Contents


```
options(knitr.kable.NA = '')
```

1 Climate futures for Haskell Indian Nations University

1.1 Prepare your environment

1. Install Climate Futures Toolbox

```
install.packages("cft")
```

2. Load other packages

```
library(tidyverse)
library(tidync)
library(cft)
library(sf)
library(ggplot2)
library(ggthemes)
library(ggpattern)
library(magick)
library(future)
library(tidytable)
library(janitor)
```

1.2 Finding basic layers in OpenStreetMap

Explain the basics of APIs and the premise of fast downloads.

1.2.1 Use plain lnaguage to request a bounding box

1. Find the general area on Open Street Map We use a function from the osmdata package to find a bounding box for our area of interest. This is a nice function for this purpose because it can use plan language declarations (e.g. “Lawrence, Kansas” or “Boulder, Colorado”) for the location. You do not need to use this function to define a bounding box. You can define your bounding box from any source. The benefit of this method is that is it is rather easy and reliable.

```
bb <- getbb("Lawrence, Kansas")
bb
```

```
##          min       max
## x -95.34454 -95.16662
## y  38.90447  39.03350
```

```
bbb <- bb
bbb[1,1] <- bbb[1,1] - 0.001
bbb[1,2] <- bbb[1,2] + 0.001
bbb[2,1] <- bbb[2,1] - 0.03
bbb[2,2] <- bbb[2,2] + 0.001
xlimit <- bbb[1,]
ylimit <- bbb[2,]
xmid <- xlimit[1] + diff(xlimit) / 2
ratio <- diff(xlimit) / diff(ylimit)
```

2. Find any buildings associated with any University in our “Lawrence, Kansas” bounding box. This request first calls the opq() function, which mounts to the OpenStreetMap database and then queries the “building” key (i.e. all the building footprints) for any building types matching the value “university”. This is a representation of the “key” and “value” system that OSM uses to query data. The final step is to convert the OSM output into a spatial format that works well in R, called sf.

```
my_boundary <- opq(bb) %>%
  add_osm_feature(key = "building", value = "university") %>%
osmdata_sf()

summary(my_boundary)
```

```
##                   Length Class  Mode
## bbox                1     -none- character
## overpass_call       1     -none- character
## meta                3     -none- list
## osm_points          5     sf     list
## osm_lines          40     sf     list
## osm_polygons       40     sf     list
## osm_multilines      0     -none- NULL
## osm_multipolygons  17     sf     list
```

3. The output from this request shows a list of multipolygons, polygons, linestrings, and points. Each of these data types have a different storage structure, so we can’t look at them all at the

same time. Instead, lets start with polygons, which likely represent a single building footprint. Printing 'my_boundary$osm_polygons' shows that there are two Universities in Lawrence and we need to filter those results down to only include Haskell building.

```
boundaries <- my_boundary$osm_polygons %>%
  filter(operator == "Haskell Indian Nations University")
boundaries1 <- boundaries[1,] #take the first building (e.g. first row) of
↪  the returns
```

osm_id	name	building	ele	gnis.county_id	gnis.created	gnis.feature_id	gnis.state_id	operator	geometry
172577406	Winona Hall	university	264	045	12/08/2008	2510678	20	Haskell Indian Nations University	POLYGON ((-95.23447 38.9396...

4. Plot our discovered footprint to visually confirm It looks like we found Winona Hall in the OpenStreetMap database. This is how we plot the perimeters associated with it.

```
basemap <- ggplot(data = st_as_sf(boundaries1)) +
  geom_sf(fill = "blueviolet") +
  geom_sf_text(aes(label = name), size=10) +
  theme_tufte()

basemap
```

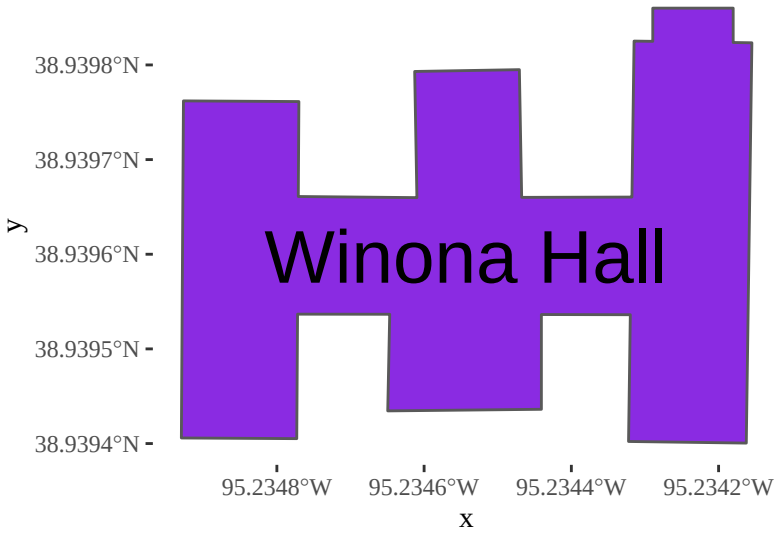


Figure 1: This seems to match.

1.3 Build a basemap from OpenStreetMap

1.3.1 Set your color palette

1. Download the Haskell university logo

```
seamless_image_filenames <- c(
  'Haskell_logo copy.jpg'
)
```

2. Sample the colors on that logo to make a custom color palette for our basemap

```
our_blue <- "#3A4E8B"
our_yellow <- "#FFD60F"
our_beige <- "#EDEDF0"
our_purple <- "#3E1471"
our_purple_alpha <- "#3E1471999"
```

1.3.2 Download all the layers you want to include in your basemap

1. Download the Haskell University footprint

```
my_boundary <- opq(bb) %>%
  add_osm_feature(key = "amenity", value = "university") %>%
osmdata_sf()

haskell_poly <- my_boundary$osm_multipolygons[1,]
```

2. Download street vector layers The street vector is divided into two different downloads in order to create two different objects for coloring in the final figure. This first download will be in the foreground. It includes the larger and faster roadways.

```
# the big streets
streets <-
  opq(bb) %>%
  add_osm_feature(key = "highway",
                  value = c("motorway", "trunk",  "primary",
                            "secondary", "tertiary")) %>%
  osmdata_sf()

streets_crop <- streets$osm_lines %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))
```

The second street download is for the small side streets and footpaths. These lines will be more faint and in the background.

```
small_streets <-
  opq(bb) %>%
  add_osm_feature(
    key = "highway",
    value = c("residential", "living", "unclassified",
              "service", "footway")) %>%
  osmdata_sf()

small_streets_crop <- small_streets$osm_lines %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))
```

3. Download water features. The water features are first divided into moving and stationary water. We will download the river layer from the waterway key.

```
water <-
  opq(bb) %>%
  add_osm_feature(key = "waterway", value = "river") %>%
  osmdata_sf()
```

We divide the water into large and small waterways in the same way we did with the road. We are interested in making the main river much larger and the remaining waterways collectively smaller. The Kansas river is the large feature in this map so, we pull it out first.

```
Kansas_river_multi <- water$osm_multilines %>%
  filter(name == "Kansas River")  %>%
  st_as_sf() %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))
```

After removing the Kansas river, we are left with a number of remaining waterways that are stored as both linestrings and multilinestrings. We need to download each of those data types individually.

```
small_water_multi <- water$osm_multilines %>%
  filter(name != "Kansas River")%>%
  st_as_sf() %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))

small_water_lines <- water$osm_lines %>%
  filter(name != "Kansas River")%>%
  st_as_sf() %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))
```

The stationary water bodies are divided into two categories: lakes and reservoirs. We need to pull each of these individually. First, the reservoir.

```
reservoir <-
  opq(bb) %>%
  add_osm_feature(key = "water", value = "reservoir") %>%
  osmdata_sf()

reservoir_crop_multi <- reservoir$osm_multipolygons %>%
  st_as_sf() %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))

reservoir_crop_poly <- reservoir$osm_multipolygons %>%
  st_as_sf() %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))

reservoir_crop <- cbind(reservoir_crop_multi, reservoir_crop_poly)
```

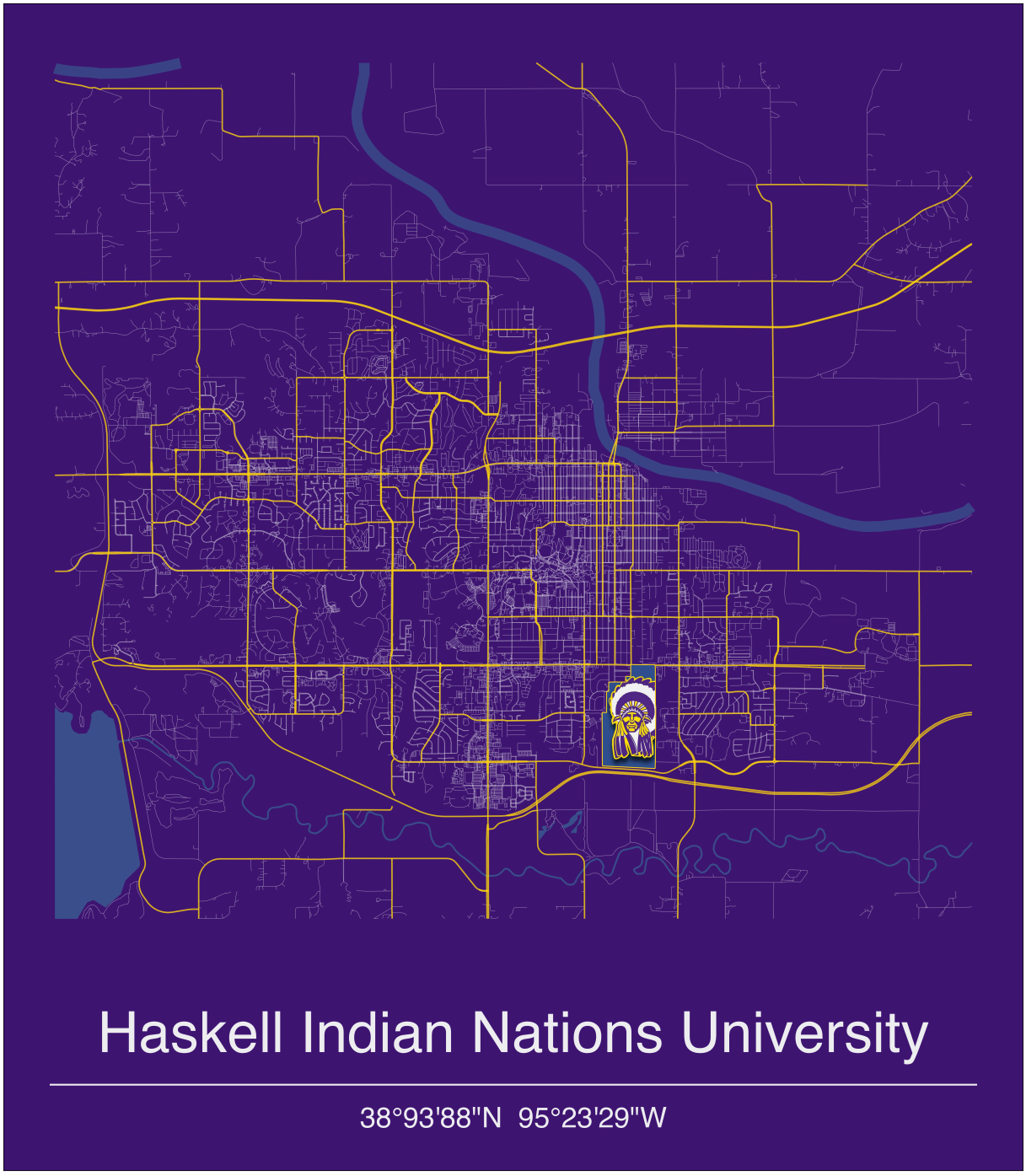
Now the lakes.

```
lake <-
  opq(bb) %>%
  add_osm_feature(key = "water", value = "lake") %>%
  osmdata_sf()

lake_crop <- lake$osm_polygons %>%
  st_as_sf() %>%
  st_crop(y = c(ymin = bb[2,1], ymax = bb[2,2], xmin = bb[1,1], xmax =
  ↪  bb[1,2]))
```

1.3.3 Stack downloaded OSM layers into a final basemap.

```
haskel_basemap <- ggplot() +
  # plot moving water layers first
  geom_sf(data = Kansas_river_multi, alpha = .8,
          size = 3, colour = our_blue) +
  geom_sf(data = small_water_multi, alpha = .8,
          size = 0.5, colour = our_blue) +
  geom_sf(data = small_water_lines, alpha = .8,
          size = 0.5, colour = our_blue) +
  # Layer bodies of water over the moving water layers
  geom_sf(data = lake_crop, alpha = 1, fill = our_blue, size=0) +
  geom_sf(data = reservoir_crop, alpha = 1, fill = our_blue, size=0) +

  # Plot small streets in the background with a light fade
  geom_sf(data = small_streets_crop, alpha = .6,
          size = .1, colour = our_beige) +
  # Layer large streets over the top of the small streets with a bold
  ↪  color.
  geom_sf(data = streets_crop, alpha = .8,
          size = .4, colour = our_yellow ) +
  # Layer Haskell university property polygon in the foreground
  geom_sf( data=haskell_poly, color=our_yellow, size=0.5) +
  # Fill Haskell property polygon with Haskell logo
   geom_sf_pattern(
     data = haskell_poly,
     size=0,
    pattern       = 'image',
```

```
  pattern_type  = 'tile',
  pattern_scale = 1.1,
  pattern_filename = seamless_image_filenames
) +
# set limits on final figure
coord_sf(ylim = ylimit, xlim = xlimit, expand = TRUE) +
# adding labels
annotate(geom = "text", y = bbb[2,1]+ 0.013, x = xmid,
         label = "Haskell Indian Nations University", size = 12, colour =
         ↪  our_beige
         ) +
annotate(geom = "errorbarh", xmin = xlimit[1], xmax = xlimit[2], y =
↪  bbb[2,1]+ 0.005,
         height = 0, size = 0.5, colour = our_beige) +
annotate(geom = "text", y = bbb[2,1]+ 0.0001, x =  xmid,
         label = "38°93'88\"N  95°23'29\"W",  size = 6,
         colour = our_beige) +
# clean out unused elements and set background color
theme_void() +
theme(panel.background = element_rect(fill = our_purple),
      plot.background = element_rect(fill = NA))
```

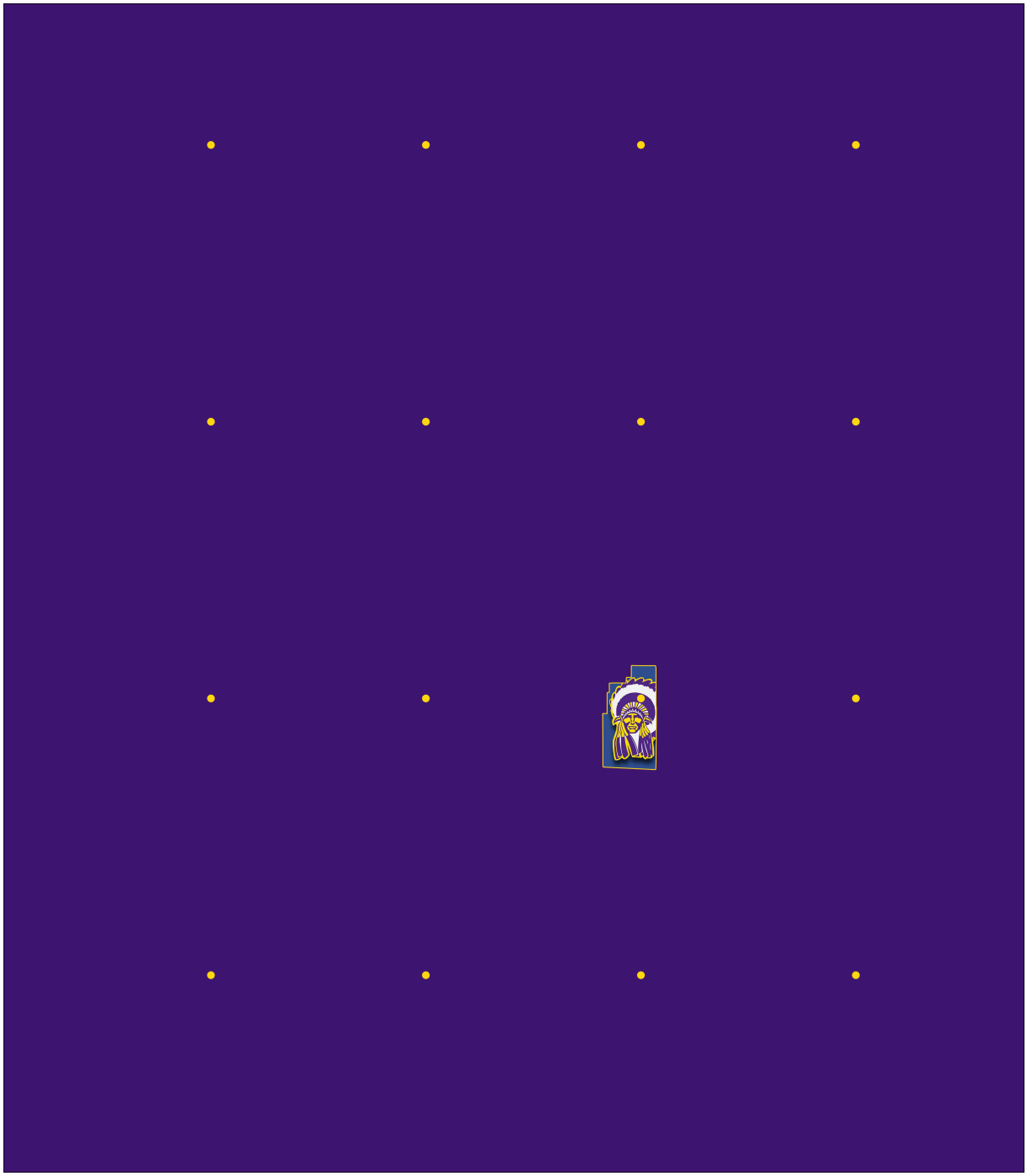

Haskell Indian Nations University
38°93'88"N 95°23'29"W

1.3.4 Save the basemap in high resolution print

```
ggsave(haskel_basemap, filename = "All_roads_lead_to_Haskell.png", height =
↪  11, width=8.5,
       units="in", dpi=600)
```

1.4 Mount the climate dataset

This dataset is way too big to download to a particular machine. Instead, you mount to the analysis ready data cube (i.e. netCDF) and only download the subsetted data that you want to pull.



1. We calculate the center point for measuring to the nearest climate data point.

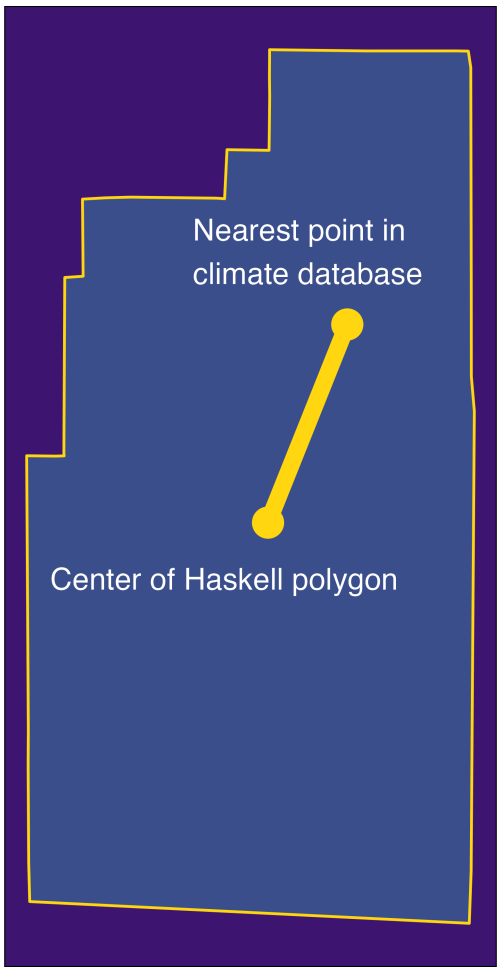
```
haskel_centroid <- st_coordinates(st_centroid(haskell_poly))
lat_pt <- haskel_centroid[1,2]
lon_pt <- haskel_centroid[1,1]
```

2. Connect to the web server and activate the proper data dimensions.

```
web_link = "https://cida.usgs.gov/thredds/dodsC/macav2metdata_daily_future"

# Change to
↪  "https://cida.usgs.gov/thredds/catalog.html?dataset=cida.usgs.gov/macav2metdata_dail
↪  for historical data.

src <- tidync::tidync(web_link)
lons <- src %>% activate("D2") %>% hyper_tibble()
lats <- src %>% activate("D1") %>% hyper_tibble()
```

3. Search through the database of climate prediction points to find which one is closest to our centroid. We then spatially project that chosen pt into an sf object.

```
known_lon <- lons[which(abs(lons-lon_pt)==min(abs(lons-lon_pt))),]
known_lat <- lats[which(abs(lats-lat_pt)==min(abs(lats-lat_pt))),]

chosen_pt <- st_as_sf(cbind(known_lon,known_lat), coords = c("lon", "lat"),
↪  crs = "WGS84", agr = "constant")
```



1.5 Downloading climate data for our bounding box.

1.5.1 Connect to the downscaled dataset.

We cannot download the entire dataset. Instead, we mount to that dataset by connecting to an api that route us to a particular part of the dataset based on the bounding box specified. Mount to the USGS downscaled dataset. The resultant object called 'input' includes three elements The first is the full list of available data at each timestep. The second is a list of possible time steps. The third is a list of verbatim copy of the raw return from the server. This raw return shows the dimensions of the data and how many elements are available in each of those dimensions.

```
# Mount to the USGS downscaled dataset.
inputs <- cft::available_data()
```

```
## Trying to connect to the USGS.gov API
```

```
## not a file:
## ' https://cida.usgs.gov/thredds/dodsC/macav2metdata_daily_future '
##
## ... attempting remote connection

## Connection succeeded.

## Reading results

## Converting into an R data.table
```

Element 1. Dataframe of availble data and descriptions of each of those variables.

```
head(inputs[[1]])
```

```
## # A tibble: 6 x 9
##   Available variab~1 Varia~2 Units Model Model~3 Scena~4 Varia~5 Model~6 Scen
##   <chr>              <chr>   <chr> <chr> <chr>   <chr>   <chr>   <chr>   <chr
## 1 huss_BNU-ESM_r1i1~ Specif~ kg k~ Beij~ r1i1p1  RCP 4.5 huss    BNU-
ESM rcp45
## 2 huss_BNU-ESM_r1i1~ Specif~ kg k~ Beij~ r1i1p1  RCP 8.5 huss    BNU-
ESM rcp85
## 3 huss_CCSM4_r6i1p1~ Specif~ kg k~ Comm~ r6i1p1  RCP 4.5 huss    CCSM4   rcp4
## 4 huss_CCSM4_r6i1p1~ Specif~ kg k~ Comm~ r6i1p1  RCP 8.5 huss    CCSM4   rcp8
## 5 huss_CNRM-CM5_r1i~ Specif~ kg k~ Cent~ r1i1p1  RCP 4.5 huss    CNRM-
C~ rcp45
## 6 huss_CNRM-CM5_r1i~ Specif~ kg k~ Cent~ r1i1p1  RCP 8.5 huss    CNRM-
C~ rcp85
## # ... with abbreviated variable names 1: `Available variable`, 2: Variable,
## #   3: `Model ensemble type (only CCSM4 relevant)`, 4: Scenario,
## #   5: `Variable abbreviation`, 6: `Model abbreviation`,
## #   7: `Scenario abbreviation`
```

Element 2. Short list of available data you can request

```
head(unique(inputs[[1]]$Variable))
```

```
## [1] "Specific Humidity"                    "Precipitation"
## [3] "Maximum Relative Humidity"            "Minimum Relative Humidity"
## [5] "Surface Downswelling Shortwave Flux" "Maximum Temperature"
```

Element 3. Dataframe of available times

```
head(inputs[[2]])
```

```
##   Available times      dates
## 1           38716 2006-01-01
## 2           38717 2006-01-02
## 3           38718 2006-01-03
## 4           38719 2006-01-04
## 5           38720 2006-01-05
## 6           38721 2006-01-06
```

1.6 Compile your data request to send to the server.

Your data order needs to include three things: Variable, Scenario, and Model. Your options can be found in the input elements we explored in the previous section.

```
input_variables <- inputs$variable_names %>%
  filter(Variable %in% c("Maximum Relative Humidity",
                         "Minimum Relative Humidity",
                         "Maximum Temperature",
                         "Minimum Temperature",
                         "Precipitation",
                         "Eastward Wind",
                         "Northward Wind")) %>%
  filter(Scenario %in% c( "RCP 8.5")) %>%
  filter(Model %in% c(
    "Beijing Climate Center - Climate System Model 1.1",
    "Beijing Normal University - Earth System Model",
    "Canadian Earth System Model 2",
     ↪
  "Centre National de Recherches Météorologiques - Climate Model 5",
   ↪
```

```
  "Commonwealth Scientific and Industrial Research Organisation - Mk3.6.0",
  ↪
  "Community Climate System Model 4",
  ↪
  "Geophysical Fluid Dynamics Laboratory - Earth System Model 2 Generalized
  ↪  Ocean Layer Dynamics",
  "Geophysical Fluid Dynamics Laboratory - Earth System Model 2 Modular
  ↪  Ocean",
  "Hadley Global Environment Model 2 - Climate Chemistry 365 (day) ",
  ↪
 "Hadley Global Environment Model 2 - Earth System 365 (day)",
 ↪
 "Institut Pierre Simon Laplace (IPSL) - Climate Model 5A - Low
 ↪  Resolution",
 "Institut Pierre Simon Laplace (IPSL) - Climate Model 5A - Medium
 ↪  Resolution",
 "Institut Pierre Simon Laplace (IPSL) - Climate Model 5B - Low
 ↪  Resolution",
 "Institute of Numerical Mathematics Climate Model 4",
 ↪
 "Meteorological Research Institute - Coupled Global Climate Model 3",
 ↪
 "Model for Interdisciplinary Research On Climate - Earth System Model",
 ↪
 "Model for Interdisciplinary Research On Climate - Earth System Model -
 ↪  Chemistry",
 "Model for Interdisciplinary Research On Climate 5",
 ↪
 "Norwegian Earth System Model 1 - Medium Resolution"  )) %>%

  pull("Available variable")

head(as.data.frame(input_variables))
```

```
##                 input_variables
## 1       pr_BNU-ESM_r1i1p1_rcp85
## 2         pr_CCSM4_r6i1p1_rcp85
## 3      pr_CNRM-CM5_r1i1p1_rcp85
## 4 pr_CSIRO-Mk3-6-0_r1i1p1_rcp85
## 5       pr_CanESM2_r1i1p1_rcp85
## 6    pr_GFDL-ESM2G_r1i1p1_rcp85
```

1.7 Prepare for parallelization

```
n_cores <- availableCores() - 1
plan(multisession, workers = n_cores)
```

1.8 Make parallel call to USGS server.

```
out <- single_point_firehose(input_variables, known_lat, known_lon )
head(out)
```

1.9 Save output

```
haskell <- out
save(haskell, file = "haskell.RData")
```

1.10 Load saved output

```
load(file = "haskell.RData")
```

```
df_precip <- haskell %>%
  st_drop_geometry() %>%
  select(time, which(startsWith(colnames(haskell), "pr"))) %>%
  gather(key = "variable", value = "value", -time)
head(df_precip)
```

```
##    time                variable value
## 1 38716 pr_BNU-ESM_r1i1p1_rcp85     0
## 2 38717 pr_BNU-ESM_r1i1p1_rcp85     0
## 3 38718 pr_BNU-ESM_r1i1p1_rcp85     0
## 4 38719 pr_BNU-ESM_r1i1p1_rcp85     0
## 5 38720 pr_BNU-ESM_r1i1p1_rcp85     0
## 6 38721 pr_BNU-ESM_r1i1p1_rcp85     0
```

```
df_min_temp <- haskell %>%
  st_drop_geometry() %>%
  select(time, which(startsWith(colnames(haskell), "tasmin"))) %>%
  gather(key = "variable", value = "value", -time)

df_min_temp <- df_min_temp[which(df_min_temp$value > 200), ]
df_min_temp$variable <-
↪  as.character(as.numeric(as.factor(df_min_temp$variable)))

head(df_min_temp)
```

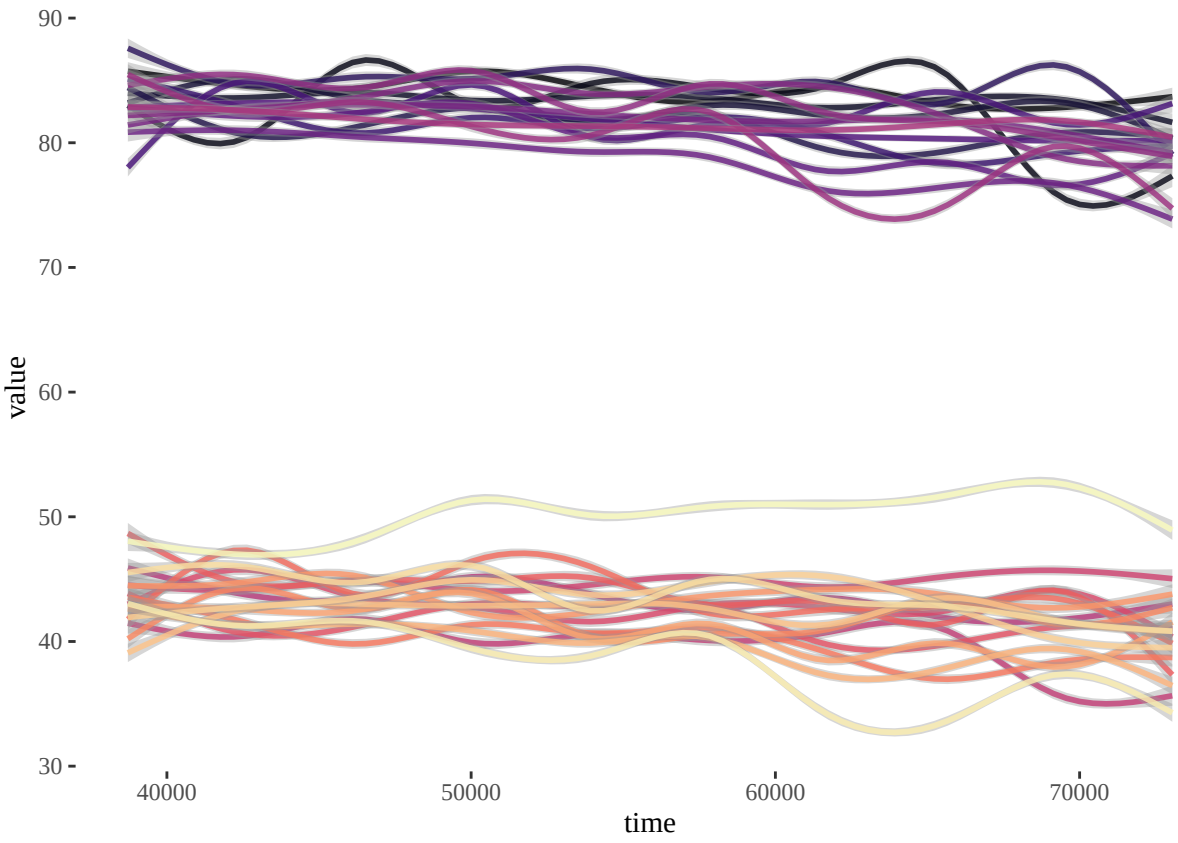
```
##    time variable value
## 1 38716        2 278.9
## 2 38717        2 277.7
## 3 38718        2 280.8
## 4 38719        2 276.5
## 5 38720        2 282.7
## 6 38721        2 284.6
```

```
df_RH <- haskell %>%
  st_drop_geometry() %>%
  select(time,which(startsWith(colnames(haskell), "rhs")) ) %>%
  gather(key = "variable", value = "value", -time)

df_RH_min <- haskell %>%
  st_drop_geometry() %>%
  select(time,which(startsWith(colnames(haskell), "rhsmin"))) %>%
  gather(key = "variable", value = "value", -time)

df_RH_max <- haskell %>%
  st_drop_geometry() %>%
  select(time,which(startsWith(colnames(haskell), "rhsmax"))) %>%
  gather(key = "variable", value = "value", -time)
head(df_RH_max)
```

```
##    time                    variable value
## 1 38716 rhsmax_BNU-ESM_r1i1p1_rcp85   100
## 2 38717 rhsmax_BNU-ESM_r1i1p1_rcp85   100
## 3 38718 rhsmax_BNU-ESM_r1i1p1_rcp85   100
```

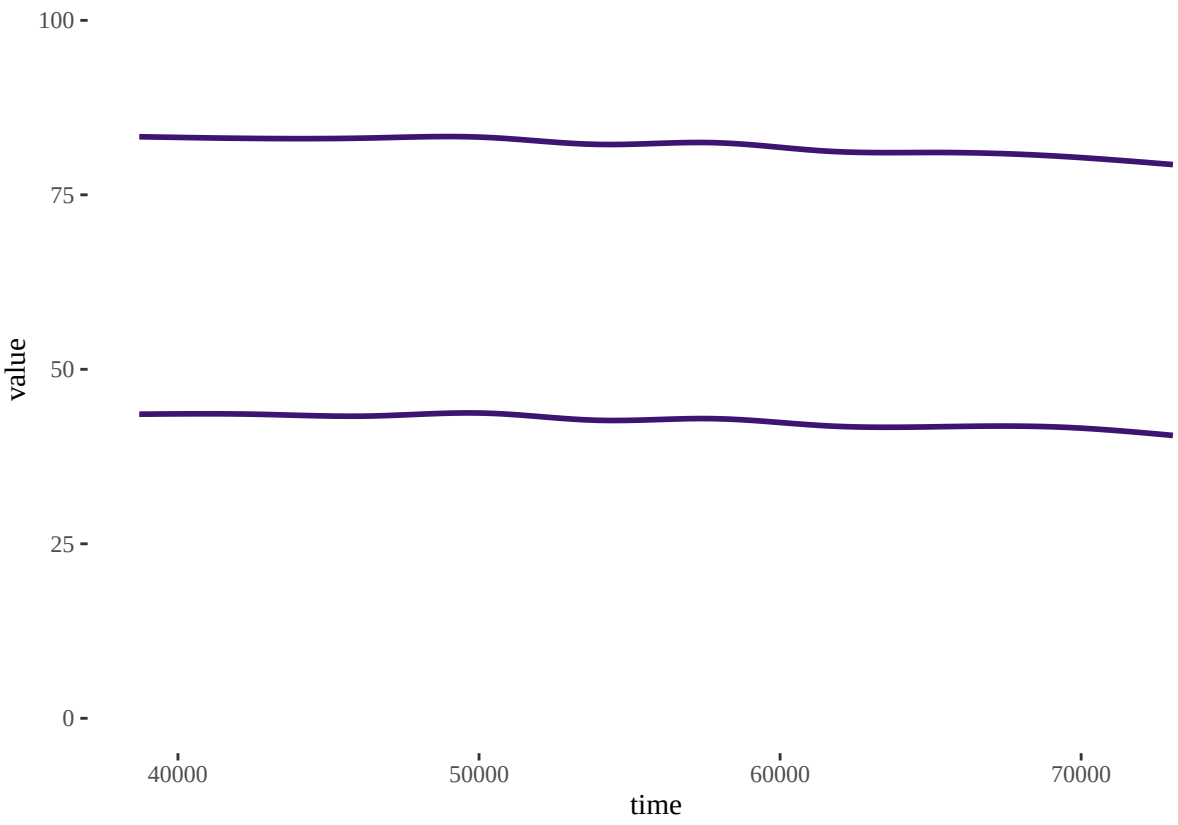
```
## 4 38719 rhsmax_BNU-ESM_r1i1p1_rcp85    95
## 5 38720 rhsmax_BNU-ESM_r1i1p1_rcp85   100
## 6 38721 rhsmax_BNU-ESM_r1i1p1_rcp85   100
```

```
all_climate_projections_RH <- ggplot(data= df_RH, aes(x = time, y = value,
↪  color = variable)) +
  #geom_line()+

  scale_colour_viridis_d(option="A", alpha = 0.8) +
  geom_smooth() +
  theme_tufte() +
  theme(legend.position = "none")


all_climate_projections_RH
```

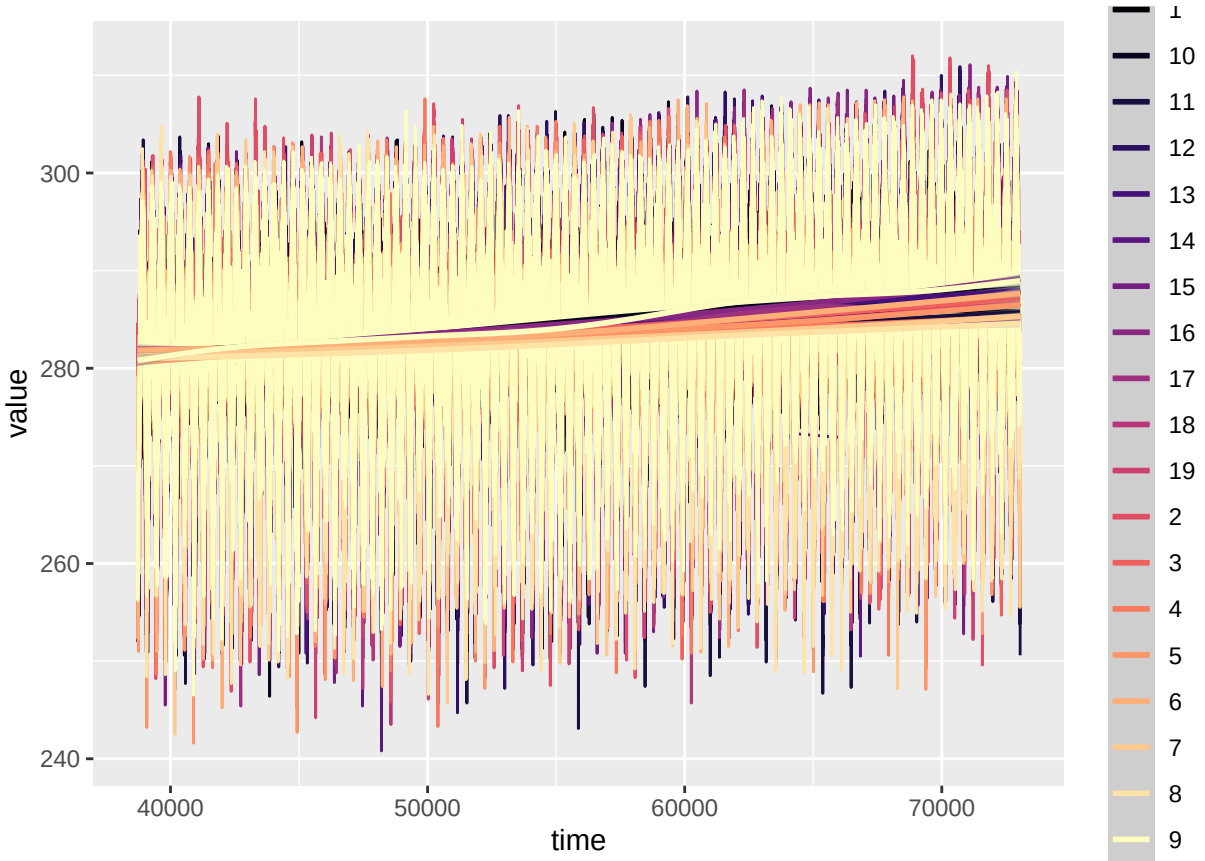



```
all_climate_projections_RH <- ggplot(data= df_RH_min, aes(x = time, y =
↪  value) )+
  geom_smooth(data= df_RH_min, color=our_purple) +
```

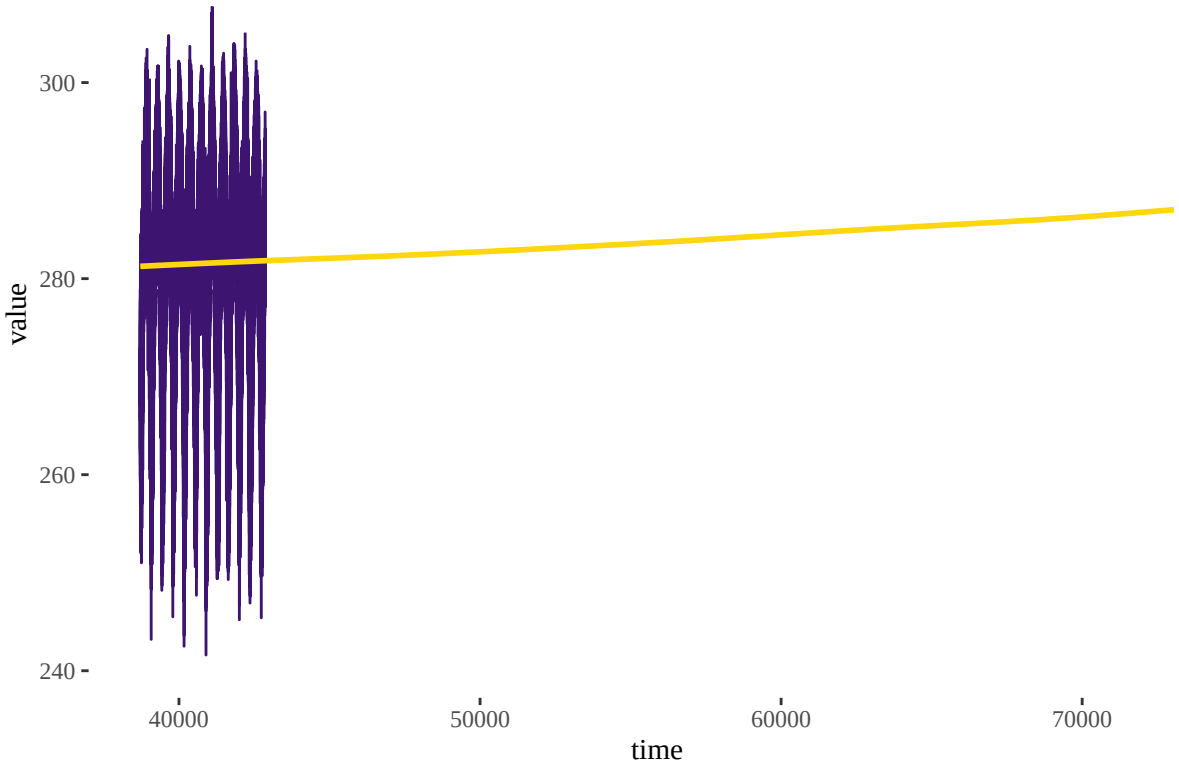
```
  geom_smooth(data= df_RH_max, color=our_purple) +
  theme_tufte() +
  ylim(0,100)


all_climate_projections_RH
```



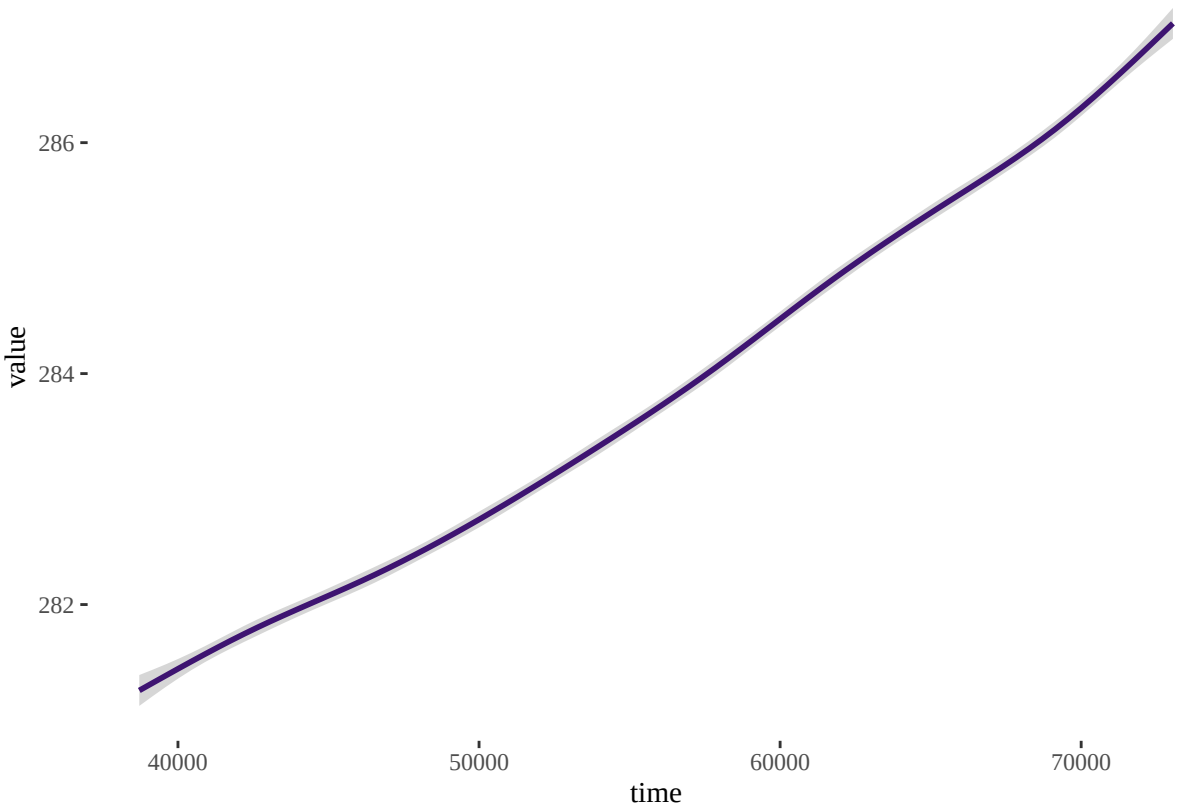

```
all_climate_projections <- ggplot(data= df_min_temp, aes(x = time, y =
↪  value, color = variable)) +
  geom_line()+
  geom_smooth() +
  scale_colour_viridis_d(option="A")


all_climate_projections
```



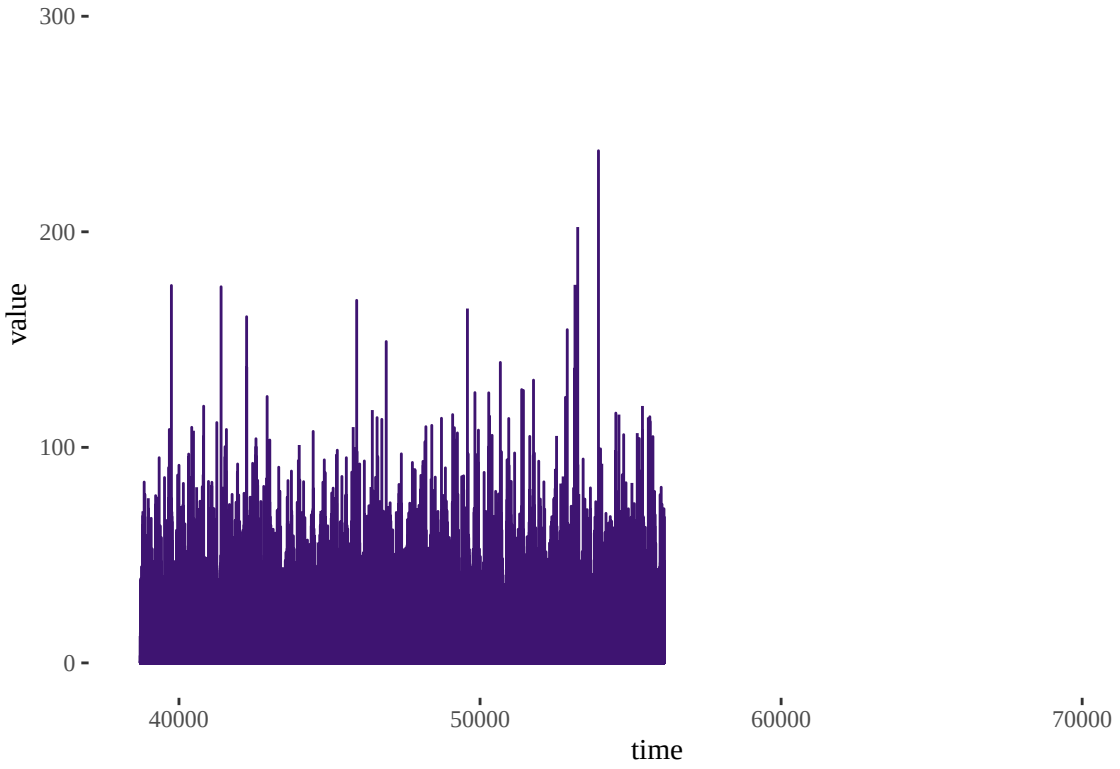

```
ensemble_climate_projections <- ggplot(data= df_min_temp, aes(x = time, y =
↪  value)) +
  geom_line(color=our_purple)+
  geom_smooth(color=our_yellow) +
  theme_tufte()


ensemble_climate_projections
```



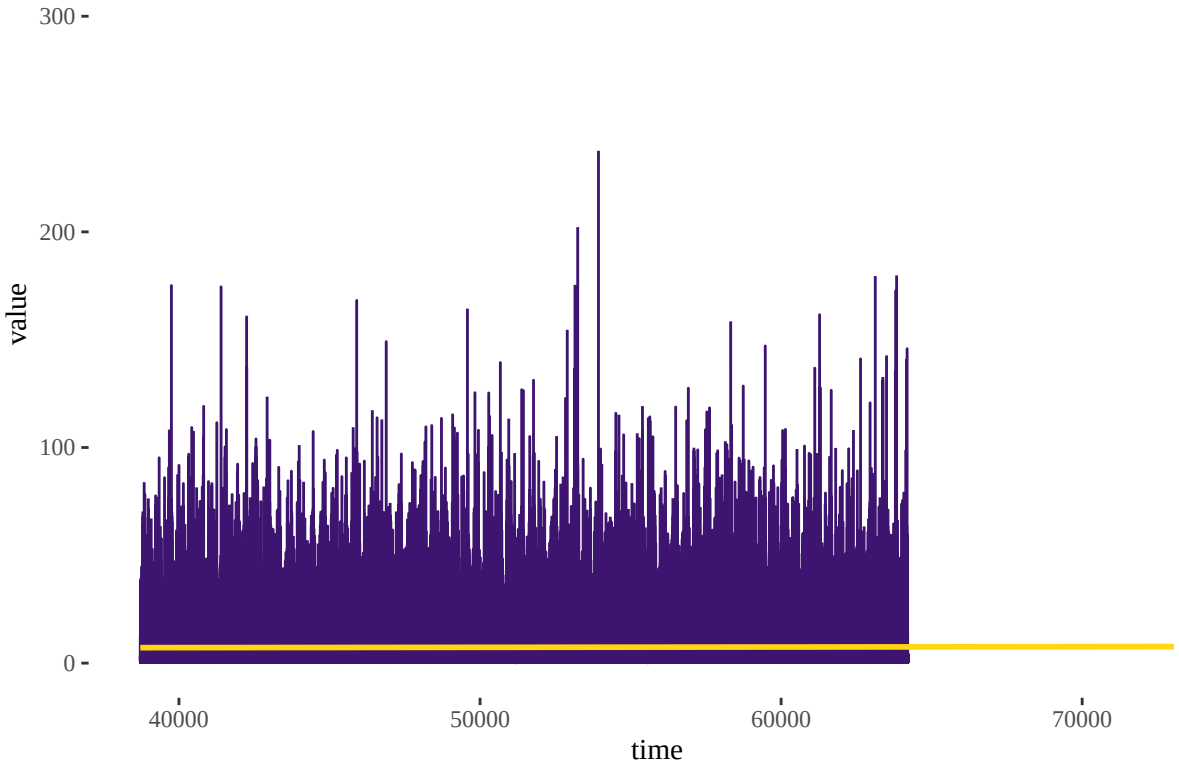

```
ggplot(data= df_min_temp, aes(x = time, y = value)) +
  geom_smooth(color=our_purple) +
  theme_tufte()
```



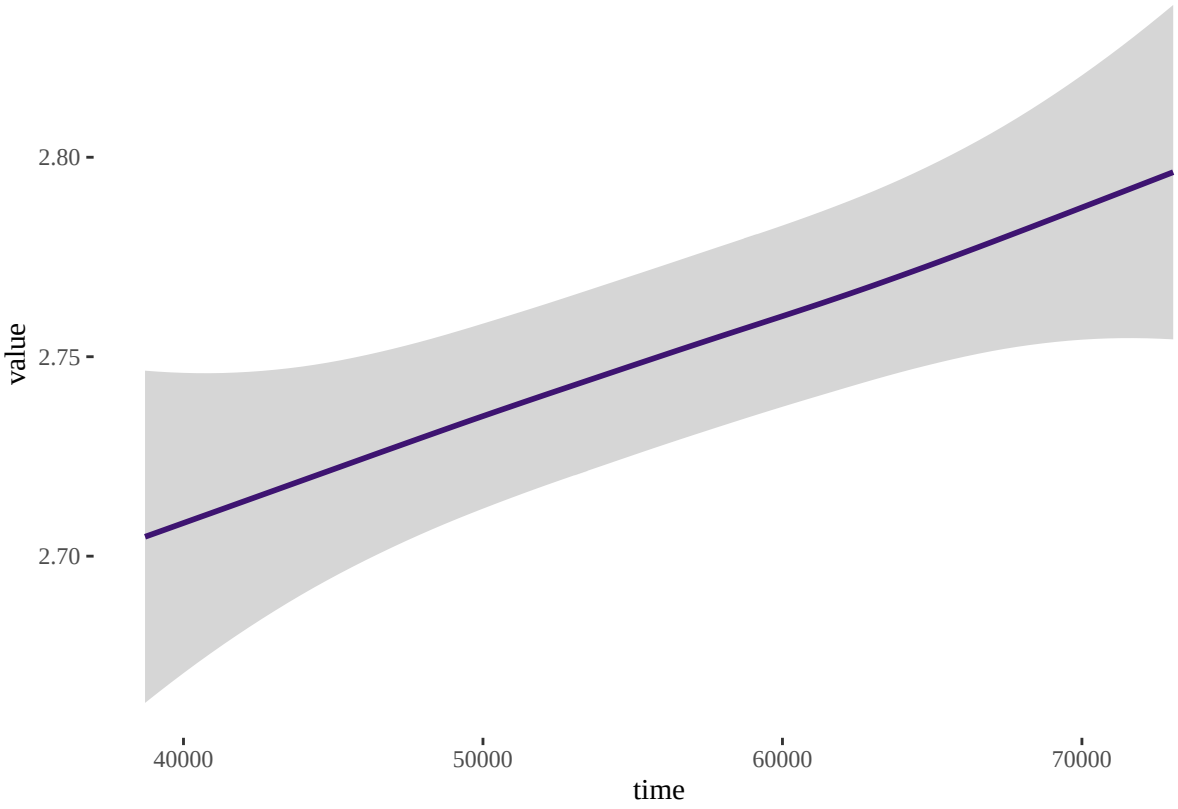

```
all_climate_projections <- ggplot(data= df_precip, aes(x = time, y = value,
↪  color = variable)) +
  geom_line(color=our_purple)+
  theme_tufte()


all_climate_projections
```




```
df_precip_2 <- df_precip[which(df_precip$value != 0),]

ensemble_climate_projections <- ggplot(data= df_precip_2, aes(x = time, y =
↪  value)) +
   geom_line(color=our_purple)+
  geom_smooth(color=our_yellow)+
  theme_tufte()


ensemble_climate_projections
```




```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(color=our_purple)+
  theme_tufte()
```

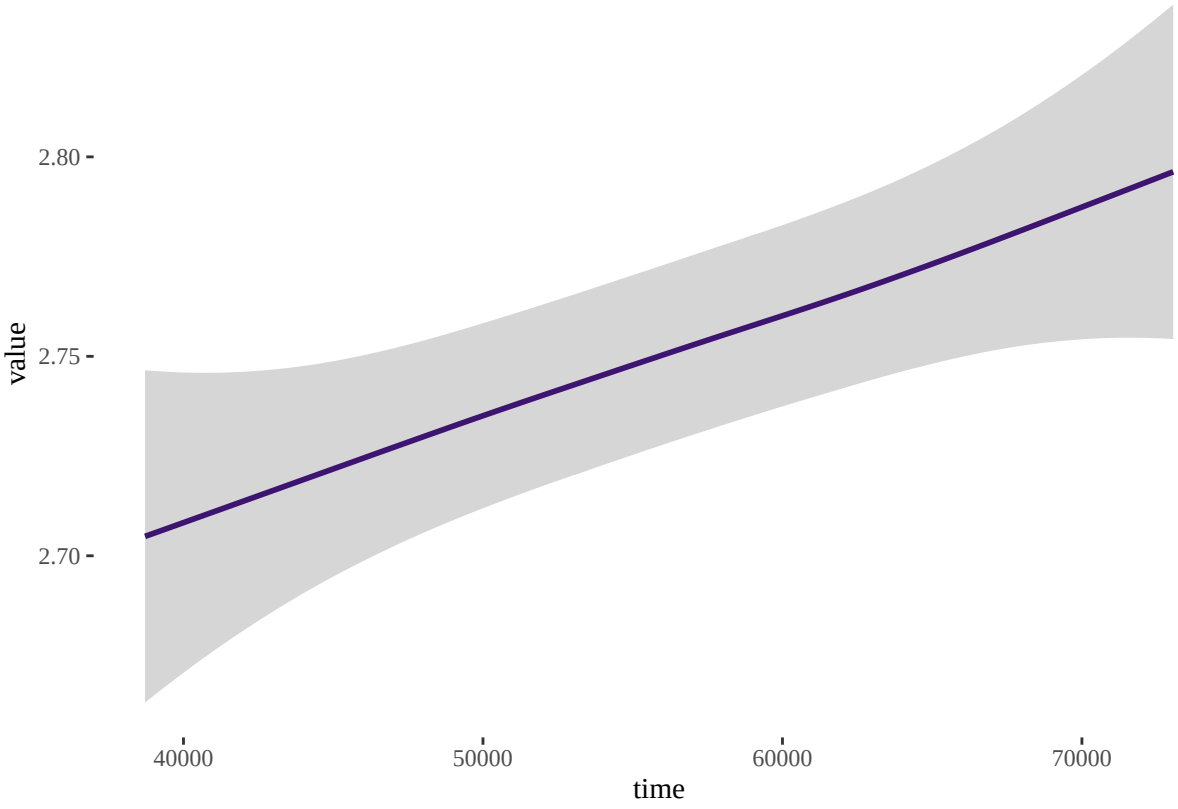



```
df_precip_quant_95 <- df_precip %>%
  mutate(quant = ntile(value, 100)) %>%
  filter(quant == 95)

df_precip_quant_96 <- df_precip %>%
  mutate(quant = ntile(value, 100)) %>%
  filter(quant == 96)

df_precip_quant_97 <- df_precip %>%
  mutate(quant = ntile(value, 100)) %>%
  filter(quant == 97)

df_precip_quant_98 <- df_precip %>%
  mutate(quant = ntile(value, 100)) %>%
  filter(quant == 98)

df_precip_quant_99 <- df_precip %>%
  mutate(quant = ntile(value, 100)) %>%
  filter(quant == 99)
```

```
df_precip_quant_100 <- df_precip %>%
  mutate(quant = ntile(value, 100)) %>%
  filter(quant == 100)

df_precip_quant_1000 <- df_precip %>%
  mutate(quant = ntile(value, 1000)) %>%
  filter(quant == 1000)

df_precip_quant_100000 <- df_precip %>%
  mutate(quant = ntile(value, 100000)) %>%
  filter(quant == 100000)
```

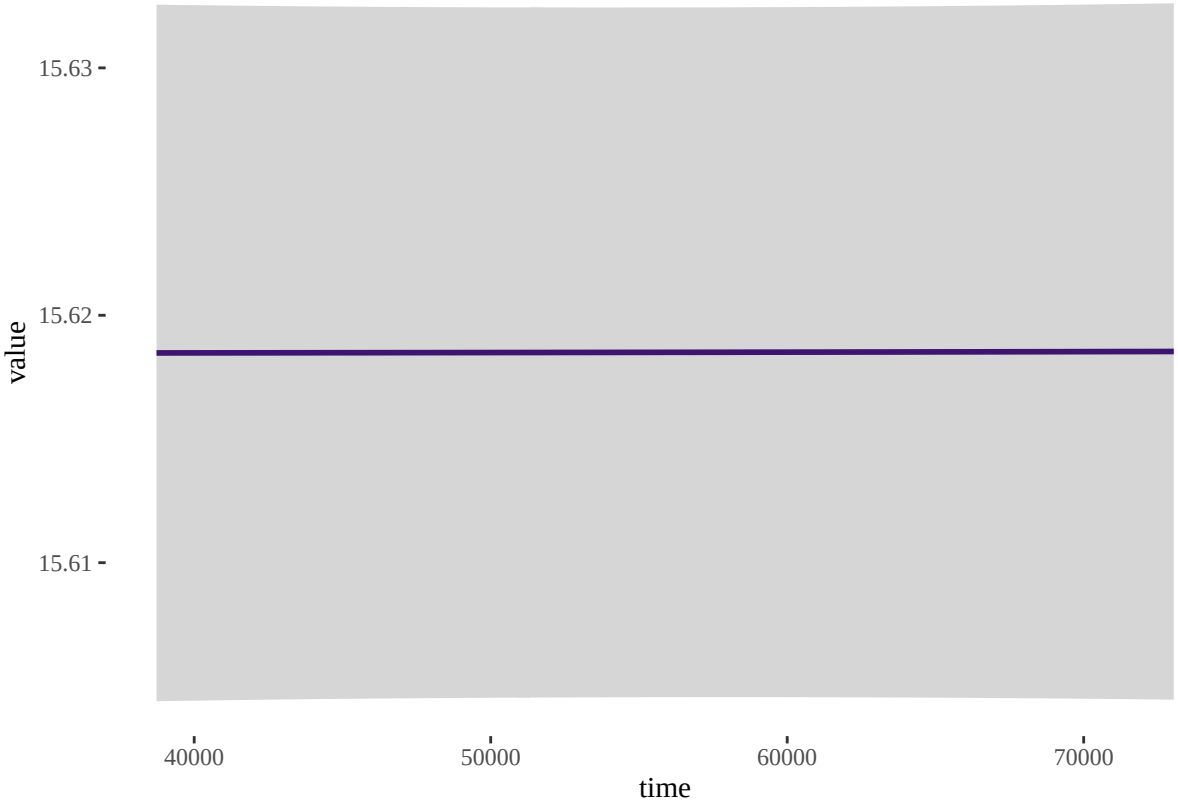
```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



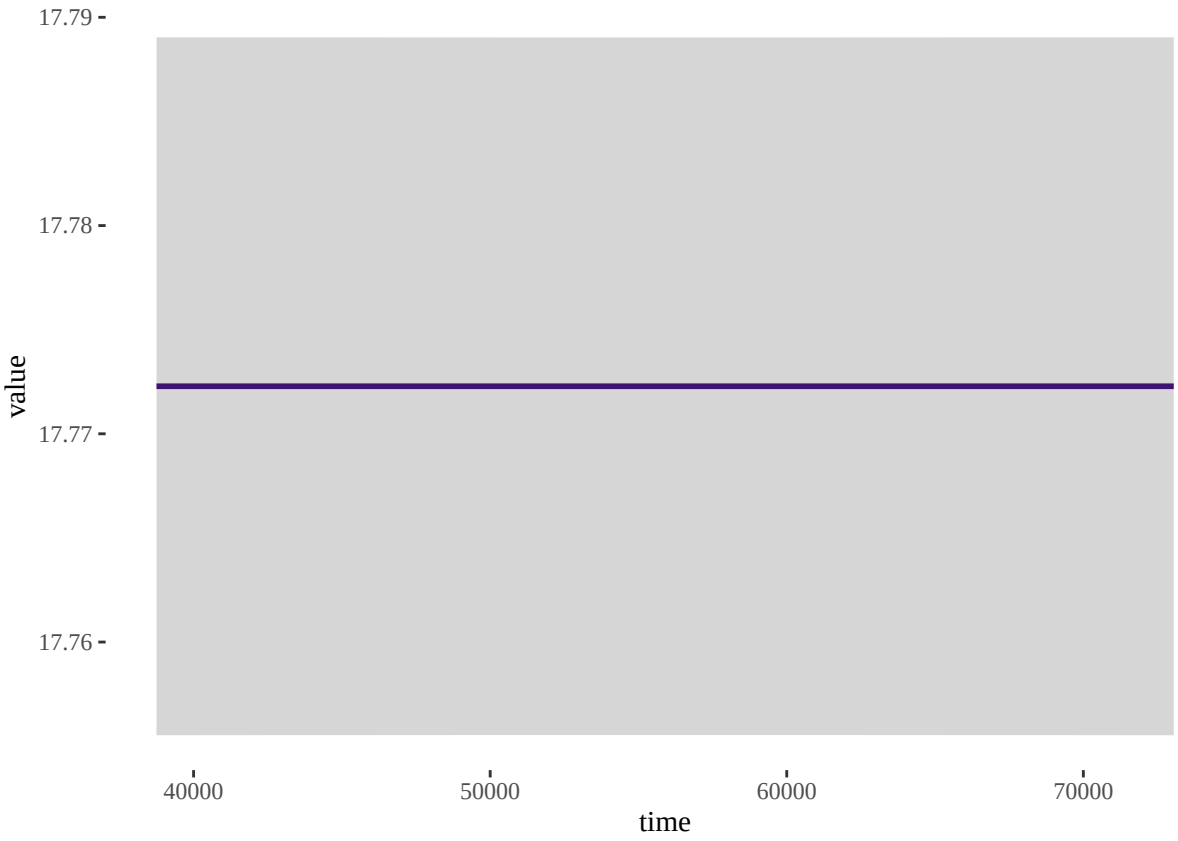
```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_95, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



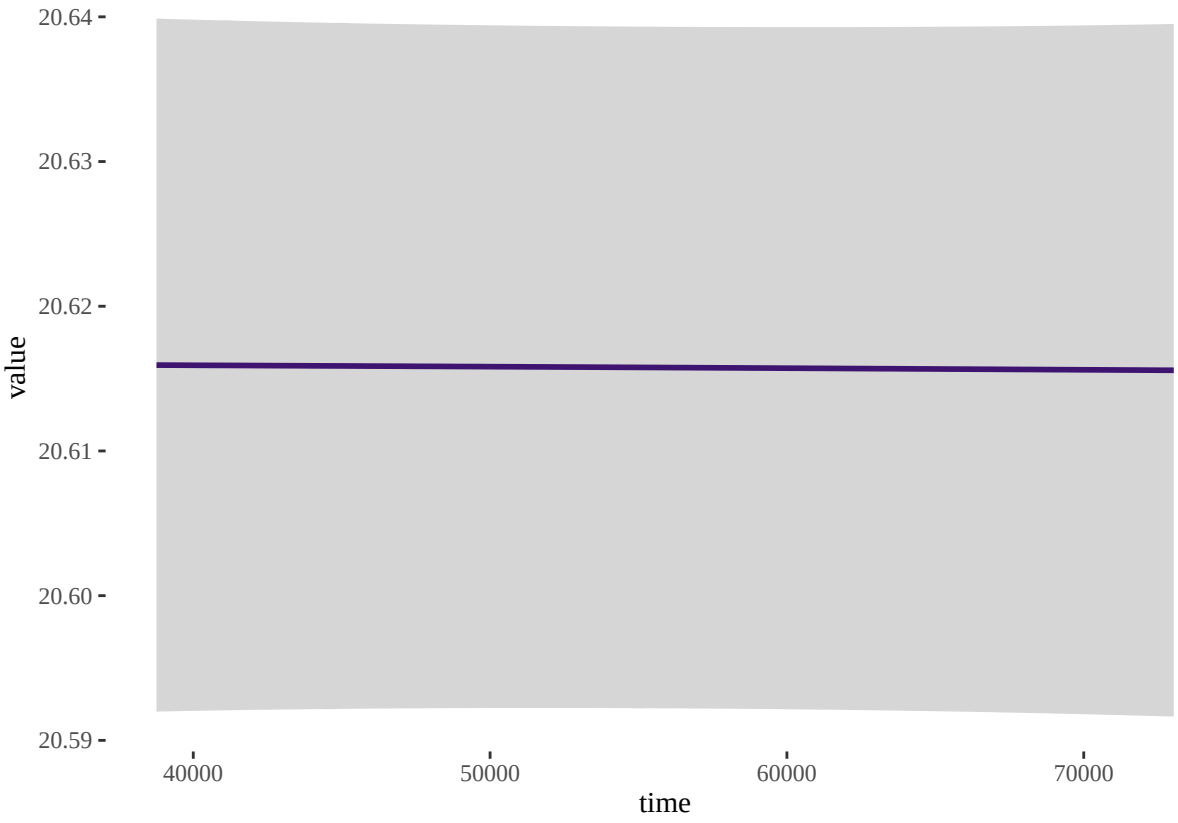

```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_96, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



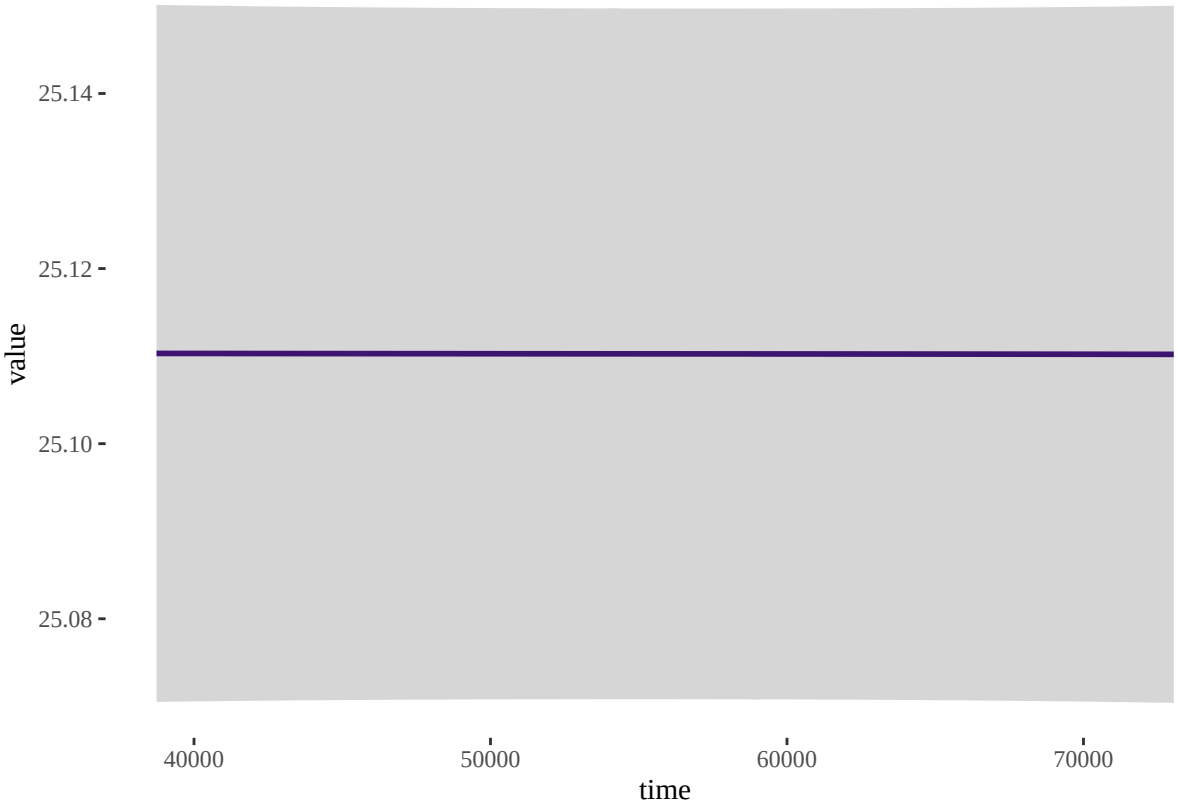

```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_97, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



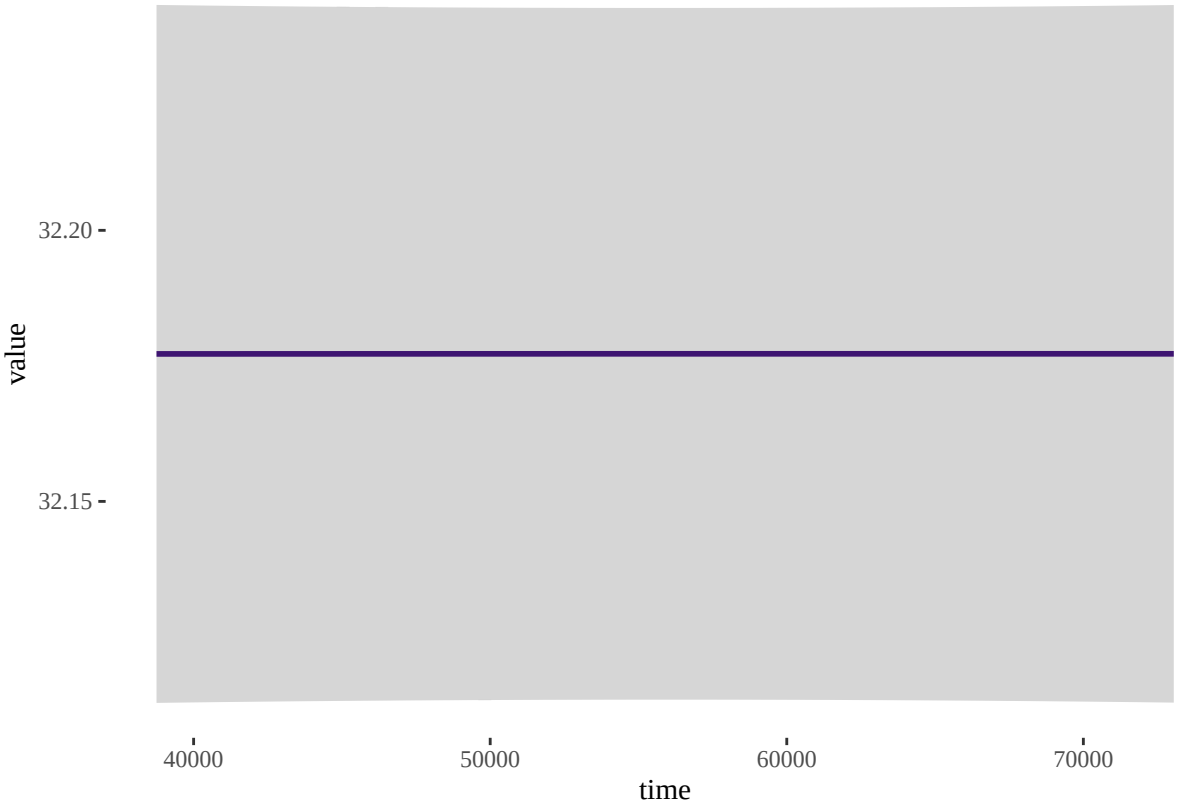

```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_98, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



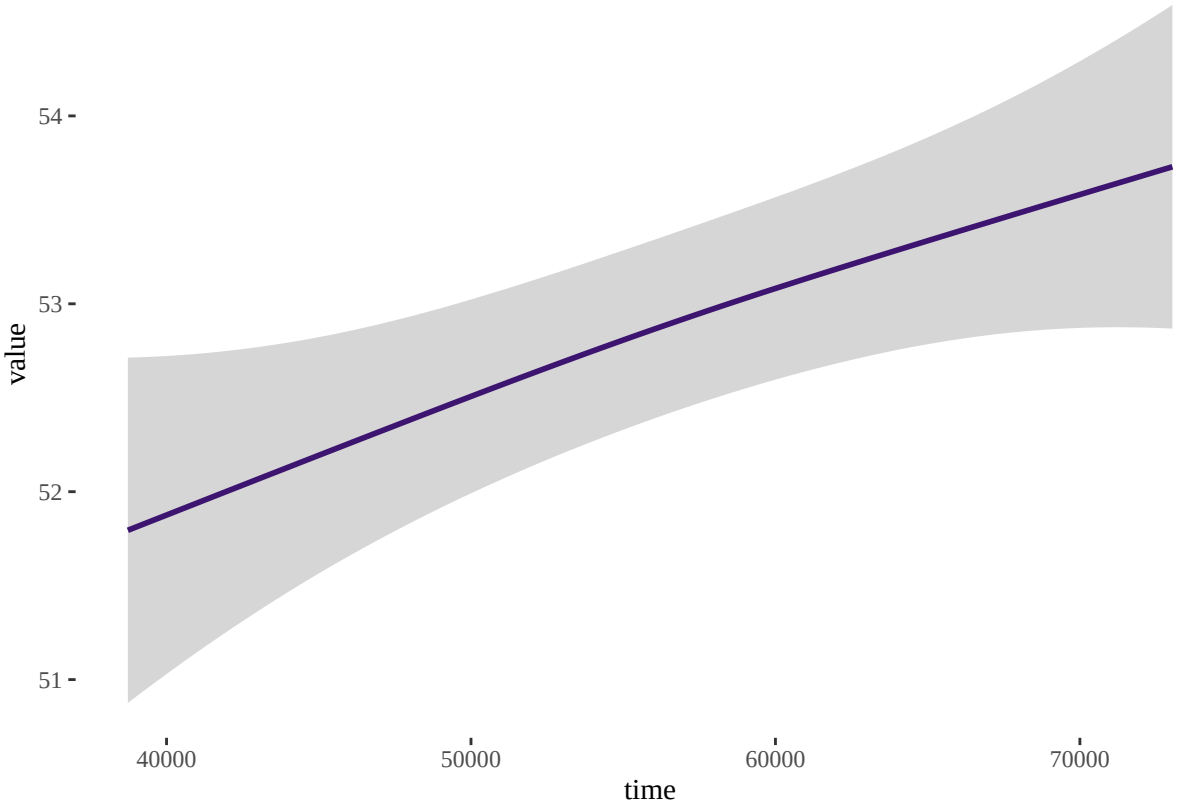

```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_99, color=our_purple) +
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



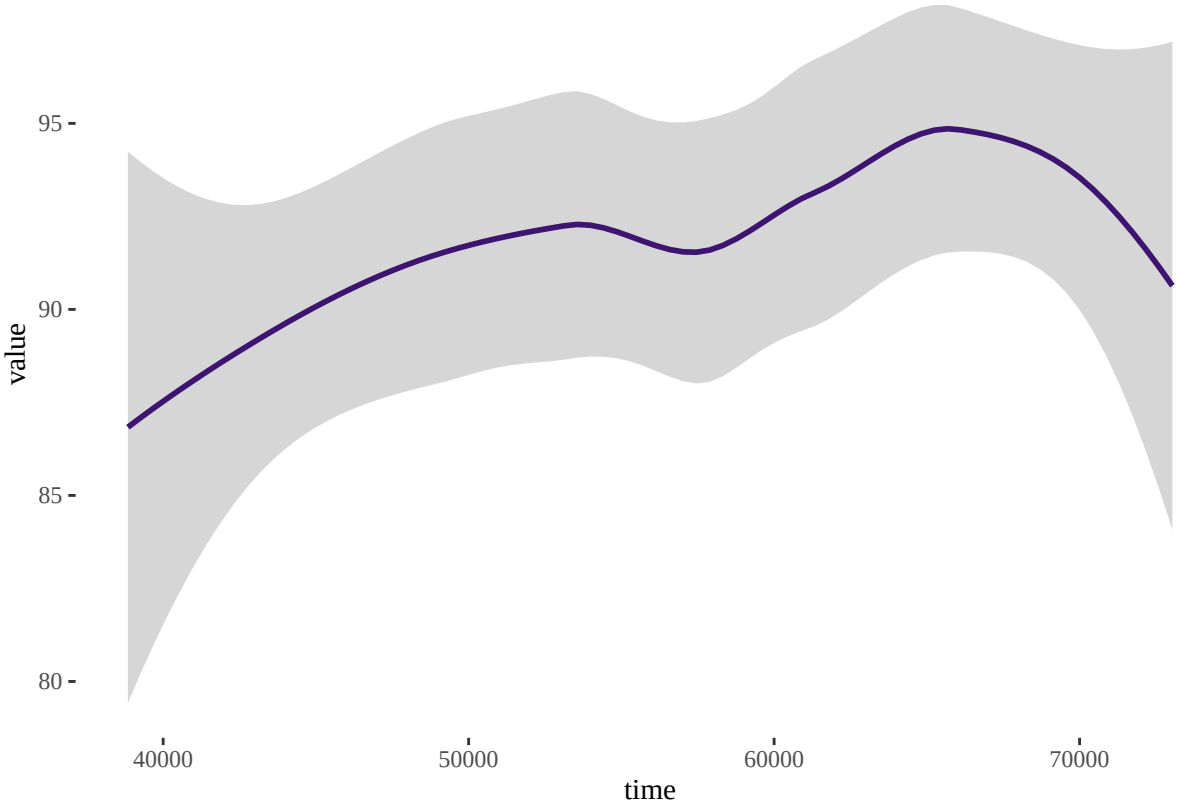

```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_100, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



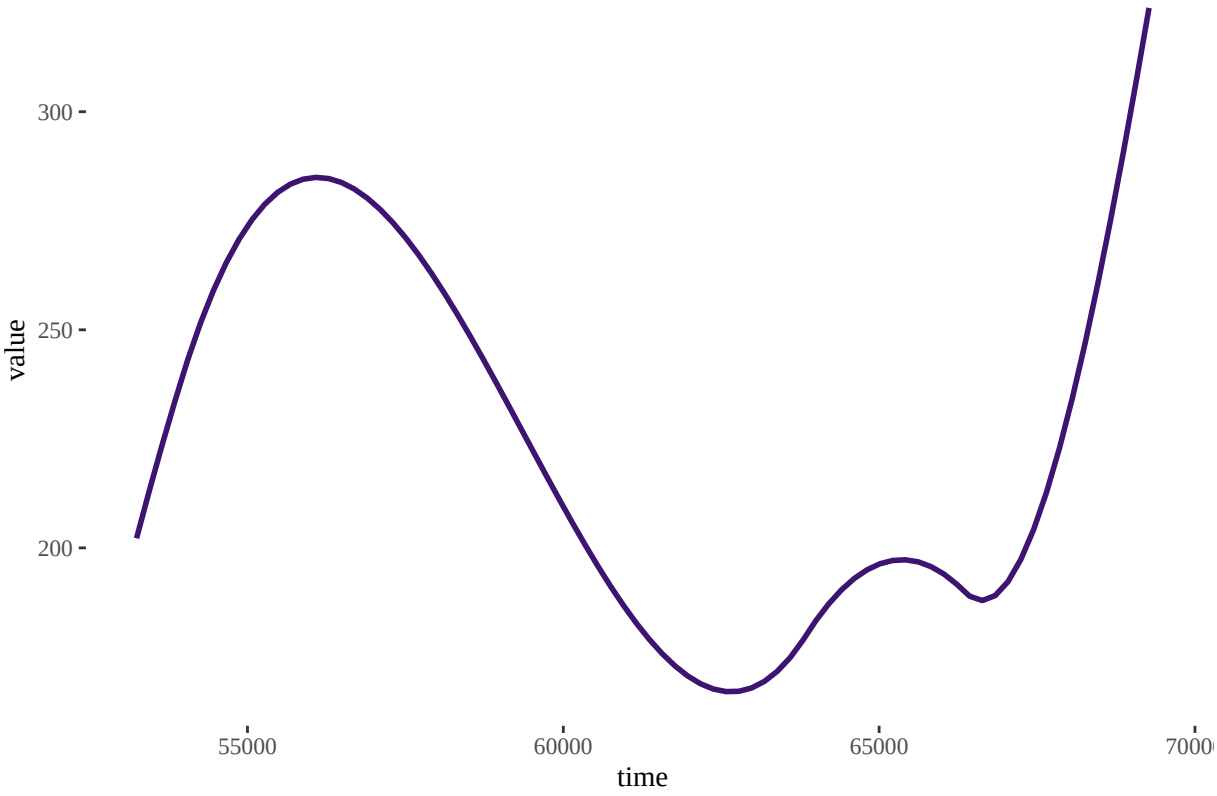

```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_1000, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

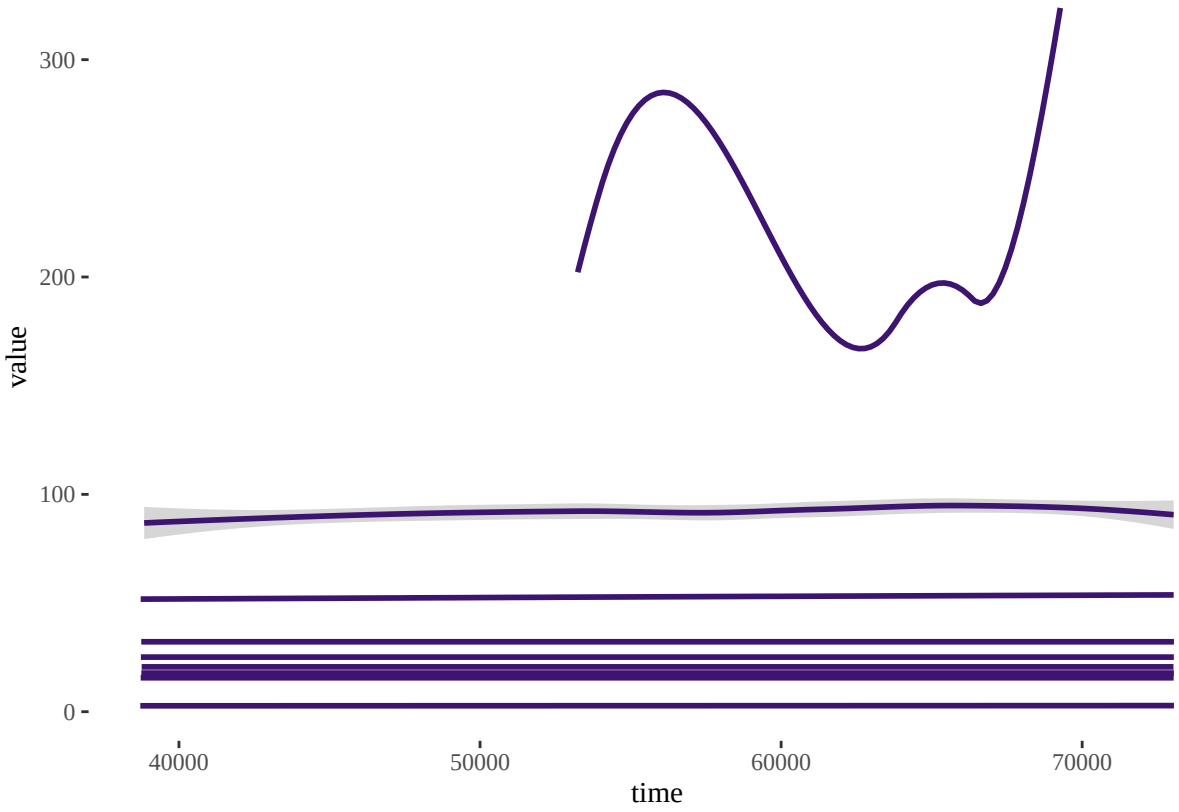



```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip_quant_100000, color=our_purple)+
  theme_tufte()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(data= df_precip, aes(x = time, y = value)) +
     geom_smooth(data= df_precip, color=our_purple)+
     geom_smooth(data= df_precip_quant_95, color=our_purple) +
    geom_smooth(data= df_precip_quant_96, color=our_purple) +
    geom_smooth(data= df_precip_quant_97, color=our_purple) +
    geom_smooth(data= df_precip_quant_98, color=our_purple) +
    geom_smooth(data= df_precip_quant_99, color=our_purple) +
  geom_smooth(data= df_precip_quant_100, color=our_purple) +
  geom_smooth(data= df_precip_quant_1000, color=our_purple) +
    geom_smooth(data= df_precip_quant_100000, color=our_purple) +
  theme_tufte()
```


300
200
100
0
value
40000
50000
60000
70000
time