Natural-Disaster-Damage-Prediction

Final Report

Author: Group 5

GitHub- https://github.com/siddas18/Natural-Disaster-Damage-Prediction/tree/main/FinalProject-Group5

Introduction

A new research field emerged in climate science in the early 2000s that wanted to explore the increasing prevalence of extreme weather events like floods, storms, cyclones, etc. The field is known as "extreme event attribution" and has gained momentum in recent years in media in addition to the scientific world. There is mounting evidence that human activity is to blame for the increased risk of these extreme weather-type events. Researchers have also given importance to analyzing the economic costs linked to the human contribution to weather events. A study in 2020 approximated that nearly $67bn of damages caused by Hurricane Harvey in 2017 could attribute to human influences on climate. There are numerous methods to carry out attribution analysis. One way is to record instances of an extreme weather event and see their frequencies change with changes in environmental factors. We aim to build a model that accurately predicts the estimated damage to property while considering various event-related factors, in addition to external factors that might be influencing the extent of the damage.

NOAA (National Oceanic and Atmospheric Administration) records the occurrence of storms and other significant weather phenomena having sufficient intensity to cause loss of life, injuries, significant property damage, and/or disruption to commerce.

Dataset

For this project, we have used publicly available data from the National Oceanic and Atmospheric Administration (NOAA) that contains event details on disaster incidents occurring in the US ranging from 1950 to August 2021. Some of the variables that we use from this dataset are as follows:

- begin and end date-time of event
- state where the event occurred
- the type of event (Hail, Storm, Drought, etc.)
- number of injuries and deaths
- starting and ending latitudes and longitudes of the event

The complete data dictionary for reference is accessible through this link.

We have also pulled in environmental indicators from yearly data collected by the United States Environmental Protection Agency (EPA). We have joined this data as additional information against the event year. The datasets that we have considered from the EPA source are as follows:

- emissions of greenhouse gases from 1990 to 2019
- events of heavy precipitation by land area percentage
- yearly earth surface temperature
- CSIRO and NOAA data for yearly sea-level changes
- variations in average seasonal temperature for fall, winter, summer, and spring
- artic ice coverage in March (yearly high) and September (yearly low)
- Glacier mass balance and number of observed glaciers

Additional dataset information is available at this source.

Algorithms Used

1. Linear Regression

It is a method to model a relationship between one or more independent variables and a response variable by fitting a linear equation on the observed data. Regression tells us the value of the response variable for an arbitrary explanatory variable value. The regression equation is:

$\hat{y} = b0 + b1\ x1 + b2\ x2 + \ldots$ where
b_0: intercept
b_i: slope/rate of change

2. Bootstrap aggregation

Bootstrapping is a sampling technique to create subsets of observations from the original data and is also known as bagging. In this technique, a generalized result combines the results of various predictive models. The subset size for bagging may be smaller than the original dataset.

3. Random Forest Regression

It is a supervised machine learning algorithm that uses bagging to solve regression and classification problems. The algorithm works by training multiple decision tree estimators concurrently and outputting the mean or mode of all the individual predictions. It helps against individual trees overfitting the data and getting stuck in locally optimal solutions.

4. Extreme Gradient Boosting Regression

Gradient boosting is a class of ensemble machine learning algorithms constructed from decision tree models. It fits the model using any arbitrary differentiable loss function and gradient descent optimization algorithm. This technique is known as gradient boosting as we minimize the loss gradient while training the model.
Extreme Gradient Boosting, or XGBoost for short, is an efficient open-source implementation of the gradient boosting algorithm. XGBoost is a powerful approach for building supervised regression models.

Figure 1

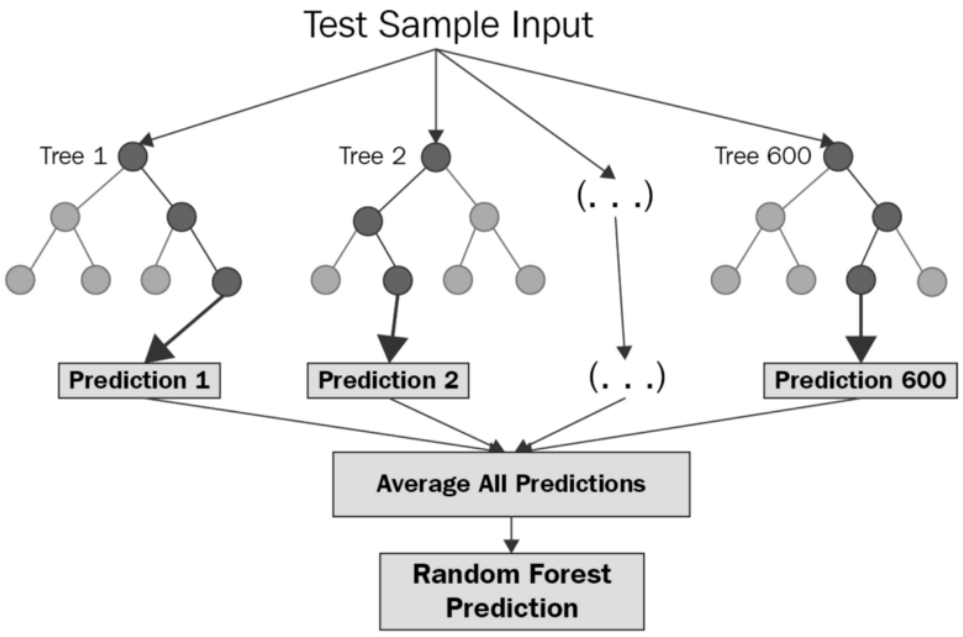
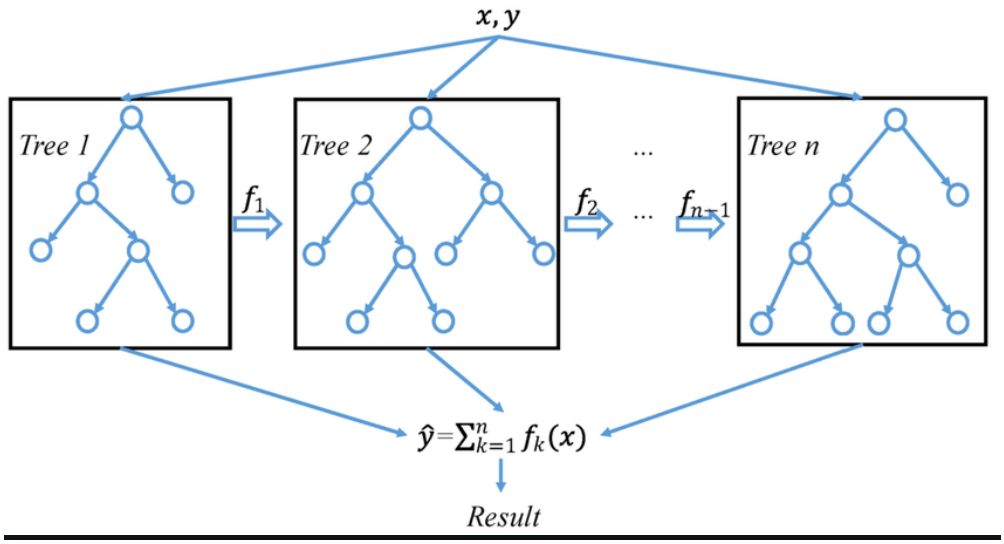


Figure 2



Experimental Setup

1. Extraction

a. **NOAA Data**:

The NOAA data files are extracted from this link and have the following naming structure: StormEvents_details-ftp_v1.0_d1950_c20210803.csv.gz. The files are then concatenated into one to form our source data frame. We save this as a pickle file. This is done through get_NOAA_data() function.

b. **EPA Data**:

We use Pandas to read the various EPA data CSV files and collate them into one. We interpolate the missing data ranging back to the year 1950 by using the impute_EPA_data () function. We use the interp1d method from SciPy to get the extrapolated variable values.

2. Preprocessing

a. Data Manipulation Functions

i. replace_str2num ()

- The replace_str2num () function cleans up the DAMAGE_CROPS and DAMAGE_PROPERTY variables to convert them into numeric values.

ii. Winds() and hail ()

- The winds () and hail () functions split the MAGNITUDE variable based on the values of MAGNITUDE_TYPE into WIND_SPEED and HAIL_SIZE.

iii. Missing_swap ()

- The missing_swap () function imputes missing values for variables where the counterpart has valid values. For example, if the BEGIN_LAT is present and the END_LAT is not present, we fill it with the BEGIN_LAT value.

iv. calc_duration ()

- The calc_duration() function calculates the time difference between the event start and end.

v. geo_distance ()

- The geo_distance() function uses the Haversine formula to calculate the geographical distance covered by the event (Tornado, etc.)

vi. Dict_mapping ()

- The dict_mapping() function replaces junk values from CZ_TIMEZONE, BEGIN_AZIMUTH, and END_AZIMUTH with appropriate values.

vii. impute_NOAA_data ()

- We use the EVENT_TYPE variable to derive three variables: COLD_WEATHER_EVENT, WINDY_EVENT, and WATER_EVENT, based on keywords like Snow, Storm, Hurricane, etc.
- Tor_Scale () converts the values of F_Scale for the tornado strength into numeric values.
- Then, we fill the missing values in continuous variables with 0 and the categorical variables by N/A.

Finally, we join the entire data into one data frame and remove the outliers from all the numerical variables.

3. Modeling

3.1 Advanced feature engineering:

1. Encoding categorical columns:

For categorical columns, based on type of data available, we did label encoding and one-hot encoding.

Columns for label encoding: 'CZ_NAME', 'BEGIN_LOCATION', 'END_LOCATION', 'TOR_OTHER_CZ_STATE', 'TOR_OTHER_CZ_NAME'

Columns for one-hot encoding:
'STATE', 'MONTH_NAME', 'EVENT_TYPE', 'CZ_TYPE', 'CZ_TIMEZONE', 'BEGIN_AZIMUTH', 'MAGNITUDE_TYPE', 'FLOOD_CAUSE', 'TOR_F_SCALE', 'END_AZIMUTH'

2. **Imputation of logically important columns:**

For column "DAMAGE_CROPS", we believed instead of simply removing all NAN's it is better to impute them with the average value of DAMAGE_CROPS per EVENT.

3. **Split the data into training and validation sets:**

Using from sklearn. model_selection import train_test_split, able to create the training data and validation data sets.

4. **Standardize and normalize the data**:

Using, from sklearn. preprocessing import StandardScaler, able to standardize the training data before running the regression models. In addition to this, using mean and standard deviation I normalized the training data.

3.2 Training Models:

1. **Linear Regression**:

Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

```
# Model 1 - Linear Regression

reg_lr = LinearRegression()
reg_lr.fit(X_train, y_train)

# make predictions for test data
y_pred = reg_lr.predict(X_test)

MSE = mean_squared_error(y_pred, y_test)
print("LR Mean Square Value", MSE)

print("LR Training R-square value", reg_lr.score(X_train, y_train))

print("LR R-Square Value", r2_score(y_test, y_pred))

pickle.dump(reg_lr, open('Data/LR_Model.pkl', 'wb'))
```

2. Random Forest:

A supervised learning algorithm that is based on the ensemble learning method and many Decision Trees. Random Forest uses a Bagging technique, so all calculations are run in parallel and there is no interaction between the Decision Trees when building them.

```
# Model 2 - Random Forest

reg_rf = RandomForestRegressor(n_estimators=100, oob_score='TRUE', n_jobs=-1, random_state=50, max_features="auto",
                               min_samples_leaf=50)

# perform training
reg_rf.fit(X_train, y_train)

# make predictions

# prediction on test using all features
y_pred = reg_rf.predict(X_test)

MSE = mean_squared_error(y_pred, y_test)
print("RF Mean Square Value", MSE)

print("RF Training R-square value", reg_rf.score(X_train, y_train))

print("RF R-Square Value", r2_score(y_test, y_pred))

pickle.dump(reg_rf, open('Data/RF_Model.pkl', 'wb'))
```

3. XGBoost Regressor:

Gradient boosting refers to a class of ensemble machine learning algorithms constructed from decision tree models. Models are fit using loss function and gradient descent algorithm. This gives the name, “gradient boosting,” as the loss gradient is minimized as the model is fitted. Extreme Gradient Boosting, or XGBoost, is an efficient implementation of the gradient boosting algorithm and is a powerful approach for building supervised regression models.

```
# Model 3 - XGBoost

xgb = XGBRegressor(learning_rate=0.01, subsample=0.7, max_depth=5, n_estimators=500, colsample_bytree=0.8)
xgb.fit(X_train, y_train, eval_set=[(X_train, y_train)])

# make predictions for test data
y_pred = xgb.predict(X_test)
y_pred_score = [round(value) for value in y_pred]

MSE = mean_squared_error(y_pred, y_test)
print("XGB Mean Square Value", MSE)

print("XGB Training R-square value", xgb.score(X_train, y_train))

print("XGB R-Square Value", r2_score(y_test, y_pred))
```

4. Ensemble Model:

For ensemble learning, we've used the sklearn function "VotingRegressor". Simply put, this regressor uses individual model predictions and then averages them out to form a final prediction.

```
    # Model 4 - Ensemble Model

    # create a dictionary of our models
    estimators = [('LR', reg_lr), ('RF', reg_rf), ('XGB', xgb)]

    # create our voting classifier, inputting our models
    ensemble = VotingRegressor(estimators)

    # fit model to training data
    ensemble.fit(X_train, y_train)
    # test our model on the test data

    y_pred = ensemble.predict(X_test)

    MSE = mean_squared_error(y_pred, y_test)
    print("Ensemble Mean Square Value", MSE)

    print("Ensemble Training R-square value", xgb.score(X_train, y_train))

    print("Ensemble R-Square Value", r2_score(y_test, y_pred))

    pickle.dump(ensemble, open('Data/Ensemble_Model.pkl', 'wb'))
```

3.3 Additional options tried to increase model efficiency

Outlier Removal:

Using the Inter-Quartile Range method, I was able to identify the outliers and remove them. This method helped improve the R-square of the model by 5%.

```
# Removing Outliers

NOAA_df.shape

Quart1 = NOAA_df.quantile(0.25)
Quart3 = NOAA_df.quantile(0.75)
Range = Quart3 - Quart1

NOAA_df = NOAA_df[~((NOAA_df < (Q1 - 1.5 * Range)) |(NOAA_df > (Q3 + 1.5 * Range))).any(axis=1)]

NOAA_df.shape
```

Principal Component Analysis:

Principal Component Analysis, or PCA, is a very popular dimensionality reduction technique. PCA is trying to rearrange the features by their linear combinations. One characteristic of PCA is that the first principal component holds the most information about the dataset. The second principal component is more informative than the third, and so on.

```
# PCA code if we use one-hot encoding

pca = PCA()
# pca = PCA(n_components=25)
pca.fit(X_train)


X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# print(X_train.shape)
# pca.components_
```

K-Means clustering for feature engineering

After multiple models tuning and feature creation iterations, the team could observe the increase in the model performance plateaued. The team decided to take help of the unsupervised K-Mean clustering model to create new feature hoping to increase the model performance.

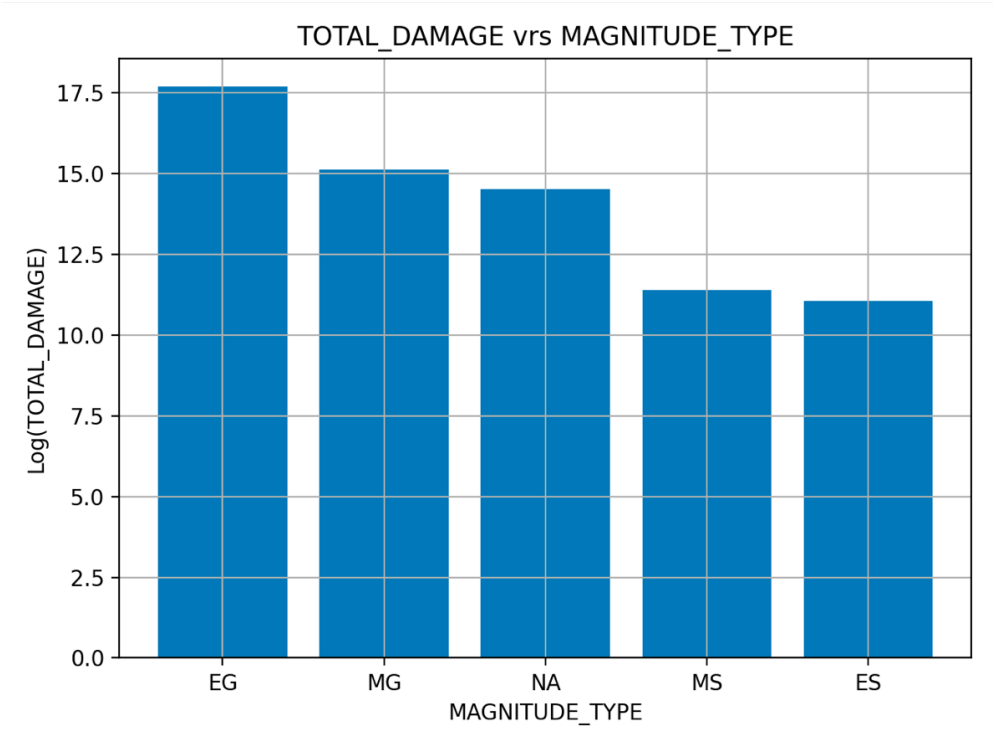
```
#Feature engineering: kmeans
from sklearn import preprocessing
from sklearn import metrics
from sklearn.cluster import KMeans, MiniBatchKMeans

df_train.reset_index()
fe = ['GEO_DISTANCE','DURATION_OF_STORM','WIND_SPEED','INJURIES_DIRECT', 'INJURIES_INDIRECT', 'DEATHS_DIRECT','DEATHS_INDIRECT','YEAR','DAMAGE_CROPS']
X_fe = df_train[fe]

X_fe.reset_index()
X_fe_scale = (X_fe - X_fe.mean(axis=0)) / X_fe.std(axis=0)
X_fe.reset_index()

###
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
Sum_of_squared_distances = []
silhouette_avg = []
for num_clusters in range_n_clusters:
    print("Cluster: "+str(num_clusters))
    # initialise kmeans
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(X_fe_scale)
    cluster_labels = kmeans.labels_

    # silhouette score
    silhouette_avg.append(metrics.silhouette_score(X_fe_scale, cluster_labels))
    Sum_of_squared_distances.append(kmeans.inertia_)
# plt.plot(range_n_clusters,silhouette_avg)
plt.plot(range_n_clusters,Sum_of_squared_distances)

plt.show()
###
```

Results

1. **Exploratory Data Analysis**

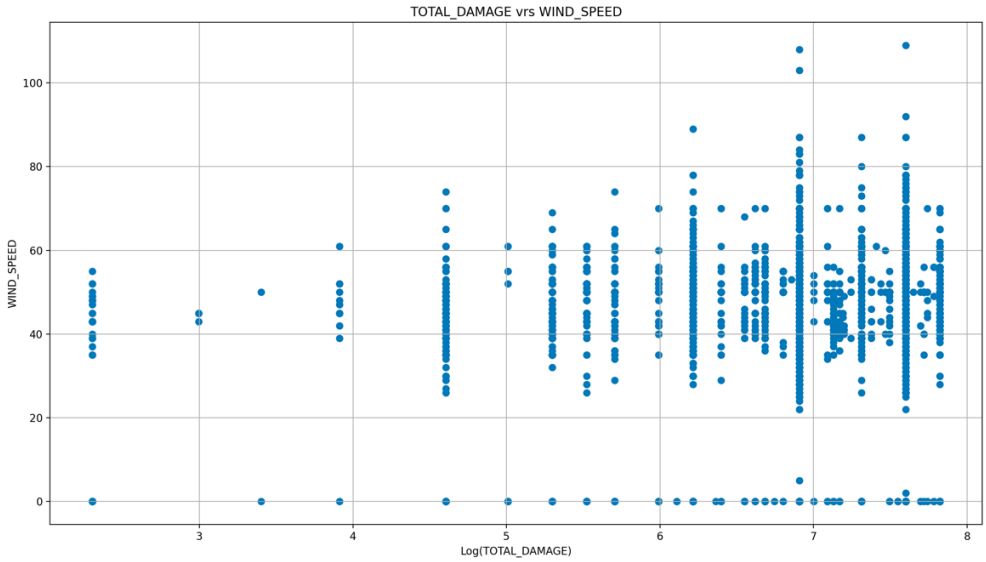
After obtaining the cleaned dataset, our objective was to get better insights about our data so that we can fix any data inconsistencies and get a clearer picture of event attributes that are explaining the variance in our target variable TOTAL_DAMAGE. We start by plotting the distributions of our target variable against various features to gauge their overall importance in our final model. For our plots, we take the logarithm of the total damage sum across different groups to plot our graphs. Here to plot our features with respect to target variable we have performed log transformation over our target variable.

Plot 1: TOTAL_DAMAGE V/S MAGNITUDE_TYPE



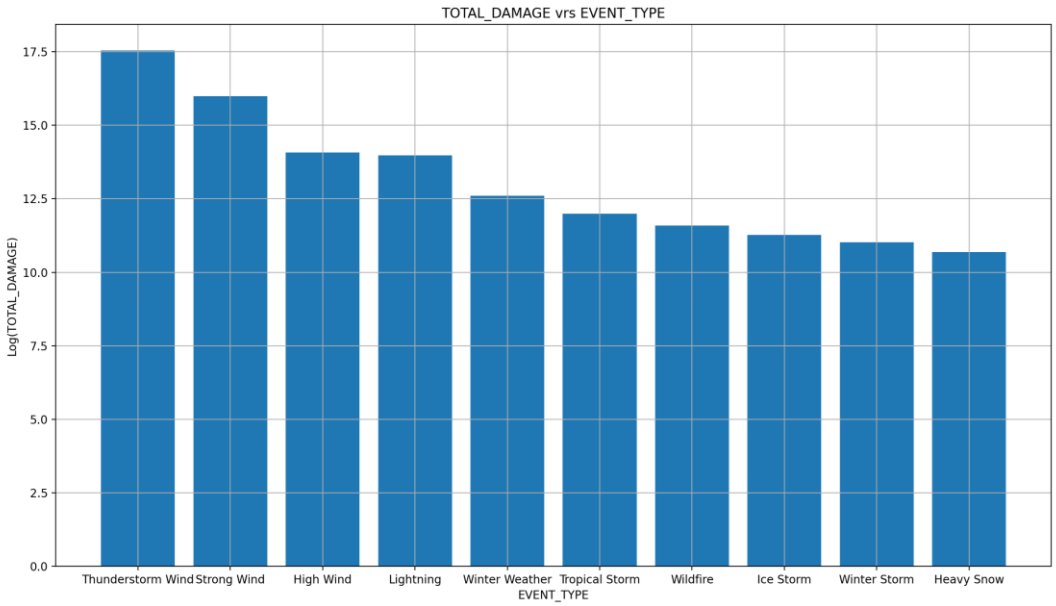
Our target variable shows that Wind estimated gust (EG) has the highest total damage and Estimated Sustained Wind (ES) has the least.

Plot 2: TOTAL_DAMAGE V/S WIND_SPEED



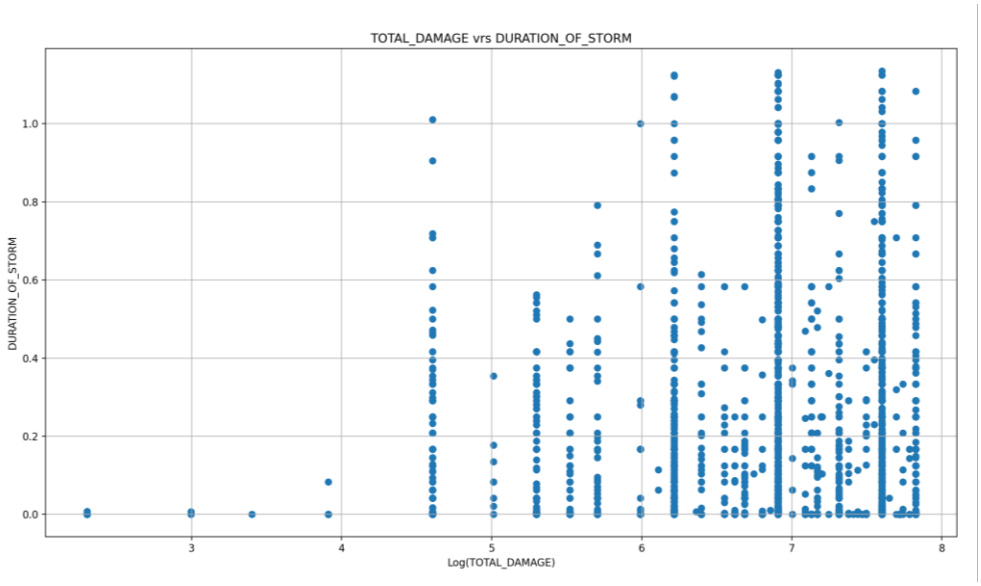
There's a slight positive correlation between wind speed and total damage. Maximum damage is caused by wind speed ranging between 20 knots to 80 knots.

Plot 3: TOTAL_DAMAGE V/S EVENT_TYPE



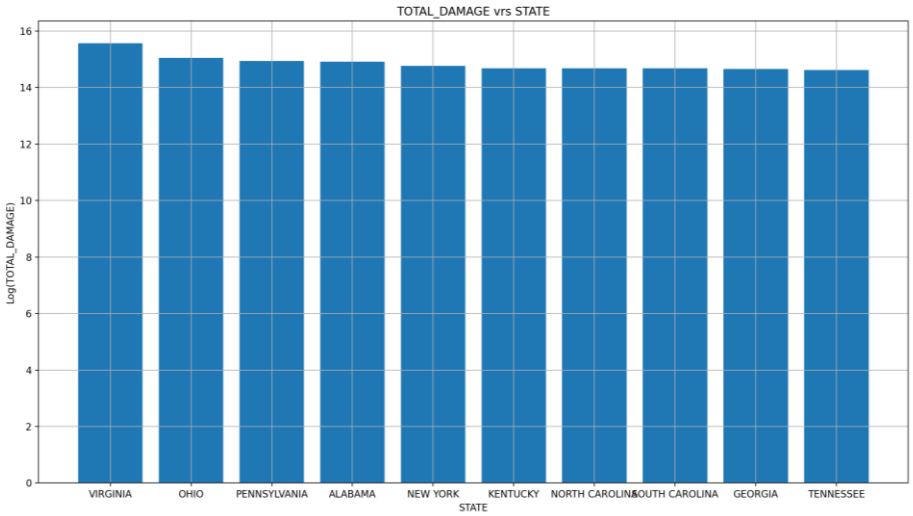
The graph depicts the top 10 events that have the highest total damage. Wind events seem to have high damage followed by the winter related events.

Plot 4: TOTAL_DAMAGE V/S DURATION_OF_STORM



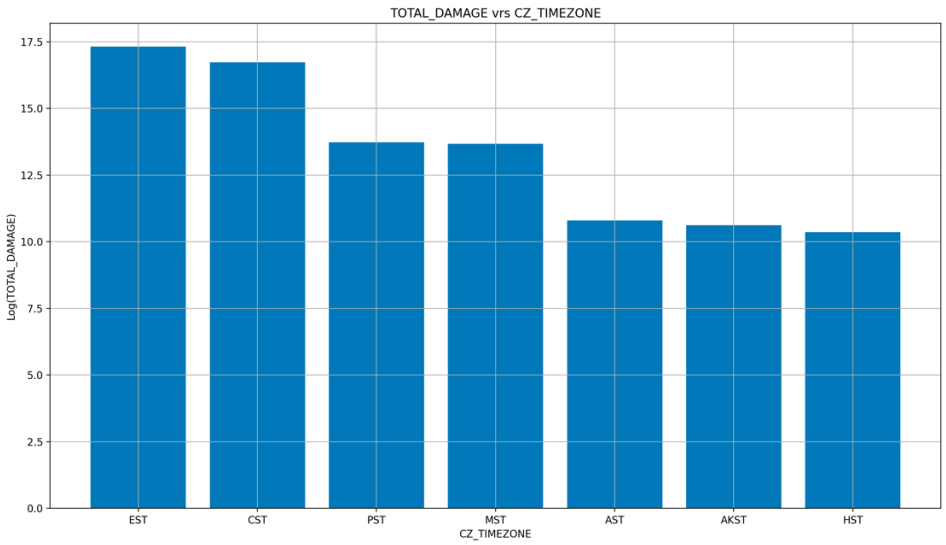
There's a positive correlation between duration of storm and total damage.

Plot 5: TOTAL_DAMAGE V/S STATE



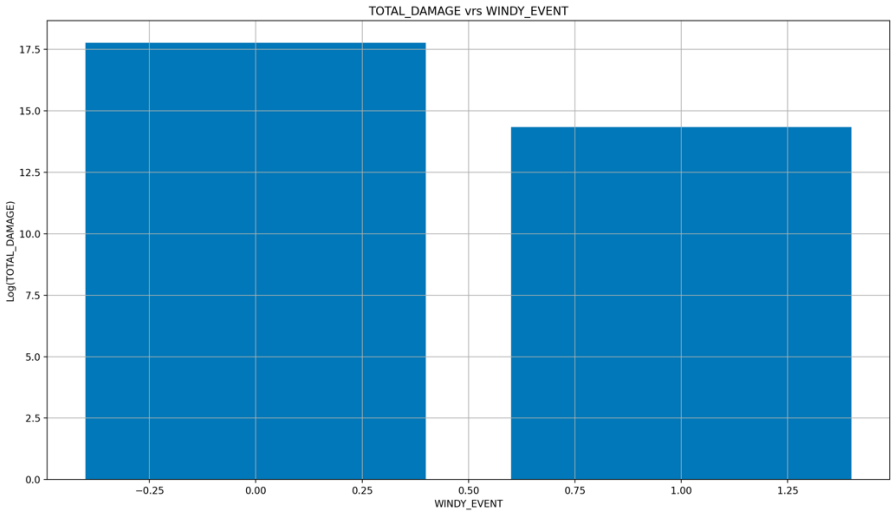
The graph depicts the top 10 states that have the most total damage Virginia being the highest.

Plot 6: TOTAL_DAMAGE V/S CZ_TIMEZONE



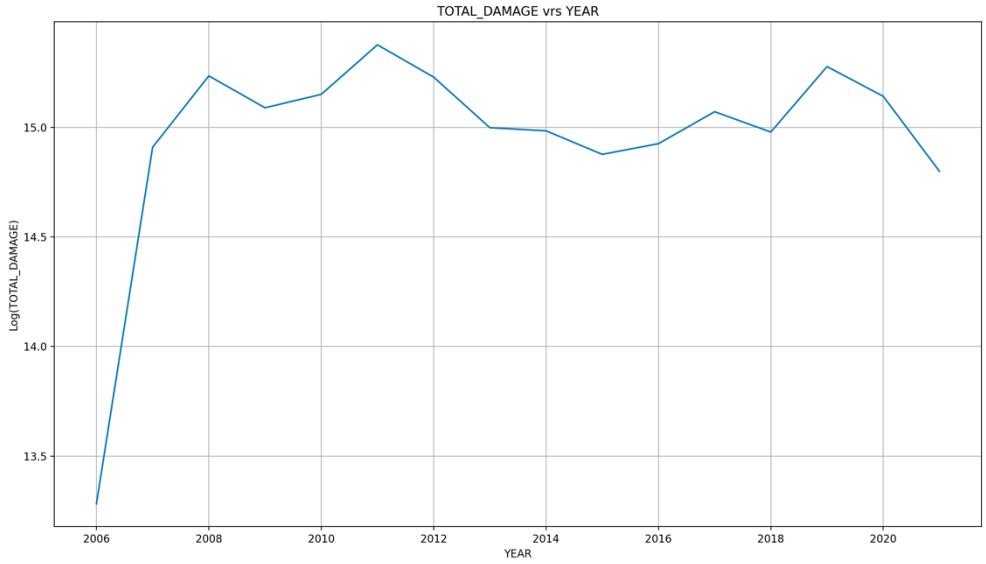
The time zone classification shows the region where the total damage was the highest. Eastern and Central region being the highest.

Plot 7: TOTAL_DAMAGE V/S WINDY_EVENT



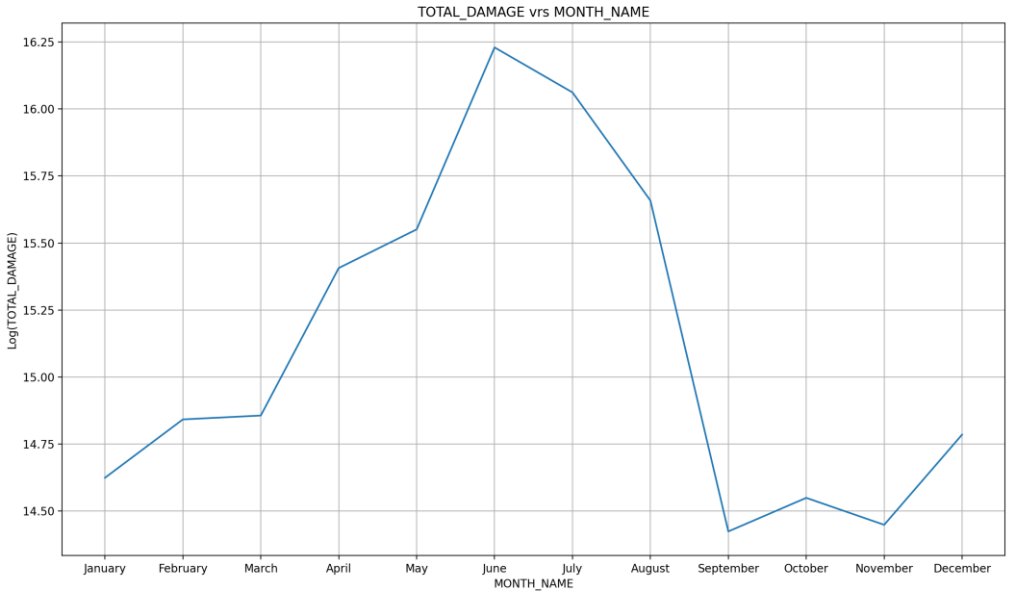
This graph depicts the total damage occurred due to a windy event when compared to a hail event.

Plot 8: TOTAL_DAMAGE V/S YEAR



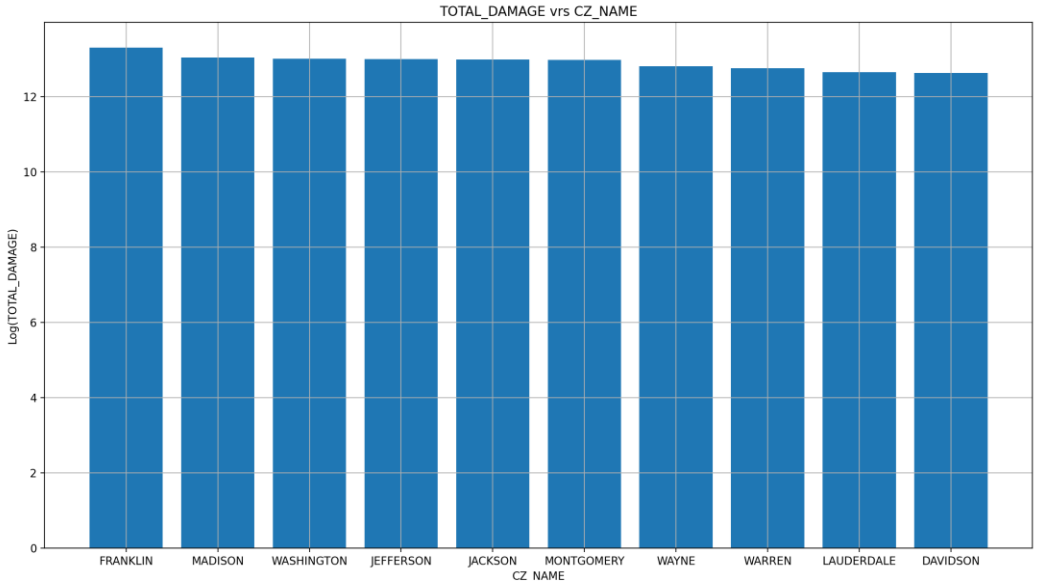
This graph shows the trend of total damage with respect to year. Here the graph is from 2006 (as before that NOAA didn't capture all the event types) which shows the increase in total damage with 2011 showing the highest total damage.

Plot 9: TOTAL_DAMAGE V/S MONTH_NAME



The months from May to August show the highest total damage as these months are the months for tornado season.

Plot 10: TOTAL_DAMAGE V/S CZ_NAME



The graph shows the top 10 counties with high total damage, Franklin being the highest.

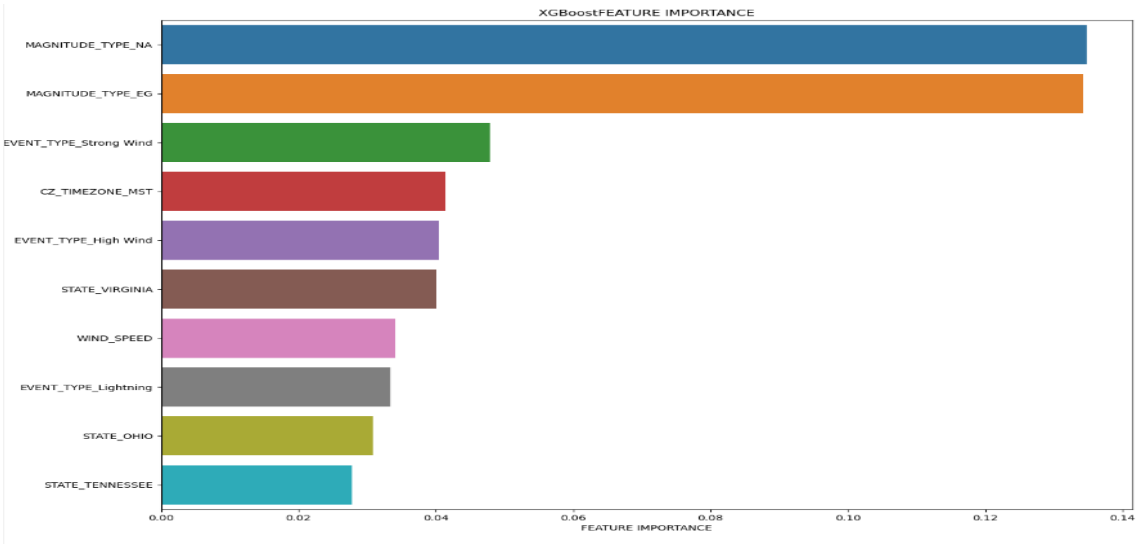
2. Modeling

We compare the performance of the different models by making use of the following metrics:

1. Mean squared error
2. Train R-squared value
3. Test R-squared value

2.1 XGBoost Regressor:

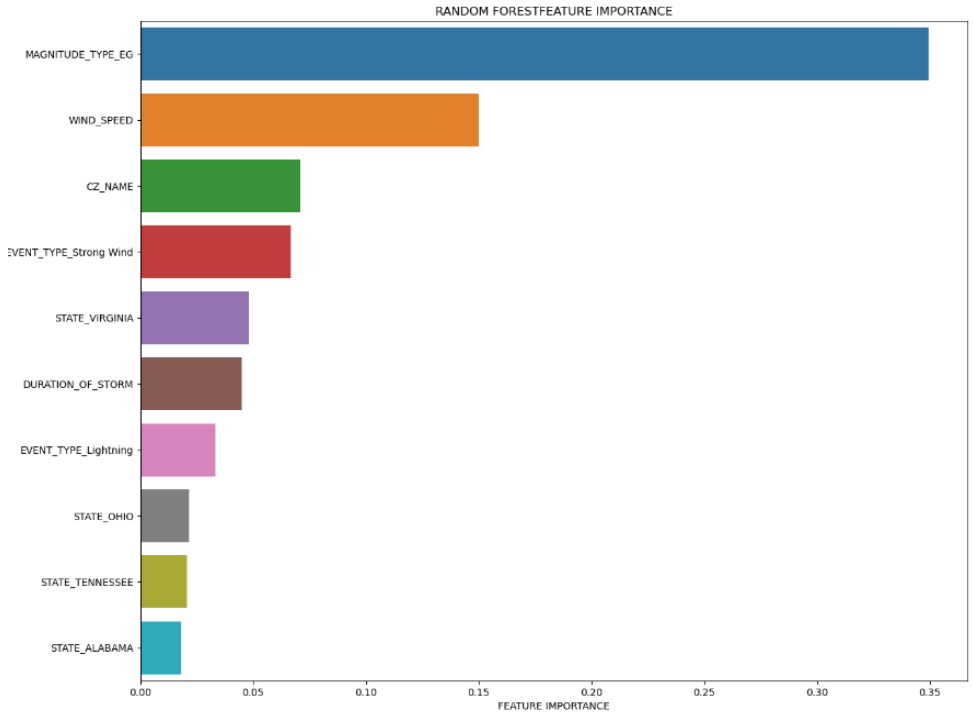
For XGBoost Regressor, our feature importance after training the model is given below. The RMSE values and train and test R-squared values are also included in the output.



We see that some of our important features from the model come out to be Magnitude_Type, Event_Type, State, CZ_Timezone, and Wind_Speed. Additionally, we get an R-squared value of around 44% which is not that great.

2.2 Random Forest:

For Random Forest Regressor, our feature importance after training the model is given below. The RMSE values and train and test R-squared values are also included in the output.



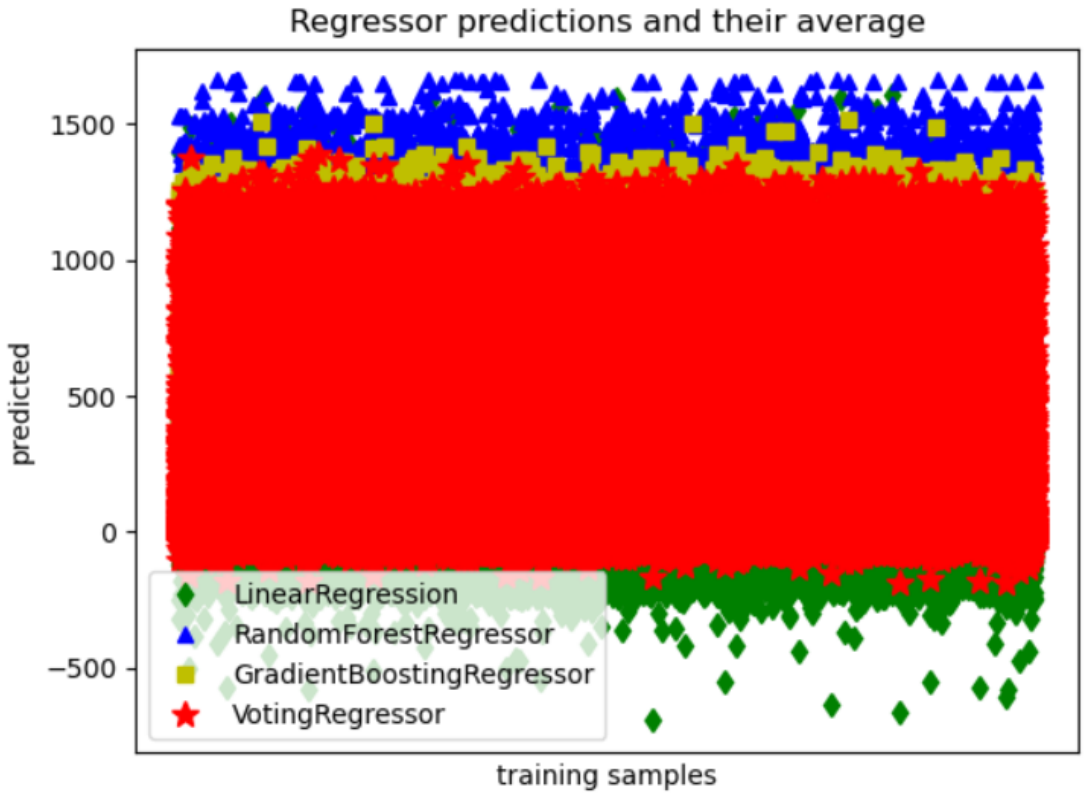
Similarly, we see that the important features are Magnitude_Type, Wind_Speed, State, Event_Type, Duration_of_Storm, and CZ_Timezone. We get better performance with R-squared value of about 50%.

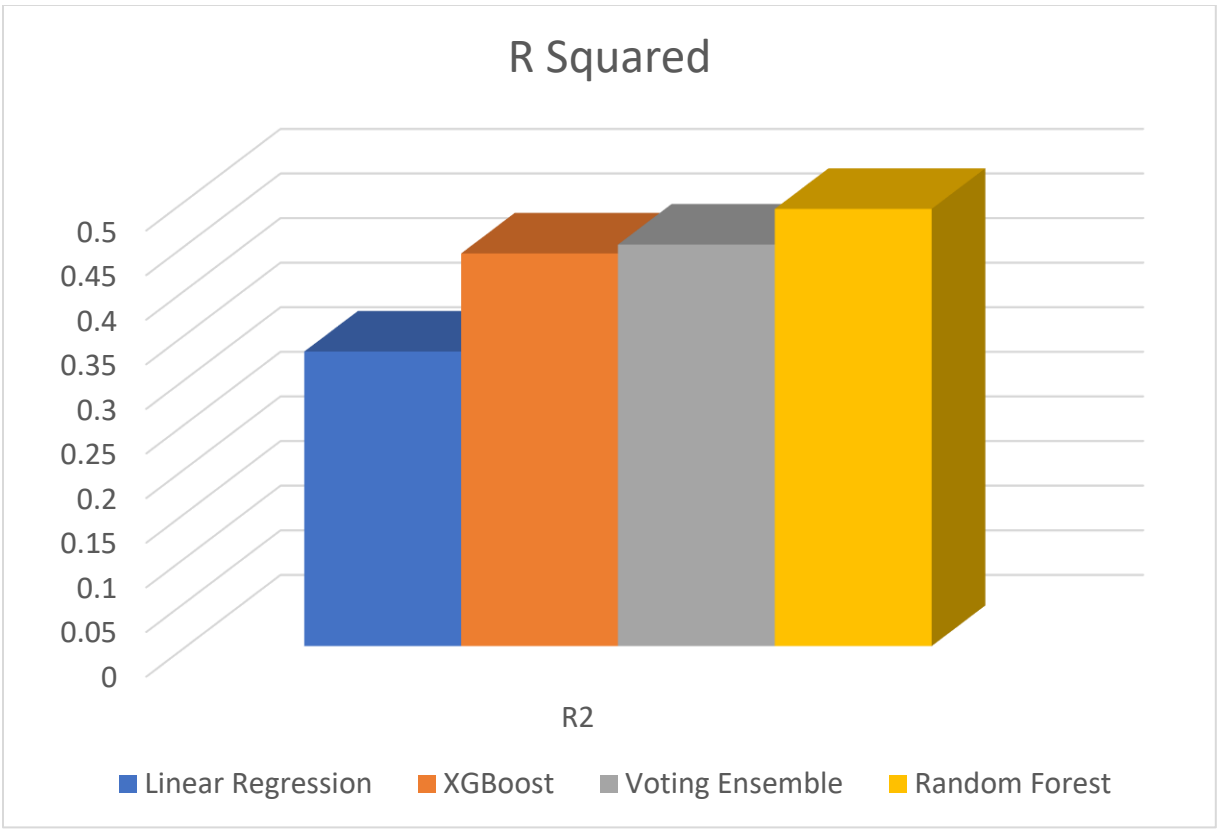
2.3 Rest of Models:

We also ran Linear Regression and Ensemble Model to predict the TOTAL_DAMAGE but both these models had a lower R-squared value than Random Forest.

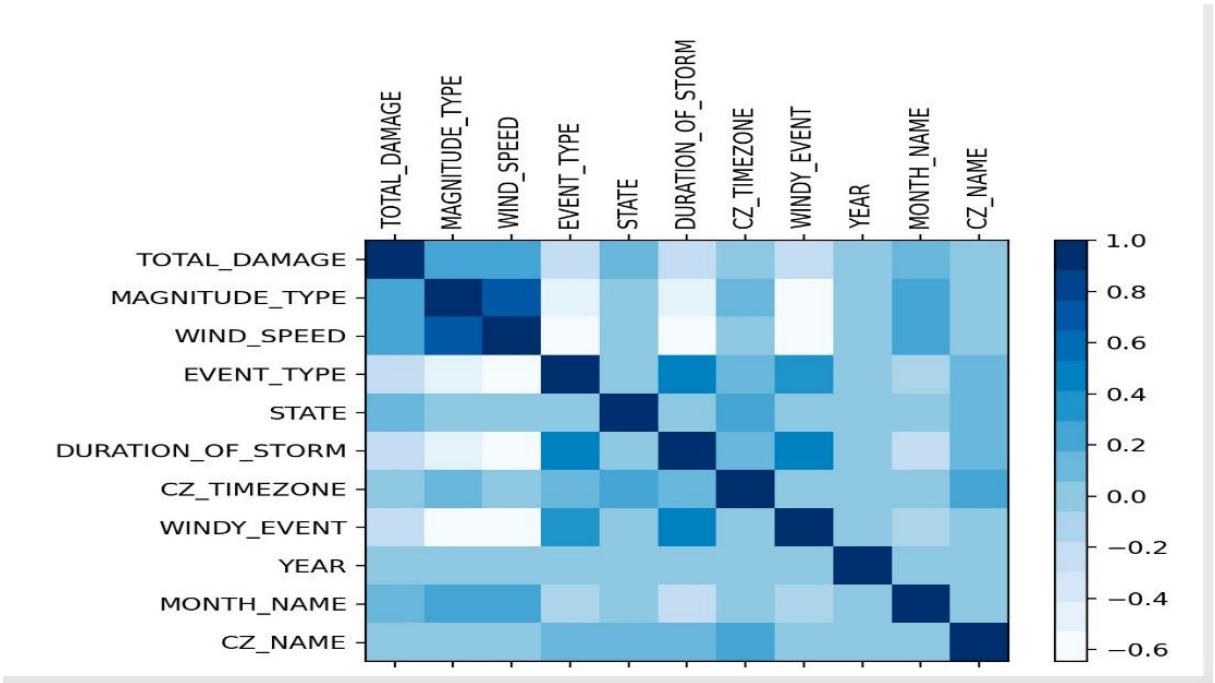
A detailed comparison of the models can be found below:

Model Name	Mean Square Error	Training R-squared Error	R-square error
Linear Regression	174992.1851	**33.70%**	**33.54%**
Random Forest	132421.8389	**52.93%**	**49.71%**
XGBoost	146355.2002	**45.22%**	**44.42%**
Ensemble	143642.2061	**44.39%**	**45.45%**



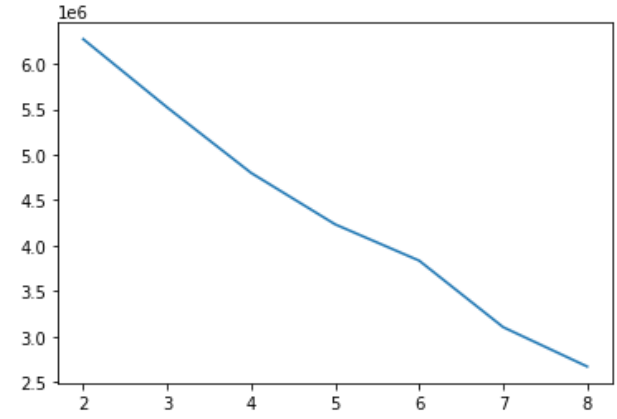


A correlation plot of our most important features against the TOTAL_DAMAGE variable returns the following result.



2.4. K-Means clustering for feature engineering

The team observed that after plotting the Sum of squared distances from the cluster mean for multiple number of clusters, the model could not provide a suitable number of clusters to use in the final model. Also the silhouette_score method resulted in inconclusive results due to the long run time of the model



The kmeans.inertia_ for the number of clusters

Summary and Conclusions

As seen from the results section, we have tried to build a disaster damage predictor by analyzing the attributes of the event and modeling them using regression. We have identified the variables that are having the most effect on the target variable. We find that wind speed type Extreme gust is a very important feature while predicting damage. Similarly, we see that the states of Alabama and Virginia are good indicators of damage prediction. We have used a variety of models with the Random Forest Regressor giving the best results. It gives a testing R-squared value of 49.7% whereas XGBoost gives 44.4% and ensemble model gives 44.3%. Linear regression does very poorly and gives a testing R-squared value of 33.5%. We have also used Grid Search for hyperparameter tuning to obtain the best scores. We also see that the environmental indicators like CO2 levels, Arctic Ice coverage, Earth Surface Temperature, etc. do not have any significant impact on our final predictions. This could be attributed to a mismatch in the granularity of the environmental data and the total damage that is caused by these disaster events. Future work will focus on further refining our feature engineering process to improve the performance of our model.

References

1. https://www.carbonbrief.org/mapped-how-climate-change-affects-extreme-weather-around-the-world
2. https://towardsdatascience.com/a-quick-and-dirty-guide-to-random-forest-regression-52ca0af157f8
3. https://www.oreilly.com/library/view/tensorflow-machine-learning/9781789132212/d3d388ea-3e0b-4095-b01e-a0fe8cb3e575.xhtml
4. http://en.wikipedia.org/wiki/Storm
5. sklearn.linear_model.LinearRegression — scikit-learn 1.0.1 documentation
6. https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097
7. https://machinelearningmastery.com/principal-component-analysis-for-visualization/
8. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
9. https://scikit-learn.org/stable/auto_examples/ensemble/plot_voting_regressor.html