Final Project Tutorial

Geog 418

December 01, 2024

Introduction

British Columbia is home to a diverse range of crops and livestock, many of which are highly sensitive to the ever-changing climate. One such example is honeybees, whose productivity is closely tied not only to temperature fluctuations but also to the plants they rely on to produce honey. As a result, beekeepers in BC are faced with a double challenge: the health of their bees is impacted by temperature variations, and the plants they depend on for nectar are also affected by these environmental changes. Given the significant role that honey production plays in the province's economy, understanding how these climate shifts influence beekeeping is essential for both farmers and policymakers (Wheeler & von Braun, 2013.

In this tutorial, we will explore how temperature data can be used to assess the effects on honey production across different regions of BC. By integrating temperature data with honey production statistics, we can analyze how the conditions of the climate effect farm productivity. We will visualize these effects using spatial interpolation techniques such as inverse distance weighting (IDW) and geographically weighted regression (GWR), giving you better insight into the relationship between temperature and honey production (Li & Heap, 2014; Fotheringham, Brunsdon, & Charlton, 2002).

The objective of this tutorial is to guide you through the process of cleaning raw climate data, merging it with agricultural data, and conducting spatial analysis to better understand the correlation between temperature and honey production. By the end of the tutorial, you will have acquired the skills necessary to perform similar analyses on other datasets, empowering you to explore the impact of environmental factors on various agricultural outcomes.

This tutorial will walk you through the following steps:

1. **Data Collection**: Downloading raw climate data (temperature) and honey production statistics.
2. **Data Cleaning and Preparation**: Transforming the raw data into a usable format for spatial analysis.
3. **Spatial Analysis**: Applying spatial interpolation techniques, like IDW, to model temperature distribution and its relationship with honey production.
4. **Analysis of Results**: Using geographically weighted regression to examine how temperature variations influence honey production across regions.

By following this step-by-step approach, you will gain valuable experience in data cleaning, spatial analysis, and interpretation, helping you understand how temperature and other environmental factors can influence agricultural productivity in British Columbia and beyond.

```
# install.packages("dplyr")
# install.packages("gstat")
# install.packages("ggplot2")
# install.packages("spgwr")
# install.packages("knitr")
```

```
# install.packages("rgdal")
# install.packages("tmap")
# install.packages("spdep")
# install.packages("raster")
# install.packages("shinyjs")
# install.packages("e1071")
# install.packages("st")
# install.packages("sf")
# install.packages("lubridate")
# install.packages("tidyverse")
```

```
# Load in libraries:
library("dplyr")
library("gstat")
library("ggplot2")
library("spgwr")
library("st")
library("sf")
library("e1071")
library("knitr")
library("shiny")
library("tmap")
library("spdep")
library("sp")
library("raster")
library("lubridate")
library(tidyverse)
```

Prepare RStudio

To begin, we need to set up RStudio for working with our data. Start by copying and pasting the code provided above into your own RStudio project. Before running the code, make sure that your versions of R, RTools, and RStudio are compatible with each other. If they aren't, you may encounter pathing errors. To fix this, the easiest way is to just update RStudio, R, and RTools to their newest versions (Weaver et al., 2016). You can find the newest versions at the following links:

- RStudio Desktop
- RTools

Updating to the latest versions will ensure smooth operation and help you avoid common setup issues.

Collect your data

Next, we need to download our data. You have two options for doing this:

Use the files provided with this tutorial: Simply download the "weather_station_files" folder and place it into your project folder. Also, make sure to download the "stations.csv" file and place it in the same project folder, but keep it as is. After that, you can run the following code.

Download the data from the Pacific Climate Impacts Consortium:

First, go to the Pacific Climate Impacts Consortium website.

Choose your desired time period by selecting a start and end date. For this tutorial, we'll focus on the summer of 2016 first, which was an unusually hot summer and then we will do this a second time for 2015.

Next, select the data attributes you need. For this analysis, we're only interested in air temperature data, both hourly and daily observations. Feel free to include additional attributes for your own analysis, but be aware that larger datasets may take hours or even days to download.

Before downloading, double-check the Station Metadata tab to ensure everything looks correct. It's much easier to avoid issues now than to go through the process again later. Trust me. If everything looks satisfactory, click download metadata, we will be using that in the next step.

Finally, go to the Station Data tab, select "CSV" as your output format, and check the box for "Clip series to filter date range." Click "Download Timeseries" and grab a coffee (or lunch).

After the download finishes, you'll receive a zip folder that you'll need to extract. The code in this tutorial assumes that these files will be stored in a folder called "weather_station_files." You're welcome to rename the folder if you prefer, but be sure to update the folder name in the code or it won't work.

Do this again but instead for the summer of 2015

Because of the time it can takes to download these files, it's recommended you use the first option to save time.

Once your data is ready, run the following code to merge the files.

```
# First, list all CSV files in the directory to make sure you are in the right folder.
csv_files <- list.files(path = "./weather_station_files_2015", pattern =
                          "\\.csv$", full.names = TRUE)

# We are giving the file this name as an example of calculating average temperatures in BC.
csv_file_name <- "BC_AVG_TEMP_2015.csv"

# Create an empty data frame
empty_data <- data.frame(Native.ID = character(0), TEMP = numeric(0),
                         Longitude = numeric(0), Latitude = numeric(0))

# Write the empty data frame to a CSV file
write.csv(empty_data, file = csv_file_name, row.names = FALSE)

# Next, loop through each CSV file and perform your calculations
for (file in csv_files) {

  # Read each file
  hourly_data <- read.csv(file, skip = 1, header = TRUE)

  # Adjust the date/time column so that it is usable in calculations
  # Remove any leading/trailing spaces from 'time' column
  hourly_data$time <- trimws(hourly_data$time)
  # Convert to POSIXct
  hourly_data$time <- as.POSIXct(hourly_data$time, format="%Y-%m-%d %H:%M:%S")

  # Clean and convert the 'air_temp_1' column to numeric
  # Remove non-numeric characters
  hourly_data$air_temp_1 <- as.numeric(gsub("[^0-9.-]", "", hourly_data$air_temp_1))


  # Remove rows with NA temperature values
  hourly_data <- hourly_data %>% filter(!is.na(air_temp_1))
```

```r
  # Example: Calculate daily average temperature from all the data
  daily_avg_temp <- hourly_data %>%
    group_by(date = as.Date(time)) %>%
    # Using 'air_temp_1'
    summarize(daily_avg_temp = mean(air_temp_1, na.rm = TRUE))

  # Example: Calculate the average temperature from May to October
  average_temp_may_october <- hourly_data %>%
    filter(month(time) >= 5 & month(time) <= 10) %>%
    summarize(TEMP = mean(air_temp_1, na.rm = TRUE))  # Using 'air_temp_1'


  # Extract the filename as this is the name of your weather station
  file_name <- basename(file)  # Extract file name from path
  # Remove the file extension
  file_name_no_ext <- sub("\\.[^.]*$", "", file_name)

  # Read the existing CSV file where you store results
  data <- read.csv(csv_file_name)

  # Round the temperature values to two decimals
  rounded_temp <- round(average_temp_may_october$TEMP, 2)

  # Convert the weather station ID column to character
  data$Native.ID <- as.character(data$Native.ID)

  # Add your weather station and temperature values to the file
  new_values <- data.frame(Native.ID = file_name_no_ext,
                           TEMP = rounded_temp,
                           Longitude = NA,  # Add Longitude data if available
                           Latitude = NA,   # Add Latitude data if available
                           stringsAsFactors = FALSE)

  # Add the new row to the existing data
  data <- bind_rows(data, new_values)

  # Save the updated data frame back to a new CSV file
  write.csv(data, file = csv_file_name, row.names = FALSE)
}
```

```r
# First, list all CSV files in the directory to make sure you are in the right folder.
csv_files <- list.files(path = "./weather_station_files_2016", pattern =
                          "\\.csv$", full.names = TRUE)
# We are giving the file this name as an example of calculating average temperatures in BC
csv_file_name <- "BC_AVG_TEMP_2016.csv"

# Create an empty data frame
empty_data <- data.frame(Native.ID = character(0), TEMP = numeric(0),
                         Longitude = numeric(0), Latitude = numeric(0))

# Write the empty data frame to a CSV file
write.csv(empty_data, file = csv_file_name, row.names = FALSE)
```

```r
# Next, loop through each CSV file and perform your calculations
for (file in csv_files) {

  # Read each file
  hourly_data <- read.csv(file, skip = 1, header = TRUE)

  # Adjust the date/time column so that it is usable in calculations
  # Remove any leading/trailing spaces from 'time' column
  hourly_data$time <- trimws(hourly_data$time)
  # Convert to POSIXct
  hourly_data$time <- as.POSIXct(hourly_data$time, format="%Y-%m-%d %H:%M:%S")

  # Clean and convert the 'air_temp_1' column to numeric
  # Remove non-numeric characters
  hourly_data$air_temp_1 <- as.numeric(gsub("[^0-9.-]", "", hourly_data$air_temp_1))

  # Filter out NA values for temperature
  # Remove rows with NA temperature values
  hourly_data <- hourly_data %>% filter(!is.na(air_temp_1))

  # Example: Calculate daily average temperature from all the data
  daily_avg_temp <- hourly_data %>%
    group_by(date = as.Date(time)) %>%
    summarize(daily_avg_temp = mean(air_temp_1, na.rm = TRUE))  # Using 'air_temp_1'

  # Example: Calculate the average temperature from May to October
  average_temp_may_october <- hourly_data %>%
    filter(month(time) >= 5 & month(time) <= 10) %>%
    summarize(TEMP = mean(air_temp_1, na.rm = TRUE))  # Using 'air_temp_1'


  # Extract the filename as this is the name of your weather station
  file_name <- basename(file)  # Extract file name from path
  file_name_no_ext <- sub("\\.[^.]*$", "", file_name)  # Remove the file extension

  # Read the existing CSV file where you store results
  data <- read.csv(csv_file_name)

  # Round the temperature values to two decimals
  rounded_temp <- round(average_temp_may_october$TEMP, 2)

  # Convert the weather station ID column to character
  data$Native.ID <- as.character(data$Native.ID)

  # Add your weather station and temperature values to the file
  new_values <- data.frame(Native.ID = file_name_no_ext,
                           TEMP = rounded_temp,
                           Longitude = NA,  # Add Longitude data if available
                           Latitude = NA,   # Add Latitude data if available
                           stringsAsFactors = FALSE)

  # Add the new row to the existing data
  data <- bind_rows(data, new_values)
```

```
  # Save the updated data frame back to a new CSV file
  write.csv(data, file = csv_file_name, row.names = FALSE)
}
```

Merge location with data

At this point, you should have two new CSV files in your project folder named `BC_AVG_TEMP_2015.csv` and 'BC_AVG_TEMP_2016.csv' . These files contain the summarized average temperature data for each weather station we downloaded in the previous steps.

However, you may have noticed that these new CSV files lack any information about the **location** of these stations. Without spatial data, how can we perform spatial analysis? This next step will solve that problem by turning our data into spatial data.

Recall the metadata file (`stations.csv`) that we downloaded earlier. This file contains the crucial information we need: the latitude and longitude of each weather station. By joining this location data with our temperature data, we can create a spatial dataset. Run the following code to merge the location information with your temperature data and turn it into spatial data.

```
# Load the datasets
metadata <- read.csv("Stations_2015.csv")
climatedata <- read.csv("BC_AVG_TEMP_2015.csv")

# Trim whitespace from Native.ID and ensure they are of the same type (character)
metadata$Native.ID <- trimws(as.character(metadata$Native.ID))
climatedata$Native.ID <- trimws(as.character(climatedata$Native.ID))

# Perform the merge
merged_data <- merge(metadata, climatedata, by = "Native.ID")

# Remove the last two columns which are duplicate Latitude and Longitude
merged_data <- merged_data[, -((ncol(merged_data)-1):ncol(merged_data))]

# Change column names for Latitude and Longitude to remove the 'x'
colnames(merged_data)[colnames(merged_data) %in%
                        c("Latitude.x", "Longitude.x")] <- c("Longitude", "Latitude")

# Omit rows with NA values in critical columns (Latitude, Longitude, or TEMP)
merged_data <- merged_data[!is.na(merged_data$Latitude) & !is.na(merged_data$Longitude)
                           & !is.na(merged_data$TEMP), ]

# Filter out erroneous temperature values greater than 100
merged_data <- merged_data[merged_data$TEMP <= 100, ]

# Write the cleaned and merged dataset to a new CSV file
write.csv(merged_data, file = "ClimateData_2015.csv", row.names = FALSE)
```

```
# Load the datasets
metadata <- read.csv("Stations_2016.csv")
climatedata <- read.csv("BC_AVG_TEMP_2016.csv")

# Trim whitespace from Native.ID and ensure they are of the same type (character)
metadata$Native.ID <- trimws(as.character(metadata$Native.ID))
```

```r
climatedata$Native.ID <- trimws(as.character(climatedata$Native.ID))

# Perform the merge
merged_data <- merge(metadata, climatedata, by = "Native.ID")

# Remove the last two columns which are duplicate Latitude and Longitude
merged_data <- merged_data[, -((ncol(merged_data)-1):ncol(merged_data))]

# Change column names for Latitude and Longitude to remove the 'x'
colnames(merged_data)[colnames(merged_data) %in%
                        c("Latitude.x", "Longitude.x")] <- c("Longitude", "Latitude")

# Omit rows with NA values in critical columns (Latitude, Longitude, or TEMP)
merged_data <- merged_data[!is.na(merged_data$Latitude) & !is.na(merged_data$Longitude)
                           & !is.na(merged_data$TEMP), ]

# Filter out erroneous temperature values greater than 100
merged_data <- merged_data[merged_data$TEMP <= 100, ]

# Write the cleaned and merged dataset to a new CSV file
write.csv(merged_data, file = "ClimateData_2016.csv", row.names = FALSE)
```

Looking at Climate Data

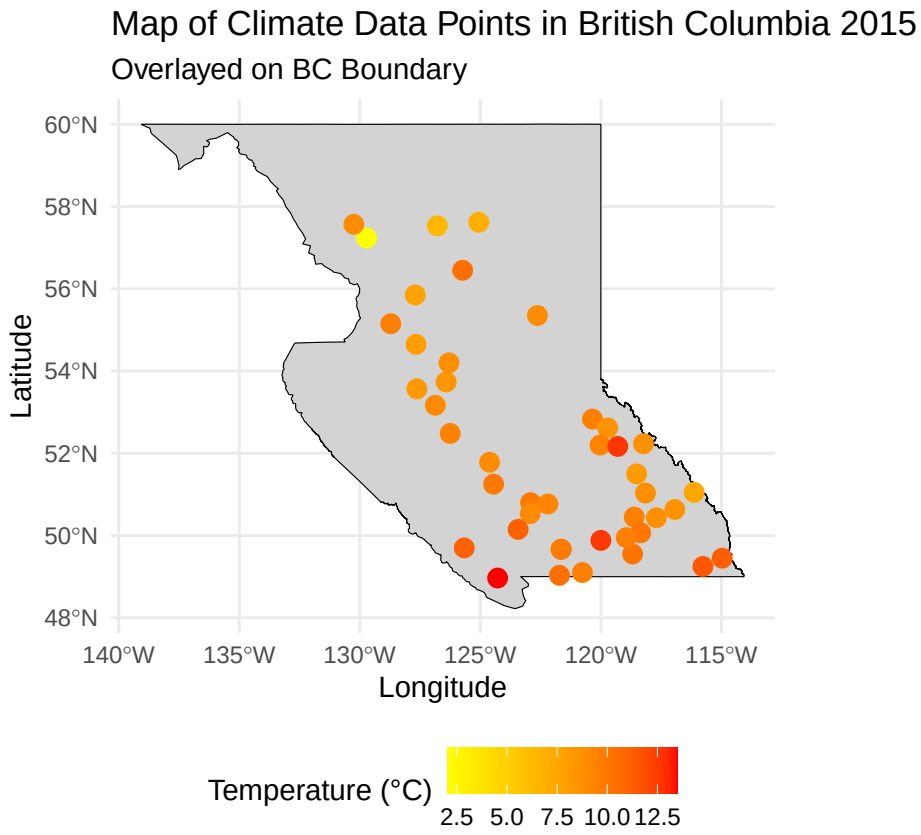
Let's take a closer look at our weather station data to better understand their locations. Ideally, we'd want weather stations to be evenly distributed across BC to ensure a reliable and comprehensive spread of data. However, British Columbia is known for its extremely diverse and rugged geography, which includes some of the most remote and impassable areas in the world. While this creates stunning landscapes, it also presents significant challenges when it comes to placing weather stations.

As a result, most of BC's weather stations are situated in more accessible locations, such as along highways and in densely populated regions. This leads to an uneven distribution of stations, which is evident in the map below. However, this unevenness isn't necessarily a major setback. In fact, over 80% of beekeepers in the province are located in the Lower Mainland, Fraser Valley, and Okanagan areas. These regions are also where the majority of agricultural activity takes place, including honey production (Potts et al., 2010).

So, while the station distribution may not be perfectly uniform, the concentration of weather stations in the areas most relevant to beekeepers allows us to focus our analysis on the regions that matter most. This ensures that our study of honey production and temperature impacts is based on data that accurately reflects the conditions in key farming areas (Wheeler & von Braun, 2013).

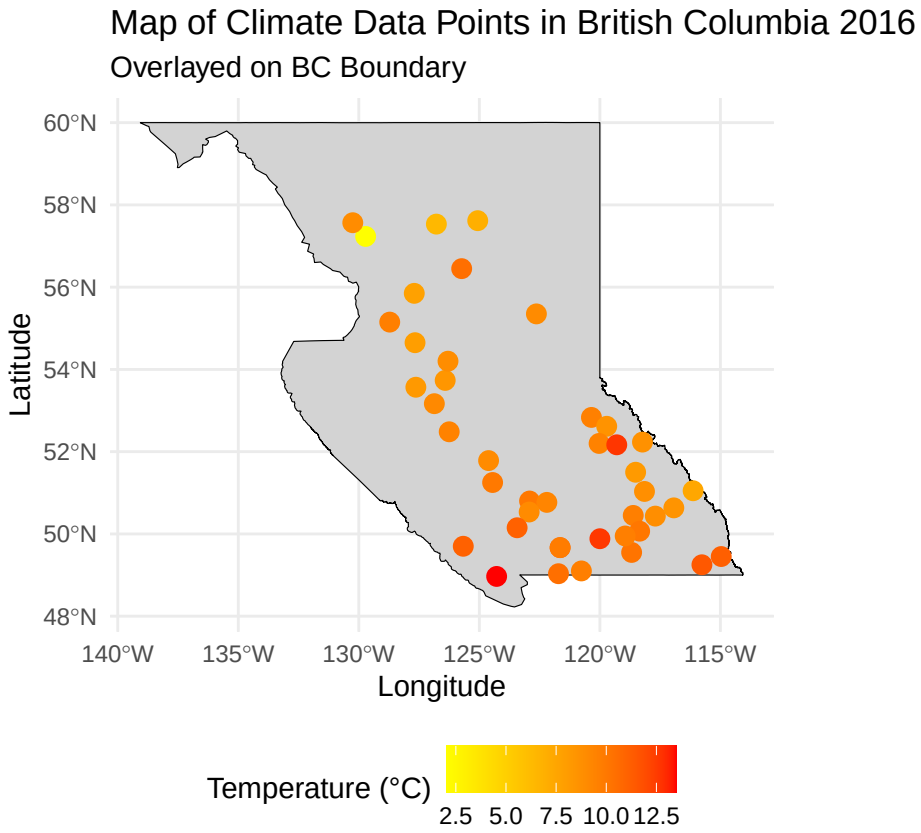
```r
#Mapping your Climate Data

# Read the CSV file
climate_data <- read.csv("ClimateData_2015.csv")

# Check first few rows of Latitude and Longitude columns
head(climate_data$Latitude)

# Ensure Latitude and Longitude are in character format
climate_data$Latitude <- as.character(climate_data$Latitude)
climate_data$Longitude <- as.character(climate_data$Longitude)
```

```r
# Clean the columns by removing non-numeric characters
# (except for negative sign and decimal)
climate_data$Latitude <- as.numeric(gsub("[^0-9.-]", "", climate_data$Latitude))
climate_data$Longitude <- as.numeric(gsub("[^0-9.-]", "", climate_data$Longitude))

# Check if there are any missing values after cleaning
sum(is.na(climate_data$Latitude))  # Check for missing latitudes
sum(is.na(climate_data$Longitude)) # Check for missing longitudes

# Optionally, filter out rows with missing latitude or longitude
climate_data <- climate_data %>%
  filter(!is.na(Latitude) & !is.na(Longitude))

# If necessary, adjust longitude values
climate_data <- climate_data %>%
  mutate(Longitude = ifelse(Longitude > 0, Longitude * -1, Longitude))

# Create a simple feature object (sf) using Latitude and Longitude
climate_sf <- st_as_sf(climate_data, coords = c("Longitude", "Latitude"), crs = 4326)

# Optionally, you can select columns that you want to keep in the shapefile
# climate_sf <- climate_sf %>% select(Your_Columns_Here)

# Write the shapefile to disk
st_write(climate_sf, "ClimateData_2015.shp")

# Confirmation message
print("Shapefile has been created: ClimateData_2015.shp")

# Load the shapefiles
climate_sf <- st_read("ClimateData_2015.shp")
bc_boundary <- st_read("ABMS_PROV_polygon.shp")
bc_boundary <- st_transform(bc_boundary, crs = 4326)

# Create the map
ggplot() +
  geom_sf(data = bc_boundary, fill = "lightgrey", color = "black") +
  # Map the TEMP variable to color
  geom_sf(data = climate_sf, aes(color = TEMP), size = 3) +
  # Adjust color gradient as needed
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal() +
  labs(title = "Map of Climate Data Points in British Columbia 2015",
       subtitle = "Overlayed on BC Boundary",
       x = "Longitude",  # Use Longitude for x-axis
       y = "Latitude",   # Use Latitude for y-axis
       color = "Temperature (°C)") + # Label for color legend
  theme(legend.position = "bottom")
```




```
#Mapping your Climate Data

# Read the CSV file
climate_data <- read.csv("ClimateData_2016.csv")

climate_data$Latitude <- as.numeric(gsub("[^0-9.-]", "",
                                         climate_data$Latitude))
climate_data$Longitude <- as.numeric(gsub("[^0-9.-]", "",
                                          climate_data$Longitude))


# Ensure Latitude and Longitude columns are correctly formatted
# Assuming the columns are named "Latitude" and "Longitude"
climate_data <- climate_data %>%
  mutate(Latitude = as.numeric(Latitude),
         Longitude = as.numeric(Longitude))

climate_data <- climate_data %>%
  mutate(Longitude = ifelse(Longitude > 0, Longitude * -1, Longitude))

#Create a simple feature object (sf) using Latitude and Longitude
climate_sf <- st_as_sf(climate_data, coords = c("Longitude", "Latitude"), crs = 4326)

#Optionally, you can select columns that you want to keep in the shapefile
#climate_sf <- climate_sf %>% select(Your_Columns_Here)
```

```r
# Write the shapefile to disk
st_write(climate_sf, "ClimateData_2016.shp")

# Confirmation message
print("Shapefile has been created: ClimateData_2016.shp")

# Load the shapefiles
climate_sf <- st_read("ClimateData_2015.shp")
bc_boundary <- st_read("ABMS_PROV_polygon.shp")
bc_boundary <- st_transform(bc_boundary, crs = 4326)

# Create the map
ggplot() +
  geom_sf(data = bc_boundary, fill = "lightgrey", color = "black") +
  # Map the TEMP variable to color
  geom_sf(data = climate_sf, aes(color = TEMP), size = 3) +
  # Adjust color gradient as needed
  scale_color_gradient(low = "yellow", high = "red") +
  theme_minimal() +
  labs(title = "Map of Climate Data Points in British Columbia 2016",
       subtitle = "Overlayed on BC Boundary",
       x = "Longitude",  # Use Longitude for x-axis
       y = "Latitude",   # Use Latitude for y-axis
       color = "Temperature (°C)") + # Label for color legend
  theme(legend.position = "bottom")
```



Using IDW to esitimate the climate the areas not covered by weather stations

We will be using Inverse Distance Weighting (IDW) to estimate the temperature for all of British Columbia based on the data points we currently have. IDW is a spatial interpolation technique that assumes that the value of a variable (in this case, temperature) at any location between two known points will be somewhere between their values. The closer a new location is to a known point, the more influence that point will have on the estimated temperature for the new location (Li & Heap, 2014).

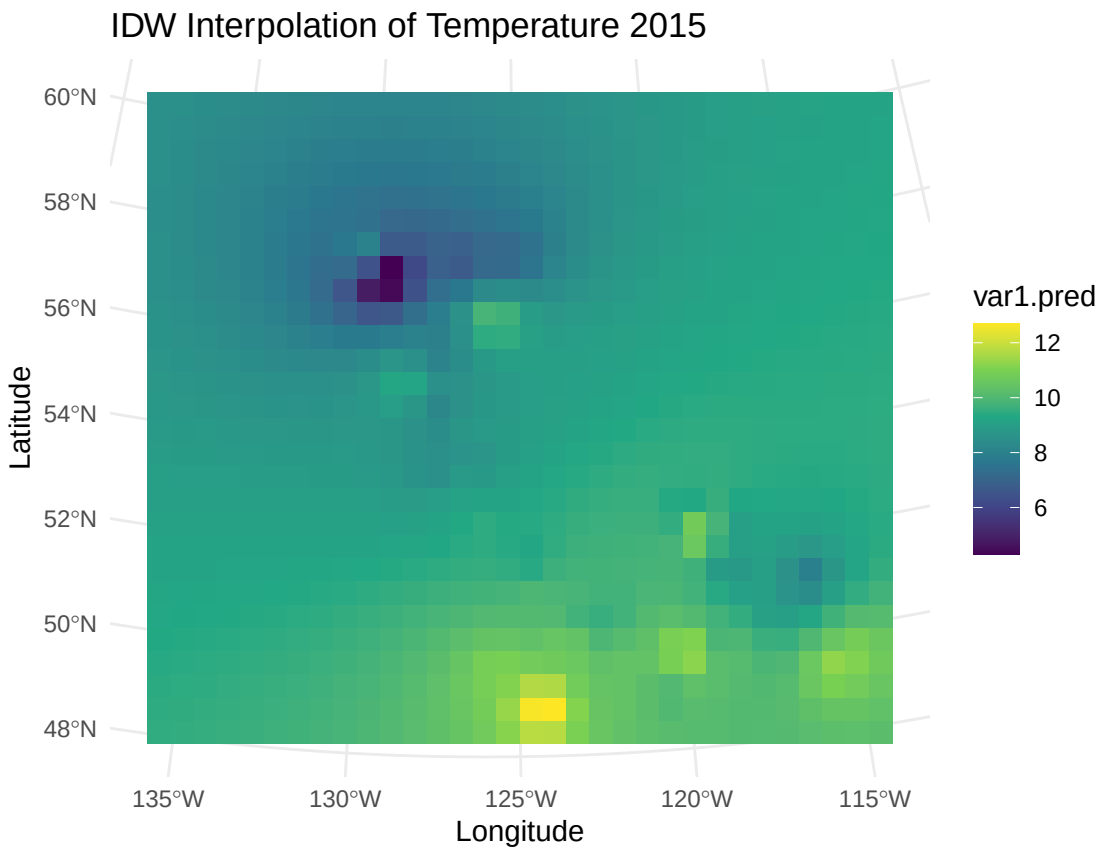
For example, if one weather station has a recorded temperature of 10°C and another has 20°C, we might assume that a location halfway between the two would have a temperature of 15°C. Similarly, if a point is closer to the 20°C station than the 10°C station, we might estimate its temperature to be closer to 20°C—let's say around 19°C.

IDW applies this logic across every possible location in our study area, using the nearest weather stations to estimate the temperature at unmeasured locations. The output of this process is a continuous surface of temperature estimates, which we can visualize as a color-scaled map. This map will help us understand temperature gradients across British Columbia and better analyze how temperature may affect honey production in different regions (Li & Heap, 2014).

Now, let's take a look at the IDW output.

```
#Spatial Interpolation of Your Climate Data
#Inverse Distance Weighting

library(sf)
library(gstat)
library(sp)

#Read your shapefiles
climate_data_2015 <- st_read("ClimateData_2015.shp")
bc_boundary <- st_read("ABMS_PROV_polygon.shp")

#Check validity of the polygons in the bc_boundary shapefile
valid_polygons <- st_is_valid(bc_boundary)
#print(valid_polygons)

#Fix invalid polygons if any exist
bc_boundary <- st_make_valid(bc_boundary)

#Check the area of each polygon to filter out zero-area polygons
bc_boundary$area <- st_area(bc_boundary)

#Remove zero-area polygons from the bc_boundary (if any)
#Convert area to numeric (without units)
bc_boundary$area <- as.numeric(st_area(bc_boundary))
bc_boundary <- bc_boundary[bc_boundary$area > 0, ]  # Filter out zero-area polygons

#Ensure grid is a valid sf object and check for zero-area polygons
grid <- st_make_grid(bc_boundary, cellsize = c(50000, 50000))

# Ensure the grid is an sf object and check its geometry column
grid_sf <- st_as_sf(grid)  # Convert grid to sf if it's not already

# Calculate the area of each grid cell
grid_sf$area <- as.numeric(st_area(grid_sf))  # Calculate area (as numeric to remove units)
```

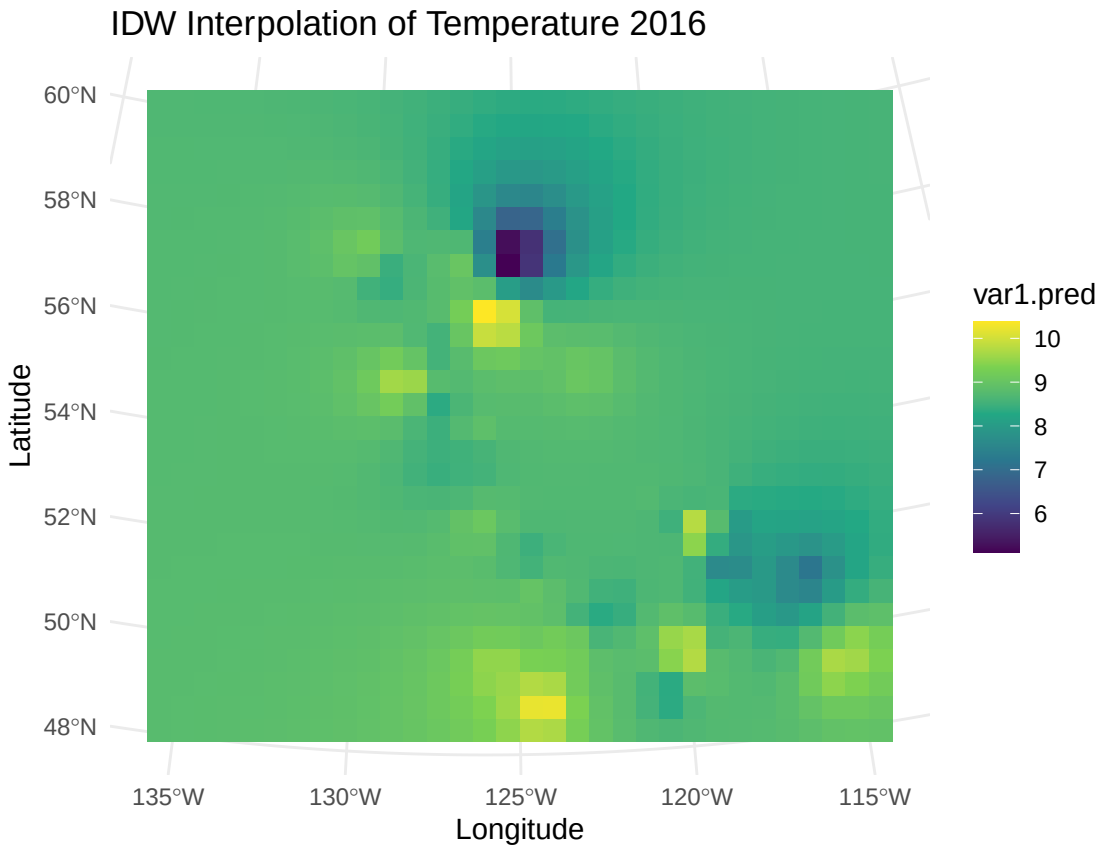
```r
# Remove zero-area grid cells (if any)
grid_sf <- grid_sf[grid_sf$area > 0, ]  # Filter out cells with zero area



# Ensure the CRS for both climate_data and bc_boundary
# (and grid) are the same, for example, EPSG:3005
climate_data_2015 <- st_transform(climate_data_2015, crs = 3005)
bc_boundary <- st_transform(bc_boundary, crs = 3005)
grid <- st_transform(grid, crs = 3005)

#Convert sf objects to sp objects for use with gstat
climate_data_sp_2015 <- as(climate_data_2015, "Spatial")
grid_sp <- as(st_as_sf(grid), "Spatial")

#Perform IDW interpolation using the gstat package
idw_result_2015 <- gstat::idw(TEMP ~ 1,
                          locations = climate_data_sp_2015,
                          newdata = grid_sp,
                          idp = 2)


#Convert idw_result to an sf object
idw_sf_2015 <- st_as_sf(idw_result_2015)

#Extract coordinates
idw_sf_2015 <- st_as_sf(idw_result_2015)

#Plot the results using geom_sf() for better handling of sf objects
ggplot(data = idw_sf_2015) +
  geom_sf(aes(fill = var1.pred), color = NA) +  # Fill based on predicted values
  scale_fill_viridis_c() +
  labs(title = "IDW Interpolation of Temperature 2015", x = "Longitude", y = "Latitude") +
  theme_minimal() +
  theme(legend.position = "right")
```




```
#Save the result to a shapefile if needed
st_write(idw_sf_2015, "IDW_Result_2015.shp", driver = "ESRI Shapefile", delete_dsn = TRUE)
```

```
#Spatial Interpolation of Your Climate Data
#Inverse Distance Weighting

library(sf)
library(gstat)
library(sp)

#Read your shapefiles
climate_data_2016 <- st_read("ClimateData_2016.shp")
bc_boundary <- st_read("ABMS_PROV_polygon.shp")

#Check validity of the polygons in the bc_boundary shapefile
valid_polygons <- st_is_valid(bc_boundary)
#print(valid_polygons)

#Fix invalid polygons if any exist
bc_boundary <- st_make_valid(bc_boundary)

#Check the area of each polygon to filter out zero-area polygons
bc_boundary$area <- st_area(bc_boundary)

#Remove zero-area polygons from the bc_boundary (if any)
#Convert area to numeric (without units)
```

```r
bc_boundary$area <- as.numeric(st_area(bc_boundary))
bc_boundary <- bc_boundary[bc_boundary$area > 0, ]  # Filter out zero-area polygons

#Ensure grid is a valid sf object and check for zero-area polygons
grid <- st_make_grid(bc_boundary, cellsize = c(50000, 50000))

# Ensure the grid is an sf object and check its geometry column
grid_sf <- st_as_sf(grid)  # Convert grid to sf if it's not already

# Calculate the area of each grid cell
grid_sf$area <- as.numeric(st_area(grid_sf))# Calculate area (as numeric to remove units)

# Remove zero-area grid cells (if any)
grid_sf <- grid_sf[grid_sf$area > 0, ]  # Filter out cells with zero area



#Ensure the CRS for both climate_data and bc_boundary (and grid) are the same,
#for example, EPSG:3005
climate_data_2016 <- st_transform(climate_data_2016, crs = 3005)
bc_boundary <- st_transform(bc_boundary, crs = 3005)
grid <- st_transform(grid, crs = 3005)

#Convert sf objects to sp objects for use with gstat
climate_data_sp_2016 <- as(climate_data_2016, "Spatial")
grid_sp <- as(st_as_sf(grid), "Spatial")

#Perform IDW interpolation using the gstat package
idw_result_2016 <- gstat::idw(TEMP ~ 1,
                         locations = climate_data_sp_2016,
                         newdata = grid_sp,
                         idp = 2)


#Convert idw_result to an sf object
idw_sf_2016 <- st_as_sf(idw_result_2016)

#Extract coordinates
idw_sf_2016 <- st_as_sf(idw_result_2016)

#Plot the results using geom_sf() for better handling of sf objects
ggplot(data = idw_sf_2016) +
  geom_sf(aes(fill = var1.pred), color = NA) +  # Fill based on predicted values
  scale_fill_viridis_c() +
  labs(title = "IDW Interpolation of Temperature 2016", x = "Longitude", y = "Latitude") +
  theme_minimal() +
  theme(legend.position = "right")
```



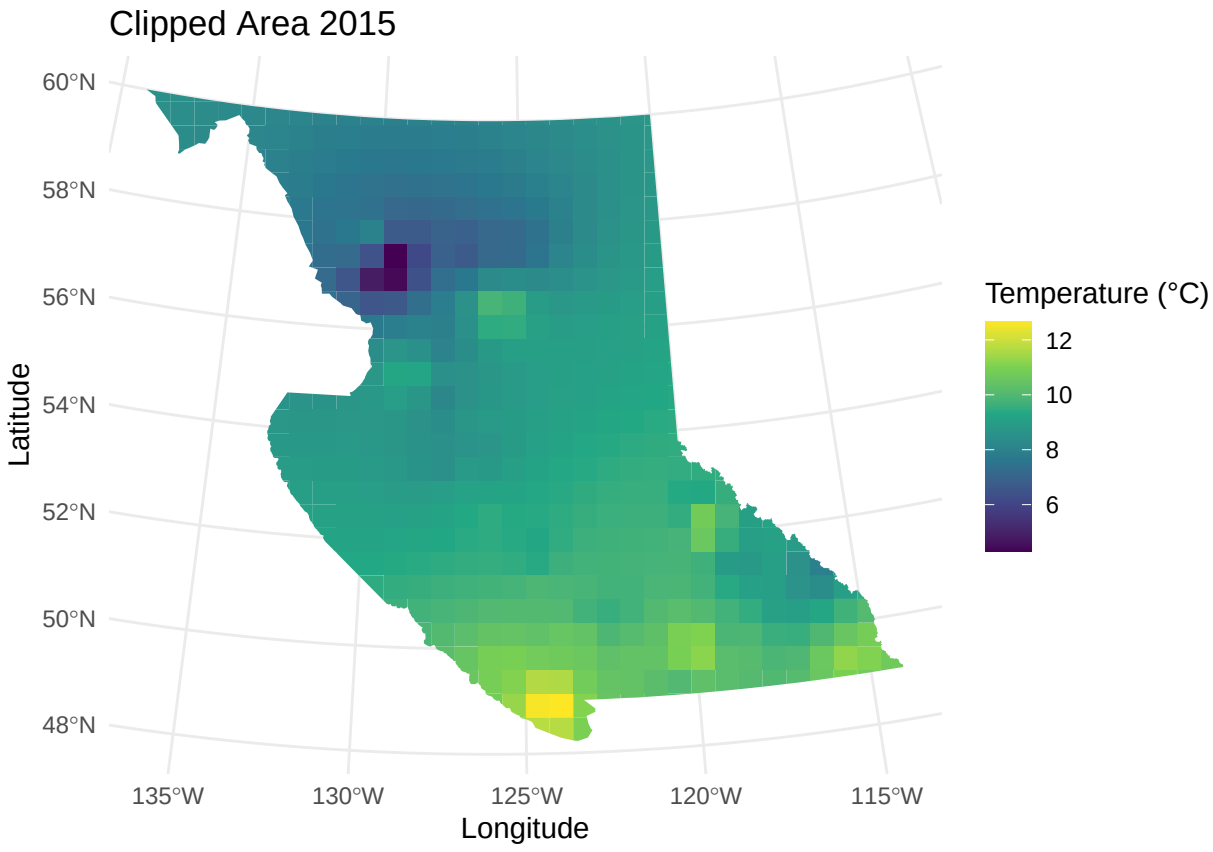

```
#Save the result to a shapefile if needed
st_write(idw_sf_2016, "IDW_Result_2016.shp", driver = "ESRI Shapefile", delete_dsn = TRUE)
```

This output may be a bit overwhelming, so let's simplify it by clipping the data to the boundary of British Columbia. Clipping is when we use a polygon to define a specific area and remove all data outside of that polygon, effectively "clipping" the data and leaving only the area we care about.

In our case, we will use the boundary of British Columbia as the polygon to clip the temperature data. This will allow us to better understand and visualize the data. Let's go ahead and clip the temperature data.

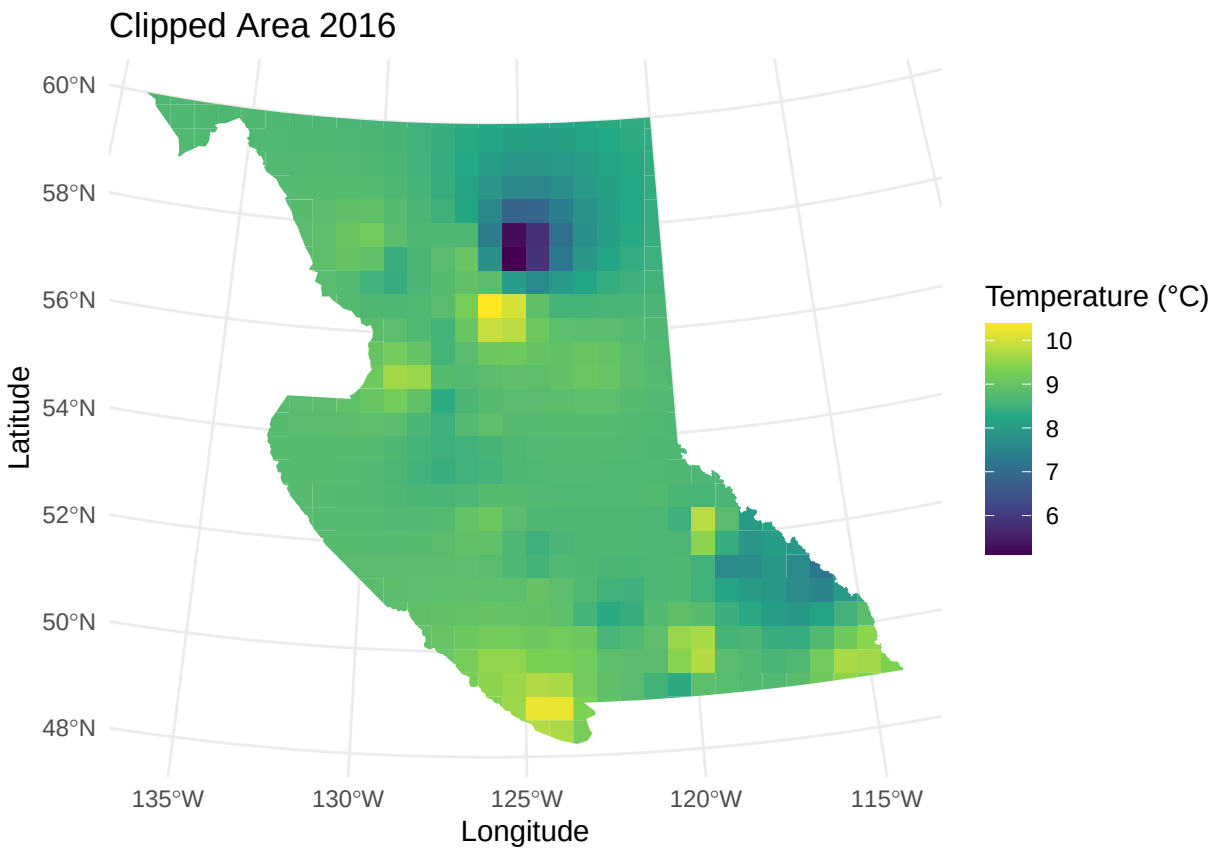
```
#We will now clip the IDW results to the BC bounday.


# Load the polygon shapefile for clipping
abms_prov_polygon <- st_read("ABMS_PROV_polygon.shp")

# Check the CRS of both objects
crs_idw_2015 <- st_crs(idw_sf_2015)  # CRS of IDW result
crs_polygon <- st_crs(abms_prov_polygon)  # CRS of the polygon shapefile

# Transform the CRS of either shapefile if they do not match
if (crs_idw_2015 != crs_polygon) {
  # Transform the IDW result to match the CRS of the polygon
  idw_sf_2015 <- st_transform(idw_sf_2015, crs = crs_polygon)
  # Transform IDW result to polygon's CRS
  message("Transformed IDW result CRS to match the polygon.")
```

```r
} else {
  message("CRS of IDW result and polygon already match.")
}

# Now attempt the intersection again
idw_clipped_2015 <- st_intersection(idw_sf_2015, abms_prov_polygon)

# Check the results of clipping
print(st_geometry(idw_clipped_2015))# Check geometry to ensure it's clipped correctly

#Create the map of the clipped results
ggplot(data = idw_clipped_2015) +
  geom_sf(aes(fill = var1.pred), color = NA) +
  # Fill based on predicted temperature values
  scale_fill_viridis_c(option = "D") +
  # Use viridis color scale for better readability
  labs(title = "Clipped Area 2015",
       fill = "Temperature (°C)",  # Change label as appropriate
       x = "Longitude",
       y = "Latitude") +
  theme_minimal() +
  theme(legend.position = "right")
```




```r
#Save the map as an image file (optional)
ggsave("Clipped_IDW_Interpolation_Map_2015.png", width = 10, height = 8, dpi = 300)
```

```r
#We will now clip the IDW results to the BC bounday.


# Load the polygon shapefile for clipping
abms_prov_polygon <- st_read("ABMS_PROV_polygon.shp")  # Ensure the path is correct

# Verify the structure of the polygon shapefile
print(head(abms_prov_polygon))
# Check the CRS of both objects
crs_idw_2016 <- st_crs(idw_sf_2016)  # CRS of IDW result
crs_polygon <- st_crs(abms_prov_polygon)  # CRS of the polygon shapefile

# Transform the CRS of either shapefile if they do not match
if (crs_idw_2016 != crs_polygon) {
  # Transform the IDW result to match the CRS of the polygon
  idw_sf_2016 <- st_transform(idw_sf_2016, crs = crs_polygon)
  message("Transformed IDW result CRS to match the polygon.")
} else {
  message("CRS of IDW result and polygon already match.")
}

# Now attempt the intersection again
idw_clipped_2016 <- st_intersection(idw_sf_2016, abms_prov_polygon)

# Check the results of clipping
print(st_geometry(idw_clipped_2016))  # Check geometry to ensure it's clipped correctly

#Create the map of the clipped results
ggplot(data = idw_clipped_2016) +
  geom_sf(aes(fill = var1.pred), color = NA) +# Fill based on predicted temperature values
  scale_fill_viridis_c(option = "D") +  # Use viridis color scale for better readability
  labs(title = "Clipped Area 2016",
       fill = "Temperature (°C)",  # Change label as appropriate
       x = "Longitude",
       y = "Latitude") +
  theme_minimal() +
  theme(legend.position = "right")
```




```
#Save the map as an image file (optional)
ggsave("Clipped_IDW_Interpolation_Map_2016.png", width = 10, height = 8, dpi = 300)
```

Now the map is much clearer. As expected, the Lower Mainland shows generally higher temperatures compared to the northern regions, which aligns with the typical pattern of it getting colder the further north you go.

You'll also notice the presence of more "hot spots" in the Lower Mainland. These areas are particularly important because they represent regions with higher temperatures, which also happens to be where most of our beekeeping operations are. Understanding these hot spots will be crucial for analyzing how temperature impacts honey production in BC, particularly in these high-risk areas.

Bee Data Analysis

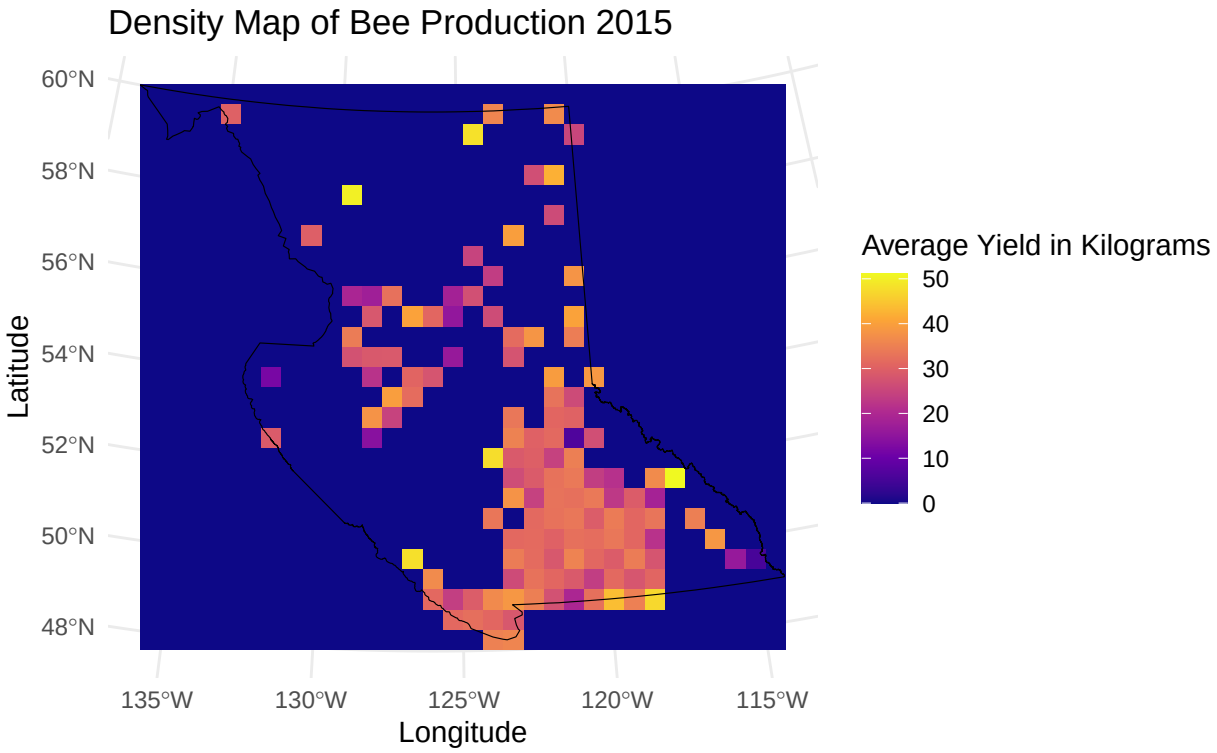
Let's now take a closer look at the bee data. For this analysis, we will focus on the amount of honey produced by each bee colony, as honey production is directly influenced by temperature. High temperatures can cause significant stress to bee colonies, leading to reduced foraging, diminished colony health, and in some cases, partial or complete colony failure (Potts et al., 2010). As a result, less production ultimately means lower honey yield. By examining the honey yield per colony, we can gain valuable insights into how extreme heat affects production.

To provide context for our analysis, we will compare honey yield data from 2015 and 2016. By contrasting the yields from the unusually hot summer of 2016 with the more temperate conditions of 2015, we can illustrate how the higher temperatures of 2016 likely had a significant impact on honey production. The maps we create will allow us to visually compare honey production across British Columbia, helping us identify areas

that were most affected by the heat and highlighting regions where beekeepers may have experienced reduced yields due to the extreme weather conditions.

```
#Creating a Density Map of Your Events Data For KG Density 2015

# Load required libraries
library(sf)
library(raster)
library(ggplot2)
library(dplyr)

# 1. Load and clean the beekeeping data
bee_data_2015 <- read.csv("Beekeeping_Census.csv")

# Clean Latitude and Longitude (ensure they are numeric and properly formatted)
bee_data_2015$Latitude <- as.numeric(gsub("[^0-9.-]", "", bee_data_2015$Latitude))
bee_data_2015$Longitude <- as.numeric(gsub("[^0-9.-]", "", bee_data_2015$Longitude))

# Ensure Longitude is negative for Western Hemisphere coordinates
bee_data_2015 <- bee_data_2015 %>%
  mutate(Longitude = ifelse(Longitude > 0, Longitude * -1, Longitude))

# Create the spatial object (sf) using Latitude and Longitude
bee_sf_2015 <- st_as_sf(bee_data_2015, coords = c("Longitude", "Latitude"), crs = 4326)

#Load the boundary shapefile (e.g., for a province or study area)
abms_prov_polygon <- st_read("ABMS_PROV_polygon.shp")  # Adjust the path as needed

#Set a common CRS (Choose a projection suitable for your area, e.g., UTM for BC: EPSG:3005)
target_crs <- 3005  # UTM projection (adjust based on your area)

#Transform both the bee data and boundary polygon to the same CRS (target CRS: EPSG:3005)
bee_sf_2015 <- st_transform(bee_sf_2015, crs = target_crs)
abms_prov_polygon <- st_transform(abms_prov_polygon, crs = target_crs)

#Create a grid to cover the bounding box of the study area (e.g., BC)
bbox <- st_bbox(abms_prov_polygon)  # Get the bounding box of the boundary
raster_res_2015 <- 50000  # Grid resolution (50 km x 50 km cells)
raster_template_2015 <- raster::raster(extent(bbox), res = c(raster_res_2015, raster_res_2015))

# Set CRS for the raster template (matching the target CRS of the polygon and bee data)
crs(raster_template_2015) <- CRS("+init=epsg:3005")

#Rasterize the bee data (count the number of colonies in each grid cell)
density_raster_2015 <- raster::rasterize(bee_sf_2015, raster_template_2015, field
                                         = "average_yield_2015.kg.", fun = mean)

#Ensure NAs are turned to zeros (for cells with no data)
density_raster_2015[is.na(density_raster_2015)] <- 0

#Convert the raster to a data frame for ggplot
density_df_2015 <- as.data.frame(density_raster_2015, xy = TRUE)

#Replace NAs with zeros (if any)
```

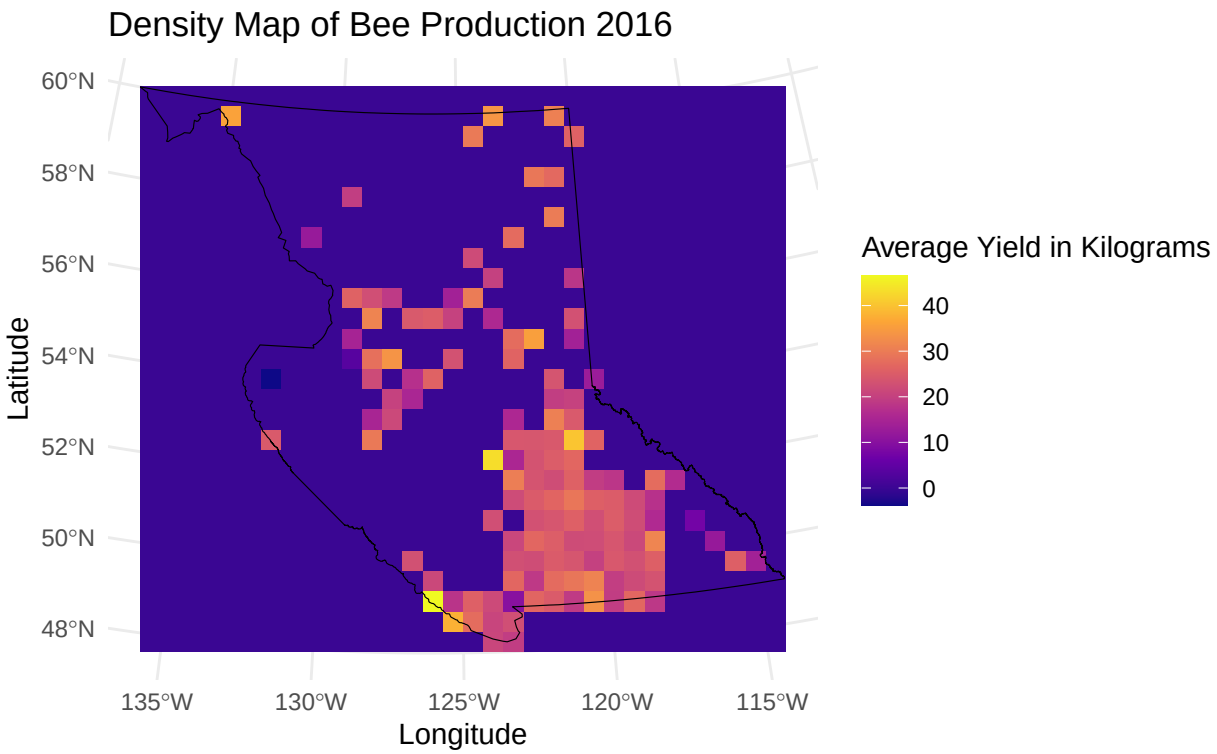
```r
density_df_2015[is.na(density_df_2015)] <- 0

#Rename the column to "bees" to clarify what we're visualizing
colnames(density_df_2015)[colnames(density_df_2015) == "layer"] <- "Yield"


#Create the plot
ggplot() +
  # Use 'bees' for density
  geom_raster(data = density_df_2015, aes(x = x, y = y, fill = Yield)) +
  geom_sf(data = abms_prov_polygon, fill = NA, color = "black") +  # Boundary polygon
  scale_fill_viridis_c(option = "plasma") +  # Color scale for density (plasma)
  theme_minimal() +
  labs(title = "Density Map of Bee Production 2015",
       x = "Longitude",
       y = "Latitude",
       fill = "Average Yield in Kilograms")  # Customize labels
```




```r
#Creating a Density Map of Your Events Data For KG Density 2016

# reload required libraries
library(sf)
library(raster)
library(ggplot2)
library(dplyr)
```

```r
#Load and clean the beekeeping data
bee_data_2016 <- read.csv("Beekeeping_Census.csv")

#Clean Latitude and Longitude (ensure they are numeric and properly formatted)
bee_data_2016$Latitude <- as.numeric(gsub("[^0-9.-]", "", bee_data_2016$Latitude))
bee_data_2016$Longitude <- as.numeric(gsub("[^0-9.-]", "", bee_data_2016$Longitude))

#Ensure Longitude is negative for Western Hemisphere coordinates
bee_data_2016 <- bee_data_2016 %>%
  mutate(Longitude = ifelse(Longitude > 0, Longitude * -1, Longitude))

#Create the spatial object (sf) using Latitude and Longitude
# WGS84 (lat/lon) = 4326
bee_sf_2016 <- st_as_sf(bee_data_2016, coords = c("Longitude", "Latitude"), crs = 4326)

#Load the boundary shapefile (e.g., for a province or study area)
abms_prov_polygon <- st_read("ABMS_PROV_polygon.shp")  # Adjust the path as needed

#Set a common CRS (Choose a projection suitable for your area, e.g., UTM for BC: EPSG:3005)
target_crs <- 3005  # UTM projection (adjust based on your area)

#Transform both the bee data and boundary polygon to the same CRS (target CRS: EPSG:3005)
bee_sf_2016 <- st_transform(bee_sf_2016, crs = target_crs)
abms_prov_polygon <- st_transform(abms_prov_polygon, crs = target_crs)

#Create a grid to cover the bounding box of the study area (e.g., BC)
bbox <- st_bbox(abms_prov_polygon)  # Get the bounding box of the boundary
raster_res_2016 <- 50000  # Grid resolution (50 km x 50 km cells)
raster_template_2016 <- raster::raster(extent(bbox), res = c(raster_res_2016, raster_res_2016))

#Set CRS for the raster template (matching the target CRS of the polygon and bee data)
crs(raster_template_2016) <- CRS("+init=epsg:3005")

#Rasterize the bee data (count the number of colonies in each grid cell)
density_raster_2016 <- raster::rasterize(bee_sf_2016, raster_template_2016, field
                                         = "average_yield_2016.kg.", fun = mean)

#Ensure NAs are turned to zeros (for cells with no data)
density_raster_2016[is.na(density_raster_2016)] <- 0

#Convert the raster to a data frame for ggplot
density_df_2016 <- as.data.frame(density_raster_2016, xy = TRUE)

#Replace NAs with zeros (if any)
density_df_2016[is.na(density_df_2016)] <- 0

#Rename the column to "bees" to clarify what we're visualizing
colnames(density_df_2016)[colnames(density_df_2016) == "layer"] <- "Yield"


#Create the plot
ggplot() +
  # Use 'bees' for density
```

```
  geom_raster(data = density_df_2016, aes(x = x, y = y, fill = Yield)) +
  geom_sf(data = abms_prov_polygon, fill = NA, color = "black") +  # Boundary polygon
  scale_fill_viridis_c(option = "plasma") +  # Color scale for density (plasma)
  theme_minimal() +
  labs(title = "Density Map of Bee Production 2016",
       x = "Longitude",
       y = "Latitude",
       fill = "Average Yield in Kilograms")  # Customize labels
```



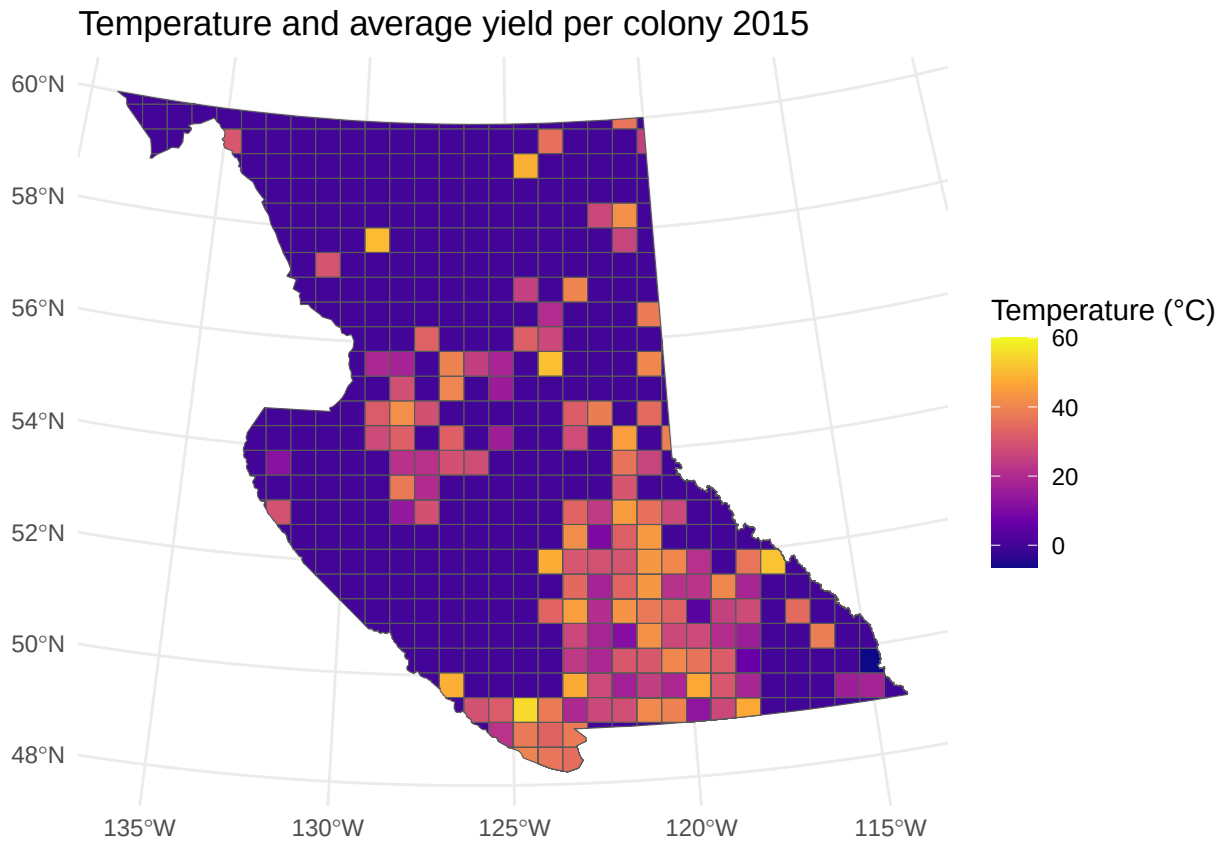
Combining Temperature Location Data with Bee Colony Data

Now, let's take our analysis a step further by combining the honey yield data with the temperature data we generated earlier. By overlaying these two datasets for each year, we can create a clear visual representation of how higher temperatures in certain areas of the province correlate with lower honey yields.

Looking at the temperature data, we can identify regions experiencing significant heat, particularly in the Lower Mainland and Okanagan. These areas, with their elevated temperatures, are where we would expect to see a notable decrease in honey production. This combination of data allows us to directly link temperature extremes to their impact on honey yields, providing valuable insights into how climate variations affect bee colonies and agricultural output.

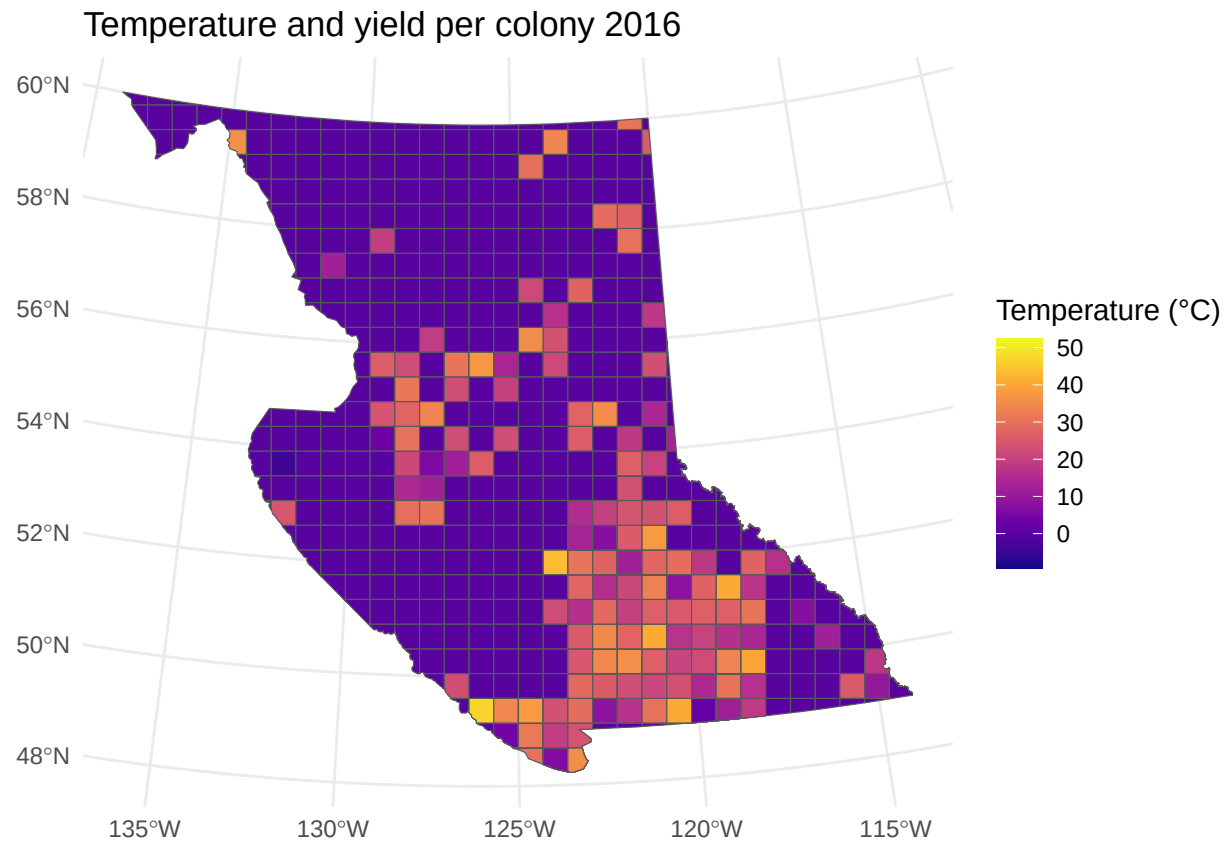
```
#Combining Your Climate and Events Data 2015 yields


# Ensure the CRS of both datasets are the same
```

```r
# Loading necessary libraries
library(sf)
library(tidyverse)
library(viridis)


# Perform a spatial join between 'idw_clipped' and 'density_sf'
joined_sf_2015 <- st_join(idw_clipped_2015, bee_sf_2015, join = st_intersects)


final_data_2015 <- joined_sf_2015[, c("var1.pred", "average_yield_2015.kg.")]

# Rename column
final_data_2015 <- final_data_2015 %>%
  rename(temperature = var1.pred)

# Replace NA values in the fires column with 0
final_data_2015 <- final_data_2015 %>%
  mutate(average_yield_2015.kg. = ifelse(is.na(average_yield_2015.kg.),
                                         0, average_yield_2015.kg.))

#Create the map
ggplot(data = final_data_2015) +
  geom_sf(aes(fill = average_yield_2015.kg.)) +
  scale_fill_viridis_c(option = "C") +
  theme_minimal() +
  labs(title = "Temperature and average yield per colony 2015",
       fill = "Temperature (°C)") +
  theme(legend.position = "right")
```




```
#Save final_data as a shapefile
st_write(final_data_2015, "final_data_2015.shp", delete_dsn = TRUE)

final_data_df_2015 <- st_drop_geometry(final_data_2015)

#Write as CSV
write.csv(final_data_df_2015, "final_data_2015.csv", row.names = FALSE)


#Combining Your Climate and Events Data 2016 yields


# Ensure the CRS of both datasets are the same
# Loading necessary libraries
library(sf)
library(tidyverse)
library(viridis)


#Perform a spatial join between 'idw_clipped' and 'density_sf'
joined_sf_2016 <- st_join(idw_clipped_2016, bee_sf_2016, join = st_intersects)


final_data_2016 <- joined_sf_2016[, c("var1.pred", "average_yield_2016.kg.")]

#Rename column
```

```r
final_data_2016 <- final_data_2016 %>%
  rename(temperature = var1.pred)

#Replace NA values in the fires column with 0
final_data_2016 <- final_data_2016 %>%
  mutate(average_yield_2016.kg. = ifelse(is.na(average_yield_2016.kg.),
                                         0, average_yield_2016.kg.))

#Create the map
ggplot(data = final_data_2016) +
  geom_sf(aes(fill = average_yield_2016.kg.)) +
  scale_fill_viridis_c(option = "C") +
  theme_minimal() +
  labs(title = "Temperature and yield per colony 2016",
       fill = "Temperature (°C)") +
  theme(legend.position = "right")
```




```r
# Save final_data as a shapefile
st_write(final_data_2016, "final_data_2016.shp", delete_dsn = TRUE)

final_data_df_2016 <- st_drop_geometry(final_data_2016)

# Write as CSV
write.csv(final_data_df_2016, "final_data_2016.csv", row.names = FALSE)
```

Conclusion

This tutorial has guided you through the process of analyzing the effects of extreme temperatures on honey production in British Columbia. By combining air temperature data with honey yield statistics, we have demonstrated that severe heat correlates with reduced honey production.

Using spatial analysis techniques like Inverse Distance Weighting (IDW) and the creation of density maps, we were able to visualize how temperature is distributed across the province and how it correlates with areas of lower honey yields. By comparing data from 2015, a milder year, to 2016, we observed a significant decline in honey production across various regions, particularly in the Lower Mainland where most of the province's honey is produced.

These findings show the importance of understanding climate variability, especially temperature, and its direct impact on agricultural output. For beekeepers, recognizing this correlation will be crucial in adapting to the changing environmental conditions that affect their colonies and honey yields.

Through this tutorial, you've gained practical skills in data cleaning, spatial analysis, and integrating multiple datasets. While our focus here was on temperature and honey production, the methods and techniques you've learned can be applied to analyze the effects of climate on a wide range of agricultural and environmental topics.

References

Fotheringham, A. S., Brunsdon, C., & Charlton, M. (2002). Geographically weighted regression: The analysis of spatially varying relationships. Wiley.

Li, H., & Heap, A. D. (2014). Spatial interpolation methods: A review. The Geographer, 52(4), 174-181. https://doi.org/10.1080/0042098X.2013.863778

Potts, S. G., et al. (2010). The effects of climate change on honeybee health and productivity. Trends in Ecology & Evolution, 25(10), 585-592. https://doi.org/10.1016/j.tree.2010.07.001

Weaver, C. P., et al. (2016). Climate data for agriculture: A guide for British Columbia. Pacific Climate Impacts Consortium. https://www.pacificclimate.org/

Wheeler, T., & von Braun, J. (2013). Climate change and agriculture: Impacts, adaptation, and mitigation. CABI.