Storm Predictor

October 27, 2024

1 Storm Damage and Magnitude Predictor

Owen Kroeger

Sprint 4 - STG-451

Video Link: This notebook documents the implementation of a machine learning model designed to predict storm damage and magnitude based on historical storm event data. The model utilizes TensorFlow and scikit-learn libraries for data preprocessing, model building, and evaluation.

2 Overview of Storm Damage and Magnitude Prediction Notebook

This Jupyter notebook is designed to implement a machine learning model that predicts storm-related damages, specifically property damage and crop damage, as well as the magnitude of the storm events. The analysis is based on historical storm event data obtained from the National Oceanic and Atmospheric Administration (NOAA).

2.1 Objectives:

- **Predict Storm Damage**: Estimate property and crop damages based on storm characteristics and historical data.
- **Predict Magnitude**: Assess the intensity of storms through a magnitude metric.

2.2 Methodology:

1. **Data Loading**:
   - The dataset is loaded from a CSV file that contains records of storm events, including their characteristics, damage reports, and time stamps.
2. **Data Inspection**:
   - The file inspects the data for missing values and provides a summary of these missing values. This step is crucial for understanding the quality of the data and for subsequent preprocessing.
3. **Data Preprocessing**:
   - **Damage Parsing**: A helper function converts damage values expressed in various formats (e.g., '0.00K', '1.20M') into numeric values (e.g., 0.00, 1200000).
   - **Feature Selection**: Relevant features are selected from the dataset for model training, excluding the target variables to avoid data leakage.

- **Handling Missing Values**: Missing values are imputed using appropriate strategies: categorical variables are filled with 'Unknown', while numerical variables are filled with their median values.
- **Feature Scaling**: Numerical features are standardized using `StandardScaler`, while categorical features are transformed using one-hot encoding to convert them into a suitable format for the model.

4. **Model Building**:
   - A neural network model is constructed using TensorFlow/Keras. The model consists of multiple layers with dropout regularization to prevent overfitting. The architecture is designed to handle multiple outputs corresponding to the target variables.
5. **Model Training**:
   - The data is split into training, validation, and testing sets to evaluate the model's performance accurately. The model is trained using the training data, with early stopping implemented to halt training when the model performance on the validation set ceases to improve.
6. **Model Evaluation**:
   - The trained model is evaluated using the test set to calculate metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE).
   - Additionally, R^2 scores are computed for each target variable to quantify the proportion of variance explained by the model.
7. **Results Visualization**:
   - Various plots are generated to visualize the training history, including loss and MAE over epochs. Furthermore, scatter plots comparing actual versus predicted values for each target variable are provided for qualitative assessment.
8. **Model Persistence**:
   - The trained model and the preprocessor are saved to disk for future use. This allows for easy deployment and inference on new data.

2.3 Conclusion:

This notebook serves as a comprehensive guide to building a predictive model for storm damages. The structured approach encompasses data loading, preprocessing, model training, and evaluation, thereby providing a clear workflow for predictive modeling in the context of natural disasters.

2.4 Import Required Libraries

In this cell, we import all necessary libraries and modules for data handling, preprocessing, model building, and evaluation.

[1]:
```
# Import required libraries
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import joblib

# Suppress TensorFlow warnings
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
```

```
2024-10-27 22:34:18.159296: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

2.5 Configuration and Constants

In this cell, we define the path to the CSV file containing storm event data and specify the target variables we want to predict.

[2]:
```
# Configuration and Constants

# Path to your extracted CSV file
CSV_FILE_PATH = 'StormEvents_details-ftp_v1.0_d2024_c20241017.csv'  # <--␣
 ↪Replace with your actual path

# Target Variables
TARGETS = ['DAMAGE_PROPERTY', 'DAMAGE_CROPS', 'MAGNITUDE']  # List of targets␣
 ↪to predict
```

2.6 Function Definitions

Here we define several functions that will be used throughout the notebook for: - Parsing damage values - Loading data from the CSV file - Inspecting missing values - Ensuring categorical data types - Preprocessing the data for model training

[3]:
```
# Function Definitions

def parse_damage(value):
    """
    Convert damage strings like '0.00K', '1.20M' to numeric values.
    """
    if pd.isnull(value) or value == "":
        return np.nan  # Handle empty strings and NaN

    try:
        # Check the type of the value
```

```
        if isinstance(value, (float, int)):
            return float(value)  # Directly return numeric values

        if isinstance(value, str):
            value = value.strip()  # Clean the value by stripping whitespace

            # Handle different suffixes
            if value.endswith('K'):
                return float(value[:-1]) * 1_000
            elif value.endswith('M'):
                return float(value[:-1]) * 1_000_000
            elif value.endswith('B'):
                return float(value[:-1]) * 1_000_000_000

            # Attempt to convert any remaining direct numeric values
            return float(value)
        else:
            print(f"Unexpected type encountered: {type(value)}. Value: {value}")
            return np.nan
    except Exception as e:
        print(f"Could not convert value: {value}. Error: {e}")  # Log the␣
↪problematic value and error
        return np.nan  # If conversion fails, return NaN


def load_data(csv_path):
    """
    Load the CSV data into a pandas DataFrame.
    """
    if not os.path.exists(csv_path):
        raise FileNotFoundError(f"CSV file not found at '{csv_path}'. Please␣
↪check the path.")

    print("Loading data...")
    try:
        df = pd.read_csv(
            csv_path,
            parse_dates=['BEGIN_DATE_TIME', 'END_DATE_TIME'],
            date_parser=lambda x: pd.to_datetime(x, format='%Y-%m-%d %H:%M:%S',␣
↪errors='coerce'),
            low_memory=False
        )
    except ValueError as e:
        print(f"Error parsing dates: {e}")
        df = pd.read_csv(csv_path, low_memory=False)
        df['BEGIN_DATE_TIME'] = pd.to_datetime(df['BEGIN_DATE_TIME'],␣
↪errors='coerce')
```

```
        df['END_DATE_TIME'] = pd.to_datetime(df['END_DATE_TIME'],␣
↪errors='coerce')

    print(f"Data loaded successfully. Number of records: {len(df)}\n")
    return df


def inspect_missing_values(df):
    """
    Inspect missing values in the DataFrame.
    """
    print("Missing Values per Column:")
    missing_counts = df.isnull().sum()
    missing_percent = (missing_counts / len(df)) * 100
    missing_df = pd.DataFrame({
        'Missing Count': missing_counts,
        'Missing Percentage': missing_percent
    })
    print(missing_df[missing_df['Missing Count'] > 0].sort_values(by='Missing␣
↪Percentage', ascending=False))
    print("\n")


def ensure_categorical_string(X, categorical_features):
    """
    Ensure that all categorical features are of string type.
    """
    for col in categorical_features:
        if col in X.columns:
            X[col] = X[col].astype(str)
    return X


def check_numerical_columns(X, numerical_features):
    """
    Ensure all numerical columns contain numeric data.
    """
    for col in numerical_features:
        if not pd.api.types.is_numeric_dtype(X[col]):
            print(f"Column '{col}' is not numeric. Attempting to convert...")
            X[col] = pd.to_numeric(X[col], errors='coerce')
            non_converted = X[col].isnull().sum()
            if non_converted > 0:
                print(f"{non_converted} entries in '{col}' could not be␣
↪converted and are set to NaN.")
```

```
def preprocess_data(df):
    """
    Preprocess the DataFrame for modeling.
    """
    print("Preprocessing data...")

    # 1. Feature Engineering: Calculate storm duration in hours
    df['duration_hours'] = (df['END_DATE_TIME'] - df['BEGIN_DATE_TIME']).dt.
 ↪total_seconds() / 3600

    # 2. Select relevant columns
    selected_columns = [
        'EVENT_TYPE', 'CZ_TYPE', 'STATE', 'YEAR', 'MONTH_NAME', 'MAGNITUDE',␣
 ↪'MAGNITUDE_TYPE',
        'CATEGORY', 'TOR_F_SCALE', 'TOR_LENGTH', 'TOR_WIDTH',
        'INJURIES_DIRECT', 'INJURIES_INDIRECT', 'DEATHS_DIRECT',␣
 ↪'DEATHS_INDIRECT',
        'END_LAT', 'END_LON', 'BEGIN_RANGE', 'BEGIN_AZIMUTH', 'BEGIN_LOCATION',
        'END_RANGE', 'END_AZIMUTH', 'END_LOCATION', 'BEGIN_LAT', 'BEGIN_LON'
    ]

    existing_columns = [col for col in selected_columns if col in df.columns]
    X = df.loc[:, existing_columns].copy()

    # 3. Extract and process target variables
    y = df[TARGETS].copy()

    print("Data types before conversion:")
    print(y.dtypes)

    for col in TARGETS:
        if col in y.columns:
            print(f"Unique values in {col} before conversion: {y[col].
 ↪unique()}")
            y[col] = y[col].apply(parse_damage)
            non_converted = y[col].isnull().sum()
            if non_converted > 0:
                print(f"{non_converted} missing or invalid entries found in␣
 ↪{col} after conversion.")
                print(f"Remaining values in {col}: {y[col].unique()}")

    # 4. Handle Missing Values in Features
    categorical_features = ['EVENT_TYPE', 'CZ_TYPE', 'STATE', 'MONTH_NAME',␣
 ↪'MAGNITUDE_TYPE', 'CATEGORY', 'TOR_F_SCALE', 'TOR_WIDTH']
    numerical_features = [col for col in X.columns if col not in␣
 ↪categorical_features]
```

```
    check_numerical_columns(X, numerical_features)
    X = ensure_categorical_string(X, categorical_features)

    original_num_features = numerical_features.copy()
    numerical_features = [col for col in numerical_features if not X[col].
↪isnull().all()]
    dropped_features = set(original_num_features) - set(numerical_features)
    if dropped_features:
        print(f"Removing numerical features with all NaN values: {', '.
↪join(dropped_features)}")
        X = X.drop(columns=list(dropped_features))

    numerical_imputer = SimpleImputer(strategy='median')
    if numerical_features:
        X.loc[:, numerical_features] = numerical_imputer.
↪fit_transform(X[numerical_features])
    else:
        print("No numerical features to impute.")

    categorical_imputer = SimpleImputer(strategy='constant',␣
↪fill_value='Unknown')
    if categorical_features:
        X.loc[:, categorical_features] = categorical_imputer.
↪fit_transform(X[categorical_features])
    else:
        print("No categorical features to impute.")

    print("Missing values imputed.")

    # 5. Define Preprocessing for Numerical and Categorical Data
    numerical_transformer = StandardScaler()
    categorical_transformer = OneHotEncoder(handle_unknown='ignore')

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_transformer, numerical_features),
            ('cat', categorical_transformer, categorical_features)
        ])

    # 6. Apply Preprocessing
    X_processed = preprocessor.fit_transform(X)

    if isinstance(X_processed, (tf.SparseTensor, tf.Tensor)) or␣
↪hasattr(X_processed, 'toarray'):
        X_processed = X_processed.toarray()
```

```
    else:
        print("X_processed is already a dense array.")

    print("Preprocessing complete.\n")

    # 7. Handle Missing Values in Targets
    initial_length = X_processed.shape[0]
    valid_indices = y.notnull().all(axis=1).values
    X_processed = X_processed[valid_indices]
    y = y[valid_indices]
    final_length = X_processed.shape[0]
    dropped_rows = initial_length - final_length
    if dropped_rows > 0:
        print(f"Dropped {dropped_rows} records due to NaN values in target␣
↪variables.")

    # 8. Validate Data
    assert not np.isnan(X_processed).any(), "There are still NaNs in the␣
↪features."
    assert not y.isnull().any().any(), "There are still NaNs in the target␣
↪variables."

    return X_processed, y, preprocessor
```

2.7 Model Building Function

In this cell, we define a function to build a simple neural network model using Keras. This model will be used for predicting the target variables.

[4]:
```
def build_model(input_dim):
    """
    Build a simple neural network model with Dropout layers for regularization.
    """
    print("Building the neural network model...")
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(32, activation='relu'),
        layers.Dense(len(TARGETS))  # Output layer for multiple regression
    ])

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
    print("Model built successfully.\n")
    return model
```

2.8 Plot Training History Function

This function visualizes the training and validation loss and mean absolute error (MAE) over the epochs, allowing us to track the model's performance during training.

[5]:
```
def plot_history(history):
    """
    Plot the training and validation loss and MAE.
    """
    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Mean Squared Error')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['mae'], label='Train MAE')
    plt.plot(history.history['val_mae'], label='Validation MAE')
    plt.title('Model Mean Absolute Error')
    plt.xlabel('Epoch')
    plt.ylabel('MAE')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

2.9 Main Function

In this cell, we define the `main()` function which executes the entire workflow from loading the data, preprocessing, training the model, evaluating its performance, and saving the model.

[6]:
```
def main():
    # Step 1: Load Data
    df = load_data(CSV_FILE_PATH)

    # Step 1a: Inspect Missing Values
    inspect_missing_values(df)

    # Step 2: Preprocess Data
    X, y, preprocessor = preprocess_data(df)

    # Step 3: Train-Test Split
    print("Splitting data into training and testing sets...")
    X_train, X_test, y_train, y_test = train_test_split(
```

```
        X, y, test_size=0.2, random_state=42
    )
    print(f"Training set size: {X_train.shape[0]} records")
    print(f"Testing set size: {X_test.shape[0]} records\n")

    # Step 4: Further Split Training into Training and Validation Sets
    print("Splitting training data into training and validation sets...")
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=0.2, random_state=42
    )
    print(f"New Training set size: {X_train.shape[0]} records")
    print(f"Validation set size: {X_val.shape[0]} records\n")

    # Step 5: Build the Model
    input_dim = X_train.shape[1]
    model = build_model(input_dim)

    # Step 6: Train the Model with Early Stopping
    print("Training the model...")
    early_stop = keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_data=(X_val, y_val),
        callbacks=[early_stop],
        verbose=1
    )
    print("Model training complete.\n")

    # Step 7: Plot Training History
    plot_history(history)

    # Step 8: Evaluate the Model
    print("Evaluating the model on the test set...")
    test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
    print(f"Test Mean Squared Error: {test_loss:.2f}")
    print(f"Test Mean Absolute Error (MAE): {test_mae:.2f}\n")

    # Step 9: Make Predictions and Calculate R² Score
    predictions = model.predict(X_test)
    r2_scores = [r2_score(y_test.iloc[:, i], predictions[:, i]) for i in␣
↪range(predictions.shape[1])]
```

```
    for i, target in enumerate(TARGETS):
        print(f"Test R² Score for {target}: {r2_scores[i]:.2f}")

    # Step 10: Print some predictions for each target variable
    print("\nSome predictions vs actual values:")
    for i, target in enumerate(TARGETS):
        print(f"\nTarget: {target}")
        for actual, predicted in zip(y_test[target].head(10), predictions[:10,␣
↪i]):
            print(f"Actual: {actual:.2f}, Predicted: {predicted:.2f}")

    # Optional: Summary statistics of predictions
    print("\nPrediction Summary Statistics:")
    for i, target in enumerate(TARGETS):
        print(f"\nTarget: {target}")
        print(f"Mean Prediction: {np.mean(predictions[:, i]):.2f}")
        print(f"Median Prediction: {np.median(predictions[:, i]):.2f}")
        print(f"Standard Deviation of Predictions: {np.std(predictions[:, i]):.
↪2f}")

    # Step 11: Visualization of Actual vs Predicted for each target
    for i, target in enumerate(TARGETS):
        plt.figure(figsize=(8, 6))
        plt.scatter(y_test[target], predictions[:, i], alpha=0.5)
        plt.xlabel(f'Actual {target}')
        plt.ylabel(f'Predicted {target}')
        plt.title(f'Actual vs. Predicted {target}')
        plt.plot([y_test[target].min(), y_test[target].max()],
                 [y_test[target].min(), y_test[target].max()], 'r--')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    # Step 12: Save the Model and Preprocessor
    model.save('storm_damage_magnitude_model.h5')
    joblib.dump(preprocessor, 'preprocessor.pkl')
    print("Model and preprocessor saved successfully.")

if __name__ == "__main__":
    main()
```

```
Loading data…

/tmp/ipykernel_347373/4172206719.py:45: FutureWarning: The argument
'date_parser' is deprecated and will be removed in a future version. Please use
'date_format' instead, or read your data in as 'object' dtype and then call
'to_datetime'.
```

```
  df = pd.read_csv(
Data loaded successfully. Number of records: 50623

Missing Values per Column:
                    Missing Count  Missing Percentage
BEGIN_DATE_TIME             50623          100.000000
END_DATE_TIME               50623          100.000000
CATEGORY                    50621           99.996049
TOR_OTHER_CZ_NAME           50333           99.427138
TOR_OTHER_CZ_STATE          50333           99.427138
TOR_OTHER_WFO               50333           99.427138
TOR_OTHER_CZ_FIPS           50333           99.427138
TOR_LENGTH                  48960           96.714932
TOR_WIDTH                   48960           96.714932
TOR_F_SCALE                 48960           96.714932
FLOOD_CAUSE                 45717           90.308753
MAGNITUDE_TYPE              30178           59.613219
MAGNITUDE                   22298           44.047172
END_AZIMUTH                 18077           35.709065
END_LON                     18077           35.709065
END_LAT                     18077           35.709065
BEGIN_LON                   18077           35.709065
BEGIN_LAT                   18077           35.709065
END_LOCATION                18077           35.709065
BEGIN_AZIMUTH               18077           35.709065
END_RANGE                   18077           35.709065
BEGIN_LOCATION              18077           35.709065
BEGIN_RANGE                 18077           35.709065
DAMAGE_PROPERTY             11058           21.843826
DAMAGE_CROPS                11046           21.820121
EVENT_NARRATIVE              8608           17.004129


Preprocessing data…
Data types before conversion:
DAMAGE_PROPERTY     object
DAMAGE_CROPS        object
MAGNITUDE          float64
dtype: object
Unique values in DAMAGE_PROPERTY before conversion: ['0.00K' nan '1.00K' '0.50K'
'2.00K' '25.00K' '5.00K' '4.00K' '10.00K'
 '20.00K' '15.00K' '50.00K' '7.50K' '1.50K' '1.10K' '75.00K' '0.25K'
 '100.00K' '3.50K' '0.30K' '0.20K' '0.01K' '125.00K' '30.00K' '3.00K'
 '300.00K' '150.00K' '35.00K' '250.00K' '11.00K' '6.00K' '7.00K' '45.00K'
 '5.50K' '200.00K' '51.17K' '36.07K' '428.00K' '26.00K' '0.80K' '0.10K'
 '600.00K' '80.00K' '8.00K' '500.00K' '525.00K' '1000.00K' '60.00K'
 '1.20K' '395.00K' '40.00K' '2.50K' '1.30M' '23.79K' '190.88K' '120.91K'
```

```
 '53.04K' '400.00K' '190.00K' '17.00K' '13.00K' '4.50K' '180.00K'
 '983.00K' '559.00K' '118.00K' '12.00K' '2.00M' '25.00M' '1250.00K'
 '4.00M' '37.00K' '27.00K' '1.00M' '3.50M' '700.00K' '14.00K' '16.00K'
 '32.50K' '17.50K' '120.00K' '5.00M' '3.60K' '130.00K' '7.00M' '30.00M'
 '9.00K' '20.00M' '15.00M' '12.50K' '55.00K' '37.50K' '52.50K' '750.00K'
 '1.80M' '15.35M' '6.00M' '326.00K' '0.20M' '350.00K' '0.40K' '10.25M'
 '70.00K' '34.40M' '2.20K' '1.50M' '315.00K' '54.00K' '105.00K' '150.00M'
 '10.00M' '7000.00K' '18.00K' '50.00M' '85.00K' '1.60K' '5.20M' '800.00K'
 '673.00K' '2.26M' '3.20M' '126.00K' '79.00K' '140.00K' '88.00K' '270.00K'
 '1.17M' '137.00K' '110.00K' '33.00K' '330.00K' '90.00K' '92.10M' '92.00M'
 '2000.00K' '22.00K' '6.50K' '175.00K' '8.50K' '48.00K' '114.00K'
 '164.00K' '1.07M' '1.21M' '3.40M' '194.00K' '11.50K' '47.00K' '450.00K'
 '224.00K' '23.50K' '67.00K' '95.00K' '3.75M' '28.00K' '19.00K' '64.00K'
 '65.00K' '575.00K' '275.00K' '80.00M' '1.40M' '550.00K' '1.20M' '2.70M'
 '40.00M' '160.00K' '4.18M' '721.00K' '3.00M' '1.10M' '241.00K' '2.15M'
 '19.00M' '125.00M' '0.75K' '53.30K' '100.00M' '34.00K' '260.00K'
 '155.00K' '12.50M' '112.00K' '58.00K' '290.00K' '975.00K' '6.50M'
 '215.00K' '31.00K' '6.80M' '540.00K' '21.50K' '4.50M' '138.00K' '78.00K'
 '1.40K' '23.00K' '706.00K' '165.00K' '230.00K' '128.00K' '82.00K'
 '86.00M' '16.00M' '2.50M' '35.00M' '5200.00K' '289.00K' '4500.00K'
 '2.90M' '2.80M' '12.00M' '950.00K' '32.00K' '6.40M' '3.10M' '670.00K'
 '933.00K' '570.00K' '880.00K' '564.00K' '333.00K' '5000.00K' '2.20M'
 '409.00K' '132.00K' '394.00K' '21.00K' '3.57M' '9.55M' '0.05K' '24.80M'
 '172.00K' '2.54M' '115.00K' '1.18M' '939.00K' '18.50K' '332.00K'
 '134.00K' '3.86M' '1.72M' '54.52M' '355.00K' '2.30M' '191.00K' '13.50K'
 '1.25K' '758.00K' '259.00K' '582.00K' '2.25M' '25.59M' '59.00K' '352.00K'
 '251.00K' '1.02M' '4.13M' '1.58M' '227.00K' '1.90M' '2.44M' '13.20M'
 '108.00K' '3.60M' '20.70M' '11.30M' '211.00K' '311.00K' '273.50K'
 '225.00K' '8000.00K' '6.10K' '42.00K' '850.00K' '24.00K' '23.00M'
 '70.00M']
11058 missing or invalid entries found in DAMAGE_PROPERTY after conversion.
Remaining values in DAMAGE_PROPERTY: [0.0000e+00        nan 1.0000e+03
5.0000e+02 2.0000e+03 2.5000e+04
 5.0000e+03 4.0000e+03 1.0000e+04 2.0000e+04 1.5000e+04 5.0000e+04
 7.5000e+03 1.5000e+03 1.1000e+03 7.5000e+04 2.5000e+02 1.0000e+05
 3.5000e+03 3.0000e+02 2.0000e+02 1.0000e+01 1.2500e+05 3.0000e+04
 3.0000e+03 3.0000e+05 1.5000e+05 3.5000e+04 2.5000e+05 1.1000e+04
 6.0000e+03 7.0000e+03 4.5000e+04 5.5000e+03 2.0000e+05 5.1170e+04
 3.6070e+04 4.2800e+05 2.6000e+04 8.0000e+02 1.0000e+02 6.0000e+05
 8.0000e+04 8.0000e+03 5.0000e+05 5.2500e+05 1.0000e+06 6.0000e+04
 1.2000e+03 3.9500e+05 4.0000e+04 2.5000e+03 1.3000e+06 2.3790e+04
 1.9088e+05 1.2091e+05 5.3040e+04 4.0000e+05 1.9000e+05 1.7000e+04
 1.3000e+04 4.5000e+03 1.8000e+05 9.8300e+05 5.5900e+05 1.1800e+05
 1.2000e+04 2.0000e+06 2.5000e+07 1.2500e+06 4.0000e+06 3.7000e+04
 2.7000e+04 3.5000e+06 7.0000e+05 1.4000e+04 1.6000e+04 3.2500e+04
 1.7500e+04 1.2000e+05 5.0000e+06 3.6000e+03 1.3000e+05 7.0000e+06
 3.0000e+07 9.0000e+03 2.0000e+07 1.5000e+07 1.2500e+04 5.5000e+04
 3.7500e+04 5.2500e+04 7.5000e+05 1.8000e+06 1.5350e+07 6.0000e+06
```

```
 3.2600e+05 3.5000e+05 4.0000e+02 1.0250e+07 7.0000e+04 3.4400e+07
 2.2000e+03 1.5000e+06 3.1500e+05 5.4000e+04 1.0500e+05 1.5000e+08
 1.0000e+07 1.8000e+04 5.0000e+07 8.5000e+04 1.6000e+03 5.2000e+06
 8.0000e+05 6.7300e+05 2.2600e+06 3.2000e+06 1.2600e+05 7.9000e+04
 1.4000e+05 8.8000e+04 2.7000e+05 1.1700e+06 1.3700e+05 1.1000e+05
 3.3000e+04 3.3000e+05 9.0000e+04 9.2100e+07 9.2000e+07 2.2000e+04
 6.5000e+03 1.7500e+05 8.5000e+03 4.8000e+04 1.1400e+05 1.6400e+05
 1.0700e+06 1.2100e+06 3.4000e+06 1.9400e+05 1.1500e+04 4.7000e+04
 4.5000e+05 2.2400e+05 2.3500e+04 6.7000e+04 9.5000e+04 3.7500e+06
 2.8000e+04 1.9000e+04 6.4000e+04 6.5000e+04 5.7500e+05 2.7500e+05
 8.0000e+07 1.4000e+06 5.5000e+05 1.2000e+06 2.7000e+06 4.0000e+07
 1.6000e+05 4.1800e+06 7.2100e+05 3.0000e+06 1.1000e+06 2.4100e+05
 2.1500e+06 1.9000e+07 1.2500e+08 7.5000e+02 5.3300e+04 1.0000e+08
 3.4000e+04 2.6000e+05 1.5500e+05 1.2500e+07 1.1200e+05 5.8000e+04
 2.9000e+05 9.7500e+05 6.5000e+06 2.1500e+05 3.1000e+04 6.8000e+06
 5.4000e+05 2.1500e+04 4.5000e+06 1.3800e+05 7.8000e+04 1.4000e+03
 2.3000e+04 7.0600e+05 1.6500e+05 2.3000e+05 1.2800e+05 8.2000e+04
 8.6000e+07 1.6000e+07 2.5000e+06 3.5000e+07 2.8900e+05 2.9000e+06
 2.8000e+06 1.2000e+07 9.5000e+05 3.2000e+04 6.4000e+06 3.1000e+06
 6.7000e+05 9.3300e+05 5.7000e+05 8.8000e+05 5.6400e+05 3.3300e+05
 2.2000e+06 4.0900e+05 1.3200e+05 3.9400e+05 2.1000e+04 3.5700e+06
 9.5500e+06 5.0000e+01 2.4800e+07 1.7200e+05 2.5400e+06 1.1500e+05
 1.1800e+06 9.3900e+05 1.8500e+04 3.3200e+05 1.3400e+05 3.8600e+06
 1.7200e+06 5.4520e+07 3.5500e+05 2.3000e+06 1.9100e+05 1.3500e+04
 1.2500e+03 7.5800e+05 2.5900e+05 5.8200e+05 2.2500e+06 2.5590e+07
 5.9000e+04 3.5200e+05 2.5100e+05 1.0200e+06 4.1300e+06 1.5800e+06
 2.2700e+05 1.9000e+06 2.4400e+06 1.3200e+07 1.0800e+05 3.6000e+06
 2.0700e+07 1.1300e+07 2.1100e+05 3.1100e+05 2.7350e+05 2.2500e+05
 8.0000e+06 6.1000e+03 4.2000e+04 8.5000e+05 2.4000e+04 2.3000e+07
 7.0000e+07]
Unique values in DAMAGE_CROPS before conversion: ['0.00K' nan '30.00K' '0.50K'
'10.00K' '12.00K' '5.00K' '200.00K' '50.00K'
 '100.00K' '1.00K' '15.00K' '3.30M' '2.00K' '4.00K' '0.10K' '70.00K'
 '150.00K' '35.00K' '40.00K' '92.00K' '700.00K' '7.00K' '3.00K' '6.00K'
 '115.00K' '45.00K' '11.00K' '8.00K' '28.00K' '300.00K' '280.00K' '18.00K'
 '25.00K' '500.00K' '360.00K' '14.00K' '345.00K' '55.00K' '3.00M'
 '1000.00K' '80.00K' '20.00K' '9.00K' '2500.00K' '125.00K' '250.00K'
 '103.00K' '0.75K' '1.00M' '17.00K' '83.00K' '9.50M' '650.00K' '5.00M'
 '56.00K' '141.00K' '2.00M' '2.50K' '0.20K' '298.60K' '16.95K' '75.00K'
 '43.00K' '0.70K' '6.00M' '4.00M' '87.25K' '2.50M' '75.00M']
11046 missing or invalid entries found in DAMAGE_CROPS after conversion.
Remaining values in DAMAGE_CROPS: [0.000e+00       nan 3.000e+04 5.000e+02
1.000e+04 1.200e+04 5.000e+03
 2.000e+05 5.000e+04 1.000e+05 1.000e+03 1.500e+04 3.300e+06 2.000e+03
 4.000e+03 1.000e+02 7.000e+04 1.500e+05 3.500e+04 4.000e+04 9.200e+04
 7.000e+05 7.000e+03 3.000e+03 6.000e+03 1.150e+05 4.500e+04 1.100e+04
 8.000e+03 2.800e+04 3.000e+05 2.800e+05 1.800e+04 2.500e+04 5.000e+05
 3.600e+05 1.400e+04 3.450e+05 5.500e+04 3.000e+06 1.000e+06 8.000e+04
```

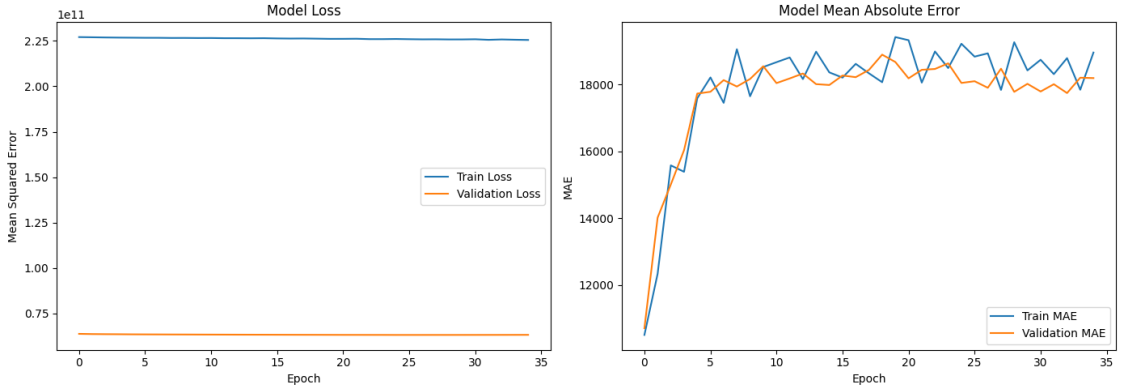
```
 2.000e+04 9.000e+03 2.500e+06 1.250e+05 2.500e+05 1.030e+05 7.500e+02
 1.700e+04 8.300e+04 9.500e+06 6.500e+05 5.000e+06 5.600e+04 1.410e+05
 2.000e+06 2.500e+03 2.000e+02 2.986e+05 1.695e+04 7.500e+04 4.300e+04
 7.000e+02 6.000e+06 4.000e+06 8.725e+04 7.500e+07]
Unique values in MAGNITUDE before conversion: [ 56.    50.      nan  59.     1.
54.     1.25  61.    66.    42.
  35.    72.    37.    58.     1.5   38.    36.    53.     2.    39.
  52.    55.     1.75  70.     0.75  65.    51.    71.    41.     0.88
   2.5    3.5    2.75  43.     1.82   2.25   3.     3.75  47.    45.
  63.    60.    73.    57.    34.    62.    30.    25.    29.    27.
  32.    28.    33.    26.    40.    46.    31.    68.    17.   129.
  67.   121.    97.    69.    96.    85.    89.   110.    90.    74.
  86.   114.   111.    91.   126.   122.    92.   109.    77.    76.
  99.   120.    75.    64.    80.    82.    48.    78.    44.     4.
  83.    93.   104.     4.5    3.25  49.    88.    79.    81.   100.
  84.     6.     1.1    0.7    0.95   0.25  87.     2.85  20.     3.73
   1.9    0.5   21.     0.38   4.4  105.   134.   103.     3.2    5.
   2.3   95.     6.12   4.25   2.4  125.     2.52   1.19   1.13   1.3
   2.2   15.     2.1   94.     2.65  23.     7.02   3.4    1.2    3.8
   1.8 ]
22298 missing or invalid entries found in MAGNITUDE after conversion.
Remaining values in MAGNITUDE: [ 56.    50.      nan  59.     1.    54.     1.25
61.    66.    42.
  35.    72.    37.    58.     1.5   38.    36.    53.     2.    39.
  52.    55.     1.75  70.     0.75  65.    51.    71.    41.     0.88
   2.5    3.5    2.75  43.     1.82   2.25   3.     3.75  47.    45.
  63.    60.    73.    57.    34.    62.    30.    25.    29.    27.
  32.    28.    33.    26.    40.    46.    31.    68.    17.   129.
  67.   121.    97.    69.    96.    85.    89.   110.    90.    74.
  86.   114.   111.    91.   126.   122.    92.   109.    77.    76.
  99.   120.    75.    64.    80.    82.    48.    78.    44.     4.
  83.    93.   104.     4.5    3.25  49.    88.    79.    81.   100.
  84.     6.     1.1    0.7    0.95   0.25  87.     2.85  20.     3.73
   1.9    0.5   21.     0.38   4.4  105.   134.   103.     3.2    5.
   2.3   95.     6.12   4.25   2.4  125.     2.52   1.19   1.13   1.3
   2.2   15.     2.1   94.     2.65  23.     7.02   3.4    1.2    3.8
   1.8 ]
Column 'BEGIN_AZIMUTH' is not numeric. Attempting to convert…
50623 entries in 'BEGIN_AZIMUTH' could not be converted and are set to NaN.
Column 'BEGIN_LOCATION' is not numeric. Attempting to convert…
50623 entries in 'BEGIN_LOCATION' could not be converted and are set to NaN.
Column 'END_AZIMUTH' is not numeric. Attempting to convert…
50623 entries in 'END_AZIMUTH' could not be converted and are set to NaN.
Column 'END_LOCATION' is not numeric. Attempting to convert…
50623 entries in 'END_LOCATION' could not be converted and are set to NaN.
Removing numerical features with all NaN values: BEGIN_LOCATION, END_AZIMUTH,
BEGIN_AZIMUTH, END_LOCATION
Missing values imputed.
```

```
Preprocessing complete.

Dropped 30621 records due to NaN values in target variables.
Splitting data into training and testing sets…
Training set size: 16001 records
Testing set size: 4001 records

Splitting training data into training and validation sets…
New Training set size: 12800 records
Validation set size: 3201 records

Building the neural network model…
Model built successfully.

Training the model…
Epoch 1/100
400/400             2s 3ms/step -
loss: 133243961344.0000 - mae: 8930.7188 - val_loss: 63757979648.0000 - val_mae:
10713.2930
Epoch 2/100
400/400             1s 3ms/step -
loss: 82368847872.0000 - mae: 8680.0146 - val_loss: 63628607488.0000 - val_mae:
14022.0771
Epoch 3/100
400/400             1s 3ms/step -
loss: 205072367616.0000 - mae: 16708.0020 - val_loss: 63571849216.0000 -
val_mae: 15014.2764
Epoch 4/100
400/400             1s 2ms/step -
loss: 70509346816.0000 - mae: 12689.0527 - val_loss: 63533457408.0000 - val_mae:
16039.5791
Epoch 5/100
400/400             1s 3ms/step -
loss: 99829530624.0000 - mae: 14987.7617 - val_loss: 63481221120.0000 - val_mae:
17732.8281
Epoch 6/100
400/400             1s 3ms/step -
loss: 278647472128.0000 - mae: 17570.0332 - val_loss: 63452016640.0000 -
val_mae: 17787.4746
Epoch 7/100
400/400             1s 3ms/step -
loss: 148969160704.0000 - mae: 14397.6660 - val_loss: 63429681152.0000 -
val_mae: 18136.2207
Epoch 8/100
400/400             1s 3ms/step -
loss: 258799828992.0000 - mae: 20185.4082 - val_loss: 63401132032.0000 -
val_mae: 17944.5977
Epoch 9/100
```

```
400/400            1s 3ms/step -
loss: 122133979136.0000 - mae: 14536.7119 - val_loss: 63386046464.0000 -
val_mae: 18170.1660
Epoch 10/100
400/400            1s 2ms/step -
loss: 177885380608.0000 - mae: 17424.8574 - val_loss: 63359913984.0000 -
val_mae: 18552.3828
Epoch 11/100
400/400            1s 3ms/step -
loss: 391820247040.0000 - mae: 20748.3770 - val_loss: 63339982848.0000 -
val_mae: 18046.6328
Epoch 12/100
400/400            1s 2ms/step -
loss: 562256871424.0000 - mae: 24336.4219 - val_loss: 63321714688.0000 -
val_mae: 18185.9766
Epoch 13/100
400/400            1s 3ms/step -
loss: 154343735296.0000 - mae: 16055.4434 - val_loss: 63302287360.0000 -
val_mae: 18333.9863
Epoch 14/100
400/400            1s 3ms/step -
loss: 291287433216.0000 - mae: 19852.2324 - val_loss: 63287037952.0000 -
val_mae: 18015.6328
Epoch 15/100
400/400            1s 3ms/step -
loss: 149532721152.0000 - mae: 17571.6895 - val_loss: 63273447424.0000 -
val_mae: 17990.2969
Epoch 16/100
400/400            1s 2ms/step -
loss: 194655748096.0000 - mae: 16858.6133 - val_loss: 63257382912.0000 -
val_mae: 18273.4941
Epoch 17/100
400/400            1s 2ms/step -
loss: 139147395072.0000 - mae: 17014.0703 - val_loss: 63241256960.0000 -
val_mae: 18224.0625
Epoch 18/100
400/400            1s 2ms/step -
loss: 109062389760.0000 - mae: 16310.8643 - val_loss: 63228166144.0000 -
val_mae: 18440.8555
Epoch 19/100
400/400            1s 3ms/step -
loss: 68800585728.0000 - mae: 15387.0137 - val_loss: 63218946048.0000 - val_mae:
18898.5176
Epoch 20/100
400/400            1s 3ms/step -
loss: 315668299776.0000 - mae: 21014.8711 - val_loss: 63203913728.0000 -
val_mae: 18677.7637
Epoch 21/100
```

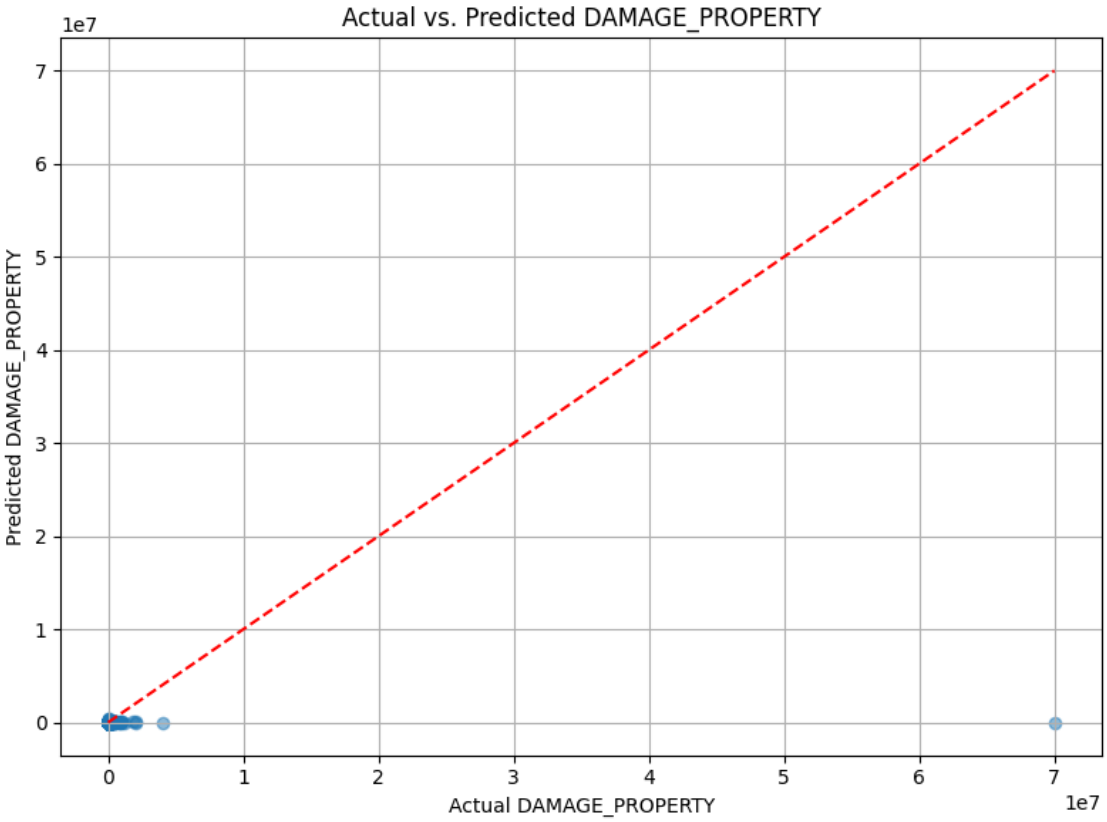
```
400/400            1s 3ms/step -
loss: 292363567104.0000 - mae: 21888.2422 - val_loss: 63186599936.0000 -
val_mae: 18189.3418
Epoch 22/100
400/400            1s 3ms/step -
loss: 134409904128.0000 - mae: 15784.5879 - val_loss: 63182401536.0000 -
val_mae: 18444.7363
Epoch 23/100
400/400            1s 3ms/step -
loss: 261463769088.0000 - mae: 20160.3145 - val_loss: 63174004736.0000 -
val_mae: 18469.7617
Epoch 24/100
400/400            1s 3ms/step -
loss: 118780329984.0000 - mae: 15752.6758 - val_loss: 63169384448.0000 -
val_mae: 18638.0215
Epoch 25/100
400/400            1s 3ms/step -
loss: 219464105984.0000 - mae: 19679.7539 - val_loss: 63150137344.0000 -
val_mae: 18051.6992
Epoch 26/100
400/400            1s 3ms/step -
loss: 233179086848.0000 - mae: 18080.0332 - val_loss: 63153397760.0000 -
val_mae: 18103.3828
Epoch 27/100
400/400            1s 3ms/step -
loss: 269264027648.0000 - mae: 19743.0410 - val_loss: 63152951296.0000 -
val_mae: 17907.0996
Epoch 28/100
400/400            1s 2ms/step -
loss: 70359105536.0000 - mae: 15250.2676 - val_loss: 63158685696.0000 - val_mae:
18477.2090
Epoch 29/100
400/400            1s 2ms/step -
loss: 432567418880.0000 - mae: 22763.7070 - val_loss: 63155773440.0000 -
val_mae: 17785.5547
Epoch 30/100
400/400            1s 2ms/step -
loss: 163270197248.0000 - mae: 18296.0117 - val_loss: 63161950208.0000 -
val_mae: 18025.0137
Epoch 31/100
400/400            1s 3ms/step -
loss: 435720880128.0000 - mae: 22120.7188 - val_loss: 63169146880.0000 -
val_mae: 17795.8691
Epoch 32/100
400/400            1s 3ms/step -
loss: 155776122880.0000 - mae: 16450.8027 - val_loss: 63176327168.0000 -
val_mae: 18013.7480
Epoch 33/100
```

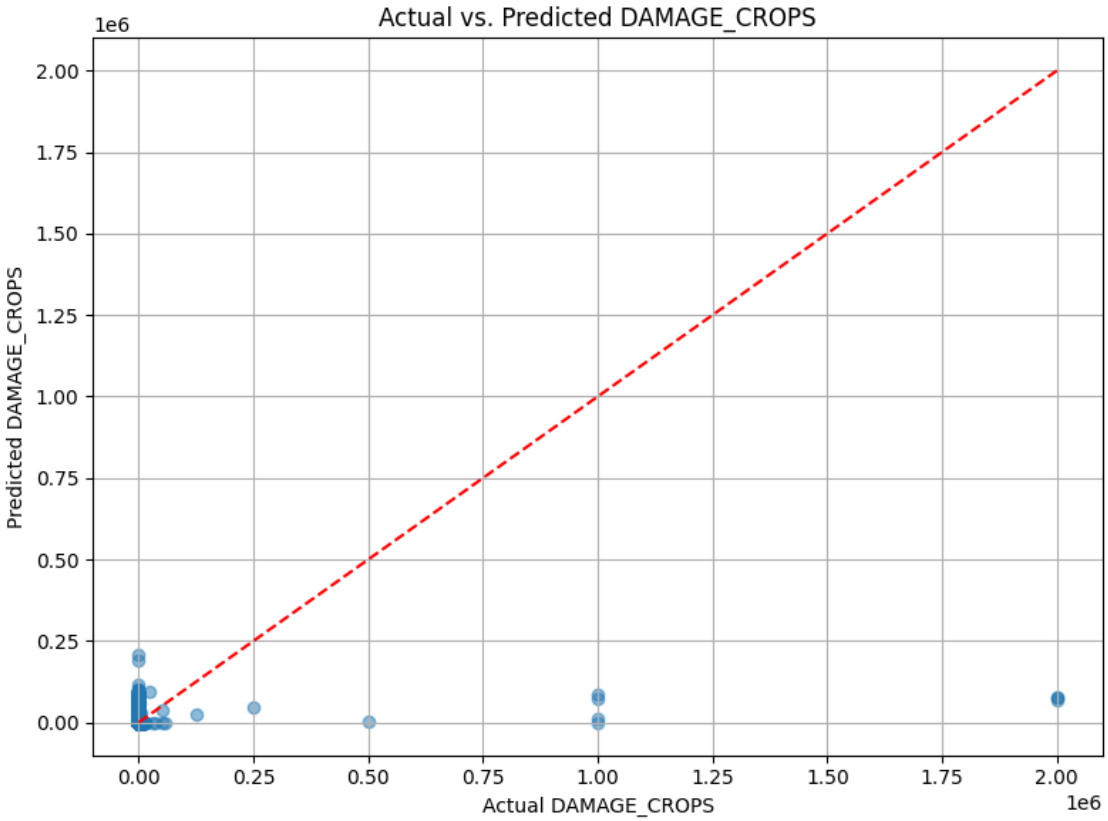
```
400/400             1s 3ms/step -
loss: 206723825664.0000 - mae: 18800.4668 - val_loss: 63182491648.0000 -
val_mae: 17749.1465
Epoch 34/100
400/400             1s 3ms/step -
loss: 65105465344.0000 - mae: 14635.5000 - val_loss: 63195561984.0000 - val_mae:
18208.8301
Epoch 35/100
400/400             1s 3ms/step -
loss: 243450937344.0000 - mae: 20536.8906 - val_loss: 63208812544.0000 -
val_mae: 18196.6074
Model training complete.
```

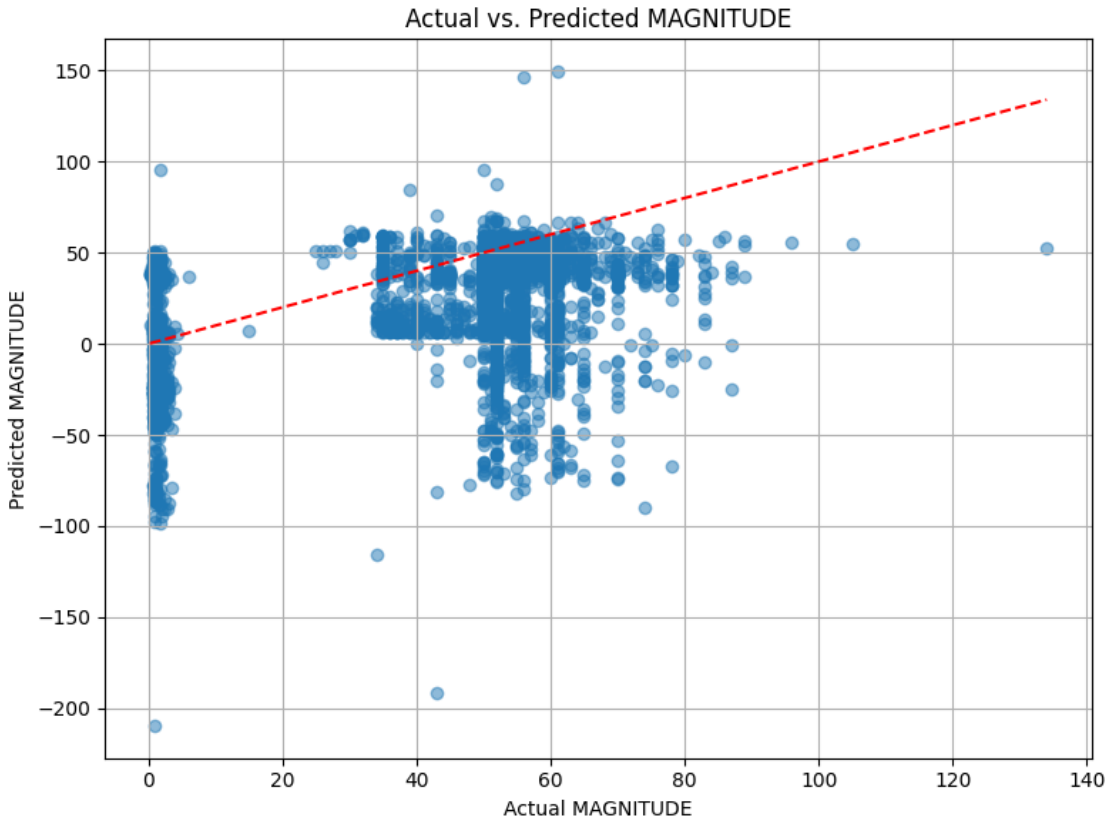



```
Evaluating the model on the test set…
Test Mean Squared Error: 412814606336.00
Test Mean Absolute Error (MAE): 18300.85

126/126             0s 2ms/step
Test R² Score for DAMAGE_PROPERTY: -0.00
Test R² Score for DAMAGE_CROPS: -0.03
Test R² Score for MAGNITUDE: -1.29

Some predictions vs actual values:

Target: DAMAGE_PROPERTY
Actual: 5000.00, Predicted: 2175.40
Actual: 2000.00, Predicted: 2875.51
Actual: 0.00, Predicted: 2854.03
Actual: 0.00, Predicted: 1363.05
Actual: 1000.00, Predicted: 2189.93
Actual: 0.00, Predicted: 2670.29
Actual: 25000.00, Predicted: 2307.76
Actual: 0.00, Predicted: 3921.85
```

```
Actual: 0.00, Predicted: 151961.61
Actual: 0.00, Predicted: 36311.86

Target: DAMAGE_CROPS
Actual: 0.00, Predicted: 32.06
Actual: 0.00, Predicted: 37.70
Actual: 5000.00, Predicted: 41.13
Actual: 0.00, Predicted: 19.76
Actual: 0.00, Predicted: 31.48
Actual: 0.00, Predicted: 36.10
Actual: 0.00, Predicted: 31.45
Actual: 0.00, Predicted: 48.82
Actual: 0.00, Predicted: 83016.12
Actual: 0.00, Predicted: 19685.96

Target: MAGNITUDE
Actual: 50.00, Predicted: 29.89
Actual: 50.00, Predicted: 36.89
Actual: 50.00, Predicted: 37.45
Actual: 61.00, Predicted: 20.61
Actual: 50.00, Predicted: 29.85
Actual: 50.00, Predicted: 34.69
Actual: 66.00, Predicted: 30.77
Actual: 50.00, Predicted: 48.17
Actual: 1.25, Predicted: -78.73
Actual: 60.00, Predicted: -7.38

Prediction Summary Statistics:

Target: DAMAGE_PROPERTY
Mean Prediction: 19265.21
Median Prediction: 3464.64
Standard Deviation of Predictions: 33589.38

Target: DAMAGE_CROPS
Mean Prediction: 9343.12
Median Prediction: 46.79
Standard Deviation of Predictions: 18855.71

Target: MAGNITUDE
Mean Prediction: 20.49
Median Prediction: 34.13
Standard Deviation of Predictions: 31.36
```


Actual vs. Predicted DAMAGE_PROPERTY
1e7
Predicted DAMAGE_PROPERTY
Actual DAMAGE_PROPERTY
1e7
0
1
2
3
4
5
6
7


Actual vs. Predicted DAMAGE_CROPS
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Predicted DAMAGE_CROPS
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
Actual DAMAGE_CROPS




```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

Model and preprocessor saved successfully.
```

2.10 Execute the Main Function

Finally, we call the `main()` function to execute the workflow.

3 Execute the main function

main()