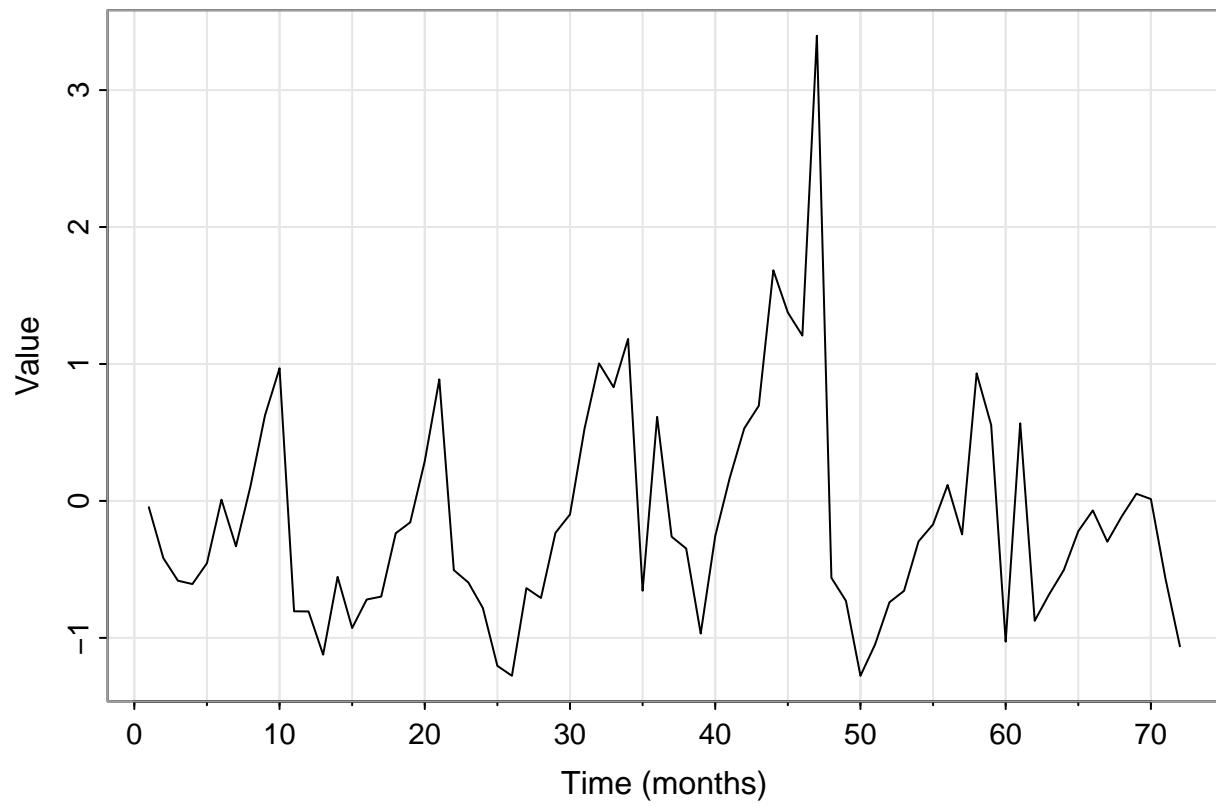
Pollutant 81102 – Cluster 5



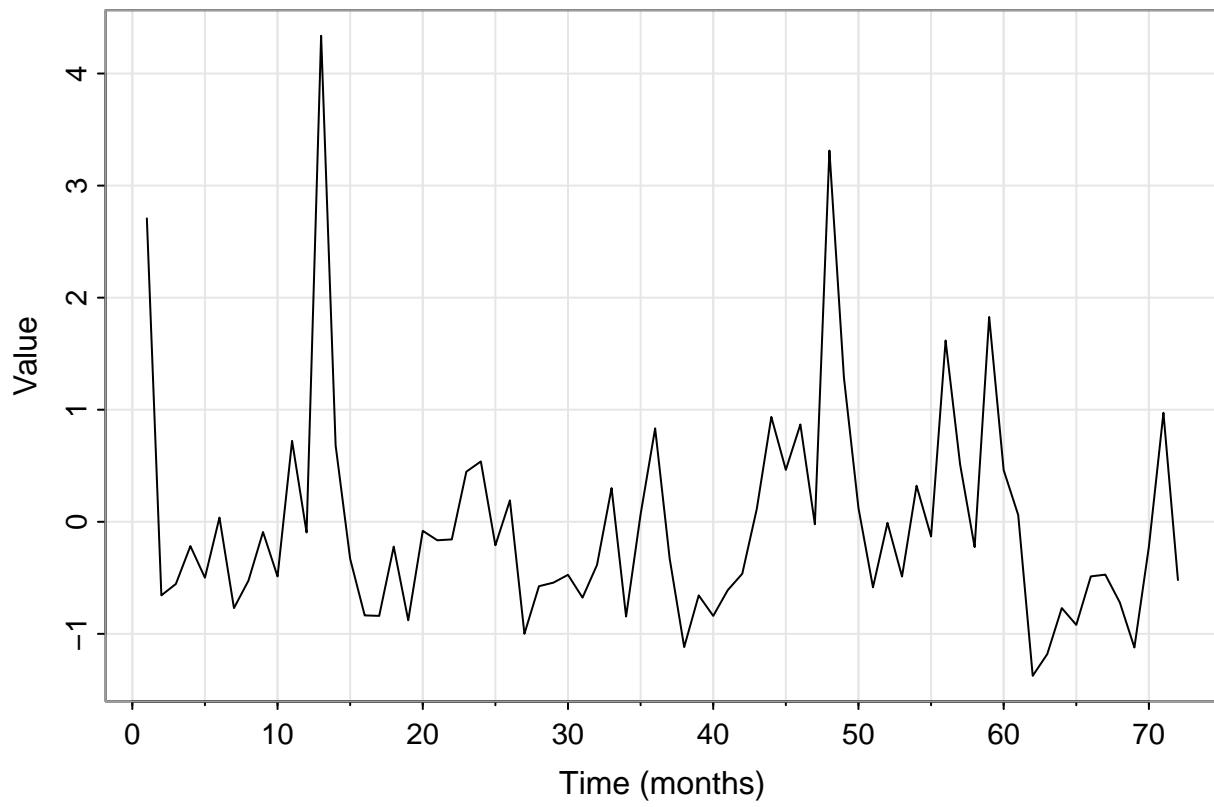
Pollutant 81102 – Cluster 6



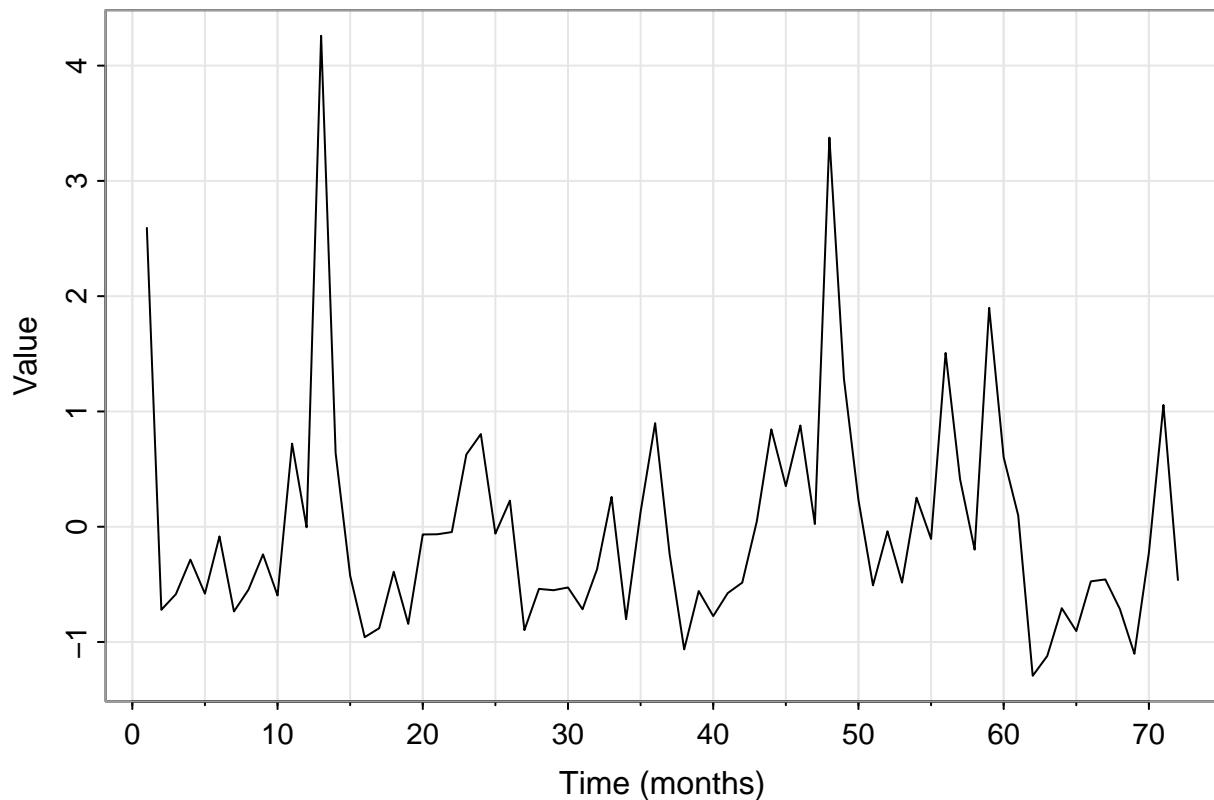
Pollutant 81102 – Cluster 7



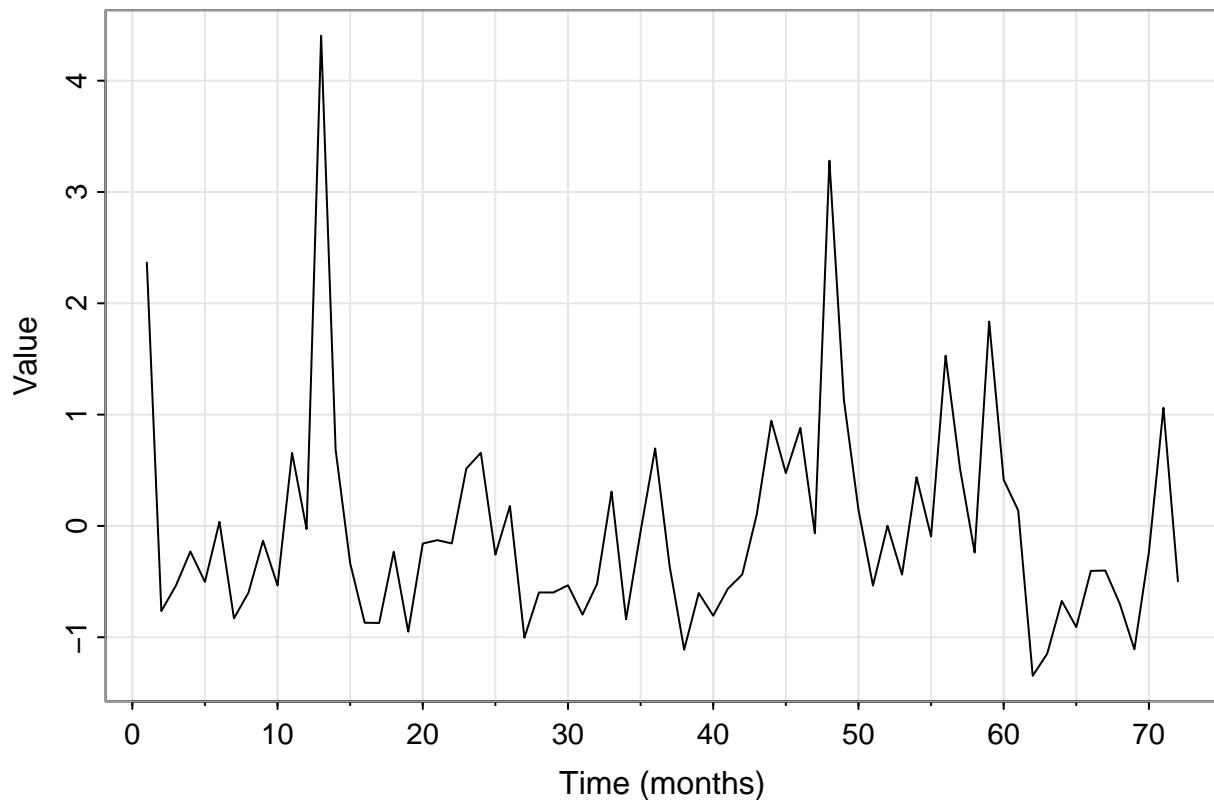
Pollutant 88101 – Cluster 1



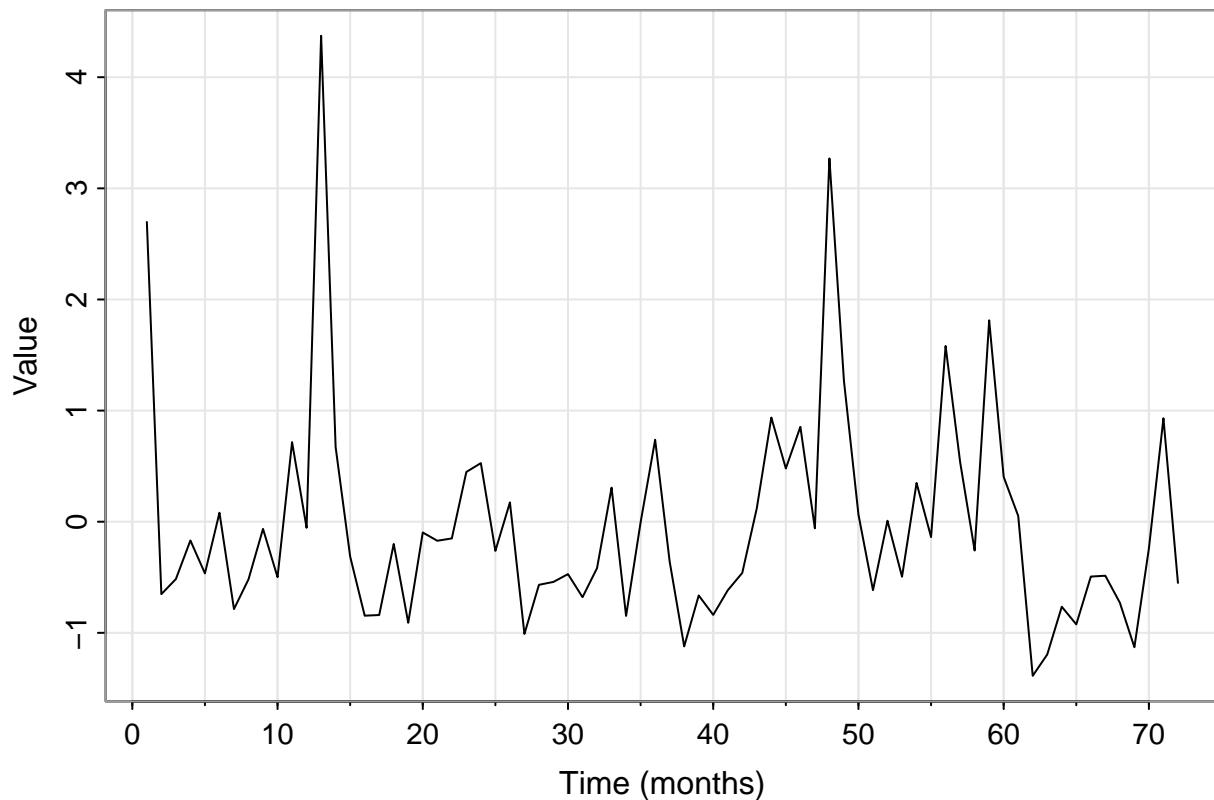
Pollutant 88101 – Cluster 2



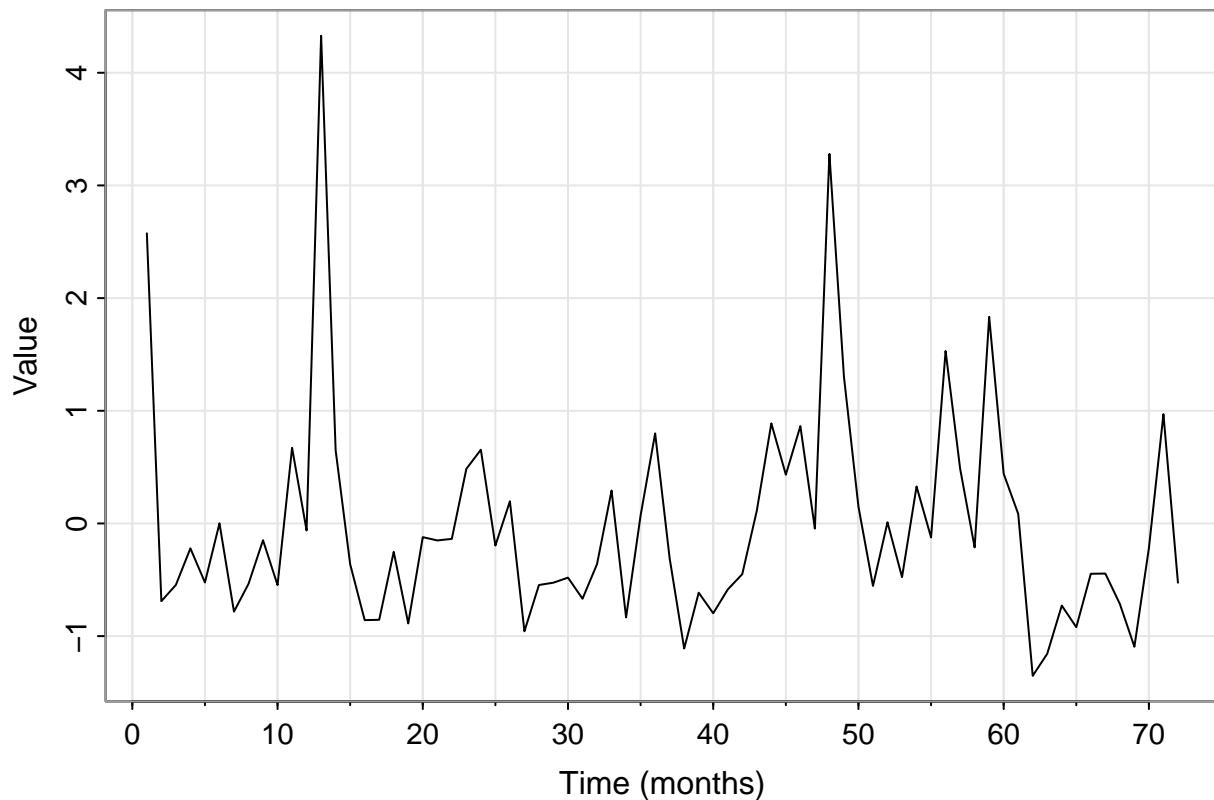
Pollutant 88101 – Cluster 3



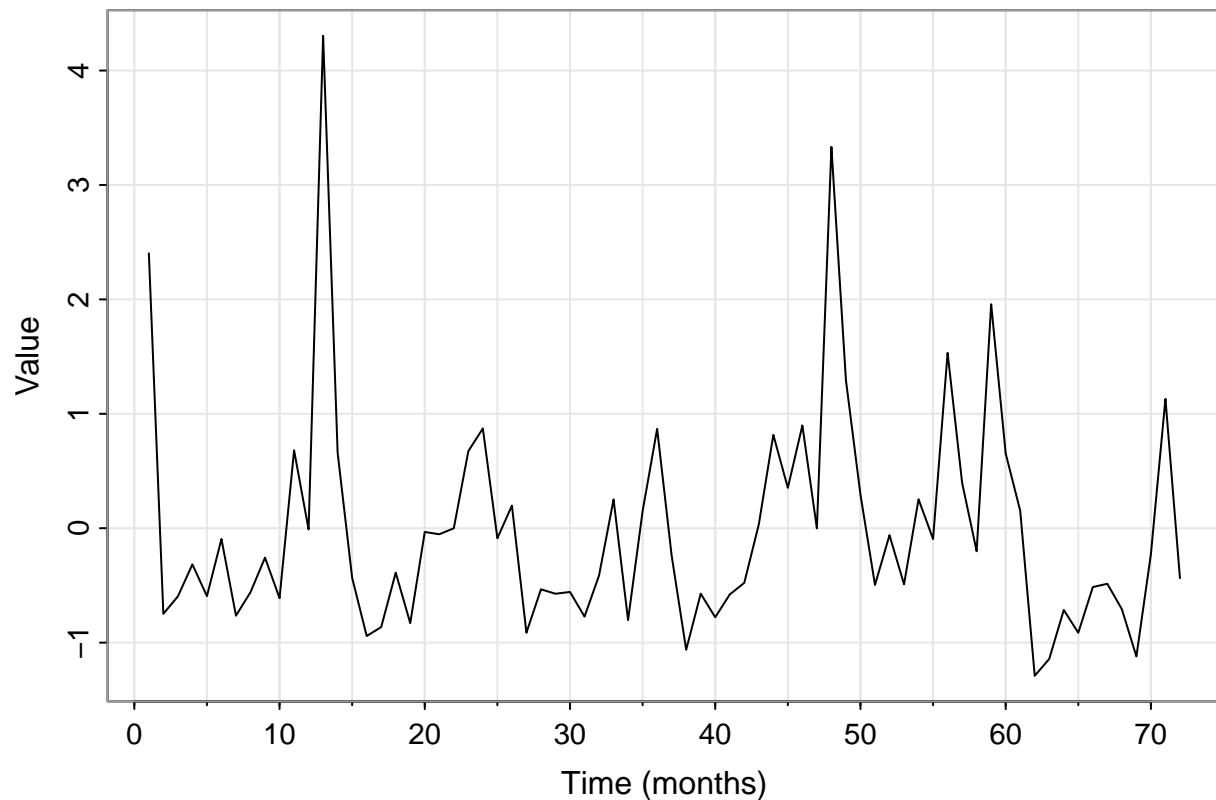
Pollutant 88101 – Cluster 4



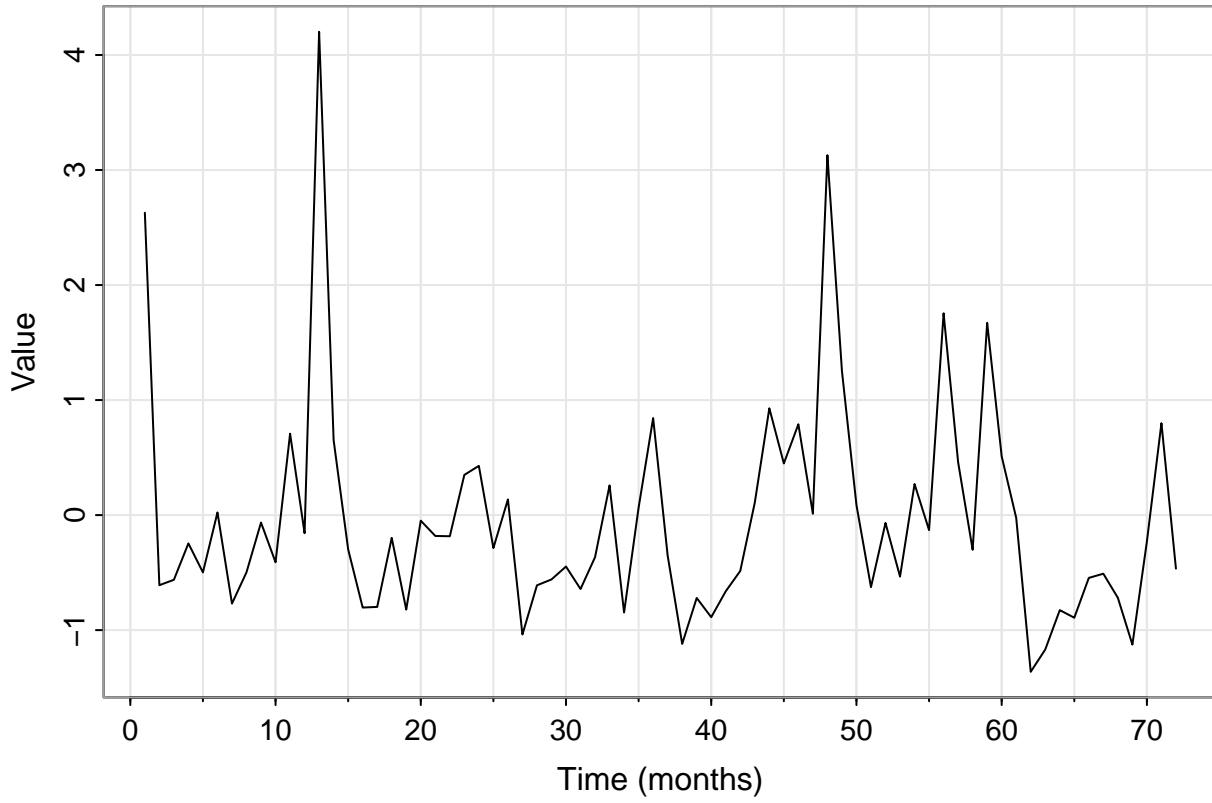
Pollutant 88101 – Cluster 5



Pollutant 88101 – Cluster 6



Pollutant 88101 – Cluster 7



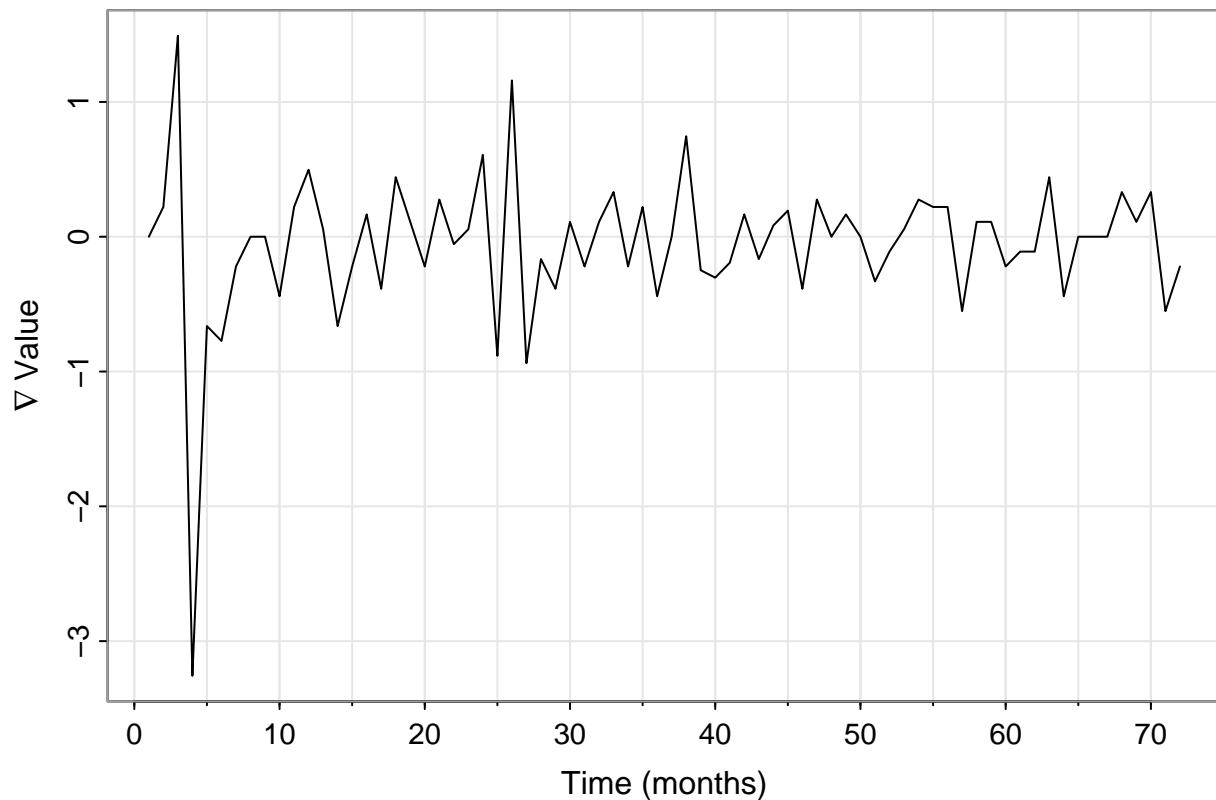
It turns out that the time series for Lead was not stationary, so we need to detrend the data before using it

```
for (i in pollutants$parametercodes.code){
  for (j in 1:num_clus){
    EPA_clus_ts = final_EPA_agg_data %>% filter(Pollutant == i) %>% filter(Cluster == j)
    lead_idx = which(final_EPA_agg_data$Pollutant == "14129" & final_EPA_agg_data$Cluster == j)

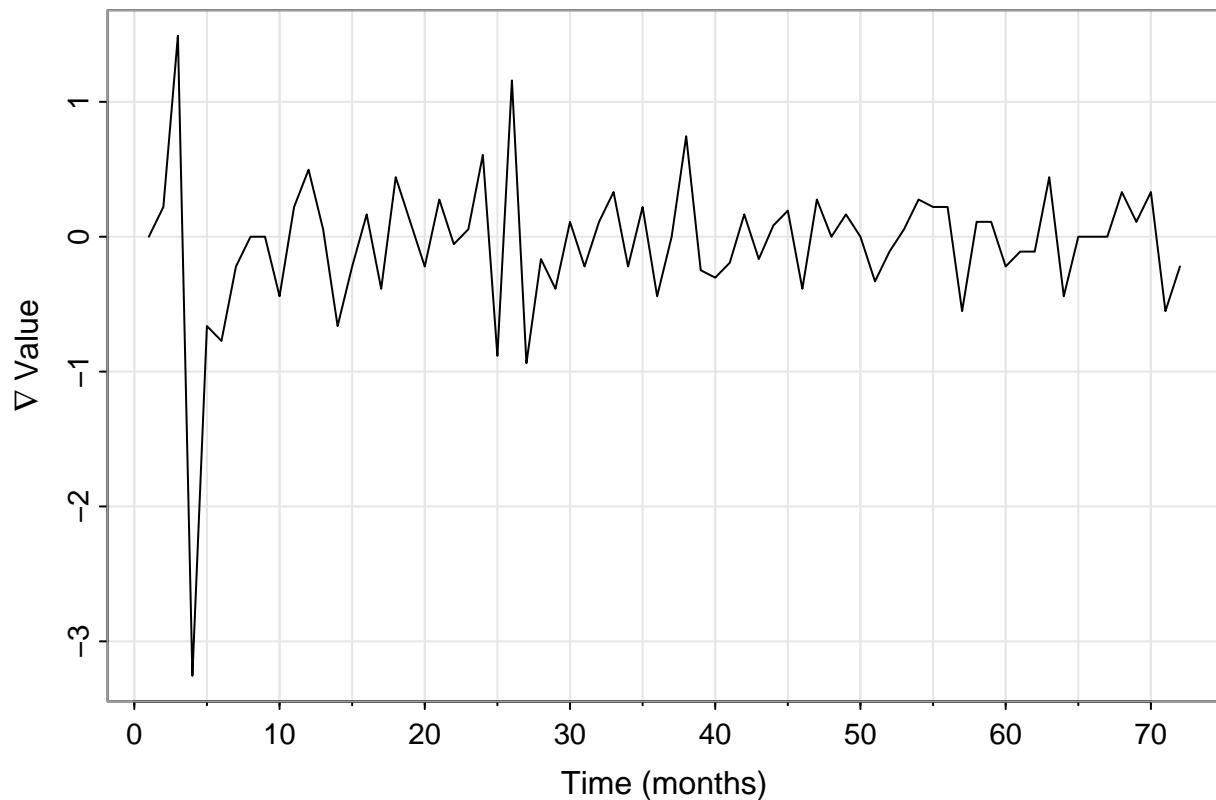
    if (i == "14129"){
      detrended1 = c(0,diff(EPA_clus_ts$Value,lag=1))
      final_EPA_agg_data[lead_idx,2] = detrended1

      title = sprintf("Pollutant %s - Cluster %1.0f",i,j)
      tsplot(detrended1,main = title,xlab = "Time (months)",ylab = expression(nabla~Value))
    } else{
      title = sprintf("Pollutant %s - Cluster %1.0f",i,j)
      tsplot(EPA_clus_ts$Value,main = title,xlab = "Time (months)",ylab = "Value")
    }
  }
}
```

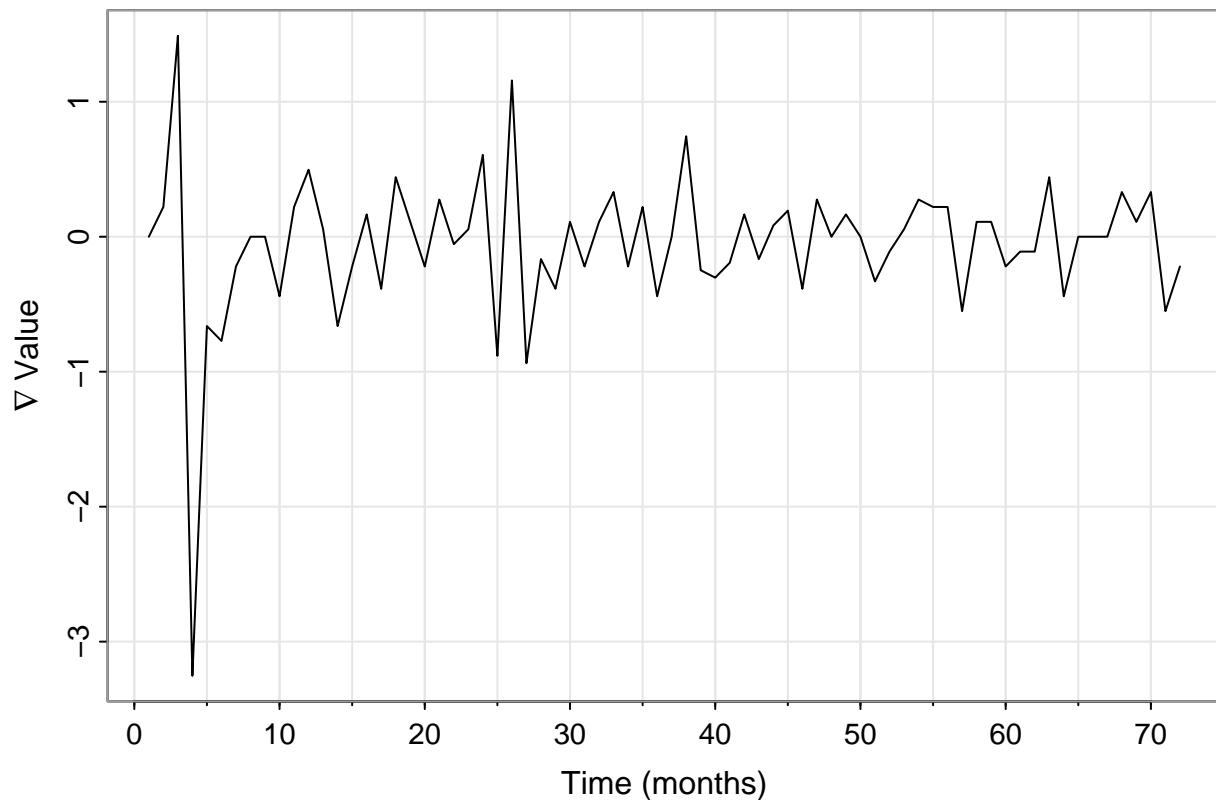
Pollutant 14129 – Cluster 1



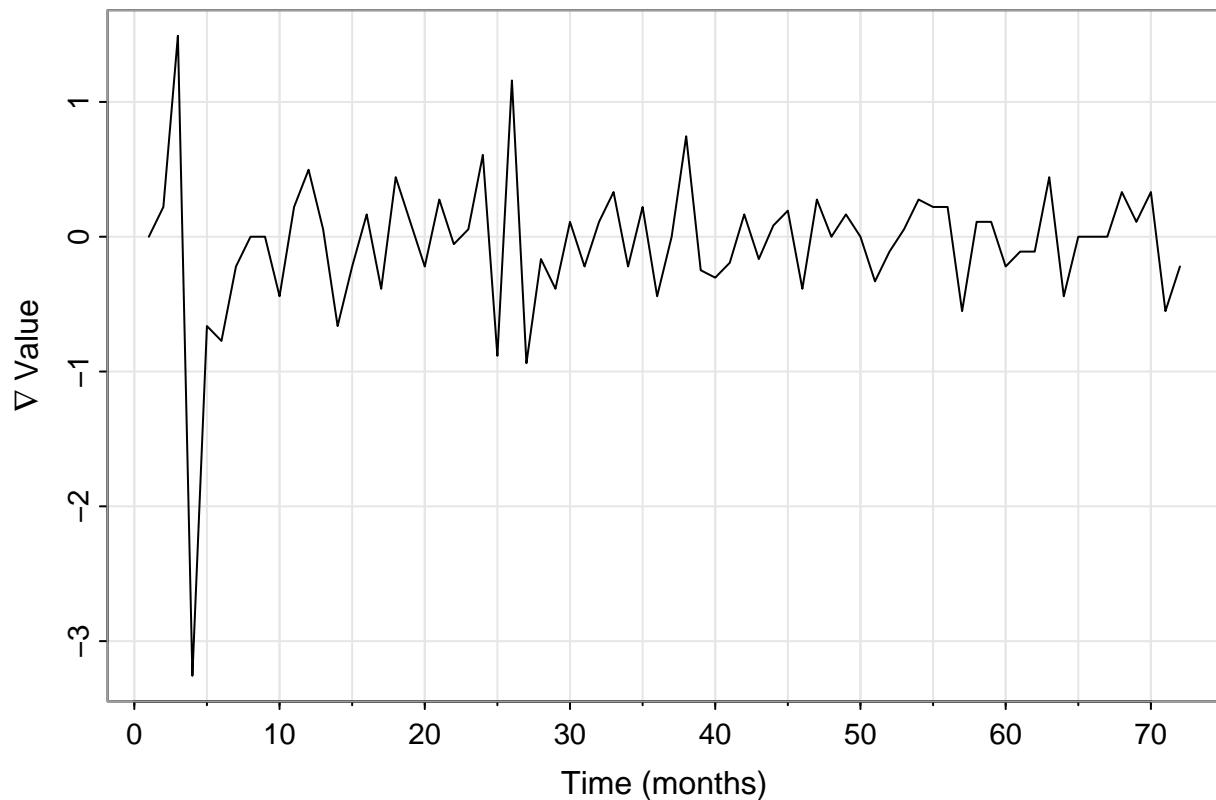
Pollutant 14129 – Cluster 2



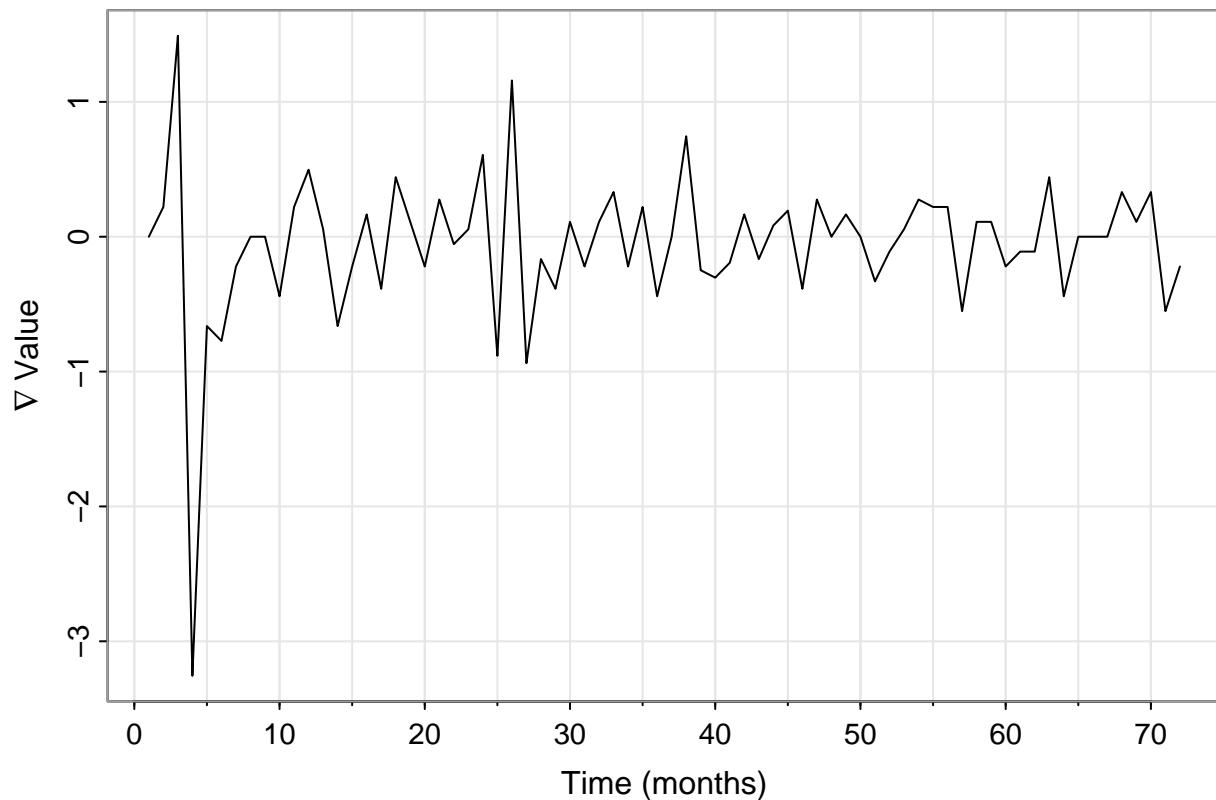
Pollutant 14129 – Cluster 3



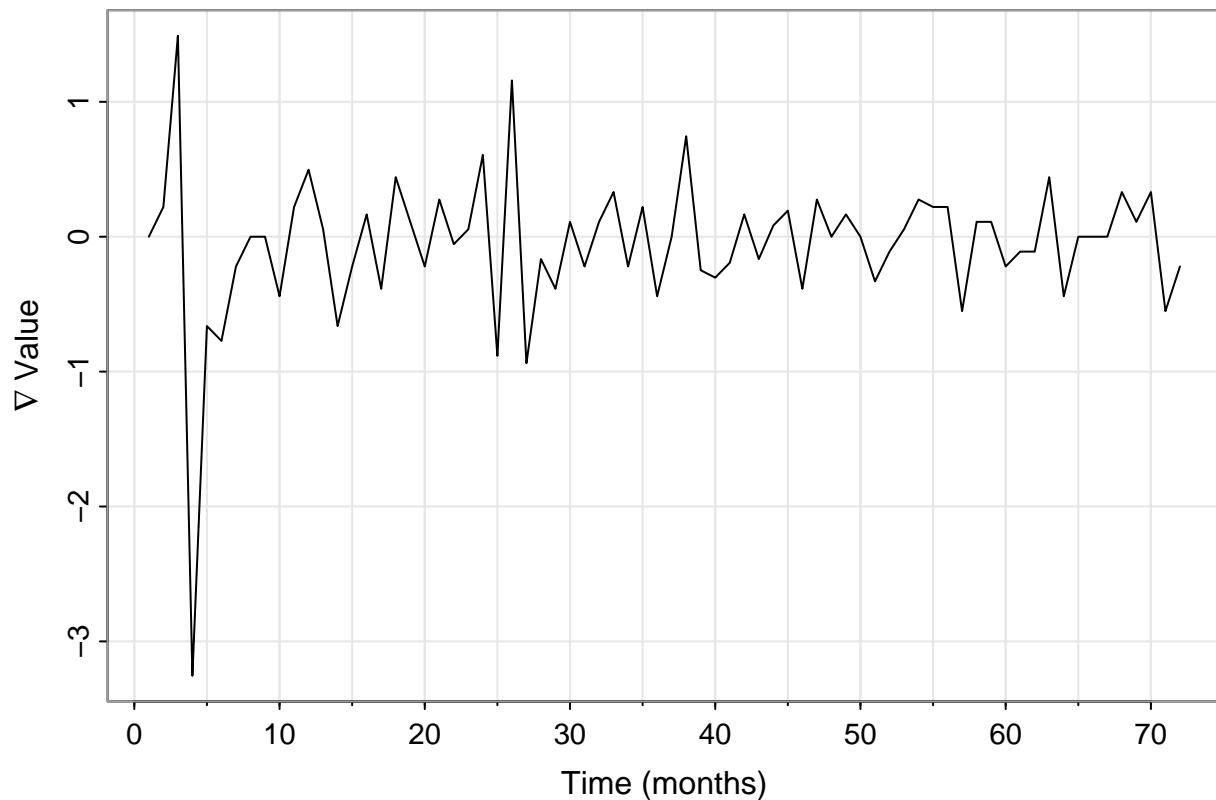
Pollutant 14129 – Cluster 4



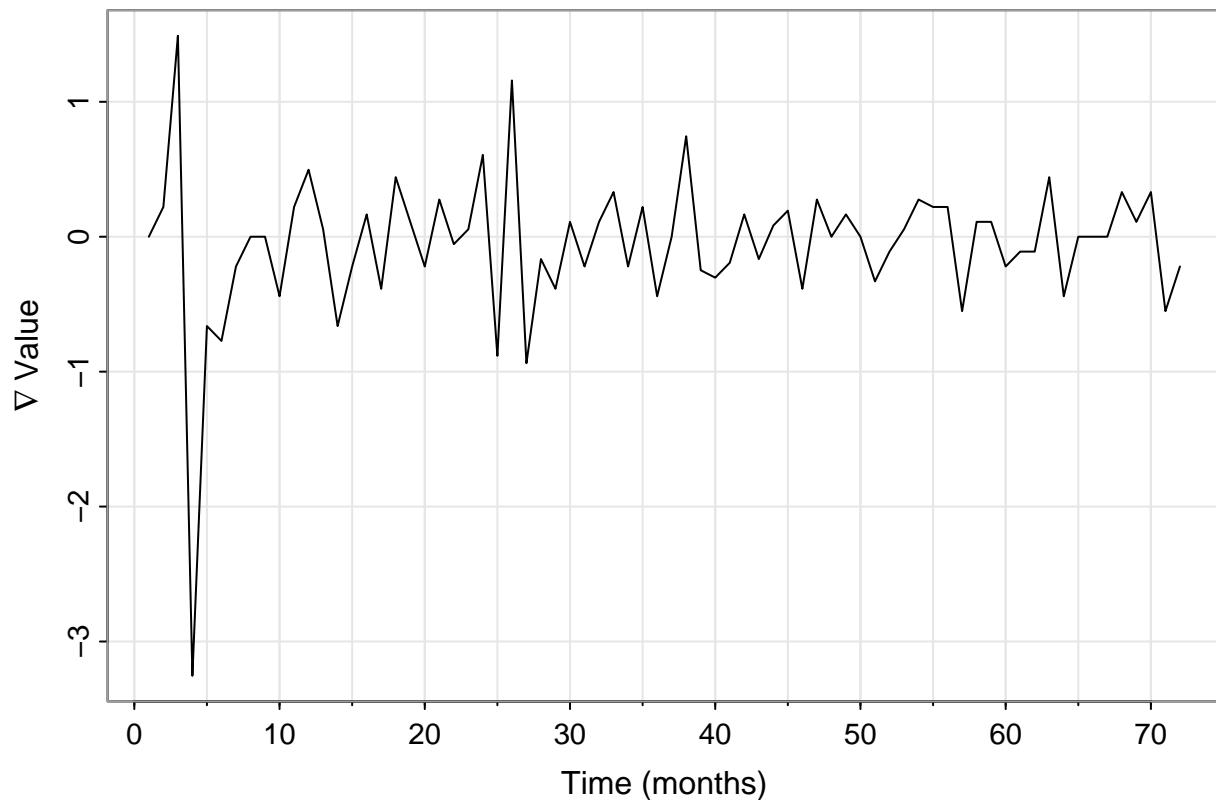
Pollutant 14129 – Cluster 5



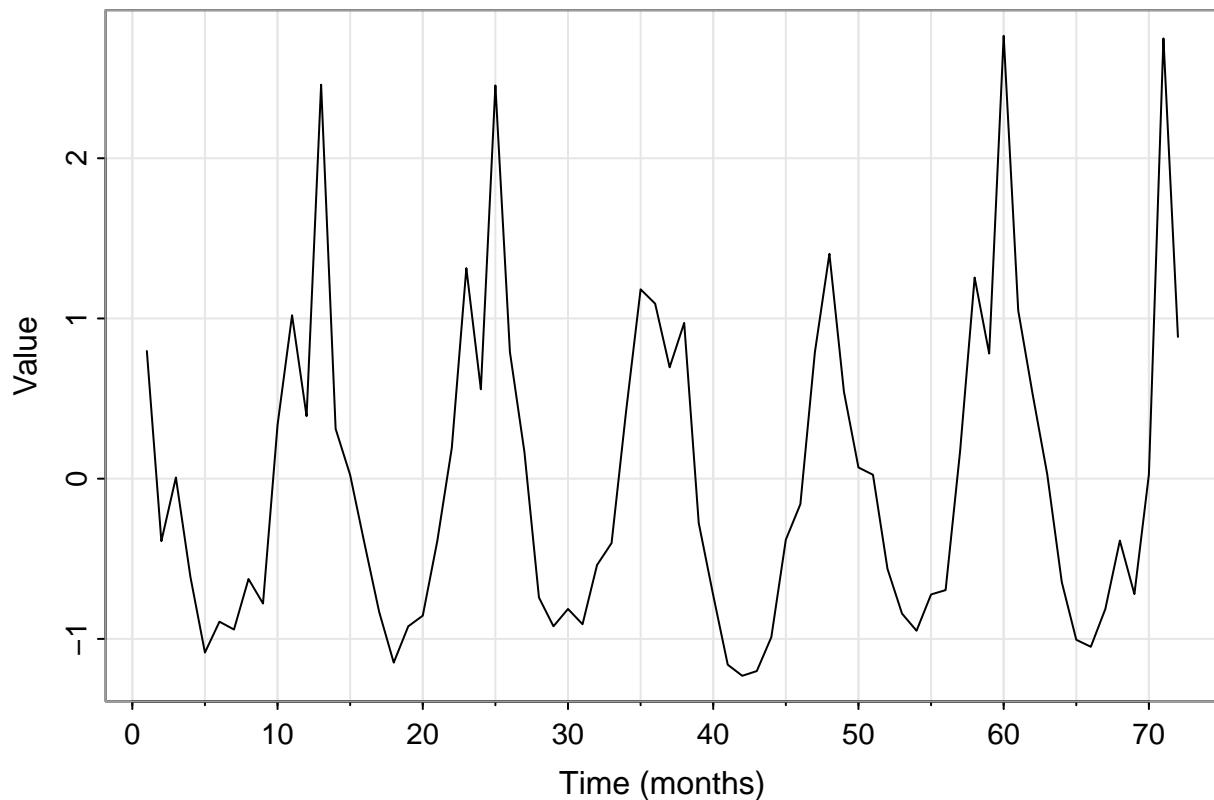
Pollutant 14129 – Cluster 6



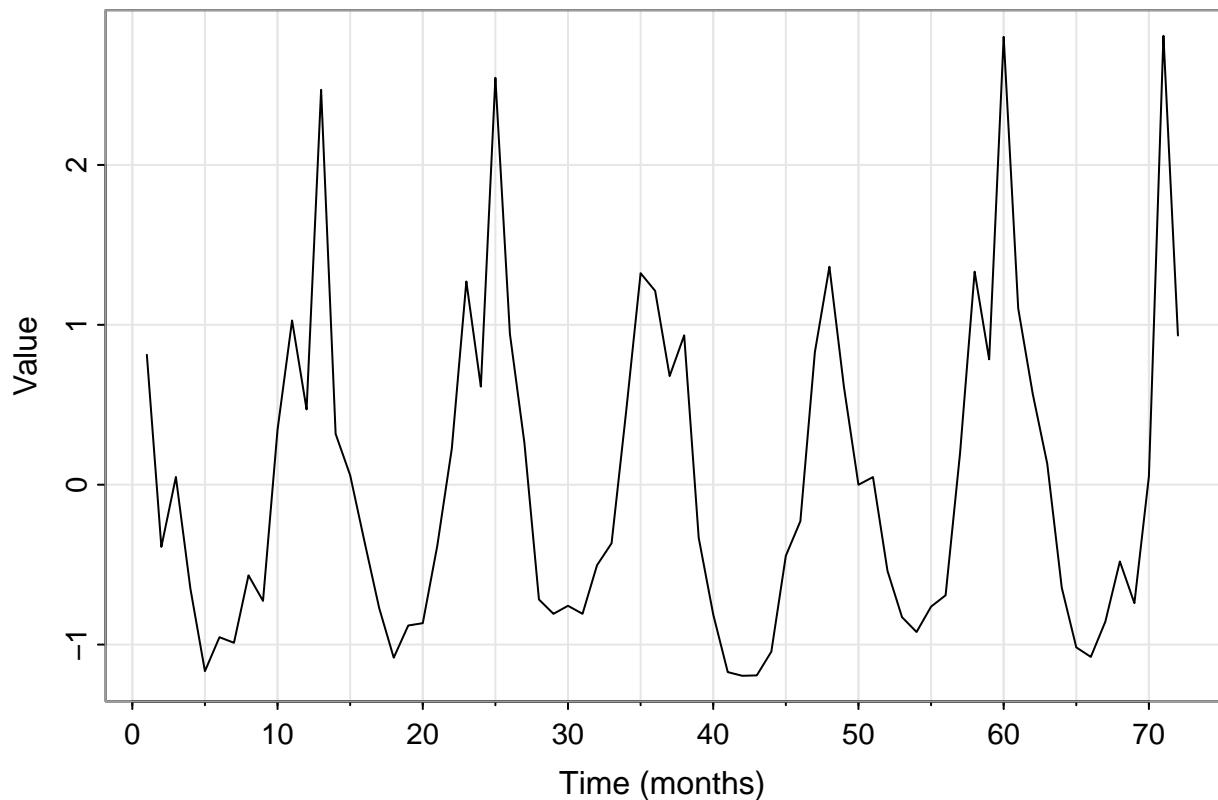
Pollutant 14129 – Cluster 7



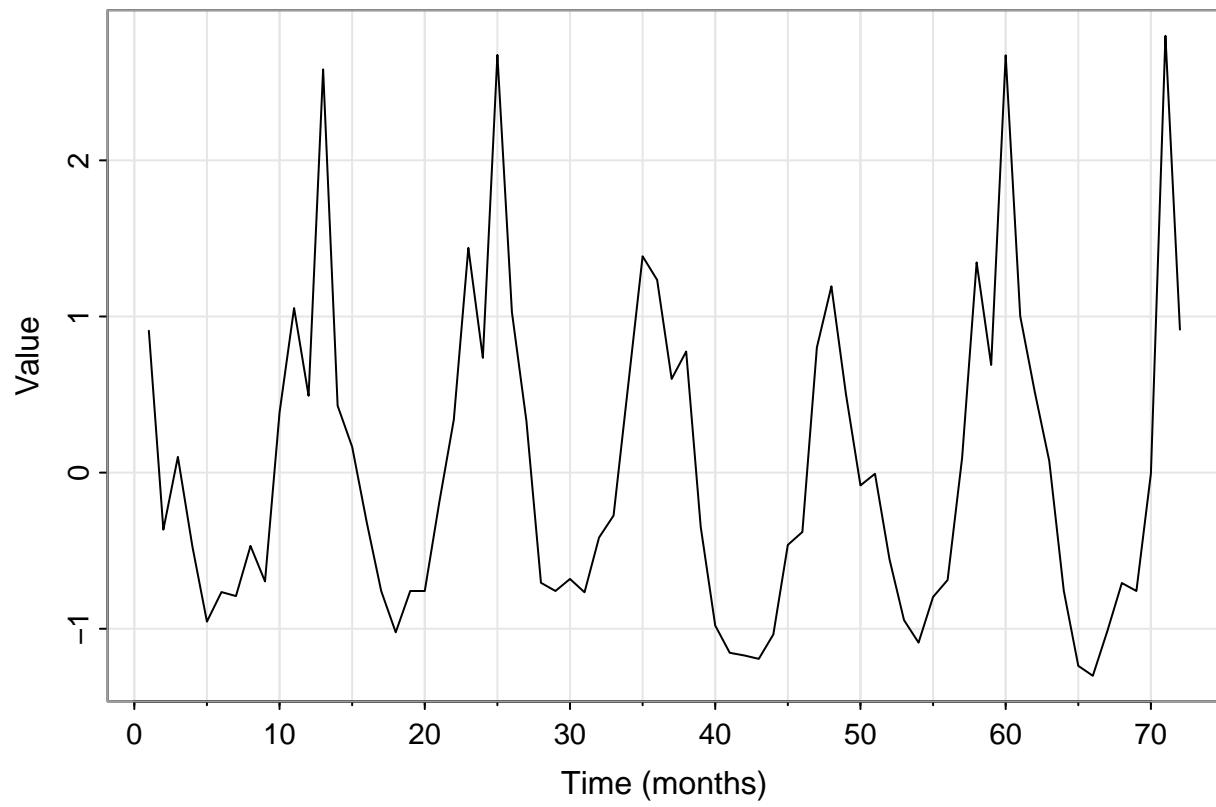
Pollutant 42101 – Cluster 1



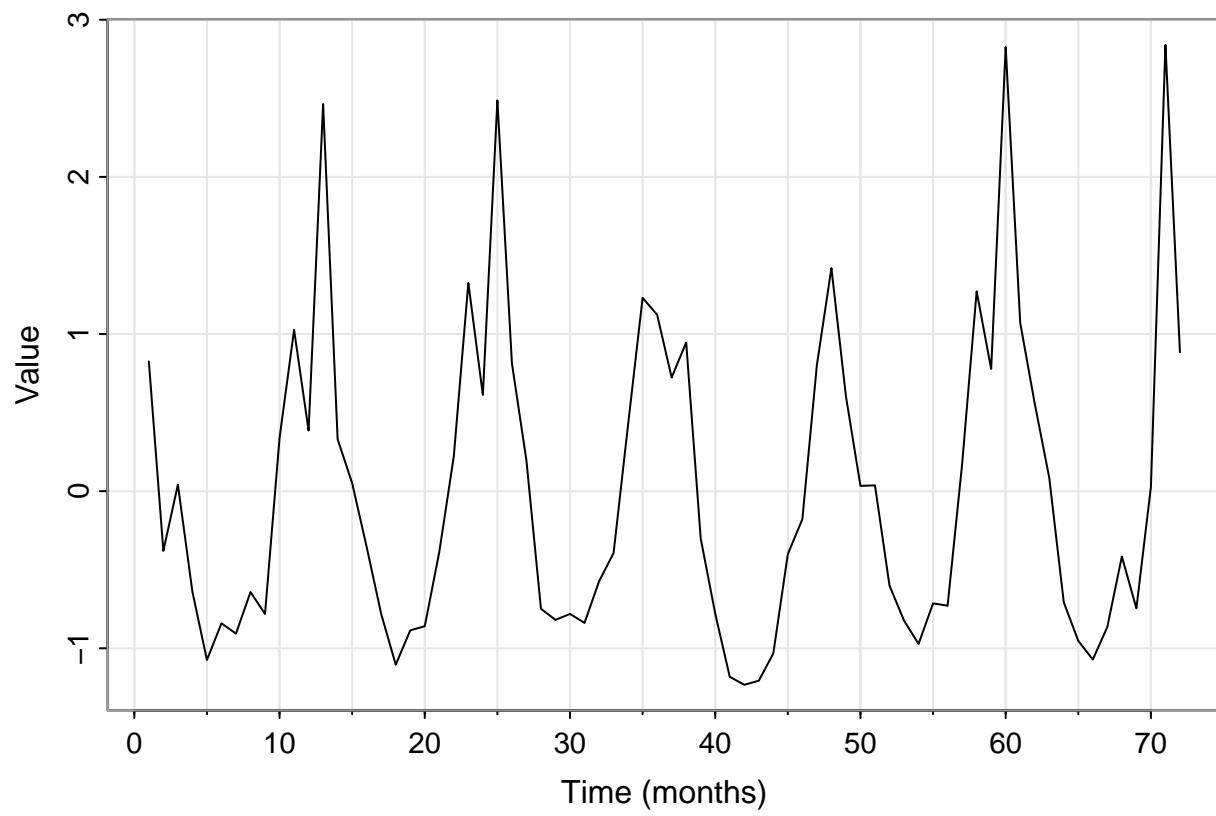
Pollutant 42101 – Cluster 2



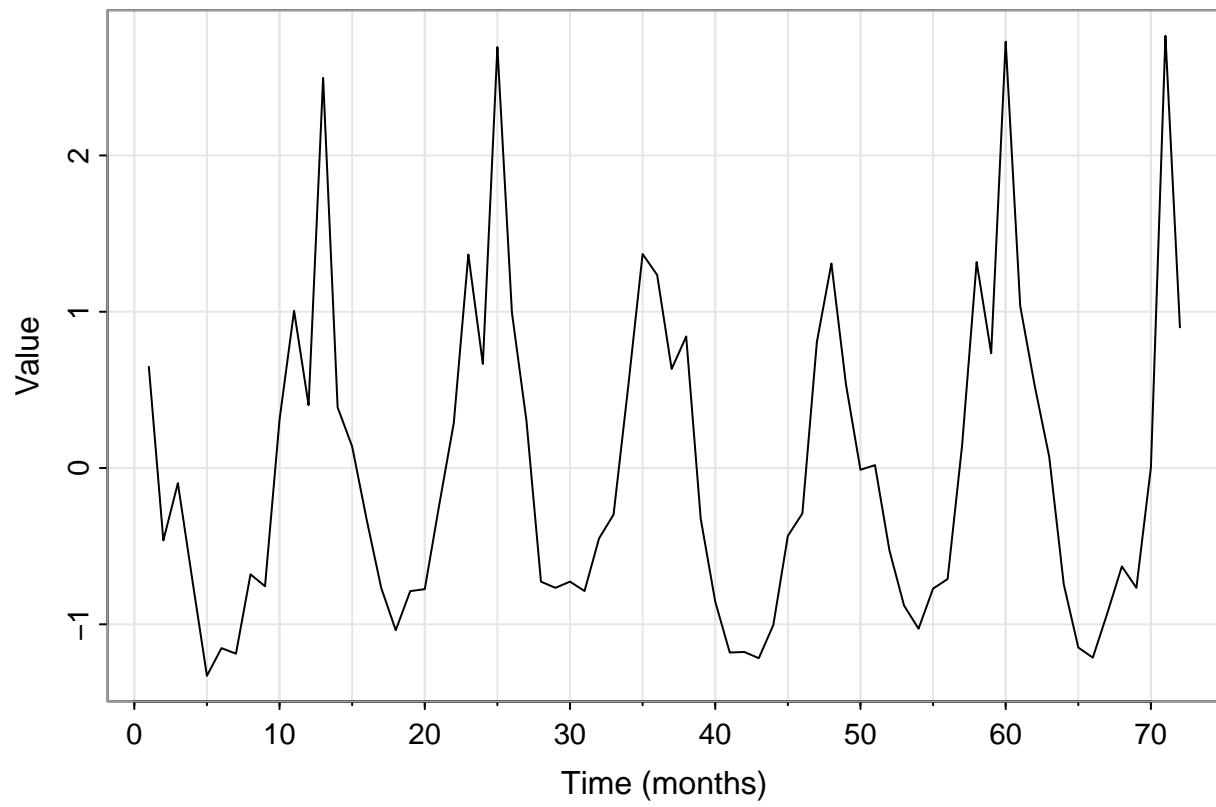
Pollutant 42101 – Cluster 3



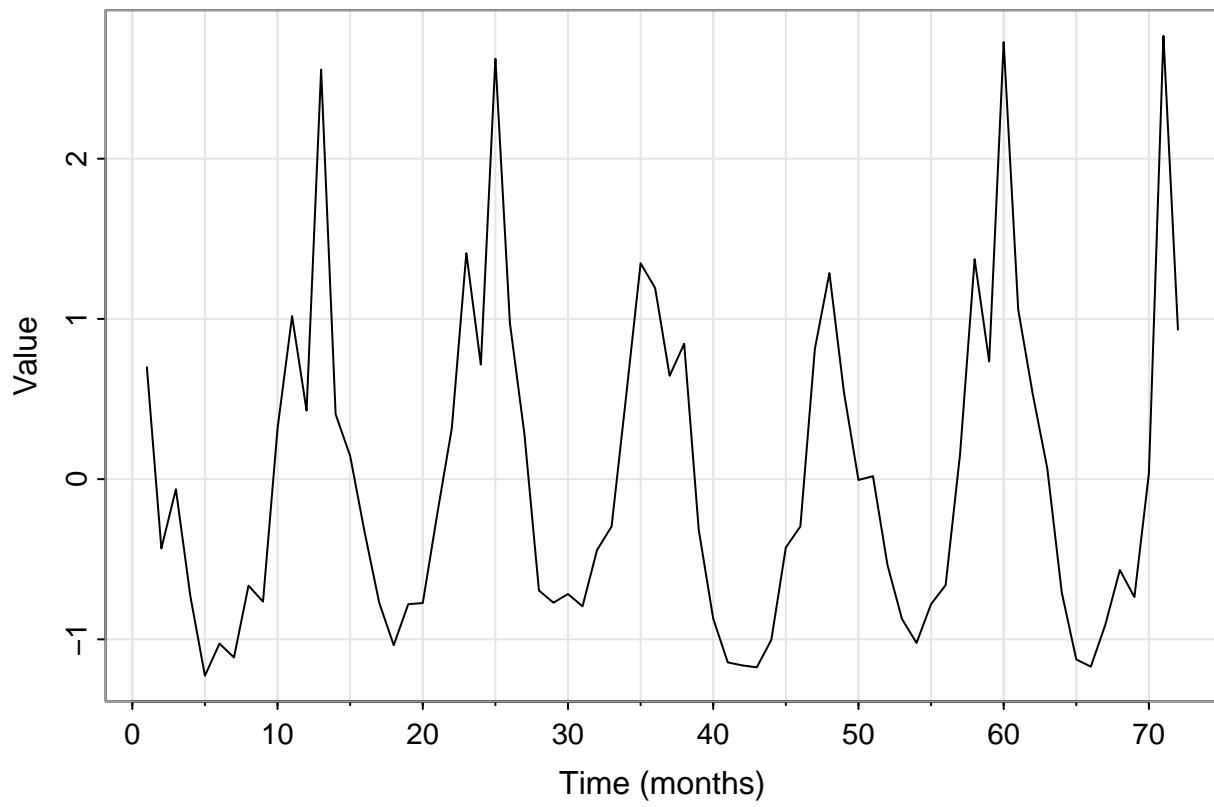
Pollutant 42101 – Cluster 4



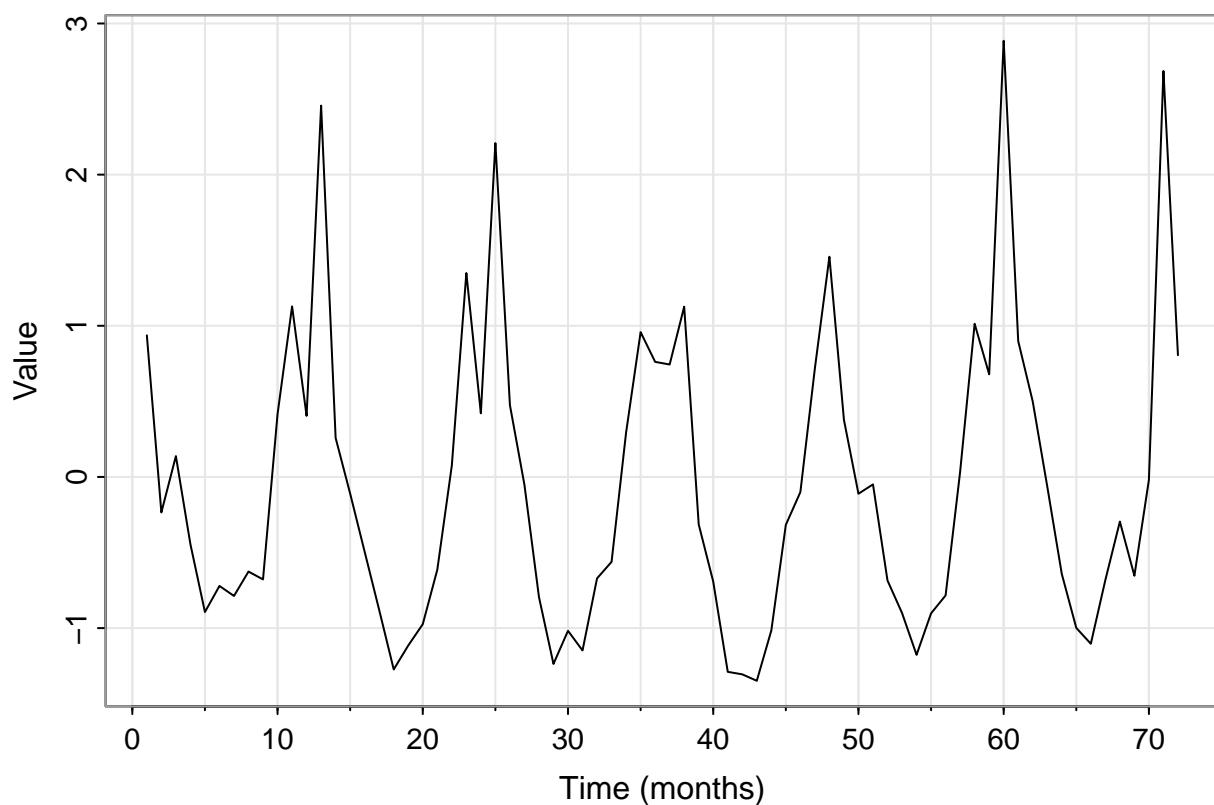
Pollutant 42101 – Cluster 5



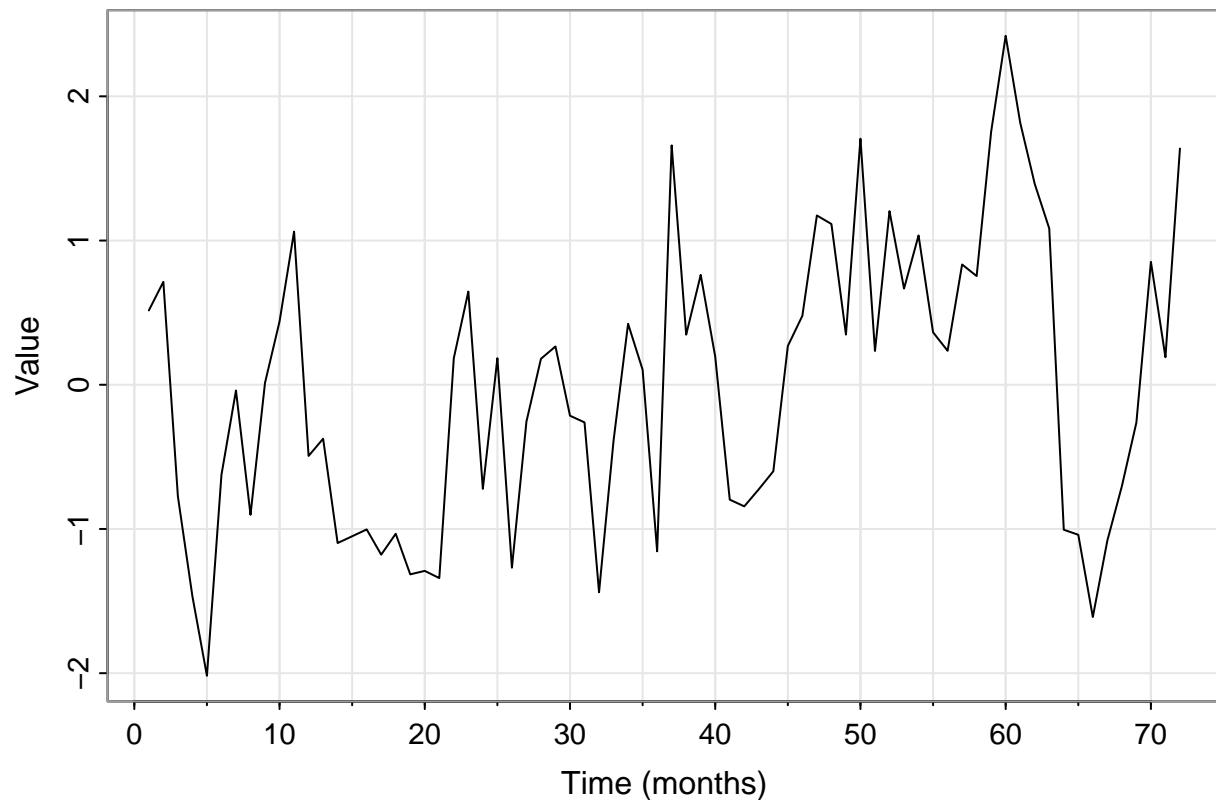
Pollutant 42101 – Cluster 6



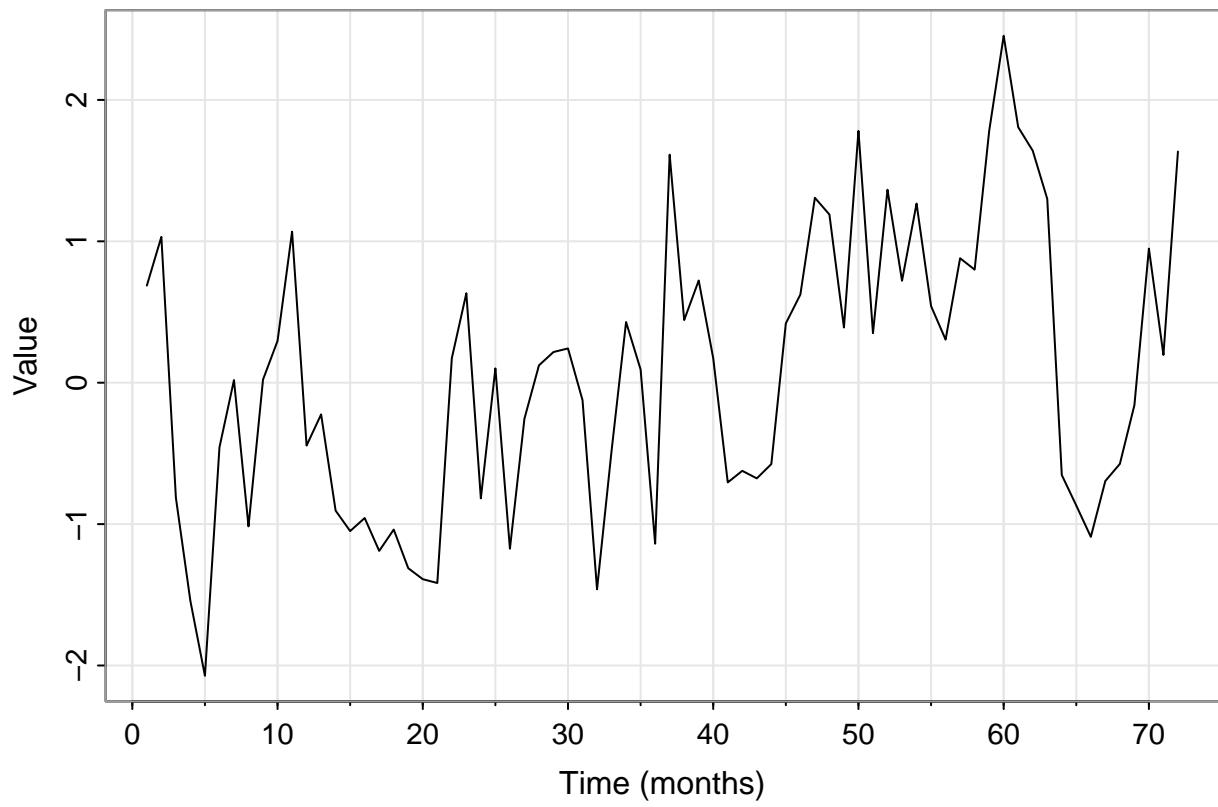
Pollutant 42101 – Cluster 7



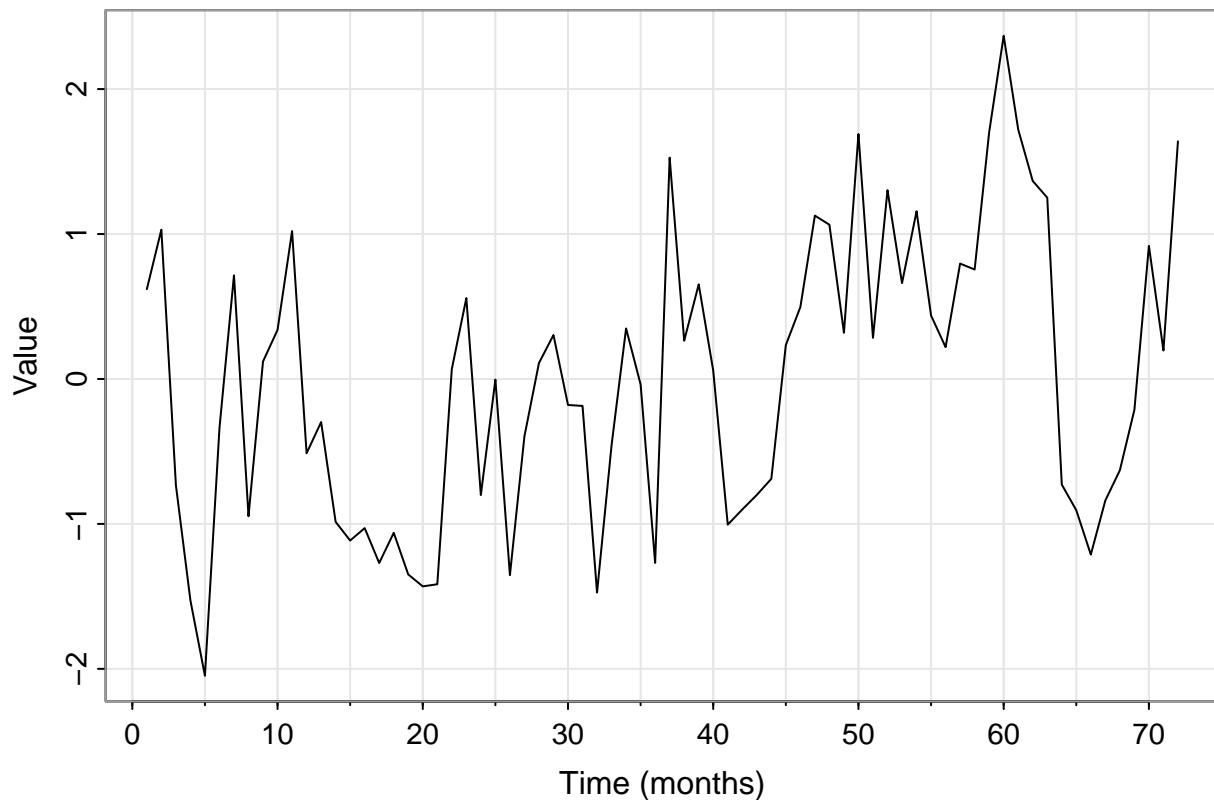
Pollutant 42401 – Cluster 1



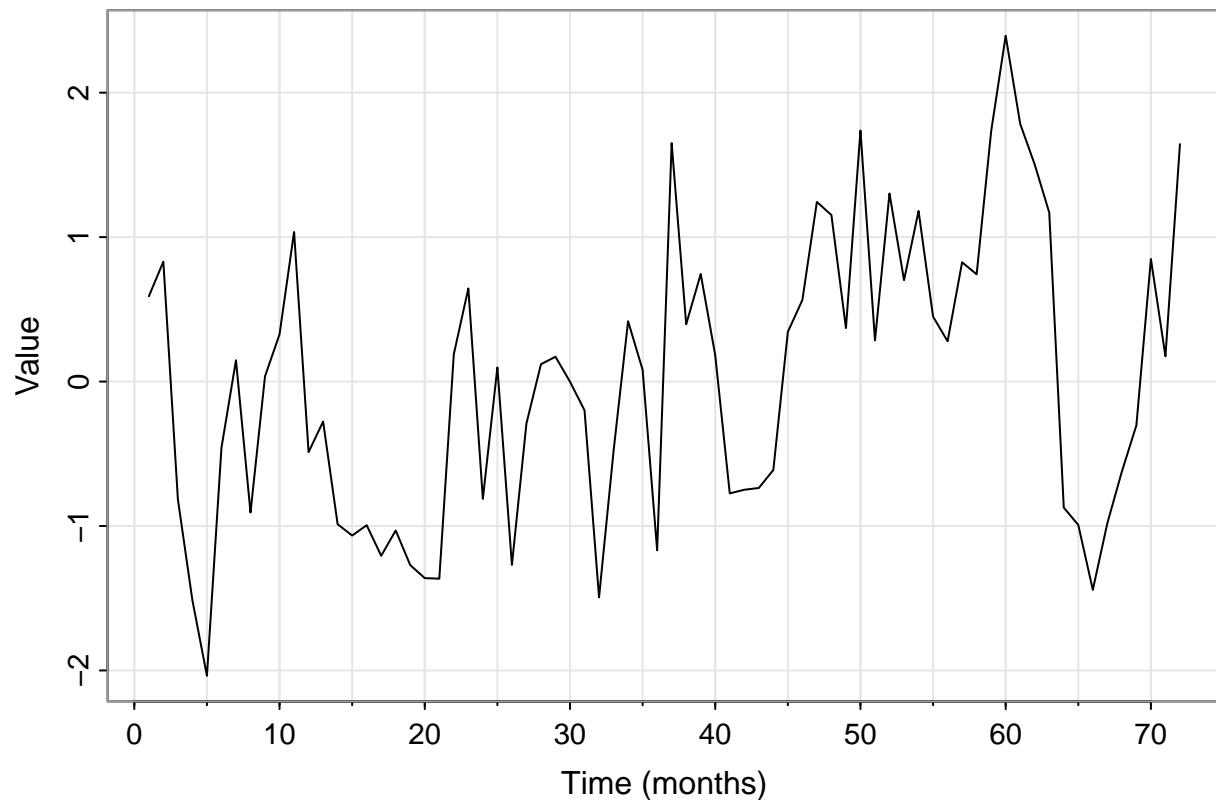
Pollutant 42401 – Cluster 2



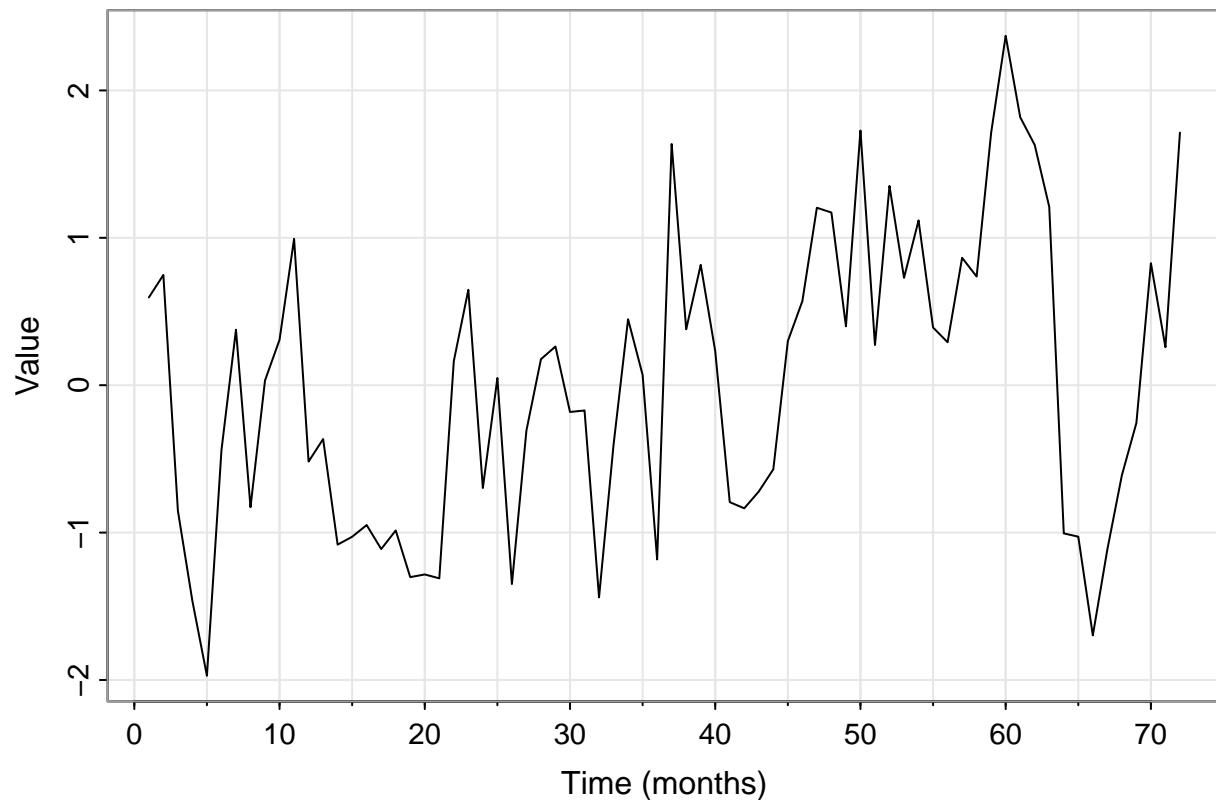
Pollutant 42401 – Cluster 3



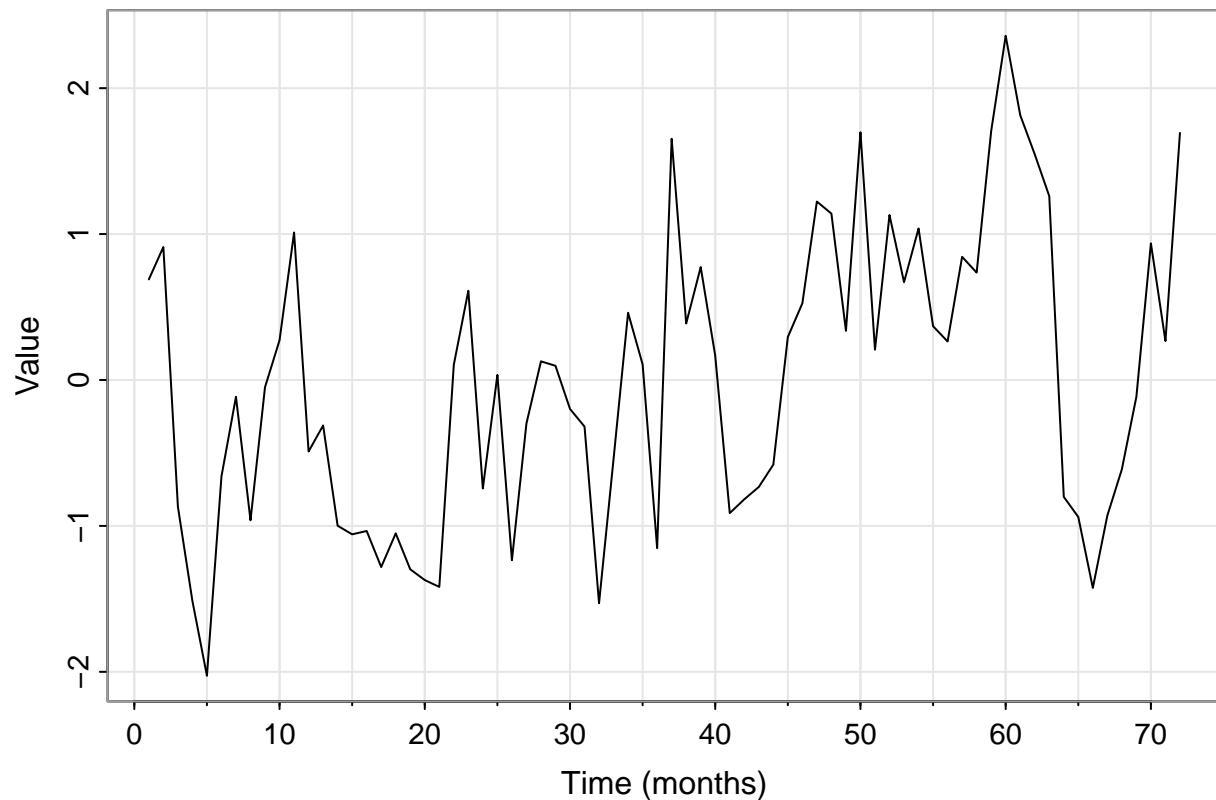
Pollutant 42401 – Cluster 4



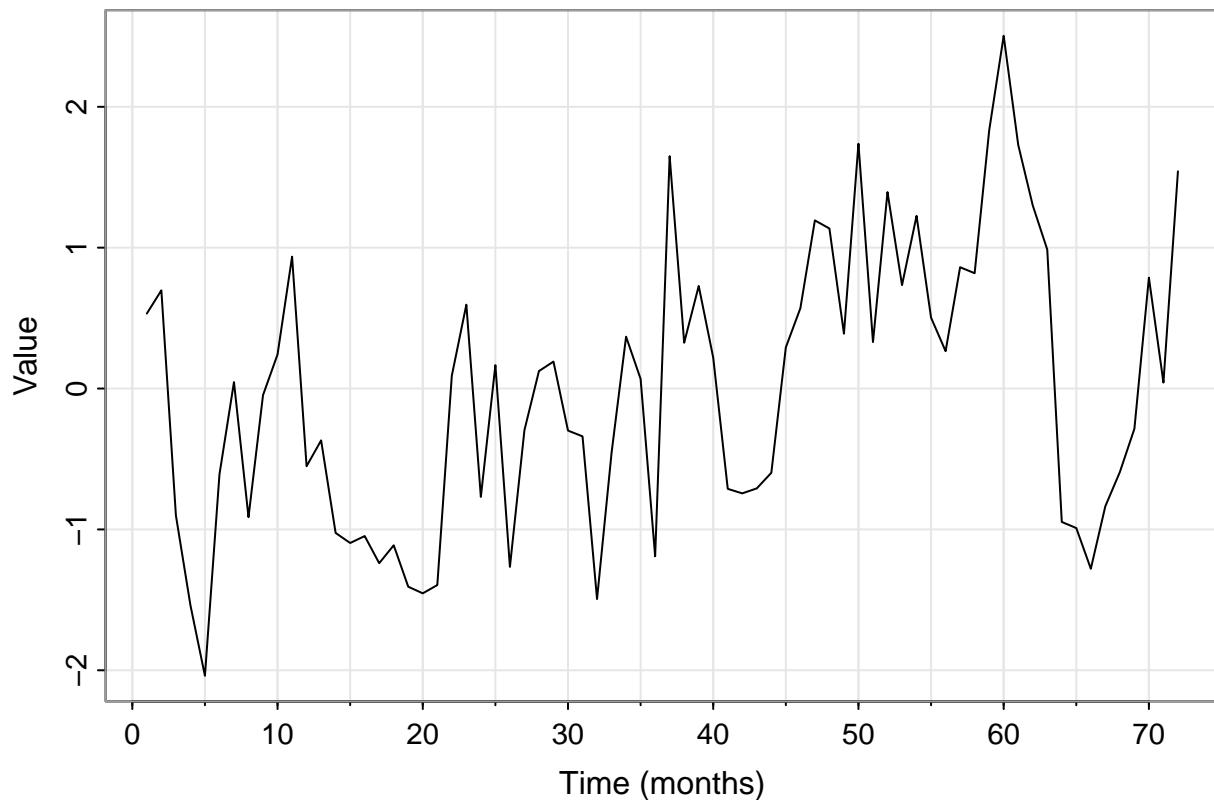
Pollutant 42401 – Cluster 5



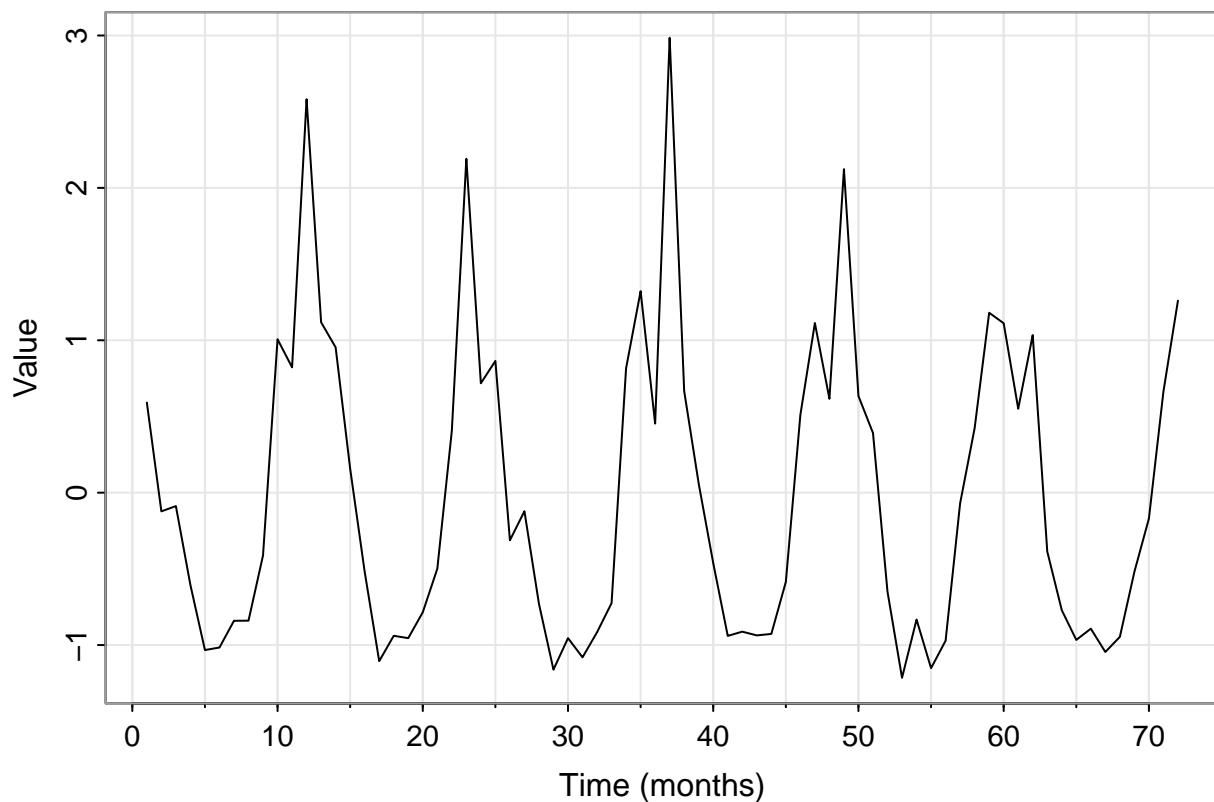
Pollutant 42401 – Cluster 6



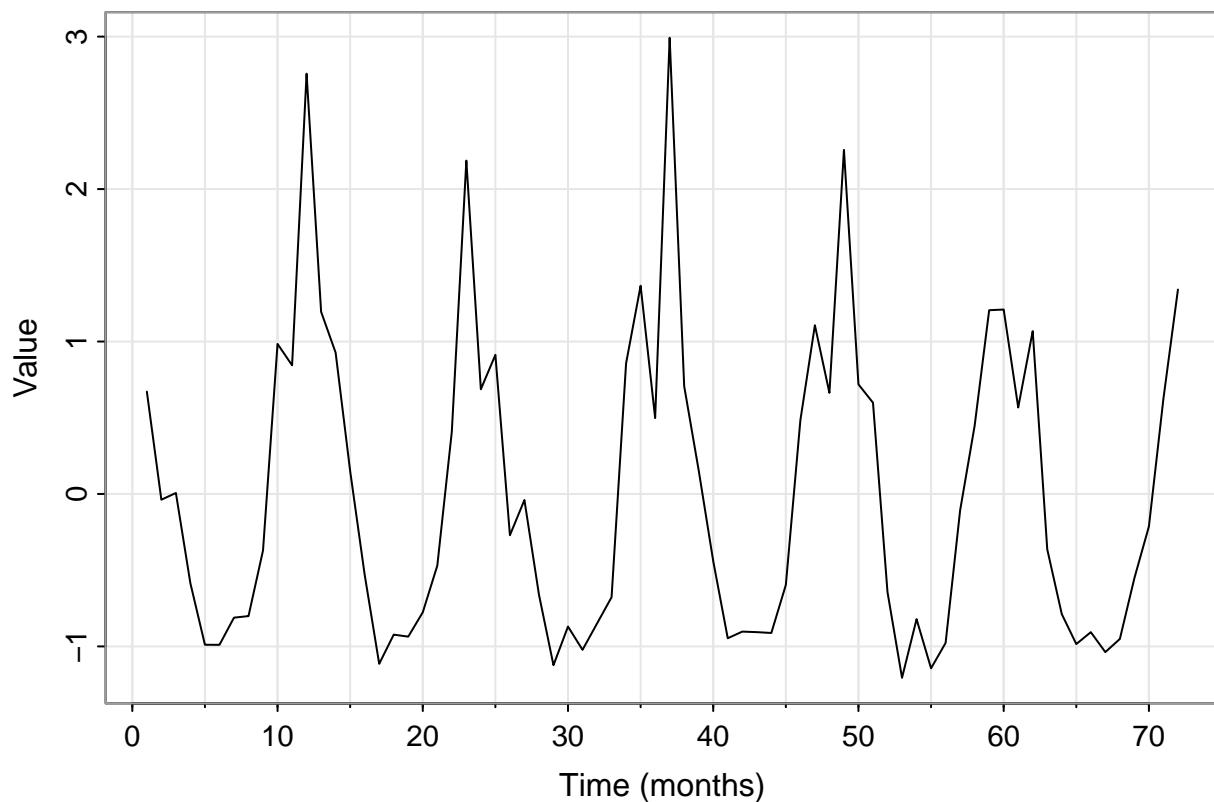
Pollutant 42401 – Cluster 7



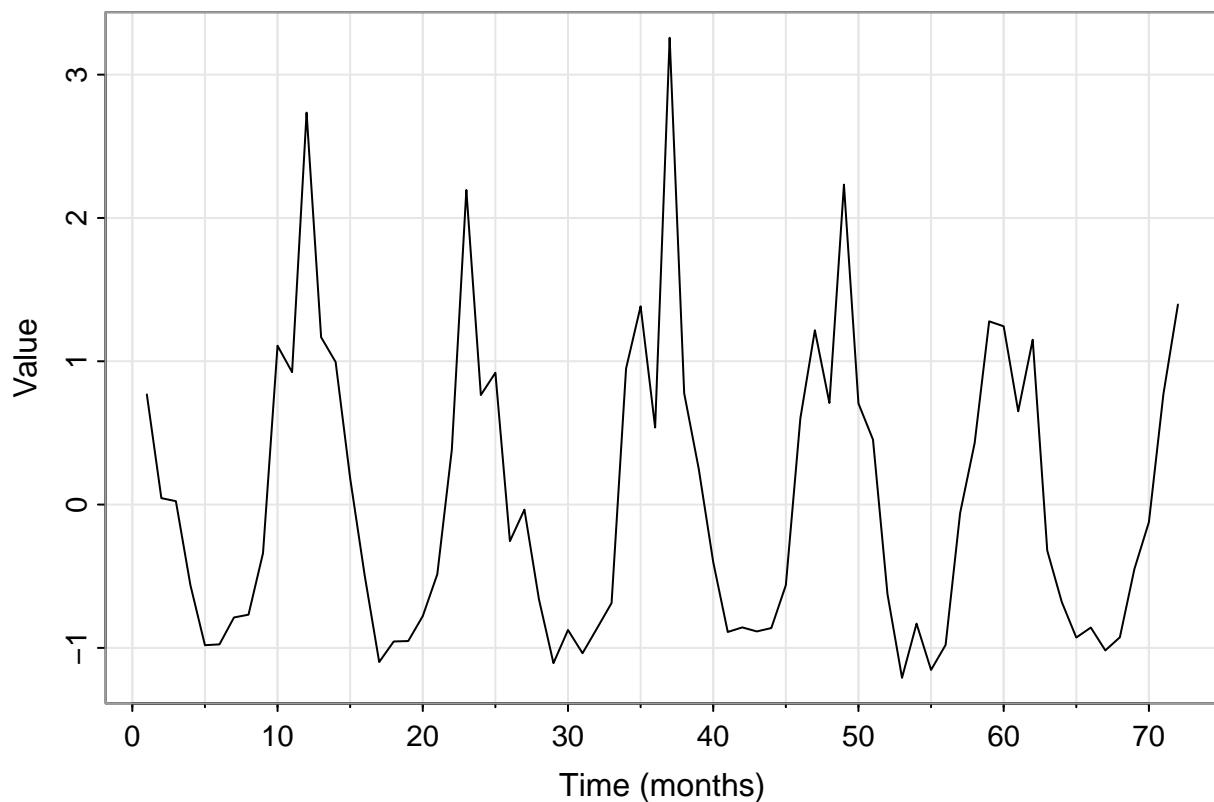
Pollutant 42602 – Cluster 1



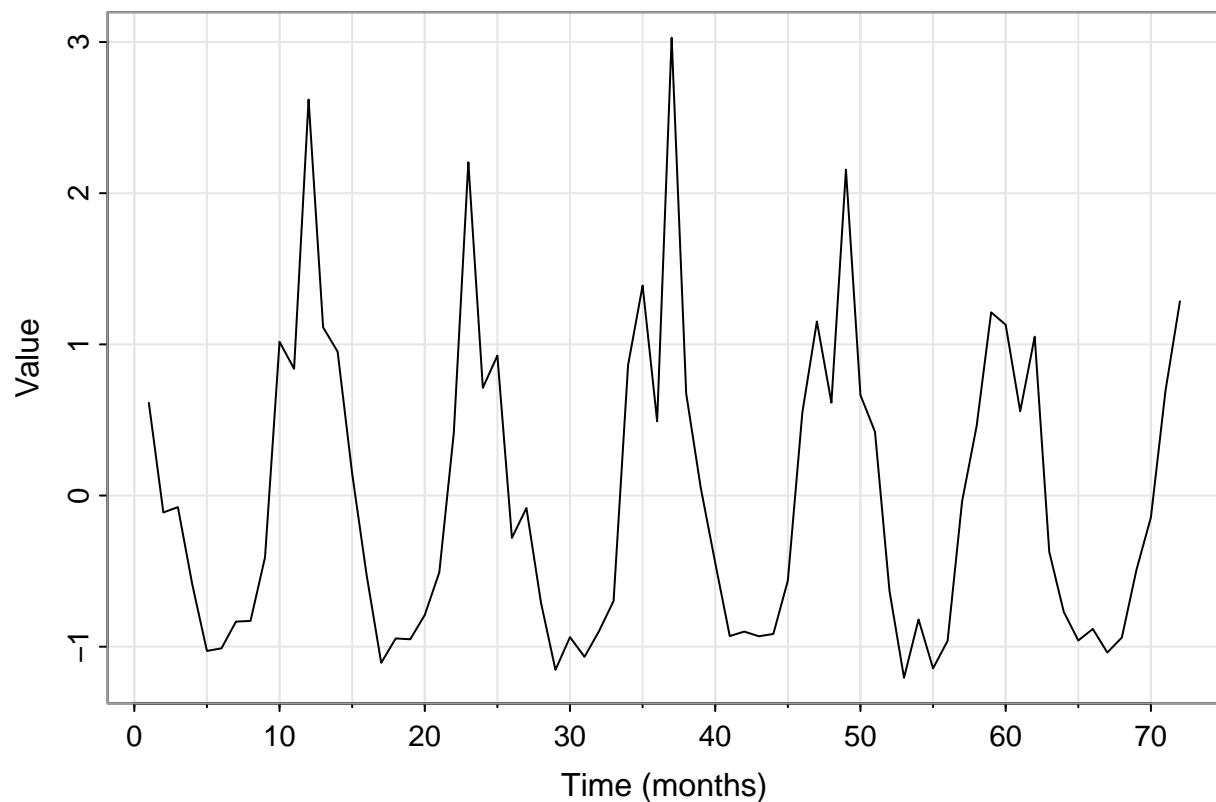
Pollutant 42602 – Cluster 2



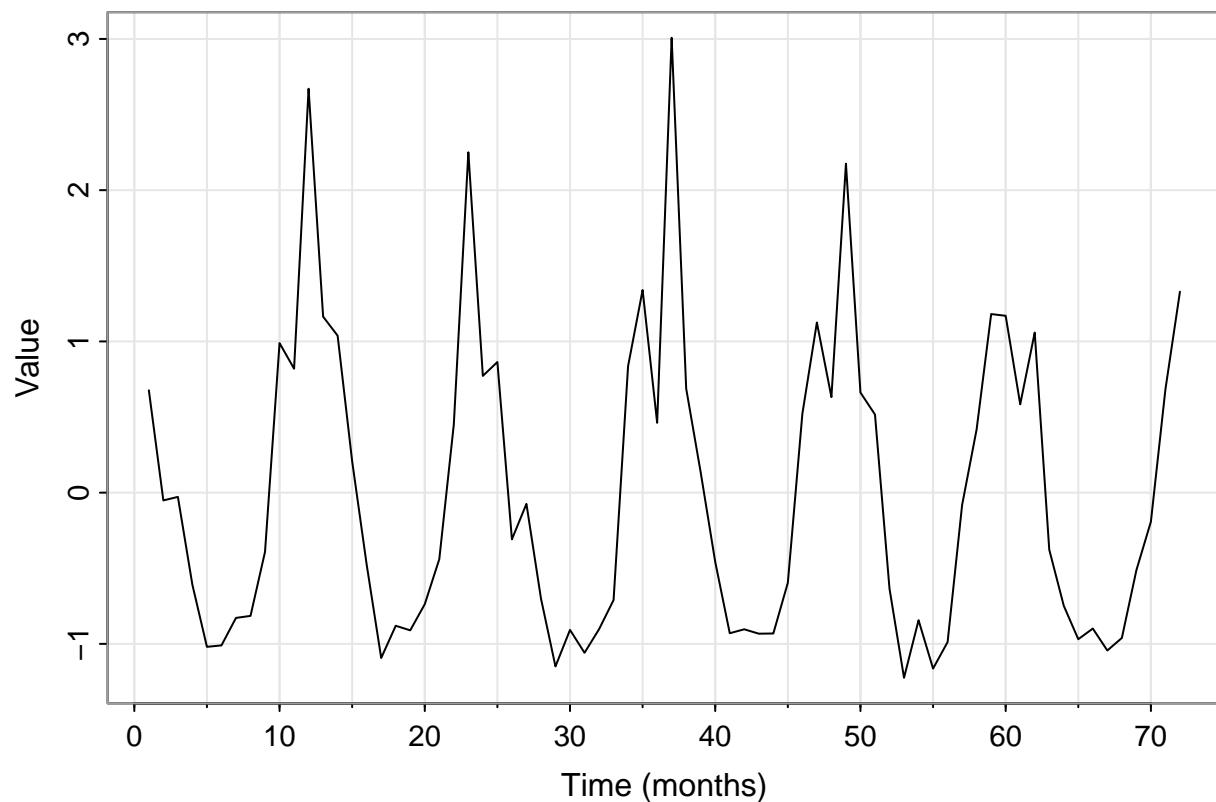
Pollutant 42602 – Cluster 3



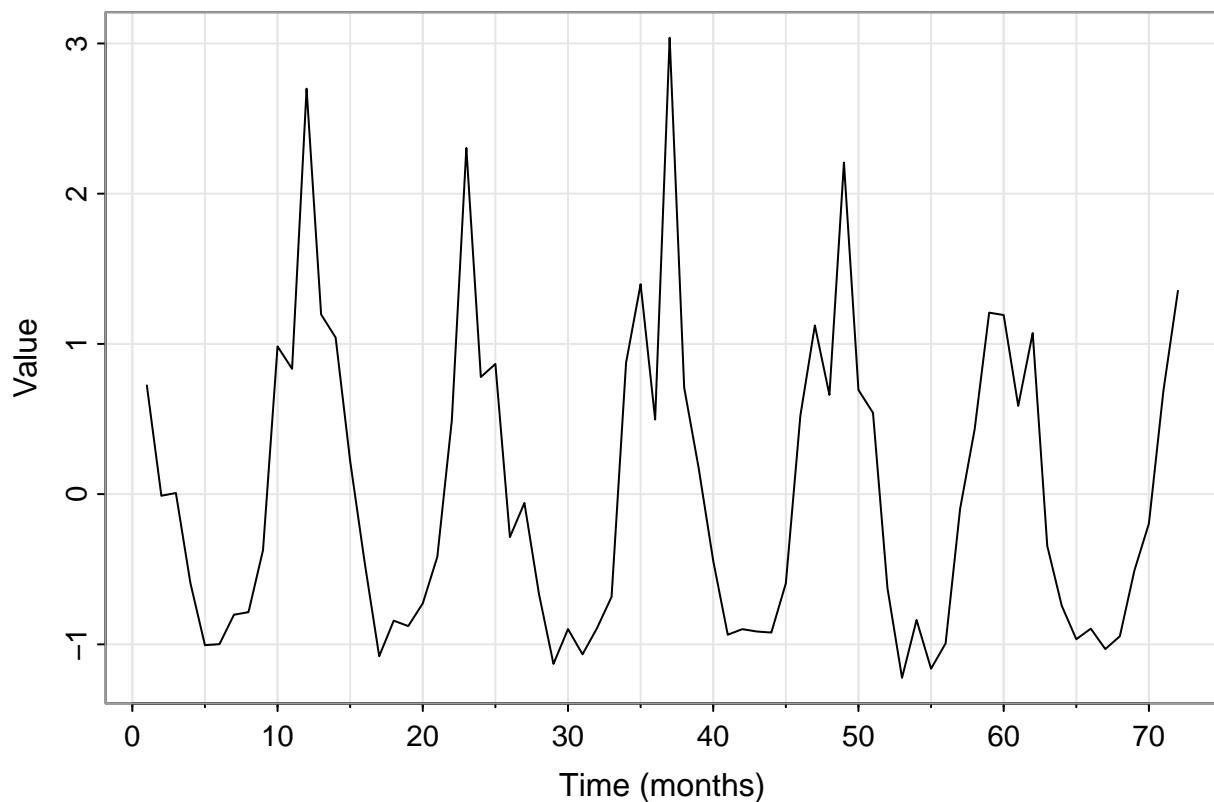
Pollutant 42602 – Cluster 4



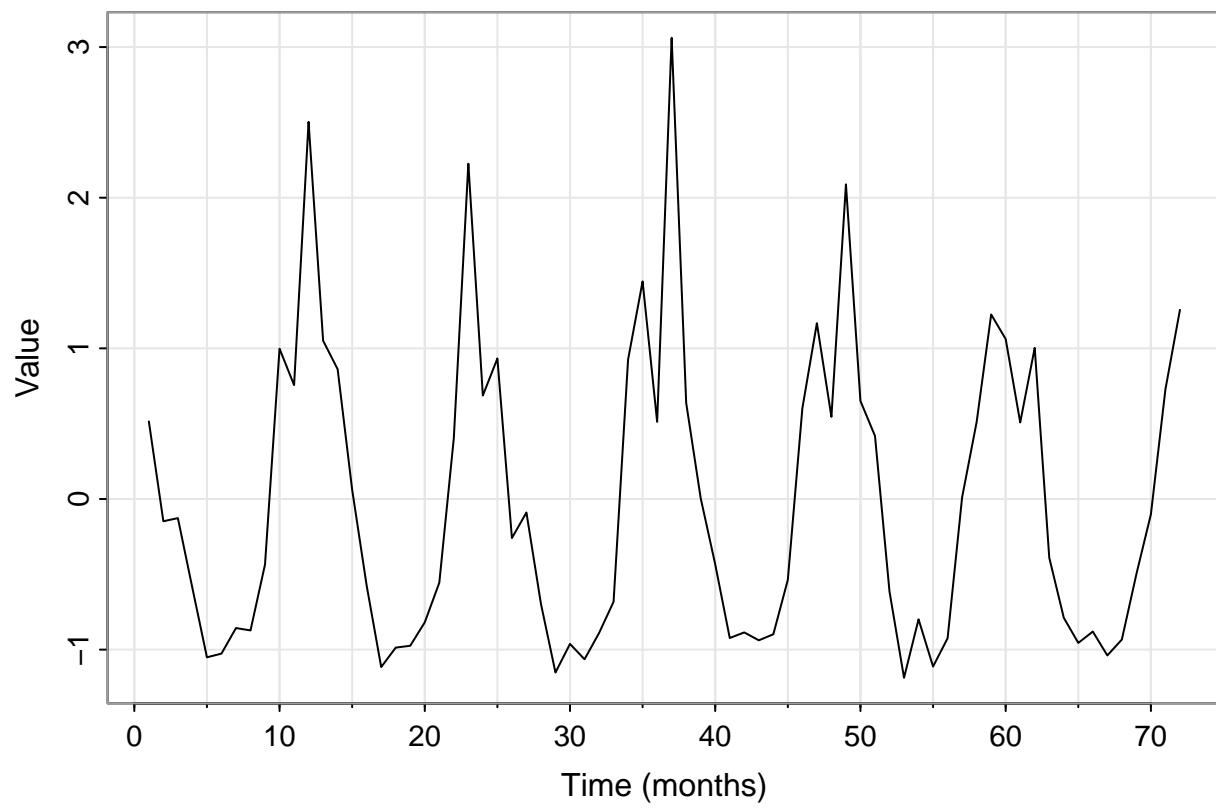
Pollutant 42602 – Cluster 5



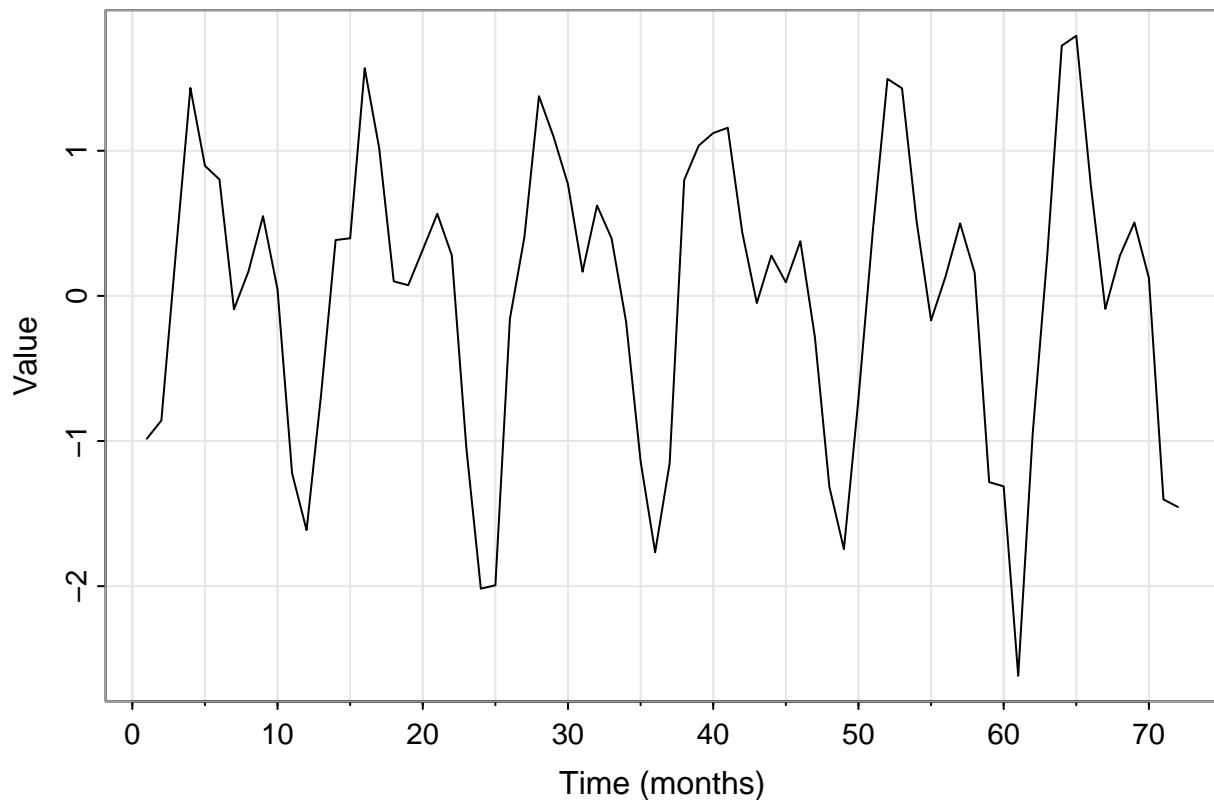
Pollutant 42602 – Cluster 6



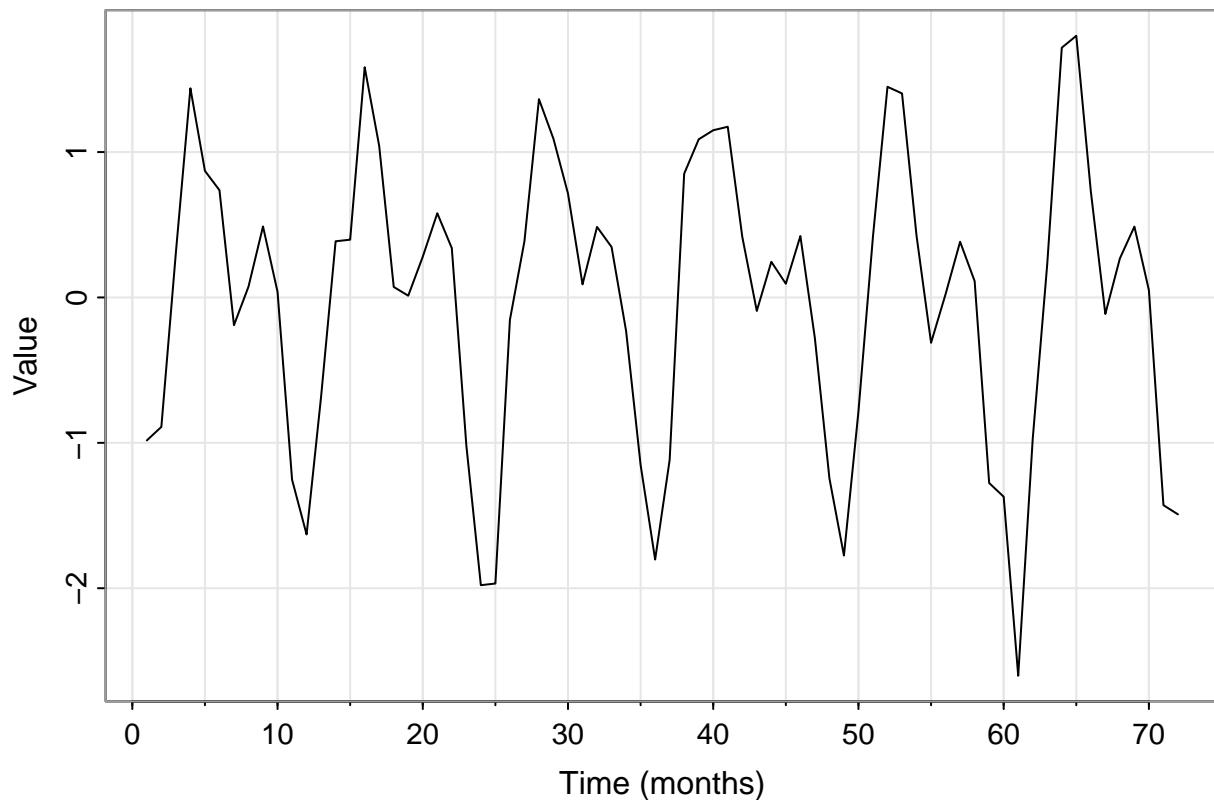
Pollutant 42602 – Cluster 7



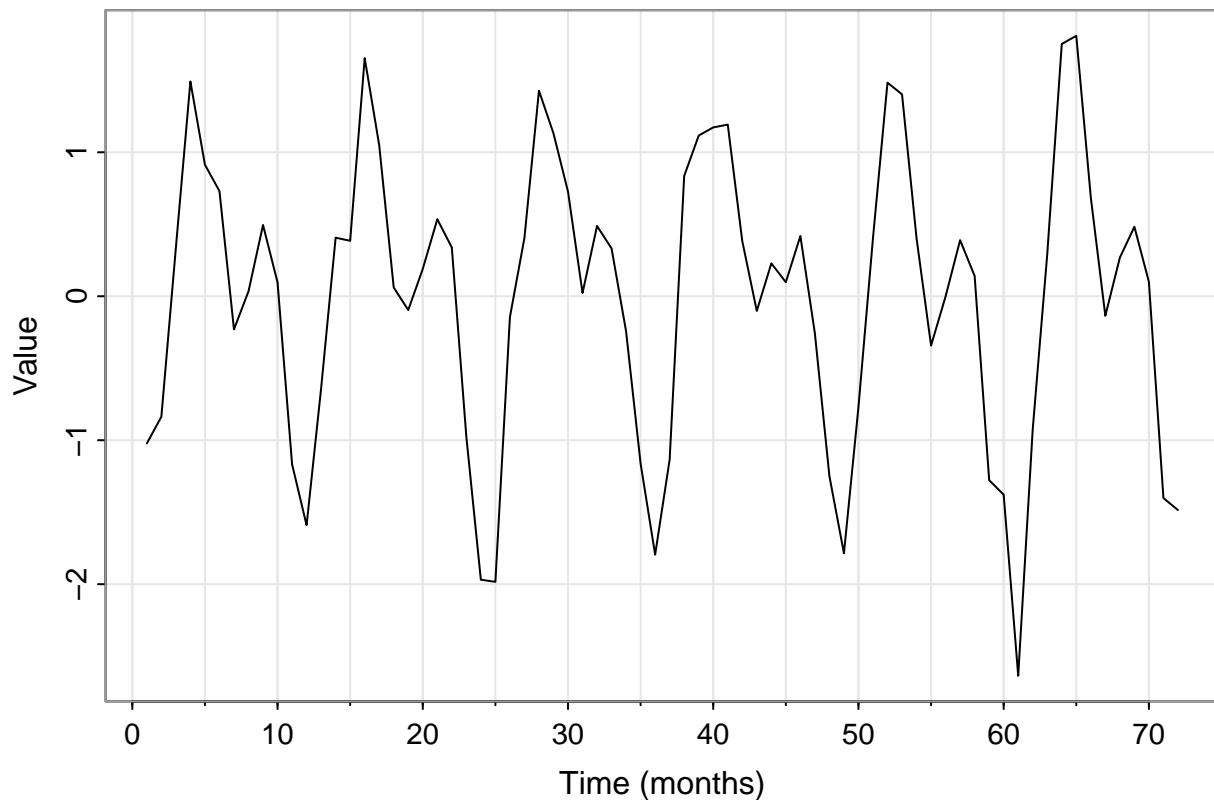
Pollutant 44201 – Cluster 1



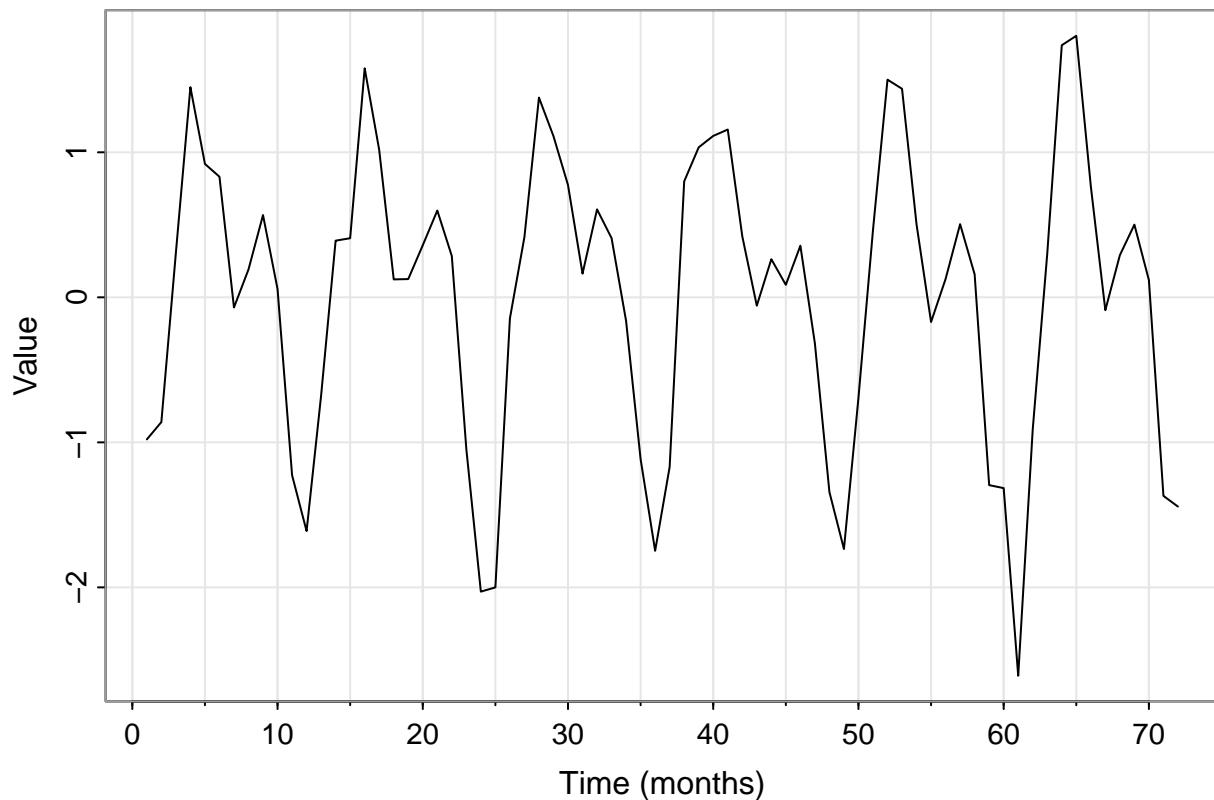
Pollutant 44201 – Cluster 2



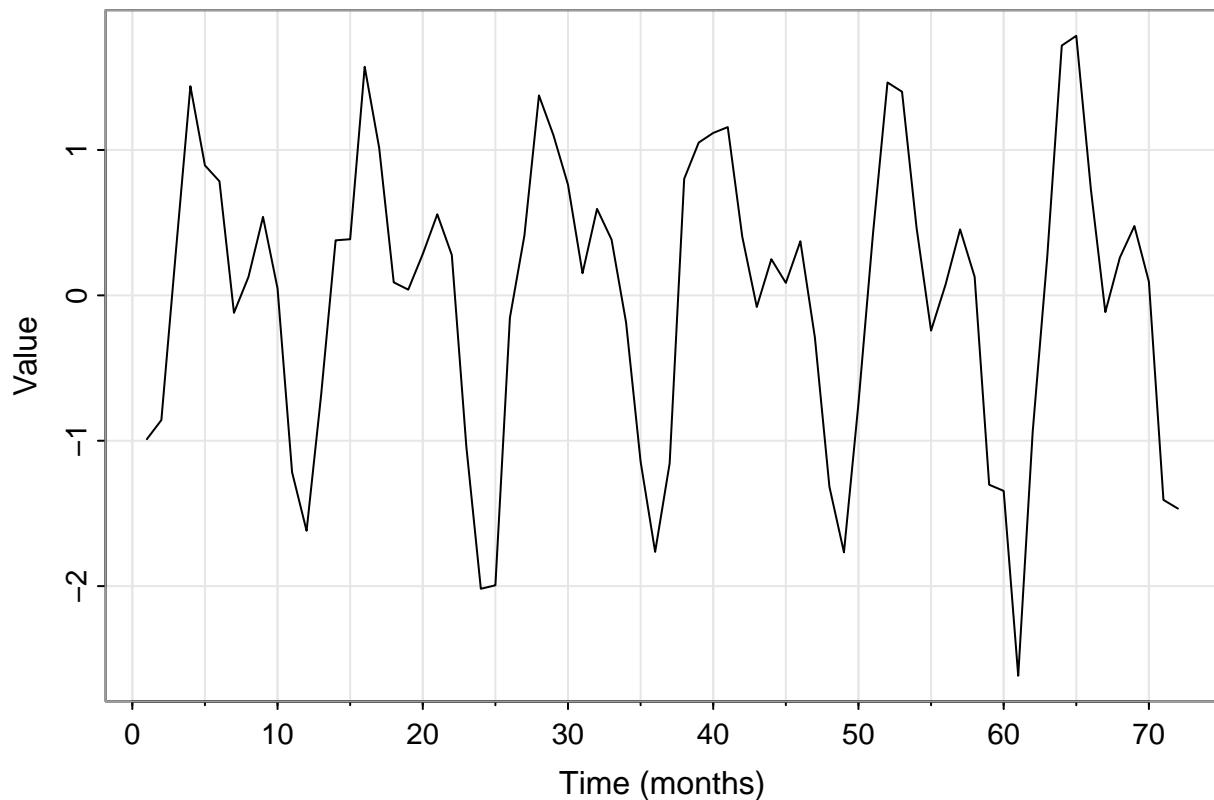
Pollutant 44201 – Cluster 3



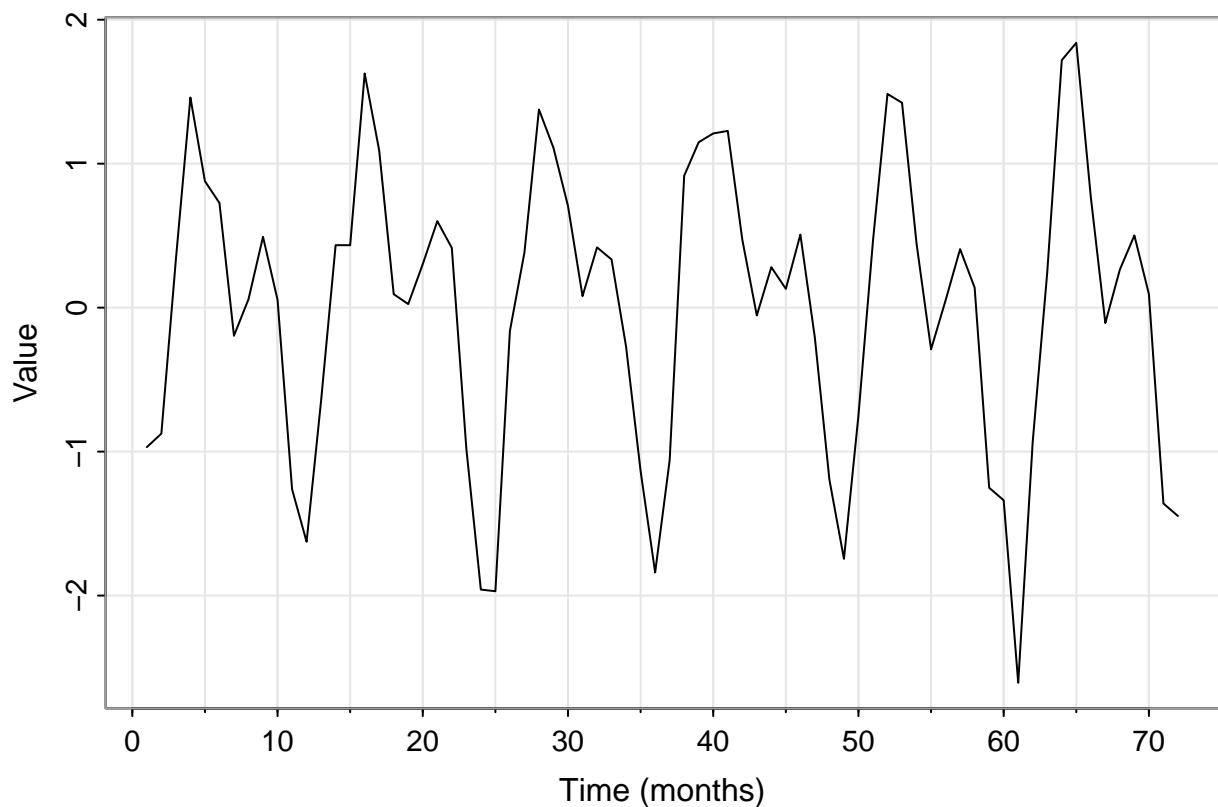
Pollutant 44201 – Cluster 4



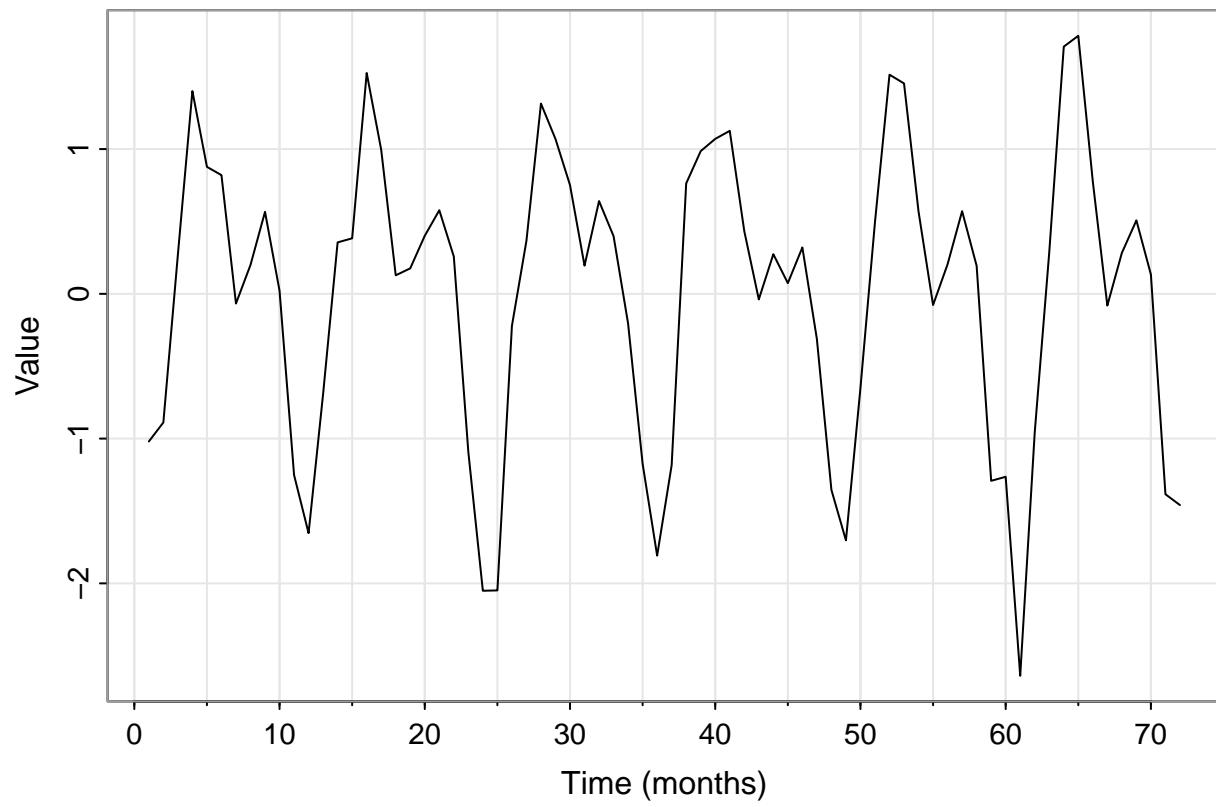
Pollutant 44201 – Cluster 5



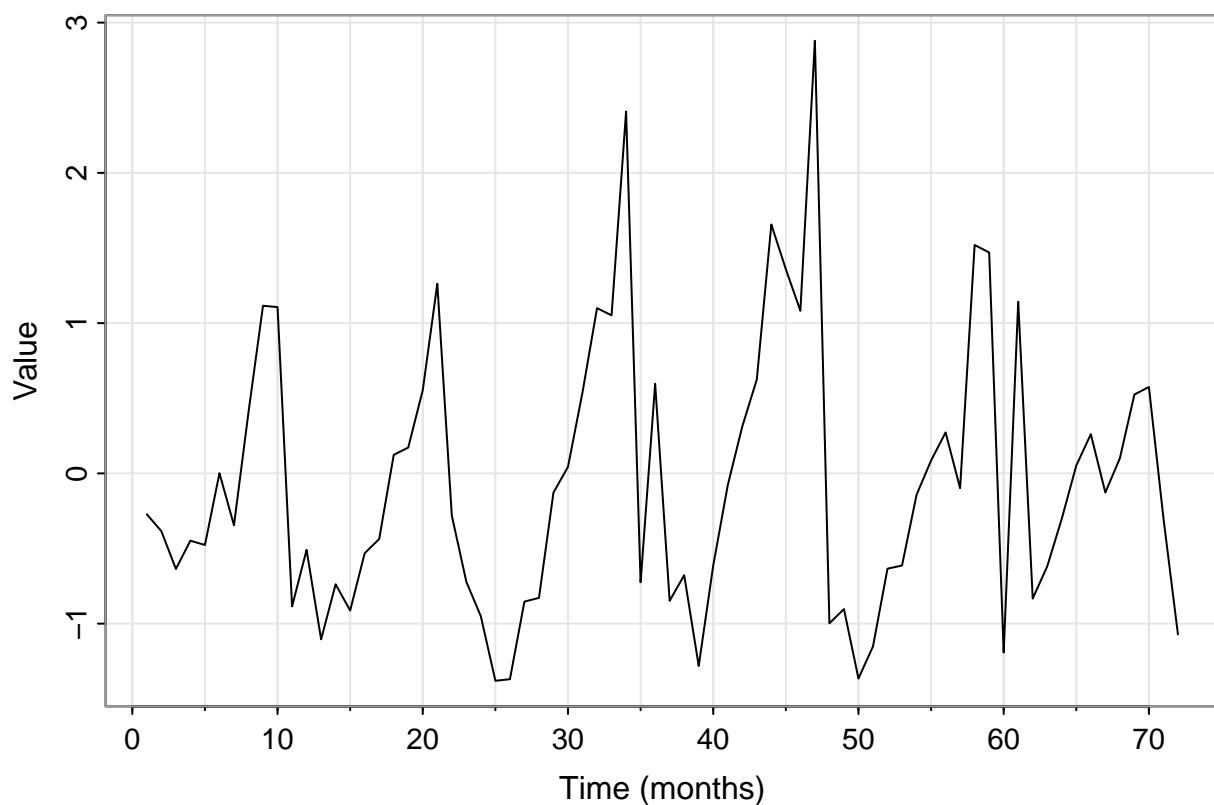
Pollutant 44201 – Cluster 6



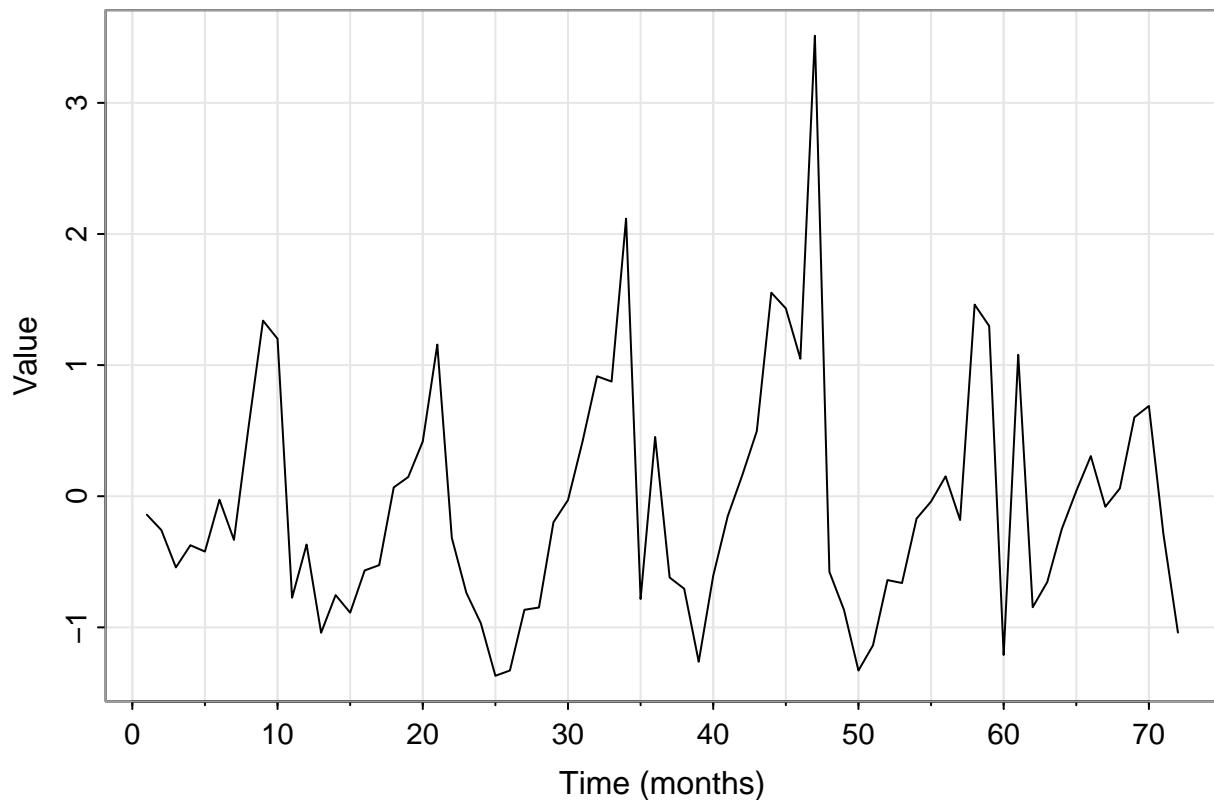
Pollutant 44201 – Cluster 7



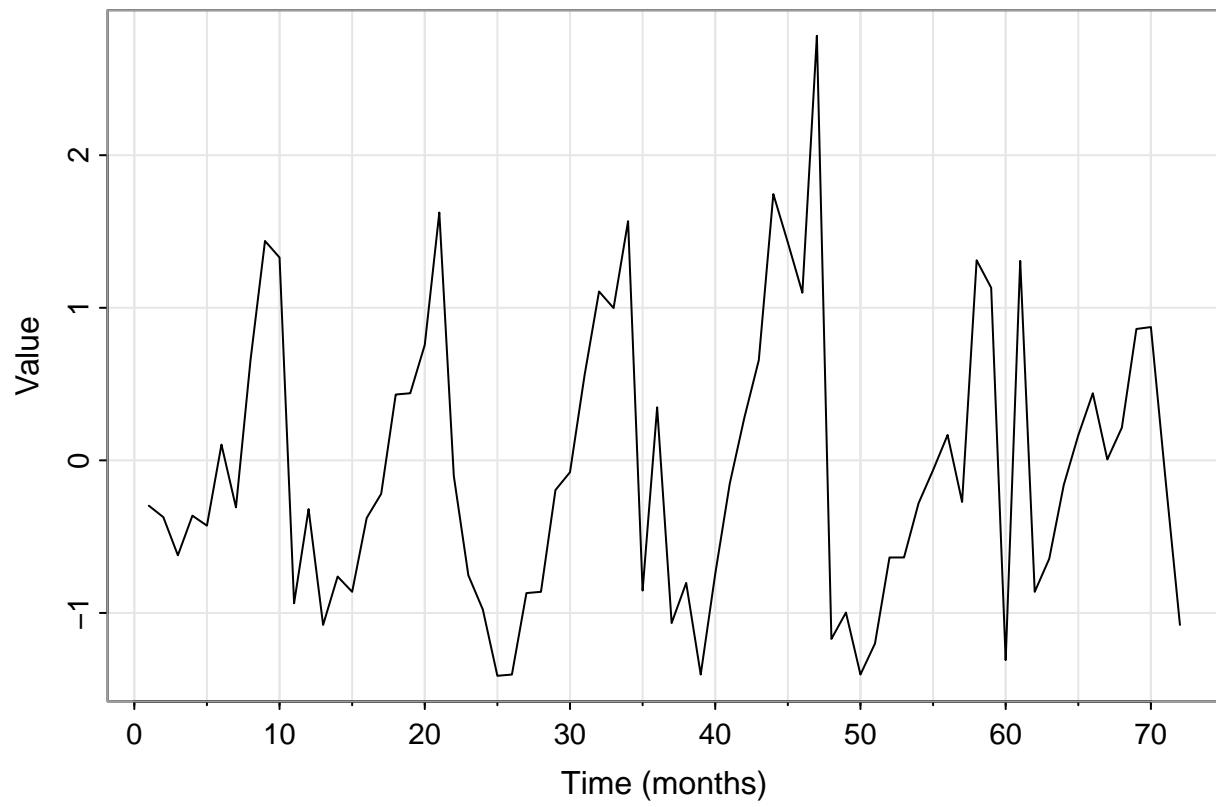
Pollutant 81102 – Cluster 1



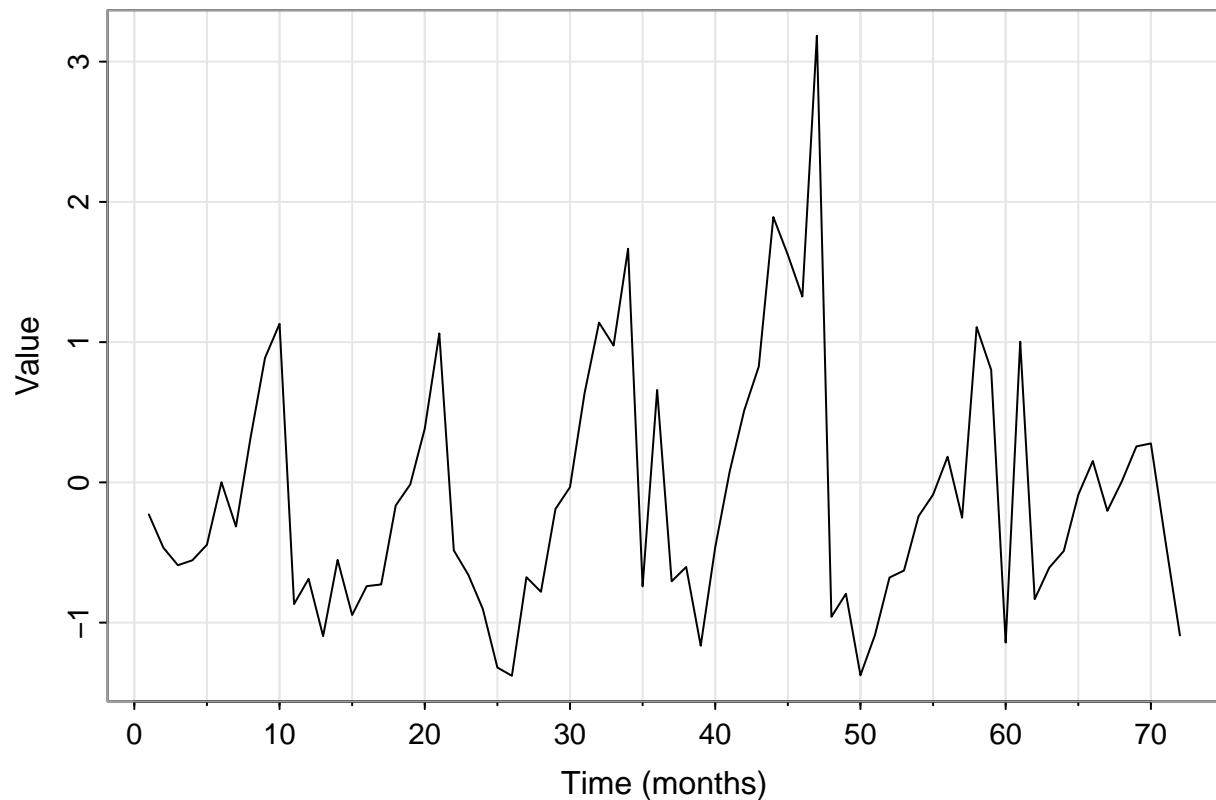
Pollutant 81102 – Cluster 2



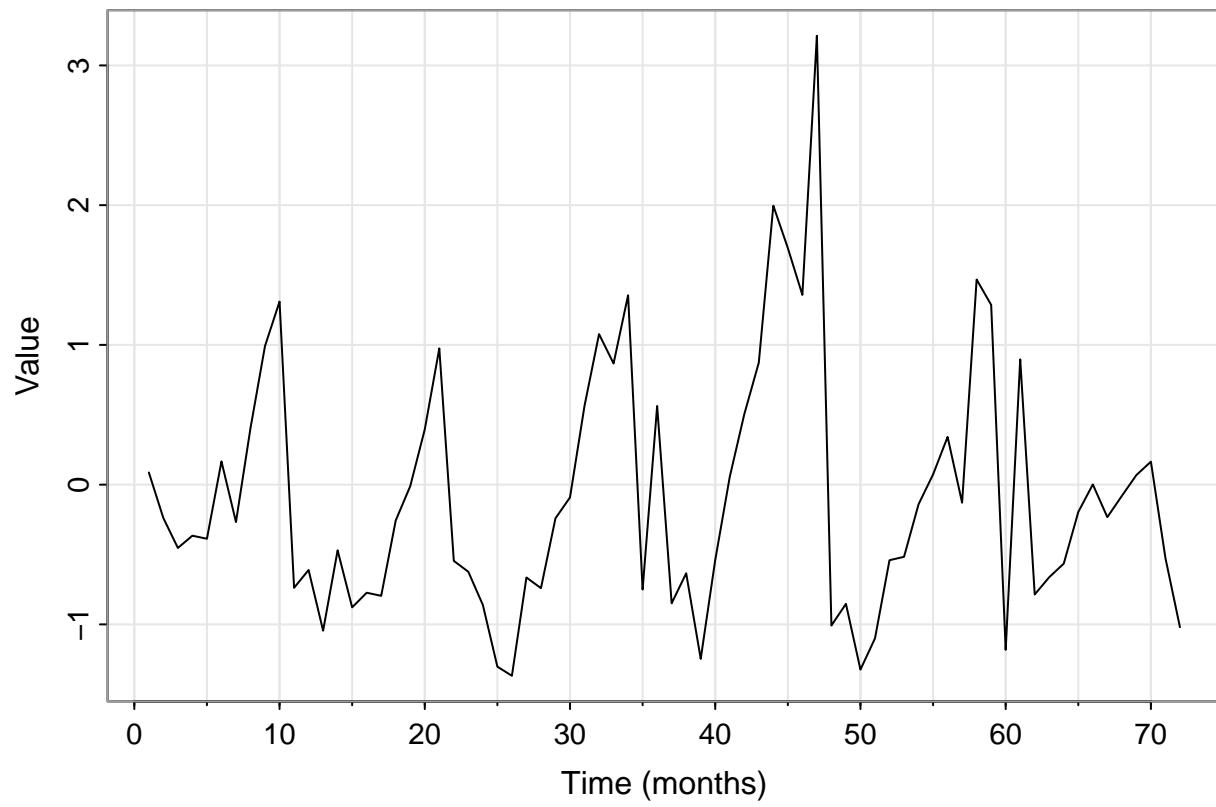
Pollutant 81102 – Cluster 3



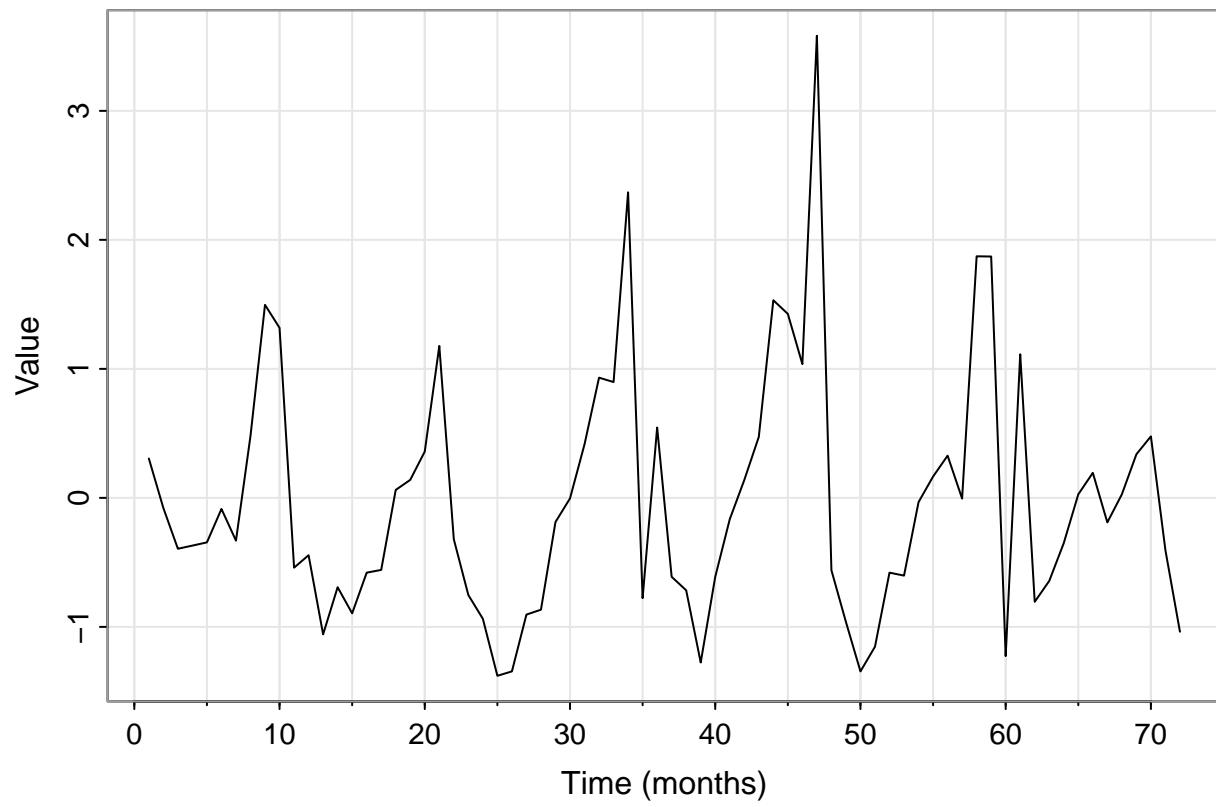
Pollutant 81102 – Cluster 4



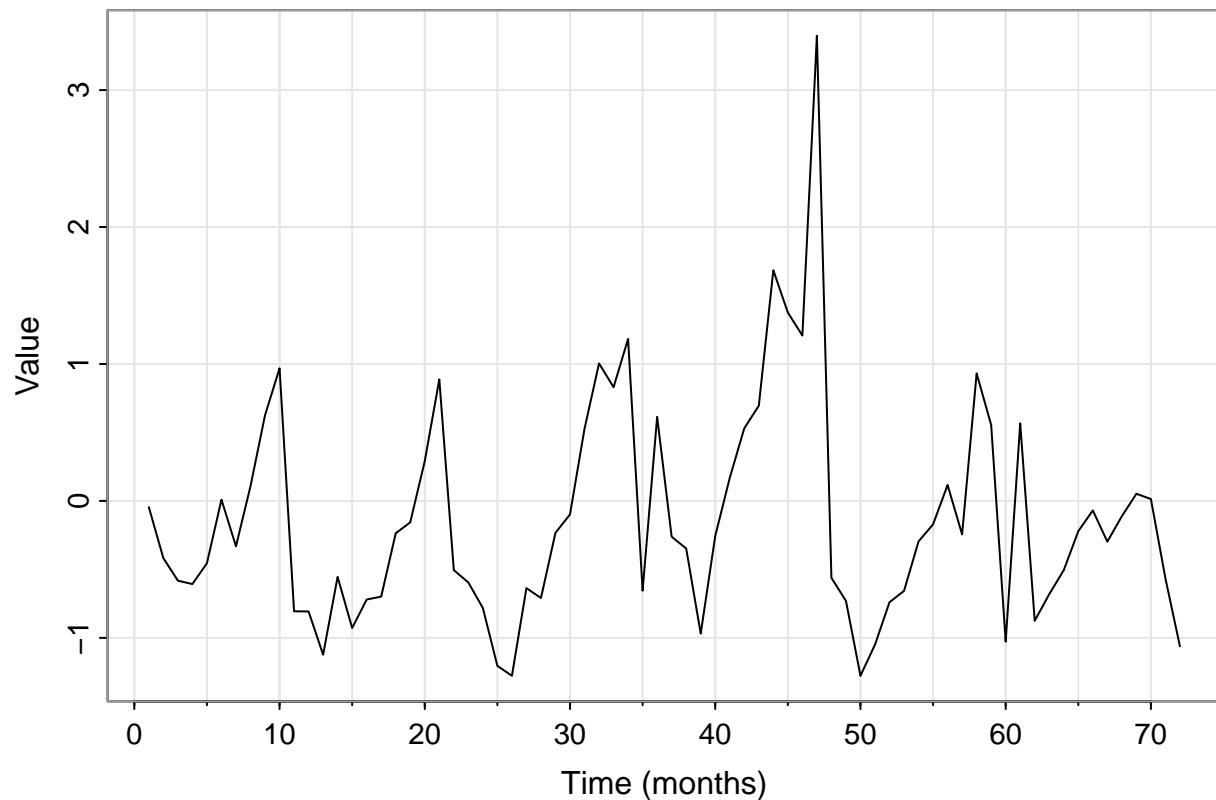
Pollutant 81102 – Cluster 5



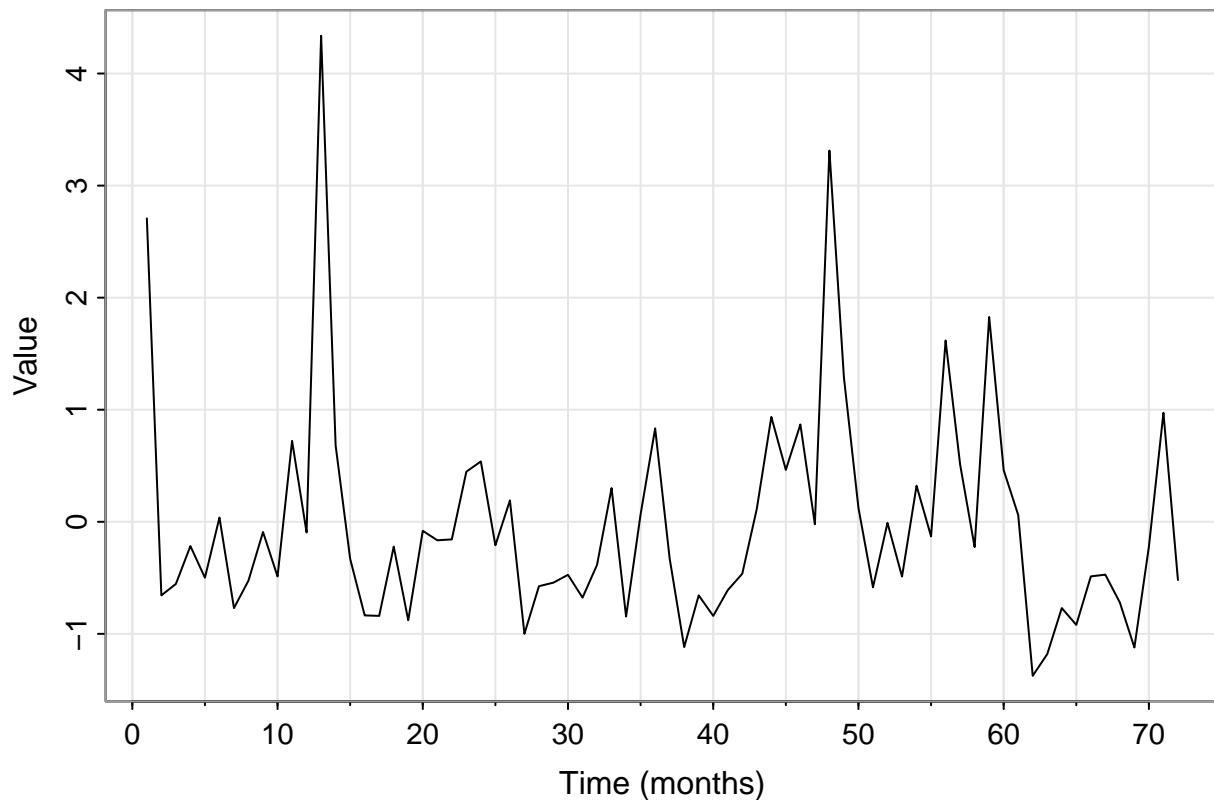
Pollutant 81102 – Cluster 6



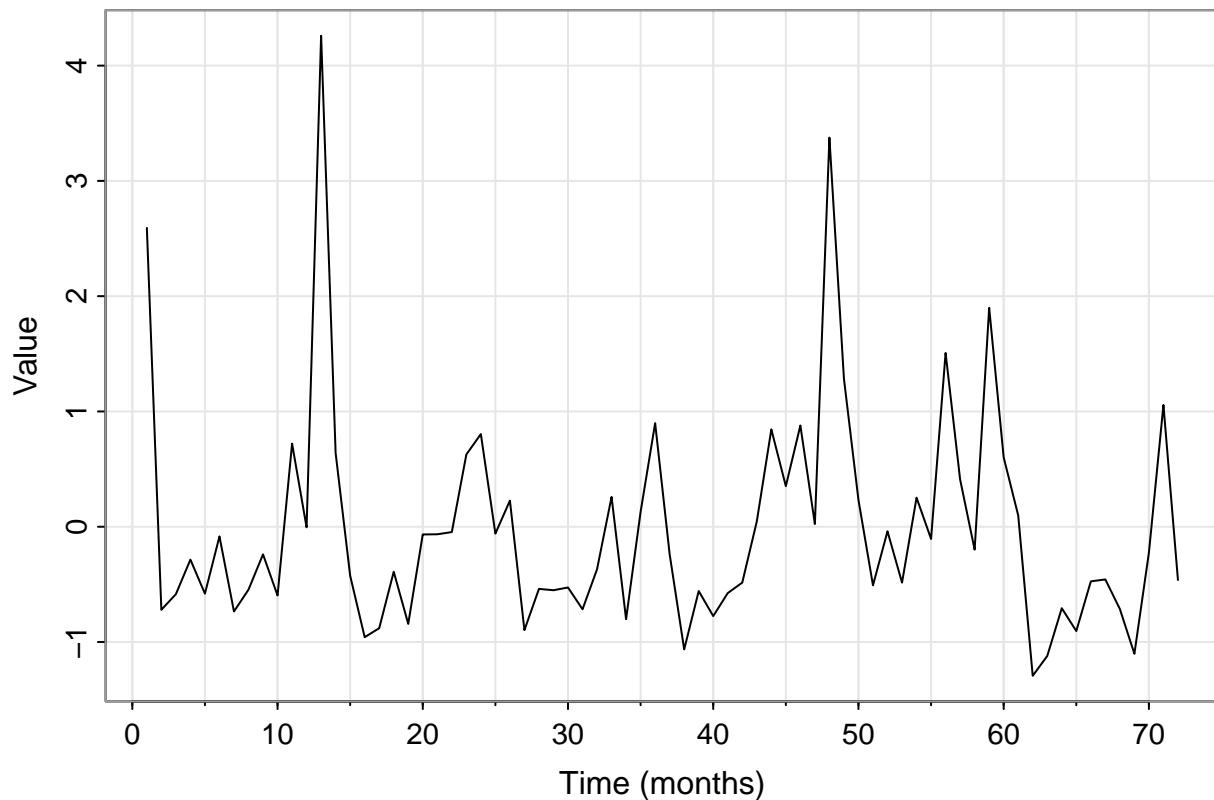
Pollutant 81102 – Cluster 7



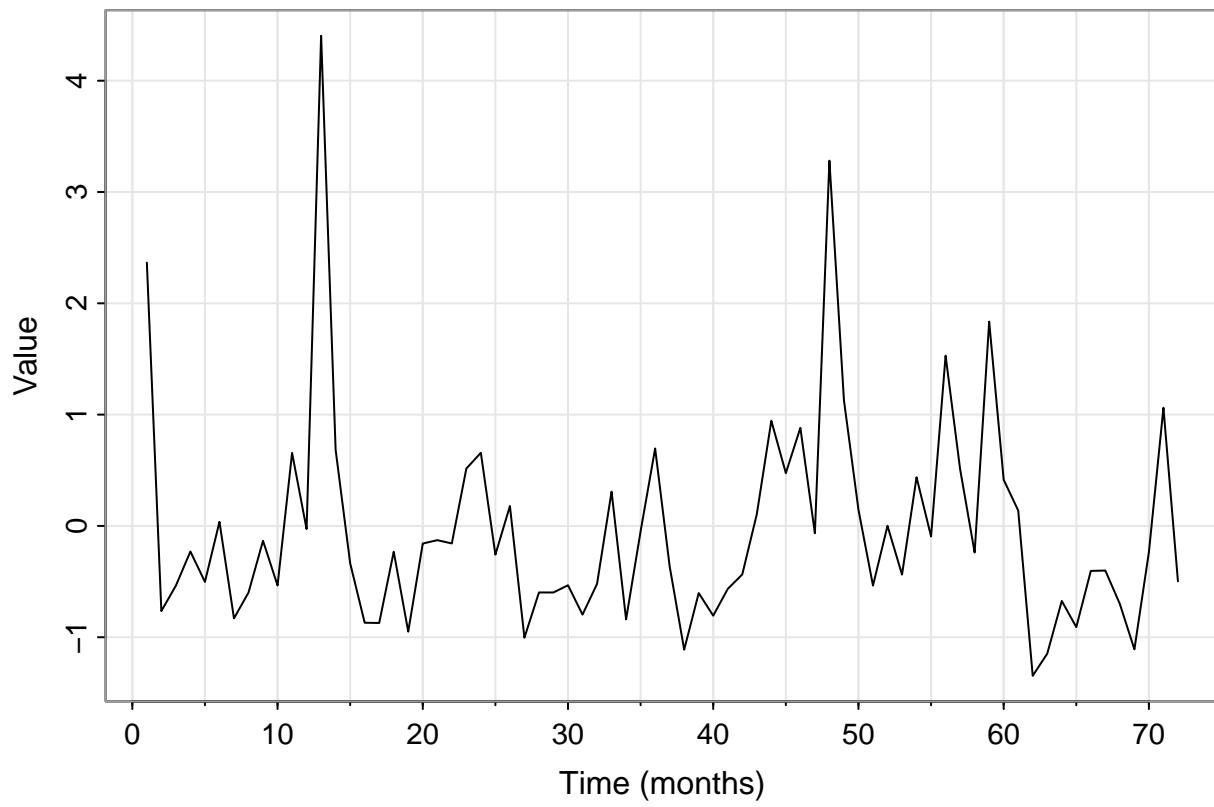
Pollutant 88101 – Cluster 1



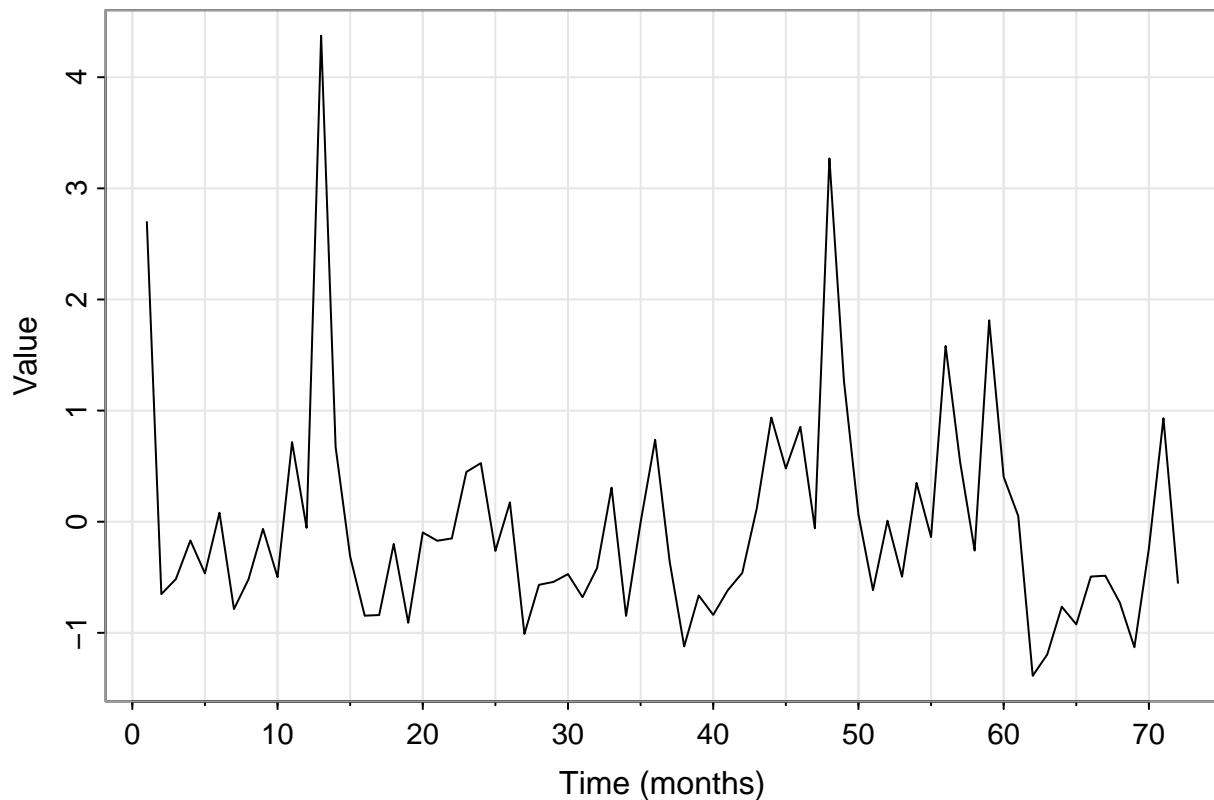
Pollutant 88101 – Cluster 2



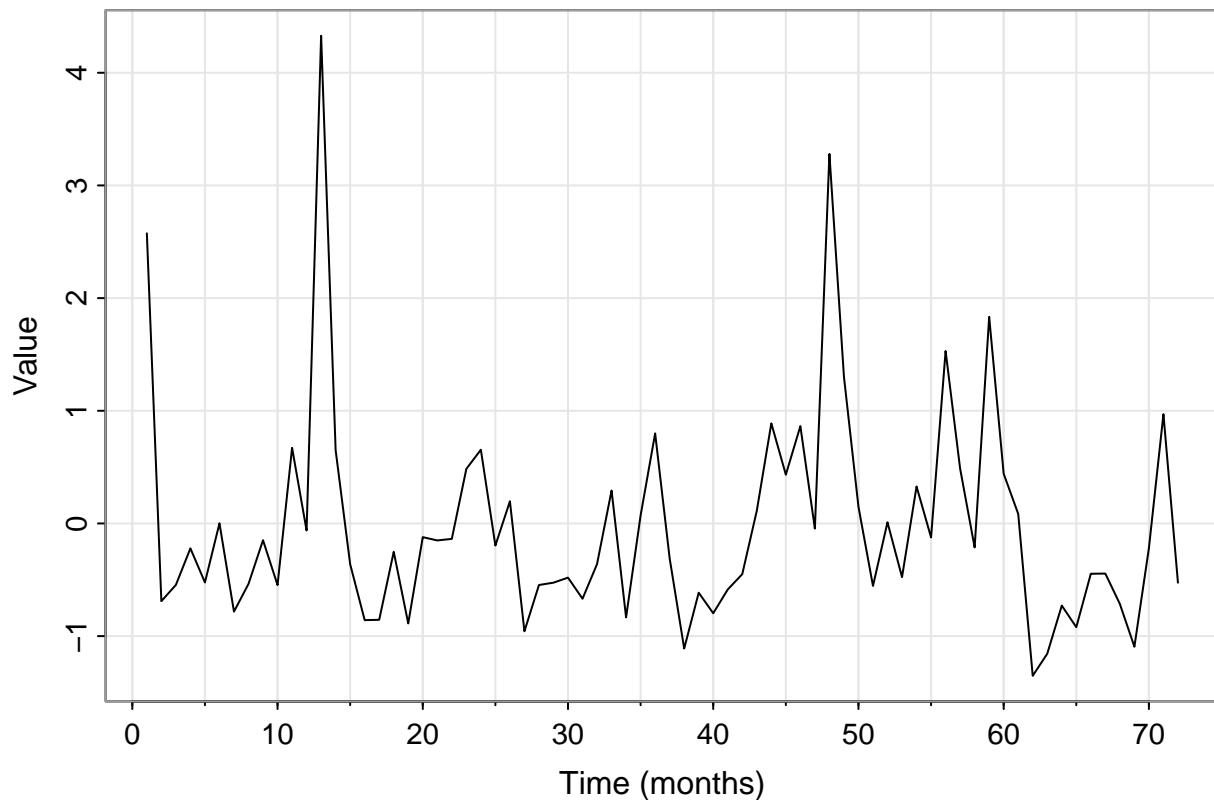
Pollutant 88101 – Cluster 3



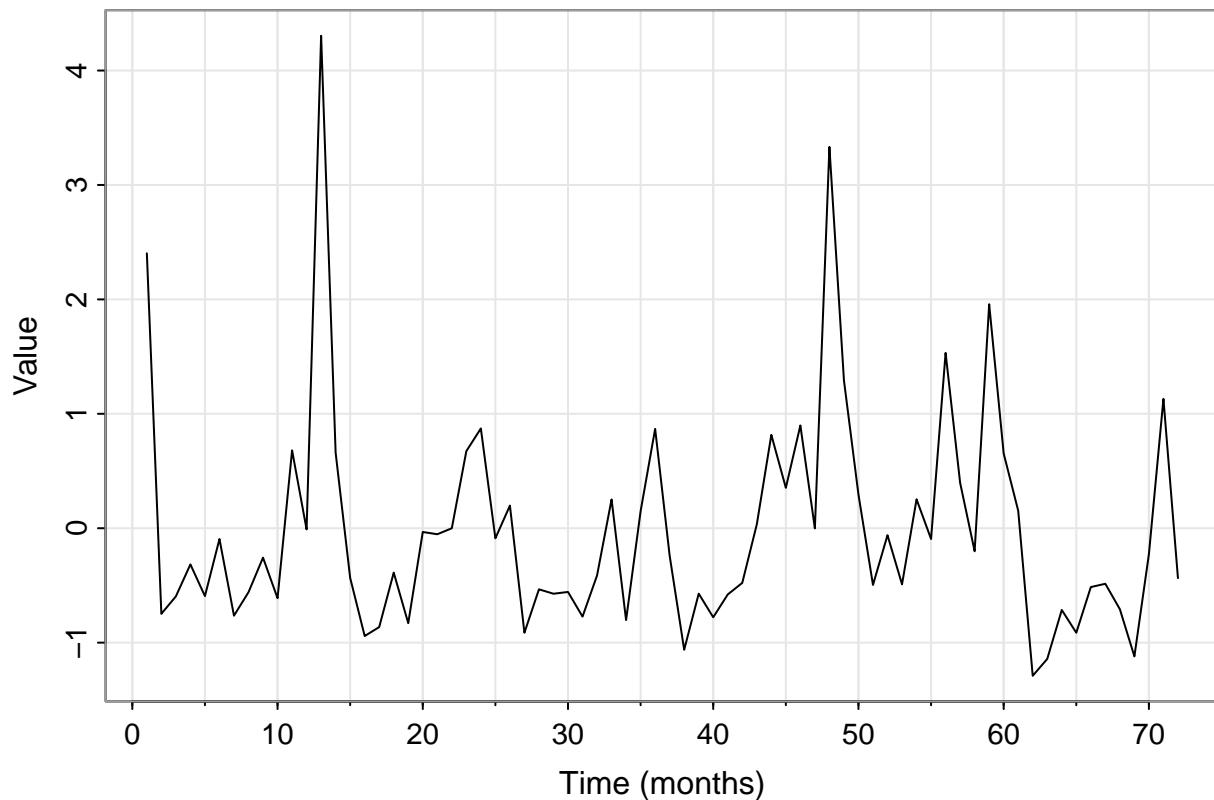
Pollutant 88101 – Cluster 4



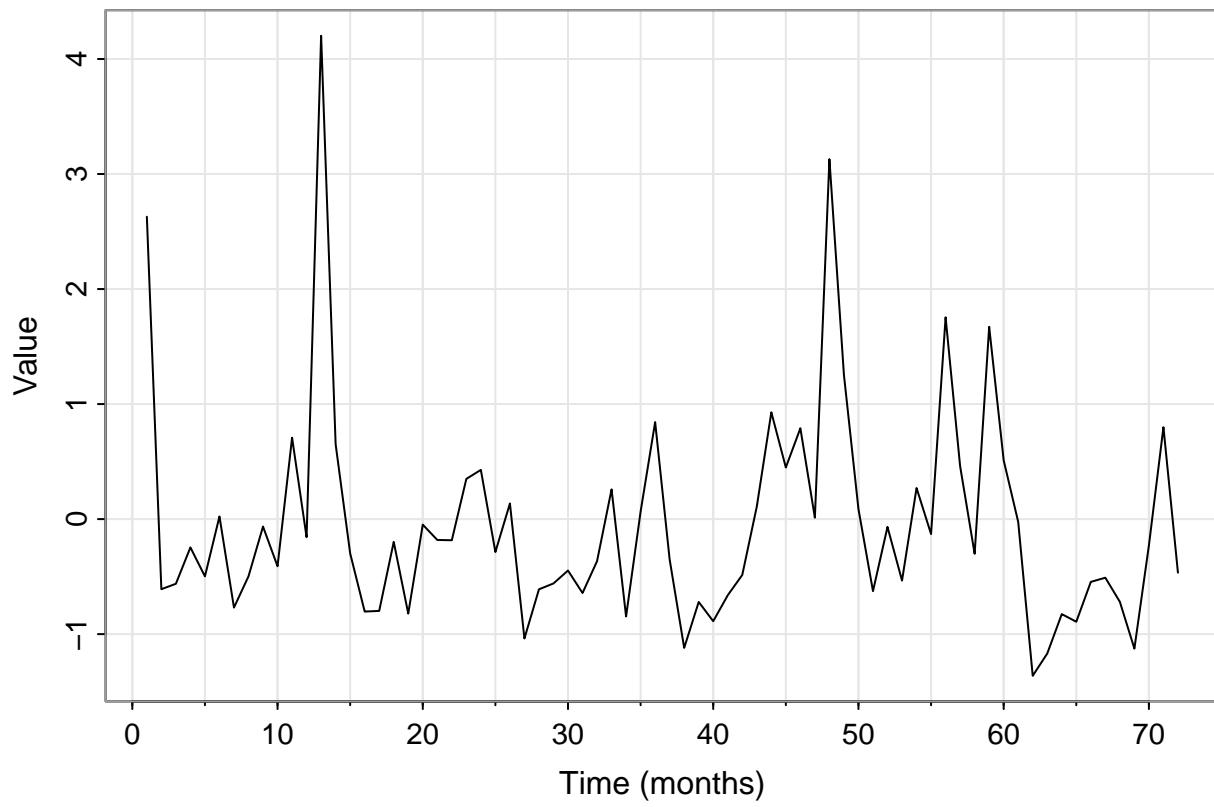
Pollutant 88101 – Cluster 5



Pollutant 88101 – Cluster 6

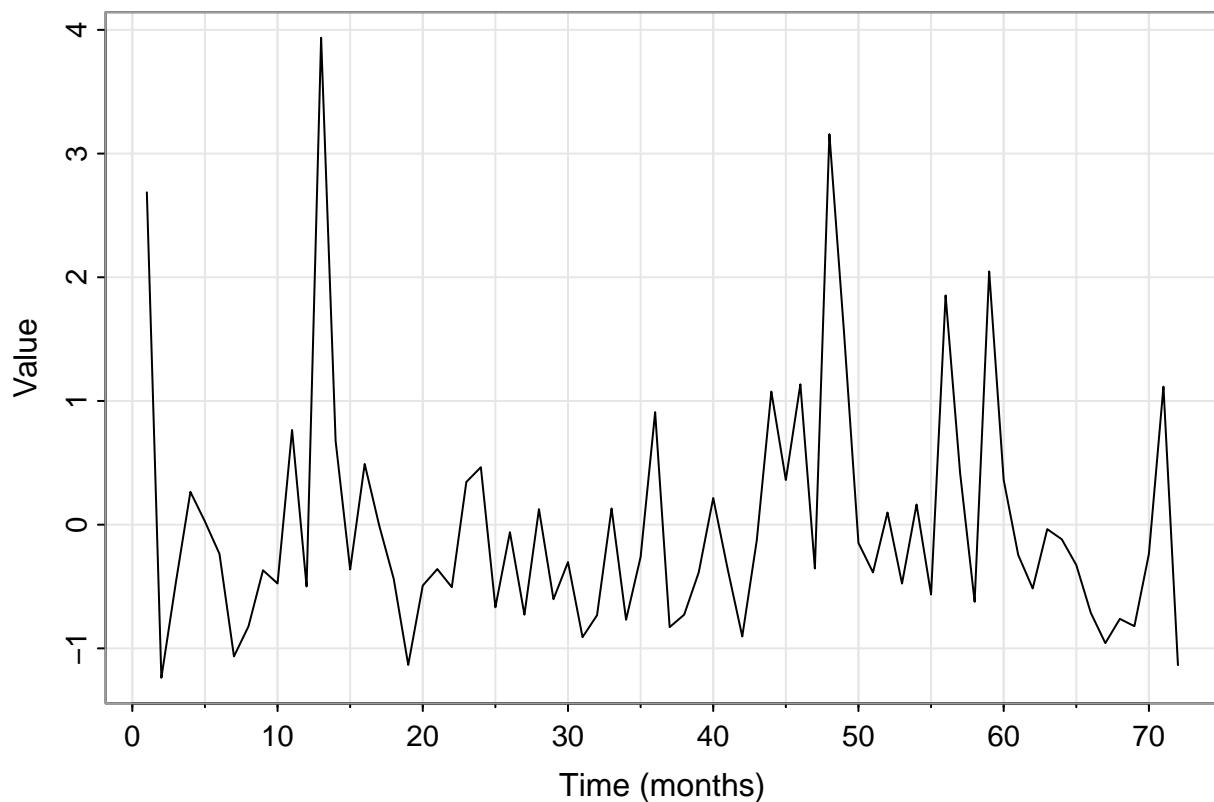


Pollutant 88101 – Cluster 7

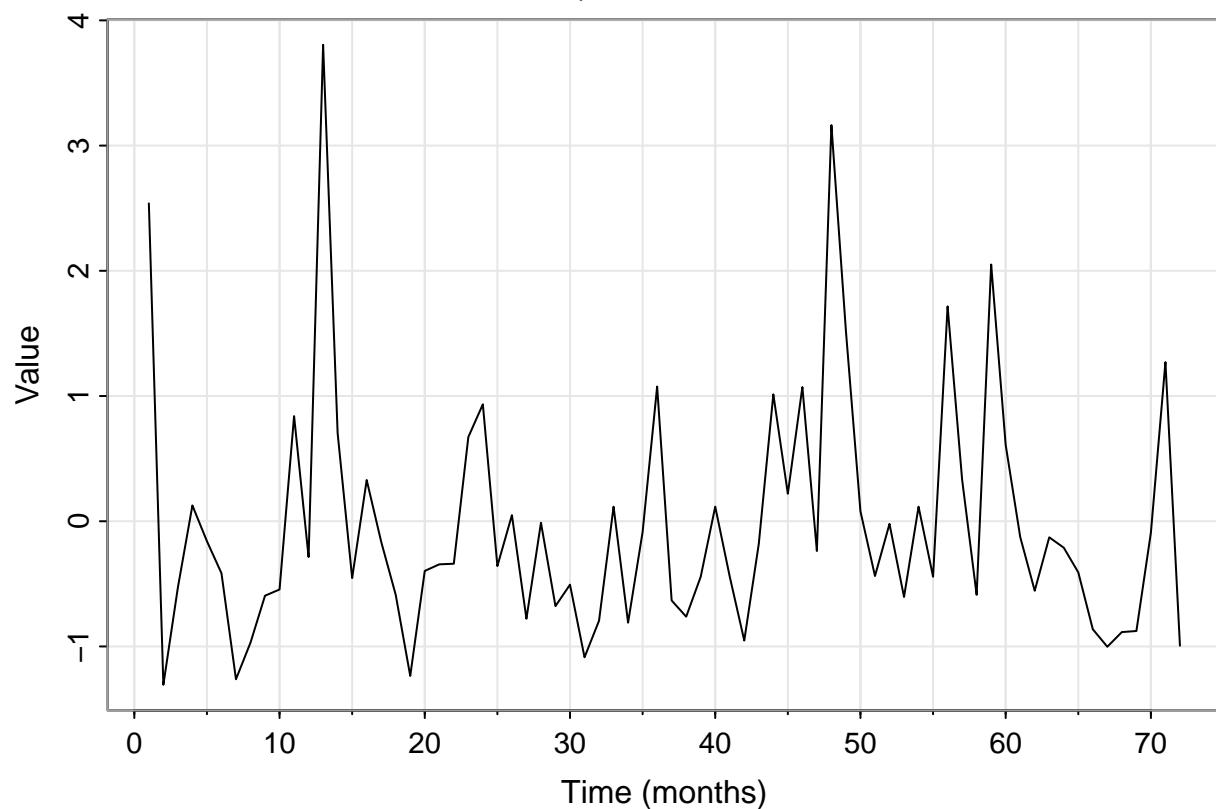


```
for (k in 1:num_clus){  
  AQI_clus_ts = final_EPA_agg_data %>% filter(Pollutant == "14129") %>% filter(Cluster == k)  
  title = sprintf("AQI - Cluster %.0f",k)  
  tsplot(AQI_clus_ts$AQI,main = title,xlab = "Time (months)",ylab = "Value")  
}
```

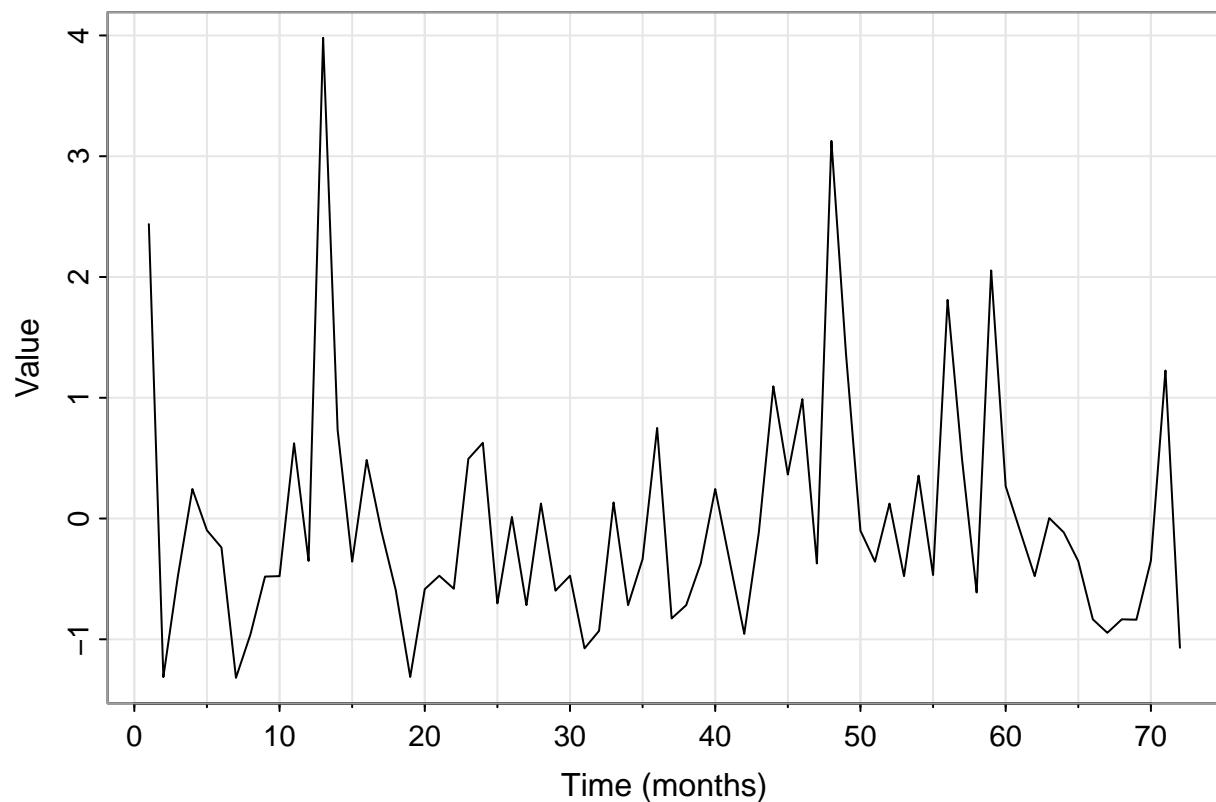
AQI – Cluster 1



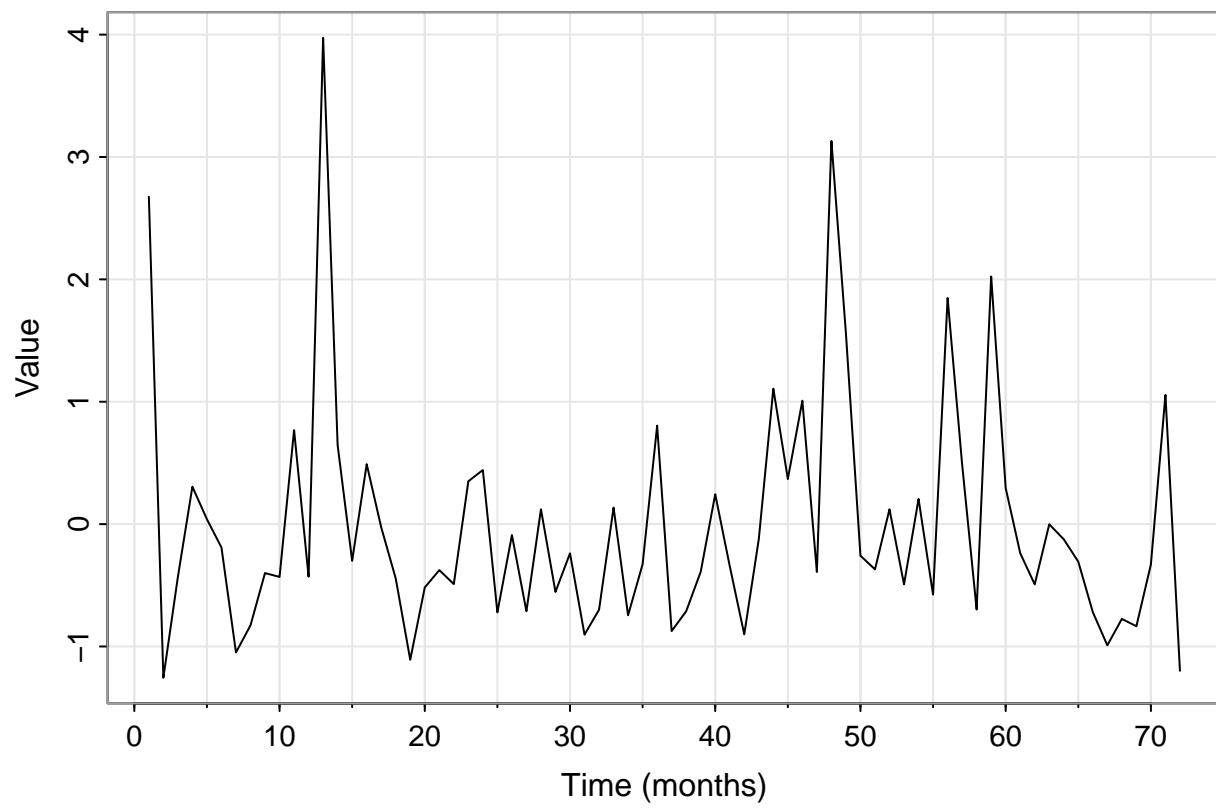
AQI – Cluster 2



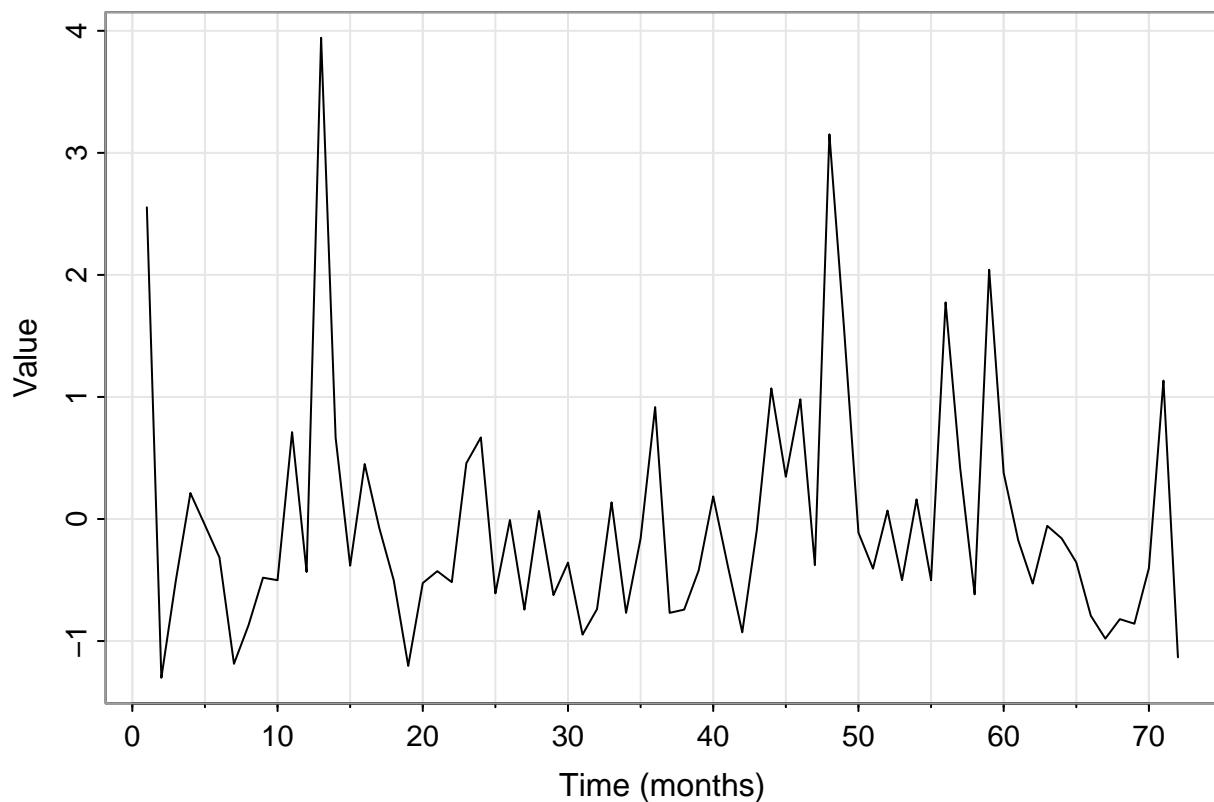
AQI – Cluster 3



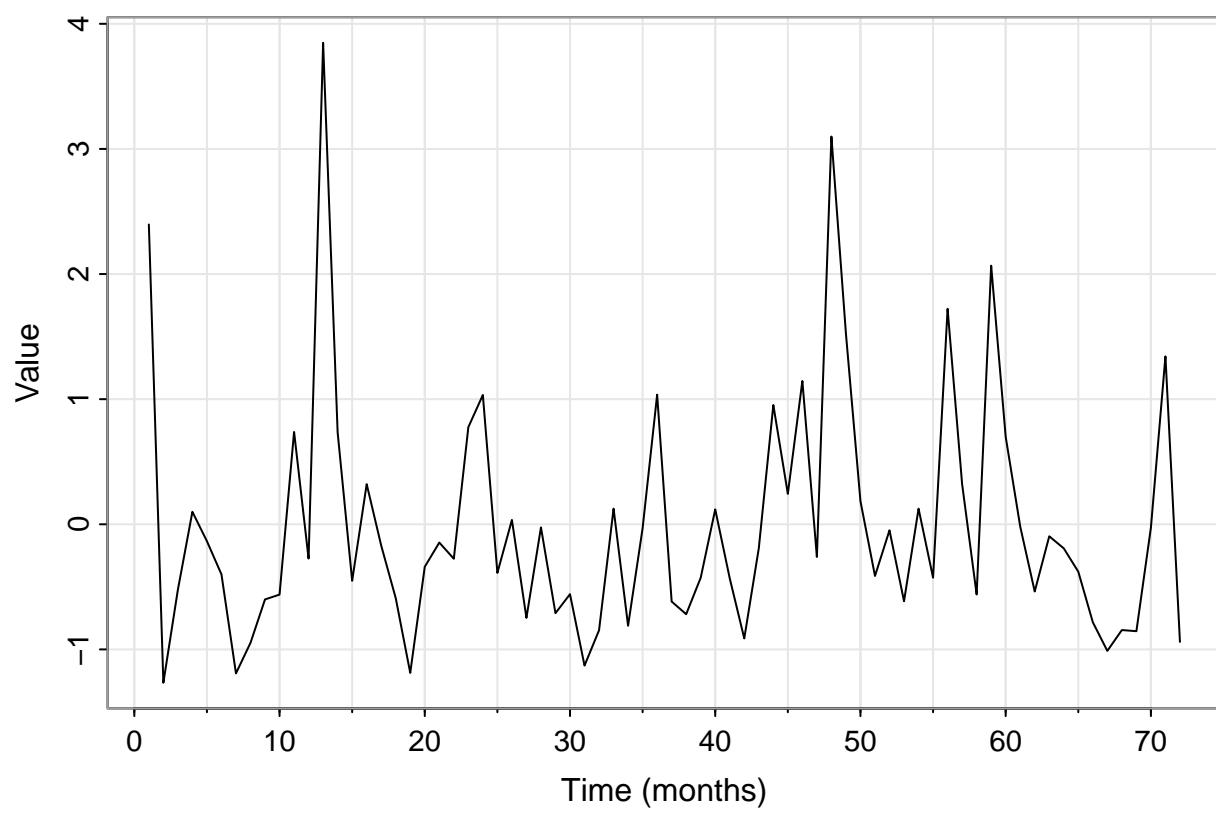
AQI – Cluster 4



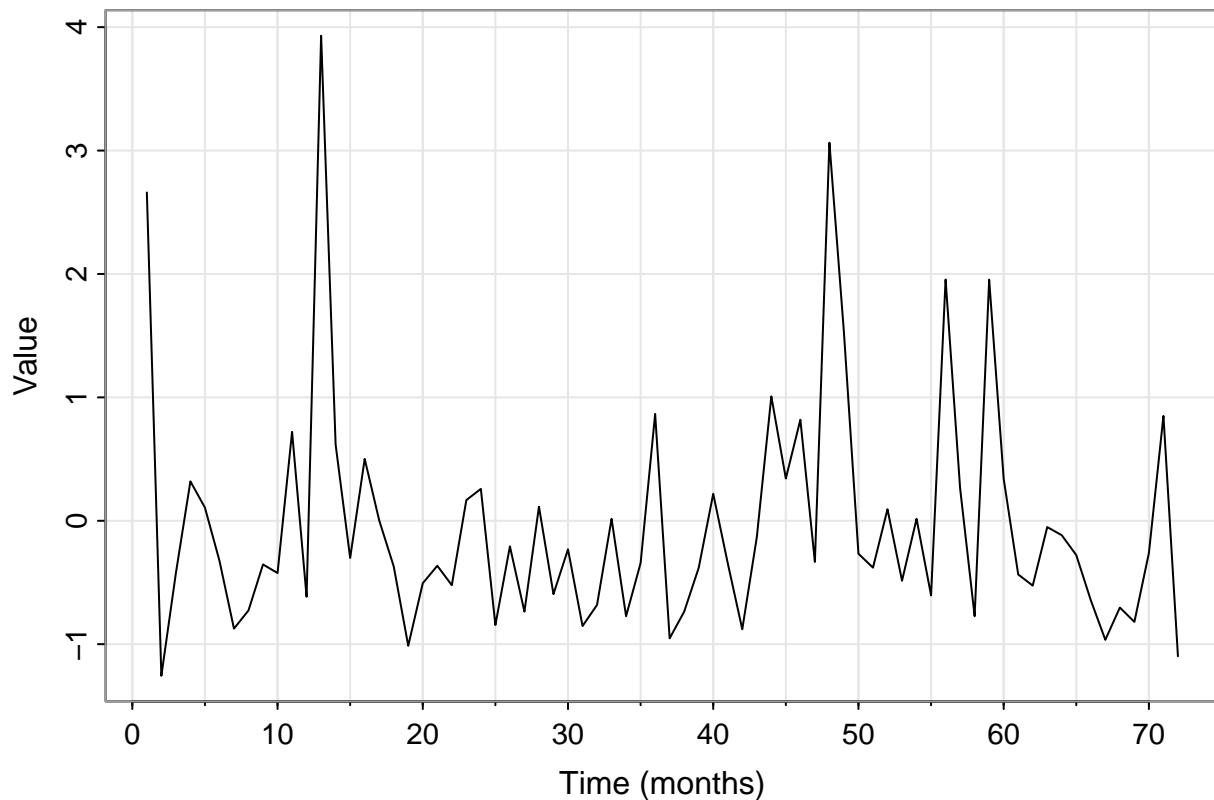
AQI – Cluster 5



AQI – Cluster 6



AQI – Cluster 7



Building covariate matrices to be used for gram matrix calculations

The kernel structures presented below reflect the findings of Chen et al. (2019) that distributed lags and interaction effects should be considered when evaluating the effects of air pollutants. Additionally, we wanted these lagged and joint effects to also vary in time. So, instead of calculating true autoregressive (AR), distributed lag (DL), and interaction structures, which assume stationarity, we construct localized kernels which have dynamic localized conditional correlations through time.

```
EPA_cluster_list = list()

cluster_EPA_data = function(cluster){
  EPA_clus_ts = matrix(nrow=72)

  for (i in pollutants$parametercodes.code){
    covariate_ts = final_EPA_agg_data %>% filter(Pollutant == i) %>% filter(Cluster == cluster)
    EPA_clus_ts = cbind(EPA_clus_ts,covariate_ts$Value)
  }
  EPA_clus_ts[,1] = covariate_ts$AQI
  colnames(EPA_clus_ts) = c("AQI", "Lead", "CO", "SO2", "NO2", "O3", "PM10", "PM2.5")

  return(EPA_clus_ts)
}

for (i in 1:num_clus){
  EPA_cluster_list[[i]] = cluster_EPA_data(i)
```

```
}
```

The first step is to decompose our covariate time series into its trend, seasonal, and residual components because we want to learn each kernel on a different component.

```
decompose_ts = function(EPA_ts){  
  S_scaled = EPA_ts  
  colnames(S_scaled) = c("AQI", "Lead", "CO", "SO2", "NO2", "O3", "PM10", "PM2.5")  
  S_scaled_ts = ts(S_scaled, frequency = 12)  
  
  # S_decomposed = decompose(S_scaled_ts)  
  
  ts_decomposed <- lapply(colnames(S_scaled_ts), function(x) {stl(S_scaled_ts[, x], s.window = "periodic")})  
  names(ts_decomposed) = colnames(S_scaled_ts)  
  
  S_seasonal = ts_decomposed[[1]]$time.series[, 1]  
  S_trend = ts_decomposed[[1]]$time.series[, 2]  
  S_random = ts_decomposed[[1]]$time.series[, 3]  
  
  for (i in 2:8){  
    S_seasonal = cbind(S_seasonal, ts_decomposed[[i]]$time.series[, 1])  
    S_trend = cbind(S_trend, ts_decomposed[[i]]$time.series[, 2])  
    S_random = cbind(S_random, ts_decomposed[[i]]$time.series[, 3])  
  }  
  
  colnames(S_trend) = colnames(S_scaled_ts)  
  colnames(S_seasonal) = colnames(S_scaled_ts)  
  colnames(S_random) = colnames(S_scaled_ts)  
  
  S_DL = S_seasonal + S_random  
  colnames(S_DL) = colnames(S_scaled_ts)  
  
  S_random_int = S_random[12:72,]  
  S_random = S_random[13:72,]  
  S_seasonal = S_seasonal[13:72,]  
  S_trend = S_trend[13:72,]  
  S_DL = S_DL[13:72,]  
  
  S_DL = data.frame(S_DL)  
  S_DL2 = matrix(nrow=60)  
  dl = c(3, 6, 12)  
  col_num = 2  
  for (i in dl){  
    for (j in 1:ncol(S_DL)){  
      extract = S_DL[(72-59-i):(72-i), j]  
      S_DL2 = cbind(S_DL2, extract)  
      colnames(S_DL2)[col_num] = sprintf("B%1.0f-%s", i, colnames(S_DL)[j])  
      col_num = col_num+1  
    }  
  }  
  
  S_DL2 = S_DL2[,-1]  
  S_DL_final = cbind(S_DL[13:72,], S_DL2)
```

```

W = matrix(nrow=(nrow(S_random))^2)
num_cols = ncol(S_random)
# num_cols = ncol(S_DL_final) #for now, just calculate interaction pairs for actual covariates
col_num = 2

for (i in 1:num_cols){
  for (j in 1:num_cols){
    interaction_col = kronecker(S_random[,i],S_random[,j]) #replace S_random with S_DL_final for DL i
    W = cbind(W,interaction_col)

    colnames(W)[col_num] = sprintf("%sx%s",colnames(S_scaled)[i],colnames(S_scaled)[j])
    col_num = col_num+1
  }
}
W = W[,-1]

row1 = c()

for (k in 1:ncol(S_random_int)){
  row1 = c(row1,S_random_int[2,k]*S_random_int[1,])
}

W2 = rbind(as.numeric(row1),W)
W2 = W2[,-seq(1,64,by=9)] #need to change if we include DL interactions

list = list(S_random,S_random_int,S_seasonal,S_DL,
            S_DL2,S_DL_final,S_trend,W2)
names(list) = c("S_random","S_random_int","S_seasonal","S_DL",
               "S_DL2","S_DL_final","S_trend","W2")
return(list)
}

decompose_clus1 = decompose_ts(EPA_cluster_list[[1]])
decompose_clus2 = decompose_ts(EPA_cluster_list[[2]])
decompose_clus3 = decompose_ts(EPA_cluster_list[[3]])
decompose_clus4 = decompose_ts(EPA_cluster_list[[4]])
decompose_clus5 = decompose_ts(EPA_cluster_list[[5]])
decompose_clus6 = decompose_ts(EPA_cluster_list[[6]])
decompose_clus7 = decompose_ts(EPA_cluster_list[[7]])

decomposed_cluster_data = list(decompose_clus1,decompose_clus2,decompose_clus3,
                                decompose_clus4,decompose_clus5,decompose_clus6,
                                decompose_clus7)

S_random_all = cbind(decompose_clus1$$S_random,decompose_clus2$$S_random,
                      decompose_clus3$$S_random,decompose_clus4$$S_random,
                      decompose_clus5$$S_random,decompose_clus6$$S_random,
                      decompose_clus7$$S_random)

S_DL_all = cbind(cbind(decompose_clus1$$S_DL,decompose_clus2$$S_DL,
                       decompose_clus3$$S_DL,decompose_clus4$$S_DL,
                       decompose_clus5$$S_DL,decompose_clus6$$S_DL,
                       decompose_clus7$$S_DL))

```

```

W2_all = cbind(cbind(decompose_clus1$W2, decompose_clus2$W2,
                      decompose_clus3$W2, decompose_clus4$W2,
                      decompose_clus5$W2, decompose_clus6$W2,
                      decompose_clus7$W2))

```

Calculating autoregressive structure

Using the residual component of each time series, we construct a kernel that calculates the autocorrelation at one lag for all time points

Linear time invariant approach:

Let $\sigma_{AR}^2 = \gamma(0)$ be the variance of one of our time series, we can find ACVF and ACF from Yule-Walker as

$$\gamma(k) = a_1\gamma(k-1) + a_2\gamma(k-2) + \dots + a_p\gamma(k-p)$$

$$\rho(k) = \frac{\gamma(k)}{\gamma(0)} = a_1\rho(k-1) + a_2\rho(k-2) + \dots + a_p\rho(k-p)$$

```

AR_invariant_list = list()

for (c in 1:num_clus){
  #Grab S_random data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  S_random_clus = cluster_data$S_random

  ar.corr.values = c()
  ar.cov.values = c()

  for (i in 1:ncol(S_random_clus)){
    var = var(S_random_clus[,i])
    fit.ar = ar(S_random_clus[,i], order.max = 1, aic = FALSE, method = "yule-walker")

    corr.ar1 = fit.ar$ar
    cov.ar1 = fit.ar$ar * var

    ar.corr.values = c(ar.corr.values,corr.ar1)
    ar.cov.values = c(ar.cov.values,cov.ar1)
  }

  for (j in 1:ncol(S_random_clus)){
    AR_invariant_covmatrix = diag(nrow(S_random_clus))

    AR_invariant_covmatrix[row(AR_invariant_covmatrix) == col(AR_invariant_covmatrix) - 1] = ar.cov.values
    AR_invariant_covmatrix[row(AR_invariant_covmatrix) == col(AR_invariant_covmatrix) + 1] = ar.cov.values
  }

  AR_invariant = diag(nrow(S_random_clus))
  AR_invariant[row(AR_invariant) == col(AR_invariant) - 1] = (1/length(ar.cov.values))*sum(ar.cov.values)
  AR_invariant[row(AR_invariant) == col(AR_invariant) + 1] = (1/length(ar.cov.values))*sum(ar.cov.values)

  # corrplot(AR_invariant, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1)
  #           title = "Invariant AR 1 Covariance Structure")

  matrix_heatmap(AR_invariant,title = "Invariant AR 1 Covariance Structure")
}

```

```

    AR_invariant_list[[c]] = AR_invariant
}

K_AR_invariant = matrix(0,nrow=60,ncol=60)

for(i in 1:num_clus){
  K_AR_invariant = K_AR_invariant + ((1/num_clus)*AR_invariant_list[[i]])
}

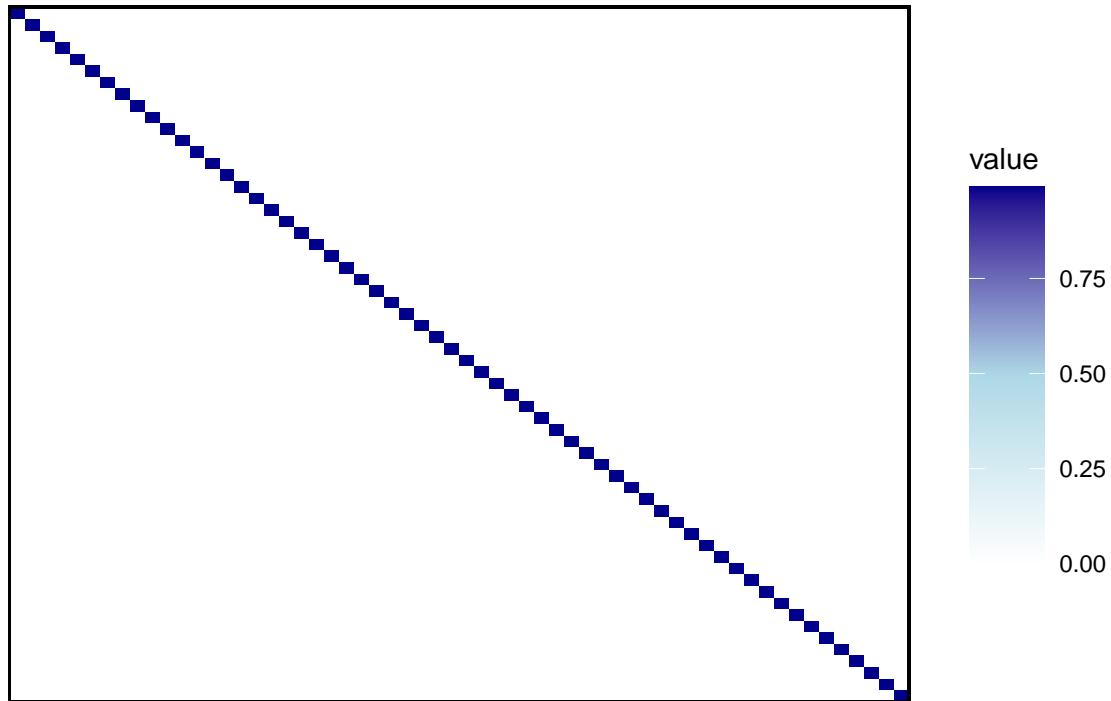
# corrplot(K_AR_invariant, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1)
#           title = "Invariant AR 1 Covariance Structure")

matrix_heatmap(K_AR_invariant,title = "Invariant AR 1 Covariance Structure")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

Invariant AR 1 Covariance Structure



```

K_AR_cluster = list()
K_AR_periodic_cluster = list()

for (c in 1:num_clus){

  #Grab S_random data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  S_random_clus = cluster_data$S_random

```

```

#Create a list to contain covariance matrix for each pollutant (8)
K_AR_list = list()
K_AR_periodic_list = list()

rho_AR = 1
sigma2_AR = 1

time_span = nrow(S_random_clus)

#Calculate a AR 1 covariance matrix for each pollutant and store in list
for (i in 1:8){
  ts = S_random_clus[,i]

  K_covariate = matrix(nrow=time_span,ncol=time_span)
  K_covariate_periodic = matrix(nrow=time_span,ncol=time_span)

  for(j in 1:time_span){
    for (k in 1:time_span){
      if (abs(j-k) <= 1){

        K_covariate[j,k] = exp(- ((ts[j] - ts[k])^2) #RBF kernel
                               / (2*rho_AR)) * sigma2_AR

        K_covariate_periodic[j,k] = exp(- ((ts[j] - ts[k])^2) #Locally periodic kernel
                                         / (2*rho_AR)) * exp(- (2*sin(abs(ts[j] - ts[k]))*pi/12)^2)
                                         / (rho_AR)) * sigma2_AR
      }
      else{
        K_covariate_periodic[j,k] = 0
        K_covariate[j,k] = 0
      }
    }
  }

  K_AR_list[[i]] = K_covariate
  K_AR_periodic_list[[i]] = K_covariate_periodic
}

names(K_AR_list) = colnames(S_random_clus)
names(K_AR_periodic_list) = colnames(S_random_clus)

#Add each pollutant's covariance matrix to get AR 1 matrix for each cluster
K_AR = matrix(0,nrow=60,ncol=60)
K_AR_periodic = matrix(0,nrow=60,ncol=60)

K_AR_periodic_weights = rep(1,length(K_AR_periodic_list))

for(i in 1:length(K_AR_periodic_list)){
  K_AR = K_AR + ((1/8)*K_AR_list[[i]])
  K_AR_periodic = K_AR_periodic + ((1/8)*K_AR_periodic_list[[i]])

  K_AR_periodic_weights[i] = norm(K_AR_periodic_list[[i]],type = "F")
}

```

```

K_AR_periodic_weights = K_AR_periodic_weights / sum(K_AR_periodic_weights)
print(K_AR_periodic_weights)

K_AR_cluster[[c]] = K_AR
K_AR_periodic_cluster[[c]] = K_AR_periodic
}

## [1] 0.1179131 0.1258756 0.1279283 0.1207317 0.1295156 0.1320599 0.1251074
## [8] 0.1208684
## [1] 0.1180238 0.1257031 0.1279176 0.1208326 0.1290610 0.1318444 0.1254549
## [8] 0.1211627
## [1] 0.1177953 0.1260674 0.1281143 0.1206671 0.1292772 0.1321721 0.1251538
## [8] 0.1207528
## [1] 0.1178528 0.1258298 0.1278816 0.1206277 0.1293356 0.1320934 0.1257028
## [8] 0.1206765
## [1] 0.1182129 0.1258859 0.1279207 0.1202214 0.1291226 0.1320892 0.1254036
## [8] 0.1211437
## [1] 0.1183861 0.1257906 0.1279683 0.1210789 0.1290587 0.1317483 0.1247048
## [8] 0.1212643
## [1] 0.1183055 0.1254444 0.1268239 0.1206665 0.1289935 0.1315729 0.1272763
## [8] 0.1209169

for (i in 1:num_clus){
  title1 = sprintf("AR 1 Covariance for Cluster %s",i)
  title2 = sprintf("Periodic AR 1 Covariance for Cluster %s",i)

  # corrplot(K_AR_cluster[[i]], order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col =
  #           title = title1)
  # corrplot(K_AR_periodic_cluster[[i]], order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color",
  #           title = title2)

  matrix_heatmap(K_AR_cluster[[i]],title = title1)
  matrix_heatmap(K_AR_periodic_cluster[[i]],title = title2)
}

```

Combine each cluster's AR 1 kernel together:

```

K_AR = matrix(0,nrow=60,ncol=60)
K_AR_periodic = matrix(0,nrow=60,ncol=60)

K_AR_periodic_weights = rep(1,num_clus)

for(i in 1:num_clus){
  K_AR = K_AR + ((1/num_clus)*K_AR_cluster[[i]])
  K_AR_periodic = K_AR_periodic + ((1/num_clus)*K_AR_periodic_cluster[[i]])

  K_AR_periodic_weights[i] = norm(K_AR_cluster[[i]],type = "F")
}

K_AR_periodic_weights = K_AR_periodic_weights / sum(K_AR_periodic_weights)

print(K_AR_periodic_weights)

## [1] 0.1426904 0.1429864 0.1425446 0.1428154 0.1428045 0.1428429 0.1433158

```

```

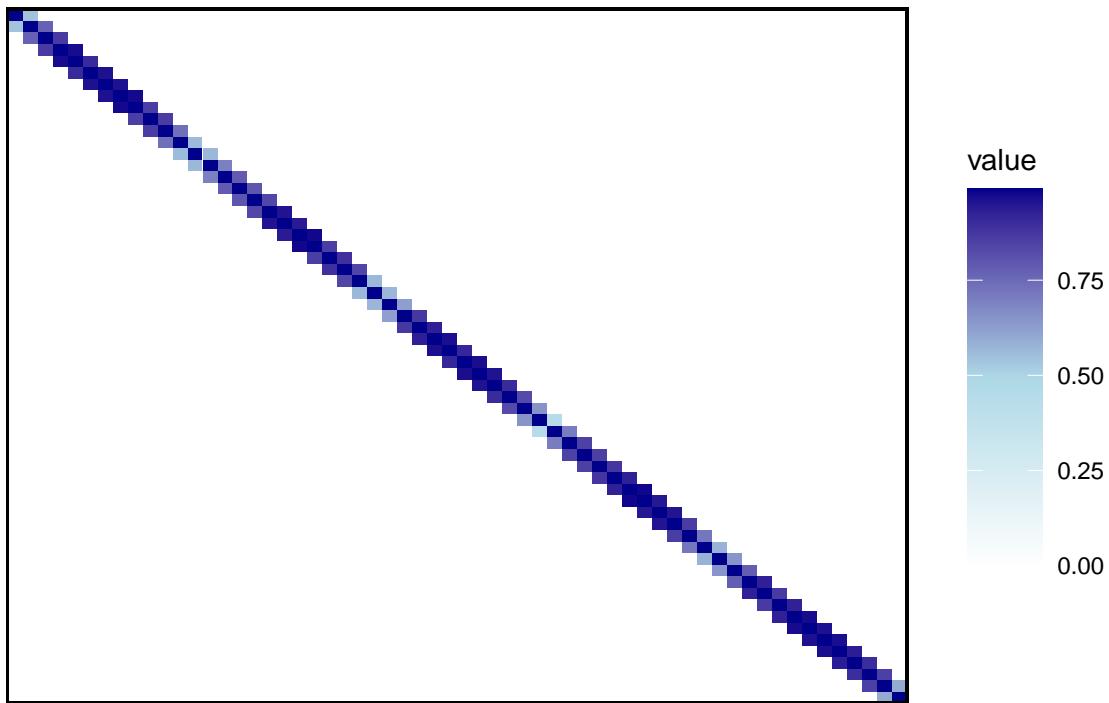
#Heatmap of resulting K
# corrplot(K_AR, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1('YlOrRd',100), title = "AR 1 Covariance Structure")
# corrplot(K_AR_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1('Blues',100), title = "Periodic AR 1 Covariance Structure")

matrix_heatmap(K_AR,title = "AR 1 Covariance Structure")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

AR 1 Covariance Structure



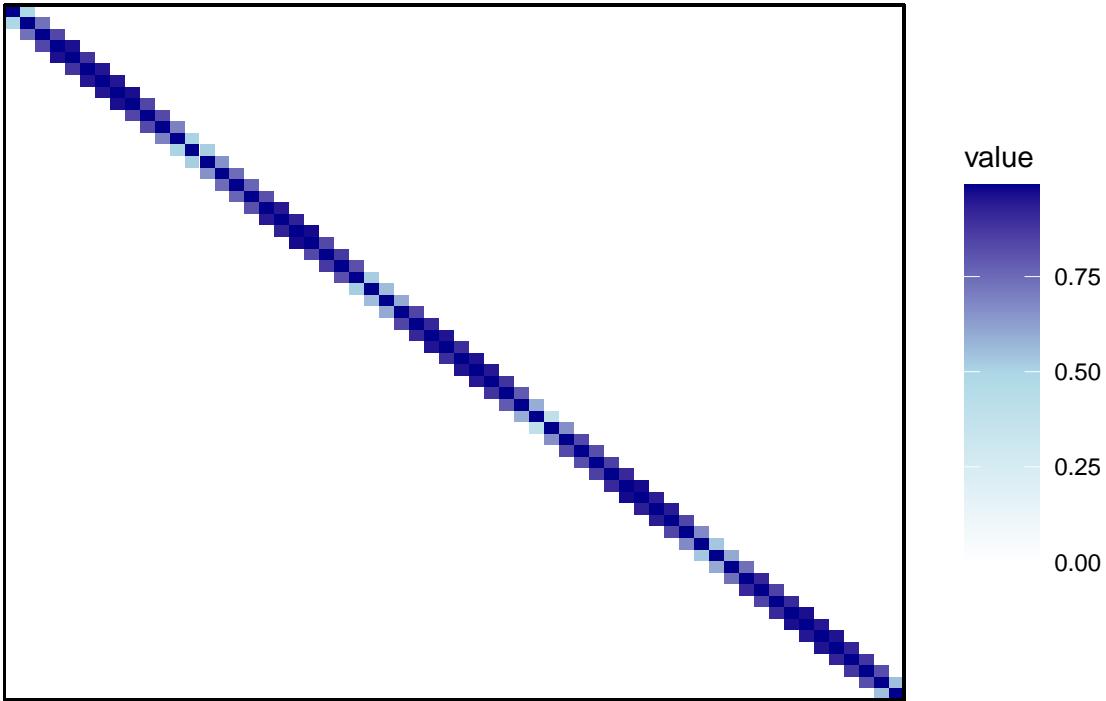
```
matrix_heatmap(K_AR_periodic,title = "Periodic AR 1 Covariance Structure")
```

```

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

Periodic AR 1 Covariance Structure



Calculating distributed lag structure

Here, we wish to account for lagged effects that we believe are significant (3,6, and 12 months). Note that we add the seasonal component to the residual component of our covariate time series decomposition to get our input for this kernel.

```
DL_invariant_list = list()

for (c in 1:num_clus){

  #Grab S_DL data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  S_DL_clus = cluster_data$S_DL

  dl3.corr.values = c()
  dl3.cov.values = c()

  dl6.corr.values = c()
  dl6.cov.values = c()

  dl12.corr.values = c()
  dl12.cov.values = c()

  for (i in 1:ncol(S_DL_clus)){
    var = var(S_DL_clus[,i])
    fit.ar = ar(S_DL_clus[,i],order.max = 1, aic = FALSE, method = "yule-walker")
    ...}
```

```

#Fit a parametric AR model for each lag
dl3 = arima(S_DL_clus[,i],order = c(3,0,0),seasonal = c(0,0,0),include.mean = FALSE,fixed = c(0,0,NA))

dl6 = arima(S_DL_clus[,i],order = c(6,0,0),seasonal = c(0,0,0),include.mean = FALSE,fixed = c(0,0,0))

dl12 = arima(S_DL_clus[,i],order = c(12,0,0),seasonal = c(0,0,0),include.mean = FALSE,
             fixed = c(0,0,0,0,0,0,0,0,0,0,0,NA))

#Calculate correlations and covariances from coefficient estimates
corr.dl3 = as.numeric(dl3$coef[3])
cov.dl3 = as.numeric(dl3$coef[3]) * var

corr.dl6 = as.numeric(dl6$coef[6])
cov.dl6 = as.numeric(dl6$coef[6]) * var

corr.dl12 = as.numeric(dl12$coef[12])
cov.dl12 = as.numeric(dl12$coef[12]) * var

dl3.corr.values = c(dl3.corr.values,corr.dl3)
dl3.cov.values = c(dl3.cov.values,cov.dl3)

dl6.corr.values = c(dl6.corr.values,corr.dl6)
dl6.cov.values = c(dl6.cov.values,cov.dl6)

dl12.corr.values = c(dl12.corr.values,corr.dl12)
dl12.cov.values = c(dl12.cov.values,cov.dl12)
}

# Run if you want to create a DL_invariant matrix for each pollutant
# for (j in 1:ncol(S_DL_clus)){
#   DL_invariant_covmatrix = diag(nrow(S_DL_clus))
#
#   DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) - 3] = dl3.cov.#
#   DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) + 3] = dl3.cov.#
#
#   DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) - 6] = dl6.cov.#
#   DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) + 6] = dl6.cov.#
#
#   DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) - 12] = dl12.cov.#
#   DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) + 12] = dl12.cov.#
#
#   # title = sprintf("Covariance of %s",colnames(S_DL_all)[j])
#   # corrplot(DL_invariant_covmatrix, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color"
#   #         title = title)
# }

DL_invariant_covmatrix = diag(nrow(S_DL_clus))

DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) - 3] = sum(dl3.cov.#
DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) + 3] = sum(dl3.cov.#

DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) - 6] = sum(dl6.cov.#

```

```

DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) + 6] = sum(dl6.cov.v)
DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) - 12] = sum(dl12.cov.v)
DL_invariant_covmatrix[row(DL_invariant_covmatrix) == col(DL_invariant_covmatrix) + 12] = sum(dl12.cov.v)

DL_invariant_list[[c]] = DL_invariant_covmatrix

# corrplot(DL_invariant_covmatrix, order = 'original', cl.pos = 'b', tl.pos = 'n', method = "color", col = COL1)
#   title = "Invariant DL (3,6,12) Covariance Structure")

matrix_heatmap(DL_invariant_covmatrix, title = "Invariant DL (3,6,12) Covariance Structure")
}

#Combine DL covariance matrices from each cluster together
K_DL_invariant = matrix(0,nrow=60,ncol=60)

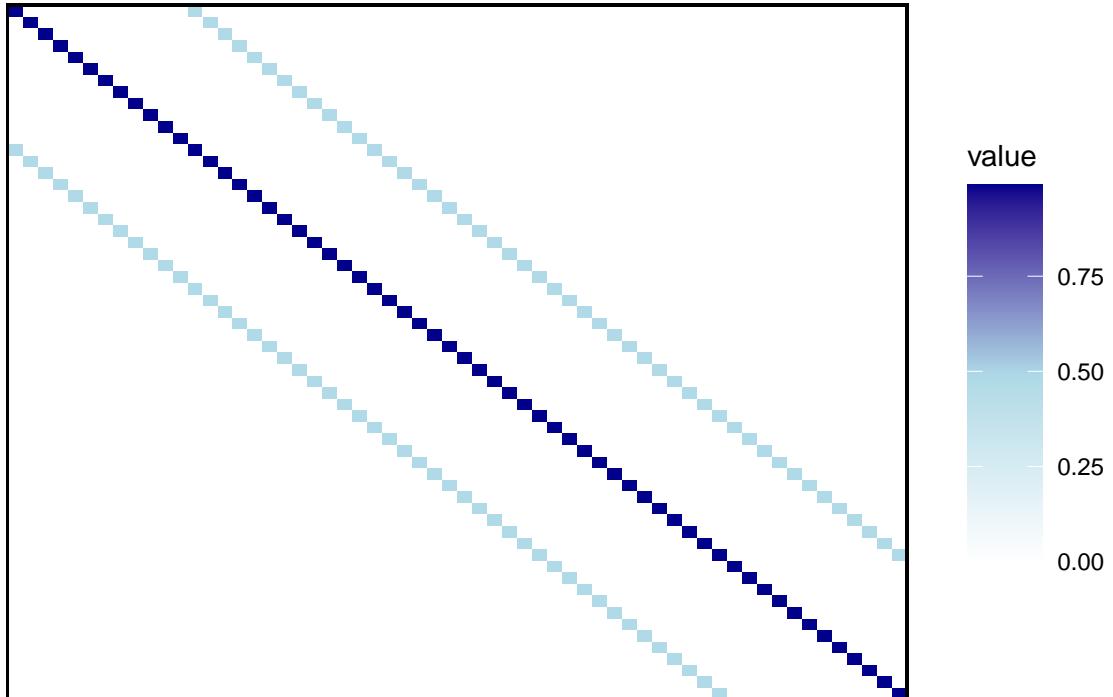
for(i in 1:num_clus){
  K_DL_invariant = K_DL_invariant + ((1/num_clus)*DL_invariant_list[[i]])
}

# corrplot(K_DL_invariant, order = 'original', cl.pos = 'b', tl.pos = 'n', method = "color", col = COL1)
#   title = "Invariant DL (3,6,12) Covariance Structure")

matrix_heatmap(K_DL_invariant, title = "Invariant DL (3,6,12) Covariance Structure")

```

Invariant DL (3,6,12) Covariance Structure



```

K_DL_cluster = list()
K_DL_periodic_cluster = list()

for (c in 1:num_clus){

  #Grab S_DL data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  S_DL_clus = cluster_data$S_DL

  #Create a list to store covariance matrix for each DL
  K_DL_list = list()
  K_DL_periodic_list = list()

  dl_lags = c(3,6,12)
  tracker = 1

  for (i in dl_lags){

    K_DL = matrix(nrow=time_span,ncol=time_span)
    K_DL_periodic = matrix(nrow=time_span,ncol=time_span)

    rho_DL = 1
    sigma2_DL = 1

    #Calculate DL covariance matrix for specified lag
    for(j in 1:nrow(S_DL_clus)){
      for (k in 1:nrow(S_DL_clus)){

        if ((abs(j-k) == 0) || (abs(j-k) == i)){

          K_DL[j,k] = exp(-(sum(S_DL_clus[j,] - S_DL_clus[k,])^2) / (2*rho_DL)) * sigma2_DL

          K_DL_periodic[j,k] = exp(-(sum(S_DL_clus[j,] - S_DL_clus[k,])^2)
                                    / (2*rho_DL)) * exp(-(2*sin(sum(abs(S_DL_clus[j,] - S_DL_clus[k,])))*pi/
                                    (rho_DL)) * sigma2_DL

        }
        else{
          K_DL_periodic[j,k] = 0
          K_DL[j,k] = 0
        }
      }
    }

    K_DL_list[[tracker]] = K_DL
    K_DL_periodic_list[[tracker]] = K_DL_periodic
    tracker = tracker+1
  }

  #Combine the 3 DL covariance matrices together
  K_DL = matrix(0,nrow=time_span,ncol=time_span)
  K_DL_periodic = matrix(0,nrow=time_span,ncol=time_span)
}

```

```

K_DL_periodic_weights = rep(1,length(K_DL_periodic_list))

for(i in 1:length(K_DL_periodic_list)){
  K_DL = K_DL + ((1/3)*K_DL_list[[i]])
  K_DL_periodic = K_DL_periodic + ((1/3)*K_DL_periodic_list[[i]])

  K_DL_periodic_weights[i] = norm(K_DL_periodic_list[[i]],type = "F")
}

K_DL_periodic_weights = K_DL_periodic_weights / sum(K_DL_periodic_weights)
print(K_DL_periodic_weights)

#Store DL(3,6,12) covariance matrix for each cluster
K_DL_cluster[[c]] = K_DL
K_DL_periodic_cluster[[c]] = K_DL
}

## [1] 0.3318032 0.3263415 0.3418554
## [1] 0.3299808 0.3265991 0.3434201
## [1] 0.3332712 0.3269659 0.3397629
## [1] 0.3317627 0.3274553 0.3407819
## [1] 0.3309092 0.3280791 0.3410116
## [1] 0.3299765 0.3262256 0.3437979
## [1] 0.3316097 0.3263056 0.3420847

Combining DL kernels for each cluster together:

K_DL = matrix(0,nrow=time_span,ncol=time_span)
K_DL_periodic = matrix(0,nrow=time_span,ncol=time_span)

K_DL_periodic_weights = rep(1,num_clus)

for(i in 1:num_clus){
  K_DL = K_DL + ((1/num_clus)*K_DL_cluster[[i]])
  K_DL_periodic = K_DL_periodic + ((1/num_clus)*K_DL_periodic_cluster[[i]])

  K_DL_periodic_weights[i] = norm(K_DL_periodic_cluster[[i]],type = "F")
}

K_DL_periodic_weights = K_DL_periodic_weights / sum(K_DL_periodic_weights)
print(K_DL_periodic_weights)

## [1] 0.1431142 0.1430462 0.1431990 0.1426814 0.1425824 0.1430127 0.1423643

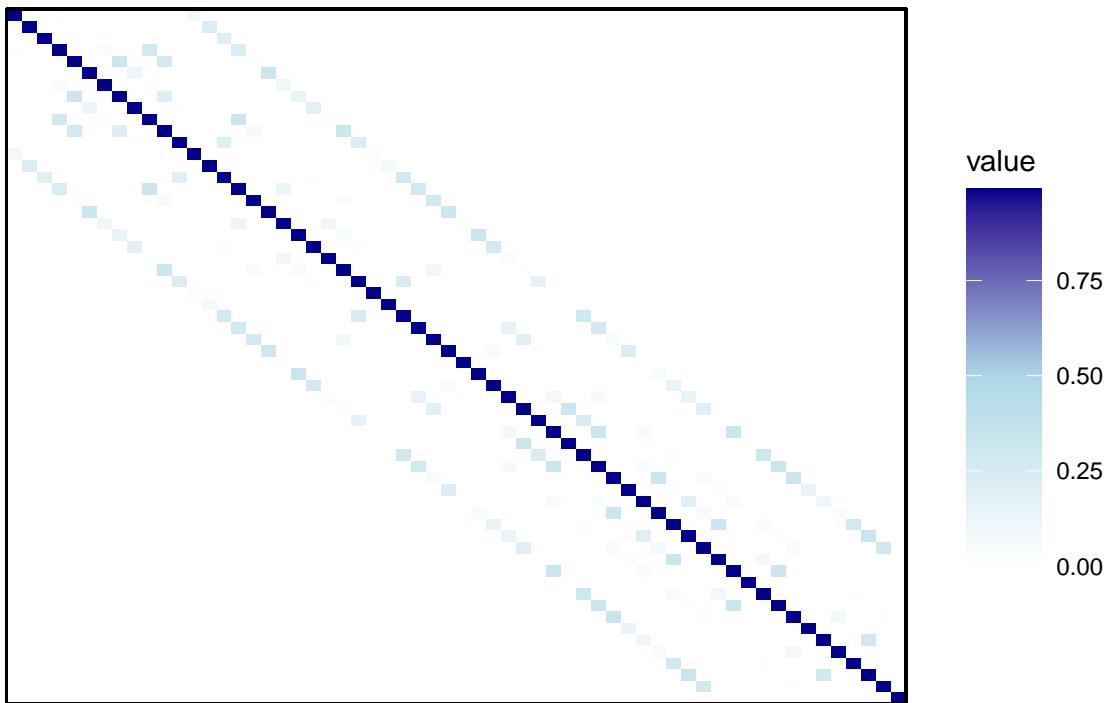
#Heatmap of resulting K
# corrplot(K_DL, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1('YlOrRd',12))
# title = "DL (3,6,12) Covariance Structure"
# corrplot(K_DL_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1('YlOrRd',12))
# title = "Periodic DL (3,6,12) Covariance Structure"

matrix_heatmap(K_DL,title = "DL (3,6,12) Covariance Structure")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
## a single row.

```

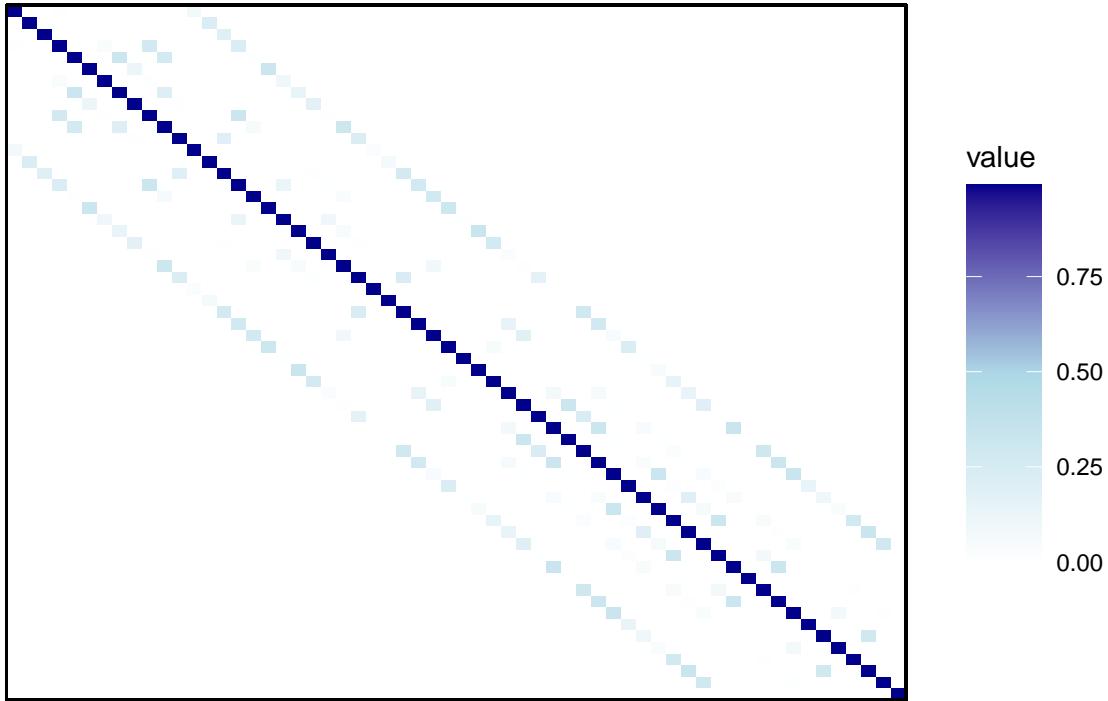
DL (3,6,12) Covariance Structure



```
matrix_heatmap(K_DL_periodic, title = "Periodic DL (3,6,12) Covariance Structure")
```

```
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```

Periodic DL (3,6,12) Covariance Structure



Calculating interaction structure

Finally, we want to include the two way interaction structures for every pair of EPA covariate time series. It is reasonable to think that the value of an air pollutant covariate at one time point may affect the value of another air pollutant at the same or even future time point. Interaction pairs are calculated by performing the kronecker product on two of time series vectors.

```
K_Interaction_cluster = list()
K_Interaction_periodic_cluster = list()

for (c in 1:num_clus){

  #Grab interaction pair data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  W2_clus = cluster_data$W2

  K_interaction_list = list()
  K_interaction_periodic_list = list()

  column_names = colnames(W2_clus)
  time_span = nrow(W2_clus)

  #Create sequence of indices corresponding to comparisons for real time and one lag interaction effects
  lag0_idx = seq(2,3601,by=61)
  lag1_idx = seq(1,3600,by=61)}
```

```

#Calculate a kernel for each interaction pair
for (a in 1:length(column_names)){
  interaction = W2_clus[,a]

  #First calculate these two interaction kernels separately
  K_int0 = matrix(nrow = 60,ncol = 60)
  K_int1 = matrix(nrow = 60,ncol = 60)

  K_int0_periodic = matrix(nrow = 60,ncol = 60)
  K_int1_periodic = matrix(nrow = 60,ncol = 60)

  rho_int = 1
  sigma2_int = 1

  for (i in 1:60){
    for (j in 1:60){

      #RBF kernels
      K_int0[i,j] = exp(- ((interaction[lag0_idx[i]] - interaction[lag0_idx[j]])^2)
                          / (2*rho_int)) * sigma2_int

      K_int1[i,j] = exp(- ((interaction[lag1_idx[i]] - interaction[lag1_idx[j]])^2)
                          / (2*rho_int)) * sigma2_int

      #Locally periodic kernels
      K_int0_periodic[i,j] = exp(- ((interaction[lag0_idx[i]] - interaction[lag0_idx[j]])^2)
                                  / (2*rho_int)) * exp(- (2*sin((abs(interaction[lag0_idx[i]] - interaction[lag0_idx[j]])) / (rho_int)) * sigma2_int

      K_int1_periodic[i,j] = exp(- ((interaction[lag1_idx[i]] - interaction[lag1_idx[j]])^2)
                                  / (2*rho_int)) * exp(- (2*sin((abs(interaction[lag1_idx[i]] - interaction[lag1_idx[j]])) / (rho_int)) * sigma2_int
    }
  }

  #Combine real time and one lag interaction kernels together
  K_interaction = 0.5*K_int0 + 0.5*K_int1
  K_interaction_list[[a]] = K_interaction

  K_interaction_periodic = 0.5*K_int0_periodic + 0.5*K_int1_periodic
  K_interaction_periodic_list[[a]] = K_interaction_periodic
}

#Combine kernels for each interaction pair together
K_interaction = matrix(0,nrow=60,ncol=60)
K_interaction_periodic = matrix(0,nrow=60,ncol=60)

K_interaction_periodic_weights = rep(1,length(K_interaction_periodic_list))

for(i in 1:length(K_interaction_periodic_list)){
  K_interaction = K_interaction + ((1/length(K_interaction_list))*K_interaction_list[[i]])

  K_interaction_periodic = K_interaction_periodic + ((1/length(K_interaction_periodic_list))*K_interactio

```

```

    K_interaction_periodic_weights[i] = norm(K_interaction_periodic_list[[i]], type = "F")
}

K_interaction_periodic_weights = K_interaction_periodic_weights / sum(K_interaction_periodic_weights)
print(K_interaction_periodic_weights)

# corrplot(K_interaction, order = 'original', cl.pos = 'b', tl.pos = 'n', method = "color", col = COL1
# title = "Interaction Covariance Structure")
# corrplot(K_interaction_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n', method = "color", c
# title = "Periodic Interaction Covariance Structure")

matrix_heatmap(K_interaction, title = "Interaction Covariance Structure")
matrix_heatmap(K_interaction_periodic, title = "Periodic Interaction Covariance Structure")

#Store final interaction kernel (for all pairs) for each cluster
K_Interaction_cluster[[c]] = K_interaction
K_Interaction_periodic_cluster[[c]] = K_interaction_periodic
}

## [1] 0.01798815 0.01707250 0.01694199 0.01782006 0.01785618 0.01704239
## [7] 0.01638232 0.01785643 0.01847610 0.01814356 0.01865467 0.01878815
## [13] 0.01861039 0.01794113 0.01707292 0.01860913 0.01798194 0.01830786
## [19] 0.01867758 0.01808722 0.01738208 0.01662694 0.01801765 0.01768447
## [25] 0.01789614 0.01822566 0.01744549 0.01696939 0.01749136 0.01878553
## [31] 0.01847234 0.01805161 0.01874162 0.01786479 0.01769166 0.01758662
## [37] 0.01886923 0.01835239 0.01820703 0.01853056 0.01816756 0.01775636
## [43] 0.01685123 0.01865585 0.01816681 0.01771010 0.01849356 0.01829750
## [49] 0.01716661 0.01599038 0.01810632 0.01725543 0.01723000 0.01782367
## [55] 0.01789775 0.01722766
## [1] 0.01799398 0.01709268 0.01706268 0.01772355 0.01782421 0.01716357
## [7] 0.01645371 0.01787718 0.01840045 0.01814090 0.01858716 0.01873067
## [13] 0.01855974 0.01795085 0.01713023 0.01852411 0.01792645 0.01821370
## [19] 0.01863021 0.01806931 0.01739713 0.01679001 0.01802977 0.01771089
## [25] 0.01784844 0.01823677 0.01752284 0.01708067 0.01751396 0.01872658
## [31] 0.01839206 0.01796957 0.01865719 0.01782524 0.01768153 0.01756624
## [37] 0.01881072 0.01830463 0.01822472 0.01844018 0.01808511 0.01767478
## [43] 0.01715139 0.01860286 0.01811352 0.01780259 0.01846529 0.01824598
## [49] 0.01734650 0.01610391 0.01809783 0.01723231 0.01735257 0.01775926
## [55] 0.01784222 0.01733938
## [1] 0.01797992 0.01699144 0.01697681 0.01777718 0.01784029 0.01701896
## [7] 0.01637073 0.01785271 0.01844724 0.01818964 0.01863244 0.01878839
## [13] 0.01858867 0.01795503 0.01716656 0.01855519 0.01795217 0.01825505
## [19] 0.01869061 0.01806611 0.01747365 0.01672483 0.01804956 0.01770563
## [25] 0.01792620 0.01829659 0.01756626 0.01704488 0.01747957 0.01876871
## [31] 0.01841102 0.01805965 0.01870195 0.01779189 0.01767880 0.01757292
## [37] 0.01886938 0.01834226 0.01827614 0.01847973 0.01812173 0.01772425
## [43] 0.01687882 0.01864848 0.01813857 0.01779204 0.01842348 0.01829411
## [49] 0.01720398 0.01599703 0.01810676 0.01721032 0.01731120 0.01778245
## [55] 0.01788553 0.01716650
## [1] 0.01796224 0.01701220 0.01692662 0.01778606 0.01785974 0.01725829
## [7] 0.01634749 0.01782948 0.01844827 0.01812726 0.01863357 0.01876688
## [13] 0.01856995 0.01790556 0.01702322 0.01857154 0.01794633 0.01828354

```

```

## [19] 0.01866044 0.01823927 0.01734590 0.01661930 0.01800915 0.01769305
## [25] 0.01787700 0.01826233 0.01752535 0.01697313 0.01745899 0.01876345
## [31] 0.01843720 0.01802069 0.01872466 0.01786430 0.01765832 0.01759966
## [37] 0.01884744 0.01833805 0.01824918 0.01850735 0.01824258 0.01776540
## [43] 0.01691454 0.01860446 0.01822114 0.01775709 0.01848905 0.01827943
## [49] 0.01720040 0.01597595 0.01806987 0.01720394 0.01723912 0.01779865
## [55] 0.01789749 0.01740841
## [1] 0.01801276 0.01703143 0.01694759 0.01778825 0.01788158 0.01721644
## [7] 0.01642633 0.01788767 0.01844540 0.01811813 0.01863162 0.01880566
## [13] 0.01860215 0.01797847 0.01715008 0.01856778 0.01795322 0.01827947
## [19] 0.01870554 0.01812662 0.01744781 0.01661799 0.01798394 0.01763119
## [25] 0.01778816 0.01815186 0.01738221 0.01692601 0.01747119 0.01877287
## [31] 0.01841553 0.01797934 0.01874718 0.01787089 0.01766895 0.01762439
## [37] 0.01888591 0.01836894 0.01820732 0.01854172 0.01825139 0.01778447
## [43] 0.01684561 0.01862202 0.01808052 0.01764450 0.01849106 0.01829118
## [49] 0.01715437 0.01607265 0.01813384 0.01722352 0.01722589 0.01781774
## [55] 0.01792503 0.01739660
## [1] 0.01803884 0.01713361 0.01713705 0.01777179 0.01784107 0.01705886
## [7] 0.01651129 0.01792731 0.01845711 0.01817844 0.01860422 0.01876322
## [13] 0.01854934 0.01799890 0.01723745 0.01857292 0.01798694 0.01826528
## [19] 0.01866658 0.01795032 0.01749327 0.01683844 0.01804676 0.01771537
## [25] 0.01786577 0.01818352 0.01735629 0.01708117 0.01753902 0.01875002
## [31] 0.01841447 0.01802720 0.01866245 0.01779154 0.01768402 0.01758671
## [37] 0.01884653 0.01831461 0.01819993 0.01846422 0.01800512 0.01768366
## [43] 0.01697388 0.01860367 0.01792730 0.01766821 0.01843175 0.01817202
## [49] 0.01717160 0.01615916 0.01813972 0.01726979 0.01736962 0.01780464
## [55] 0.01784823 0.01725975
## [1] 0.01793093 0.01703255 0.01699054 0.01774645 0.01778528 0.01741396
## [7] 0.01643922 0.01777190 0.01836883 0.01810608 0.01856923 0.01867281
## [13] 0.01857209 0.01786711 0.01691149 0.01852418 0.01780964 0.01816871
## [19] 0.01847682 0.01826081 0.01726716 0.01667661 0.01797483 0.01756242
## [25] 0.01790155 0.01825023 0.01781933 0.01708938 0.01739940 0.01869059
## [31] 0.01833717 0.01800166 0.01861137 0.01802058 0.01764887 0.01751726
## [37] 0.01875468 0.01815959 0.01814963 0.01840496 0.01829976 0.01772545
## [43] 0.01721479 0.01859167 0.01820298 0.01795298 0.01856058 0.01832354
## [49] 0.01750714 0.01607387 0.01805412 0.01725451 0.01733893 0.01776503
## [55] 0.01784731 0.01763145

```

Combining interaction kernels from each cluster together:

```

K_interaction = matrix(0,nrow=60,ncol=60)
K_interaction_periodic = matrix(0,nrow=60,ncol=60)

K_interaction_periodic_weights = rep(1,num_clus)

for(i in 1:num_clus){
  K_interaction = K_interaction + ((1/length(K_Interaction_cluster))*K_Interaction_cluster[[i]])

  K_interaction_periodic = K_interaction_periodic + ((1/length(K_Interaction_periodic_cluster))*K_Interaction_periodic_cluster[[i]])

  K_interaction_periodic_weights[i] = norm(K_Interaction_periodic_cluster[[i]],type = "F")
}

K_interaction_periodic_weights = K_interaction_periodic_weights / sum(K_interaction_periodic_weights)
print(K_interaction_periodic_weights)

```

```

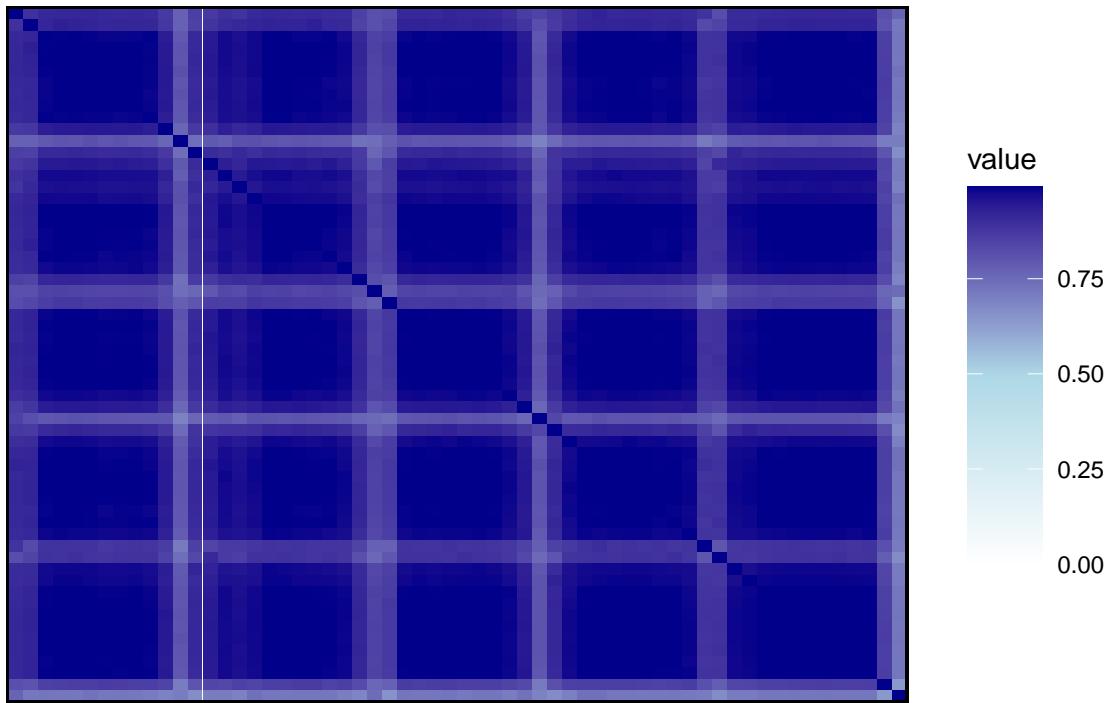
## [1] 0.1426358 0.1431007 0.1425407 0.1428563 0.1425086 0.1426961 0.1436618
# corrplot(K_interaction, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col = COL1('
# title = "Interaction Covariance Structure")
# corrplot(K_interaction_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col
# title = "Periodic Interaction Covariance Structure")

matrix_heatmap(K_interaction,title = "Interaction Covariance Structure")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
## a single row.

```

Interaction Covariance Structure



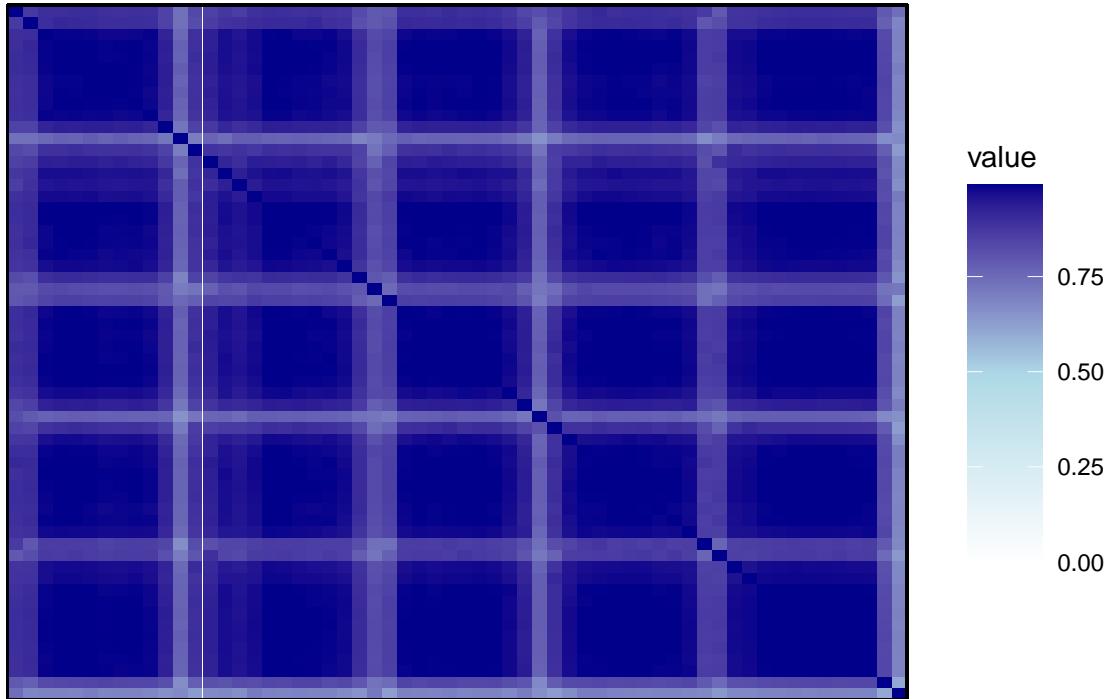
```

matrix_heatmap(K_interaction_periodic,title = "Periodic Interaction Covariance Structure")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
## a single row.

```

Periodic Interaction Covariance Structure



Kernel target alignment calculation between 3 variations of each kernel

```
F_norm = function(kernel){

  total = sum(as.numeric(abs(kernel)^2))
  Fnorm = sqrt(total)
  return(Fnorm)
}

KTA_norm = function(kernel1,kernel2){

  centered_k1 = t(diag(nrow(kernel1)) - (1/nrow(kernel1) * t(diag(nrow(kernel1))) %*% diag(nrow(kernel1) * diag(nrow(kernel1)) - (1/nrow(kernel1) * t(diag(nrow(kernel1))) %*% diag(nrow(kernel1))

  centered_k2 = t(diag(nrow(kernel2)) - (1/nrow(kernel2) * t(diag(nrow(kernel2))) %*% diag(nrow(kernel2) * diag(nrow(kernel2)) - (1/nrow(kernel2) * t(diag(nrow(kernel2))) %*% diag(nrow(kernel2))

  term1 = centered_k1 / (F_norm(centered_k1))
  term2 = centered_k2 / (F_norm(centered_k2))
  term3 = term1 - term2
  term4 = F_norm(term3)

  measure = 1 - 0.5*(term4^2)
  return(measure)
}
```

```

KTA_rownames = c("AR Invariant-AR RBF", "AR Invariant-AR LP",
                "AR RBF-AR LP ", "DL Invariant-DL RBF",
                "DL Invariant-DL LP", "DL RBF-DL LP",
                "Interaction RBF-Interaction LP")

KTA_table = matrix(c(K_AR_invariant,K_AR),
                   KTA_norm(K_AR_invariant,K_AR_periodic),
                   KTA_norm(K_AR,K_AR_periodic),
                   KTA_norm(K_DL_invariant,K_DL),
                   KTA_norm(K_DL_invariant,K_DL_periodic),
                   KTA_norm(K_DL,K_DL_periodic),
                   KTA_norm(K_interaction,K_interaction_periodic)),
                   nrow=7)

KTA_table = data.frame(KTA_table)
rownames(KTA_table) = KTA_rownames
colnames(KTA_table) = "Centered KTA via inner product"

KTA_table

##                                     Centered KTA via inner product
## AR Invariant-AR RBF                 0.5993978
## AR Invariant-AR LP                  0.6105226
## AR RBF-AR LP                      0.9997569
## DL Invariant-DL RBF                 0.7920310
## DL Invariant-DL LP                  0.7920310
## DL RBF-DL LP                      1.0000000
## Interaction RBF-Interaction LP     0.9999298

```

Cleaning and aggregating CalViDa mortality data

Imputing “< 11” values in data with EM algorithm

In the mortality dataset obtained from Cal-ViDa, all of the cells with small values i.e., less than 10 but not equal to 0, were censored. So in order to avoid using a truncated Poisson distribution, we decided to impute these censored values with an EM algorithm which is described below:

Due to interval censoring, we do not observe the exact mortality counts for some units. Let C_i be the censoring indicator such that $C_i = I(1 \leq Y_i \leq 10)$, i.e., the count is censored if it is between 1 and 10. For the EM algorithm, we will be modeling the rate using a generalized linear model where we use the variables month (our time index), cause of death, and age group as predictors. So let us assume that $\lambda_i(\beta) = \exp(\alpha_i + \mathbf{X}_i^T \beta)$, where α_i is the offset (log of county population of the age group) and \mathbf{X}_i are predictors (month, cause of death, age group). Derivations for the EM algorithm can be found in appendix B of the paper.

We coded this algorithm as follows:

First, we needed to get an initial estimate of our β coefficients in our Poisson regression model. We included age group, county of death, cause of death (either influenza+pneumonia OR chronic lower respiratory disease), and month of death as covariates.

We need to perform an initial imputation to get a complete dataset to fit a model on. We decided to do this by making a crude estimation of the rate per 100,000 people λ_i . To do this, we calculated a population weighted mean of the number of deaths across all ages and months. However, we only had observed populations at the county level, not for each specific age group included in the mortality data. So using census data which told us the approximate populations for specific age groups (for all of California), we were able to calculate

approximate population sizes for each of the age categories included in the mortality data. See death_byage2 for reference. Then using the ratio of a given county's population relative to the entire population of California, we were able to calculate approximations for the population size of each age group for each county in our mortality dataset. These served as the weights for our population weighted average of the rate of respiratory deaths in California.

```

population_age = read_xlsx("Population Categories.xlsx")

## New names:
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
## * `` -> `...8`
## * `` -> `...9`
## * `` -> `...10`
## * `` -> `...11`
## * `` -> `...12`
## * `` -> `...13`

#head(population_age)

population_age = population_age[-(1:5),2]
population_age$...2 = as.numeric(population_age$...2)

#split under 5 category into < 1 and 1-4 years old
less1 = floor(population_age$...2[1]*0.2)
onefour = floor(population_age$...2[1]*0.8)
death_byage = population_age$...2[-1]
death_byage = c(less1,onefour,death_byage)
death_byage = death_byage[1:19]

death_byage2 = death_byage[1:2]
idx = seq(from = 3, to = 17, by = 2)
for (i in idx){
  death_byage2 = c(death_byage2,(death_byage[i]+death_byage[i+1]))
}
death_byage2 = c(death_byage2,death_byage[19])

age_groups = unique(mortality$Age)
death_byage2 = data.frame(cbind(age_groups,death_byage2))
colnames(death_byage2) = c("Age_Group","Population_by_Age")
death_byage2$Population_by_Age = as.numeric(death_byage2$Population_by_Age)
#head(death_byage2)

#2010-2019 population data for CA
USpops = read.csv("CA_census_pops1019.csv")
CApops = USpops %>% filter(STNAME == "California") %>% select(CTYNAME,POPESTIMATE2019)
counties = countycodes$value_represented #from EPA data file

weights = CApops[(2:59),2]
weights = weights/CApops[1,2]
```

```

groups = unique(mortality$Age)
step1 = 1
step2 = 1

for (i in counties){
  for (j in groups){
    idx = which(mortality$Age == j & mortality$County_of_Death == i)
    mortality$Population[idx] = ceiling(death_byage2$Population[step1]*weights[step2])
    step1 = step1+1
  }
  step1 = 1
  step2 = step2+1
}

mortality$logpop = log(mortality$Population)

censored_idx = which(mortality$Total_Deaths == "<11")
censorTF = mortality$Total_Deaths == "<11"
mortality = cbind(mortality,censorTF)
#head(mortality)

```

GETTING INITIAL GUESS FOR LAMBDA: AVG DEATHS (PER 100K PEOPLE) PER MONTH FOR ONE COUNTY

```

uncensored_mortality = mortality %>% filter(censorTF == FALSE) %>% select(Total_Deaths,Population)
uncensored_mortality$Total_Deaths = as.numeric(uncensored_mortality$Total_Deaths)

theta = mean(uncensored_mortality$Total_Deaths*100000/uncensored_mortality$Population)

```

By using all the data, I obtained a crude initial guess for λ of about 1.08. Using this initial estimate λ , we calculated the expected value for each Z_i to get an initial imputed dataset. This dataset will be used to estimate a Poisson regression model which will give us our initial value for our actual parameters of interest β .

FUNCTION FOR IMPUTING CENSORED VALUE BASED ON EXPECTATION GIVEN LAMBDA

```

impute_small_values = function(lambda){
  x = 1:10
  p = dpois(x,lambda)

  value = sum(x*p)/sum(p)
  return(value)
}

```

INITIAL IMPUTATION:

```

mortality2 = mortality
mortality2$Total_Deaths[censored_idx] = 0.01
mortality2$Total_Deaths = as.numeric(mortality2$Total_Deaths)

for (i in censored_idx){
  lambda = theta*mortality2$Population[i] / 100000
  deaths = impute_small_values(lambda)

  mortality2$Total_Deaths[i] = floor(deaths)
}

```

INITIAL REGRESSION MODELS:

```

mortality2$Age = factor(mortality2$Age)
mortality2$Cause_of_Death = factor(mortality2$Cause_of_Death)
mortality2$Month = factor(mortality2$Month)

pois_reg = glm(Total_Deaths ~ Age + Cause_of_Death + Month + offset(logpop), family = "poisson", data =
# ZIP_reg = zeroInfl(Total_Deaths ~ Age + Cause_of_Death + Month + offset(logpop) / 1, data = mortality2)

vec0 = coef(pois_reg)
# vec0 = coef(ZIP_reg)

summary(pois_reg)

## 
## Call:
## glm(formula = Total_Deaths ~ Age + Cause_of_Death + Month + offset(logpop),
##      family = "poisson", data = mortality2)
## 
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -13.744433   0.087789 -156.562 < 2e-16
## Age15 - 24 years            0.568750   0.095519   5.954 2.61e-09
## Age25 - 34 years            1.081482   0.091812  11.779 < 2e-16
## Age35 - 44 years            1.667882   0.090131  18.505 < 2e-16
## Age45 - 54 years            2.381811   0.088827  26.814 < 2e-16
## Age5 - 14 years              0.236541   0.099117   2.386  0.017
## Age55 - 64 years             3.136398   0.088079  35.609 < 2e-16
## Age65 - 74 years             4.183664   0.087714  47.697 < 2e-16
## Age75 - 84 years             5.352522   0.087582  61.115 < 2e-16
## Age85 years and over        6.499169   0.087521  74.258 < 2e-16
## AgeLess than 1 year          0.965400   0.138658   6.962 3.34e-12
## Cause_of_DeathInfluenza and pneumonia -0.751349   0.007035 -106.798 < 2e-16
## Month2                      -0.253385   0.013643  -18.573 < 2e-16
## Month3                      -0.251082   0.013634  -18.416 < 2e-16
## Month4                      -0.450337   0.014455  -31.155 < 2e-16
## Month5                      -0.522552   0.014781  -35.352 < 2e-16
## Month6                      -0.646664   0.015382  -42.040 < 2e-16
## Month7                      -0.661526   0.015458  -42.796 < 2e-16
## Month8                      -0.709630   0.015707  -45.180 < 2e-16
## Month9                      -0.756347   0.015957  -47.400 < 2e-16
## Month10                     -0.651645   0.015407  -42.294 < 2e-16
## Month11                     -0.617588   0.015237  -40.532 < 2e-16
## Month12                     -0.357779   0.014059  -25.449 < 2e-16
## 
## (Intercept)                 ***
## Age15 - 24 years             ***
## Age25 - 34 years             ***
## Age35 - 44 years             ***
## Age45 - 54 years             ***
## Age5 - 14 years                  *
## Age55 - 64 years             ***
## Age65 - 74 years             ***
## Age75 - 84 years             ***
## Age85 years and over         ***

```

```

## AgeLess than 1 year           ***
## Cause_of_DeathInfluenza and pneumonia ***
## Month2                         ***
## Month3                         ***
## Month4                         ***
## Month5                         ***
## Month6                         ***
## Month7                         ***
## Month8                         ***
## Month9                         ***
## Month10                        ***
## Month11                        ***
## Month12                        ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##      Null deviance: 382636  on 91871  degrees of freedom
## Residual deviance:  74194  on 91849  degrees of freedom
## AIC: 126734
##
## Number of Fisher Scoring iterations: 7
# summary(ZIP_reg)

```

Now that we have initialized our parameters, $\beta^{(0)}$, we can proceed with the EM algorithm until our parameters (the coefficients of our regression model), converge.

The main steps implemented in the chunk below are:

1. Given a newly fitted Poisson regression model with parameter values $\beta^{(t)}$, take its fitted values for the λ 's corresponding to observations that were censored in the original mortality dataset
2. Use those fitted λ 's, calculate the expected value of our unknown values Z
3. Once all Z_i 's are imputed, we can use the now complete dataset to estimate the Poisson regression model again, which will produce the maximum likelihood estimate of our parameters β , these are our new values $\beta^{(t+1)}$.
4. Compare the difference between our new β coefficient estimates with those from the previous iteration and either perform another iteration or stop the algorithm if the maximum difference between coefficients from different iterations is less than 0.01.

Note: I experimented with a ZIP regression model as well but the log likelihood values at each iteration were generally higher for the Poisson regression model

```

mortality3 = mortality2
model = pois_reg

model_diff = 100
iter = 1
vec0 = coef(model)

while((model_diff > 0.01) & (iter < 10)){

  #impute data (should be between 1-10)
  fvs = fitted.values(model)
}

```

```

for (i in censored_idx){
  deaths = impute_small_values(fvs[i])
  mortality3$Total_Deaths[i] = floor(deaths)
}

#fit model on "new" data
model = glm(Total_Deaths ~ Age + Cause_of_Death + Month + offset(logpop), family = "poisson", data = mortality3)
vec1 = coef(model)

model_diff = max(abs(vec1 - vec0))
iter = iter+1
vec0 = vec1
}

final_pois_reg = model
summary(final_pois_reg)

## Call:
## glm(formula = Total_Deaths ~ Age + Cause_of_Death + Month + offset(logpop),
##      family = "poisson", data = mortality3)
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -14.351349   0.116534 -123.152 < 2e-16
## Age15 - 24 years            -0.054190   0.135792  -0.399 0.689845
## Age25 - 34 years             0.430115   0.127371   3.377 0.000733
## Age35 - 44 years              1.210281   0.122028   9.918 < 2e-16
## Age45 - 54 years              2.345599   0.118274  19.832 < 2e-16
## Age5 - 14 years              -0.455007   0.145826  -3.120 0.001807
## Age55 - 64 years              3.744079   0.116771  32.063 < 2e-16
## Age65 - 74 years              5.001304   0.116457  42.946 < 2e-16
## Age75 - 84 years              6.168765   0.116379  53.006 < 2e-16
## Age85 years and over          7.285988   0.116346  62.623 < 2e-16
## AgeLess than 1 year           1.536529   0.158560   9.691 < 2e-16
## Cause_of_DeathInfluenza and pneumonia -0.719013   0.006477 -111.013 < 2e-16
## Month2                         -0.245579   0.012764  -19.240 < 2e-16
## Month3                         -0.234760   0.012726  -18.448 < 2e-16
## Month4                         -0.416647   0.013416  -31.057 < 2e-16
## Month5                         -0.488946   0.013715  -35.649 < 2e-16
## Month6                         -0.603208   0.014221  -42.418 < 2e-16
## Month7                         -0.631446   0.014352  -43.998 < 2e-16
## Month8                         -0.672621   0.014547  -46.237 < 2e-16
## Month9                         -0.708569   0.014723  -48.128 < 2e-16
## Month10                        -0.618620   0.014292  -43.285 < 2e-16
## Month11                        -0.580590   0.014118  -41.125 < 2e-16
## Month12                        -0.337932   0.013106  -25.784 < 2e-16
##
## (Intercept)                   ***
## Age15 - 24 years                ***
## Age25 - 34 years                ***
## Age35 - 44 years                ***
## Age45 - 54 years                ***

```

```

## Age5 - 14 years                      **
## Age55 - 64 years                     ***
## Age65 - 74 years                     ***
## Age75 - 84 years                     ***
## Age85 years and over                ***
## AgeLess than 1 year                 ***
## Cause_of_DeathInfluenza and pneumonia ***
## Month2                                ***
## Month3                                ***
## Month4                                ***
## Month5                                ***
## Month6                                ***
## Month7                                ***
## Month8                                ***
## Month9                                ***
## Month10                               ***
## Month11                               ***
## Month12                               ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
## Null deviance: 455249  on 91871  degrees of freedom
## Residual deviance: 38553  on 91849  degrees of freedom
## AIC: 97178
##
## Number of Fisher Scoring iterations: 8
logLik(final_pois_reg)

## 'log Lik.' -48566.18 (df=23)

```

Alternatively, we can take the first and second derivative and apply a Newton-Raphson procedure if we want to solve for β numerically. To solve for the fixed point solutions for β numerically, we would first take the derivative of $Q()$ wrt each β_j which gives us:

$$\frac{dQ}{d\beta_j} = \sum_{i \in D \setminus W}^{n_{obs}} [y_i - \exp(\alpha_i + x_i \beta)] x_{ij} + \sum_{i \in W}^N [\tilde{y}_i - \exp(\alpha_i + x_i \beta)] x_{ij} \text{ where } \alpha_i \text{ represents the offset associated with each observation } i \text{ and } \tilde{y}_i \text{ represents the imputed values corresponding to censored observation } Z_i$$

Rewriting this in vector form (since we have 3 covariates, which are all categorical) we obtain the following gradient:

$$f'(\beta) = \frac{dQ}{d\beta} = X^T [\vec{y} - \exp(\vec{\alpha} + X\beta)] \text{ where first term is a } p \times n \text{ matrix and second term is a } n \times 1 \text{ vector}$$

Then, we take another derivative to get the Hessian:

$$f''_{jj'}(\beta) = - \sum_{i=1}^N \exp(\alpha_i + x_i \beta) X_{ij} X_{ij'} = -X^T \text{diag}(\exp(\vec{\alpha} + X\beta)) X$$

Finally, we solve for the next value for our β s using these two values with the following equation:

$$\beta^{(b)} = \beta^{(b-1)} - [(f''(\beta^{(b-1)}))^{-1} f'(\beta^{(b-1)})]$$

Now we need to code up a Newton Raphson function and initialize it:

```
Y = mortality2$Total_Deaths
```

```
#X is design matrix w col of 1s then each level of each categorical predictors except their baselines
intercept = rep(1,length(Y))
```

```

#Age categories
Age1524 = as.numeric(mortality2$Age == "15 - 24 years")
Age2534 = as.numeric(mortality2$Age == "25 - 34 years")
Age3544 = as.numeric(mortality2$Age == "35 - 44 years")
Age4554 = as.numeric(mortality2$Age == "45 - 54 years")
Age514 = as.numeric(mortality2$Age == "5 - 14 years")
Age5564 = as.numeric(mortality2$Age == "55 - 64 years")
Age6574 = as.numeric(mortality2$Age == "65 - 74 years")
Age7584 = as.numeric(mortality2$Age == "75 - 84 years")
Age85 = as.numeric(mortality2$Age == "85 years and over")
Age1 = as.numeric(mortality2$Age == "Less than 1 year")

#Cause of death categories
Cause2 = as.numeric(mortality2$Cause_of_Death == "Influenza and pneumonia")

#Month categories
Month2 = as.numeric(mortality2$Month == 2)
Month3 = as.numeric(mortality2$Month == 3)
Month4 = as.numeric(mortality2$Month == 4)
Month5 = as.numeric(mortality2$Month == 5)
Month6 = as.numeric(mortality2$Month == 6)
Month7 = as.numeric(mortality2$Month == 7)
Month8 = as.numeric(mortality2$Month == 8)
Month9 = as.numeric(mortality2$Month == 9)
Month10 = as.numeric(mortality2$Month == 10)
Month11 = as.numeric(mortality2$Month == 11)
Month12 = as.numeric(mortality2$Month == 12)

X = cbind(intercept, Age1524, Age2534, Age3544, Age4554, Age514, Age5564, Age6574, Age7584, Age85, Age1,
          Cause2, Month2, Month3, Month4, Month5, Month6, Month7, Month8, Month9, Month10,
          Month11, Month12)
# dim(X)

offset_vec = offset(mortality2$logpop)
offset_vec = matrix(offset_vec, ncol=1)

#Initial guesses for beta
B = coef(pois_reg)
B = matrix(B, ncol=1)

#Define first derivative of Q function
f_gradient = function(Y,X,B){
  value = t(X) %*% (Y - exp(X %*% B + offset_vec))
  return(value)
}

# f_gradient(Y,X,B)

#Define second derivative of Q function
f_hessian = function(Y,X,B){
  middle = as.numeric(exp(X %*% B + offset_vec))
  X2 = X

```

```

for (i in 1:length(middle)){
  X2[i,] = X[i,] * middle[i]
}

value = -t(X) %*% X2

return(value)
}

# dim(f_hessian(Y,X,B))

#Define Newton Raphson function and compute initial beta coefficient estimates
Newton_Raphson = function(Y,X,x0,tol = 0.001,eps = 0.01,max_iter = 100){
  for (i in 1:max_iter){
    g = f_gradient(Y,X,x0)
    h = f_hessian(Y,X,x0)
    value = abs(det(h))

    if (value < eps){
      break
    }

    x1 = x0 - (solve(h) %*% g)
    # x1 = x0 - (solve(h) %*% g * (0.01 * 0.999^i)) #gradient descent is too large at each iteration so

    if (max(abs(x1-x0)) <= tol){
      return(x1)
    }

    x0 = x1
  }

  return(x0)
}

#Initial beta coefficient estimates
new_coefs = Newton_Raphson(Y,X,B)

```

Similar to above, now that we have initialized our parameters, $\beta^{(0)}$, we can proceed with the EM algorithm until our parameters (the coefficients of our regression model), converge.

The main steps implemented in the chunk below are:

1. Given newly estimated parameter values $\beta^{(t)}$ from the Newton-Raphson procedure above, take its fitted values for the λ 's corresponding to observations that were censored in the original mortality dataset
2. Use those fitted λ 's, calculate the expected value of our unknown values Z
3. Once all Z_i 's are imputed, we can use the now complete dataset to estimate the beta coefficients with Newton-Raphson again, which will produce the maximum likelihood estimate of our parameters β , these are our new values $\beta^{(t+1)}$.
4. Compare the difference between our new β coefficient estimates with those from the previous iteration and either perform another iteration or stop the algorithm if the maximum difference between coefficients from different iterations is less than 0.01.

```

while((model_diff > 0.01) & (iter < 10)){

  #impute data (should be between 1-10)
  fvs_NR = exp((X %*% new_coefs) + offset_vec)

  for (i in censored_idx){
    deaths = impute_small_values(fvs_NR[i])
    Y[i] = floor(deaths)
  }

  new_coefs2 = Newton_Raphson(Y,X,new_coefs)

  model_diff = max(abs(new_coefs2 - new_coefs))
  new_coefs = new_coefs2
  iter = iter+1
  vec0 = vec1
}

new_coefs

##          [,1]
## intercept -13.7444329
## Age1524    0.5687496
## Age2534    1.0814825
## Age3544    1.6678819
## Age4554    2.3818112
## Age514     0.2365414
## Age5564    3.1363978
## Age6574    4.1836642
## Age7584    5.3525223
## Age85      6.4991688
## Age1       0.9653963
## Cause2     -0.7513495
## Month2     -0.2533848
## Month3     -0.2510821
## Month4     -0.4503366
## Month5     -0.5225520
## Month6     -0.6466641
## Month7     -0.6615256
## Month8     -0.7096304
## Month9     -0.7563473
## Month10    -0.6516455
## Month11    -0.6175879
## Month12    -0.3577794

summary(final_pois_reg)

##
## Call:
## glm(formula = Total_Deaths ~ Age + Cause_of_Death + Month + offset(logpop),
##       family = "poisson", data = mortality3)
##
## Coefficients:
##                                         Estimate Std. Error z value Pr(>|z|)

```

```

## (Intercept) -14.351349 0.116534 -123.152 < 2e-16
## Age15 - 24 years -0.054190 0.135792 -0.399 0.689845
## Age25 - 34 years 0.430115 0.127371 3.377 0.000733
## Age35 - 44 years 1.210281 0.122028 9.918 < 2e-16
## Age45 - 54 years 2.345599 0.118274 19.832 < 2e-16
## Age5 - 14 years -0.455007 0.145826 -3.120 0.001807
## Age55 - 64 years 3.744079 0.116771 32.063 < 2e-16
## Age65 - 74 years 5.001304 0.116457 42.946 < 2e-16
## Age75 - 84 years 6.168765 0.116379 53.006 < 2e-16
## Age85 years and over 7.285988 0.116346 62.623 < 2e-16
## AgeLess than 1 year 1.536529 0.158560 9.691 < 2e-16
## Cause_of_DeathInfluenza and pneumonia -0.719013 0.006477 -111.013 < 2e-16
## Month2 -0.245579 0.012764 -19.240 < 2e-16
## Month3 -0.234760 0.012726 -18.448 < 2e-16
## Month4 -0.416647 0.013416 -31.057 < 2e-16
## Month5 -0.488946 0.013715 -35.649 < 2e-16
## Month6 -0.603208 0.014221 -42.418 < 2e-16
## Month7 -0.631446 0.014352 -43.998 < 2e-16
## Month8 -0.672621 0.014547 -46.237 < 2e-16
## Month9 -0.708569 0.014723 -48.128 < 2e-16
## Month10 -0.618620 0.014292 -43.285 < 2e-16
## Month11 -0.580590 0.014118 -41.125 < 2e-16
## Month12 -0.337932 0.013106 -25.784 < 2e-16
##
## (Intercept) ***
## Age15 - 24 years ***
## Age25 - 34 years ***
## Age35 - 44 years ***
## Age45 - 54 years ***
## Age5 - 14 years **
## Age55 - 64 years ***
## Age65 - 74 years ***
## Age75 - 84 years ***
## Age85 years and over ***
## AgeLess than 1 year ***
## Cause_of_DeathInfluenza and pneumonia ***
## Month2 ***
## Month3 ***
## Month4 ***
## Month5 ***
## Month6 ***
## Month7 ***
## Month8 ***
## Month9 ***
## Month10 ***
## Month11 ***
## Month12 ***
##
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
## Null deviance: 455249 on 91871 degrees of freedom
## Residual deviance: 38553 on 91849 degrees of freedom

```

```

## AIC: 97178
##
## Number of Fisher Scoring iterations: 8

```

NEWTON RAPHSON APPROACH DOES NOT WORK WELL BC AT EACH ITERATION VALUES ARE CHANGING BY TOO MUCH, LEADS TO HESSIAN MATRIX BEING UNINVERTIBLE

Now that we have imputed the censored “< 11” values in the Cal-ViDa dataset, we will now aggregate the data to get total number of respiratory related deaths for each county for every month between 2014-2019.

Empirical 5 number summaries for each variable of interest

```

print("Summary for respiratory related mortality")

## [1] "Summary for respiratory related mortality"
summary(mortality3$Total_Deaths) #summary of deaths per month, for all counties i.e. all of CA, age gro

##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 0.000   0.000   0.000   1.178   0.000 209.000

print("Summary for EPA data")

## [1] "Summary for EPA data"
for (i in pollutants$parametercodes.code){
  data = final_EPA_data %>% filter(Pollutant == i)
  print(summary(data$value)) #summary of values for each pollutant, for all counties i.e. all of CA, al
  AQI = data$AQI
}

##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 0.0120  0.0140  0.0160  0.0184  0.0190  0.0655
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 0.2292  0.3000  0.3458  0.3719  0.4333  0.6500
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 0.2636  0.4864  0.6545  0.6448  0.7773  1.1318
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 3.402   4.939   7.139   8.351   11.071  21.668
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 0.008118 0.020000 0.026412 0.025109 0.028948 0.037000
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 7.00    13.00   18.00   20.09   25.00   70.50
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 4.200   6.700   7.800   8.546   9.600   23.350

summary(AQI)

##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 25.00   33.00   36.00   38.34   40.50   75.00

print("Summary for SDI data")

## [1] "Summary for SDI data"
summary(soa.data$Score) #summary of SDI score, for all counties i.e. all of CA, all years 2010-2019

##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 67.38   93.45  106.95  112.53  131.69  183.67

```

Reformatting the data set (mainly aggregating)

First, we wanted to combine the number of deaths from the two different causes into a total number of respiratory related deaths for each age group. Then, we combined the total number of deaths for each age group for a given month and county in a singular total for that month and county.

```
#Aggregating by cause of death
data = mortality3

data1 = data %>% filter(Cause_of_Death == "Chronic lower respiratory diseases")
data2 = data %>% filter(Cause_of_Death == "Influenza and pneumonia")

newdeaths = data1$Total_Deaths + data2$Total_Deaths
data1$Total_Deaths = newdeaths

respmortality = data1[,-5]

#Creates total deaths by adding deaths of all age groups together
agg respmortality = respmortality[1,]
agg respmortality$Age = as.character(agg respmortality$Age)
rows2 = seq(1,nrow(respmortality),11)

for (i in rows2){
  agg respmortality[i,] = respmortality[i,]
  agg respmortality$Total_Deaths[i] = sum(respmortality$Total_Deaths[i:(i+10)])
  agg respmortality$Age[i] = "Everyone"
}

agg respmortality = na.omit(agg respmortality)
rownames(agg respmortality) = NULL
total respmortality = agg respmortality[, -c(6:9)]
```

Reformat dataset into time series format (rows are counties, columns are months)

Total deaths:

```
months = unique(total respmortality$Month_of_Death)
years = sort(unique(total respmortality$Year_of_Death))
counties = unique(total respmortality$County_of_Death)
x = 0

total mortality.ts = matrix(1, nrow = 58, ncol = 72)

for (k in counties){
  county.ts = c()
  x = x+1

  for (i in years){
    for (j in months){
      deaths = total respmortality %>% filter(County_of_Death == k) %>% filter(Year_of_Death == i) %>%
        county.ts = c(county.ts, deaths)
    }
  }

  total mortality.ts[x,] = county.ts
```

```

}

#Label time series data
total.mortality.ts = as.data.frame(total.mortality.ts)

dates = c()
x=1
for (i in years){
  for (j in months){
    dates[x] = sprintf("%1.0f/%1.0f",j,i)
    x = x+1
  }
}

colnames(total.mortality.ts) = dates
ID = c(1:58)
total.mortality.ts = cbind(ID,counties,total.mortality.ts)

total.mortality.ts = left_join(clusterlabels,total.mortality.ts,by = "counties")
# total.mortality.ts = left_join(clusterlabels2,total.mortality.ts,by = "counties")
head(total.mortality.ts)

##   counties Cluster ID 1/2014 2/2014 3/2014 4/2014 5/2014 6/2014 7/2014 8/2014
## 1   Alameda     1   1    84    61    57    57    55    44    44    44
## 2   Alpine      4   2     0     0     0     0     0     0     0     0
## 3   Amador      4   3     2     3     2     2     2     1     3     1
## 4   Butte       2   4    15    10    11     8     5     6     7     6
## 5 Calaveras     4   5     3     4     2     3     0     0     1     0
## 6 Colusa       5   6     0     1     1     0     1     0     0     0
## 9/2014 10/2014 11/2014 12/2014 1/2015 2/2015 3/2015 4/2015 5/2015 6/2015
## 1   42      51      50     62    101    72    63    61    63    46
## 2     0      0      0     0     0     0     0     0     0     0
## 3     1      3      2     1     3     3     3     5     4     4
## 4     7      5      7     9    13     8    10    19     6     8
## 5     1      2      1     1     2     0     2     2     1     2
## 6     1      0      0     0     1     2     0     1     0     0
## 7/2015 8/2015 9/2015 10/2015 11/2015 12/2015 1/2016 2/2016 3/2016 4/2016
## 1   44      40      36     41    41    64    78    56    62    55
## 2     0      0      0     0     0     0     0     0     0     0
## 3     2      2      2     2     2     4     3     3     2     2
## 4     6     10      3     7     8     6    13     6     8     7
## 5     2      1      2     0     1     4     3     2     4     2
## 6     0      0      1     1     1     2     1     0     1     0
## 5/2016 6/2016 7/2016 8/2016 9/2016 10/2016 11/2016 12/2016 1/2017 2/2017
## 1   39      44      47     43    37    53    48    60   114    66
## 2     0      0      0     0     0     1     0     0     0     0
## 3     0      0      1     1     2     0     2     2     4     2
## 4   18      4      4     6     7     6     8     8    11    11
## 5     3      2      3     2     1     0     0     3     2     1
## 6     0      0      0     0     0     0     0     2     0     1
## 3/2017 4/2017 5/2017 6/2017 7/2017 8/2017 9/2017 10/2017 11/2017 12/2017
## 1   62      59      45     45    48    37    41    40    44    62
## 2     0      0      0     0     0     0     0     0     0     0
## 3     1      3      3     3     4     2     2     1     1     2

```

```

## 4    11    10     8    10     6     8     5    10     7     9
## 5     2     4     2     2     1     2     1     0     0     0     3
## 6     0     0     0     1     0     1     0     0     0     0     1
##   1/2018 2/2018 3/2018 4/2018 5/2018 6/2018 7/2018 8/2018 9/2018 10/2018
## 1    114     58     62     52     50     46     49     39     39     49
## 2     0     0     0     0     0     0     0     0     0     0     0
## 3     6     2     2     0     2     3     1     1     2     3
## 4    13     10     10     9     8     7     6     8     7     8
## 5     3     4     3     2     2     2     1     1     5     3
## 6     0     0     1     1     2     0     0     0     1     0
##   11/2018 12/2018 1/2019 2/2019 3/2019 4/2019 5/2019 6/2019 7/2019 8/2019
## 1    41     49     60     61     56     60     38     43     48     45
## 2     1     0     0     0     0     0     0     0     0     0
## 3     3     2     4     2     2     4     0     1     3     0
## 4     8     6    10     8     8     7     7     8     6     6
## 5     2     1     3     3     2     1     1     4     0     2
## 6     0     0     2     0     0     1     1     1     0     1
##   9/2019 10/2019 11/2019 12/2019
## 1    38     44     47     68
## 2     0     0     0     0
## 3     2     1     2     3
## 4     5     5     8     7
## 5     3     1     1     0
## 6     1     0     1     2

```

Exploring the aggregated data

HOW MANY 0s DOES EACH COUNTY HAVE?

```

numzeros_total = c()
for (i in 1:58){
  numzeros_total[i] = length(which(total.mortality.ts[i,3:74] == 0))
}

numzeros_total

## [1]  0 69  6  0 10 42  0  8  0  0 32  0  0 20  0  1  0 31  0  0  0 31  1  0 25
## [26] 62  0  0  1  0  0 25  0  0 13  0  0  0  0  0  0  0  0  0  0 61  2  0  0  0
## [51]  3  2 38  0  3  0  0  0

propzeros_total = numzeros_total/72

length(which(propzeros_total > 0.85))

## [1] 2

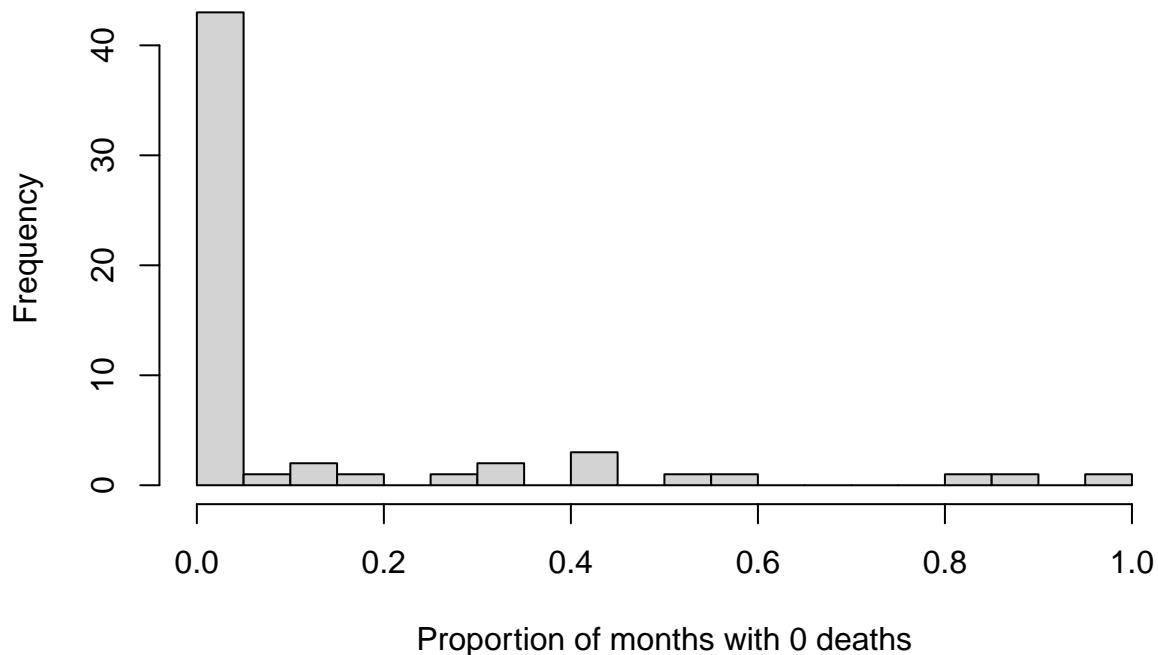
countycodes$value_represented[which(propzeros_total > 0.75)]

## [1] "Alpine" "Mono"   "Sierra"

hist(propzeros_total, breaks = 20 , xlab = "Proportion of months with 0 deaths", main = "Do some counties have many months with 0 deaths?")

```

Do some counties have more strings of 0s than others?



One aspect of the data that we wanted to examine before proceeding with our analysis was the frequency in which there were 0 deaths in a given county for a month. This would inform us about whether a standard Poisson model or a zero inflated Poisson model would be more appropriate. What I did above was first calculate the proportion of months (out of 72) that had 0 deaths observed for each county. Then identified which counties had a proportion of 0s greater than 75%, 85%, etc. Then, I made a histogram which shows there are only a couple of counties (which have very small populations) that had a high frequency of 0s. The aggregation performed in previous steps addressed the zero inflation it appears.

AGGREGATING MORTALITY DATA INTO CLUSTERS AS OPPOSED TO EACH COUNTY (ALSO AGGREGATED TO MORTALITY RATE PER CLUSTER)

Again, we want the number of respiratory related deaths at the cluster level, not the county level. So we once again aggregate the observations for each county in a given cluster. First, we simply add all the observations in cluster together to get a total number of respiratory related deaths for the months of 2014-2019 for each cluster. Then, we also calculated a mortality rate (per 100k people) for each cluster. This was done by taking the total number of deaths for a given cluster and dividing it by the total population of that cluster times 100,000 i.e. $(\text{deaths} * \frac{100000}{\text{clusterpop}})$. This second dataset will be used for our Gaussian process regression model which needs to be fit on a continuous response variable.

```
#County populations by year pulled from SoA data
countypops = CA_data %>% filter(Year > 2013) %>% select(Total_Pop, County, Year) %>% unique()
countypops = cbind(countypops, Cluster = rep(clusterlabels$Cluster, each=6))

cluster_mortality_total = matrix(NA, nrow = 72, ncol = num_clus)

cluster_mortality_rate = matrix(NA, nrow = 72, ncol = num_clus)

for (i in 1:num_clus){
```

```

cluster = total.mortality.ts %>% filter(Cluster == i)
year = 2014

for(j in 1:72){
  col = cluster[,j+3]

  #Sum of deaths across counties in a cluster
  cluster_mortality_total[j,i] = sum(na.omit(col))

  #Rate of deaths (per 100,000) across counties in a cluster
  pops = countypops %>% filter(Year == year,Cluster == i) %>% select(Total_Pop)
  cluster.pop = sum(pops)
  cluster_mortality_rate[j,i] = (sum(na.omit(col))/cluster.pop)*100000

  if ((j>12) & (j<25)){
    year = 2015
  }

  else if ((j>24) & (j<37)){
    year = 2016
  }

  else if ((j>36) & (j<49)){
    year = 2017
  }

  else if ((j>48) & (j<61)){
    year = 2018
  }

  else if ((j>60) & (j<73)){
    year = 2019
  }

  else{
    year = 2014
  }
}

#Time series of total deaths for each cluster
colnames(cluster_mortality_total) = c("Cluster 1","Cluster 2","Cluster 3",
                                      "Cluster 4","Cluster 5","Cluster 6",
                                      "Cluster 7")
rownames(cluster_mortality_total) = colnames(total.mortality.ts[4:75])
cluster_mortality_total = data.frame(cluster_mortality_total)

head(cluster_mortality_total)

```

	Cluster.1	Cluster.2	Cluster.3	Cluster.4	Cluster.5	Cluster.6	Cluster.7
## 1/2014	601	80	80	310	55	15	1101
## 2/2014	465	51	44	250	39	6	867
## 3/2014	437	47	42	234	32	15	854
## 4/2014	395	45	38	203	36	12	705

```

## 5/2014      360      45      36     195      31      17     710
## 6/2014      308      37      43     166      25       9     663

```

```

#Time series of rate of deaths (per 100,000) for each cluster
colnames(cluster_mortality_rate) = c("Cluster 1", "Cluster 2", "Cluster 3",
                                      "Cluster 4", "Cluster 5", "Cluster 6",
                                      "Cluster 7")
rownames(cluster_mortality_rate) = colnames(total.mortality.ts[4:75])
cluster_mortality_rate = data.frame(cluster_mortality_rate)

head(cluster_mortality_rate)

```

```

##           Cluster.1 Cluster.2 Cluster.3 Cluster.4 Cluster.5 Cluster.6 Cluster.7
## 1/2014  5.644215  6.635007  6.711308  5.944301  5.431745  6.013784  5.891607
## 2/2014  4.366988  4.229817  3.691219  4.793791  3.851601  2.405513  4.639440
## 3/2014  4.104030  3.898066  3.523437  4.486989  3.160288  6.013784  4.569875
## 4/2014  3.709592  3.732191  3.187871  3.892558  3.555324  4.811027  3.772555
## 5/2014  3.380894  3.732191  3.020089  3.739157  3.061529  6.815621  3.799310
## 6/2014  2.892543  3.068691  3.607328  3.183077  2.468975  3.608270  3.547807

```

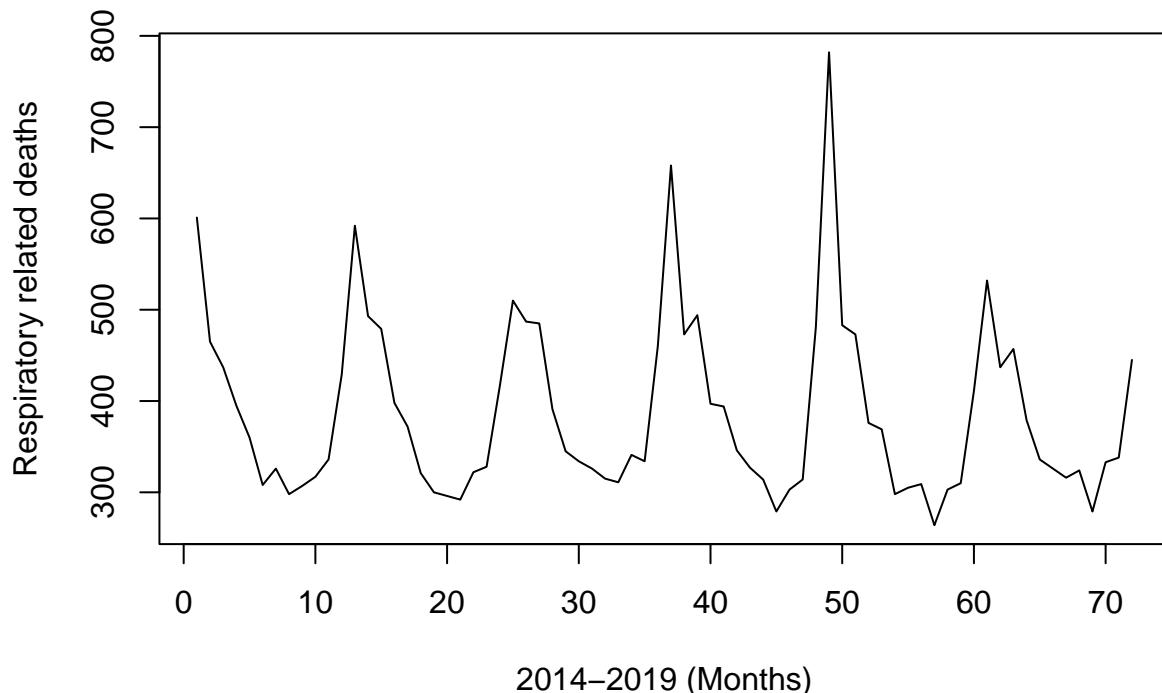
MAKE A TIME SERIES FOR EACH CLUSTER:

```

#Mortality
plot(ts(cluster_mortality_total$Cluster.1), xlab = "2014–2019 (Months)", ylab = "Respiratory related deaths")

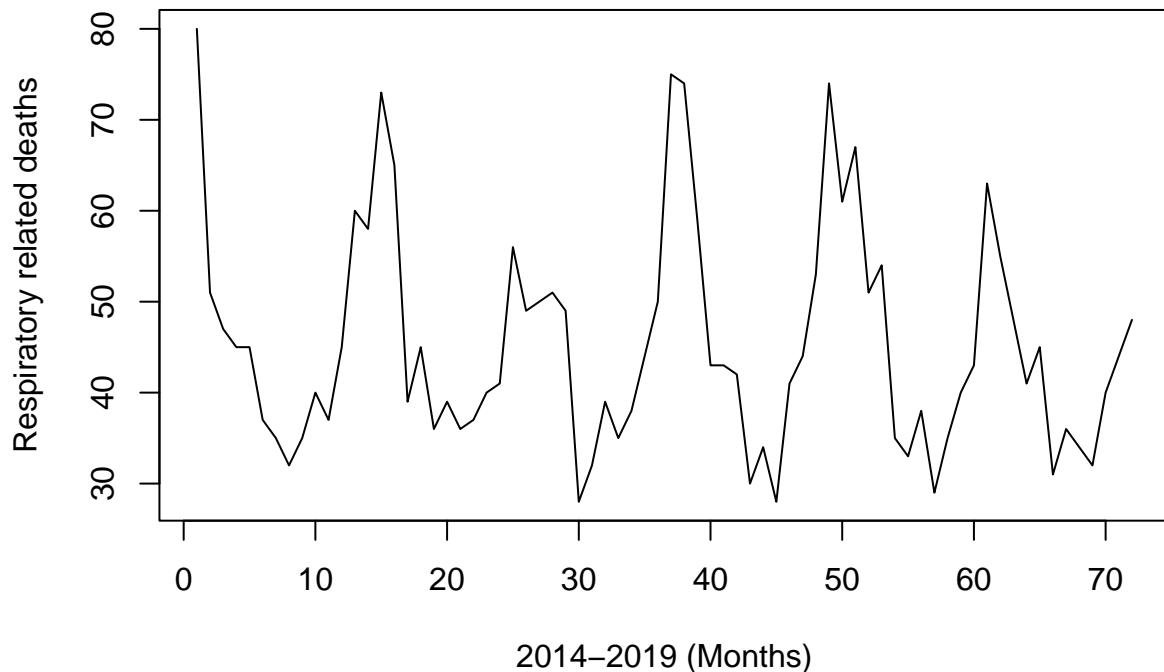
```

Monthly population weighted means for Cluster 1



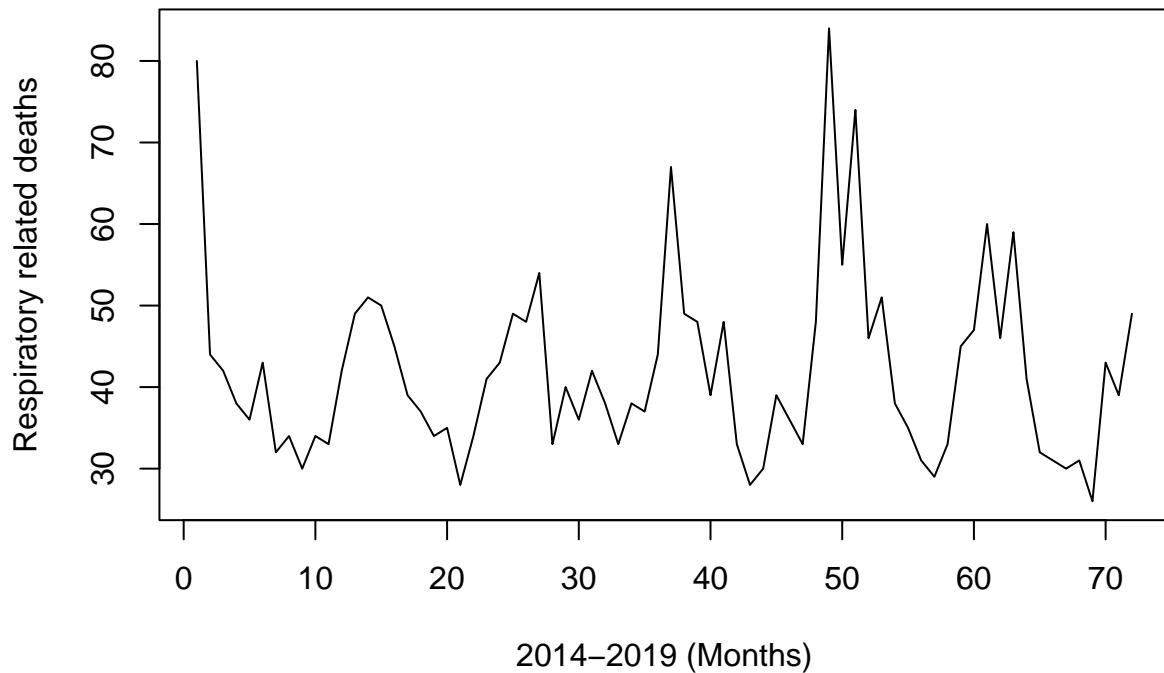
```
plot(ts(cluster_mortality_total$Cluster.2),xlab = "2014-2019 (Months)",ylab = "Respiratory related deat
```

Monthly population weighted means for Cluster 2



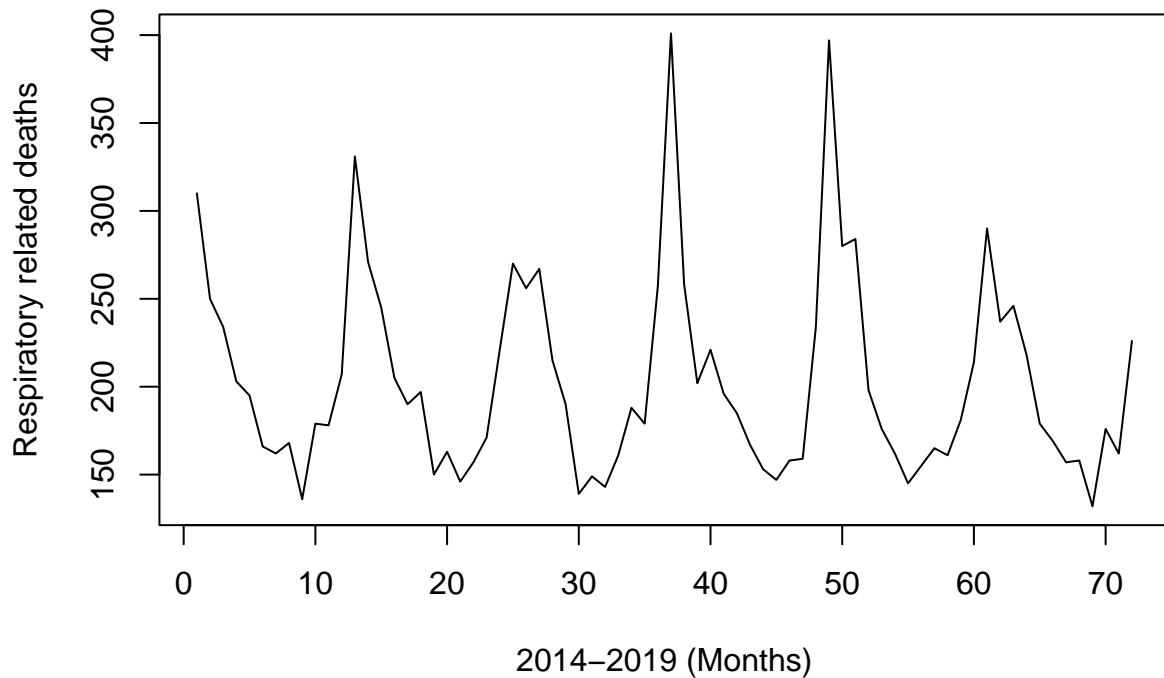
```
plot(ts(cluster_mortality_total$Cluster.3),xlab = "2014-2019 (Months)",ylab = "Respiratory related deat
```

Monthly population weighted means for Cluster 3



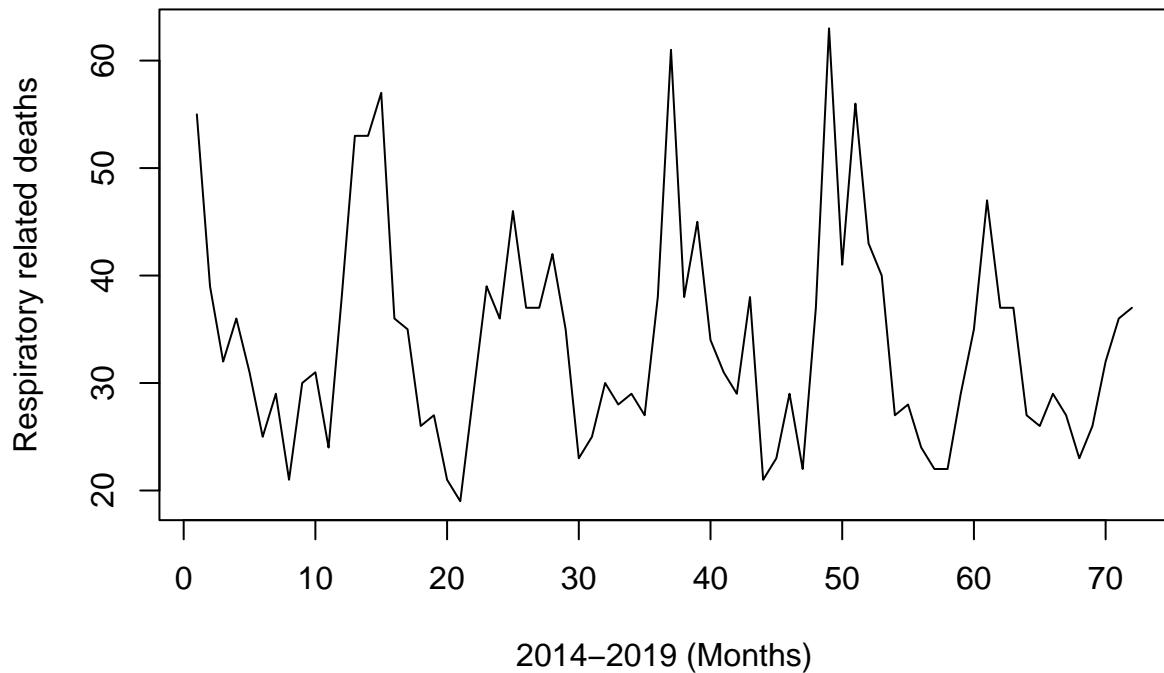
```
plot(ts(cluster_mortality_total$Cluster.4),xlab = "2014–2019 (Months)",ylab = "Respiratory related deaths")
```

Monthly population weighted means for Cluster 4



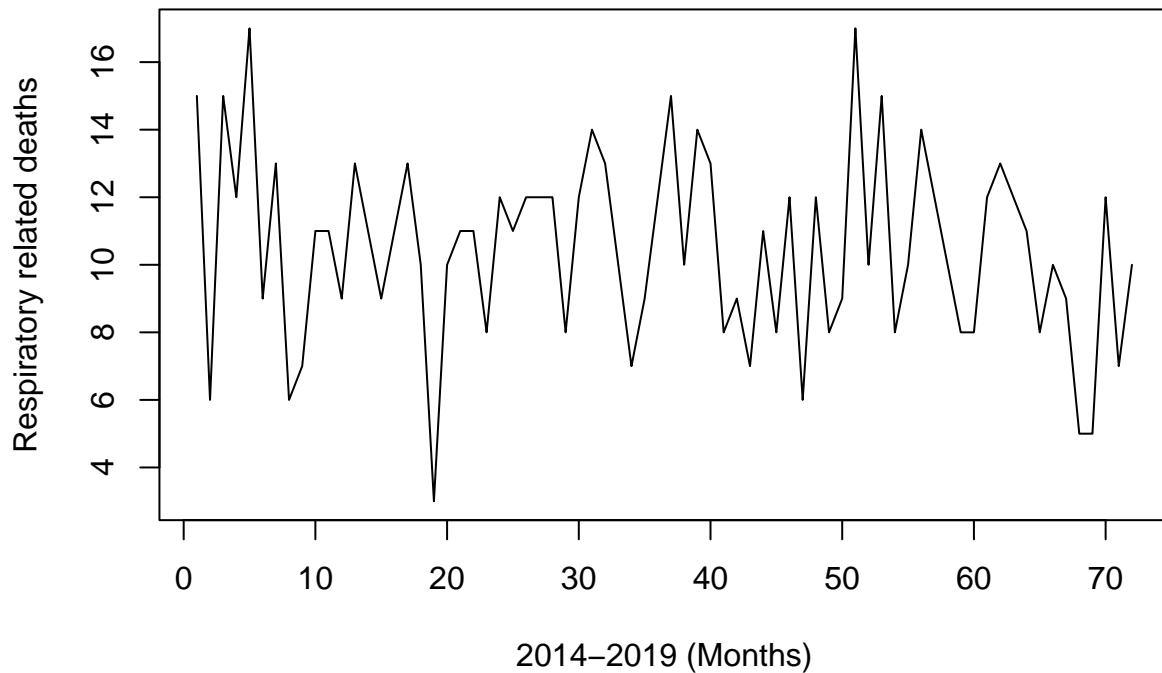
```
plot(ts(cluster_mortality_total$Cluster.5),xlab = "2014–2019 (Months)",ylab = "Respiratory related deaths")
```

Monthly population weighted means for Cluster 5



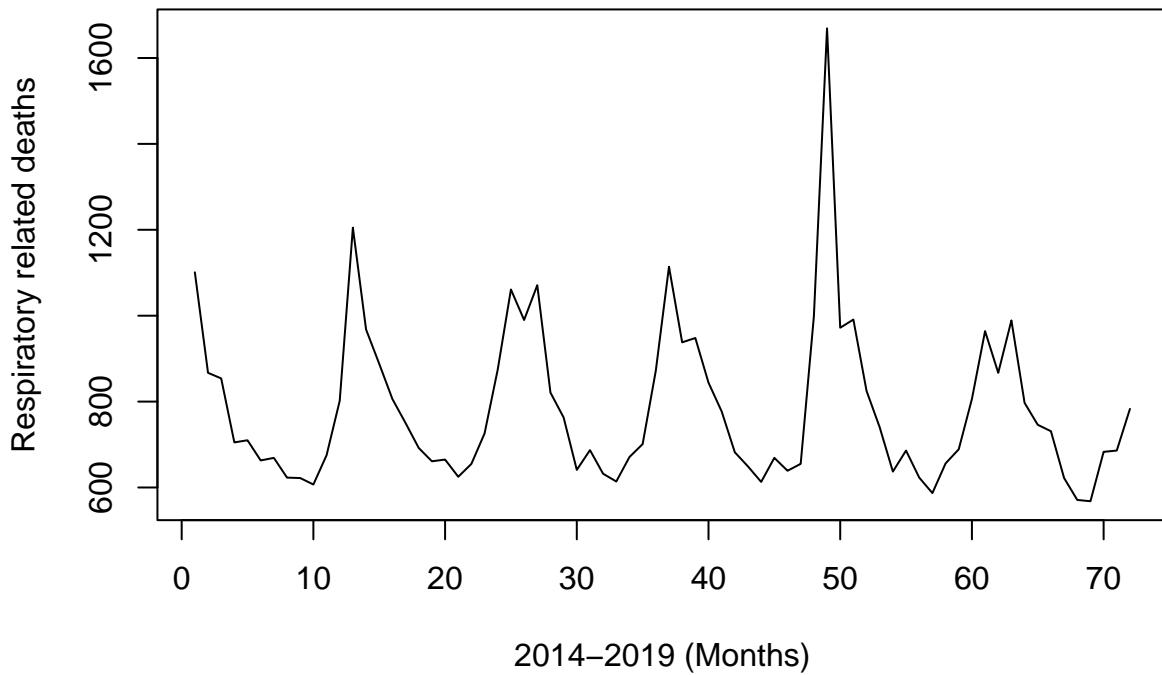
```
plot(ts(cluster_mortality_total$Cluster.6),xlab = "2014–2019 (Months)",ylab = "Respiratory related deaths")
```

Monthly population weighted means for Cluster 6



```
plot(ts(cluster_mortality_total$Cluster.7),xlab = "2014–2019 (Months)",ylab = "Respiratory related deaths")
```

Monthly population weighted means for Cluster 7



Grouped time series plot for important variables:

```
deaths_df = data.frame()

for(i in 1:num_clus){
  deaths = cluster_mortality_total[,i]
  Cluster = rep(i,length(deaths))
  month = seq(1,length(deaths))

  deaths_df = rbind(deaths_df,cbind(month,Cluster,deaths))
}

deaths_df = deaths_df %>% filter(month >= 13)
dates = as.Date("2014-01-01") + months(as.numeric(deaths_df$month) - 1)

# Define tick marks
jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
all_breaks <- sort(c(jan_breaks, jul_breaks))

deaths_df$date = dates
deaths_df$Cluster = factor(deaths_df$Cluster)

distinct_colors <- c("red", "blue", "green", "purple", "orange", "yellow", "brown")

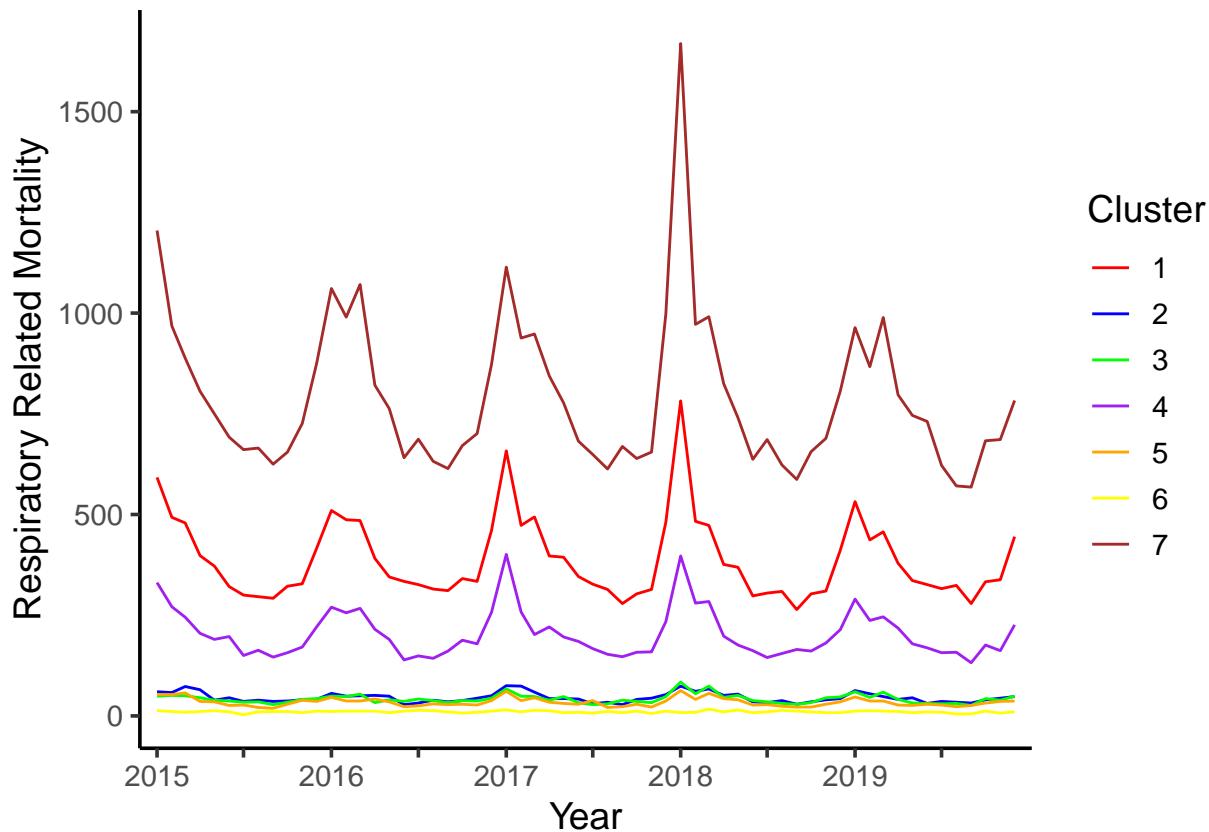
mortality_ts_plot = ggplot(data = deaths_df, aes(x = date, y = deaths, group = Cluster, colour = Cluste
  geom_line() +
```

```

  labs(y = "Respiratory Related Mortality", x = "Year") +
  scale_x_date(
    breaks = all_breaks,
    labels = ifelse(format(all_breaks, "%m") == "01", format(all_breaks, "%Y"), ""),
    expand = expansion(mult = 0.02)
  ) +
  scale_colour_manual(values = distinct_colors) +
  theme_classic(base_size = 14)

mortality_ts_plot

```



```

#AQI
AQI_df = data.frame()

for (i in 1:num_clus){
  aqi = final_EPA_agg_data %>% filter(Pollutant == "14129") %>% filter(Cluster == i) %>% select(AQI)
  Cluster = rep(i,length(aqi$AQI))
  month = seq(1,length(aqi$AQI))

  AQI_df = rbind(AQI_df, cbind(month,Cluster,aqi$AQI))
}

AQI_df = AQI_df %>% filter(month >= 13)
AQI_df$date = dates

AQI_df$Cluster = factor(AQI_df$Cluster)

```

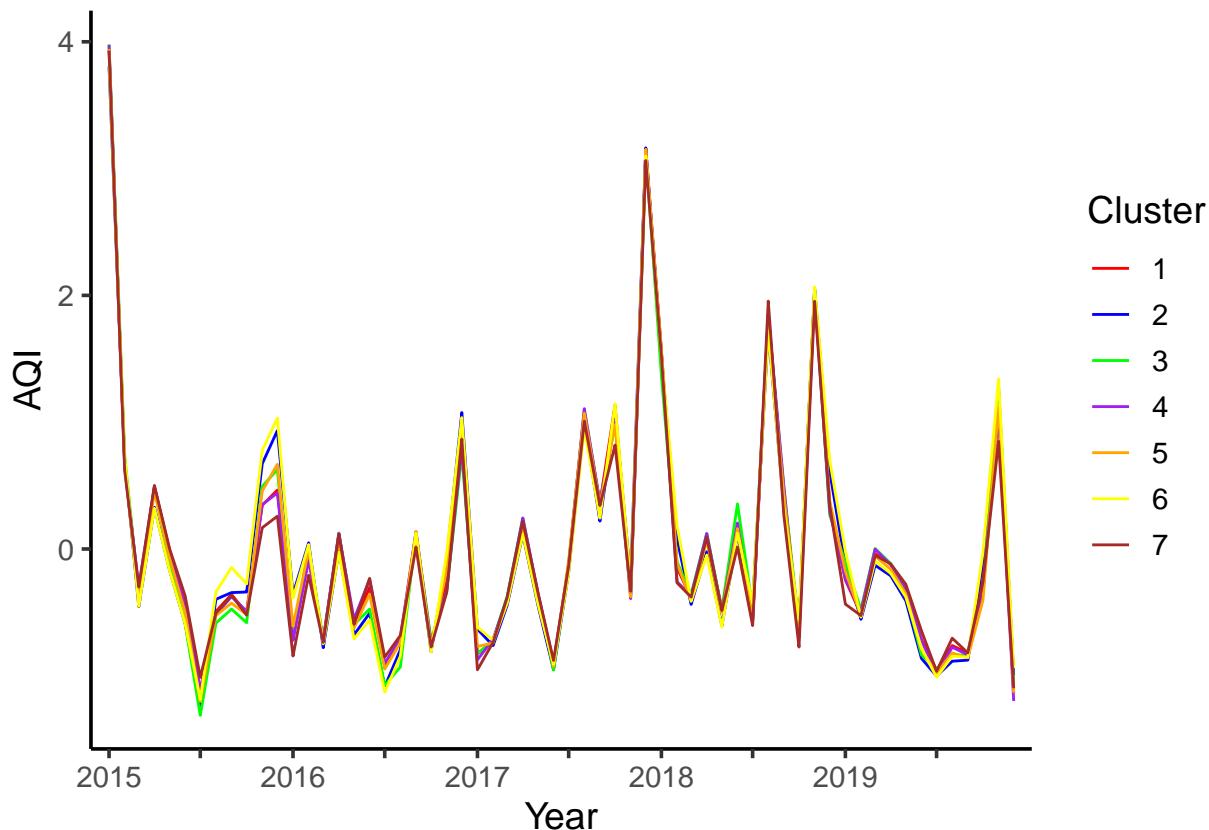
```

colnames(AQI_df)[3] = "aqi"

AQI_ts_plot = ggplot(data = AQI_df, aes(x=date,y=aqi,group=Cluster,colour=Cluster)) +
  geom_line() + labs(y="AQI",x="Year") +
  scale_x_date(
    breaks = all_breaks,
    labels = ifelse(format(all_breaks, "%m") == "01", format(all_breaks, "%Y"), ""),
    expand = expansion(mult = 0.02)
  ) + scale_colour_manual(values = distinct_colors) +
  theme_classic(base_size = 14)

AQI_ts_plot

```



```

#SDI
SDI_df = data.frame()

for (i in 1:num_clus){
  sdi = cluster_features[,i]
  Cluster = rep(i,length(sdi))
  year = seq(2010,2019)

  SDI_df = rbind(SDI_df,cbind(year,Cluster,sdi))
}

# SDI_df = SDI_df %>% filter(year >= 2015)

```

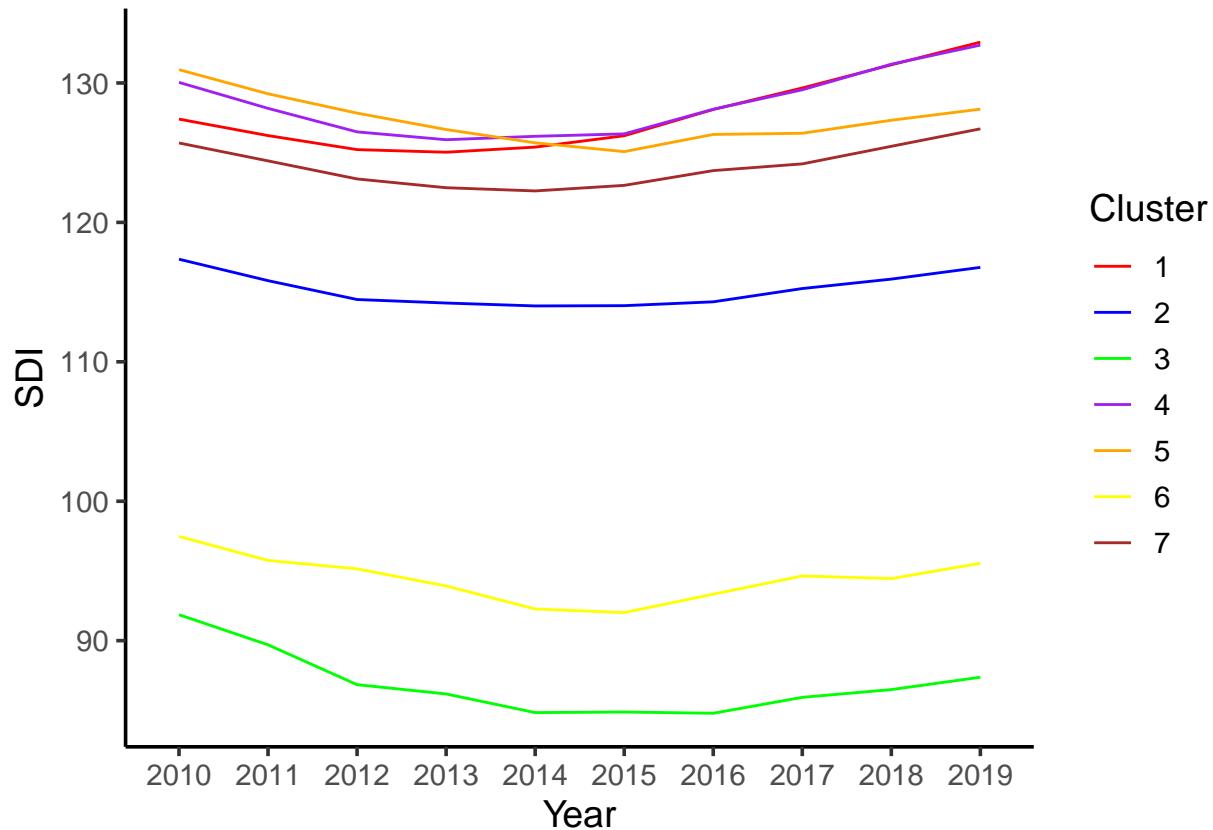
```

SDI_df$year = factor(SDI_df$year)
SDI_df$Cluster = factor(SDI_df$Cluster)

SDI_ts_plot = ggplot(data = SDI_df, aes(x=year,y=SDI,group=Cluster,colour=Cluster)) +
  geom_line() + labs(y="SDI",x="Year") +
  scale_colour_manual(values = distinct_colors) +
  theme_classic(base_size = 14)

SDI_ts_plot

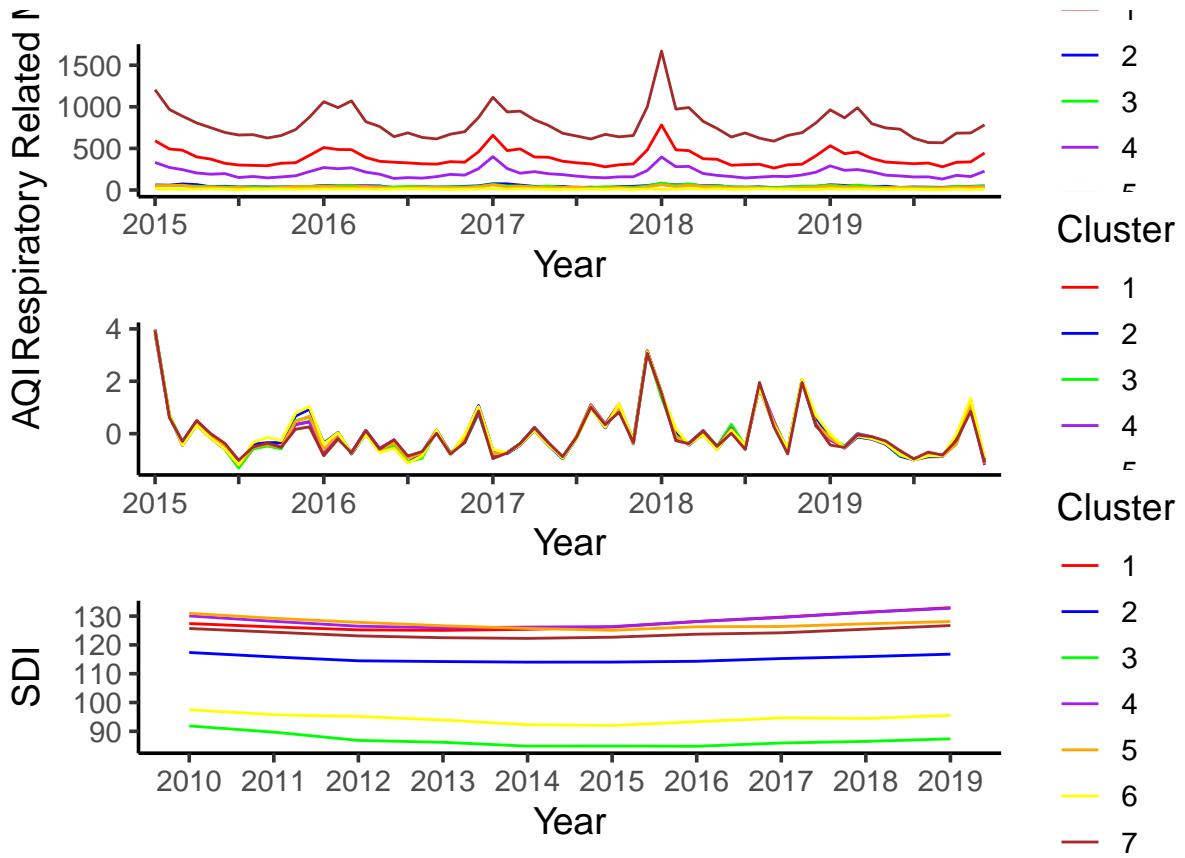
```



```

ts_plot_list = list(mortality_ts_plot,AQI_ts_plot,SDI_ts_plot)
combined = wrap_plots(plotlist = ts_plot_list, nrow = 3)
combined

```



```
# Save one combined figure
ggsave(
  filename = paste0("ts_combined_plot.png"),
  plot = combined,
  path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/Paper1-images/",
  width = 8,
  height = 10,
  dpi = 600
)
```

Fitting INLA models: spatial GLMM, Besag-York-Mollie (BYM) model, LGCPs

Now that all of the data from the SoA (used for SKATER and HUGE to get graph filter H), EPA (used to get gram matrix K), and Cal-ViDa (response) is downloaded, cleaned, and well formatted, we can now fit our kernel graph regression model as well as a few reference models, which we will compare against each other. For now, I have implemented a training-test data fitting approach to evaluating model performance.

First, I created an in sample dataset (`inla_insample_data`) which has variables ID (which represents the cluster label), ID2 (which is basically an index label), response (cluster mortality), time (time index label 1-66), and months (month label 1-12). Then, I create an out of sample dataset (`inla_outsample_data`) in which I decided to hold out the last 6 months of the data (67-72 or July-Dec 2019) so I replaced those response values with NAs. This is how you get INLA to make predictions/forecasts because it does so based on the posterior predictive distribution.

```
cluster_mortality_total_red = cluster_mortality_total[13:72,]
cluster_mortality_rate_red = cluster_mortality_rate[13:72,]
```

```

response = t(cluster_mortality_total_red)
response = as.vector(response)
response = ceiling(response)

response2 = t(cluster_mortality_rate_red)
response2 = as.vector(response2)

id = rep(c(1:7),60)
id2 = 1:(7*60)
time = rep(c(1:60),each = 7)

inla_full_data = data.frame(id,id2,response,time)

inla_full_data2 = data.frame(id,id2,response2,time)

months = rep(c(1:12),each = 7)
months = rep(months,5)
inla_full_data = cbind(inla_full_data,months)

#Experimented with defining each of these as factors
# inla_full_data$id = factor(inla_full_data$id)
# inla_full_data$id2 = factor(inla_full_data$id2)
# inla_full_data$time = factor(inla_full_data$time)
inla_full_data$months = factor(inla_full_data$months)

#Add multiple intercept columns, one for each cluster
Intercept1 = rep(c(1,NA,NA,NA,NA,NA,NA),60)
Intercept2 = rep(c(NA,1,NA,NA,NA,NA,NA),60)
Intercept3 = rep(c(NA,NA,1,NA,NA,NA,NA),60)
Intercept4 = rep(c(NA,NA,NA,1,NA,NA,NA),60)
Intercept5 = rep(c(NA,NA,NA,NA,1,NA,NA),60)
Intercept6 = rep(c(NA,NA,NA,NA,NA,1,NA),60)
Intercept7 = rep(c(NA,NA,NA,NA,NA,NA,1),60)

inla_full_data = cbind(inla_full_data,Intercept1,Intercept2,Intercept3,Intercept4,
                       Intercept5,Intercept6,Intercept7)

inla_full_data$pop = rep(cluster_pops,60)

# response = replace_na(response,0.01)
inla_gp_data = inla_full_data2

year = rep(2015:2019,each = 12)
inla_gp_data = cbind(inla_gp_data,year)

###Split into in sample and out of sample dataset
inla_outsample_data = inla_full_data

#Omit values for months 55-60 (out of sample dataset)
omit_idx = which(inla_outsample_data$time > 54)
inla_outsample_data$response[omit_idx] = NA
inla_insample_data = inla_full_data[-omit_idx,]

```

```

omit_idx = which(inla_gp_data$time > 54)
inla_gp_data$response2[omit_idx] = NA

Setting up data for reference model 4

lead = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 14129) %>%
  select(Value)
co = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 42101) %>%
  select(Value)
so2 = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 42401) %>%
  select(Value)
no2 = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 42602) %>%
  select(Value)
o3 = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 44201) %>%
  select(Value)
pm10 = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 81102) %>%
  select(Value)
pm25 = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 88101) %>%
  select(Value)
aqi = final_EPA_agg_data %>% filter(Time <= 60, Pollutant == 88101) %>%
  select(AQI)

inla_full_data_wcovariates = inla_full_data %>% arrange(id)
end = ncol(inla_full_data_wcovariates)

inla_full_data_wcovariates = cbind(inla_full_data_wcovariates, lead, co,
                                    so2, no2, o3, pm10, pm25, aqi)
colnames(inla_full_data_wcovariates)[(end+1):(ncol(inla_full_data_wcovariates))] = c("lead", "co", "so2",
                                                                                   "no2", "o3", "pm10", "pm25", "aqi")

inla_outsample_data_wcovariates = inla_full_data_wcovariates
omit_idx = which(inla_outsample_data_wcovariates$time > 54)

inla_outsample_data_wcovariates$response[omit_idx] = NA
inla_insample_data_wcovariates = inla_full_data_wcovariates[-omit_idx,]

```

IN SAMPLE FITTING ANALYSIS

Fit a simple Poisson GLMM for our mortality data (Reference model 1)

We wanted to compare the performance of our proposed model with a few reference models. The first one is a Poisson generalized linear mixed model. This model assumes the observed data follows a Poisson distribution and the hyperparameter λ_i can be modeled using a mixed effects model with a log link.

In other words,

$Y_{c,t} \sim Pois(\Lambda_{c,t})$ for $c = 1, \dots, 7$ and $t = 1, \dots, 54$ where $\Lambda_{c,t} = \exp(\beta_{c1}I\{c = 1\} + \beta_{c2}I\{c = 2\} + \dots + \beta_{C}I\{c = C\} + \beta_1I\{t \text{ mod } 12 = 1\} + \dots + \beta_{11}I\{t \text{ mod } 12 = 11\} + \mathbf{F}_c)$

where the random effect $\mathbf{F}|\tau \sim MVN(\mathbf{0}, \tau \Sigma)$

We wanted the first reference model to be simple, so we assumed that the random effects \mathbf{F}_c are iid. This means that Σ is simply a diagonal matrix of scaling factors. The hyperparameter $\log(\tau)$ is by default assigned a $\log \Gamma(1, 0.00005)$ prior.

```
#Write a function to fit our poisson glmm in INLA
ref_model1 = function(dataset,a_prior = 1,b_prior = 5e-05,link=1){
```

```

####Fit INLA model
prec_prior <- list(prec = list(prior = "loggamma", param = c(a_prior,b_prior)))
ref_formula1 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
    Intercept5 + Intercept6 + Intercept7 + f(id,model = "iid", hyper = prec_prior) #could use id or id2
model = inla(formula = ref_formula1,family = "poisson",data = dataset,
             control.compute = list(dic=TRUE,waic=TRUE),
             control.inla = list(strategy = "laplace"),
             control.predictor = list(compute = TRUE, link = link))

####Extract relevant information and store in the list

model_summary <- model$summary.fixed
bri_hyperpar_summary <- bri.hyperpar.summary(model)
modelDIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf,Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
    scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
    xlab("linear predictor")

#Store the results in the list
ref_model1_results = list(
    model_summary = model_summary,
    bri_hyperpar_summary = bri_hyperpar_summary,
    exp_effects = multeff,
    param_plot = param_plot,
    hyperparam_plot = hyperparam_plot,
    modelDIC = modelDIC,
    model_WAIC = model_WAIC,
    fitted_values = preds_model
)
return(ref_model1_results)
}

#Run model
ref_model1_fit = ref_model1(dataset = inla_insample_data,a_prior = 1,b_prior = 5e-5)

```

```

#Extract DIC and WAIC
ref_model1_DIC = ref_model1_fit$modelDIC
ref_model1_WAIC = ref_model1_fit$modelWAIC

#Get summaries of parameter estimates
ref_model1_fit$model_summary

##               mean        sd 0.025quant 0.5quant 0.975quant      mode
## months1    2.1039996 7.254715 -12.121777 2.1039996   16.32978 2.1039996
## months2    1.8618770 7.254716 -12.363902 1.8618770   16.08766 1.8618770
## months3    1.8812179 7.254716 -12.344561 1.8812179   16.10700 1.8812179
## months4    1.6930262 7.254718 -12.532756 1.6930262   15.91881 1.6930262
## months5    1.6124053 7.254718 -12.613378 1.6124053   15.83819 1.6124053
## months6    1.4969691 7.254719 -12.728816 1.4969691   15.72275 1.4969691
## months7    1.4681952 7.254722 -12.757595 1.4681952   15.69399 1.4681952
## months8    1.4344600 7.254723 -12.791331 1.4344600   15.66025 1.4344600
## months9    1.4008381 7.254723 -12.824954 1.4008381   15.62663 1.4008381
## months10   1.4724083 7.254722 -12.753382 1.4724083   15.69820 1.4724083
## months11   1.5162767 7.254722 -12.709513 1.5162767   15.74207 1.5162767
## months12   1.7814816 7.254719 -12.444302 1.7814816   16.00727 1.7814816
## Intercept1 4.2895288 7.254726 -9.936270 4.2895288   18.51533 4.2895288
## Intercept2 2.1524553 7.254748 -12.073386 2.1524553   16.37830 2.1524553
## Intercept3 2.0751099 7.254750 -12.150735 2.0751099   16.30096 2.0751099
## Intercept4 3.6523903 7.254729 -10.573413 3.6523903   17.87819 3.6523903
## Intercept5 1.8619677 7.254756 -12.363890 1.8619677   16.08783 1.8619677
## Intercept6 0.6814343 7.254831 -13.544569 0.6814343   14.90744 0.6814343
## Intercept7 5.0102689 7.254725 -9.215527 5.0102689   19.23606 5.0102689
##               kld
## months1    5.526823e-11
## months2    5.526817e-11
## months3    5.526817e-11
## months4    5.526817e-11
## months5    5.526817e-11
## months6    5.526811e-11
## months7    5.526817e-11
## months8    5.526823e-11
## months9    5.526823e-11
## months10   5.526817e-11
## months11   5.526817e-11
## months12   5.526817e-11
## Intercept1 5.526817e-11
## Intercept2 5.526815e-11
## Intercept3 5.526820e-11
## Intercept4 5.526819e-11
## Intercept5 5.526820e-11
## Intercept6 5.526821e-11
## Intercept7 5.526816e-11

ref_model1_fit$bri_hyperpar_summary

##               mean        sd       q0.025      q0.5       q0.975      mode
## SD for id 0.01149621 0.00966155 0.003662817 0.008442955 0.04014991 0.005741776
ref_model1_fit$exp_effects

```

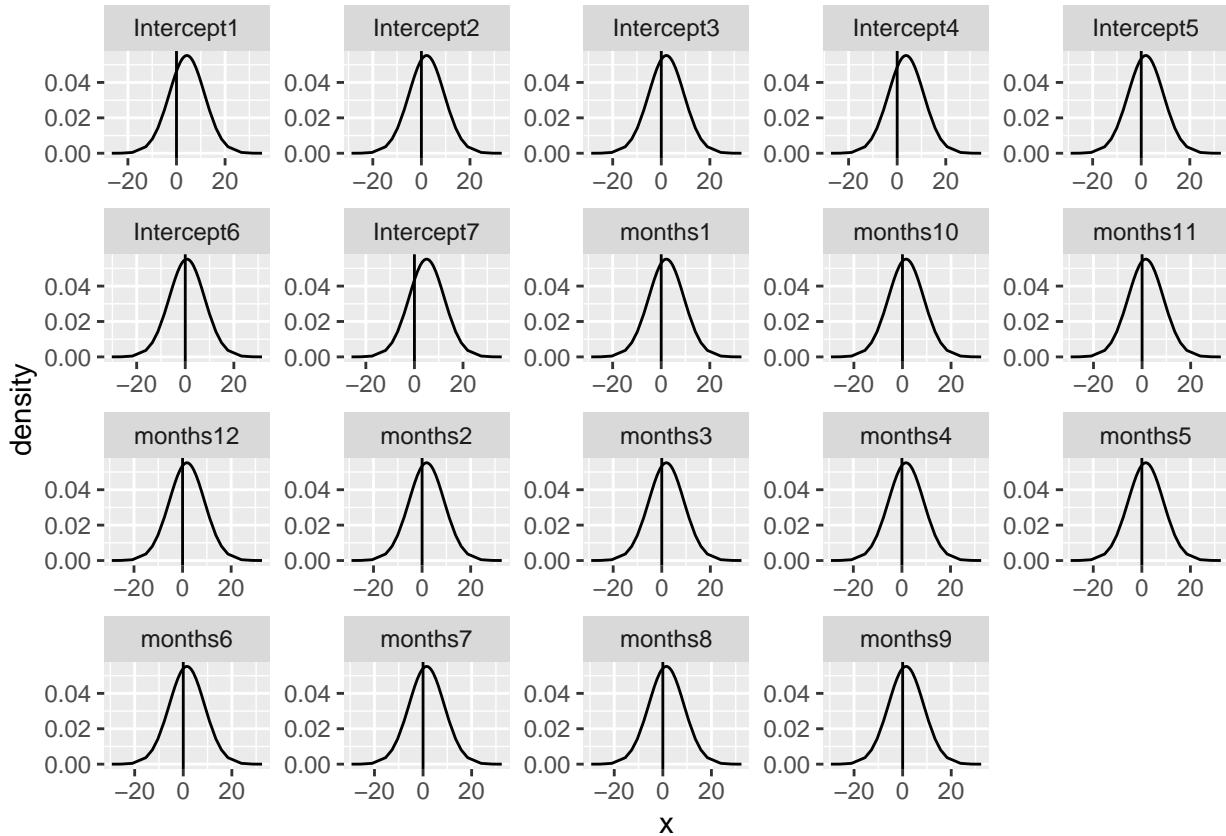
```

##    months1     months2     months3     months4     months5     months6     months7
## 8.198897  6.435806  6.561491  5.435906  5.014859  4.468126  4.341393
##    months8     months9    months10    months11    months12 Intercept1 Intercept2
## 4.197378  4.058600  4.359722  4.555233  5.938649  72.932093  8.605963
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
## 7.965422 38.566740  6.436389  1.976711 149.945049

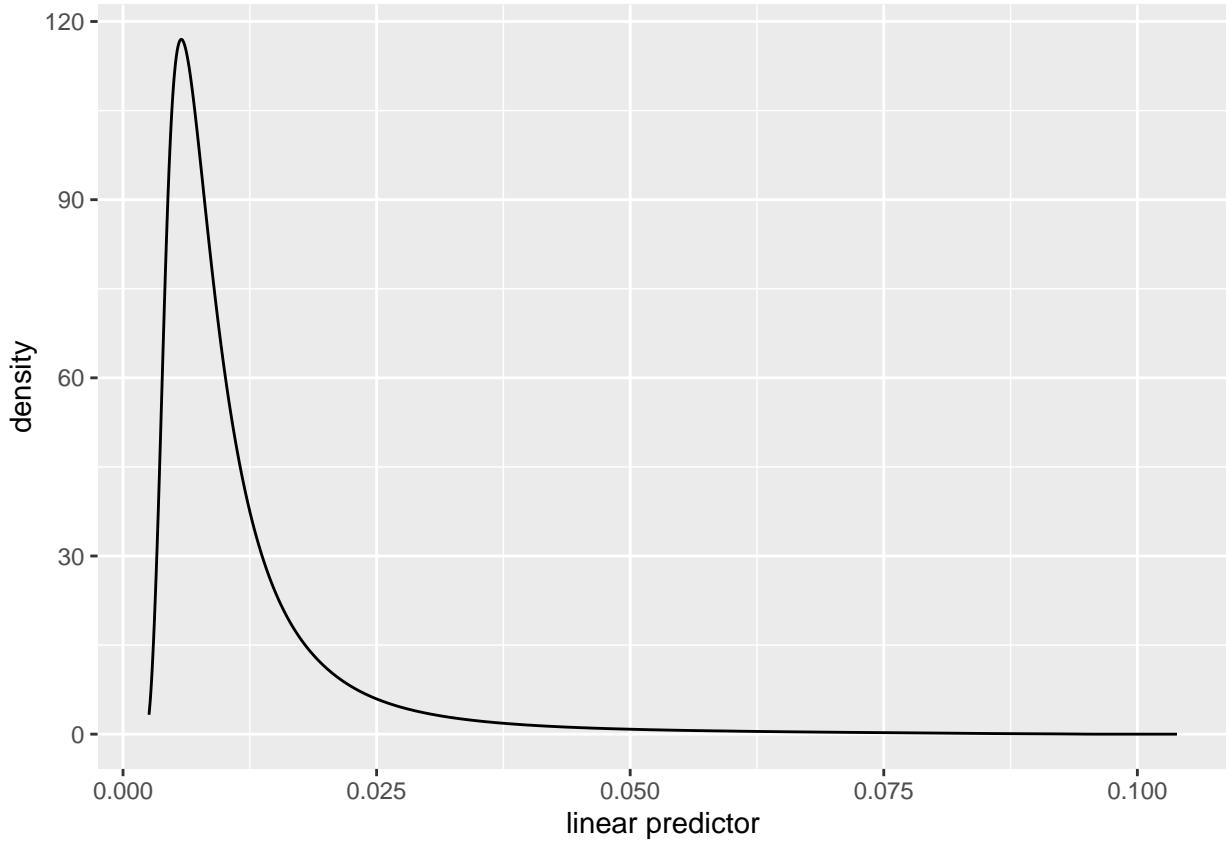
```

#Show plots

```
ref_model1_fit$param_plot
```



```
ref_model1_fit$hyperparam_plot
```



Note: SD for ID: standard deviation for the means (avg intensities) corresponding to the 8 different clusters was 0.116

Plot of posterior predictive estimates with credible interval bands OVERLAID on response:

```

poisson_pp_sampling = function(inla_fvs,n=100000){
  inla_fvs$y_ppmean = NA
  inla_fvs$y_pplower = NA
  inla_fvs$y_ppupper = NA

  for(i in 1:nrow(inla_fvs)){
    pp_samples = rpois(n,inla_fvs$mean[i])

    inla_fvs$y_ppmean[i] = mean(pp_samples)
    inla_fvs$y_pplower[i] = quantile(pp_samples,0.025)
    inla_fvs$y_ppupper[i] = quantile(pp_samples,0.975)
  }

  return(inla_fvs)
}

ref_model1_fit$fitted_values = poisson_pp_sampling(ref_model1_fit$fitted_values,n=100000)

#Write a function to make plot of posterior predictive estimates with credible interval bands OVERLAID
pp_insample_plot = function(num_plots = num_clus, ref_data = inla_insample_data, pred_data){
  for (i in 1:num_plots){
    df = ref_data %>% filter(id == i) %>% select(response)
  }
}

```

```

preds = pred_data %>% filter(id == i)
df = cbind(df,preds)
df$date = as.Date(paste0("2015-01-01")) + months(df$time - 1)

captured = (df$response >= df`^0.025quant` & df$response <= df`^0.975quant`)
coverage = round(sum(captured)/length(captured)*100,2)

# title = sprintf("Posterior Predictive Fits for Cluster %s",i)
title = sprintf("Cluster %s",i)

# post_pred_plot = df %>% ggplot(aes(x=time,y=response)) + geom_point() +
#   geom_line(aes(y=mean),color = "red") + geom_ribbon(aes(ymin = `0.025quant`,ymax = `0.975quant`),a
#   # print(post_pred_plot)
# }

}

pp_insample_plot = function(num_plots = num_clus, ref_data = inla_insample_data, pred_data, save_path =
  for (i in 1:num_plots){
  df = ref_data %>% filter(id == i) %>% select(response)
  preds = pred_data %>% filter(id == i)
  df = cbind(df,preds)
  df$date = as.Date(paste0("2015-01-01")) + months(df$time - 1)

  captured = (df$response >= df$y_pplower & df$response <= df$y_ppupper)
  coverage = round(sum(captured)/length(captured)*100,2)

  title = sprintf("Cluster %s (Coverage: %s%%)",i,coverage)

  # Define tick marks
  jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
  jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
  all_breaks <- sort(c(jan_breaks, jul_breaks))

  post_pred_plot = df %>%
    ggplot(aes(x = date, y = response)) +
    geom_point() +
    geom_line(aes(y = mean), color = "red") +
    geom_ribbon(aes(ymin = y_pplower, ymax = y_ppupper), alpha = 0.3) +
    ggtitle(title) +
    scale_x_date(
      name = "Year",
      breaks = all_breaks,
      labels = ifelse(format(all_breaks, "%m") == "01",
                     format(all_breaks, "%Y"), ""),
      expand = expansion(mult = 0.02) # ~2% space on each side
    ) +
    scale_y_continuous(name = "Respiratory Related Deaths") +
    theme_classic(base_size = 14)

  print(post_pred_plot)

  # Save plot automatically with prefix
  ggsave(

```

```

        filename = paste0(prefix, "cluster", i, ".png"),
        plot = post_pred_plot,
        path = save_path,
        width = 8,
        height = 6,
        dpi = 300
    )
}
}

library(patchwork)

pp_insample_plot_combined = function(num_plots = num_clus,
                                      ref_data = inla_insample_data,
                                      pred_data,
                                      save_path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inferen",
                                      prefix = ""){

plot_list = list()

for (i in 1:num_plots){
  df = ref_data %>% filter(id == i) %>% select(response)
  preds = pred_data %>% filter(id == i)
  df = cbind(df,preds)
  df$date = as.Date("2015-01-01") + months(df$time - 1)

  captured = (df$response >= df$y_pplower & df$response <= df$y_ppupper)
  coverage = round(mean(captured) * 100, 2)

  title = sprintf("Cluster %s (Coverage: %s%%)", i, coverage)

  # Define tick marks
  jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
  jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
  all_breaks <- sort(c(jan_breaks, jul_breaks))

  p = df %>%
    ggplot(aes(x = date, y = response)) +
    geom_point(size = 1) +
    geom_line(aes(y = mean), color = "red") +
    geom_ribbon(aes(ymin = y_pplower, ymax = y_ppupper), alpha = 0.3) +
    ggtitle(title) +
    scale_x_date(
      name = "Year",
      breaks = all_breaks,
      labels = ifelse(format(all_breaks, "%m") == "01",
                     format(all_breaks, "%Y"), ""),
      expand = expansion(mult = 0.02)
    ) +
    scale_y_continuous(name = "Deaths") +
    theme_classic(base_size = 14)

  plot_list[[i]] = p
}


```

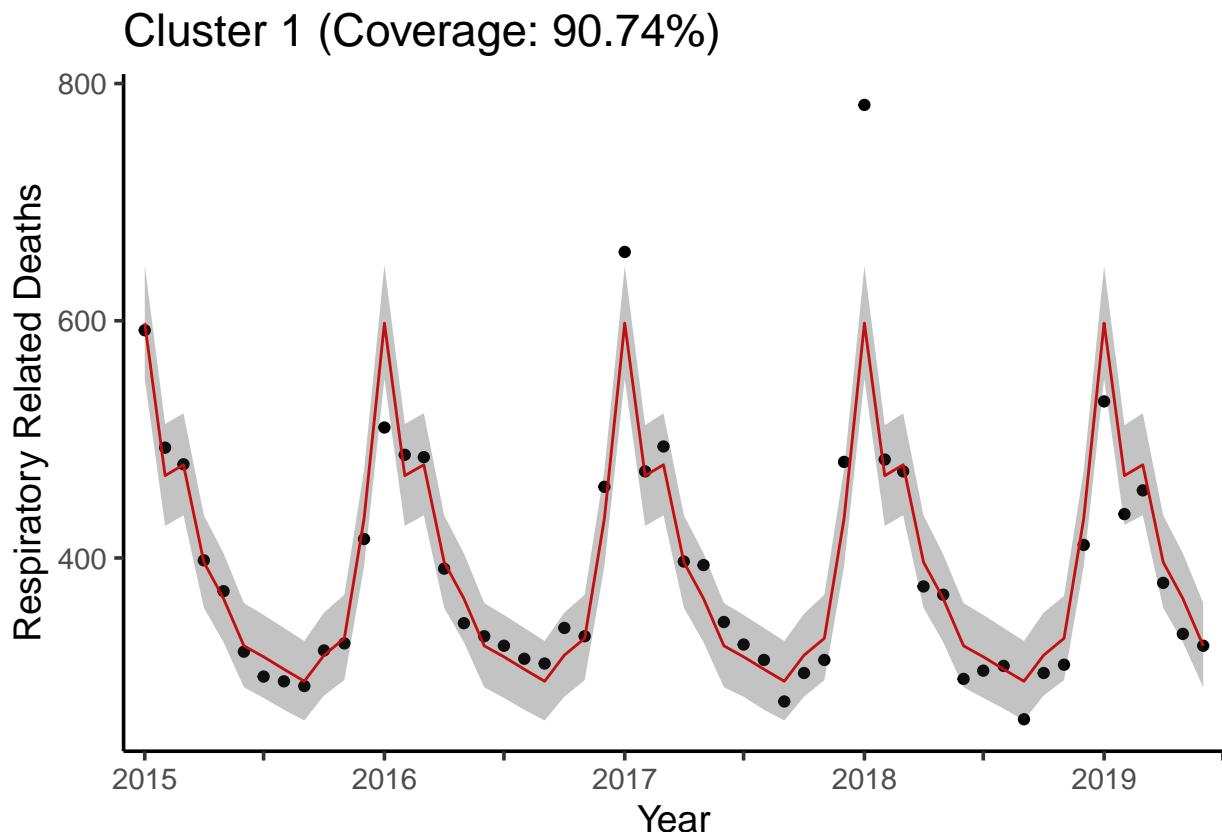
```

# Arrange plots in a grid (2 rows x 4 cols for up to 8 clusters)
combined = wrap_plots(plotlist = plot_list, ncol = 2)

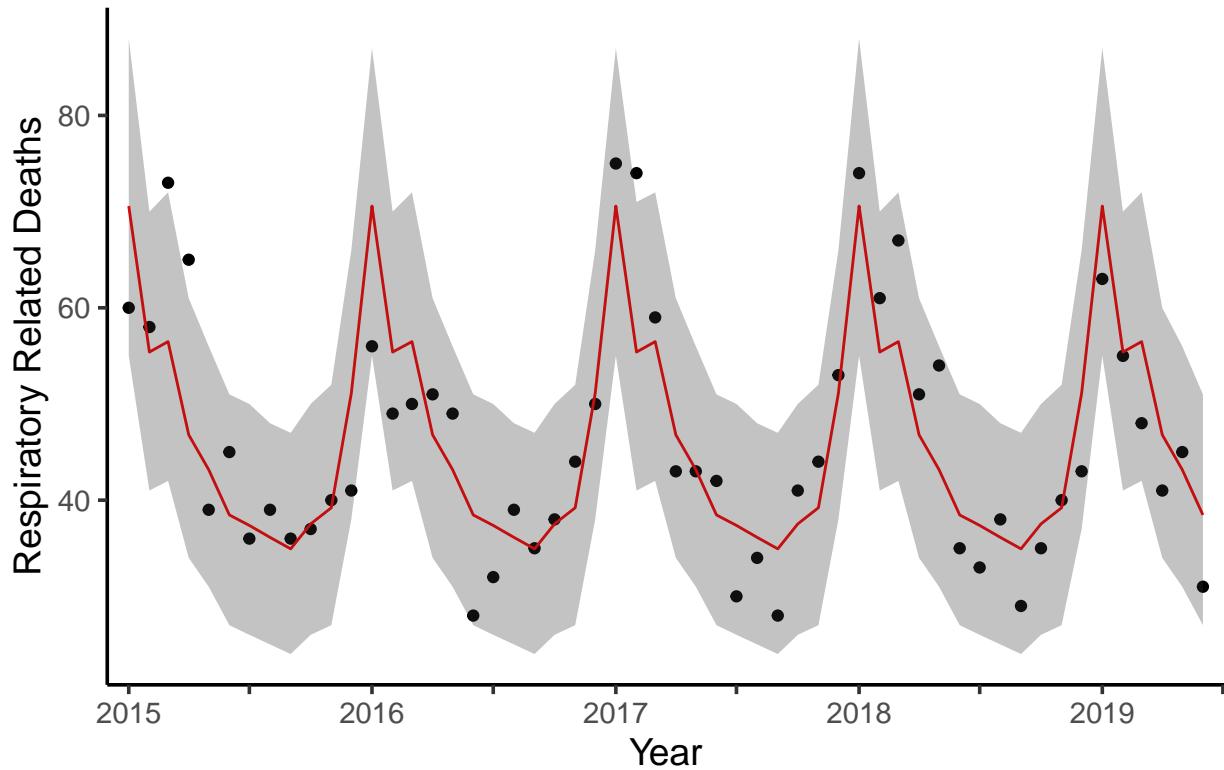
# Save one combined figure
ggsave(
  filename = paste0(prefix, "combined.png"),
  plot = combined,
  path = save_path,
  width = 10,
  height = 8,
  dpi = 600
)
}

#Plot ref_model1 pp plot
pp_insample_plot(pred_data = ref_model1_fit$fitted_values,prefix = "ref1-insample-")

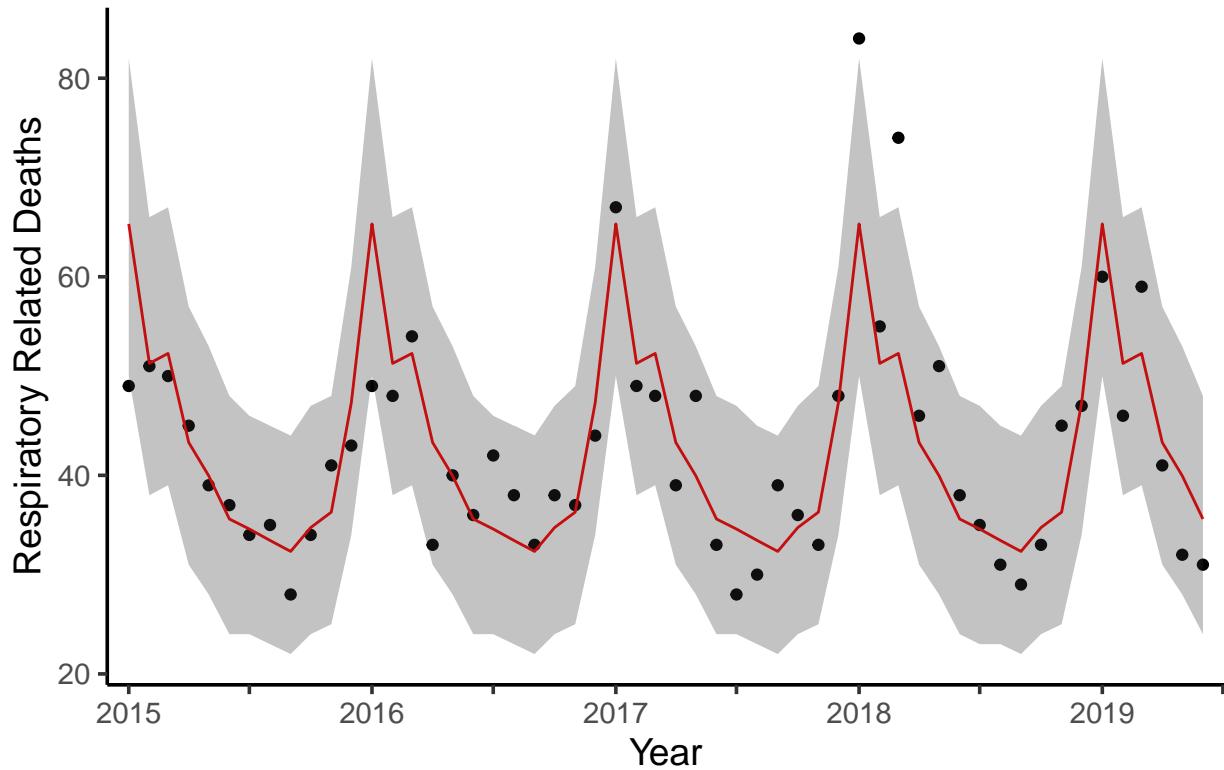
```



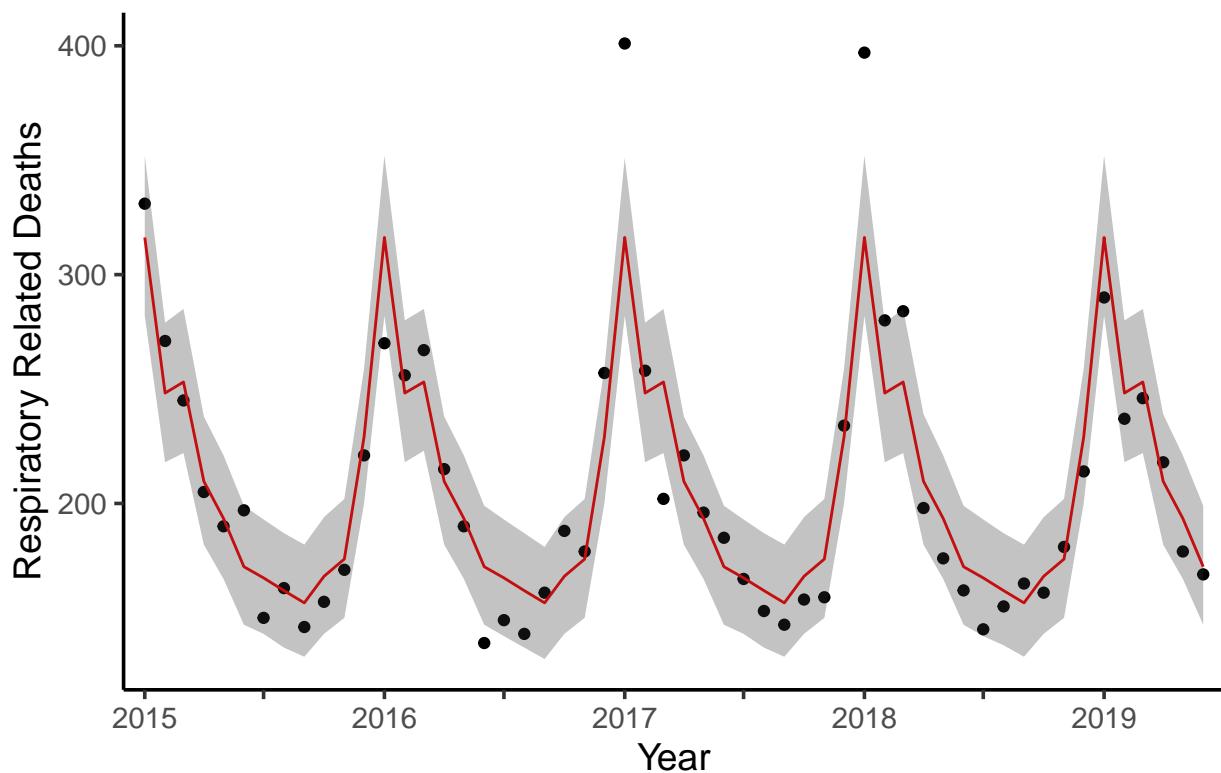
Cluster 2 (Coverage: 94.44%)



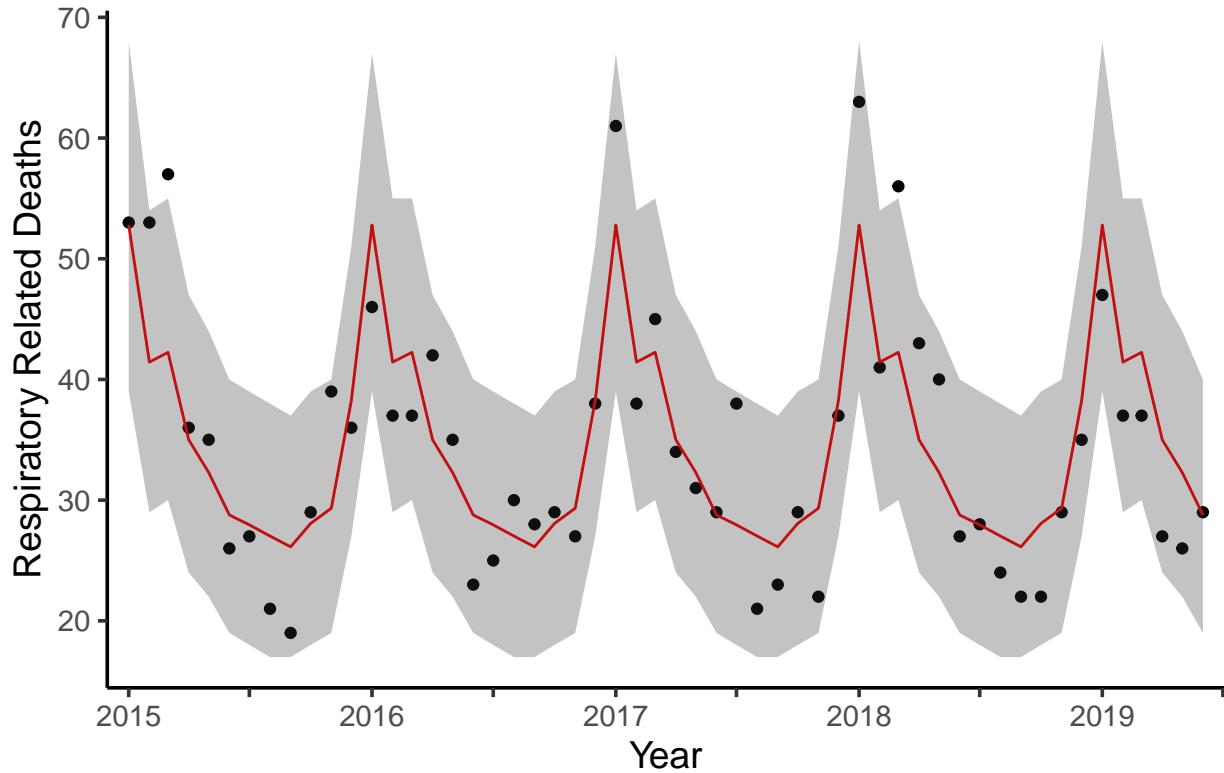
Cluster 3 (Coverage: 92.59%)



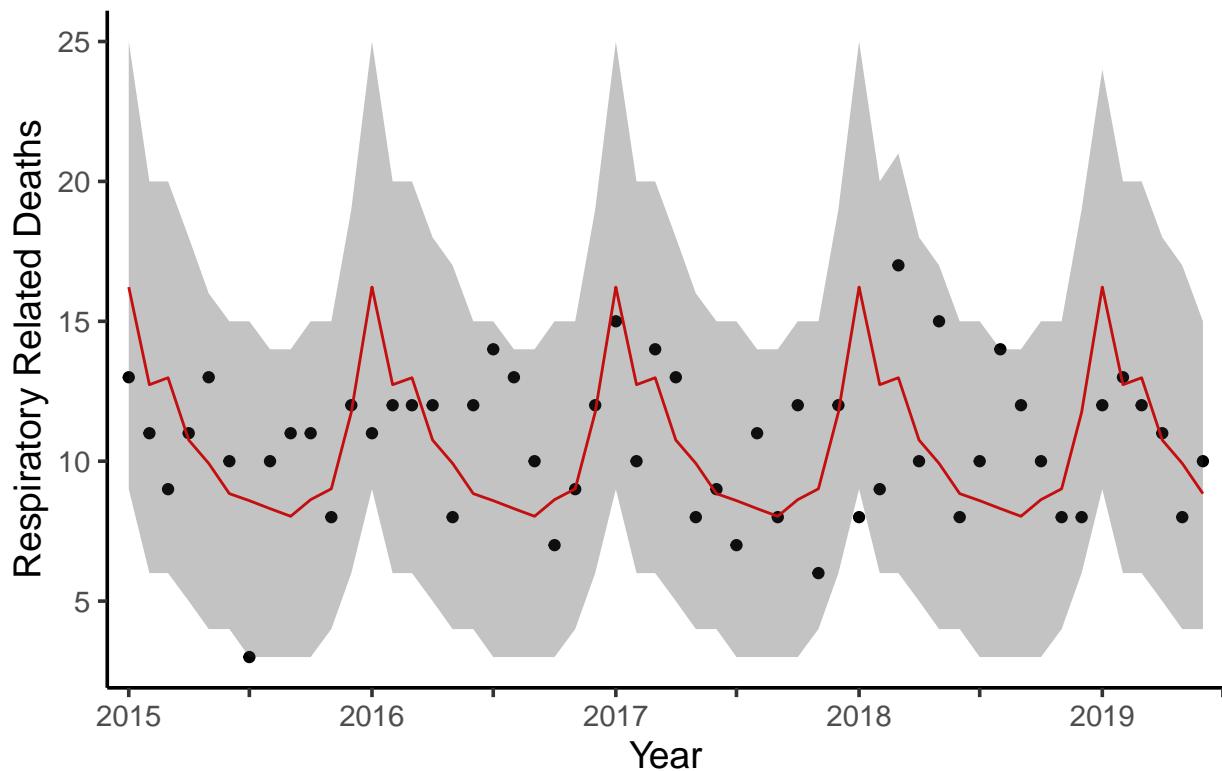
Cluster 4 (Coverage: 88.89%)



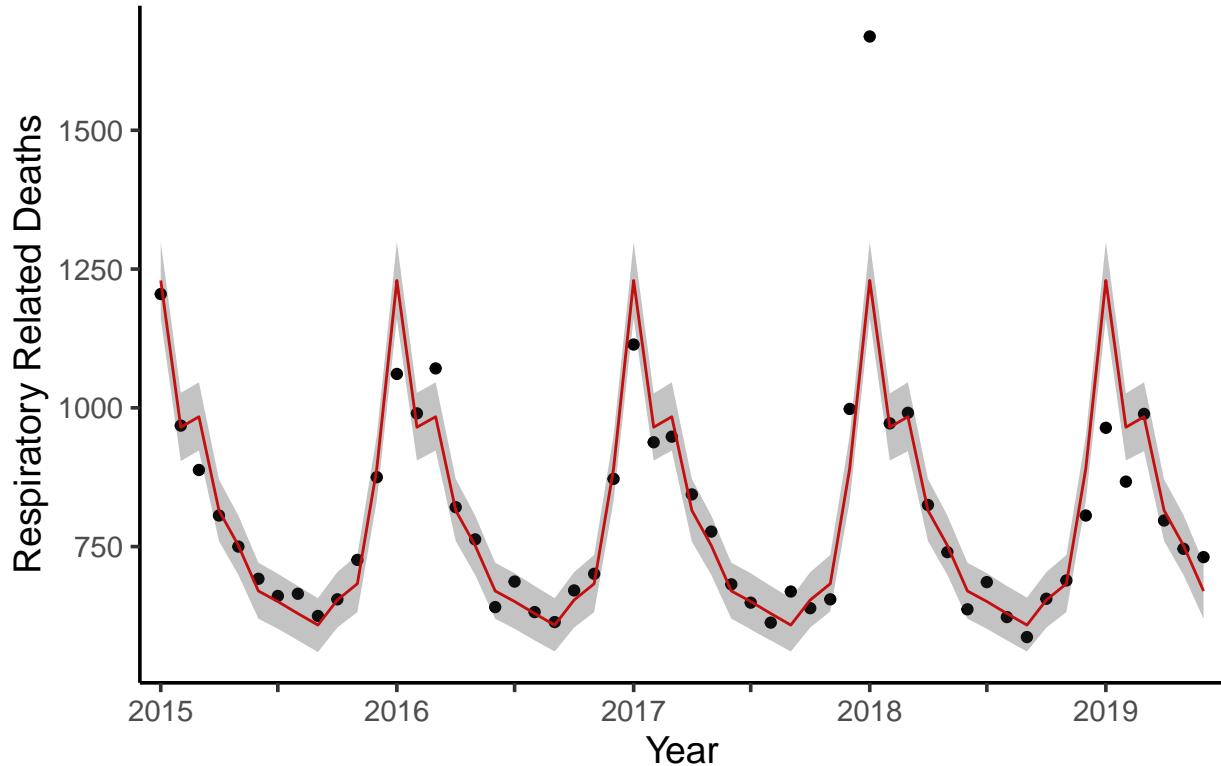
Cluster 5 (Coverage: 96.3%)



Cluster 6 (Coverage: 98.15%)



Cluster 7 (Coverage: 79.63%)



```
pp_insample_plot_combined(pred_data = ref_model1_fit$fitted_values,prefix = "ref1-insample-")
```

Fitting a Besag-York-Mollie model (Reference model 2)

For our second reference model, we decided to fit a Besag-York-Mollie model, which is a log-normal Poisson model with an intrinsic conditional autoregressive component to capture spatial autocorrelations i.e. a Besag model, plus a standard random effects term which is included to capture non-spatial heterogeneity. Obviously, this model is less naive than reference model 1 because it does not assume iid random effects.

The BYM model can be written as,

$$Y_{c,t} \sim Pois(\Lambda_{c,t}) \text{ for } c = 1, \dots, 7 \text{ and } t = 1, \dots, 54 \text{ where } \Lambda_{c,t} | \mathbf{S} = \exp(\beta_{c1}I\{c=1\} + \beta_{c2}I\{c=2\} + \dots + \beta_C I\{c=C\} + \beta_1 I\{t \text{ mod } 12 = 1\} + \dots + \beta_{11} I\{t \text{ mod } 12 = 11\} + \phi_c + \mathbf{F}_c)$$

where $p(\phi) \propto \exp(-\frac{1}{2} \sum_{c_1 \sim c_2} (\phi_{c_1} - \phi_{c_2})^2)$ and $\mathbf{F} | \mathbf{S}, \tau \sim MVN(\mathbf{0}, \tau \Sigma)$.

Note: it is more commonly known that ICAR components are conditionally normally distributed.

As one can see below, the summary outputs indicate that this model is very similar to the Poisson GLMM (reference model 1). The intercept and SD for the random effect component are estimated to almost the exact same as those estimated by the Poisson GLMM, indicating that including the spatial ICAR component is seemingly not very impactful.

```
#Write a function to fit our BYM model in INLA
ref_model2 = function(dataset,a_prec_prior = 1,b_prec_prior = 5e-04,a_phi_prior = 1,b_phi_prior = 5e-04
  #####Fit INLA model
  bym_prior <- list(
    prec.unstruct = list(
      prior = "loggamma",
```

```

    param = c(a_prec_prior,b_prec_prior)),
prec.spatial = list(
  prior = "loggamma",
  param = c(a_phi_prior,b_phi_prior))
)
ref_formula2 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
Intercept5 + Intercept6 + Intercept7 +
f(id, model = "bym", graph = huge.est, hyper = bym_prior) #ID2 in formula results in error
model = inla(formula = ref_formula2,family = "poisson",data = dataset,
            control.compute = list(dic=TRUE,waic=TRUE),
            control.inla = list(strategy = "laplace"),
            control.predictor = list(compute = TRUE, link = link))

####Extract relevant information and store in the list

model_summary <- model$summary.fixed
bri_hyperpar_summary <- bri.hyperpar.summary(model)
modelDIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$ marginals.fixed)
cf <- spread(mf,Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$ marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
ref_model2_results = list(
  model_summary = model_summary,
  bri_hyperpar_summary = bri_hyperpar_summary,
  exp_effects = multeff,
  param_plot = param_plot,
  hyperparam_plot = hyperparam_plot,
  modelDIC = modelDIC,
  model_WAIC = model_WAIC,
  fitted_values = preds_model
)

return(ref_model2_results)
}

```

```

#Fit ref_model2
ref_model2_fit = ref_model2(dataset = inla_insample_data,a_prec_prior = 1,b_prec_prior = 5e-4,
                            a_phi_prior = 1,b_phi_prior = 5e-4)

#Posterior predictive sampling from estimated intensities
ref_model2_fit$fitted_values = poisson_pp_sampling(ref_model2_fit$fitted_values,n=100000)

#Extract DIC and WAIC
ref_model2_DIC = ref_model2_fit$modelDIC
ref_model2_WAIC = ref_model2_fit$modelWAIC

#Get summaries of parameter estimates
ref_model2_fit$model_summary

##                                mean      sd 0.025quant 0.5quant 0.975quant      mode
## months1    2.1039919 7.254718 -12.121790 2.1039919   16.32977 2.1039919
## months2    1.8618693 7.254719 -12.363915 1.8618693   16.08765 1.8618693
## months3    1.8812102 7.254719 -12.344574 1.8812102   16.10699 1.8812102
## months4    1.6930185 7.254720 -12.532768 1.6930185   15.91881 1.6930185
## months5    1.6123975 7.254721 -12.613391 1.6123975   15.83819 1.6123975
## months6    1.4969614 7.254722 -12.728829 1.4969614   15.72275 1.4969614
## months7    1.4681874 7.254725 -12.757608 1.4681874   15.69398 1.4681874
## months8    1.4344522 7.254725 -12.791344 1.4344522   15.66025 1.4344522
## months9    1.4008303 7.254726 -12.824967 1.4008303   15.62663 1.4008303
## months10   1.4724005 7.254725 -12.753395 1.4724005   15.69820 1.4724005
## months11   1.5162689 7.254724 -12.709526 1.5162689   15.74206 1.5162689
## months12   1.7814738 7.254721 -12.444315 1.7814738   16.00726 1.7814738
## Intercept1 4.2895333 7.254879 -9.936564 4.2895333   18.51563 4.2895333
## Intercept2 2.1524452 7.254894 -12.073682 2.1524452   16.37857 2.1524452
## Intercept3 2.0750958 7.254903 -12.151049 2.0750958   16.30124 2.0750958
## Intercept4 3.6524015 7.254893 -10.573724 3.6524015   17.87853 3.6524015
## Intercept5 1.8619519 7.254903 -12.364193 1.8619519   16.08810 1.8619519
## Intercept6 0.6813661 7.254984 -13.544938 0.6813661   14.90767 0.6813661
## Intercept7 5.0102682 7.254871 -9.215814 5.0102682   19.23635 5.0102682
##                                kld
## months1    5.526823e-11
## months2    5.526823e-11
## months3    5.526817e-11
## months4    5.526817e-11
## months5    5.526822e-11
## months6    5.526817e-11
## months7    5.526823e-11
## months8    5.526817e-11
## months9    5.526817e-11
## months10   5.526811e-11
## months11   5.526816e-11
## months12   5.526817e-11
## Intercept1 5.526708e-11
## Intercept2 5.526713e-11
## Intercept3 5.526710e-11
## Intercept4 5.526703e-11
## Intercept5 5.526712e-11
## Intercept6 5.526715e-11

```

```

## Intercept7 5.526717e-11
ref_model2_fit$bri_hyperpar_summary

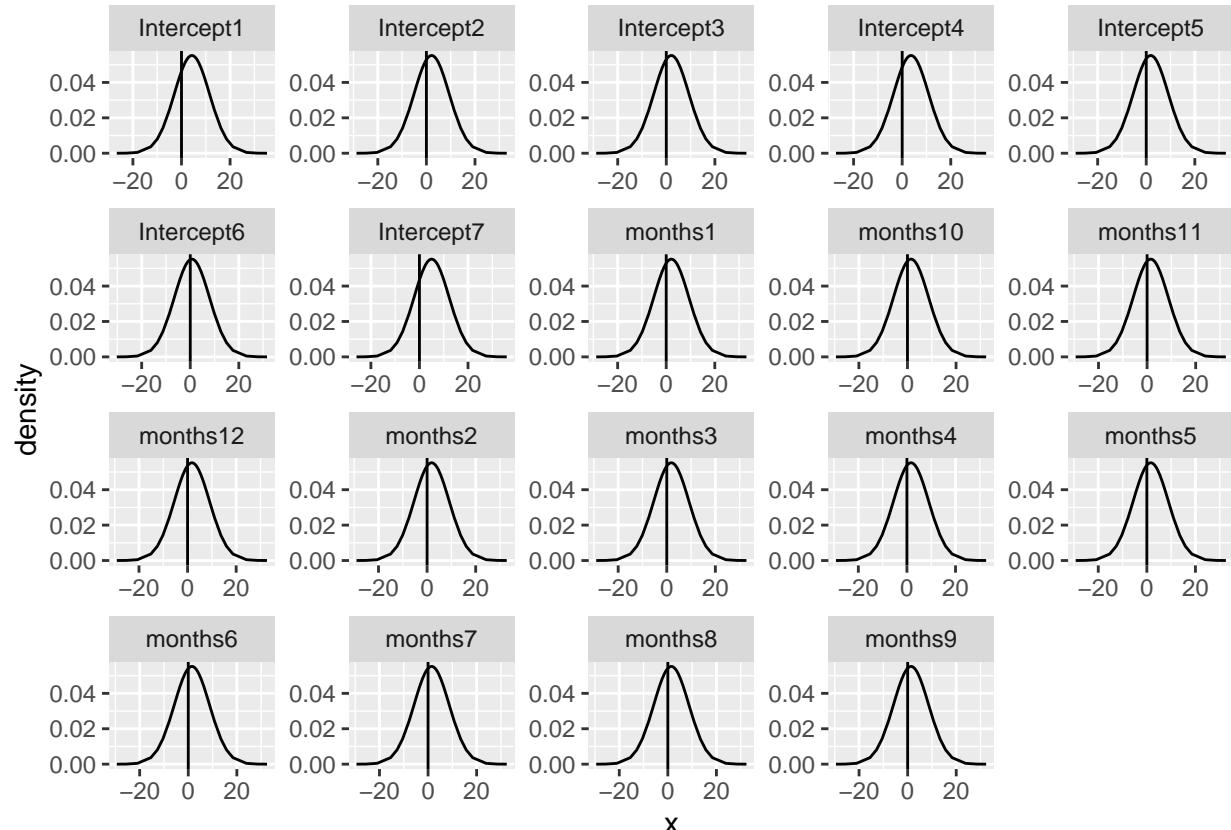
##                               mean        sd      q0.025      q0.5
## SD for id (idd component) 0.03117834 0.01874727 0.01079952 0.02597501
## SD for id (spatial component) 0.03117845 0.01874742 0.01079963 0.02597504
##                               q0.975      mode
## SD for id (idd component) 0.08143821 0.01914941
## SD for id (spatial component) 0.08143880 0.01914945
ref_model2_fit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
## 8.198833  6.435756  6.561441  5.435864  5.014820  4.468091  4.341359
##    months8    months9   months10   months11   months12 Intercept1 Intercept2
## 4.197345  4.058568  4.359688  4.555198  5.938602 72.932423  8.605876
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
## 7.965309 38.567175  6.436287  1.976576 149.944945

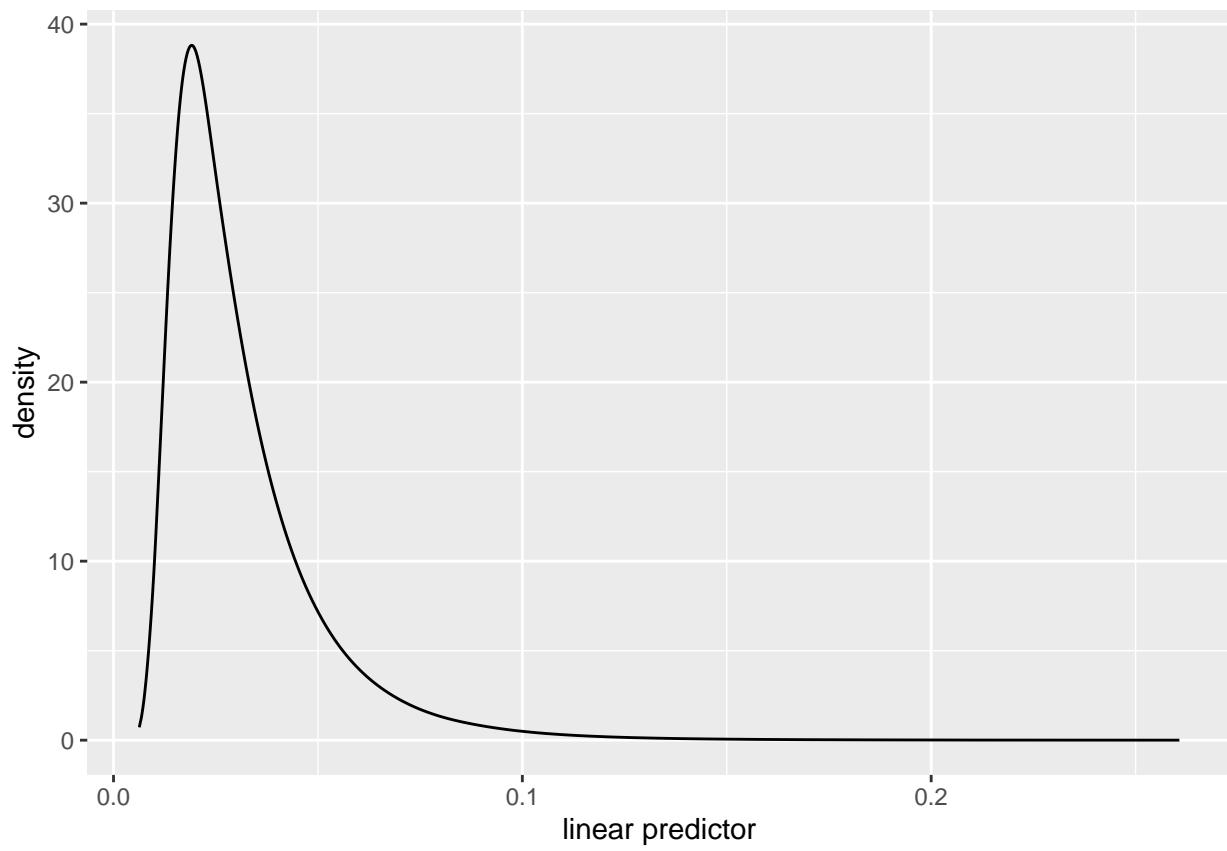
```

#Show plots

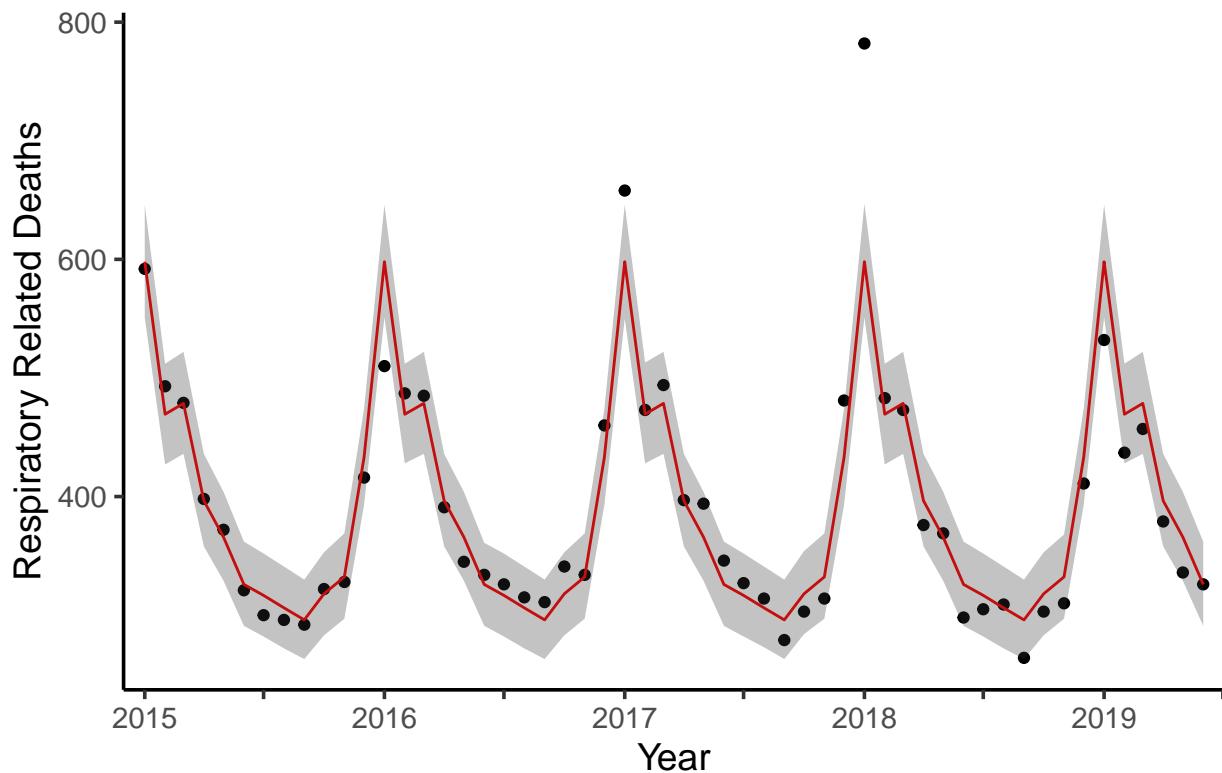
```
ref_model2_fit$param_plot
```



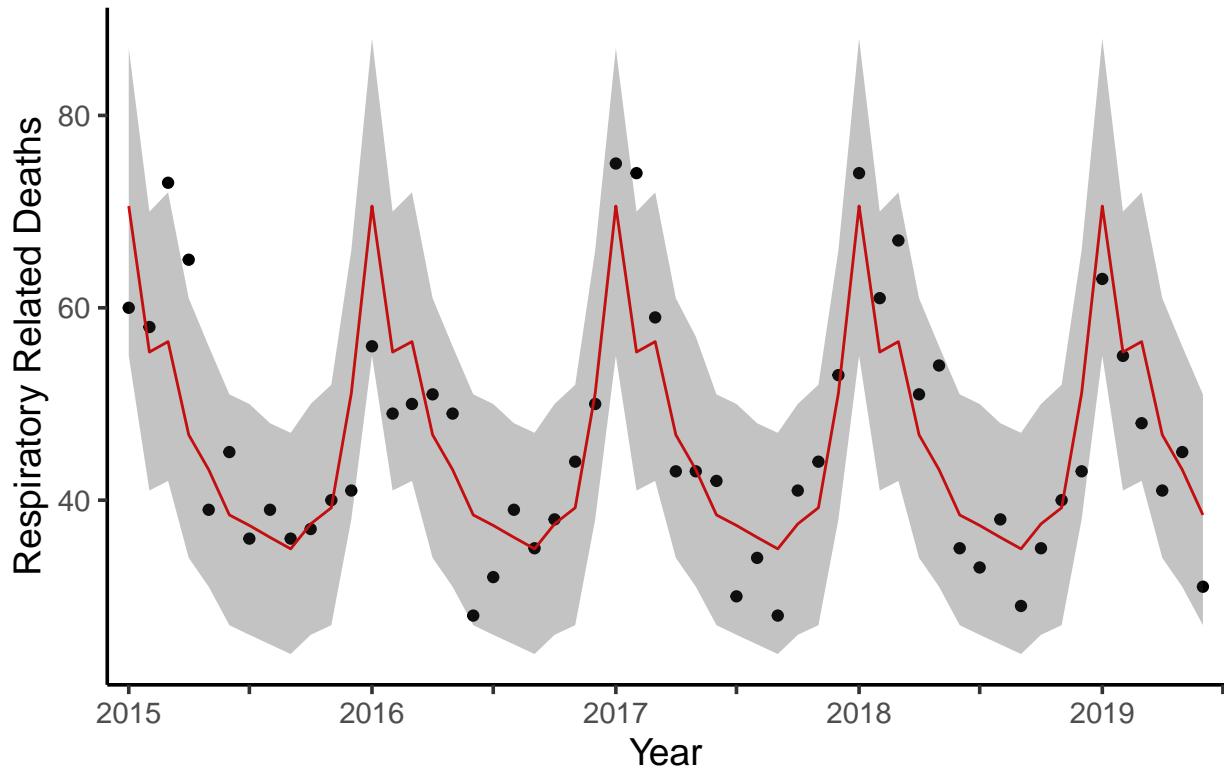
```
ref_model2_fit$hyperparam_plot
```



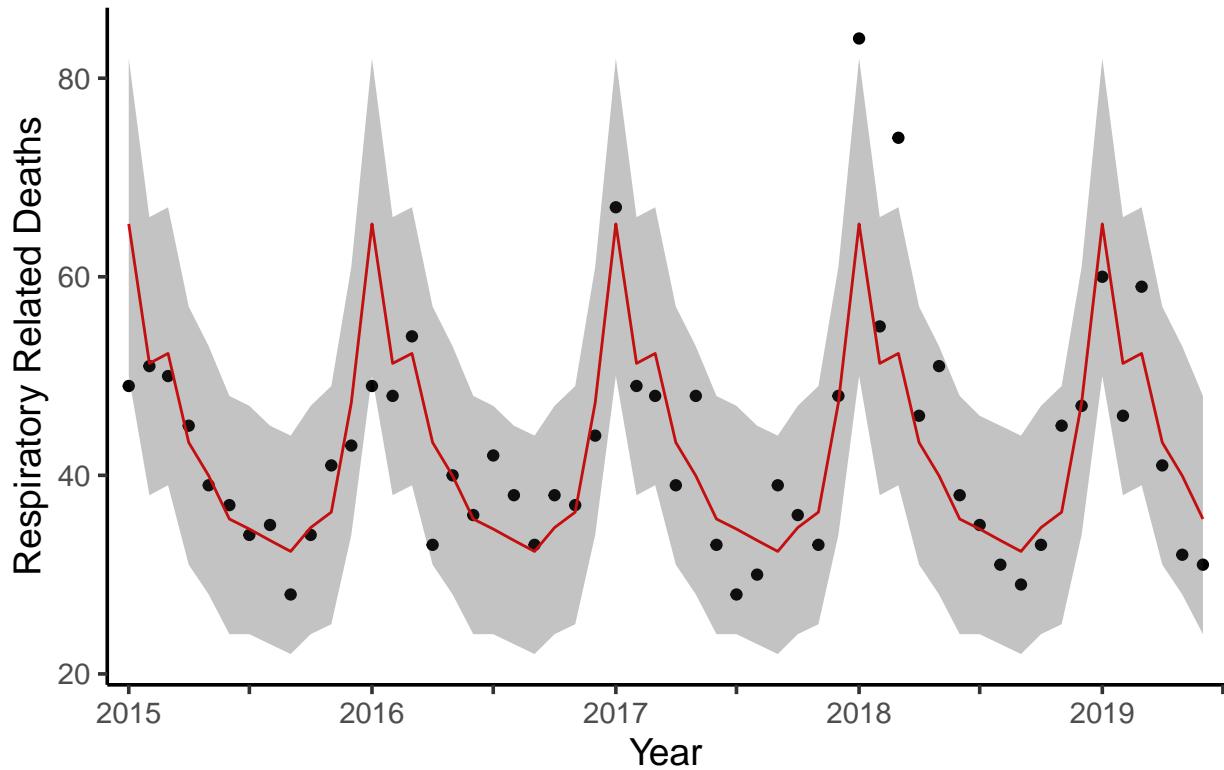
Cluster 1 (Coverage: 90.74%)



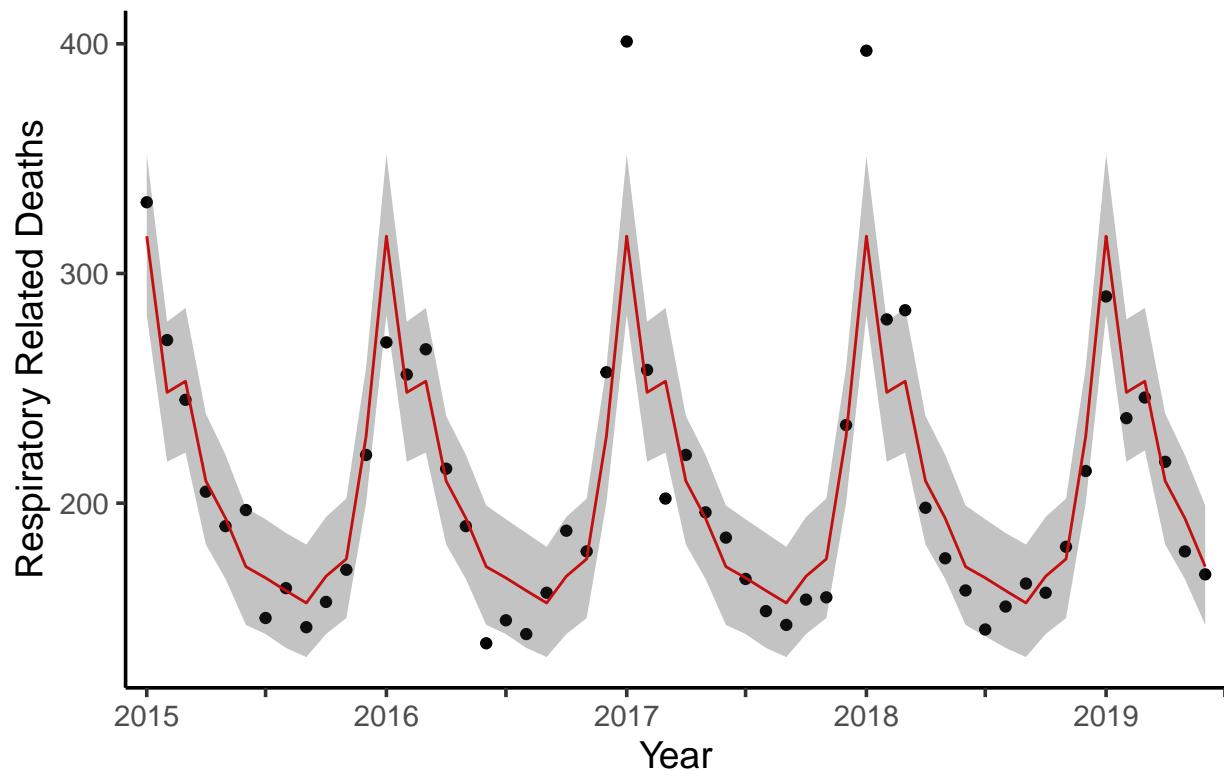
Cluster 2 (Coverage: 94.44%)



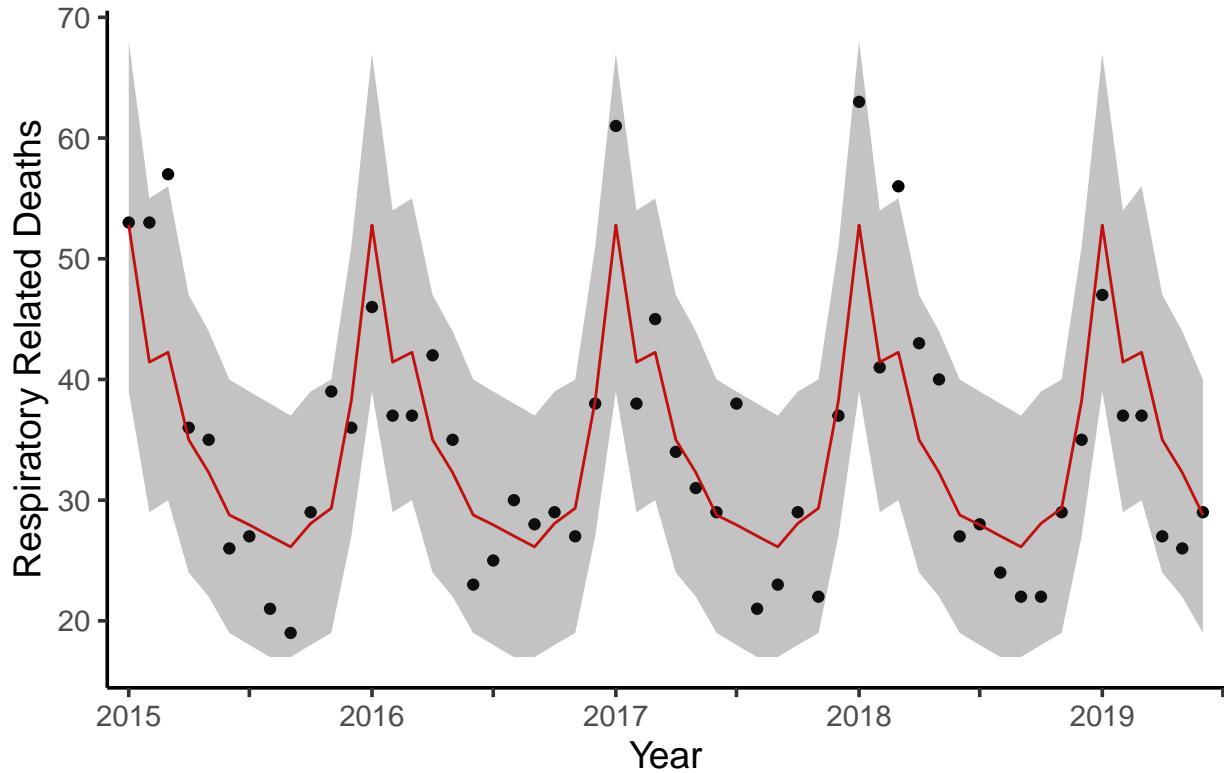
Cluster 3 (Coverage: 92.59%)



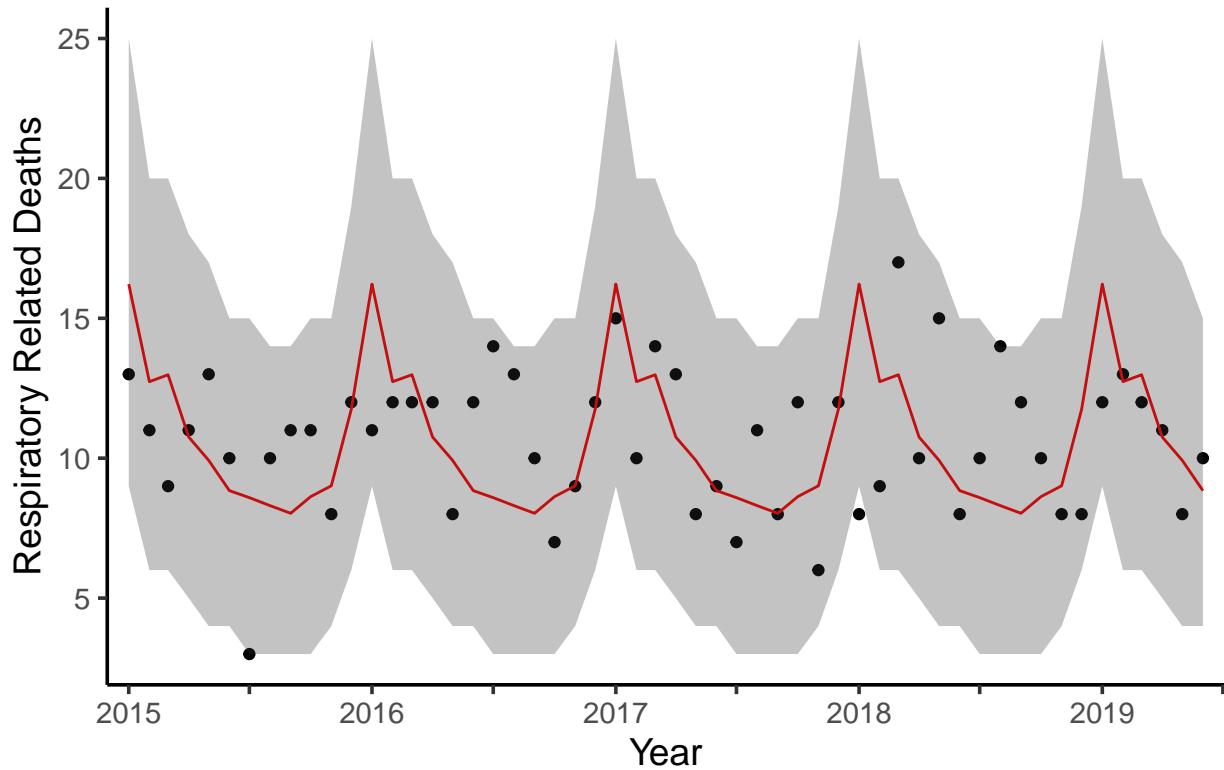
Cluster 4 (Coverage: 88.89%)



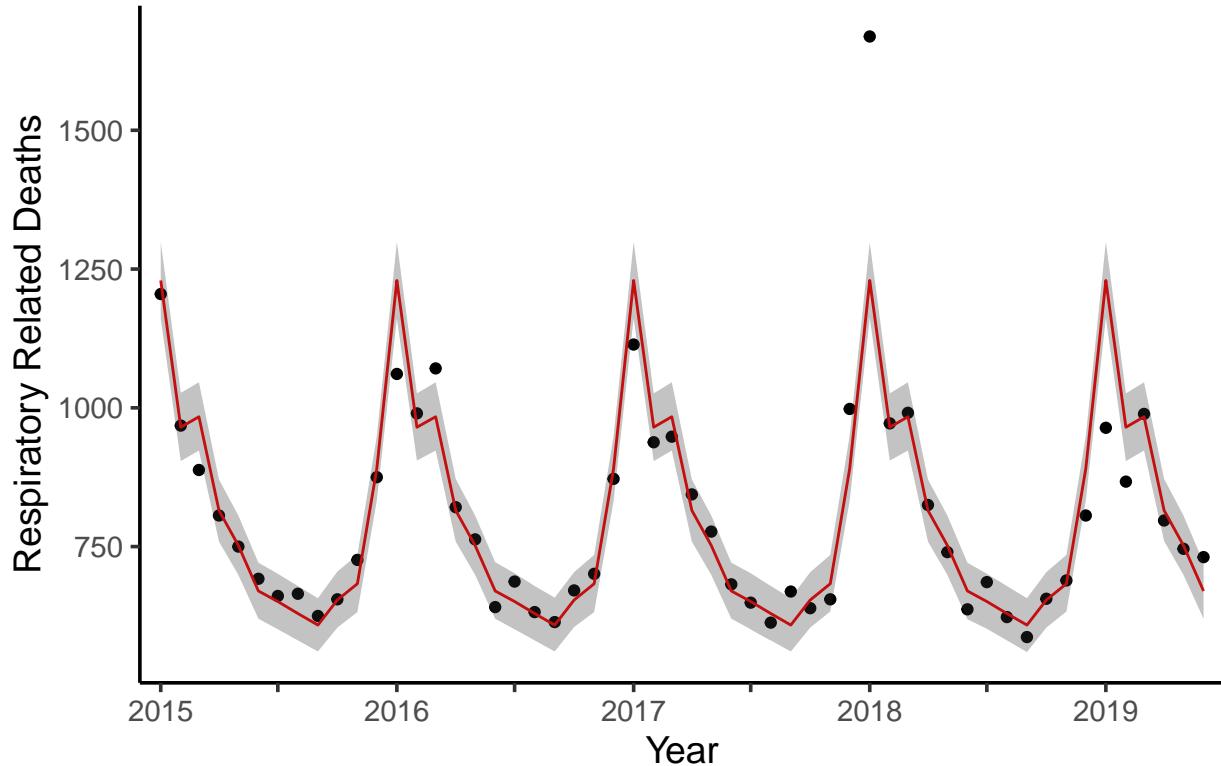
Cluster 5 (Coverage: 96.3%)



Cluster 6 (Coverage: 98.15%)



Cluster 7 (Coverage: 79.63%)



```
pp_insample_plot_combined(pred_data = ref_model2_fit$fitted_values,prefix = "ref2-insample-")
```

Fitting kernel graph regression models

KGR model with time series kernel x graph filter (Proposed model 2)

Finally, we fit our proposed model which we call a kernel graph regression model. It also takes the form of a latent Gaussian model as shown below:

$$\Lambda_{c,t} | \mathbf{F}, \mathbf{S} = \exp(\beta_{c1} I\{c = 1\} + \beta_{c2} I\{c = 2\} + \dots + \beta_7 I\{c = 7\} + \beta_1 I\{t \bmod 12 = 1\} + \dots + \beta_{11} I\{t \bmod 12 = 11\} + \mathbf{F}_{c,t})$$

where the graph signal $\mathbf{F} | \mathbf{S}, \rho_{rbf}, \rho_p, \sigma_{EPA}^2 \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_{EPA} \otimes \mathbf{H}^2)$ with $Cov(F_{c_1, t_1}, F_{c_2, t_2}) = [\mathbf{K}^{EPA}]_{t_1, t_2} [\mathbf{H}^2]_{c_1, c_2}$.

The key difference here is that the covariance matrix of this GP is specified by the kronecker product of K , which is the time kernel gram matrix calculated from the EPA air quality data, and H , which is the graph filter which is calculated from the adjacency matrix estimated by glasso using the HUGE package. This matrix has been estimated already so it can be directly plugged into INLA as the covariance matrix of our underlying GP using the “generic0” specification as shown below:

Calculating gram matrix K from EPA data

Using the EPA air quality data, we can calculate the gram matrix K which will characterize the dependence structure of air quality (across 7 different pollutants and AQI) over time. This is done by calculating the squared difference between all of the observations at two different time points e.g. 56 observations for Jan 2014 compared with the 64 observations for Feb 2014. For the in sample analysis, the resulting matrix is 54x54 because we are holding out the last 6 months of observations.

```

EPA_kernel = function(EPA_data = final_EPA_agg_data,time_span,rho_rbf,rho_periodic,sigma2){
  K_EPA = matrix(0,nrow=time_span,ncol=time_span)
  i = 1
  j = 1

  for(t1 in 1:time_span){
    for (t2 in 1:time_span){
      A = EPA_data %>% filter(Time == t1)
      B = EPA_data %>% filter(Time == t2)
      AQIa = unique(A$AQI)
      AQIb = unique(B$AQI)

      ABtest = c((A$Value-B$Value)^2,(AQIa-AQIb)^2) #7 clusters * 8 measurements
      # K_EPA[i,j] = exp(-sum(ABtest)) / (2*rho_rbf)) * sigma2

      # K_EPA[i,j] = exp(- (sum(ABtest))) ###square this sum or remove it???
      #           / (2*rho_rbf)) * exp(- (2*sin(sum(abs(ABtest))*pi/12)^2)
      #           / (rho_periodic)) * sigma2

      K_EPA[i,j] = exp(- (mean(ABtest)) ###mean or sum???
                         / (2*rho_rbf)) * exp(- (2*sin(sum(abs(ABtest))*pi/12)^2)
                         / (rho_periodic)) * sigma2

      j = j+1
    }
    j = 1
    i = i+1
  }

  return(K_EPA)
}

```

Ensuring precision matrix is not computationally singular, so we jitter eigenvalues, using reciprocal condition number as constraint

```

desingularize = function(covmatrix,threshold = 1e-2,increment = 0.01){

  tracker = 0

  while (rcond(covmatrix) <= threshold){
    #Perform spectral decomposition
    ev = eigen(covmatrix)
    L = ev$values
    V = ev$vectors

    # #Add a little noise to eigenvalues to bring away from 0
    L = L + increment

    # #Add 0.01 to eigenvalues in bottom five percentile to bring away from 0
    # cutoff = quantile(abs(L),0.05)
    # L[which(abs(L) < cutoff)] = L[which(abs(L) < cutoff)] + 0.01
  }
}

```

```

#Calculate new precision matrix
covmatrix = V %*% diag(L) %*% t(V)

    tracker = tracker + 1
}

results_list = list(covmatrix,tracker)
sprintf("%s iterations of desingularizer applied",tracker)
return(results_list)
}

# test = desingularize(K_time)

GLMM with type 0 generic specification (known covariance matrix)
kgr_model2 = function(dataset, rho_EPA_rbf = 1, rho_EPA_periodic = 1, sigma2_EPA = 1, link=1){

#Calculate gram matrix K_EPA
K_EPA = EPA_kernel(time_span = length(unique(dataset$time)),
rho_rbf = rho_EPA_rbf, rho_periodic = rho_EPA_periodic, sigma2 = sigma2_EPA)

#Heatmap of resulting K
K_EPA_heatmap = matrix_heatmap(K_EPA,title = "")

#Calculate trace norm of gram matrix
K_EPA_weight = norm((1/60)*K_EPA,type = "F")

###Load graph regression kernel
covGP = kronecker(K_EPA/60,(H^2)/7)

#Need to ensure precision matrix is not computationally singular i.e det > 0
covGP_jittered = desingularize(covGP,threshold = 1e-2,increment = 0.01)
covGP = covGP_jittered[[1]]
inv_covGP = solve(covGP)
# cov_Fnorm = norm(covGP,type = "F")

#Heatmap of resulting K
inv_covGP_heatmap = matrix_heatmap2(inv_covGP,title = "",legend_title = "Precision",prefix = "kgr2-pr")

###Fit INLA model
# kgr_formula2 = response ~ -1 + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
#   Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP)

kgr_formula2 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
  Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP)

model = inla(formula = kgr_formula2,family = "poisson",data = dataset,
control.compute = list(dic=TRUE,waic=TRUE,
return.marginals.predictor=TRUE),
control.inla = list(strategy = "laplace"),
control.predictor = list(compute = TRUE, link = link))

```

```

####Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hyperpar_summary <- bri.hyperpar.summary(model)
model_DIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf, Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
kgr_model2_results = list(
  K_EPA_heatmap = K_EPA_heatmap,
  K_EPA_weight = K_EPA_weight/(K_EPA_weight + gfilter_weight),
  gfilter_weight = gfilter_weight/(K_EPA_weight + gfilter_weight),
  covmatrix = covGP,
  prec = inv_covGP,
  num_jitters = covGP_jittered[[2]],
  prec_heatmap = inv_covGP_heatmap,
  model_summary = model_summary,
  bri_hyperpar_summary = bri_hyperpar_summary,
  exp_effects = multeff,
  param_plot = param_plot,
  hyperparam_plot = hyperparam_plot,
  model_DIC = model_DIC,
  model_WAIC = model_WAIC,
  fitted_values = preds_model,
  marg_fitted_values = marginal_fvs
)
return(kgr_model2_results)
}

#Fit kgr_model2
kgr_model2_fit = kgr_model2(dataset = inla_insample_data,rho_EPA_rbf = 23.009,rho_EPA_periodic = 5253.009)

```

```

#Posterior predictive sampling from estimated intensities
kgr_model2_fit$fitted_values = poisson_pp_sampling(kgr_model2_fit$fitted_values, n=100000)

#Extract DIC and WAIC
kgr_model2_DIC = kgr_model2_fit$modelDIC
kgr_model2_WAIC = kgr_model2_fit$modelWAIC

#Get summaries of parameter estimates
kgr_model2_fit$model_summary

##          mean        sd 0.025quant 0.5quant 0.975quant      mode
## months1    2.098868 7.254730 -12.126937 2.098868   16.32467 2.098868
## months2    1.869505 7.254731 -12.356304 1.869505   16.09531 1.869505
## months3    1.891071 7.254731 -12.334738 1.891071   16.11688 1.891071
## months4    1.696463 7.254733 -12.529349 1.696463   15.92228 1.696463
## months5    1.616083 7.254734 -12.609731 1.616083   15.84190 1.616083
## months6    1.491741 7.254735 -12.734076 1.491741   15.71756 1.491741
## months7    1.455218 7.254742 -12.770612 1.455218   15.68105 1.455218
## months8    1.434208 7.254742 -12.791623 1.434208   15.66004 1.434208
## months9    1.392895 7.254743 -12.832937 1.392895   15.61873 1.392895
## months10   1.473388 7.254742 -12.752441 1.473388   15.69922 1.473388
## months11   1.516417 7.254741 -12.709411 1.516417   15.74224 1.516417
## months12   1.776299 7.254738 -12.449523 1.776299   16.00212 1.776299
## Intercept1  4.285925 7.254753 -9.939926 4.285925   18.51178 4.285925
## Intercept2  2.151779 7.254758 -12.074082 2.151779   16.37764 2.151779
## Intercept3  2.074403 7.254779 -12.151500 2.074403   16.30031 2.074403
## Intercept4  3.647509 7.254767 -10.578368 3.647509   17.87339 3.647509
## Intercept5  1.860457 7.254759 -12.365405 1.860457   16.08632 1.860457
## Intercept6  0.681925 7.254874 -13.544163 0.681925   14.90801 0.681925
## Intercept7  5.010156 7.254727 -9.215644 5.010156   19.23596 5.010156
##          kld
## months1    5.526822e-11
## months2    5.526812e-11
## months3    5.526823e-11
## months4    5.526823e-11
## months5    5.526817e-11
## months6    5.526817e-11
## months7    5.526823e-11
## months8    5.526811e-11
## months9    5.526823e-11
## months10   5.526817e-11
## months11   5.526817e-11
## months12   5.526823e-11
## Intercept1 5.526817e-11
## Intercept2 5.526822e-11
## Intercept3 5.526818e-11
## Intercept4 5.526823e-11
## Intercept5 5.526823e-11
## Intercept6 5.526817e-11
## Intercept7 5.526823e-11

kgr_model2_fit$bri_hyperpar_summary

##          mean        sd     q0.025     q0.5     q0.975      mode

```

```

## SD for id2 0.6519292 0.05113658 0.5562614 0.6501584 0.7571085 0.6467981
kgr_model2_fit$exp_effects

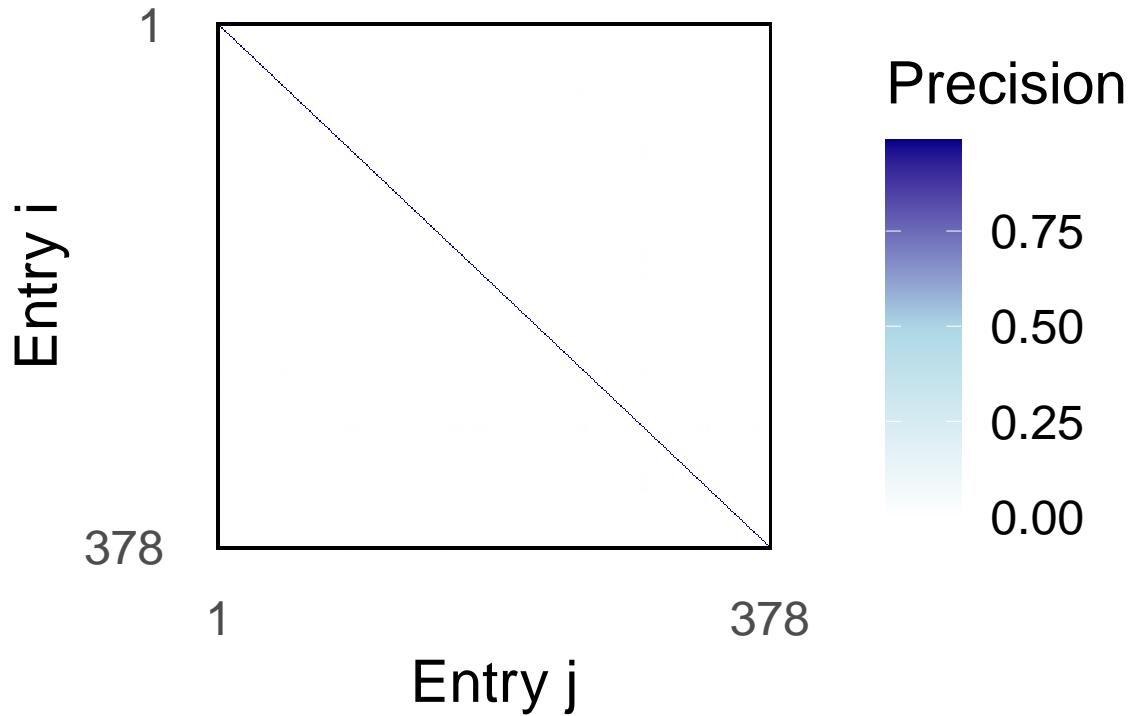
##    months1    months2    months3    months4    months5    months6    months7
## 8.156934 6.485084 6.626461 5.454621 5.033334 4.444827 4.285416
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
## 4.196319 4.026490 4.363994 4.555871 5.907949 72.669748 8.600143
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
## 7.959790 38.378962 6.426676 1.977681 149.928084
kgr_model2_fit$K_EPA_weight

## [1] 0.7751928
kgr_model2_fit$gfilter_weight

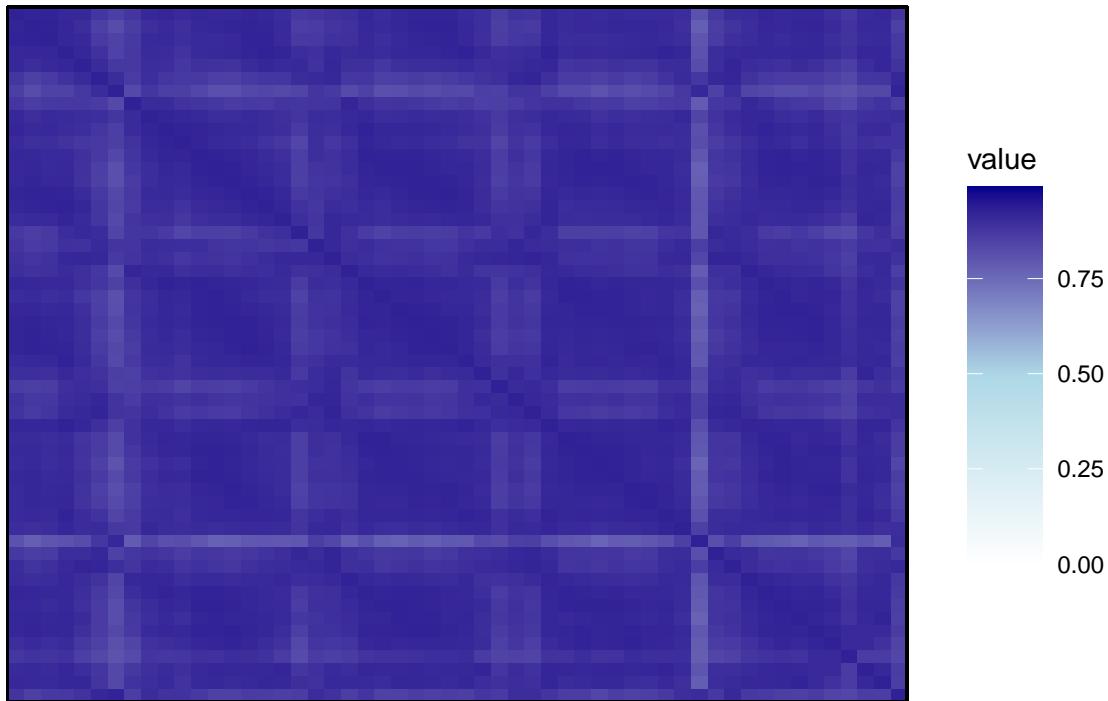
## [1] 0.22248072
kgr_model2_fit$num_jitters

## [1] 1
#Show plots
kgr_model2_fit$prec_heatmap

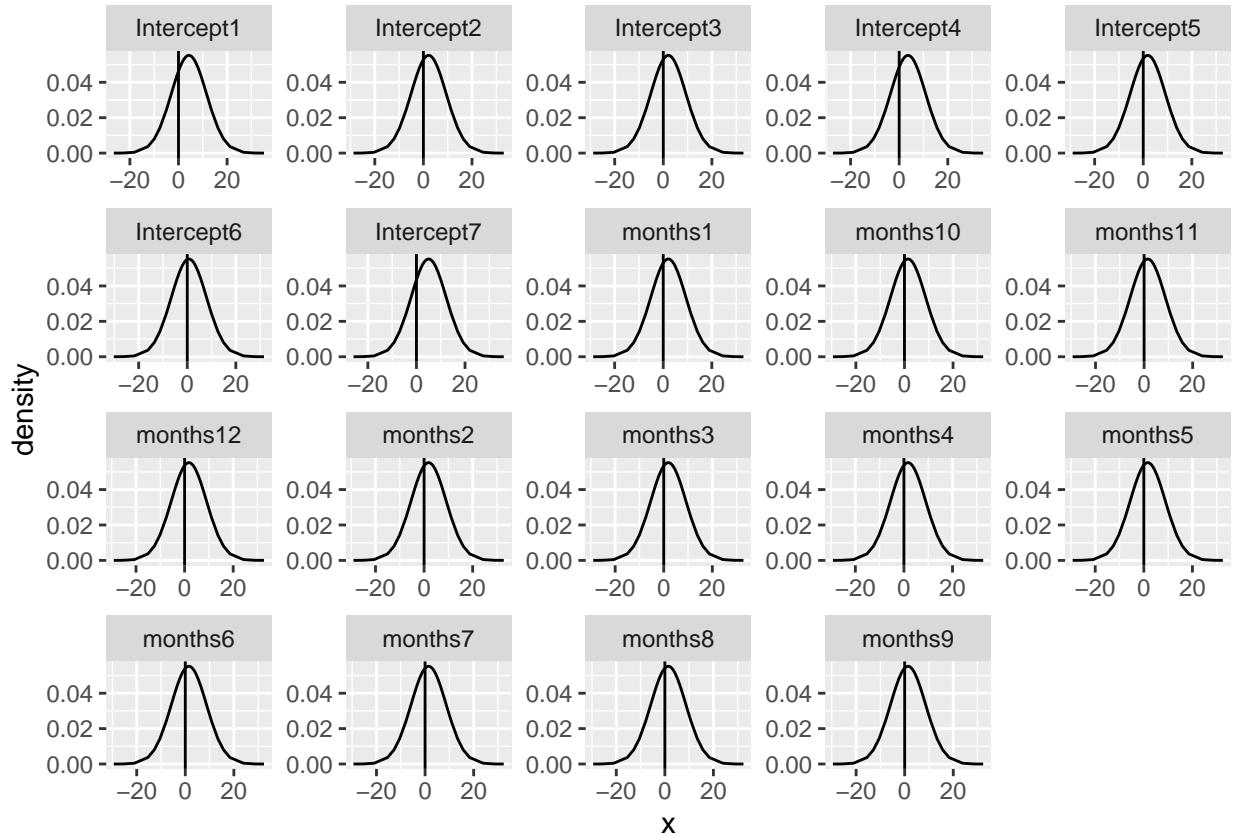
```



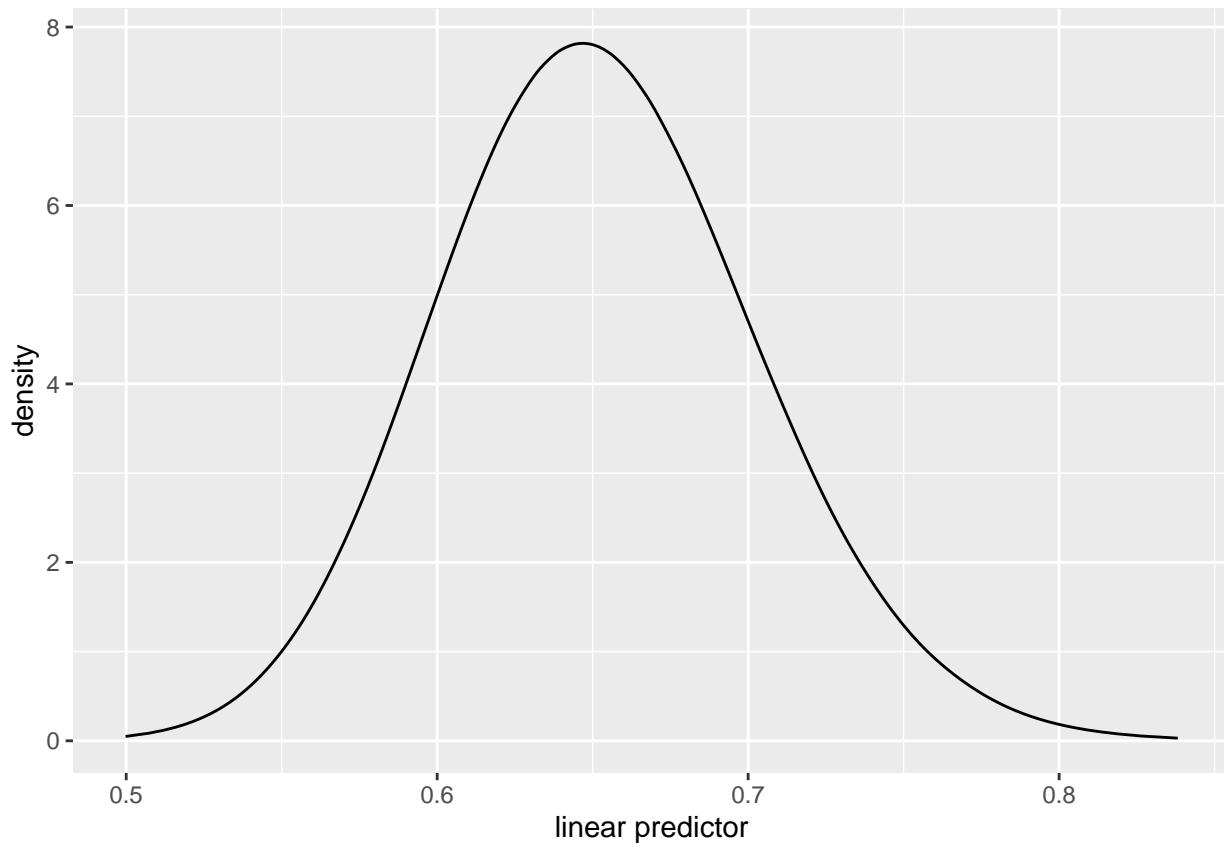
```
kgr_model2_fit$K_EPA_heatmap
```



```
kgr_model2_fit$param_plot
```

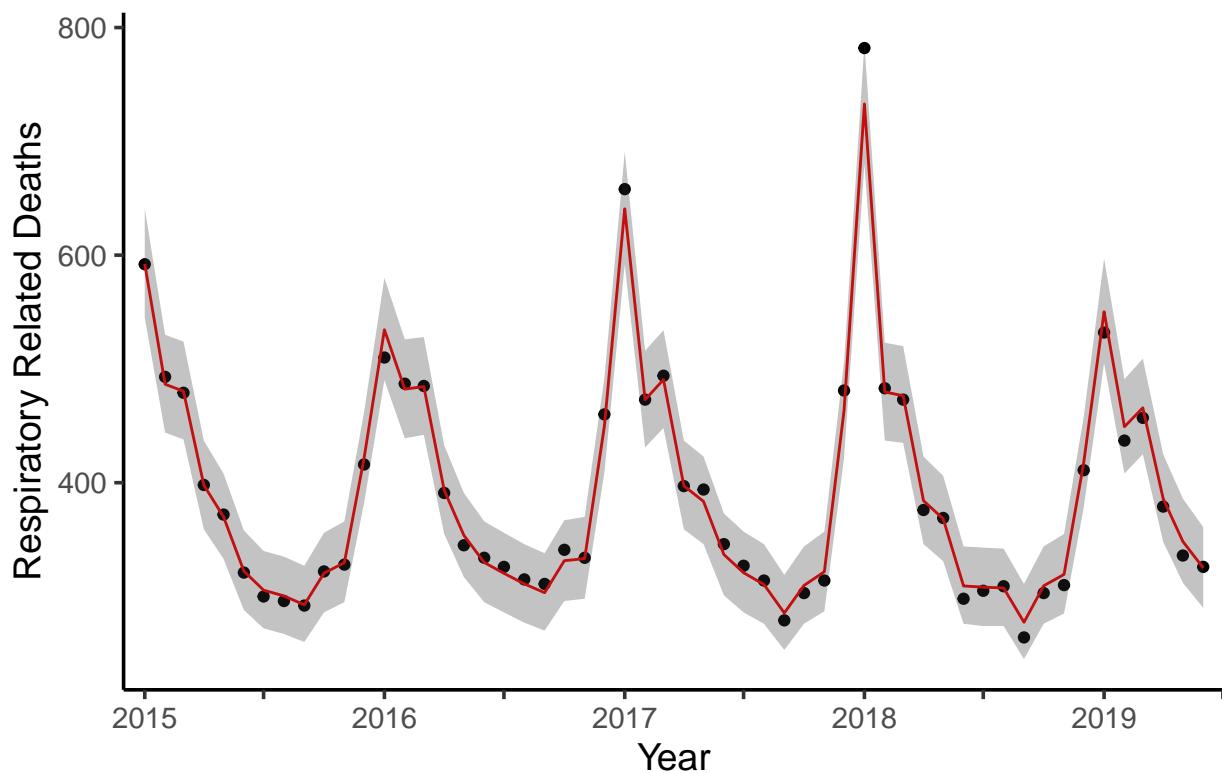


```
kgr_model12_fit$hyperparam_plot
```

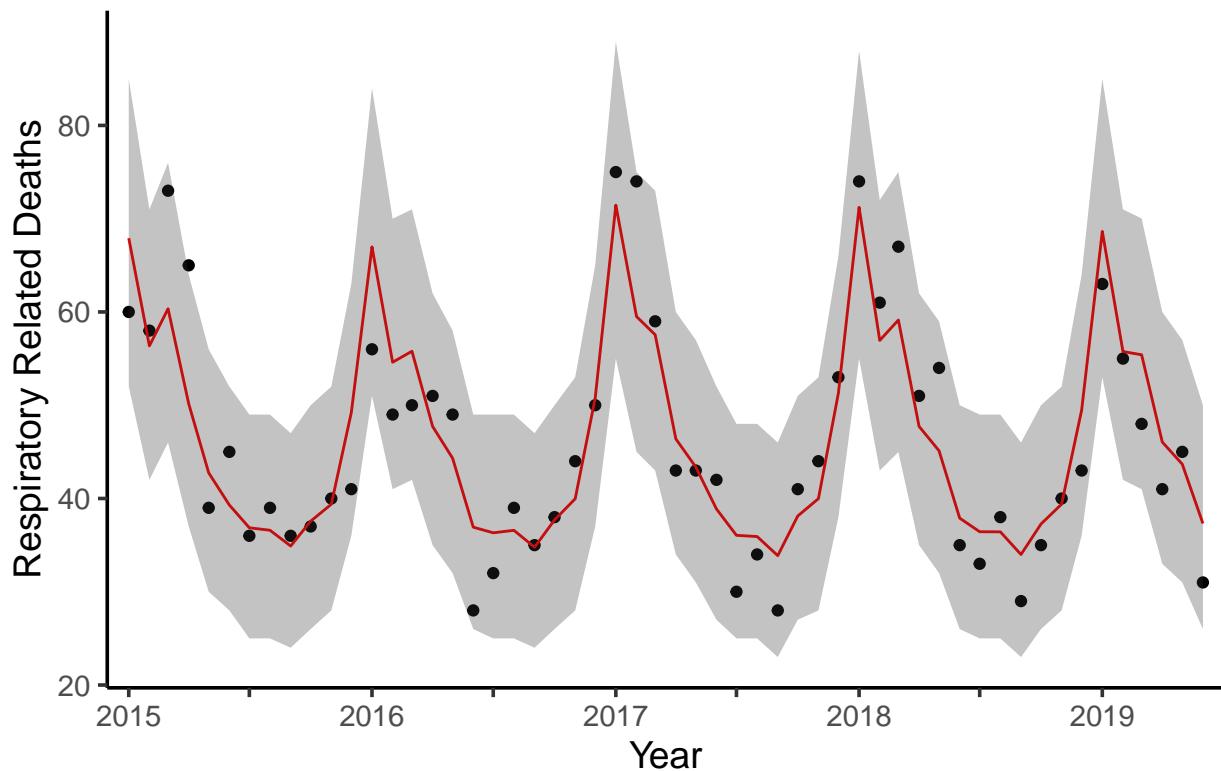


```
pp_insample_plot(pred_data = kgr_model2_fit$fitted_values,prefix = "kgr2-insample-")
```

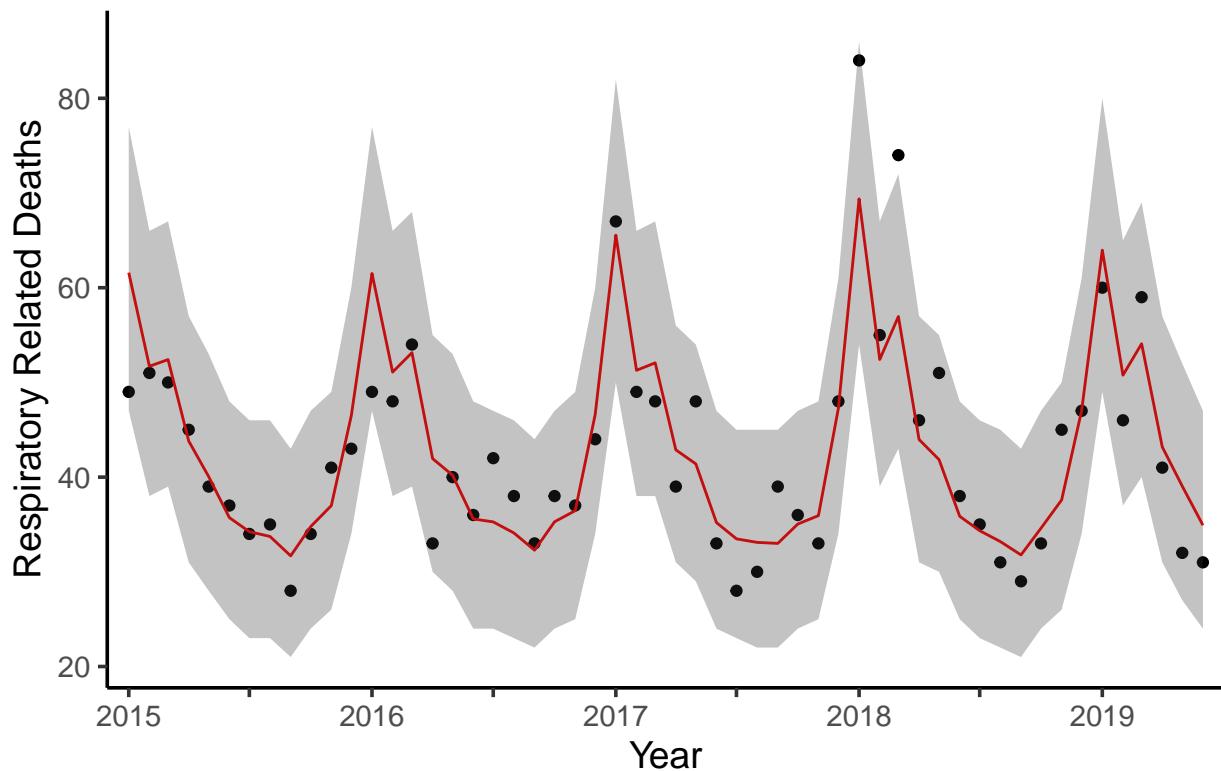
Cluster 1 (Coverage: 100%)



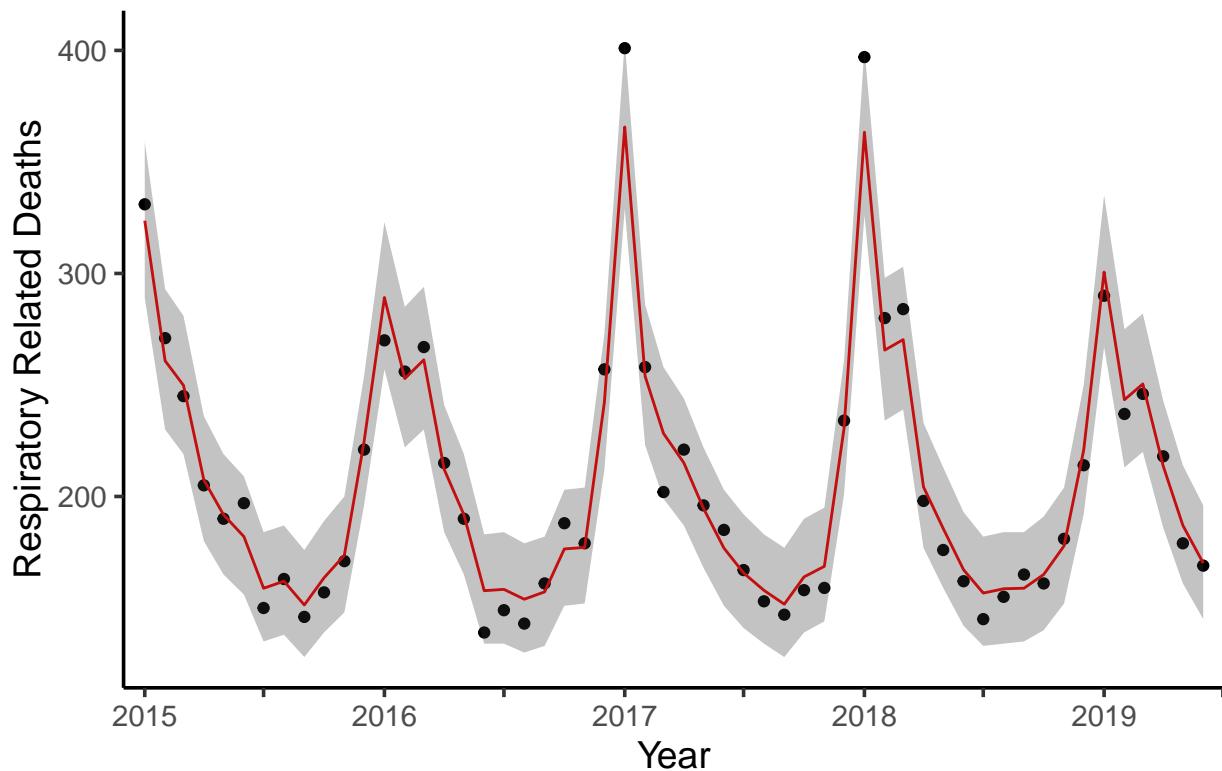
Cluster 2 (Coverage: 98.15%)



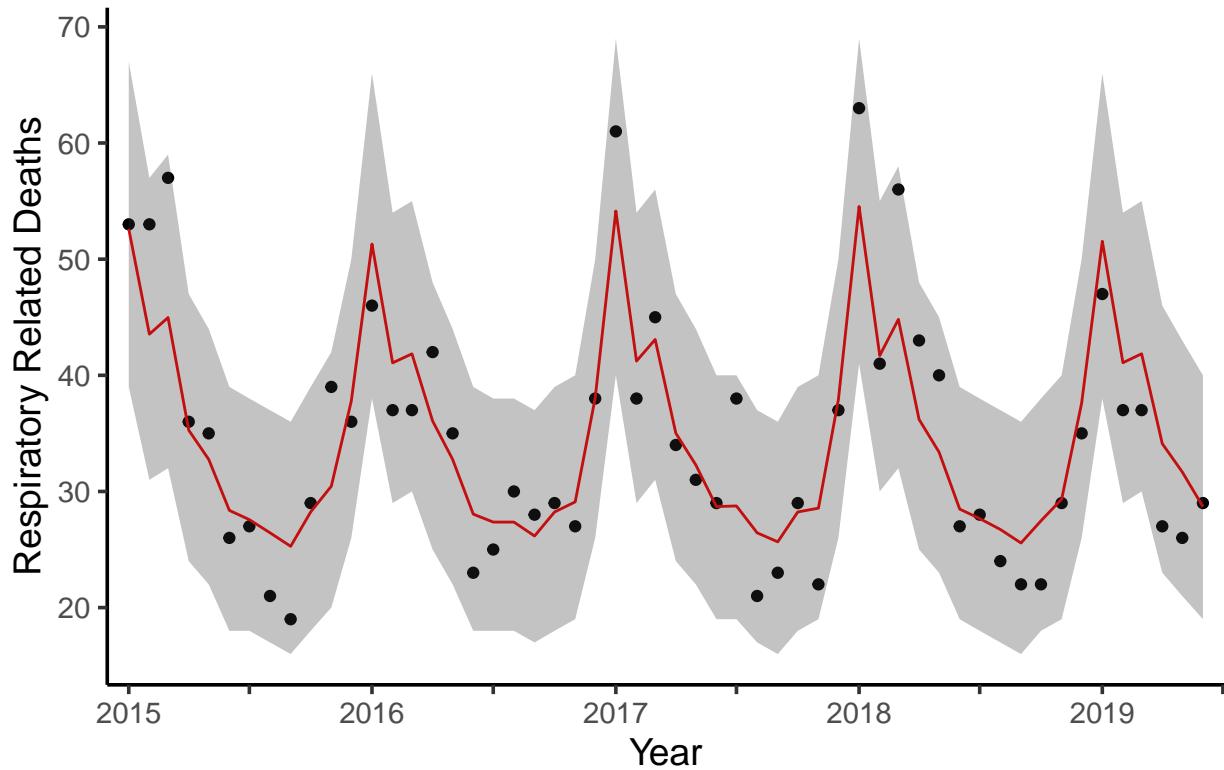
Cluster 3 (Coverage: 98.15%)



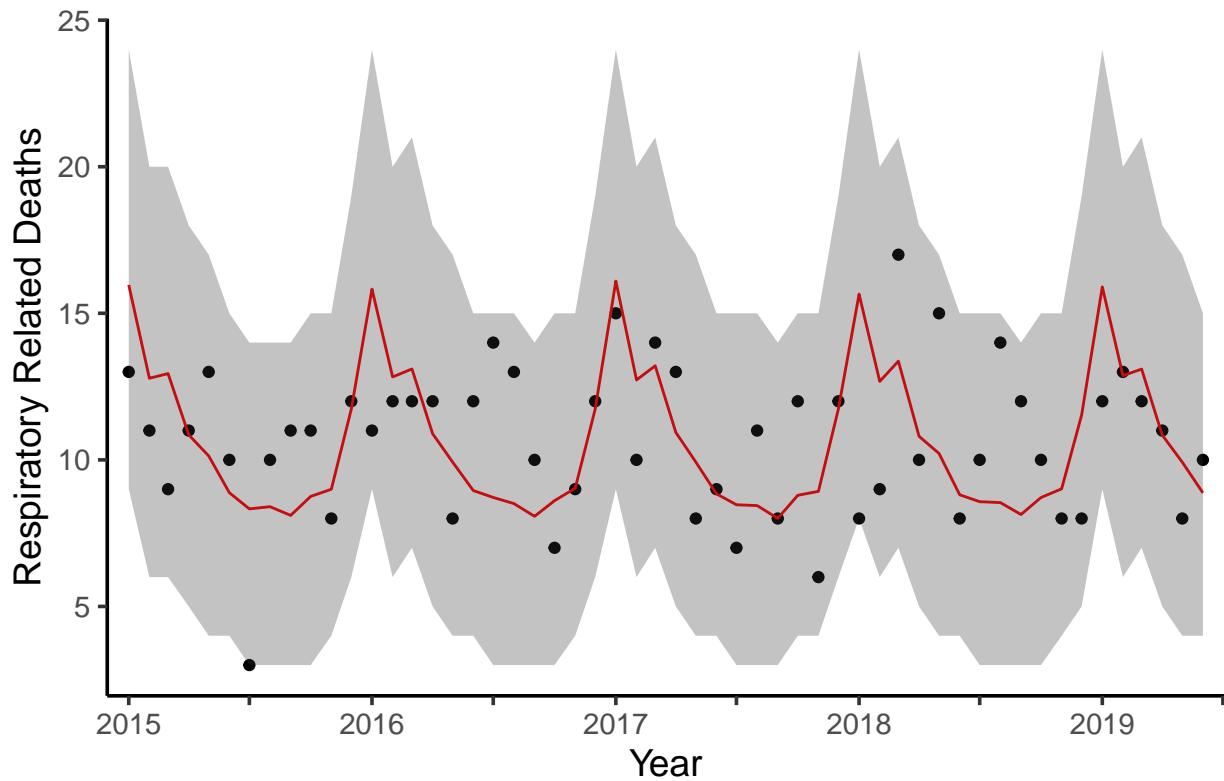
Cluster 4 (Coverage: 100%)



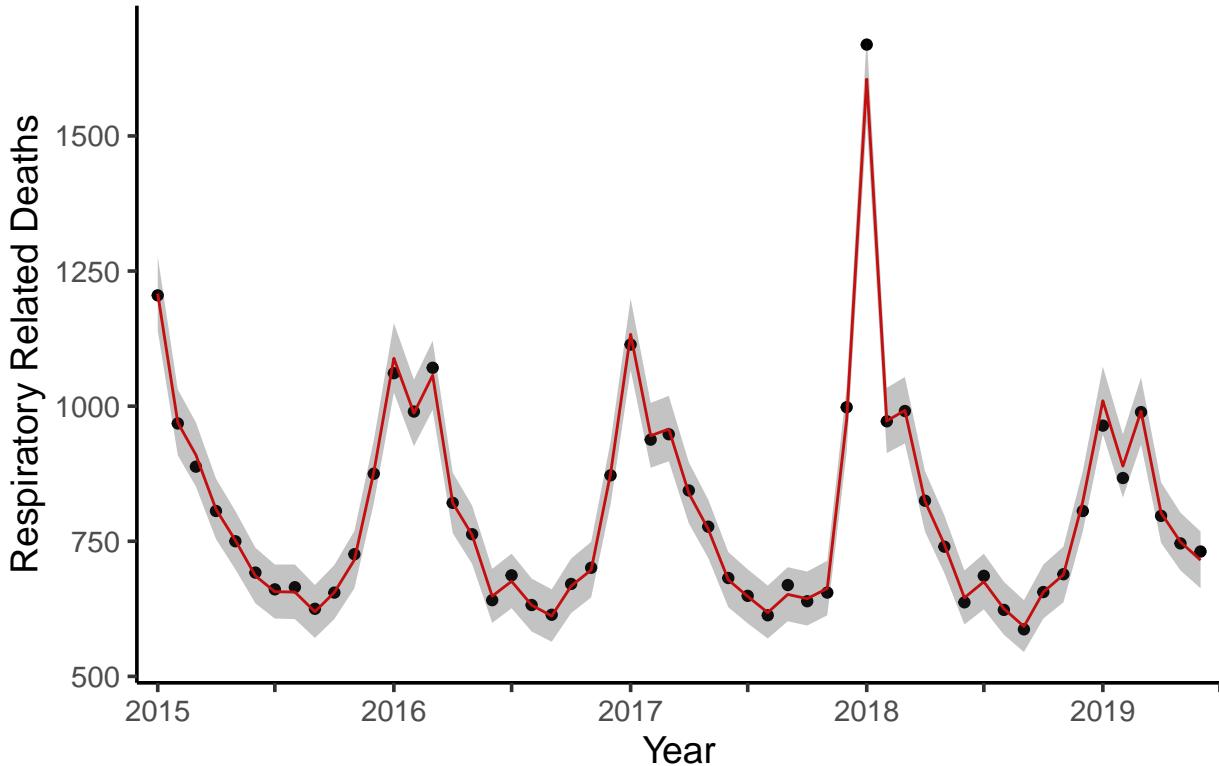
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 100%)



Cluster 7 (Coverage: 100%)



```
pp_insample_plot_combined(pred_data = kgr_model2_fit$fitted_values,prefix = "kgr2-insample-")
```

Negative Binomial model

```
kgr_model2_nb = function(dataset, rho_EPA_rbf = 1, rho_EPA_periodic = 1, sigma2_EPA = 1, link=1){

  #Calculate gram matrix K_EPA
  K_EPA = EPA_kernel(time_span = length(unique(dataset$time)),
                      rho_rbf = rho_EPA_rbf, rho_periodic = rho_EPA_periodic, sigma2 = sigma2_EPA)

  #Heatmap of resulting K
  K_EPA_heatmap = matrix_heatmap(K_EPA,title = "K_EPA Heatmap")

  #Calculate trace norm of gram matrix
  K_EPA_weight = norm((1/60)*K_EPA,type = "F")

  ###Load graph regression kernel
  covGP = kronecker(K_EPA/60,(H^2)/7)

  #Need to ensure precision matrix is not computationally singular i.e det > 0
  covGP_jittered = desingularize(covGP,threshold = 1e-2,increment = 0.01)
  covGP = covGP_jittered[[1]]
  inv_covGP = solve(covGP)
  # cov_Fnorm = norm(covGP,type = "F")
}
```

```

#Heatmap of resulting K
inv_covGP_heatmap = matrix_heatmap2(inv_covGP,title = "",legend_title = "Precision",prefix = "kgr2-nb"

####Fit INLA model
kgr_formula2 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP)

# kgr_formula2 = response ~ -1 + offset(log(pop)) + months + f(id2,model = "generic0",Cmatrix = inv_c

model = inla(formula = kgr_formula2,family = "nbinomial",data = dataset,
control.compute = list(dic=TRUE,waic=TRUE,
return.marginals.predictor=TRUE),
control.family = list(hyper = list(theta = list(prior = "pc.mgamma", param = 1))),
control.inla = list(strategy = "laplace"),
control.predictor = list(compute = TRUE, link = link))

####Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hypersummary <- bri.hypersummary(model)
model_DIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

####NBINOMIAL SAMPLING
pp_ysamples = list()
n_density = data.frame(model$marginals.hyperpar[[1]])
sample_n = inla.rmarginal(1000,inla.smarginal(n_density))
pp_nsamples = sample_n

####Generate uncertainty bands from the posterior predictive distribution as opposed to posterior of t
for (i in 1:nrow(preds_model)){
  ####Sample 1000 mu for each row
  row = data.frame(model$marginals.fitted.values[[i]])

  sample_mu = inla.rmarginal(1000,inla.smarginal(row))
  lower = quantile(sample_mu,0.025)
  mean = mean(sample_mu)
  upper = quantile(sample_mu,0.975)

  ####Sample from nbinom with these a and b
  sample_y = rnbinom(1000, n = sample_n, mu = sample_mu)
  pp_ysamples[[i]] = sample_y

  ####Obtain 95% credible interval
  preds_model$y_ppmean[i] = mean(sample_y, na.rm = TRUE)
  preds_model$y_pplower[i] = quantile(sample_y, 0.025, na.rm = TRUE)
  preds_model$y_ppupper[i] = quantile(sample_y, 0.975, na.rm = TRUE)
}

#Exponentiating parameter to get better interpretation of estimates

```

```

multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$ marginals.fixed)
cf <- spread(mf, Var2, value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf, aes(x=x, y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$ marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden, aes(x,y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
kgr_model2_results = list(
  K_EPA_heatmap = K_EPA_heatmap,
  K_EPA_weight = K_EPA_weight/(K_EPA_weight + gfilter_weight),
  gfilter_weight = gfilter_weight/(K_EPA_weight + gfilter_weight),
  covmatrix = covGP,
  prec = inv_covGP,
  num_jitters = covGP_jittered[[2]],
  prec_heatmap = inv_covGP_heatmap,
  model_summary = model_summary,
  bri_hyperpar_summary = bri_hyperpar_summary,
  exp_effects = multeff,
  param_plot = param_plot,
  hyperparam_plot = hyperparam_plot,
  modelDIC = modelDIC,
  modelWAIC = modelWAIC,
  fitted_values = preds_model,
  marg_fitted_values = marginal_fvs
)

return(kgr_model2_results)
}

#Fit kgr_model2
kgr_model2_nb_fit = kgr_model2_nb(dataset = inla_insample_data, rho_EPA_rbf = 23.009, rho_EPA_periodic = 1)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
## a single row.

#Posterior predictive sampling from estimated intensities
# kgr_model2_nb_fit$fitted_values = nb_pp_sampling(kgr_model2_nb_fit$fitted_values, n=100000)

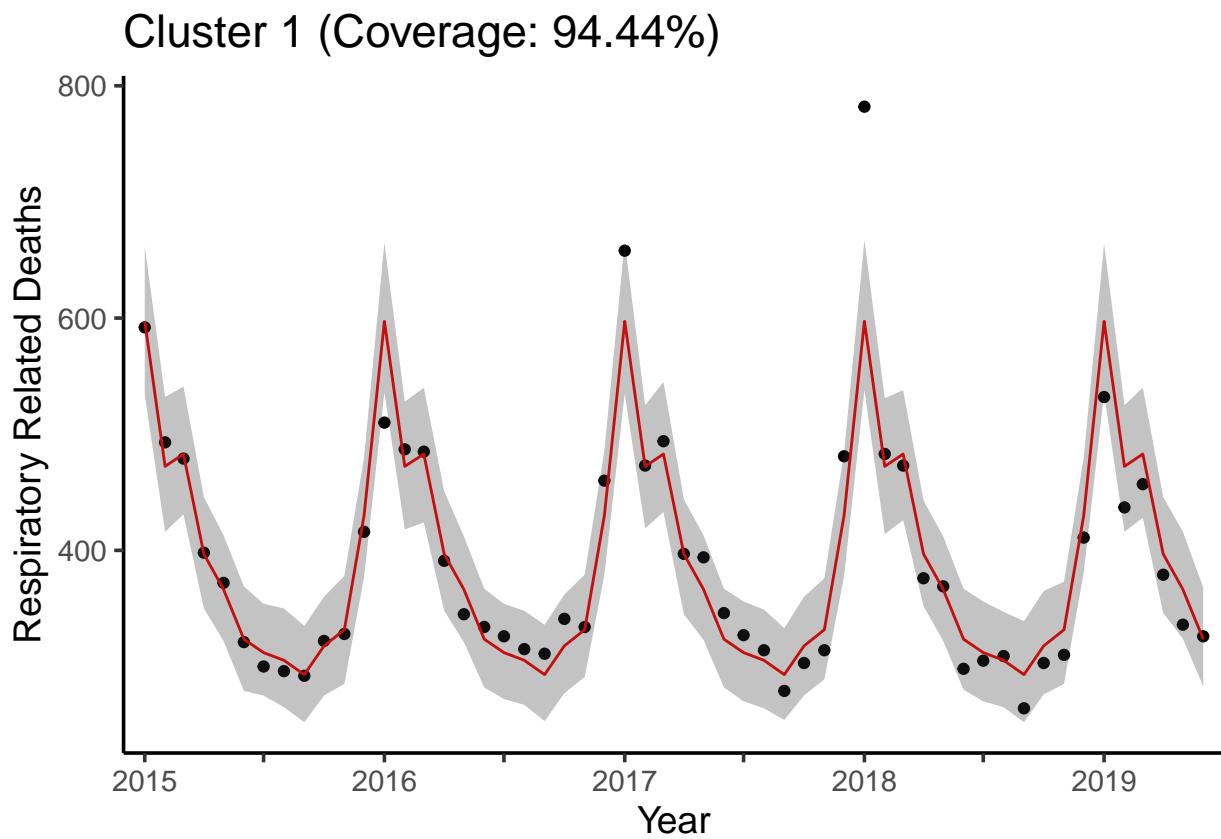
# #Extract DIC and WAIC
# kgr_model2_DIC = kgr_model2_nb_fit$modelDIC
# kgr_model2_WAIC = kgr_model2_nb_fit$modelWAIC
#
# #Get summaries of parameter estimates
# kgr_model2_fit$model_summary

```

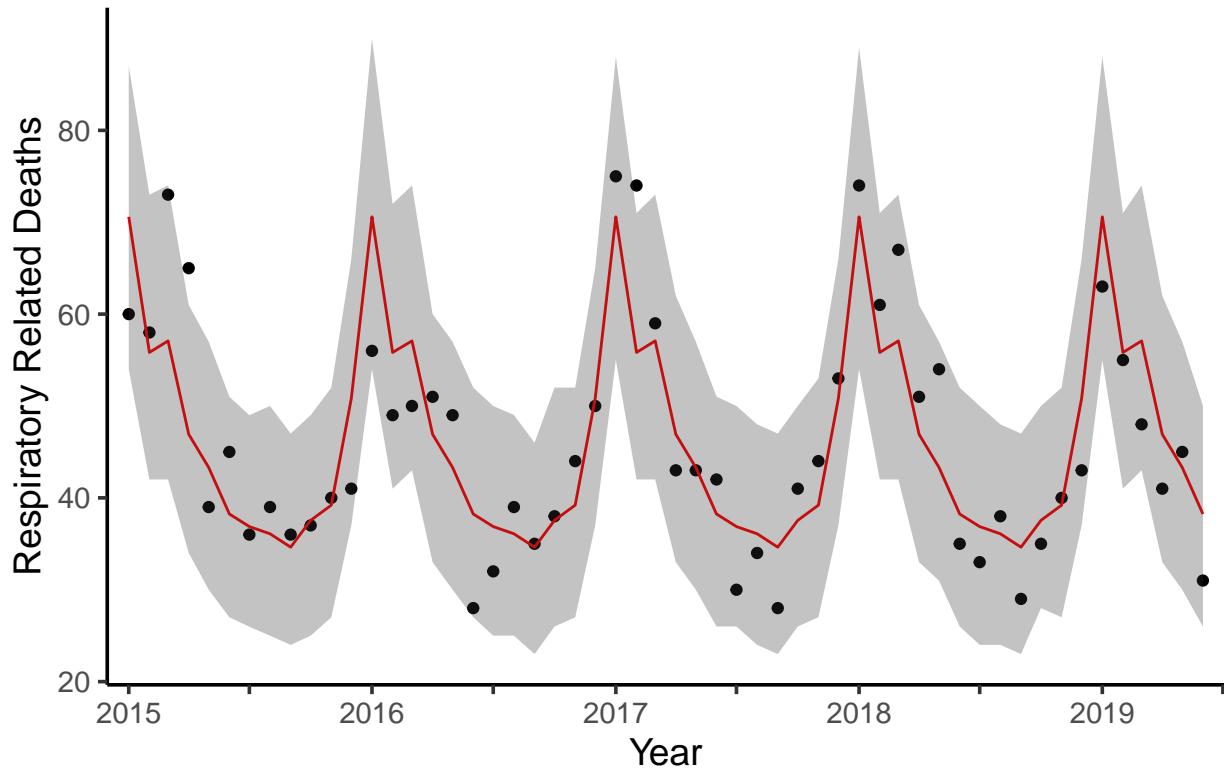
```

# kgr_model2_fit$bri_hyperpar_summary
# kgr_model2_fit$exp_effects
# kgr_model2_fit$K_EPA_weight
# kgr_model2_fit$gfilter_weight
# kgr_model2_fit$num_jitters
#
# #Show plots
# kgr_model2_fit$prec_heatmap
# kgr_model2_fit$K_EPA_heatmap
# kgr_model2_fit$param_plot
# kgr_model2_fit$hyperparam_plot
pp_insample_plot(pred_data = kgr_model2_nb_fit$fitted_values,prefix = "kgr2-nb-insample-")

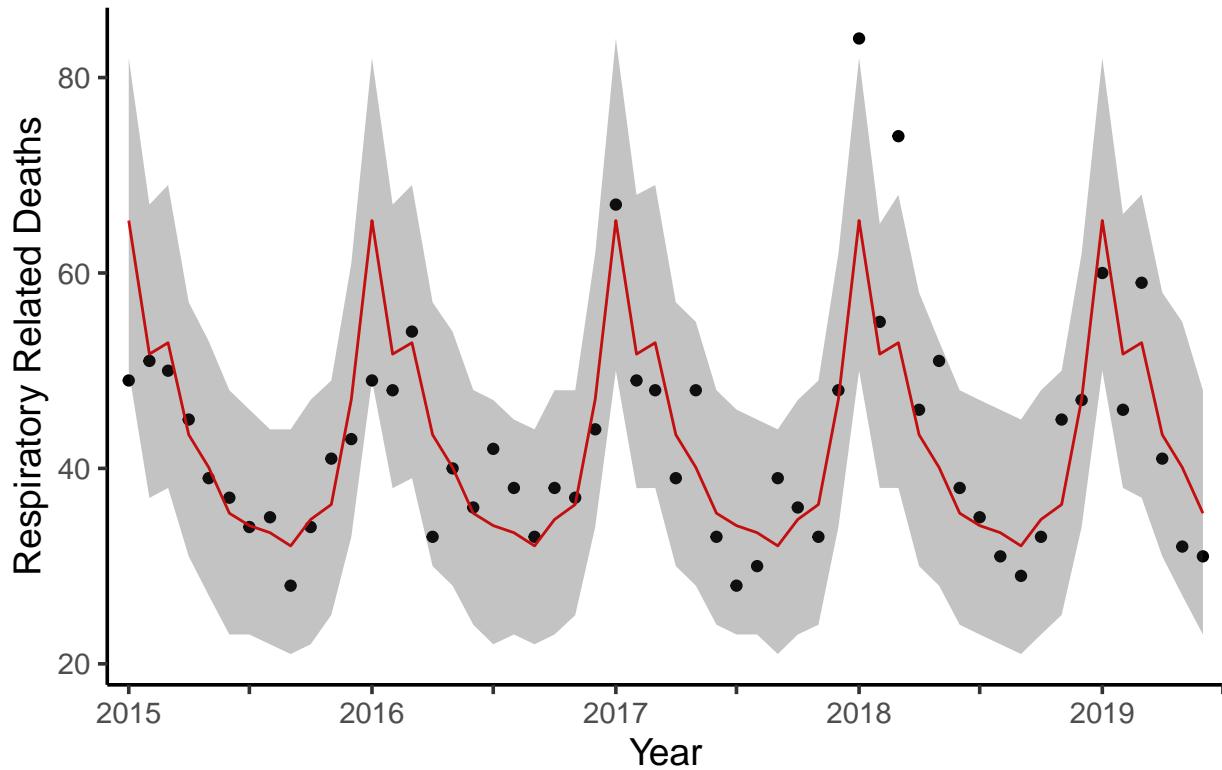
```



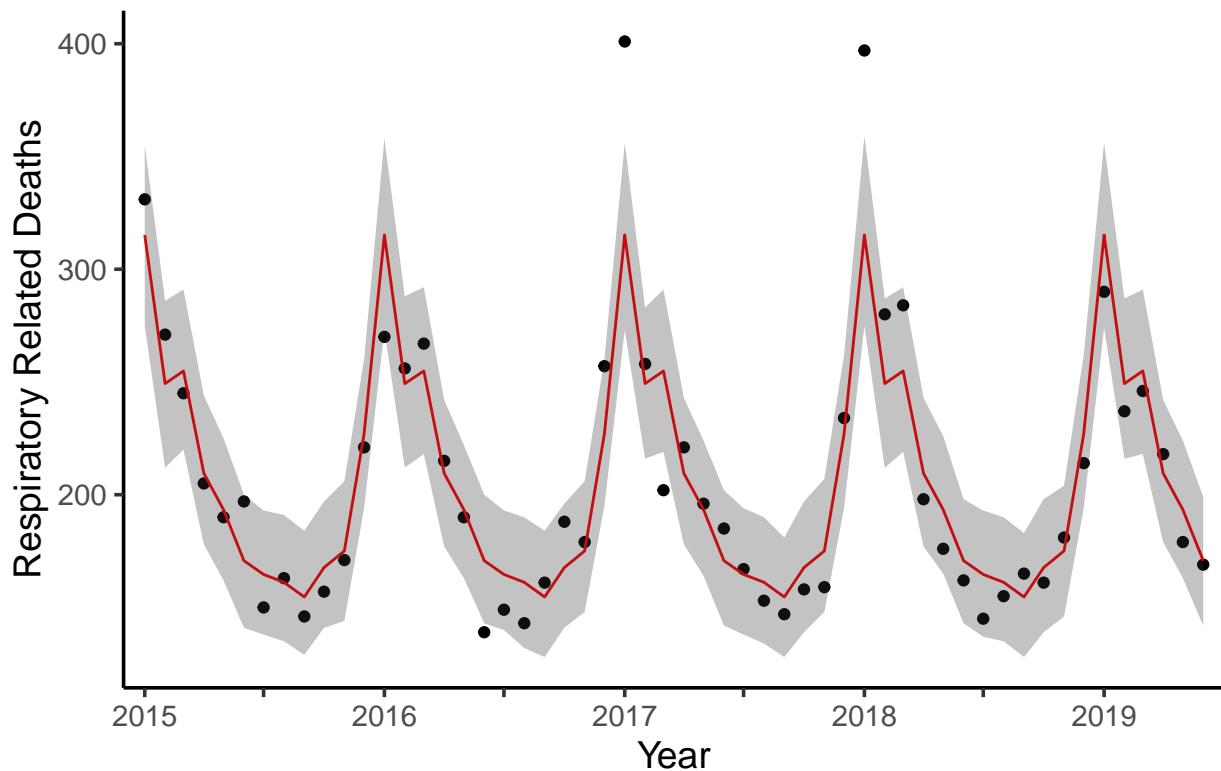
Cluster 2 (Coverage: 96.3%)



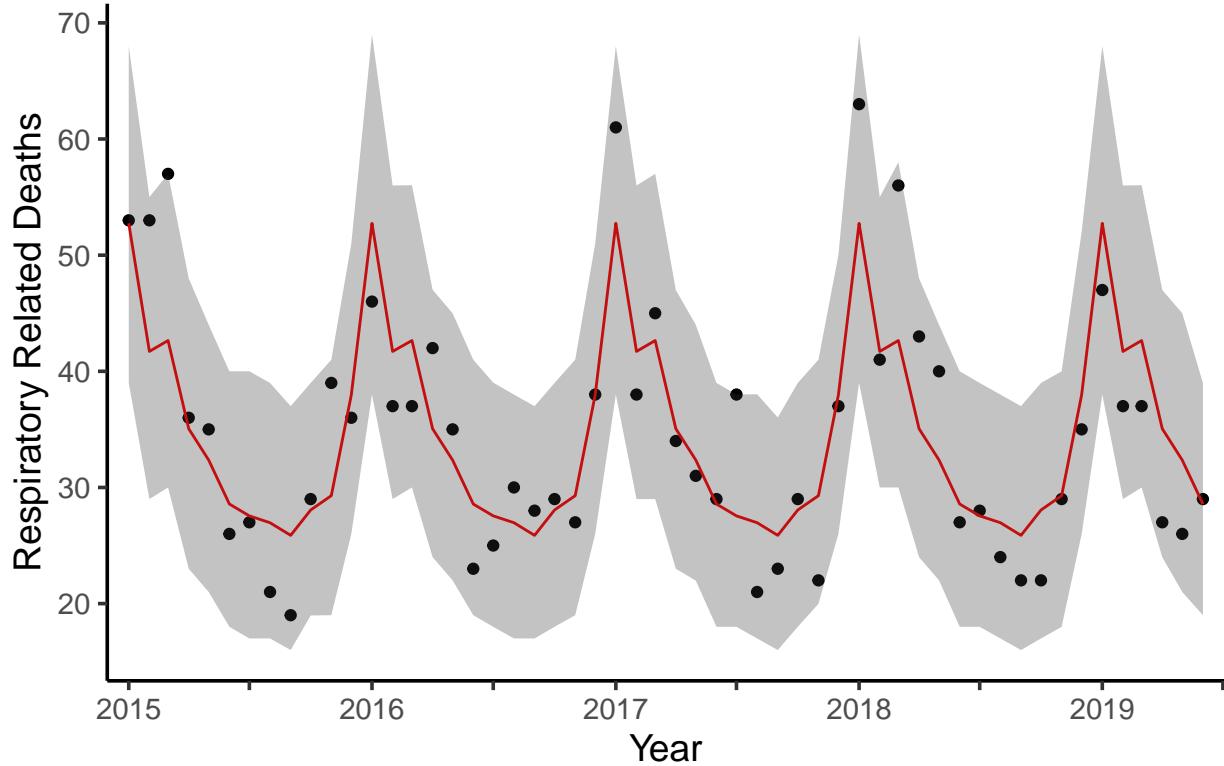
Cluster 3 (Coverage: 94.44%)



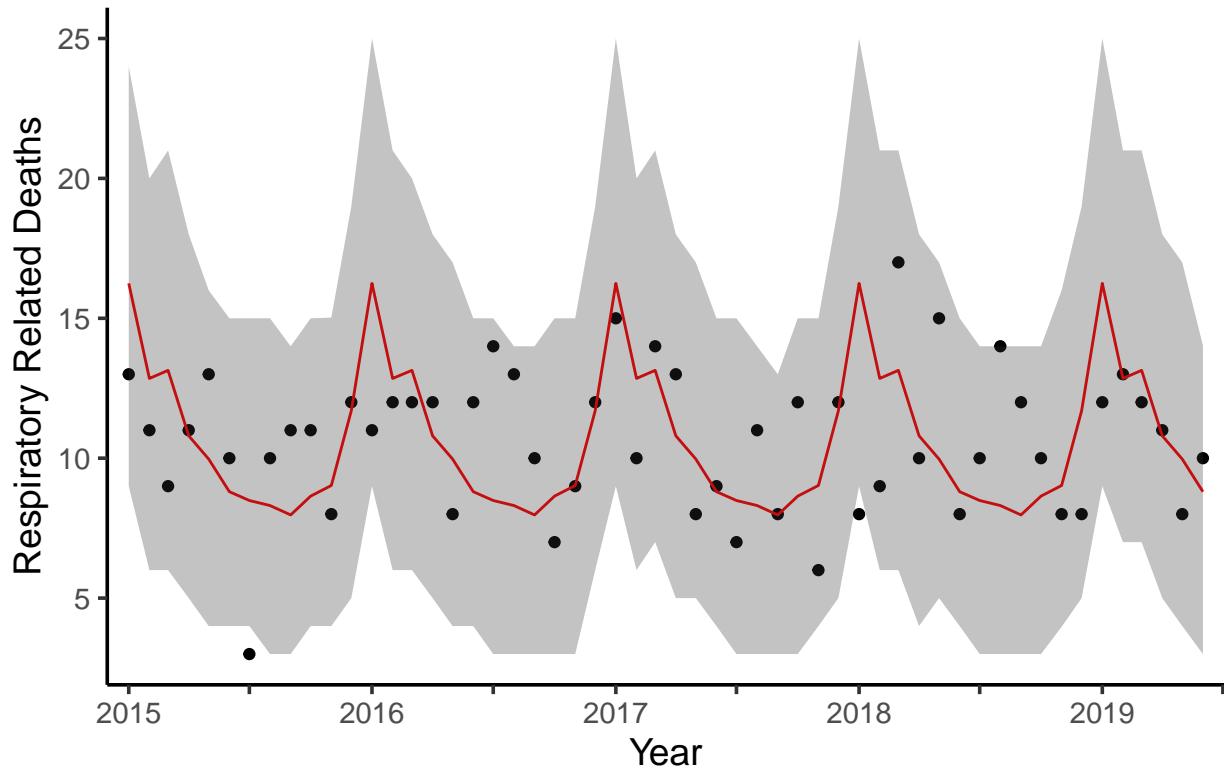
Cluster 4 (Coverage: 90.74%)



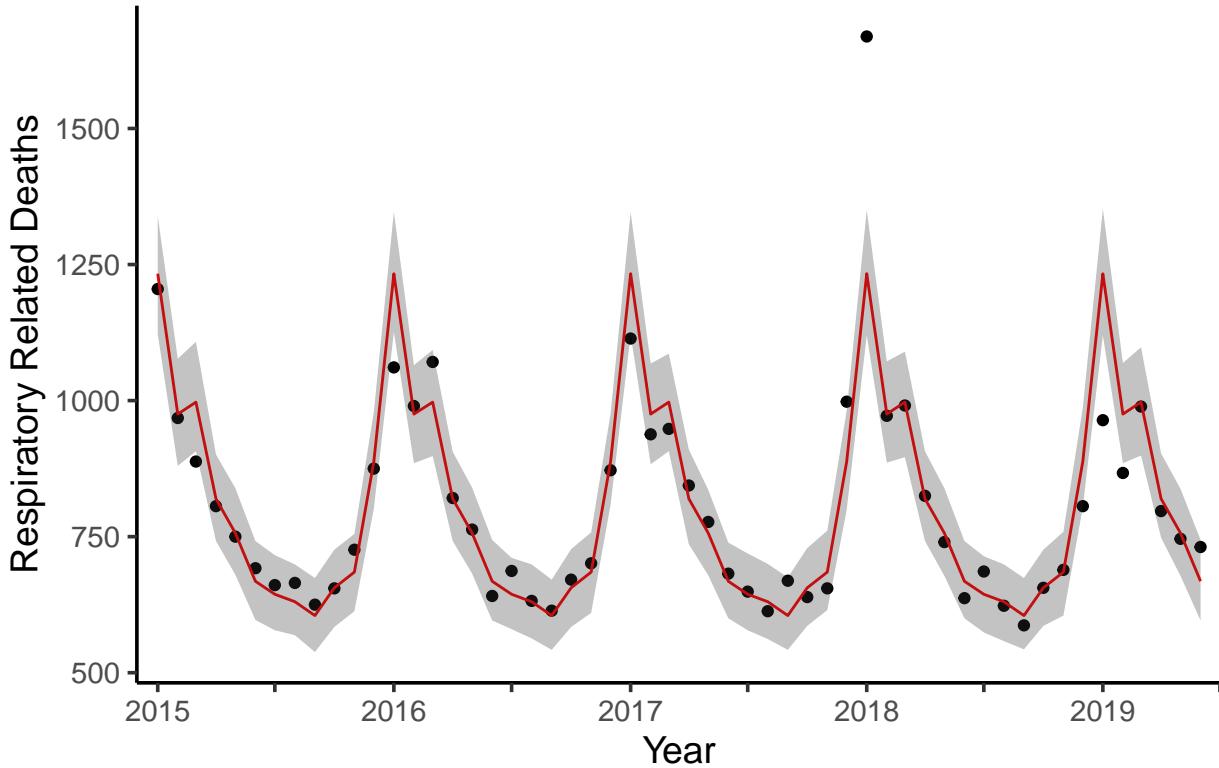
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 96.3%)



Cluster 7 (Coverage: 87.04%)



```
pp_insample_plot_combined(pred_data = kgr_model2_nb_fit$fitted_values, prefix = "kgr2-nb-insample-")
```

We can also simplify the covariance of our underlying GP and see how our proposed model compares with a simplified version with a simple time kernel:

$$\Lambda_{c,t} | \mathbf{F} = \exp(\beta_{c1} I\{c=1\} + \beta_{c2} I\{c=2\} + \dots + \beta_7 I\{c=7\} + \mathbf{F}_t)$$

where the graph signal $\mathbf{F} | \rho_{rbf}, \rho_p, \sigma_{time}^2 \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_{time})$ with $Cov(F_{c_1, t_1}, F_{c_2, t_2}) = [\mathbf{K}^{time}]_{t_1, t_2}$.

Instead of calculating gram matrix K based on covariate (EPA variables) similarity, our gram matrix K is simply a time kernel where similar values of t (months 1-60) have larger covariances. As a result, this model has no spatial dependence structure built in.

Calculating simple time kernel K

```
time_kernel = function(time_span,rho_rbf,rho_periodic,sigma2){
  K_time = matrix(NA,nrow = time_span, ncol = time_span)

  for (i in 1:time_span){
    for (j in 1:time_span){
      # K_time[i,j] = exp(- (abs(i-j)^2) / (2*rho)) * sigma2

      K_time[i,j] = exp(-(abs(i-j)^2) / (2*rho_rbf)) * exp(-(2*sin(sum(abs(i-j))*pi/12)^2)
                                              / (rho_periodic)) * sigma2
    }
  }
}
```

```

    return(K_time)
}

```

LGCP with temporal kernel (Reference model 3)

Since there is no spatial component in this model, each cluster can be fit separately.

```

ref_model3 = function(dataset, cluster, rho_time_rbf = 1, rho_time_periodic = 1, sigma2_time = 1, link=1)

  #Calculating gram matrix K_time
  K_time = time_kernel(time_span = length(unique(dataset$time)), rho_rbf = rho_time_rbf,
                        rho_periodic = rho_time_periodic, sigma2 = sigma2_time)

  #Heatmap of resulting K
  K_time_heatmap = matrix_heatmap(K_time, title = "K_time heatmap")

  #Calculate trace norm of gram matrix
  K_time_weight = norm(K_time, type = "F")

  #Need to ensure precision matrix is not computationally singular i.e det > 0
  covGP_jittered = desingularize(K_time, threshold = 1e-2, increment = 0.01)
  K_time = covGP_jittered[[1]]
  inv_K_time = solve(K_time)
  # cov_Fnorm = norm(K_time, type = "F")

  #Heatmap of resulting inv_K_time
  inv_K_time_heatmap = matrix_heatmap2(inv_K_time, title = "", legend_title = "Precision", prefix = "ref3-")

  ###Fitting the model on each cluster
  inla_test_clus_data = dataset %>% filter(id == cluster)

  ref_formula3 = response ~ -1 + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
    Intercept5 + Intercept6 + Intercept7 + f(time, model = "generic0", Cmatrix = inv_K_time)

  model = inla(ref_formula3, data = inla_test_clus_data, family = "poisson",
                control.compute = list(dic=TRUE, waic=TRUE),
                control.inla = list(strategy = "laplace"),
                control.predictor = list(compute = TRUE, link = link))

  ###Extract relevant information and store in the list
  model_summary <- model$summary.fixed
  bri_hyperpar_summary <- bri.hyperpar.summary(model)
  model_DIC <- model$dic$dic
  model_WAIC <- model$waic$waic
  preds_model <- model$summary.fitted.values
  preds_model <- cbind(inla_test_clus_data$id, inla_test_clus_data$time, preds_model)
  colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")

  #Exponentiating parameter to get better interpretation of estimates
  multeff <- exp(model$summary.fixed$mean)
  names(multeff) <- model$names.fixed

```

```

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf,Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
ref_model3_results = list(
  K_time_heatmap = K_time_heatmap,
  K_time_weight = K_time_weight/(K_time_weight + gfilter_weight),
  gfilter_weight = gfilter_weight/(K_time_weight + gfilter_weight),
  covmatrix = K_time,
  prec = inv_K_time,
  num_jitters = covGP_jittered[[2]],
  prec_heatmap = inv_K_time_heatmap,
  model_summary = model_summary,
  bri_hyperpar_summary = bri_hyperpar_summary,
  exp_effects = multeff,
  param_plot = param_plot,
  hyperparam_plot = hyperparam_plot,
  modelDIC = modelDIC,
  modelWAIC = modelWAIC,
  fitted_values = preds_model
)

return(ref_model3_results)
}

#Fit ref_model3 on one cluster (to test)
ref_model3_fit = ref_model3(dataset = inla_insample_data, cluster = 2, rho_time_rbf = 1,
rho_time_periodic = 1, sigma2_time = 5)

#Extract DIC and WAIC
ref_model3_DIC = ref_model3_fit$modelDIC
ref_model3_WAIC = ref_model3_fit$modelWAIC

#Get summaries of parameter estimates
ref_model3_fit$model_summary

##               mean          sd 0.025quant    0.5quant 0.975quant
## Intercept1 -1.669285e-15 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept2  3.814774e+00  0.04580621   3.723606 3.815080e+00  3.904215
## Intercept3 -1.669285e-15 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept4 -1.669285e-15 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept5 -1.669285e-15 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept6 -1.669285e-15 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept7 -1.669285e-15 31.62254151 -62.008668 5.489991e-14 62.008668
##               mode          kld

```

```

## Intercept1 -1.055939e-14 5.526823e-11
## Intercept2  3.815078e+00 4.686553e-08
## Intercept3 -1.055939e-14 5.526823e-11
## Intercept4 -1.055939e-14 5.526823e-11
## Intercept5 -1.055939e-14 5.526823e-11
## Intercept6 -1.055939e-14 5.526823e-11
## Intercept7 -1.055939e-14 5.526823e-11

ref_model3_fit$bri_hyperpar_summary

##               mean        sd      q0.025      q0.5      q0.975      mode
## SD for time 0.08981184 0.01317405 0.06656631 0.08886568 0.1183232 0.08717421

ref_model3_fit$exp_effects

## Intercept1 Intercept2 Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##      1.0000    45.3665    1.0000    1.0000    1.0000    1.0000    1.0000

ref_model3_fit$K_time_weight

## [1] 0.9949444

ref_model3_fit$gfilter_weight

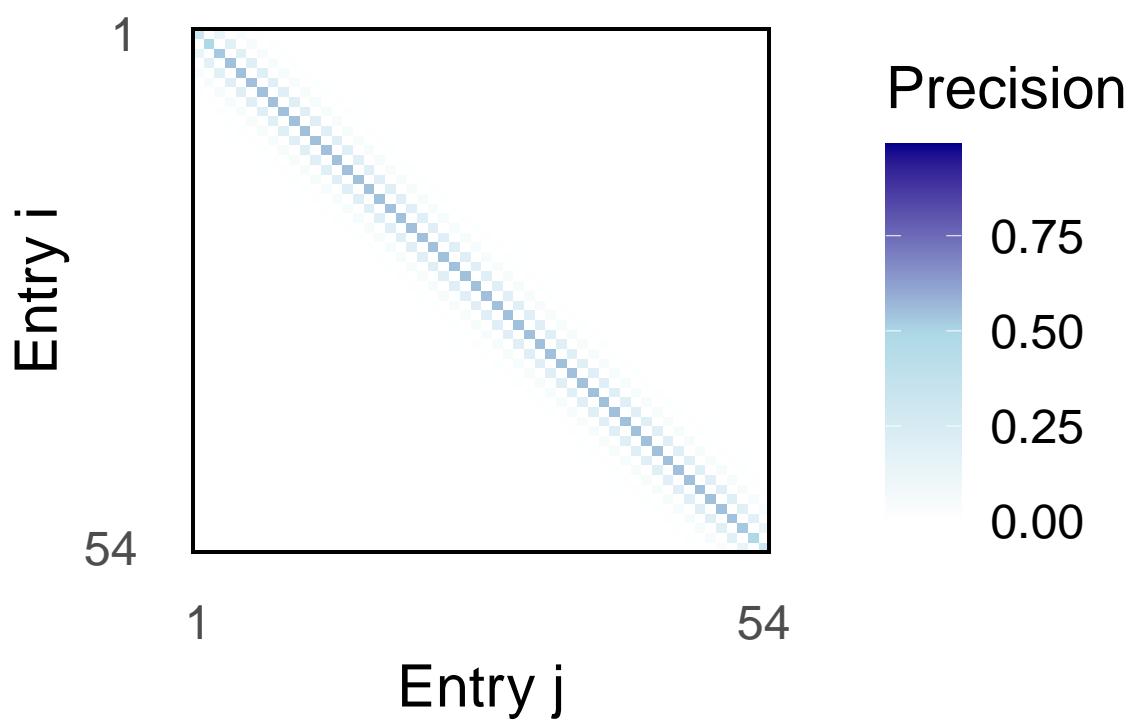
## [1] 0.005055571

ref_model3_fit$num_jitters

## [1] 0

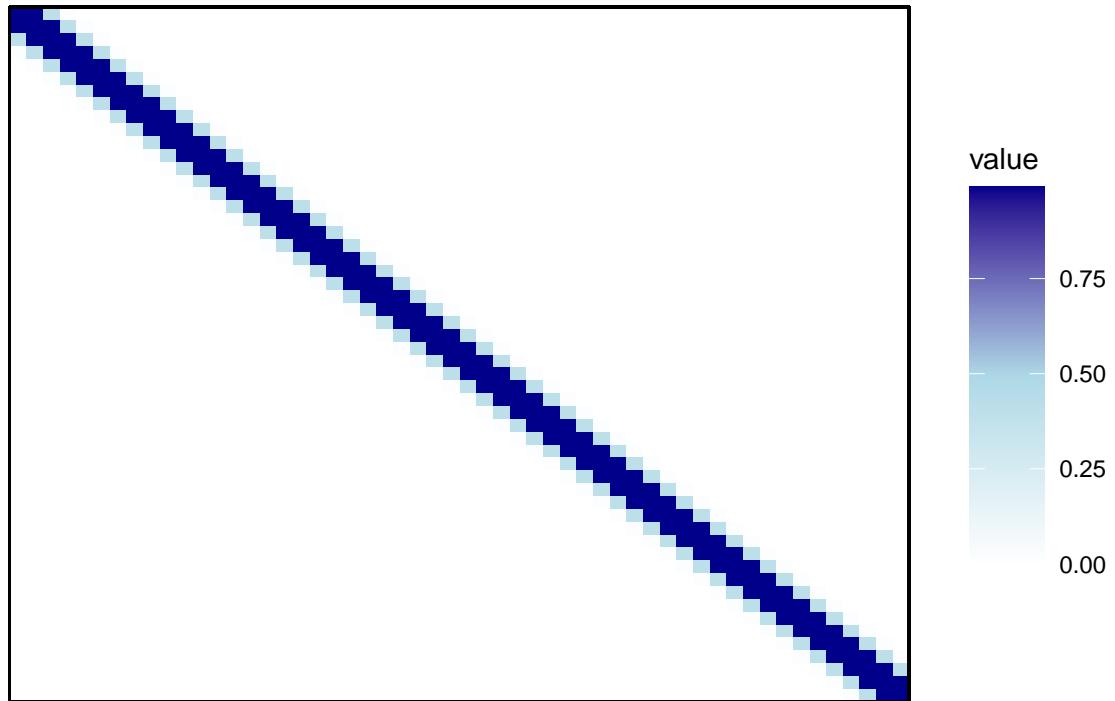
#Show plots
ref_model3_fit$prec_heatmap

```

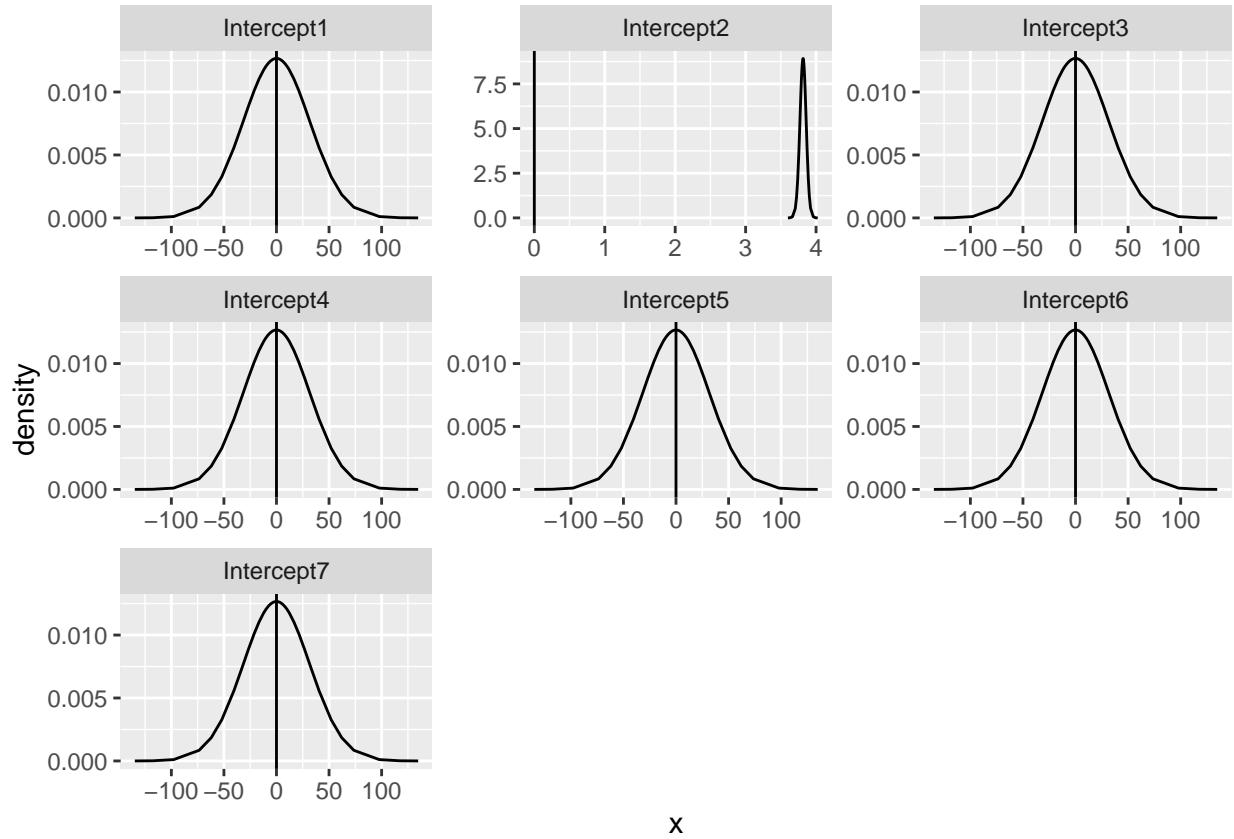


ref_model3_fit\$K_time_heatmap

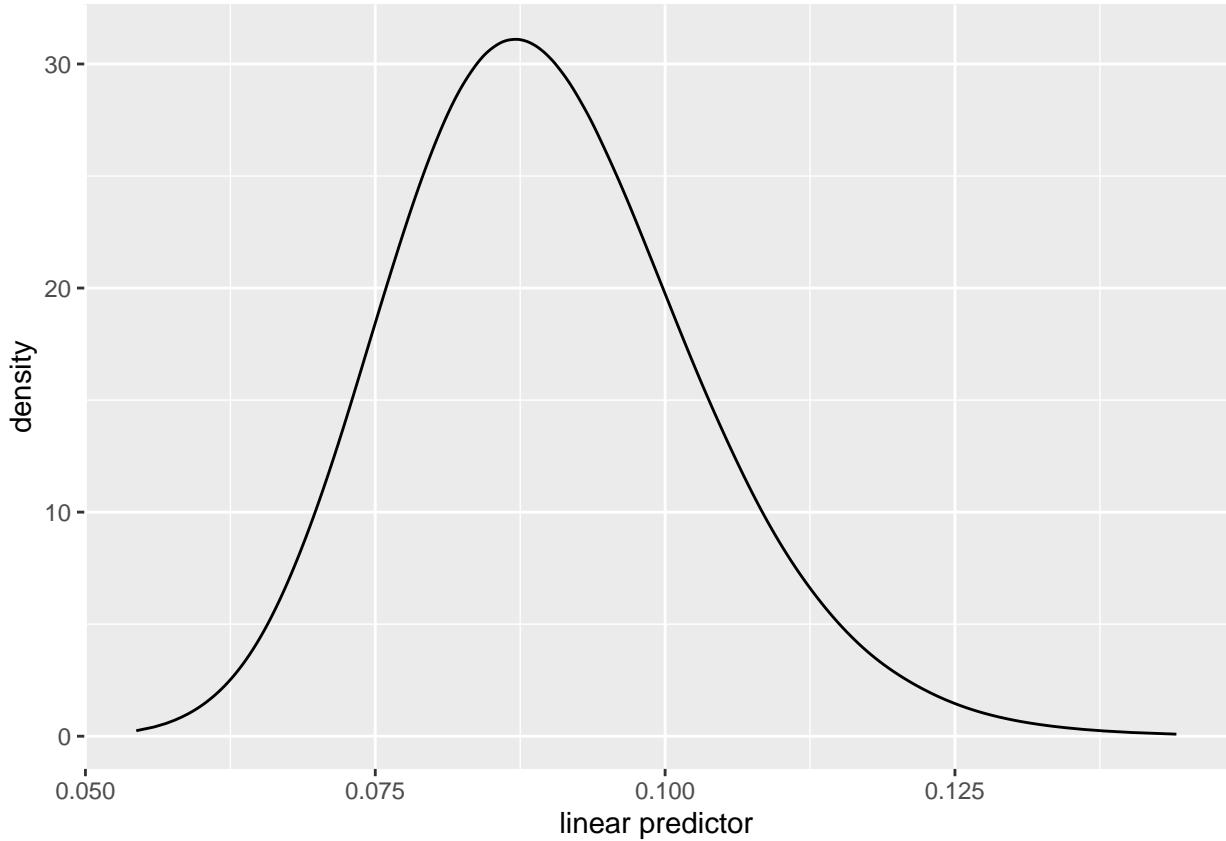
K_time heatmap



```
ref_model3_fit$param_plot
```



```
ref_model3_fit$hyperparam_plot
```



```

test1 = ref_model3(dataset = inla_insample_data, cluster = 1, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test2 = ref_model3(dataset = inla_insample_data, cluster = 2, rho_time_rbf = 498.918,
                   rho_time_periodic = 120.307, sigma2_time = 0.014)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test3 = ref_model3(dataset = inla_insample_data, cluster = 3, rho_time_rbf = 8.090,
                   rho_time_periodic = 468.170, sigma2_time = 2.428)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test4 = ref_model3(dataset = inla_insample_data, cluster = 4, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802) #why does sigma2 have to be so big here

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

```

test5 = ref_model3(dataset = inla_insample_data, cluster = 5, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test6 = ref_model3(dataset = inla_insample_data, cluster = 6, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802) #why does sigma2 have to be so big here

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test7 = ref_model3(dataset = inla_insample_data, cluster = 7, rho_time_rbf = 474.985,
                   rho_time_periodic = 2.050, sigma2_time = 1.489)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

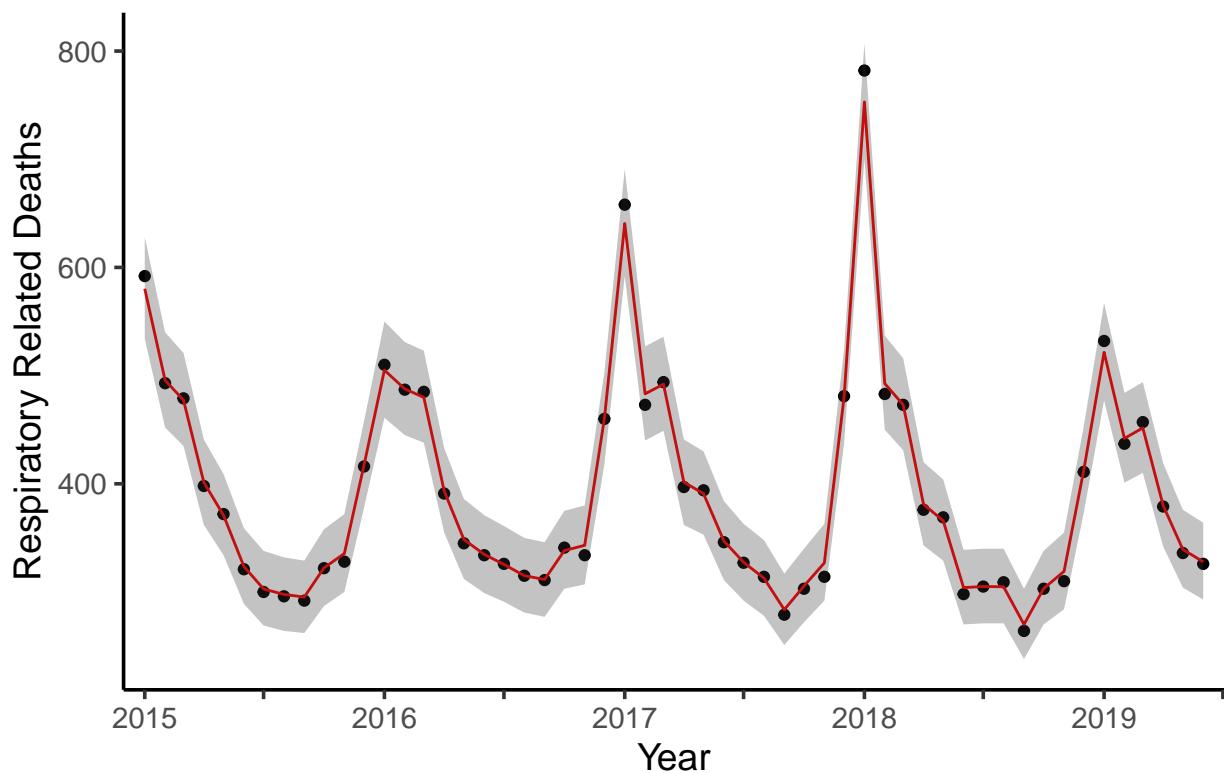
test1$fitted_values = poisson_pp_sampling(test1$fitted_values,n=100000)
test2$fitted_values = poisson_pp_sampling(test2$fitted_values,n=100000)
test3$fitted_values = poisson_pp_sampling(test3$fitted_values,n=100000)
test4$fitted_values = poisson_pp_sampling(test4$fitted_values,n=100000)
test5$fitted_values = poisson_pp_sampling(test5$fitted_values,n=100000)
test6$fitted_values = poisson_pp_sampling(test6$fitted_values,n=100000)
test7$fitted_values = poisson_pp_sampling(test7$fitted_values,n=100000)

ref_model3_fvs = rbind(test1$fitted_values,test2$fitted_values,test3$fitted_values,
                       test4$fitted_values,test5$fitted_values,test6$fitted_values,
                       test7$fitted_values)

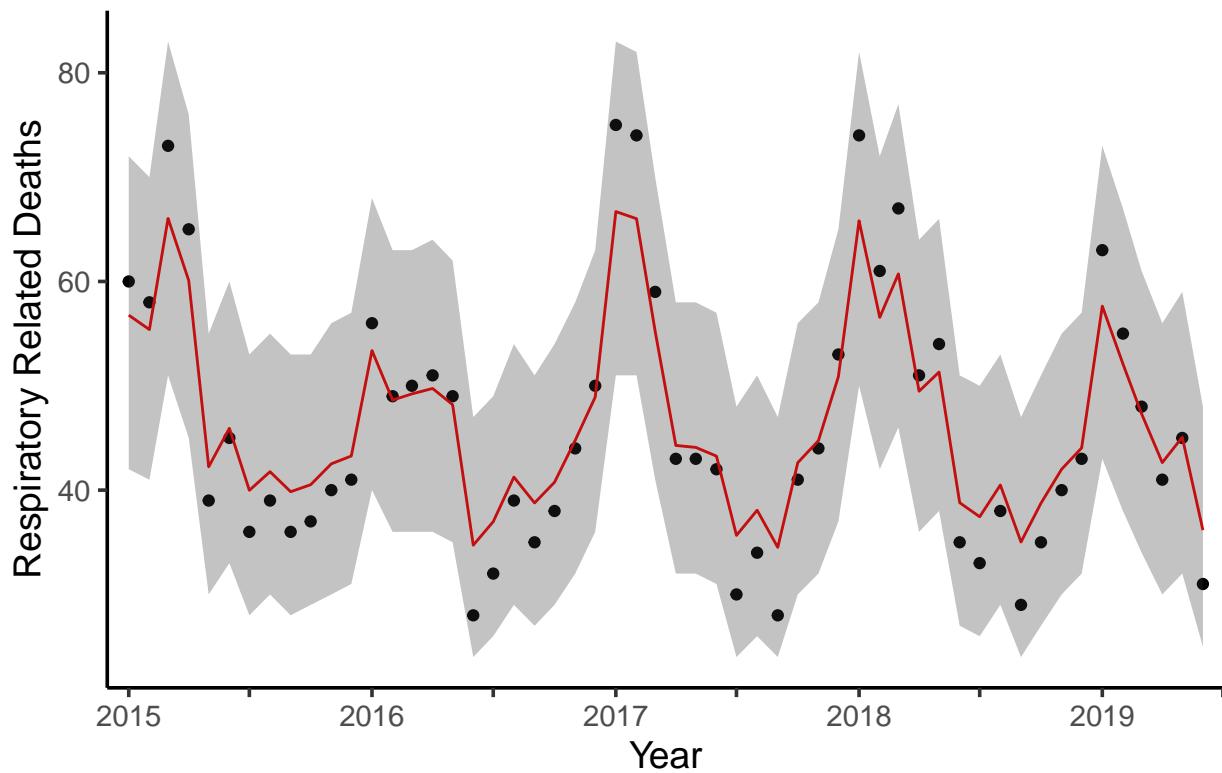
pp_insample_plot(num_plots = num_clus,ref_data = inla_insample_data,pred_data = ref_model3_fvs,prefix =

```

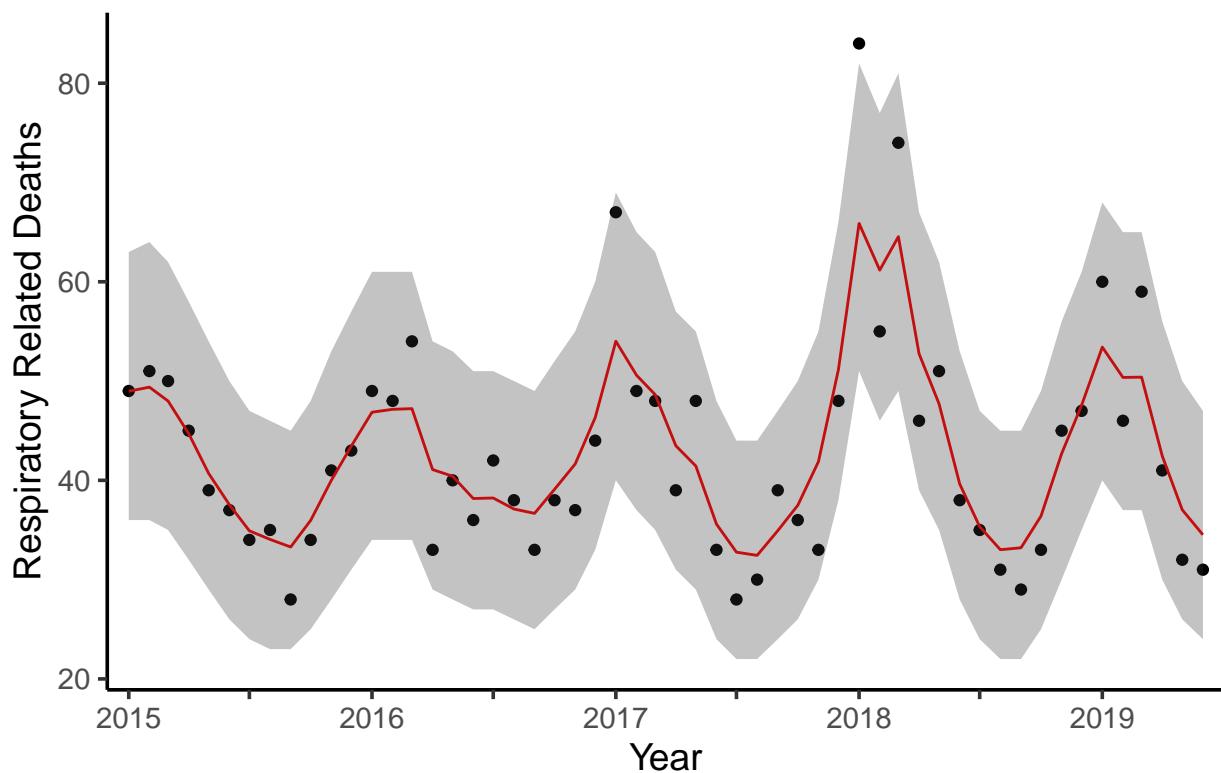
Cluster 1 (Coverage: 100%)



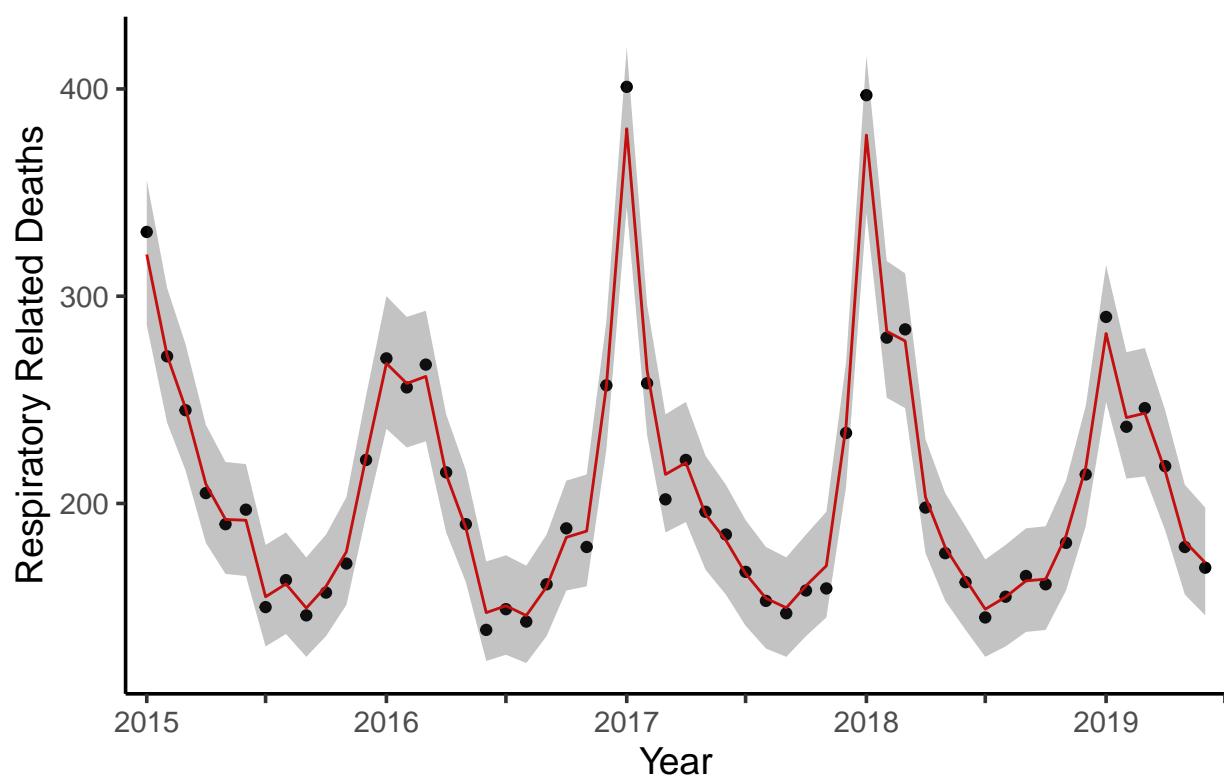
Cluster 2 (Coverage: 100%)



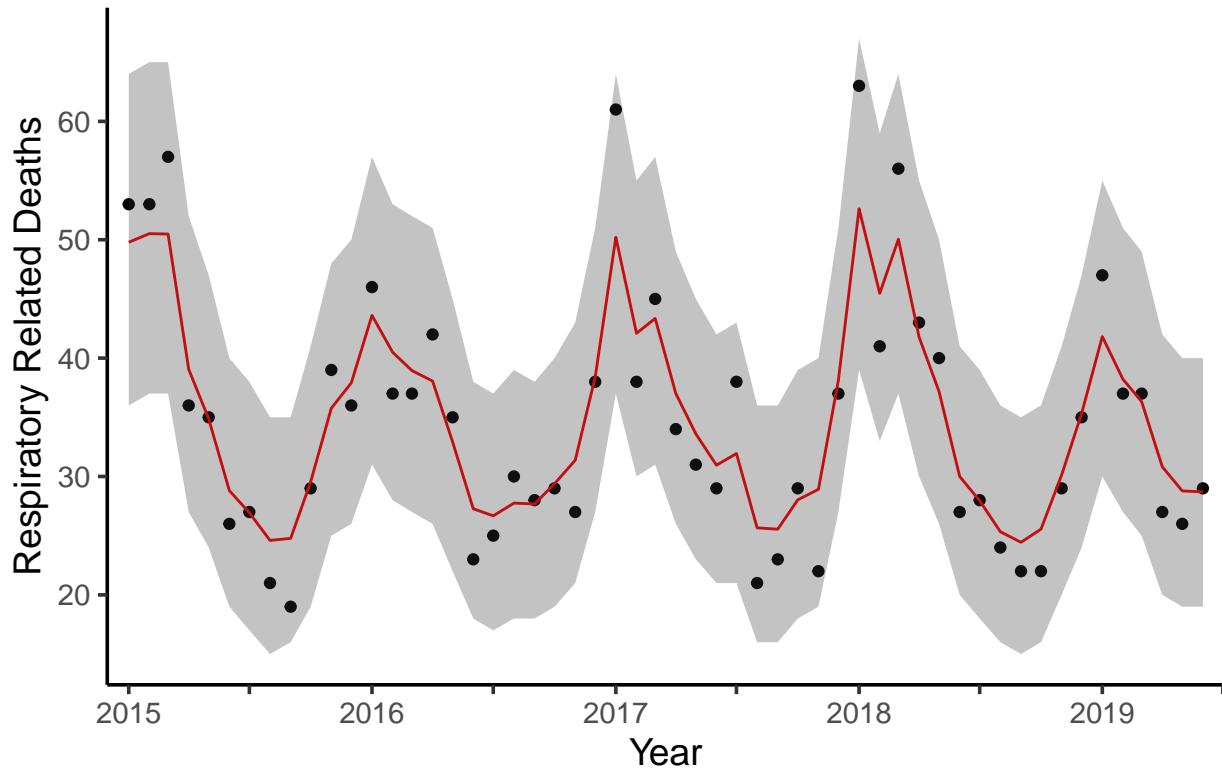
Cluster 3 (Coverage: 98.15%)



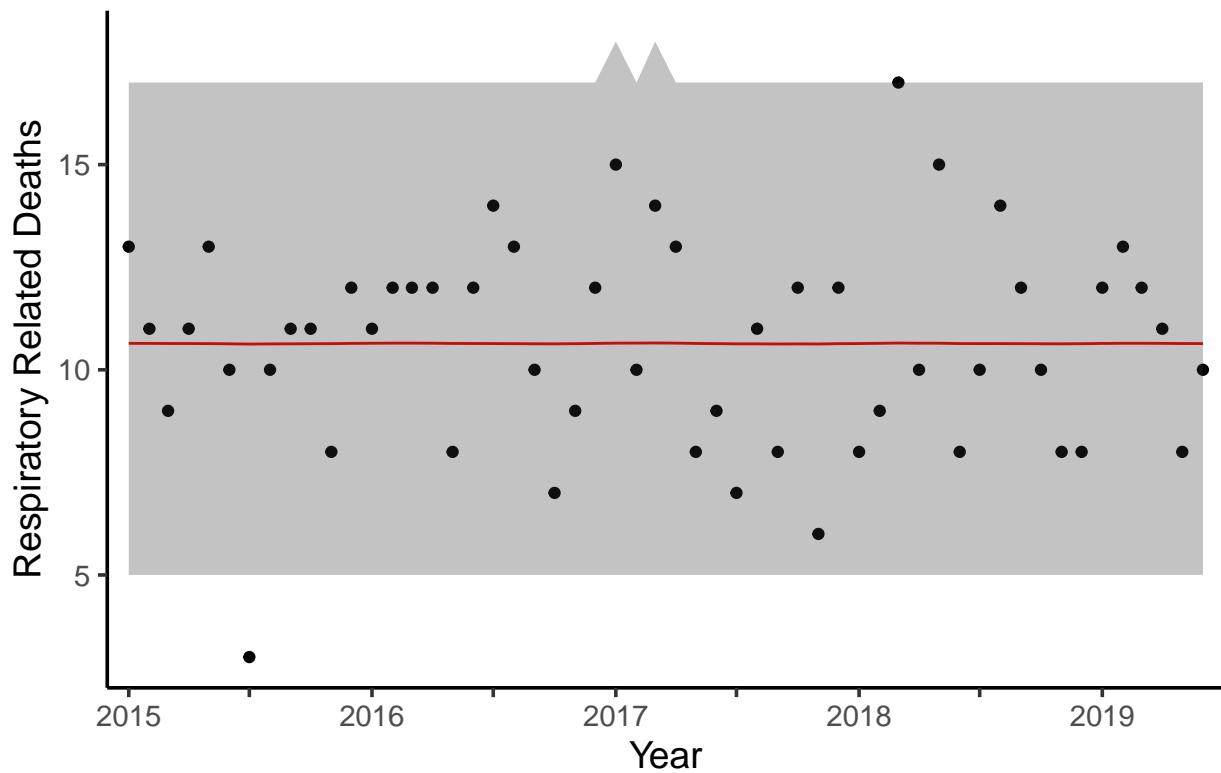
Cluster 4 (Coverage: 100%)



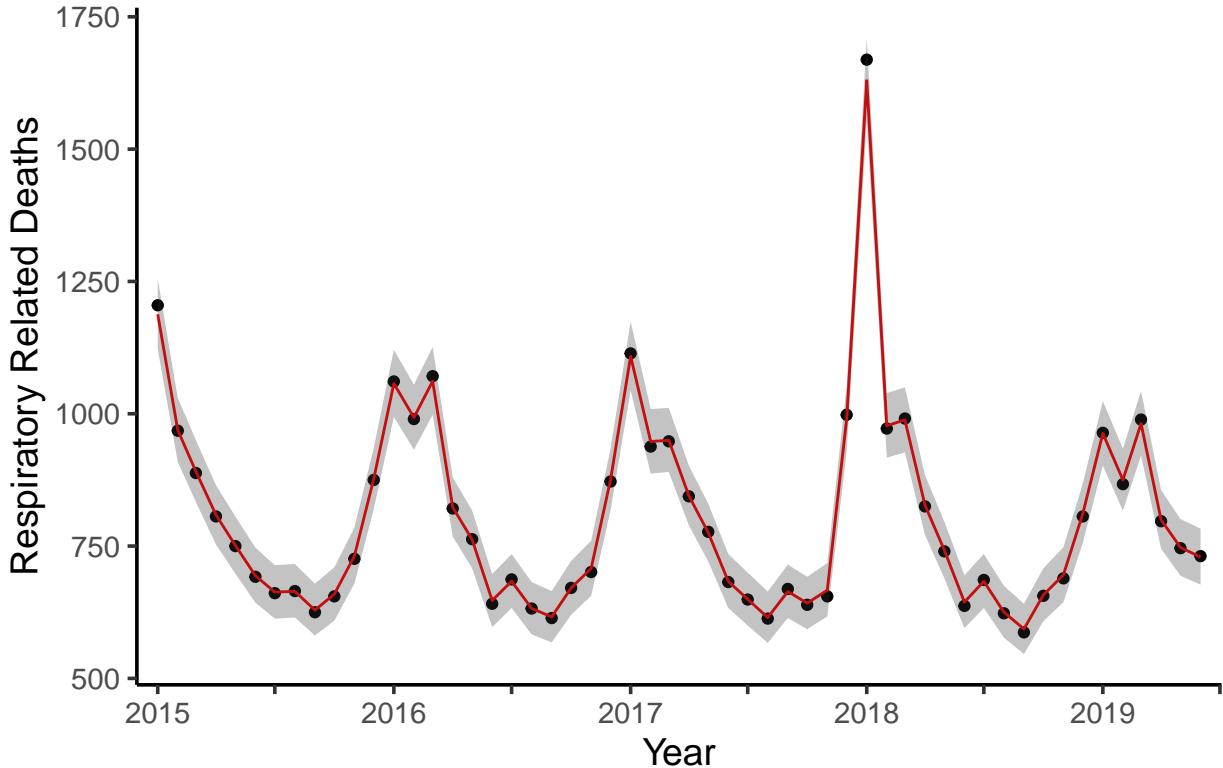
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 98.15%)



Cluster 7 (Coverage: 100%)



```
pp_insample_plot_combined(num_plots = num_clus, ref_data = inla_insample_data, pred_data = ref_model3_fvs
```

Reference model 4 (simple time kernel + EPA covariates)

```
ref_model4 = function(dataset, rho_time_rbf = 1, rho_time_periodic = 1, sigma2_time = 1, link=1){
  #Calculating gram matrix K_time
  K_time = time_kernel(time_span = length(unique(dataset$time)), rho_rbf = rho_time_rbf,
                        rho_periodic = rho_time_periodic, sigma2 = sigma2_time)

  #Heatmap of resulting K
  K_time_heatmap = matrix_heatmap(K_time, title = "K_time heatmap")

  #Calculate trace norm of gram matrix
  K_time_weight = norm((1/60)*K_time, type = "F")

  #Calculate proposed kernel
  covGP2 = kronecker(K_time/60, (H^2)/7)

  #Need to ensure precision matrix is not computationally singular i.e det > 0
  covGP_jittered = desingularize(covGP2, threshold = 1e-2, increment = 0.01)
  covGP2 = covGP_jittered[[1]]

  inv_covGP2 = solve(covGP2)

  #Heatmap of resulting inv_covGP2
  inv_covGP2_heatmap = matrix_heatmap2(inv_covGP2, title = "", legend_title = "Precision", prefix = "ref4-")
```

```

####Fit INLA model
ref_model4 = response ~ -1 + lead + co + so2 + no2 + o3 + pm10 + pm25 +
Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5 + Intercept6 +
Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP2)

# ref_model4 = response ~ -1 + months + lead + co + so2 + no2 + o3 + pm10 + pm25 +
# Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5 + Intercept6 +
# Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP2)

model = inla(formula = ref_model4,family = "poisson",data = dataset,
control.compute = list(dic=TRUE,waic=TRUE,
return.marginals.predictor=TRUE),
control.inla = list(strategy = "laplace"),
control.predictor = list(compute = TRUE, link = link))

####Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hyperpar_summary <- bri.hyperpar.summary(model)
modelDIC <- model$dic$dic
modelWAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf,Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
ref_model4_results = list(
K_time_heatmap = K_time_heatmap,
K_time_weight = K_time_weight/(K_time_weight + gfilter_weight),
gfilter_weight = gfilter_weight/(K_time_weight + gfilter_weight),
covmatrix = covGP2,
prec = inv_covGP2,
num_jitters = covGP_jittered[[2]],
prec_heatmap = inv_covGP2_heatmap,
model_summary = model_summary,
bri_hyperpar_summary = bri_hyperpar_summary,
exp_effects = multeff,

```

```

    param_plot = param_plot,
    hyperparam_plot = hyperparam_plot,
    modelDIC = modelDIC,
    modelWAIC = modelWAIC,
    fitted_values = preds_model,
    marg_fitted_values = marginal_fvs
  )

  return(ref_model4_results)
}

#Fit ref_model4
ref_model4_fit = ref_model4(dataset = inla_insample_data_wcovariates,rho_time_rbf = 483.957,rho_time_pe...
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha...
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
#Posterior predictive sampling from estimated intensities
ref_model4_fit$fitted_values = poisson_pp_sampling(ref_model4_fit$fitted_values,n=100000)

#Extract DIC and WAIC
ref_model4_DIC = ref_model4_fit$modelDIC
ref_model4_WAIC = ref_model4_fit$modelWAIC

#Get summaries of parameter estimates
ref_model4_fit$model_summary

##               mean        sd  0.025quant   0.5quant   0.975quant
## lead      -0.024199245 0.01488364 -0.05341095 -0.024196128 0.004994640
## co         0.054393437 0.01911385  0.01688352  0.054395466 0.091891761
## so2        -0.002898188 0.01292425 -0.02829349 -0.002885409 0.022424328
## no2        0.054061270 0.01586268  0.02290415  0.054072809 0.085152737
## o3         -0.010810220 0.01523287 -0.04071606 -0.010804188 0.019061247
## pm10       -0.076350666 0.01250546 -0.10079408 -0.076383701 -0.051719195
## pm25       -0.016413067 0.01131100 -0.03862461 -0.016406780 0.005762739
## Intercept1 5.940529542 0.07022276  5.80265709  5.940533893 6.078377341
## Intercept2 3.809115339 0.05774869  3.69570024  3.809141455 3.922381603
## Intercept3 3.719515117 0.07590623  3.57041635  3.719553466 3.868395576
## Intercept4 5.312603740 0.07904668  5.15741677  5.312606230 5.467776573
## Intercept5 3.520289156 0.04981116  3.42245160  3.520321693 3.617941075
## Intercept6 2.345872853 0.09490577  2.15959082  2.345888392 2.532065997
## Intercept7 6.672843180 0.04374688  6.58696140  6.672842741 6.758727468
##               mode        kld
## lead      -0.02419614 5.197442e-10
## co         0.05439546 5.932652e-10
## so2        -0.00288539 8.016699e-10
## no2        0.05407285 6.788581e-10
## o3         -0.01080418 5.953444e-10
## pm10       -0.07638377 2.302581e-09
## pm25       -0.01640675 6.521671e-10
## Intercept1 5.94053393 2.438847e-09
## Intercept2 3.80914143 1.408791e-09
## Intercept3 3.71955349 1.711296e-09
## Intercept4 5.31260624 2.346651e-09

```

```

## Intercept5 3.52032164 9.137449e-10
## Intercept6 2.34588830 8.650204e-10
## Intercept7 6.67284274 2.434757e-09
ref_model4_fit$bri_hyperpar_summary

##               mean        sd   q0.025    q0.5   q0.975     mode
## SD for id2 1.278435 0.06845017 1.150027 1.276184 1.418838 1.271884
ref_model4_fit$exp_effects

##      lead       co       so2       no2       o3       pm10
## 0.9760912 1.0559000 0.9971060 1.0555493 0.9892480 0.9264913
##      pm25 Intercept1 Intercept2 Intercept3 Intercept4 Intercept5
## 0.9837209 380.1361742 45.1105137 41.2443906 202.8777822 33.7941988
## Intercept6 Intercept7
## 10.4423834 790.6403448
ref_model4_fit$K_time_weight

## [1] 0.8959784
ref_model4_fit$gfilter_weight

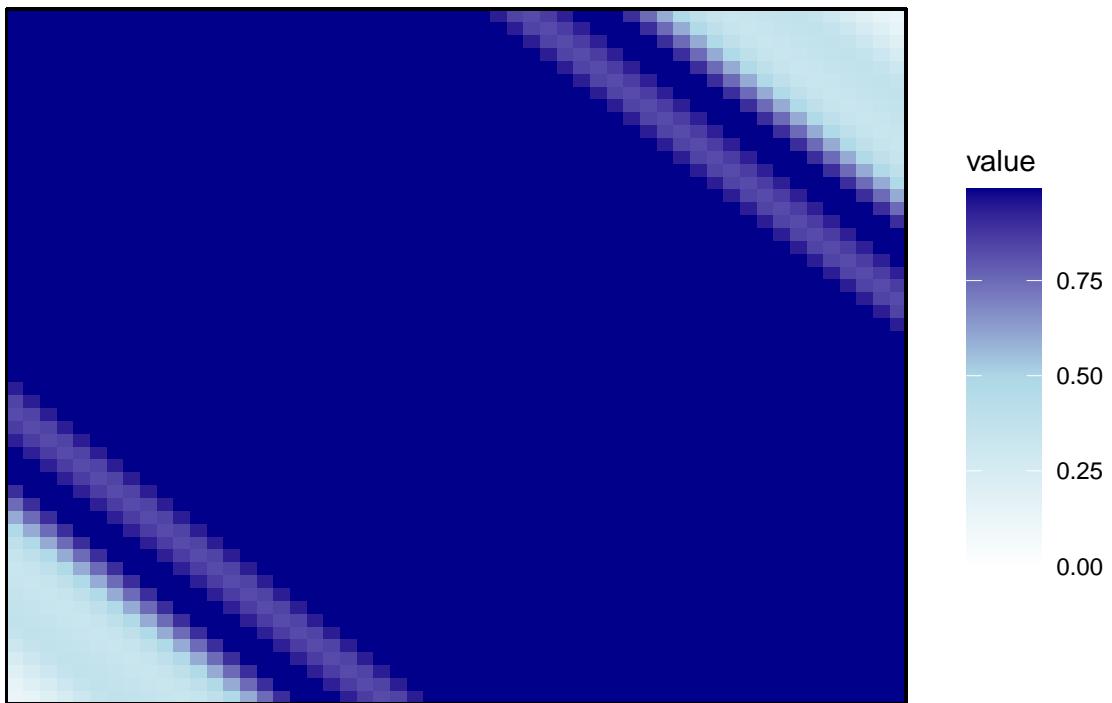
## [1] 0.1040216
ref_model4_fit$num_jitters

## [1] 1
#Show plots
ref_model4_fit$K_time_heatmap

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
## a single row.

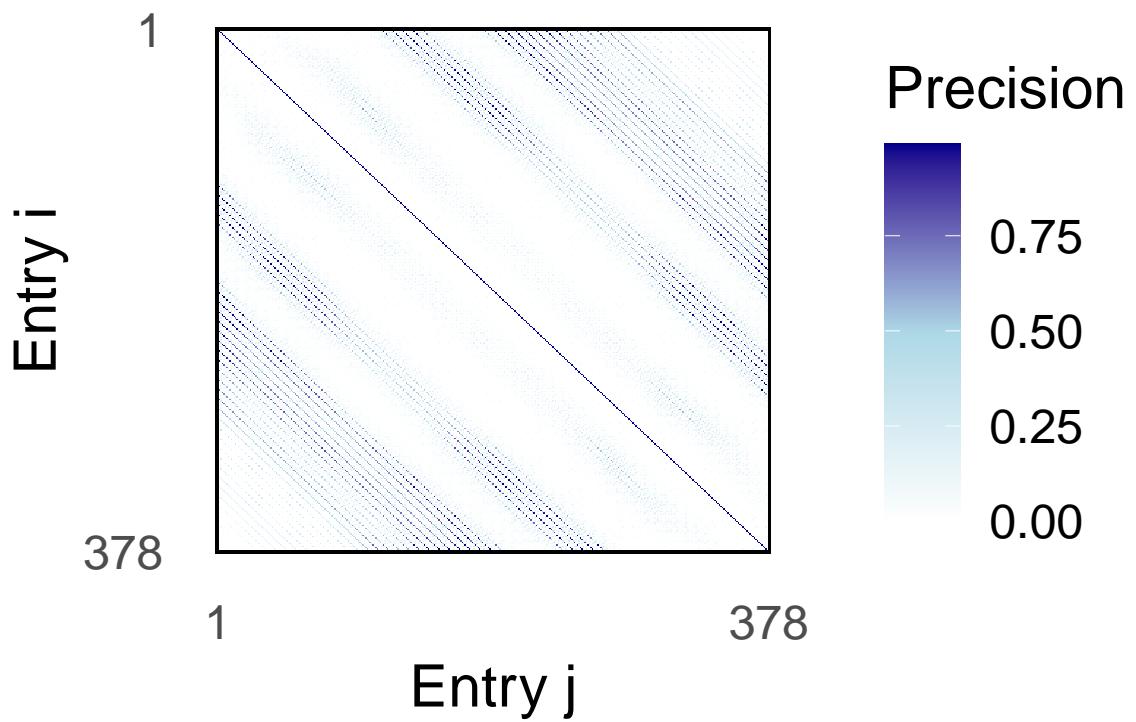
```

K_time heatmap

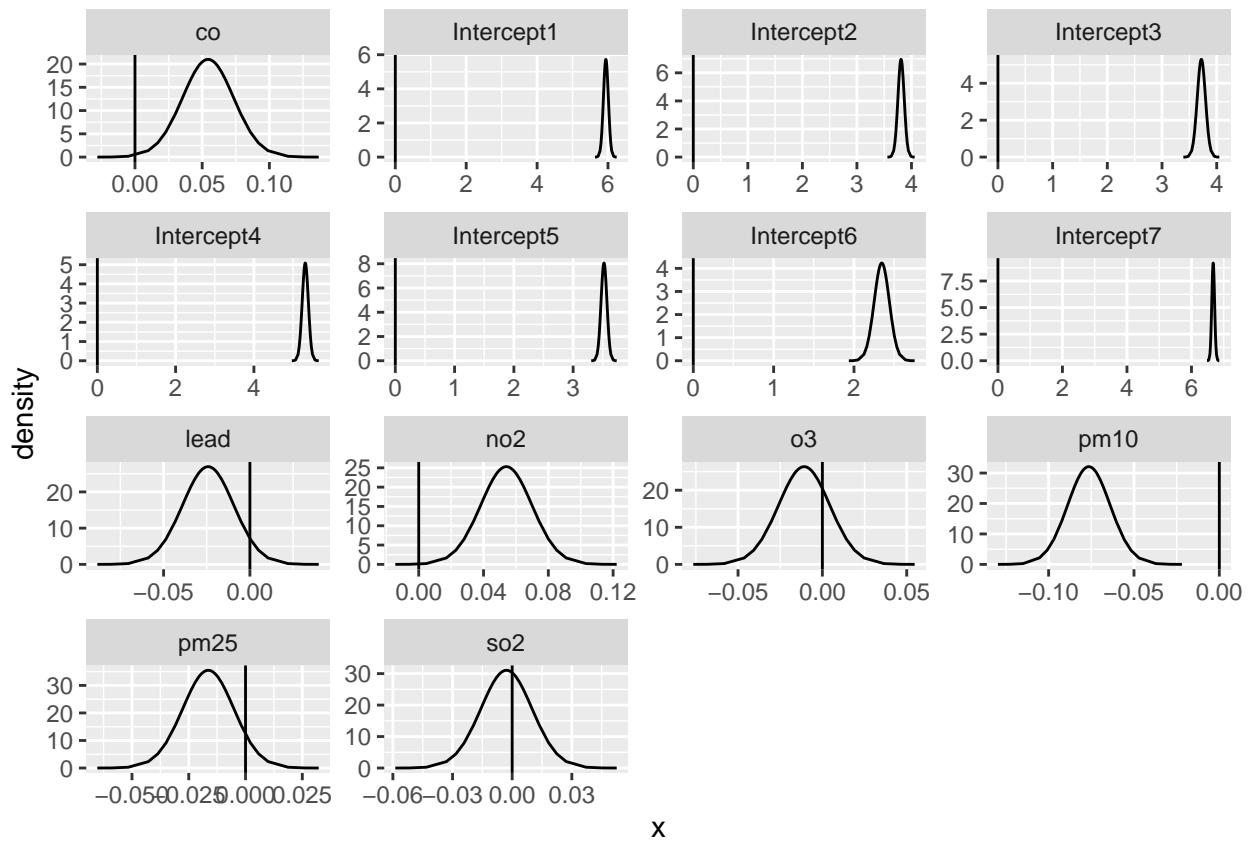


```
ref_model4_fit$prec_heatmap
```

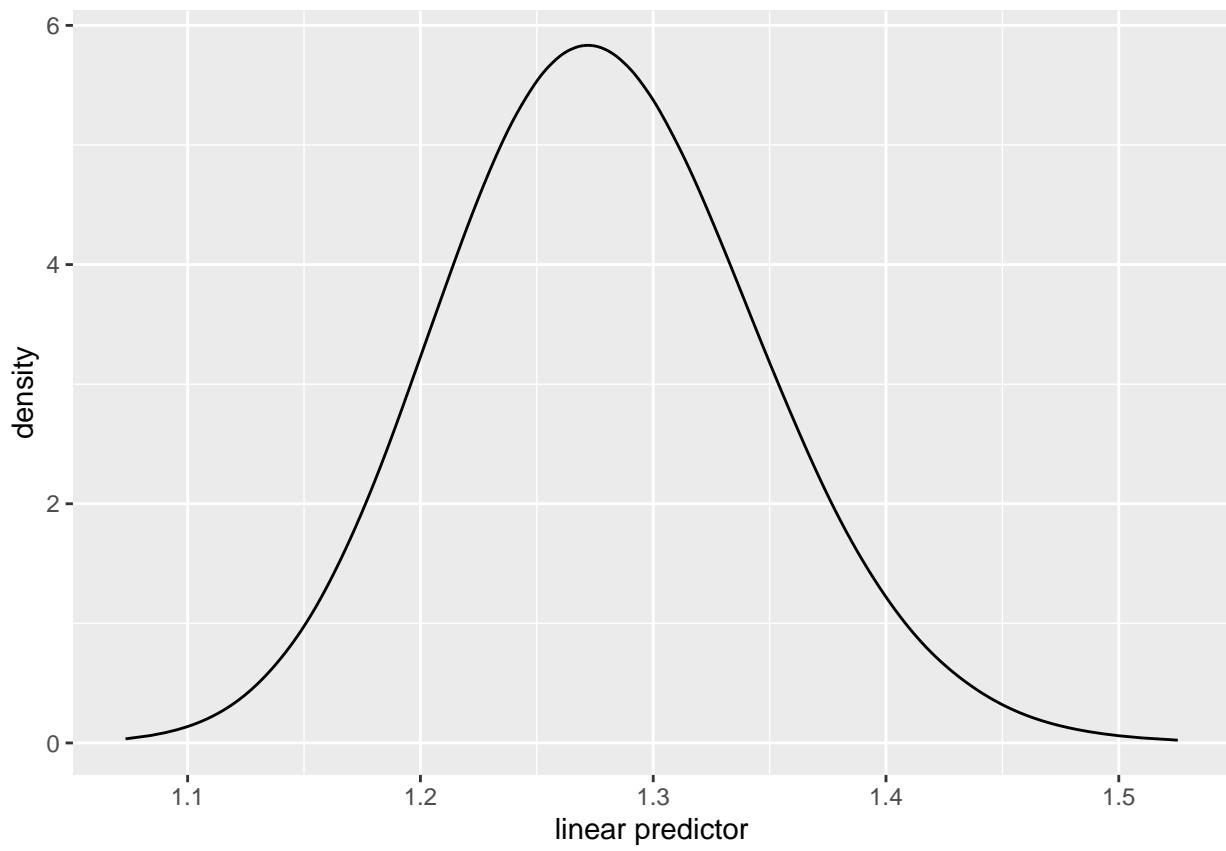
```
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



```
ref_model4_fit$param_plot
```

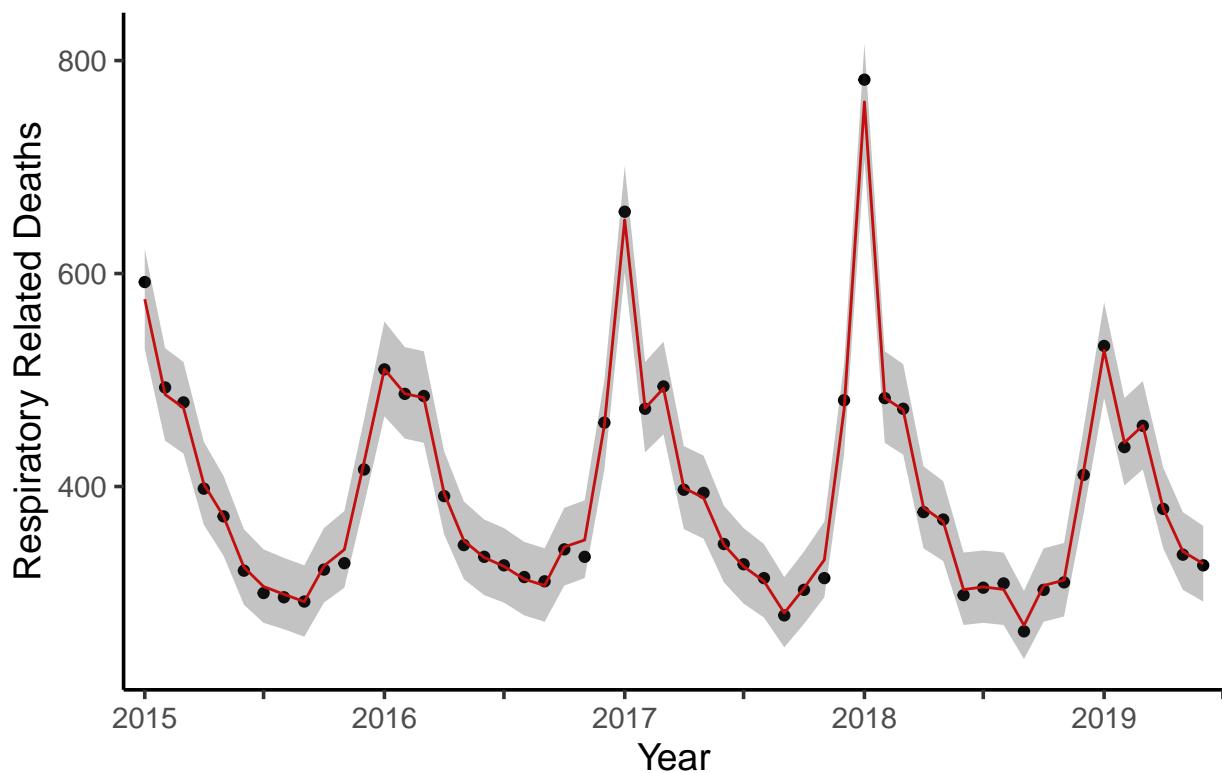


```
ref_model4_fit$hyperparam_plot
```

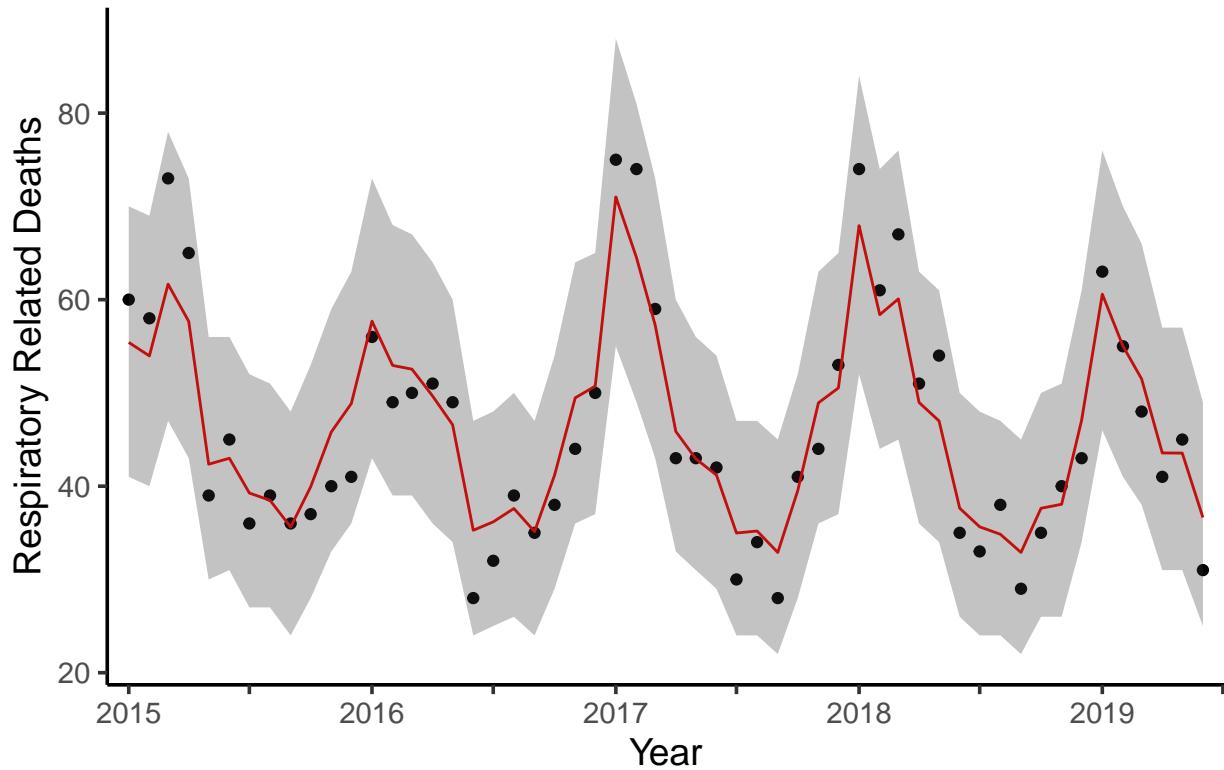


```
pp_insample_plot(pred_data = ref_model4_fit$fitted_values, prefix = "ref4-insample-")
```

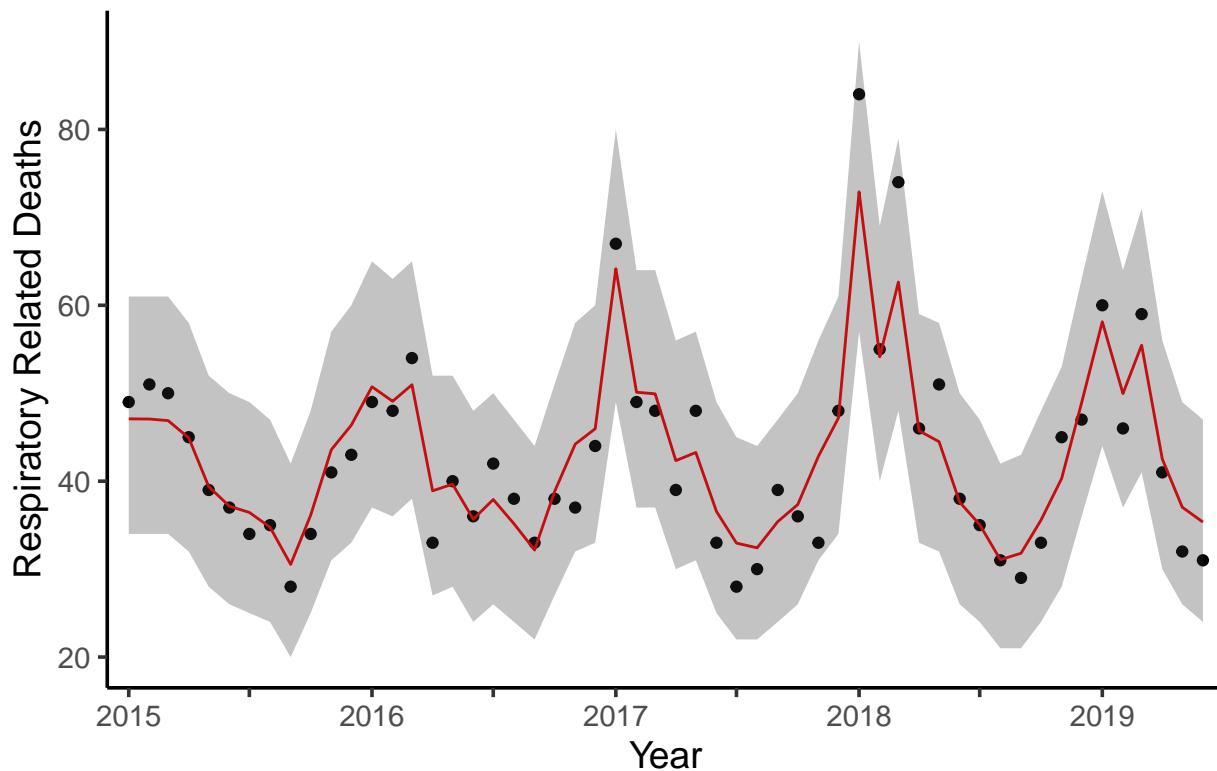
Cluster 1 (Coverage: 100%)



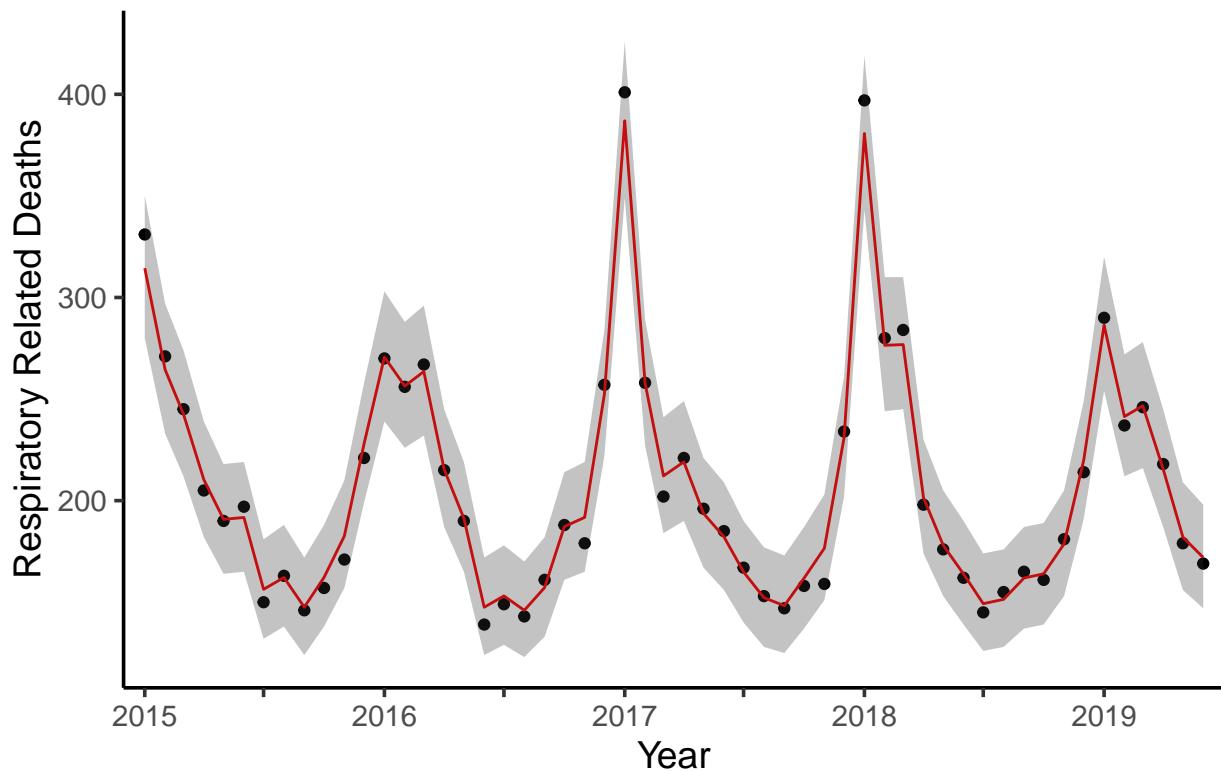
Cluster 2 (Coverage: 100%)



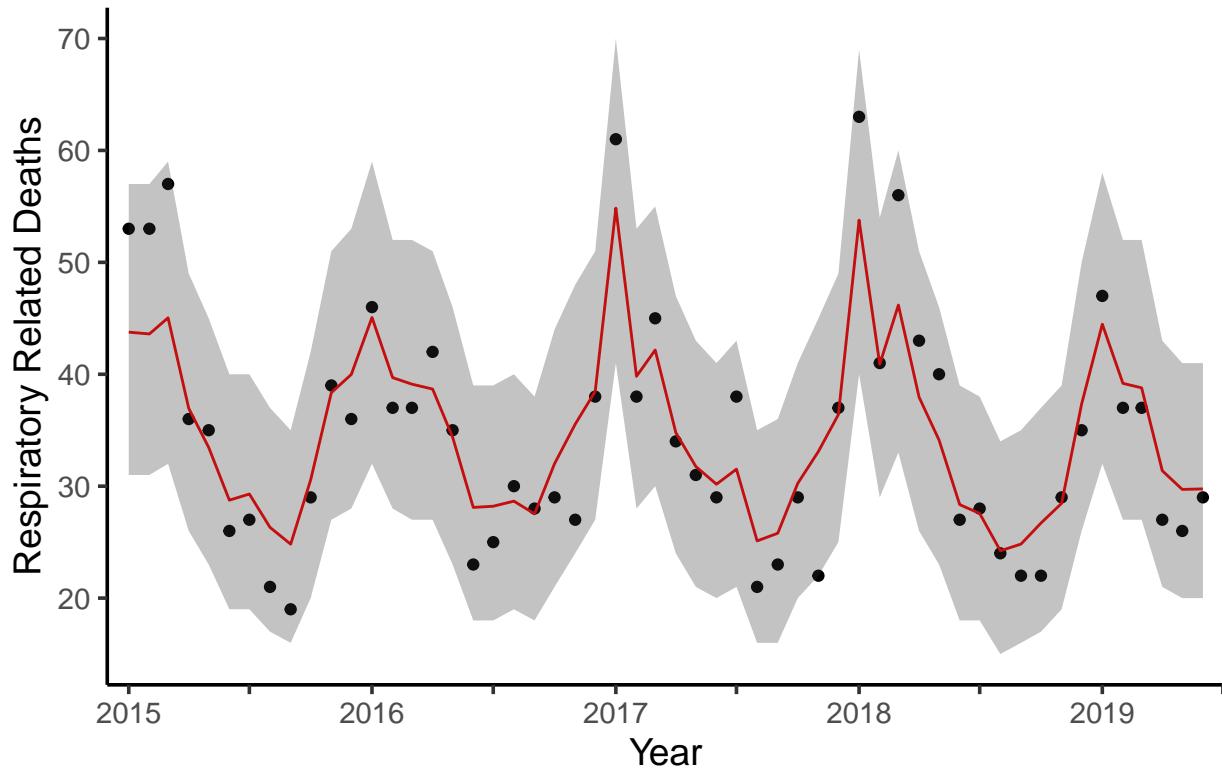
Cluster 3 (Coverage: 100%)



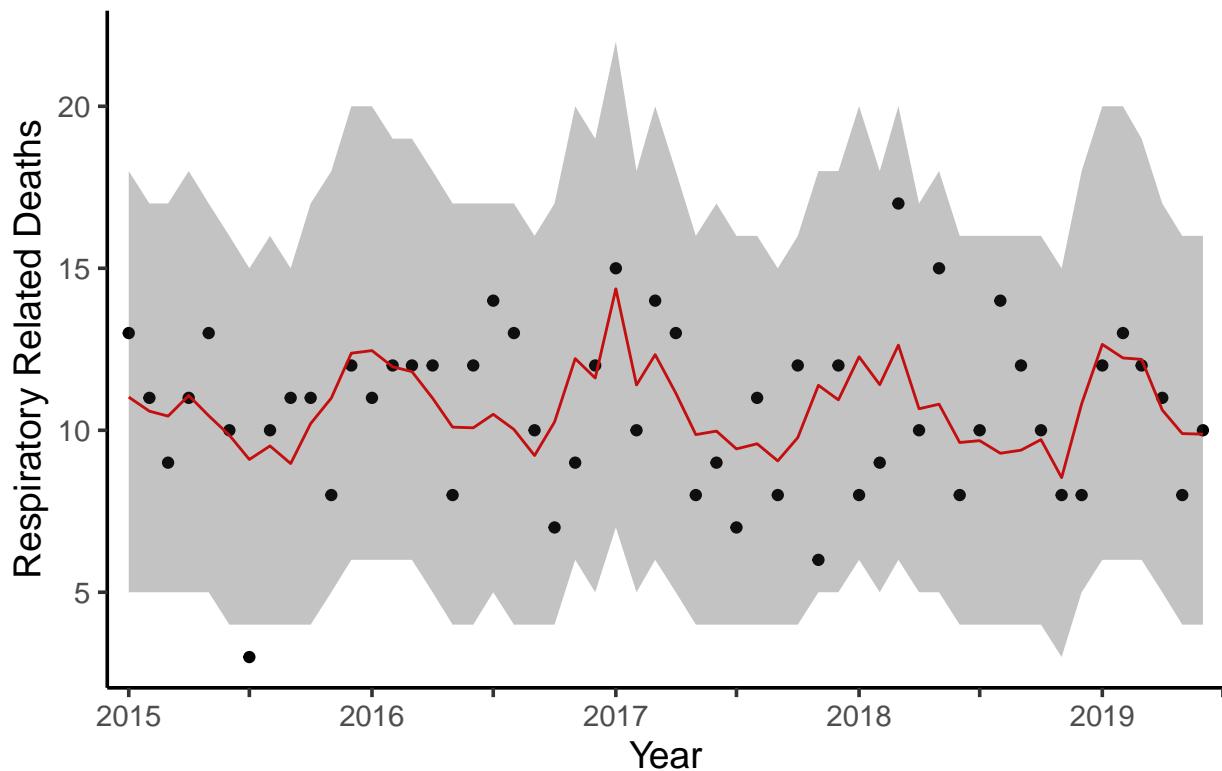
Cluster 4 (Coverage: 100%)

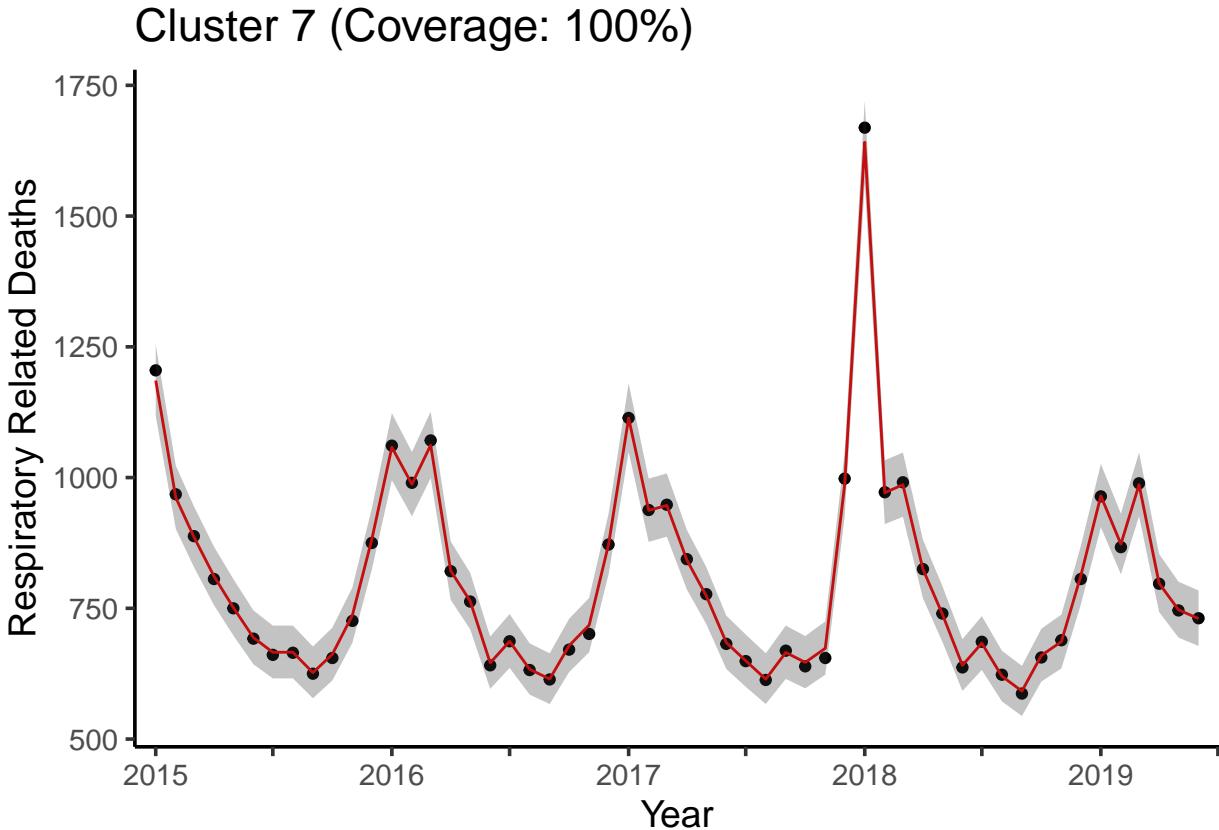


Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 98.15%)





```
pp_insample_plot_combined(pred_data = ref_model4_fit$fitted_values,prefix = "ref4-insample-")
```

KGR model with temporal kernel x graph filter (Proposed model 1)

$$\Lambda_{c,t} | \mathbf{F}, \mathbf{S} = \exp(\beta_{c1}I\{c = 1\} + \beta_{c2}I\{c = 2\} + \dots + \beta_7I\{c = 7\} + \beta_1I\{t \bmod 12 = 1\} + \dots + \beta_{11}I\{t \bmod 12 = 11\} + \mathbf{F}_{c,t})$$

where the graph signal $\mathbf{F} | \mathbf{S}, \rho_{rbf}, \rho_p, \sigma_{time}^2 \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_{time} \otimes \mathbf{H}^2)$ with $Cov(F_{c_1, t_1}, F_{c_2, t_2}) = [\mathbf{K}^{time}]_{t_1, t_2} [\mathbf{H}^2]_{c_1, c_2}$.

```
kgr_model1 = function(dataset, rho_time_rbf = 1, rho_time_periodic = 1, sigma2_time = 1, link=1){
```

```
#Calculating gram matrix K_time
K_time = time_kernel(time_span = length(unique(dataset$time)), rho_rbf = rho_time_rbf,
                      rho_periodic = rho_time_periodic, sigma2 = sigma2_time)
```

```
#Heatmap of resulting K
K_time_heatmap = matrix_heatmap(K_time, title = "K_time heatmap")
```

```
#Calculate trace norm of gram matrix
K_time_weight = norm((1/60)*K_time, type = "F")
```

```
#Calculate proposed kernel
covGP2 = kronecker(K_time/60, (H^2)/7)
```

```
#Need to ensure precision matrix is not computationally singular i.e det > 0
covGP_jittered = desingularize(covGP2, threshold = 1e-2, increment = 0.01)
covGP2 = covGP_jittered[[1]]
```

```

inv_covGP2 = solve(covGP2)

#Heatmap of resulting inv_covGP2
inv_covGP2_heatmap = matrix_heatmap2(inv_covGP2,title = "",legend_title = "Precision",prefix = "kgr1-"

####Fit INLA model
# kgr_formula1 = response ~ -1 + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
#   Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP2)

kgr_formula1 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
  Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP2)

model = inla(formula = kgr_formula1,family = "poisson",data = dataset,
             control.compute = list(dic=TRUE,waic=TRUE,
                                    return.marginals.predictor=TRUE),
             control.inla = list(strategy = "laplace"),
             control.predictor = list(compute = TRUE, link = link))

####Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hyperpar_summary <- bri.hyperpar.summary(model)
modelDIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf,Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
  scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
  xlab("linear predictor")

#Store the results in the list
kgr_model1_results = list(
  K_time_heatmap = K_time_heatmap,
  K_time_weight = K_time_weight/(K_time_weight + gfilter_weight),
  gfilter_weight = gfilter_weight/(K_time_weight + gfilter_weight),
  covmatrix = covGP2,
  prec = inv_covGP2,
  num_jitters = covGP_jittered[[2]],
  prec_heatmap = inv_covGP2_heatmap,

```

```

    model_summary = model_summary,
    bri_hyperpar_summary = bri_hyperpar_summary,
    exp_effects = multeff,
    param_plot = param_plot,
    hyperparam_plot = hyperparam_plot,
    modelDIC = modelDIC,
    modelWAIC = modelWAIC,
    fitted_values = preds_model,
    marg_fitted_values = marginal_fvs
  )

  return(kgr_model1_results)
}

#Fit kgr_model1
kgr_model1_fit = kgr_model1(dataset = inla_insample_data,rho_time_rbf = 63.888,rho_time_periodic = 5.84)

#Posterior predictive sampling from estimated intensities
kgr_model1_fit$fitted_values = poisson_pp_sampling(kgr_model1_fit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model1_DIC = kgr_model1_fit$modelDIC
kgr_model1_WAIC = kgr_model1_fit$modelWAIC

#Get summaries of parameter estimates
kgr_model1_fit$model_summary

##          mean      sd 0.025quant 0.5quant 0.975quant      mode
## months1   2.0980142 7.254728 -12.127789 2.0980142   16.32382 2.0980142
## months2   1.8695839 7.254730 -12.356222 1.8695839   16.09539 1.8695839
## months3   1.8914900 7.254730 -12.334316 1.8914900   16.11730 1.8914900
## months4   1.6970297 7.254732 -12.528780 1.6970297   15.92284 1.6970297
## months5   1.6170621 7.254733 -12.608749 1.6170621   15.84287 1.6170621
## months6   1.4933316 7.254734 -12.732483 1.4933316   15.71915 1.4933316
## months7   1.4550677 7.254741 -12.770760 1.4550677   15.68089 1.4550677
## months8   1.4333935 7.254741 -12.792434 1.4333935   15.65922 1.4333935
## months9   1.3922575 7.254742 -12.833572 1.3922575   15.61809 1.3922575
## months10  1.4728782 7.254740 -12.752948 1.4728782   15.69870 1.4728782
## months11  1.5157025 7.254740 -12.710123 1.5157025   15.74153 1.5157025
## months12  1.7749707 7.254736 -12.450848 1.7749707   16.00079 1.7749707
## Intercept1 4.2849911 7.254724 -9.940804 4.2849911   18.51079 4.2849911
## Intercept2 2.1517188 7.254743 -12.074112 2.1517188   16.37755 2.1517188
## Intercept3 2.0736399 7.254749 -12.152203 2.0736399   16.29948 2.0736399
## Intercept4 3.6474254 7.254729 -10.578379 3.6474254   17.87323 3.6474254
## Intercept5 1.8604876 7.254750 -12.365357 1.8604876   16.08633 1.8604876
## Intercept6 0.6824637 7.254833 -13.543543 0.6824637   14.90847 0.6824637
## Intercept7 5.0100552 7.254718 -9.215727 5.0100552   19.23584 5.0100552
##          kld
## months1   5.526823e-11
## months2   5.526818e-11
## months3   5.526811e-11
## months4   5.526823e-11
## months5   5.526817e-11
## months6   5.526822e-11

```

```

## months7      5.526822e-11
## months8      5.526817e-11
## months9      5.526822e-11
## months10     5.526817e-11
## months11     5.526818e-11
## months12     5.526817e-11
## Intercept1   5.526825e-11
## Intercept2   5.526817e-11
## Intercept3   5.526812e-11
## Intercept4   5.526815e-11
## Intercept5   5.526823e-11
## Intercept6   5.526811e-11
## Intercept7   5.526819e-11

kgr_model1_fit$bri_hyperpar_summary

##               mean        sd    q0.025    q0.5    q0.975      mode
## SD for id2 0.6336763 0.04940089 0.5412813 0.6319565 0.7353104 0.6286965

kgr_model1_fit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##  8.149969  6.485597  6.629239  5.457712  5.038267  4.451903  4.284774
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##  4.192904  4.023924  4.361771  4.552618  5.900108 72.601901  8.599627
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##  7.953721 38.375738  6.426870  1.978747 149.913012

kgr_model1_fit$K_time_weight

## [1] 0.5390401

kgr_model1_fit$gfilter_weight

## [1] 0.4609599

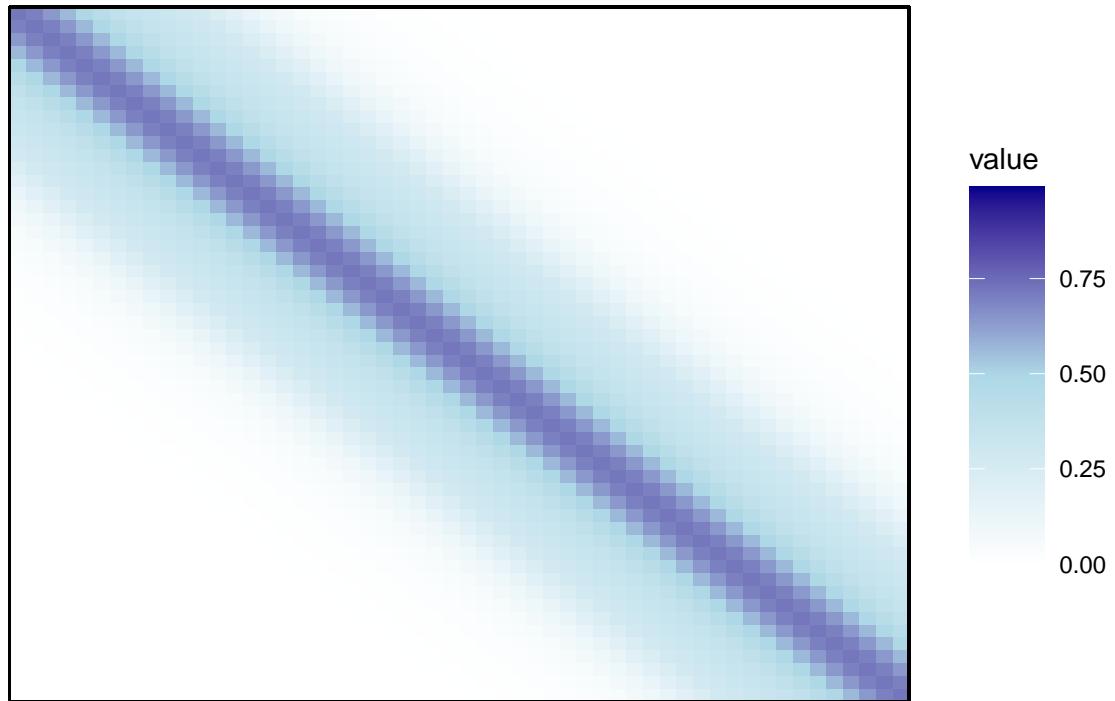
kgr_model1_fit$num_jitters

## [1] 1

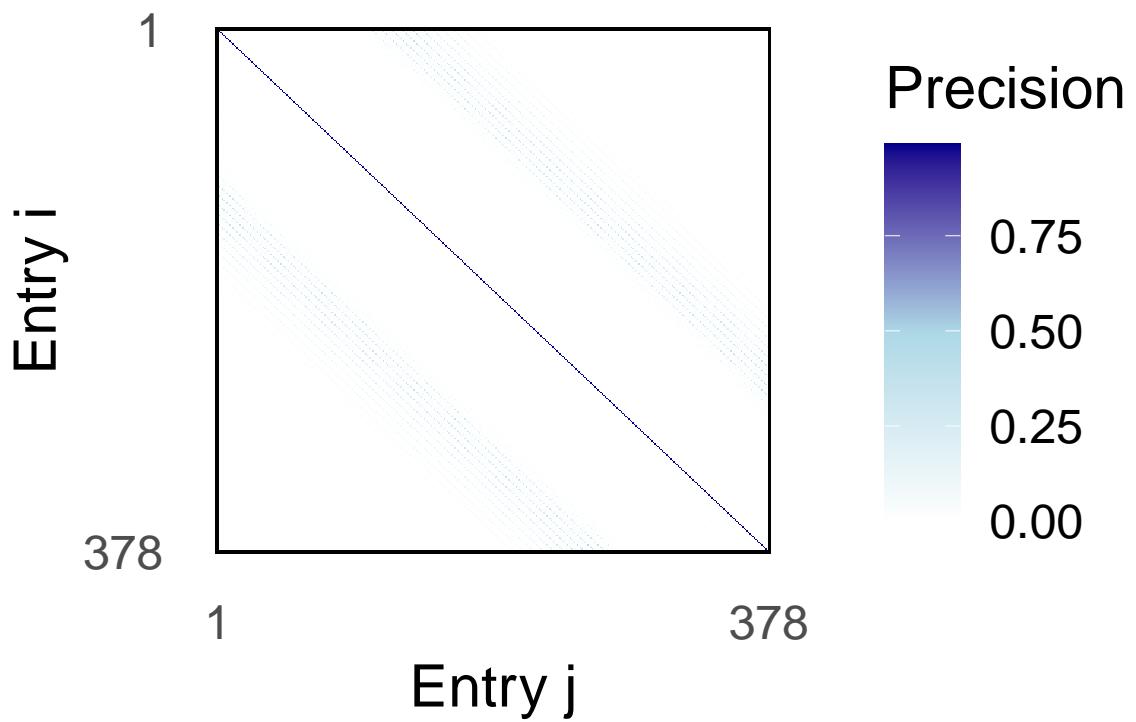
#Show plots
kgr_model1_fit$K_time_heatmap

```

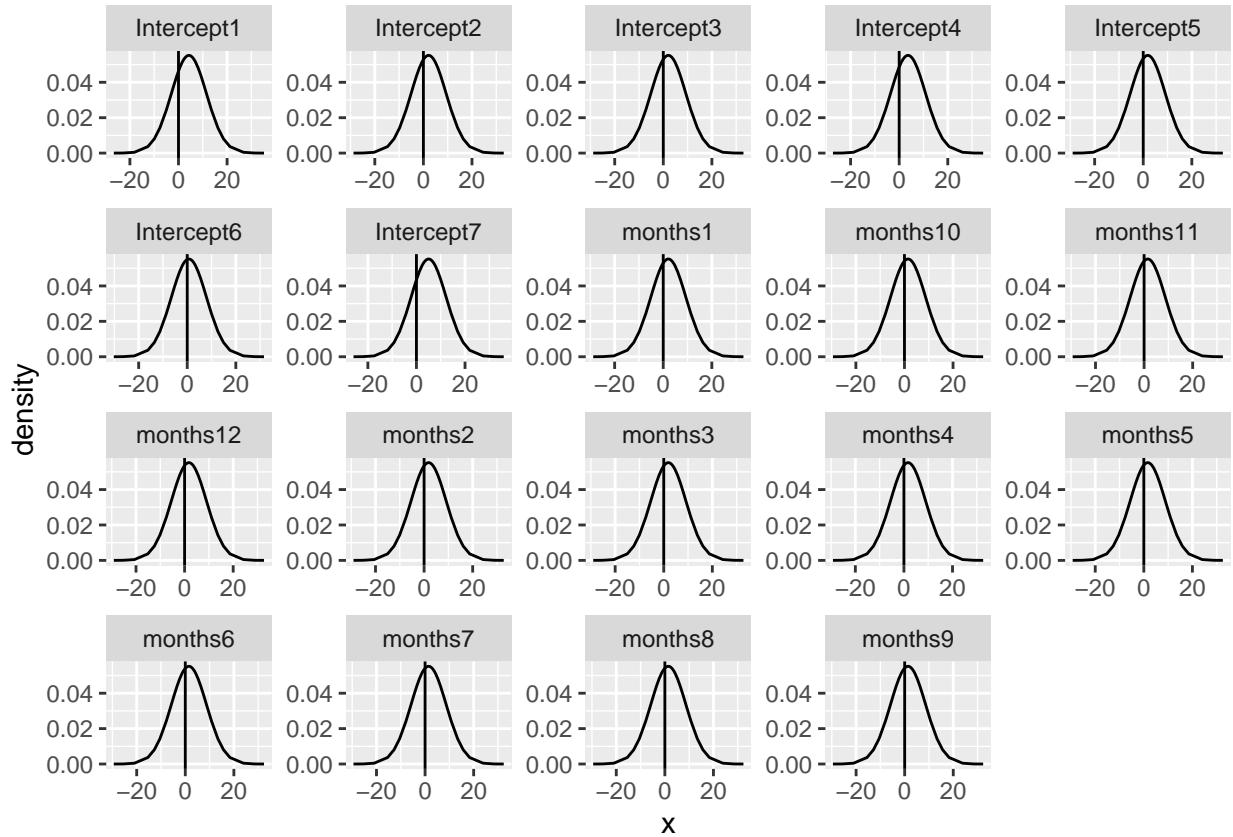
K_time heatmap



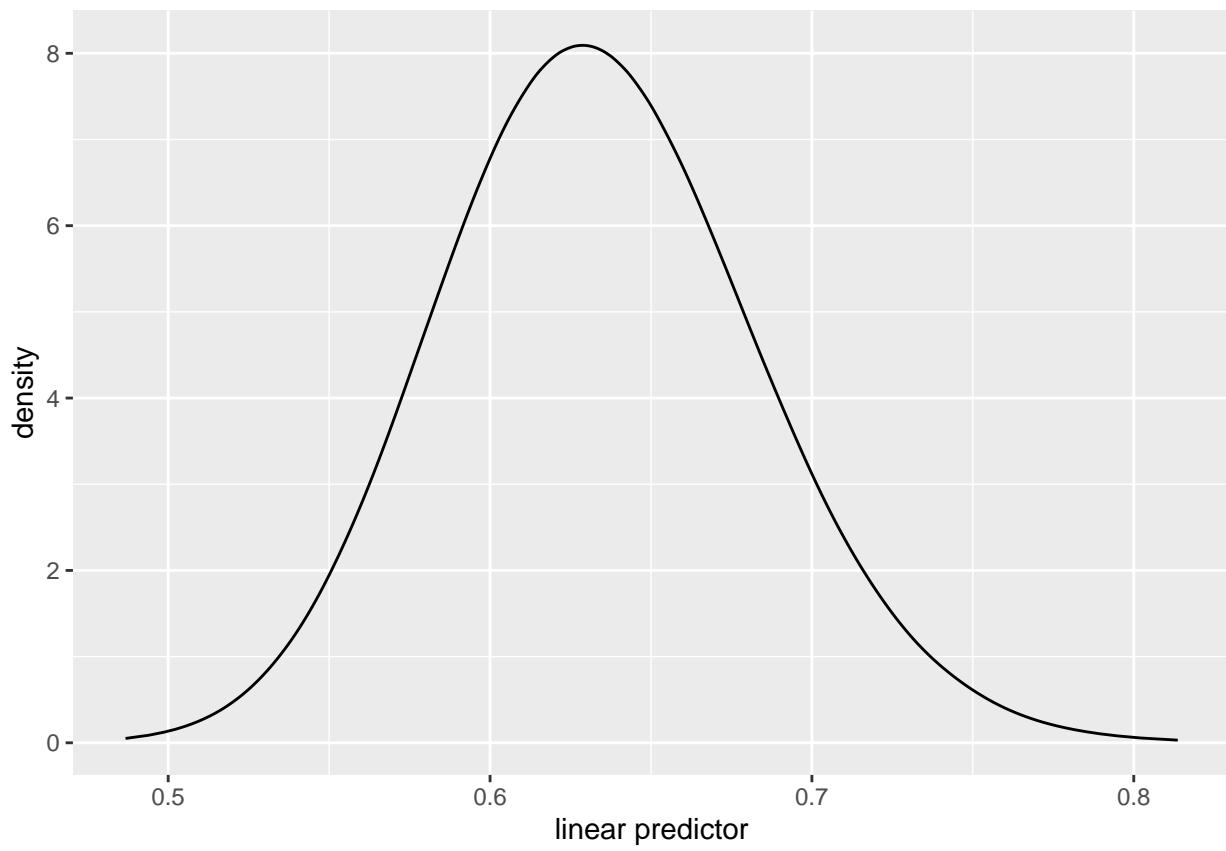
```
kgr_model1_fit$prec_heatmap
```



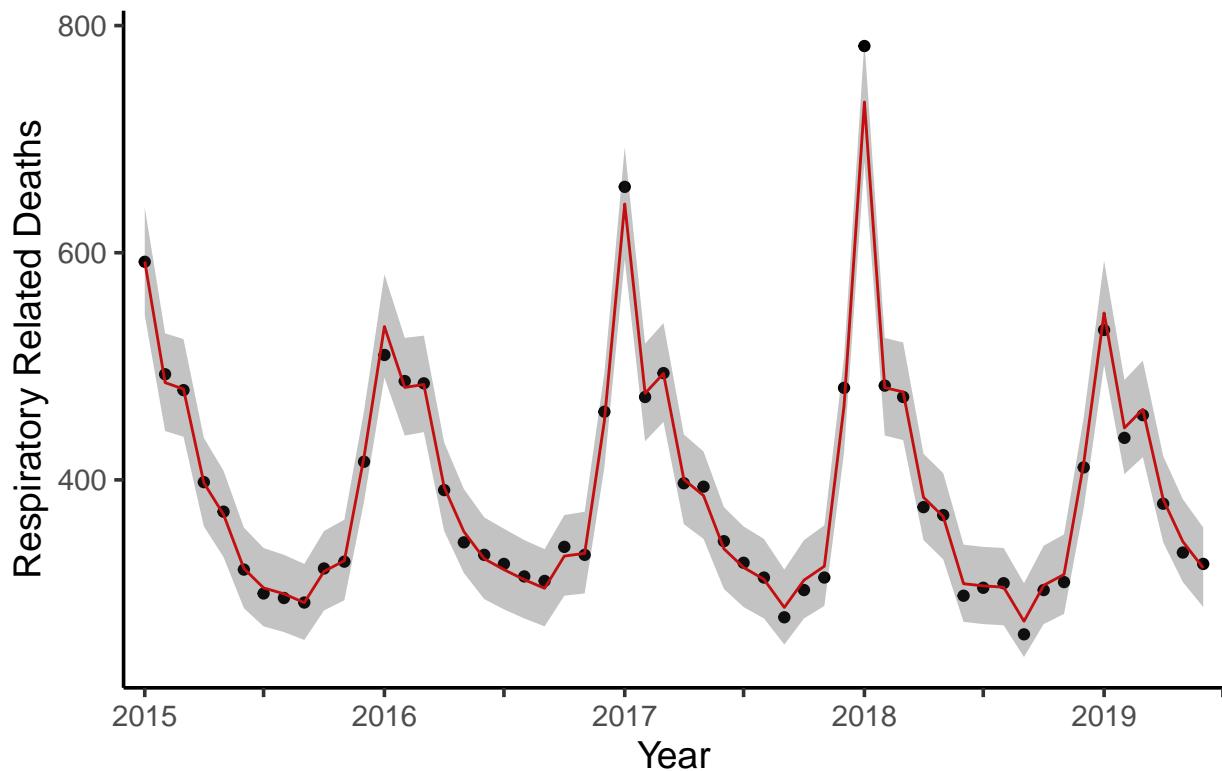
```
kgr_model1_fit$param_plot
```



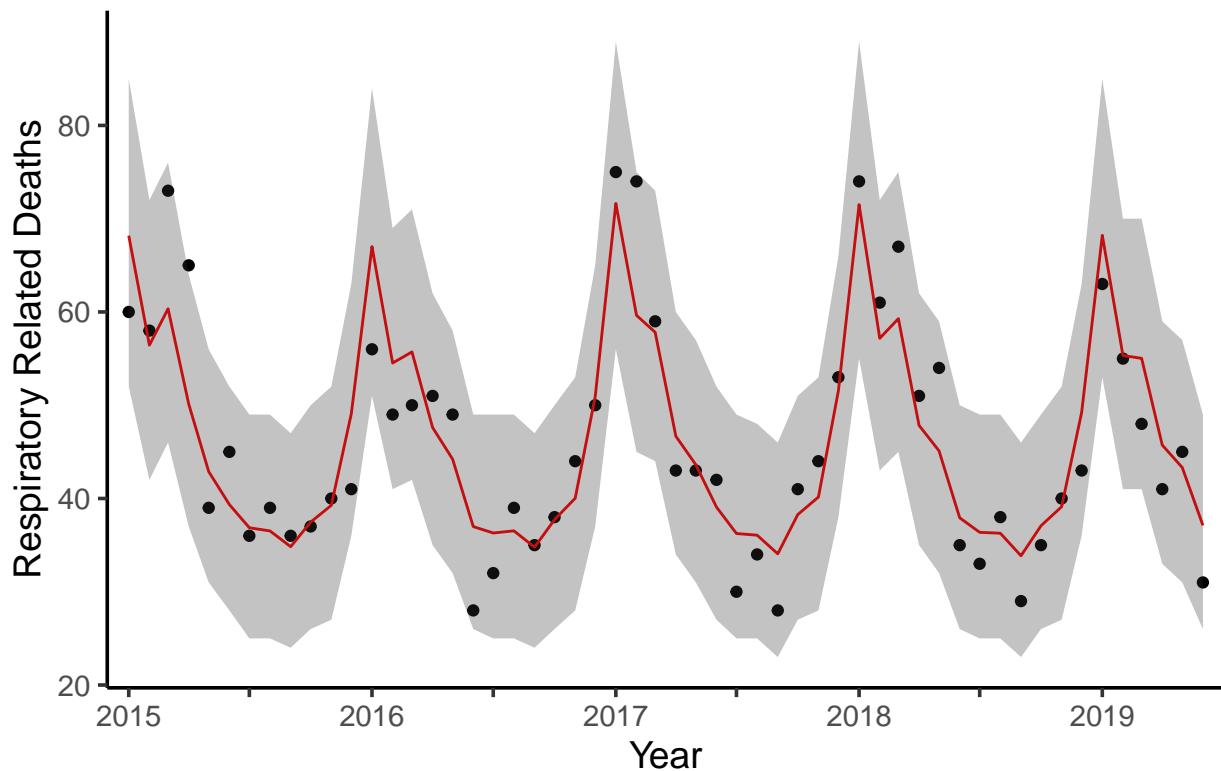
```
kgr_model1_fit$hyperparam_plot
```



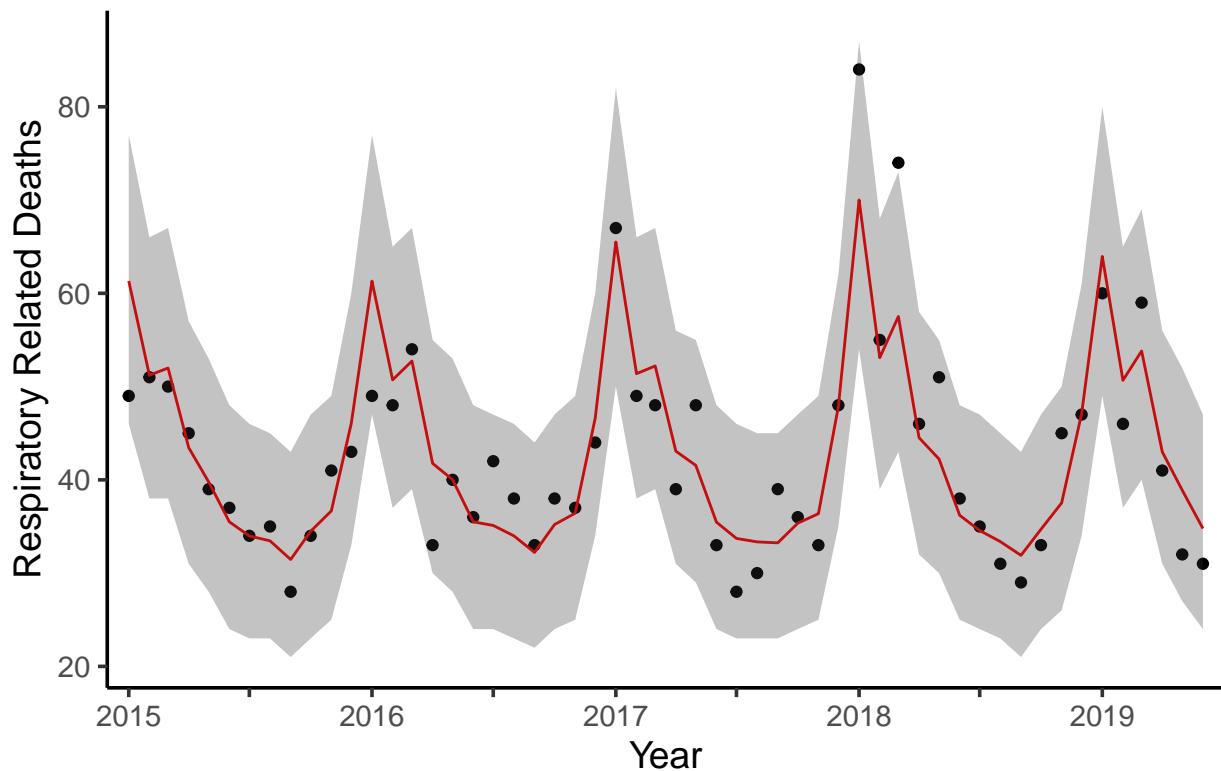
Cluster 1 (Coverage: 100%)



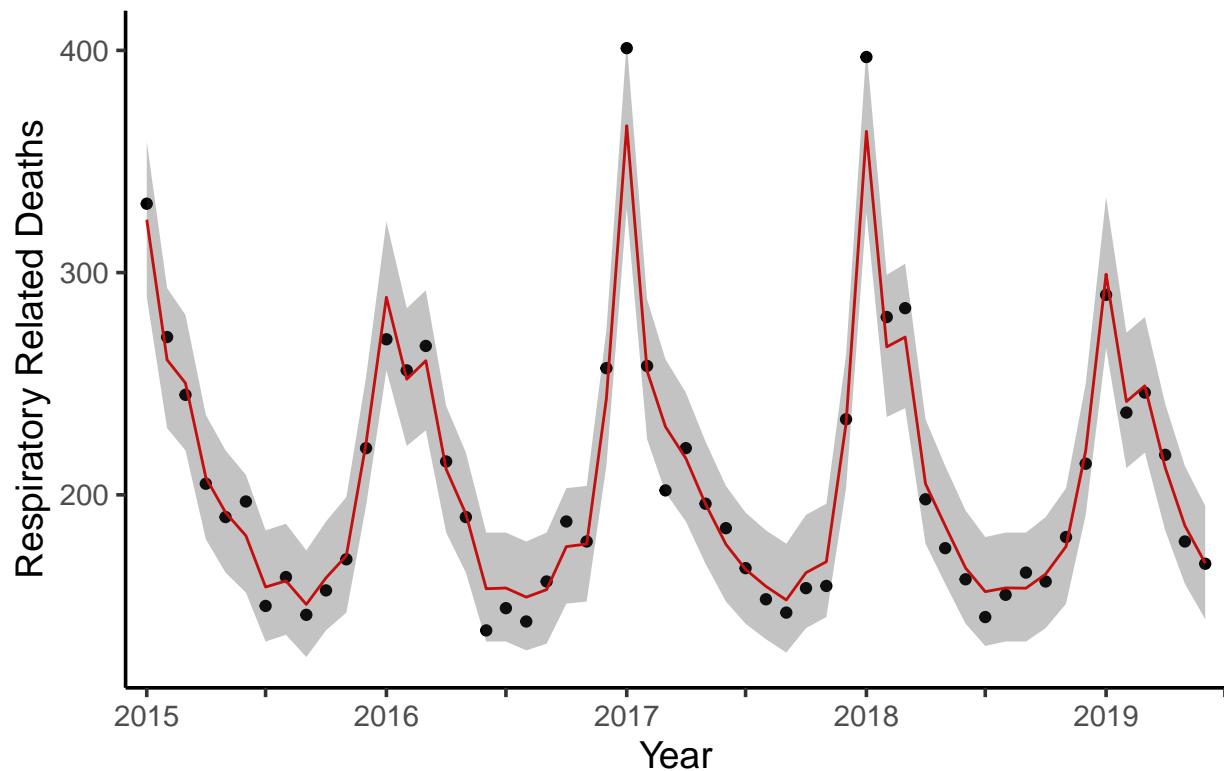
Cluster 2 (Coverage: 98.15%)



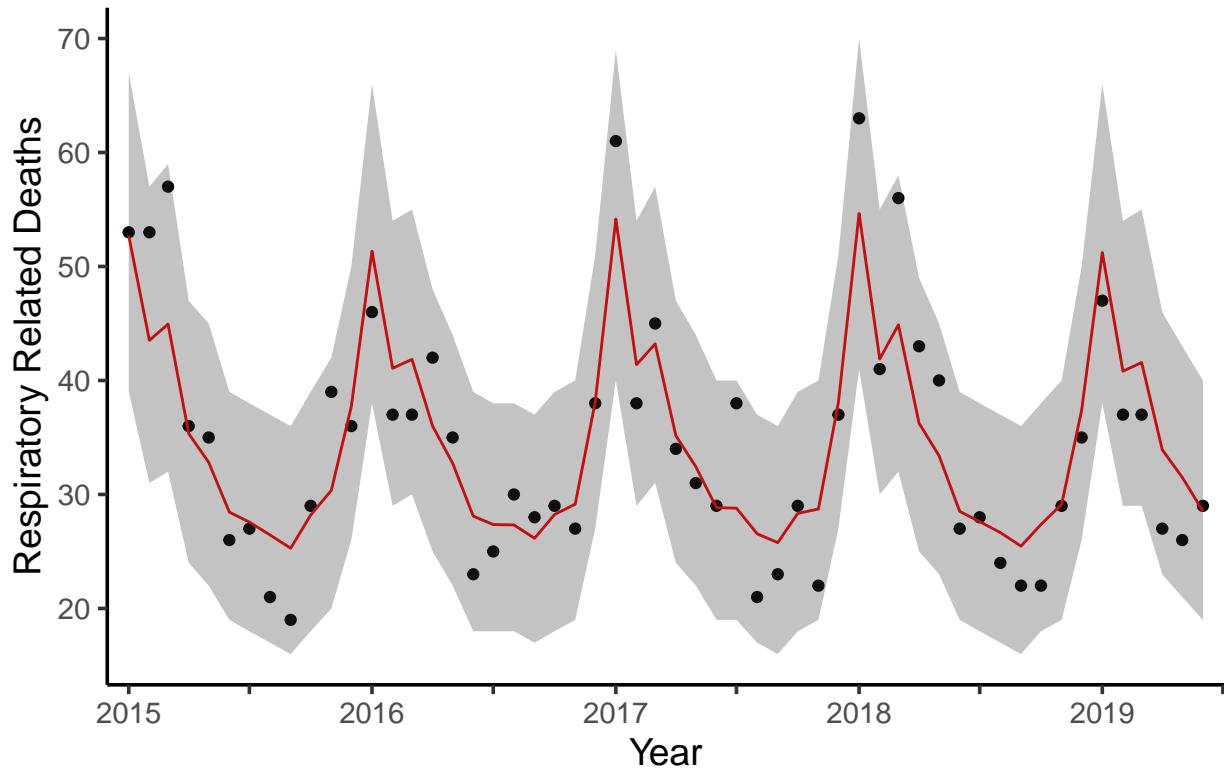
Cluster 3 (Coverage: 98.15%)



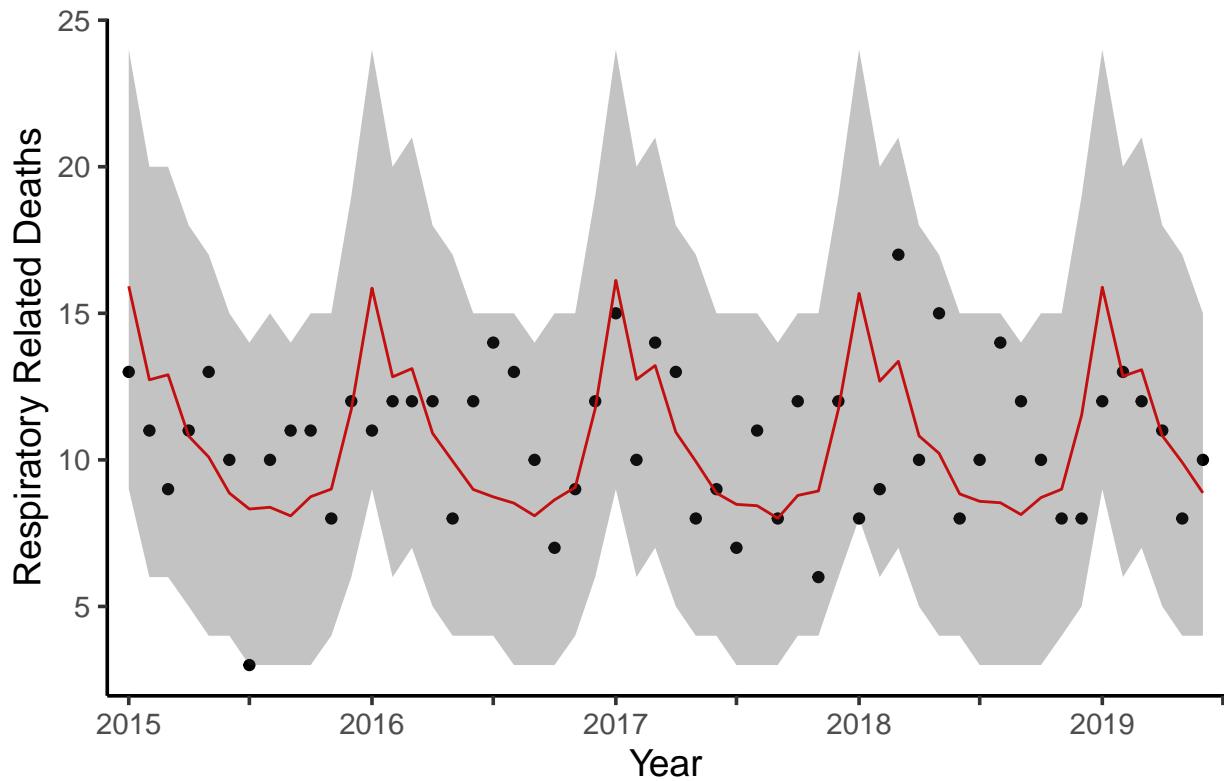
Cluster 4 (Coverage: 100%)



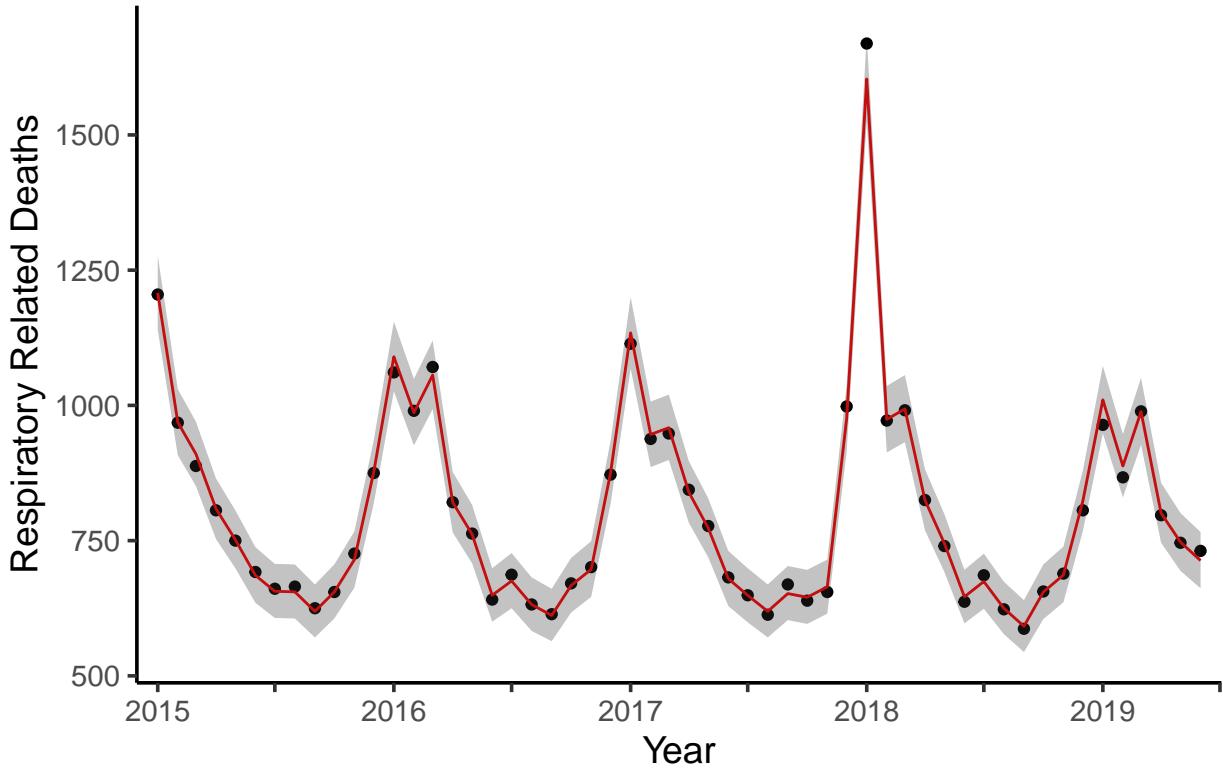
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 100%)



Cluster 7 (Coverage: 100%)



```
pp_insample_plot_combined(pred_data = kgr_model1_fit$fitted_values,prefix = "kgr1-insample-")
```

Finally, we could also increase the complexity of our proposed model by including our time kernel in the covariance structure of the underlying GP. Notice that K does not explicitly have a temporal dependence structure; instead, it represents EPA covariate similarity compared across months. We can explicitly include the time kernel above by either element wise adding or multiplying K_{EPA} and K_{time} together before taking the kronecker product with H^2

Proposed model 3

$$\Lambda_{c,t}|\mathbf{F}, \mathbf{S} = \exp(\beta_{c1}I\{c = 1\} + \beta_{c2}I\{c = 2\} + \dots + \beta_7I\{c = 7\} + \beta_1I\{t \bmod 12 = 1\} + \dots + \beta_{11}I\{t \bmod 12 = 11\} + \mathbf{F}_{c,t})$$

where the graph signal $\mathbf{F}|\mathbf{S}, \rho_{rbf}^{time}, \rho_p^{time}, \rho_{rbf}^{EPA}, \rho_p^{EPA}, \sigma^2 \sim \mathcal{GP}(\mathbf{0}, (\mathbf{K}_{time} \odot \mathbf{K}_{EPA}) \otimes \mathbf{H}^2)$ with $Cov(F_{c_1,t_1}, F_{c_2,t_2}) = [\mathbf{K}^{time}]_{t_1,t_2} [\mathbf{K}^{EPA}]_{t_1,t_2} [\mathbf{H}^2]_{c_1,c_2}$.

```
kgr_model3 = function(dataset,rho_EPA_rbf = 1,rho_EPA_periodic = 1,rho_time_rbf = 1,rho_time_periodic = 1)

    ###Calculating gram matrix K_EPA
    K_EPA = EPA_kernel(time_span = length(unique(dataset$time)),
                        rho_rbf = rho_EPA_rbf,rho_periodic = rho_EPA_periodic,sigma2 = 1)

    #Heatmap of resulting K
    K_EPA_heatmap = matrix_heatmap(K_EPA,title = "K_EPA heatmap")

    ###Calculating gram matrix K_time
    K_time = time_kernel(time_span = length(unique(dataset$time)),rho_rbf = rho_time_rbf,
```

```

rho_periodic = rho_time_periodic, sigma2 = sigma2)

#Heatmap of resulting K
K_time_heatmap = matrix_heatmap(K_time,title = "K_time heatmap")

K_EPA_norm = norm(K_EPA,type = "F")
K_time_norm = norm(K_time,type = "F")

#Calculate trace norm of gram matrix
gram = (K_EPA*K_time)/sigma2
K_weight = norm((1/60)*gram,type = "F")

###Load graph regression kernel
# covGP3 = kronecker(gram,H^2)
covGP3 = kronecker(gram/60,(H^2)/7)

#Need to ensure precision matrix is not computationally singular i.e det > 0
covGP_jittered = desingularize(covGP3,threshold = 1e-2,increment = 0.01)
covGP3 = covGP_jittered[[1]]

inv_covGP3 = solve(covGP3)

#Heatmap of resulting K
inv_covGP3_heatmap = matrix_heatmap2(inv_covGP3,title = "",legend_title = "Precision",prefix = "kgr3-p")

###Fit INLA model
# kgr_formula3 = response ~ -1 + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
#   Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP3)

kgr_formula3 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
  Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP3)

model = inla(formula = kgr_formula3,family = "poisson",data = dataset,
            control.compute = list(dic=TRUE,waic=TRUE,
                                   return.marginals.predictor=TRUE),
            control.inla = list(strategy = "gaussian"),
            control.predictor = list(compute = TRUE, link = link))

###Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hyparpar_summary <- bri.hyperpar.summary(model)
model_DIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

```

```

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf, Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
kgr_model3_results = list(
  K_EPA_heatmap = K_EPA_heatmap,
  K_time_heatmap = K_time_heatmap,
  K_EPA_weight = K_EPA_norm / (K_EPA_norm + K_time_norm),
  K_time_weight = K_time_norm / (K_EPA_norm + K_time_norm),
  K_weight = K_weight/(K_weight + gfilter_weight),
  gfilter_weight = gfilter_weight/(K_weight + gfilter_weight),
  gram_matrix = gram,
  covmatrix = covGP3,
  prec = inv_covGP3,
  num_jitters = covGP_jittered[[2]],
  prec_heatmap = inv_covGP3_heatmap,
  model_summary = model_summary,
  bri_hyperpar_summary = bri_hyperpar_summary,
  exp_effects = multeff,
  param_plot = param_plot,
  hyperparam_plot = hyperparam_plot,
  modelDIC = modelDIC,
  model_WAIC = model_WAIC,
  fitted_values = preds_model,
  marg_fitted_values = marginal_fvs
)

return(kgr_model3_results)
}

#Fit kgr_model3
kgr_model3_fit = kgr_model3(dataset = inla_insample_data, rho_EPA_rbf = 77.429, rho_EPA_periodic = 5164
                           rho_time_rbf = 6802.120, rho_time_periodic = 8977.554, sigma2 = 4.997)

#Posterior predictive sampling from estimated intensities
kgr_model3_fit$fitted_values = poisson_pp_sampling(kgr_model3_fit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model3_DIC = kgr_model3_fit$modelDIC
kgr_model3_WAIC = kgr_model3_fit$modelWAIC

#Get summaries of parameter estimates
kgr_model3_fit$model_summary

```

##	mean	sd	0.025quant	0.5quant	0.975quant	mode
----	------	----	------------	----------	------------	------

```

## months1    2.0982603 7.254730 -12.127545 2.0982603   16.32407 2.0982603
## months2    1.8695318 7.254731 -12.356277 1.8695318   16.09534 1.8695318
## months3    1.8912298 7.254731 -12.334579 1.8912298   16.11704 1.8912298
## months4    1.6963042 7.254733 -12.529508 1.6963042   15.92212 1.6963042
## months5    1.6160437 7.254734 -12.609770 1.6160437   15.84186 1.6160437
## months6    1.4917144 7.254735 -12.734102 1.4917144   15.71753 1.4917144
## months7    1.4551395 7.254742 -12.770690 1.4551395   15.68097 1.4551395
## months8    1.4338533 7.254742 -12.791977 1.4338533   15.65968 1.4338533
## months9    1.3925646 7.254743 -12.833267 1.3925646   15.61840 1.3925646
## months10   1.4734481 7.254742 -12.752381 1.4734481   15.69928 1.4734481
## months11   1.5165189 7.254741 -12.709309 1.5165189   15.74235 1.5165189
## months12   1.7759337 7.254738 -12.449887 1.7759337   16.00175 1.7759337
## Intercept1  4.2846679 7.254756 -9.941188 4.2846679   18.51052 4.2846679
## Intercept2  2.1517407 7.254759 -12.074122 2.1517407   16.37760 2.1517407
## Intercept3  2.0742004 7.254782 -12.151707 2.0742004   16.30011 2.0742004
## Intercept4  3.6470251 7.254770 -10.578859 3.6470251   17.87291 3.6470251
## Intercept5  1.8604719 7.254759 -12.365392 1.8604719   16.08634 1.8604719
## Intercept6  0.6822513 7.254877 -13.543844 0.6822513   14.90835 0.6822513
## Intercept7  5.0101850 7.254727 -9.215616 5.0101850   19.23599 5.0101850
##                               kld
## months1    5.526823e-11
## months2    5.526823e-11
## months3    5.526823e-11
## months4    5.526822e-11
## months5    5.526822e-11
## months6    5.526817e-11
## months7    5.526817e-11
## months8    5.526823e-11
## months9    5.526817e-11
## months10   5.526817e-11
## months11   5.526817e-11
## months12   5.526822e-11
## Intercept1 5.526813e-11
## Intercept2 5.526823e-11
## Intercept3 5.526812e-11
## Intercept4 5.526822e-11
## Intercept5 5.526822e-11
## Intercept6 5.526817e-11
## Intercept7 5.526822e-11
kgr_model3_fit$bri_hyperpar_summary

##               mean        sd      q0.025      q0.5      q0.975       mode
## SD for id2 0.6520801 0.05114906 0.5563814 0.6503115 0.7572779 0.6469541
kgr_model3_fit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##    8.151975   6.485259   6.627515   5.453754   5.033138   4.444709   4.285081
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##    4.194832   4.025160   4.364258   4.556336   5.905793  72.578437  8.599815
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##    7.958180  38.360380  6.426769   1.978327 149.932478
kgr_model3_fit$K_EPA_weight

```

```
## [1] 0.1699687
kgr_model3_fit$K_time_weight

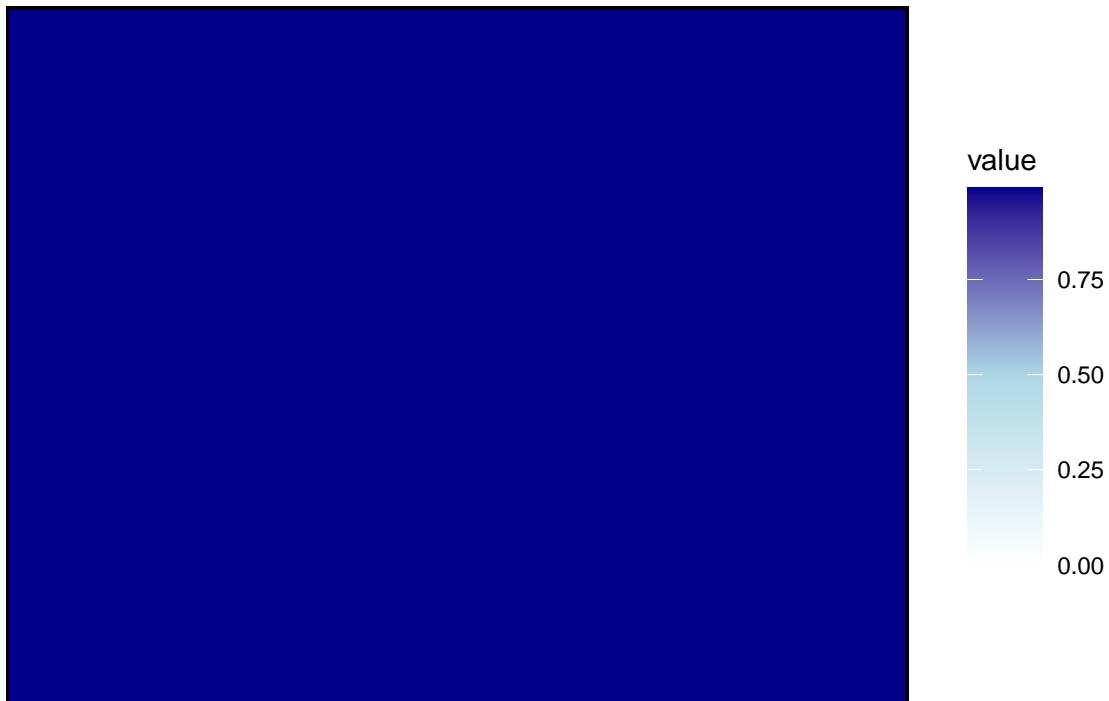
## [1] 0.8300313
kgr_model3_fit$K_weight

## [1] 0.7864507
kgr_model3_fit$gfilter_weight

## [1] 0.2135493
kgr_model3_fit$num_jitters

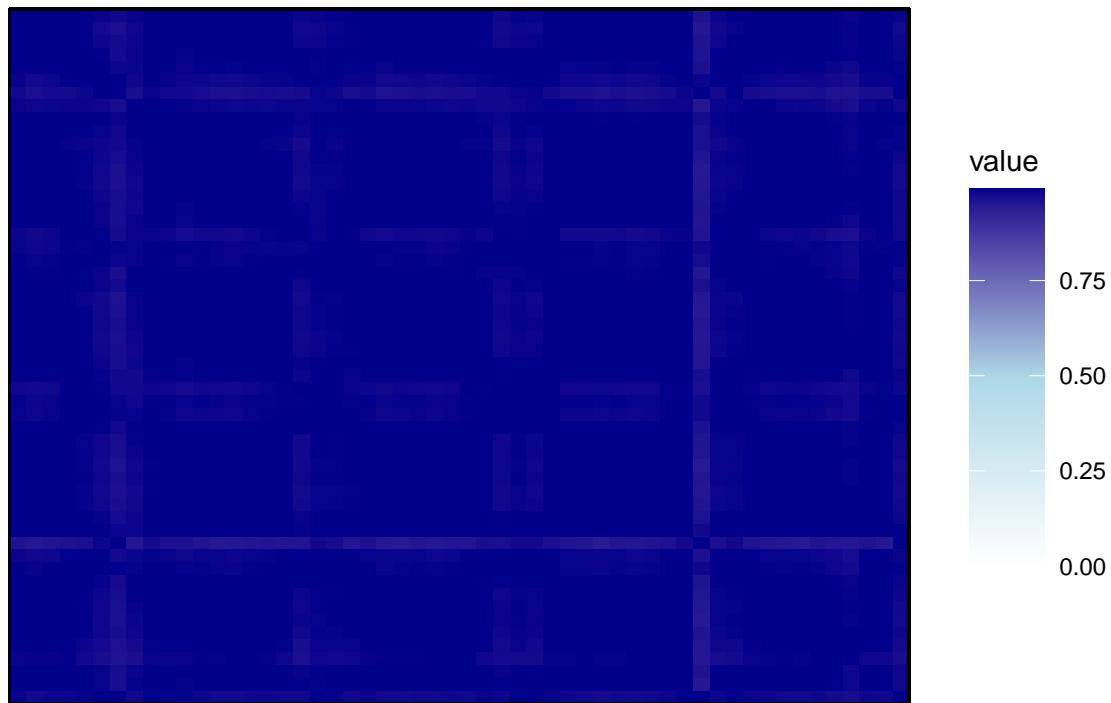
## [1] 1
#Show plots
kgr_model3_fit$K_time_heatmap
```

K_time heatmap

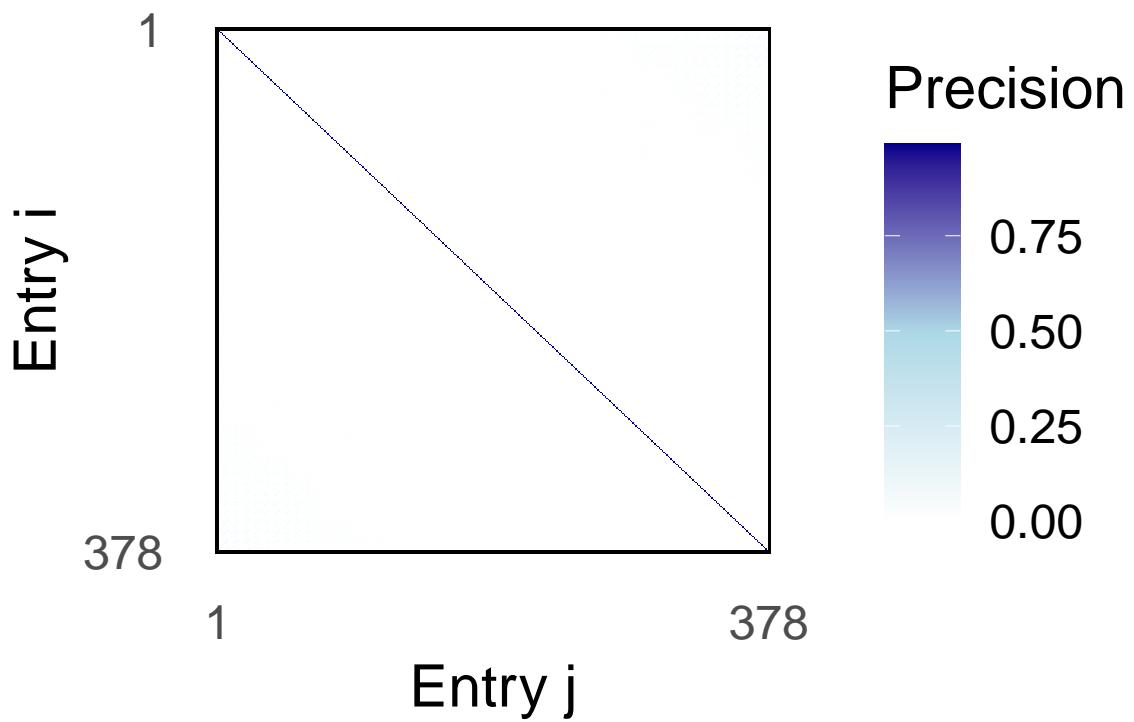


```
kgr_model3_fit$K_EPA_heatmap
```

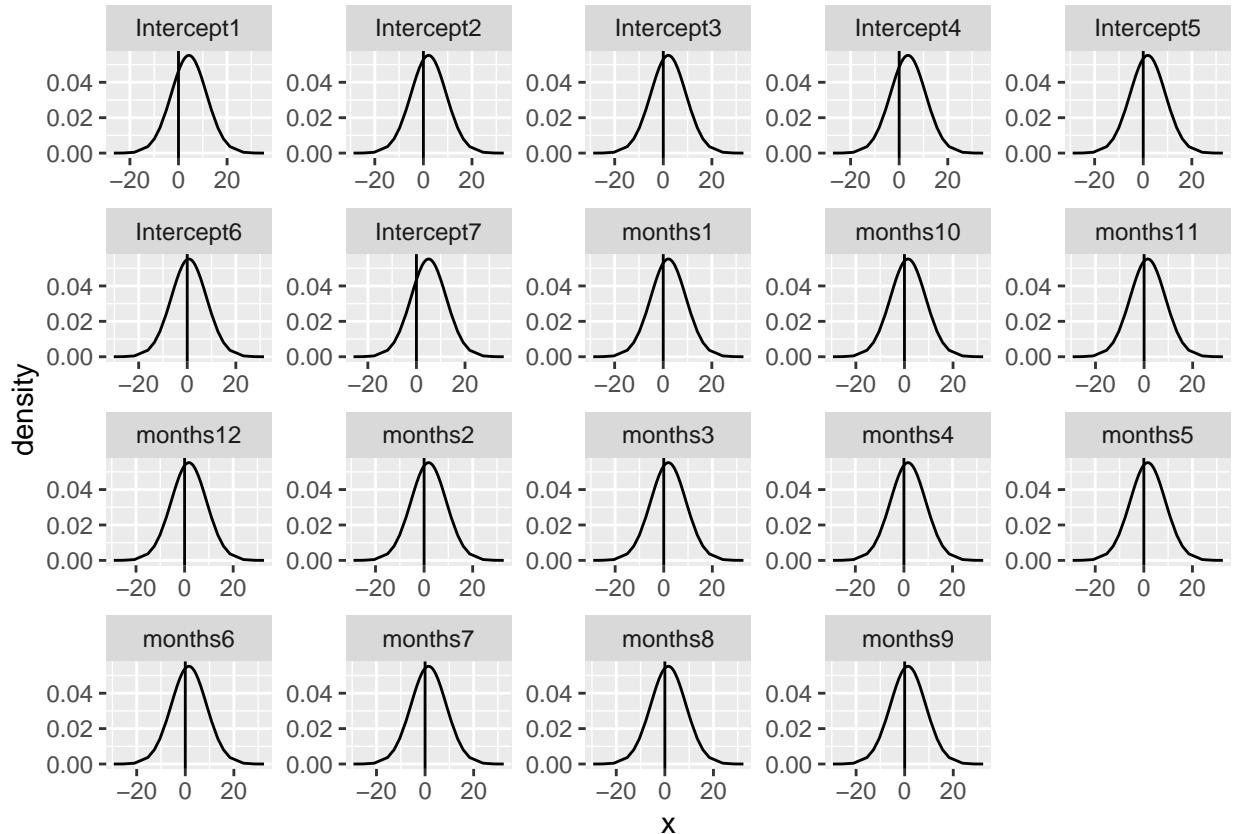
K_EPA heatmap



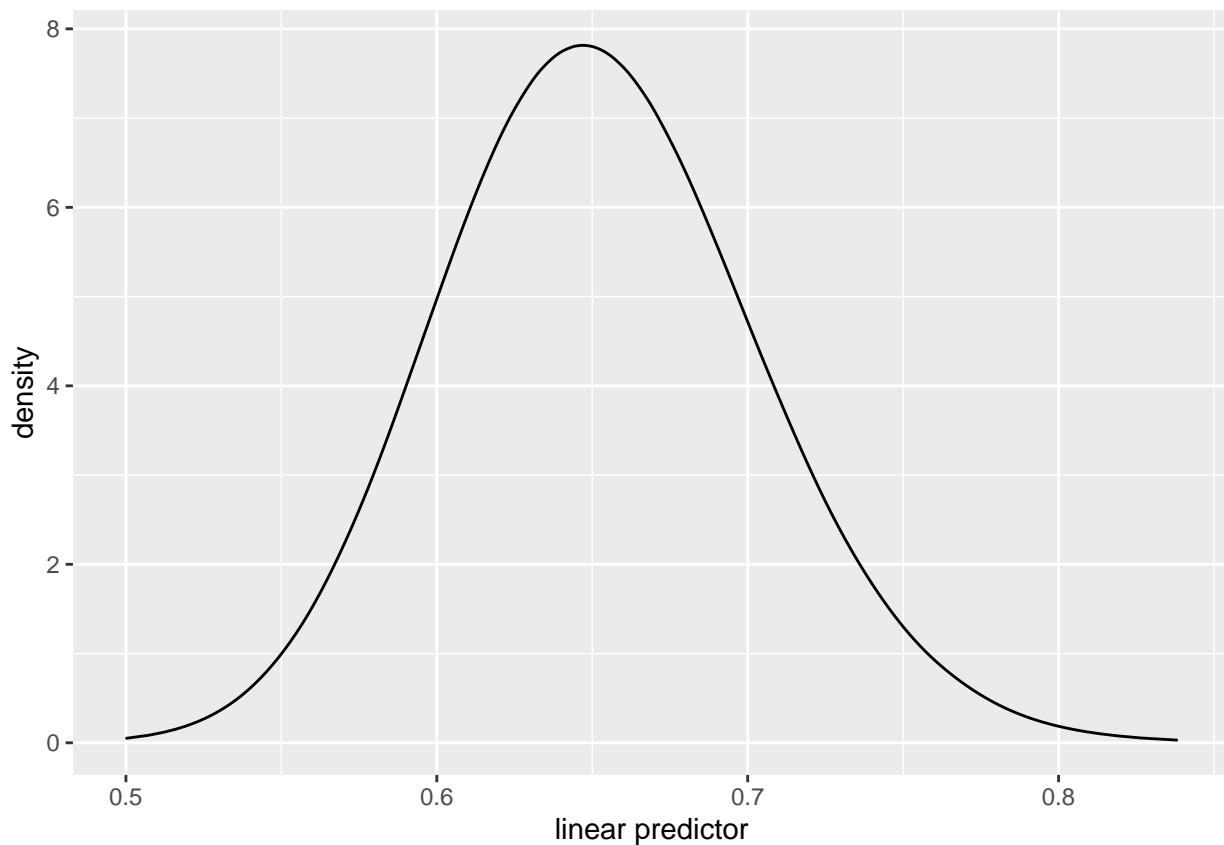
```
kgr_model3_fit$prec_heatmap
```



```
kgr_model3_fit$param_plot
```

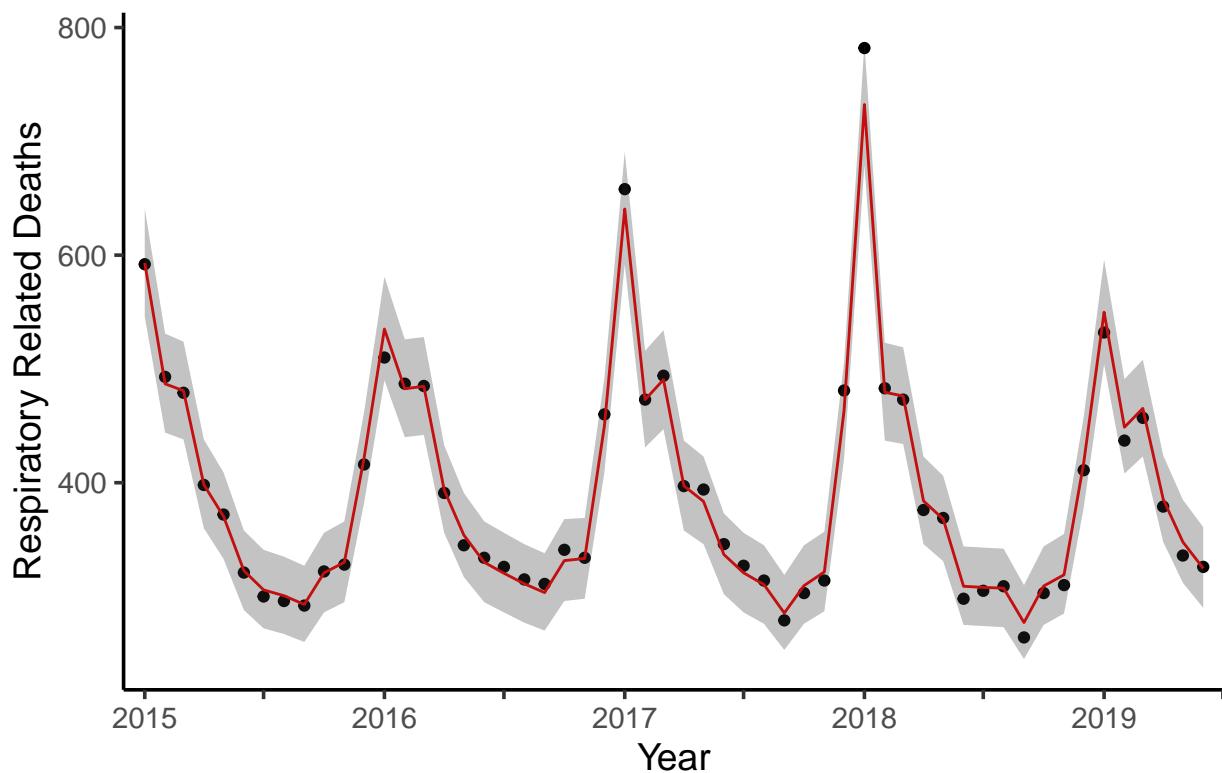


```
kgr_model3_fit$hyperparam_plot
```

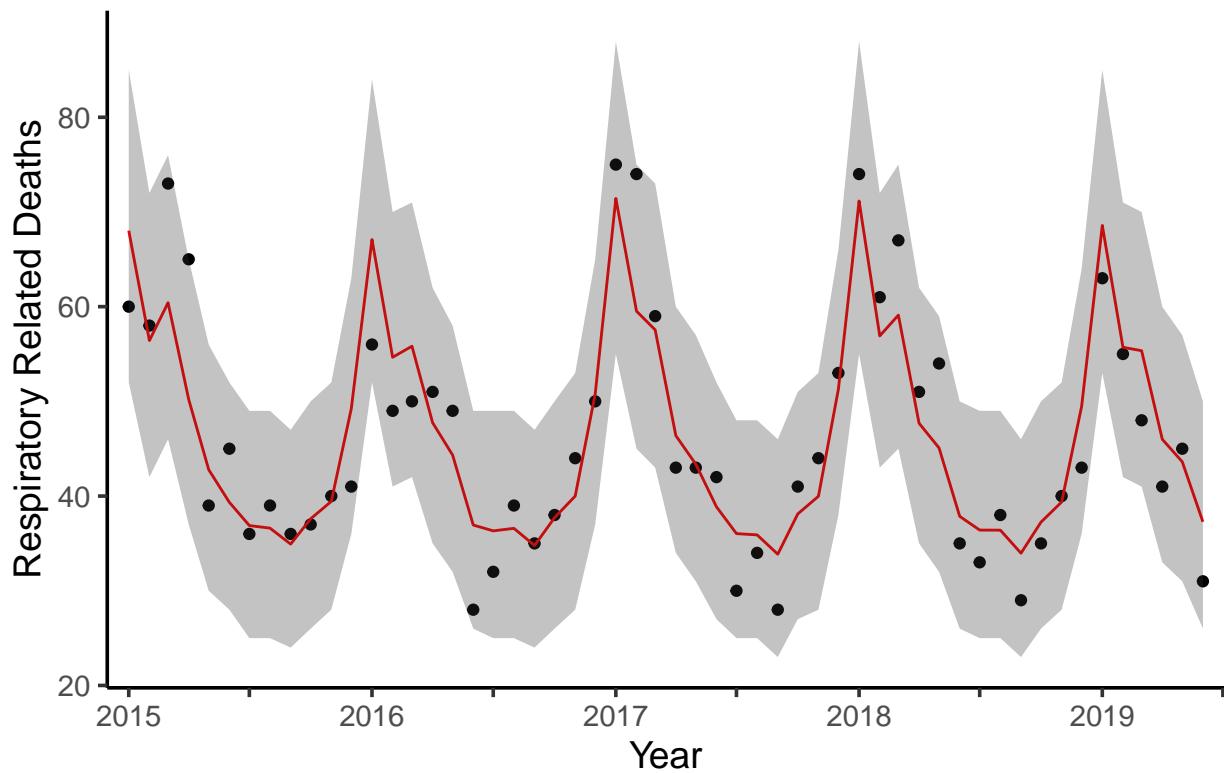


```
pp_insample_plot(pred_data = kgr_model3_fit$fitted_values,prefix = "kgr3-insample-")
```

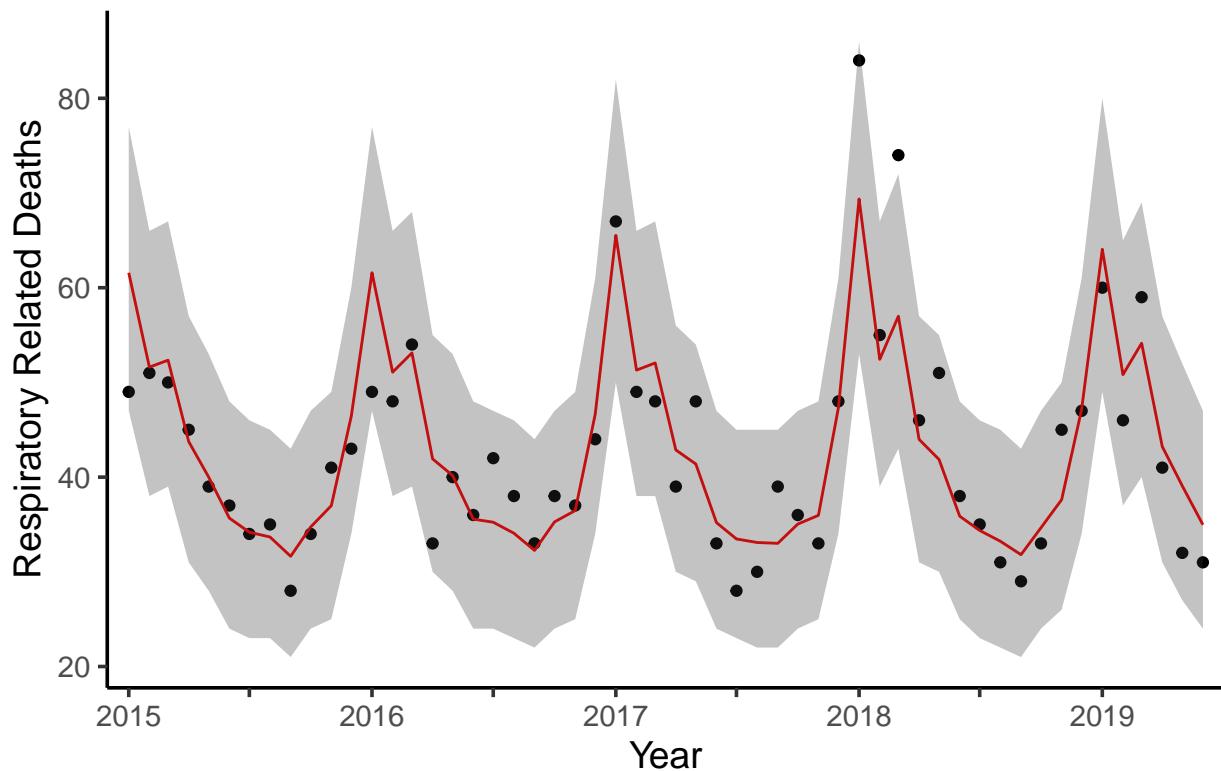
Cluster 1 (Coverage: 100%)



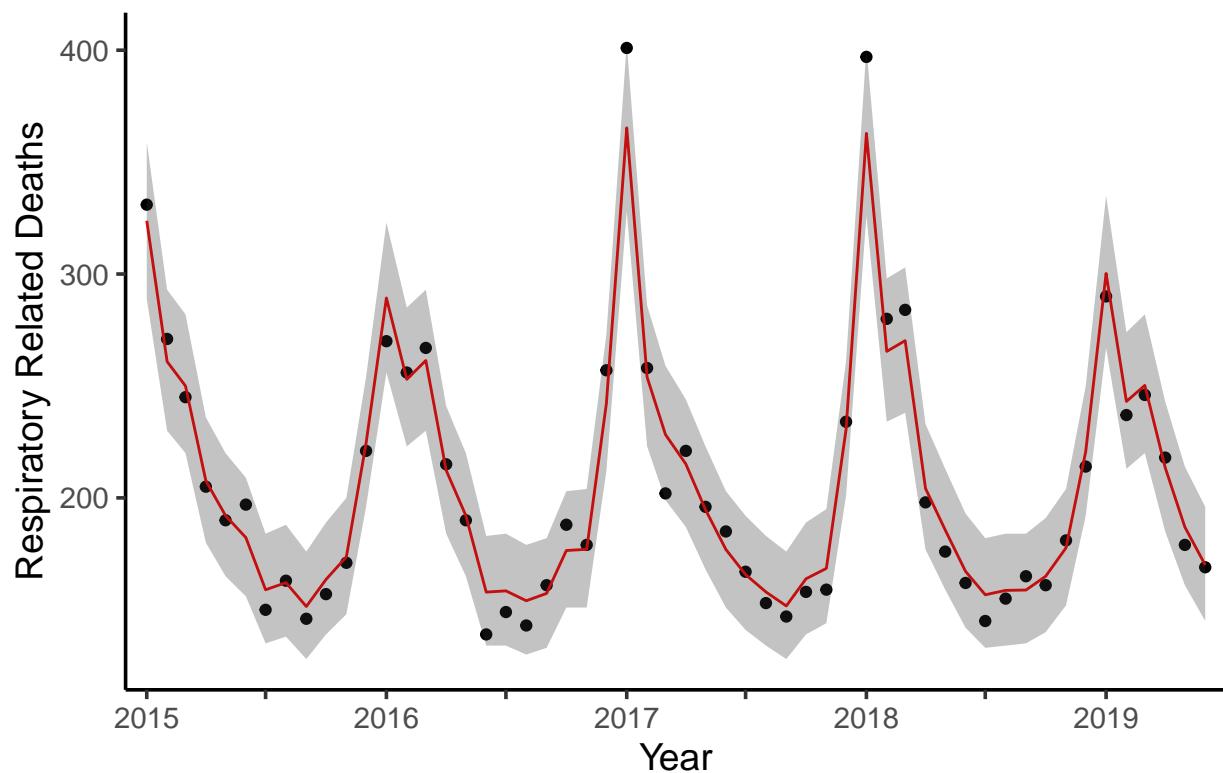
Cluster 2 (Coverage: 100%)



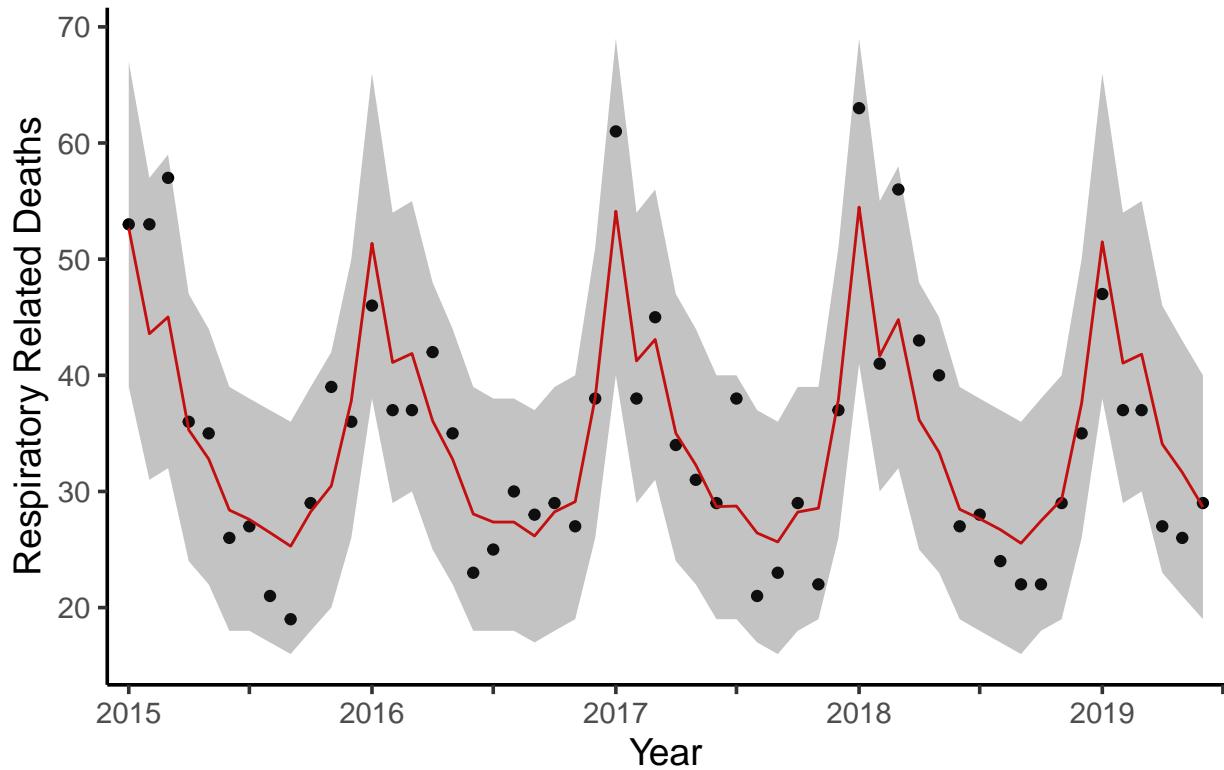
Cluster 3 (Coverage: 98.15%)



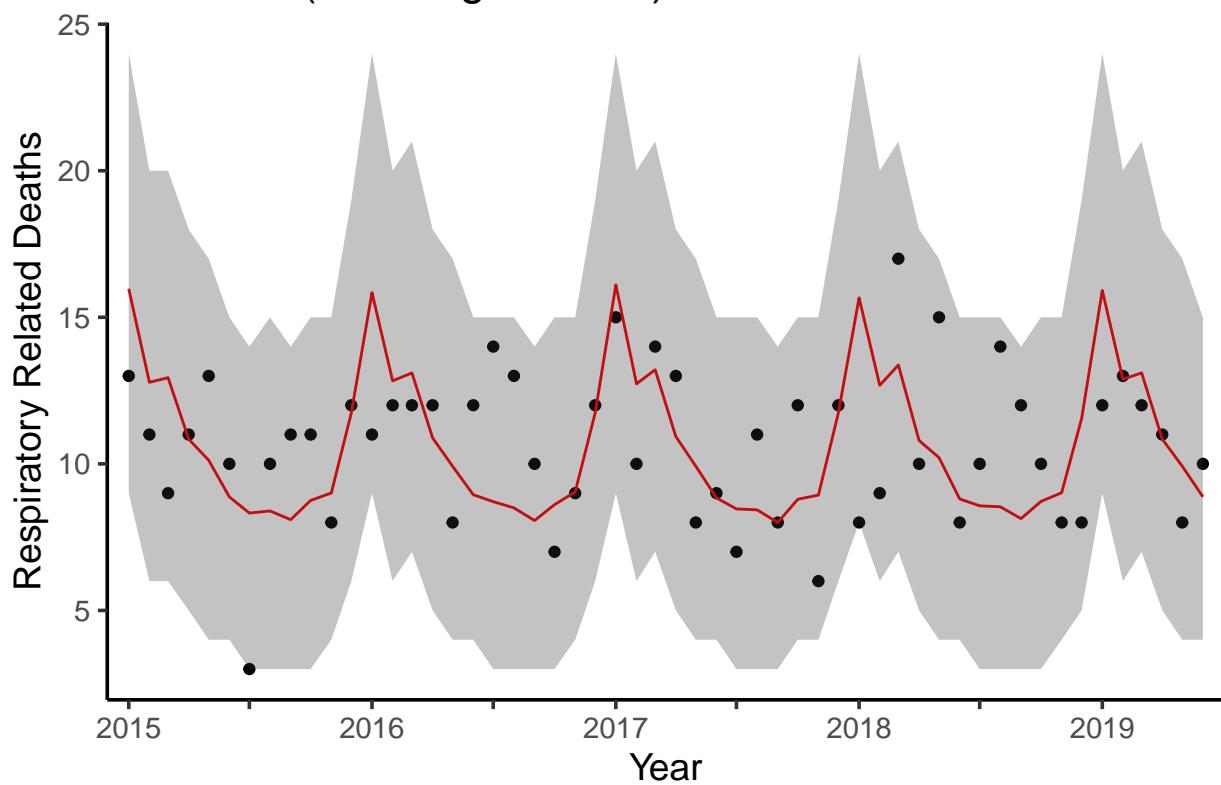
Cluster 4 (Coverage: 100%)



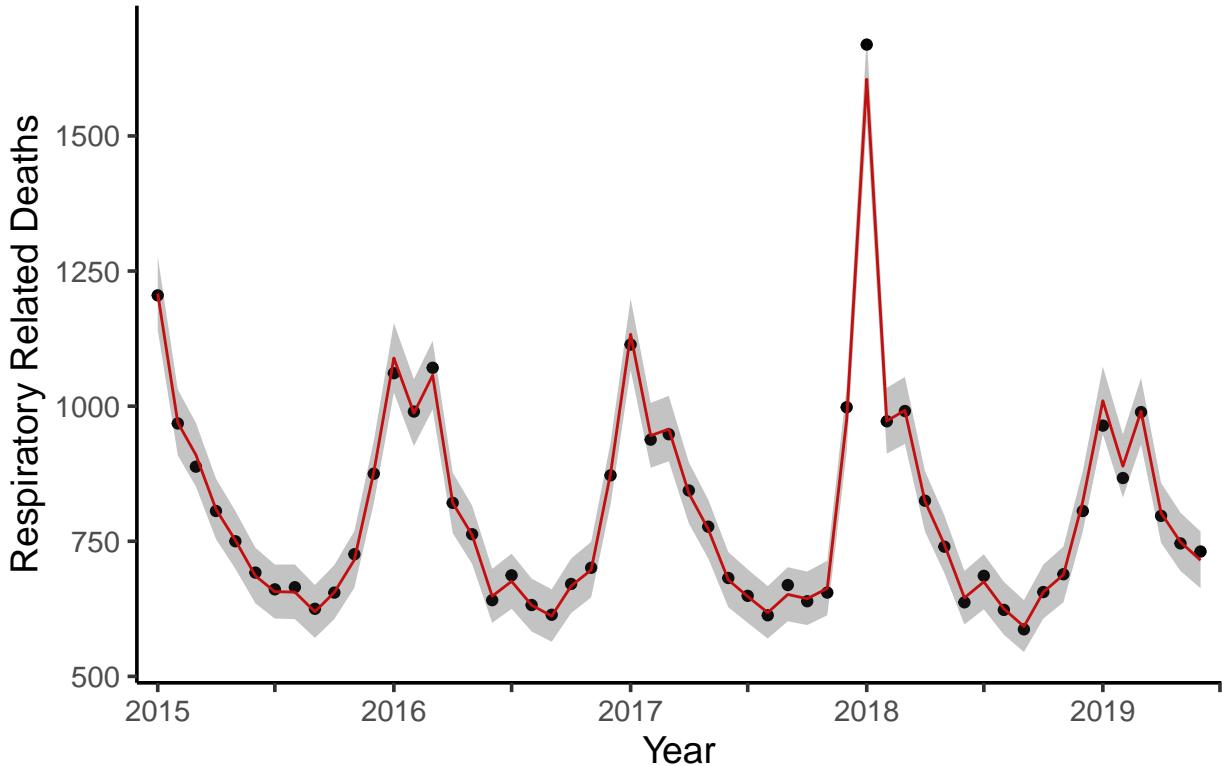
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 100%)



Cluster 7 (Coverage: 100%)



```
pp_insample_plot_combined(pred_data = kgr_model3_fit$fitted_values,prefix = "kgr3-insample-")
```

Proposed model 4

$$\Lambda_{c,t} | \mathbf{F}, \mathbf{S} = \exp(\beta_{c1} I\{c=1\} + \beta_{c2} I\{c=2\} + \dots + \beta_7 I\{c=7\} + \beta_1 I\{t \bmod 12 = 1\} + \dots + \beta_{11} I\{t \bmod 12 = 11\} + \mathbf{F}_{c,t})$$

where the graph signal $\mathbf{F} | \mathbf{S}, \rho_{rbf}^{time}, \rho_p^{time}, \rho_{rbf}^{EPA}, \rho_p^{EPA}, \sigma_{time}^2, \sigma_{EPA}^2 \sim \mathcal{GP}(\mathbf{0}, (\frac{1}{2}(\mathbf{K}_{time} + \mathbf{K}_{EPA}) \otimes \mathbf{H}^2))$ with $Cov(F_{c_1, t_1}, F_{c_2, t_2}) = \frac{1}{2} \left([\mathbf{K}^{time}]_{t_1, t_2} + [\mathbf{K}^{EPA}]_{t_1, t_2} \right) [\mathbf{H}^2]_{c_1, c_2}$.

```
kgr_model4 = function(dataset,rho_EPA_rbf = 1, rho_EPA_periodic = 1,
                      rho_time_rbf = 1, rho_time_periodic = 1, sigma2_EPA = 1, sigma2_time = 3)

###Calculating gram matrix K_EPA
K_EPA = EPA_kernel(time_span = length(unique(dataset$time)),
                     rho_rbf = rho_EPA_rbf,rho_periodic = rho_EPA_periodic,sigma2 = 1)

#Heatmap of resulting K
K_EPA_heatmap = matrix_heatmap(K_EPA,title = "K_EPA heatmap")

###Calculating gram matrix K_time
K_time = time_kernel(time_span = length(unique(dataset$time)),rho_rbf = rho_time_rbf,
                      rho_periodic = rho_time_periodic, sigma2 = sigma2_time)

#Heatmap of resulting K
K_time_heatmap = matrix_heatmap(K_time,title = "K_time heatmap")
```

```

K_EPA_norm = norm(K_EPA,type = "F")
K_time_norm = norm(K_time,type = "F")

gram = 0.5*(K_time+K_EPA)
K_weight = norm((1/60)*gram,type = "F")

###Load graph regression kernel
# covGP4 = kronecker(gram,(H^2))
covGP4 = kronecker(gram/60,(H^2)/7)

#Need to ensure precision matrix is not computationally singular i.e det > 0
covGP_jittered = desingularize(covGP4,threshold = 1e-2,increment = 0.01)
covGP4 = covGP_jittered[[1]]

inv_covGP4 = solve(covGP4)

#Heatmap of resulting K
# inv_covGP4_heatmap = matrix_heatmap(inv_covGP4,title = "")
inv_covGP4_heatmap = matrix_heatmap2(inv_covGP4,title = "",legend_title = "Precision",prefix = "kgr4-")

###Fit INLA model
# kgr_formula4 = response ~ -1 + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
#   Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP4)

kgr_formula4 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
  Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP4)

model = inla(formula = kgr_formula4,family = "poisson",data = dataset,
             control.compute = list(dic=TRUE,waic=TRUE,
                                    return.marginals.predictor=TRUE),
             control.inla = list(strategy = "laplace"),
             control.predictor = list(compute = TRUE, link = link))

###Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hypersummary <- bri.hyperpar.summary(model)
model_DIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf,Var2,value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf,aes(x=x,y=y)) + geom_line() + facet_wrap(~ parameter,

```

```

    scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$ marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden,aes(x,y)) + geom_line() + ylab("density") +
  xlab("linear predictor")

#Store the results in the list
kgr_model4_results = list(
  K_EPA_heatmap = K_EPA_heatmap,
  K_time_heatmap = K_time_heatmap,
  K_EPA_weight = K_EPA_norm / (K_EPA_norm + K_time_norm),
  K_time_weight = K_time_norm / (K_EPA_norm + K_time_norm),
  K_weight = K_weight/(K_weight + gfilter_weight),
  gfilter_weight = gfilter_weight/(K_weight + gfilter_weight),
  gram_matrix = gram,
  covmatrix = covGP4,
  prec = inv_covGP4,
  num_jitters = covGP_jittered[[2]],
  prec_heatmap = inv_covGP4_heatmap,
  model_summary = model_summary,
  bri_hyperpar_summary = bri_hyperpar_summary,
  exp_effects = multeff,
  param_plot = param_plot,
  hyperparam_plot = hyperparam_plot,
  model_DIC = model_DIC,
  model_WAIC = model_WAIC,
  fitted_values = preds_model,
  marg_fitted_values = marginal_fvs
)
}

return(kgr_model4_results)
}

#Fit kgr_model4
kgr_model4_fit = kgr_model4(dataset = inla_insample_data, rho_EPA_rbf = 259.326, rho_EPA_periodic = 246
                           rho_time_rbf = 2.699, rho_time_periodic = 86.005, sigma2_EPA = 3.215, sigma2_time = 1.8678684)

#Posterior predictive sampling from estimated intensities
kgr_model4_fit$fitted_values = poisson_pp_sampling(kgr_model4_fit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model4_DIC = kgr_model4_fit$model_DIC
kgr_model4_WAIC = kgr_model4_fit$model_WAIC

#Get summaries of parameter estimates
kgr_model4_fit$model_summary

##               mean        sd 0.025quant 0.5quant 0.975quant      mode
## months1     2.0962549 7.254729 -12.129549 2.0962549   16.32206 2.0962549
## months2     1.8678684 7.254731 -12.357939 1.8678684   16.09368 1.8678684
## months3     1.8905178 7.254731 -12.335289 1.8905178   16.11632 1.8905178
## months4     1.6974479 7.254732 -12.528363 1.6974479   15.92326 1.6974479
## months5     1.6173799 7.254733 -12.608433 1.6173799   15.84319 1.6173799

```

```

## months6    1.4938692 7.254735 -12.731946 1.4938692 15.71968 1.4938692
## months7    1.4578806 7.254741 -12.767947 1.4578806 15.68371 1.4578806
## months8    1.4355938 7.254742 -12.790235 1.4355938 15.66142 1.4355938
## months9    1.3948557 7.254742 -12.830974 1.3948557 15.62069 1.3948557
## months10   1.4742742 7.254741 -12.751553 1.4742742 15.70010 1.4742742
## months11   1.5158388 7.254740 -12.709987 1.5158388 15.74166 1.5158388
## months12   1.7741877 7.254737 -12.451632 1.7741877 16.00001 1.7741877
## Intercept1  4.2865207 7.254738 -9.939300 4.2865207 18.51234 4.2865207
## Intercept2  2.1518843 7.254749 -12.073960 2.1518843 16.37773 2.1518843
## Intercept3  2.0750270 7.254763 -12.150844 2.0750270 16.30090 2.0750270
## Intercept4  3.6470703 7.254747 -10.578769 3.6470703 17.87291 3.6470703
## Intercept5  1.8609588 7.254753 -12.364893 1.8609588 16.08681 1.8609588
## Intercept6  0.6836237 7.254852 -13.542422 0.6836237 14.90967 0.6836237
## Intercept7  5.0108841 7.254721 -9.214905 5.0108841 19.23667 5.0108841
##                               kld
## months1    5.526823e-11
## months2    5.526822e-11
## months3    5.526817e-11
## months4    5.526823e-11
## months5    5.526823e-11
## months6    5.526823e-11
## months7    5.526817e-11
## months8    5.526818e-11
## months9    5.526822e-11
## months10   5.526817e-11
## months11   5.526817e-11
## months12   5.526817e-11
## Intercept1 5.526815e-11
## Intercept2 5.526823e-11
## Intercept3 5.526822e-11
## Intercept4 5.526822e-11
## Intercept5 5.526817e-11
## Intercept6 5.526817e-11
## Intercept7 5.526820e-11
kgr_model4_fit$bri_hyperpar_summary

##          mean        sd      q0.025      q0.5      q0.975       mode
## SD for id2 0.5982378 0.0461062 0.5120181 0.5966276 0.6931084 0.5935777
kgr_model4_fit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##    8.135644   6.474481   6.622797   5.459995   5.039868   4.454297   4.296843
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##    4.202140   4.034393   4.367864   4.553239   5.895491  72.713035   8.601050
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##    7.964762  38.362113   6.429899   1.981044 150.037322
kgr_model4_fit$K_EPA_weight

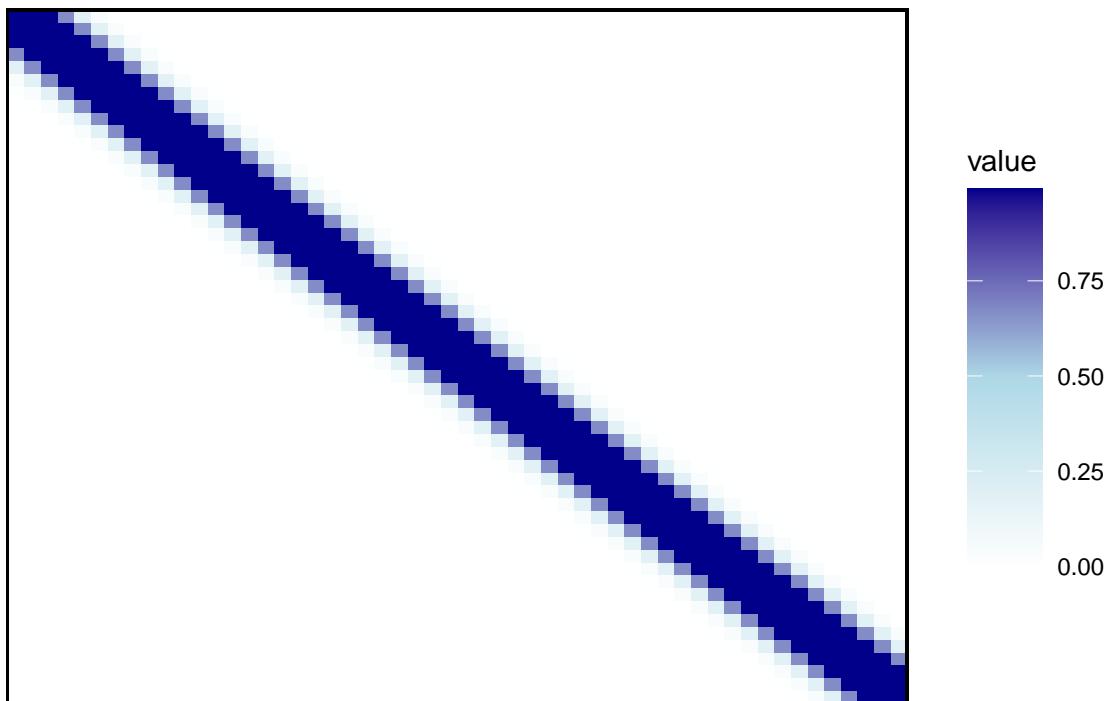
## [1] 0.5464999
kgr_model4_fit$K_time_weight

## [1] 0.4535001

```

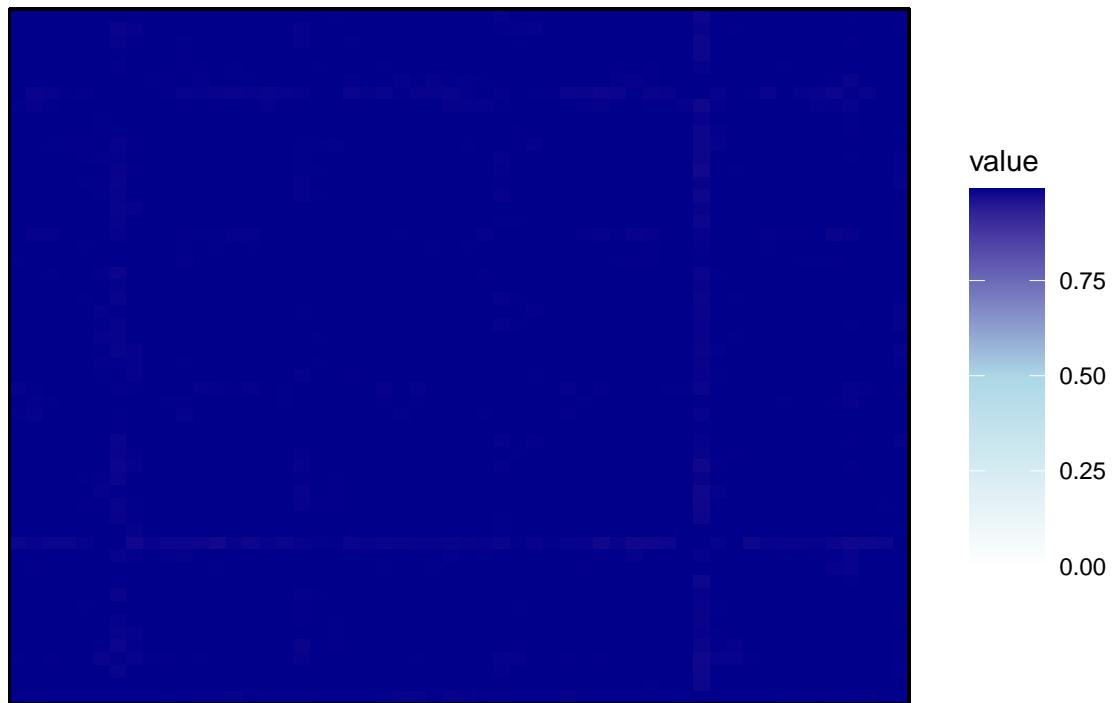
```
kgr_model4_fit$K_weight  
## [1] 0.740457  
kgr_model4_fit$gfilter_weight  
## [1] 0.259543  
kgr_model4_fit$num_jitters  
## [1] 1  
#Show plots  
kgr_model4_fit$K_time_heatmap
```

K_time heatmap

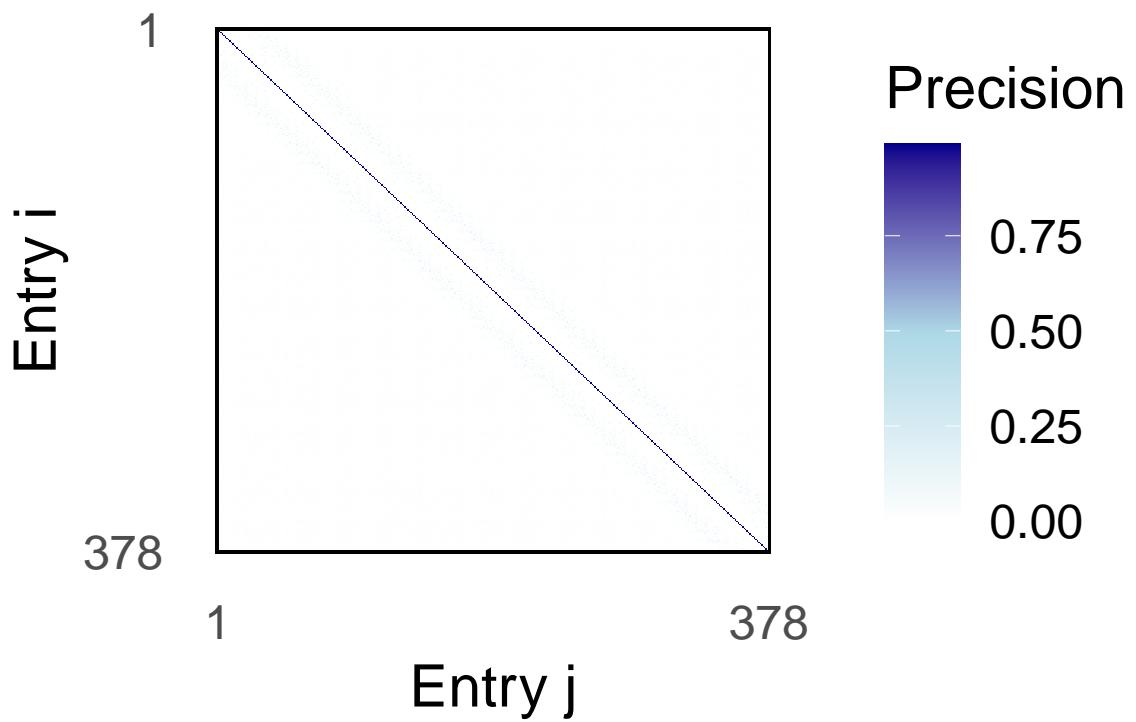


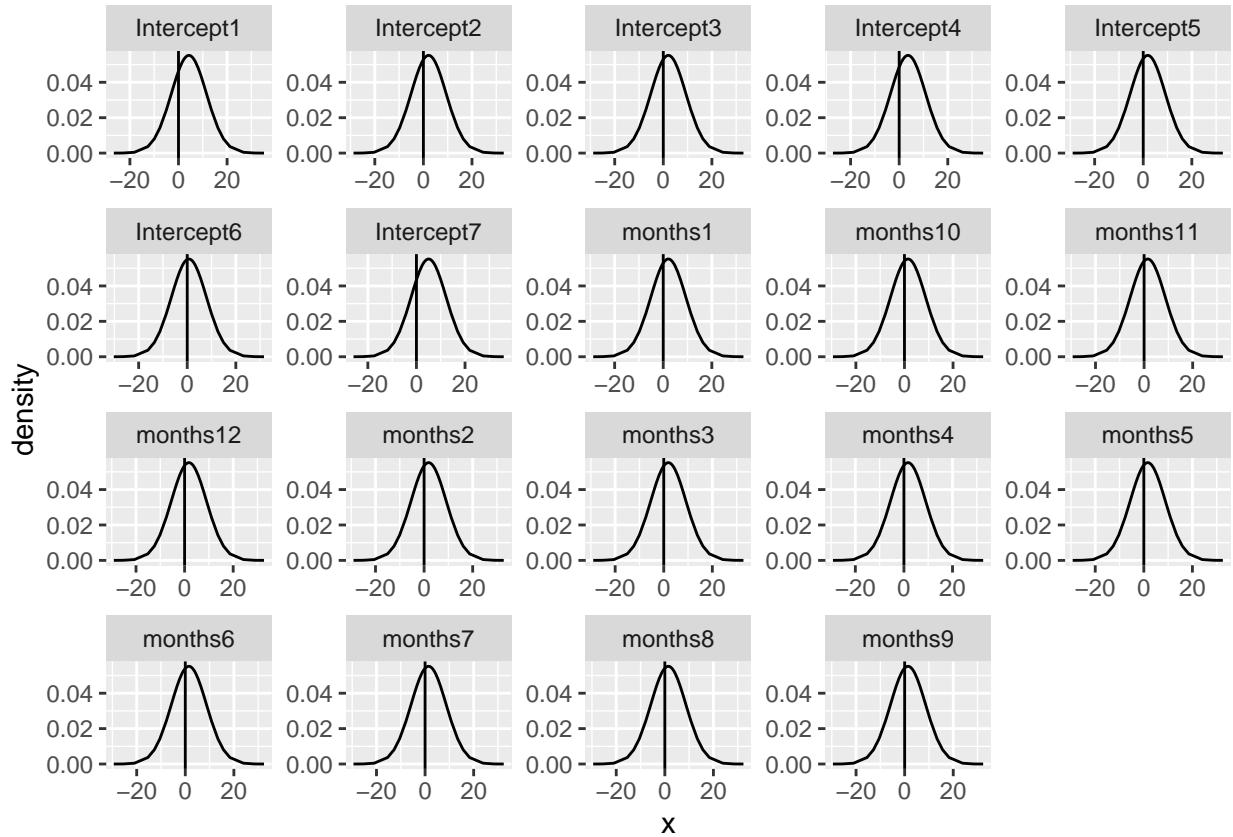
```
kgr_model4_fit$K_EPA_heatmap
```

K_EPA heatmap

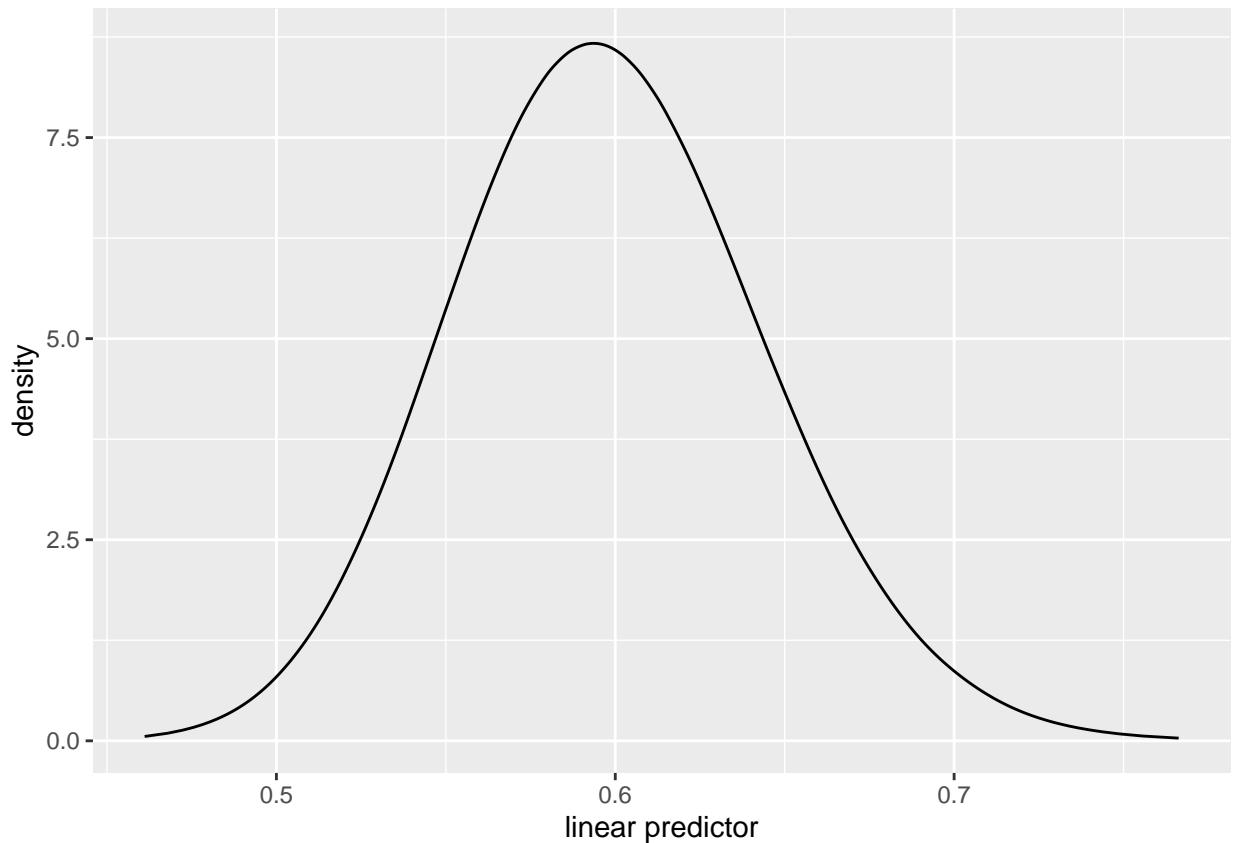


```
kgr_model4_fit$prec_heatmap
```

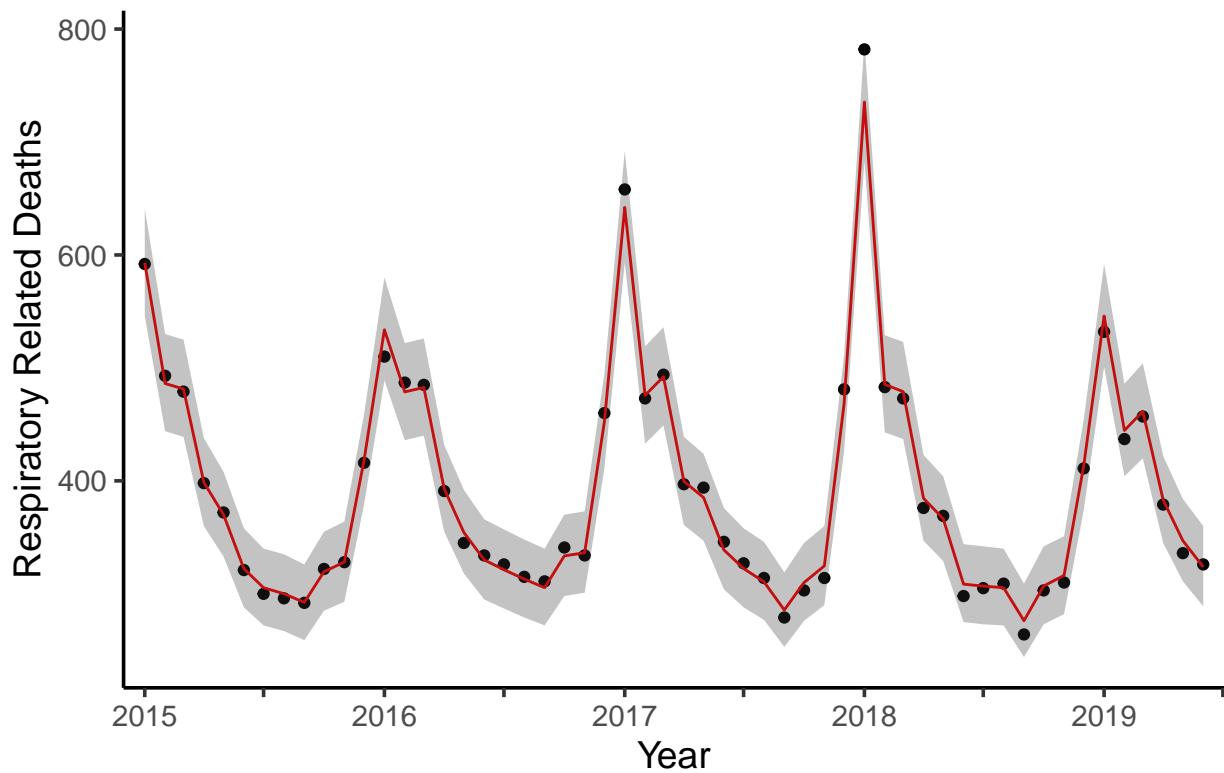




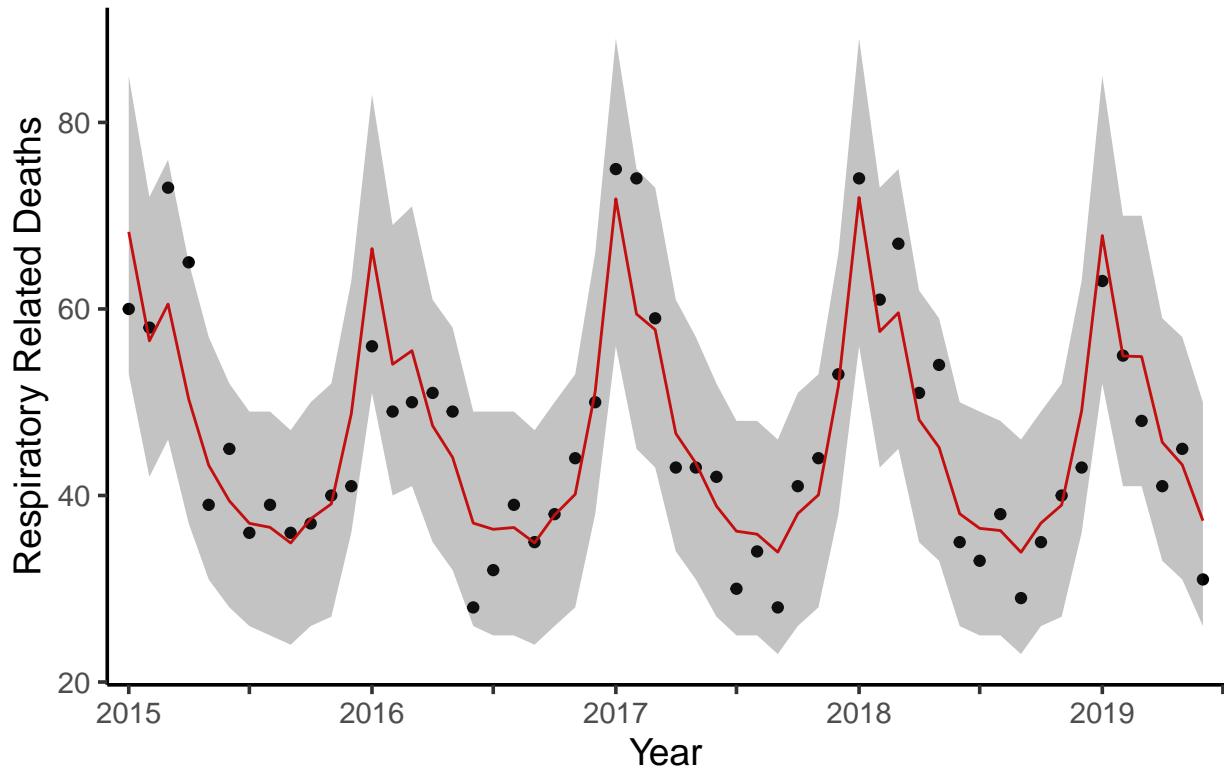
```
kgr_model4_fit$hyperparam_plot
```



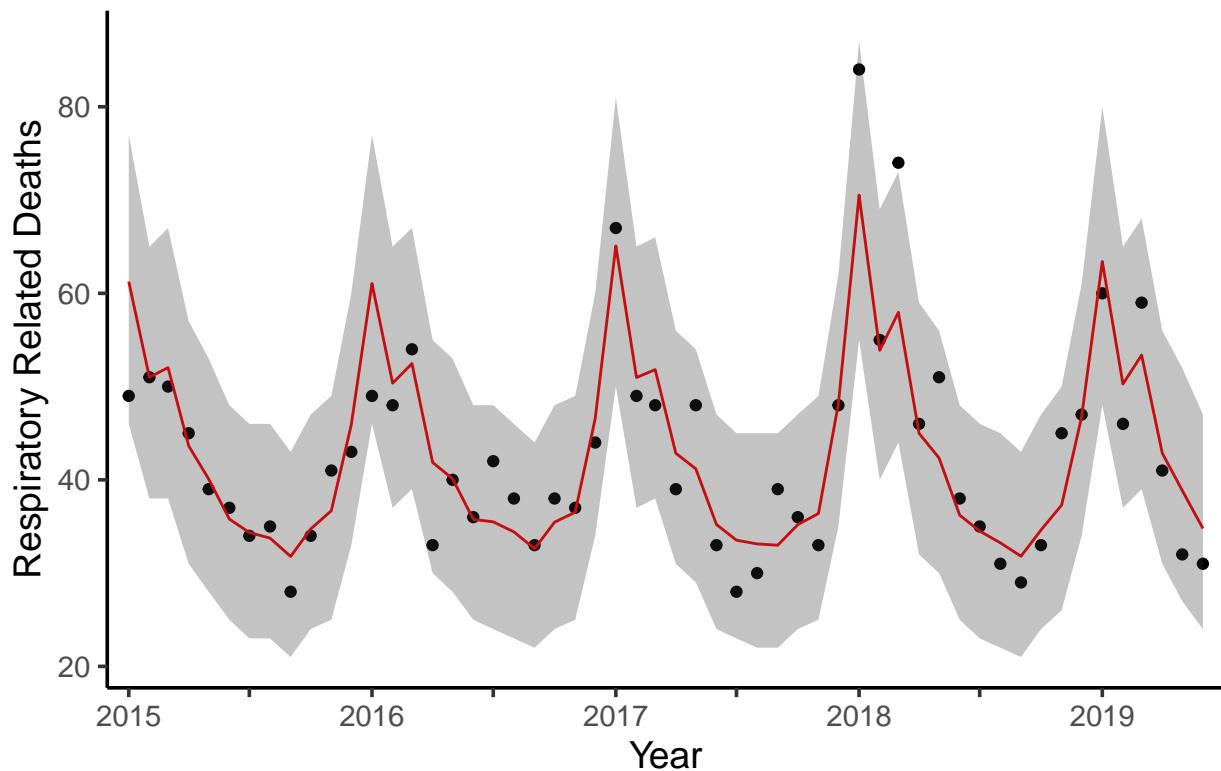
Cluster 1 (Coverage: 100%)



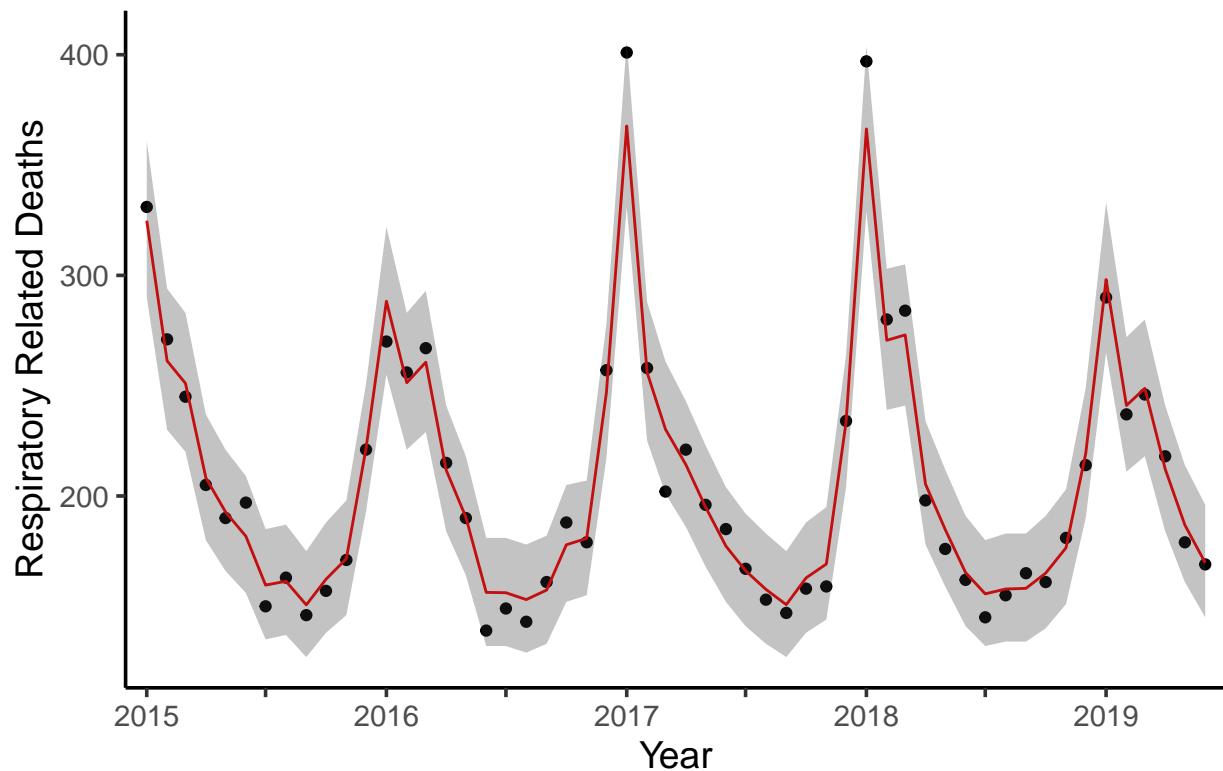
Cluster 2 (Coverage: 100%)



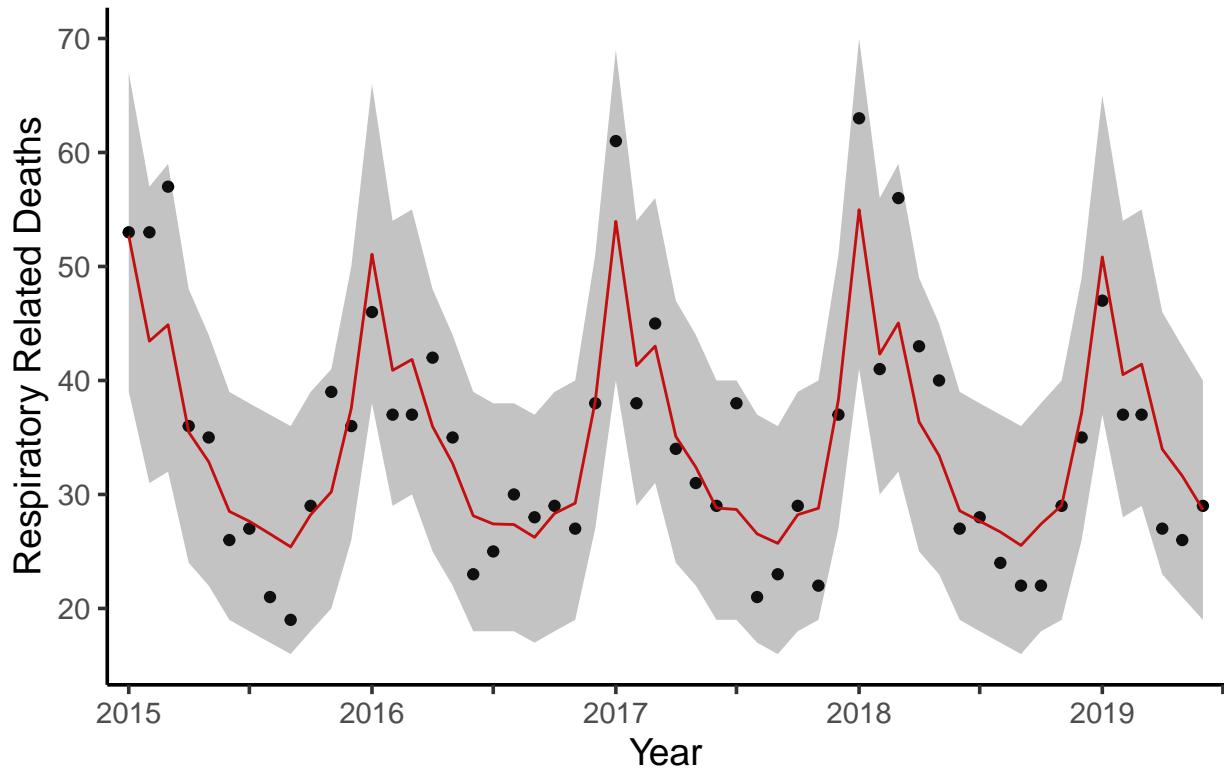
Cluster 3 (Coverage: 98.15%)



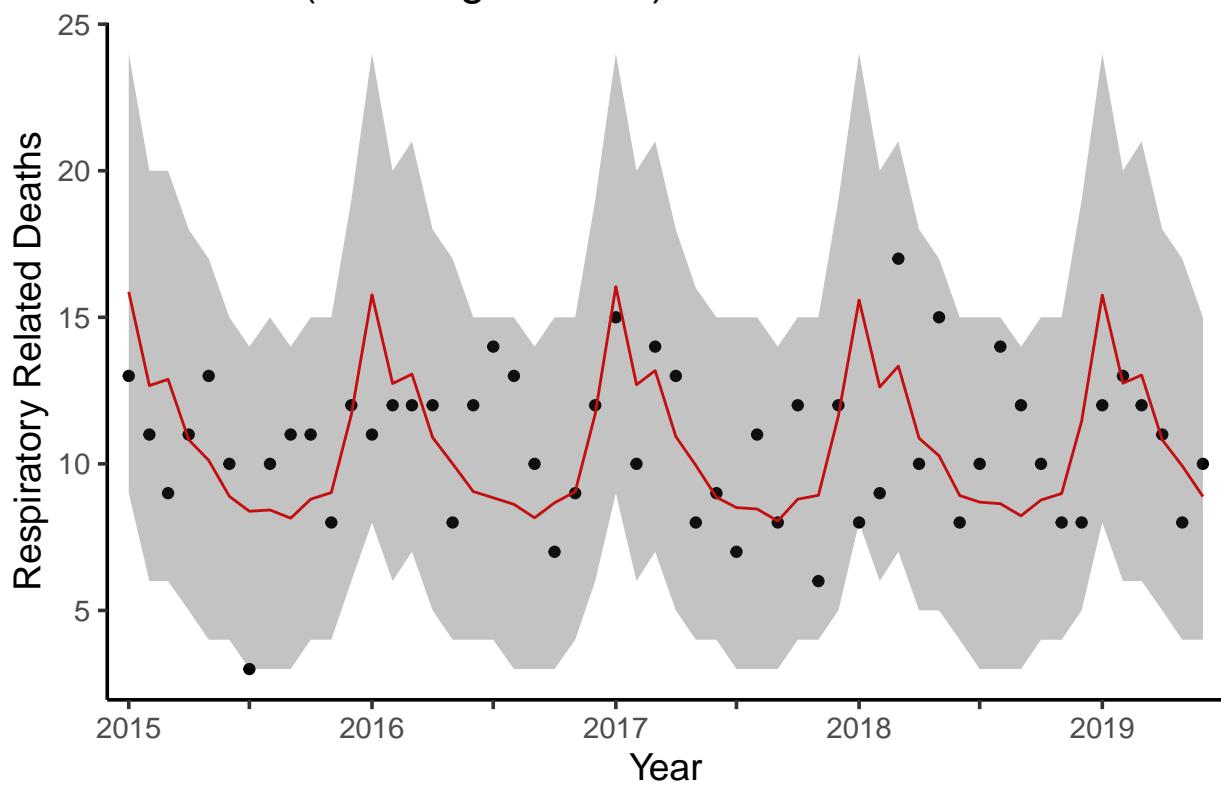
Cluster 4 (Coverage: 100%)



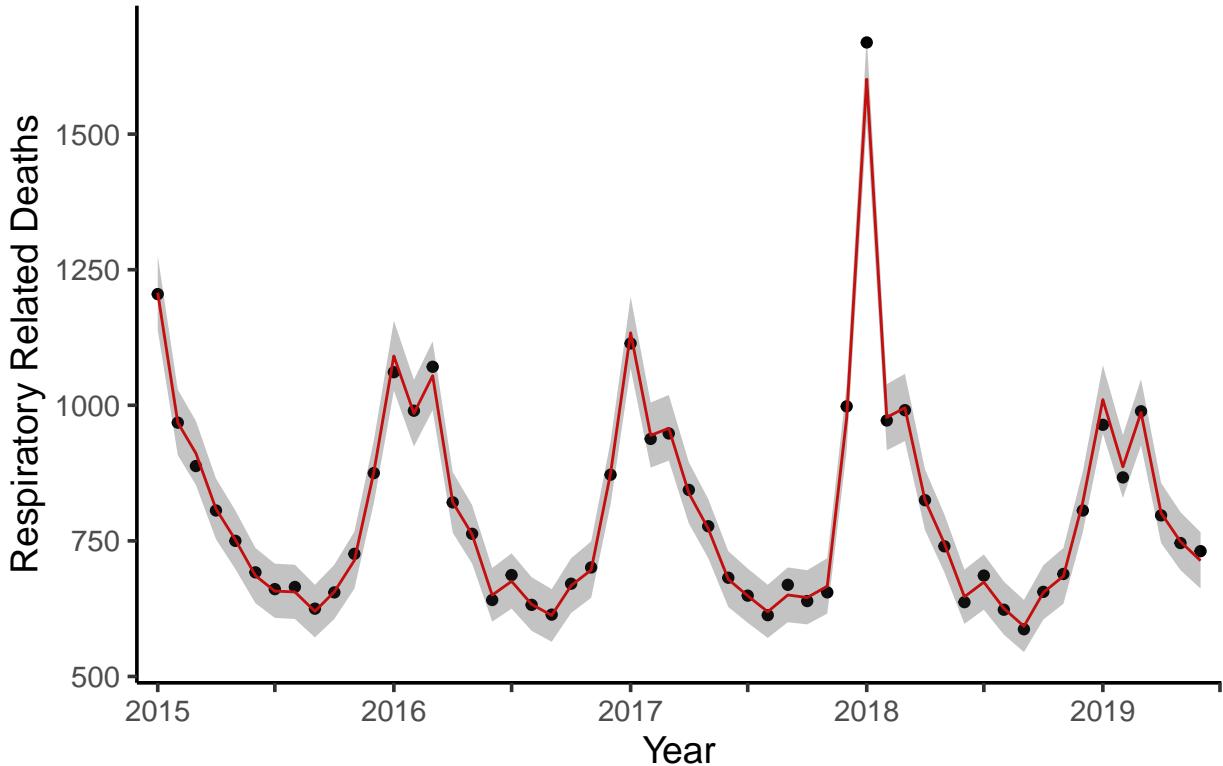
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 100%)



Cluster 7 (Coverage: 100%)



```
pp_insample_plot_combined(pred_data = kgr_model4_fit$fitted_values,prefix = "kgr4-insample-")
```

Proposed model 5

$$\Lambda_{c,t} | \mathbf{F}, \mathbf{S} = \exp(\beta_{c1} I\{c=1\} + \beta_{c2} I\{c=2\} + \dots + \beta_7 I\{c=7\} + \beta_1 I\{t \bmod 12 = 1\} + \dots + \beta_{11} I\{t \bmod 12 = 11\} + \mathbf{F}_{c,t})$$

where the graph signal $\mathbf{F} | \mathbf{S}, \rho_{rbf}^{AR}, \rho_p^{AR}, \rho_{rbf}^{DL}, \rho_p^{DL}, \rho_{rbf}^{Int}, \rho_p^{Int}, \sigma_{AR}^2, \sigma_{DL}^2, \sigma_{Int}^2 \sim \mathcal{GP}(\mathbf{0}, ((\frac{1}{3}\mathbf{K}_{AR} + \frac{1}{3}\mathbf{K}_{DL} + \frac{1}{3}\mathbf{K}_{Interaction}) \otimes \mathbf{H}^2))$ with $Cov[F_{n_1, t_1}, F_{n_2, t_2}] = (k(t_1, t_2) + k(x_{t_1}, x_{t_2}))(H^2)_{n_1, n_2}$.

```
kgr_model5 = function(dataset, rho_AR_rbf = 1, rho_AR_periodic = 1, rho_DL_rbf = 1, rho_DL_periodic = 1,
                      rho_int_rbf = 1, rho_int_periodic = 1, sigma2_AR = 1, sigma2_DL = 1, sigma2_int = 1)

#Calculating gram matrix K_AR
K_AR_cluster = list()
K_AR_periodic_cluster = list()

for (c in 1:num_clus){

  #Grab S_random data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  S_random_clus = cluster_data$S_random

  #Create a list to contain covariance matrix for each pollutant (8)
  K_AR_list = list()
  K_AR_periodic_list = list()
```

```

time_span = nrow(S_random_clus)

#Calculate a AR 1 covariance matrix for each pollutant and store in list
for (i in 1:8){
  ts = S_random_clus[,i]

  K_covariate = matrix(nrow=time_span,ncol=time_span)
  K_covariate_periodic = matrix(nrow=time_span,ncol=time_span)

  for(j in 1:time_span){
    for (k in 1:time_span){
      if (abs(j-k) <= 1){

        K_covariate[j,k] = exp(- ((ts[j] - ts[k])^2) #RBF kernel
                               / (2*rho_AR_rbf)) * sigma2_AR

        K_covariate_periodic[j,k] = exp(- ((ts[j] - ts[k])^2) #Locally periodic kernel
                                         / (2*rho_AR_rbf)) * exp(- (2*sin((abs(ts[j] - ts[k]))*pi/12)^2)
                                         / (rho_AR_periodic)) * sigma2_AR
      }
      else{
        K_covariate_periodic[j,k] = 0
        K_covariate[j,k] = 0
      }
    }
  }

  K_AR_list[[i]] = K_covariate
  K_AR_periodic_list[[i]] = K_covariate_periodic
}

names(K_AR_list) = colnames(S_random_clus)
names(K_AR_periodic_list) = colnames(S_random_clus)

#Add each pollutant's covariance matrix to get AR 1 matrix for each cluster
K_AR = matrix(0,nrow=60,ncol=60)
K_AR_periodic = matrix(0,nrow=60,ncol=60)

for(i in 1:length(K_AR_periodic_list)){
  K_AR = K_AR + ((1/8)*K_AR_list[[i]])
  K_AR_periodic = K_AR_periodic + ((1/8)*K_AR_periodic_list[[i]])
}

K_AR_cluster[[c]] = K_AR
K_AR_periodic_cluster[[c]] = K_AR_periodic
}

K_AR = matrix(0,nrow=60,ncol=60)
K_AR_periodic = matrix(0,nrow=60,ncol=60)

for(i in 1:num_clus){
  K_AR = K_AR + ((1/num_clus)*K_AR_cluster[[i]])
  K_AR_periodic = K_AR_periodic + ((1/num_clus)*K_AR_periodic_cluster[[i]])
}

```

```

}

K_AR_norm = norm(K_AR_periodic,type = "F")

#Heatmap of resulting K
# K_AR_heatmap = corrplot(K_AR, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col
# title = "AR 1 Covariance Structure")
# K_AR_heatmap = corrplot(K_AR_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color

# K_AR_heatmap = matrix_heatmap(K_AR,title = "AR 1 Covariance Structure")
K_AR_heatmap = matrix_heatmap(K_AR_periodic,title = "Periodic AR 1 Covariance Structure")

###Calculating gram matrix K_DL
K_DL_cluster = list()
K_DL_periodic_cluster = list()

for (c in 1:num_clus){

  #Grab S_DL data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  S_DL_clus = cluster_data$S_DL

  #Create a list to store covariance matrix for each DL
  K_DL_list = list()
  K_DL_periodic_list = list()

  dl_lags = c(3,6,12)
  tracker = 1

  for (i in dl_lags){

    K_DL = matrix(nrow=time_span,ncol=time_span)
    K_DL_periodic = matrix(nrow=time_span,ncol=time_span)

    #Calculate DL covariance matrix for specified lag
    for(j in 1:nrow(S_DL_clus)){
      for (k in 1:nrow(S_DL_clus)){

        if ((abs(j-k) == 0) || (abs(j-k) == i)){

          K_DL[j,k] = exp(-(sum(S_DL_clus[j,] - S_DL_clus[k,])^2) / (2*rho_DL_rbf)) * sigma2_DL

          K_DL_periodic[j,k] = exp(-(sum(S_DL_clus[j,] - S_DL_clus[k,])^2)
                                    / (2*rho_DL_rbf)) * exp(-(2*sin(sum(abs(S_DL_clus[j,] - S_DL_clus[k,])))))
                                    / (rho_DL_periodic)) * sigma2_DL

        }
        else{
          K_DL_periodic[j,k] = 0
          K_DL[j,k] = 0
        }
      }
    }
  }
}

```

```

K_DL_list[[tracker]] = K_DL
K_DL_periodic_list[[tracker]] = K_DL_periodic
tracker = tracker+1
}

#Combine the 3 DL covariance matrices together
K_DL = matrix(0,nrow=time_span,ncol=time_span)
K_DL_periodic = matrix(0,nrow=time_span,ncol=time_span)

for(i in 1:length(K_DL_periodic_list)){
  K_DL = K_DL + ((1/3)*K_DL_list[[i]])
  K_DL_periodic = K_DL_periodic + ((1/3)*K_DL_periodic_list[[i]])
}

#Store DL(3,6,12) covariance matrix for each cluster
K_DL_cluster[[c]] = K_DL
K_DL_periodic_cluster[[c]] = K_DL
}

K_DL = matrix(0,nrow=time_span,ncol=time_span)
K_DL_periodic = matrix(0,nrow=time_span,ncol=time_span)

for(i in 1:num_clus){
  K_DL = K_DL + ((1/num_clus)*K_DL_cluster[[i]])
  K_DL_periodic = K_DL_periodic + ((1/num_clus)*K_DL_periodic_cluster[[i]])
}

K_DL_norm = norm(K_DL_periodic,type = "F")

#Heatmap of resulting K
# K_DL_heatmap = corrplot(K_DL, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col ...
# title = "DL (3,6,12) Covariance Structure")
# K_DL_heatmap = corrplot(K_DL_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n',method = "color", col ...
# title = "Periodic DL (3,6,12) Covariance Structure")

# K_DL_heatmap = matrix_heatmap(K_DL,title = "DL (3,6,12) Covariance Structure")
K_DL_heatmap = matrix_heatmap(K_DL_periodic,title = "Periodic DL (3,6,12) Covariance Structure")

###Calculating gram matrix K_Interaction
K_Interaction_cluster = list()
K_Interaction_periodic_cluster = list()

for (c in 1:num_clus){

  #Grab interaction pair data for cluster c
  cluster_data = decomposed_cluster_data[[c]]
  W2_clus = cluster_data$W2

  K_interaction_list = list()
  K_interaction_periodic_list = list()

  column_names = colnames(W2_clus)
  time_span = nrow(W2_clus)
}

```

```

#Create sequence of indices corresponding to comparisons for real time and one lag interaction effe
lag0_idx = seq(2,3601,by=61)
lag1_idx = seq(1,3600,by=61)

#Calculate a kernel for each interaction pair
for (a in 1:length(column_names)){
  interaction = W2_clus[,a]

  #First calculate these two interaction kernels separately
  K_int0 = matrix(nrow = 60,ncol = 60)
  K_int1 = matrix(nrow = 60,ncol = 60)

  K_int0_periodic = matrix(nrow = 60,ncol = 60)
  K_int1_periodic = matrix(nrow = 60,ncol = 60)

  for (i in 1:60){
    for (j in 1:60){

      #RBF kernels
      K_int0[i,j] = exp(-( (interaction[lag0_idx[i]] - interaction[lag0_idx[j]])^2
                            / (2*rho_int_rbf)) * sigma2_int

      K_int1[i,j] = exp(-( (interaction[lag1_idx[i]] - interaction[lag1_idx[j]])^2
                            / (2*rho_int_rbf)) * sigma2_int

      #Locally periodic kernels
      K_int0_periodic[i,j] = exp(-( (interaction[lag0_idx[i]] - interaction[lag0_idx[j]])^2
                                    / (2*rho_int_rbf)) *
        exp(-(2*sin(abs(interaction[lag0_idx[i]] - interaction[lag0_idx[j]]))*pi/12)^2
            / (rho_int_periodic)) * sigma2_int

      K_int1_periodic[i,j] = exp(-( (interaction[lag1_idx[i]] - interaction[lag1_idx[j]])^2
                                    / (2*rho_int_rbf)) *
        exp(-(2*sin(abs(interaction[lag1_idx[i]] - interaction[lag1_idx[j]]))*pi/12)^2
            / (rho_int_periodic)) * sigma2_int
    }
  }

  #Combine real time and one lag interaction kernels together
  K_interaction = 0.5*K_int0 + 0.5*K_int1
  K_interaction_list[[a]] = K_interaction

  K_interaction_periodic = 0.5*K_int0_periodic + 0.5*K_int1_periodic
  K_interaction_periodic_list[[a]] = K_interaction_periodic
}

#Combine kernels for each interaction pair together
K_interaction = matrix(0,nrow=60,ncol=60)
K_interaction_periodic = matrix(0,nrow=60,ncol=60)

for(i in 1:length(K_interaction_periodic_list)){
  K_interaction = K_interaction + ((1/length(K_interaction_list))*K_interaction_list[[i]])
}

```

```

        K_interaction_periodic = K_interaction_periodic + ((1/length(K_interaction_periodic_list))*K_interaction_periodic)

    }

#Store final interaction kernel (for all pairs) for each cluster
K_Interaction_cluster[[c]] = K_interaction
K_Interaction_periodic_cluster[[c]] = K_interaction_periodic
}

K_interaction = matrix(0,nrow=60,ncol=60)
K_interaction_periodic = matrix(0,nrow=60,ncol=60)

for(i in 1:num_clus){
  K_interaction = K_interaction + ((1/length(K_Interaction_cluster))*K_Interaction_cluster[[i]])

  K_interaction_periodic = K_interaction_periodic + ((1/length(K_Interaction_periodic_cluster))*K_Interaction_periodic)
}

K_Int_norm = norm(K_interaction_periodic,type = "F")

#Heatmap of resulting K
# K_Interaction_heatmap = corrplot(K_interaction, order = 'original', cl.pos = 'b', tl.pos = 'n',method = 'lower')
# K_Interaction_heatmap = corrplot(K_interaction_periodic, order = 'original', cl.pos = 'b', tl.pos = 'n',method = 'lower')

# K_Interaction_heatmap = matrix_heatmap(K_interaction,title = "Interaction Covariance Structure")
K_Interaction_heatmap = matrix_heatmap(K_interaction_periodic,title = "Periodic Interaction Covariance Structure")

gram = (1/3)*(K_AR_periodic+K_DL_periodic+K_interaction_periodic)
K_weight = norm((1/60)*gram,type = "F")

####Load graph regression kernel
# covGP5 = kronecker(gram,H^2)
covGP5 = kronecker(gram/60,(H^2)/7)

#Need to ensure precision matrix is not computationally singular i.e det > 0
covGP_jittered = desingularize(covGP5,threshold = 1e-2,increment = 0.01)
covGP5 = covGP_jittered[[1]]

inv_covGP5 = solve(covGP5)

#Heatmap of resulting K
#inv_covGP5_heatmap = matrix_heatmap(inv_covGP5,title = "")
inv_covGP5_heatmap = matrix_heatmap2(inv_covGP5,title = "",legend_title = "Precision",prefix = "kgr5-")

####Fit INLA model
# kgr_formula5 = response ~ -1 + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
#   Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP5)

kgr_formula5 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 +
  Intercept5 + Intercept6 + Intercept7 + f(id2,model = "generic0",Cmatrix = inv_covGP5)

model = inla(formula = kgr_formula5,family = "poisson",data = dataset,
             control.compute = list(dic=TRUE,waic=TRUE,

```

```

                    return.marginals.predictor=TRUE),
control.inla = list(strategy = "laplace"),
control.predictor = list(compute = TRUE, link = link))

###Extract relevant information and store in the list
model_summary <- model$summary.fixed
bri_hyperpar_summary <- bri.hyperpar.summary(model)
modelDIC <- model$dic$dic
model_WAIC <- model$waic$waic
preds_model <- model$summary.fitted.values
preds_model <- cbind(dataset$id, dataset$time, preds_model)
colnames(preds_model) <- c("id", "time", "mean", "sd", "0.025quant", "0.5quant", "0.975quant", "mode")
marginal_fvs <- model$marginals.fitted.values

#Exponentiating parameter to get better interpretation of estimates
multeff <- exp(model$summary.fixed$mean)
names(multeff) <- model$names.fixed

#Plot of each parameters' posterior density
mf <- melt(model$marginals.fixed)
cf <- spread(mf, Var2, value)
names(cf)[2] <- 'parameter'
param_plot = ggplot(cf, aes(x=x, y=y)) + geom_line() + facet_wrap(~ parameter,
scales="free") + geom_vline(xintercept=0) + ylab("density")

#Plot of precision of random effect (main hyperparameter of interest)
sden <- data.frame(bri.hyper.sd(model$marginals.hyperpar[[1]]))
hyperparam_plot = ggplot(sden, aes(x, y)) + geom_line() + ylab("density") +
xlab("linear predictor")

#Store the results in the list
kgr_model5_results = list(
K_AR_heatmap = K_AR_heatmap,
K_DL_heatmap = K_DL_heatmap,
K_Interaction_heatmap = K_Interaction_heatmap,
K_AR_weight = K_AR_norm / (K_AR_norm + K_DL_norm + K_Int_norm),
K_DL_weight = K_DL_norm / (K_AR_norm + K_DL_norm + K_Int_norm),
K_Int_weight = K_Int_norm / (K_AR_norm + K_DL_norm + K_Int_norm),
K_weight = K_weight / (K_weight + gfilter_weight),
gfilter_weight = gfilter_weight / (K_weight + gfilter_weight),
gram_matrix = gram,
covmatrix = covGP5,
prec = inv_covGP5,
num_jitters = covGP_jittered[[2]],
prec_heatmap = inv_covGP5_heatmap,
model_summary = model_summary,
bri_hyperpar_summary = bri_hyperpar_summary,
exp_effects = multeff,
param_plot = param_plot,
hyperparam_plot = hyperparam_plot,
modelDIC = modelDIC,
model_WAIC = model_WAIC,
fitted_values = preds_model,
)

```

```

        marg_fitted_values = marginal_fvs
    )

    return(kgr_model5_results)
}

#Fit kgr_model5
kgr_model5_fit = kgr_model5(dataset = inla_insample_data, rho_AR_rbf = 0.016, rho_AR_periodic = 0.019,
                             rho_DL_rbf = 0.013, rho_DL_periodic = 0.003, rho_int_rbf = 0.002,
                             rho_int_periodic = 0.011, sigma2_AR = 4.002, sigma2_DL = 0.486, sigma2_int ...

#Posterior predictive sampling from estimated intensities
kgr_model5_fit$fitted_values = poisson_pp_sampling(kgr_model5_fit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model5_DIC = kgr_model5_fit$modelDIC
kgr_model5_WAIC = kgr_model5_fit$modelWAIC

#Get summaries of parameter estimates
kgr_model5_fit$model_summary

##          mean      sd 0.025quant 0.5quant 0.975quant      mode
## months1   2.0953307 7.254730 -12.130475 2.0953307   16.32114 2.0953307
## months2   1.8677250 7.254731 -12.358084 1.8677250   16.09353 1.8677250
## months3   1.8913754 7.254731 -12.334433 1.8913754   16.11718 1.8913754
## months4   1.6974133 7.254733 -12.528399 1.6974133   15.92323 1.6974133
## months5   1.6169280 7.254734 -12.608886 1.6169280   15.84274 1.6169280
## months6   1.4929665 7.254735 -12.732850 1.4929665   15.71878 1.4929665
## months7   1.4582944 7.254742 -12.767535 1.4582944   15.68412 1.4582944
## months8   1.4358546 7.254742 -12.789976 1.4358546   15.66169 1.4358546
## months9   1.3937548 7.254743 -12.832076 1.3937548   15.61959 1.3937548
## months10  1.4741204 7.254741 -12.751708 1.4741204   15.69995 1.4741204
## months11  1.5174906 7.254741 -12.708337 1.5174906   15.74332 1.5174906
## months12  1.7748719 7.254738 -12.450949 1.7748719   16.00069 1.7748719
## Intercept1 4.2889358 7.254727 -9.936864 4.2889358   18.51474 4.2889358
## Intercept2 2.1515814 7.254744 -12.074252 2.1515814   16.37741 2.1515814
## Intercept3 2.0712802 7.254752 -12.154568 2.0712802   16.29713 2.0712802
## Intercept4 3.6581737 7.254733 -10.567637 3.6581737   17.88398 3.6581737
## Intercept5 1.8605166 7.254750 -12.365329 1.8605166   16.08636 1.8605166
## Intercept6 0.6781004 7.254837 -13.547915 0.6781004   14.90412 0.6781004
## Intercept7 5.0075374 7.254718 -9.218246 5.0075374   19.23332 5.0075374

##          kld
## months1   5.526824e-11
## months2   5.526817e-11
## months3   5.526822e-11
## months4   5.526822e-11
## months5   5.526817e-11
## months6   5.526817e-11
## months7   5.526817e-11
## months8   5.526817e-11
## months9   5.526823e-11
## months10  5.526817e-11
## months11  5.526817e-11
## months12  5.526816e-11

```

```

## Intercept1 5.526827e-11
## Intercept2 5.526822e-11
## Intercept3 5.526812e-11
## Intercept4 5.526823e-11
## Intercept5 5.526822e-11
## Intercept6 5.526823e-11
## Intercept7 5.526821e-11
kgr_model5_fit$bri_hyperpar_summary

##               mean        sd     q0.025      q0.5     q0.975      mode
## SD for id2 0.587171 0.04499062 0.5029803 0.5856187 0.6796956 0.5826715

kgr_model5_fit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##  8.128128   6.473552   6.628479   5.459806   5.037591   4.450278   4.298622
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##  4.203235   4.029953   4.367193   4.560766   5.899525  72.888861   8.598445
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##  7.934975  38.790433   6.427056   1.970132 149.536030

kgr_model5_fit$K_AR_weight

## [1] 0.3121421
kgr_model5_fit$K_DL_weight

## [1] 0.03542519
kgr_model5_fit$K_Int_weight

## [1] 0.6524327
kgr_model5_fit$K_weight

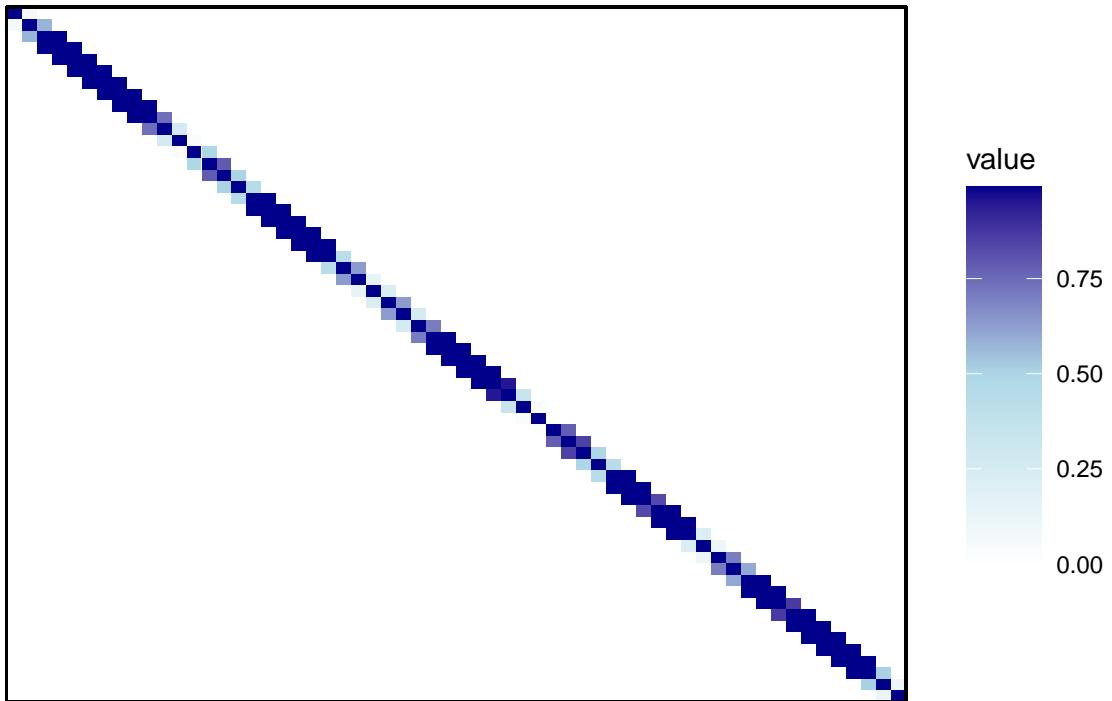
## [1] 0.6824774
kgr_model5_fit$gfilter_weight

## [1] 0.3175226
kgr_model5_fit$num_jitters

## [1] 1
#Show plots
kgr_model5_fit$K_AR_heatmap

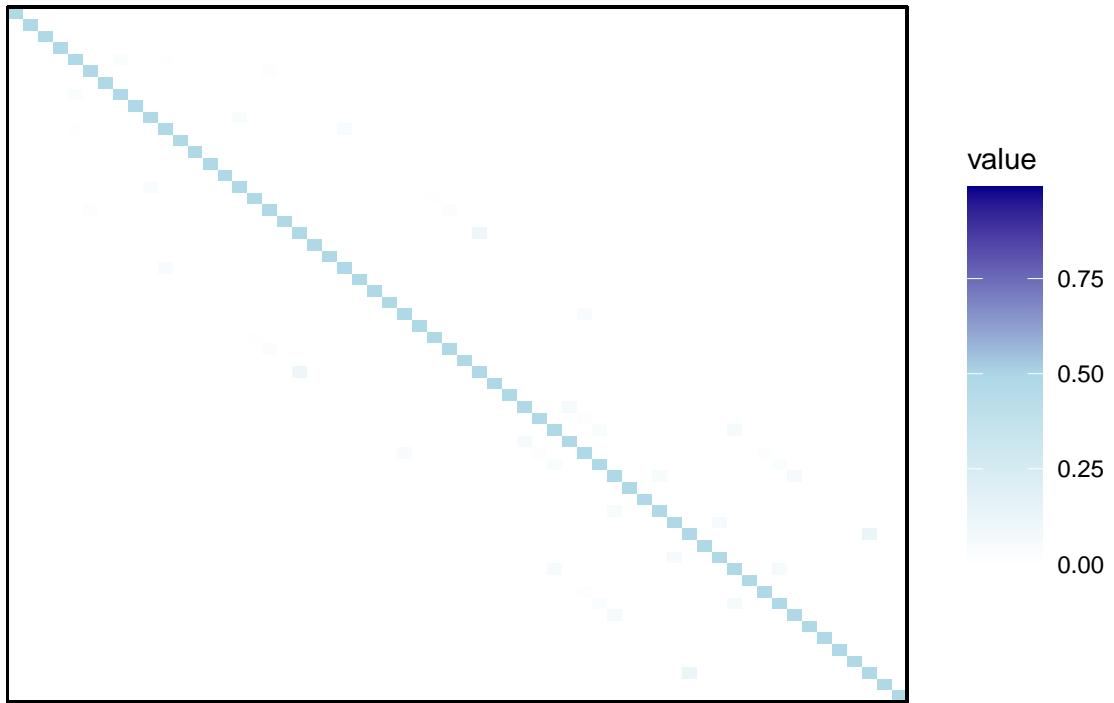
```

Periodic AR 1 Covariance Structure



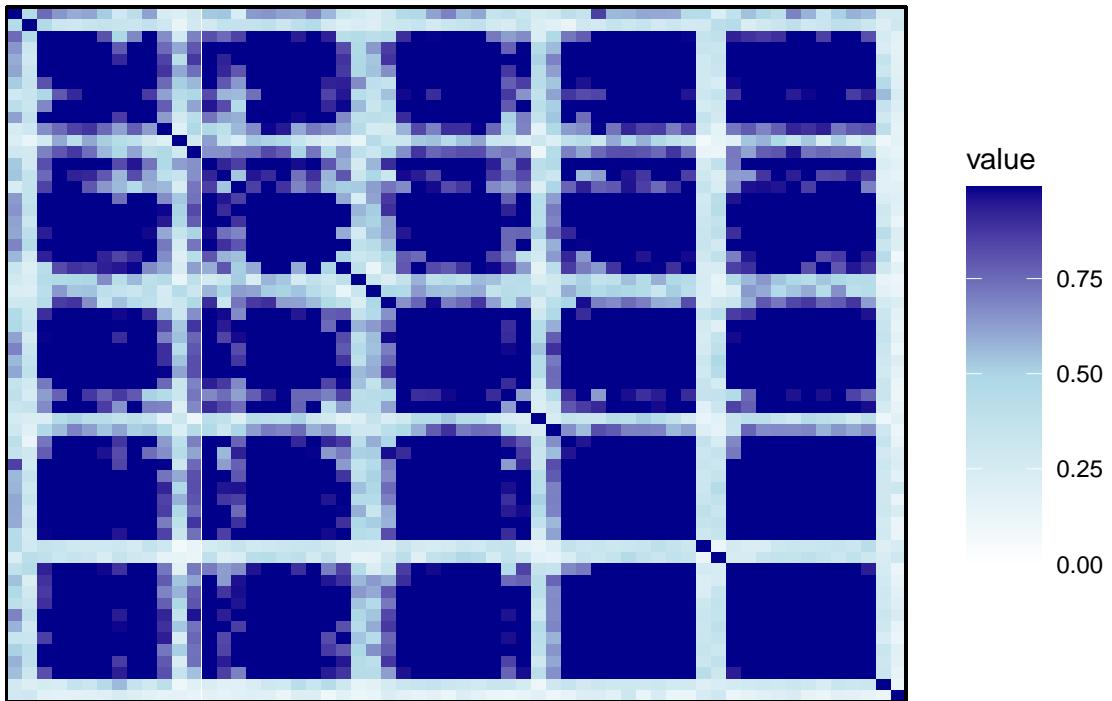
kgr_model5_fit\$K_DL_heatmap

Periodic DL (3,6,12) Covariance Structure

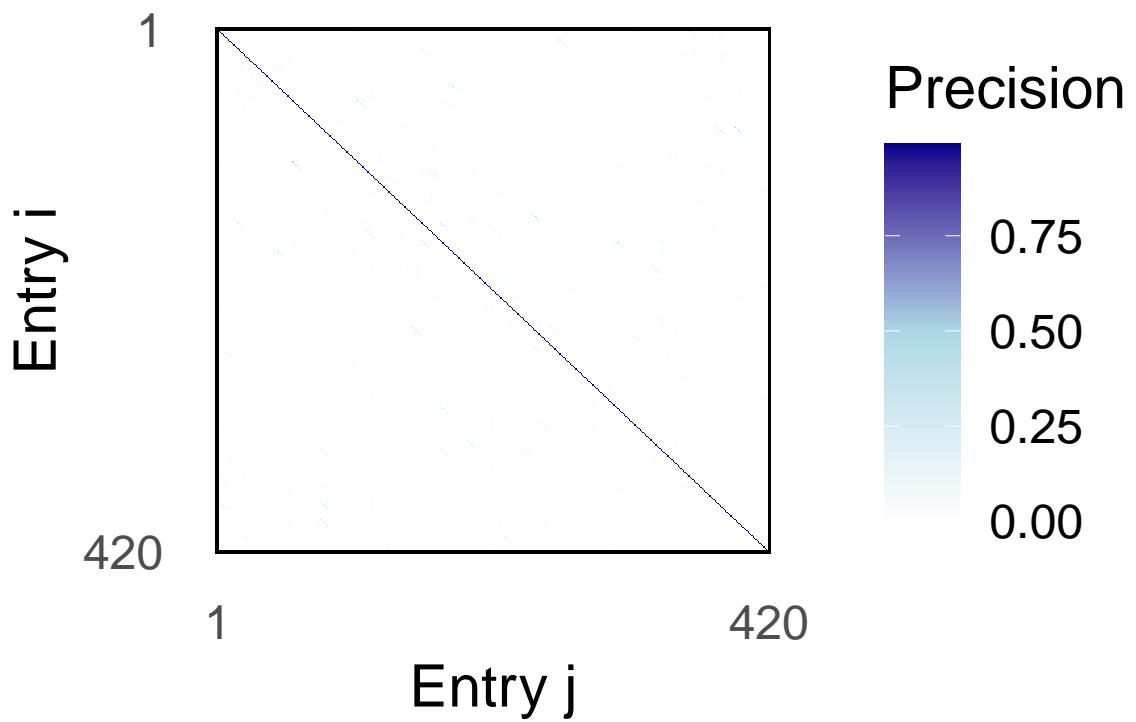


kgr_model5_fit\$K_Interaction_heatmap

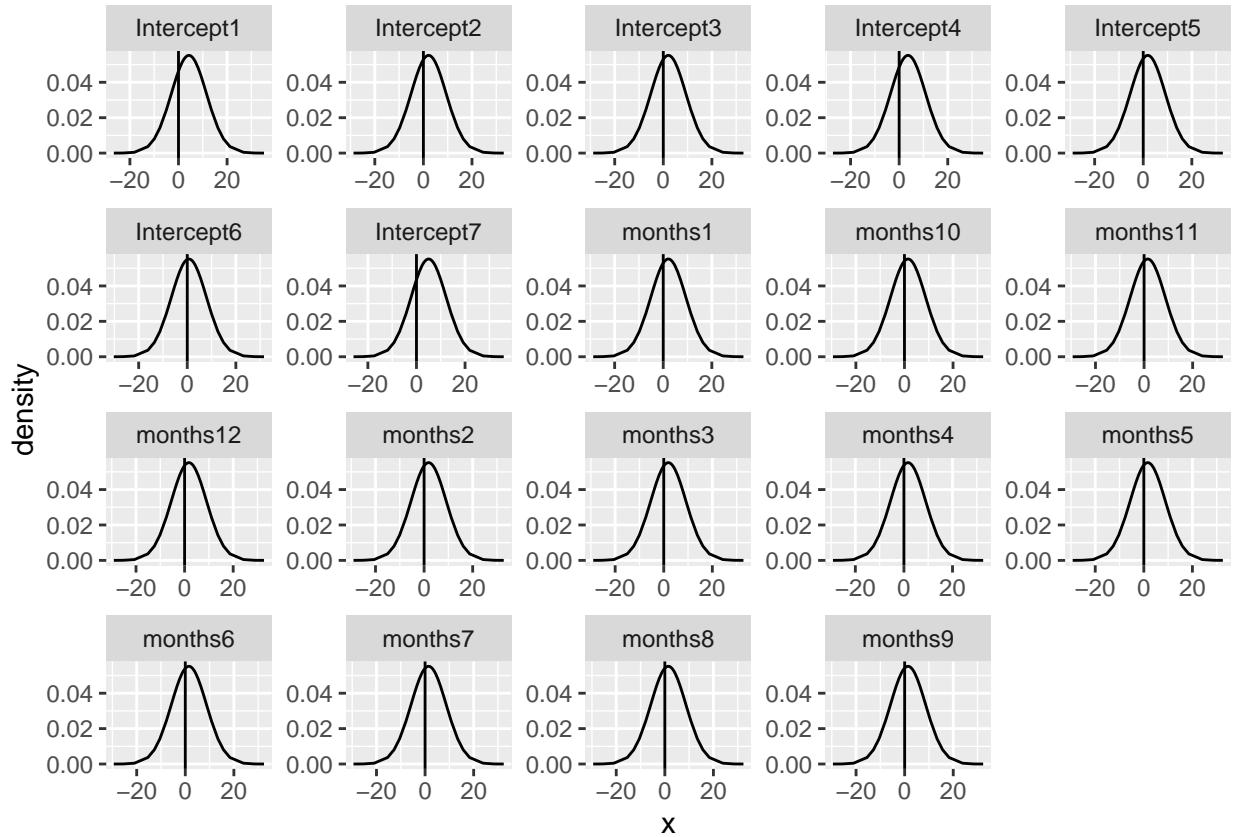
Periodic Interaction Covariance Structure



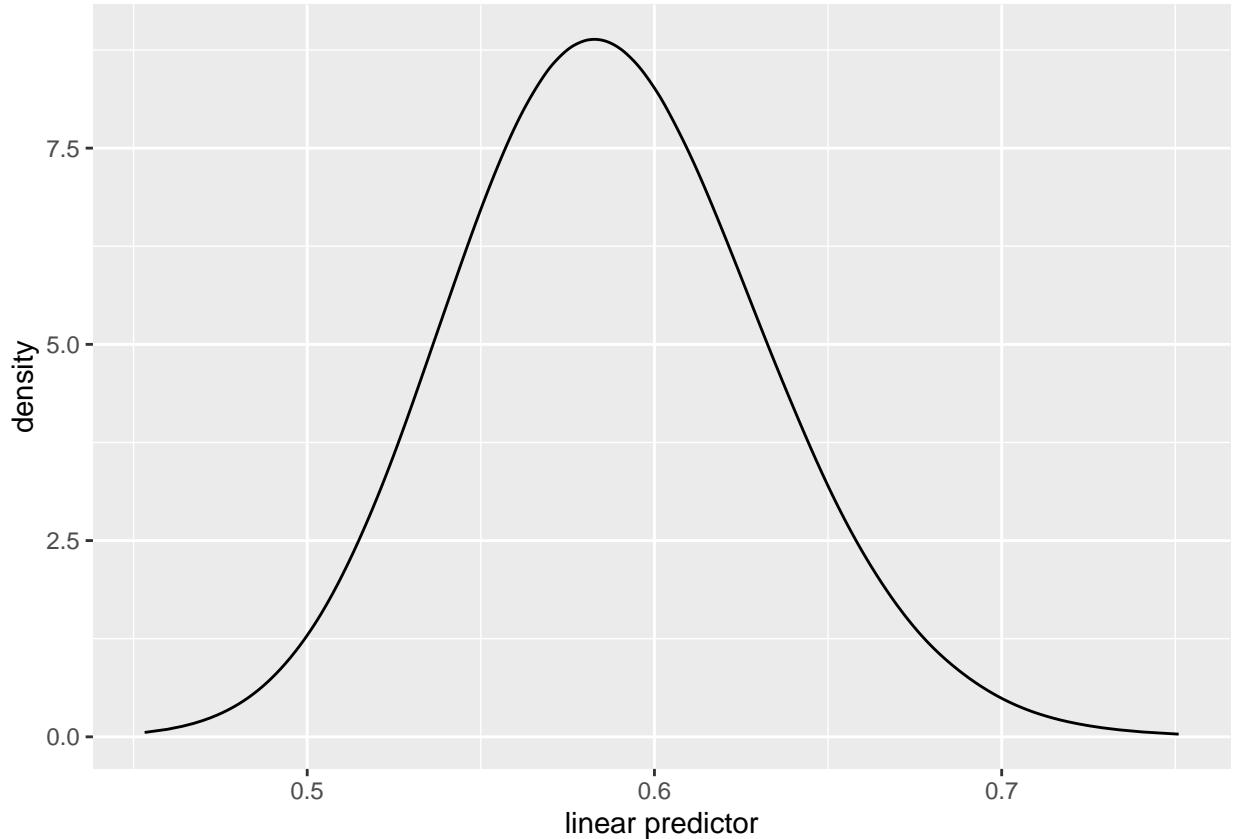
```
kgr_model5_fit$prec_heatmap
```



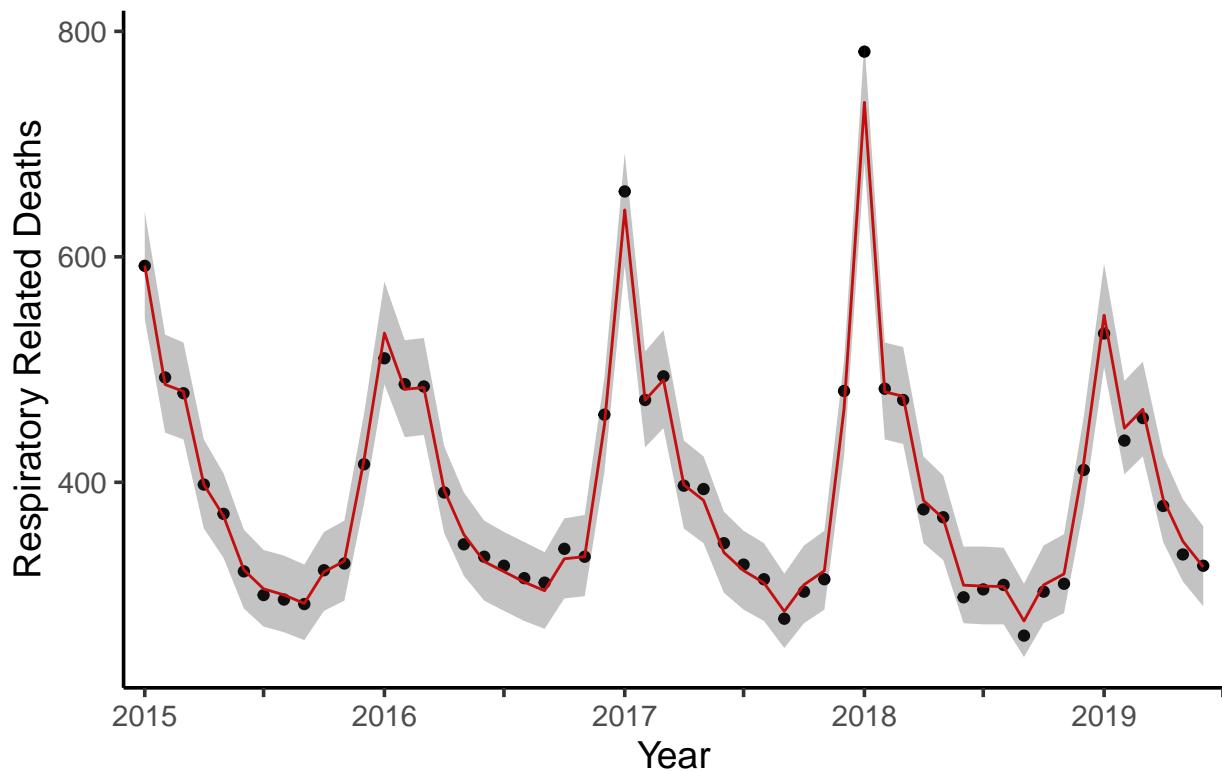
```
kgr_model5_fit$param_plot
```



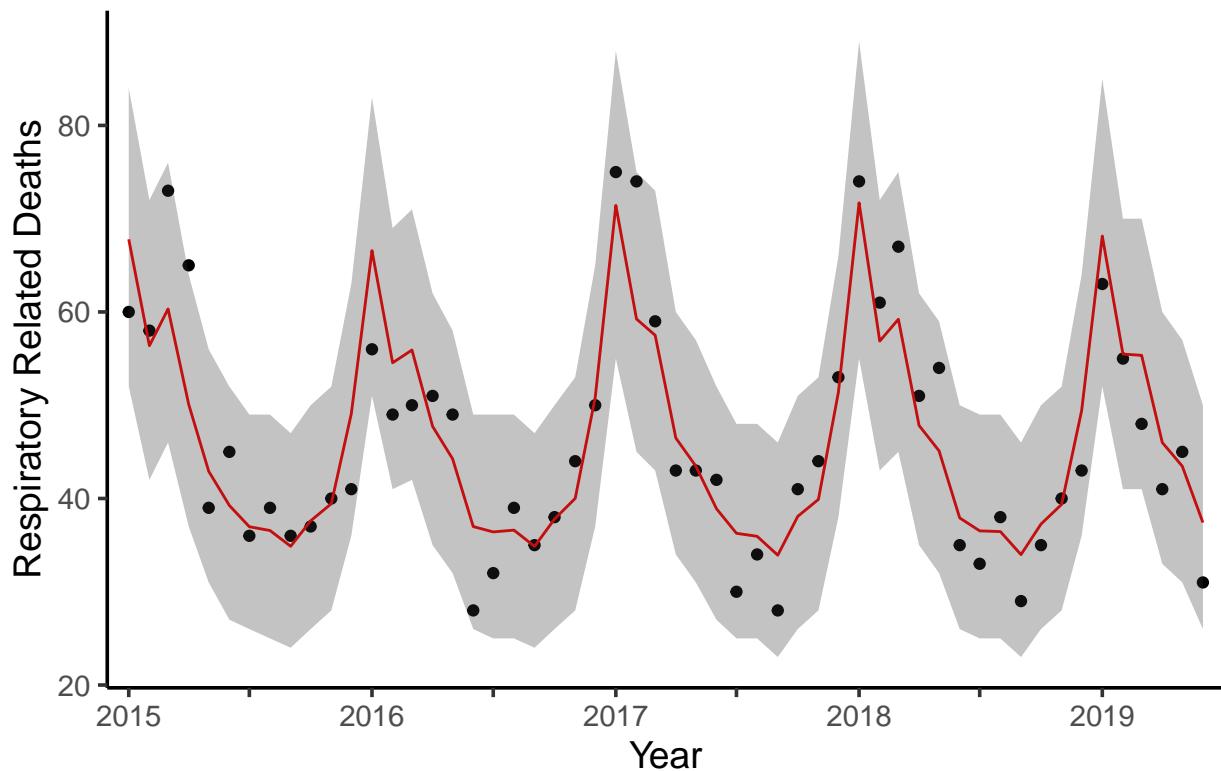
```
kgr_model5_fit$hyperparam_plot
```



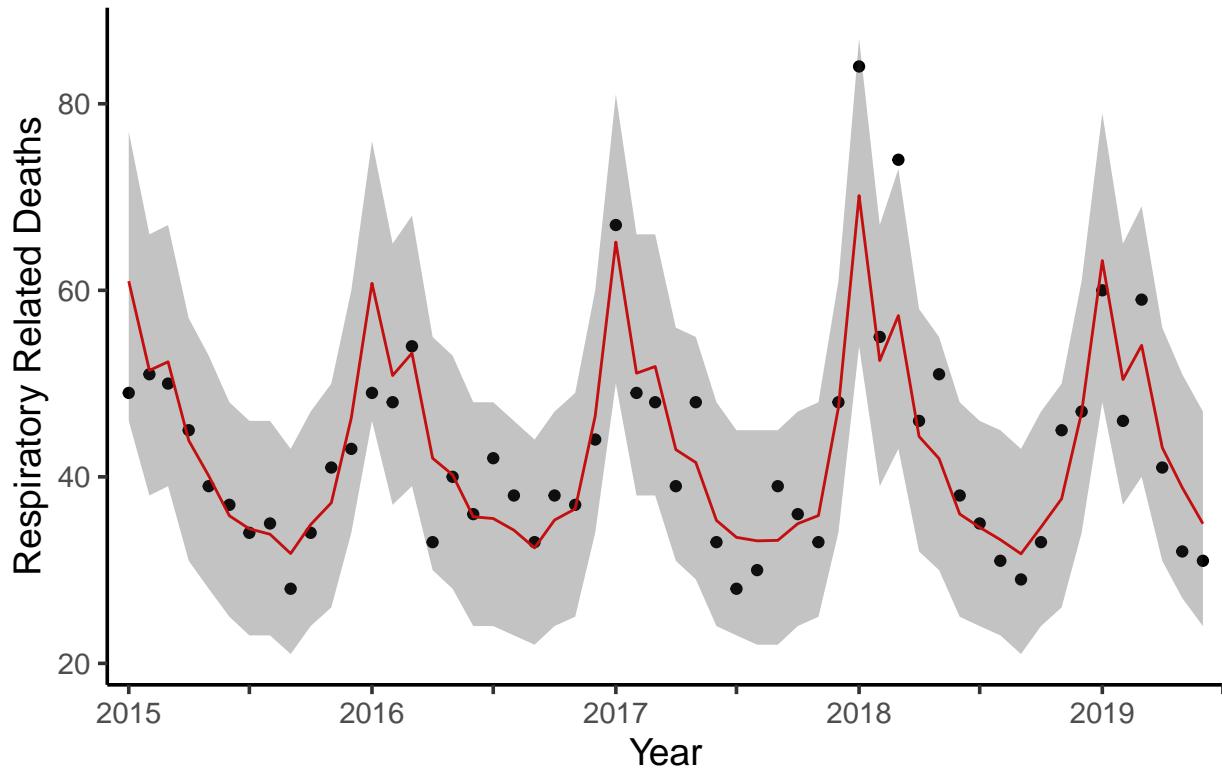
Cluster 1 (Coverage: 100%)



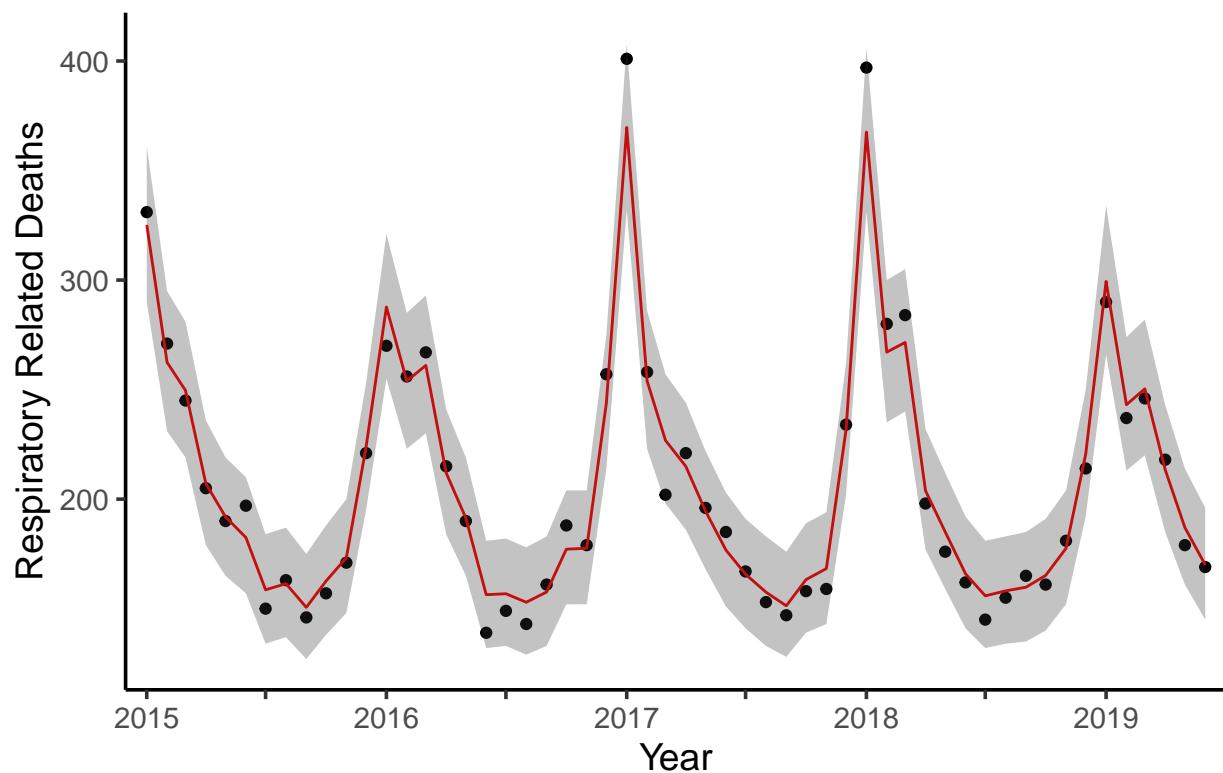
Cluster 2 (Coverage: 98.15%)



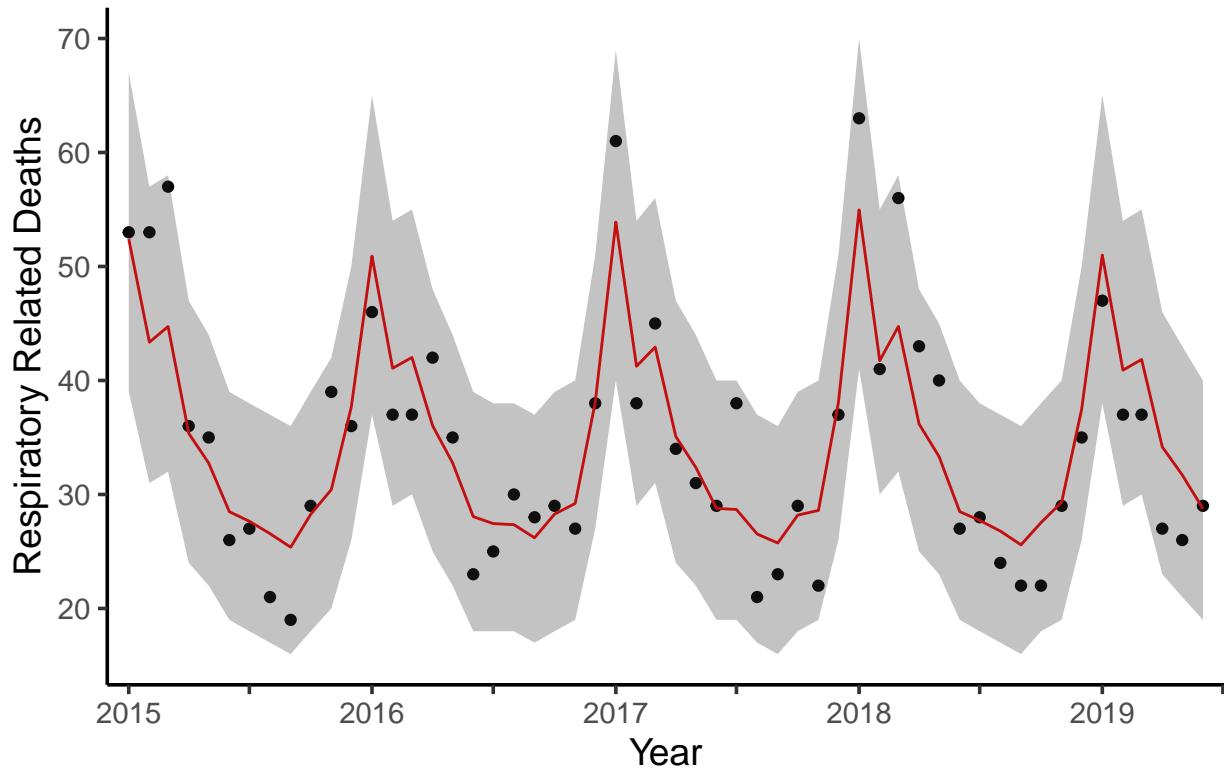
Cluster 3 (Coverage: 98.15%)



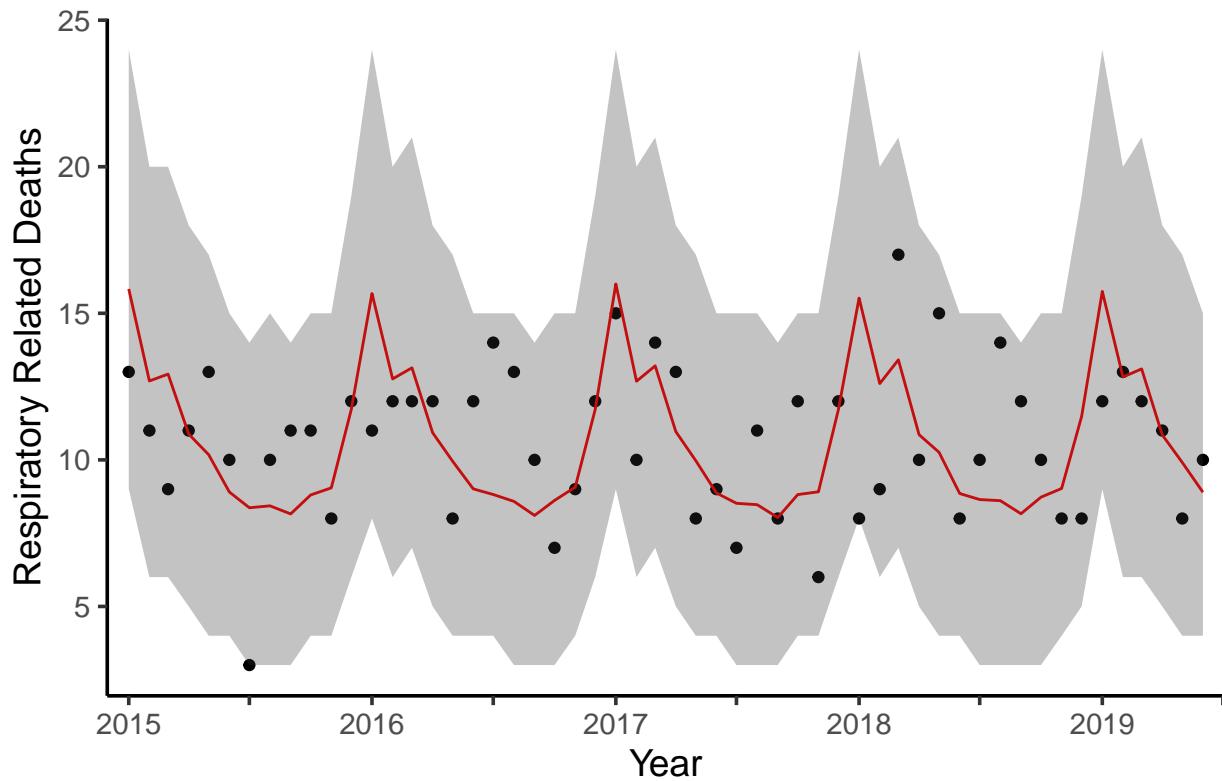
Cluster 4 (Coverage: 100%)



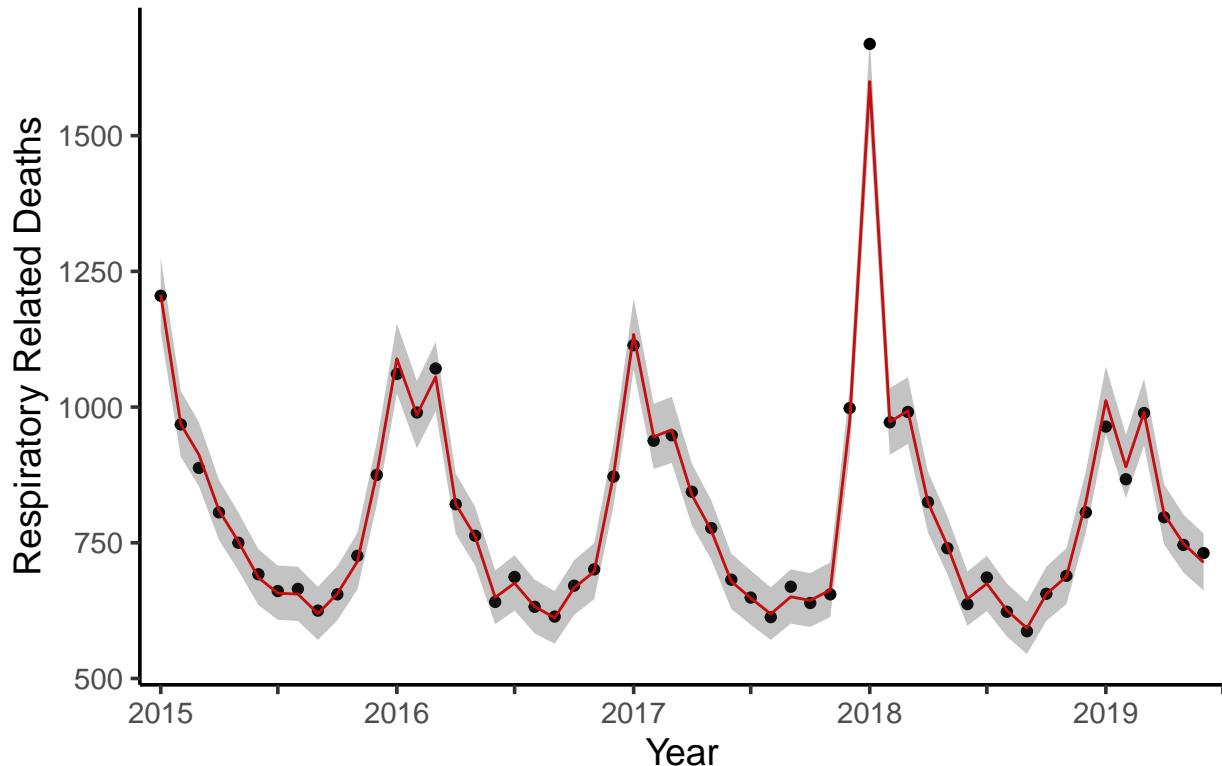
Cluster 5 (Coverage: 100%)



Cluster 6 (Coverage: 100%)



Cluster 7 (Coverage: 100%)



```
pp_insample_plot_combined(pred_data = kgr_model5_fit$fitted_values,prefix = "kgr5-insample-")
```

What if we cluster, estimate graph, apply kernel to response data?

SKATER clustering on response in 2014

```
CA_sf = st_read(getwd(),"CA_Counties_TIGER2016")

## Reading layer `CA_Counties_TIGER2016` from data source
##   `C:\Users\jeffr\Desktop\Spatiotemporal + Causal Inference\Wildfire Paper 1 Code'
##   using driver `ESRI Shapefile'
## Simple feature collection with 58 features and 17 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: -13857270 ymin: 3832931 xmax: -12705030 ymax: 5162404
## Projected CRS: WGS 84 / Pseudo-Mercator

CA_spdf = as_Spatial(CA_sf)

response_scaled = data.frame(scale(total.mortality.ts[,4:15]))
CA_spdf@data = response_scaled

#Identify neighborhood list for counties
CA_nb = poly2nb(CA_spdf)

#summary(CA_nb)
```

```

# plot(CA_spdf, main = "With queen")
# plot(CA_nb, coords = coordinates(CA_spdf), col="blue", add = TRUE)

#Calculate edge costs (dissimilarity matrix) based on Euclidean distance
costs <- nbcosts(CA_nb, data = response_scaled)

####Get adjacency matrix using nb2mat() (SEPARATE STEP FOR INLA)
adj = nb2mat(CA_nb,style = "B")

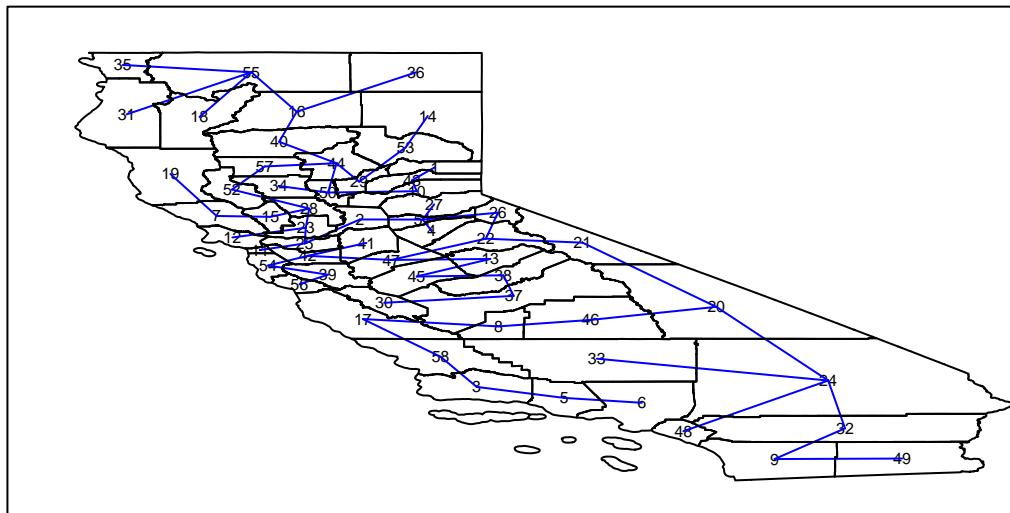
#Style means the coding scheme style used to create the weighting matrix
# B: basic binary coding scheme
# W: row standardized coding scheme
# C: globally standardized coding scheme
# U: values of C / number of neighbors
# S: variance stabilizing coding scheme

#Transform edge costs to spatial weights
ct_w <- nb2listw(CA_nb,costs,style="B")

#Create minimum spanning tree
ct_mst <- mstree(ct_w)

plot(ct_mst,coordinates(CA_spdf),col="blue", cex.lab=0.5)
plot(CA_spdf, add=TRUE)

```



```

#Run SKATER algorithm to get 7 contiguous clusters (cluster idx is in order of CA_sf)
clus7_response <- skater(edges = ct_mst[,1:2], data = response_scaled, ncuts = 6)

#Determine an appropriate minimum population threshold based on???
pops_summary = summary(unique(CA_data$Total_Pop))
pops_summary

##      Min.    1st Qu.     Median      Mean    3rd Qu.      Max.
## 12700    63275    219705   710796   750235 10105722

#Idea 1: Use median * (how many counties should be in a cluster at minimum)
min_pop = as.numeric(pops_summary[3] * 4)

#Idea 2: If we assume CA population is 39M, divide total pop by # clusters
min_pop2 = 39000000 / 7

#Add a min population constraint
clus7_response_min <- skater(edges = ct_mst[,1:2],
                                data = response_scaled,
                                crit = min_pop,
                                vec.crit = CA_data$Total_Pop,
                                ncuts = 6)

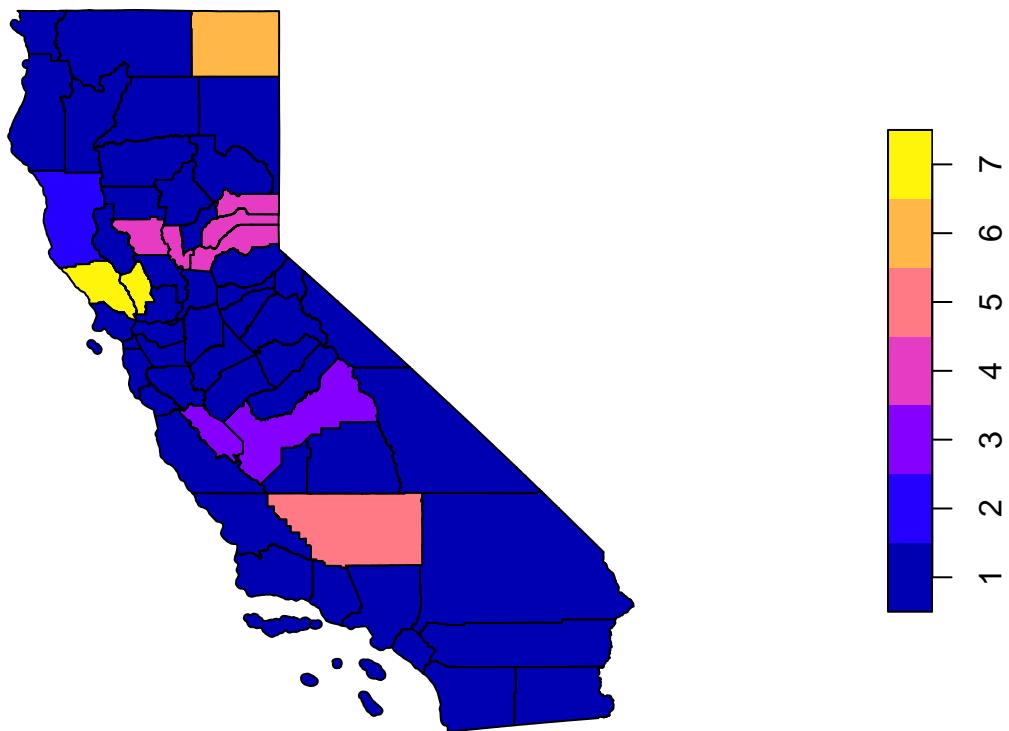
#Add a minimum number of areas in each cluster constraint
clus7_response_minarea = skater(edges = ct_mst[,1:2], data = response_scaled, ncuts = 6, 4)

CA_data_cluster = (CA_sf %>% mutate(clus = clus7_min$groups))

#Plot clustered CA
plot((CA_sf %>% mutate(clus = clus7_response$groups))['clus'], main = "7 cluster example")

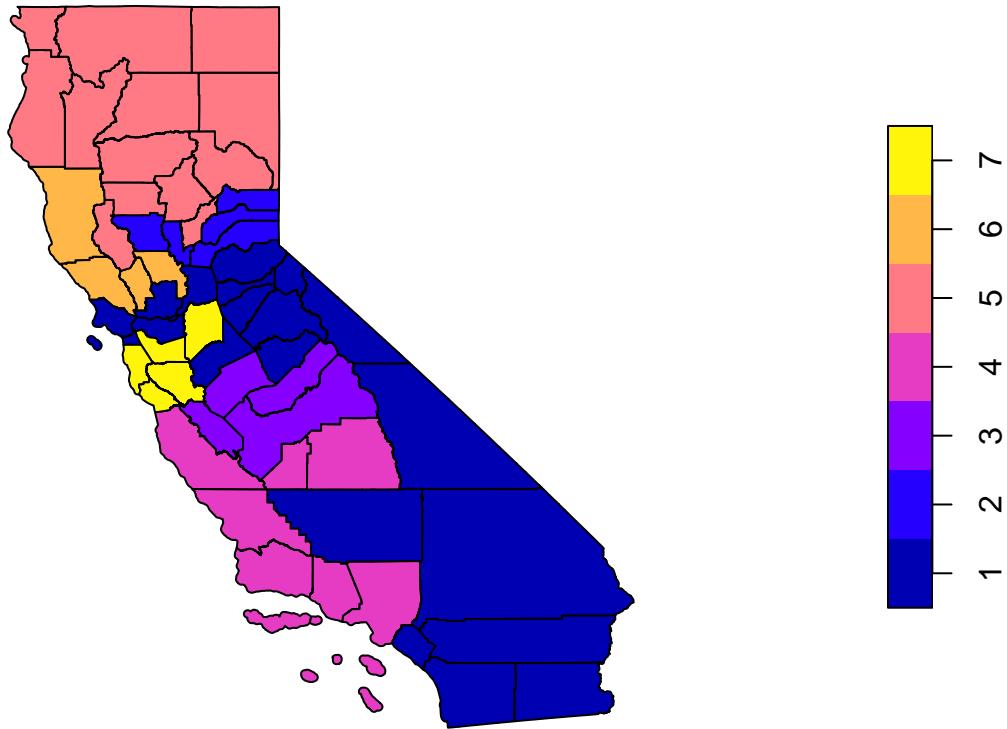
```

7 cluster example



```
plot((CA_sf %>% mutate(clus = clus7_response_min$groups))['clus'], main = "7 cluster example with popula")
```

7 cluster example with population constraint



```
plot((CA_sf %>% mutate(clus = clus7_response_minarea$groups))['clus'], main = "7 cluster example with p
```

For reference, here are the cluster labels for each county:

```
clusterlabels2 = data.frame(CA_data_cluster$NAME,clus7_response_min$groups)
names(clusterlabels2) = c("counties","Cluster")

o = order(clusterlabels2$counties)
clusterlabels2 = clusterlabels2[o,]
rownames(clusterlabels2) = NULL

clusterlabels2

##          counties Cluster
## 1          Alameda      7
## 2          Alpine      1
## 3          Amador      1
## 4          Butte       5
## 5        Calaveras      1
## 6          Colusa      2
## 7    Contra Costa      1
## 8        Del Norte      5
## 9        El Dorado      1
## 10         Fresno      3
## 11         Glenn       5
## 12        Humboldt      5
## 13        Imperial      1
```

```

## 14          Inyo      1
## 15          Kern      1
## 16          Kings     4
## 17          Lake      5
## 18          Lassen    5
## 19          Los Angeles 4
## 20          Madera    3
## 21          Marin     1
## 22          Mariposa   1
## 23          Mendocino  6
## 24          Merced    3
## 25          Modoc     5
## 26          Mono      1
## 27          Monterey   4
## 28          Napa      6
## 29          Nevada    2
## 30          Orange    1
## 31          Placer    2
## 32          Plumas    5
## 33          Riverside  1
## 34          Sacramento 1
## 35          San Benito 3
## 36          San Bernardino 1
## 37          San Diego   1
## 38          San Francisco 1
## 39          San Joaquin  7
## 40          San Luis Obispo 4
## 41          San Mateo   7
## 42          Santa Barbara 4
## 43          Santa Clara  7
## 44          Santa Cruz   7
## 45          Shasta     5
## 46          Sierra     2
## 47          Siskiyou   5
## 48          Solano     1
## 49          Sonoma     6
## 50          Stanislaus 1
## 51          Sutter     2
## 52          Tehama     5
## 53          Trinity    5
## 54          Tulare     4
## 55          Tuolumne   1
## 56          Ventura    4
## 57          Yolo       6
## 58          Yuba       5

clusterlabels$Cluster == clusterlabels2$Cluster

## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE
## [13] FALSE TRUE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE
## [37] FALSE FALSE
## [49] FALSE FALSE
```

HUGE clustering on clusters grouped by response

```
#Aggregate feature vectors into one vector for each SKATER cluster
clustered_response = cluster_mortality_total

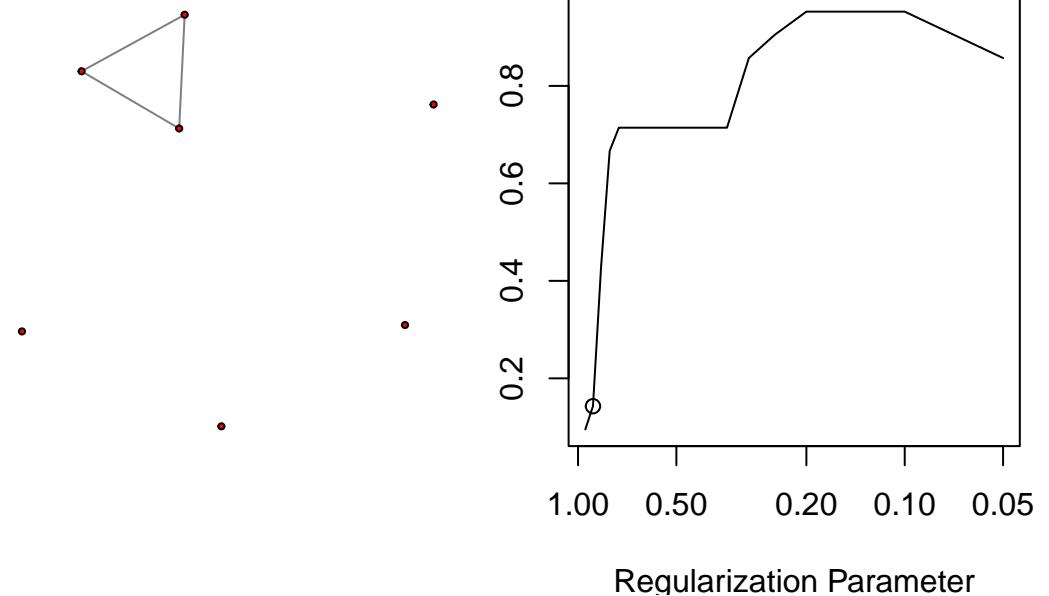
#Graph learning w HUGE
out.glasso = huge(as.matrix(clustered_response),lambda = seq(0.95,0.05,by=-0.05),method="glasso")

## Conducting the graphical lasso (glasso) wtih lossless screening....in progress: 5%Conducting the graph
## Conducting the graphical lasso (glasso)....done.
glasso.stars = huge.select(out.glasso,criterion = "stars",stars.thresh = 0.1)

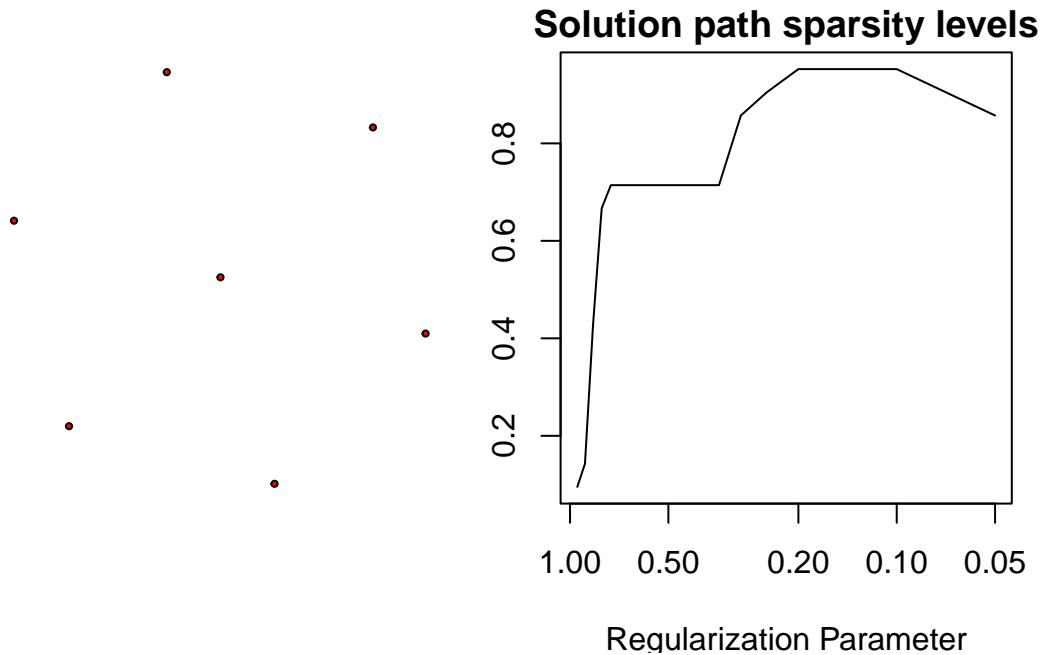
## Conducting Subsampling....in progress:5% Conducting Subsampling....in progress:10% Conducting Subsampl
glasso.ric = huge.select(out.glasso,criterion = "ric")

## Conducting rotation information criterion (ric) selection....done
## Computing the optimal graph....done
glasso.ebic = huge.select(out.glasso,criterion = "ebic")

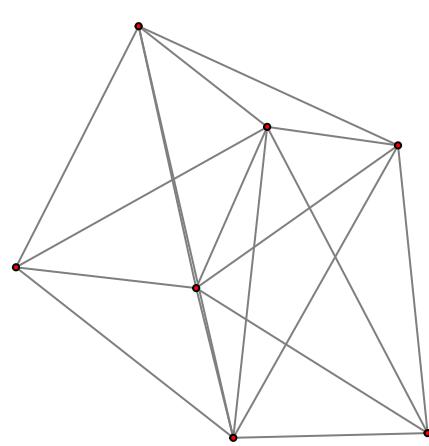
## Conducting extended Bayesian information criterion (ebic) selection....done
plot(glasso.stars)
```



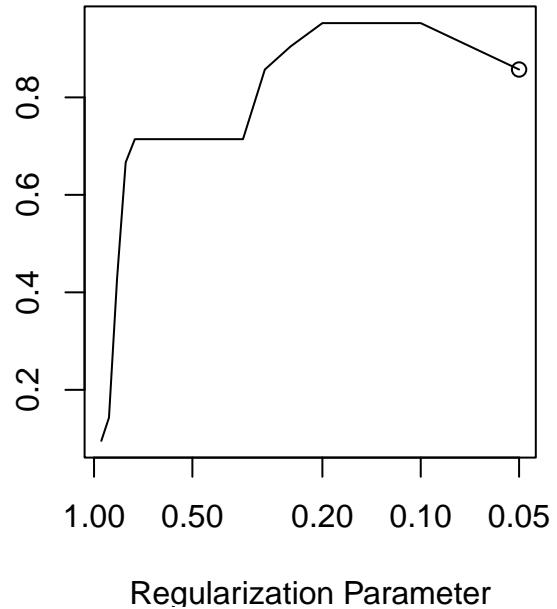
```
plot(glasso.ric)
```



```
plot(glasso.ebic)
```



Solution path sparsity levels



```
huge.est2 = glasso.ebic$refit
huge.est2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    0    1    1    1    1    0    1
## [2,]    1    0    1    1    1    1    0
## [3,]    1    1    0    1    1    1    1
## [4,]    1    1    1    0    1    1    1
## [5,]    1    1    1    1    0    1    1
## [6,]    0    1    1    1    1    0    0
## [7,]    1    0    1    1    1    0    0
```

#Identify which clusters/nodes are the most connected on the graph i.e. has the most association with t

```
degree_connectivity = data.frame(colSums(huge.est2))
colnames(degree_connectivity) = "node_connections"
degree_connectivity = cbind(c(1:num_clus),degree_connectivity)
```

Calculate graph filter based on response SKATER and HUGE results

```
A = as.matrix(huge.est2)
```

```
p = nrow(A)
```

#obtain graph Laplacian L

```
D = diag(p)
```

```
for (i in 1:p){
```

```
  d = sum(A[,i])
```

```

    D[i,i] = d
}

L = D - A

#eigendecomposition of L
Ldecomp = eigen(L)
U = as.matrix(Ldecomp$vectors)
Lambdas = Ldecomp$values

#quantile(Lambdas,2/3)
transformed.L = cutoff.transform(Lambdas,2/3)
eta.L = diag(p)*transformed.L

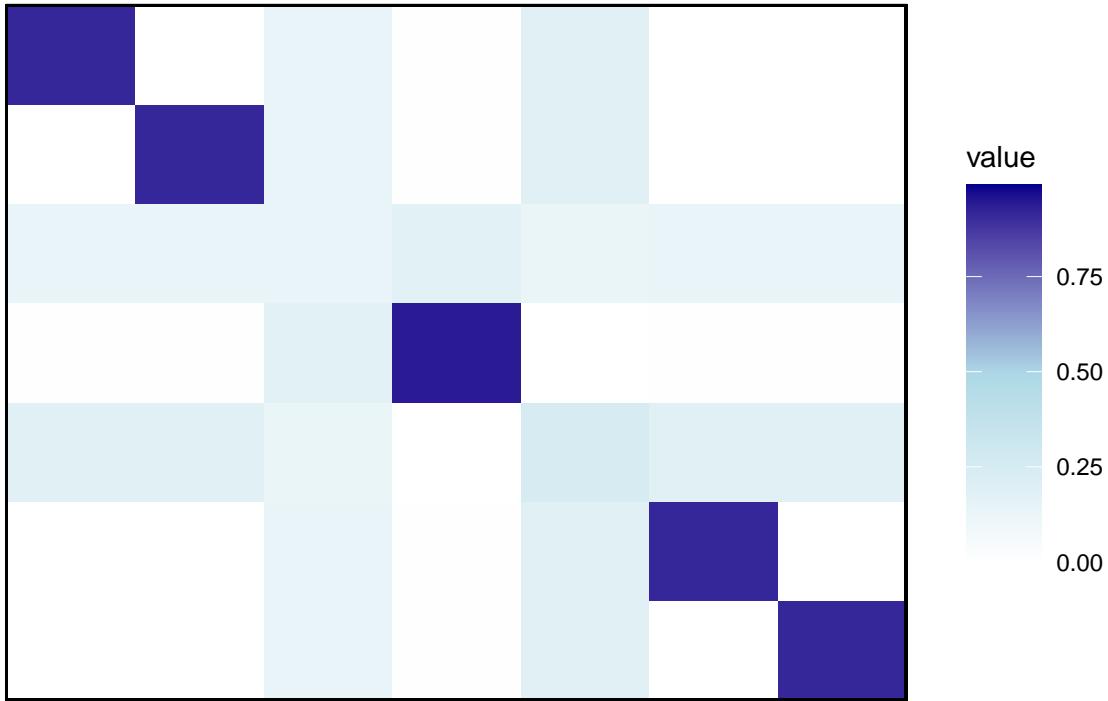
#obtain graph filter
H_response = U %*% eta.L %*% t(U)
H_response

##          [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
## [1,]  0.915546124 -0.084453876  0.1928632  0.008358106  0.1365942 -0.084453876
## [2,] -0.084453876  0.915546124  0.1928632  0.008358106  0.1365942 -0.084453876
## [3,]  0.192863202  0.192863202  0.2530695 -0.153576040  0.1290537  0.192863202
## [4,]  0.008358106  0.008358106 -0.1535760  0.940160078  0.1799835  0.008358106
## [5,]  0.136594198  0.136594198  0.1290537  0.179983539  0.1445859  0.136594198
## [6,] -0.084453876 -0.084453876  0.1928632  0.008358106  0.1365942  0.915546124
## [7,] -0.084453876 -0.084453876  0.1928632  0.008358106  0.1365942 -0.084453876
##          [,7]
## [1,] -0.084453876
## [2,] -0.084453876
## [3,]  0.192863202
## [4,]  0.008358106
## [5,]  0.136594198
## [6,] -0.084453876
## [7,]  0.915546124

matrix_heatmap(H_response,title="")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```



Calculating response similarity kernel (need to perform hyperparameter learning)

```

response_kernel = function(data = clustered_response,time_span,rho_rbf,rho_periodic,sigma2){
  K_response = matrix(0,nrow=time_span,ncol=time_span)
  rownames(data) = NULL
  i = 1
  j = 1

  for(t1 in 1:time_span){
    for (t2 in 1:time_span){
      A = data[t1,]
      B = data[t2,]

      ABtest = as.numeric((A-B)^2) #7 clusters * 8 measurements
      # K_response[i,j] = exp(-sum(ABtest)) / (2*rho_rbf)) * sigma2

      # K_response[i,j] = exp(- (sum(ABtest))) ###square this sum or remove it???
      #           / (2*rho_rbf)) * exp(- (2*sin(sum(abs(ABtest))*pi/12)^2)
      #           / (rho_periodic)) * sigma2

      K_response[i,j] = exp(- (mean(ABtest))) ###mean or sum???
      / (2*rho_rbf)) * exp(- (2*sin(sum(abs(ABtest))*pi/12)^2)
      / (rho_periodic)) * sigma2
    }
  }
}
  
```

```

        j = j+1
    }

    j = 1
    i = i+1
}

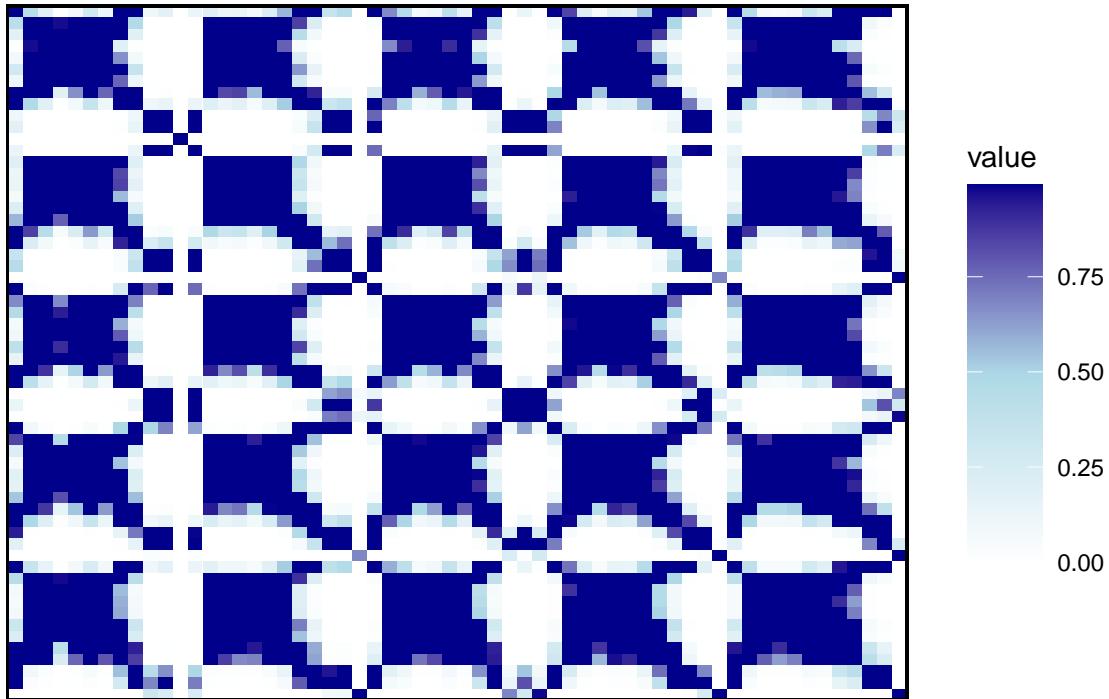
return(K_response)
}

K_response = response_kernel(time_span=60,rho_rbf=1000,rho_periodic=2135,sigma2=2.5)
matrix_heatmap(K_response,title="Response similarity kernel")

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

Response similarity kernel



Comparing DIC and WAIC between models

With INLA, we can obtain the deviance information criterion (DIC) and the widely applicable (or Watanabe-Akaike) information criterion (WAIC) which have the following formulas:

$DIC = \bar{D} + p_D$ where the first term is the posterior mean deviance i.e., a measure of fit $\bar{D} = E_{\theta|y}[D(\theta)]$ and the second term is the effective number of parameters i.e. a measure of model complexity $p_D = E_{\theta|y}[D(\theta)] - D(E_{\theta|y}[\theta]) = \bar{D} - D(\bar{\theta})$

where $D(\theta) = -2\log(p(y|\theta))$

$WAIC = T_n + \frac{V_n}{n}$ where $T_n = -\frac{1}{n} \sum_{i=1}^n \log p^*(Y_i|w)$ and $V_n = \sum_{i=1}^n \{E_w[(\log p(Y_i|w))^2] - E_w[\log p(Y_i|w)]^2\}$

where T_n is the log loss function and w are the parameters in our model.

Also note that for both criteria, the smaller the value, the better the model

```
infocrit_table = matrix(nrow = 8, ncol = 2)

dics = c(ref_model1DIC, ref_model2DIC, kgr_model1DIC,
         kgr_model2DIC, kgr_model3DIC, kgr_model4DIC, kgr_model5DIC)

waics = c(ref_model1WAIC, ref_model2WAIC, kgr_model1WAIC,
           kgr_model2WAIC, kgr_model3WAIC, kgr_model4WAIC, kgr_model5WAIC)

infocrit_table = cbind(dics, waics)
colnames(infocrit_table) = c("DIC", "WAIC")
rownames(infocrit_table) = c("Poisson GLM model", "BYM model",
                           "Proposed KGR model 1", "Proposed KGR model 2",
                           "Proposed KGR model 3", "Proposed KGR model 4",
                           "Proposed KGR model 5")

infocrit_table = data.frame(infocrit_table)
infocrit_table

##          DIC      WAIC
## Poisson GLM model 2886.360 4043.573
## BYM model        2886.360 4043.573
## Proposed KGR model 1 2840.392 2818.443
## Proposed KGR model 2 2843.759 2823.320
## Proposed KGR model 3 2844.210 2823.925
## Proposed KGR model 4 2834.327 2809.223
## Proposed KGR model 5 2839.593 2815.818
```

Comparing in sample RMSE for different clusters between models

One way to compare performance between the models fit above is to calculate RMSEs for each model's fit on each cluster's time series. Since INLA makes predictions based on the posterior predictive distribution, I actually calculated two sets of RMSEs. The first one is the RMSE of the predictions made by each model on the observed training data points i.e. not months 55-60. These were the observations that the models were fit on so we would expect small discrepancies between the observed values and the posterior predictive means for those time periods. The second one is the RMSE of the predictions made by each model on the test data points i.e. months 55-60. There was a lot more variation in the RMSEs calculated for these data points obviously.

Another important thing to note here is that the RMSEs calculated for each cluster were drastically different because the population sizes between clusters varied by a lot (think thousands compared to hundred thousands). So in order to make actual comparisons, the RMSEs had to be scaled which involves dividing the calculated RMSE by the average of the actual observed data points. The resulting RMSE values for each cluster, which are presented in tables below, can now be interpreted relative to the average number of respiratory related deaths in that cluster.

```
#Overall fit
RMSE_table = matrix(nrow=8, ncol=num_clus)

for (i in 1:num_clus){
```

```

actual = inla_insample_data %>% filter(id == i) %>% select(response) %>% data.frame()

actual.mean = mean(actual$response)

pm_1 = ref_model2_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_2 = ref_model3_fvs %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_3 = kgr_model1_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_4 = kgr_model2_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_5 = kgr_model3_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_6 = kgr_model4_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_7 = kgr_model5_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_8 = kgr_model2_nb_fit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()

rmse1 = sqrt(mean((actual.mean - pm_1$mean)^2))
rmse2 = sqrt(mean((actual.mean - pm_2$mean)^2))
rmse3 = sqrt(mean((actual.mean - pm_3$mean)^2))
rmse4 = sqrt(mean((actual.mean - pm_4$mean)^2))
rmse5 = sqrt(mean((actual.mean - pm_5$mean)^2))
rmse6 = sqrt(mean((actual.mean - pm_6$mean)^2))
rmse7 = sqrt(mean((actual.mean - pm_7$mean)^2))
rmse8 = sqrt(mean((actual.mean - pm_8$mean)^2))

RMSE_table[,i] = c(rmse1,rmse2,rmse3,rmse4,rmse5,rmse6,rmse7,rmse8)
RMSE_table[,i] = RMSE_table[,i] / actual.mean
}

#Table 1: In sample RMSE
RMSE_table = data.frame(RMSE_table)

colnames(RMSE_table) = c("Cluster 1","Cluster 2","Cluster 3","Cluster 4",
                        "Cluster 5","Cluster 6","Cluster 7")
rownames(RMSE_table) = c("BYM model","LGCP model","Proposed KGR model 1",
                        "Proposed KGR model 2","Proposed KGR model 3",
                        "Proposed KGR model 4","Proposed KGR model 5", "NB model")

RMSE_table

##                                     Cluster 1 Cluster 2 Cluster 3 Cluster 4 Cluster 5
## BYM model                  0.2293690 0.2294094 0.2294130 0.2293738 0.2294248
## LGCP model                 0.2473416 0.1863686 0.1849034 0.2675127 0.2291370
## Proposed KGR model 1      0.2480194 0.2309662 0.2296718 0.2568028 0.2362070
## Proposed KGR model 2      0.2478003 0.2304885 0.2294322 0.2565523 0.2362671
## Proposed KGR model 3      0.2477991 0.2305797 0.2296181 0.2561972 0.2363297
## Proposed KGR model 4      0.2483933 0.2309104 0.2281534 0.2595734 0.2351330
## Proposed KGR model 5      0.2482920 0.2295268 0.2270737 0.2599086 0.2343074
## NB model                   0.2323153 0.2327045 0.2328153 0.2318819 0.2324671
##                                     Cluster 6 Cluster 7
## BYM model                  0.229562765 0.2293662
## LGCP model                 0.001136826 0.2316173
## Proposed KGR model 1      0.222658001 0.2350362
## Proposed KGR model 2      0.222921301 0.2350937
## Proposed KGR model 3      0.223359601 0.2351279
## Proposed KGR model 4      0.218009552 0.2346100
## Proposed KGR model 5      0.218482566 0.2344733

```

```
## NB model          0.233203487 0.2333276
```

OUT OF SAMPLE FITTING (FORECASTING) ANALYSIS

For out of sample model fitting, we now use `inla_outsample_data` which has $t=60$ now instead of $t=54$. Recall that the response values for $t=55, \dots, 60$ are NA in order for INLA to make posterior predictive predictions.

Plots of true mortality values

```
true_mortality = inla_full_data
true_mortality$time = as.numeric(true_mortality$time)

#Combine plots with library patchwork
true1 = true_mortality %>% filter(id == 1) %>% ggplot(aes(x=time,y=response)) + geom_line()

true2 = true_mortality %>% filter(id == 2) %>% ggplot(aes(x=time,y=response)) + geom_line()

true3 = true_mortality %>% filter(id == 3) %>% ggplot(aes(x=time,y=response)) + geom_line()

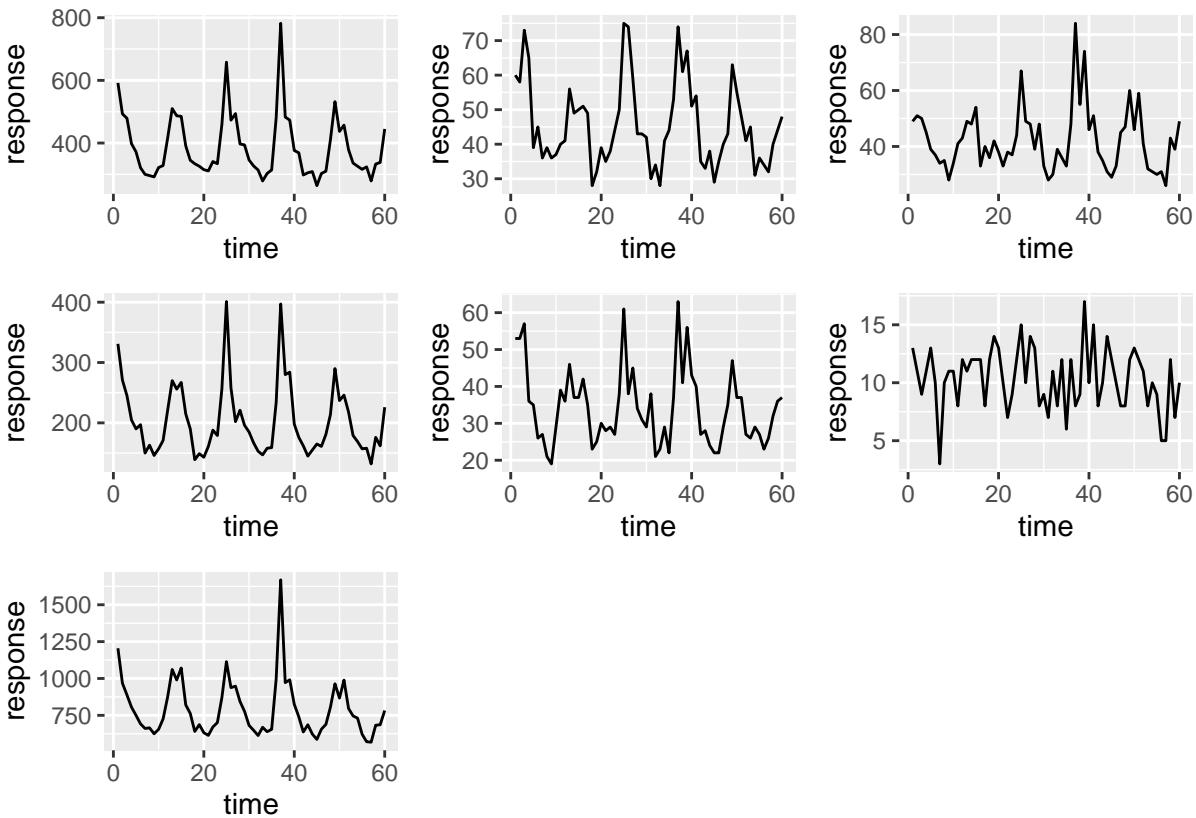
true4 = true_mortality %>% filter(id == 4) %>% ggplot(aes(x=time,y=response)) + geom_line()

true5 = true_mortality %>% filter(id == 5) %>% ggplot(aes(x=time,y=response)) + geom_line()

true6 = true_mortality %>% filter(id == 6) %>% ggplot(aes(x=time,y=response)) + geom_line()

true7 = true_mortality %>% filter(id == 7) %>% ggplot(aes(x=time,y=response)) + geom_line()

true1 + true2 + true3 + true4 + true5 + true6 + true7
```



```
#Write a function to make plot of posterior predictive estimates with credible interval bands OVERLAI
pp_outsample_plot = function(num_plots = num_clus, ref_data = inla_full_data, pred_data){
  for (i in 1:num_plots){
    df = ref_data %>% filter(id == i) %>% select(response)
    preds = pred_data %>% filter(id == i)
    df = cbind(df,preds)

    # title = sprintf("Posterior Predictive Fits for Cluster %s",i)
    title = sprintf("Cluster %s",i)

    post_pred_plot = df %>% ggplot(aes(x=time,y=response)) + geom_point() +
      geom_line(aes(y=mean),color = "red") + geom_ribbon(aes(ymax = `0.025quant`,ymin = `0.975quant`),alpha = 0.2)
    print(post_pred_plot)
  }
}

pp_outsample_plot = function(num_plots = num_clus, ref_data = inla_full_data, pred_data, save_path = "C:/Users/..."){
  for (i in 1:num_plots){
    df = ref_data %>% filter(id == i) %>% select(response)
    preds = pred_data %>% filter(id == i)
    df = cbind(df,preds)
    df$date = as.Date(paste0("2015-01-01")) + months(df$time - 1)

    test_idx = which(df$time >= 55)

    captured = (df$response[test_idx] >= df$y_pplower[test_idx] & df$response[test_idx] <= df$y_ppupper
  }
}
```

```

coverage = round(sum(captured)/length(captured)*100,2)

title = sprintf("Cluster %s (Out of Sample Coverage: %s%%)",i,coverage)

# Define tick marks
jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
all_breaks <- sort(c(jan_breaks, jul_breaks))

post_pred_plot = df %>%
  ggplot(aes(x = date, y = response)) +
  geom_point() +
  geom_line(aes(y = mean), color = "red") +
  geom_ribbon(aes(ymin = y_pplower, ymax = y_ppupper), alpha = 0.3) +
  geom_vline(xintercept = as.Date("2019-06-01"),linetype = "dashed",color = "blue",linewidth = 1.5)
  ggttitle(title) +
  scale_x_date(
    name = "Year",
    breaks = all_breaks,
    labels = ifelse(format(all_breaks, "%m") == "01",
                   format(all_breaks, "%Y"), ""),
    expand = expansion(mult = 0.02) # ~2% space on each side
  ) +
  scale_y_continuous(name = "Respiratory Related Deaths") +
  theme_classic(base_size = 14)

print(post_pred_plot)

# Save plot automatically with prefix
ggsave(
  filename = paste0(prefix, "cluster", i, ".png"),
  plot = post_pred_plot,
  path = save_path,
  width = 8,
  height = 6,
  dpi = 300
)
}

}

pp_outsample_plot_combined = function(num_plots = num_clus,
                                      ref_data = inla_full_data,
                                      pred_data,
                                      save_path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Infer
prefix = ""){

plot_list = list()

for (i in 1:num_plots){
  df = ref_data %>% filter(id == i) %>% select(response)
  preds = pred_data %>% filter(id == i)
  df = cbind(df,preds)
  df$date = as.Date("2015-01-01") + months(df$time - 1)

  # test indices
}
}

```

```

test_idx = which(df$time >= 55)

# coverage calc
captured = (df$response[test_idx] >= df$y_pplower[test_idx] &
            df$response[test_idx] <= df$y_ppupper[test_idx])
coverage = round(mean(captured) * 100, 2)

title = sprintf("Cluster %s (Out of Sample Coverage: %s%%)", i, coverage)

# tick marks
jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
all_breaks <- sort(c(jan_breaks, jul_breaks))

p = df %>%
  ggplot(aes(x = date, y = response)) +
  geom_point(size = 1) +
  geom_line(aes(y = mean), color = "red") +
  geom_ribbon(aes(ymin = y_pplower, ymax = y_ppupper), alpha = 0.3) +
  geom_vline(xintercept = as.Date("2019-06-01"),
             linetype = "dashed", color = "blue", linewidth = 1.2) +
  ggtitle(title) +
  scale_x_date(
    name = "Year",
    breaks = all_breaks,
    labels = ifelse(format(all_breaks, "%m") == "01",
                   format(all_breaks, "%Y"), ""),
    expand = expansion(mult = 0.02)
  ) +
  scale_y_continuous(name = "Deaths") +
  theme_classic(base_size = 14)

plot_list[[i]] = p
}

# combine into a grid (adjust ncol if needed)
combined = wrap_plots(plotlist = plot_list, ncol = 2)

# save one combined figure
ggsave(
  filename = paste0(prefix, "combined.png"),
  plot = combined,
  path = save_path,
  width = 12,
  height = 8,
  dpi = 600
)
}

```

Reference model 1

```

#Run model
ref_model1_outfit = ref_model1(inla_outsample_data, a_prior=1, b_prior=5e-5)

```

```

#Posterior predictive sampling from estimated intensities
ref_model1_outfit$fitted_values = poisson_pp_sampling(ref_model1_outfit$fitted_values,n=100000)

#Extract DIC and WAIC
ref_model1_DIC = ref_model1_outfit$modelDIC
ref_model1_WAIC = ref_model1_outfit$modelWAIC

#Get summaries of parameter estimates
ref_model1_outfit$model_summary

##               mean        sd 0.025quant 0.5quant 0.975quant      mode
## months1    2.1039996 7.254715 -12.121777 2.1039996   16.32978 2.1039996
## months2    1.8618770 7.254716 -12.363902 1.8618770   16.08766 1.8618770
## months3    1.8812179 7.254716 -12.344561 1.8812179   16.10700 1.8812179
## months4    1.6930262 7.254718 -12.532756 1.6930262   15.91881 1.6930262
## months5    1.6124053 7.254718 -12.613378 1.6124053   15.83819 1.6124053
## months6    1.4969691 7.254719 -12.728816 1.4969691   15.72275 1.4969691
## months7    1.4681952 7.254722 -12.757595 1.4681952   15.69399 1.4681952
## months8    1.4344600 7.254723 -12.791331 1.4344600   15.66025 1.4344600
## months9    1.4008381 7.254723 -12.824954 1.4008381   15.62663 1.4008381
## months10   1.4724083 7.254722 -12.753382 1.4724083   15.69820 1.4724083
## months11   1.5162767 7.254722 -12.709513 1.5162767   15.74207 1.5162767
## months12   1.7814816 7.254719 -12.444302 1.7814816   16.00727 1.7814816
## Intercept1  4.2895288 7.254726 -9.936270 4.2895288   18.51533 4.2895288
## Intercept2  2.1524553 7.254748 -12.073386 2.1524553   16.37830 2.1524553
## Intercept3  2.0751099 7.254750 -12.150735 2.0751099   16.30096 2.0751099
## Intercept4  3.6523903 7.254729 -10.573413 3.6523903   17.87819 3.6523903
## Intercept5  1.8619677 7.254756 -12.363890 1.8619677   16.08783 1.8619677
## Intercept6  0.6814343 7.254831 -13.544569 0.6814343   14.90744 0.6814343
## Intercept7  5.0102689 7.254725 -9.215527 5.0102689   19.23606 5.0102689
##               kld
## months1    5.526817e-11
## months2    5.526823e-11
## months3    5.526817e-11
## months4    5.526817e-11
## months5    5.526817e-11
## months6    5.526817e-11
## months7    5.526817e-11
## months8    5.526818e-11
## months9    5.526823e-11
## months10   5.526817e-11
## months11   5.526823e-11
## months12   5.526818e-11
## Intercept1 5.526821e-11
## Intercept2 5.526820e-11
## Intercept3 5.526815e-11
## Intercept4 5.526819e-11
## Intercept5 5.526815e-11
## Intercept6 5.526815e-11
## Intercept7 5.526815e-11

ref_model1_outfit$bri_hyperpar_summary

##               mean        sd      q0.025      q0.5      q0.975      mode

```

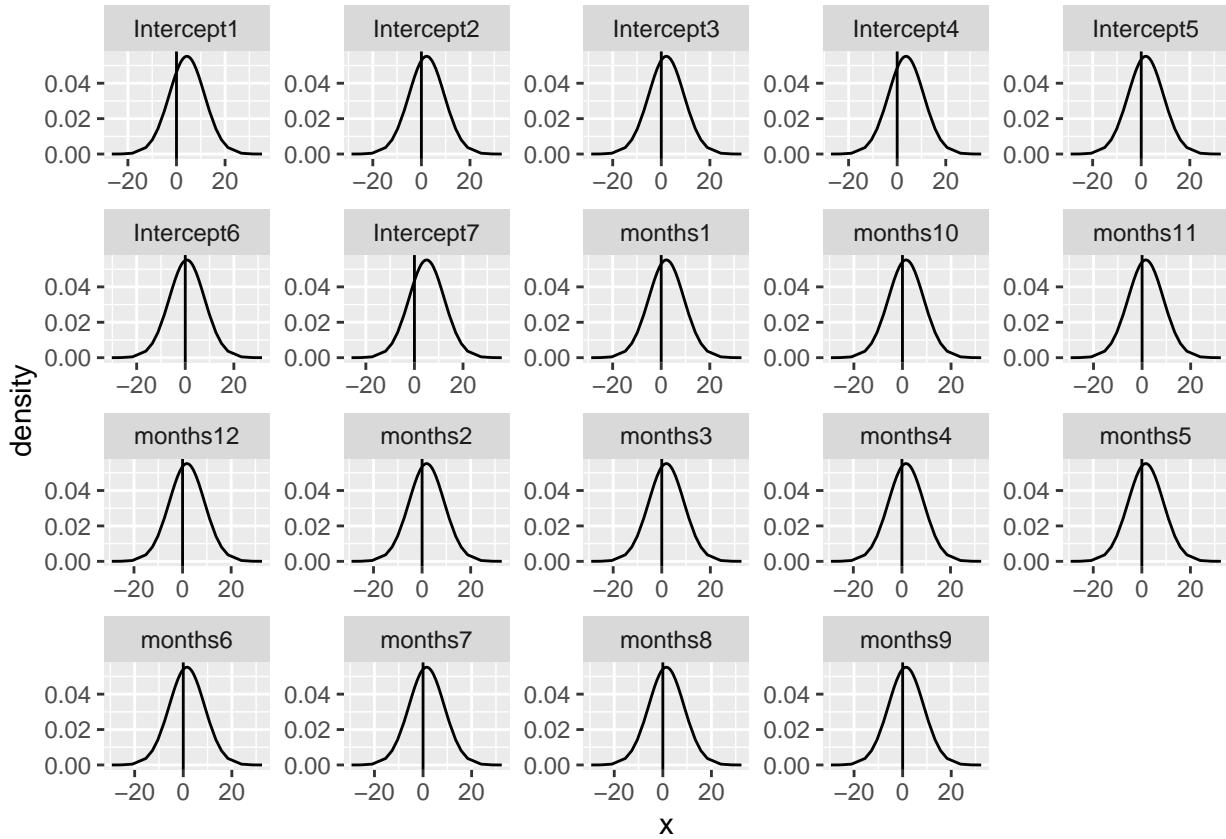
```

## SD for id 0.01149625 0.009661678 0.003662818 0.008442957 0.04015017 0.005741776
ref_model1_outfit$exp_effects

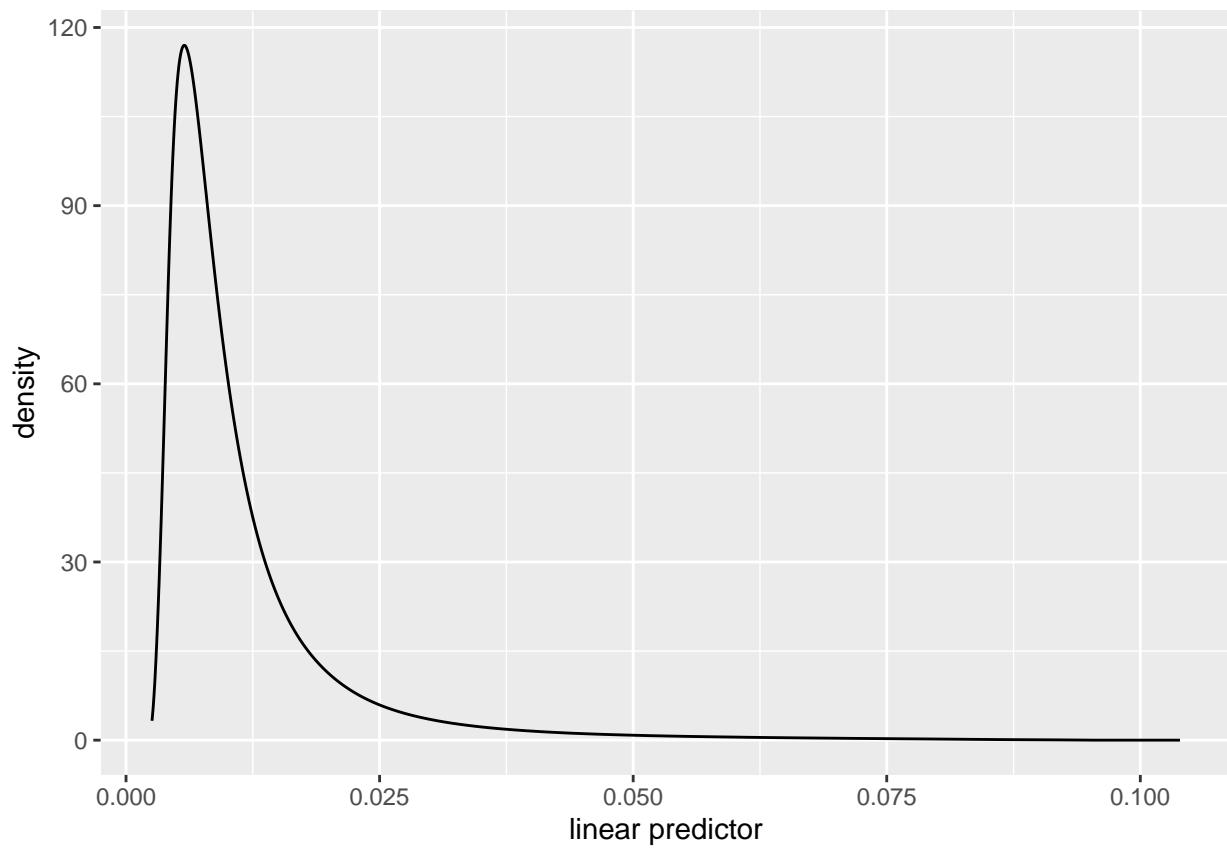
##    months1    months2    months3    months4    months5    months6    months7
## 8.198897  6.435806  6.561491  5.435906  5.014859  4.468126  4.341393
##    months8    months9   months10   months11   months12 Intercept1 Intercept2
## 4.197378  4.058600  4.359722  4.555233  5.938649  72.932093  8.605963
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
## 7.965422  38.566740  6.436389  1.976711 149.945049

#Show plots
ref_model1_outfit$param_plot

```

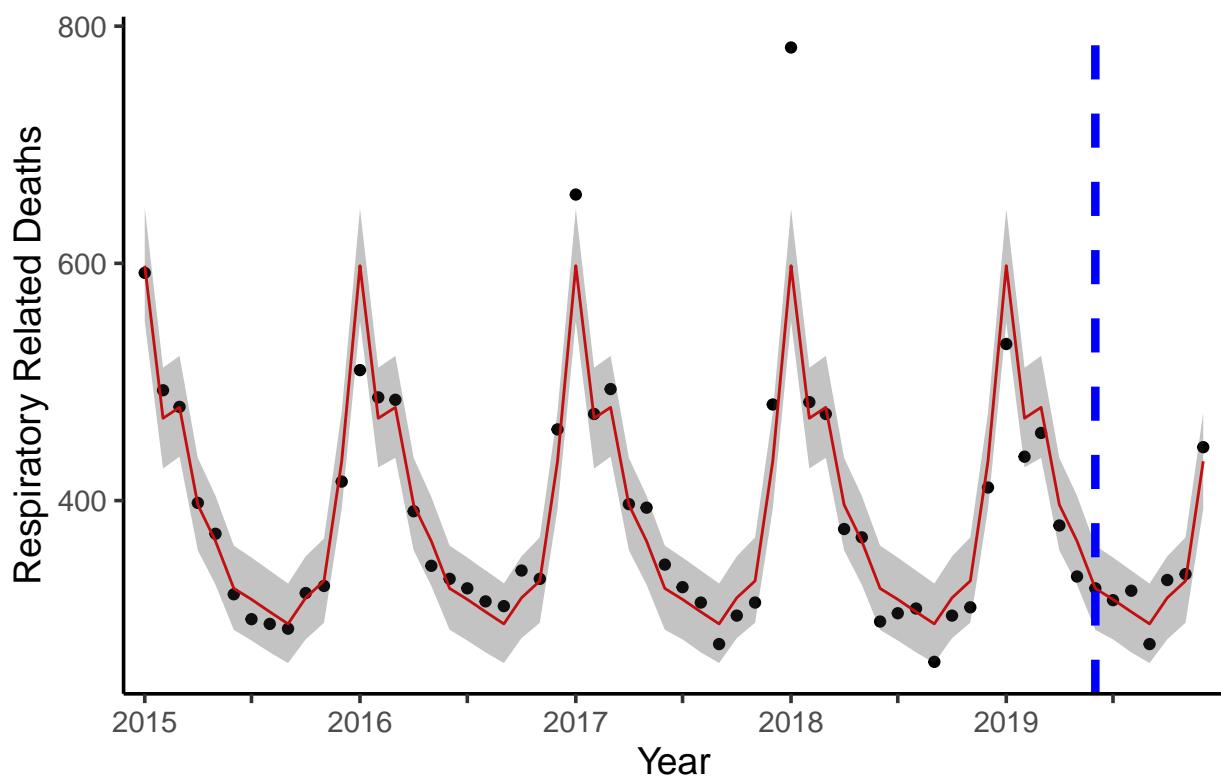


```
ref_model1_outfit$hyperparam_plot
```

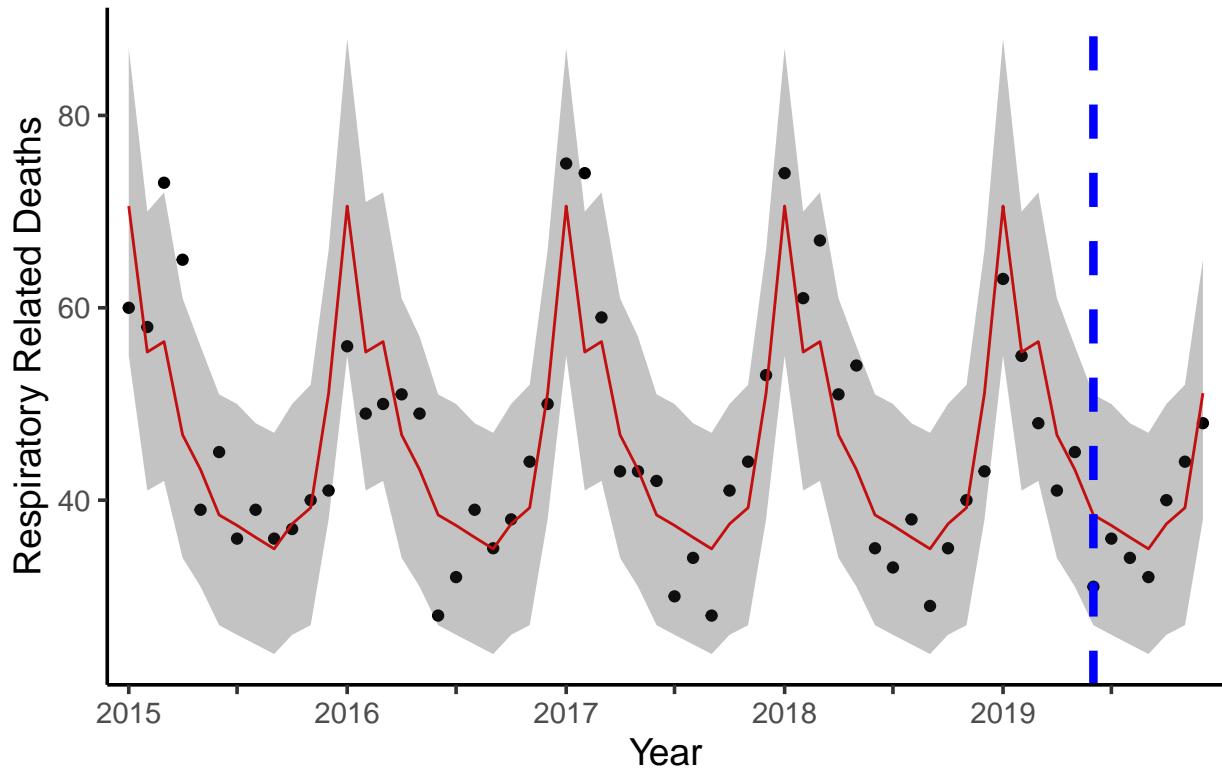


```
pp_outsample_plot(pred_data = ref_model1_outfit$fitted_values,prefix = "ref1-outsamp-")
```

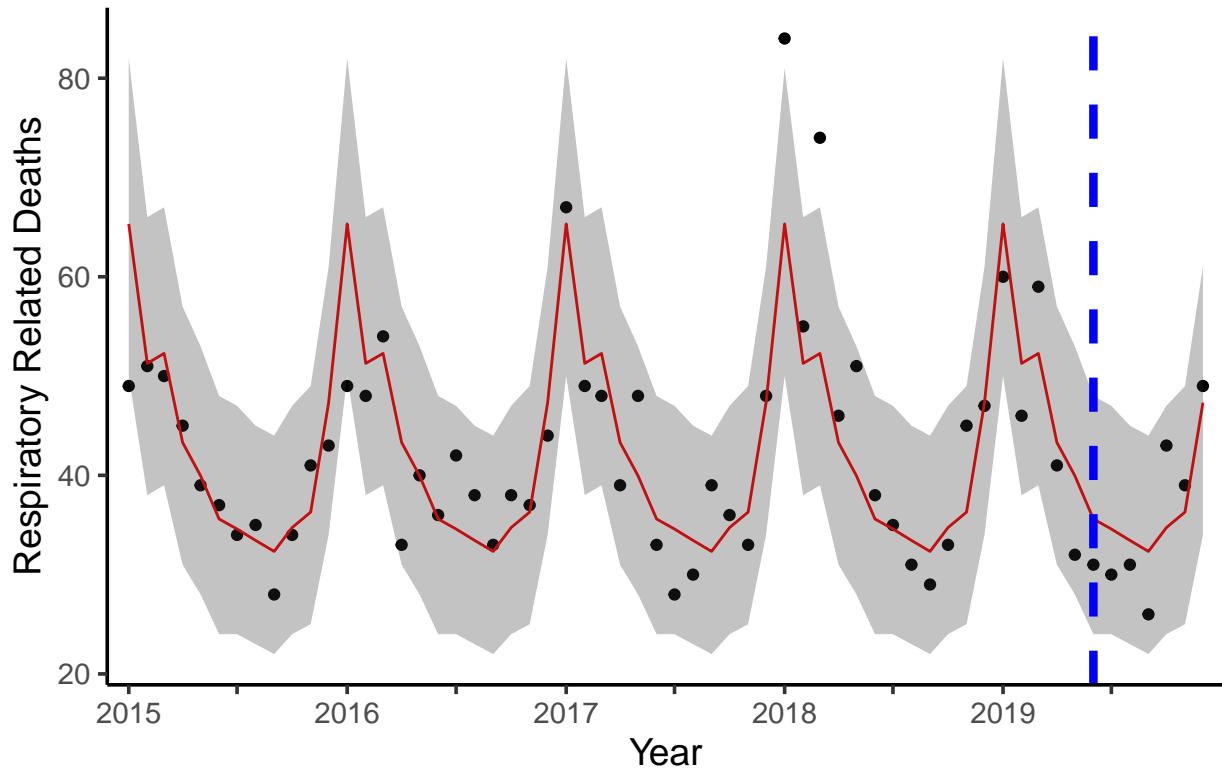
Cluster 1 (Out of Sample Coverage: 100%)



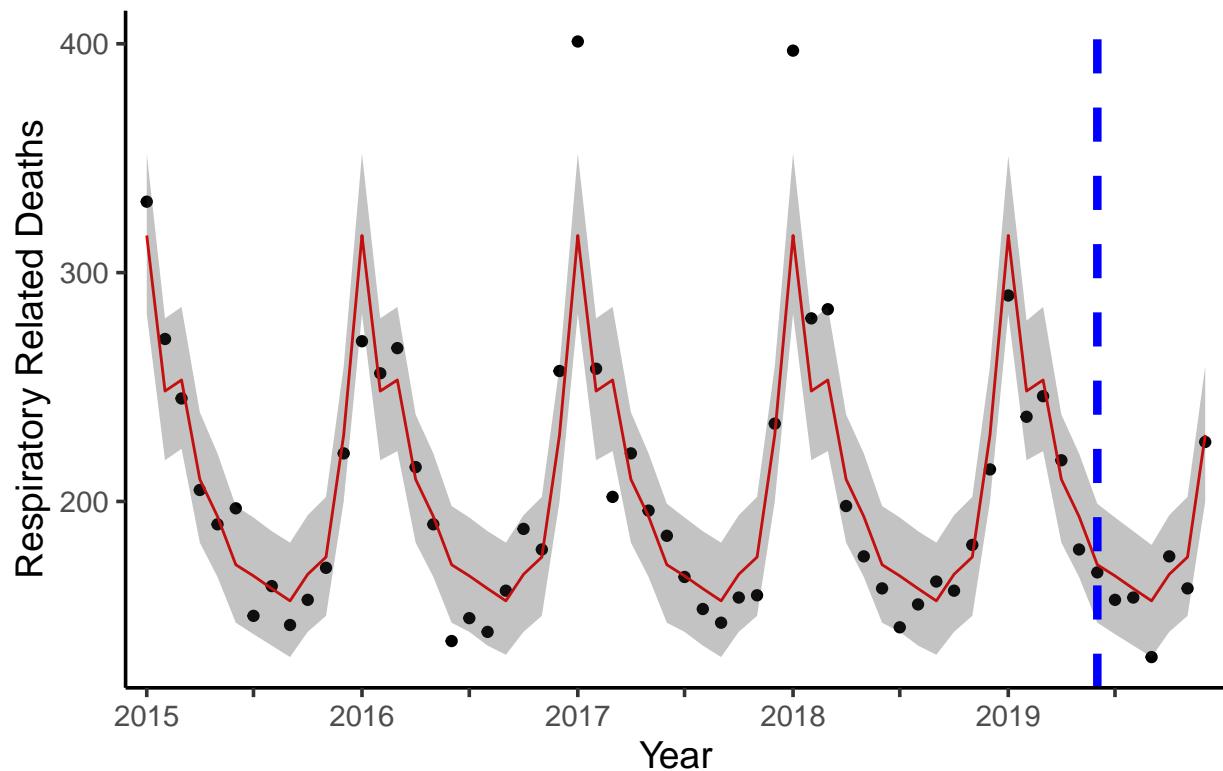
Cluster 2 (Out of Sample Coverage: 100%)



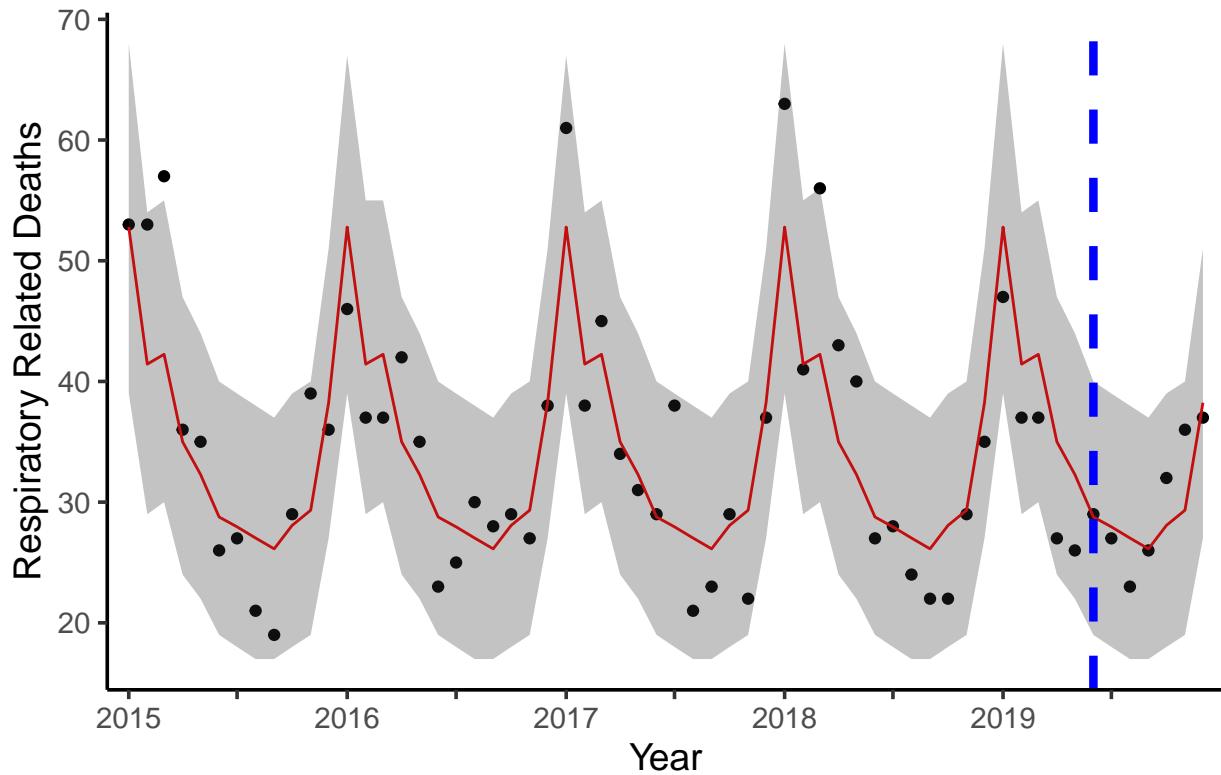
Cluster 3 (Out of Sample Coverage: 100%)



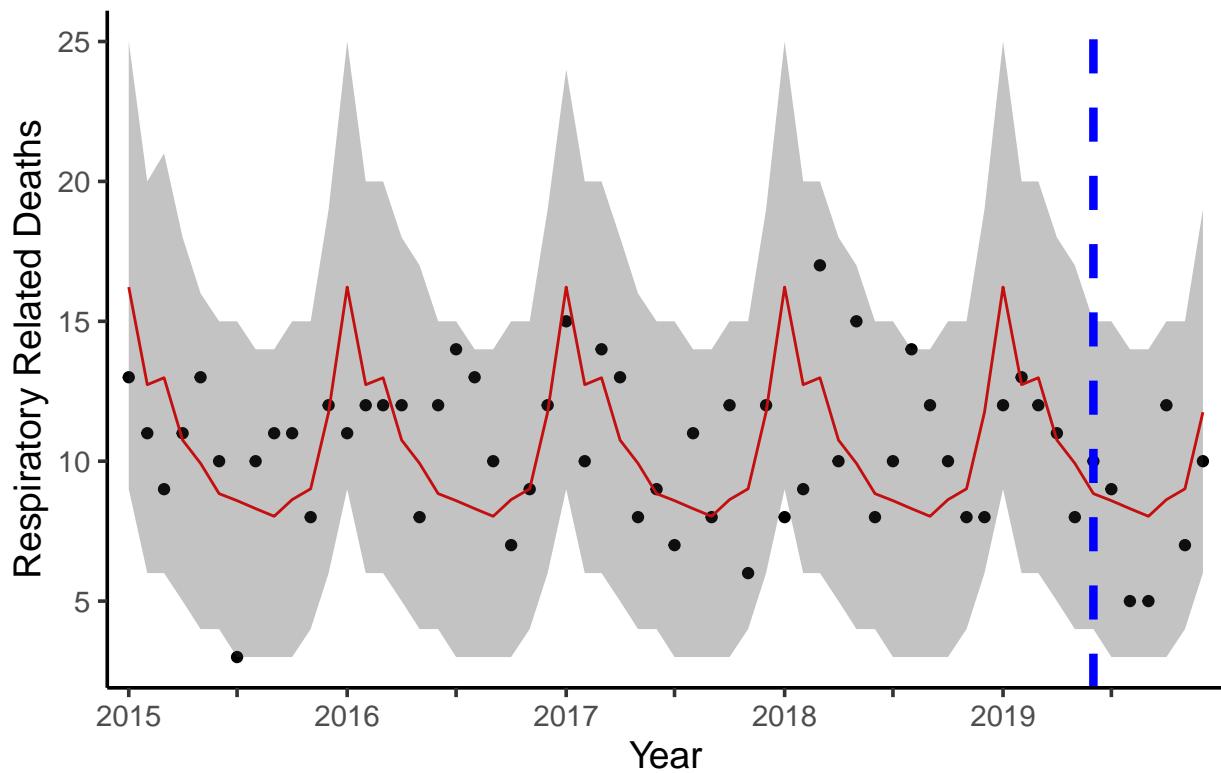
Cluster 4 (Out of Sample Coverage: 100%)



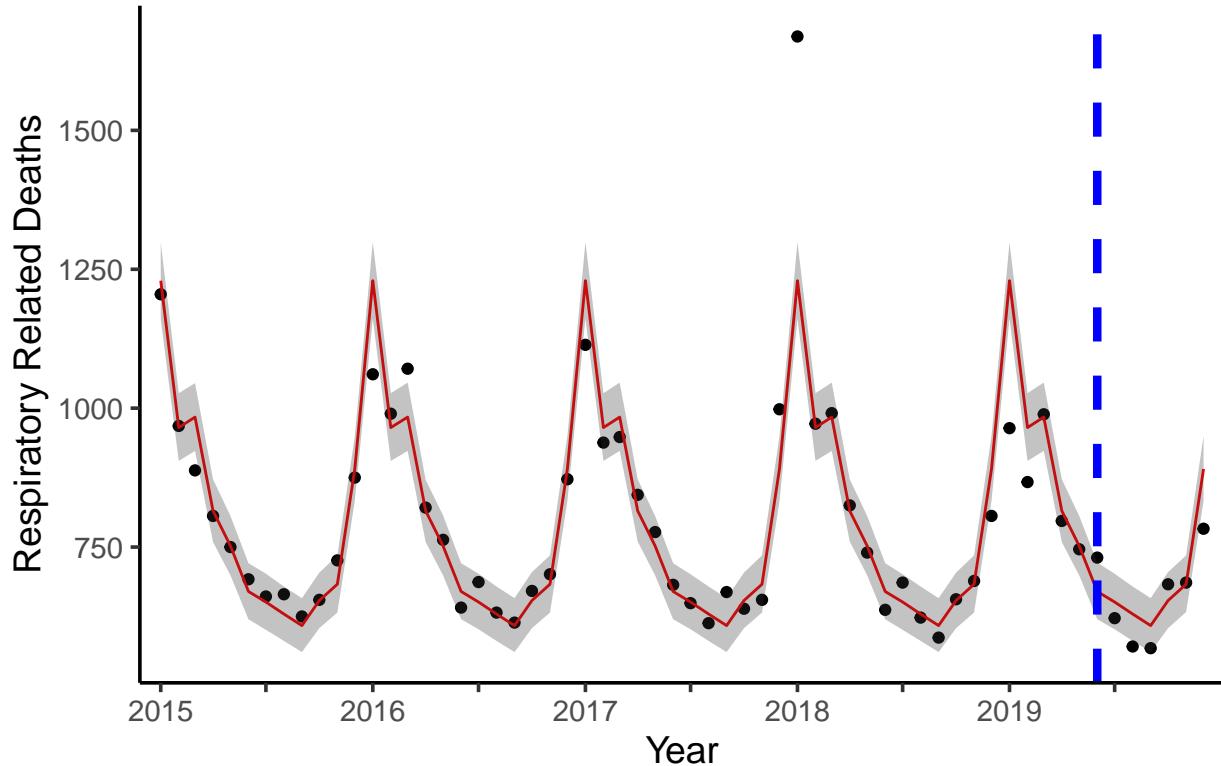
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 66.67%)



```
pp_outsample_plot_combined(pred_data = ref_model1_outfit$fitted_values, prefix = "ref1-outsample-")
```

Reference model 2

```
#Run model
ref_model2_outfit = ref_model2(dataset = inla_outsample_data,a_prec_prior = 1,b_prec_prior = 5e-4,
                                 a_phi_prior = 1,b_phi_prior = 5e-4)

#Posterior predictive sampling from estimated intensities
ref_model2_outfit$fitted_values = poisson_pp_sampling(ref_model2_outfit$fitted_values,n=100000)

#Extract DIC and WAIC
ref_model2_DIC = ref_model2_outfit$model_DIC
ref_model2_WAIC = ref_model2_outfit$model_WAIC

#Get summaries of parameter estimates
ref_model2_outfit$model_summary %>% kable %>% kable_styling()
```

	mean	sd	0.025quant	0.5quant	0.975quant	mode	kld
months1	2.1039919	7.254718	-12.121790	2.1039919	16.32977	2.1039919	0
months2	1.8618693	7.254719	-12.363915	1.8618693	16.08765	1.8618693	0
months3	1.8812102	7.254719	-12.344574	1.8812102	16.10699	1.8812102	0
months4	1.6930185	7.254720	-12.532768	1.6930185	15.91881	1.6930185	0
months5	1.6123976	7.254721	-12.613391	1.6123976	15.83819	1.6123976	0
months6	1.4969614	7.254722	-12.728829	1.4969614	15.72275	1.4969614	0

months7	1.4681875	7.254725	-12.757608	1.4681875	15.69398	1.4681875	0
months8	1.4344523	7.254725	-12.791344	1.4344523	15.66025	1.4344523	0
months9	1.4008303	7.254726	-12.824967	1.4008303	15.62663	1.4008303	0
months10	1.4724005	7.254725	-12.753395	1.4724005	15.69820	1.4724005	0
months11	1.5162689	7.254724	-12.709526	1.5162689	15.74206	1.5162689	0
months12	1.7814739	7.254721	-12.444315	1.7814739	16.00726	1.7814739	0
Intercept1	4.2895333	7.254878	-9.936564	4.2895333	18.51563	4.2895333	0
Intercept2	2.1524452	7.254894	-12.073682	2.1524452	16.37857	2.1524452	0
Intercept3	2.0750958	7.254903	-12.151049	2.0750958	16.30124	2.0750958	0
Intercept4	3.6524016	7.254893	-10.573723	3.6524016	17.87853	3.6524016	0
Intercept5	1.8619519	7.254902	-12.364192	1.8619519	16.08810	1.8619519	0
Intercept6	0.6813662	7.254984	-13.544937	0.6813662	14.90767	0.6813662	0
Intercept7	5.0102682	7.254870	-9.215813	5.0102682	19.23635	5.0102682	0

```
ref_model2_outfit$bri_hyperpar_summary %>% kable %>% kable_styling()
```

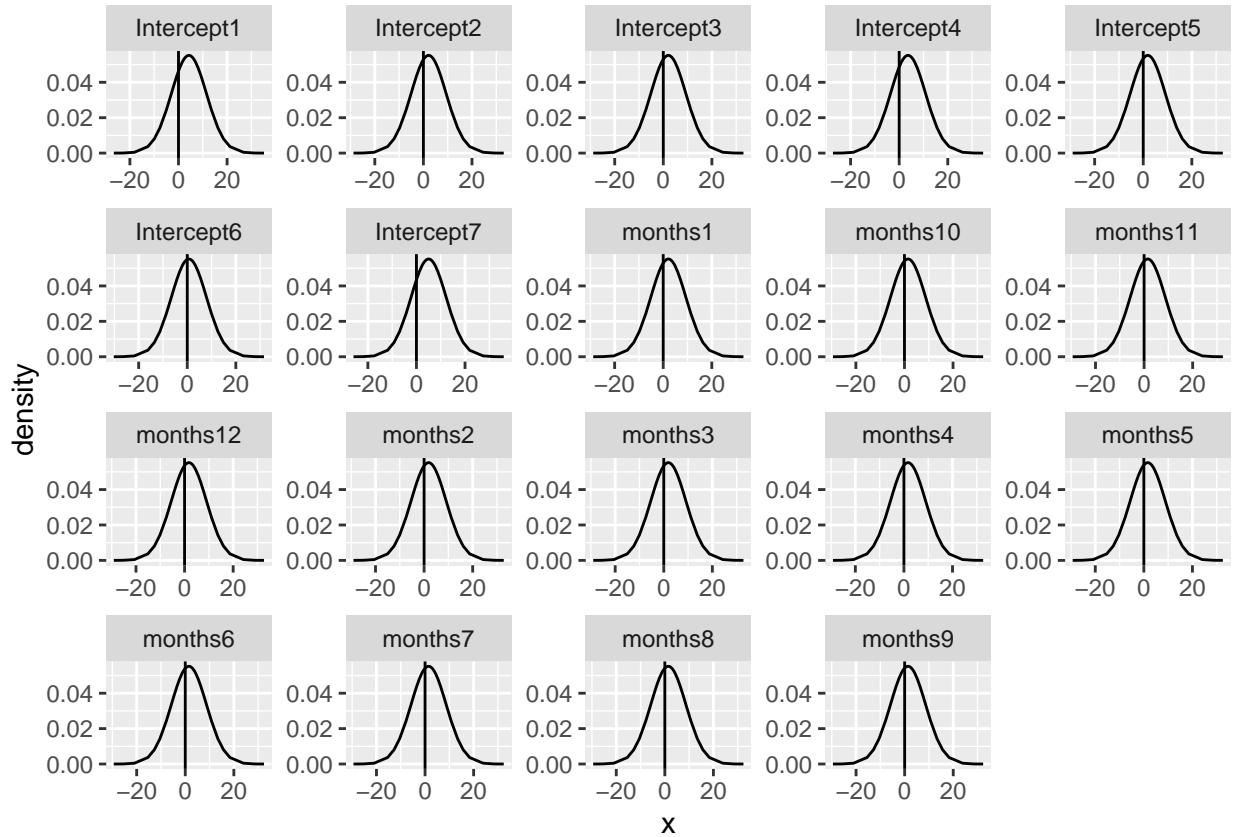
	mean	sd	q0.025	q0.5	q0.975	mode
SD for id (idd component)	0.0311780	0.0187465	0.0107997	0.0259749	0.0814358	0.0191494
SD for id (spatial component)	0.0311778	0.0187464	0.0107997	0.0259748	0.0814352	0.0191494

```
ref_model2_outfit$exp_effects
```

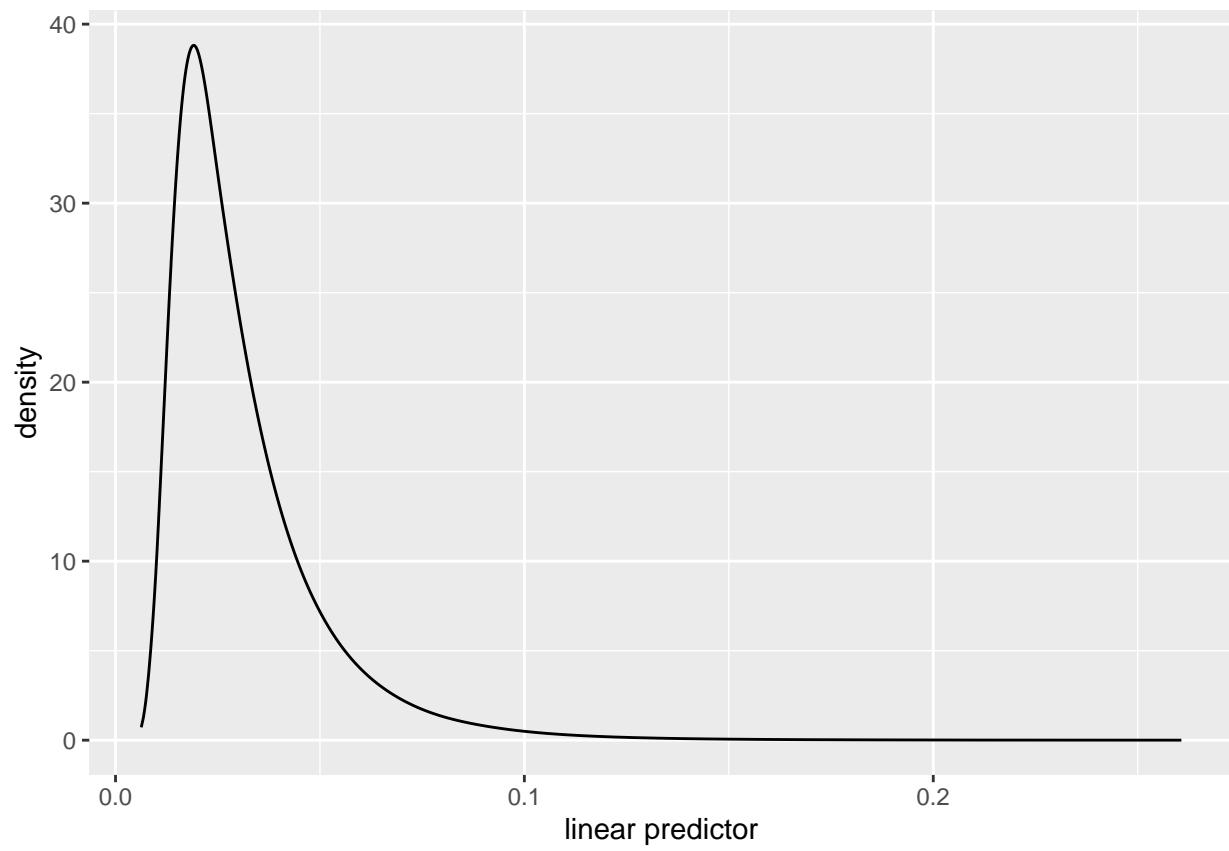
```
##   months1    months2    months3    months4    months5    months6    months7
## 8.198834  6.435756  6.561441  5.435864  5.014820  4.468092  4.341359
##   months8    months9   months10   months11   months12 Intercept1 Intercept2
## 4.197345  4.058568  4.359688  4.555198  5.938603  72.932425  8.605876
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
## 7.965310 38.567178  6.436287  1.976576 149.944949
```

#Show plots

```
ref_model2_outfit$param_plot
```

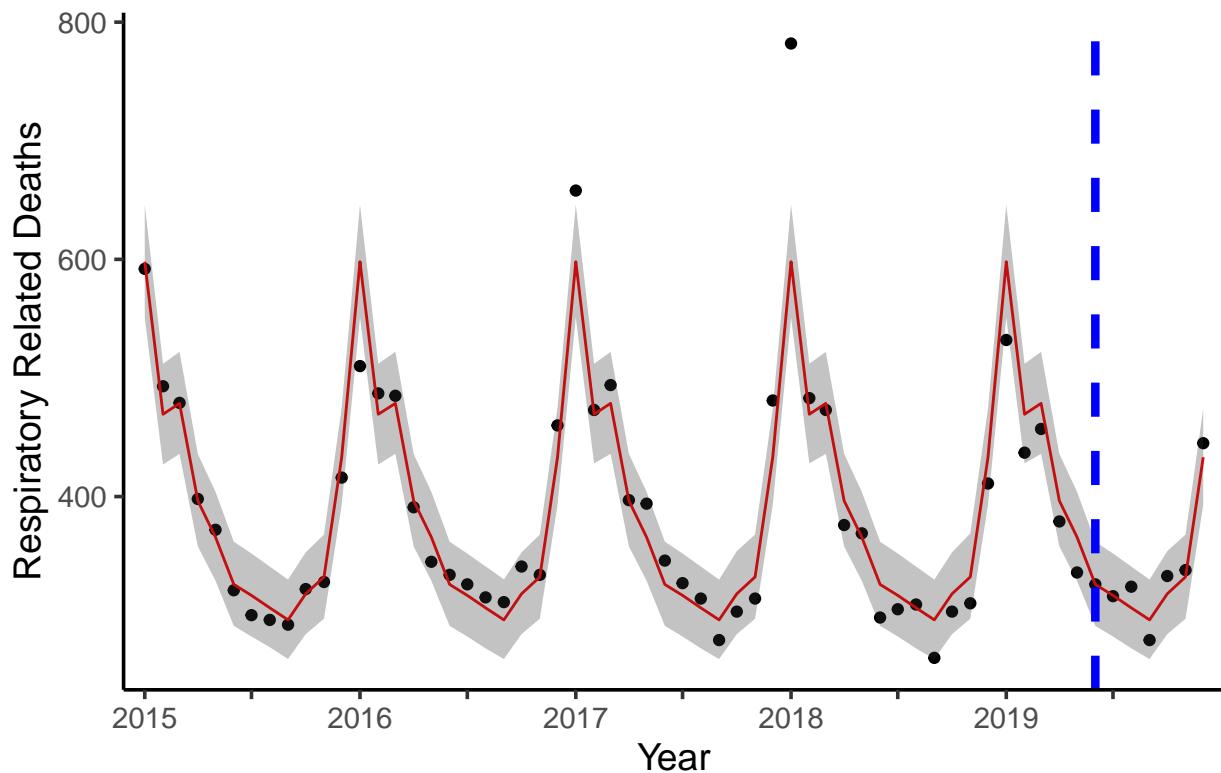


```
ref_model2_outfit$hyperparam_plot
```

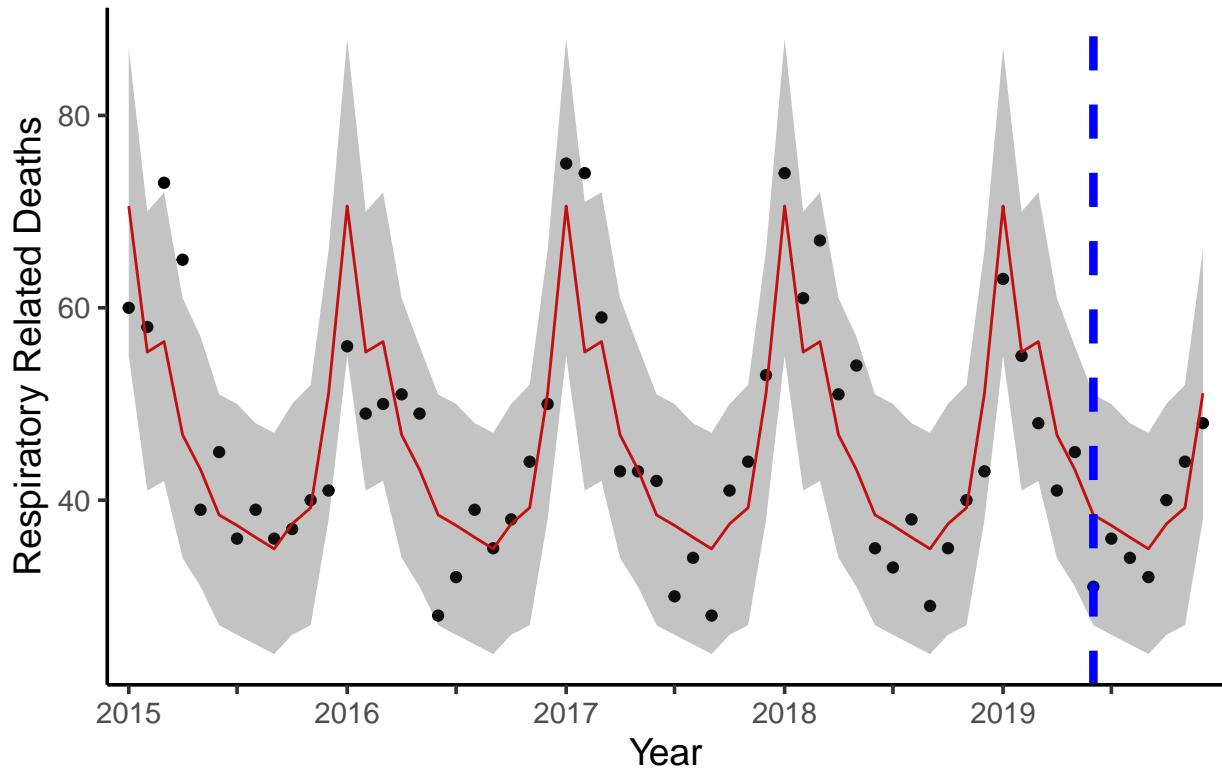


```
pp_outsample_plot(pred_data = ref_model2_outfit$fitted_values,prefix = "ref2-outsamp-")
```

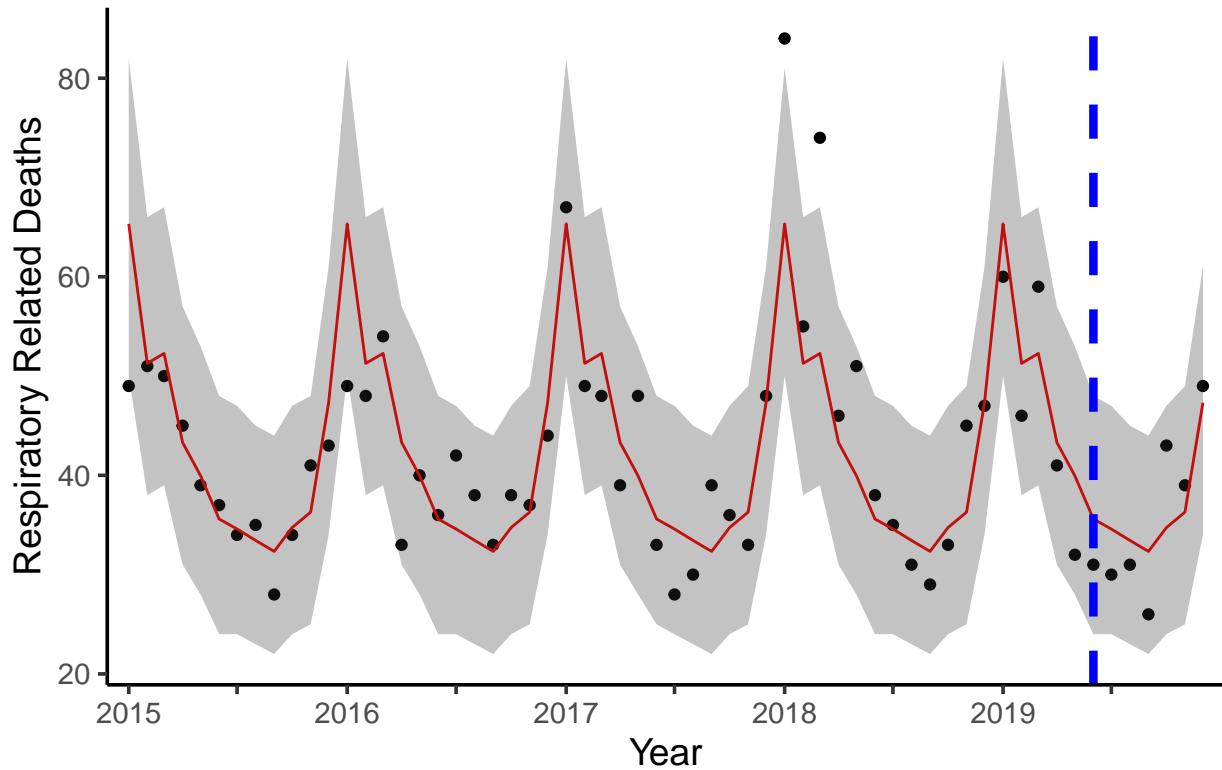
Cluster 1 (Out of Sample Coverage: 100%)



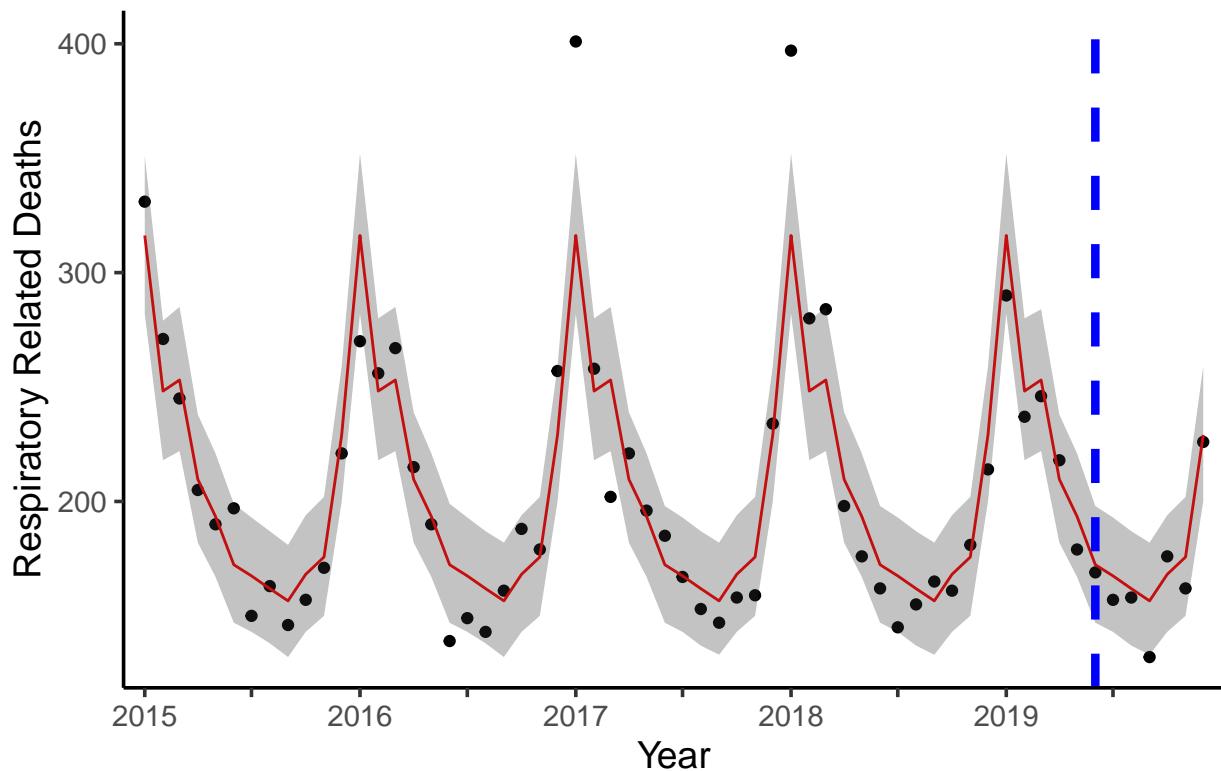
Cluster 2 (Out of Sample Coverage: 100%)



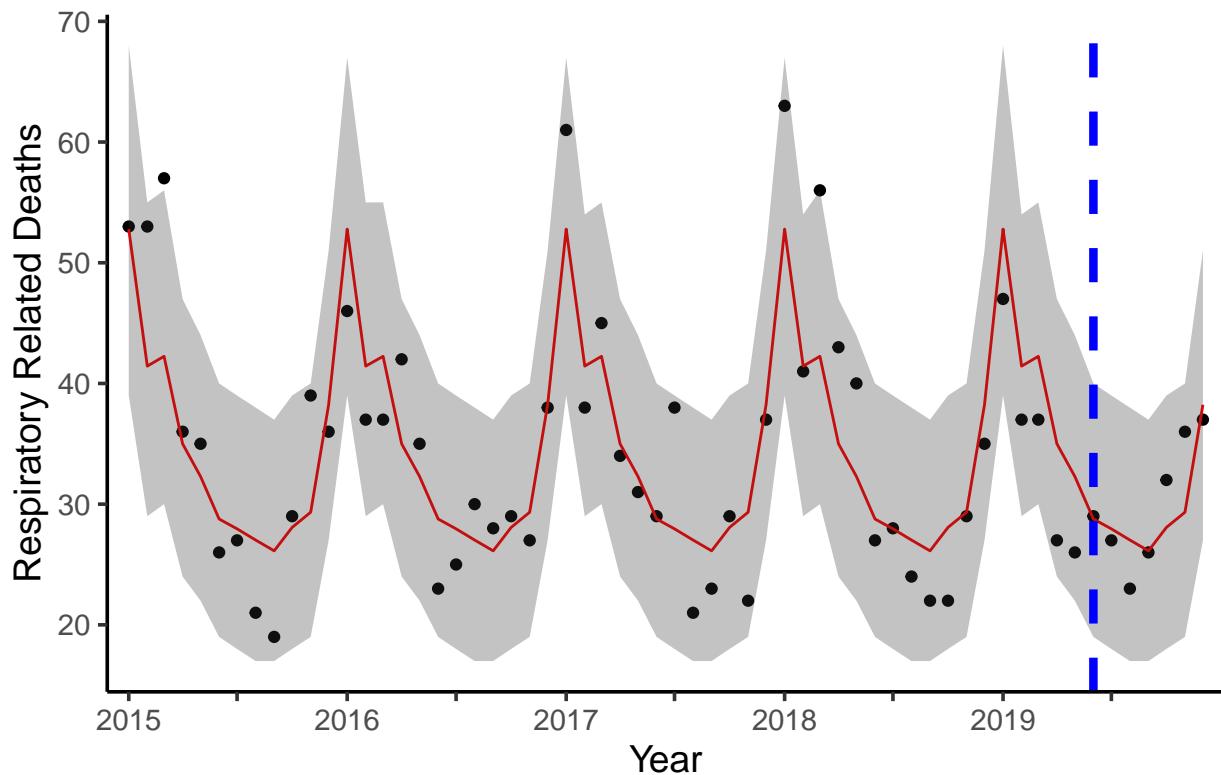
Cluster 3 (Out of Sample Coverage: 100%)



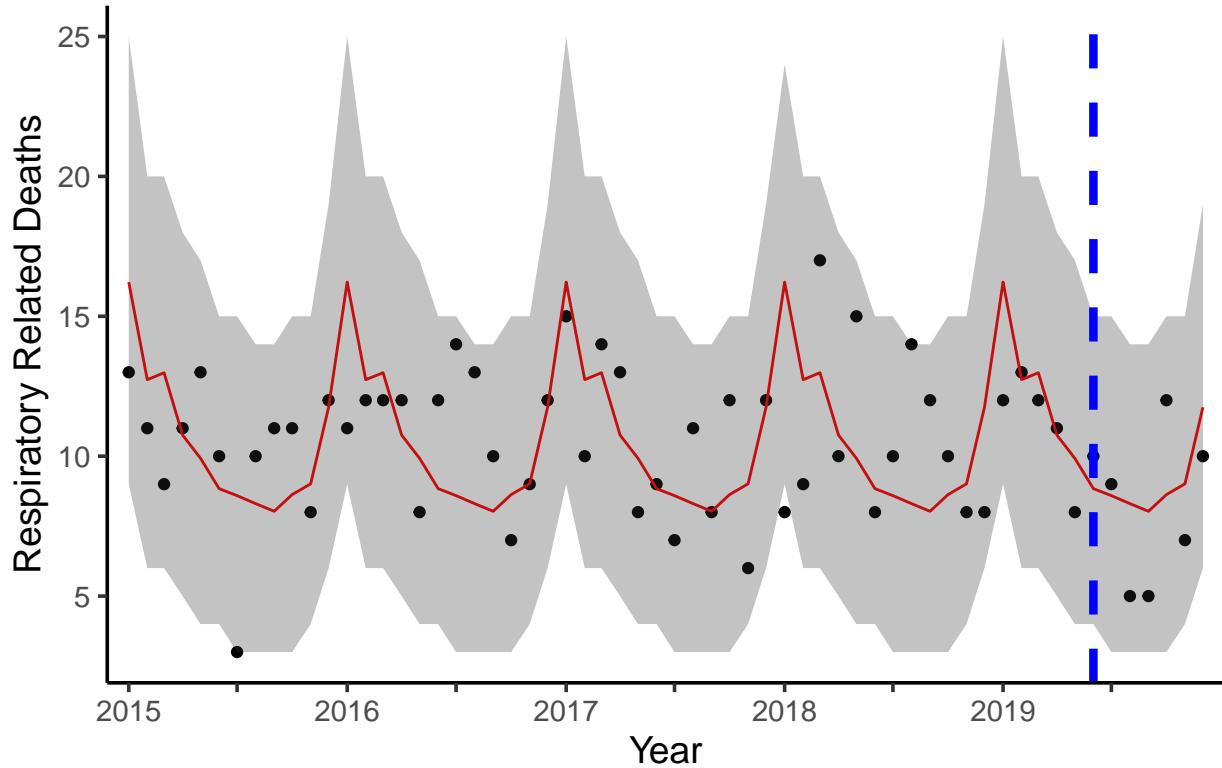
Cluster 4 (Out of Sample Coverage: 83.33%)



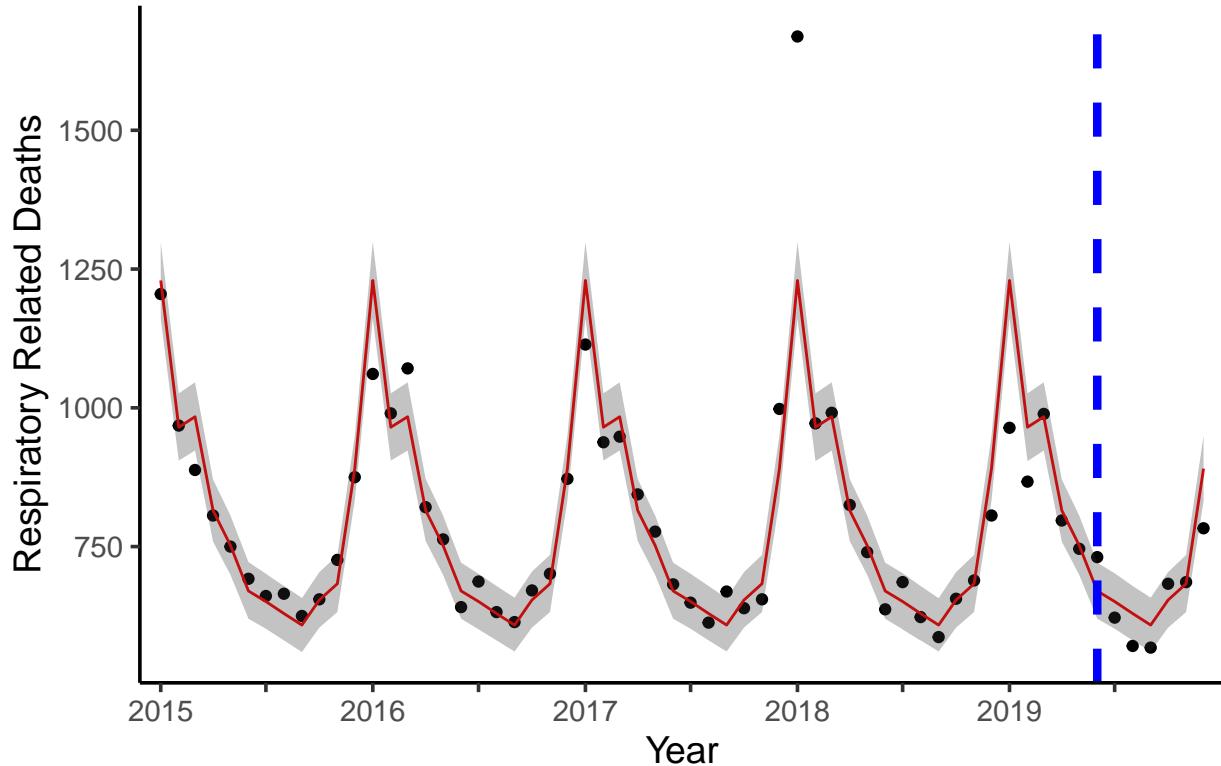
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 66.67%)



```
pp_outsample_plot_combined(pred_data = ref_model2_outfit$fitted_values, prefix = "ref2-outsample-")
```

Fitting kernel graph regression models

Proposed model 2

```
#Run model
kgr_model2_outfit = kgr_model2(data = inla_outsample_data,rho_EPA_rbf = 23.009,rho_EPA_periodic = 5253.

#Posterior predictive sampling from estimated intensities
kgr_model2_outfit$fitted_values = poisson_pp_sampling(kgr_model2_outfit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model2_DIC = kgr_model2_outfit$model_DIC
kgr_model2_WAIC = kgr_model2_outfit$model_WAIC

#Get summaries of parameter estimates
kgr_model2_outfit$model_summary
```

	mean	sd	0.025quant	0.5quant	0.975quant	mode
## months1	2.0988878	7.254730	-12.126918	2.0988878	16.32469	2.0988878
## months2	1.8695211	7.254731	-12.356288	1.8695211	16.09533	1.8695211
## months3	1.8910895	7.254731	-12.334719	1.8910895	16.11690	1.8910895
## months4	1.6964798	7.254733	-12.529332	1.6964798	15.92229	1.6964798
## months5	1.6161055	7.254734	-12.609709	1.6161055	15.84192	1.6161055
## months6	1.4917579	7.254735	-12.734059	1.4917579	15.71757	1.4917579

```

## months7    1.4552417 7.254742 -12.770588 1.4552417   15.68107 1.4552417
## months8    1.4342282 7.254742 -12.791602 1.4342282   15.66006 1.4342282
## months9    1.3929132 7.254743 -12.832919 1.3929132   15.61874 1.3929132
## months10   1.4734124 7.254742 -12.752417 1.4734124   15.69924 1.4734124
## months11   1.5164342 7.254741 -12.709394 1.5164342   15.74226 1.5164342
## months12   1.7763170 7.254738 -12.449504 1.7763170   16.00214 1.7763170
## Intercept1  4.2859103 7.254753 -9.939941 4.2859103   18.51176 4.2859103
## Intercept2  2.1517758 7.254758 -12.074085 2.1517758   16.37764 2.1517758
## Intercept3  2.0744570 7.254779 -12.151446 2.0744570   16.30036 2.0744570
## Intercept4  3.6475208 7.254767 -10.578357 3.6475208   17.87340 3.6475208
## Intercept5  1.8604498 7.254759 -12.365412 1.8604498   16.08631 1.8604498
## Intercept6  0.6821203 7.254874 -13.543968 0.6821203   14.90821 0.6821203
## Intercept7  5.0101543 7.254727 -9.215645 5.0101543   19.23595 5.0101543
##          kld
## months1    5.526811e-11
## months2    5.526823e-11
## months3    5.526822e-11
## months4    5.526823e-11
## months5    5.526817e-11
## months6    5.526817e-11
## months7    5.526822e-11
## months8    5.526822e-11
## months9    5.526822e-11
## months10   5.526823e-11
## months11   5.526817e-11
## months12   5.526823e-11
## Intercept1 5.526826e-11
## Intercept2 5.526818e-11
## Intercept3 5.526816e-11
## Intercept4 5.526817e-11
## Intercept5 5.526823e-11
## Intercept6 5.526811e-11
## Intercept7 5.526818e-11
kgr_model2_outfit$bri_hyperpar_summary

##           mean        sd      q0.025      q0.5      q0.975      mode
## SD for id2 0.6519292 0.05113661 0.5562614 0.6501584 0.7571086 0.646798
kgr_model2_outfit$exp_effects

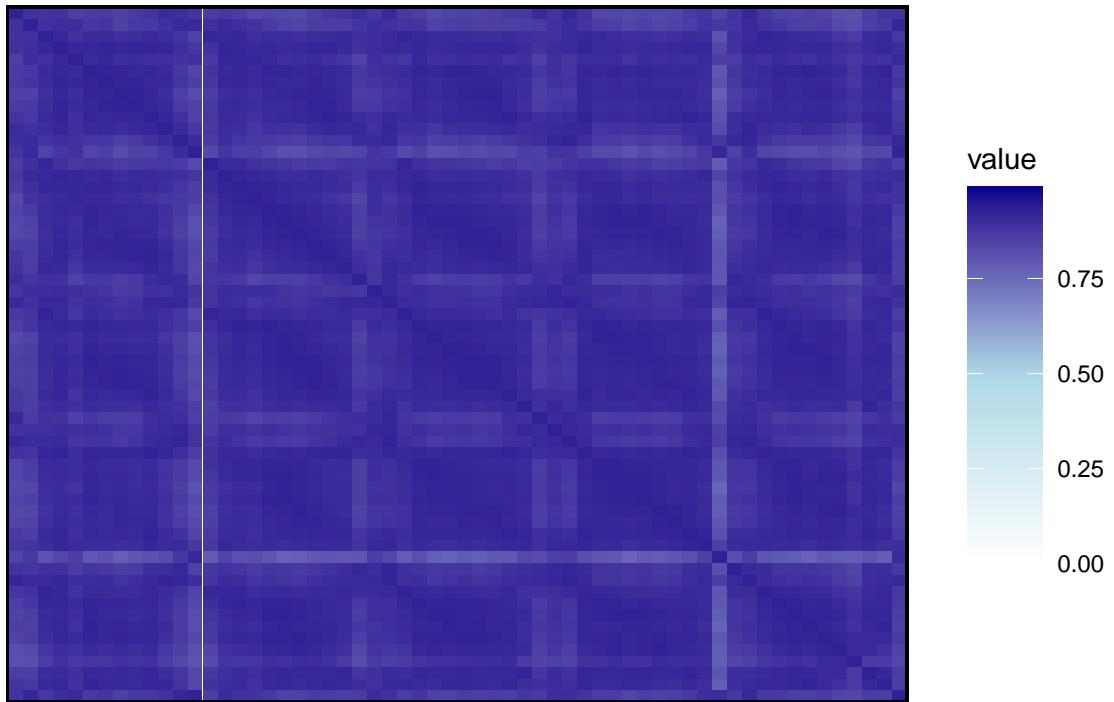
##    months1    months2    months3    months4    months5    months6    months7
##  8.157093  6.485190  6.626585  5.454712  5.033449  4.444902  4.285519
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##  4.196405  4.026563  4.364102  4.555950  5.908057 72.668663  8.600117
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##  7.960223 38.379399  6.426627  1.978067 149.927871
kgr_model2_outfit$K_EPA_weight

## [1] 0.7928439
kgr_model2_outfit$gfilter_weight

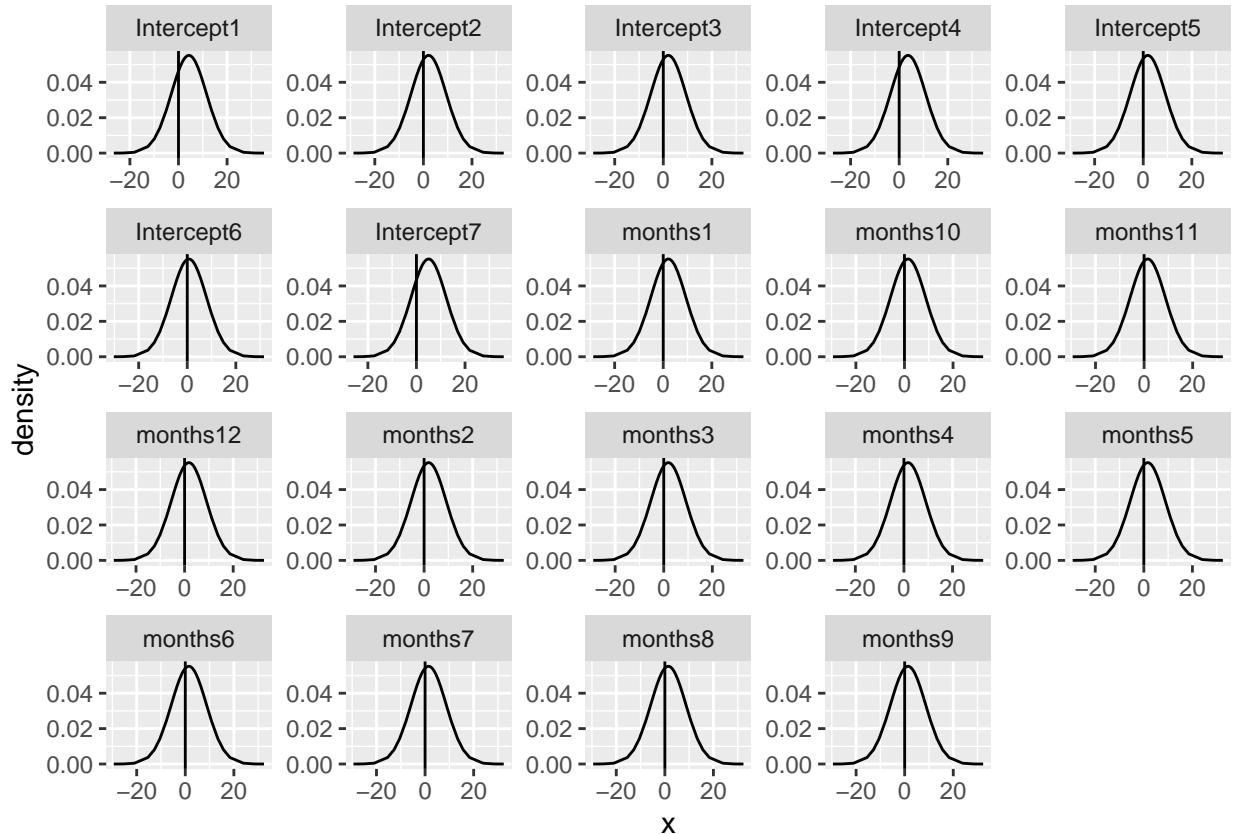
## [1] 0.2071561

```

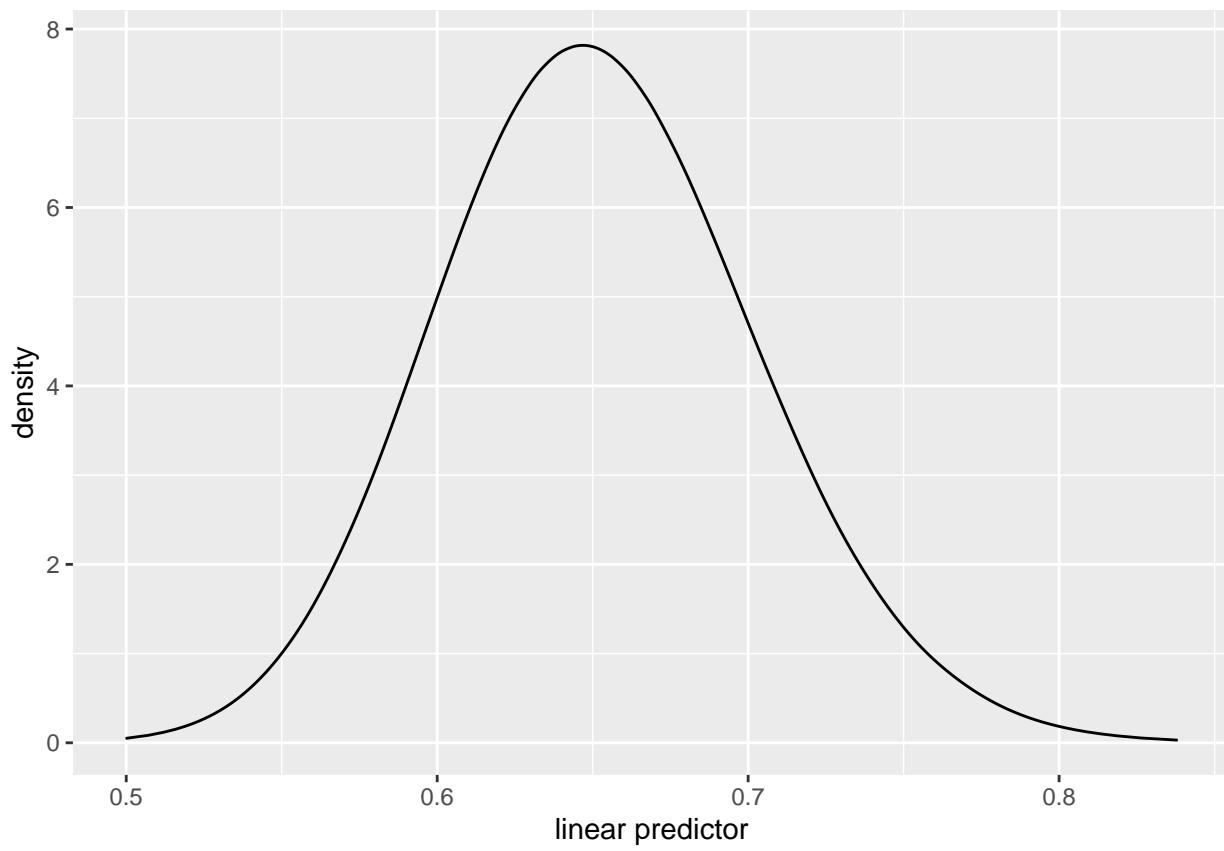
```
#Show plots  
kgr_model2_outfit$K_EPA_heatmap
```



```
kgr_model2_outfit$param_plot
```

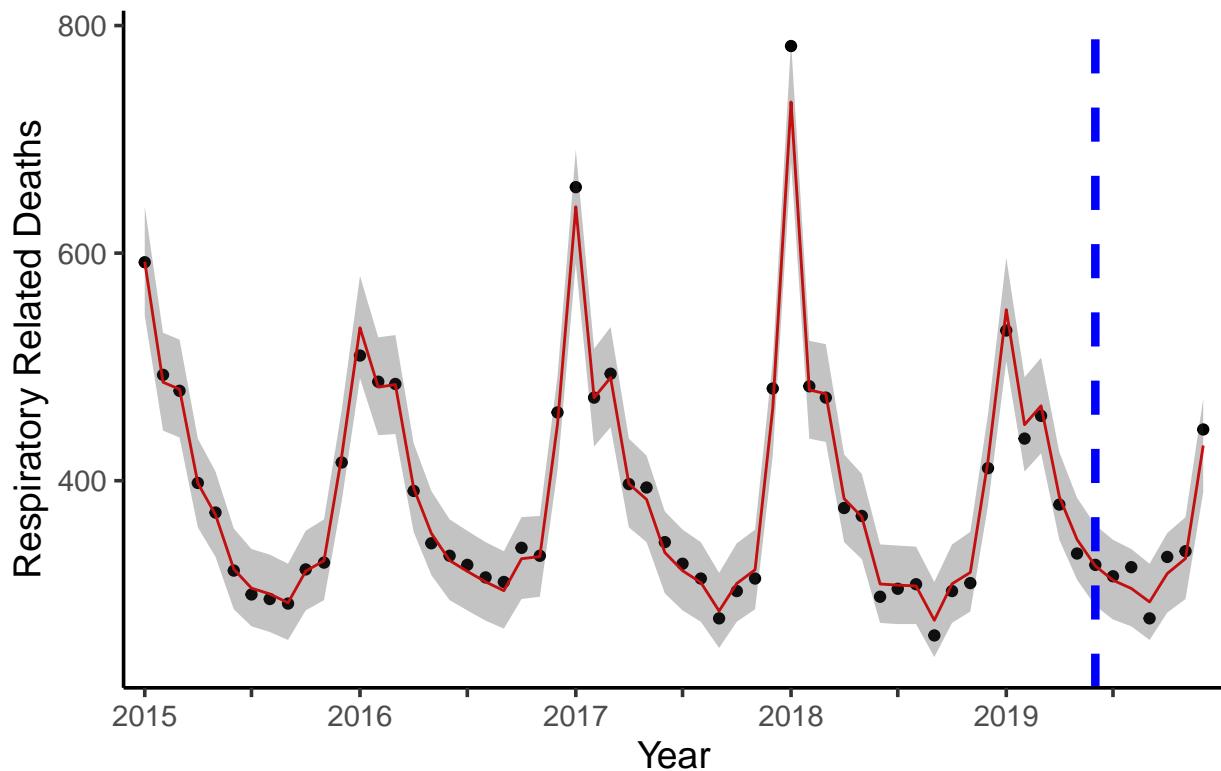


```
kgr_model12_outfit$hyperparam_plot
```

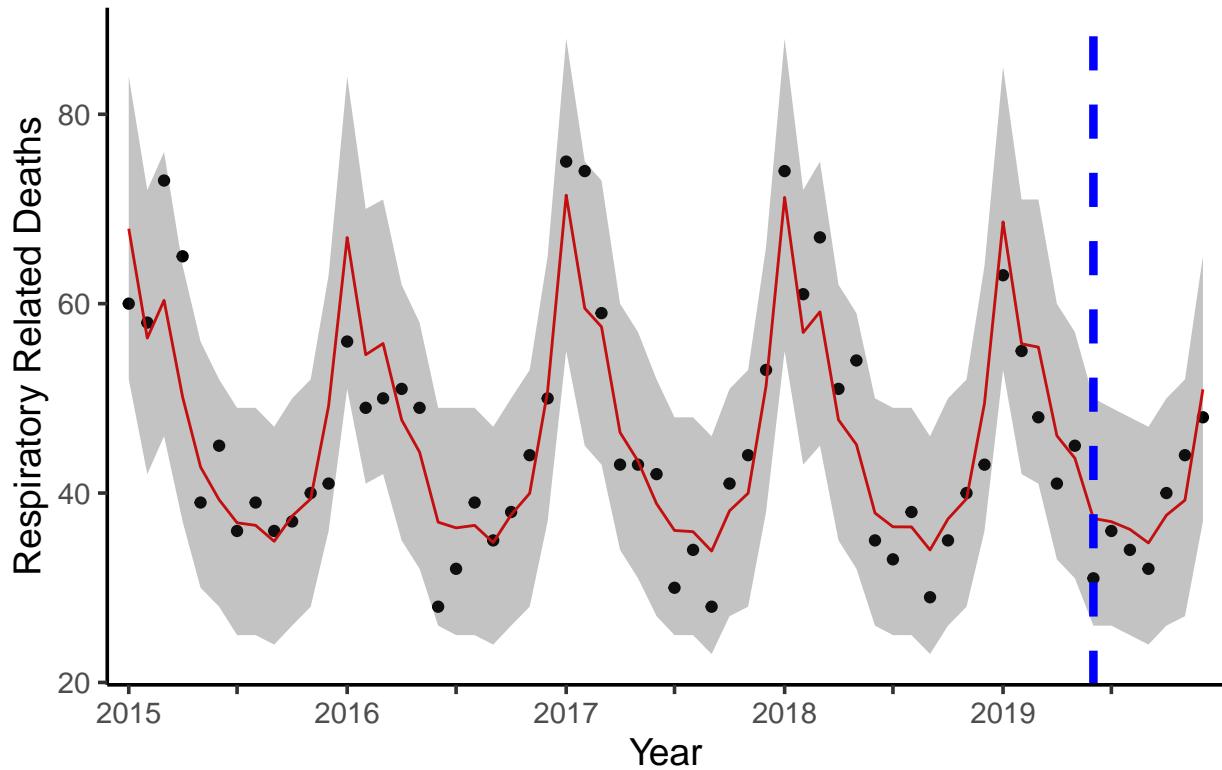


```
pp_outsample_plot(pred_data = kgr_model2_outfit$fitted_values, prefix = "kgr2-outsamp-")
```

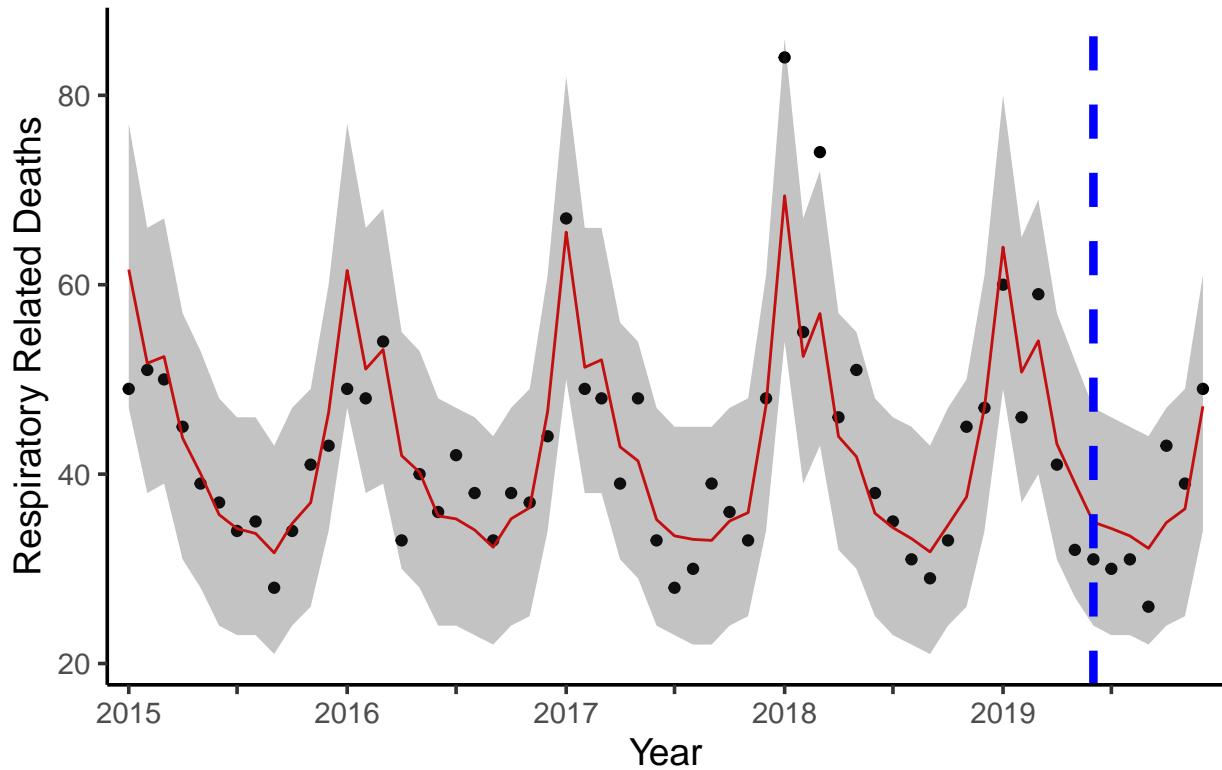
Cluster 1 (Out of Sample Coverage: 100%)



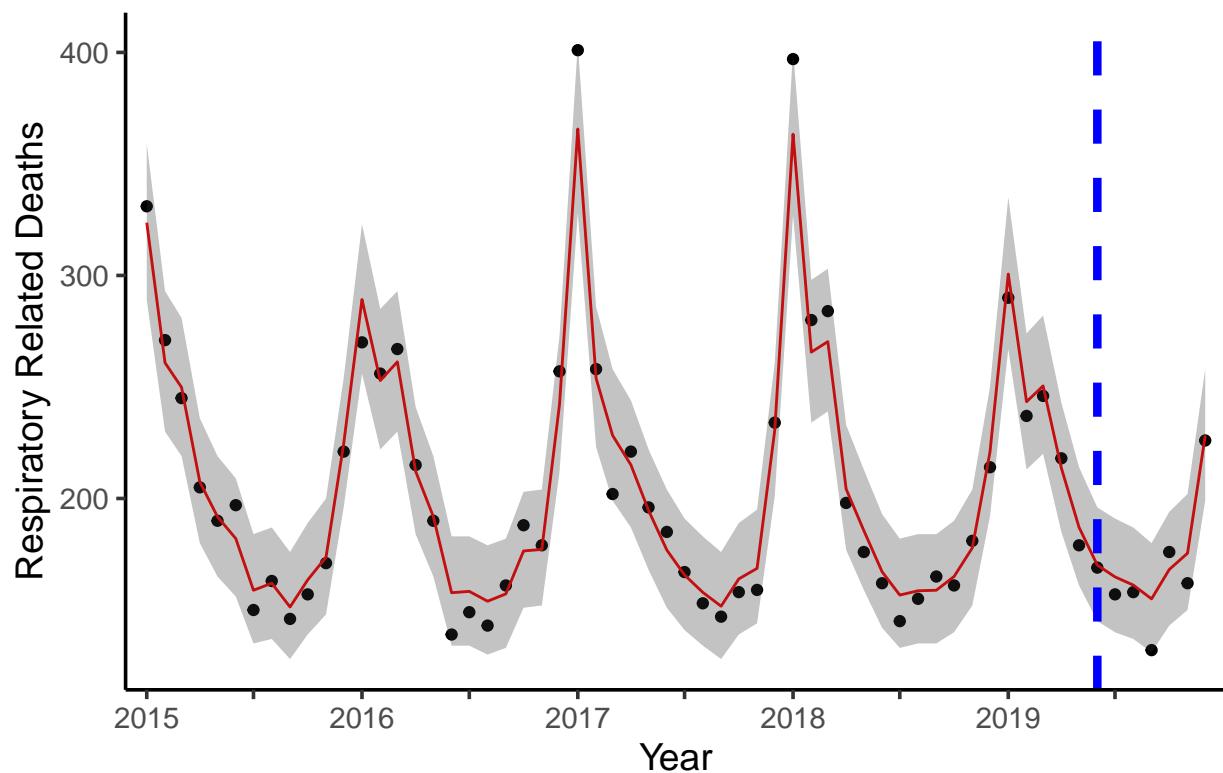
Cluster 2 (Out of Sample Coverage: 100%)



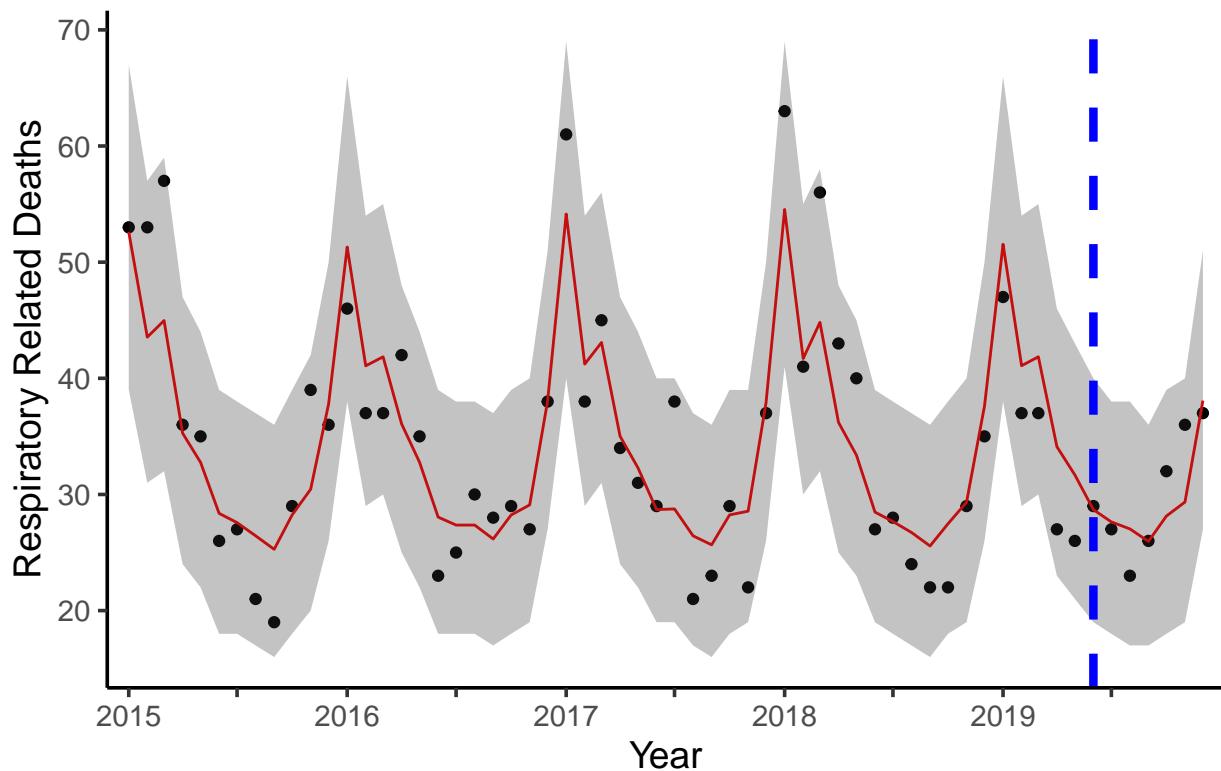
Cluster 3 (Out of Sample Coverage: 100%)



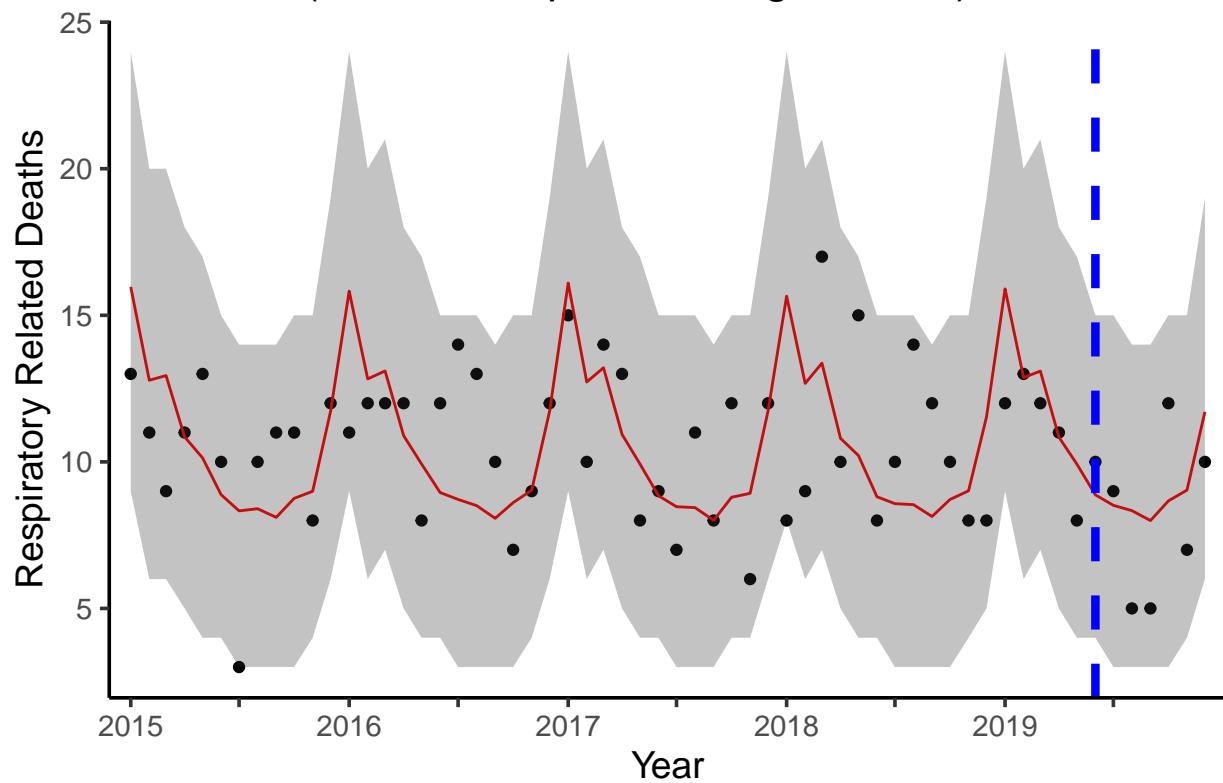
Cluster 4 (Out of Sample Coverage: 100%)



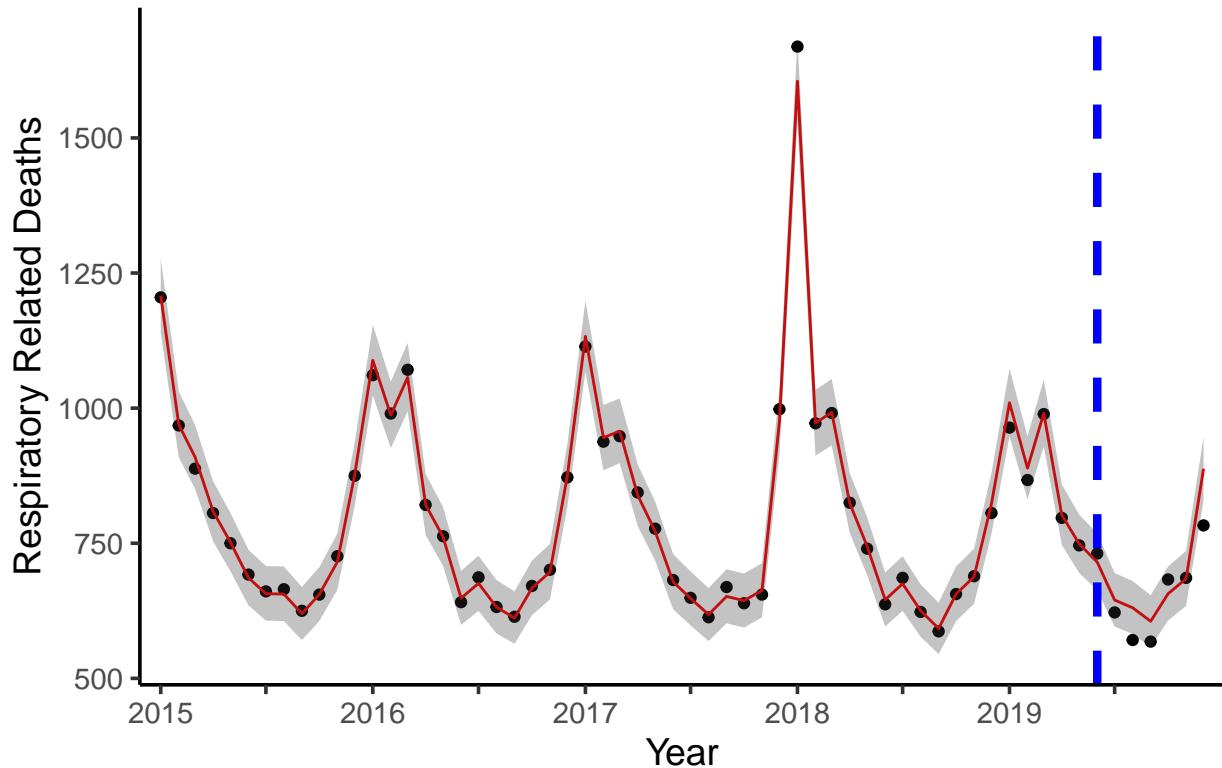
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)

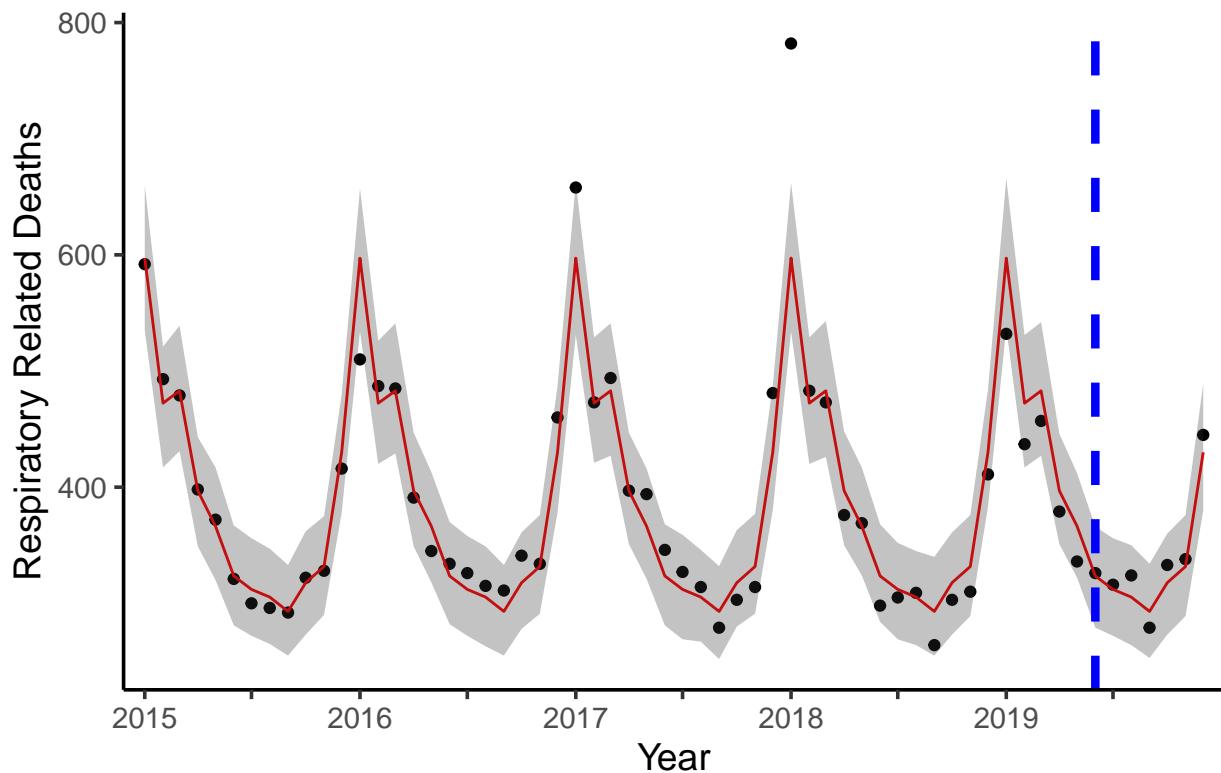


Cluster 7 (Out of Sample Coverage: 66.67%)

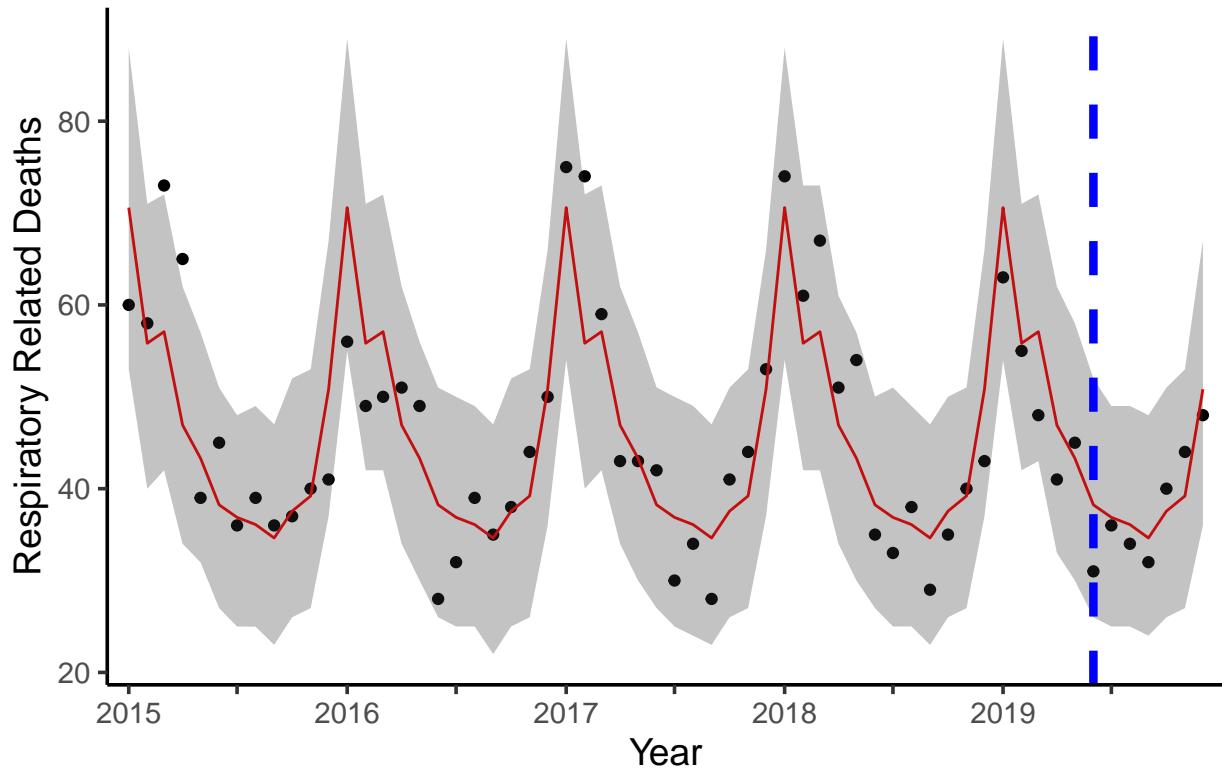


```
pp_outsample_plot_combined(pred_data = kgr_model2_outfit$fitted_values,prefix = "kgr2-outsample-")  
  
kgr_model2_nb_outfit = kgr_model2_nb(data = inla_outsample_data,rho_EPA_rbf = 23.009,rho_EPA_periodic =  
pp_outsample_plot(pred_data = kgr_model2_nb_outfit$fitted_values,prefix = "kgr2-nb-outsample-")
```

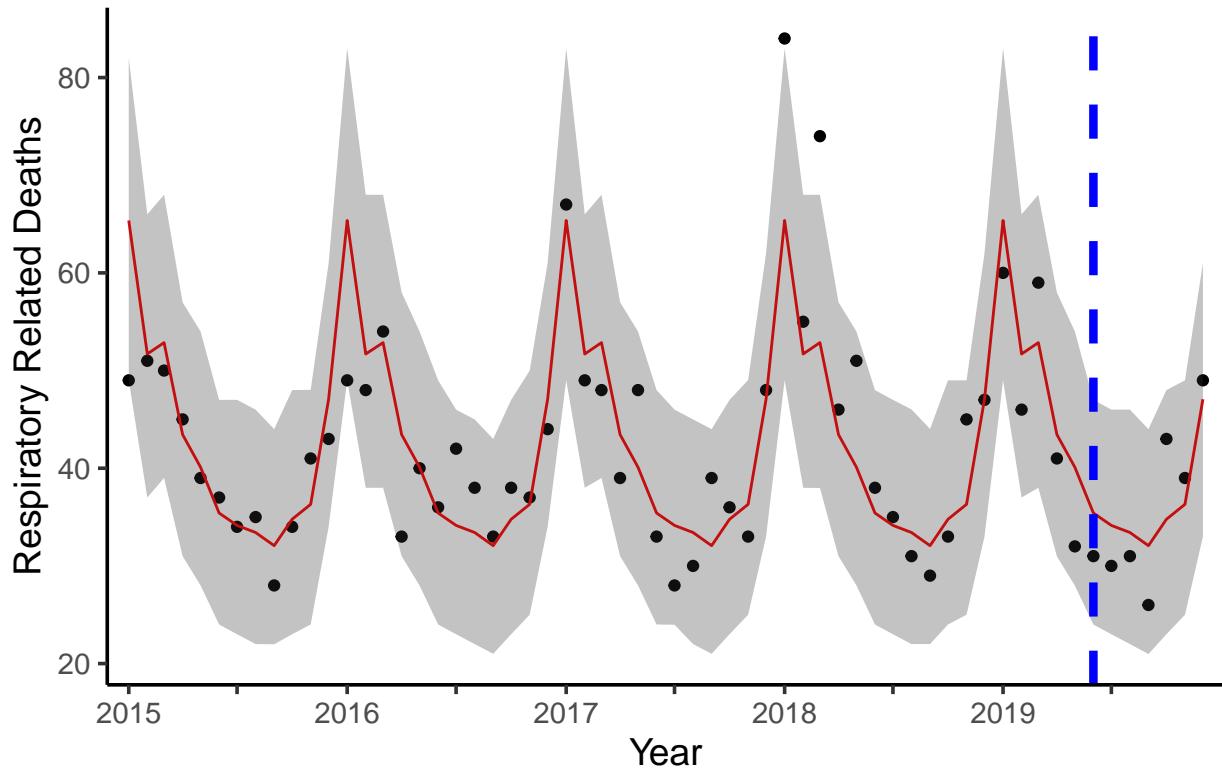
Cluster 1 (Out of Sample Coverage: 100%)



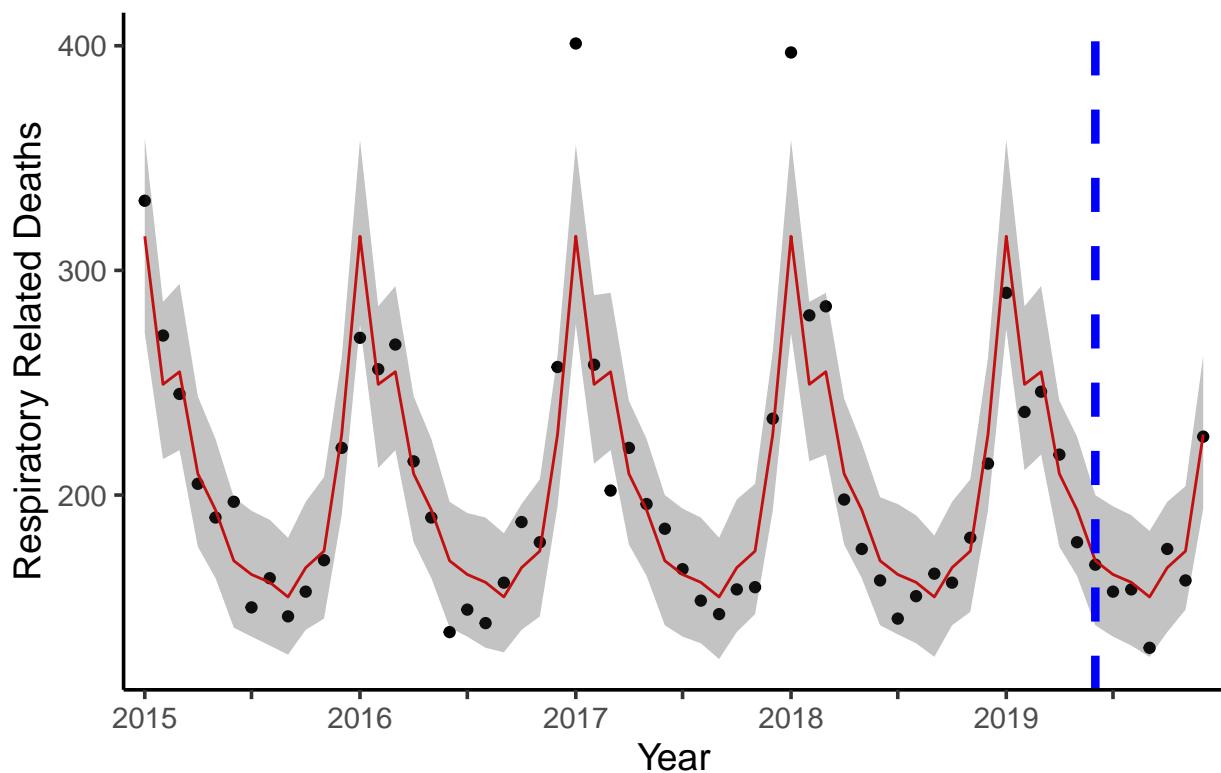
Cluster 2 (Out of Sample Coverage: 100%)



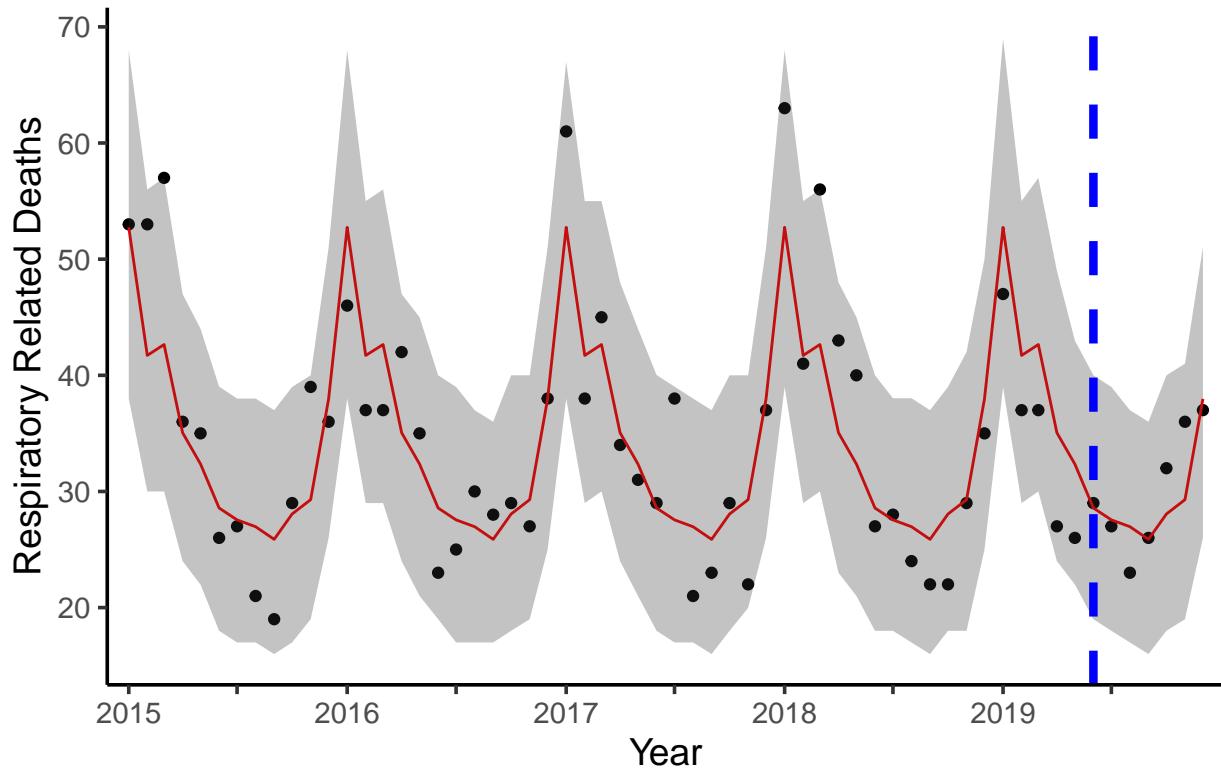
Cluster 3 (Out of Sample Coverage: 100%)



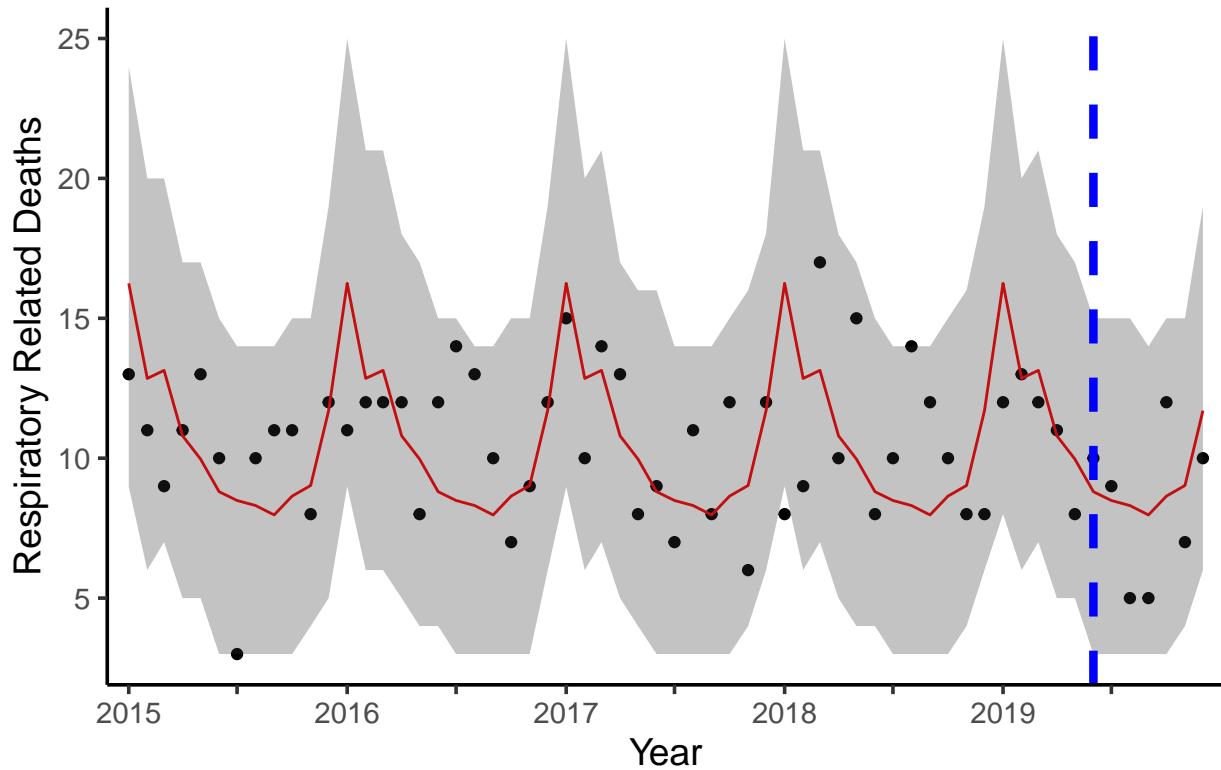
Cluster 4 (Out of Sample Coverage: 100%)



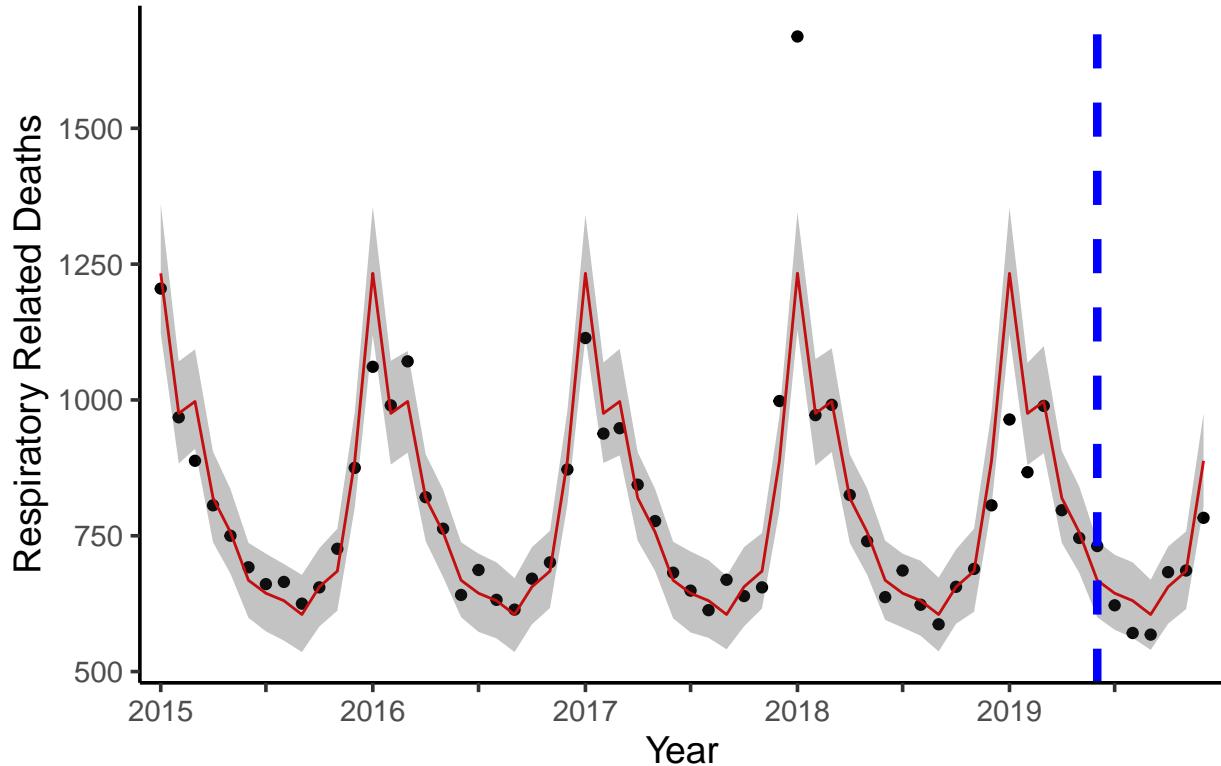
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 83.33%)



```
pp_outsample_plot_combined(pred_data = kgr_model2_nb_outfit$fitted_values, prefix = "kgr2-nb-outsample-")
```

Reference model 3

```
#Fit ref_model3 on one cluster (to test)
ref_model3_outfit = ref_model3(dataset = inla_outsample_data, cluster = 1, rho_time_rbf = 1,
                                rho_time_periodic = 1, sigma2_time = 5)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
##   All aesthetics have the same value.
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

#Extract DIC and WAIC
ref_model3_DIC = ref_model3_outfit$modelDIC
ref_model3_WAIC = ref_model3_outfit$modelWAIC

#Get summaries of parameter estimates
ref_model3_outfit$model_summary

##                                     mean          sd 0.025quant    0.5quant    0.975quant
## Intercept1  5.948496e+00  0.04327453  5.863177 5.948531e+00  6.033617
## Intercept2 -9.939822e-16 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept3 -9.939822e-16 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept4 -9.939822e-16 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept5 -9.939822e-16 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept6 -9.939822e-16 31.62254151 -62.008668 5.489991e-14 62.008668
## Intercept7 -9.939822e-16 31.62254151 -62.008668 5.489991e-14 62.008668
```

```

##          mode      kld
## Intercept1 5.948531e+00 3.011730e-08
## Intercept2 -1.055939e-14 5.526817e-11
## Intercept3 -1.055939e-14 5.526817e-11
## Intercept4 -1.055939e-14 5.526817e-11
## Intercept5 -1.055939e-14 5.526817e-11
## Intercept6 -1.055939e-14 5.526817e-11
## Intercept7 -1.055939e-14 5.526817e-11
ref_model3_outfit$bri_hyperpar_summary

##          mean      sd   q0.025    q0.5   q0.975      mode
## SD for time 0.09452262 0.01002996 0.07700184 0.0937447 0.1163704 0.09228759
ref_model3_outfit$exp_effects

## Intercept1 Intercept2 Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##     383.1767      1.0000      1.0000      1.0000      1.0000      1.0000
ref_model3_outfit$K_time_weight

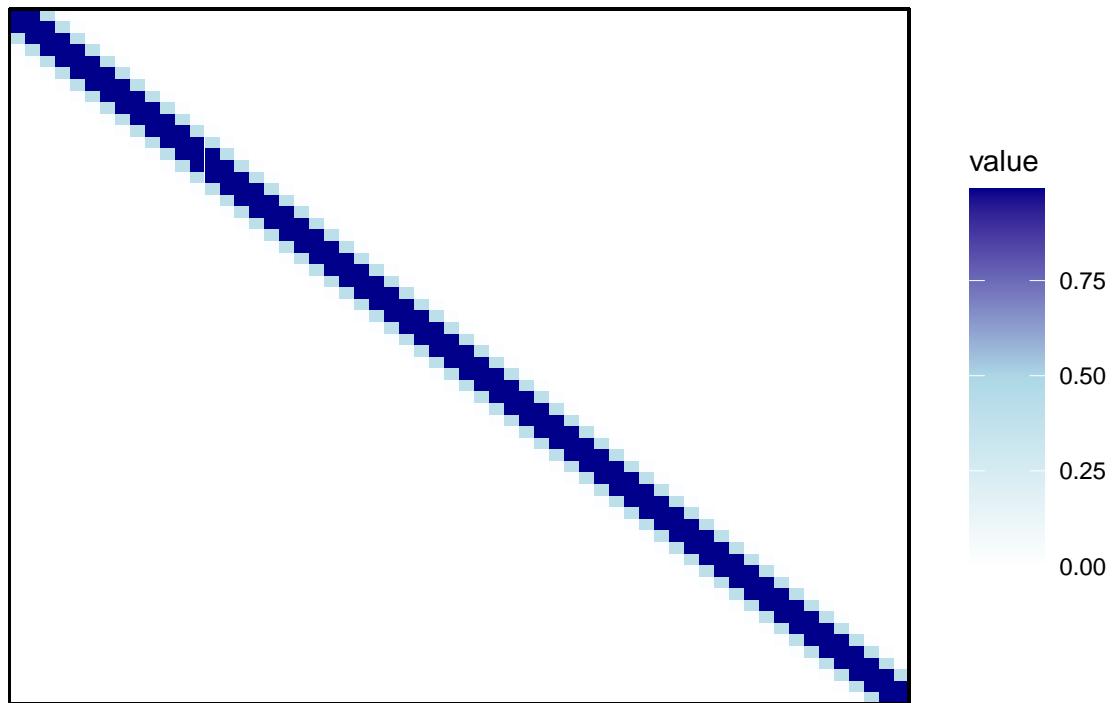
## [1] 0.9952043
ref_model3_outfit$gfilter_weight

## [1] 0.004795715
#Show plots
ref_model3_outfit$K_time_heatmap

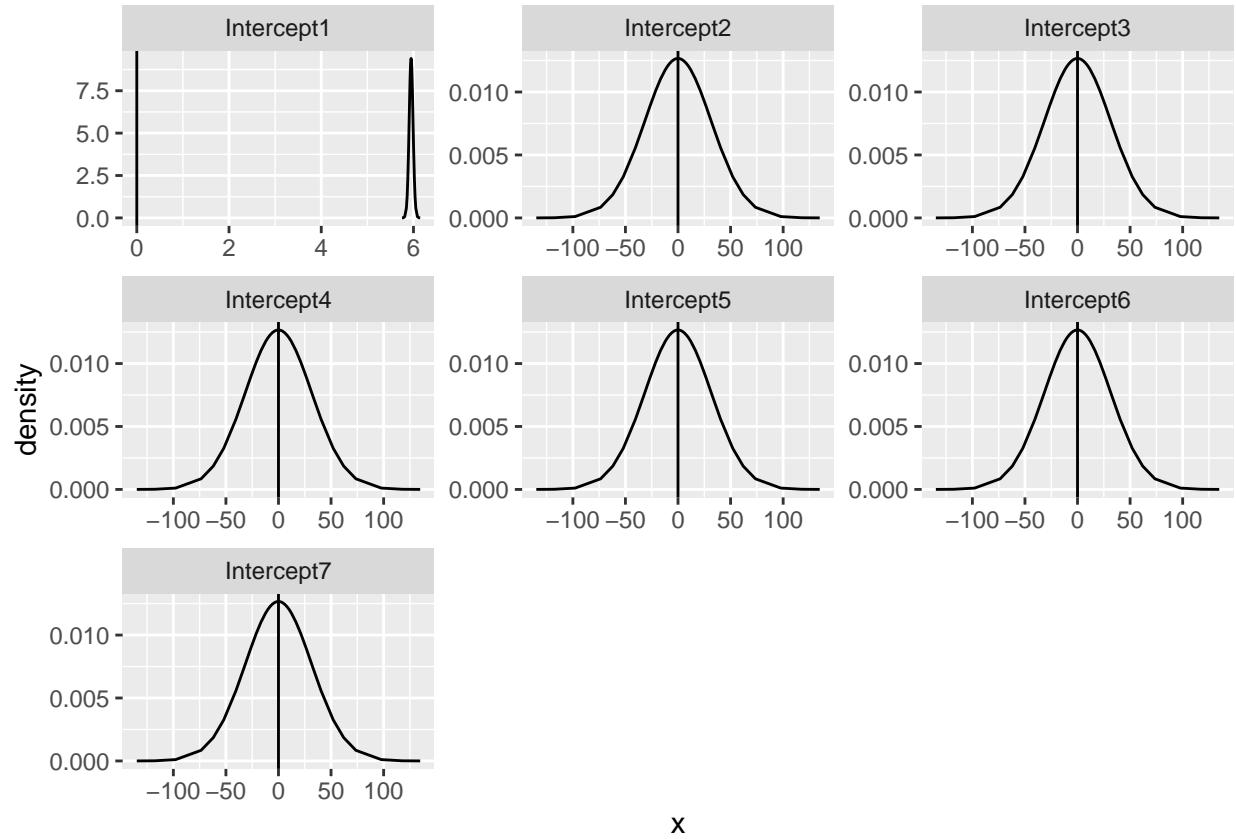
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

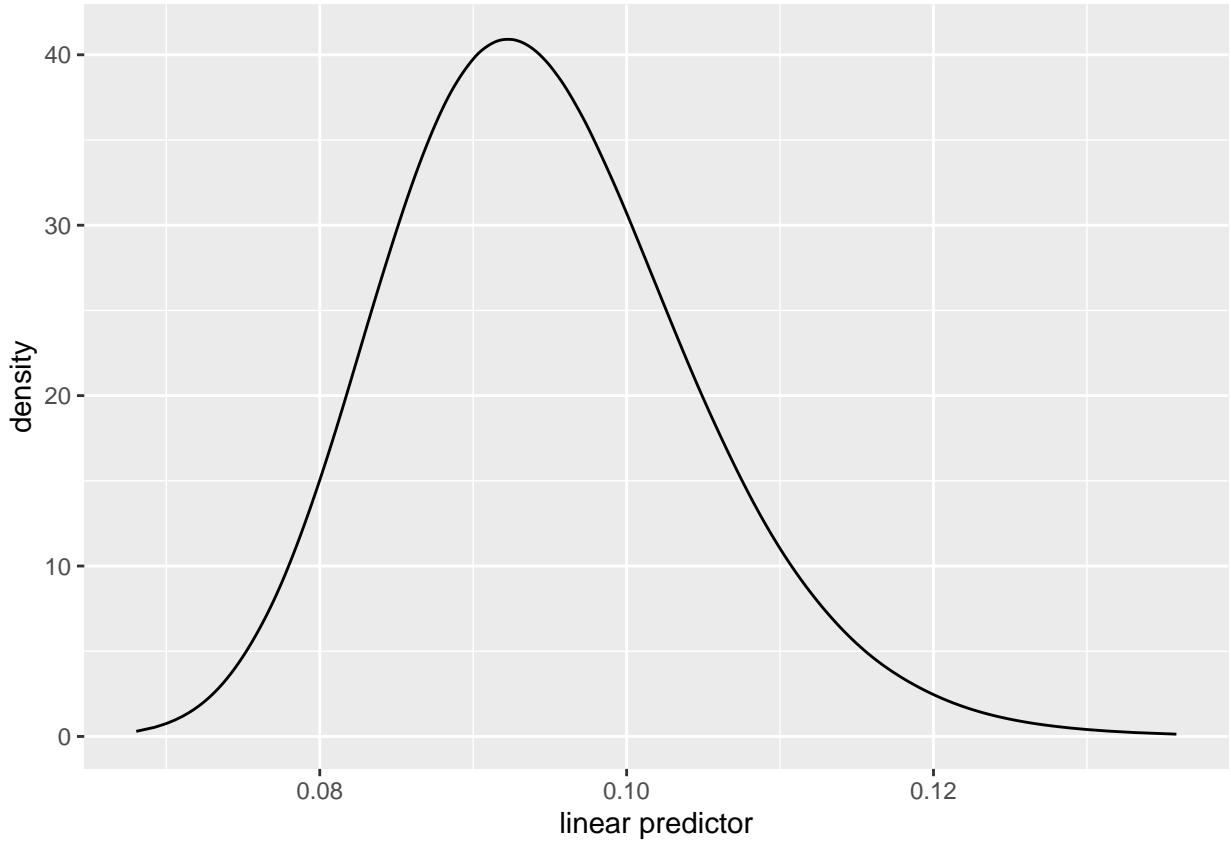
K_time heatmap



ref_model3_outfit\$param_plot



```
ref_model3_outfit$hyperparam_plot
```



```

test1 = ref_model3(dataset = inla_outsample_data, cluster = 1, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test2 = ref_model3(dataset = inla_outsample_data, cluster = 2, rho_time_rbf = 498.918,
                   rho_time_periodic = 120.307, sigma2_time = 0.014)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test3 = ref_model3(dataset = inla_outsample_data, cluster = 3, rho_time_rbf = 8.090,
                   rho_time_periodic = 468.170, sigma2_time = 2.428)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test4 = ref_model3(dataset = inla_outsample_data, cluster = 4, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802) #why does sigma2 have to be so big h

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

```

```

test5 = ref_model3(dataset = inla_outsample_data, cluster = 5, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test6 = ref_model3(dataset = inla_outsample_data, cluster = 6, rho_time_rbf = 105.474,
                   rho_time_periodic = 1.902, sigma2_time = 1.802) #why does sigma2 have to be so big h

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

test7 = ref_model3(dataset = inla_outsample_data, cluster = 7, rho_time_rbf = 474.985,
                   rho_time_periodic = 2.050, sigma2_time = 1.489)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

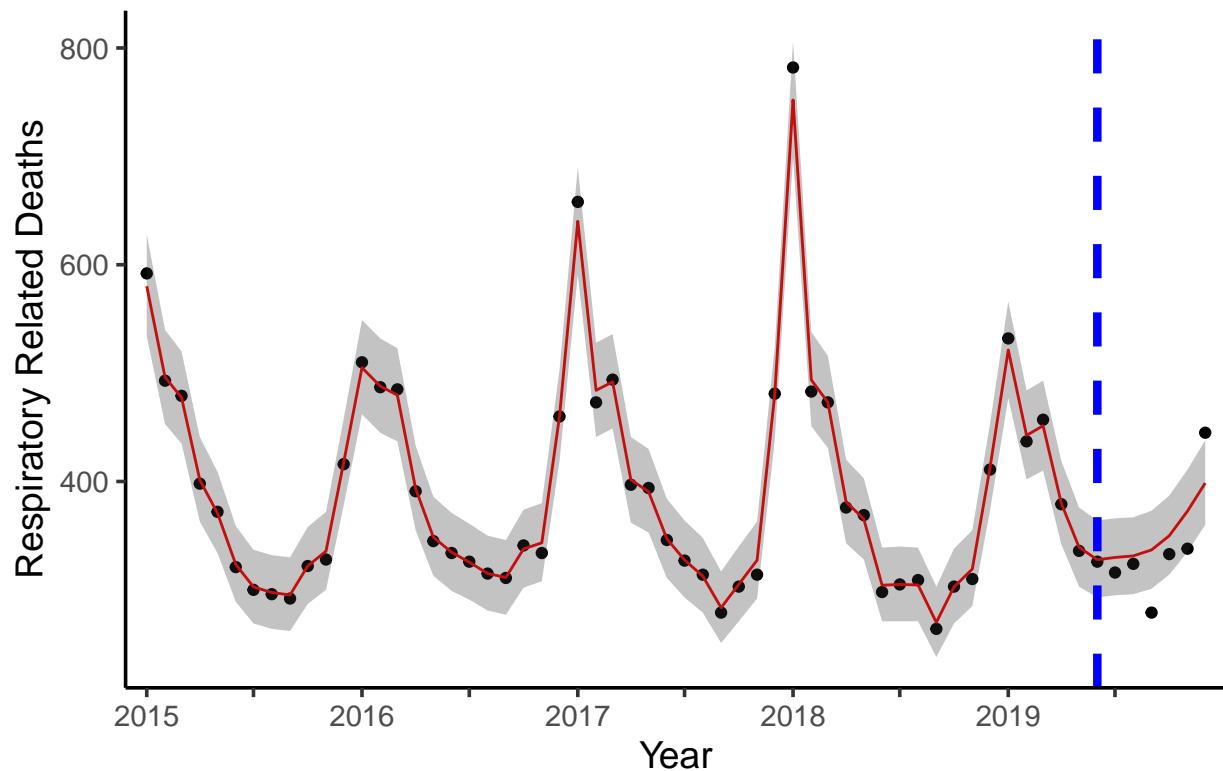
#Posterior predictive sampling from estimated intensities
test1$fitted_values = poisson_pp_sampling(test1$fitted_values,n=100000)
test2$fitted_values = poisson_pp_sampling(test2$fitted_values,n=100000)
test3$fitted_values = poisson_pp_sampling(test3$fitted_values,n=100000)
test4$fitted_values = poisson_pp_sampling(test4$fitted_values,n=100000)
test5$fitted_values = poisson_pp_sampling(test5$fitted_values,n=100000)
test6$fitted_values = poisson_pp_sampling(test6$fitted_values,n=100000)
test7$fitted_values = poisson_pp_sampling(test7$fitted_values,n=100000)

ref_model3_outfvs = rbind(test1$fitted_values,test2$fitted_values,test3$fitted_values,
                           test4$fitted_values,test5$fitted_values,test6$fitted_values,test7$fitted_values)

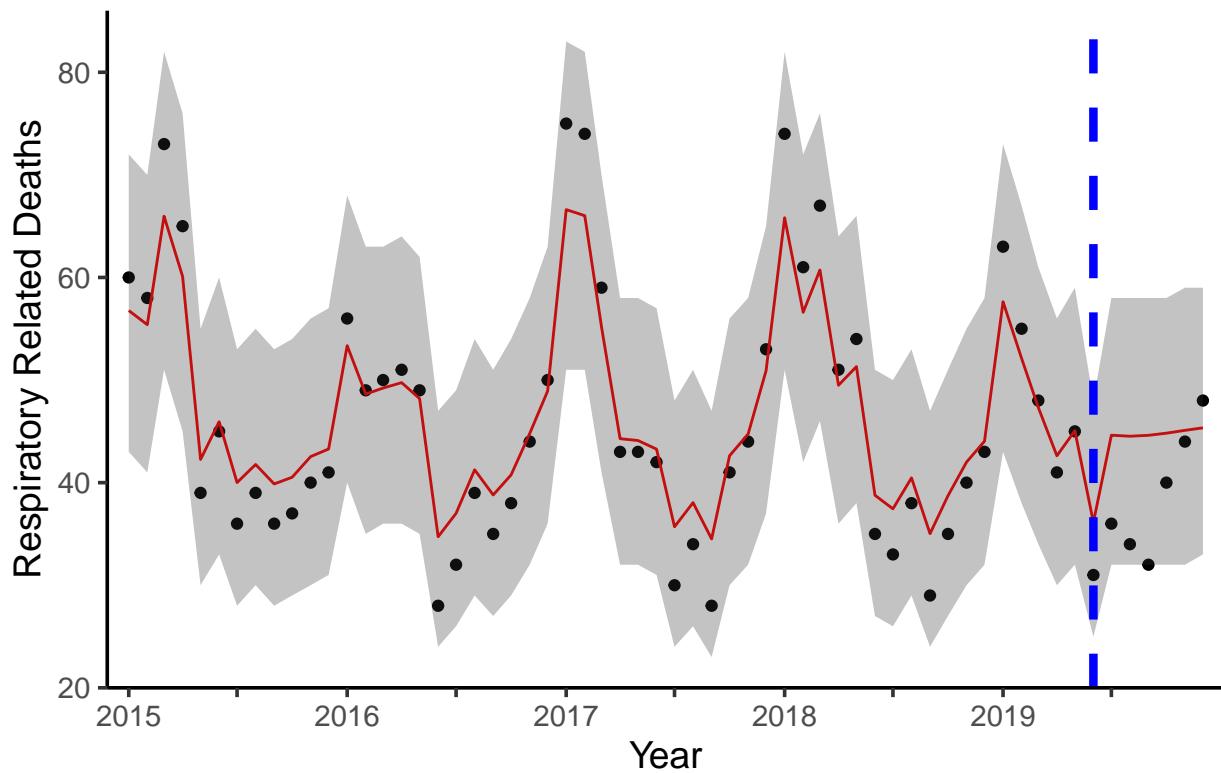
pp_outsample_plot(num_plots = num_clus,ref_data = inla_full_data,pred_data = ref_model3_outfvs,prefix =

```

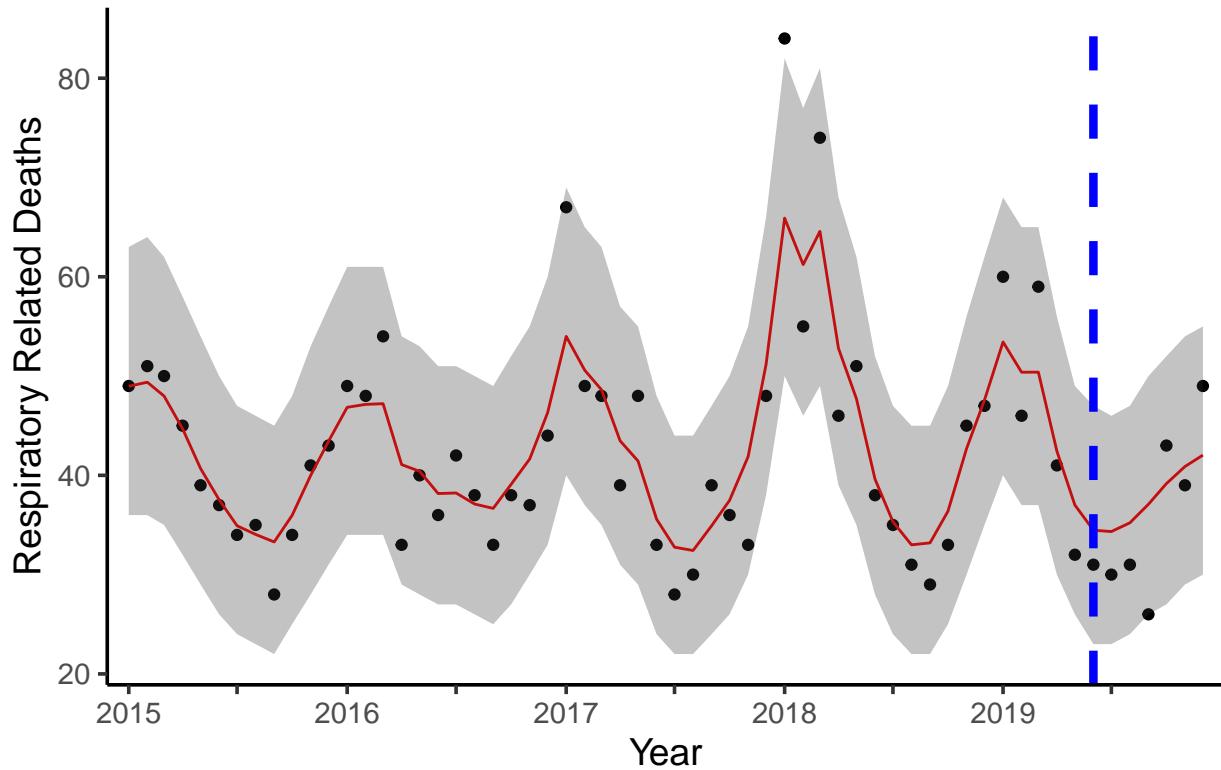
Cluster 1 (Out of Sample Coverage: 66.67%)



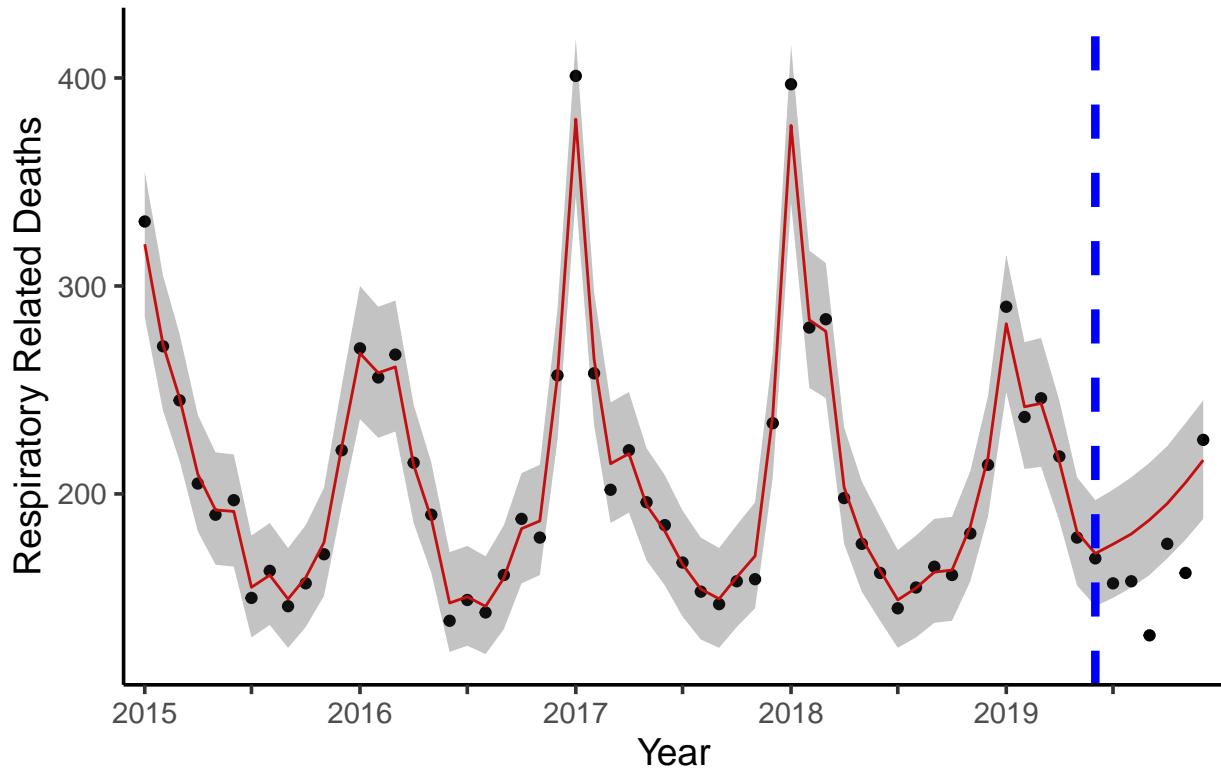
Cluster 2 (Out of Sample Coverage: 100%)



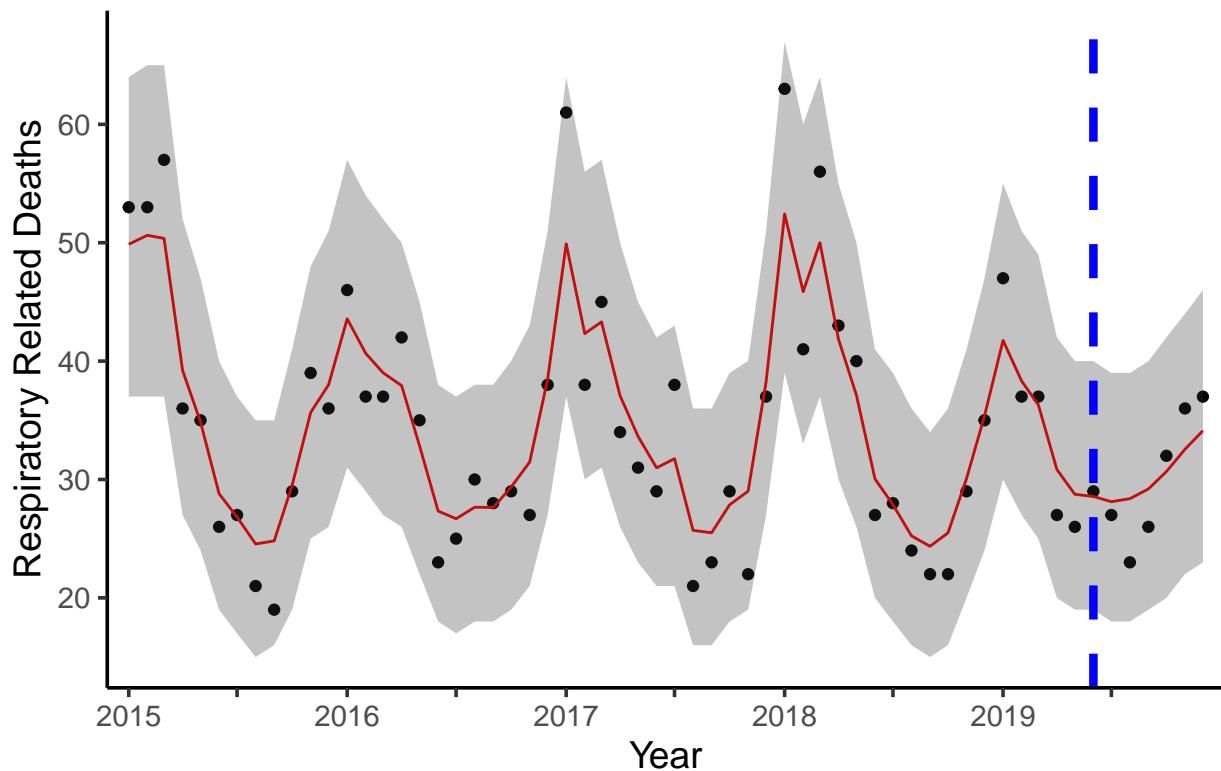
Cluster 3 (Out of Sample Coverage: 100%)



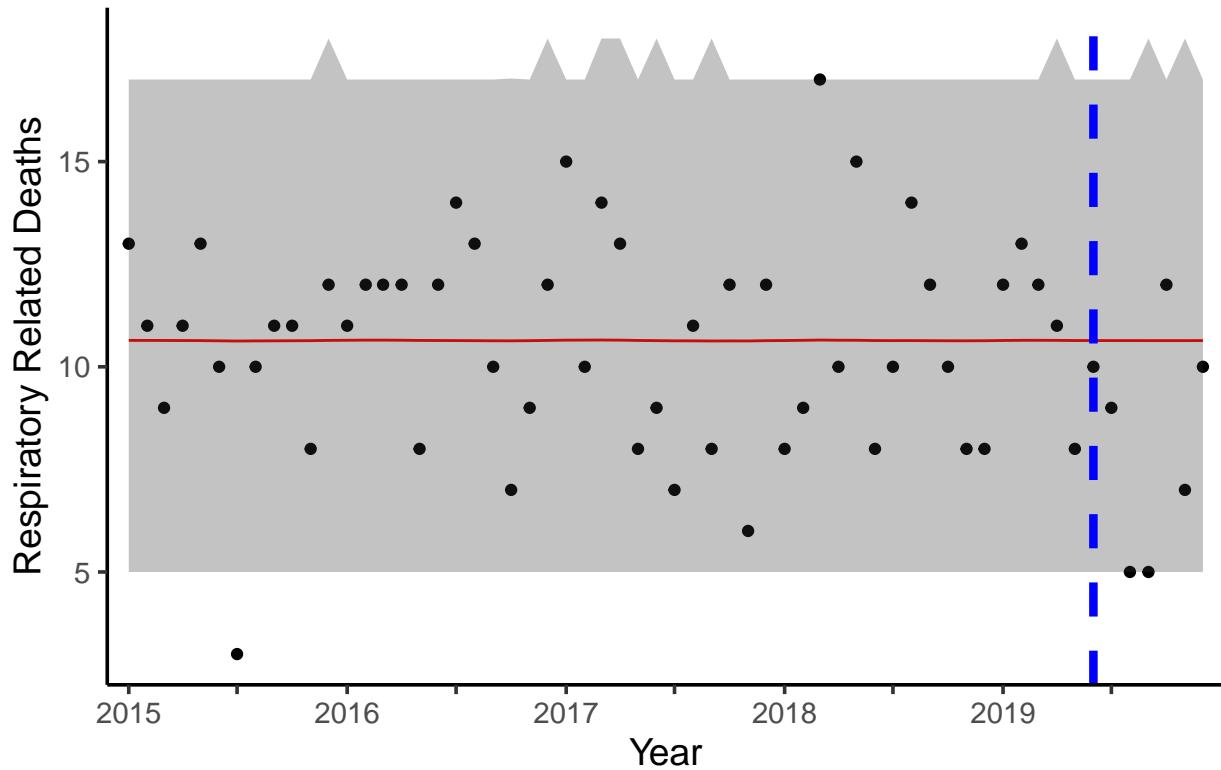
Cluster 4 (Out of Sample Coverage: 66.67%)



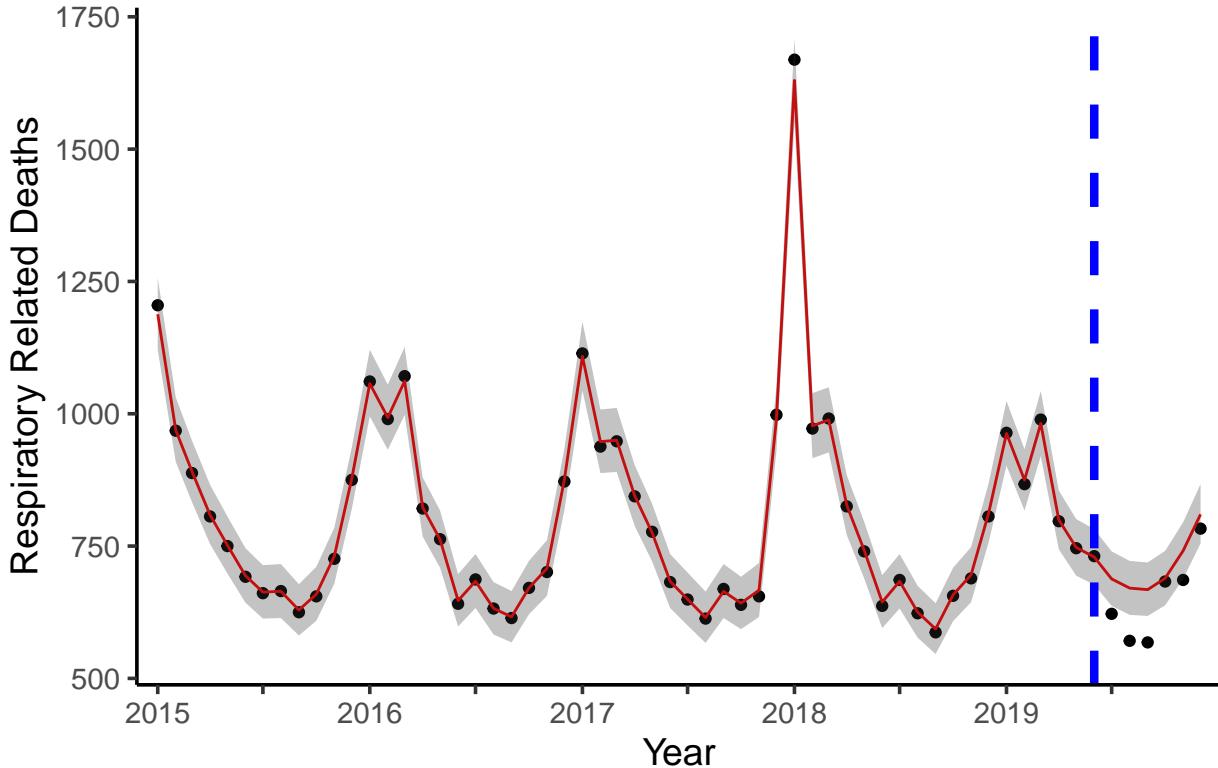
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 33.33%)



```
pp_outsample_plot_combined(pred_data = ref_model3_outfvs,prefix = "ref3-outsample-")
```

Reference model 4 (simple time kernel + EPA covariates)

```
#Fit ref_model4
ref_model4_outfit = ref_model4(dataset = inla_outsample_data_wcovariates,rho_time_rbf = 483.957,rho_time_spatial = 1.5,rho_spatial = 1000000)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

#Posterior predictive sampling from estimated intensities
ref_model4_outfit$fitted_values = poisson_pp_sampling(ref_model4_outfit$fitted_values,n=1000000)

#Extract DIC and WAIC
ref_model4_DIC = ref_model4_outfit$model_DIC
ref_model4_WAIC = ref_model4_outfit$model_WAIC

#Get summaries of parameter estimates
ref_model4_outfit$model_summary

##               mean        sd  0.025quant    0.5quant    0.975quant
## lead      -0.024199244 0.01488364 -0.05341095 -0.024196126 0.004994647
## co        0.054393492 0.01911386  0.01688357  0.054395520 0.091891826
## so2       -0.002897965 0.01292425 -0.02829326 -0.002885188 0.022424558
## no2       0.054061322 0.01586268  0.02290419  0.054072860 0.085152796
## o3        -0.010810130 0.01523288 -0.04071597 -0.010804099 0.019061344
```

```

## pm10      -0.076350749 0.01250546 -0.10079417 -0.076383783 -0.051719279
## pm25      -0.016413020 0.01131100 -0.03862456 -0.016406733 0.005762791
## Intercept1 5.940529857 0.07022278 5.80265738 5.940534208 6.078377693
## Intercept2 3.809113444 0.05774871 3.69569830 3.809139565 3.922379718
## Intercept3 3.719512905 0.07590625 3.57041408 3.719551261 3.868393380
## Intercept4 5.312603449 0.07904670 5.15741644 5.312605940 5.467776320
## Intercept5 3.520286880 0.04981117 3.42244927 3.520319427 3.617938798
## Intercept6 2.345863373 0.09490579 2.15958113 2.345878969 2.532056403
## Intercept7 6.672843213 0.04374690 6.58696141 6.672842774 6.758727524
##           mode      kld
## lead      -0.024196139 5.197455e-10
## co        0.054395511 5.932623e-10
## so2       -0.002885169 8.016090e-10
## no2       0.054072896 6.788471e-10
## o3        -0.010804095 5.953343e-10
## pm10     -0.076383856 2.302480e-09
## pm25     -0.016406702 6.521665e-10
## Intercept1 5.940534241 2.438846e-09
## Intercept2 3.809139545 1.408923e-09
## Intercept3 3.719551287 1.711295e-09
## Intercept4 5.312605951 2.346365e-09
## Intercept5 3.520319370 9.139234e-10
## Intercept6 2.345878874 8.649953e-10
## Intercept7 6.672842776 2.432900e-09

ref_model4_outfit$bri_hyperpar_summary

##           mean      sd   q0.025    q0.5   q0.975      mode
## SD for id2 1.278435 0.06845018 1.150027 1.276184 1.418838 1.271884

ref_model4_outfit$exp_effects

##           lead      co      so2      no2      o3      pm10
## 0.9760912 1.0559000 0.9971062 1.0555493 0.9892481 0.9264912
##          pm25 Intercept1 Intercept2 Intercept3 Intercept4 Intercept5
## 0.9837209 380.1362941 45.1104282 41.2442993 202.8777233 33.7941219
## Intercept6 Intercept7
## 10.4422844 790.6403710

ref_model4_outfit$K_time_weight

## [1] 0.9020751

ref_model4_outfit$gfilter_weight

## [1] 0.09792492

ref_model4_outfit$num_jitters

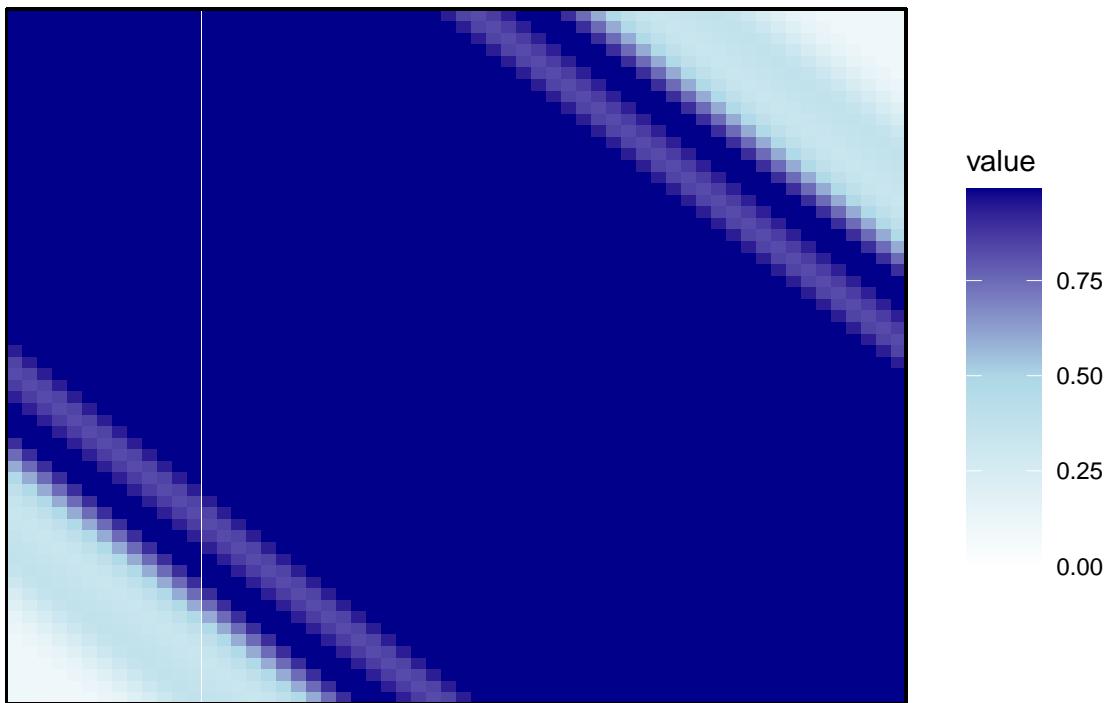
## [1] 1

#Show plots
ref_model4_outfit$K_time_heatmap

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
## a single row.

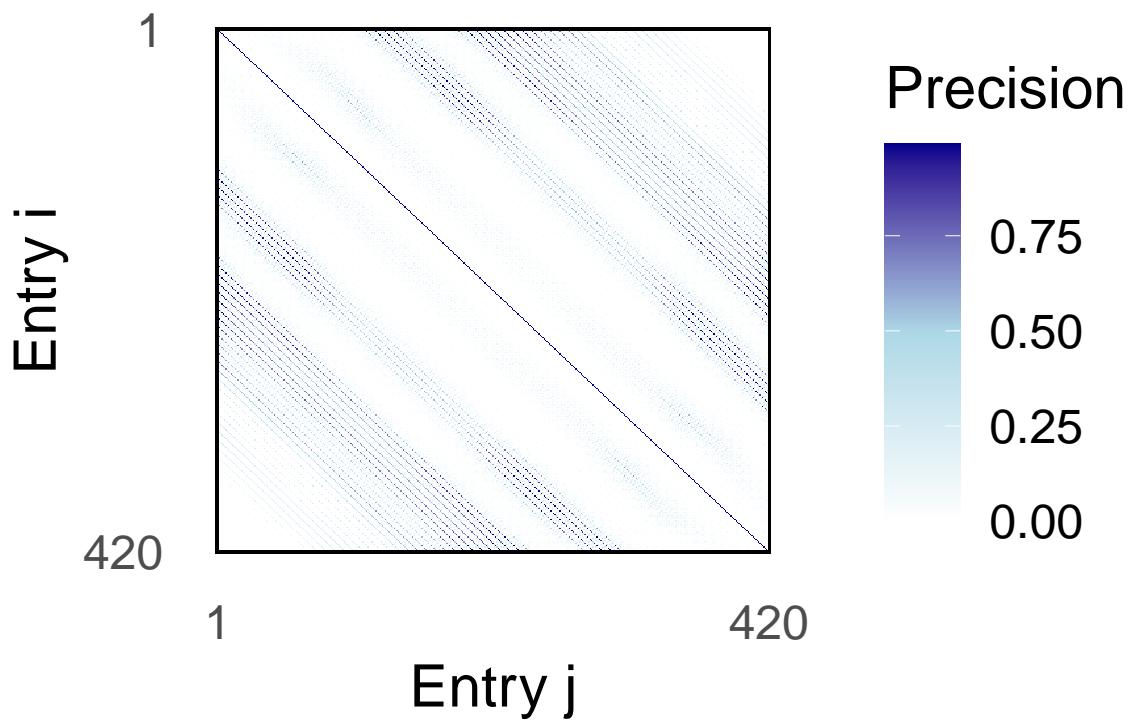
```

K_time heatmap

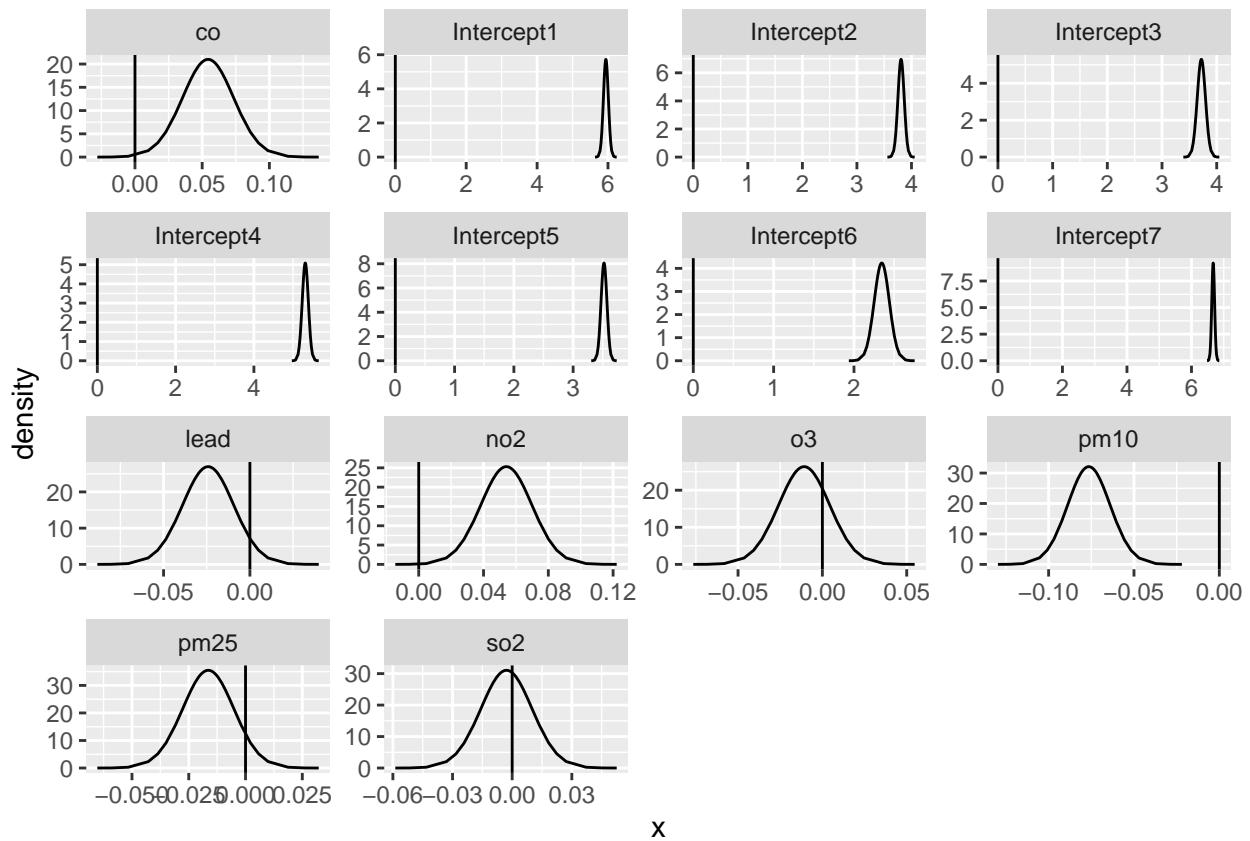


```
ref_model4_outfit$prec_heatmap
```

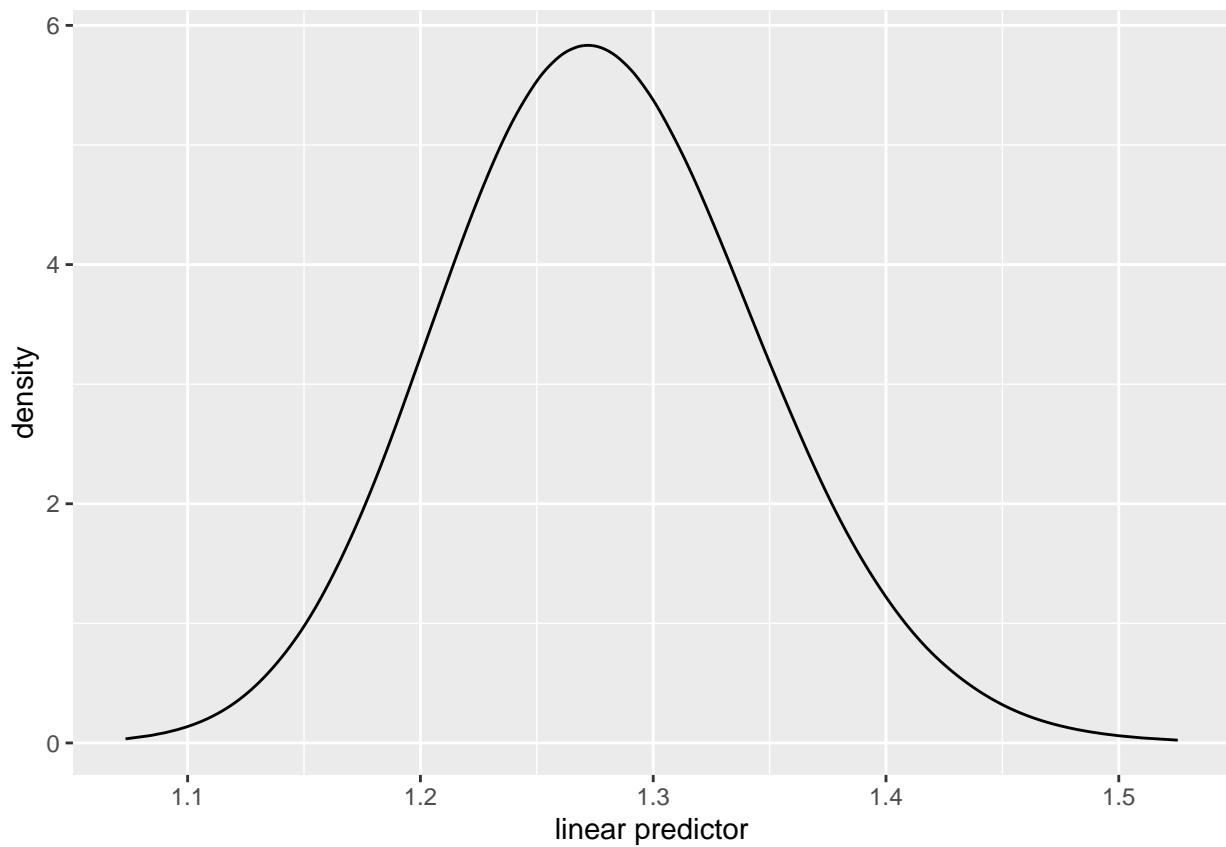
```
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



```
ref_model4_outfit$param_plot
```

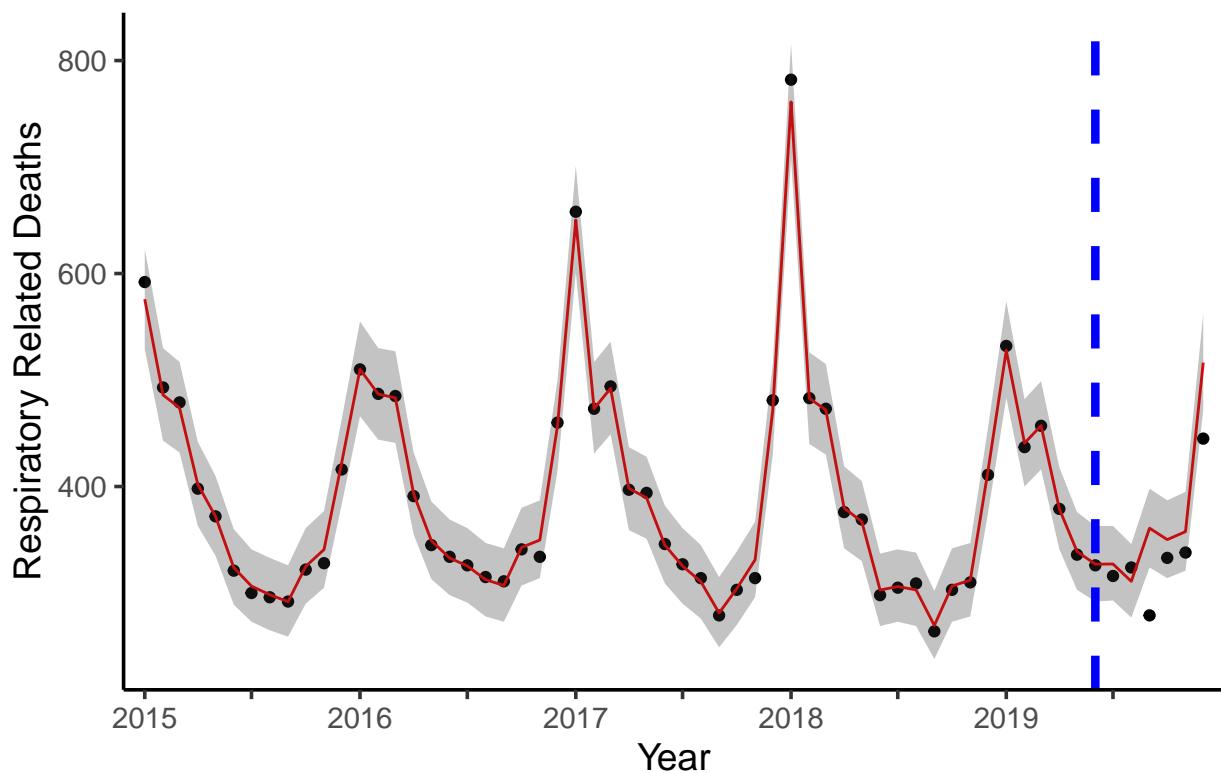


```
ref_model4_outfit$hyperparam_plot
```

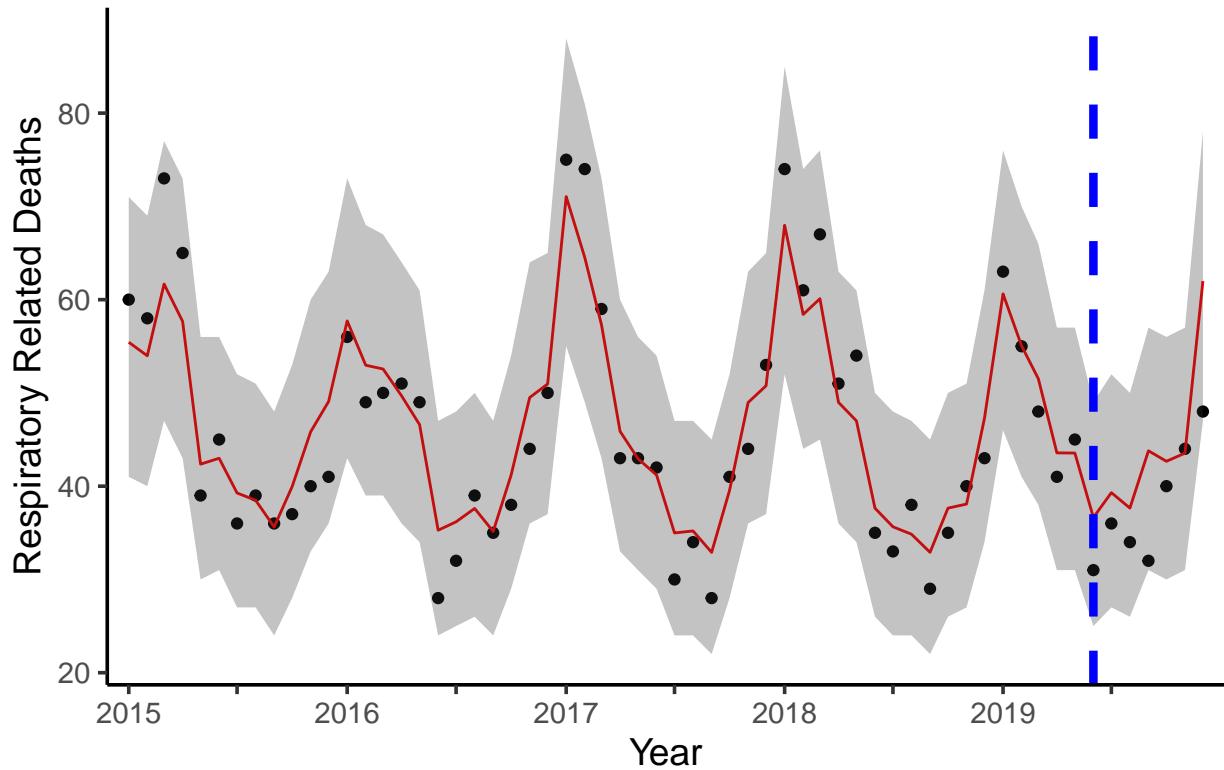


```
pp_outsample_plot(pred_data = ref_model4_outfit$fitted_values, prefix = "ref4-outsamp-")
```

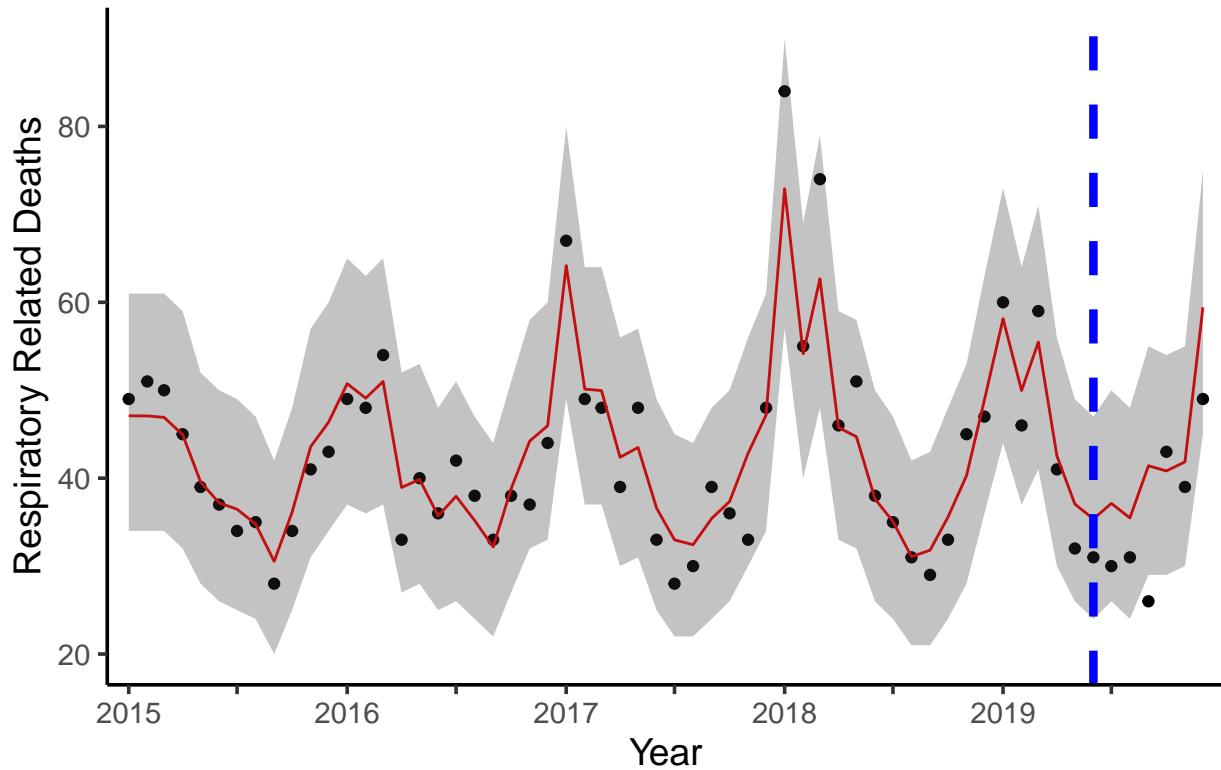
Cluster 1 (Out of Sample Coverage: 66.67%)



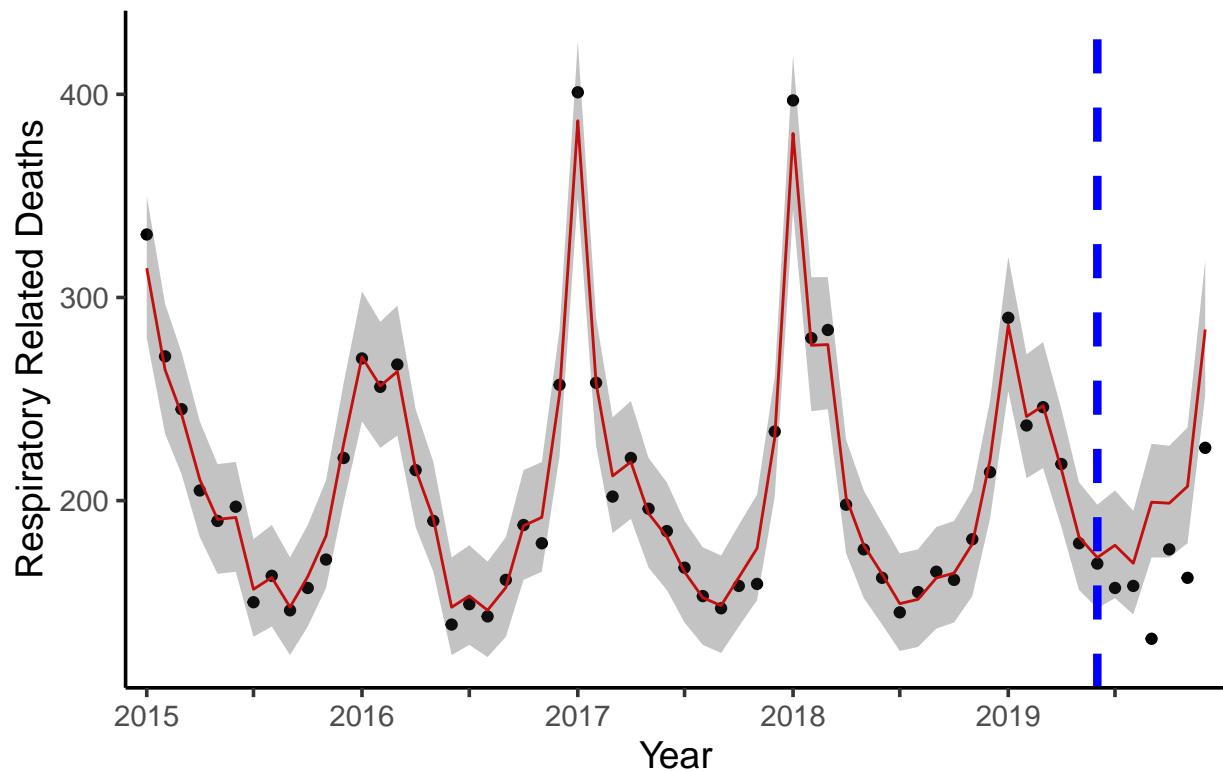
Cluster 2 (Out of Sample Coverage: 100%)



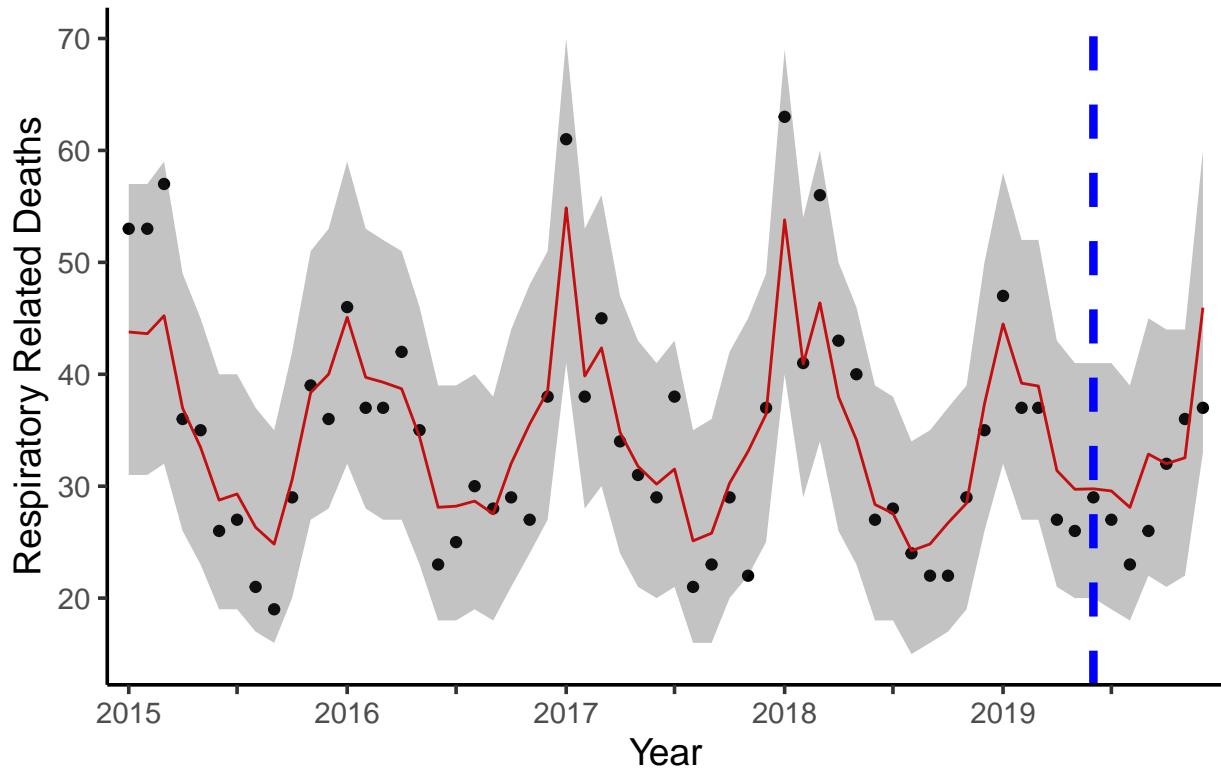
Cluster 3 (Out of Sample Coverage: 83.33%)



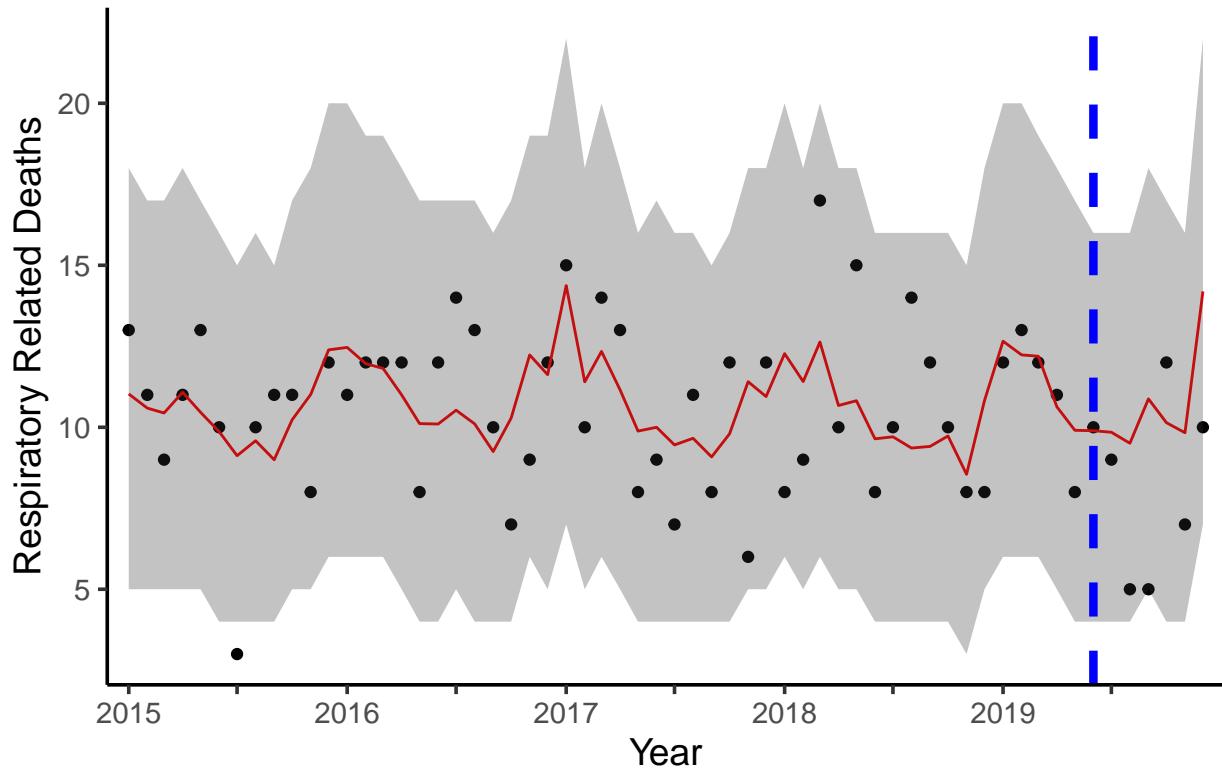
Cluster 4 (Out of Sample Coverage: 50%)



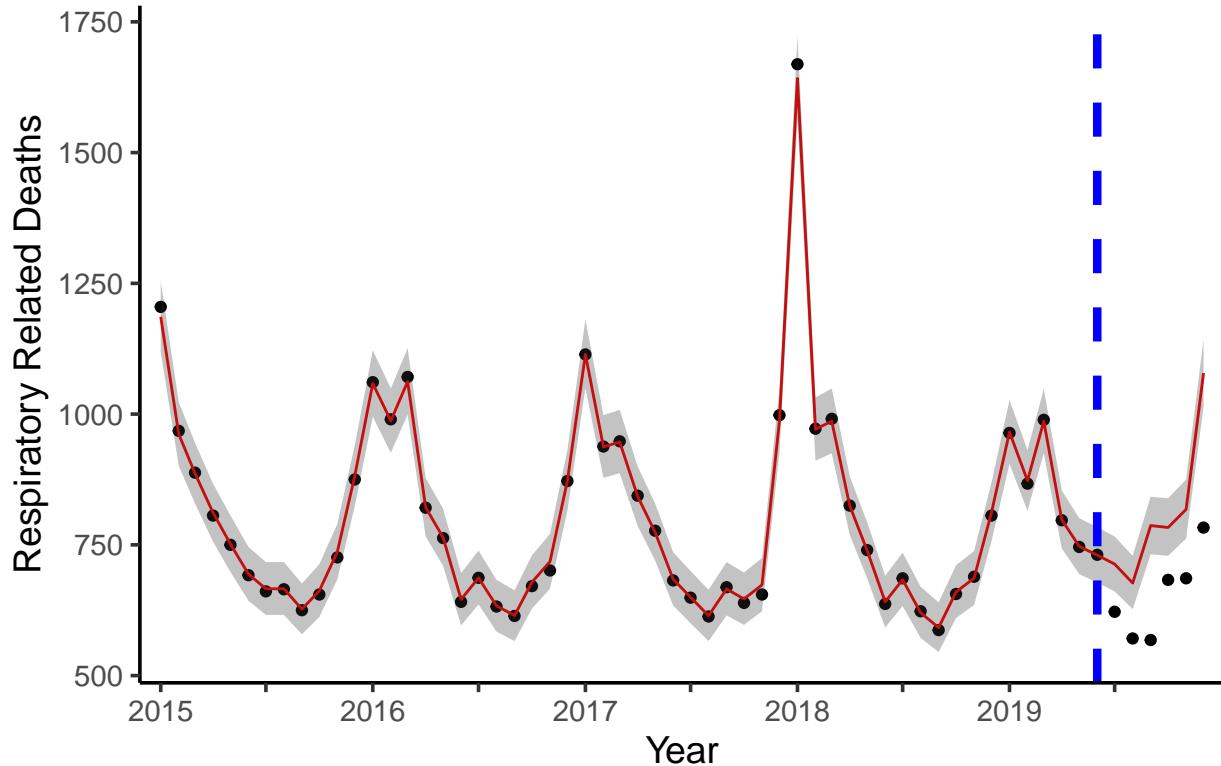
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 0%)



```
pp_outsample_plot_combined(pred_data = ref_model4_outfit$fitted_values, prefix = "ref4-outsample-")
```

Proposed model 1

```
#Fit kgr_model1
kgr_model1_outfit = kgr_model1(dataset = inla_outsample_data,rho_time_rbf = 63.888,rho_time_periodic = 1)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), :
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

#Posterior predictive sampling from estimated intensities
kgr_model1_outfit$fitted_values = poisson_pp_sampling(kgr_model1_outfit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model1_DIC = kgr_model1_outfit$model_DIC
kgr_model1_WAIC = kgr_model1_outfit$model_WAIC

#Get summaries of parameter estimates
kgr_model1_outfit$model_summary

##          mean      sd 0.025quant 0.5quant 0.975quant      mode
## months1    2.0980147 7.254728 -12.127788 2.0980147   16.32382 2.0980147
## months2    1.8695830 7.254730 -12.356223 1.8695830   16.09539 1.8695830
## months3    1.8914905 7.254730 -12.334316 1.8914905   16.11730 1.8914905
## months4    1.6970288 7.254732 -12.528781 1.6970288   15.92284 1.6970288
## months5    1.6170627 7.254733 -12.608749 1.6170627   15.84287 1.6170627
```

```

## months6    1.4933306 7.254734 -12.732484 1.4933306 15.71914 1.4933306
## months7    1.4550682 7.254741 -12.770759 1.4550682 15.68090 1.4550682
## months8    1.4333926 7.254741 -12.792435 1.4333926 15.65922 1.4333926
## months9    1.3922580 7.254742 -12.833571 1.3922580 15.61809 1.3922580
## months10   1.4728771 7.254740 -12.752950 1.4728771 15.69870 1.4728771
## months11   1.5157029 7.254740 -12.710122 1.5157029 15.74153 1.5157029
## months12   1.7749696 7.254736 -12.450849 1.7749696 16.00079 1.7749696
## Intercept1  4.2849839 7.254724 -9.940811 4.2849839 18.51078 4.2849839
## Intercept2  2.1517184 7.254743 -12.074113 2.1517184 16.37755 2.1517184
## Intercept3  2.0736402 7.254749 -12.152203 2.0736402 16.29948 2.0736402
## Intercept4  3.6474256 7.254729 -10.578379 3.6474256 17.87323 3.6474256
## Intercept5  1.8604876 7.254750 -12.365357 1.8604876 16.08633 1.8604876
## Intercept6  0.6824678 7.254833 -13.543539 0.6824678 14.90847 0.6824678
## Intercept7  5.0100553 7.254718 -9.215727 5.0100553 19.23584 5.0100553
##                               kld
## months1    5.526824e-11
## months2    5.526823e-11
## months3    5.526823e-11
## months4    5.526823e-11
## months5    5.526823e-11
## months6    5.526823e-11
## months7    5.526817e-11
## months8    5.526811e-11
## months9    5.526822e-11
## months10   5.526812e-11
## months11   5.526822e-11
## months12   5.526812e-11
## Intercept1 5.526822e-11
## Intercept2 5.526818e-11
## Intercept3 5.526816e-11
## Intercept4 5.526819e-11
## Intercept5 5.526817e-11
## Intercept6 5.526823e-11
## Intercept7 5.526817e-11
kgr_model1_outfit$bri_hyperpar_summary

##               mean        sd      q0.025      q0.5      q0.975       mode
## SD for id2 0.6336763 0.04940089 0.5412813 0.6319565 0.7353104 0.6286964
kgr_model1_outfit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##  8.149974  6.485591  6.629242  5.457707  5.038270  4.451899  4.284776
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##  4.192900  4.023926  4.361766  4.552620  5.900102  72.601376  8.599624
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##  7.953723 38.375744  6.426870  1.978755 149.913028
kgr_model1_outfit$K_time_weight

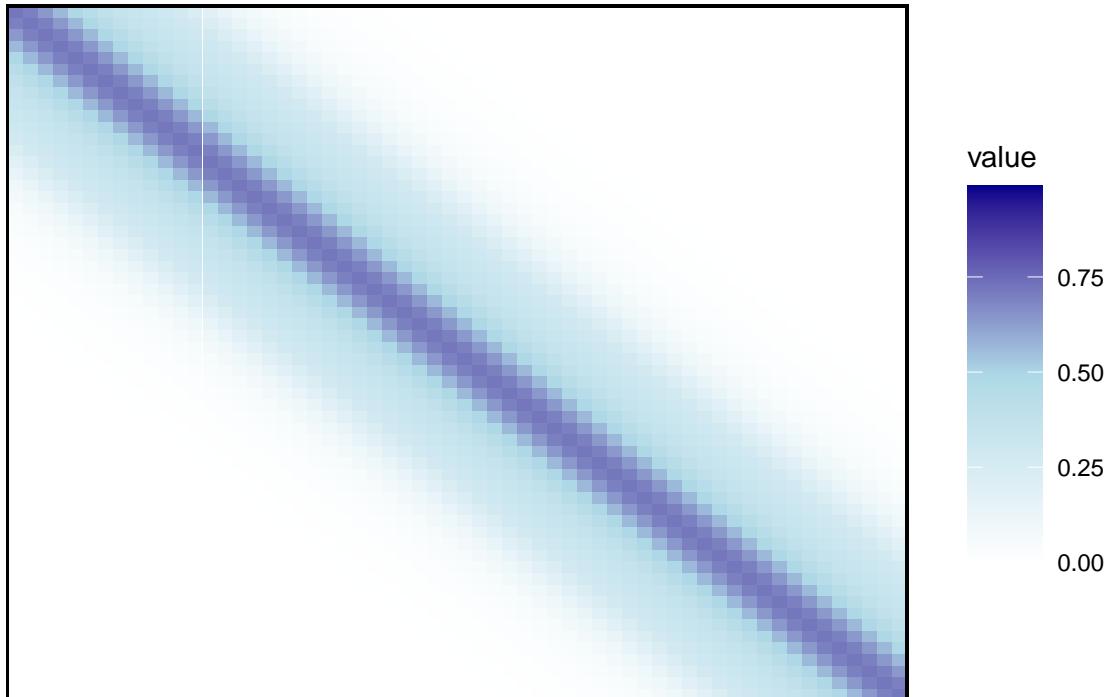
## [1] 0.5531353
kgr_model1_outfit$gfilter_weight

## [1] 0.4468647

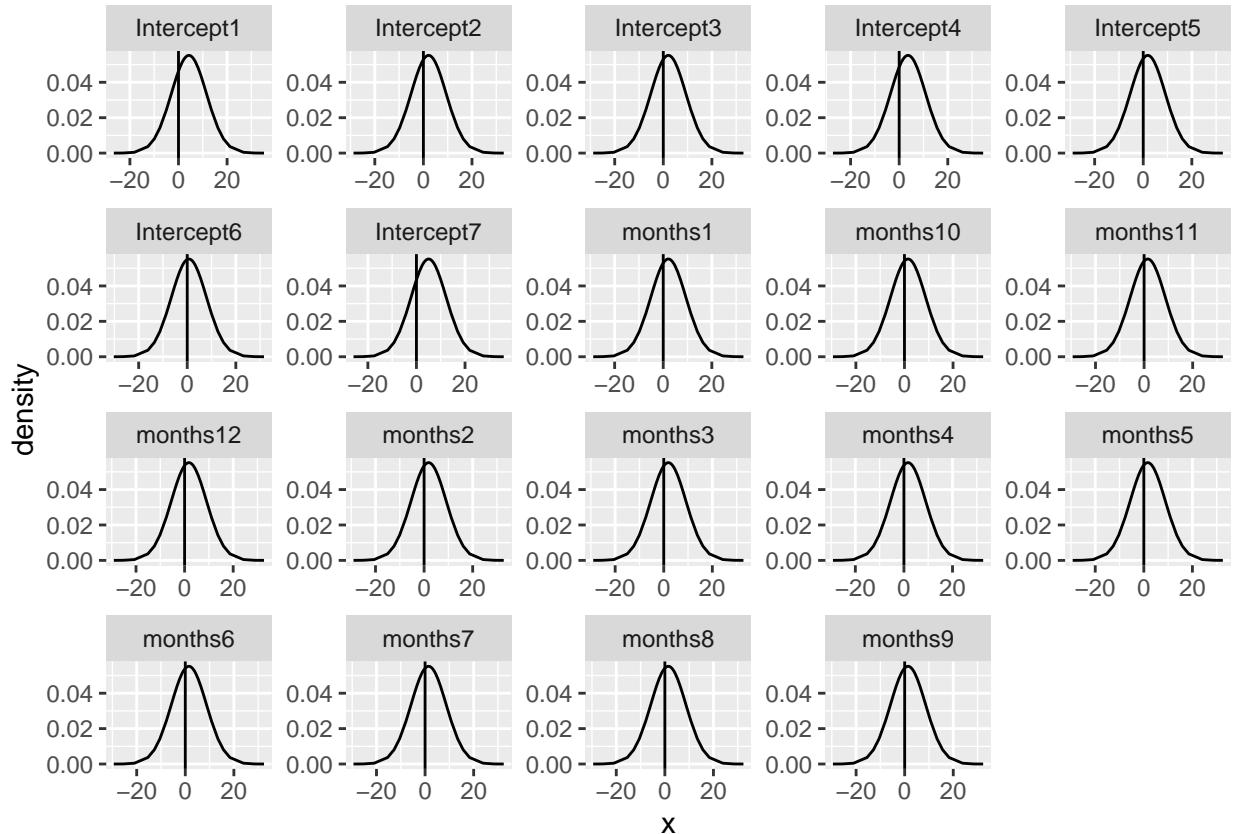
```

```
#Show plots  
kgr_model1_outfit$K_time_heatmap  
  
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha  
## i Please consider using `annotate()` or provide this layer with data containing  
## a single row.
```

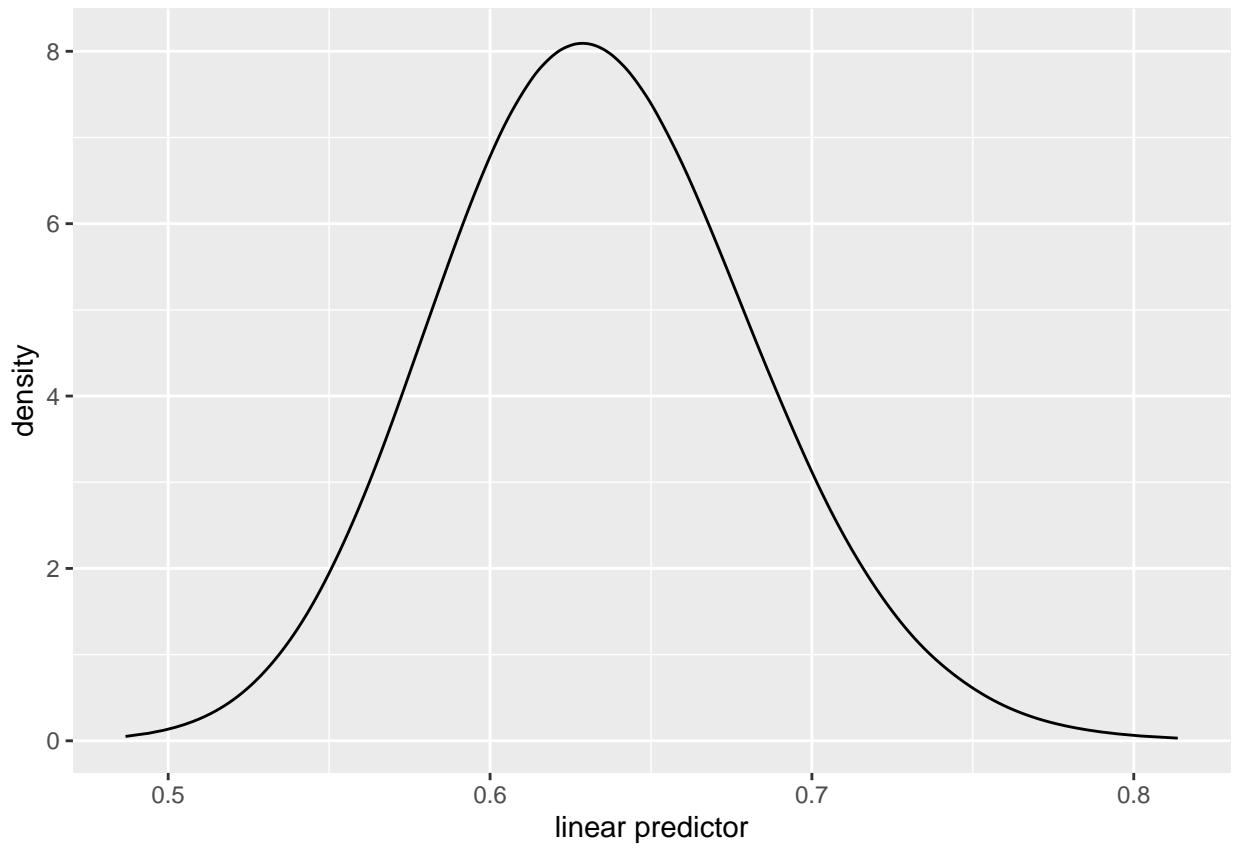
K_time heatmap



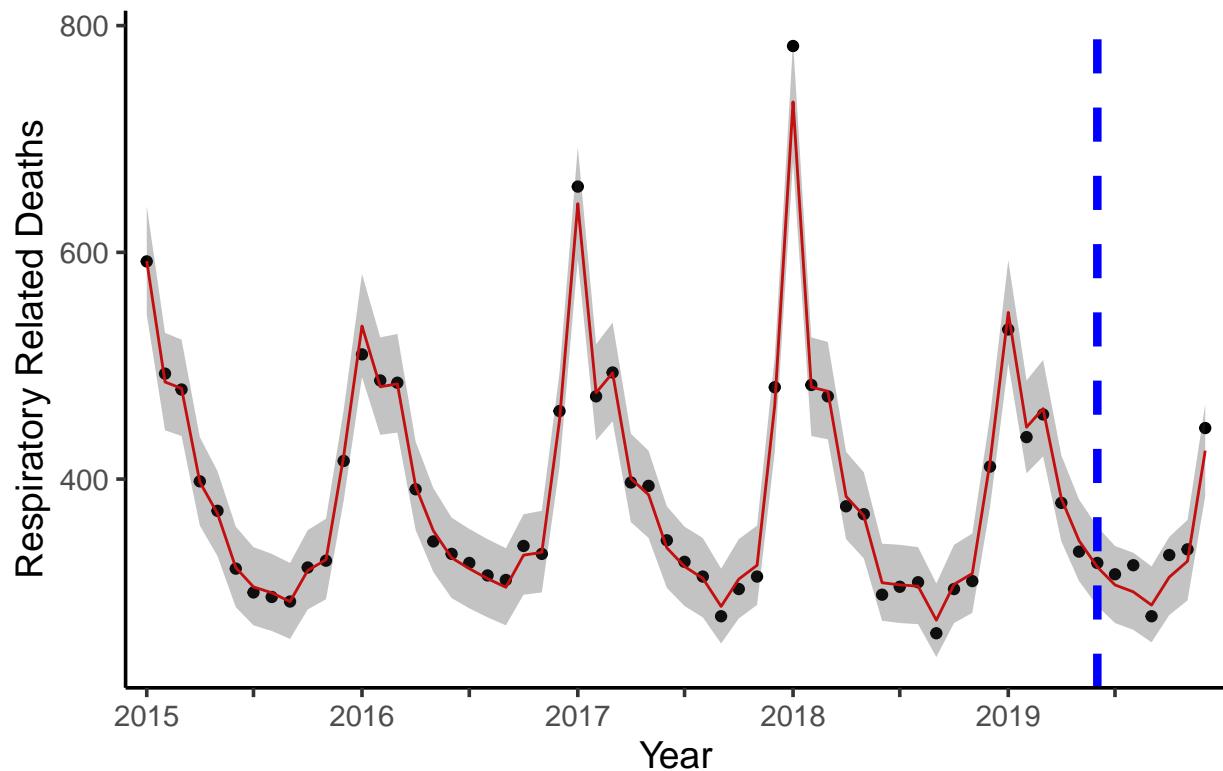
```
kgr_model1_outfit$param_plot
```



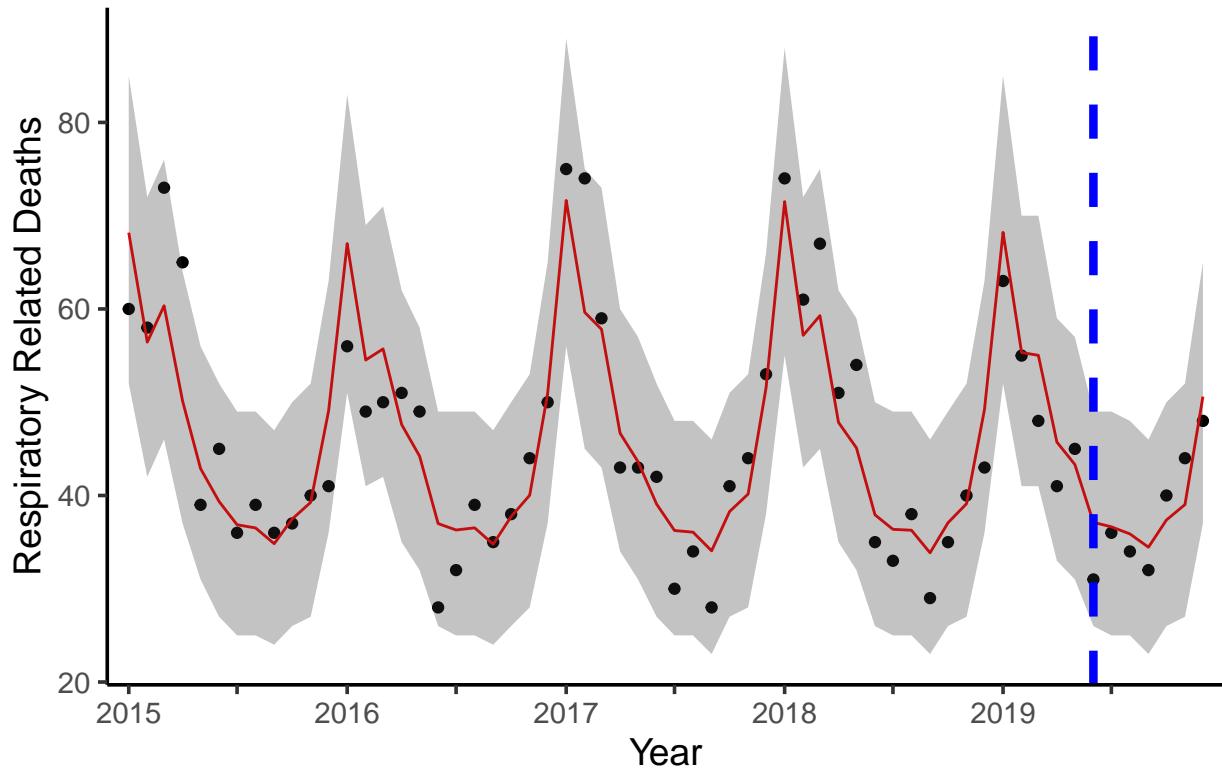
```
kgr_model1_outfit$hyperparam_plot
```



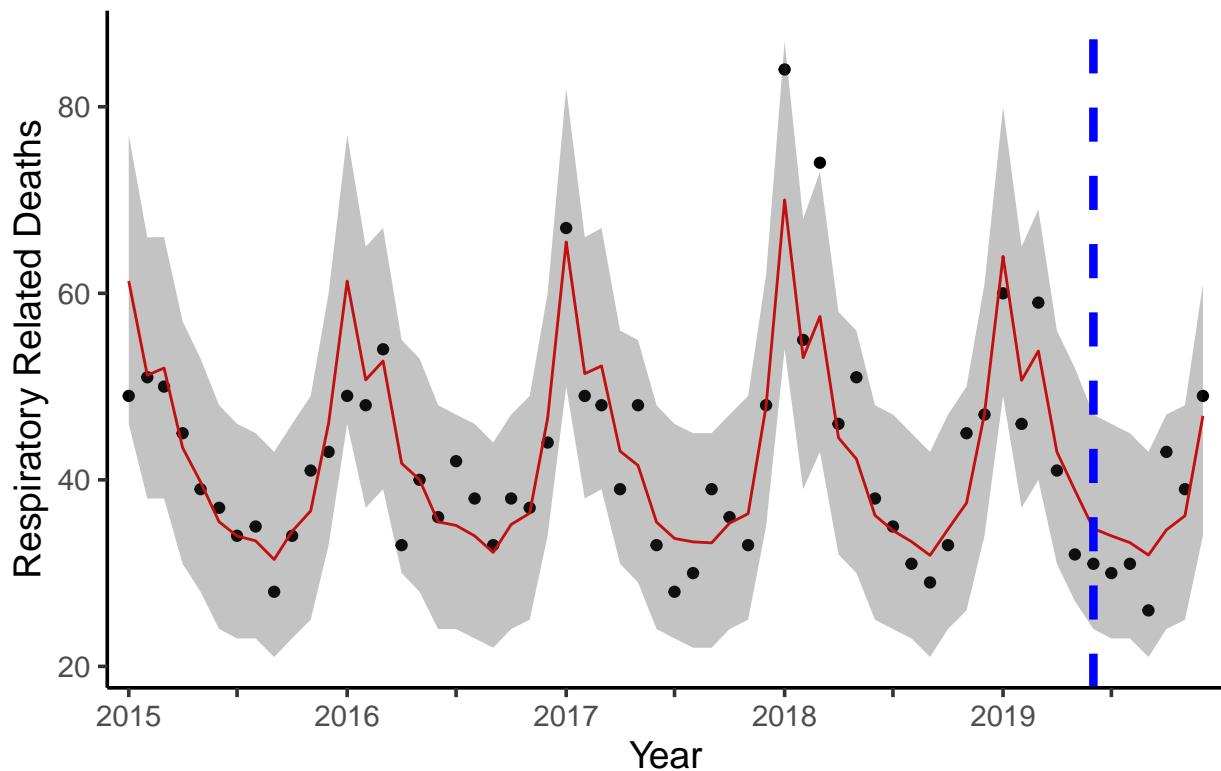
Cluster 1 (Out of Sample Coverage: 100%)



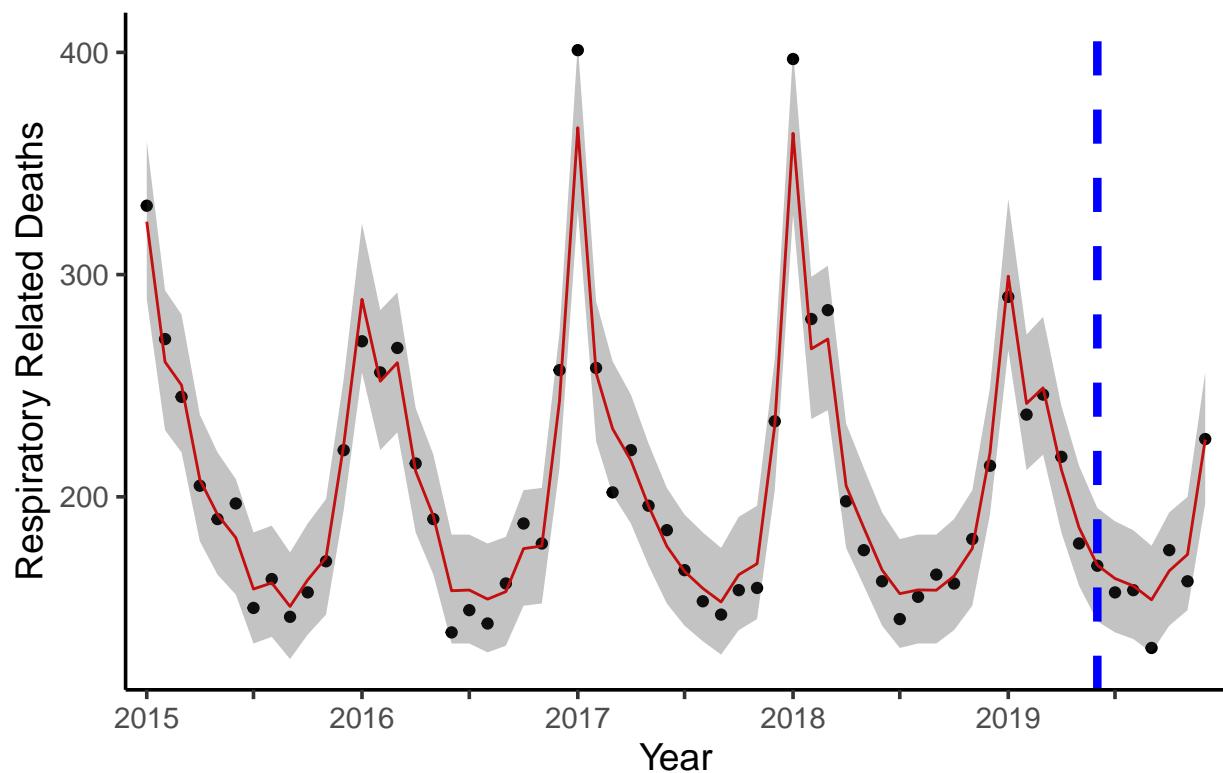
Cluster 2 (Out of Sample Coverage: 100%)



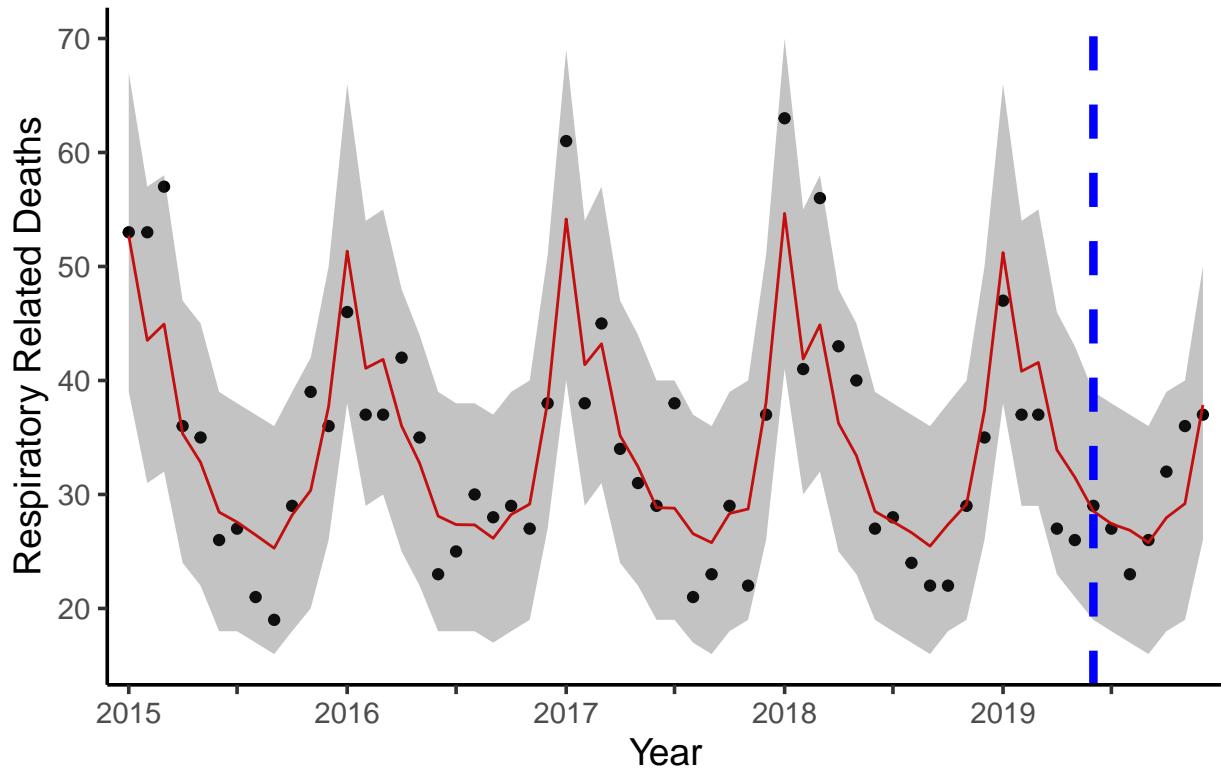
Cluster 3 (Out of Sample Coverage: 100%)



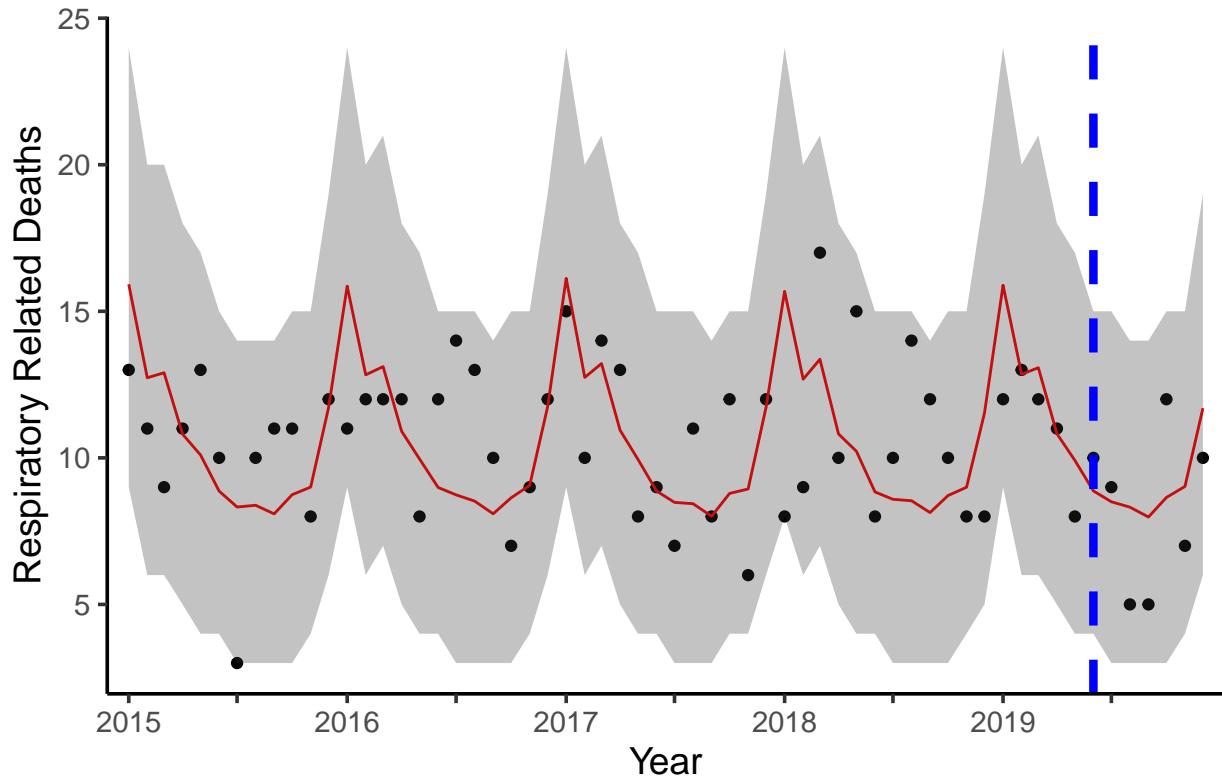
Cluster 4 (Out of Sample Coverage: 100%)



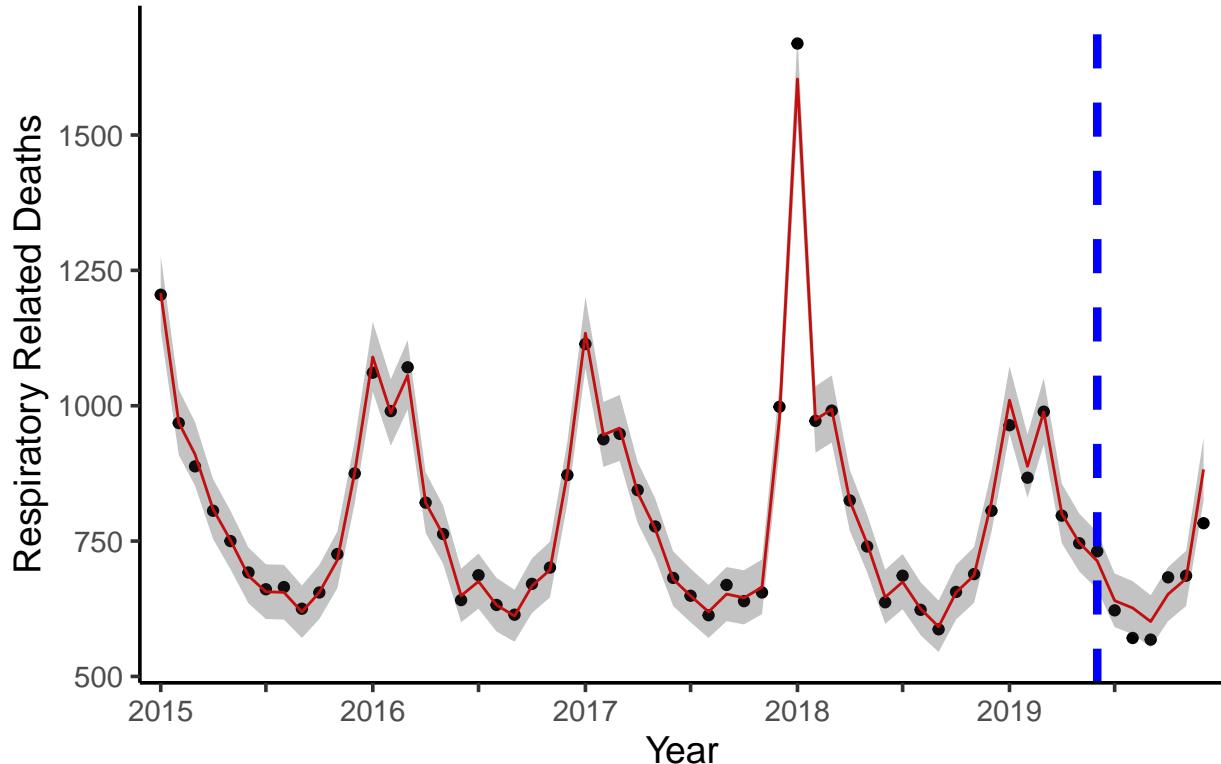
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 66.67%)



```
pp_outsample_plot_combined(pred_data = kgr_model1_outfit$fitted_values,prefix = "kgr1-outsample-")
```

Proposed model 3

```
#Fit kgr_model3
kgr_model3_outfit = kgr_model3(dataset = inla_outsample_data, rho_EPA_rbf = 77.429, rho_EPA_periodic =
                                rho_time_rbf = 6802.120, rho_time_periodic = 8977.554, sigma2 = 4.997, link=
```

#Posterior predictive sampling from estimated intensities

```
kgr_model3_outfit$fitted_values = poisson_pp_sampling(kgr_model3_outfit$fitted_values,n=100000)
```

#Extract DIC and WAIC

```
kgr_model3_DIC = kgr_model3_outfit$model_DIC
kgr_model3_WAIC = kgr_model3_outfit$model_WAIC
```

#Get summaries of parameter estimates

```
kgr_model3_outfit$model_summary
```

	mean	sd	0.025quant	0.5quant	0.975quant	mode
## months1	2.0982847	7.254730	-12.127520	2.0982847	16.32409	2.0982847
## months2	1.8695515	7.254731	-12.356257	1.8695515	16.09536	1.8695515
## months3	1.8912521	7.254731	-12.334556	1.8912521	16.11706	1.8912521
## months4	1.6963241	7.254733	-12.529488	1.6963241	15.92214	1.6963241
## months5	1.6160699	7.254734	-12.609744	1.6160699	15.84188	1.6160699
## months6	1.4917346	7.254735	-12.734082	1.4917346	15.71755	1.4917346
## months7	1.4551717	7.254742	-12.770658	1.4551717	15.68100	1.4551717

```

## months8    1.4338775 7.254742 -12.791953 1.4338775   15.65971 1.4338775
## months9    1.3925864 7.254743 -12.833245 1.3925864   15.61842 1.3925864
## months10   1.4734797 7.254742 -12.752350 1.4734797   15.69931 1.4734797
## months11   1.5165402 7.254741 -12.709288 1.5165402   15.74237 1.5165402
## months12   1.7759570 7.254738 -12.449864 1.7759570   16.00178 1.7759570
## Intercept1  4.2846622 7.254756 -9.941194 4.2846622   18.51052 4.2846622
## Intercept2  2.1517428 7.254759 -12.074120 2.1517428   16.37761 2.1517428
## Intercept3  2.0742577 7.254782 -12.151650 2.0742577   16.30017 2.0742577
## Intercept4  3.6470499 7.254770 -10.578834 3.6470499   17.87293 3.6470499
## Intercept5  1.8604659 7.254759 -12.365398 1.8604659   16.08633 1.8604659
## Intercept6  0.6824671 7.254877 -13.543628 0.6824671   14.90856 0.6824671
## Intercept7  5.0101834 7.254727 -9.215618 5.0101834   19.23598 5.0101834
##                      kld
## months1    5.526823e-11
## months2    5.526823e-11
## months3    5.526822e-11
## months4    5.526823e-11
## months5    5.526817e-11
## months6    5.526817e-11
## months7    5.526817e-11
## months8    5.526817e-11
## months9    5.526817e-11
## months10   5.526817e-11
## months11   5.526822e-11
## months12   5.526822e-11
## Intercept1 5.526817e-11
## Intercept2 5.526823e-11
## Intercept3 5.526823e-11
## Intercept4 5.526820e-11
## Intercept5 5.526817e-11
## Intercept6 5.526823e-11
## Intercept7 5.526817e-11
kgr_model3_outfit$bri_hyperpar_summary

##                  mean        sd      q0.025      q0.5      q0.975      mode
## SD for id2 0.6520801 0.05114907 0.5563814 0.6503115 0.7572779 0.6469541
kgr_model3_outfit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##  8.152174   6.485387   6.627662   5.453862   5.033270   4.444799   4.285219
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##  4.194933   4.025247   4.364395   4.556433   5.905930  72.578023  8.599833
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##  7.958637  38.361329   6.426731   1.978753 149.932237
kgr_model3_outfit$K_weight

## [1] 0.8023821
kgr_model3_outfit$gfilter_weight

## [1] 0.1976179
#Show plots
kgr_model3_outfit$K_time_heatmap

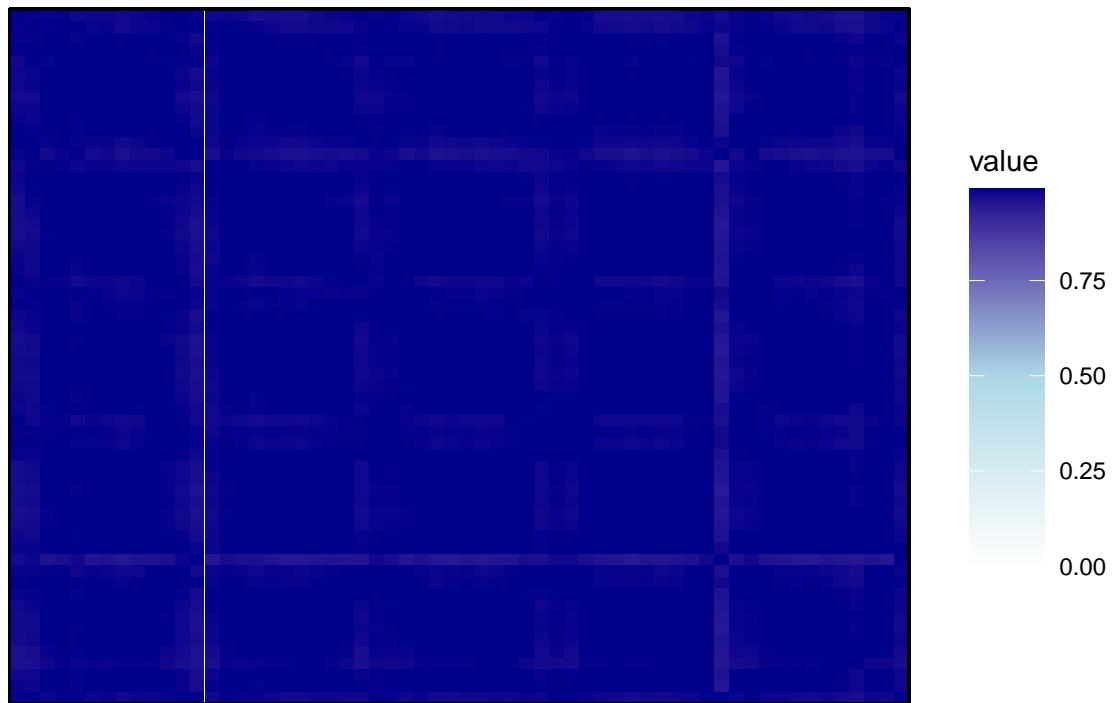
```

K_time heatmap

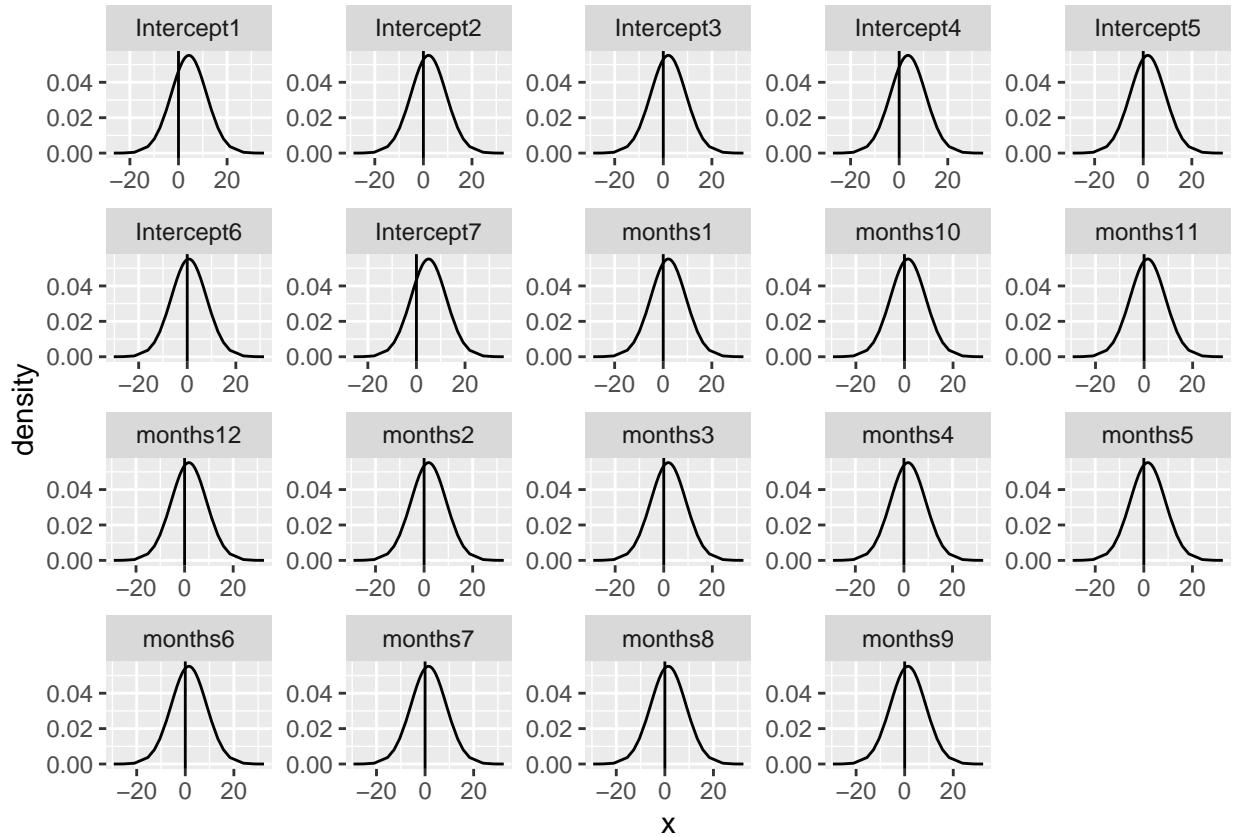


kgr_model3_outfit\$K_EPA_heatmap

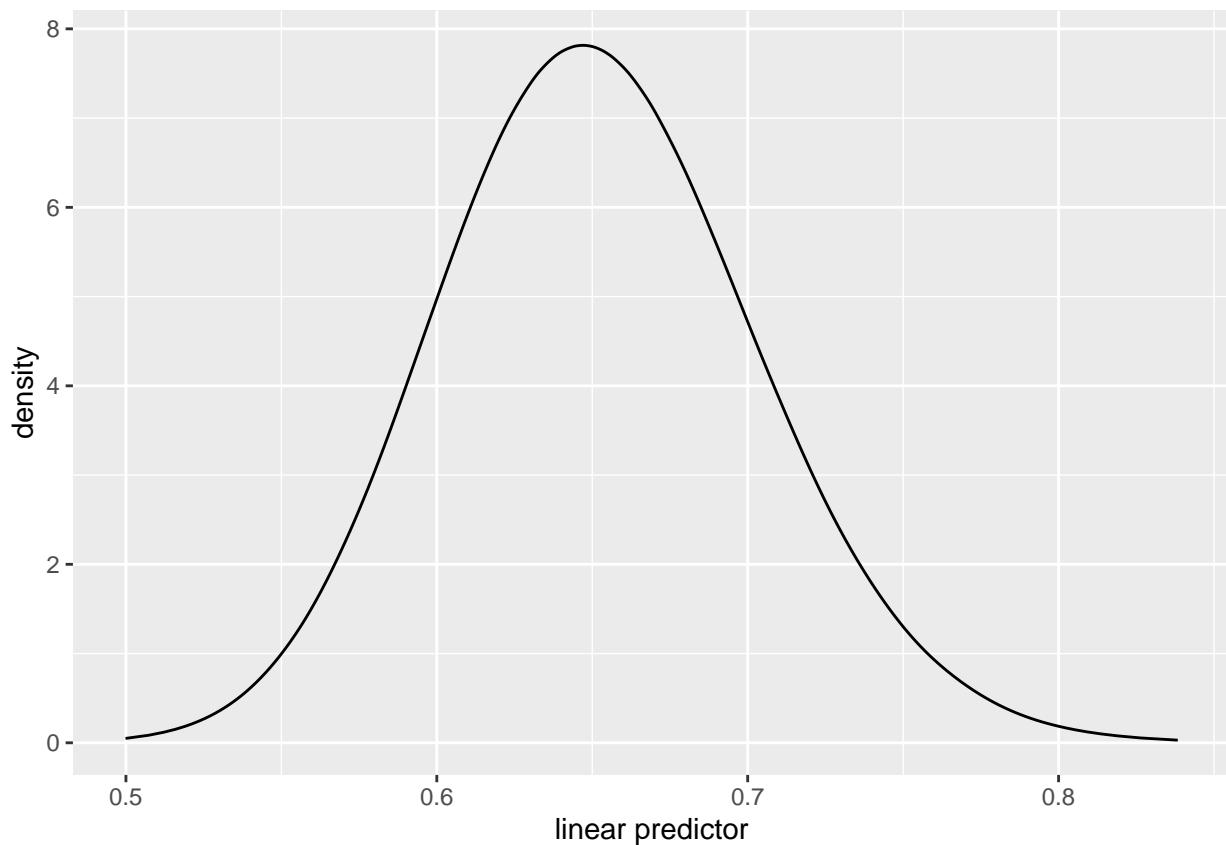
K_EPA heatmap



```
kgr_model3_outfit$param_plot
```

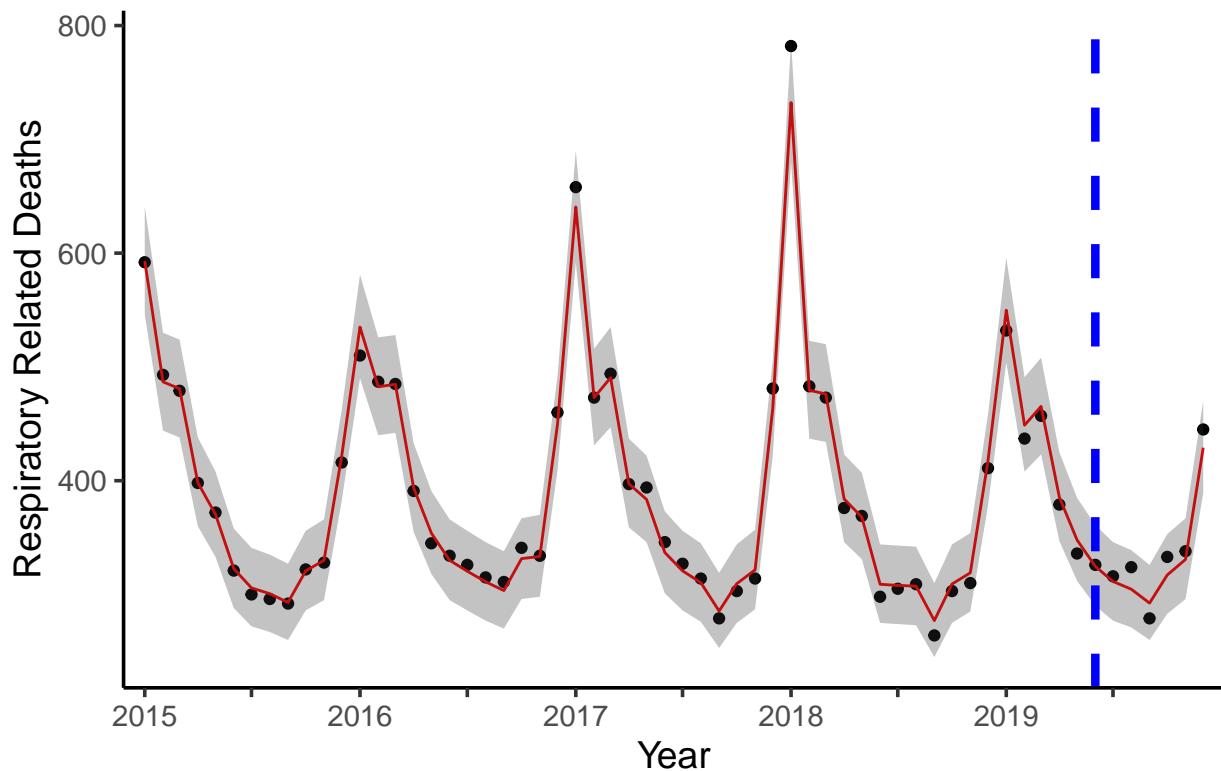


```
kgr_model3_outfit$hyperparam_plot
```

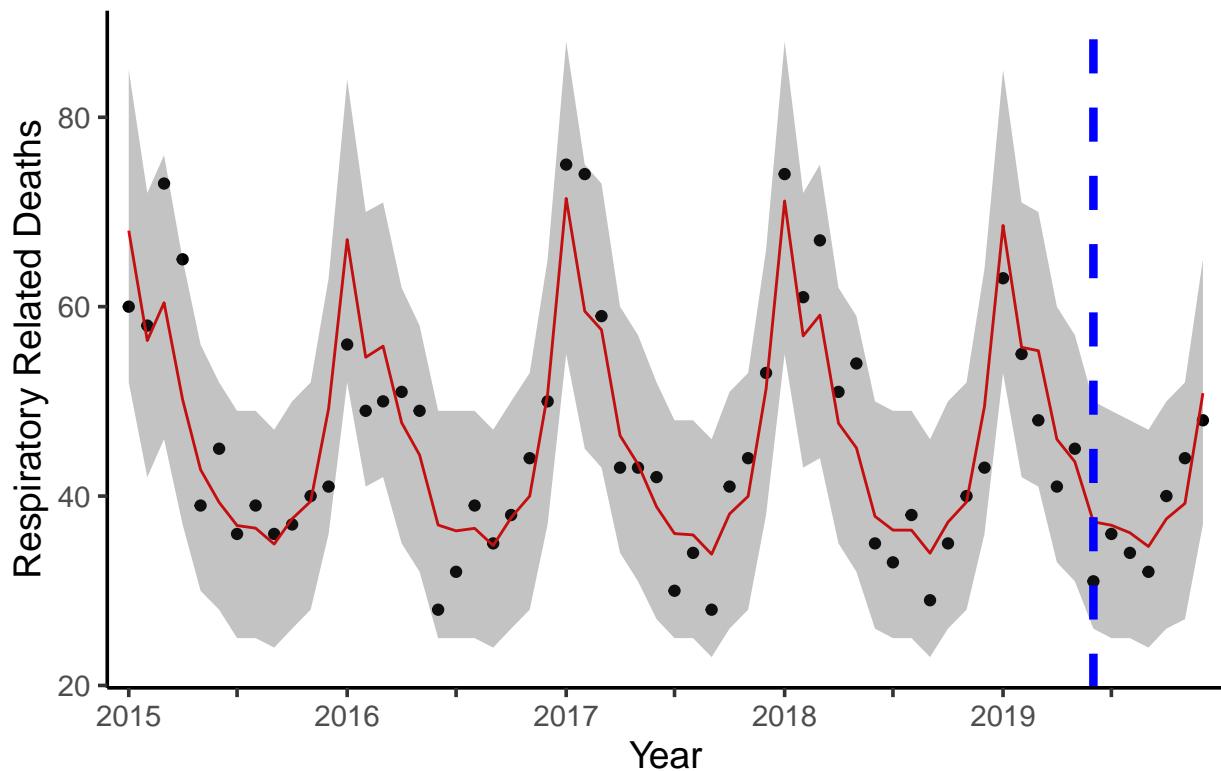


```
pp_outsample_plot(pred_data = kgr_model3_outfit$fitted_values,prefix = "kgr3-outsamp...")
```

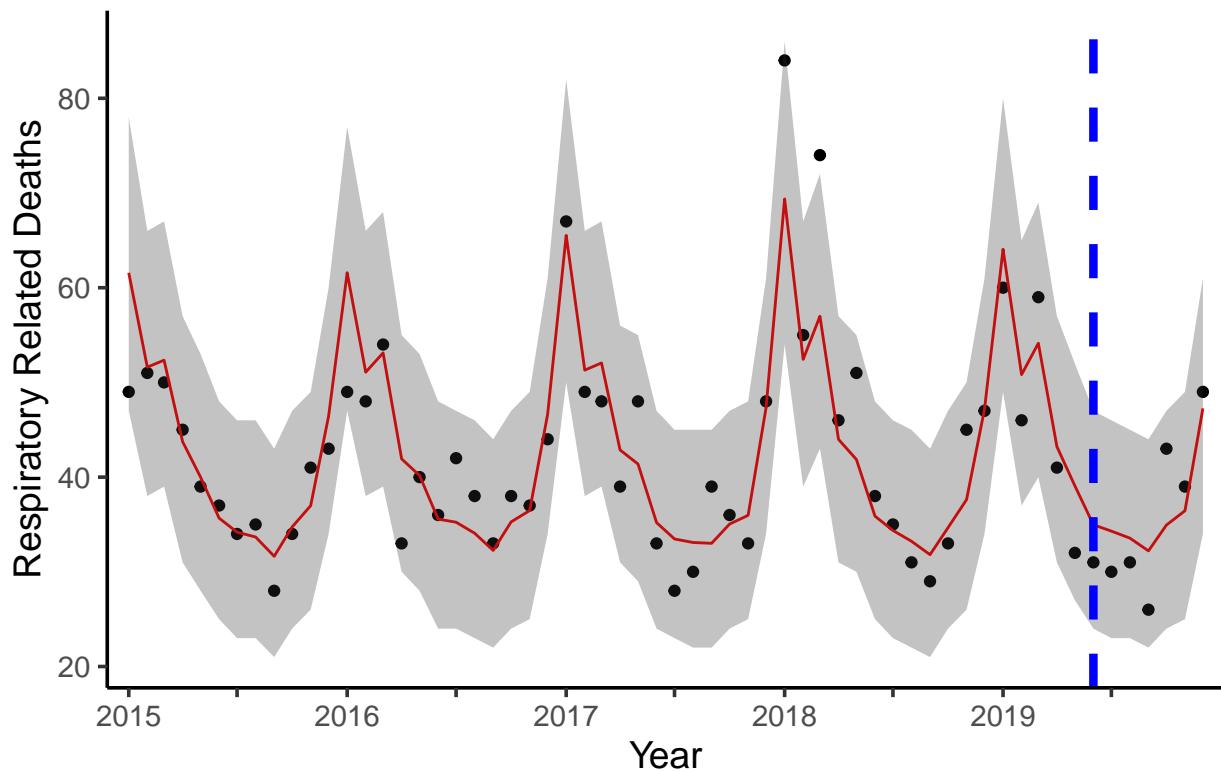
Cluster 1 (Out of Sample Coverage: 100%)



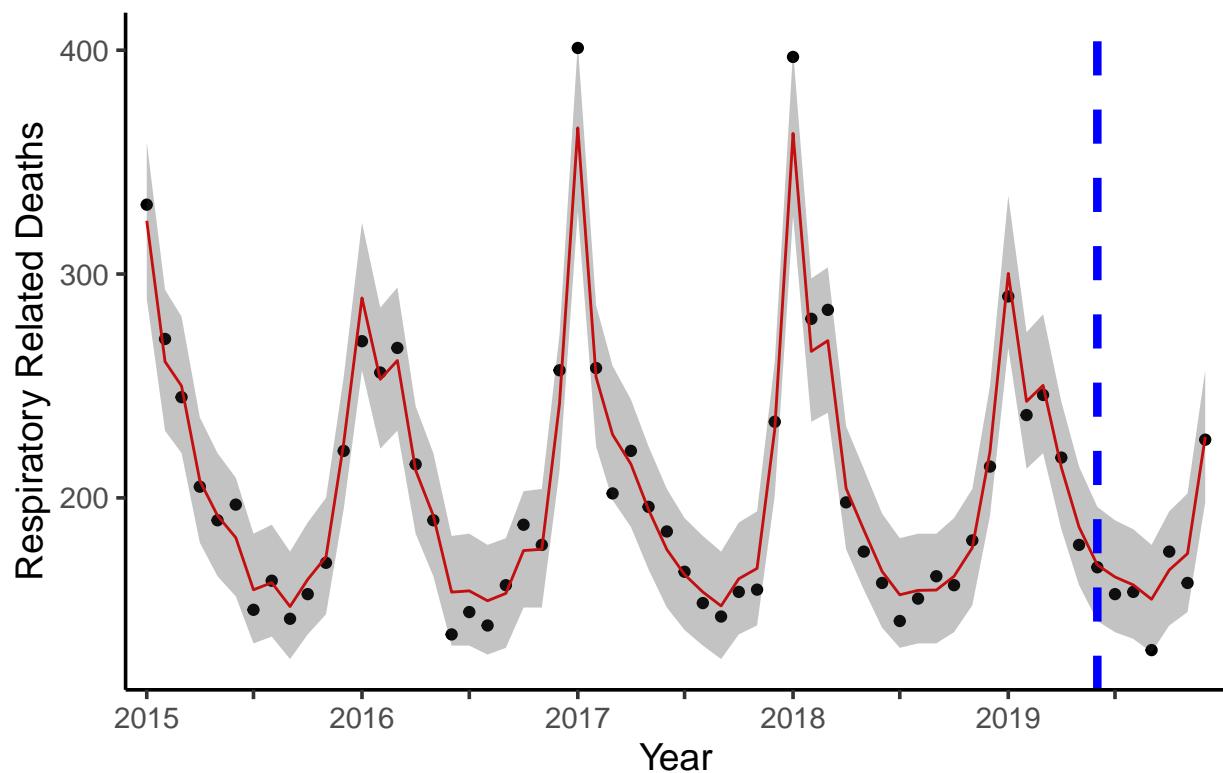
Cluster 2 (Out of Sample Coverage: 100%)



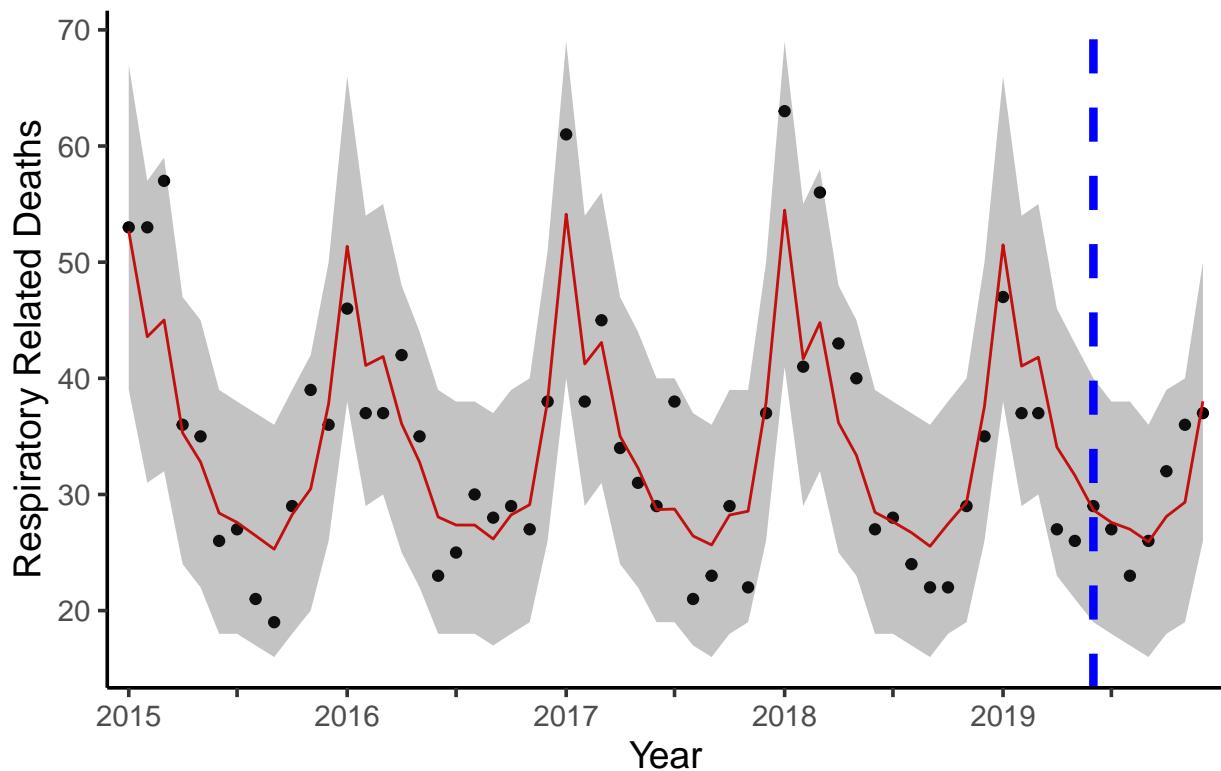
Cluster 3 (Out of Sample Coverage: 100%)



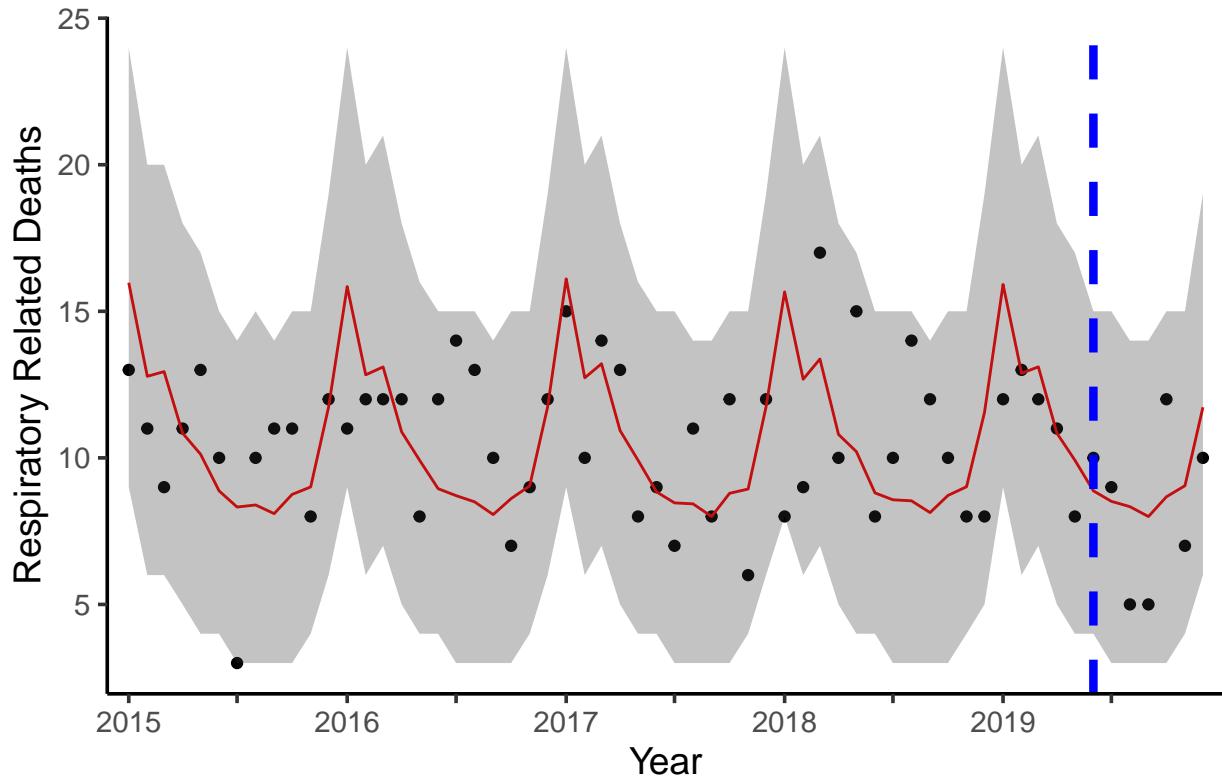
Cluster 4 (Out of Sample Coverage: 100%)



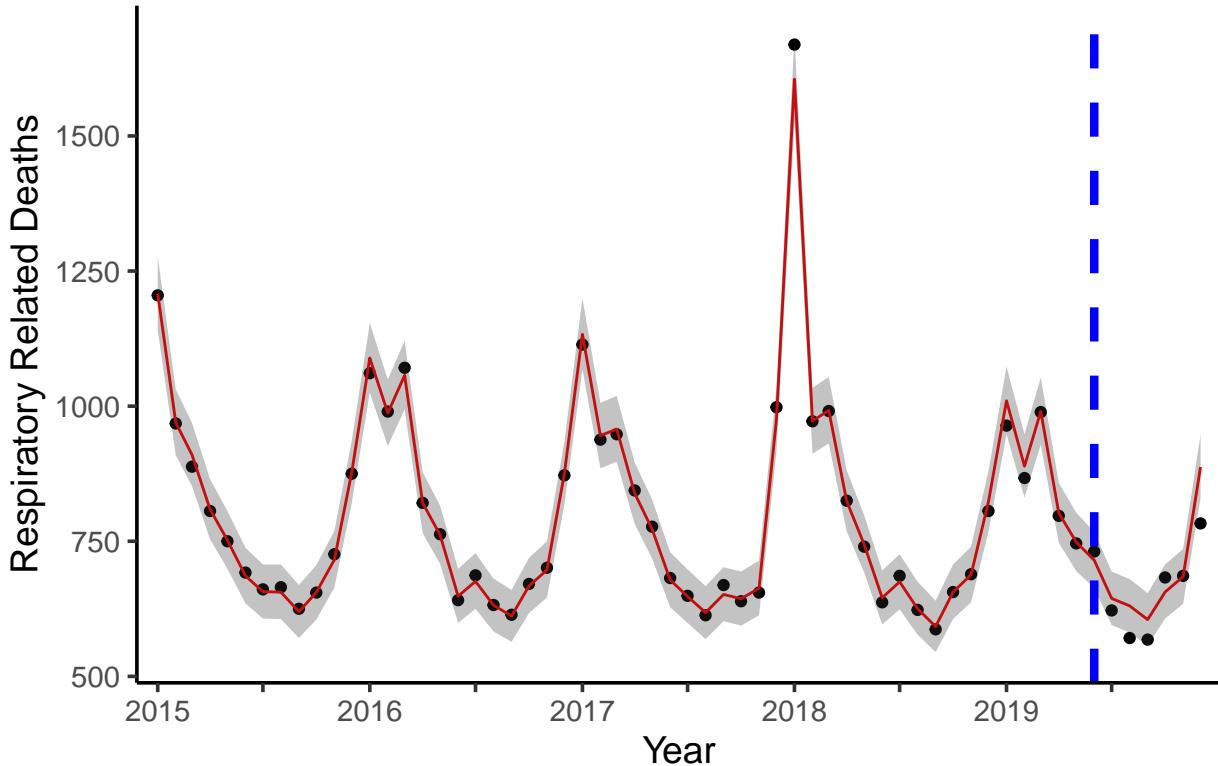
Cluster 5 (Out of Sample Coverage: 100%)



Cluster 6 (Out of Sample Coverage: 100%)



Cluster 7 (Out of Sample Coverage: 66.67%)



```
pp_outsample_plot_combined(pred_data = kgr_model3_outfit$fitted_values, prefix = "kgr3-outsample-")
```

Proposed model 4

```
#Fit kgr_model4
kgr_model4_outfit = kgr_model4(dataset = inla_outsample_data, rho_EPA_rbf = 259.326, rho_EPA_periodic =
                           rho_time_rbf = 2.699, rho_time_periodic = 86.005, sigma2_EPA = 3.215, sigma2_time = 1.25)

#Posterior predictive sampling from estimated intensities
kgr_model4_outfit$fitted_values = poisson_pp_sampling(kgr_model4_outfit$fitted_values, n=100000)

#Extract DIC and WAIC
kgr_model4_DIC = kgr_model4_outfit$model_DIC
kgr_model4_WAIC = kgr_model4_outfit$model_WAIC

#Get summaries of parameter estimates
kgr_model4_outfit$model_summary %>% kbl() %>% kable_styling()

kgr_model4_outfit$bri_hyperpar_summary %>% kbl() %>% kable_styling()

kgr_model4_outfit$exp_effects

##    months1     months2     months3     months4     months5     months6     months7
##  8.135682   6.474502   6.622827   5.460016   5.039901   4.454317   4.296872
##    months8     months9     months10    months11    months12 Intercept1 Intercept2
##  4.202169   4.034416   4.367885   4.553261   5.895512  72.711907   8.601006
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
```

	mean	sd	0.025quant	0.5quant	0.975quant	mode	kld
months1	2.096259	7.254729	-12.129544	2.096259	16.32206	2.096259	0
months2	1.867872	7.254731	-12.357936	1.867872	16.09368	1.867872	0
months3	1.890522	7.254731	-12.335285	1.890522	16.11633	1.890522	0
months4	1.697452	7.254732	-12.528359	1.697452	15.92326	1.697452	0
months5	1.617386	7.254733	-12.608426	1.617386	15.84320	1.617386	0
months6	1.493874	7.254735	-12.731941	1.493874	15.71969	1.493874	0
months7	1.457887	7.254741	-12.767941	1.457887	15.68372	1.457887	0
months8	1.435601	7.254741	-12.790228	1.435601	15.66143	1.435601	0
months9	1.394862	7.254742	-12.830968	1.394862	15.62069	1.394862	0
months10	1.474279	7.254741	-12.751548	1.474279	15.70011	1.474279	0
months11	1.515844	7.254740	-12.709983	1.515844	15.74167	1.515844	0
months12	1.774191	7.254737	-12.451628	1.774191	16.00001	1.774191	0
Intercept1	4.286505	7.254738	-9.939316	4.286505	18.51233	4.286505	0
Intercept2	2.151879	7.254749	-12.073965	2.151879	16.37772	2.151879	0
Intercept3	2.075035	7.254763	-12.150836	2.075035	16.30091	2.075035	0
Intercept4	3.647066	7.254747	-10.578773	3.647066	17.87291	3.647066	0
Intercept5	1.860954	7.254753	-12.364898	1.860954	16.08681	1.860954	0
Intercept6	0.683708	7.254852	-13.542337	0.683708	14.90975	0.683708	0
Intercept7	5.010882	7.254721	-9.214907	5.010882	19.23667	5.010882	0

	mean	sd	q0.025	q0.5	q0.975	mode
SD for id2	0.5982378	0.0461062	0.5120181	0.5966276	0.6931084	0.5935776

```

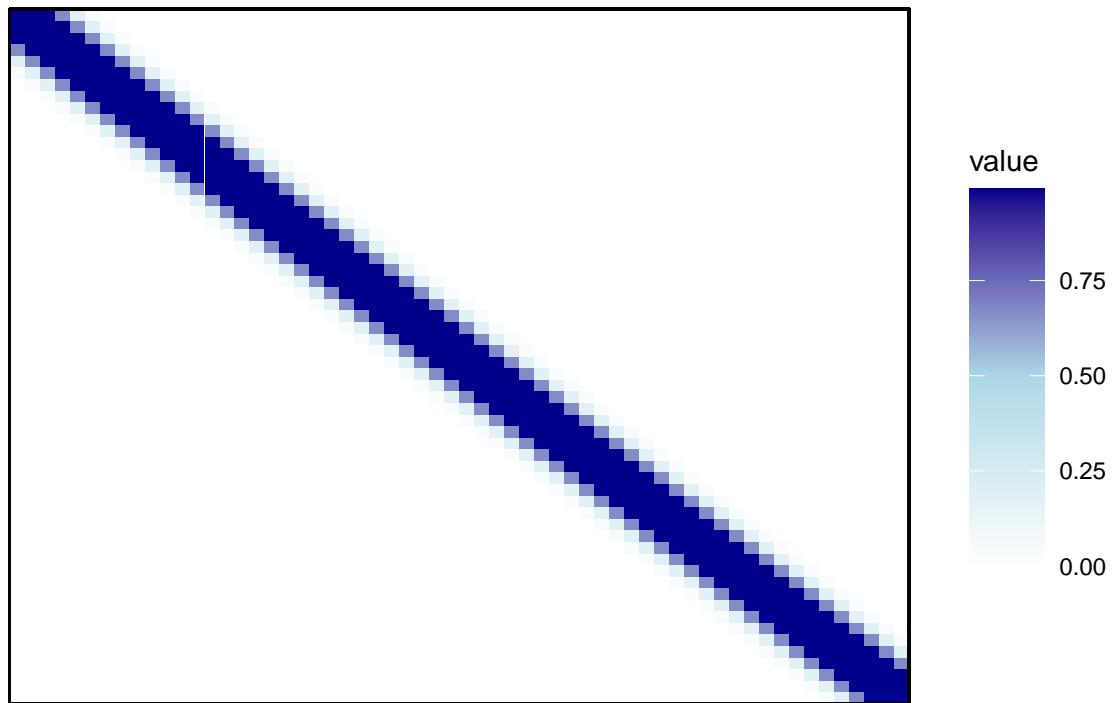
##    7.964823 38.361955  6.429867  1.981211 150.037010
kgr_model4_outfit$K_weight

## [1] 0.7550725
kgr_model4_outfit$gfilter_weight

## [1] 0.2449275
>Show plots
kgr_model4_outfit$K_time_heatmap

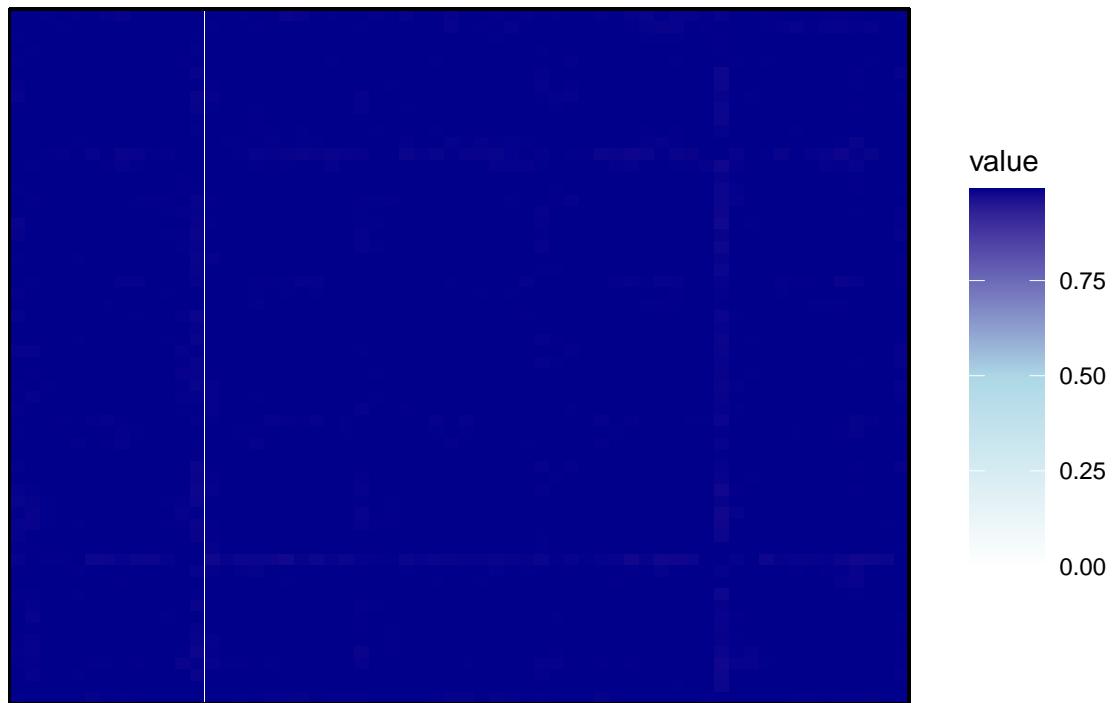
```

K_time heatmap

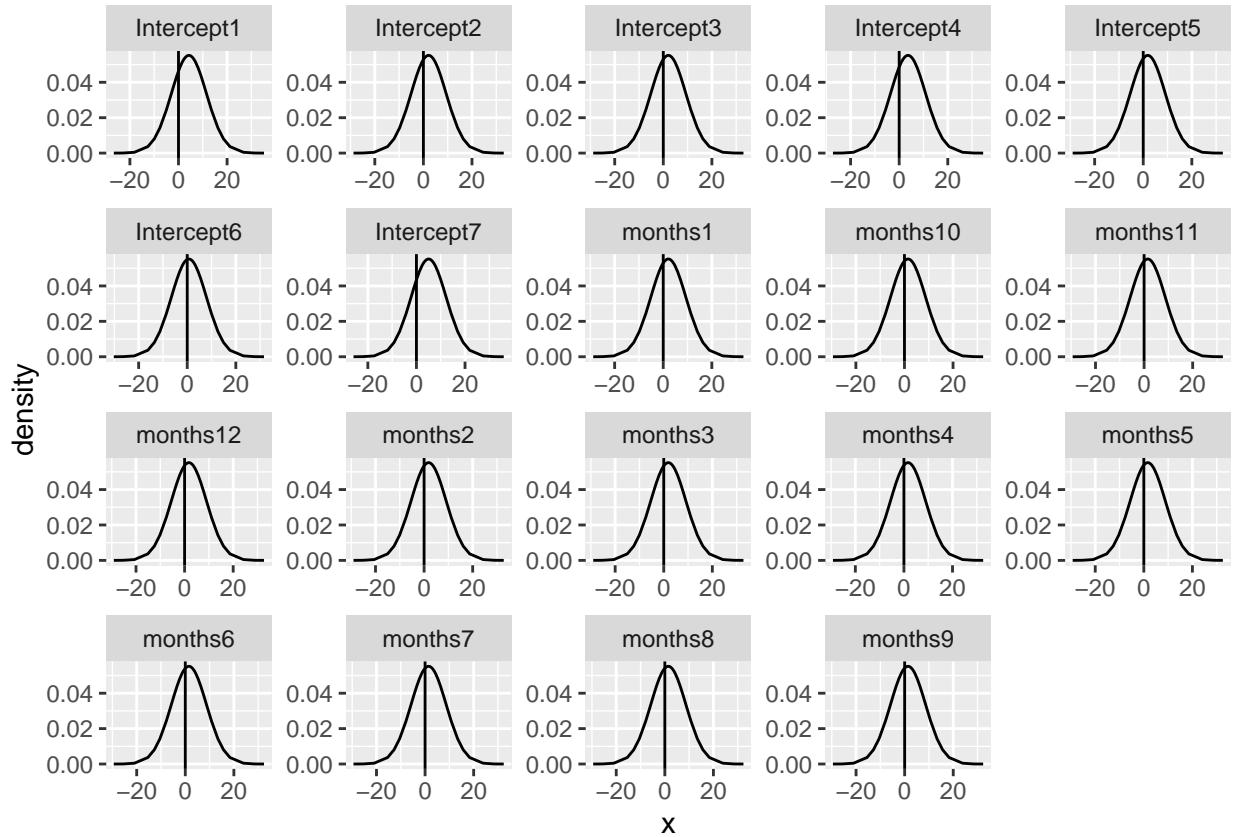


kgr_model4_outfit\$K_EPA_heatmap

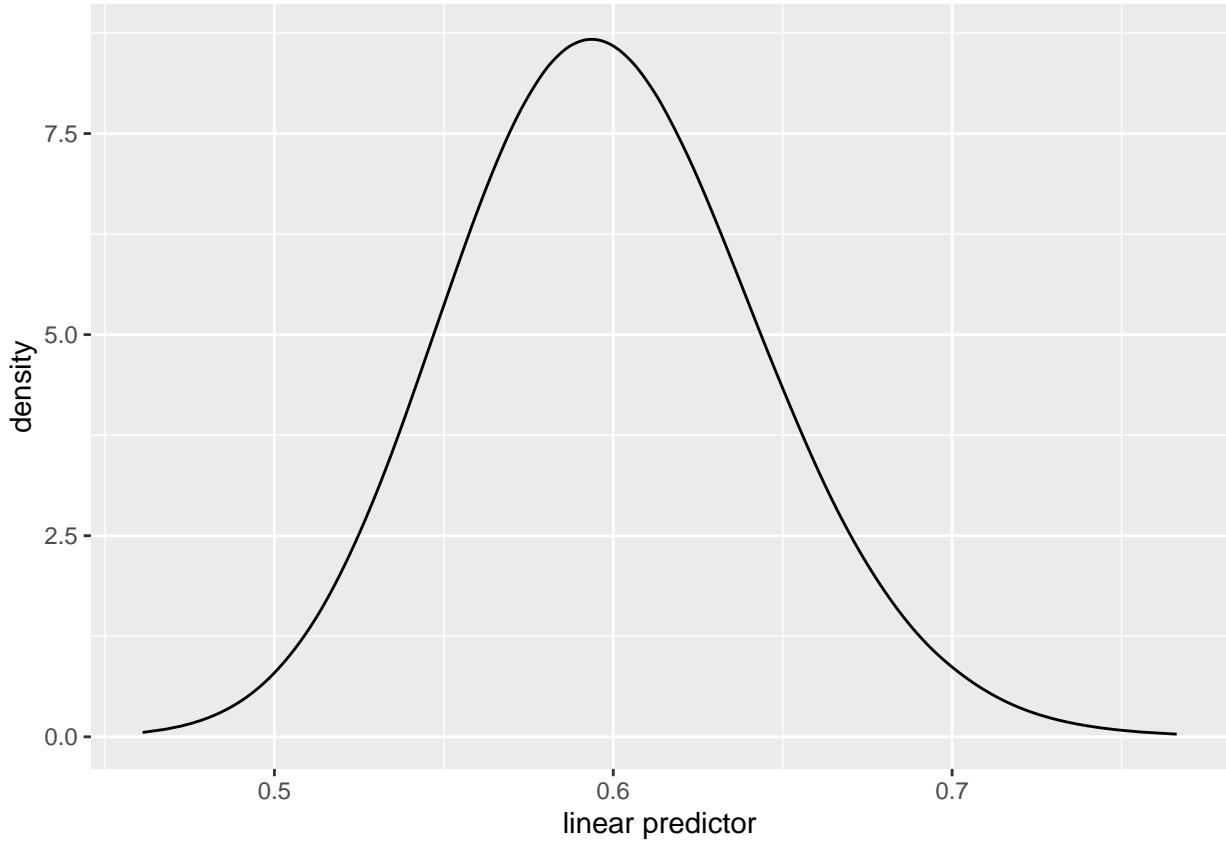
K_EPA heatmap



kgr_model4_outfit\$param_plot



```
kgr_model4_outfit$hyperparam_plot
```



```
#pp_outsample_plot(pred_data = kgr_model4_outfit$fitted_values,prefix = "kgr4-outsample-")
pp_outsample_plot_combined(pred_data = kgr_model4_outfit$fitted_values,prefix = "kgr4-outsample-nofe-")
```

Proposed model 5

```
#Fit kgr_model5
kgr_model5_outfit = kgr_model5(dataset = inla_outsample_data, rho_AR_rbf = 0.016, rho_AR_periodic = 0.013,
                                 rho_DL_rbf = 0.013, rho_DL_periodic = 0.003, rho_int_rbf = 0.002,
                                 rho_int_periodic = 0.011, sigma2_AR = 4.002, sigma2_DL = 0.486, sigma2_int = 0.001)

## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.

#Posterior predictive sampling from estimated intensities
kgr_model5_outfit$fitted_values = poisson_pp_sampling(kgr_model5_outfit$fitted_values,n=100000)

#Extract DIC and WAIC
kgr_model5_DIC = kgr_model5_outfit$model_DIC
kgr_model5_WAIC = kgr_model5_outfit$model_WAIC

#Get summaries of parameter estimates
kgr_model5_outfit$model_summary

##           mean      sd 0.025quant 0.5quant 0.975quant      mode
## months1    2.0953307 7.254730 -12.130475 2.0953307   16.32114 2.0953307
## months2    1.8677250 7.254731 -12.358084 1.8677250   16.09353 1.8677250
```

```

## months3    1.8913754 7.254731 -12.334433 1.8913754   16.11718 1.8913754
## months4    1.6974133 7.254733 -12.528399 1.6974133   15.92323 1.6974133
## months5    1.6169280 7.254734 -12.608886 1.6169280   15.84274 1.6169280
## months6    1.4929665 7.254735 -12.732850 1.4929665   15.71878 1.4929665
## months7    1.4582944 7.254742 -12.767535 1.4582944   15.68412 1.4582944
## months8    1.4358546 7.254742 -12.789976 1.4358546   15.66169 1.4358546
## months9    1.3937548 7.254743 -12.832076 1.3937548   15.61959 1.3937548
## months10   1.4741204 7.254741 -12.751708 1.4741204   15.69995 1.4741204
## months11   1.5174906 7.254741 -12.708337 1.5174906   15.74332 1.5174906
## months12   1.7748719 7.254738 -12.450949 1.7748719   16.00069 1.7748719
## Intercept1  4.2889358 7.254727 -9.936864 4.2889358   18.51474 4.2889358
## Intercept2  2.1515814 7.254744 -12.074252 2.1515814   16.37741 2.1515814
## Intercept3  2.0712802 7.254752 -12.154568 2.0712802   16.29713 2.0712802
## Intercept4  3.6581737 7.254733 -10.567637 3.6581737   17.88398 3.6581737
## Intercept5  1.8605166 7.254750 -12.365329 1.8605166   16.08636 1.8605166
## Intercept6  0.6781004 7.254837 -13.547915 0.6781004   14.90412 0.6781004
## Intercept7  5.0075374 7.254718 -9.218246 5.0075374   19.23332 5.0075374
##                      kld
## months1    5.526816e-11
## months2    5.526823e-11
## months3    5.526823e-11
## months4    5.526818e-11
## months5    5.526822e-11
## months6    5.526822e-11
## months7    5.526817e-11
## months8    5.526822e-11
## months9    5.526817e-11
## months10   5.526823e-11
## months11   5.526818e-11
## months12   5.526817e-11
## Intercept1 5.526823e-11
## Intercept2 5.526817e-11
## Intercept3 5.526816e-11
## Intercept4 5.526815e-11
## Intercept5 5.526823e-11
## Intercept6 5.526823e-11
## Intercept7 5.526813e-11

kgr_model5_outfit$bri_hyperpar_summary

##               mean        sd      q0.025     q0.5     q0.975       mode
## SD for id2 0.5871709 0.04499063 0.5029803 0.5856186 0.6796956 0.5826715

kgr_model5_outfit$exp_effects

##    months1    months2    months3    months4    months5    months6    months7
##    8.128128   6.473552   6.628479   5.459806   5.037591   4.450278   4.298622
##    months8    months9    months10   months11   months12 Intercept1 Intercept2
##    4.203235   4.029953   4.367193   4.560766   5.899525  72.888861   8.598445
## Intercept3 Intercept4 Intercept5 Intercept6 Intercept7
##    7.934975  38.790433   6.427056   1.970132 149.536030

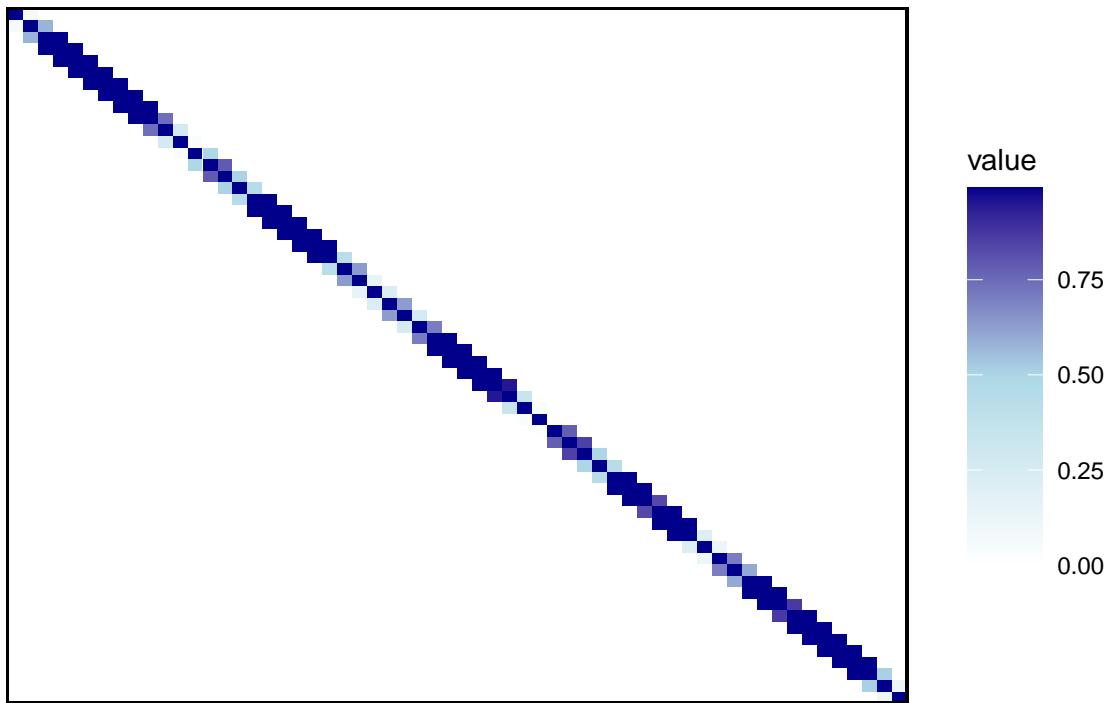
kgr_model5_outfit$K_weight

## [1] 0.6824774

```

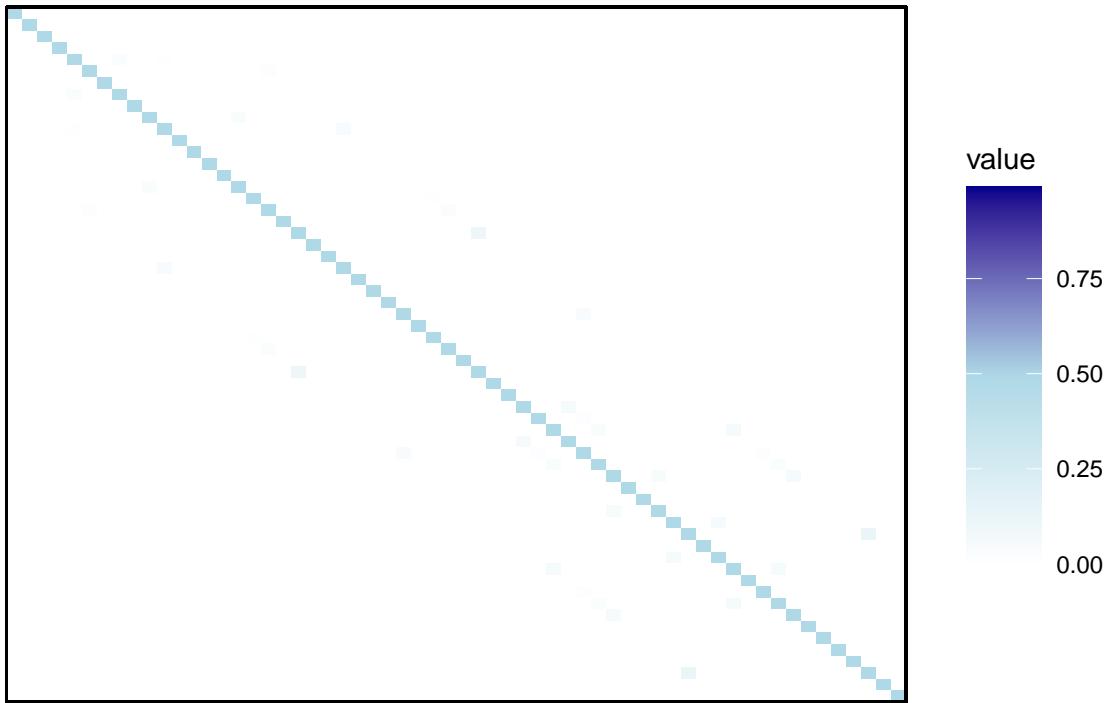
```
kgr_model5_outfit$gfilter_weight  
## [1] 0.3175226  
#Show plots  
kgr_model5_outfit$K_AR_heatmap  
  
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha  
## i Please consider using `annotate()` or provide this layer with data containing  
## a single row.
```

Periodic AR 1 Covariance Structure



```
kgr_model5_outfit$K_DL_heatmap  
  
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics ha  
## i Please consider using `annotate()` or provide this layer with data containing  
## a single row.
```

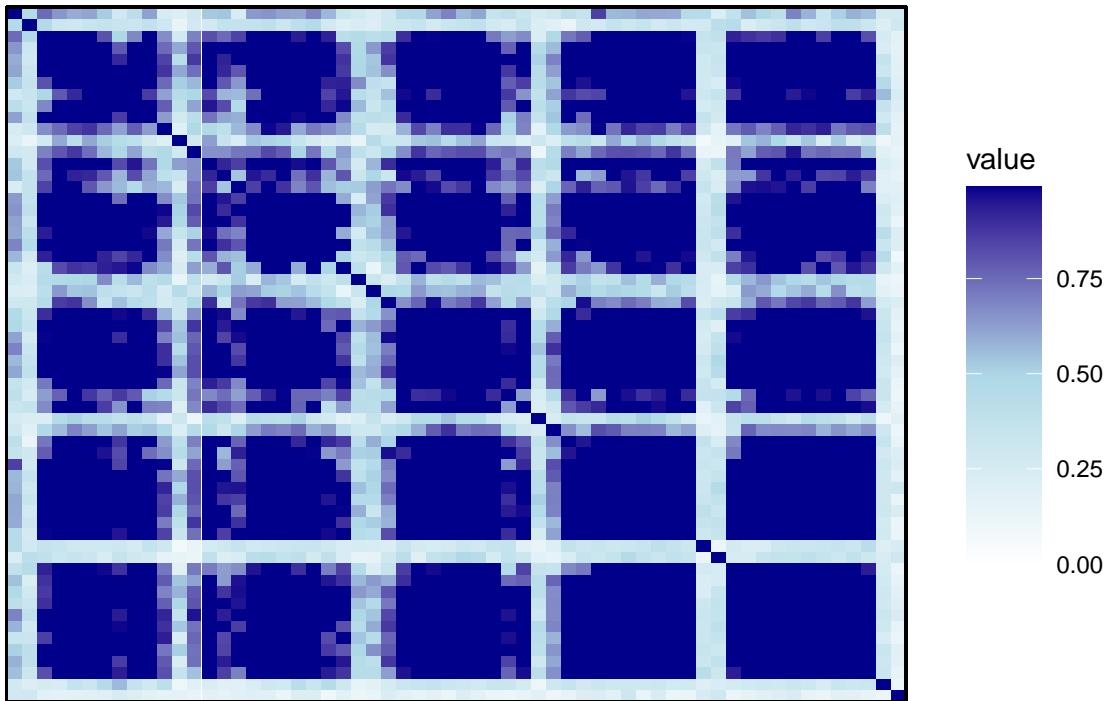
Periodic DL (3,6,12) Covariance Structure



```
kgr_model5_outfit$K_Interaction_heatmap
```

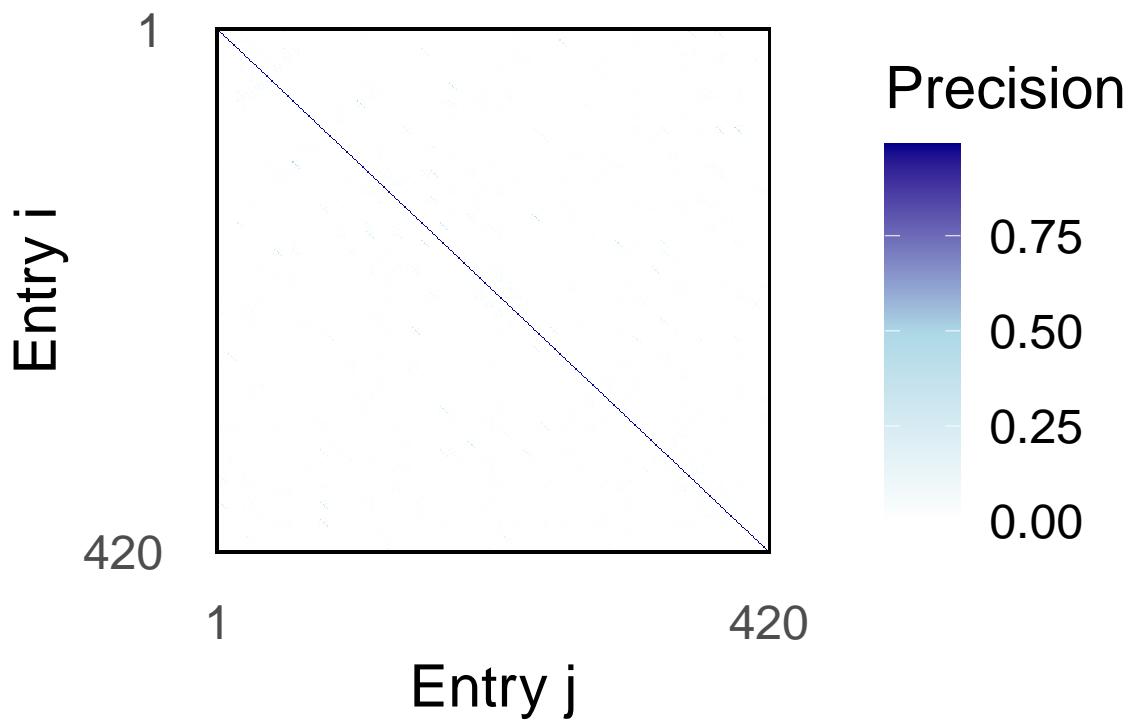
```
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```

Periodic Interaction Covariance Structure

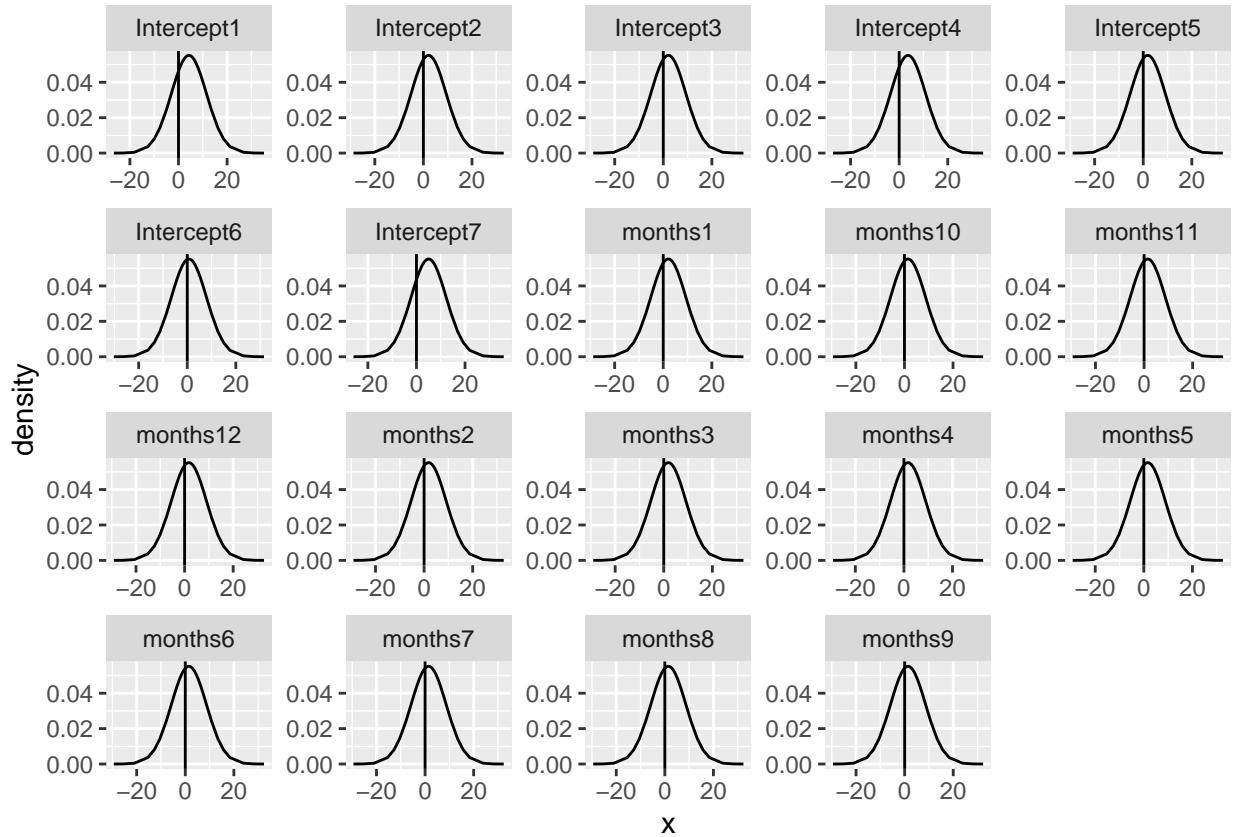


```
kgr_model5_outfit$prec_heatmap
```

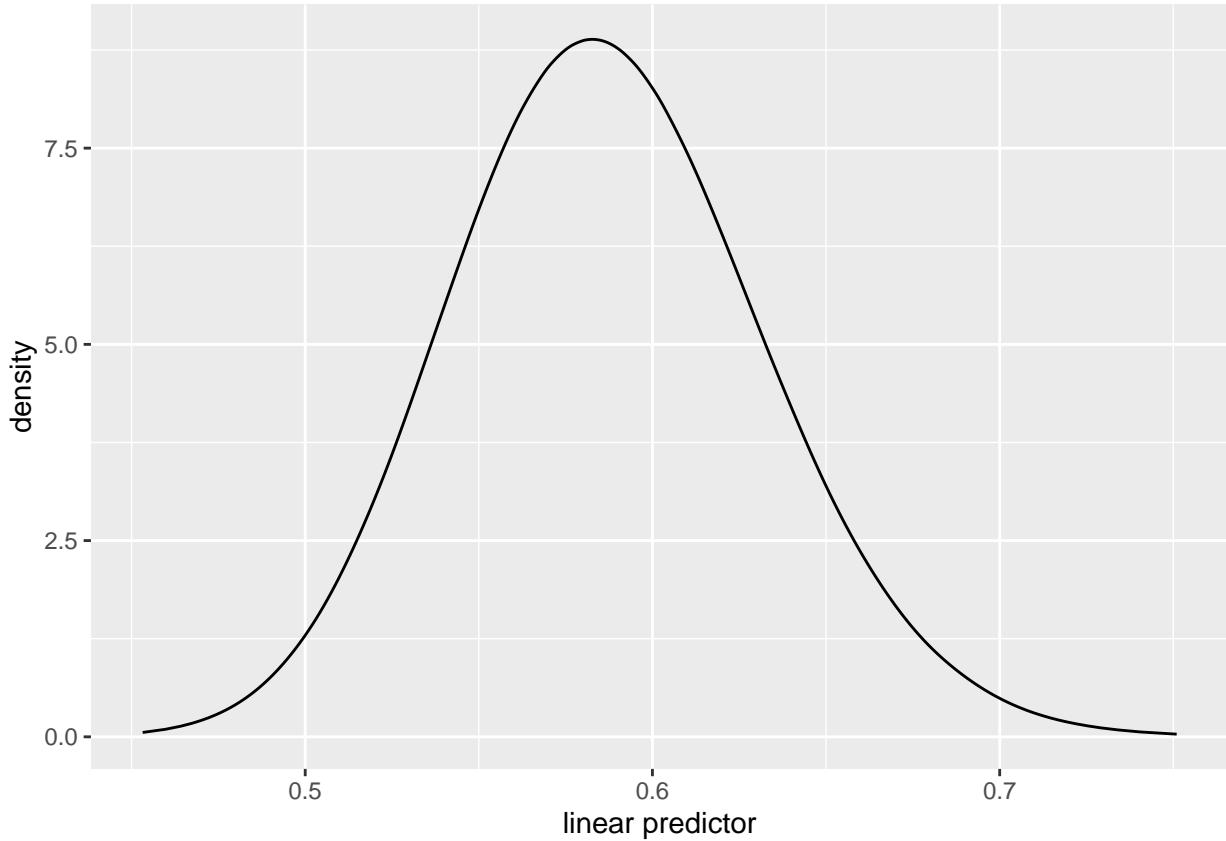
```
## Warning in geom_rect(aes(xmin = x_min, xmax = x_max, ymin = y_min, ymax = y_max), : All aesthetics have
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



```
kgr_model5_outfit$param_plot
```



```
kgr_model5_outfit$hyperparam_plot
```



```
#pp_outsample_plot(pred_data = kgr_model5_outfit$fitted_values,prefix = "kgr5-outsample-")
pp_outsample_plot_combined(pred_data = kgr_model5_outfit$fitted_values,prefix = "kgr5-outsample-nofe-")
```

Comparing out of sample MAE, MASE, and MAPE for different clusters between models

```
print(degree_connectivity)
```

```
##   c(1:num_clus) node_connections
## 1             1             5
## 2             2             5
## 3             3             6
## 4             4             6
## 5             5             6
## 6             6             4
## 7             7             4
```

To compare out of sample fit, we calculate a variety of metrics. The first three, MAE, MASE, and MAPE specifically evaluate forecast accuracy i.e. the prediction accuracy for the time points $t=55, \dots, 60$. The last one is RMSE which is calculated based on the entire sample to get an idea of wholistic fit. Note that once again, we scale the RMSEs so that they can be interpreted relative to the average number of respiratory related deaths in that cluster. Lastly, we also calculate a Frequentist coverage rate based on the credible intervals produced by INLA to get an idea of our models' uncertainty quantification.

Mean absolute error (MAE) is a measure of the average size of the mistakes in a collection of predictions, without taking their direction into account

$MAE = \frac{1}{h_{max}} \sum_{h=1}^{h_{max}} |\hat{\lambda}_{t+h}^{obs} - \hat{\lambda}_{t+h}|$ where $\hat{\lambda}_{t+h}^{obs}$ is the average of the number of deaths observed for month $t+h$ over all years

```

MAE_table = matrix(nrow=7, ncol=num_clus)

for (i in 1:num_clus){
  actual = inla_full_data %>% filter(id == i) %>% data.frame()

  pm_1 = ref_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_2 = ref_model3_outfvs %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_3 = kgr_model1_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_4 = kgr_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_5 = kgr_model3_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_6 = kgr_model4_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_7 = kgr_model5_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()

  actual_test = c()

  for (j in 7:12){
    actual_j = actual %>% filter(months == j) %>% select(response)
    est_lambda = mean(actual_j$response)
    actual_test = c(actual_test, as.numeric(est_lambda))
  }

  pm_1_test = pm_1[55:60,]
  pm_2_test = pm_2[55:60,]
  pm_3_test = pm_3[55:60,]
  pm_4_test = pm_4[55:60,]
  pm_5_test = pm_5[55:60,]
  pm_6_test = pm_6[55:60,]
  pm_7_test = pm_7[55:60,]

  actual_test_mean = mean(actual_test)

  mae1 = mean(abs(actual_test - pm_1_test$mean))
  mae2 = mean(abs(actual_test - pm_2_test$mean))
  mae3 = mean(abs(actual_test - pm_3_test$mean))
  mae4 = mean(abs(actual_test - pm_4_test$mean))
  mae5 = mean(abs(actual_test - pm_5_test$mean))
  mae6 = mean(abs(actual_test - pm_6_test$mean))
  mae7 = mean(abs(actual_test - pm_7_test$mean))

  MAE_table[,i] = c(mae1, mae2, mae3, mae4, mae5, mae6, mae7)
  # MAE_table[,i] = MAE_table[,i] / actual_test_mean
}

#Table 2: MAE on test dataset

MAE_table = data.frame(MAE_table)

colnames(MAE_table) = c("Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4",
                       "Cluster 5", "Cluster 6", "Cluster 7")
rownames(MAE_table) = c("BYM model", "LGCP model", "Proposed KGR model 1",
                       "Proposed KGR model 2", "Proposed KGR model 3",

```

```
"Proposed KGR model 4", "Proposed KGR model 5")
```

```
MAE_table
```

```
##          Cluster 1 Cluster 2 Cluster 3 Cluster 4 Cluster 5
## BYM model      6.274058  2.592604  1.409831  5.752874  1.640119
## LGCP model     34.695832  7.089688  2.877373 27.071209  2.993841
## Proposed KGR model 1 8.389914  2.404287  1.179996  4.715383  1.617876
## Proposed KGR model 2 6.250490  2.431381  1.272305  5.024925  1.624942
## Proposed KGR model 3 6.767787  2.416762  1.289851  5.070064  1.619310
## Proposed KGR model 4 6.580509  2.404159  1.244311  5.206200  1.619211
## Proposed KGR model 5 6.147730  2.413198  1.339202  5.088489  1.610315
##          Cluster 6 Cluster 7
## BYM model      1.268562 10.26971
## LGCP model     1.158639 44.45333
## Proposed KGR model 1 1.278389 12.21282
## Proposed KGR model 2 1.270791 10.86421
## Proposed KGR model 3 1.275580 11.00176
## Proposed KGR model 4 1.259610 10.19292
## Proposed KGR model 5 1.244917 10.31371
```

Mean absolute scaled error (MASE) is a measure of the accuracy of forecasts. It is the mean absolute error of the forecast values, divided by the mean absolute error of the in-sample one-step naive forecast.

$$MASE = \frac{\frac{1}{n} \sum_{t=1}^n |\hat{\lambda}_{t+h}^{obs} - \hat{\lambda}_{t+h}^{obs}|}{\frac{1}{n-1} \sum_{t=2}^n |\hat{\lambda}_{t+h}^{obs} - \hat{\lambda}_{t+h-1}^{obs}|}$$

```
MASE_table = matrix(nrow=7, ncol=num_clus)
```

```
for (i in 1:num_clus){
  actual = inla_full_data %>% filter(id == i) %>% data.frame()

  pm_1 = ref_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_2 = ref_model3_outfvs %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_3 = kgr_model1_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_4 = kgr_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_5 = kgr_model3_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_6 = kgr_model4_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_7 = kgr_model5_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()

  actual_test = c()

  for (j in 6:12){
    actual_j = actual %>% filter(months == j) %>% select(response)
    est_lambda = mean(actual_j$response)
    actual_test = c(actual_test, as.numeric(est_lambda))
  }

  pm_1_test = pm_1$mean[54:60]
  pm_2_test = pm_2$mean[54:60]
  pm_3_test = pm_3$mean[54:60]
  pm_4_test = pm_4$mean[54:60]
  pm_5_test = pm_5$mean[54:60]
  pm_6_test = pm_6$mean[54:60]
  pm_7_test = pm_7$mean[54:60]
```

```

actual_test_mean = mean(actual_test)

values1 = c()
values2 = c()
values3 = c()
values4 = c()
values5 = c()
values6 = c()
values7 = c()

for (j in 2:length(actual_test)){
  error1 = (abs(actual_test[j] - pm_1_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 1)
  values1 = c(values1,error1)

  error2 = (abs(actual_test[j] - pm_2_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 2)
  values2 = c(values2,error2)

  error3 = (abs(actual_test[j] - pm_3_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 3)
  values3 = c(values3,error3)

  error4 = (abs(actual_test[j] - pm_4_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 4)
  values4 = c(values4,error4)

  error5 = (abs(actual_test[j] - pm_5_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 5)
  values5 = c(values5,error5)

  error6 = (abs(actual_test[j] - pm_6_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 6)
  values6 = c(values6,error6)

  error7 = (abs(actual_test[j] - pm_7_test[j])) / (abs(actual_test[j] - actual_test[j-1])) / (length(actual_test) - 7)
  values7 = c(values7,error7)
}

mase1 = mean(values1)
mase2 = mean(values2)
mase3 = mean(values3)
mase4 = mean(values4)
mase5 = mean(values5)
mase6 = mean(values6)
mase7 = mean(values7)

MASE_table[,i] = c(mase1,mase2,mase3,mase4,mase5,mase6,mase7)
# MASE_table[,i] = MASE_table[,i] / actual_test_mean
}

#Table 3: MASE on test dataset

MASE_table = data.frame(MASE_table)

colnames(MASE_table) = c("Cluster 1","Cluster 2","Cluster 3","Cluster 4",
                        "Cluster 5","Cluster 6","Cluster 7")
rownames(MASE_table) = c("BYM model","LGCP model","Proposed KGR model 1",
                        "Proposed KGR model 2","Proposed KGR model 3",

```

```
"Proposed KGR model 4", "Proposed KGR model 5")
```

```
MASE_table
```

```
##          Cluster 1 Cluster 2 Cluster 3 Cluster 4 Cluster 5
## BYM model      0.1150280 0.1108288 0.10068914 0.3876802 0.4052299
## LGCP model     0.6026151 0.3049695 0.22538553 1.6519588 0.8628729
## Proposed KGR model 1 0.1496440 0.1016874 0.07567425 0.2779533 0.3632288
## Proposed KGR model 2 0.1170152 0.1036205 0.08972494 0.3450353 0.3834677
## Proposed KGR model 3 0.1211275 0.1028704 0.09289741 0.3378773 0.3785432
## Proposed KGR model 4 0.1156773 0.1023401 0.08728742 0.3491306 0.3901577
## Proposed KGR model 5 0.1125927 0.1030027 0.10171755 0.3611477 0.3852852
##          Cluster 6 Cluster 7
## BYM model      0.1186376 0.05246646
## LGCP model     0.1132963 0.33932512
## Proposed KGR model 1 0.1205859 0.09702111
## Proposed KGR model 2 0.1194229 0.07133461
## Proposed KGR model 3 0.1196398 0.07412209
## Proposed KGR model 4 0.1177340 0.06185849
## Proposed KGR model 5 0.1166126 0.06461876
```

Mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method in statistics, expressing accuracy as a ratio

$$\text{MAPE} = \frac{100}{n} \sum_{t=1}^n \left| \frac{\hat{\lambda}_{t+h}^{obs} - \hat{\lambda}_{t+h}}{\hat{\lambda}_{t+h}^{obs}} \right|$$

```
MAPE_table = matrix(nrow=7, ncol=num_clus)
```

```
for (i in 1:num_clus){
  actual = inla_full_data %>% filter(id == i) %>% data.frame()

  pm_1 = ref_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_2 = ref_model3_outvfs %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_3 = kgr_model1_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_4 = kgr_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_5 = kgr_model3_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_6 = kgr_model4_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_7 = kgr_model5_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()

  actual_test = c()

  for (j in 7:12){
    actual_j = actual %>% filter(months == j) %>% select(response)
    est_lambda = mean(actual_j$response)
    actual_test = c(actual_test, as.numeric(est_lambda))
  }

  pm_1_test = pm_1[55:60,]
  pm_2_test = pm_2[55:60,]
  pm_3_test = pm_3[55:60,]
  pm_4_test = pm_4[55:60,]
  pm_5_test = pm_5[55:60,]
  pm_6_test = pm_6[55:60,]
  pm_7_test = pm_7[55:60,]
```

```

actual_test_mean = mean(actual_test)

mape1 = mean(abs((actual_test - pm_1_test$mean)/actual_test))
mape2 = mean(abs((actual_test - pm_2_test$mean)/actual_test))
mape3 = mean(abs((actual_test - pm_3_test$mean)/actual_test))
mape4 = mean(abs((actual_test - pm_4_test$mean)/actual_test))
mape5 = mean(abs((actual_test - pm_5_test$mean)/actual_test))
mape6 = mean(abs((actual_test - pm_6_test$mean)/actual_test))
mape7 = mean(abs((actual_test - pm_7_test$mean)/actual_test))

MAPE_table[,i] = c(mape1, mape2, mape3, mape4, mape5, mape6, mape7)
# MAPE_table[,i] = MAPE_table[,i] / actual_test_mean
}

#Table 2: RMSE on test dataset

MAPE_table = data.frame(MAPE_table)

colnames(MAPE_table) = c("Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4",
                        "Cluster 5", "Cluster 6", "Cluster 7")
rownames(MAPE_table) = c("BYM model", "LGCP model", "Proposed KGR model 1",
                        "Proposed KGR model 2", "Proposed KGR model 3",
                        "Proposed KGR model 4", "Proposed KGR model 5")

MAPE_table

##                               Cluster 1   Cluster 2   Cluster 3   Cluster 4   Cluster 5
## BYM model           0.01898377 0.06821119 0.03827458 0.03650015 0.06162319
## LGCP model          0.10540792 0.20205498 0.08040964 0.16560380 0.11321216
## Proposed KGR model 1 0.02413559 0.06279817 0.03173702 0.02855467 0.06062466
## Proposed KGR model 2 0.01844258 0.06371923 0.03442227 0.03145985 0.06105471
## Proposed KGR model 3 0.01974231 0.06331897 0.03494186 0.03140223 0.06084929
## Proposed KGR model 4 0.01934617 0.06309082 0.03372914 0.03216143 0.06109242
## Proposed KGR model 5 0.01813836 0.06338134 0.03657593 0.03179537 0.06072548
##                               Cluster 6   Cluster 7
## BYM model           0.1315404 0.01424397
## LGCP model          0.1390807 0.06520897
## Proposed KGR model 1 0.1329921 0.01774027
## Proposed KGR model 2 0.1322369 0.01539471
## Proposed KGR model 3 0.1327687 0.01564039
## Proposed KGR model 4 0.1311103 0.01447426
## Proposed KGR model 5 0.1294794 0.01470626

#Overall fit
test_RMSE_table = matrix(nrow=8, ncol=num_clus)

for (i in 1:num_clus){
  actual = inla_full_data %>% filter(id == i) %>% data.frame()

  pm_1 = ref_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_2 = ref_model3_outfvs %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_3 = kgr_model1_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_4 = kgr_model2_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
  pm_5 = kgr_model3_outfit$fitted_values %>% filter(id == i) %>% select(mean, sd) %>% data.frame()
}

```

```

pm_6 = kgr_model4_outfit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_7 = kgr_model5_outfit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()
pm_8 = kgr_model2_nb_outfit$fitted_values %>% filter(id == i) %>% select(mean,sd) %>% data.frame()

actual_test = c()

for (j in 7:12){
  actual_j = actual %>% filter(months == j) %>% select(response)
  est_lambda = mean(actual_j$response)
  actual_test = c(actual_test,as.numeric(est_lambda))
}

pm_1_test = pm_1[55:60,]
pm_2_test = pm_2[55:60,]
pm_3_test = pm_3[55:60,]
pm_4_test = pm_4[55:60,]
pm_5_test = pm_5[55:60,]
pm_6_test = pm_6[55:60,]
pm_7_test = pm_7[55:60,]
pm_8_test = pm_7[55:60,]

actual_test_mean = mean(actual_test)

test_rmse1 = sqrt(mean((actual_test - pm_1_test$mean)^2))
test_rmse2 = sqrt(mean((actual_test - pm_2_test$mean)^2))
test_rmse3 = sqrt(mean((actual_test - pm_3_test$mean)^2))
test_rmse4 = sqrt(mean((actual_test - pm_4_test$mean)^2))
test_rmse5 = sqrt(mean((actual_test - pm_5_test$mean)^2))
test_rmse6 = sqrt(mean((actual_test - pm_6_test$mean)^2))
test_rmse7 = sqrt(mean((actual_test - pm_7_test$mean)^2))
test_rmse8 = sqrt(mean((actual_test - pm_8_test$mean)^2))

test_RMSE_table[,i] = c(test_rmse1,test_rmse2,test_rmse3,test_rmse4,
                        test_rmse5,test_rmse6,test_rmse7,test_rmse8)
test_RMSE_table[,i] = test_RMSE_table[,i] / actual_test_mean
}

#Table 2: RMSE on test dataset

test_RMSE_table = data.frame(test_RMSE_table)

colnames(test_RMSE_table) = c("Cluster 1","Cluster 2","Cluster 3","Cluster 4",
                             "Cluster 5","Cluster 6","Cluster 7")
rownames(test_RMSE_table) = c("BYM model","LGCP model","Proposed KGR model 1",
                             "Proposed KGR model 2","Proposed KGR model 3",
                             "Proposed KGR model 4","Proposed KGR model 5","NB model")

test_RMSE_table

##                               Cluster 1   Cluster 2   Cluster 3   Cluster 4   Cluster 5
## BYM model           0.02141523 0.07719888 0.04377833 0.04257111 0.06721311
## LGCP model          0.11238140 0.21280864 0.09242386 0.16449726 0.11861162
## Proposed KGR model 1 0.02917817 0.06944887 0.04208864 0.03134029 0.06405213
## Proposed KGR model 2 0.02132844 0.07268997 0.04091446 0.03579592 0.06632393

```

```

## Proposed KGR model 3 0.02280352 0.07173541 0.04037654 0.03542099 0.06573521
## Proposed KGR model 4 0.02261739 0.07171135 0.04046836 0.03608462 0.06672927
## Proposed KGR model 5 0.02120384 0.07262390 0.04194394 0.03602800 0.06650422
## NB model          0.02120384 0.07262390 0.04194394 0.03602800 0.06650422
##                           Cluster 6   Cluster 7
## BYM model           0.1524910 0.01757785
## LGCP model          0.1687961 0.06723023
## Proposed KGR model 1 0.1520058 0.01922794
## Proposed KGR model 2 0.1511963 0.01803904
## Proposed KGR model 3 0.1515460 0.01809742
## Proposed KGR model 4 0.1502002 0.01687912
## Proposed KGR model 5 0.1490934 0.01658478
## NB model           0.1490934 0.01658478

```

Comparing coverage rates of credible intervals produced by each model

```

coverage = rep(0,8)
true_values = inla_full_data$response
models_fvs = list(ref_model2_outfit$fitted_values,ref_model3_outfvs,kgr_model1_outfit$fitted_values,
                   kgr_model2_outfit$fitted_values,kgr_model3_outfit$fitted_values,kgr_model4_outfit$fitted_values,
                   kgr_model5_outfit$fitted_values,kgr_model2_nb_outfit$fitted_values)

for (i in 1:8){
  lci = models_fvs[[i]] %>% select(y_pplower)
  uci = models_fvs[[i]] %>% select(y_ppupper)

  if(i == 2){
    inla_full_data2 = inla_full_data %>% arrange(id)
    true_values2 = inla_full_data2$response

    captured = (true_values2 >= lci$y_pplower & true_values2 <= uci$y_ppupper)
    coverage[2] = sum(captured)/length(captured)
  } else{
    captured = (true_values >= lci$y_pplower & true_values <= uci$y_ppupper)
    coverage[i] = sum(captured)/length(captured)
  }
}

coverage = data.frame(coverage)
colnames(coverage) = "95% coverage"
rownames(coverage) = c("BYM model","LGCP model","Proposed KGR model 1",
                      "Proposed KGR model 2","Proposed KGR model 3","Proposed KGR model 4",
                      "Proposed KGR model 5","NB model")

coverage
##                               95% coverage
## BYM model           0.9190476
## LGCP model          0.9761905
## Proposed KGR model 1 0.9904762
## Proposed KGR model 2 0.9904762
## Proposed KGR model 3 0.9928571
## Proposed KGR model 4 0.9928571

```

```
## Proposed KGR model 5      0.9904762
## NB model                 0.9476190
```

Plotting heatmaps of CA with estimated mean and variance of intensity function for each model

```
#Plot heatmap for time = 60
clusterlabels$counties = tolower(clusterlabels$counties)

colnames(clusterlabels) = c("subregion","cluster")
merged_response = join(ca_map,clusterlabels,by = "subregion")

true_values = inla_full_data %>% filter(time == 60) %>% select(id,response)
colnames(true_values) = c("cluster","response")
merged_response = join(merged_response,true_values,by = "cluster")

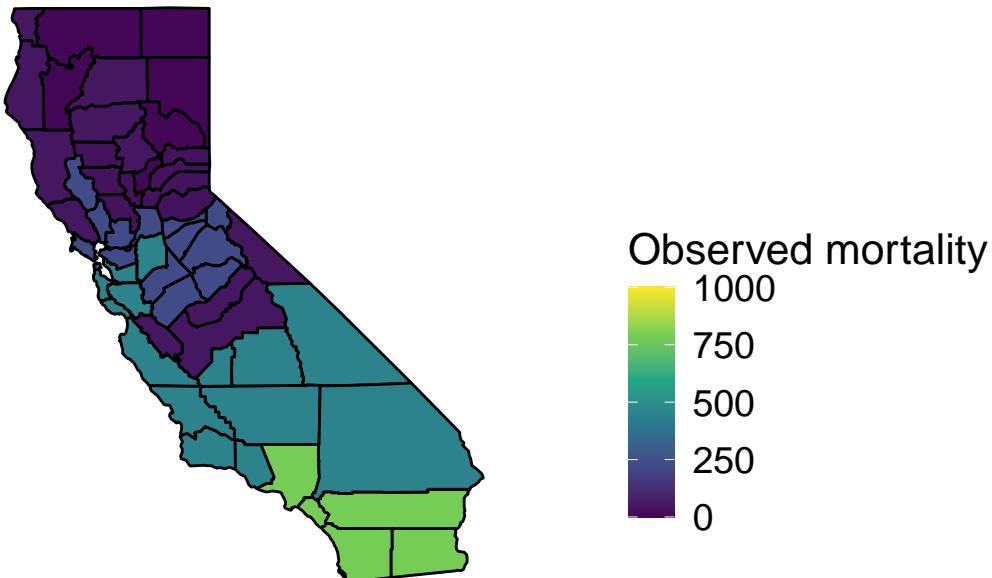
heatmap_limits = c(0,1000)
legend_titles = c("Ref model 2 fitted values","Ref model 3 fitted values","Prop model 1 fitted values",
                  "Prop model 2 fitted values","Prop model 3 fitted values","Prop model 4 fitted values"

#Plot of observed mortality
# gg_pop <- ggplot() +
#   geom_polygon(data = merged_response, aes(x = long, y = lat, group = group, fill = response),
#               color = "black") +
#   coord_fixed(ratio = 1.3, xlim = c(-125, -112), ylim = c(30, 42)) +
#   scale_fill_viridis_c(limits = heatmap_limits, name = "Observed mortality") +
#   theme_void() +
#   labs(title = "Dec 2019",
#        x = "Longitude",
#        y = "Latitude")
#
# print(gg_pop)

gg_pop <- ggplot() +
  geom_polygon(
    data = merged_response,
    aes(x = long, y = lat, group = group, fill = response),
    color = "black"
  ) +
  coord_fixed(ratio = 1.3, xlim = c(-125, -112), ylim = c(30, 42)) +
  scale_fill_viridis_c(limits = heatmap_limits, name = "Observed mortality") +
  labs(
    title = "Dec 2019"
  ) +
  theme_void(base_size = 14) +    # keeps axes/ticks gone, scales base font
  theme(
    plot.title = element_text(size = 18, face = "bold", hjust = 0.5),
    legend.title = element_text(size = 16),
    legend.text = element_text(size = 14)
  )

print(gg_pop)
```

Dec 2019



Plotting heatmaps of mean of intensity function

```
models_intensity_fvs = list(ref_model1_outfit$fitted_values,ref_model2_outfit$fitted_values,ref_model3_outfit$fitted_values,kgr_model1_outfit$fitted_values,kgr_model2_outfit$fitted_values,kgr_model3_outfit$fitted_values,kgr_model4_outfit$fitted_values,kgr_model5_outfit$fitted_values)

merged_response = join(ca_map,clusterlabels,by = "subregion")

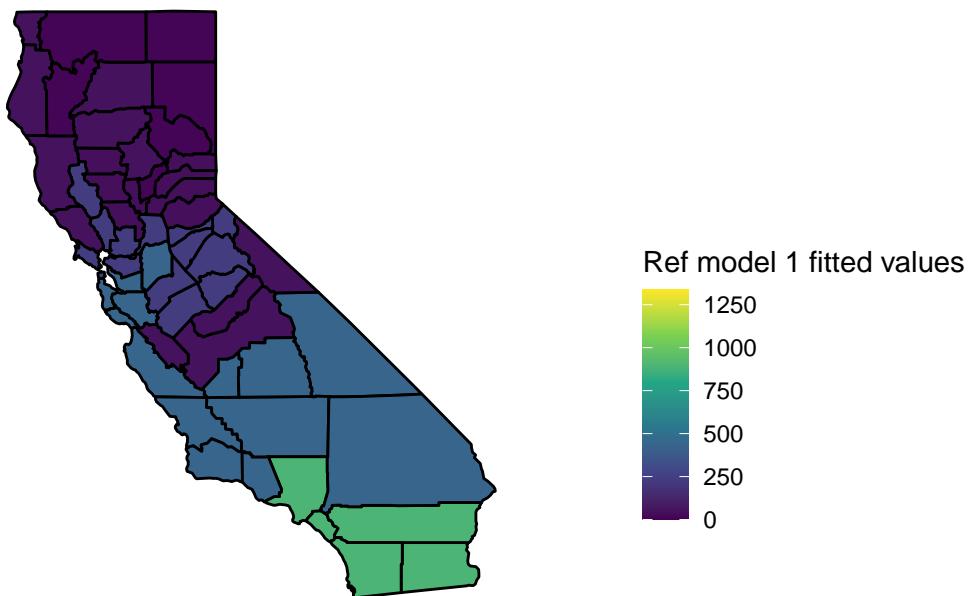
legend_titles = c("Ref model 1 fitted values","Ref model 2 fitted values","Ref model 3 fitted values",
                  "Prop model 1 fitted values","Prop model 2 fitted values","Prop model 3 fitted values",
                  "Prop model 4 fitted values","Prop model 5 fitted values")

for (i in 1:8){
  fitted_values = models_intensity_fvs[[i]] %>% filter(time == 60) %>% select(id,mean)
  heatmap_limits = c(0,1.5*max(fitted_values$mean))
  colname = sprintf("prediction.%s",i)
  colnames(fitted_values) = c("cluster",colname)
  merged_response = join(merged_response,fitted_values,by = "cluster")

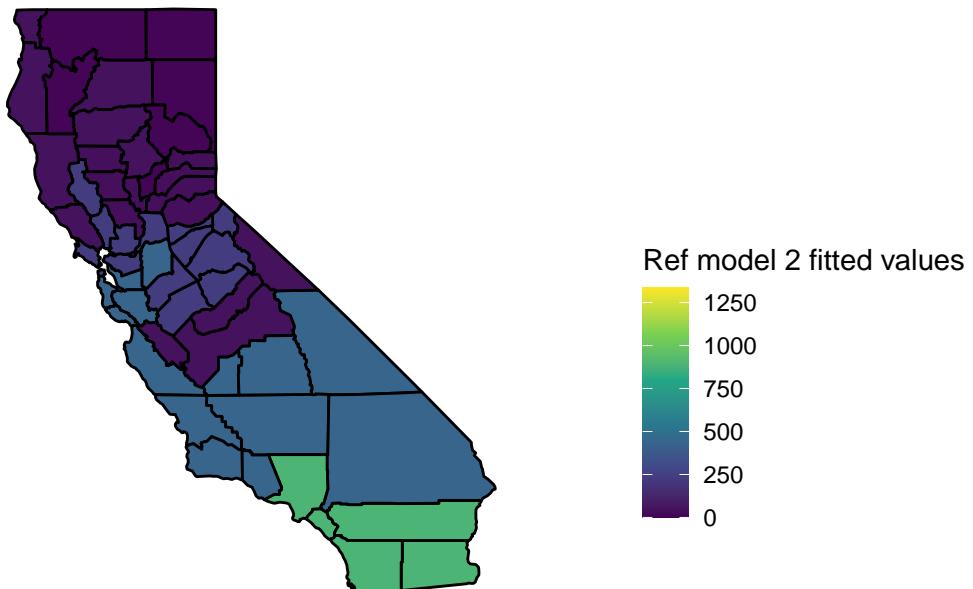
  #Heatmap of each model's fvs
  gg_pop <- ggplot() +
    geom_polygon(data = merged_response, aes(x = long, y = lat, group = group, fill = merged_response[,colname],
                                               color = "black") +
    coord_fixed(ratio = 1.3, xlim = c(-125, -112), ylim = c(30, 42)) +
    scale_fill_viridis_c(limits = heatmap_limits, name = legend_titles[i]) +
```

```
theme_void() +  
  labs(title = "Dec 2019",  
       x = "Longitude",  
       y = "Latitude")  
  
print(gg_pop)  
}
```

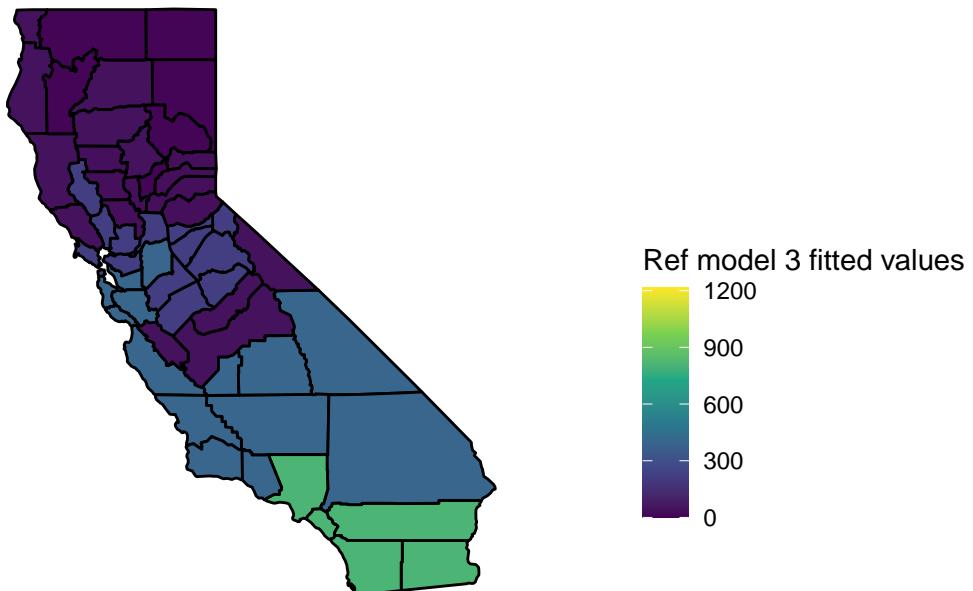
Dec 2019



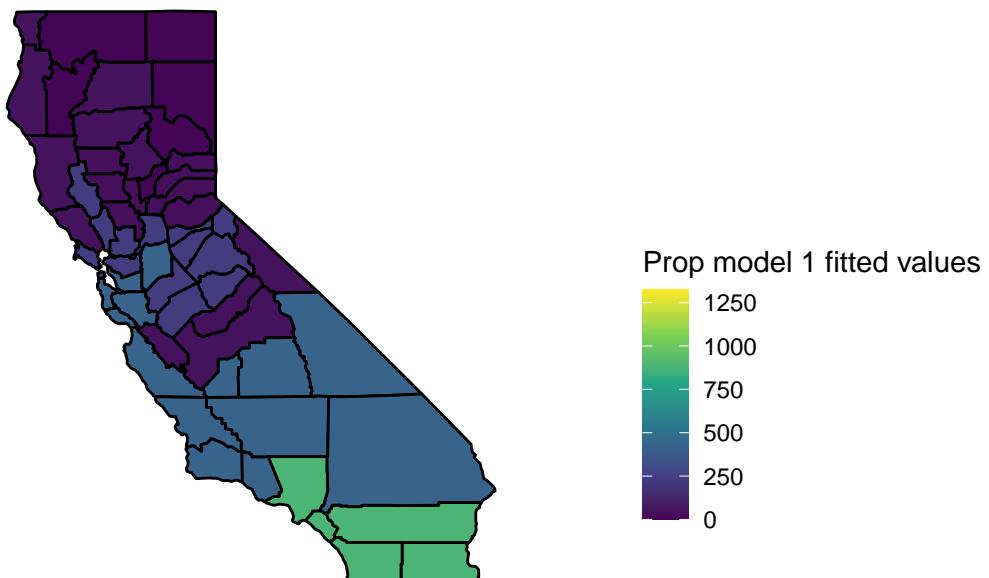
Dec 2019



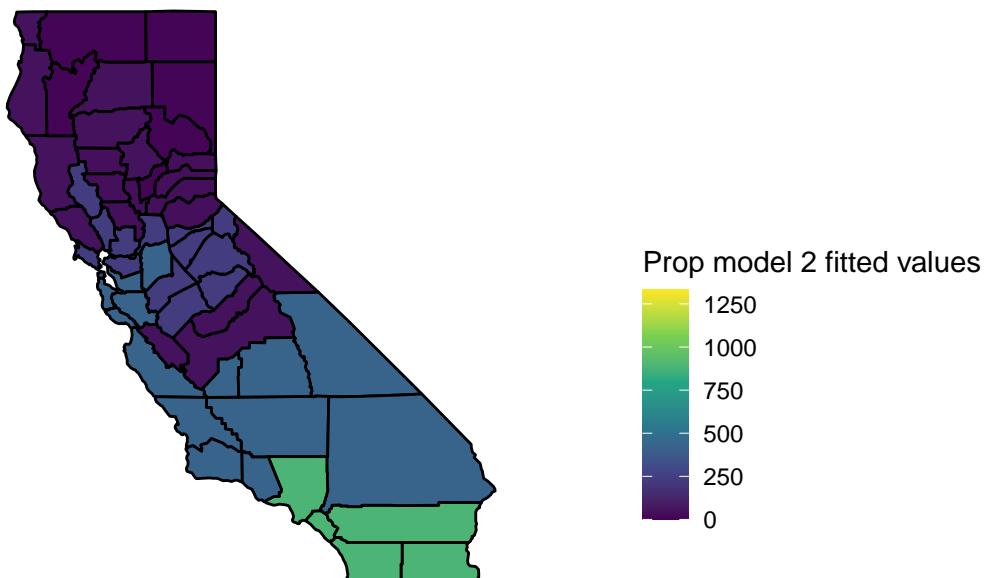
Dec 2019



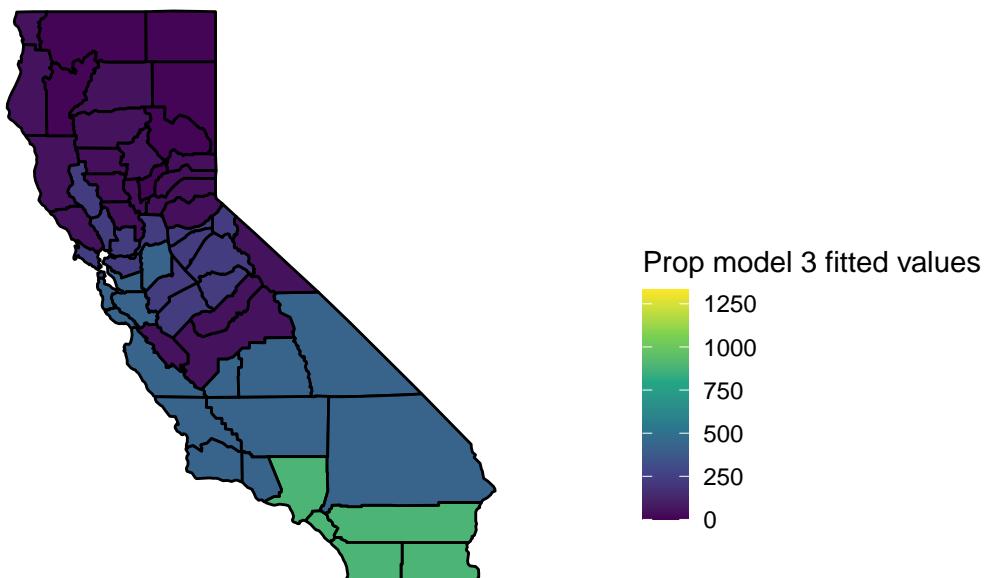
Dec 2019



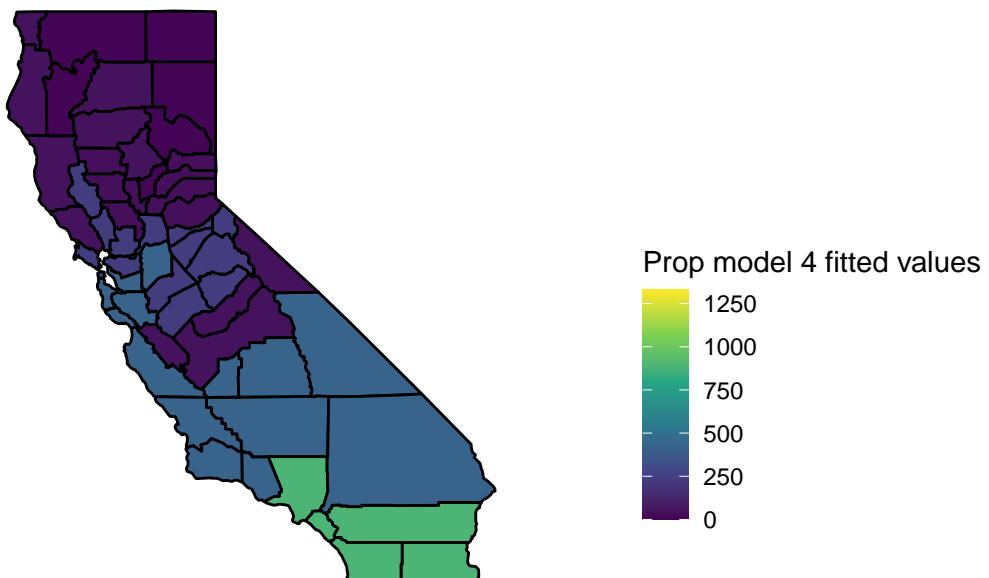
Dec 2019



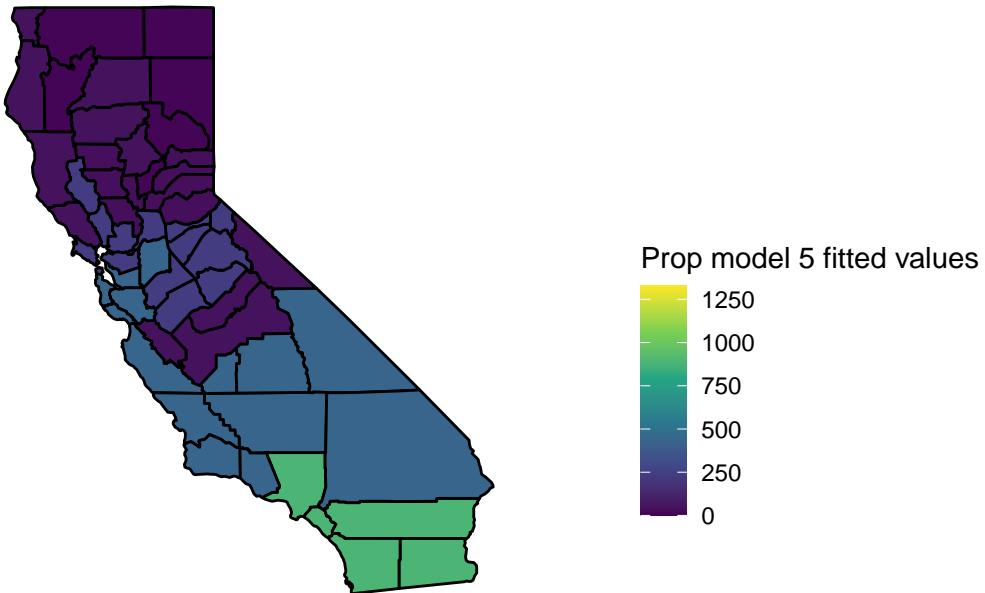
Dec 2019



Dec 2019



Dec 2019



Plotting heatmaps of variance of intensity function

```
merged_response = join(ca_map,clusterlabels,by = "subregion")

legend_titles = c("Ref model 1 variance","Ref model 2 variance","Ref model 3 variance",
                 "Prop model 1 variance","Prop model 2 variance","Prop model 3 variance",
                 "Prop model 4 variance","Prop model 5 variance")

for (j in 1:8){
  fitted_values = models_intensity_fvs[[j]] %>% filter(time == 60) %>% select(id,sd)
  fitted_values$sd = fitted_values$sd^2

  if (j == 3){
    heatmap_limits = c(0,20000)
  } else{
    heatmap_limits = c(0,6000)
  }

  # heatmap_limits = c(0,8000)
  colname = sprintf("prediction.%s",j)
  colnames(fitted_values) = c("cluster",colname)
  merged_response = join(merged_response,fitted_values,by = "cluster")

  #Heatmap of each model's fvs
  gg_pop <- ggplot() +
```

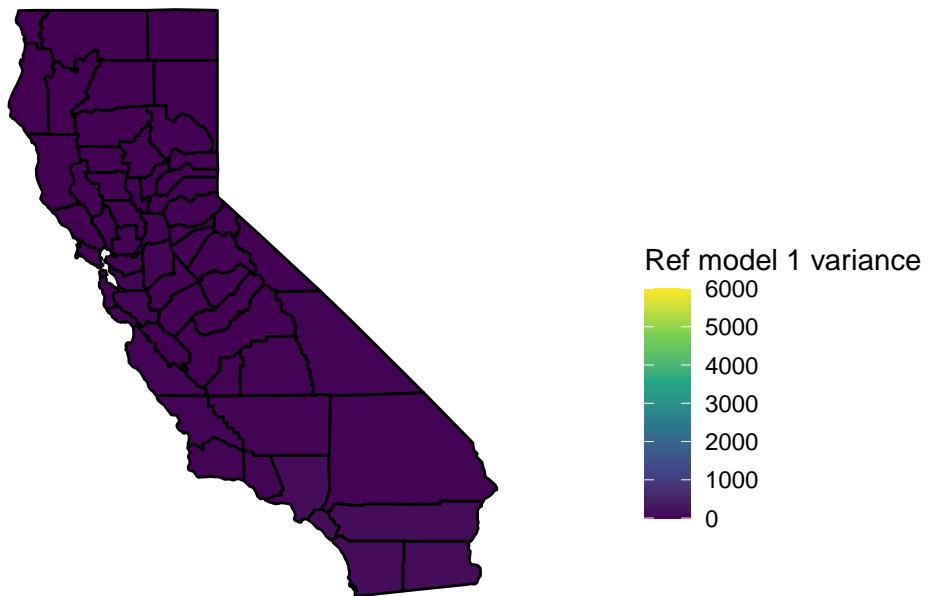
```

geom_polygon(data = merged_response, aes(x = long, y = lat, group = group, fill = merged_response[,],
                                         color = "black") +
  coord_fixed(ratio = 1.3, xlim = c(-125, -112), ylim = c(30, 42)) +
  scale_fill_viridis_c(limits = heatmap_limits, name = legend_titles[j]) +
  theme_void() +
  labs(title = "Dec 2019",
       x = "Longitude",
       y = "Latitude")

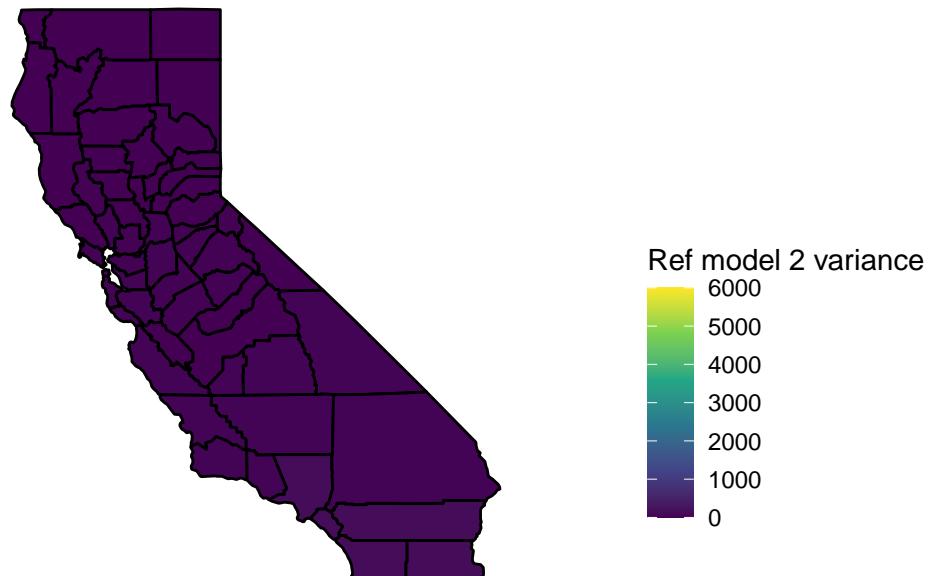
print(gg_pop)
}

```

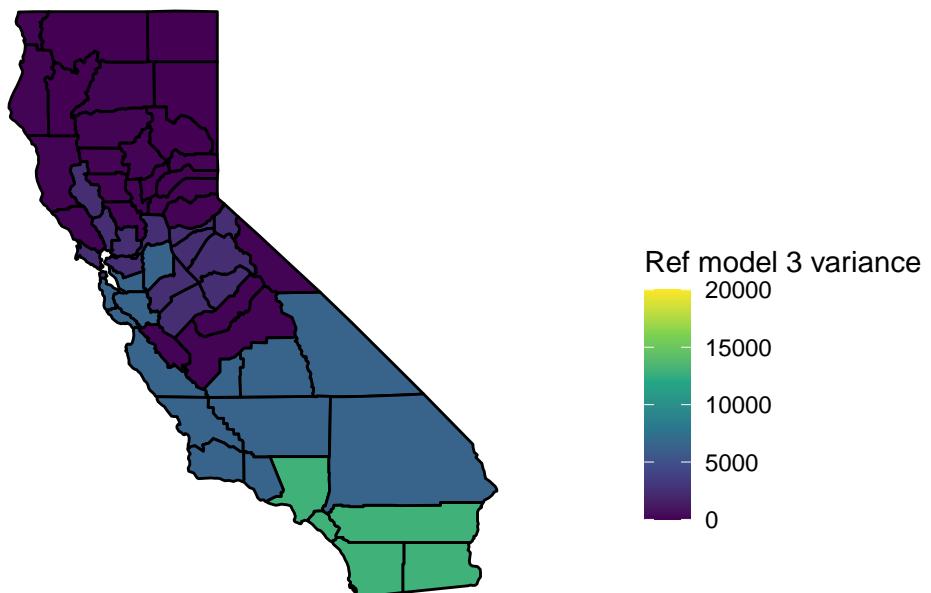
Dec 2019



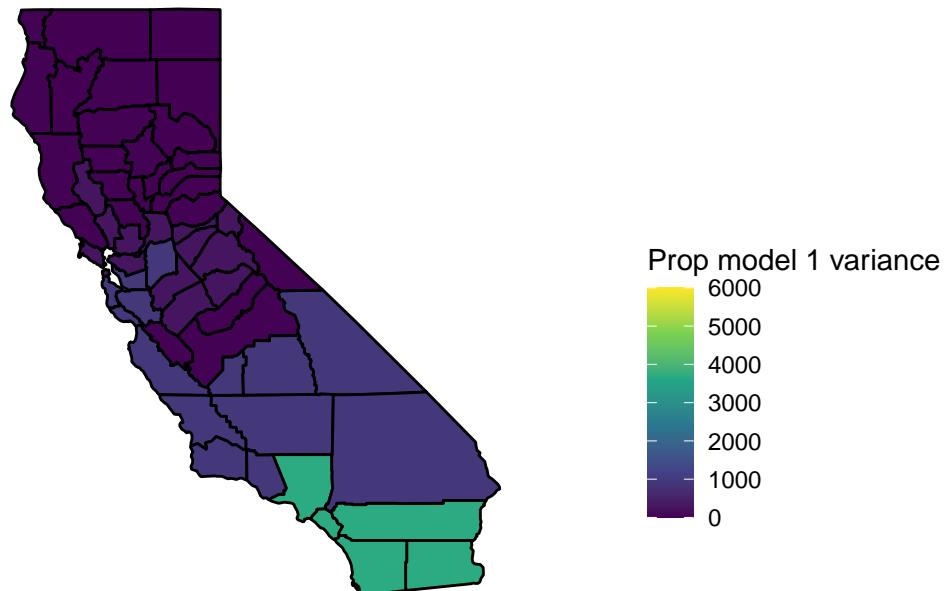
Dec 2019



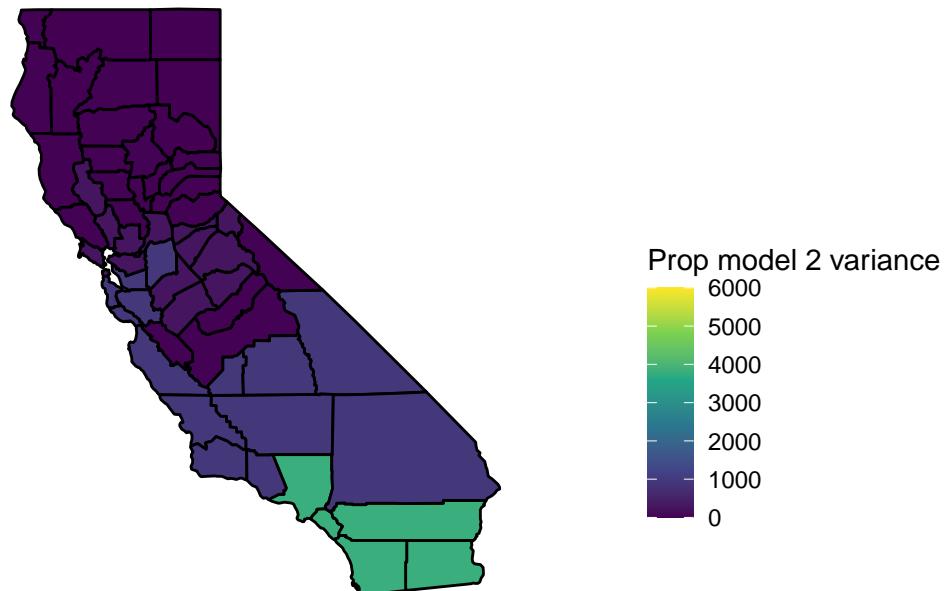
Dec 2019



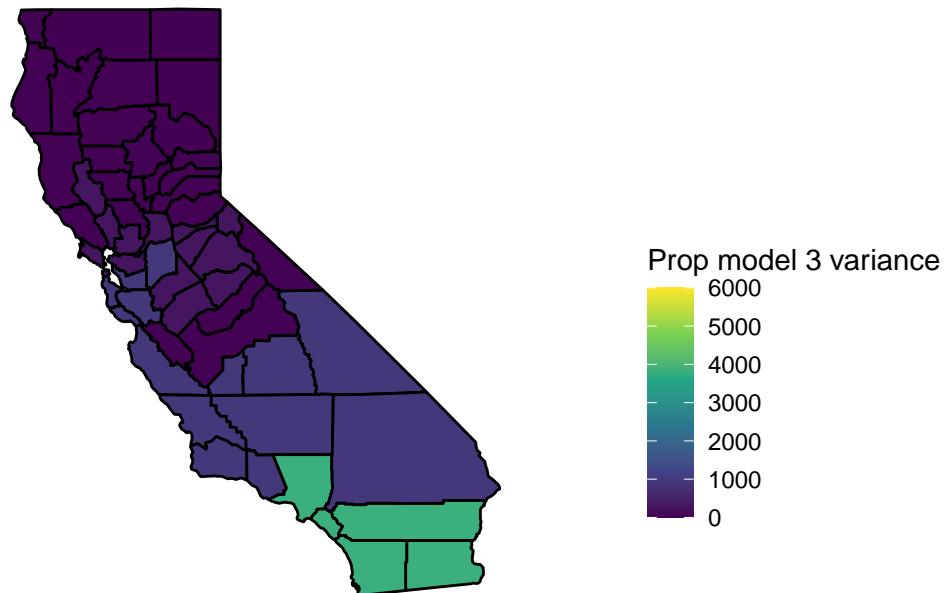
Dec 2019



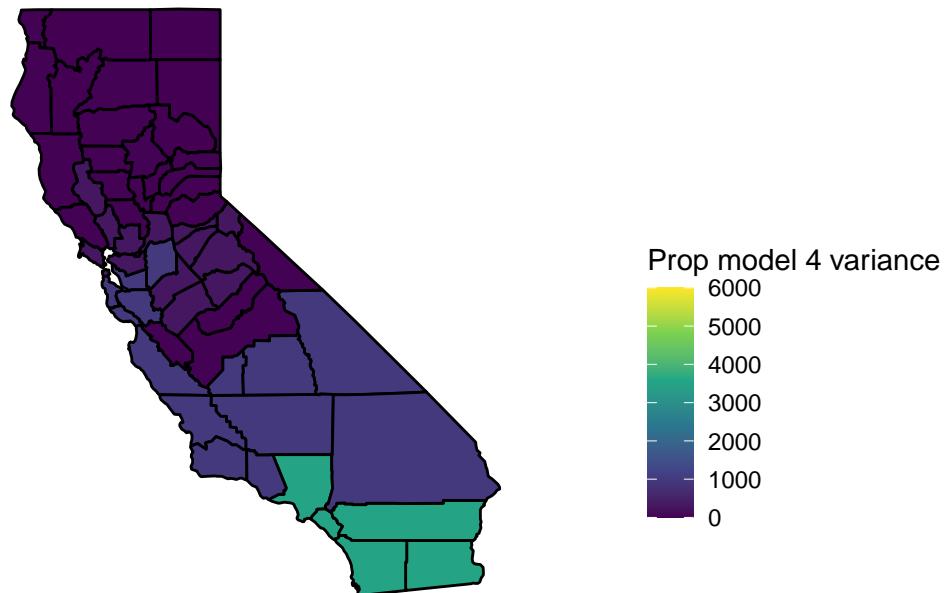
Dec 2019



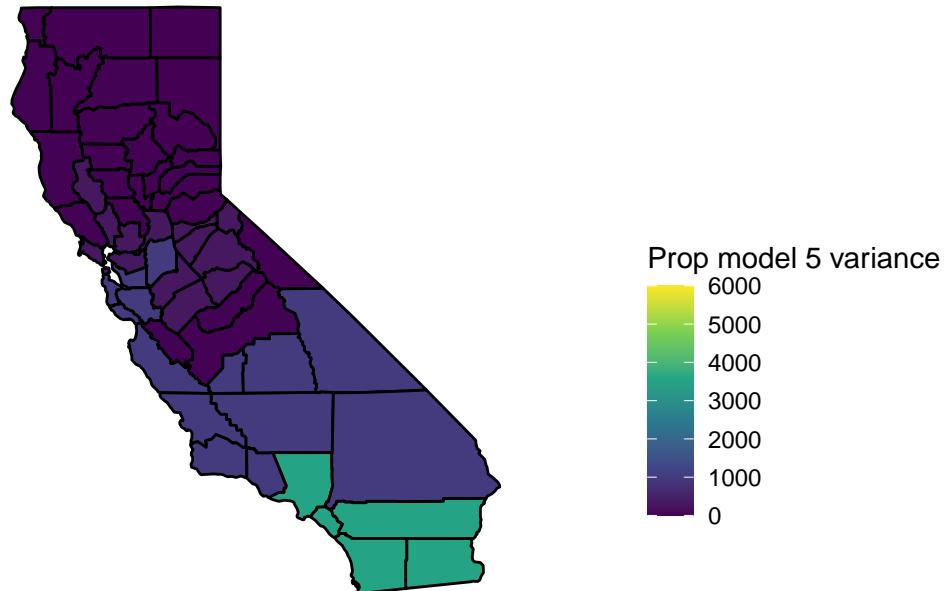
Dec 2019



Dec 2019



Dec 2019



Sliding timeframe forecasting exercise:

One step ahead forecasting

In this section, I implemented a forecasting exercise in which I start with 36 months of training data. I will use that data to estimate a model and then forecast one month ahead. Using our starting data AND the forecasted values, I will re-estimate the model and forecast again to get the next month's predictions. This process continues until the original 36 months of data have been used to produce a complete time series of 60 months (the last 3 years are forecasted). This exercise gives us another way to compare the predictive ability of our various models.

Reference model

```
ref_model_error = cbind(MAE,MASE,MAPE,RMSPE)
ref_model_error

##          MAE        MASE        MAPE        RMSPE
## [1,] 18.228571 0.020729671 0.08875661 25.497563
## [2,] 7.600000 0.018738550 0.04932430 12.802678
## [3,] 3.000000 0.026709075 0.04295158  3.796991
## [4,] 3.400000 0.012414181 0.02372278  6.187083
## [5,] 4.171429 0.066365751 0.03513482  5.475139
## [6,] 2.571429 0.039829518 0.04001582  3.140519
## [7,] 4.971429 0.084333436 0.04883699  6.975058
## [8,] 3.628571 0.049233209 0.06536543  4.872664
## [9,] 5.828571 0.092609032 0.08009851  8.024961
```

```

## [10,]  1.714286 0.042161765 0.03469257  2.140761
## [11,]  4.657143 0.080198043 0.06214016  5.826540
## [12,]  9.371429 0.004038253 0.05799603 17.473245
## [13,] 18.228571 0.013031877 0.08875661 25.497563
## [14,]  7.600000 0.015874065 0.04932430 12.802678
## [15,]  3.000000 0.027994560 0.04295158  3.796991
## [16,]  3.400000 0.012598345 0.02372278  6.187083
## [17,]  4.171429 0.066006542 0.03513482  5.475139
## [18,]  2.571429 0.038961132 0.04001582  3.140519
## [19,]  4.971429 0.083501151 0.04883699  6.975058
## [20,]  3.628571 0.047773676 0.06536543  4.872664
## [21,]  5.828571 0.091153982 0.08009851  8.024961
## [22,]  1.714286 0.041326319 0.03469257  2.140761
## [23,]  4.800000 0.079698886 0.06680869  5.853448
## [24,]  9.371429 0.003106388 0.05799603 17.473245

#Plot of posterior predictive estimates (months 37-60) with credible interval bands OVERLAID on responses
true_mortality = inla_full_data

for (i in 1:num_clus){
  df = true_mortality %>% filter(id == i) %>% select(response)
  preds = starting_data %>% filter(id == i)
  colnames(preds)[3] = "mean"
  df = cbind(df,preds)

  title = sprintf("Cluster %s",i)

  post_pred_plot = df %>% ggplot(aes(x=time,y=response)) + geom_point() +
    geom_line(data = df %>% filter(time > 36), aes(y=mean),color = "red") +
    geom_ribbon(aes(ymin = lower,ymax = upper),alpha = 0.3) +
    geom_vline(xintercept = 36,linetype = "dashed",color = "blue",linewidth = 1.5) + ggtitle(title) +
    print(post_pred_plot)
}

true_mortality = inla_full_data
plot_list = list()

for (i in 1:num_clus){
  df = true_mortality %>% filter(id == i) %>% select(response)
  preds = starting_data %>% filter(id == i)
  colnames(preds)[3] = "mean"
  df = cbind(df,preds)
  df$date = as.Date(paste0("2015-01-01")) + months(df$time - 1)

  test_idx = which(df$time >= 37)

  captured = (df$response[test_idx] >= df$lower[test_idx] & df$response[test_idx] <= df$upper[test_idx])
  coverage = round(sum(captured)/length(captured)*100,2)

  title = sprintf("Cluster %s (Coverage: %s%%)",i,coverage)

  # Define tick marks
  jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
  jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
  all_breaks <- sort(c(jan_breaks, jul_breaks))
}

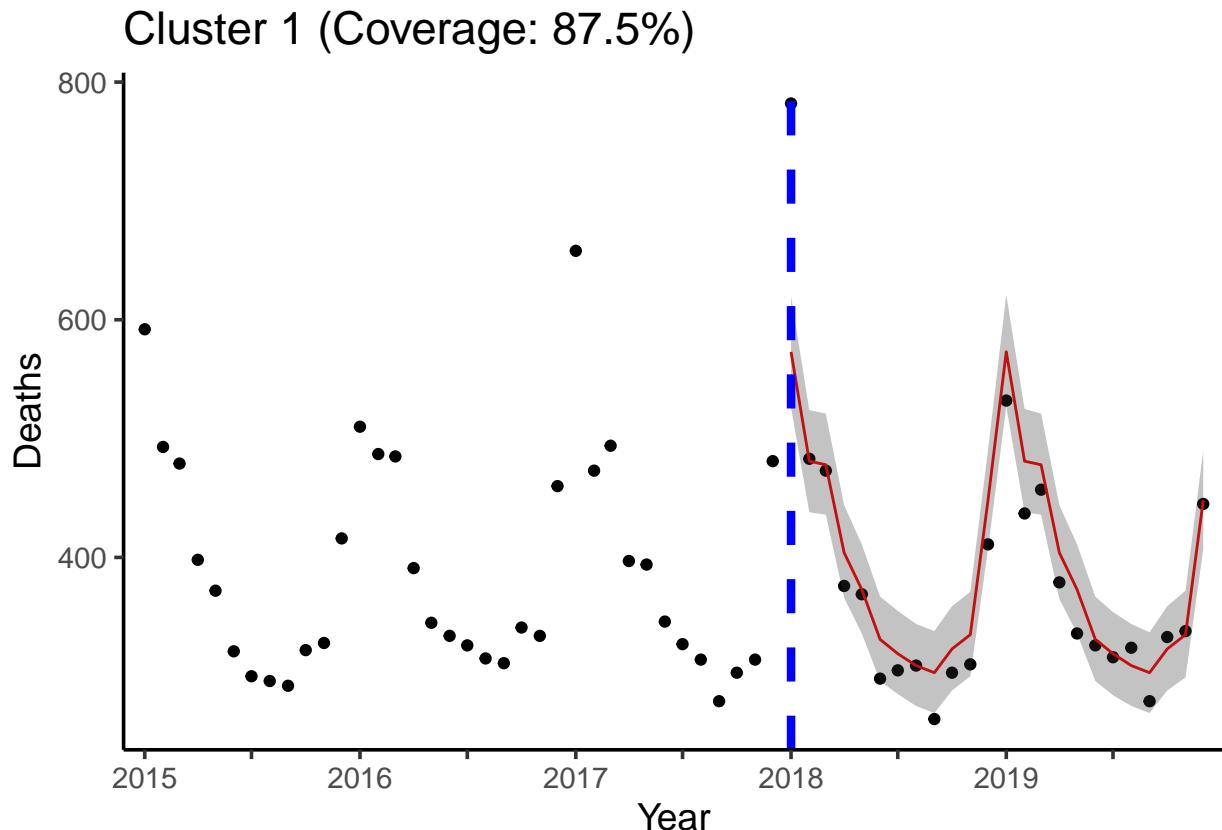
```

```

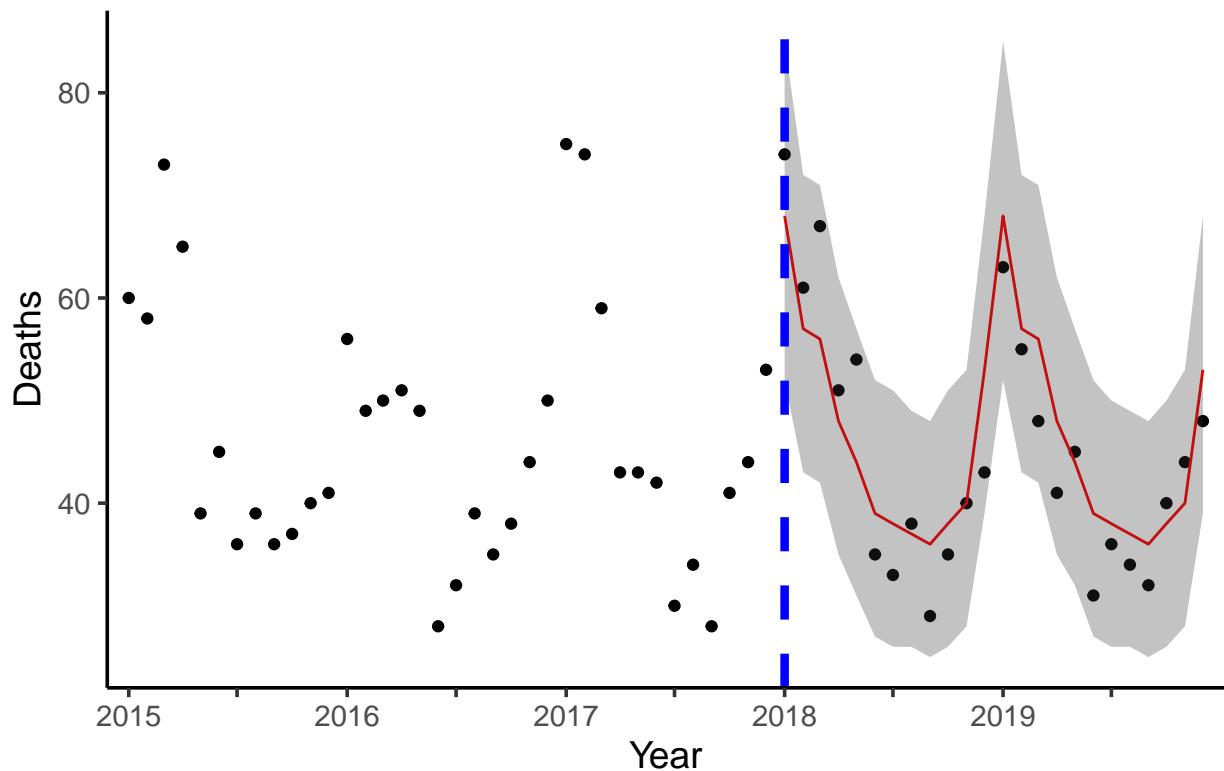
post_pred_plot = df %>%
  ggplot(aes(x = date, y = response)) +
  geom_point() +
  geom_line(data = df %>% filter(time > 36), aes(y = mean), color = "red") +
  geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.3) +
  geom_vline(xintercept = as.Date("2018-01-01"), linetype = "dashed", color = "blue", linewidth = 1.5) +
  ggtitle(title) +
  scale_x_date(
    name = "Year",
    breaks = all_breaks,
    labels = ifelse(format(all_breaks, "%m") == "01",
                   format(all_breaks, "%Y"), ""),
    expand = expansion(mult = 0.02) # ~2% space on each side
  ) +
  scale_y_continuous(name = "Deaths") +
  theme_classic(base_size = 14)

print(post_pred_plot)
plot_list[[i]] = post_pred_plot
}

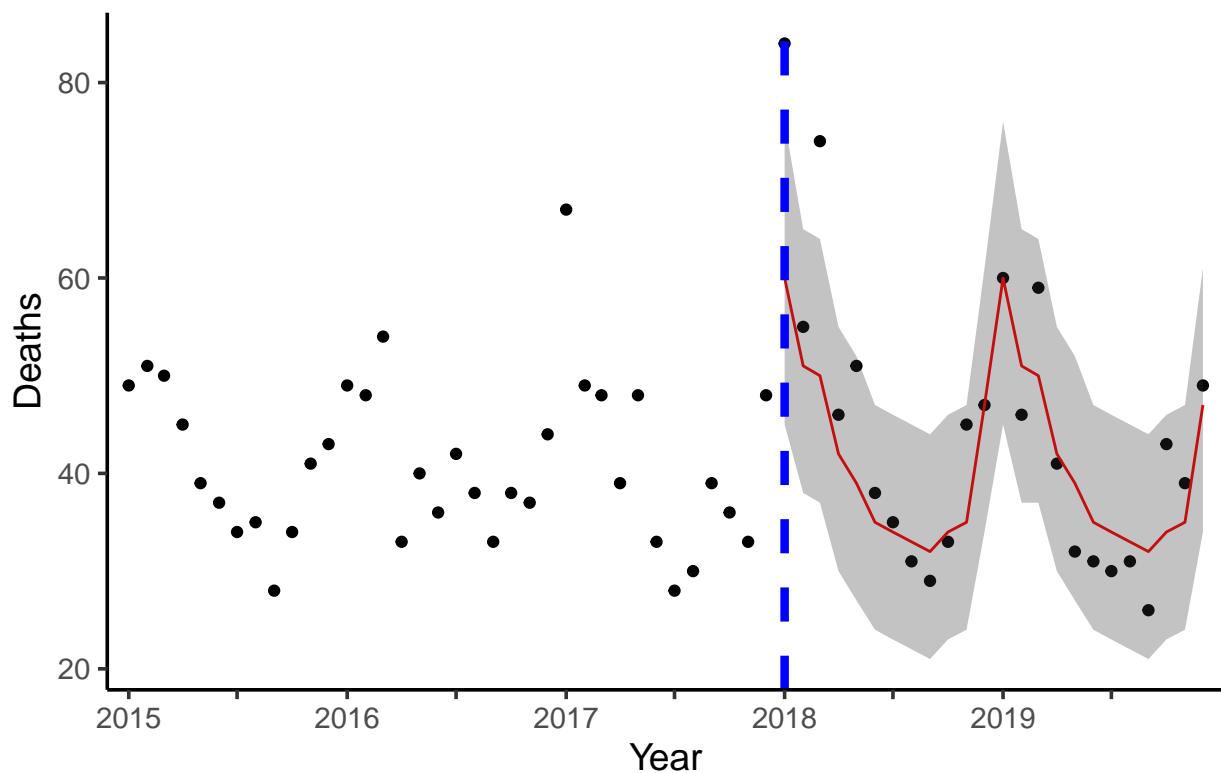
```



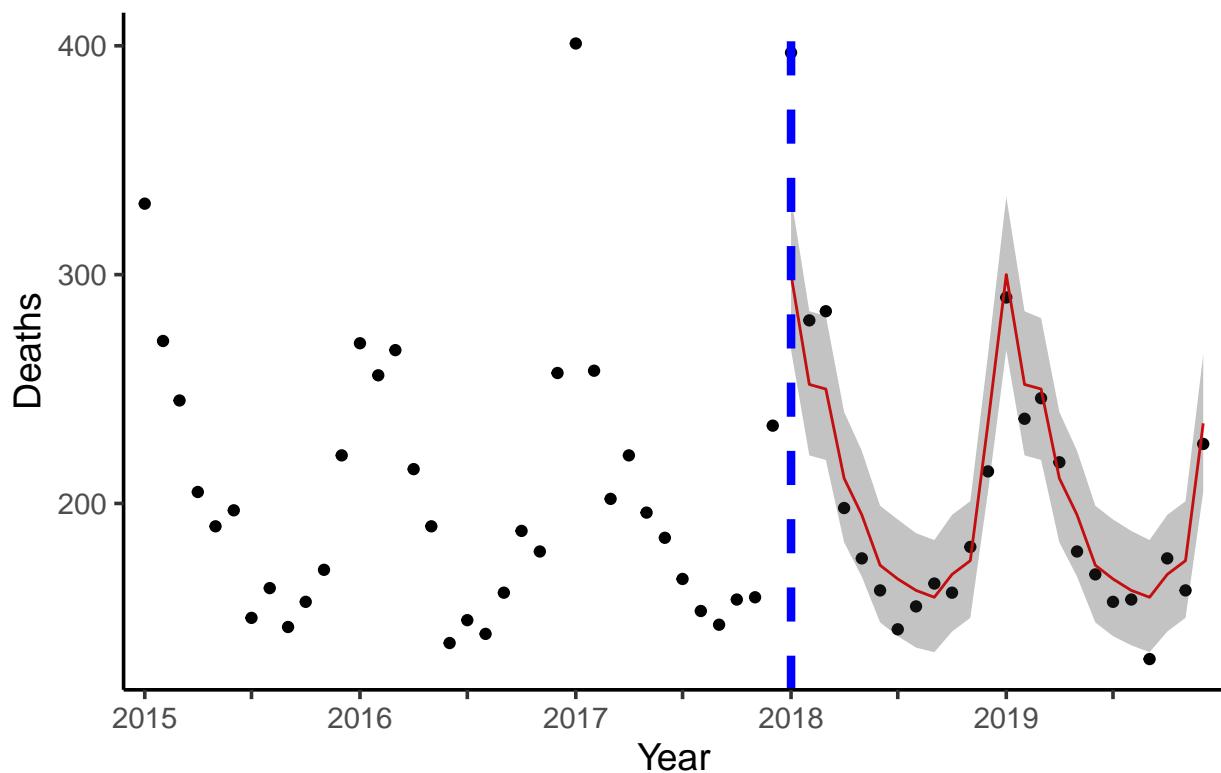
Cluster 2 (Coverage: 100%)



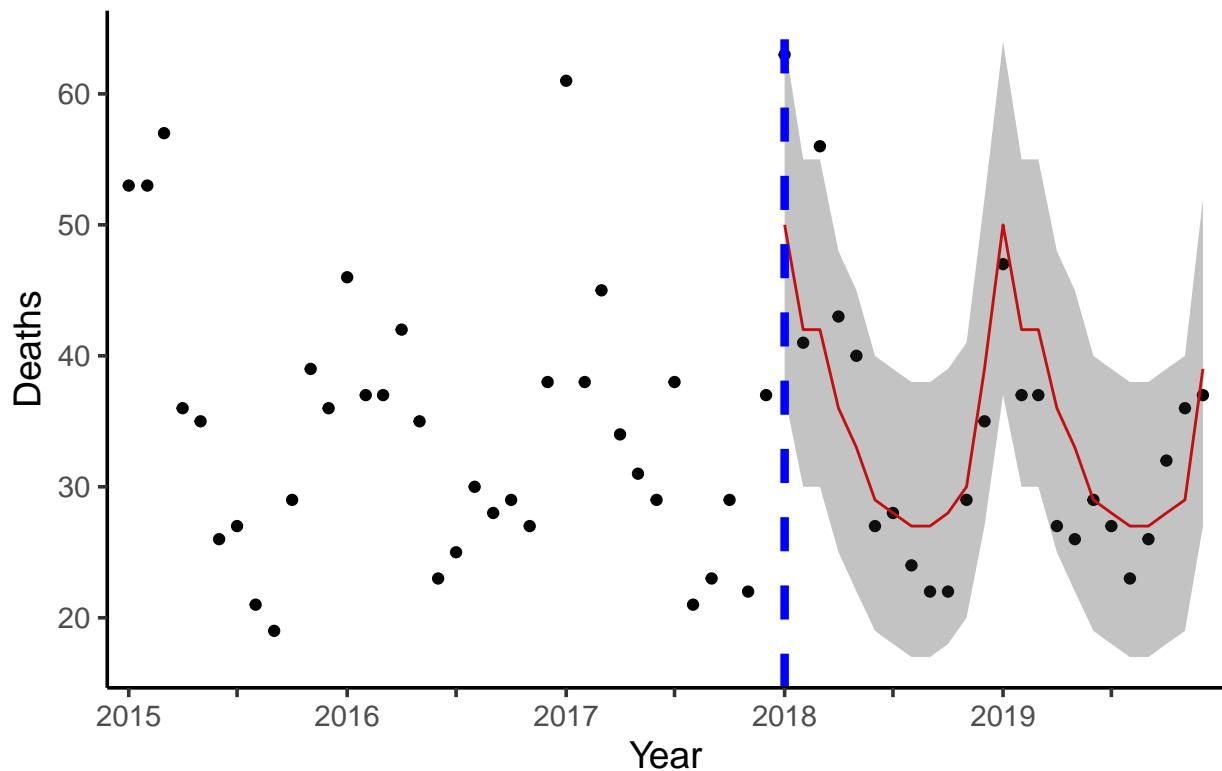
Cluster 3 (Coverage: 91.67%)



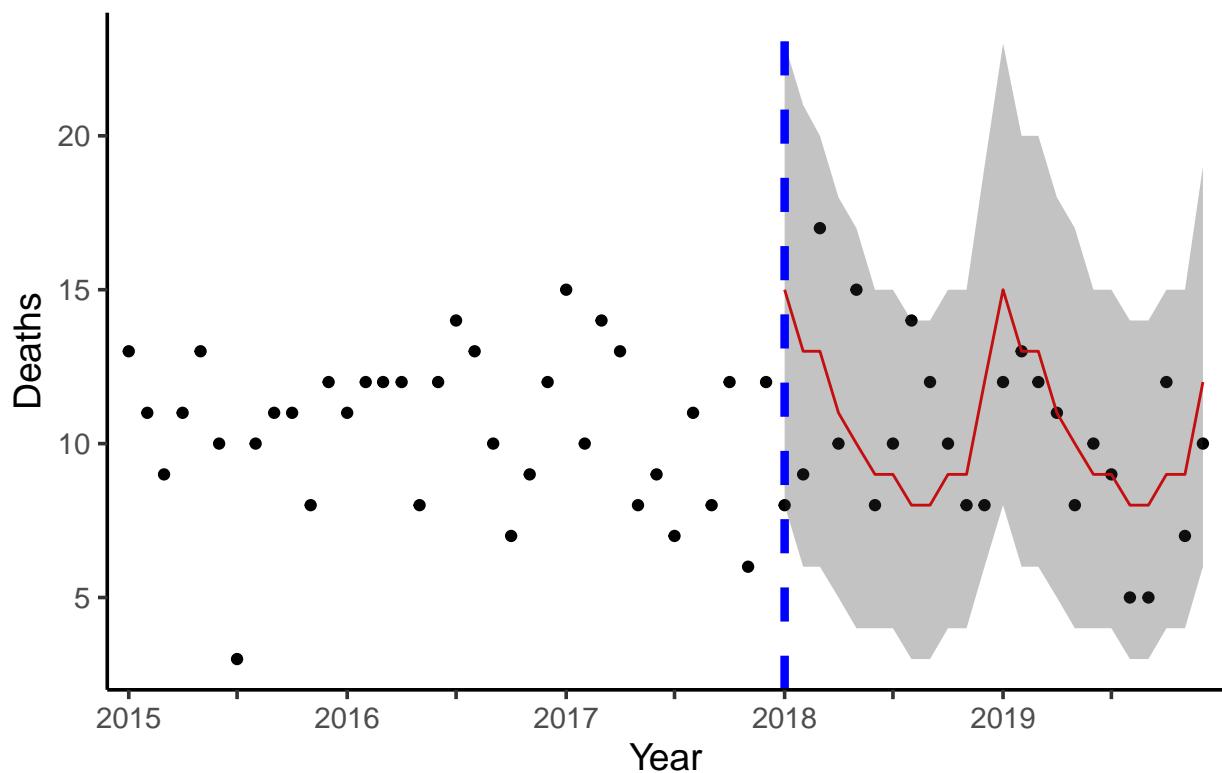
Cluster 4 (Coverage: 87.5%)



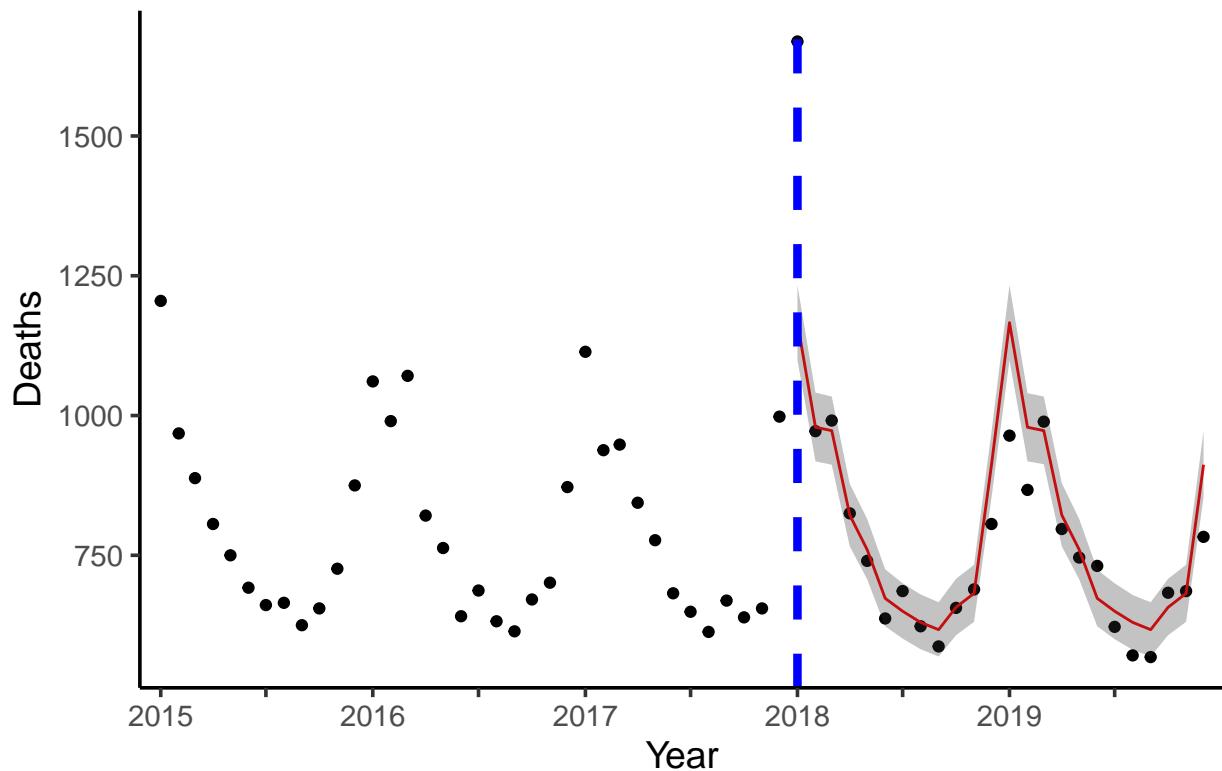
Cluster 5 (Coverage: 95.83%)



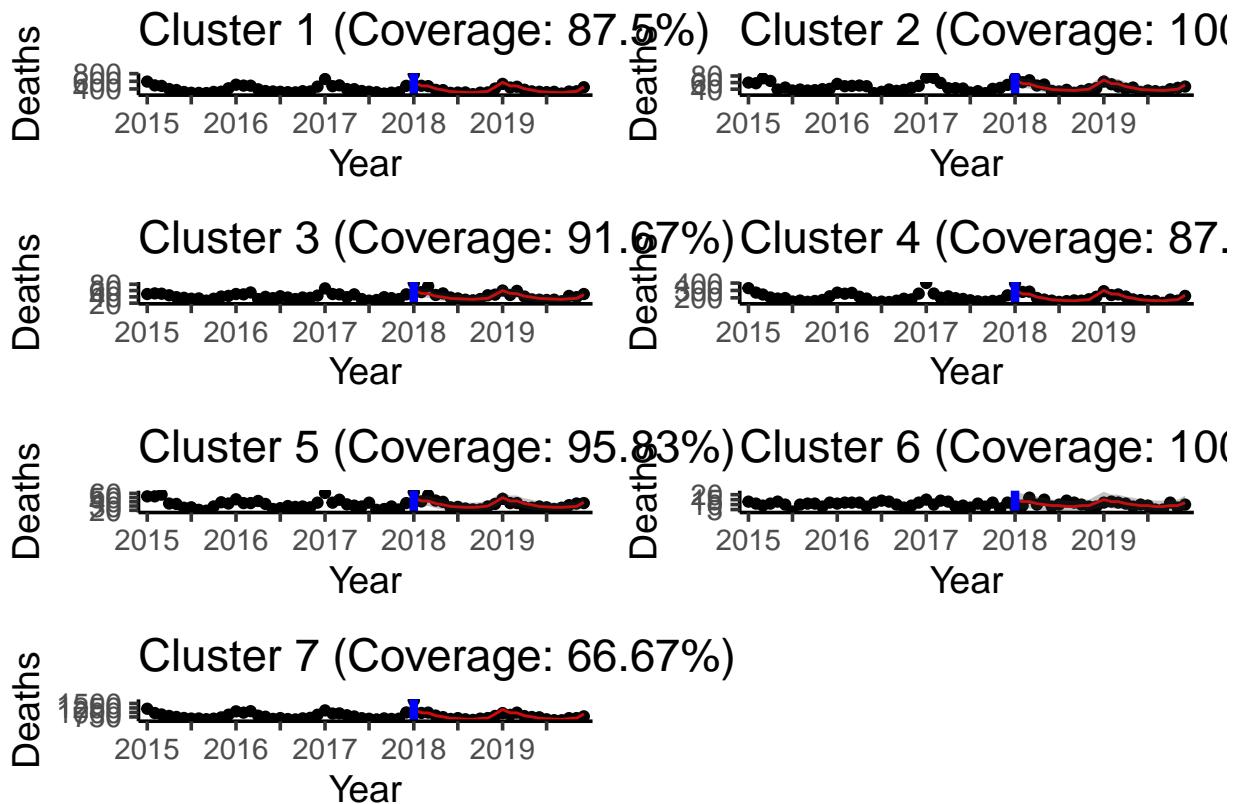
Cluster 6 (Coverage: 100%)



Cluster 7 (Coverage: 66.67%)



```
combined = wrap_plots(plotlist = plot_list, ncol = 2)
combined
```



```
# Save one combined figure
ggsave(
  filename = paste0("sliding_ref2_combined_plot.png"),
  plot = combined,
  path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/Paper1-images/",
  width = 10,
  height = 8,
  dpi = 600
)
```

Proposed model

```
prop_model_error = cbind(MAE2,MASE2,MAPE2,RMSPE2)
prop_model_error
```

	MAE2	MASE2	MAPE2	RMSPE2
[1,]	15.657143	0.026894549	0.09312820	22.107788
[2,]	9.028571	0.019388453	0.05444968	15.476064
[3,]	2.571429	0.027316508	0.04251310	3.454190
[4,]	3.971429	0.011154463	0.02774772	6.554824
[5,]	4.314286	0.065020968	0.03532398	5.611722
[6,]	2.285714	0.037842655	0.03936512	2.721344
[7,]	4.771429	0.083380386	0.04569034	6.993466
[8,]	3.714286	0.047612448	0.07132673	4.510306
[9,]	4.971429	0.090235454	0.07672738	6.948792
[10,]	1.428571	0.040076402	0.03426019	1.727095

```

## [11,]  4.800000 0.078982078 0.06973813  5.685068
## [12,]  9.085714 0.004112485 0.05705322 17.399507
## [13,] 16.228571 0.014778291 0.09371693 22.848820
## [14,]  8.600000 0.015762233 0.05384697 14.681183
## [15,]  2.714286 0.028693837 0.04265926  3.531895
## [16,]  3.828571 0.011254504 0.02757321  6.424951
## [17,]  4.314286 0.064446191 0.03532398  5.611722
## [18,]  2.285714 0.036653866 0.03936512  2.721344
## [19,]  5.057143 0.081792099 0.05042130  7.131019
## [20,]  3.714286 0.045759987 0.07132673  4.510306
## [21,]  5.400000 0.088242904 0.07841295  7.477586
## [22,]  1.571429 0.038931068 0.03447638  1.907878
## [23,]  4.800000 0.077793445 0.06973813  5.685068
## [24,]  9.085714 0.003146753 0.05705322 17.399507

#Plot of posterior predictive estimates (months 37-60) with credible interval bands OVERLAI on responses
true_mortality = inla_full_data

for (i in 1:num_clus){
  df = true_mortality %>% filter(id == i) %>% select(response)
  preds = starting_data2 %>% filter(id == i)
  colnames(preds)[3] = "mean"
  df = cbind(df,preds)

  title = sprintf("Cluster %s",i)

  post_pred_plot = df %>% ggplot(aes(x=time,y=response)) + geom_point() +
    geom_line(data = df %>% filter(time > 36), aes(y=mean),color = "red") + geom_ribbon(aes(ymin = lower,
    geom_vline(xintercept = 36,linetype = "dashed",color = "blue",linewidth = 1.5) + ggttitle(title) +
    theme_classic()
  print(post_pred_plot)
}

true_mortality = inla_full_data
plot_list = list()

for (i in 1:num_clus){
  df = true_mortality %>% filter(id == i) %>% select(response)
  preds = starting_data2 %>% filter(id == i)
  colnames(preds)[3] = "mean"
  df = cbind(df,preds)
  df$date = as.Date(paste0("2015-01-01")) + months(df$time - 1)

  test_idx = which(df$time >= 37)

  captured = (df$response[test_idx] >= df$lower[test_idx] & df$response[test_idx] <= df$upper[test_idx])
  coverage = round(sum(captured)/length(captured)*100,2)

  title = sprintf("Cluster %s (Coverage: %s%%)",i,coverage)

  # Define tick marks
  jan_breaks <- seq(as.Date("2015-01-01"), as.Date("2019-01-01"), by = "1 year")
  jul_breaks <- seq(as.Date("2015-07-01"), as.Date("2019-07-01"), by = "1 year")
  all_breaks <- sort(c(jan_breaks, jul_breaks))
}

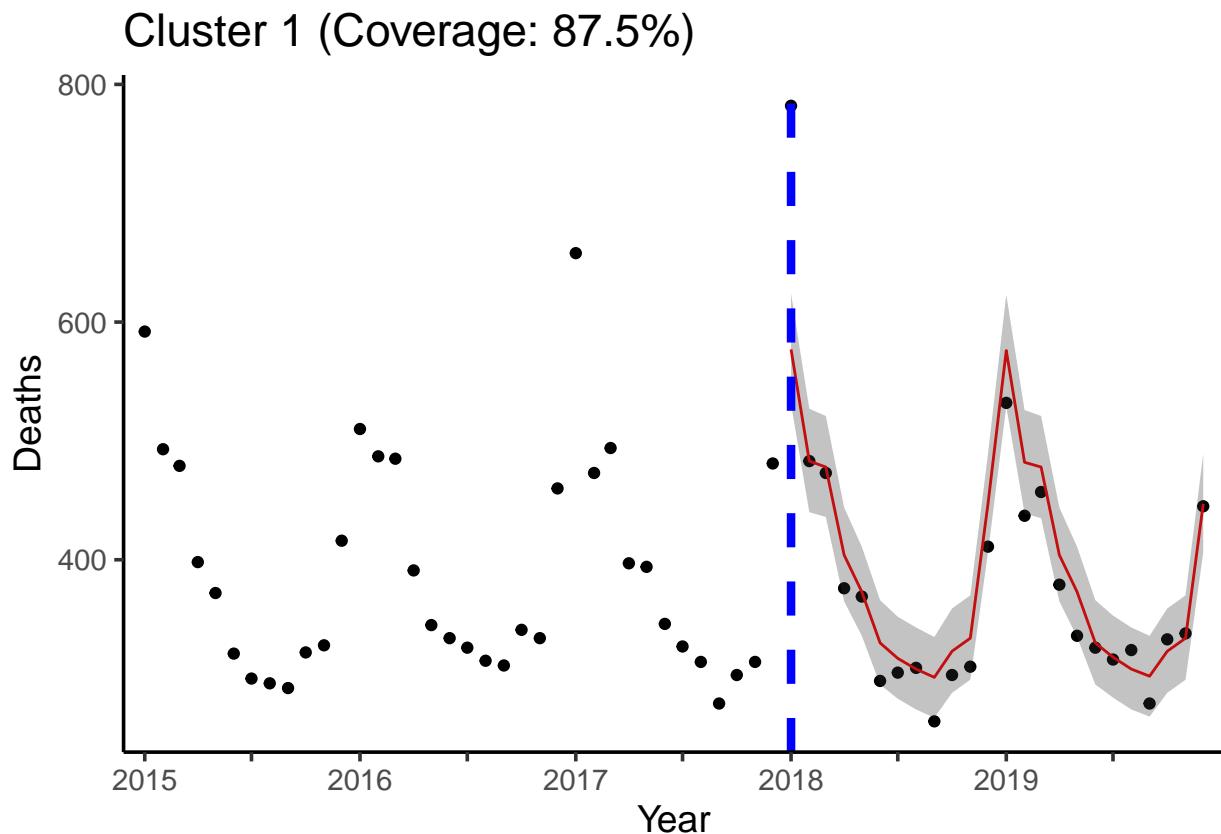
```

```

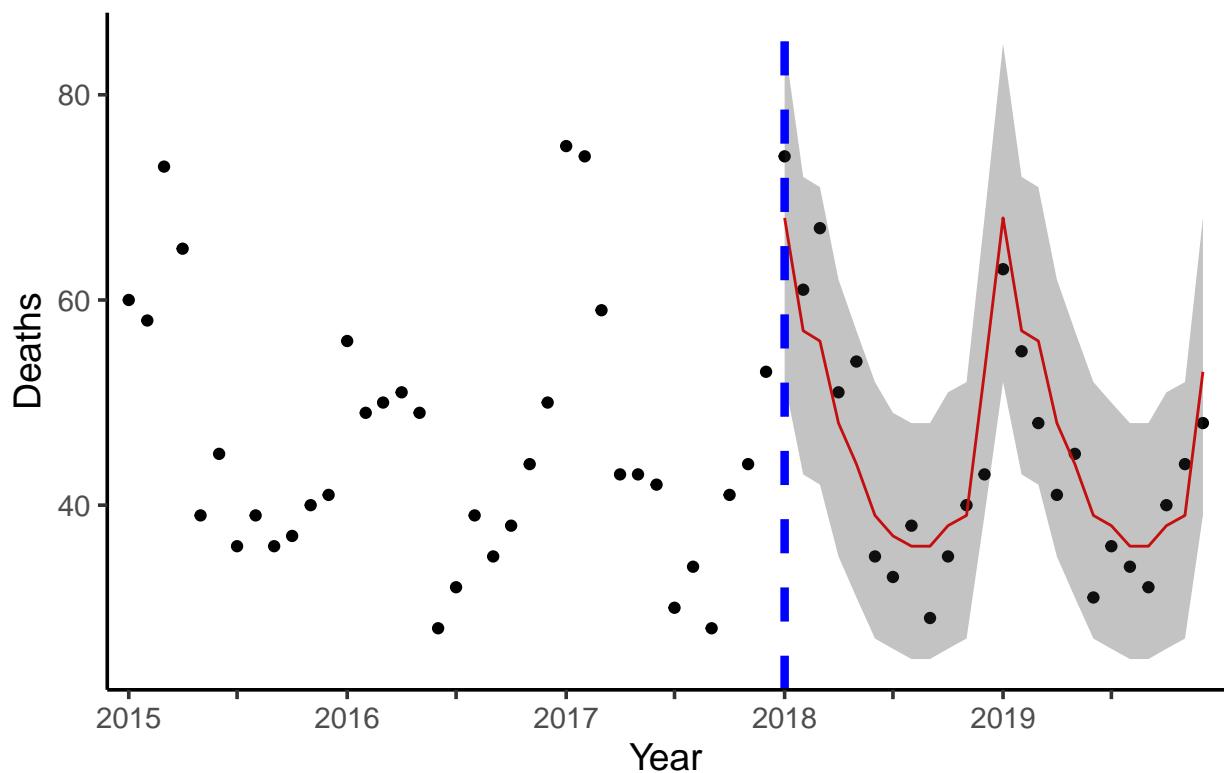
post_pred_plot = df %>%
  ggplot(aes(x = date, y = response)) +
  geom_point() +
  geom_line(data = df %>% filter(time > 36), aes(y = mean), color = "red") +
  geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.3) +
  geom_vline(xintercept = as.Date("2018-01-01"), linetype = "dashed", color = "blue", linewidth = 1.5) +
  ggttitle(title) +
  scale_x_date(
    name = "Year",
    breaks = all_breaks,
    labels = ifelse(format(all_breaks, "%m") == "01",
                   format(all_breaks, "%Y"), ""),
    expand = expansion(mult = 0.02) # ~2% space on each side
  ) +
  scale_y_continuous(name = "Deaths") +
  theme_classic(base_size = 14)

print(post_pred_plot)
plot_list[[i]] = post_pred_plot
}

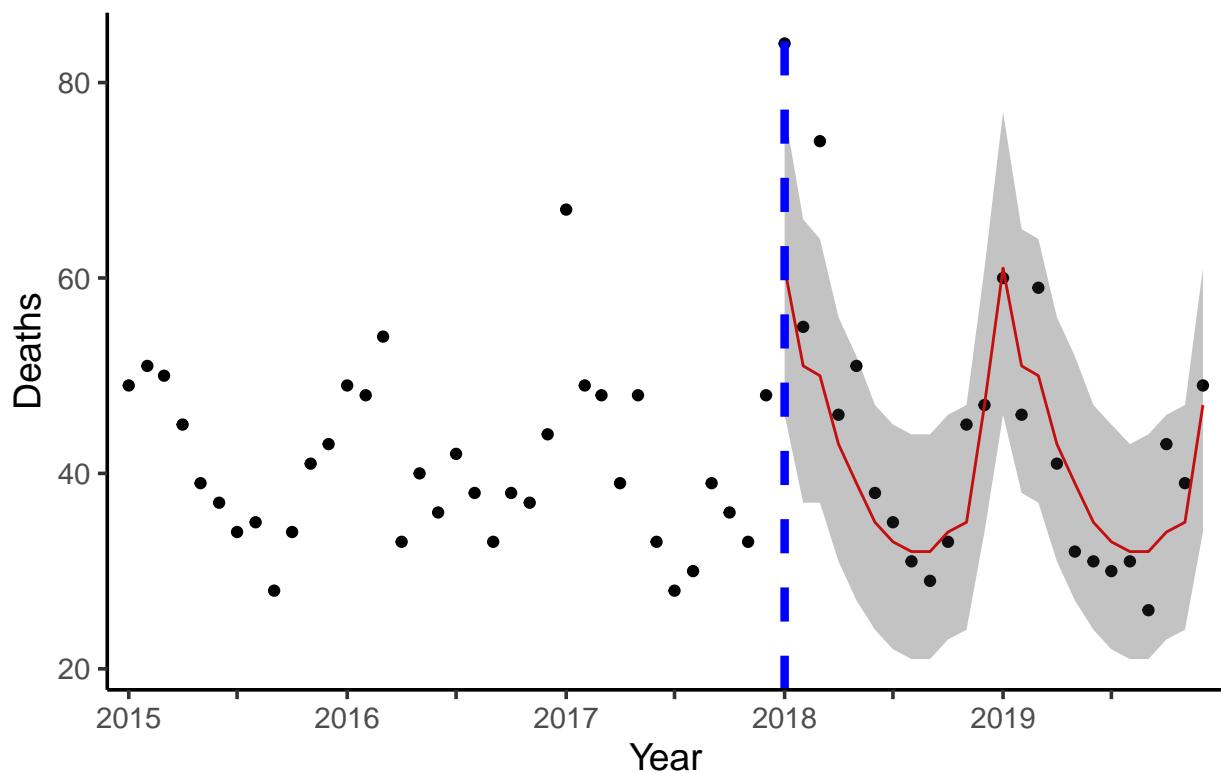
```



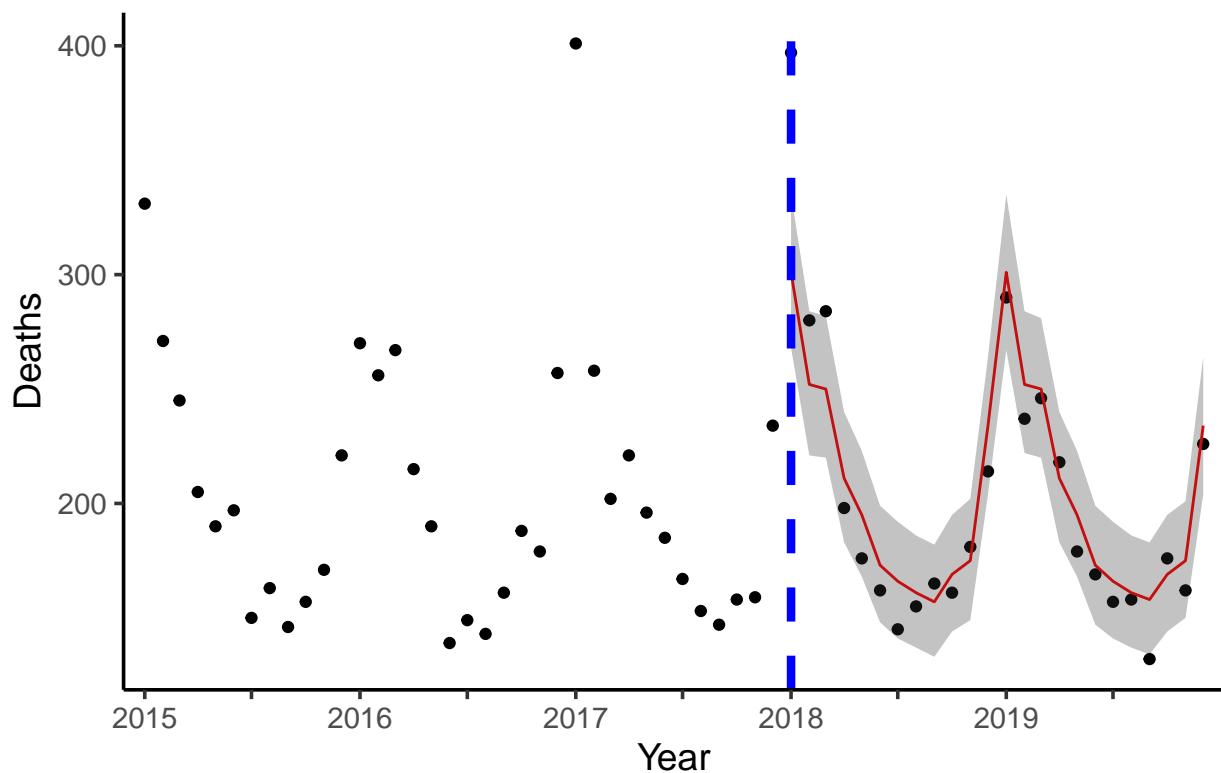
Cluster 2 (Coverage: 100%)



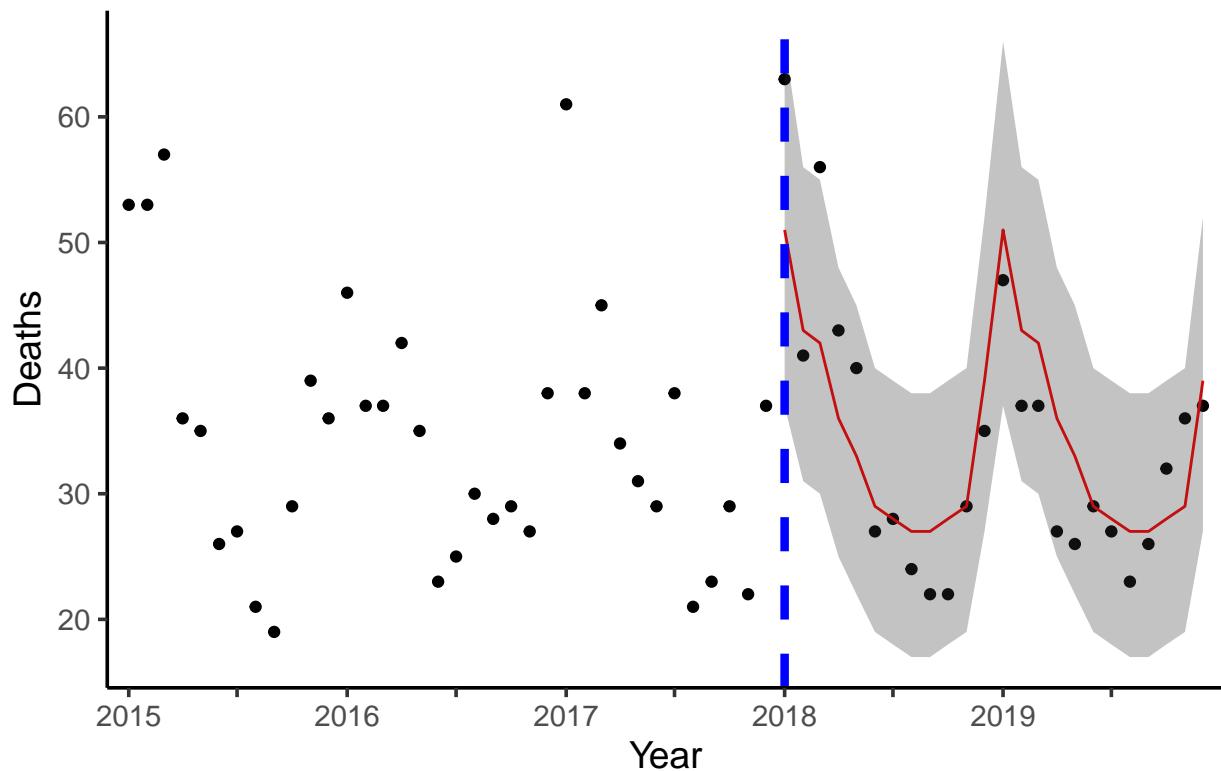
Cluster 3 (Coverage: 91.67%)

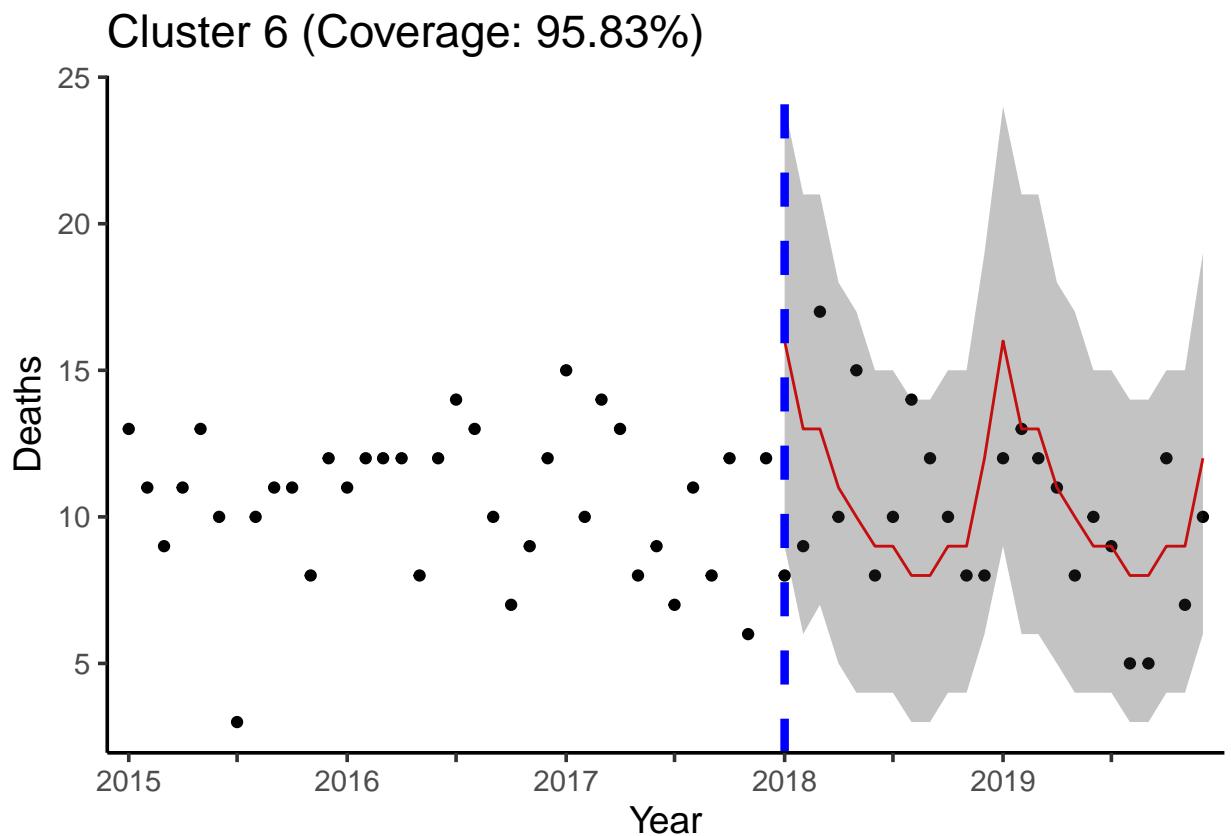


Cluster 4 (Coverage: 87.5%)

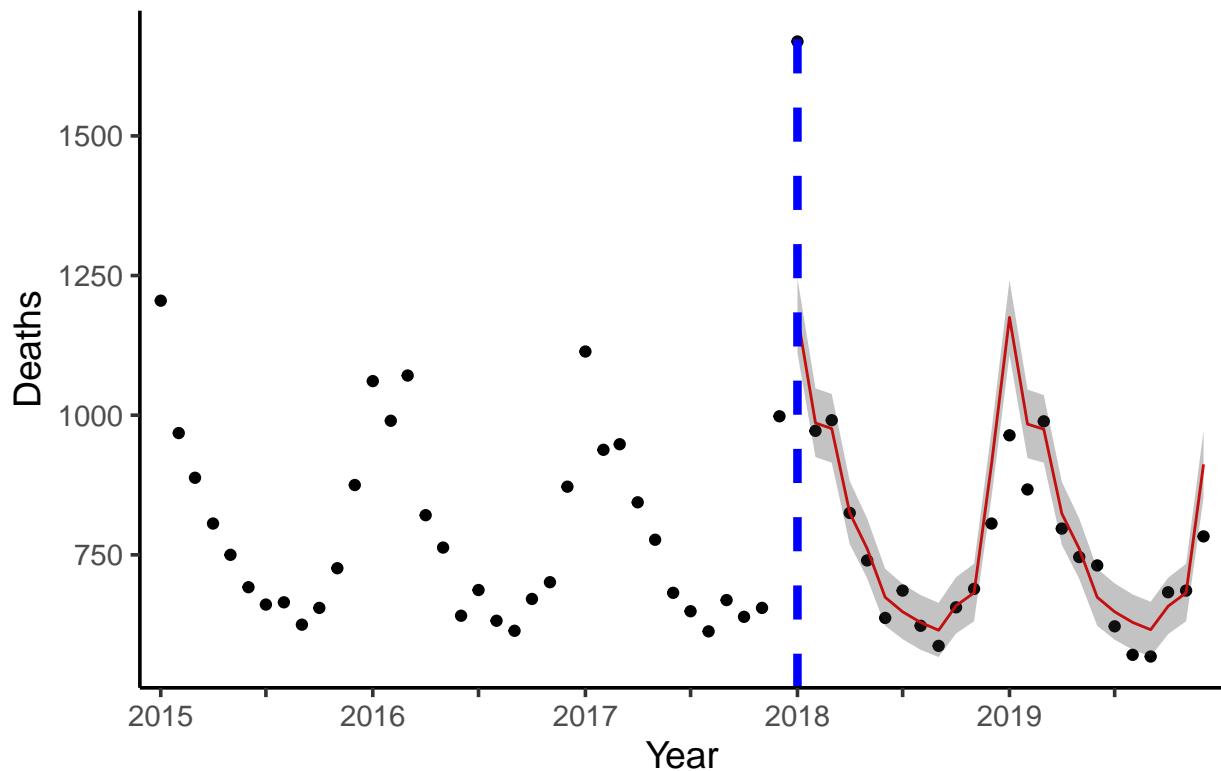


Cluster 5 (Coverage: 95.83%)

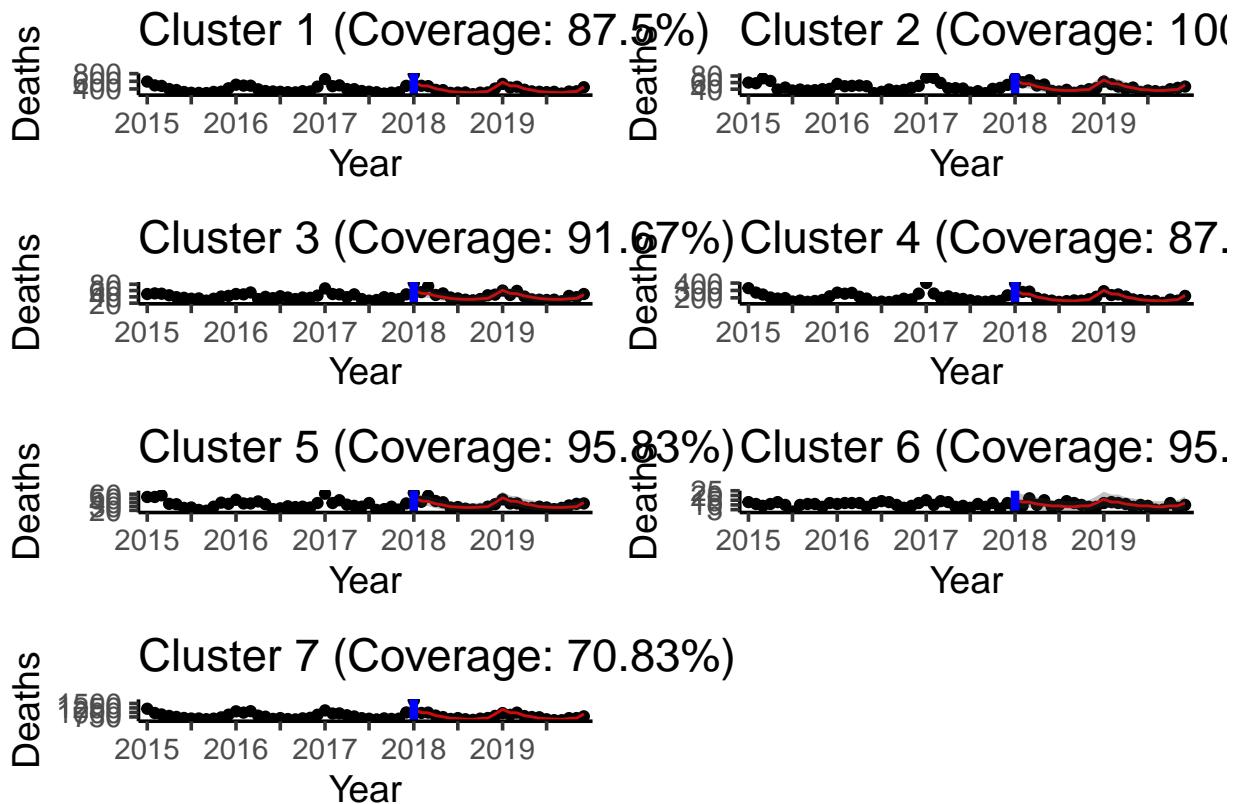




Cluster 7 (Coverage: 70.83%)



```
combined = wrap_plots(plotlist = plot_list, ncol = 2)
combined
```



```
# Save one combined figure
ggsave(
  filename = paste0("sliding_kgr4_combined_plot.png"),
  plot = combined,
  path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/Paper1-images/",
  width = 10,
  height = 8,
  dpi = 600
)
```

Grouped barplot for each error metric over 24 time points

```
# Create data frames for both tables
data1 <- data.frame(
  Time = 37:60, MAE, MASE, MAPE, RMSPE
)

data2 <- data.frame(
  Time = 37:60, MAE2, MASE2, MAPE2, RMSPE2
)
colnames(data2) = colnames(data1)

MAE_ratio = MAE/MAE2
MASE_ratio = MASE/MASE2
MAPE_ratio = MAPE/MAPE2
MAE_ratio = MAE/MAE2
```

```

# Add a column to each data frame to indicate the source table
data1$Source <- 'Ref model'
data2$Source <- 'Prop model'

# Combine the two data frames
combined_data <- rbind(data1, data2)

# Melt the combined data to long format
data_long <- melt(combined_data, id.vars = c("Time", "Source"), variable.name = "Metric", value.name = "Value")

plot1 = data_long |> pivot_wider(names_from="Source", values_from="Value") |>
  mutate(log_ratio = log(`Prop model`/`Ref model`)) |> ggplot() +
  geom_line(aes(x=Time, y=log_ratio)) +
  facet_wrap(~Metric, scales = "free_y") +
  labs(x = "Forecasted Month", y = "Log Ratio (Proposed/Reference)", title = "") +
  theme_classic(base_size = 14) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  geom_hline(yintercept=0, linetype="dashed")

# data_long |> pivot_wider(names_from="Source", values_from="Value") |>
#   mutate(log_ratio = log(`Prop model`/`Ref model`)) |> ggplot() +
#   geom_histogram(aes(x=log_ratio)) +
#   facet_wrap(~Metric, scales = "free_y") +
#   labs(x = "log Ratio", title = "") +
#   theme_bw(base_size = 14) +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
#   geom_vline(xintercept=0, linetype="dashed")

# Plot grouped bar charts using ggplot2
plot2 = ggplot(data_long, aes(x = factor(Time), y = Value, fill = Source)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.9)) +
  facet_wrap(~ Metric, scales = "free_y") +
  labs(x = "Forecasted Month", y = "Error Values", title = "") +
  theme_bw(base_size = 14) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

ggsave(
  filename = "rolling_forecast_error_ratio.png",
  plot = plot1,
  path = "C:/Users/jeffr/Desktop/Spatiotemporal + Causal Inference/Wildfire Paper 1 Code/Paper1-images/",
  width = 10,
  height = 8,
  dpi = 600
)

```

Looks the same visually but if we compare the actual numbers:

```

ref_model_error[,1] >= prop_model_error[,1]

## [1] TRUE FALSE TRUE FALSE FALSE TRUE TRUE FALSE TRUE TRUE FALSE TRUE
## [13] TRUE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TRUE TRUE TRUE TRUE
ref_model_error[,2] >= prop_model_error[,2]

## [1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE

```

```

## [13] FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE
ref_model_error[,3] >= prop_model_error[,3]

## [1] FALSE FALSE TRUE FALSE FALSE TRUE TRUE FALSE TRUE TRUE FALSE TRUE
## [13] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TRUE TRUE FALSE TRUE
ref_model_error[,4] >= prop_model_error[,4]

## [1] TRUE FALSE TRUE FALSE FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE
## [13] TRUE FALSE TRUE FALSE FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE

```

We see the proposed model does slightly better or as well in every forecasting metric

Coverage on forecasted values for each model:

```

coverage_sliding = rep(0,2)
true_values = inla_full_data$response[inla_full_data$time > 36]
models_sliding_fvs = list(starting_data[starting_data$time > 36,],starting_data2[starting_data2$time > 36,])

for (i in 1:2){
  lci = models_sliding_fvs[[i]] %>% select(lower)
  uci = models_sliding_fvs[[i]] %>% select(upper)

  captured = (true_values >= lci$lower & true_values <= uci$upper)
  coverage_sliding[i] = sum(captured)/length(captured)
}

coverage_sliding = data.frame(coverage_sliding)
colnames(coverage_sliding) = "95% coverage"
rownames(coverage_sliding) = c("BYM model","Proposed KGR model 4")
coverage_sliding

##                               95% coverage
## BYM model                  0.8988095
## Proposed KGR model 4      0.8988095

```

6 month sliding window forecast (IGNORE)

In this section, I implemented a forecasting exercise in which I start with 36 months of training data. I will use that data to estimate a model and then forecast 6 months ahead. Now, we slide the time window of interest and use months 7-36 AND the newly forecasted values to re-estimate the model and forecast again to get the next 6 months. This process continues until the original 36 months of data have been used to produce a complete time series of 60 months (the last 3 years are forecasted). This exercise gives us another way to compare the predictive ability of our various models.

Reference model iteration

```

starting_data = inla_full_data[1:252,]
starting_data$months = as.numeric(starting_data$months)
rownames(starting_data) = NULL
starting_data$lower = NA
starting_data$upper = NA

while(max(starting_data$time) < 60){

  ###Attach df for next 6 months with NAs in response

```

```

end = nrow(starting_data)
id = rep(1:7,6)
id2 = (starting_data$id2[end]+1):(starting_data$id2[end]+42)
response = rep(NA,42)
lower = rep(NA,7)
upper = rep(NA,7)
pop = cluster_pops
time = rep((starting_data$time[end]+1):(starting_data$time[end]+6),each=7)
Intercept1 = rep(c(1,NA,NA,NA,NA,NA,NA),6)
Intercept2 = rep(c(NA,1,NA,NA,NA,NA,NA),6)
Intercept3 = rep(c(NA,NA,1,NA,NA,NA,NA),6)
Intercept4 = rep(c(NA,NA,NA,1,NA,NA,NA),6)
Intercept5 = rep(c(NA,NA,NA,NA,1,NA,NA),6)
Intercept6 = rep(c(NA,NA,NA,NA,NA,1,NA),6)
Intercept7 = rep(c(NA,NA,NA,NA,NA,NA,1),6)

if (starting_data$months[end] == 6){
  months = rep(c(7,8,9,10,11,12),each=7)
} else if (starting_data$months[end] == 12){
  months = rep(c(1,2,3,4,5,6),each=7)
}

new_data = data.frame(id,id2,response,time,months,Intercept1,Intercept2,Intercept3,
                      Intercept4,Intercept5,Intercept6,Intercept7,pop,lower,upper)
starting_data = rbind(starting_data,new_data)
starting_data$months = factor(starting_data$months)

###Fit KGR model on most recent 36 months
starting_data_subset = starting_data[(nrow(starting_data)-293):end,]

ref_formula2 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5 +
  Intercept6 + Intercept7 + f(id, model = "bym", graph = huge.est)
ref_model2 = inla(ref_formula2,family = "poisson",data = starting_data_subset,
                  control.compute = list(dic=TRUE,waic=TRUE),
                  control.predictor = list(compute = TRUE, link = 1))

###Append ref model 2 predictions to starting data
preds_ref_model2 = ref_model2$summary.fitted.values
preds_ref_model2$mean = round(preds_ref_model2$mean)

end2 = nrow(preds_ref_model2)

pred_data = preds_ref_model2$mean[(end2-41):end2]
starting_data$response[(end+1):(end+42)] = pred_data

starting_data$months = as.numeric(starting_data$months)
}

#Plot of posterior predictive estimates (months 37-60) with credible interval bands OVERLAID on response
true_mortality = inla_full_data

for (i in 1:num_clus){
  df = true_mortality %>% filter(id == i) %>% select(response)
  preds = starting_data %>% filter(id == i)
}

```

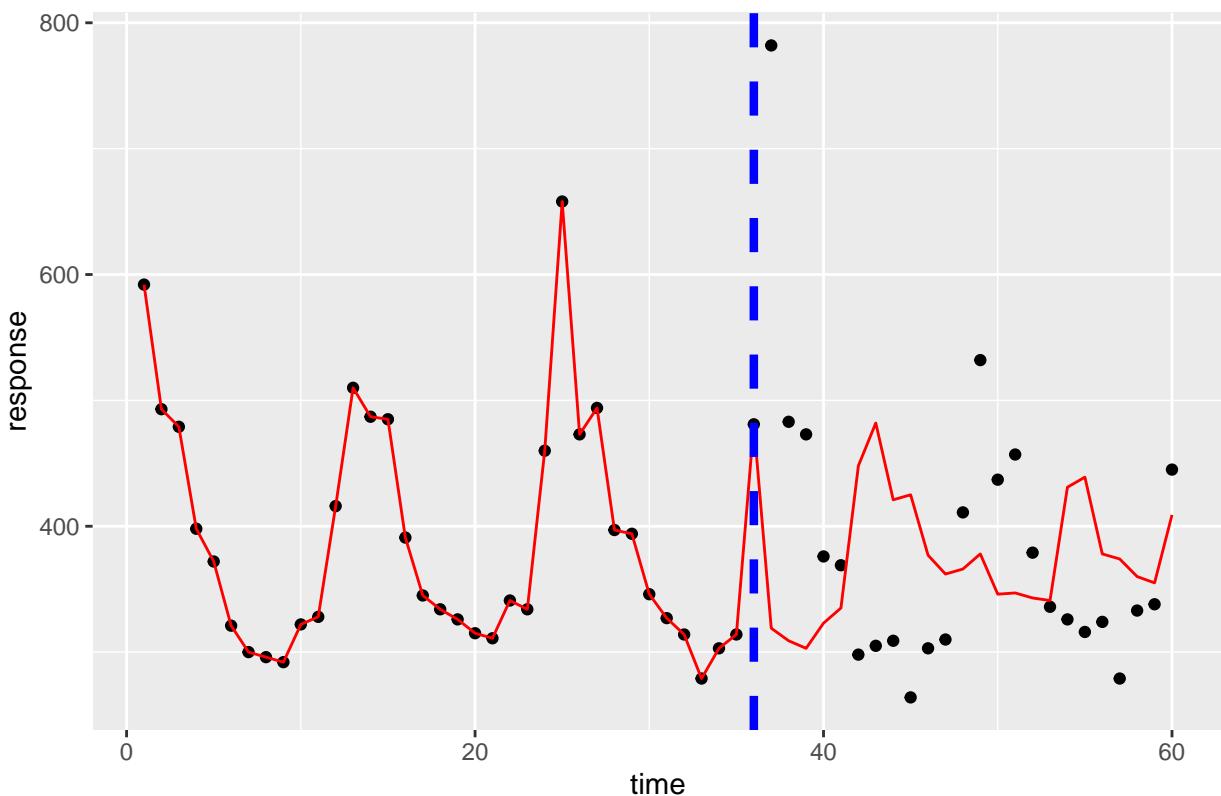
```

colnames(preds)[3] = "mean"
df = cbind(df,preds)

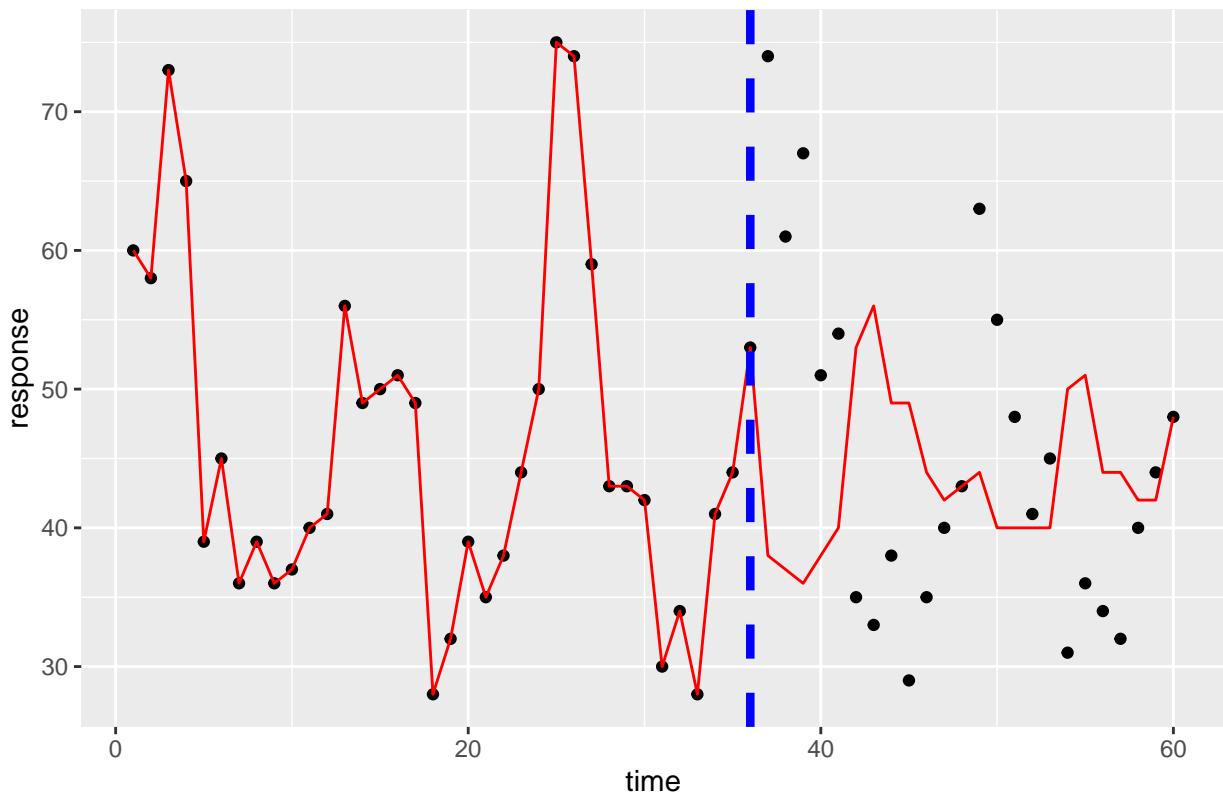
post_pred_plot = df %>% ggplot(aes(x=time,y=response)) + geom_point() +
  geom_line(aes(y=mean),color = "red") + geom_vline(xintercept = 36,linetype = "dashed",color = "blue")
print(post_pred_plot)
}

```

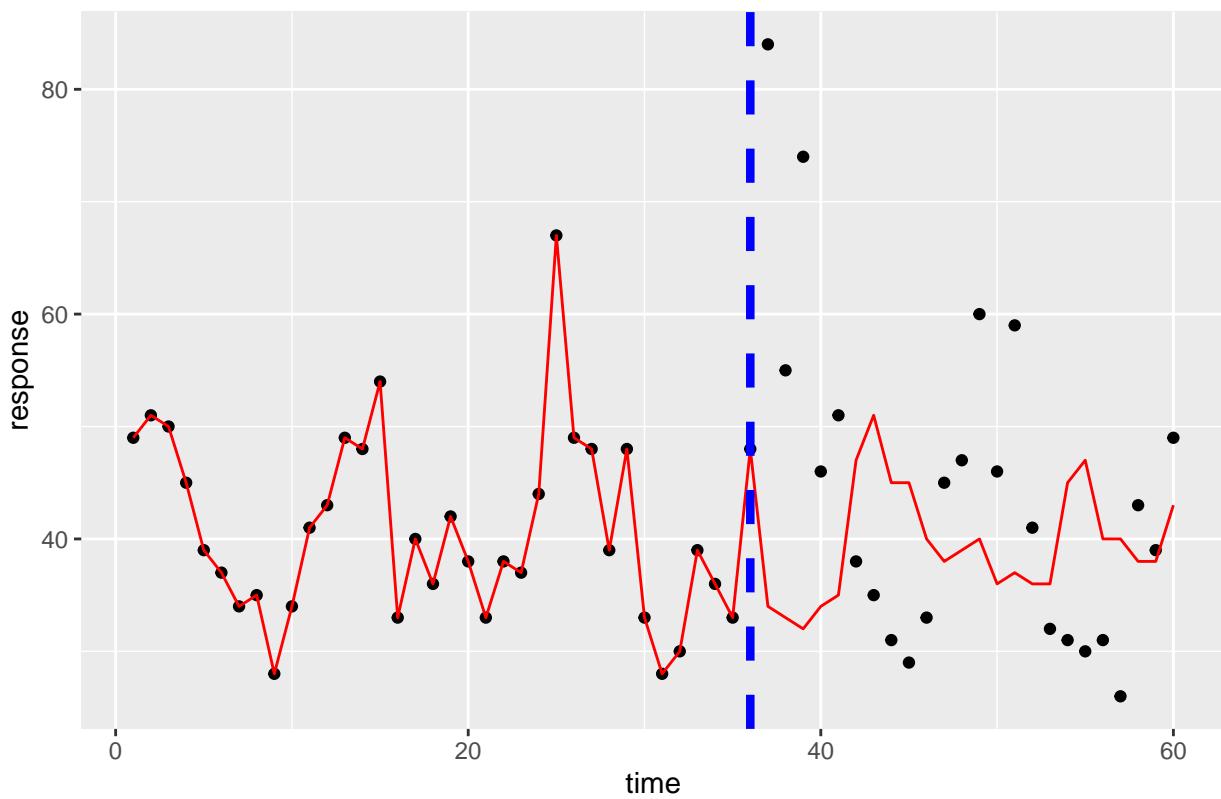
Sliding Timeframe Forecast for Cluster 1



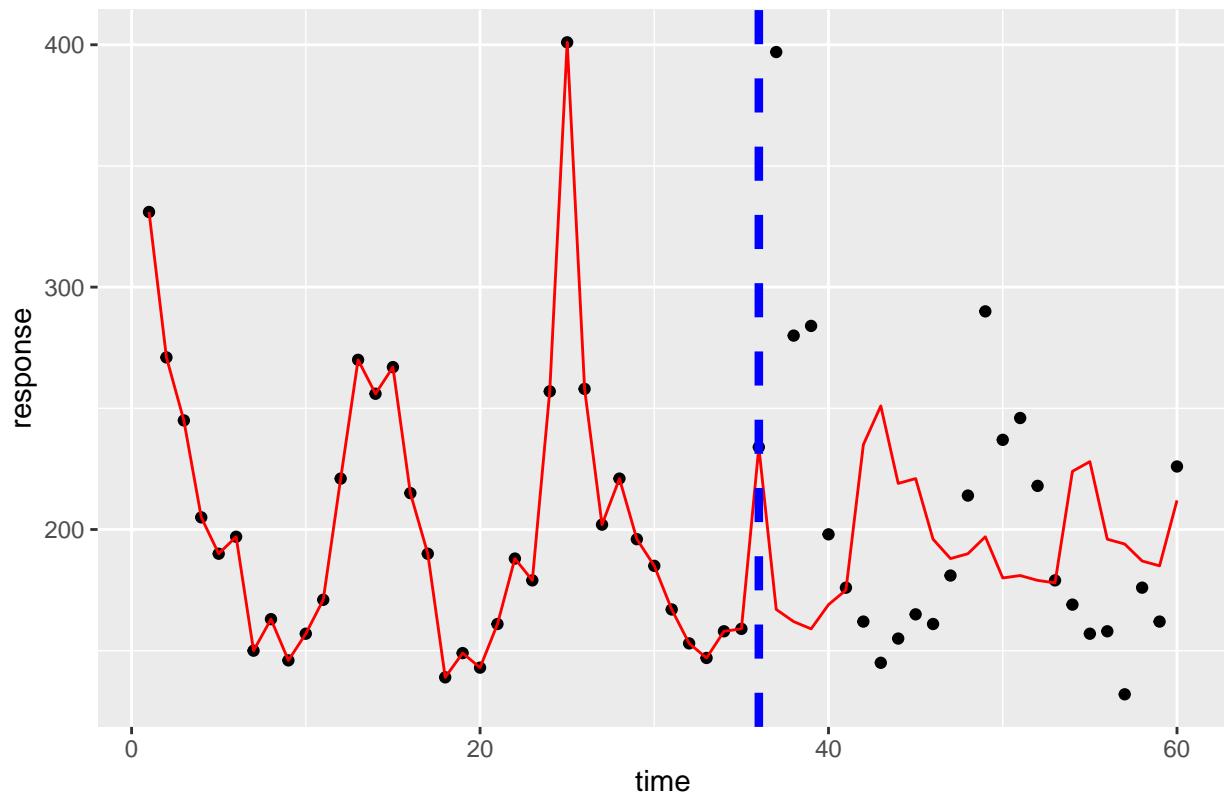
Sliding Timeframe Forecast for Cluster 2



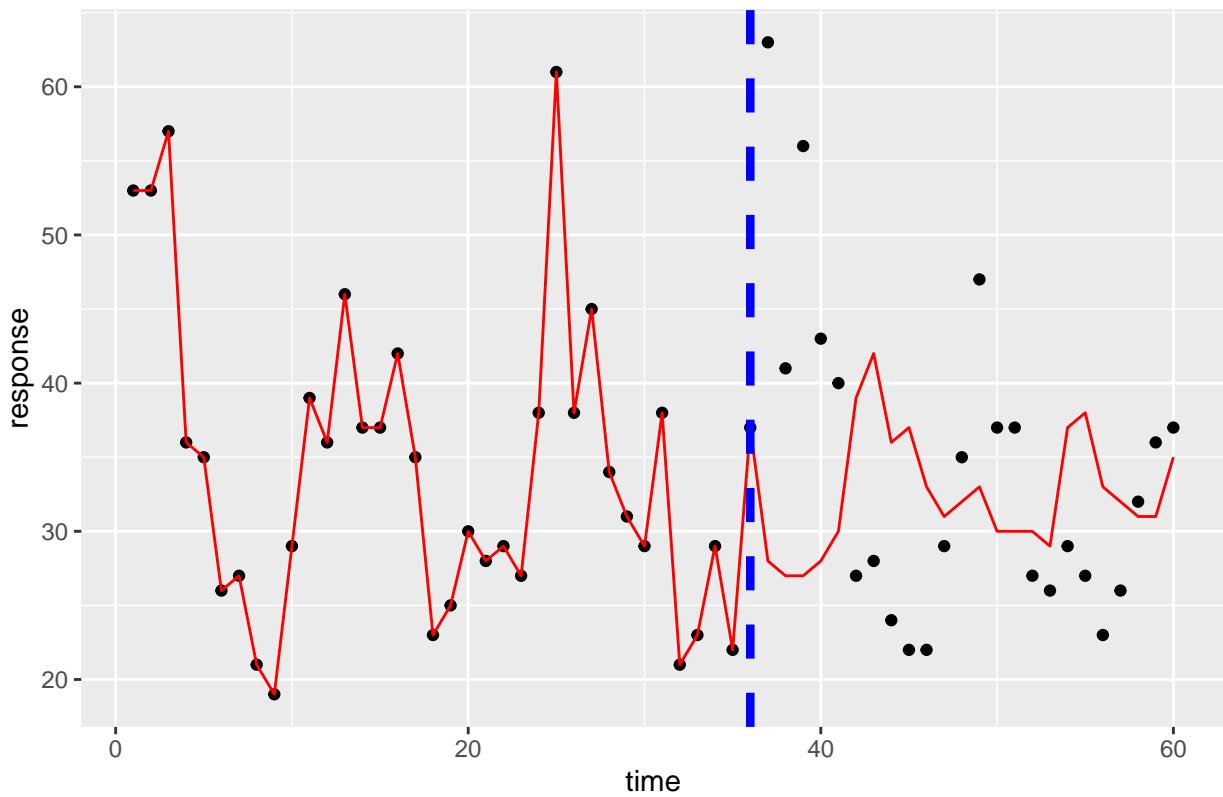
Sliding Timeframe Forecast for Cluster 3



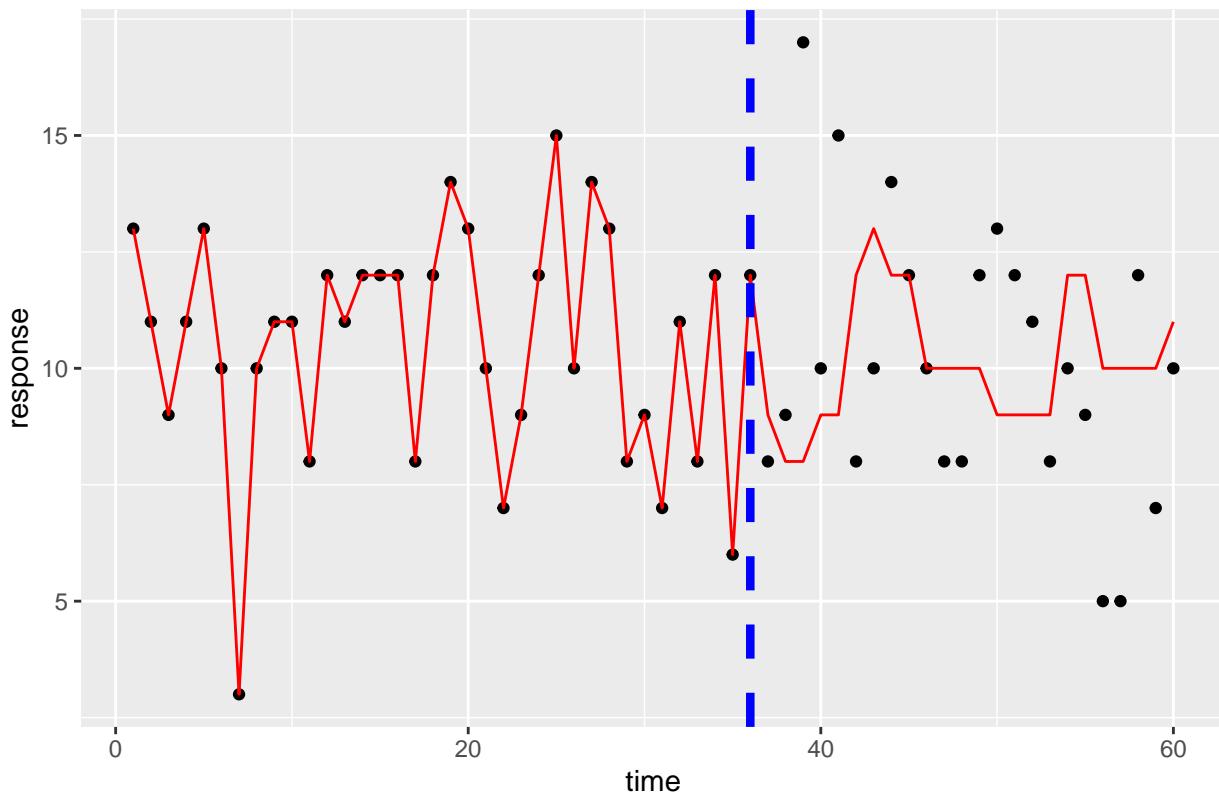
Sliding Timeframe Forecast for Cluster 4



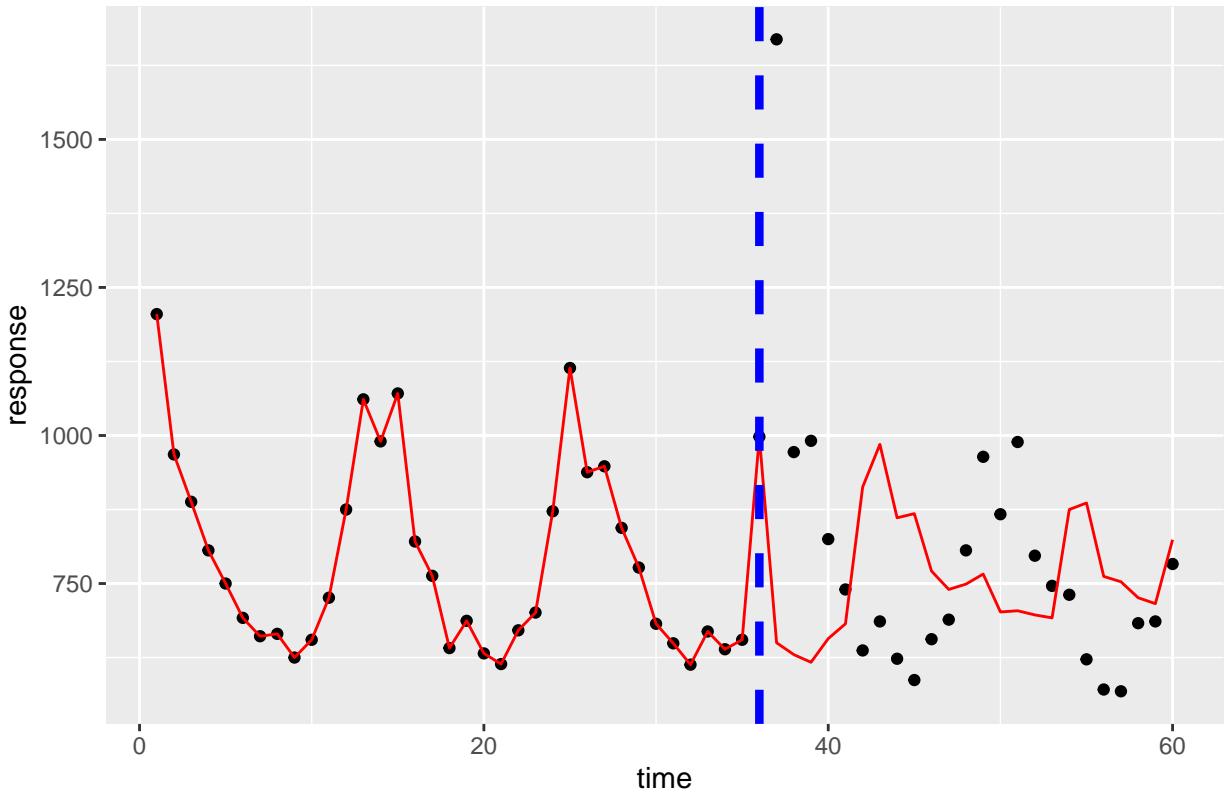
Sliding Timeframe Forecast for Cluster 5



Sliding Timeframe Forecast for Cluster 6



Sliding Timeframe Forecast for Cluster 7



Proposed model iteration

```

inv_covGP = kgr_model2_outfit$prec
inv_covGP3 = kgr_model3_outfit$prec
inv_covGP4 = kgr_model4_outfit$prec
inv_covGP5 = kgr_model5_outfit$prec

starting_data = inla_full_data[1:252,]
starting_data$months = as.numeric(starting_data$months)
rownames(starting_data) = NULL
starting_data$lower = NA
starting_data$upper = NA

while(max(starting_data$time) < 60){

  ###Attach df for next 6 months with NAs in response
  end = nrow(starting_data)
  id = rep(1:7,6)
  id2 = (starting_data$id2[end]+1):(starting_data$id2[end]+42)
  response = rep(NA,42)
  lower = rep(NA,7)
  upper = rep(NA,7)
  pop = cluster_pops
  time = rep((starting_data$time[end]+1):(starting_data$time[end]+6),each=7)
  Intercept1 = rep(c(1,NA,NA,NA,NA,NA,NA),6)
}

```

```

Intercept2 = rep(c(NA,1,NA,NA,NA,NA),6)
Intercept3 = rep(c(NA,NA,1,NA,NA,NA),6)
Intercept4 = rep(c(NA,NA,NA,1,NA,NA),6)
Intercept5 = rep(c(NA,NA,NA,NA,1,NA),6)
Intercept6 = rep(c(NA,NA,NA,NA,NA,1),6)
Intercept7 = rep(c(NA,NA,NA,NA,NA,NA),6)

if (starting_data$months[end] == 6){
  months = rep(c(7,8,9,10,11,12),each=7)
} else if (starting_data$months[end] == 12){
  months = rep(c(1,2,3,4,5,6),each=7)
}

new_data = data.frame(id,id2,response,time,months,Intercept1,Intercept2,Intercept3,
                      Intercept4,Intercept5,Intercept6,Intercept7,pop,lower,upper)
starting_data = rbind(starting_data,new_data)
starting_data$months = factor(starting_data$months)

###Fit KGR model on most recent 36 months
starting_data_subset = starting_data[(nrow(starting_data)-293):end,]

# #Proposed model 2
# kgr_formula2 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5
#
# kgr_model2 = inla(kgr_formula2, data = starting_data_subset, family = "poisson",
#                     control.predictor = list(compute = TRUE, link = 1))
#
# preds_kgr_model = kgr_model2$summary.fitted.values

# #Proposed model 3
# kgr_formula3 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5
#
# kgr_model3 = inla(kgr_formula3, data = starting_data_subset, family = "poisson",
#                     control.predictor = list(compute = TRUE, link = 1))
#
# preds_kgr_model = kgr_model3$summary.fitted.values

# #Proposed model 4
# kgr_formula4 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5
#
# kgr_model4 = inla(kgr_formula4, data = starting_data_subset, family = "poisson",
#                     control.predictor = list(compute = TRUE, link = 1))
#
# preds_kgr_model = kgr_model4$summary.fitted.values

#Proposed model 5
kgr_formula5 = response ~ -1 + months + Intercept1 + Intercept2 + Intercept3 + Intercept4 + Intercept5

kgr_model5 = inla(kgr_formula5, data = starting_data_subset, family = "poisson",
                  control.predictor = list(compute = TRUE, link = 1))

preds_kgr_model = kgr_model5$summary.fitted.values

```

```

###Append KGR model predictions to starting data
preds_kgr_model$mean = round(preds_kgr_model$mean)
end2 = nrow(preds_kgr_model)

pred_data = preds_kgr_model$mean[(end2-41):end2]
starting_data$response[(end+1):(end+42)] = pred_data

# starting_data$response = replace(starting_data$response, which(starting_data$response < 0), 0)
starting_data$months = as.numeric(starting_data$months)
}

#Plot of posterior predictive estimates (months 37-60) with credible interval bands OVERLAID on responses
true_mortality = inla_full_data

for (i in 1:num_clus){
  df = true_mortality %>% filter(id == i) %>% select(response)
  preds = starting_data %>% filter(id == i)
  colnames(preds)[3] = "mean"
  df = cbind(df,preds)

  post_pred_plot = df %>% ggplot(aes(x=time,y=response)) + geom_point() +
    geom_line(aes(y=mean),color = "red") + geom_vline(xintercept = 36,linetype = "dashed",color = "blue")
  print(post_pred_plot)
}

```

