MA 679 Final Project

Zihao Zhang, Keyu Feng, Jinran Li, Ziheng Li

2025-04-22

Abstract

In this project, we developed and evaluated predictive models to estimate the number of affected customers ("customers_day") due to weather-related power outages. Using historical storm event data combined with outage reports, we performed extensive data preprocessing and feature engineering to create a structured dataset suitable for machine learning. We implement four models: Lasso regression, Ridge regression, Random Forest (via ranger), and XGBoost. Among these, XGBoost delivers the best performance, achieving an RMSE of 35.91 and an MAE of 1.26 on the test set. The results demonstrate the effectiveness of ensemble tree-based models in capturing the nonlinear and complex relationships between storm characteristics and outage severity. Our analysis supports the deployment of XGBoost for proactive outage forecasting, helping utility companies and emergency planners better prepare for extreme weather events.

Introduction

Power outages caused by severe weather events—such as storms, floods, and hurricanes—can have significant social, economic, and health impacts. Accurate prediction of power outages is essential for utility companies to allocate resources effectively and for emergency response teams to mitigate risk to vulnerable populations. The objective of this study is to develop data-driven models capable of forecasting the number of customers likely to be affected during such events.

We leverage publicly available storm event records and historical power outage data in the U.S. from 2014 to 2023. The raw dataset includes structured and unstructured fields related to the timing, location, type, and magnitude of storm events. A series of preprocessing steps—including missing value imputation, duration calculation, damage normalization, and categorical encoding—are applied to convert the data into a format suitable for machine learning.

To evaluate model performance, we test four predictive algorithms: Lasso (L1-regularized linear regression), Ridge (L2-regularized regression), Random Forest, and XGBoost. We use RMSE, MAE, MSE and Loss function as the primary evaluation metrics. Our pipeline includes cross-validation for hyperparameter tuning and uses consistent train/test splits to ensure comparability. This report details the modeling workflow and presents insights into model performance, with particular attention to the high accuracy and robustness of the XGBoost model.

Separate Data Cleaning and Modeling Data Combination

Outage Data (No need to run again)

We first loaded the outage data and converted the time column into a proper datetime format. We then grouped the data by state, county, and date, summed the number of customers affected, and calculated the value of customers per day.

```r
# Load the data
outage_data <- read.csv("eaglei_outages_2014_2023.csv")
#head(outage_data)

# Rearrange the data
# Step 1: Make sure time column is proper datetime
df <- outage_data %>%
  mutate(run_start_time = ymd_hms(run_start_time),
         date = as.Date(run_start_time))

# Step 2: Group by state + date, sum the customers, and convert to customers.day
summary <- df %>%
  group_by(state, county, date) %>%
  summarise(
    total_customers = sum(customers_out, na.rm = TRUE),
    customers_day = total_customers / 96,
    .groups = "drop"
  )

# Load the new dataset
#write.csv(summary, "outage.csv")
```

Stromevents Data (No need to run again)

Next, we loaded the storm events data and cleaned the time format by adding colons and padding zeros. We joined a timezone offset table to adjust all times to UTC. Then, we created full datetime columns for the start and end of each event and extracted the date. We also standardized state and county names.

To focus on the most common or impactful events, we filtered the dataset to keep only the top 20 most frequent event types and some important ones like Hurricane, Wildfire, and Ice Storm. We removed many unnecessary columns, such as IDs and narrative text fields, and arranged the final dataset by state and date.

Finally, we created a function to calculate the percentage of missing values in each column, allowing us to check data completeness before merging datasets. From the function, We found that "CATEGORY", "TOR_OTHER_CZ_FIPS", "TOR_LENGTH", and "TOR_WIDTH" coulumns had large proportions of missing values ($> 97\%$), so we removed them along with some columns that are meaningless.

```r
# Access the data
storm_data <- read.csv("StormEvents_2014_2024.csv")
#head(storm_data)

# Timezone offset table
timezone_offset <- tibble(
  CZ_TIMEZONE = c("CST", "CDT", "EST", "EDT", "MST", "MDT", "PST", "PDT", "HST", "AKST", "AKDT"),
  UTC_OFFSET = c(-6, -5, -5, -4, -7, -6, -8, -7, -10, -9, -8)
)

# Time format cleaning + UTC adjustment
storm_data <- storm_data %>%
  mutate(
    BEGIN_TIME = str_pad(as.character(BEGIN_TIME), 4, pad = "0"),
    END_TIME = str_pad(as.character(END_TIME), 4, pad = "0"),
    BEGIN_TIME = paste0(str_sub(BEGIN_TIME, 1, 2), ":", str_sub(BEGIN_TIME, 3, 4)),
```

```
    END_TIME = paste0(str_sub(END_TIME, 1, 2), ":", str_sub(END_TIME, 3, 4))
  ) %>%
  left_join(timezone_offset, by = "CZ_TIMEZONE") %>%
  mutate(
    UTC_OFFSET = ifelse(is.na(UTC_OFFSET), 0, UTC_OFFSET),
    MONTH_NAME = str_pad(match(MONTH_NAME, month.name), 2, pad = "0"),
    BEGIN_DAY = str_pad(BEGIN_DAY, 2, pad = "0"),
    END_DAY = str_pad(END_DAY, 2, pad = "0"),
    BEGIN_DATE_TIME = ymd_hm(paste0(YEAR, "-", MONTH_NAME, "-", BEGIN_DAY, " ", BEGIN_TIME)) + hours(UTC
    END_DATE_TIME = ymd_hm(paste0(YEAR, "-", MONTH_NAME, "-", END_DAY, " ", END_TIME)) + hours(UTC_OFFSE
    DATE = as.Date(BEGIN_DATE_TIME),
    state = str_to_upper(STATE),
    county = str_to_lower(str_trim(CZ_NAME))
  )

# Select and clean final stormevents_filtered for join
stormevents_filtered <- storm_data %>%
  filter(EVENT_TYPE %in% {
    top_events <- count(., EVENT_TYPE, sort = TRUE) %>%
      top_n(20, n) %>%
      pull(EVENT_TYPE)
    union(top_events, c("Hurricane", "Wildfire", "Lightning", "Ice Storm"))
  }) %>%
  mutate(
    date = DATE
  ) %>%
  select(date, state, county, EVENT_TYPE, BEGIN_TIME, END_TIME, everything()) %>%
  arrange(state, date)

# Optional: Check NA summary
calculate_na_summary <- function(data) {
  na_info <- sapply(data, function(x) sum(is.na(x)))
  data.frame(
    Column = names(na_info),
    NA_Count = as.vector(na_info),
    Total_Rows = nrow(data),
    NA_Percentage = round((as.vector(na_info) / nrow(data)) * 100, 2)
  ) %>%
    arrange(desc(NA_Percentage))
}

na_summary <- calculate_na_summary(stormevents_filtered)
print(na_summary)

stormevents_filtered <- stormevents_filtered %>%
  arrange(STATE, DATE) %>% select(-BEGIN_YEARMONTH, -END_YEARMONTH, -EPISODE_ID, -EVENT_ID, -STATE, -STA
#head(stormevents_filtered)
```

Combine the datasets by matching state and date. (No need to run again)

We loaded the cleaned outage dataset and standardized the state, county, and date formats to make sure they match with the storm events dataset. Similarly, we updated the storm events dataset to use the same

format for these three key columns.

After making sure the formats were aligned, we merged the two datasets using state, county, and date as the joining keys. This allowed us to combine power outage data with storm event information that occurred on the same day in the same area. Finally, we sorted the merged dataset by state and date for better organization.

```
# Loading cleaning outage set for convenience
outage_data <- read.csv("outage.csv")
#head(outage_data)

# Modify the outage dataset: standardize fields
outage_data <- outage_data %>%
  mutate(
    state = str_to_upper(state),
    county = str_to_lower(county),
    date = as.Date(date)
  )
#head(outage_data)

# Modify the stormevents_filtered dataset: ensure matchable format
stormevents_filtered <- stormevents_filtered %>%
  mutate(
    state = str_to_upper(state),
    county = str_to_lower(county),
    date = as.Date(date)
  )
#head(stormevents_filtered)

# Combine datasets using state + county + date as key
modeling_data <- inner_join(
  outage_data,
  stormevents_filtered,
  by = c("state", "county", "date")
) %>%
  arrange(state, date)

# Write the combination dataset (optional)
#write.csv(modeling_data, "raw_modeling_data.csv", row.names = FALSE)
```

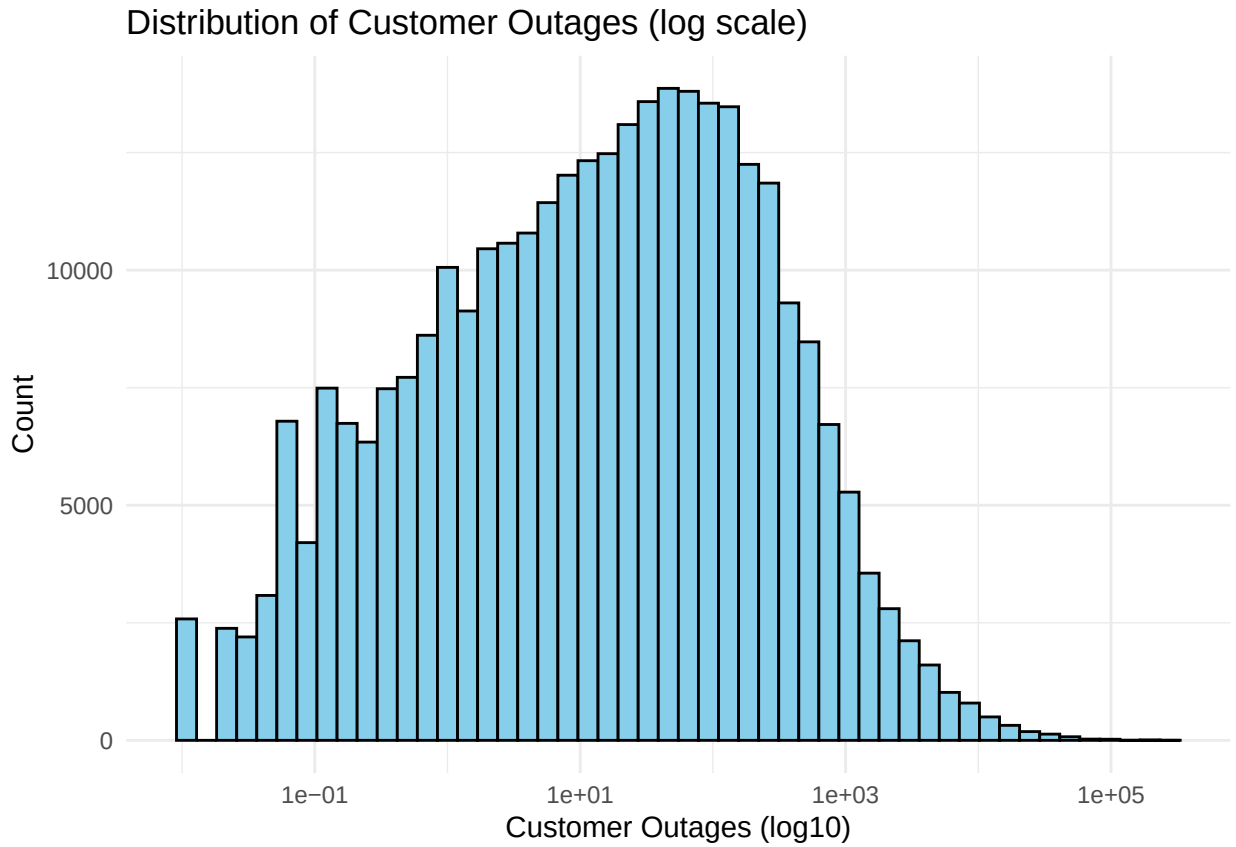
Exploratory Data Analysis (EDA)

In our research, we created 6 plots to visualize the outage and stormevents, including the distribution of outages (log-scaled), the top 15 states with higher outages, the storm events that caused higher outages, the distribution of events duration, the yearly storm events frequency, and the correlation plots for numeric variables.

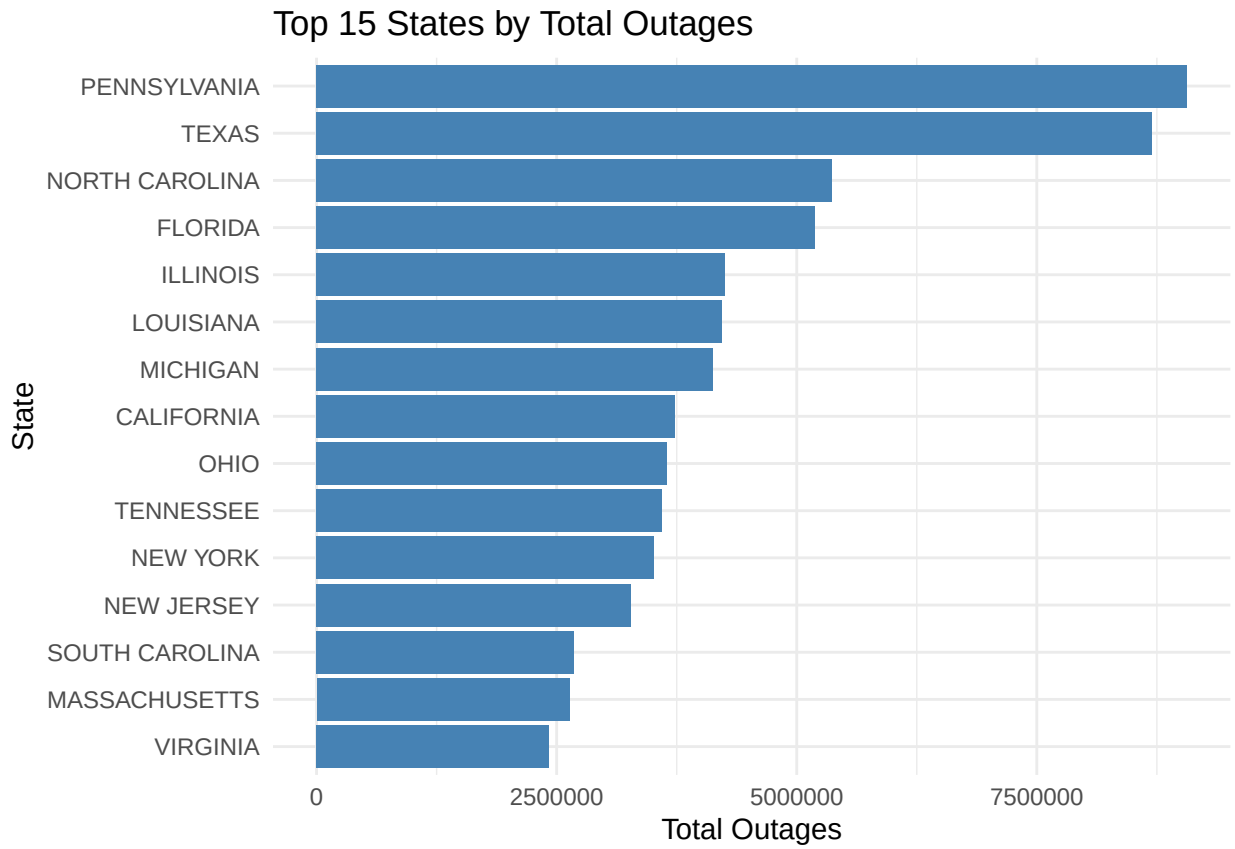
This histogram displays the distribution of customer outages on a log10 scale. The data exhibits a strong right-skew, indicating that most outage events affect a relatively small number of customers, while a few rare events result in extremely large-scale outages.

```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
```

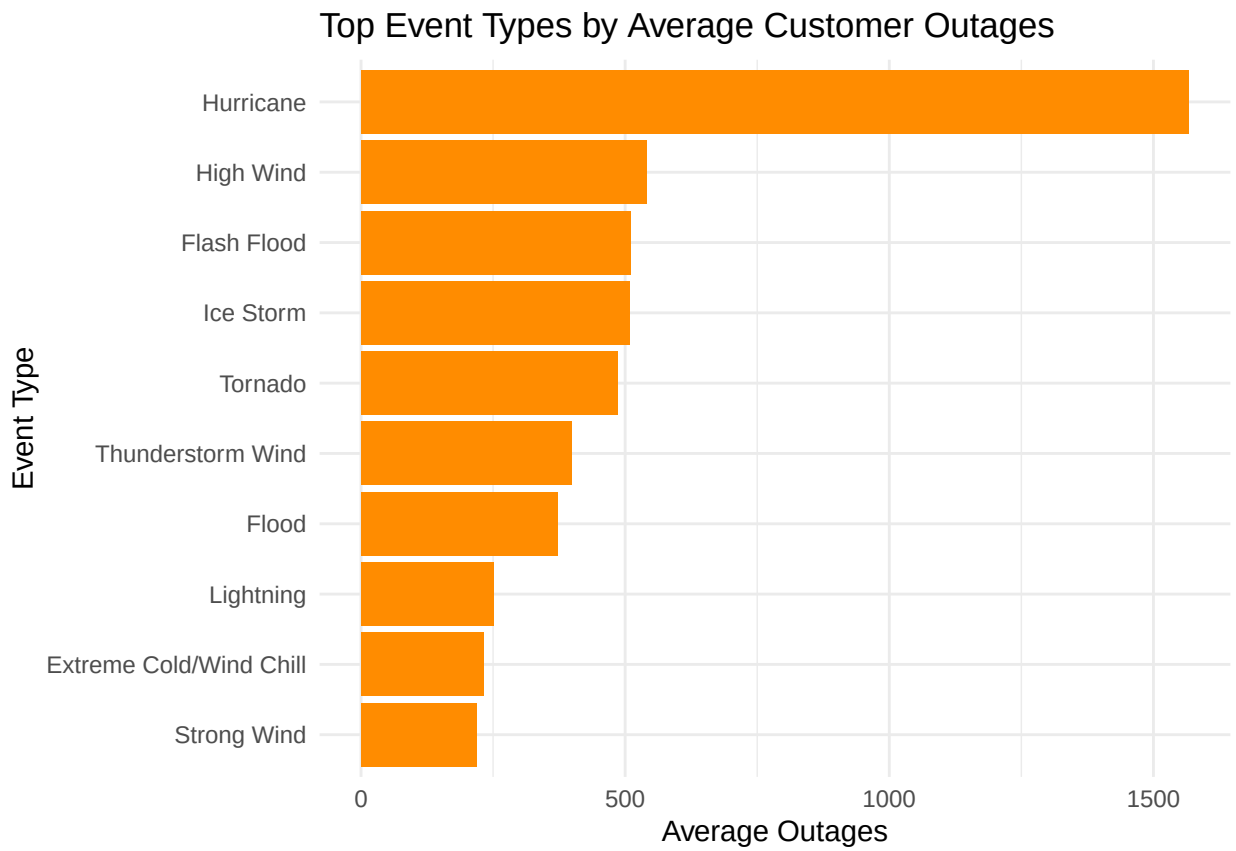
```
## Warning: Removed 5479 rows containing non-finite outside the scale range
## (‘stat_bin()‘).
```



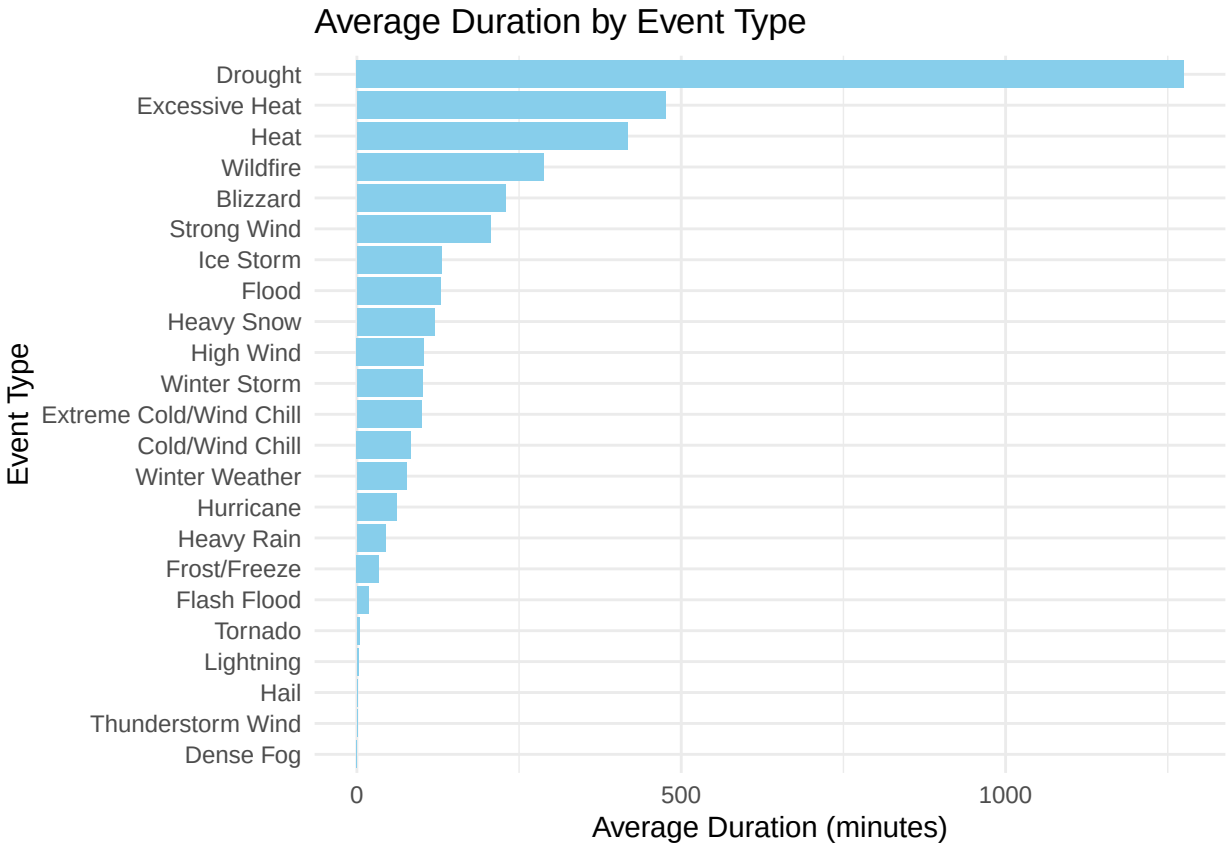
This bar chart highlights the top 15 U.S. states with the highest cumulative number of customer outages. Pennsylvania and Texas lead by a significant margin, each exceeding 7.5 million affected customers, suggesting a high frequency or severity of weather events in those regions. The distribution indicates that power outages are not uniformly spread across the country and are heavily concentrated in certain states, which may be linked to factors such as population density, storm exposure, and infrastructure vulnerability.



This chart ranks weather event types by their average customer impact. Hurricanes stand out as the most disruptive, causing over 1,500 outages on average per event. High wind, flash floods, and ice storms also rank high, suggesting that wind- and water-related phenomena tend to be the most damaging to power infrastructure. The results highlight the importance of event-specific modeling to anticipate outages and prioritize emergency response based on event severity.

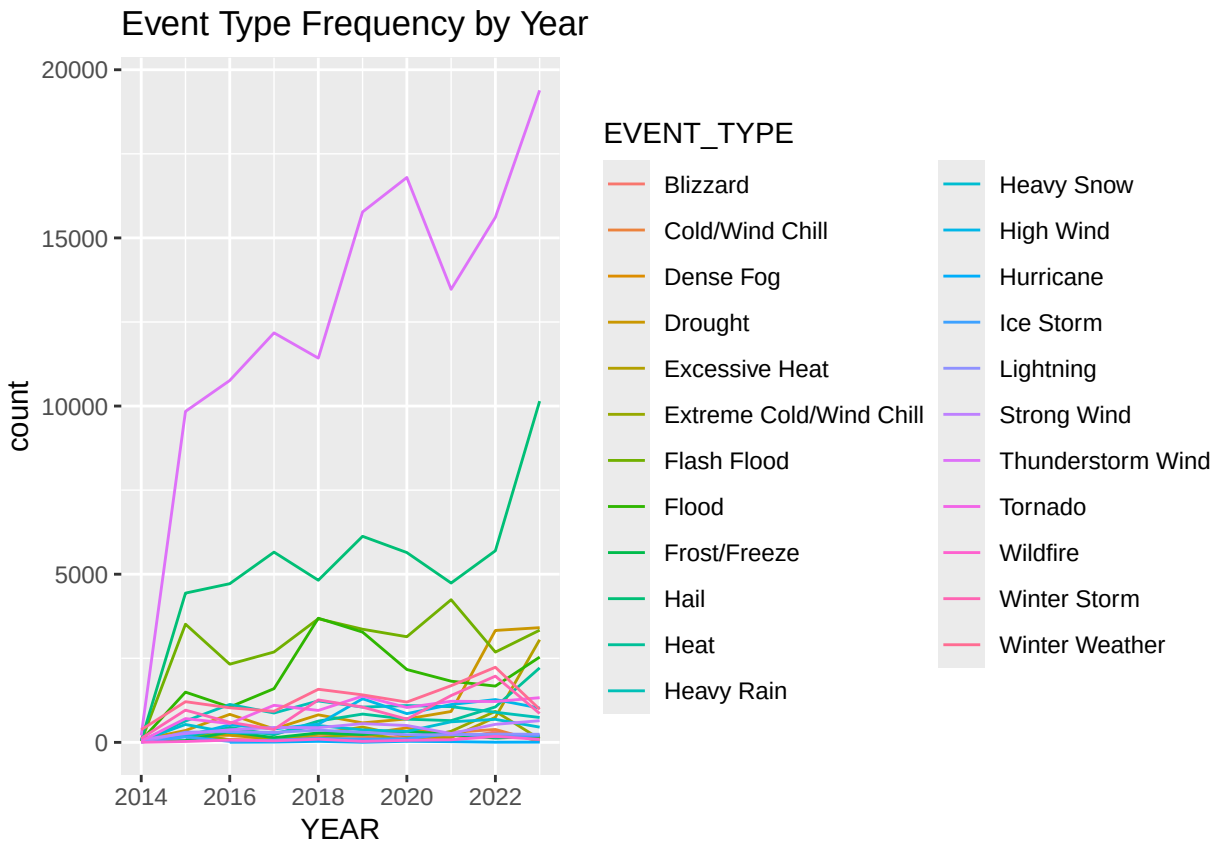


This chart ranks weather event types by their average duration in minutes. Droughts, excessive heat, and wildfires are among the longest-lasting events, often persisting for several days. In contrast, high-impact but short-lived events such as tornadoes, lightning, and hail occur over much shorter time spans. The variation in duration emphasizes the importance of incorporating temporal dynamics into outage prediction models, as different event types pose distinct operational challenges over time.

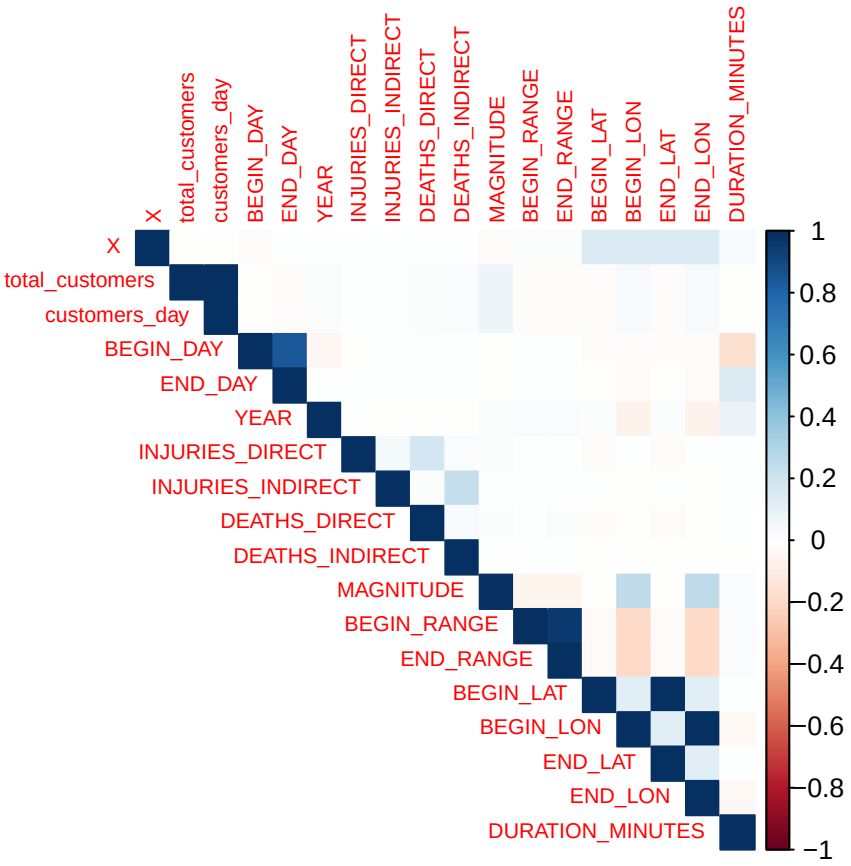


This line chart illustrates the yearly frequency of different weather event types from 2014 to 2023. Thunderstorm wind is by far the most common event type, showing a steady upward trend over the decade. Flash floods and floods are also increasingly prevalent, especially in recent years. The overall rise in event counts across categories may reflect growing climate volatility, improved detection/reporting, or both. These trends highlight the need for adaptive forecasting models that account for shifting weather patterns over time.

```
## `summarise()` has grouped output by 'YEAR'. You can override using the
## `.groups` argument.
```



This plot showed that total_customers and customers_day are perfectly related, which makes sense since one is calculated from the other. Location columns like BEGIN_LAT, END_LAT, BEGIN_LON, and END_LON are also strongly related, showing that the start and end points of each event are often close together. Most of the other columns, such as INJURIES_DIRECT, DEATHS_DIRECT, and MAGNITUDE, are not strongly related to customer outages. This means they might describe different parts of the storm impact.



Modeling

Further Data Cleaning

Before modeling, we double checked the missing values, we treated empty strings as missing values (NA). We then calculated the NA ratio for each column, and we detected 3 high missing proportion columns ($> 40\%$) in our modeling data. In our next step, we manually removed those columns that had too many missing values or were considered irrelevant, such as location descriptions, compass directions, or identifiers. The dimension of our data was 318782 observations of 21 variables. Next, we filled in missing latitude, longitude, and range values using the median of each column to help keep the geographic data consistent.

For our time format, we extracted date-related features such as year, month, day, weekday, and whether the event occurred on a weekend. We also parsed the hour from the storm start time (BEGIN_TIME) and assigned -1 for rows where the time was missing or invalid. During our cleaning process, we detected that there were only 1368 events in 2014, which makes it statistically unreliable for learning meaningful patterns. They might introduce noise and lead to overfitting or misrepresentation in the model, so we remove information about 2014 from our data. The final data had 317414 observations of 26 variables.

```
data <- read.csv("raw_modeling_data.csv")
#head(data)

# Test the NA ratio again
data[data == ""] <- NA
calculate_na_ratio <- function(data) {
```

```
  data.frame(
    Column = names(data),
    NA_Ratio = sapply(data, function(x) mean(is.na(x)))
  ) %>%
    arrange(desc(NA_Ratio))
}

na_ratio_summary <- calculate_na_ratio(data)
print(na_ratio_summary)
```

```
##                                Column  NA_Ratio
## TOR_OTHER_WFO           TOR_OTHER_WFO 0.9960475
## TOR_OTHER_CZ_STATE TOR_OTHER_CZ_STATE 0.9960475
## TOR_F_SCALE               TOR_F_SCALE 0.9700893
## FLOOD_CAUSE               FLOOD_CAUSE 0.8483007
## MAGNITUDE_TYPE         MAGNITUDE_TYPE 0.5706282
## MAGNITUDE                   MAGNITUDE 0.4073693
## BEGIN_RANGE               BEGIN_RANGE 0.2262863
## BEGIN_AZIMUTH           BEGIN_AZIMUTH 0.2262863
## BEGIN_LOCATION         BEGIN_LOCATION 0.2262863
## END_RANGE                   END_RANGE 0.2262863
## END_AZIMUTH               END_AZIMUTH 0.2262863
## END_LOCATION             END_LOCATION 0.2262863
## BEGIN_LAT                   BEGIN_LAT 0.2262863
## BEGIN_LON                   BEGIN_LON 0.2262863
## END_LAT                       END_LAT 0.2262863
## END_LON                       END_LON 0.2262863
## DAMAGE_CROPS             DAMAGE_CROPS 0.1992020
## DAMAGE_PROPERTY       DAMAGE_PROPERTY 0.1920466
## X                                   X 0.0000000
## state                           state 0.0000000
## county                         county 0.0000000
## date                             date 0.0000000
## total_customers       total_customers 0.0000000
## customers_day           customers_day 0.0000000
## EVENT_TYPE                 EVENT_TYPE 0.0000000
## BEGIN_TIME                 BEGIN_TIME 0.0000000
## END_TIME                     END_TIME 0.0000000
## BEGIN_DAY                   BEGIN_DAY 0.0000000
## END_DAY                       END_DAY 0.0000000
## YEAR                             YEAR 0.0000000
## CZ_TYPE                       CZ_TYPE 0.0000000
## WFO                               WFO 0.0000000
## INJURIES_DIRECT       INJURIES_DIRECT 0.0000000
## INJURIES_INDIRECT   INJURIES_INDIRECT 0.0000000
## DEATHS_DIRECT           DEATHS_DIRECT 0.0000000
## DEATHS_INDIRECT       DEATHS_INDIRECT 0.0000000
## UTC_OFFSET                 UTC_OFFSET 0.0000000
```

```
# Remove columns that have higher proportion NAs or are meaningless
columns_to_remove <- c( # List of columns to remove
  "X",                  # Row ID, meaningless
  "FLOOD_CAUSE",
```

```
  "MAGNITUDE",
  "MAGNITUDE_TYPE",
  "TOR_OTHER_WFO",
  "TOR_OTHER_CZ_STATE",
  "TOR_F_SCALE",
  "CZ_TYPE",            # Area type (low variance, not useful)
  "WFO",                # Forecasting Institutions type
  "TOR_OTHER_WFO",
  "BEGIN_AZIMUTH",      # Compass direction (messy, text-based)
  "END_AZIMUTH",        # Compass direction (messy, text-based)
  "BEGIN_LOCATION",     # Text description, not structured
  "END_LOCATION",       # Text description
  "UTC_OFFSET"          # Timezone offset (redundant with 'state')
)

# Remove these columns
data <- data[, !(names(data) %in% columns_to_remove)]

# View the remaining columns
#print(names(data))

# Missing Value Imputation
data <- data %>%
  mutate(across(c(BEGIN_LAT, BEGIN_LON, END_LAT, END_LON), ~ replace_na(., median(., na.rm = TRUE))))

data <- data %>%
  mutate(
    BEGIN_RANGE = replace_na(BEGIN_RANGE, median(BEGIN_RANGE, na.rm = TRUE)),
    END_RANGE   = replace_na(END_RANGE, median(END_RANGE, na.rm = TRUE))
  )

# Rearrange the time format
data <- data %>%
  mutate(
    year = year(date),
    month = month(date),
    day = day(date),
    weekday = wday(date, week_start = 1) - 1,
    is_weekend = ifelse(weekday %in% c(5, 6), 1, 0)
  ) %>%
  select(year, month, day, weekday, is_weekend, everything(), -date)

data$begin_hour <- hour(parse_date_time(data$BEGIN_TIME, orders = "H:M"))
data$begin_hour[is.na(data$begin_hour)] <- -1

# Count the number of occurrences per year
year_counts <- table(data$year)
print(year_counts)
```

```
##
##  2014  2015  2016  2017  2018  2019  2020  2021  2022  2023
##  1368 25867 26059 28008 33488 39071 36458 35043 42191 51229
```

```r
# Remove rows where the year is 2014
data <- data[data$year != "2014", ]
```

Feature Engineering

In this step, we processed the storm start and end times by converting them into time format and calculating the total event duration in minutes. To account for events that cross over to the next day (e.g., from 11 PM to 1 AM), we corrected negative time differences by adding 24 hours. We then combined the day difference and minute difference to calculate the full event duration and removed the original time and day columns afterward.

To clean and standardize the economic impact fields, we created a function that converts damage values (e.g., "10K", "5M") into numeric amounts in dollars. This allowed us to use damage_property and damage_crops as continuous numeric features in our models.

Since most machine learning models require numeric input, categorical variables like event types or states have no natural order or numerical relationship, which may cause interpretation problems for our models. Therefore, we transformed those columns into dummy variables using one-hot encoding. This process creates a new binary column for each category—for example, EVENT_TYPE_Tornado, EVENT_TYPE_Hail, etc.—which allows models to interpret them numerically.

```r
# Time Management
data$BEGIN_TIME <- hm(data$BEGIN_TIME)
data$END_TIME <- hm(data$END_TIME)

# Calculate raw duration
minute_difference <- as.numeric(as.duration(data$END_TIME - data$BEGIN_TIME)) / 60

# Handle negative time differences
minute_difference[minute_difference < 0] <-
  minute_difference[minute_difference < 0] + 24 * 60

# Calculate day span difference
day_difference <- data$END_DAY - data$BEGIN_DAY
data$duration_minutes <- day_difference * 24 * 60 + minute_difference

# Remove useless columns after engineering
data <- data[, !(names(data) %in% c("BEGIN_TIME", "END_TIME", "BEGIN_DAY", "END_DAY"))]
#head(data)

# Damage Convention
convert_damage <- function(x) {
  x <- toupper(trimws(as.character(x)))  # standardize
  sapply(x, function(val) {
    if (is.na(val) || val == "") {
      return(0)
    } else if (grepl("K$", val)) {
      return(as.numeric(gsub("K", "", val)) * 1e3)
    } else if (grepl("M$", val)) {
      return(as.numeric(gsub("M", "", val)) * 1e6)
    } else if (grepl("B$", val)) {
      return(as.numeric(gsub("B", "", val)) * 1e9)
    } else if (!is.na(as.numeric(val))) {
```

```
      return(as.numeric(val))
    } else {
      return(0)
    }
  })
}

data$damage_property <- convert_damage(data$DAMAGE_PROPERTY)
data$damage_crops <- convert_damage(data$DAMAGE_CROPS)

# Perform one-hot encoding for the EVENT_TYPE column
# This will generate binary indicator columns like EVENT_TYPE_Tornado, EVENT_TYPE_Hail, etc.
final_data <- dummy_cols(data, select_columns = c("EVENT_TYPE", "state"), remove_first_dummy = FALSE, r
data_cleaned <- data %>% select(-EVENT_TYPE, -state, -county, -DAMAGE_PROPERTY, -DAMAGE_CROPS)

# Re-bind the cleaned original data with the new dummy variables
final_data <- bind_cols(data_cleaned, select(final_data, starts_with("EVENT_TYPE_"), starts_with("state_
```

This is the function that we used for performance evaluation.

```
# Evaluation function
calc_metrics <- function(prediction, actual) {
  # Ensure numeric vectors
  prediction <- as.numeric(prediction)
  actual <- as.numeric(actual)

  # Mean Squared Error
  mse <- mean((actual - prediction)^2)

  # Mean Absolute Error
  mae <- mean(abs(actual - prediction))


  return(list(MSE = mse, MAE = mae))
}
```

Linear regression with Lasso as the baseline

The lambda value controls how much the model should shrink less important variables. If lambda is too small, the model may overfit the data. If it is too large, the model may underfit and ignore important variables. We use cross-validation to find the best lambda that gives us the most accurate predictions while keeping the model simple.

The best lambda is 64, with the Mean Squared Error (mse) = 3993.253, which told us the average difference between actual and prediction values. And the Mean Absolute Error (MAE) is 13.13931. This means that, on average, the model's predictions differ from the actual outage values by about 13 units.

```
# Prepare Feature Matrix X and Target Variable y
X <- final_data %>%
  select(-customers_day) %>%
  as.matrix()

y <- final_data$customers_day
```

```r
# Split the dataset use the index 0.7
set.seed(123)
train_idx <- sample(1:nrow(X), size = 0.8 * nrow(X))
X_train <- X[train_idx, ]
X_test  <- X[-train_idx, ]
y_train <- y[train_idx]
y_test  <- y[-train_idx]
X_train[is.na(X_train)] <- 0
X_test[is.na(X_test)] <- 0 # Replace NAs as 0

# Fit the Lasso Model
lasso_model <- glmnet(X_train, y_train, alpha = 1)

# Use Cross Validation to select best lambda
cv_model <- cv.glmnet(X_train, y_train, alpha = 1)
best_lambda <- cv_model$lambda.min
cat("Best lambda:", best_lambda, "\n")
```

```
## Best lambda: 64.70454
```

```r
# Prediction
y_pred <- predict(cv_model, s = best_lambda, newx = X_test)
Lasso_res <- calc_metrics(prediction = y_pred, actual = y_test)

# Display results
print(Lasso_res)
```

```
## $MSE
## [1] 3993.253
##
## $MAE
## [1] 13.13931
```

Linear Regression with Ridge

The best lambda is 223, with the Mean Squared Error (MSE) = 39497.93, which tells us the average squared difference between actual and predicted values. The Mean Absolute Error (MAE) is 40.97327, meaning the model's predictions differ from the true outage values by about 41 units on average.

```r
# Fit the Ridge Model
ridge_model <- glmnet(X_train, y_train, alpha = 0)

# Use Cross Validation to select best lambda
cv_ridge <- cv.glmnet(X_train, y_train, alpha = 0)
best_lambda <- cv_ridge$lambda.min
cat("Best lambda:", best_lambda, "\n")
```

```
## Best lambda: 221.967
```

```r
# Prediction
y_pred <- predict(cv_ridge, s = best_lambda, newx = X_test)
Ridge_res <- calc_metrics(prediction = y_pred, actual = y_test)

# Display results
print(Ridge_res)
```

```
## $MSE
## [1] 39497.93
##
## $MAE
## [1] 40.97327
```

XGBoost

XGBoost is an integrated learning algorithm based on gradient boosting, which is suitable for dealing with nonlinear relationships and has strong predictive power. We use early stopping to avoid overfitting.

The performance of the final model on the test set is:

MSE: 1289.72 MAE: 1.26

The results show that XGBoost performs the best among all the models with the lowest prediction error and high accuracy.

```r
X <- final_data %>%
  select(-customers_day) %>%
  mutate(across(everything(), as.numeric)) %>%
  as.matrix()

y <- final_data$customers_day

# Split the data
set.seed(123)
train_idx <- sample(1:nrow(X), size = 0.8 * nrow(X))
X_train <- X[train_idx, ]
X_test <- X[-train_idx, ]
y_train <- y[train_idx]
y_test <- y[-train_idx]

# Create DMatrix
dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dtest <- xgb.DMatrix(data = X_test, label = y_test)

params <- list(
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 6
)

xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
```

```
  watchlist = list(train = dtrain, test = dtest),
  print_every_n = 10,
  early_stopping_rounds = 10

)
```

```
## [1]  train-rmse:2029.706092  test-rmse:1983.964999
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [11] train-rmse:774.428697   test-rmse:733.982626
## [21] train-rmse:313.326309   test-rmse:269.798612
## [31] train-rmse:134.952297   test-rmse:115.794352
## [41] train-rmse:63.571035    test-rmse:63.520413
## [51] train-rmse:33.798914    test-rmse:44.399904
## [61] train-rmse:19.368834    test-rmse:38.588254
## [71] train-rmse:11.561902    test-rmse:37.168393
## [81] train-rmse:7.183767 test-rmse:36.791581
## [91] train-rmse:4.774947 test-rmse:36.260111
## [100]    train-rmse:3.593574 test-rmse:35.881432
```

```
# Make Prediction
xgb_pred <- predict(xgb_model, newdata = dtest)
xgb_res <- calc_metrics(xgb_pred, y_test)

# Display the results
print(xgb_res)
```

```
## $MSE
## [1] 1287.478
##
## $MAE
## [1] 1.256135
```

Discussion

Throughout this project, we encountered several challenges related to data quality and variable representation. A major issue was the handling of the MAGNITUDE and MAGNITUDE_TYPE variables, which were initially considered important indicators of storm severity. However, due to a high percentage of missing values—exceeding 47%—these features were ultimately excluded. Their missingness could not be reliably addressed through imputation, as there were no strong supporting variables to inform a meaningful fill-in strategy.

Another key challenge involved handling high-cardinality categorical variables. The county variable contained hundreds of unique values, and incorporating it through one-hot encoding would have introduced excessive dimensionality, increasing the risk of overfitting and computational inefficiency. Similarly, the WFO (Weather Forecast Office) variable included a large number of unique identifiers tied to regional forecasting offices. While potentially informative, its high cardinality and overlap with state-level information made it difficult to integrate meaningfully. To address these issues, we removed both county and WFO from the modeling dataset and instead retained the state variable, which preserved essential geographic variation in a more manageable form. We applied one-hot encoding to both state and EVENT_TYPE to capture important categorical effects while maintaining a balance between model complexity and interpretability.

We also observed clear differences in outage severity across event types and states. Events such as hurricanes and ice storms produced significantly higher average outages, while certain states experienced disproportionately large disruptions. These patterns were effectively captured by XGBoost, which demonstrated strong predictive ability under complex, nonlinear relationships and high-dimensional inputs. The model successfully leveraged both engineered features (e.g., storm duration, damage estimates) and encoded categorical variables to produce highly accurate outage forecasts.

Conclusion

In this project, we evaluated multiple modeling approaches to predict the number of customers per day using a cleaned and engineered dataset. As a baseline, we implemented linear regression models with Lasso and Ridge regularization. While the Lasso model achieved a moderate level of accuracy (MSE = 3993.25, MAE = 13.14), the Ridge model performed less effectively with a significantly higher error (MSE = 39497.93, MAE = 40.97).

To improve predictive performance, we applied an XGBoost regression model. Compared to the baseline models, XGBoost significantly outperformed both linear models, yielding a much lower MSE of 1289.72 and a MAE of 1.26. The training process also showed consistent improvement and convergence without signs of overfitting, as indicated by early stopping based on test RMSE.

The results demonstrate that XGBoost is highly effective in capturing complex nonlinear relationships in the data, making it the most suitable model among those evaluated.

References

Ghasemkhani B, Kut R A, Yilmaz R, et al. Machine Learning Model Development to Predict Power Outage Duration (POD): A Case Study for Electric Utilities[J]. Sensors (Basel, Switzerland), 2024, 24(13): 4313.

Wang X, Fatehi N, Wang C, et al. Deep Learning-Based Weather-Related Power Outage Prediction with Socio-Economic and Power Infrastructure Data[C]//2024 IEEE Power & Energy Society General Meeting (PESGM). IEEE, 2024: 1-5.

Tervo R, Karjalainen J, Jung A. Short-term prediction of electricity outages caused by convective storms[J]. IEEE Transactions on Geoscience and Remote Sensing, 2019, 57(11): 8618-8626.