STAT5125_Project_Yuxin_Jen_Rene

Yuxin Zhang, Jennifer Nguyen, Rene Chang

2023-03-31

Contents

Introduction and Motivation

Climate change is not a new problem. Although the earliest research was published in the late 1800s, this topic was not taken as a serious global issue until the 1960s.

As the problem of climate change becomes increasingly urgent, data science is also being recognized as an essential tool in the fight against climate change. The skills in data science will allow better understanding of the complex interplay of various environmental factors and take proactive steps to reduce our impact on the planet.

One of the main human impact worsening climate change is our carbon foot print. This leads the motivation of our project to analyze the CO2 emissions produced. The data is sourced from the International Monetary Fund climate change data dashboard. Their mission is to achieve sustainable growth and prosperity for all of its 190 member countries by supporting economic policies that promote financial stability and monetary cooperation, which are essential to increase productivity, job creation, and economic well-being.

With that said, this project will be analyzing the combination of two data sets; Direct Investment related indicators and Climate related disaster frequency.

Our combine data consists of CO2 emissions (in Metric Tons) of 60 different countries spanning from 2005 to 2018 by 4 types of enterprise and 15 sectors. There are four types of enterprise of focus; exports of domestic controlled enterprises, exports of foreign controlled multinational enterprises, output of domestic controlled enterprises, and output of foreign controlled multinational enterprises.

With 826 total observations and 59 combined variables, we hope to account for the frequency of disaster events by country and year with models listed below in the outline.

Outline

- Data pre-processing and tidying (including test/train split)
- Reasoning of research question
- Visualizations
  -

  - –

- Modeling:

  - Random Forest
  - XG-Boosting
  - Poisson Generalized Linear Regression with Lasso
  - Poisson Generalized Linear Regression with Elastic Net
  - K-Nearest Neighbor Regression

- Uncertainty Quantification
- Result Communication

Data Importing and Cleaning

Set Up Environment

```
# calling required packages

library(readr)

library(tidyverse)

library(tidymodels)

library(randomForest)

library(parsnip)

library(workflows)

library(parallel)

library(doParallel)

library(yardstick)

library(ggplot2)

library(xgboost)

library(yardstick)

library(kknn)
library(poissonreg)

library(car)

tidymodels_prefer()

theme_set(theme_bw())


# To prevent connection lost

unregister_dopar <- function() {

  env <- foreach:::.foreachGlobals

  rm(list=ls(name=env), pos=env)

}


unregister_dopar()
```

Read in Data and Cleaning

In this project, we utilized two datasets, 11_Direct_Investment_related_indicators.csv and 24_Climate_related_Diasters_Frequency.csv. The former dataset contains CO2 emission for each country from years 2005 to 2018, for each country, sectors, and relative to either export or domestic of either domestically or foreign controlled enterprises. The later dataset contains disaster count from years 1980 to 2019 for each country and by different types of diaster.

Cleaning Dataset 11_Direct_Investment-related_Indicators

In order to organize the raw data, four different dictionaries were created and selected our variables of interest (Country, Year 2005 to 2018, Sector and Enterprise type).

The raw data consisted of 51 sectors many were related to each other. Sectors that deemed to be related or in the same job family were combined. For example, “Mining and extraction of energy producing products”,“Mining and quarrying of non-energy producing products”, and “Mining support service activities” were combined under mining.

```r
####### Cleaning data set 11_Direct_Investment-related_Indicators #######
# read the investment indicators dataset
investment_indicators <- read_csv("11_Direct_Investment-related_Indicators.csv")
```

```
## Rows: 14190 Columns: 25
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (11): Country, ISO2, ISO3, Indicator, Unit, Source, CTS Code, CTS Name, ...
## dbl (14): 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, ...
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

```r
# subset the dataset to discard unwanted columns
investment_indicators <- investment_indicators %>%
  select(Country, ISO2, ISO3, Indicator, Unit, `CTS Code`, `CTS Name`, Sector,
         `2005`:`2018`)
investment_indicators %>% glimpse()
```

```
## Rows: 14,190
## Columns: 22
## $ Country    <chr> "Argentina", "Argentina", "Argentina", "Argentina", "Argent~
## $ ISO2       <chr> "AR", "AR", "AR", "AR", "AR", "AR", "AR", "AR", "AR", "AR",~
## $ ISO3       <chr> "ARG", "ARG", "ARG", "ARG", "ARG", "ARG", "ARG", "ARG", "AR~
## $ Indicator  <chr> "CO2 emissions in exports of domestic controlled enterprise~
## $ Unit       <chr> "Metric tons of CO2", "Metric tons of CO2", "Metric tons of~
## $ ‘CTS Code‘ <chr> "ECBIXD", "ECBIXD", "ECBIXD", "ECBIXD", "ECBIXD", "ECBIXD",~
## $ ‘CTS Name‘ <chr> "CO2 Emissions in Exports; Of which: Exports of Domestic Co~
## $ Sector     <chr> "Accommodation and food services", "Agriculture, forestry a~
## $ ‘2005‘     <dbl> 67.8725926, 1954.7926108, 36.5992019, 50.0725812, 148.75835~
## $ ‘2006‘      <dbl> 81.3704731, 1720.0380663, 45.1554008, 79.4180588, 173.56809~
## $ ‘2007‘      <dbl> 92.426562, 2341.137818, 46.501763, 75.346349, 126.714502, 4~
## $ ‘2008‘      <dbl> 85.907408, 2570.054999, 51.914479, 79.524506, 171.180549, 4~
## $ ‘2009‘      <dbl> 71.2114336, 1433.6555943, 32.8004184, 67.4562106, 158.40492~
## $ ‘2010‘      <dbl> 83.558460, 2022.159504, 32.363177, 89.201459, 169.111172, 2~
## $ ‘2011‘      <dbl> 79.534370, 2244.337944, 31.128261, 91.336366, 203.536455, 2~
## $ ‘2012‘      <dbl> 64.993731, 1916.677540, 26.751951, 82.703299, 222.669928, 2~
## $ ‘2013‘      <dbl> 57.385115, 1874.305223, 26.115516, 77.160579, 210.962388, 1~
## $ ‘2014‘      <dbl> 59.141641, 1442.325438, 26.143714, 74.335338, 229.553929, 1~
## $ ‘2015‘      <dbl> 56.846428, 1216.817160, 22.016188, 60.709666, 219.951865, 1~
## $ ‘2016‘      <dbl> 14.668454, 1229.891696, 25.395741, 47.939953, 223.918129, 1~
## $ ‘2017‘      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ ‘2018‘      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

```
########################################################################
# creating dictionary for the country with ISO2 and ISO3.
Country_label <- investment_indicators %>% select(Country, ISO2, ISO3) %>% distinct()


# creating dictionary for the sectors.
Sectors_label <- investment_indicators %>% select(Sector) %>% distinct()
Sectors_label <- Sectors_label %>%
  mutate(label = sapply(1:length(Sector), function(i) paste0("sector_", i)))


# creating dictionary for the 8 types of CO2 emissions.
CTS_label <- investment_indicators %>% select(`CTS Code`, `CTS Name`) %>% distinct()


# creating dictionary for unit
unique(investment_indicators$Unit)
```

```
## [1] "Metric tons of CO2"
## [2] "Metric tons of CO2 per million US$ of output"

## [3] "Metric tons per million US$"
```

```
Unit_label <- matrix(c("Metric tons of CO2", "MT",
                       "Metric tons of CO2 per million US$ of output", "MTPMDO",
                       "Metric tons per million US$", "MTPMD"),
                     ncol = 2, byrow = TRUE) %>% as.data.frame()
colnames(Unit_label) <- c("Unit", "Unit_abb")

##########################################################################
# match Units in data set with its abbreviations, using the dictionary
# match Sectors in data set with its abbreviations, using the dictionary
investment_indicators <- investment_indicators %>%
  mutate(Unit_abb = sapply(Unit,
                           function(x) Unit_label$Unit_abb[match(x, Unit_label$Unit)])) %>%
  relocate(Unit_abb, .before = Unit) %>%
  mutate(Sector_abb = sapply(Sector,
                             function(x) Sectors_label$label[match(x, Sectors_label$Sector)]))
  relocate(Sector_abb, .before = Sector)

# select the columns of interest, save it temporarily.
tmp <- investment_indicators %>% select(ISO3, `CTS Code`, Sector_abb,
                                        `2005`:`2018`)
# pivot longer the year and pivot wider the indicators, save as emission_df
emission_df <- tmp %>% pivot_longer(cols = `2005`:`2018`,
                                    names_to = "Year",
                                    values_to = "CO2_emission") %>%
  pivot_wider(names_from = `CTS Code`,
              values_from = CO2_emission) %>%
  select(-ECBIPF, -ECBIPD, -ECBIFR, -ECBIFF) %>%
  mutate_all(~ ifelse(is.na(.), 0, .))


#########################################################################
# Combining some sectors that have overlaps
emission_df <- emission_df %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_1",
             "sector_14"),
             "Food", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_18",
             "sector_42", "sector_43", "sector_44"),
             "Mining", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_2",
             "sector_40"),
             "Agriculture", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_3",
             "sector_37"),
             "Recreation", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_4",
             "sector_12","sector_22", "sector_34"),
             "RawMaterial", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_5",
             "sector_46","sector_38"),
             "Pharma", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_6",
             "sector_11", "sector_39", "sector_50"),
             "Energy", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_7",
             "sector_10", "sector_30"),
             "Electronic", Sector_abb)) %>%
```

```r
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_8"),
             "Construction", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_9"),
             "Education", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_13",
             "sector_20", "sector_28", "sector_25", "sector_35",
             "sector_45", "sector_47", "sector_48", "sector_15",
             "sector_16", "sector_27", "sector_26"),
             "HumanActivities", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_17",
             "sector_21"),
             "Machinery", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_19",
             "sector_23", "sector_32", "sector_33", "sector_36",
             "sector_41", "sector_49", "sector_51"),
             "Transportation", Sector_abb)) %>%
  mutate(Sector_abb = if_else(Sector_abb %in% c("sector_24",
             "sector_29", "sector_31"),
             "Manufacturing", Sector_abb)) %>%
  group_by(ISO3, Year, Sector_abb) %>%
  reframe(across(c(ECBIXD, ECBIXF, ECBIOD, ECBIOF), sum))

##########################################################################
# remove data that will not be used again
rm(tmp, investment_indicators)

### Summary: all dictionaries have ending with "_label".
### We will be using emission_df for further analysis.
```

Cleaning Dataset 24_Climate-related_Diasters_Frequency

The following shows the left join of the second data set "Climate-related_Diasters_Frequency" by country and year.

```
####### Cleaning dataset 24_Climate-related_Diasters_Frequency#######
disasters_frequency <- read_csv("24_Climate-related_Disasters_Frequency.csv")
```

```
## Rows: 968 Columns: 52
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (10): Country, ISO2, ISO3, Indicator, Unit, Source, CTS Code, CTS Name, ...
## dbl (42): 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# CTS Code for climate related disasters frequency is ECCD.
# Adding ECCD to CTS_label dictionary.
# Obtain the CTS code and name for disasters
ECCD_code <- disasters_frequency$`CTS Code` %>% unique()
ECCD_name <- disasters_frequency$`CTS Name` %>% unique()
# adding the CTS code and name for disasters to CTS dictionary
ECCD_df <- data.frame(ECCD_code, ECCD_name)
names(ECCD_df) <- c("CTS Code", "CTS Name")
CTS_label <- rbind(CTS_label, ECCD_df)



# Subset the dataset to discard unwanted data.
# Narrow down the data to focus on years from 2005 to 2018 (range of years
# in the emission_df).
```

```
disasters_frequency <- disasters_frequency %>%
  select(ISO3, Indicator, `CTS Code`, `2005`:`2018`)

# mutate the Indicator column to trim away repeated information
disasters_frequency <- disasters_frequency %>%
  mutate(Indicator_abb =
           str_extract(Indicator, "(?<=: ).*")) %>%
  relocate(Indicator_abb, .after = Indicator) %>%
  select(-Indicator) %>%
  rename(Indicator = Indicator_abb)

# filter to only total count of disasters.
disasters_frequency <- disasters_frequency %>%
  filter(Indicator == "TOTAL") %>%
  select(-Indicator)

# replace all na with zeros
disasters_frequency <- disasters_frequency %>%
  mutate_all(~replace_na(., 0)) %>% glimpse()
```

```
## Rows: 215
## Columns: 16
## $ ISO3       <chr> "AFG", "ALB", "DZA", "ASM", "AGO", "AIA", "ATG", "ARG", "AR~
## $ ‘CTS Code‘ <chr> "ECCD", "ECCD", "ECCD", "ECCD", "ECCD", "ECCD", "ECCD", "EC~
## $ ‘2005‘     <dbl> 11, 2, 3, 1, 1, 0, 0, 1, 0, 4, 3, 0, 0, 1, 12, 0, 1, 3, 2, ~
## $ ‘2006‘     <dbl> 12, 0, 1, 0, 1, 0, 0, 1, 0, 8, 1, 0, 0, 0, 7, 0, 0, 1, 0, 0~
## $ ‘2007‘     <dbl> 7, 2, 7, 0, 2, 0, 0, 3, 0, 2, 2, 0, 0, 1, 5, 0, 1, 1, 1, 1,~
## $ ‘2008‘     <dbl> 3, 0, 1, 0, 1, 0, 1, 1, 0, 5, 1, 0, 0, 2, 5, 0, 0, 2, 2, 1,~
## $ ‘2009‘     <dbl> 4, 1, 2, 0, 3, 0, 0, 5, 0, 6, 4, 1, 0, 0, 6, 0, 0, 0, 0, 3,~
## $ ‘2010‘     <dbl> 4, 1, 0, 0, 3, 0, 1, 1, 0, 8, 0, 1, 0, 0, 6, 2, 0, 3, 1, 1,~
## $ ‘2011‘     <dbl> 4, 0, 1, 0, 4, 0, 0, 0, 0, 3, 0, 0, 0, 1, 5, 0, 0, 2, 0, 1,~
## $ ‘2012‘     <dbl> 10, 1, 2, 0, 1, 0, 0, 3, 0, 2, 1, 1, 0, 1, 5, 0, 1, 1, 0, 1~
## $ ‘2013‘     <dbl> 5, 0, 0, 0, 1, 0, 0, 2, 2, 3, 1, 0, 0, 1, 3, 0, 1, 2, 0, 2,~
## $ ‘2014‘     <dbl> 3, 0, 0, 0, 0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 4, 0, 1, 1, 0, 0,~
## $ ‘2015‘     <dbl> 5, 3, 2, 0, 3, 0, 0, 4, 0, 8, 0, 0, 0, 1, 7, 0, 0, 1, 1, 0,~
## $ ‘2016‘     <dbl> 4, 1, 0, 0, 4, 0, 0, 2, 1, 3, 1, 0, 0, 1, 3, 0, 0, 2, 1, 0,~
## $ ‘2017‘     <dbl> 5, 2, 1, 0, 2, 1, 1, 4, 0, 3, 2, 0, 0, 1, 5, 1, 1, 0, 0, 0,~
## $ ‘2018‘     <dbl> 6, 1, 1, 0, 1, 0, 0, 5, 1, 3, 0, 0, 0, 0, 4, 0, 1, 2, 0, 0,~
```

```
disasters_df <- disasters_frequency %>%
  pivot_longer(cols = `2005`:`2018`,
               names_to = "Year",
               values_to = "Count") %>%
  pivot_wider(names_from = `CTS Code`,
              values_from = Count)
```

Merge the Two Datasets

```
#######Combining the two data sets#######
df <- emission_df %>% left_join(disasters_df, by = c("ISO3", "Year"))
df %>% mutate_at("Year", as.factor)
```

```
## # A tibble: 11,564 x 8
##    ISO3  Year  Sector_abb       ECBIXD  ECBIXF ECBIOD  ECBIOF  ECCD
##    <chr> <fct> <chr>             <dbl>   <dbl>  <dbl>   <dbl> <dbl>
##   1 ARG   2005  Agriculture     1955.     5.86  6476.    12.8       1
##   2 ARG   2005  Construction       0      0      6817.  531.       1
##   3 ARG   2005  Education          3.63   0.158  2594.    7.43     1
##   4 ARG   2005  Electronic        83.1   57.9    1193.  634.       1
##   5 ARG   2005  Energy           604.   226.    11497. 4433.       1
##  6 ARG   2005  Food            3284.   360.    16649. 1198.       1
##  7 ARG   2005  HumanActivities  116.    79.3    9175.  920.       1
##  8 ARG   2005  Machinery         89.0  122.     1883.  914.       1
##  9 ARG   2005  Manufacturing    474.    40.3    5809.  243.       1
## 10 ARG   2005  Mining            16.6   10.7    1403.  280.       1
## # ... with 11,554 more rows
```

```
# remove data not going to be used
rm(ECCD_df, disasters_frequency)

#####More Manipulation to the Dataset#####
# relocating Year to the first
df <- df %>% relocate(Year, .before = Sector_abb)
# replace all na with 0
df <- df %>% mutate_all(~ ifelse(is.na(.), 0, .))
# renaming the branches
df <- df %>% rename(Export_Domestics = ECBIXD,
                           Export_Foreign = ECBIXF,
                           Output_Domestic = ECBIOD,
                           Output_Foreign = ECBIOF,
                           Disaster_Frequency = ECCD)

# pivot longer the branches
df_long <- df %>% pivot_longer(cols = Export_Domestics:Output_Foreign,
                           names_to = "Category",
                           values_to = "Value")
# pivot wider the combinations of branches and category
df_wide <- df_long %>% pivot_wider(names_from = c("Sector_abb", "Category"),
                                   values_from = Value,
                                   values_fill = 0,
                                   names_glue = "{Category}_{Sector_abb}")

# convert year from character to factor
df_wide <- df_wide %>%
  mutate(Year = as.factor(Year))

# save(df_wide, file = "df_wide.csv")
# save(df_long, file = "df_long.csv")
```

There are two tidy versions of the data sets saved under df_long and df_wide. Df_long depicts the four variables of interest and df_wide stratifies the type of enterprises by sectors.

Train Test Splitting

We decide to do an 80-20 split of the train and test data set.

```
set.seed(123457)
# train test split by assigning 80% to train and remaining 20% to test
df_split <- df_wide %>% initial_split(prop = 0.8)
train <- df_split %>% training()
test <- df_split %>% testing()

# dropping country indicator
train_sub <- train %>% select(-c(ISO3))
test_sub <- test %>% select(-c(ISO3))
```

Defining the Research Question

To examine the effects of domestically- versus foreign- controlled enterprises' CO2 emissions relative to sectors of activities performed for each country on climate related disasters. The models that we will approach are as the following: - (Rene) Poisson Generalized Linear Regression Model with Lasso Penalty - Lasso Penalty will be determined through cross validation. - (Rene) Poisson Generalized Linear Model with Elastic Net Penalty - cross validation to determine the best penalty rate - (Yuxin) Random Forest - pass along several possible combinations of hyper-parameters and perform cross validation - maximum depth, minimum number of samples at each split, n_estimators (4 for each) - parallel computing - (Jen) K-Nearest Neighbors Regression - cross validation to determine the best k - (Yuxin) XG-Boost - pass along several possible combinations of hyper-parameters and perform cross validation - maximum depth, minimum number of samples at each split, n_estimators (4 for each) - parallel computing

TARGET VARIABLE: Disaster_Frequency

graphics: - (Rene) line graph, count of disaster by year - color code by sector (t.s. plot) - (Rene) line graph of aggregated count of disasters cross years. - t.s. plot - (Jen) world map and co2 emission (optional) - aggregating the co2 emission across all years, and rank by country. - ranking is going to be done by color. (gradient color, red is worse, while green is better) - (Jen) co2 emission by country (do first, if time then do map) - (Jen) bar graphs of *df_long* category column, x_axis is year. - facet wrap by category - (Yuxin) scatter plot, x-axis will be co2 emission, y-axis will be count of disasters. - because count of disaster goes by year, then we will need to aggregate the co2 emission by country and by year. so we sum all of the sectors and types of co2 emissions.

Visualizations

Climate change is a global issue as it leads to higher natural disasters. The line graph depicts the total disaster event by year on a global level. Ever since the 1900s, natural disasters have been on the rise. Due to our incomplete data spanning from 2005 to 2008, this is a period of decline of disasters with 2005 having the most events.

```
# prep dataframe for plot
df_aggregate_2 <- df_wide %>%
  group_by(Year, ISO3) %>%
  reframe(diaster_count = first(Disaster_Frequency)) %>%
  group_by(Year) %>%
  summarize(total_disaster_frequency = sum(diaster_count))

# converting Year to numeric
df_aggregate_2$Year <- as.numeric(as.character(df_aggregate_2$Year))

# ggplot using aggregate dataset
ggplot(df_aggregate_2, aes(x = Year, y = total_disaster_frequency)) +
  geom_line(color = "blue") +
  geom_point(color = "blue") +
  labs(title = "Total Disaster Frequency by Year ",
       x = "Year",
       y = "Sum Disaster Frequency")
```

Carbon dioxide is a greenhouse gas that traps heat in the atmosphere, leading to global warming and climate change. It is produce primarily from human activities such as burning fossil fuels for energy, transportation, and other industry sectors. Below is a map of CO2 emissions produce in each country with red being the higher end of the spectrum and white being little CO2 produced.

```
# load world data for map
library(maps)
world <- map_data("world")

# data prepping for the map
# rename the Country to match Country names stored in the world data
Country_label <- Country_label %>%
  mutate(Country = if_else(Country == "China, P.R.: Hong Kong", "China", Country)) %>%
  mutate(Country = if_else(Country == "China, P.R.: Mainland", "China", Country)) %>%
  mutate(Country = if_else(Country == "Taiwan Province of China", "Taiwan", Country)) %>%
  mutate(Country = if_else(Country == "Korea, Rep. of", "South Korea", Country)) %>%
```

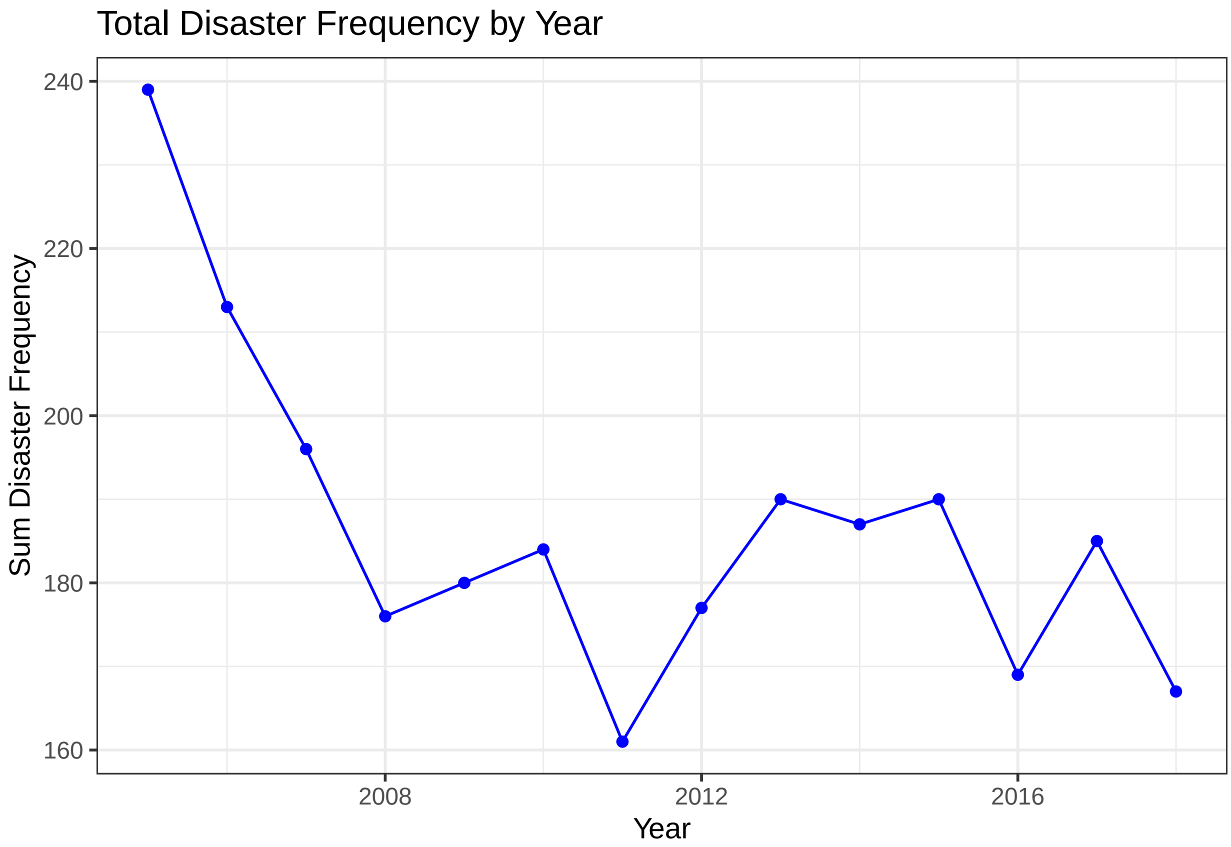


Figure 1: Diaster by year. Count of total disaster decreased from years 2005 to 2008. This is a decreasing trend because of we are focusing on a relatively short time span. If one were to look at the count of diaster at a bigger time frame, one should see a general upward trend.

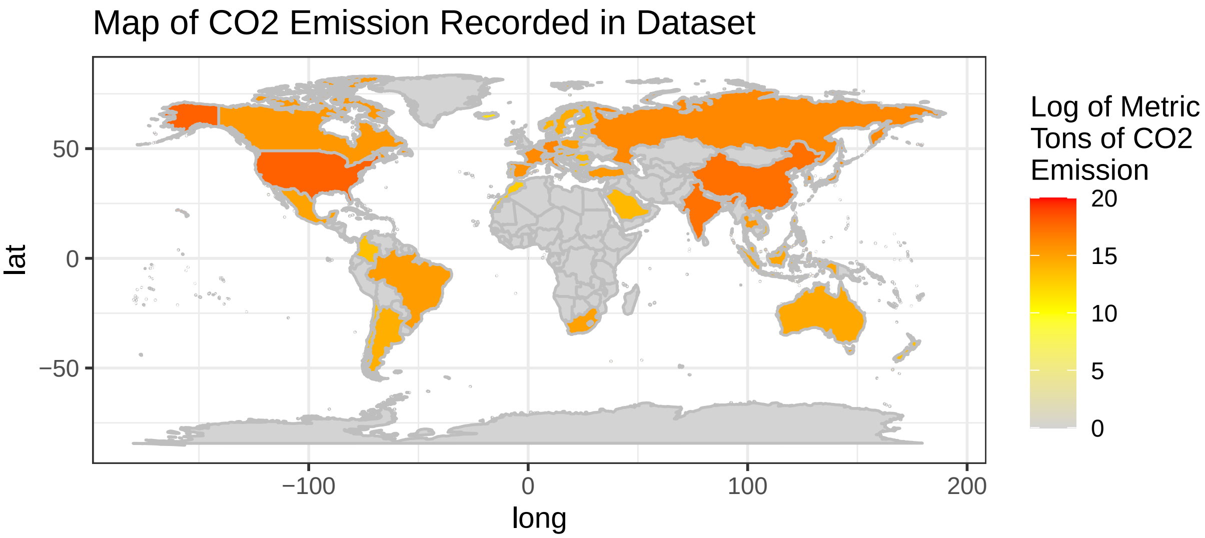
```
  mutate(Country = if_else(Country == "Netherlands, The", "Nether", Country)) %>%
  mutate(Country = if_else(Country == "Poland, Rep. of", "Poland", Country)) %>%
  mutate(Country = if_else(Country == "Russian Federation", "Russia", Country)) %>%
  mutate(Country = if_else(Country == "Slovak Rep.", "Slovakia", Country)) %>%
  mutate(Country = if_else(Country == "Slovenia, Rep. of", "Slovenia", Country)) %>%
  mutate(Country = if_else(Country == "Estonia, Rep. of", "Estonia", Country)) %>%
  mutate(Country = if_else(Country == "Czech Rep.", "Czech Republic", Country)) %>%
  mutate(Country = if_else(Country == "Croatia, Rep. of", "Croatia", Country)) %>%
  mutate(Country = if_else(Country == "United States", "USA", Country))
# create data frame for map
bind <- df_long %>%
  group_by(ISO3, Year, Category) %>%
  reframe(Value = sum(Value)) %>%
  left_join(Country_label, by="ISO3") %>%
  select(c(Country,Value)) %>%
  group_by(Country) %>% reframe(Value = sum(Value))

# combine the two China values
bind <- bind %>% group_by(Country) %>% summarize(Value = sum(Value))

# join the two datasets
world <- world %>% full_join(bind, by = c("region" ="Country"))
world <- world %>% mutate_all(~ ifelse(is.na(.), 0, .))
```

This data preparation is necessary for the map to join in order to include all the data points.

```
# plot the map, set emission as fill color.
ggplot(world, aes(long, lat, group=group, fill = log(Value+1))) +
  geom_polygon(color="gray") +
  scale_fill_gradient2(low = "lightgray", mid = "yellow", high = "red",
                                   midpoint = 10, limits = c(0, 20)) +
  coord_fixed() +
  ggtitle("Map of CO2 Emission Recorded in Dataset") +
  labs(fill='Log of Metric \nTons of CO2 \nEmission')
```



It is depicted that developed countries, such as the United States, China, Russia, and parts of Europe, produces more CO2 emission than underdeveloped countries.

After seeing the countries with prolific production of CO2 emission, we have suspects for those that may have higher frequency of natural disasters. The graph below counts the number of disasters by year colored based on countries. This figure confirms our suspicions of more disasters will occur in developed countries.

The top 3 countries with the most disasters are USA, China, and India.

```
# prepare data for the line graph.
# obtain 10 countries with the most diaster count for every year.
df_aggregate_1 <- df_long %>%
  group_by(Year, ISO3) %>%
  reframe(total_disaster = first(Disaster_Frequency)) %>%
  group_by(Year) %>%
  slice_max(total_disaster, n = 10)
```

```
# convert Year to numeric
df_aggregate_1$Year <- as.numeric(as.character(df_aggregate_1$Year))


# ggplot using aggregate dataset
ggplot(df_aggregate_1, aes(x = Year, y = total_disaster, color = ISO3)) +
  geom_line() +
  geom_point() +
  labs(title = "Count of Disaster by Year for Top 10 Countries with Most Diasters",
       x = "Year",
       y = "Sum of Disaster",
       color = "Country")
```

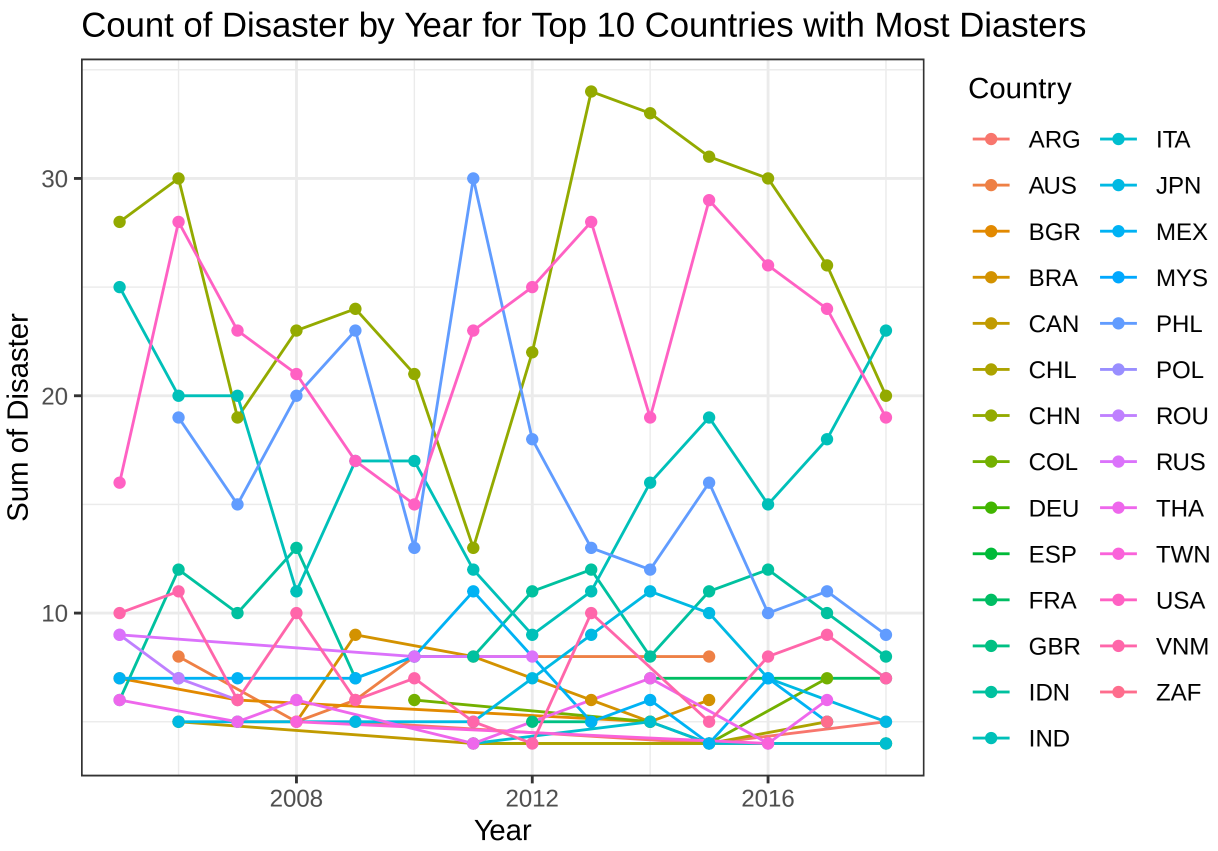


Figure 2: Count of disaster by year for countries with most diasters. This plot includes the 10 countries with most diaster count for every year. As can be seen that some country, such as China, India, and USA are consistently ranked as the top 10 for every year. These three countries have one characteristic in common, that their land area are large in comparison with other countries.

In the scatterplot below, we explored the countries with most disaster by the log of CO2 emissions by metric tons. This graph only includes countries with greater than 10 events and is faceted by year. As you can see from this faceted scatter plot, there is a lack of data for 2017 and 2018. USA, China, and India consistently is in the top producers of carbon dioxide correlating with higher count of disasters by year.

```r
# optimize data for scatter plot
scatterplot_df <- df_long %>% group_by(ISO3, Year) %>%
  reframe(emission = sum(`Value`),
          Disaster_Frequency = first(Disaster_Frequency))

# convert country (IOS3) as factor
scatterplot_df$ISO3 <- as.factor(scatterplot_df$ISO3)

# plot the scatter plot
scatterplot_df %>% ggplot(aes(x = log(emission + 0.0001),
                              y = Disaster_Frequency)) +
  geom_point() +
  geom_point(data = scatterplot_df[scatterplot_df$Disaster_Frequency>10, ],
             aes(x = log(emission + 0.0001),
                 y = Disaster_Frequency,
                 color = ISO3)) +
  facet_wrap(~Year) +
  ylab("Disaster Count") +
  xlab("Log of Emission (Log of Metric Tons of CO2)") +
  ggtitle("Emission versus Disaster Count for Each Country")
```

The four types of enterprises are exports of domestic controlled enterprises, exports of foreign controlled multinational enterprises, output of domestic controlled enterprises, and output of foreign controlled multinational enterprises.

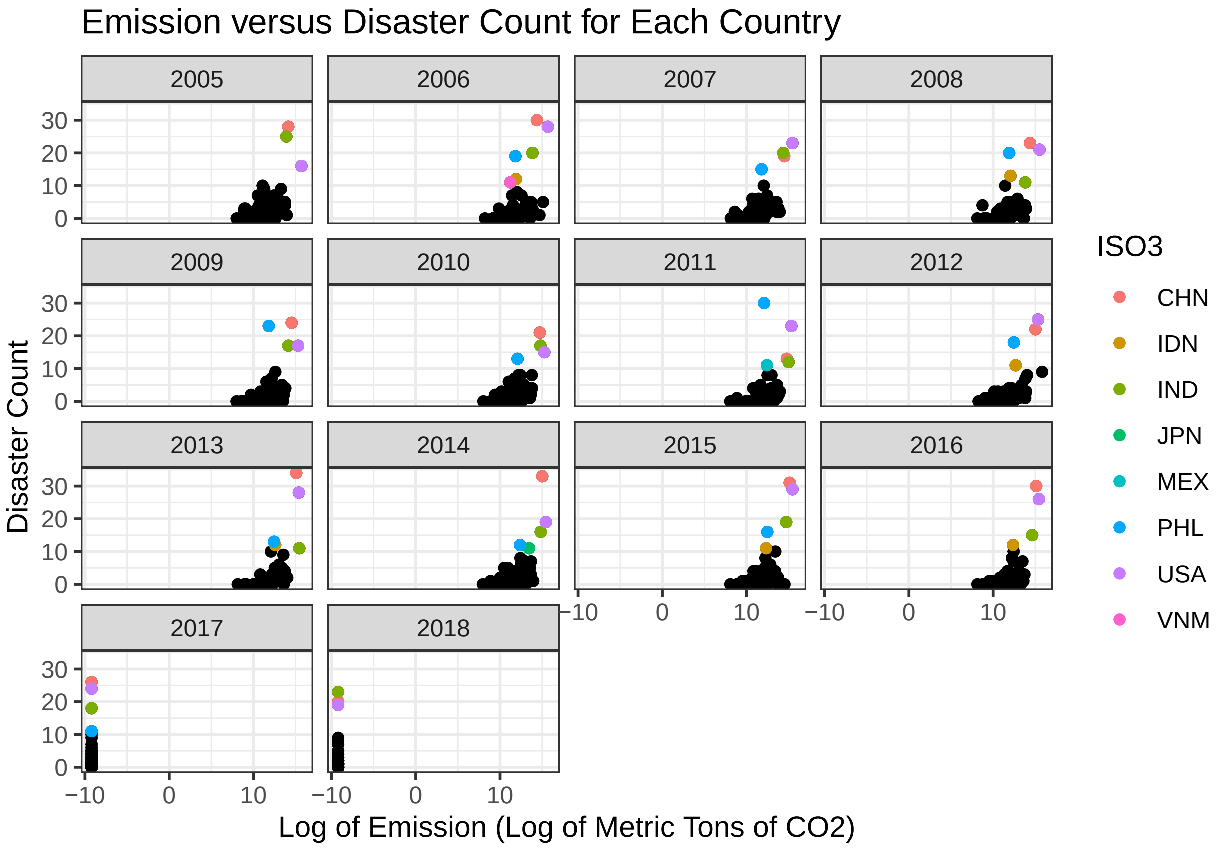


Figure 3: Emission versus diaster count for each country. In this plot, country with more than 10 diasters per year are highlighted. There appears to be a linear relationship between the log of emission and disaster count. Countries such as China and USA consistently have greater than 10 diasters for every year.

Exports are shipment of goods or services from one country to another. Output refers to the production of goods or services within a particular industry or sector. Domestic refers to products or services that are produced or provided within a particular country and intended for use or consumption within that country. Foreign refers to products or services that are produced or provided outside of one's own country and intended for use or consumption in other countries. The type of enterprises will be a combination of either export or output and foreign or domestic. For example, exports of domestic controlled enterprises ('Export_Domestics') are goods that are produced by companies that are headquartered outside the country where they are sold.

The stacked bar graph depicts the CO2 emissions produced by the category of enterprises.

```r
# prep the dataframe for bar plot.
df_agg <- df_long %>%
  group_by(ISO3, Category) %>%
  reframe(emission = sum(`Value`)) %>%
  mutate_if(is.character, as.factor) %>%
  group_by(Category) %>%
  arrange(desc(emission)) %>%
  slice(1:10)


# CO2 by top 10 country and fill by category.
ggplot(df_agg,aes(x = ISO3, y = emission, fill = Category)) +
  geom_bar(stat = "identity")  +
  # facet_wrap(~Category)+
  labs(
    title = "CO2 emissions by Category of Enterprise for Each Country",
    x = "Country",
    y = "Metric Tons of CO2 Emission"
  )
```

In developed countries, output of domestic controlled enterprises produce the most CO2 emissions

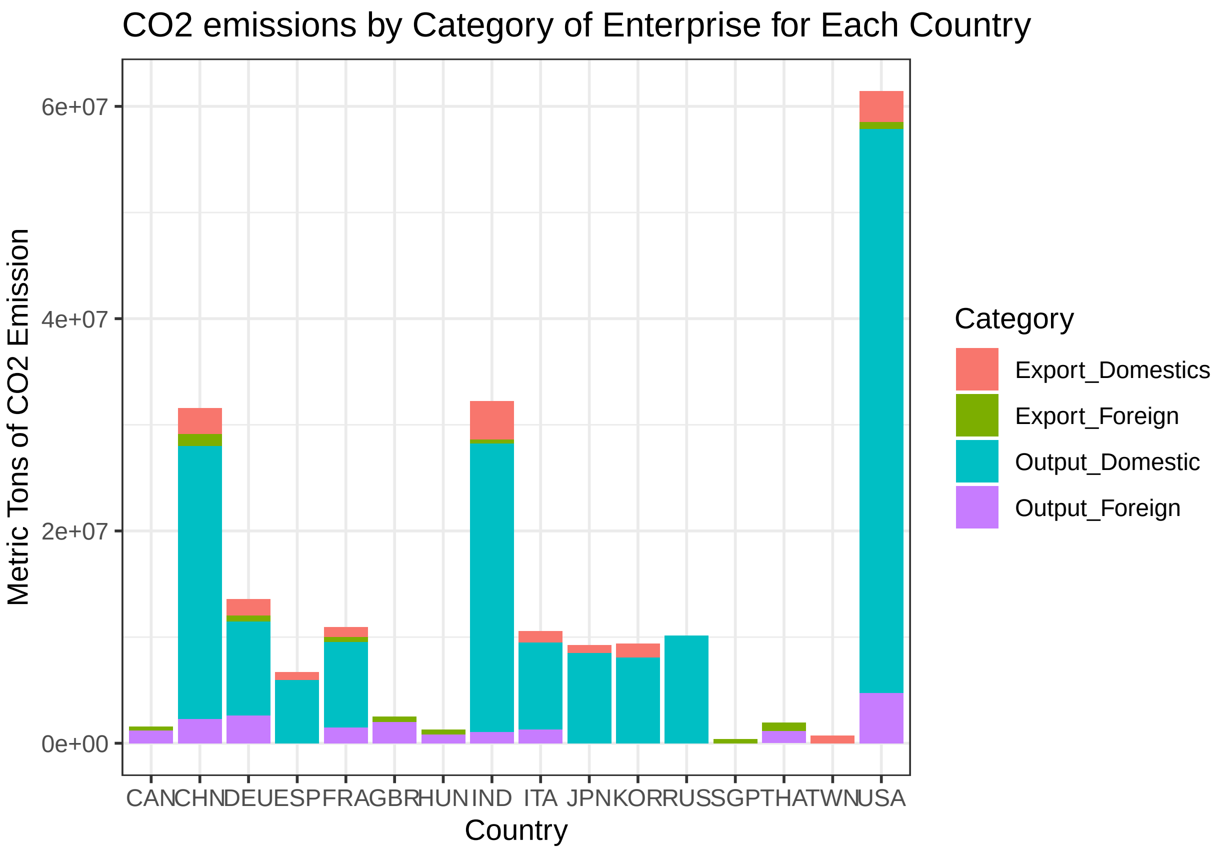


Figure 4: CO2 emissions by category of enterprise for each country. USA seems to come in the first place in terms of CO2 emission, followed by India and China. The emissions of China and India are relatively the same. As a general trend, Output Domestic seems to be the major contributor of CO2 emission.

by metric tons in comparison to other categories. Therefore, the most CO2 produced are from the goods or services produced by companies that are owned and controlled within the country where they operate. This indicates countries like the USA are producing goods that feeds back into its own economy.

Modeling

The five models we have chosen are * Random Forest * XG-Boosting * Poisson Generalized Linear Regression with Lasso * Poisson Generalized Linear Regression with Elastic Net * K-Nearest Neighbor Regression

Random Forest

In modeling Random Forest, we will attempt with tuning the hyper-parameters including number of predictors at each split (mtry), number of trees for each ensemble (trees), and minimum number of data points at each split (min_n). The tuning is performed on three levels for each of the hyper-parameters, in total of $3^3 = 27$ possible combinations. A 10-fold cross validation is performed twice to avoid sampling bias. Hence, in total 540 models will be evaluated to obtain the best model. Further, to improve the modeling, categorical variables will be encoded into dummy variables and all numerical predictors will be normalized. Metrics used to evaluate the models are Mean Absolute Error (mae), Root Mean Squared Error (rmse), and residual Squared (rsq), each are a measure of deviation or error of predicted values from observed values.

```
# set the seed
set.seed(123457)
# define parsnip, set random forest with mtry, trees, and min_n hyperparameters
# as tuning. To be tuned later.
parsnip_RF <- rand_forest(mtry = tune("mtry"),
                          trees = tune("trees"),
                          min_n = tune("min_n")) %>%
```

```r
  set_engine('randomForest') %>%
  set_mode('regression') #set random forest to regression

# define random forest recipe, code all nominal predictors dummy variables and
# normalize all numeric predictors.
RF_recipe <-
  recipe(formula = Disaster_Frequency ~ ., data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

# define workflow
workflow_RF <- workflow() %>%
  add_model(parsnip_RF) %>%
  add_recipe(RF_recipe)

# define hyperparameter tuning grid to be experimented
hyperparam_tune_grid <- crossing(min_n = seq(10, 50, by = 20),
                                 mtry = seq(10, 50, by = 20),
                                 trees = c(100, 500, 1000))
# define cross validation
rf_cv <- train_sub %>% vfold_cv(v = 10, times = 2)
# define metrics to be used
RF_metrics <- metric_set(yardstick::rmse, mae, rsq)
```

The following performs parallel processing for the hyper-parameter tuning and cross validation.

```r
# assigning resources
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
time1 <- Sys.time() #save current system time
```

```r
#run the 10 fold cross validation for each combination of hyperparameters
rf_tuning <- workflow_RF %>%
  tune_grid(resamples = rf_cv,
            grid = hyperparam_tune_grid,
            metrics = RF_metrics) %>%
  collect_metrics()

time2 <- Sys.time() #save current system time

(diff <- time2 - time1) #obtain total run time

stopCluster(cl)

save(rf_tuning, file = "RandomForest_Prediction.rds")
```

The following retrieves run result of parallel processing. Both the mean absolute error and root mean squared error suggests that hyper-parameter combination of mtry = 30, trees = 100, and min_n = 10 yields the best model. The residual squared supports that hyper-parameter combination of mtry = 10, trees = 500, and min_n = 10 will yield the best model. Hence, we proceed with the former hyper-parameter combination

```r
RF_prediction <- load("RandomForest_Prediction.rds")

# selecting the best hyperparameter combination
Best_result1 <- rf_tuning %>%
  filter(!(.metric == "rsq")) %>%
  group_by(.metric) %>%
  slice_min(mean)
Best_result2 <- rf_tuning %>%
  filter(.metric == "rsq") %>%
  group_by(.metric) %>%
  slice_max(mean)
Best_result3 <- Best_result1 %>% bind_rows(Best_result2)
Best_result3
```

```
## # A tibble: 3 x 9
## # Groups:   .metric [3]
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <dbl> <dbl> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    30   100    10 mae     standard   1.58     10  0.0846 Preprocessor1_Model04
## 2    30   100    10 rmse    standard   2.51     10  0.243  Preprocessor1_Model04
## 3    10   500    10 rsq     standard   0.757    10  0.0432 Preprocessor1_Model02
```

By fitting the model with hyper-parameter combination suggested above, the model explains roughly 70 of the variation in the model. The mean of squared residuals (or mse) is 8.411. This implies that the fitted values are on average 8.411 away from the observed values.

```
# Fitting the train data with best hyperparameter combination
# Since both rmse and rsq_trad supports mtry = 50, trees = 1000,
# and min_n = 10, we fit that to the train set.
parsnip_RF_train <- rand_forest(mtry = 30,
                                trees = 100,
                                min_n = 10) %>%
  set_engine('randomForest') %>%
  set_mode('regression')

# define workflow
workflow_RF_train <- workflow() %>%
  add_model(parsnip_RF_train) %>%
  add_recipe(RF_recipe)


# fit to the train data
RF_fit <- workflow_RF_train %>% fit(data = train_sub)
# display result
RF_rsq <- RF_fit$fit$fit$fit$rsq %>% mean()
RF_mae <- mean((train_sub$Disaster_Frequency - RF_fit$fit$fit$fit$predicted)^2)
RF_rmse <- mean((train_sub$Disaster_Frequency - RF_fit$fit$fit$fit$predicted)^2) %>%
  sqrt()
cat("Metrics of Random Forest: rsq =", RF_rsq, ", mae =", RF_mae,
    ", and rmse=", RF_rmse, ".")
```

```
## Metrics of Random Forest: rsq = 0.6973855 , mae = 8.411428 , and rmse= 2.900246 .
```

By performing prediction on both the train set and test set, we conclude that the performance of our model is fair. The prediction on the train set scored an rsq of 0.837 with rmse of 2.183 and mae of 1.059. The performance on the test set is lower in comparison to train set, roughly 0.765(rsq). The mae and rmse of the prediction on test set is only a fraction bigger than the prediction on train set. The model performance on the test set is lower than on the train set.

```
# prediction on the train set
RF_train_predict <- RF_fit %>% predict(train_sub)
RF_train_predict_metric <- RF_train_predict %>%
  mutate(truth = train_sub$Disaster_Frequency, estimate = .pred) %>%
  RF_metrics(truth = truth, estimate = .pred)
RF_train_predict_metric
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       2.18
## 2 mae     standard       1.06
## 3 rsq     standard       0.837
```

```
# predict the test set and evaluate
RF_test_predict <- RF_fit %>% predict(test_sub)
RF_test_predict_metric <- RF_test_predict %>%
  mutate(truth =test_sub$Disaster_Frequency, estimate = .pred) %>%
  RF_metrics(truth = truth, estimate = .pred)
RF_test_predict_metric
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       2.53
## 2 mae     standard       1.55
## 3 rsq     standard       0.765
```

XG-Boost

In modeling using Boosting, we will attempt to tune the hyper-parameters to improve model performance. The hyper parameter as depth of each trees (tree_depth), learning rate of XG-Boost (learn_rate), and minimum number of data points at each split (min_n). Three levels of each hyper-parameters are tried, giving $3^3 = 27$ possible combinations. A 10-fold cross validation is performed twice to avoid sampling bias. In total 540 model are being evaluated. Further more, to improve model performance, all nominal predictors are coded as dummy variables and all numerical predictors are normalized.

```
# set the seed
set.seed(123457)
# define parsnip, set random forest with mtry, trees, and min_n hyperparameters
# as tuning. To be tuned later.
parsnip_boost <- boost_tree(tree_depth = tune("tree_depth"),
                            learn_rate = tune("learn_rate"),
                            min_n = tune("min_n")) %>%
  set_engine('xgboost') %>%
  set_mode('regression')

# define recipe for boosting, one hot encode dummy variable for nominal predictors
# and normalize all numeric predictors.
boost_recipe <-
  recipe(formula = Disaster_Frequency ~ ., data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())


# define workflow
workflow_boost <- workflow() %>%
  add_model(parsnip_boost) %>%
  add_recipe(boost_recipe)

# define hyperparameter tuning grid to be experimented
hyperparam_tune_grid_boost <- crossing(min_n = seq(10, 50, by = 20),
                                       tree_depth = c(5, 10, 15),
                                       learn_rate = c(0.01, 0.05, 0.1))
# define cross validation
boost_cv <- train_sub %>% vfold_cv(v = 10, times = 2)
# define metrics to be used
boost_metrics <- metric_set(rmse, mae, rsq)
```

The following utilizes parallel processin to evaluate the models

```r
# assigning resources
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
time1 <- Sys.time() #save current system time

#run the 10 fold cross validation for each combination of hyperparameters
boost_tuning <- workflow_boost %>%
  tune_grid(resamples = boost_cv,
            grid = hyperparam_tune_grid_boost,
            metrics = boost_metrics) %>%
  collect_metrics()

time2 <- Sys.time() #save current system time

(diff <- time2 - time1) #obtain total run time

stopCluster(cl)

save(boost_tuning, file = "XGBoost_Prediction.rds")
```

After retrieving the run result from parallel processing, all three metrics rsq, mae and rmse supports hyper-parameter combination of min_n = 10, tree_depth = 15, and learn_rate = 0.1 yields the best model.

```r
Boost_prediction <- load("XGBoost_Prediction.rds")

Best_result4 <- boost_tuning %>%
  filter(!(.metric == "rsq")) %>%
  group_by(.metric) %>%
  slice_min(mean)
```

```
Best_result5 <- boost_tuning %>%
  filter(.metric == "rsq") %>%
  group_by(.metric) %>%
  slice_max(mean)
Best_result6 <- Best_result4 %>% bind_rows(Best_result5)
Best_result6
```

```
## # A tibble: 3 x 9
## # Groups:   .metric [3]
##   min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <dbl>      <dbl>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    10         15        0.1 mae     standard   1.83     10  0.0972 Preprocess~
## 2    10         15        0.1 rmse    standard   3.16     10  0.272  Preprocess~
## 3    10         15        0.1 rsq     standard   0.688    10  0.0519 Preprocess~
```

```
# Fitting the train data with best hyperparameter combination
# all metrics support the combination of min_n = 10,
# tree_depth = 10, and learn_rate = 0.1, we use it as our model for train set.
parsnip_Boost_train <- boost_tree(tree_depth = 10,
                                  learn_rate = 0.1,
                                  min_n = 10) %>%
  set_engine('xgboost') %>%
  set_mode('regression')

# define workflow
workflow_Boost_train <- workflow() %>%
  add_model(parsnip_Boost_train) %>%
  add_recipe(boost_recipe)

# fit to the train data
```

```
boost_fit <- workflow_Boost_train %>% fit(data = train_sub)
boost_fit$fit$fit$fit$evaluation_log %>% slice_min(training_rmse)
```

```
##    iter training_rmse
## 1:   15      2.703514
```

We see that the performance of boosting on the train set is roughly the same as Random Forest, however a little lower. This is also true for boosting performance on the test set.

```
# predict the train set and evaluate
boost_fit %>% predict(train_sub) %>%
  mutate(truth = train_sub$Disaster_Frequency, estimate = .pred) %>%
  boost_metrics(truth = truth, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       2.70
## 2 mae     standard       1.31
## 3 rsq     standard       0.814
```

```
# predict the test set and evaluate
boost_fit %>% predict(test_sub) %>%
  mutate(truth = test_sub$Disaster_Frequency, estimate = .pred) %>%
  boost_metrics(truth = truth, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       2.73
## 2 mae     standard       1.54
## 3 rsq     standard       0.758
```

Poisson Generalized Linear Regression Model with Lasso Penalty

```r
set.seed(123457)
# subsetting train dataset. excluding ISO3 and Year
train_sub <- train %>% select(-c(ISO3))
# Poisson Generalized Linear Regression Model with Lasso Penalty
poi_recipe <- recipe(Disaster_Frequency ~., data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
# defining model using poisson_reg()
poi_parsnip <-  poisson_reg(penalty = "lasso") %>%
  set_mode("regression") %>%
  set_engine("glm")
# defining workflow with model and recipe
poi_workflow <- workflow() %>%
  add_model(poi_parsnip) %>%
  add_recipe(poi_recipe)
```

```r
# Cross validation on train dataset
poi_result <- poi_workflow %>%
  fit_resamples(
    resamples = vfold_cv(train_sub, v = 10),
    metrics = metric_set(rmse, rsq, mae)
    )
```

```r
# showing results
poi_result %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator        mean     n      std_err .config
##   <chr>   <chr>            <dbl> <int>        <dbl> <chr>
## 1 mae     standard    181859.       10  181855.     Preprocessor1_Model1
## 2 rmse    standard   1477413.       10 1477396.     Preprocessor1_Model1
## 3 rsq     standard         0.268    10       0.0674 Preprocessor1_Model1
```

```
poi_fitted <- poi_workflow %>% fit(train_sub)
poi_fitted
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: poisson_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
##
## Call:  stats::glm(formula = ..y ~ ., family = stats::poisson, data = data)
##
## Coefficients:
##                     (Intercept)      Export_Domestics_Agriculture
##                        0.748860                         -0.309675
##       Export_Foreign_Agriculture       Output_Domestic_Agriculture
##                        0.229349                          0.934654
##       Output_Foreign_Agriculture     Export_Domestics_Construction
##                       -0.579262                         -0.061464
##      Export_Foreign_Construction      Output_Domestic_Construction
##                         0.095968                        -0.523704 
##      Output_Foreign_Construction       Export_Domestics_Education 
##                        -0.005035                        -0.267180 
##         Export_Foreign_Education        Output_Domestic_Education 
##                         0.254784                        -0.763702 
##         Output_Foreign_Education      Export_Domestics_Electronic 
##                        -0.083098                         0.676342 
##        Export_Foreign_Electronic       Output_Domestic_Electronic 
##                        -1.157403                        -0.108360 
##        Output_Foreign_Electronic          Export_Domestics_Energy 
##                         1.438619                        -0.213684 
##            Export_Foreign_Energy           Output_Domestic_Energy 
##                        -0.059051                         1.106137 
##            Output_Foreign_Energy            Export_Domestics_Food 
##                         0.238033                        -0.506095 
##              Export_Foreign_Food             Output_Domestic_Food 
##                        -0.242648                         1.645598 
##              Output_Foreign_Food Export_Domestics_HumanActivities 
##                         1.558337                         1.194722 
##   Export_Foreign_HumanActivities  Output_Domestic_HumanActivities 
##                        -0.104619                        -1.471897 
##   Output_Foreign_HumanActivities       Export_Domestics_Machinery 
##                        -0.182742                        -1.074674 
##         Export_Foreign_Machinery        Output_Domestic_Machinery 
##                         0.838602                         0.906224 
##         Output_Foreign_Machinery   Export_Domestics_Manufacturing 
##                        -0.906660                         1.110798 
##     Export_Foreign_Manufacturing    Output_Domestic_Manufacturing 
##                        -1.072520                        -0.647625 
##     Output_Foreign_Manufacturing          Export_Domestics_Mining 
##                         1.381830                          0.085472
##            Export_Foreign_Mining            Output_Domestic_Mining
##                         0.331654                          0.067772
##            Output_Foreign_Mining           Export_Domestics_Pharma
##                        -0.358383                         -0.916283
##            Export_Foreign_Pharma            Output_Domestic_Pharma
##                        -0.508127                         -1.269727
##            Output_Foreign_Pharma      Export_Domestics_RawMaterial
##                         0.323194                         -0.404092
##
## ...
## and 28 more lines.
```

Poisson Generalized Linear Regression Model with Elastic Net Penalty

```r
#  Poisson Generalized Linear Model with Elastic Net Penalty
set.seed(123457)
# defining recipe
poi_recipe_enp <- recipe(Disaster_Frequency ~., data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
# defining model using poisson_reg with elastic net penalty
poi_parsnip_enp <-  poisson_reg(penalty = "elastic_net") %>%
  set_mode("regression") %>%
  set_engine("glm")
# defining workflow with model and recipe
poi_workflow_enp <- workflow() %>%
  add_model(poi_parsnip_enp) %>%
  add_recipe(poi_recipe_enp)
```

```r
# cross validation on train dataset
poi_result_enp <- poi_workflow_enp %>%
  fit_resamples(
    resamples = vfold_cv(train_sub, v = 10),
    metrics = metric_set(rmse, rsq, mae),
    )
```

```r
# showing results
poi_result_enp %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator        mean     n      std_err .config
##   <chr>   <chr>            <dbl> <int>        <dbl> <chr>
## 1 mae     standard    181859.       10  181855.     Preprocessor1_Model1
## 2 rmse    standard   1477413.       10 1477396.     Preprocessor1_Model1
## 3 rsq     standard         0.268    10       0.0674 Preprocessor1_Model1
```

```r
#fitting model on train
poi_fitted_enp <- poi_workflow_enp %>% fit(train_sub)
poi_fitted_enp
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: poisson_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
```

```
##
## -- Model -----------------------------------------------------------------------
##
## Call:  stats::glm(formula = ..y ~ ., family = stats::poisson, data = data)
##
## Coefficients:
##                      (Intercept)      Export_Domestics_Agriculture
##                         0.748860                         -0.309675
##       Export_Foreign_Agriculture       Output_Domestic_Agriculture
##                         0.229349                          0.934654
##       Output_Foreign_Agriculture     Export_Domestics_Construction
##                        -0.579262                         -0.061464
##      Export_Foreign_Construction      Output_Domestic_Construction
##                         0.095968                         -0.523704
##      Output_Foreign_Construction        Export_Domestics_Education
##                        -0.005035                         -0.267180
##         Export_Foreign_Education         Output_Domestic_Education
##                         0.254784                         -0.763702
##         Output_Foreign_Education       Export_Domestics_Electronic
##                        -0.083098                          0.676342
##        Export_Foreign_Electronic        Output_Domestic_Electronic
##                        -1.157403                         -0.108360
##        Output_Foreign_Electronic           Export_Domestics_Energy
##                         1.438619                         -0.213684
##            Export_Foreign_Energy            Output_Domestic_Energy
##                        -0.059051                          1.106137
##            Output_Foreign_Energy             Export_Domestics_Food
##                         0.238033                         -0.506095
##              Export_Foreign_Food              Output_Domestic_Food
##                        -0.242648                          1.645598
##              Output_Foreign_Food  Export_Domestics_HumanActivities
##                         1.558337                          1.194722
##   Export_Foreign_HumanActivities   Output_Domestic_HumanActivities
##                        -0.104619                         -1.471897
##   Output_Foreign_HumanActivities        Export_Domestics_Machinery
##                        -0.182742                         -1.074674
##         Export_Foreign_Machinery         Output_Domestic_Machinery
##                         0.838602                          0.906224
##         Output_Foreign_Machinery    Export_Domestics_Manufacturing
##                        -0.906660                          1.110798
##     Export_Foreign_Manufacturing     Output_Domestic_Manufacturing
##                        -1.072520                         -0.647625
##     Output_Foreign_Manufacturing           Export_Domestics_Mining
##                         1.381830                          0.085472
##            Export_Foreign_Mining            Output_Domestic_Mining
##                         0.331654                          0.067772
##            Output_Foreign_Mining           Export_Domestics_Pharma
##                        -0.358383                         -0.916283
##            Export_Foreign_Pharma            Output_Domestic_Pharma
##                        -0.508127                         -1.269727
##            Output_Foreign_Pharma      Export_Domestics_RawMaterial
##                         0.323194                         -0.404092
##
## ...
## and 28 more lines.
```

```r
# poisson with lasso, prediction on test set
goal_metrics <- metric_set(rmse, rsq, mae)
# predict on test test
poi_prediction_1 <- poi_fitted %>% predict(test_sub) %>%
                    mutate(truth = test$Disaster_Frequency, estimate = .pred) %>%
                    goal_metrics(truth = truth, estimate = .pred)
poi_prediction_1
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       6.60
## 2 rsq     standard       0.141
## 3 mae     standard       2.55
```

```
# poisson with elastic net, preediction on test set
goal_metrics <- metric_set(rmse, rsq, mae)
# predict on test test
poi_prediction_2 <- poi_fitted_enp %>% predict(test_sub) %>%
                    mutate(truth = test$Disaster_Frequency, estimate = .pred) %>%
                    goal_metrics(truth = truth, estimate = .pred)
poi_prediction_2
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       6.60
## 2 rsq     standard       0.141
## 3 mae     standard       2.55
```

K-Nearest Neighbor Regression

K-Nearest Neighbor is a simple yet effective algorithm, but its performance can be impacted by the choice of the hyperparameter k. However, it can be computationally expensive depending on the size of the data. Here we have chosen k to be 5, 10, 20.

```r
# Set seed
set.seed(123457)
# k=5
knn_parsnip_5 <- nearest_neighbor() %>%
  set_mode("regression") %>%
  set_engine("kknn", neighbors = 5)

knn_recipe_5 <- recipe(Disaster_Frequency ~ .,
                       data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

knn_workflow_5 <- workflow() %>%
  add_model(knn_parsnip_5) %>%
  add_recipe(knn_recipe_5)

knn_5s <- knn_workflow_5 %>% fit(train_sub)


# k=10
knn_parsnip_10 <- nearest_neighbor() %>%
  set_mode("regression") %>%
  set_engine("kknn", neighbors = 10)

knn_recipe_10 <- recipe(Disaster_Frequency ~ .,
                        data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

```
knn_workflow_10 <- workflow() %>%
  add_model(knn_parsnip_10) %>%
  add_recipe(knn_recipe_10)

knn_10 <- knn_workflow_10 %>% fit(train_sub)



# k=20
knn_parsnip_20 <- nearest_neighbor() %>%
  set_mode("regression") %>%
  set_engine("kknn", neighbors = 20)

knn_recipe_20 <- recipe(Disaster_Frequency ~ .,
                        data = train_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

knn_workflow_20 <- workflow() %>%
  add_model(knn_parsnip_20) %>%
  add_recipe(knn_recipe_20)

knn_20 <- knn_workflow_20 %>% fit(train_sub)

# In-Sample validation
# k=5
knn_val1<- knn_workflow_5 %>%
  fit_resamples(
    resamples = vfold_cv(train_sub, v = 10),
    metrics = metric_set(rmse, rsq, mae),
  )
knn_val1 %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mae     standard   2.62     10  0.144  Preprocessor1_Model1
## 2 rmse    standard   4.29     10  0.274  Preprocessor1_Model1
## 3 rsq     standard   0.374    10  0.0772 Preprocessor1_Model1
```

```
# k=10
knn_val2 <- knn_workflow_10 %>%
  fit_resamples(
    resamples = vfold_cv(train_sub, v = 10),
    metrics = metric_set(rmse, rsq, mae),
  )
knn_val2 %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mae     standard   2.66     10  0.197  Preprocessor1_Model1
## 2 rmse    standard   4.29     10  0.293  Preprocessor1_Model1
## 3 rsq     standard   0.403    10  0.0527 Preprocessor1_Model1
```

```
# k=20
knn_val3 <- knn_workflow_10 %>%
  fit_resamples(
    resamples = vfold_cv(train_sub, v = 10),
    metrics = metric_set(rmse, rsq, mae),
  )
knn_val3 %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mae     standard   2.52     10  0.0763 Preprocessor1_Model1
## 2 rmse    standard   4.16     10  0.201  Preprocessor1_Model1
## 3 rsq     standard   0.434    10  0.0495 Preprocessor1_Model1
```

```
metrics_knn <- bind_rows(
  knn_val1 %>% collect_metrics(),
  knn_val2 %>% collect_metrics(),
  knn_val3 %>% collect_metrics(),
  .id = "Model"
)
metrics_knn
```

```
## # A tibble: 9 x 7
##   Model .metric .estimator  mean     n std_err .config
##   <chr> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 1     mae     standard   2.62     10  0.144  Preprocessor1_Model1
## 2 1     rmse    standard   4.29     10  0.274  Preprocessor1_Model1
## 3 1     rsq     standard   0.374    10  0.0772 Preprocessor1_Model1
## 4 2     mae     standard   2.66     10  0.197  Preprocessor1_Model1
## 5 2     rmse    standard   4.29     10  0.293  Preprocessor1_Model1
## 6 2     rsq     standard   0.403    10  0.0527 Preprocessor1_Model1
## 7 3     mae     standard   2.52     10  0.0763 Preprocessor1_Model1
## 8 3     rmse    standard   4.16     10  0.201  Preprocessor1_Model1
## 9 3     rsq     standard   0.434    10  0.0495 Preprocessor1_Model1
```

```r
# k=5
knn_parsnip_5t <- nearest_neighbor() %>%
  set_mode("regression") %>%
  set_engine("kknn", neighbors = 5)

knn_recipe_5t <- recipe(Disaster_Frequency ~ .,
                       data = test_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

knn_workflow_5t <- workflow() %>%
  add_model(knn_parsnip_5t) %>%
  add_recipe(knn_recipe_5t)

knn_5t <- knn_workflow_5t %>% fit(test_sub)

# k=10
knn_parsnip_10t <- nearest_neighbor() %>%
  set_mode("regression") %>%
  set_engine("kknn", neighbors = 10)

knn_recipe_10t <- recipe(Disaster_Frequency ~ .,
                        data = test_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

knn_workflow_10t <- workflow() %>%
  add_model(knn_parsnip_10t) %>%
  add_recipe(knn_recipe_10t)
```

```
knn_10t <- knn_workflow_10t %>% fit(test_sub)


# k=20 with standardization
knn_parsnip_20t <- nearest_neighbor() %>%
  set_mode("regression") %>%
  set_engine("kknn", neighbors = 20)

knn_recipe_20t <- recipe(Disaster_Frequency ~ .,
                        data = test_sub) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

knn_workflow_20t <- workflow() %>%
  add_model(knn_parsnip_20t) %>%
  add_recipe(knn_recipe_20t)

knn_20t <- knn_workflow_20t %>% fit(test_sub)


# validation
# k=5
knn_val1t<- knn_workflow_5t %>%
  fit_resamples(
    resamples = vfold_cv(test_sub, v = 10),
    metrics = metric_set(rmse,mae,rsq),
  )
knn_val1t %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mae     standard   2.17     10   0.230 Preprocessor1_Model1
## 2 rmse    standard   3.53     10   0.461 Preprocessor1_Model1
## 3 rsq     standard   0.380    10   0.109 Preprocessor1_Model1
```

```
# k=10
knn_val2t <- knn_workflow_10t %>%
  fit_resamples(
    resamples = vfold_cv(test_sub, v = 10),
    metrics = metric_set(rmse,mae,rsq),
  )
knn_val2t %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mae     standard   2.02     10   0.188 Preprocessor1_Model1
## 2 rmse    standard   3.36     10   0.372 Preprocessor1_Model1
## 3 rsq     standard   0.482    10   0.107 Preprocessor1_Model1
```

```
# k=20
knn_val3t <- knn_workflow_20t %>%
  fit_resamples(
    resamples = vfold_cv(test_sub, v = 10),
    metrics = metric_set(rmse,mae,rsq),
  )
knn_val3t %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mae     standard   2.11     10  0.308  Preprocessor1_Model1
## 2 rmse    standard   3.33     10  0.547  Preprocessor1_Model1
## 3 rsq     standard   0.343    10  0.0981 Preprocessor1_Model1
```

```
metrics_knn_held <- bind_rows(
  knn_val1t %>% collect_metrics(),
  knn_val2t %>% collect_metrics(),
  knn_val3t %>% collect_metrics(),
  .id = "Out of Sample"
)
```

```
metrics_knn_held
```

```
## # A tibble: 9 x 7
##   `Out of Sample` .metric .estimator  mean     n std_err .config
##   <chr>           <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 1               mae     standard   2.17     10  0.230  Preprocessor1_Model1
## 2 1               rmse    standard   3.53     10  0.461  Preprocessor1_Model1
## 3 1               rsq     standard   0.380    10  0.109  Preprocessor1_Model1
## 4 2               mae     standard   2.02     10  0.188  Preprocessor1_Model1
## 5 2               rmse    standard   3.36     10  0.372  Preprocessor1_Model1
## 6 2               rsq     standard   0.482    10  0.107  Preprocessor1_Model1
## 7 3               mae     standard   2.11     10  0.308  Preprocessor1_Model1
## 8 3               rmse    standard   3.33     10  0.547  Preprocessor1_Model1
## 9 3               rsq     standard   0.343    10  0.0981 Preprocessor1_Model1
```

Model Selection

```r
# set.seed(123457)
# train_bootstrap <- train_sub %>% bootstraps(times = 25)
# workflow_RF_boostrap <- workflow_RF %>%
#   fit_resamples(train_bootstrap, metrics = RF_metrics)
```

Model Assessment and Interpretation

From the diagnostic plot, we can interpret that our model prediction complied well with the observed values. However, the dataset seems to contain quite a few outliers of which may have a significant effect on our model.

```r
# Diagnostic Plots

# setting up plot environment
par(mfrow = c(2,2), mar = c(4,4,5,4))

# Scatter plot of Actual versus predicted
plot(train_sub$Disaster_Frequency, RF_train_predict$.pred,
     main = "Actual versus Predicted",
     ylab = "Prediction Result",
     xlab = "Actual Result")

# calculate residuals
resid <- train_sub$Disaster_Frequency - RF_train_predict$.pred

# fitted versus residual plot
fitted_resid_plot <- plot(RF_train_predict$.pred, resid,
                          ylab = "Residuals",
                          xlab = "Fitted Values",
                          main = "Fitted versus Residuals")
abline(h = 0, col = 'blue', lty = 2)

# calculate standard residuals
stand_resid <- (resid - mean(resid))/sd(resid)

# fitted versus standard residual plot
plot(RF_train_predict$.pred, stand_resid,
     ylab = "Standard Residuals",
     xlab = "Fitted Values",
     main = "Fitted versus Standard Residuals")
abline(h = 0, col = 'blue', lty = 2)

# qq plot
qqPlot(stand_resid, main = "QQ-norm Plot",
       id = FALSE,
       ylab = "Standard Residuals",
       xlab = "Norm Quantiles")

# setting a common title
mtext("Diagnostic Plots", side = 3, line = -2, cex = 1.5, outer = TRUE)
```

```r
train_PCA <- train_sub %>%
  mutate_if(~ any(!is.na(as.numeric(.))), ~as.numeric(.)) %>%
  prcomp(scale = TRUE)
train_loadings <- broom::tidy(train_PCA, matrix = "loadings")
train_scores <- broom::tidy(train_PCA, matrix = "scores")
```

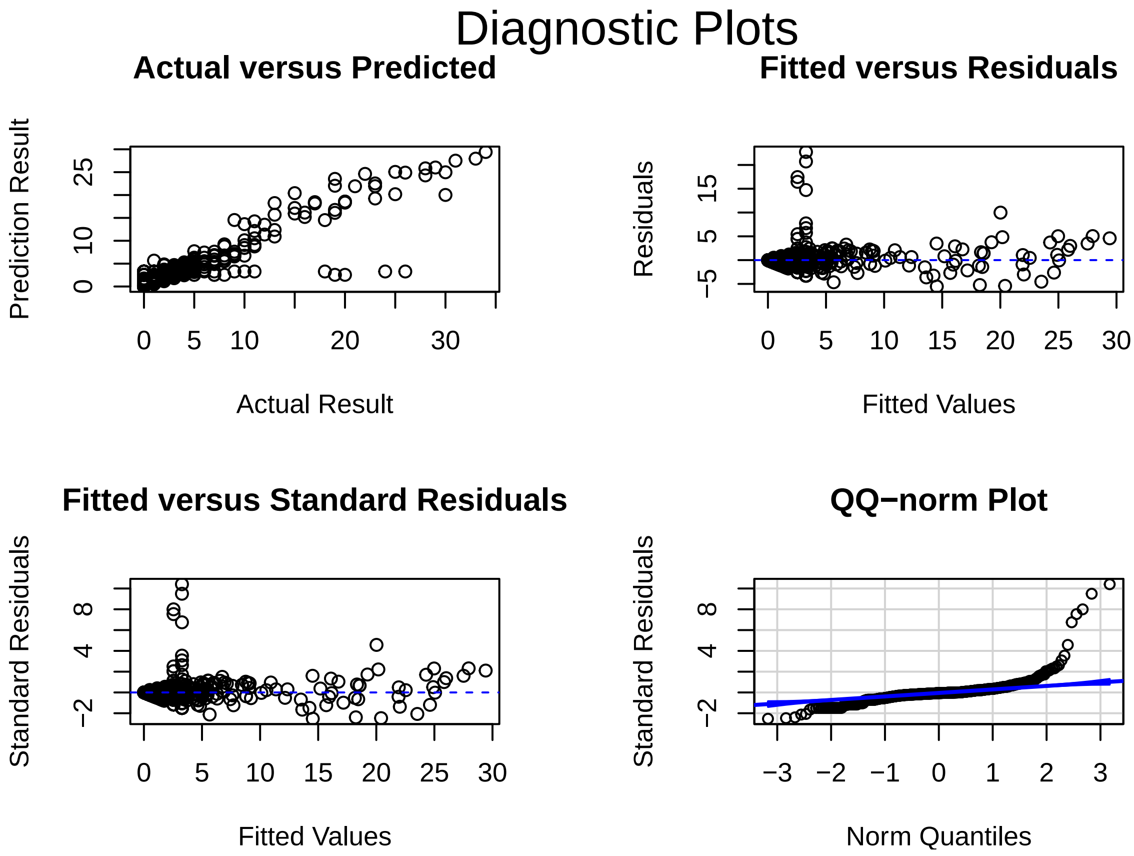


Figure 5: Diagnostic plot of the random forest model. We see somewhat of a linear trend in the actual versus predicted plot (top left) with some outliers. This implies that majority of our prediction complied with the observed values except for a few outliers. The fitted versus residuals plot (top right) indicates no specific pattern. However, there does appears to be quite a few outliers. This also occured in the fitted versus standard residuals plot (bottom left). Finally, the qq-plot appears to be heavy-tailed.

```r
train_variances <- broom::tidy(train_PCA, matrix = "pcs")


train_variances %>% ggplot(aes(x = PC, y = percent)) +
  geom_segment(aes(xend = PC), yend = 0, linewidth = 5) +
  ylab("Percentage of Variance Explained") +
  ggtitle("Percentage of Variances Explained by Each Principle Components")
```

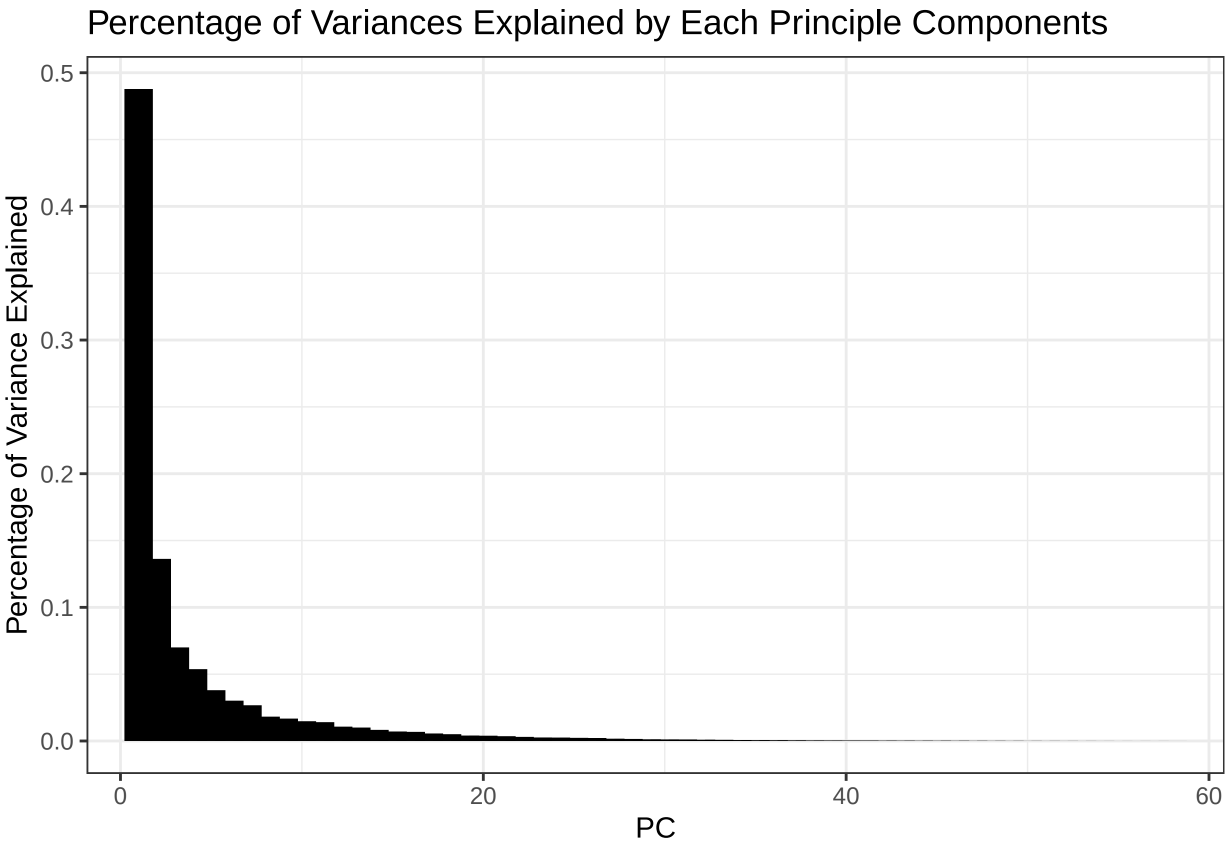


Figure 6: Percentage of Vairnaces Explained by Each Principle Components. The first couple principle components seems to add up to the majority of the percentage of variances.

```r
cat("Percentage of variances explained by the first four principle components is",
    (train_variances$percent[1:4] %>% sum())*100, "%")
```

```
## Percentage of variances explained by the first four principle components is 74.789 %
```

```
train_loadings %>%

  filter(PC <=2) %>%

  ggplot(aes(y = column, x = value)) +

  geom_segment(aes(yend = column), xend = 0, linewidth = 4) +

  facet_wrap(~PC) +

  theme(axis.text.y=element_text(size=rel(0.75)))
```

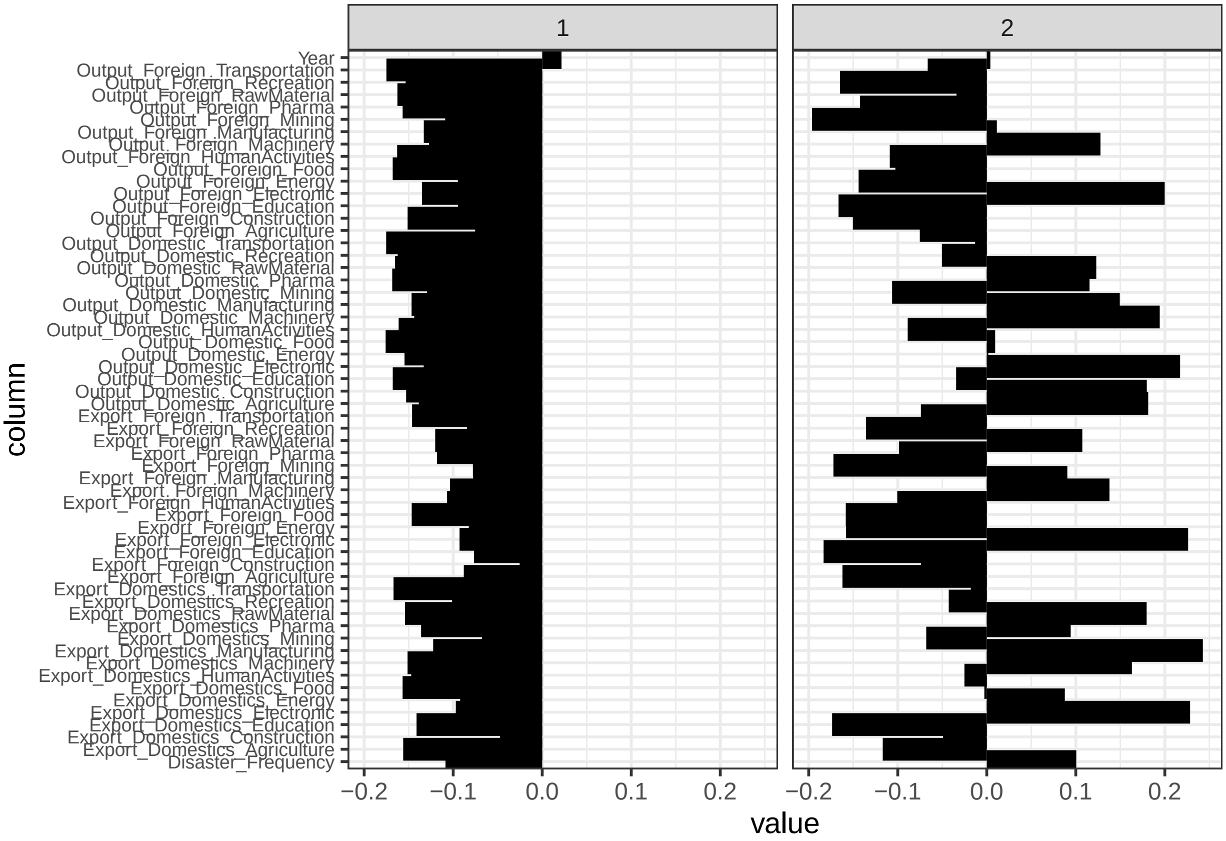


Figure 7: Factor loadings for the first two principle components. We see that for the first component, all factors have negative effects while only year has positive effect. And in the second component does not seems to be interpretable

```
train_loadings %>%

  filter(PC > 2 & PC <= 4) %>%

  ggplot(aes(y = column, x = value)) +

  geom_segment(aes(yend = column), xend = 0, linewidth = 4) +

  facet_wrap(~PC) +
theme(axis.text.y=element_text(size=rel(0.75)))
```

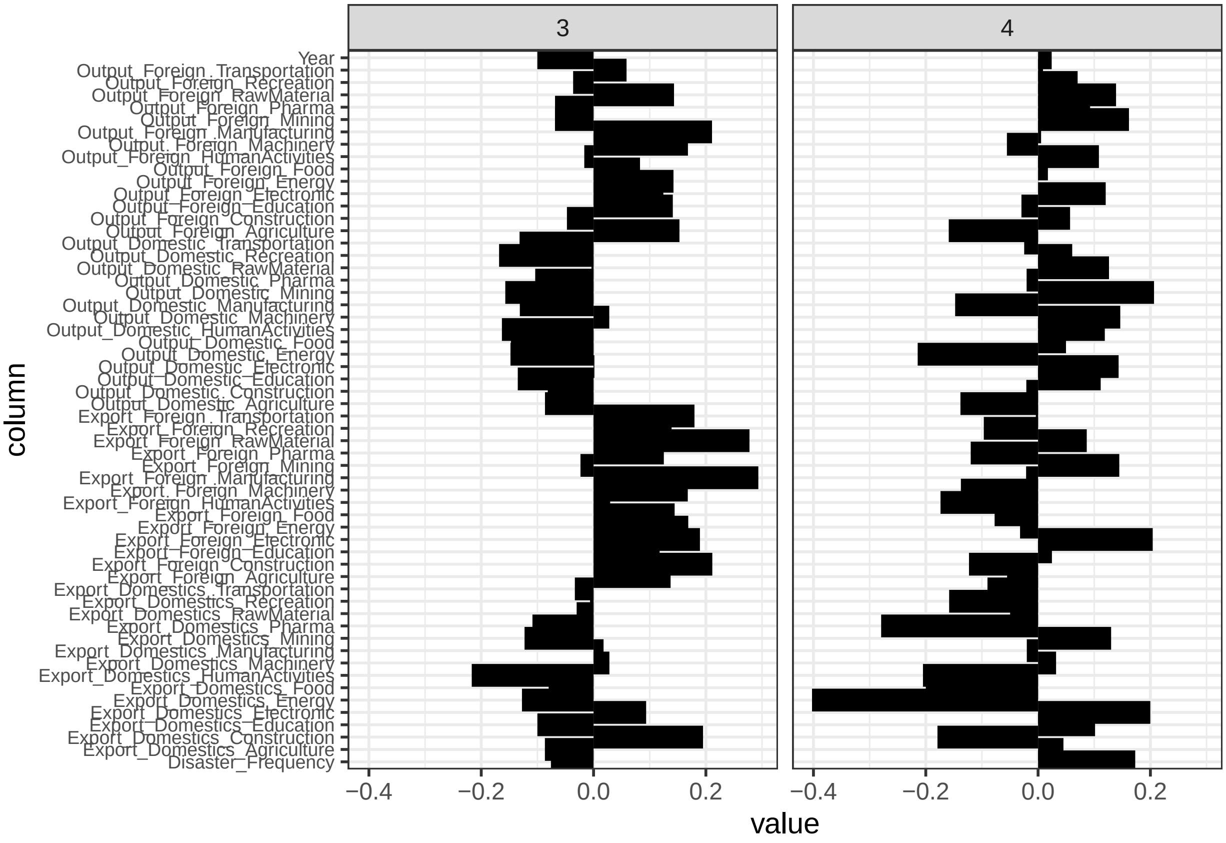


Figure 8: Factor loadings for the third and fourth principle components. Both factoring plot does not seem to be interpretable.

Uncertainty Quantification

Result Communication

- reciting what is our best model.

Citation

International Monetary Fund. 2022.Climate Change Indicators Dashboard. Climate Change Data and , https://climatedata.imf.org/pages/access-data. Accessed on [2023-03-15].