PROPERTY DAMAGE PREDICTION

Individual Final Report

Author: Hemangi Kinger

Email: hkinger1102@gwu.edu

Overview of Project

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.

What is a Storm?

According to Wikipedia, a storm is any disturbed state of an environment or in an astronomical body's atmosphere especially affecting its surface, and strongly implying severe weather. It may be marked by significant disruptions to normal conditions such as strong wind, tornadoes, hail, thunder, and lightning (a thunderstorm), heavy precipitation (snowstorm, rainstorm), heavy freezing rain (ice storm), strong winds (tropical cyclone, windstorm), or wind transporting some substance through the atmosphere as in a dust storm, blizzard, sandstorm, etc.

What are we predicting?

NOAA stores the observations of storm events in a database of csv files (https://www.ncei.noaa.gov/pub/data/swdi/stormevents/csvfiles/). We are using the features and observations from this data to predict the property damage caused by any of the storm events in United States

Roles and Responsibility

Team Member	Area of Work	Shared Responsibility
Siddharth Das	Preprocessing	EDA
Kartik Das	Extraction	Modelling
Hemangi Kinger	PyQt5 and Visualization	EDA
Ishan Kuchroo	Modelling	Preprocessing

What is my responsibility?

Once my teammates are done with extraction and data pre-processing, I'll be using their learning of the data to do exploratory data analysis. Post EDA, I'll share the insights with the team to build better regression models. Additionally, I'll also work on the GUI (i.e. PyQT5) and visualization.

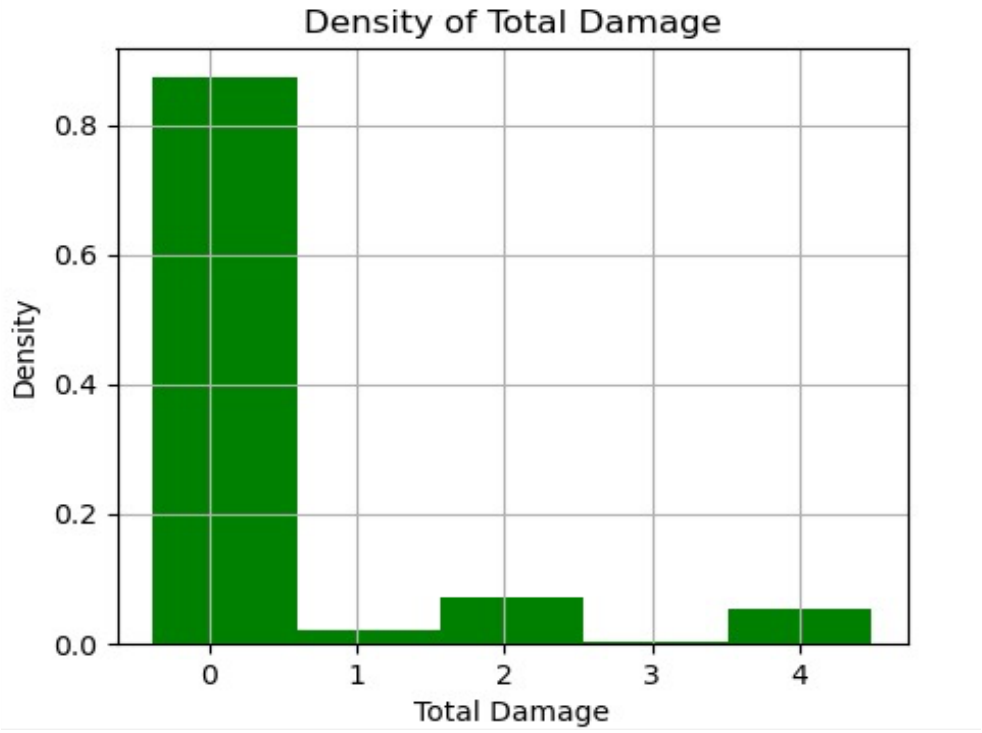
- I'll be proof-reading and making changes in the summary report created by team
- Consolidating the code of data-preprocessing and modelling and creating a pipeline to ensure the code runs smoothly for PyQt5.

PyQt5, Visualization and EDA

After Data Cleaning, I started to do following step:

Exploratory Data Analysis:

1. Initially to check the trend of our target variable i.e., TOTAL_DANAMGE

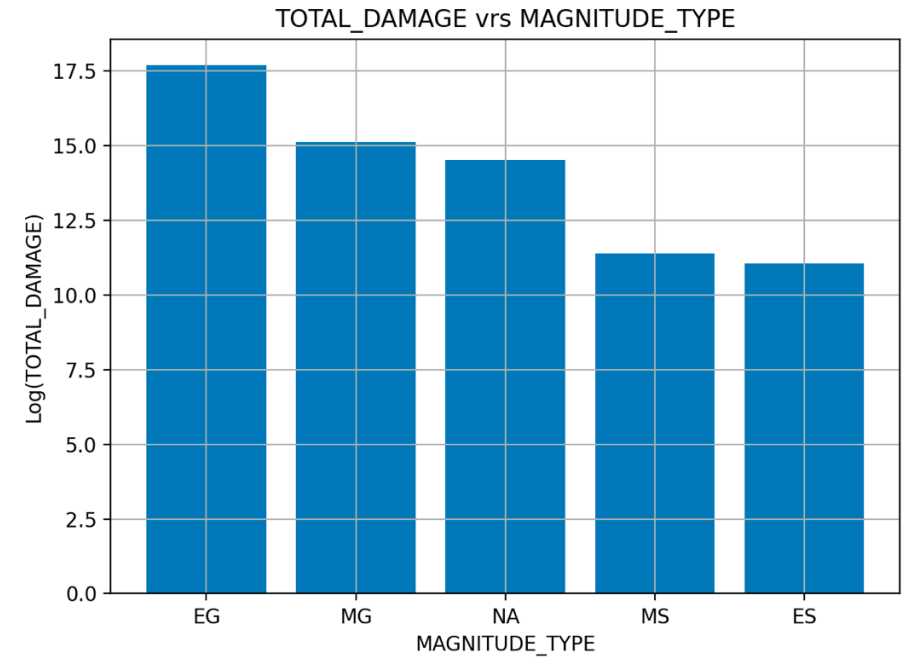


   For the trend of target variable, I normalized the data (subtracted from mean and divided by standard deviation) our target variable as majority of the values is 0.

2. Trend of target variable with respect to our target variable.

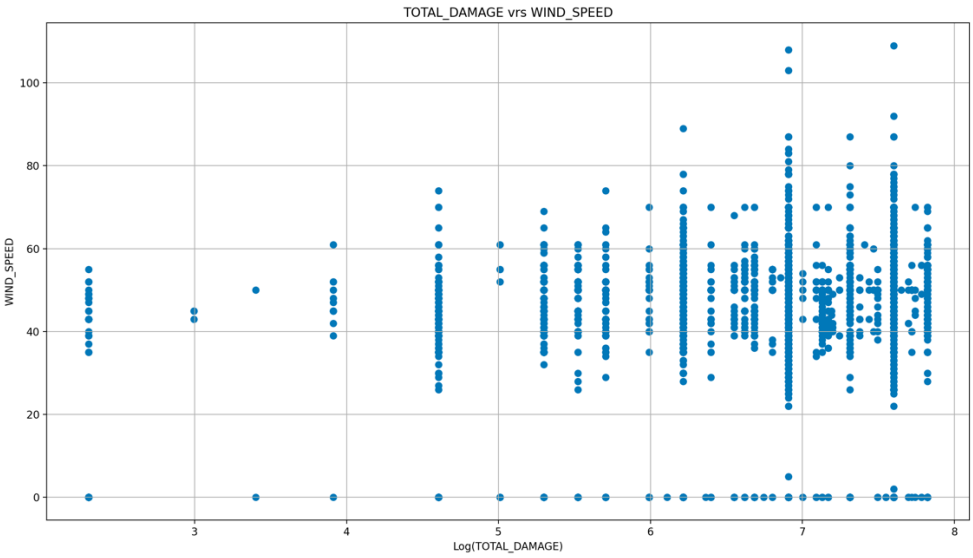
   To plot the trend with respect to our target variable we took log for our target variable (TOTAL_DAMAGE) for normalization.

I. TOTAL_DAMAGE v/s MAGNITUDE_TYPE



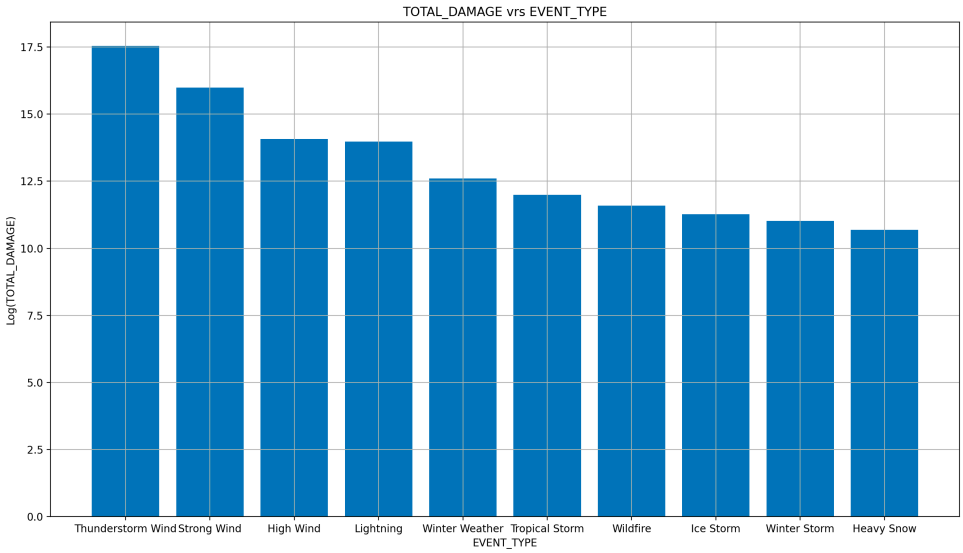
Our target variable shows that Wind estimated gust (EG) has the highest total damage and Estimated Sustained Wind (ES) has the least.

II. TOTAL_DAMAGE v/s WIND_SPEED



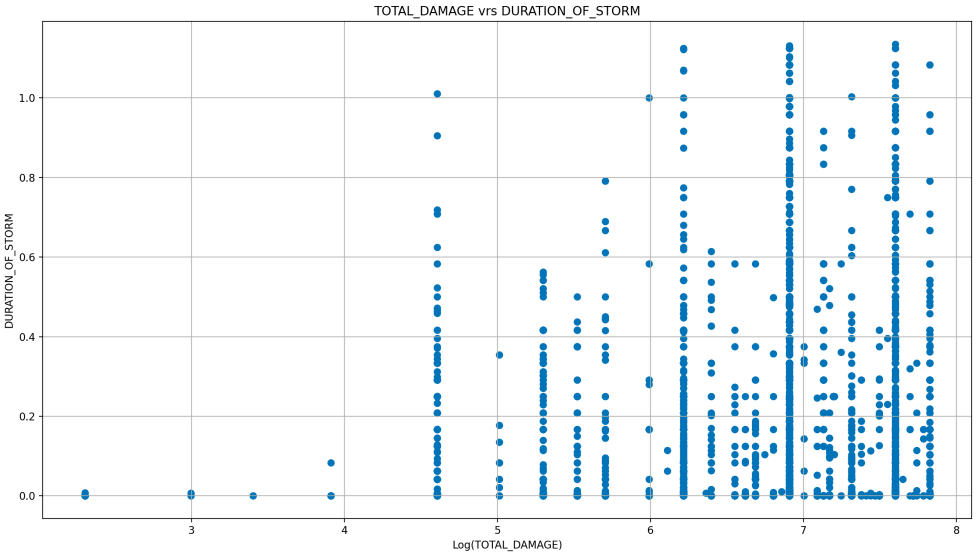
There's a slight positive correlation between wind speed and total damage. Maximum damage is caused by wind speed ranging between 20 knots to 80 knots.

III. TOTAL_DAMAGE v/s EVENT_TYPE (TOP 10)



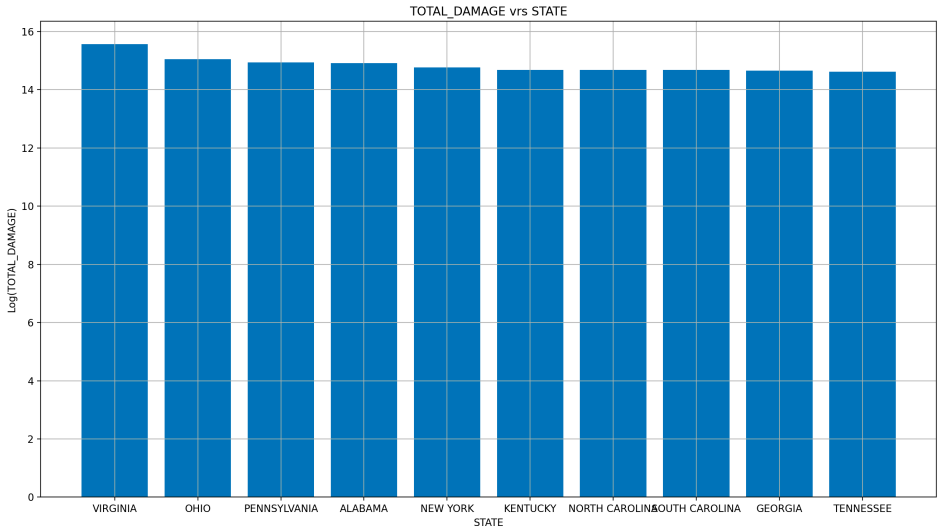
The graph depicts the top 10 events that have the highest total damage. Wind events seem to have high damage followed by the winter related events.

IV. TOTAL_DAMAGE v/s DURATION_OF_STORM



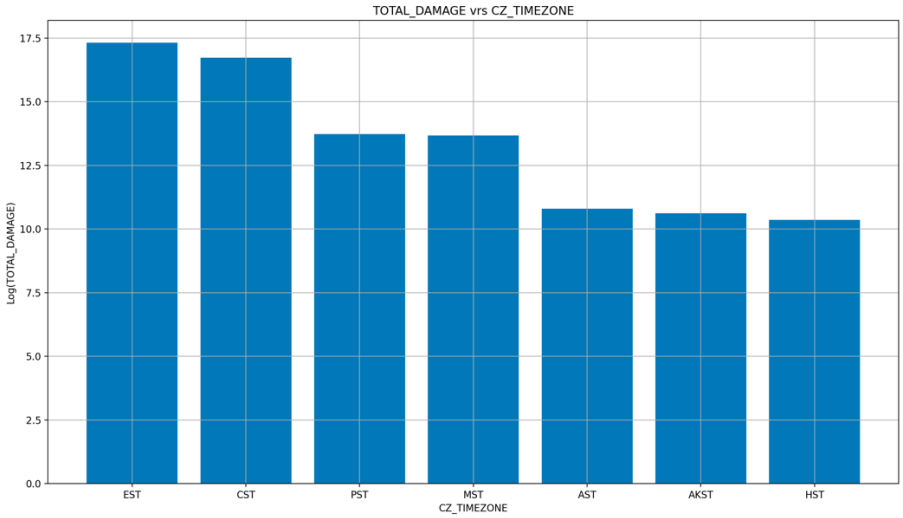
There's a positive correlation between duration of storm and total damage.

V. TOTAL_DAMAGE v/s STATE (TOP 10)



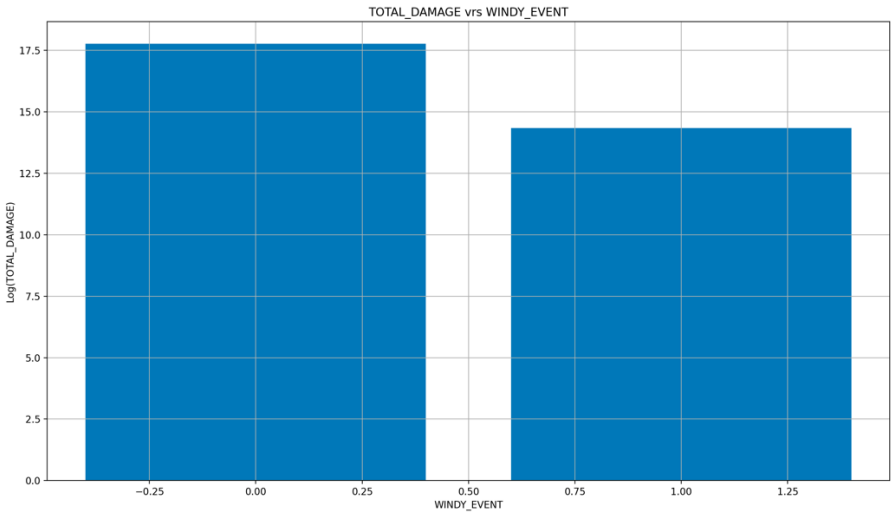
The graph depicts the top 10 states that have the most total damage Virginia being the highest.

VI. TOTAL_DAMAGE v/s CZ_TIMEZONE



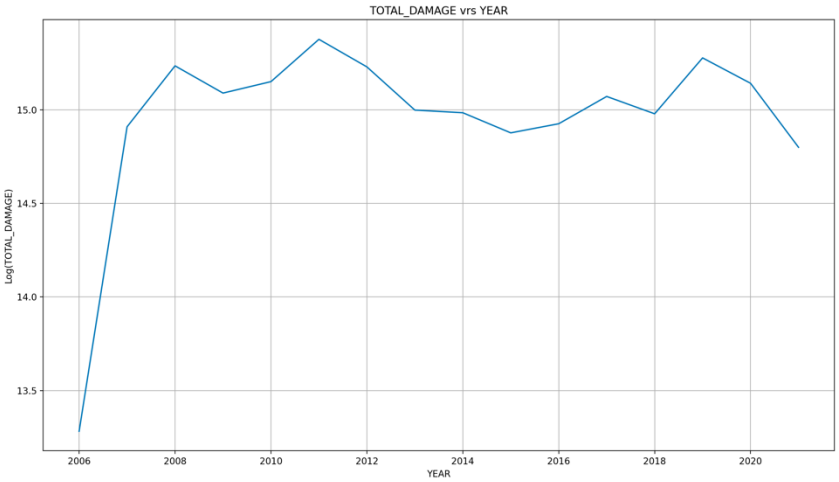
The time zone classification shows the region where the total damage was the highest. Eastern and Central region being the highest.

VII. TOTAL_DAMAGE v/s CZ_WINDY_EVENT



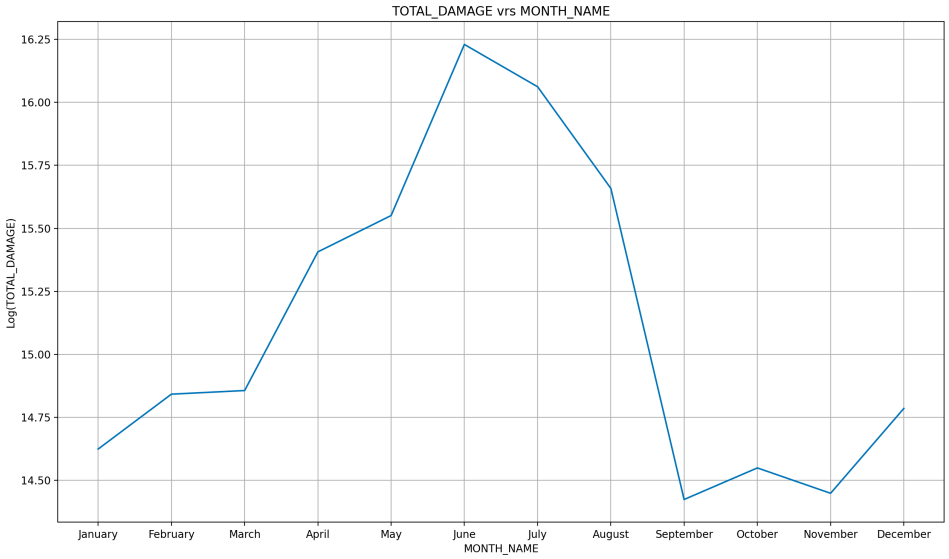
This graph depicts the total damage occurred due to a windy event when compared to a hail event.

VIII. TOTAL_DAMAGE v/s YEAR



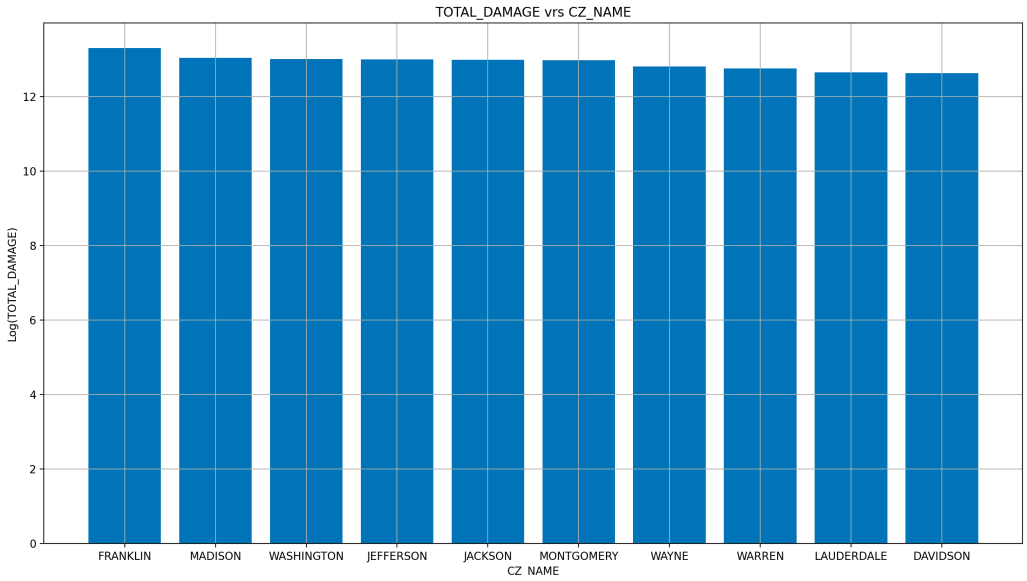
This graph shows the trend of total damage with respect to year. Here the graph is from 2006 (as before that NOAA didn't capture all the event types) which shows the increase in total damage with 2011 showing the highest total damage.

IX. TOTAL_DAMAGE v/s MONTH_NAME



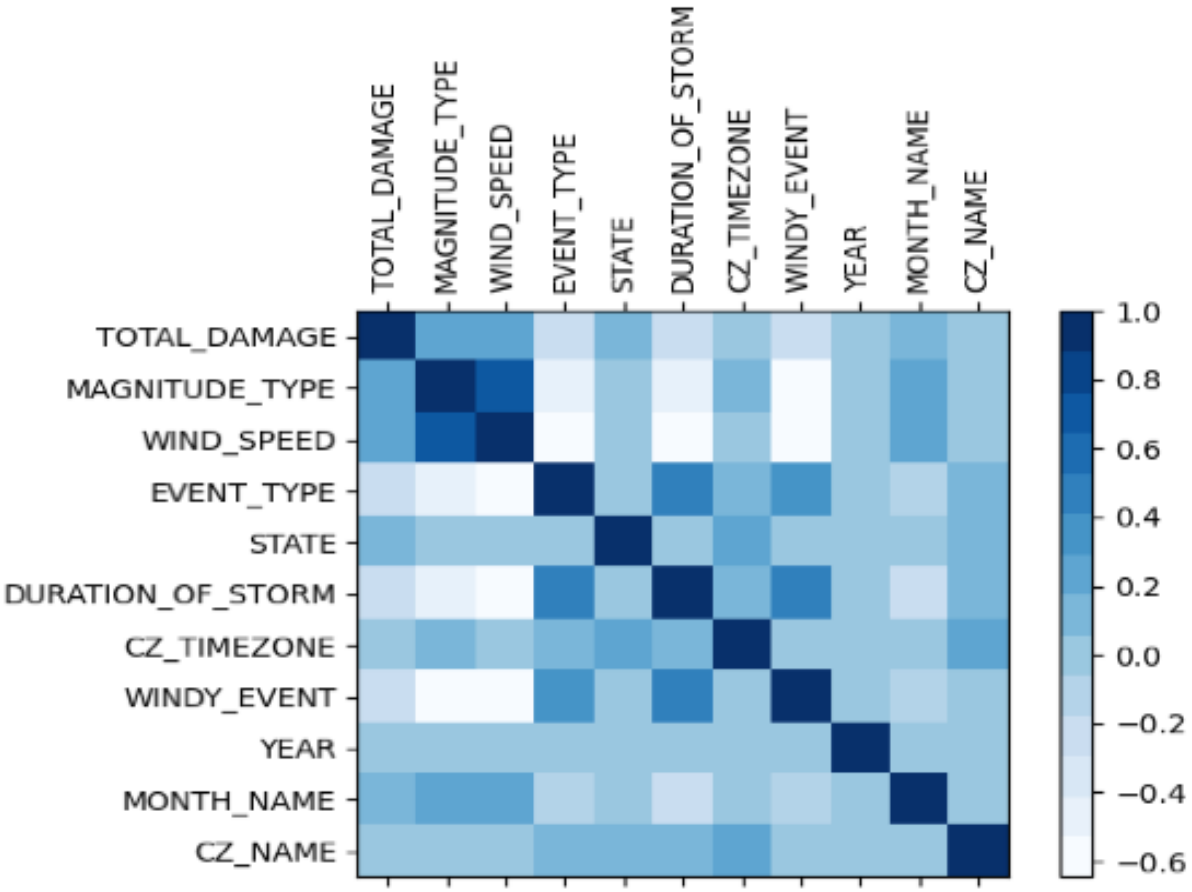
The months from May to August show the highest total damage as these months are the months for tornado season.

X. TOTAL_DAMAGE v/s CZ_NAME (TOP 10)



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

3. Feature Correlation Matrix



The correlation matrix shows the correlation between the features. From the correlation matrix we can see that MAGNITUDE_TYPE and WIND_SPEED have the highest correlation with TOTAL_DAMAGE.

PyQt5 and Visualization:

1. Created initUi function in Random Forest class to create the UI to show the output of Random Forest algorithm.

```
def initUi(self):
    #::----------------------------------------------------------------
    #  Create the canvas and all the element to create a dashboard with
    #  all the necessary elements to present the results from the algorithm
    #  The canvas is divided using a  grid layout to facilitate the drawing
    #  of the elements
    #::----------------------------------------------------------------

    self.setWindowTitle(self.Title)
    self.setStyleSheet(font_size_window)

    self.main_widget = QWidget(self)

    self.layout = QGridLayout(self.main_widget)

    self.groupBox1 = QGroupBox('Random Forest Features')
    self.groupBox1Layout= QGridLayout()   # Grid
    self.groupBox1.setLayout(self.groupBox1Layout)

    self.btnExecute = QPushButton("Execute RF")
    self.btnExecute.clicked.connect(self.update)

    self.groupBox1Layout.addWidget(self.btnExecute,5,0)

    self.groupBox2 = QGroupBox('Results from the model')
    self.groupBox2Layout = QVBoxLayout()
    self.groupBox2.setLayout(self.groupBox2Layout)


    self.lblMSE = QLabel('Mean Square Value:')
    self.txtMSE = QLineEdit()
```

2. Created update function in Random Forest class to populate the output of Random Forest algorithm.

```
def update(self):
    '''
    Random Forest
    We populate the dashboard using the parametres chosen by the user
    The parameters are processed to execute in the skit-learn Random Forest algorithm
      then the results are presented in graphics and reports in the canvas
    :return:None
    '''


    # Assign the X and y to run the Random Forest

    X_dt = df_train.loc[:, ~df_train.columns.isin(['TOTAL_DAMAGE', 'YEAR'])]
    y_dt = df_train['TOTAL_DAMAGE']

    class_le = LabelEncoder()

    # split the dataset into train and test

    X_train, X_test, y_train, y_test = train_test_split(X_dt, y_dt, test_size=0.3, random_state=100)


    #------------------------------------------------------------------------

    # predicton on test using all features
    y_pred = loaded_model_rf.predict(X_test)


    self.ff_mse = mean_squared_error(y_pred,y_test)
    self.txtMSE.setText(str(self.ff_mse))
```

3. Created __init__ function in Random Forest class to initiate the class.

```
def __init__(self):
    super(RandomForest, self).__init__()
    self.Title = "Random Forest "
    self.initUi()
```

4. Created __init__ function in XGBoost class to initiate the class.

```
def __init__(self):
    super(XGBoost, self).__init__()

    self.Title ="XGBoost"
    self.initUi()
```

5. Created initUi function in XGBoost class to create the UI to show the output of XGBoost algorithm.

```
def initUi(self):
    #:----------------------------------------------------------------------
    #  Create the canvas and all the element to create a dashboard with
    #  all the necessary elements to present the results from the algorithm
    #  The canvas is divided using a  grid loyout to facilitate the drawing
    #  of the elements
    #:----------------------------------------------------------------------

    self.setWindowTitle(self.Title)
    self.setStyleSheet(font_size_window)

    self.main_widget = QWidget(self)

    self.layout = QGridLayout(self.main_widget)

    self.groupBox1 = QGroupBox('XGBoost')
    self.groupBox1Layout= QGridLayout()
    self.groupBox1.setLayout(self.groupBox1Layout)

    self.btnExecute = QPushButton("Execute XGBoost")
    self.btnExecute.clicked.connect(self.update)


    # We create a checkbox for each feature

    self.groupBox1Layout.addWidget(self.btnExecute,6,0)

    self.groupBox2 = QGroupBox('Results from the model')
    self.groupBox2Layout = QVBoxLayout()
    self.groupBox2.setLayout(self.groupBox2Layout)

    self.lblMSE = QLabel('Mean Square Value:')
    self.txtMSE = QLineEdit()
```

6. Created update function in XGBoost class to populate the output of XGBoost algorithm.

```
def update(self):
    '''
    Decision Tree Algorithm
    We pupulate the dashboard using the parametres chosen by the user
    The parameters are processed to execute in the skit-learn Decision Tree algorithm
      then the results are presented in graphics and reports in the canvas
    :return: None
    '''

    # We process the parameters

    # Assign the X and y to run the Random Forest

    X_dt = df_train.loc[:, ~df_train.columns.isin(['TOTAL_DAMAGE', 'YEAR'])]
    y_dt = df_train['TOTAL_DAMAGE']

    class_le = LabelEncoder()

    # split the dataset into train and test

    X_train, X_test, y_train, y_test = train_test_split(X_dt, y_dt, test_size=0.3, random_state=100)

    y_pred = loaded_model_xgb.predict(X_test)

    self.ff_mse = mean_squared_error(y_pred, y_test)
    self.txtMSE.setText(str(self.ff_mse))
```

7. Created __init__ function in CorrelationPlot class to initiate the class.

```
def __init__(self):
    #::-------------------------------------------------------------
    # Initialize the values of the class
    #::-------------------------------------------------------------
    super(CorrelationPlot, self).__init__()

    self.Title = 'Correlation Plot'
    self.initUi()
```

8. Created initUi function in CorrelationPlot class to create the UI to show the output of Correlation Plot.

```
def initUi(self):
    #;;-------------------------------------------------------------------
    #  Creates the canvas and elements of the canvas
    #;;-------------------------------------------------------------------
    self.setWindowTitle(self.Title)
    self.setStyleSheet(font_size_window)

    self.main_widget = QWidget(self)

    self.layout = QVBoxLayout(self.main_widget)

    self.groupBox1 = QGroupBox('Correlation Plot Features')
    self.groupBox1Layout= QGridLayout()
    self.groupBox1.setLayout(self.groupBox1Layout)


    self.feature0 = QCheckBox(features_list[0],self)
    self.feature1 = QCheckBox(features_list[1],self)
    self.feature2 = QCheckBox(features_list[2], self)
    self.feature3 = QCheckBox(features_list[2], self)
    self.feature4 = QCheckBox(features_list[4],self)
    self.feature5 = QCheckBox(features_list[5],self)
    self.feature6 = QCheckBox(features_list[6], self)
    self.feature7 = QCheckBox(features_list[7], self)
```

9. Created update function in CorrelationPlot class to populate the output of Correlation Plot.

```
def update(self):

    #;;-------------------------------------------------------------
    # Populates the elements in the canvas using the values
    # chosen as parameters for the correlation plot
    #;;-------------------------------------------------------------
    self.ax1.clear()

    ff_noaa_cor = ff_noaa[features_list]

    def mapping(xx):

        dict = {}
        count = -1
        for x in xx:
            dict[x] = count + 1
            count = count + 1
        return dict

    for i in ["MAGNITUDE_TYPE", "EVENT_TYPE", "STATE","CZ_TIMEZONE","WINDY_EVENT","YEAR","MONTH_NAME",'CZ_NAME']
        unique_tag = ff_noaa_cor[i].value_counts().keys().values
        dict_mapping = mapping(unique_tag)
        ff_noaa_cor[i] = ff_noaa_cor[i].map(lambda x: dict_mapping[x] if x in dict_mapping.keys() else -1)

    X_1 = ff_noaa["TOTAL_DAMAGE"]

    list_corr_features = pd.DataFrame(ff_noaa["TOTAL_DAMAGE"])
```

10. Created __init__ function in DPGraphs class to initiate the class.

```
def __init__(self):
    #::-------------------------------------------------------------
    # Crate a canvas with the layout to draw a dotplot
    # The layout sets all the elements and manage the changes
    # made on the canvas
    #::-------------------------------------------------------------
    super(DPGraphs, self).__init__()

    self.Title = "Features vrs TOTAL_DAMAGE"
    self.main_widget = QWidget(self)

    self.setWindowTitle(self.Title)
    self.setStyleSheet(font_size_window)

    self.fig = Figure()
    self.ax1 = self.fig.add_subplot(111)
    self.axes=[self.ax1]
    self.canvas = FigureCanvas(self.fig)


    self.canvas.setSizePolicy(QSizePolicy.Expanding,
                              QSizePolicy.Expanding)
```

11. Created update function in DPGraphs class to show the trends of features with respect to target variable.

```
def update(self):
    #::-------------------------------------------------------------
    # This method executes each time a change is made on the canvas
    # containing the elements of the graph
    # The purpose of the method es to draw a dot graph using the
    # score of happiness and the feature chosen the canvas
    #::-------------------------------------------------------------
    colors=["b", "r", "g", "y", "k", "c"]
    self.ax1.clear()
    cat1 = self.dropdown1.currentText()

    numerical_features = ["DURATION_OF_STORM","WIND_SPEED"]

    if cat1 in numerical_features:

        X_1 = ff_noaa["TOTAL_DAMAGE"]
        y_1 = ff_noaa[cat1]

        X_1 = np.log(X_1)

        self.ax1.scatter(X_1,y_1)

        vtitle = "TOTAL_DAMAGE vrs "+ cat1
        self.ax1.set_title(vtitle)
        self.ax1.set_xlabel("Log(TOTAL_DAMAGE)")
        self.ax1.set_ylabel(cat1)
```

12. Created initUi function in App class to create the UI for the menu.

```
def initUI(self):
    #::-----------------------------------------------
    # Creates the manu and the items
    #::-----------------------------------------------
    self.setWindowTitle(self.Title)
    self.setGeometry(self.left, self.top, self.width, self.height)

    #::-----------------------------
    # Create the menu bar
    # and three items for the menu, File, EDA Analysis and ML Models
    #::-----------------------------
    mainMenu = self.menuBar()
    mainMenu.setStyleSheet('background-color: lightblue')

    fileMenu = mainMenu.addMenu('File')
    EDAMenu = mainMenu.addMenu('EDA Analysis')
    MLModelMenu = mainMenu.addMenu('ML Models')

    #::-------------------------------------
    # Exit application
    # Creates the actions for the fileMenu item
    #::-------------------------------------

    exitButton = QAction(QIcon('enter.png'), 'Exit', self)
    exitButton.setShortcut('Ctrl+Q')
    exitButton.setStatusTip('Exit application')
```

13. Created a function EDA1 that creates histogram of the target variable

```
def EDA1(self):
    #::-------------------------------------------------------
    # Creates the histogram
    # X was populated in the method data_noaa()
    # at the start of the application
    #::-------------------------------------------------------
    dialog = CanvasWindow(self)
    dialog.m.plot()
    x = (X - np.mean(X))/np.std(X)
    dialog.m.ax.hist(x, bins = 5, density=True, facecolor='green')
    dialog.m.ax.set_title('Density of Total Damage')
    dialog.m.ax.set_xlabel("Total Damage")
    dialog.m.ax.set_ylabel("Density")
    dialog.m.ax.grid(True)
    dialog.m.draw()
    self.dialogs.append(dialog)
    dialog.show()
```

14. Created the function data_noaa to load the pickle file and define global variables.

```
def data_noaa():
    global noaa
    global ff_noaa
    global X
    global y
    global features_list
    global class_names
    global df_train
    global loaded_model_rf
    global loaded_model_xgb
    ff_noaa = pd.read_pickle('Data/cleaned_NAN_removed.pkl')
    X = ff_noaa["TOTAL_DAMAGE"]
    y = ff_noaa["STATE"]
    df_train = pd.read_pickle('Data/df_train.pkl')
    loaded_model_rf = pickle.load(open('Data/RF_Model.pkl', 'rb'))
    loaded_model_xgb = pickle.load(open('Data/XGB_Model.pkl', 'rb'))
    features_list = ["MAGNITUDE_TYPE", "WIND_SPEED", "EVENT_TYPE", "STATE",
        "DURATION_OF_STORM", "CZ_TIMEZONE","WINDY_EVENT","YEAR","MONTH_NAME",'CZ_NAME']
```

Conclusion

Referenced Code %:

(675 – 300)/ 675 + 94 * 100 = 48%