Landslide Analysis

Theodore Li, Drew Keely, Dvanshu Khadka, Nami Jian

10/18/2024

Here is a data dictionary containing all features of interest to our project.

This first model is a attempt at predicting the possibility of a landslide occurring at a specific specific region of a 100 mile radius.

Data Cleaning

Step 1, get the columns we want. Not the redundant columns that contain the same information or identification numbers, or columns that we don't know what it means.

```
  landslide_  trigger fatalities injuries landslide1  distance population key_
1   Mudslide Downpour          0        0     Medium   2.12825       6523   US
2  Landslide     Rain          0        0     Medium 215.44888          0   ID
3   Mudslide     Rain          0        0     Medium 199.44893       8987   CA
4    Complex Downpour          0        0     Medium 178.23706      33430   CA
5   Mudslide Downpour          0        0      Small 176.02202      14708   CA
6   Mudslide Downpour          0        0     Medium 166.44009      33430   CA
  latitude longitude                       date_
1  41.5585  -73.4020 03/07/2011 08:00:00 AM +0000
2   0.1115  113.9171 04/01/2008 07:00:00 AM +0000
3  52.3545 -127.6980 01/30/2015 08:00:00 AM +0000
4  50.7053 -127.5062 09/24/2010 07:00:00 AM +0000
5  53.3319 -132.4149 08/31/2014 07:00:00 AM +0000
6  50.4335 -127.4846 09/24/2010 07:00:00 AM +0000
```

The month and the year are under the column "date_", we'll need to extract this into respective columns to properly use them as features in our model.

Extracting the month and the year

```
library(stringr)

# Convert to Date format
parsed_datetime <- strptime(landslide_data$date_, "%m/%d/%Y %I:%M:%S %p %z")

# Extract year, month, day, etc.
landslide_data$year <- format(parsed_datetime, "%Y")
landslide_data$month <- format(parsed_datetime, "%m")

# Convert to numeric if needed
landslide_data$year <- as.integer(landslide_data$year)
landslide_data$month <- as.integer(landslide_data$month)

head(landslide_data)
```

```
  landslide_  trigger fatalities injuries landslide1  distance population key_
1   mudslide downpour          0        0     medium   2.12825       6523   US
2  landslide     rain          0        0     medium 215.44888          0   ID
3   mudslide     rain          0        0     medium 199.44893       8987   CA
4    complex downpour          0        0     medium 178.23706      33430   CA
5   mudslide downpour          0        0      small 176.02202      14708   CA
6   mudslide downpour          0        0     medium 166.44009      33430   CA
  latitude longitude                        date_ year month
1  41.5585  -73.4020 03/07/2011 08:00:00 AM +0000 2011     3
2   0.1115  113.9171 04/01/2008 07:00:00 AM +0000 2008     4
3  52.3545 -127.6980 01/30/2015 08:00:00 AM +0000 2015     1
4  50.7053 -127.5062 09/24/2010 07:00:00 AM +0000 2010     9
5  53.3319 -132.4149 08/31/2014 07:00:00 AM +0000 2014     8
6  50.4335 -127.4846 09/24/2010 07:00:00 AM +0000 2010     9
```

Now we can remove the date column

```
landslide_data <- subset(landslide_data, select = -date_)
```

Now we'll remove all the rows containing NA:

```
landslide_data <- na.omit(landslide_data)
```

Let's visualize the triggers, are there any dominant triggers that can be identified.

```
# Load necessary libraries
library(knitr)
library(kableExtra)
library(gridExtra)

# Assuming 'landslide_data' is your dataset and 'trigger' is the feature you're interested in

# Calculate frequency of each trigger type
trigger_frequencies <- table(landslide_data$trigger)

# Convert to data frame for better readability
trigger_df <- as.data.frame(trigger_frequencies)
colnames(trigger_df) <- c("Trigger_Type", "Frequency")

# Display the table with kable
kable_table <- kable(trigger_df, caption = "Frequencies of Different Triggers") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))

# Save the table as a PNG image
png("trigger_table.png", width = 800, height = 600)  # Set PNG file size
grid.table(trigger_df)  # Plot the table
dev.off()  # Close the plotting device and save the PNG
```

```
pdf
  2
```

```
kable_table
```

Table 1: Frequencies of Different Triggers

Trigger_Type	Frequency

construction	21
continuousrain	271
damembankmentcollapse	4
downpour	3887
earthquake	34
flooding	14
freezethaw	13
miningdigging	38
monsoon	70
noapparenttrigger	4
other	12
rain	1573
snowfall	46
tropicalcyclone	457
unknown	338

```
# Calculate the number of occurrences for each month
month_count <- table(landslide_data$month)
```

Now to turn all the columns into numerical values to be able to be used in our model:

```
landslide_data$landslide_ <- factor(landslide_data$landslide_)
landslide_data$landslide_ <- as.integer(landslide_data$landslide_)

landslide_data$trigger <- factor(landslide_data$trigger)
landslide_data$trigger <- as.integer(landslide_data$trigger)

landslide_data$landslide1 <- factor(landslide_data$landslide1)
landslide_data$landslide1 <- as.integer(landslide_data$landslide1)

landslide_data$key_ <- factor(landslide_data$key)
landslide_data$key_ <- as.integer(landslide_data$key)


head(landslide_data)
```

```
  landslide_ trigger fatalities injuries landslide1  distance population key_
1          6       4          0        0          4   2.12825       6523  116
2          5      12          0        0          4 215.44888          0   54
3          6      12          0        0          4 199.44893       8987   20
4          1       4          0        0          4 178.23706      33430   20
5          6       4          0        0          5 176.02202      14708   20
6          6       4          0        0          4 166.44009      33430   20
  latitude longitude year month
1  41.5585  -73.4020 2011     3
2   0.1115  113.9171 2008     4
3  52.3545 -127.6980 2015     1
4  50.7053 -127.5062 2010     9
5  53.3319 -132.4149 2014     8
6  50.4335 -127.4846 2010     9
```

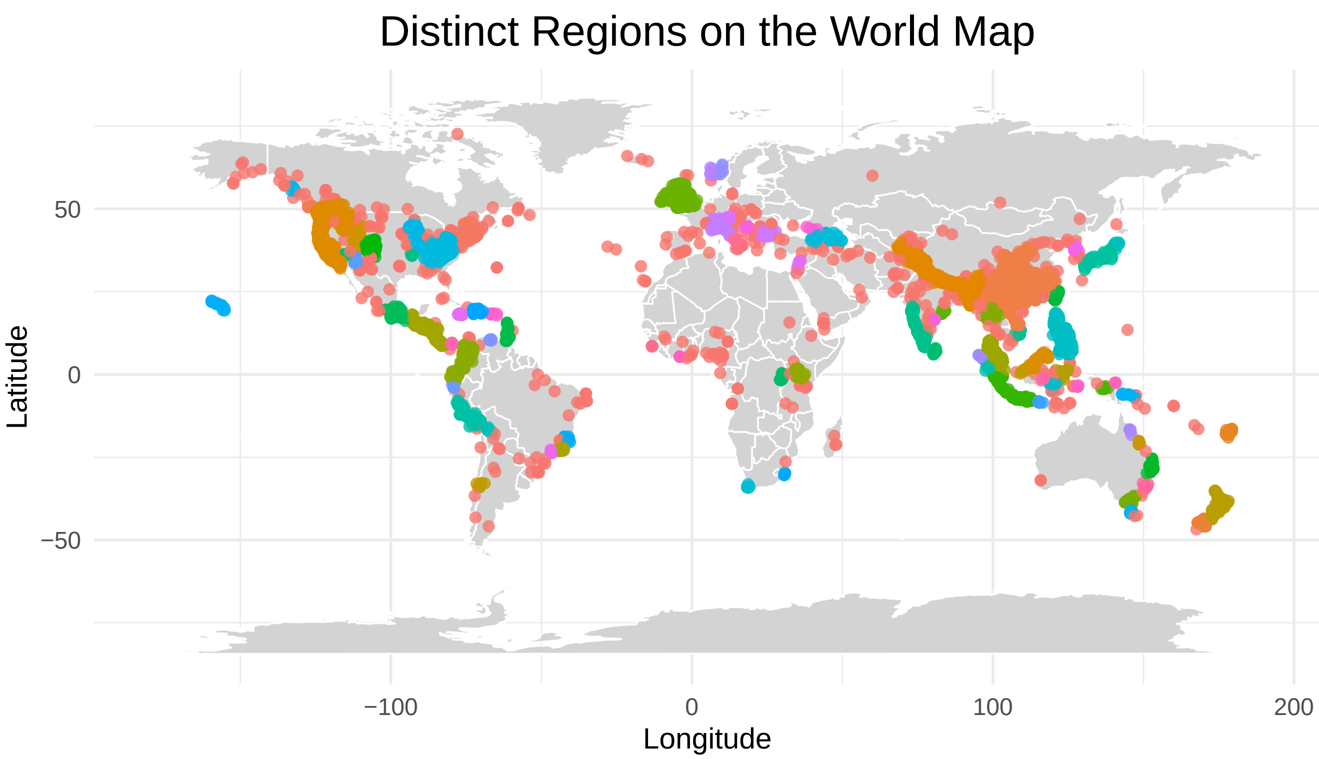
Now to make our predictions easier, we'll create another feature grouping the observations into regions of 100 mile radius. For a region to exist there needs to be at least 5 observations within it

```
library(dbscan)

coords <- landslide_data[, c("longitude", "latitude")] #Extract the lat and long cols
```

```r
clustering <- dbscan(coords, eps = 1.45, minPts = 5)  # eps = 160.934/111 where 1 degree is 111 kilometers
landslide_data$region <- clustering$cluster
landslide_data$region <- factor(landslide_data$region)
```

Now we have a new column of regions grouping together all observations together within a 100 mile radius ~~~~~~~~~~~~~~ Data Cleaning Complete

```r
library(ggplot2)
library(maps)

# Get the world map data
world_map <- map_data("world")

# Plot the clusters (regions) on the map without a legend
ggplot() +
  geom_map(data = world_map, map = world_map, aes(x = long, y = lat, map_id = region),
           fill = "lightgray", color = "white", size = 0.3) +
  geom_point(data = landslide_data, aes(x = longitude, y = latitude, color = factor(region)),
             size = 1.5, alpha = 0.8) +
  theme_minimal() +
  labs(
    title = "Distinct Regions on the World Map",
    x = "Longitude",
    y = "Latitude"
  ) +
  coord_fixed(ratio = 1.1) +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16),
    legend.position = "none"  # Hide the legend
  )
```

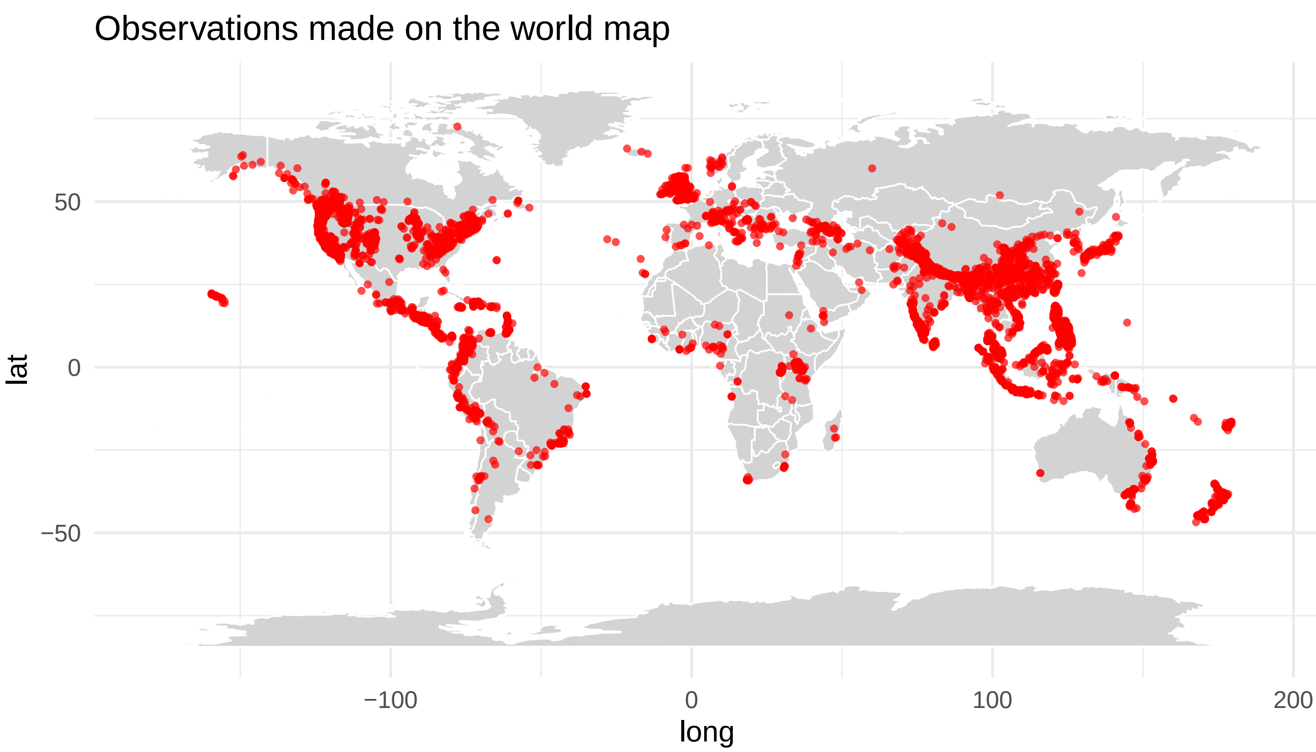


Visualization

Next, let's visualize our data and our regions. This could give us a idea on spatial dependencies between the data points where those coordinates nearby are more likely to observe landslides as well.

```
# Assuming your 'landslide_data' has columns 'longitude' and 'latitude'
world_map <- map_data("world")

# Create a basic world map
ggplot() +
  geom_map(data = world_map, map = world_map, aes(x = long, y = lat, map_id = region),
           fill = "lightgray", color = "white", size = 0.3) +
  geom_point(data = landslide_data, aes(x = longitude, y = latitude),
             color = "red", size = .75, alpha = 0.7) +
  theme_minimal() +
```

```
  labs(title = "Observations made on the world map") +
  coord_fixed(ratio = 1.1)
```



Indeed, we can see a pattern of grouping among the data points. Thus coordinates near a landslide is much more likely to experience another landslide. This violates the simple linear regression assumption of linearly independent observations, thus just by this visualization we know that a linear relationship is not a good fit for our research question.

A case we need to consider is the use of coordinates as features. The whole point of this model is to make predictions on future landslides on where they might appear, thus if we include the coordinates as predictors which is directly correlated to the landslide regions we constructed earlier, this would also lead to data leakage. And if we already know the coordinates of where the landslide will appear, then it's pretty pointless to predict the region it appears in. Thus, we'll remove coordinates from the dataset.

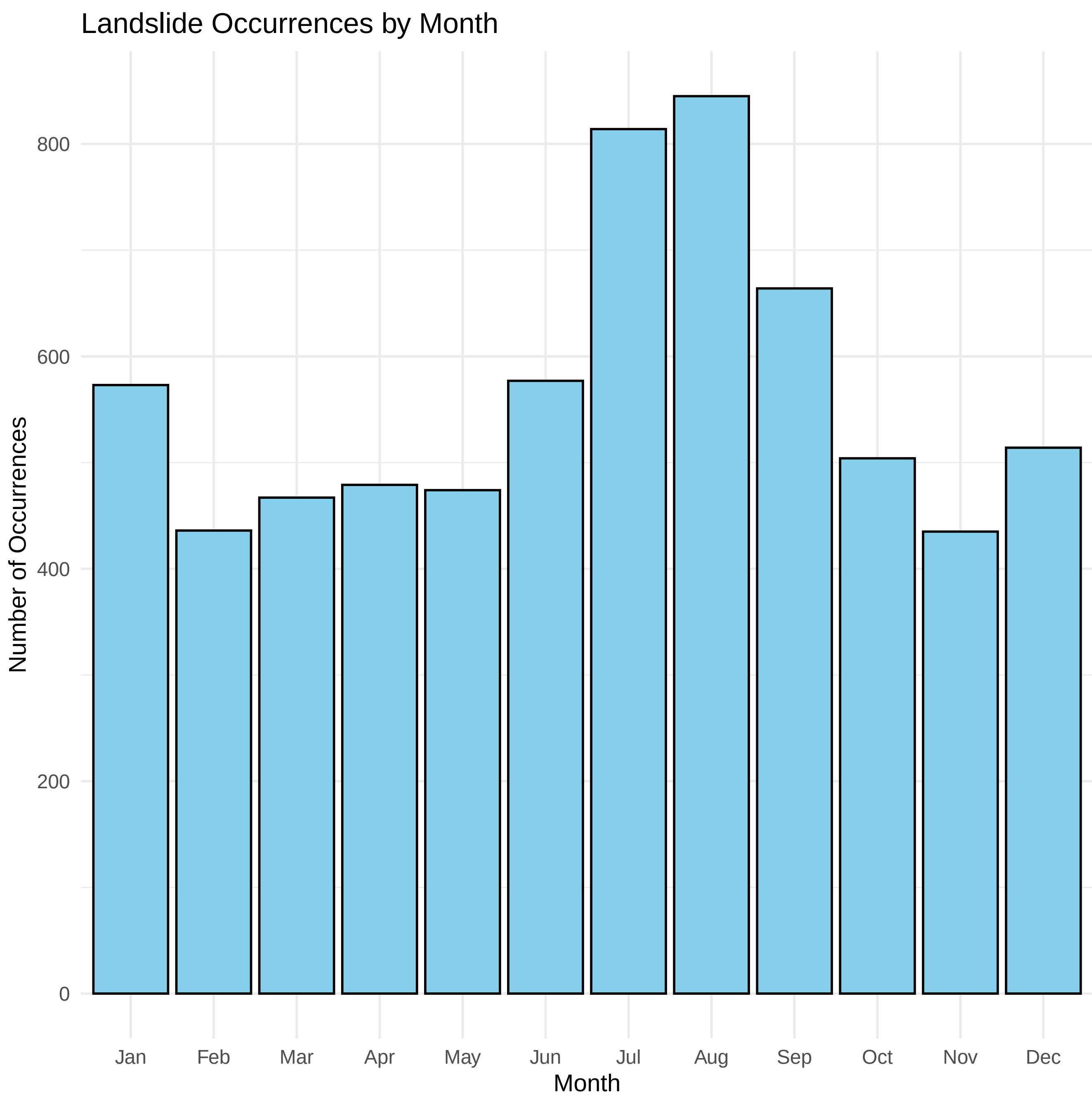
```
landslide_data <- subset(landslide_data, select = -c(latitude, longitude))
head(landslide_data)
```

```
  landslide_ trigger fatalities injuries landslide1  distance population key_
```

```
1          6       4          0        0          4   2.12825       6523  116
2          5      12          0        0          4 215.44888          0   54
3          6      12          0        0          4 199.44893       8987   20
4          1       4          0        0          4 178.23706      33430   20
5          6       4          0        0          5 176.02202      14708   20
6          6       4          0        0          4 166.44009      33430   20
  year month region
1 2011     3      1
2 2008     4      0
3 2015     1      0
4 2010     9      2
5 2014     8      0
6 2010     9      2
```

Next, let's visualize the time-related features. Let's look at the months and the frequency of landslides for each month.

```r
# Load necessary libraries
library(ggplot2)

# Make sure the 'month' column exists in your dataset
# If you haven't already, create the 'month' column (e.g., from a 'timestamp' column)
# landslide_data$month <- as.integer(str_extract(landslide_data$timestamp, "^\\d{1,2}"))

# Create a bar plot of landslides by month
ggplot(landslide_data, aes(x = factor(month))) +
  geom_bar(fill = "skyblue", color = "black") +  # Create bars with color
  labs(title = "Landslide Occurrences by Month",
       x = "Month", y = "Number of Occurrences") +
  theme_minimal() +  # Clean theme
  scale_x_discrete(labels = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "De
```

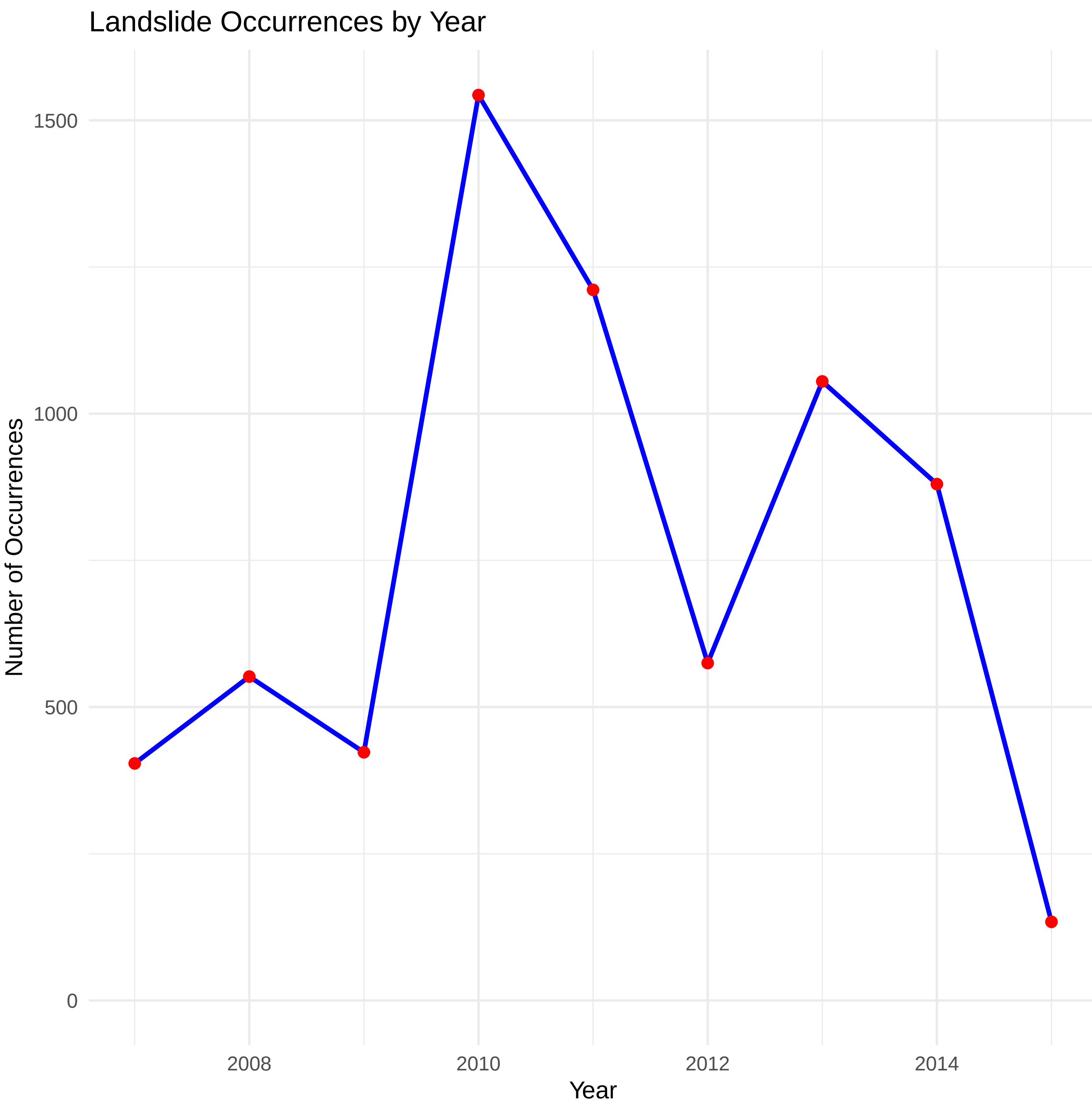


We can see that the most observations of landslides occurred in the month of August and July which is surprising considering that the most common trigger to landslides is due to downpour. Thus it's only natural to assume that the months with the most precipitation would contain the most landslide occurrences.

Now let's visualize the occurances of landslides throughout the timeline.

```
# Aggregate the data by year to count the number of occurrences per year
library(dplyr)
yearly_counts <- landslide_data %>%
  group_by(year) %>%
  summarise(occurrences = n())

# Create the plot
library(ggplot2)
# Create the line plot, starting from 2007
ggplot(yearly_counts, aes(x = year, y = occurrences)) +
  geom_line(color = "blue", size = 1) +  # Line plot
```

```
geom_point(color = "red", size = 2) +  # Add points on the line
labs(title = "Landslide Occurrences by Year",
     x = "Year", y = "Number of Occurrences") +
theme_minimal() +  # Clean theme
scale_x_continuous(limits = c(2007, NA))
```



Overall we can see harsh peaks and drops from 2007-2015. Most likely since this is a natural disaster has to relate to changes in climate, particularly if those years contain more downpour which as shown above is the most frequent trigger for the landslides. Additionally we can see a trend in the plot as generally it follows the pattern of increasing landslide occurrences for one year then followed by 2 years of decreasing occurrences.

Thus, due to the potential temporal trends (increasing or decreasing based on climate changes),if we randomly pick 70% observations for training and 30% for testing this could lead to data leakage. If the training data contains all landslides across the entire time, (2007-2015) as we randomly pick 70% from all the years this can allow the model to be overly reliant on the trends in the training data. Thus if we were to make predictions on testing data, say for 2012, but we already know about landslide occurrences before and after thus the model may be overly reliant on the trend from the training data.

Thus a better solution is to pick training data to be from 2007-2012 and testing data be the landslides after 2012. This creates a better simulation of the testing cases as we only know about the previous landslide data and need to predict on unseen data in the future.

```
# Split the data into training and testing sets
landslide_train <- subset(landslide_data, year < 2012)  # Data before 2012
landslide_test <- subset(landslide_data, year >= 2012)  # Data from 2012 onward

# Check the number of rows in each dataset
cat("Training data size:", nrow(landslide_train), "\n")
```

```
Training data size: 4138
```

```
cat("Testing data size:", nrow(landslide_test), "\n")
```

```
Testing data size: 2644
```

```
# Convert 'region' to a factor in both the training and testing sets
landslide_train$region <- as.factor(landslide_train$region)
landslide_test$region <- as.factor(landslide_test$region)
# Ensure both training and testing sets have the same factor levels
levels(landslide_test$region) <- levels(landslide_train$region)
```

Model Fitting

For our use case of predicting the likelihood of landslides based on coordinates, country, time of the year. We need to consider models that links the relationship between location, and previous occurrences of landslides. We are predicting which region the next landslide will occur in.

Thus the model I would like to use is a random forest due to their ability to handle high-dimensional data especially with spatial features, in our case coordinates. Additionally, the random forest is a flexible model that works well with a mix of categorical and numerical features which is contained in our landslide_data. However, a big difference between a random forest is that it is a supervised machine learning algorithm, contrary to kNN. The logic behind the random forest is majority rules as it runs multiple iterations through different decision trees and averages the different outcomes to finalize a prediction.

#Initial Model fitting with random forest

```
library(randomForest)

# Fit the Random Forest model to predict the region of landslide occurrences
model_rf_region <- randomForest(region ~ .,
                                data = landslide_train)

predicted_regions_test <- predict(model_rf_region, newdata = landslide_test)

#predicted_regions_test <- as.factor((predicted_regions_test))

# Calculate the accuracy
accuracy <- sum(predicted_regions_test == landslide_test$region) / length(predicted_regions_test)
cat("Accuracy: ", accuracy, "\n")
```

```
Accuracy:  0.5945537
```

This is our baseline model with accuracy of 59.2%

Not bad especially considering there are 86 regions but we can do better.

#Tuning hyperparameters

We will try to use cross validation with 5 folds to make our model more robust to untouched data.

Additionally we will iterate through different values of mtry: Number of variables to be picked as candidates for each split in the tree We will also test through different values of the tree, how many trees we use and find the average out of

```r
library(caret)

# Define training method (without cross-validation)
train_control <- trainControl(method = "none")  # Set method to "none" to remove cross-validation

# Initialize a list to store results and track the best model for each ntree
best_accuracies <- list()  # Store best accuracy for each ntree

# Loop over ntree values
for (ntree in c(100, 200, 300)) {  # Looping through ntree values

    # Initialize variables to track best accuracy for the current ntree
    best_accuracy_for_ntree <- 0  # Variable to store best accuracy for this ntree
    best_mtry_for_ntree <- NA  # Variable to store the corresponding mtry for the best accuracy

    # Loop over mtry values
    for (mtry_value in c(10, 12, 14, 16, 18, 20)) {  # Looping through mtry values
        cat("Training with ntree =", ntree, " and mtry =", mtry_value, "\n")

        # Train the model for each ntree and mtry value (without cross-validation)
        tuned_model <- train(region ~ .,
                             data = landslide_train,
                             method = "rf",
                             trControl = train_control,
                             ntree = ntree,   # Set the ntree
                             tuneGrid = data.frame(mtry = mtry_value))  # Set the mtry

        best_model <- tuned_model$finalModel  # Extract the final trained model
        predicted_regions_test <- predict(best_model, newdata = landslide_test)  # Test on testing data

        # Calculate accuracy on test data
        accuracy <- sum(predicted_regions_test == landslide_test$region) / length(predicted_regions_test)
        accuracy_percentage <- accuracy * 100  # Convert to percentage

        # Update the best accuracy for this ntree if current model is better
        if (accuracy_percentage > best_accuracy_for_ntree) {
            best_accuracy_for_ntree <- accuracy_percentage
            best_mtry_for_ntree <- mtry_value  # Track the mtry value that gave best accuracy
        }
    }

    # Store best accuracy for current ntree
    best_accuracies[[paste("ntree", ntree, sep = "_")]] <- list(
        best_accuracy = best_accuracy_for_ntree,
        best_mtry = best_mtry_for_ntree
    )

    # Print the best accuracy for the current ntree value
    cat("\nBest Accuracy for ntree =", ntree, ":", round(best_accuracy_for_ntree, 2), "%\n")
    cat("Best mtry for ntree =", ntree, ":", best_mtry_for_ntree, "\n")
}
```

```
Training with ntree = 100  and mtry = 10
Training with ntree = 100  and mtry = 12
Training with ntree = 100  and mtry = 14
```

```
Training with ntree = 100  and mtry = 16
Training with ntree = 100  and mtry = 18
Training with ntree = 100  and mtry = 20

Best Accuracy for ntree = 100 : 70.5 %
Best mtry for ntree = 100 : 20
Training with ntree = 200  and mtry = 10
Training with ntree = 200  and mtry = 12
Training with ntree = 200  and mtry = 14
Training with ntree = 200  and mtry = 16
Training with ntree = 200  and mtry = 18
Training with ntree = 200  and mtry = 20

Best Accuracy for ntree = 200 : 70.76 %
Best mtry for ntree = 200 : 20
Training with ntree = 300  and mtry = 10
Training with ntree = 300  and mtry = 12
Training with ntree = 300  and mtry = 14
Training with ntree = 300  and mtry = 16
Training with ntree = 300  and mtry = 18
Training with ntree = 300  and mtry = 20

Best Accuracy for ntree = 300 : 70.65 %
Best mtry for ntree = 300 : 18
```

```r
# Optionally, print the stored best results for each ntree
cat("\nSummary of Best Accuracy for each ntree:\n")
```

```
Summary of Best Accuracy for each ntree:
```

```r
for (ntree in names(best_accuracies)) {
  cat("\n", ntree, ":\n")
  cat("Best Accuracy: ", best_accuracies[[ntree]]$best_accuracy, "%\n")
  cat("Best mtry: ", best_accuracies[[ntree]]$best_mtry, "\n")
}
```

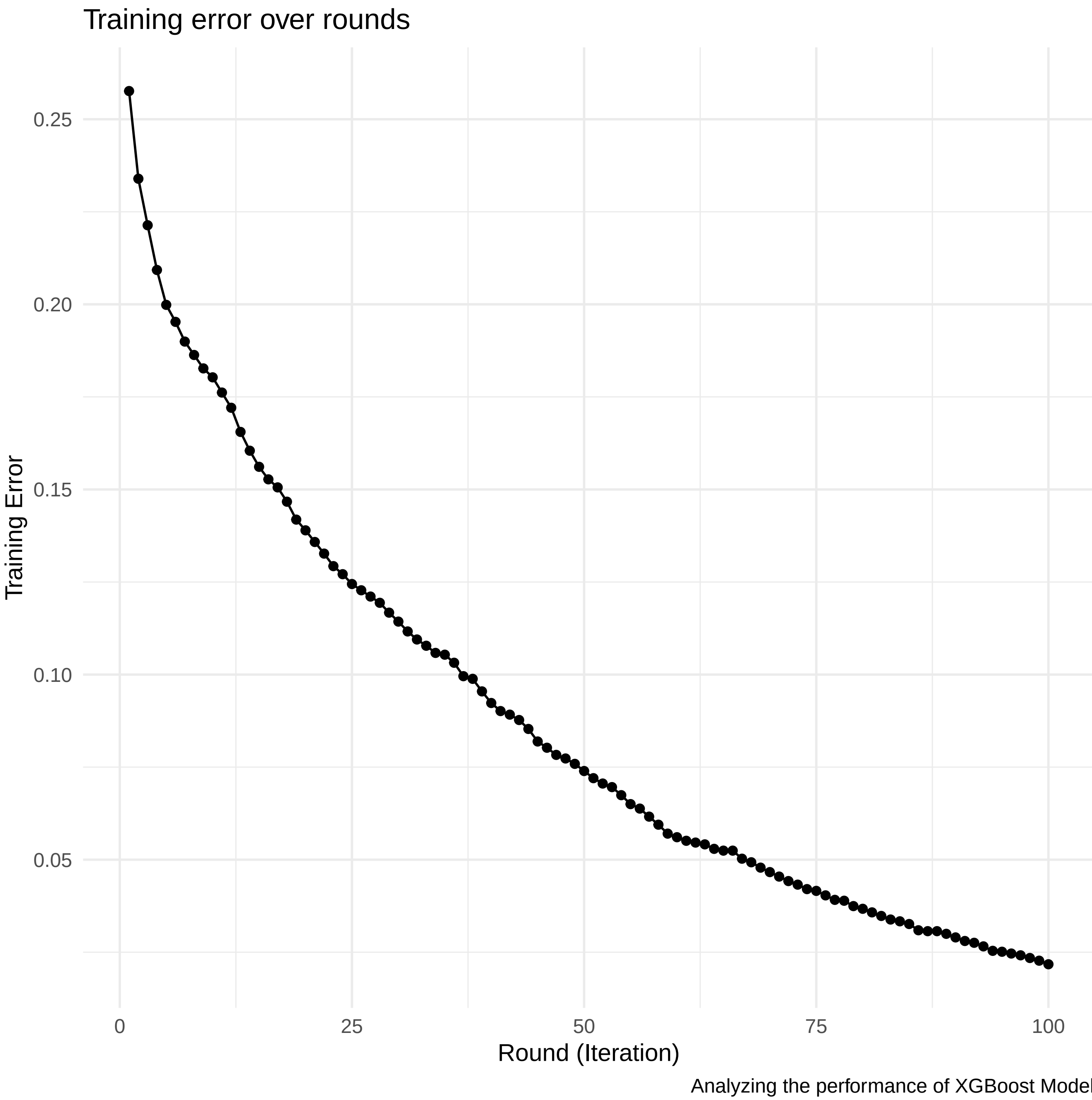
```
 ntree_100 :
Best Accuracy:  70.49924 %
Best mtry:  20

 ntree_200 :
Best Accuracy:  70.76399 %
Best mtry:  20

 ntree_300 :
Best Accuracy:  70.65053 %
Best mtry:  18
```

After tuning the hyper parameters we are able to get accuracy of 68% however our accuracies aren't really improving from increasing ntree or mtry which is usually the case.

Thus we need to analyze our data and model more and see what else we can change to help give us a better prediction.

Let us try XGBoost which is another tree based model. The main difference is that with Random Forests, trees are built during training and averaged out for a outcome, XGBoost trains one tree after another where each iteration is focused on correcting the error from the previous tree.

```r
library(xgboost)

train_matrix <- as.matrix(landslide_train[, -which(names(landslide_train) == "region")])  # Exclude the target
train_label <- as.factor(landslide_train$region)  # Convert target to factor

# Ensure labels are zero-indexed (convert factor levels to integers starting from 0)
train_label <- as.numeric(train_label) - 1  # Subtract 1 to start from 0

test_matrix <- as.matrix(landslide_test[, -which(names(landslide_test) == "region")])
test_label <- as.factor(landslide_test$region)
test_label <- as.numeric(test_label) - 1  # Zero-index the test labels as well

params <- list(
  objective = "multi:softmax",  # Multi-class classification
  num_class = length(unique(train_label)),  # Number of classes in the target
  eval_metric = "merror",       # Evaluation metric: Multi-class classification error
  max_depth = 6,                # Maximum depth of the trees
  eta = 0.1                     # Learning rate
)

# Train the model
xgb_model <- xgboost(
  data = train_matrix,
  label = train_label,
  params = params,
  nrounds = 100,                # Number of boosting rounds (trees)
  verbose = 0
)

# Plot training error during the boosting rounds
evals_result <- xgb_model$evaluation_log

# Create a line plot of the training error (merror) over boosting rounds
ggplot(evals_result, aes(x = iter, y = train_merror)) +
  geom_line() +
  geom_point() +
  labs(
    title = "Training error over rounds",
    x = "Round (Iteration)",
    y = "Training Error",
    caption = "Analyzing the performance of XGBoost Model"
  ) +
  theme_minimal()
```




```
# Make predictions
predictions <- predict(xgb_model, newdata = test_matrix)

# Calculate accuracy
accuracy <- sum(predictions == test_label) / length(test_label) * 100
cat("\nTest Accuracy: ", round(accuracy, 2), "%\n")
```

```
Test Accuracy:  70.92 %
```

Even with a basic XGBoost without hyperparameter tuning we are able to get better performance than a optimized random forest. However, it is still overfitting to the data. Next technique I would like to try is bootstrapping.