

# **EARTHQUAKE INTENSITY PREDICTION**

**MACHINE  
LEARNING PROJECT**



**DONE BY:**

**ABHI LAVANYA-2022103068  
NOLA DAISY THOMAS:2022103072  
CATHRENE JESINTHA G:2022103061  
HARSHIKA SENTHIL-2022103069**

# INDEX

Introduction .....	2
1. Dataset and Data Preprocessing .....	2
Data Augmentation and Categorization: .....	3
2. Feature Scaling and Model Training.....	4
Machine Learning Models: .....	4
3. Model Evaluation and Performance .....	7
4. Real-Time Data Visualization .....	8
Visualization Steps:.....	8
5. Web Application for User Interaction .....	9
6. Model Deployment and Continuous Learning .....	11
7. Benefits and Applications .....	11
8. Future Enhancements and Developments.....	12
Source code .....	13
Conclusion.....	17
References .....	17

# EARTHQUAKE INTENSITY PREDICTION

## Introduction

Earthquakes are unpredictable natural phenomena that can cause catastrophic damage, making understanding their intensity crucial for disaster management and preparedness. This project leverages advanced machine learning techniques to predict earthquake magnitudes based on input features such as latitude, longitude, and depth. The system integrates a Python backend using the Flask framework for model inference, coupled with a user-friendly HTML frontend that allows users to input data and receive predictions along with insightful data visualizations. The workflow encompasses data preprocessing, model training, and the integration of machine learning models into a web application for real-time predictions. Additionally, it incorporates real-time data visualization, allowing users to explore seismic activity geographically and temporally. By combining predictive modeling and interactive exploration, the Earthquake Intensity Prediction System provides a comprehensive tool for analyzing, predicting, and visualizing seismic events, supporting disaster preparedness, research, and risk mitigation in seismology.

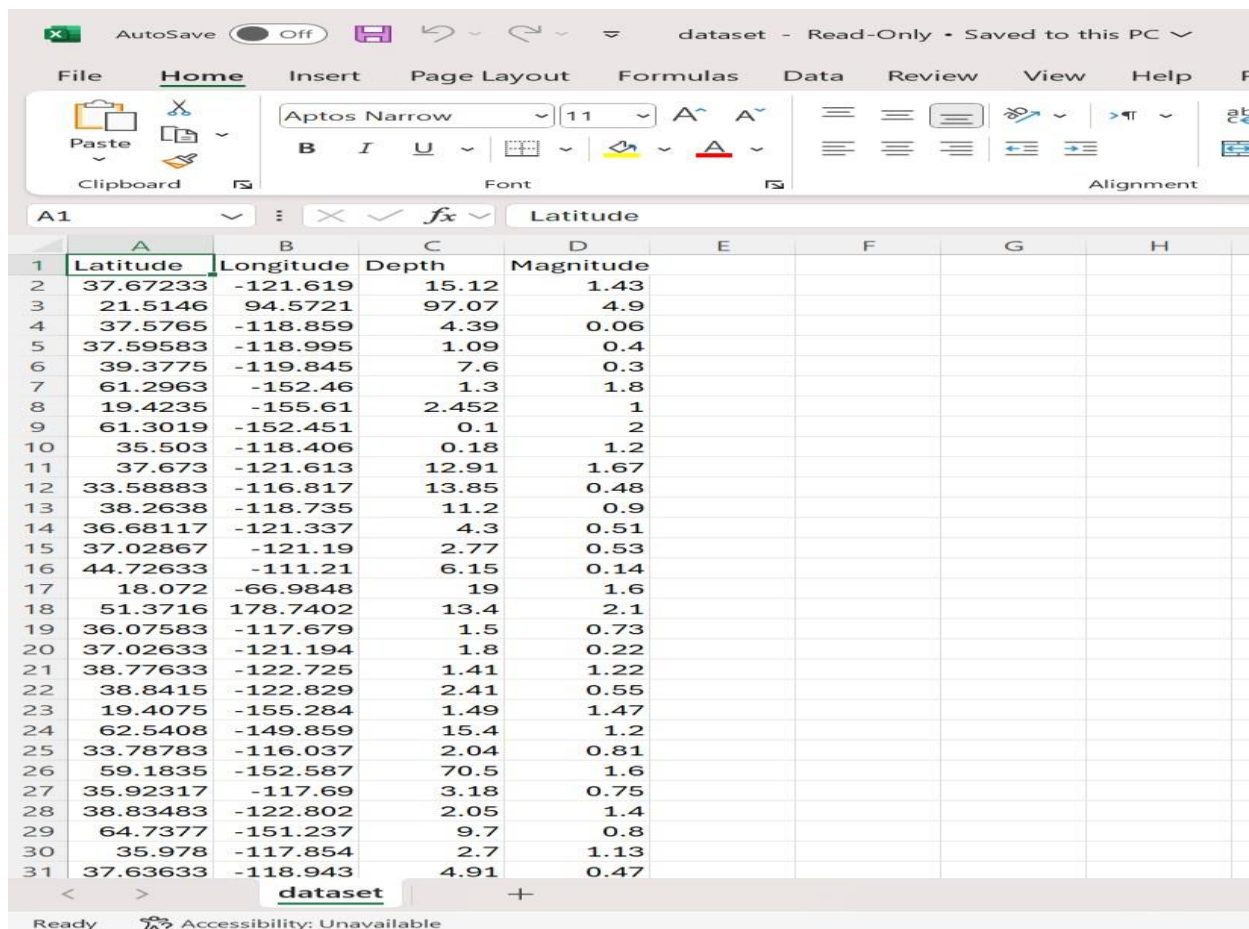
## 1. Dataset and Data Preprocessing

The foundation of the system is built upon a detailed dataset containing earthquake information. Key features typically include:

- **Latitude and Longitude:** These represent the geographical location of the earthquake event.

- **Depth:** The depth at which the earthquake originated, which often correlates with its potential intensity.
- **Magnitude:** A measure of the earthquake's energy release, which is a key feature for predicting its intensity.

The data is preprocessed to handle missing values, outliers, and date-time conversions, ensuring that it is ready for analysis and model training. Time-related features are especially important for filtering and performing time-based analysis.



	A	B	C	D	E	F	G	H
1	Latitude	Longitude	Depth	Magnitude				
2	37.67233	-121.619	15.12	1.43				
3	21.5146	94.5721	97.07	4.9				
4	37.5765	-118.859	4.39	0.06				
5	37.59583	-118.995	1.09	0.4				
6	39.3775	-119.845	7.6	0.3				
7	61.2963	-152.46	1.3	1.8				
8	19.4235	-155.61	2.452	1				
9	61.3019	-152.451	0.1	2				
10	35.503	-118.406	0.18	1.2				
11	37.673	-121.613	12.91	1.67				
12	33.58883	-116.817	13.85	0.48				
13	38.2638	-118.735	11.2	0.9				
14	36.68117	-121.337	4.3	0.51				
15	37.02867	-121.19	2.77	0.53				
16	44.72633	-111.21	6.15	0.14				
17	18.072	-66.9848	19	1.6				
18	51.3716	178.7402	13.4	2.1				
19	36.07583	-117.679	1.5	0.73				
20	37.02633	-121.194	1.8	0.22				
21	38.77633	-122.725	1.41	1.22				
22	38.8415	-122.829	2.41	0.55				
23	19.4075	-155.284	1.49	1.47				
24	62.5408	-149.859	15.4	1.2				
25	33.78783	-116.037	2.04	0.81				
26	59.1835	-152.587	70.5	1.6				
27	35.92317	-117.69	3.18	0.75				
28	38.83483	-122.802	2.05	1.4				
29	64.7377	-151.237	9.7	0.8				
30	35.978	-117.854	2.7	1.13				
31	37.63633	-118.943	4.91	0.47				

### Data Augmentation and Categorization:

To address the imbalance in the dataset, particularly in high-magnitude earthquake instances, synthetic data points are generated to represent more powerful earthquakes. This helps improve model robustness, especially for classifying earthquakes into different intensity categories:

- **Low Intensity:** Magnitude below 4
- **Medium Intensity:** Magnitude between 4 and 6
- **High Intensity:** Magnitude 6 and above

Magnitude values are categorized into these intensity levels, with additional synthetic data augmenting the representation of high-magnitude earthquakes, balancing the dataset for better prediction performance.

```
# Categorize Magnitude into Intensity Levels for Classification
def categorize_intensity(magnitude):
    """
    Categorize magnitude into intensity levels based on updated thresholds.
    """
    if magnitude < 4: # Adjusted threshold for "Low"
        return "Low"
    elif 4 <= magnitude < 6: # Adjusted range for "Medium"
        return "Medium"
    else:
        return "High" # "High" for anything 6.5 and above
```

## 2. Feature Scaling and Model Training

Feature scaling ensures that all input features contribute equally to the machine learning models. **MinMaxScaler** is used to normalize the features (Latitude, Longitude, and Depth), which helps to prevent any feature from dominating due to different scales.

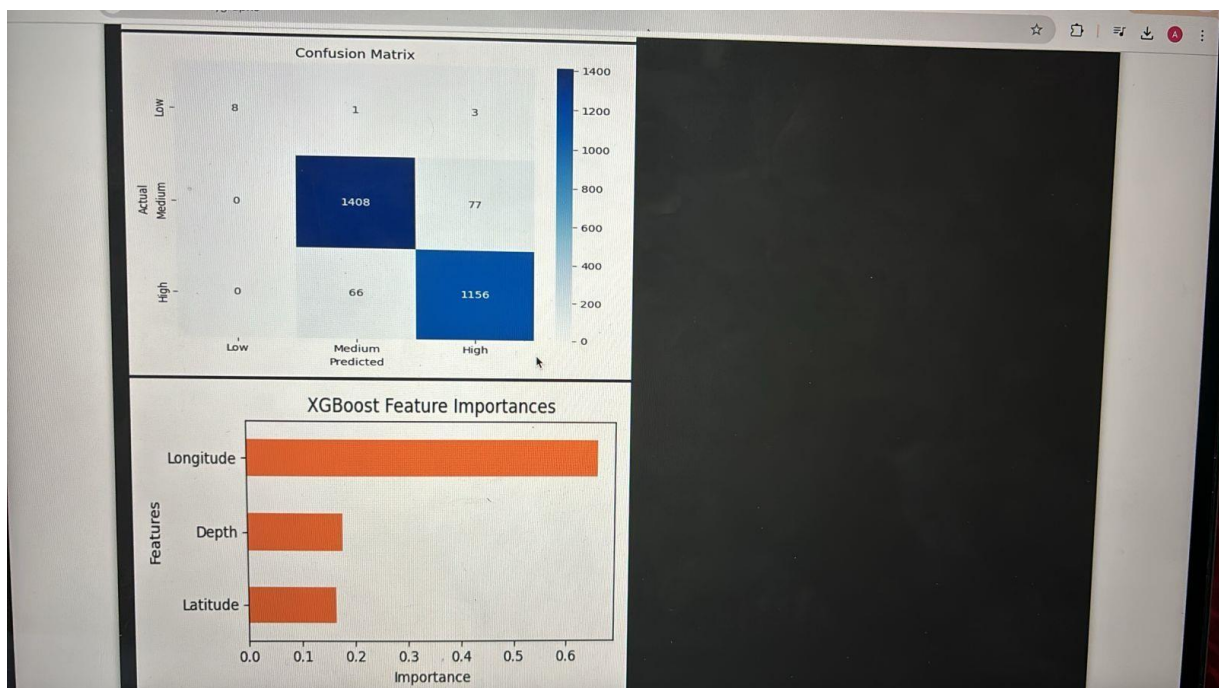
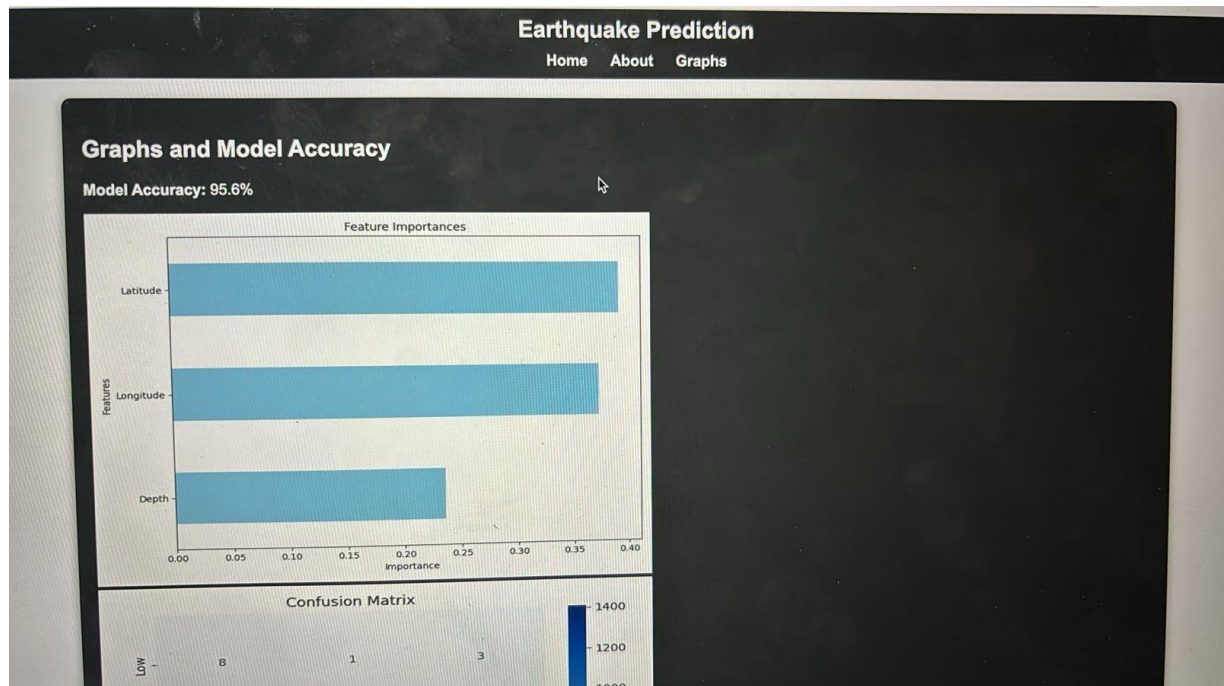
### Machine Learning Models:

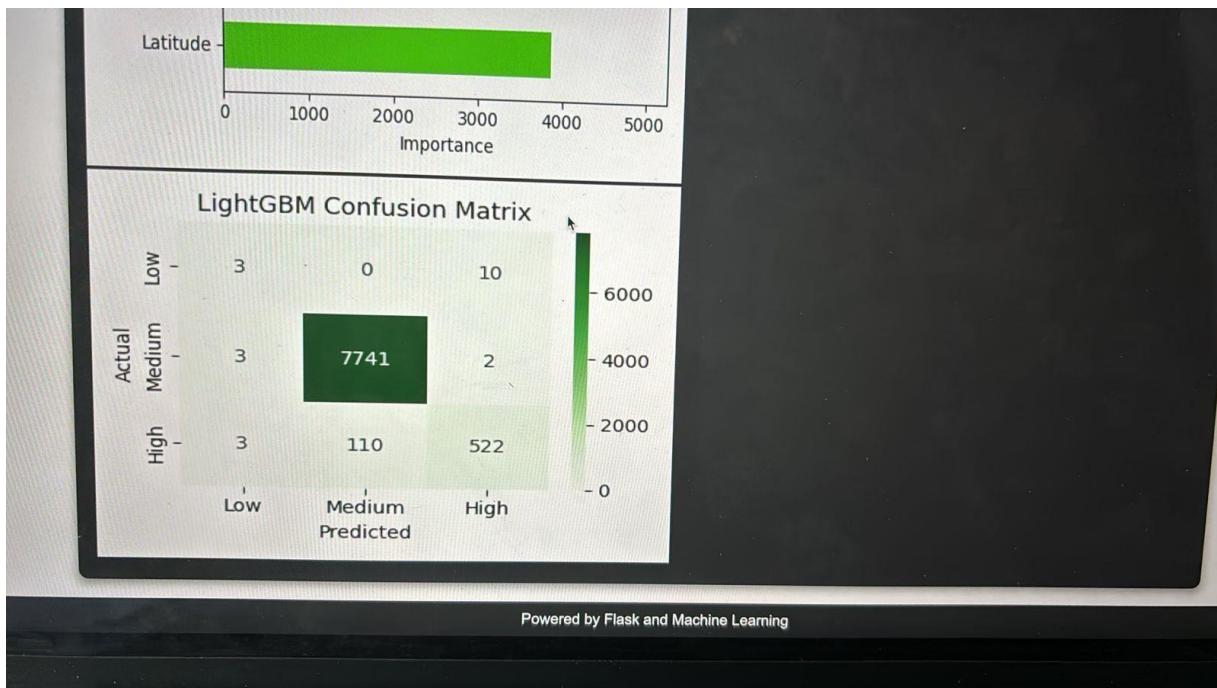
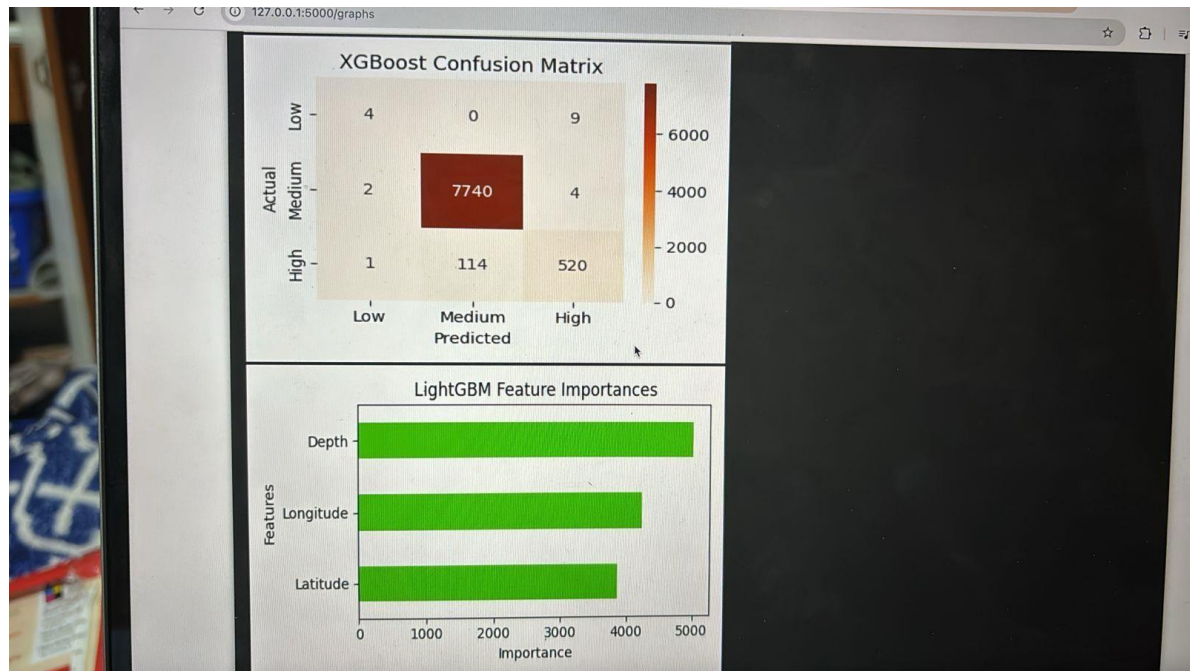
The core of the system lies in two machine learning models, both of which are based on popular gradient boosting algorithms—XGBoost (Extreme Gradient Boosting) and LightGBM (Light Gradient Boosting Machine). These models are efficient and widely used for classification and regression tasks due to their high performance and speed. The system leverages these algorithms to predict and classify earthquake intensity. Specifically, these models are trained for two main tasks:

- **Random Forest Regressor:** This model predicts the continuous earthquake magnitude based on geographical features such as latitude, longitude, and depth. The regressor helps estimate the magnitude of future earthquakes, providing crucial information for assessing potential impact.
- **Random Forest Classifier:** This model classifies earthquake intensity levels into three categories: Low, Medium, and High, based on the predicted magnitude. By categorizing the earthquake's severity, the classifier assists in understanding the potential damage and supporting disaster management efforts.
- **XGBoost Classifier:** XGBoost (Extreme Gradient Boosting) is a powerful and widely-used machine learning algorithm for classification tasks. It is based on decision tree boosting, which helps improve model performance by iteratively correcting errors from previous models. Known for its high accuracy, efficient handling of missing data, and scalability, XGBoost is particularly effective for structured datasets. Its parallel processing capabilities and robust regularization techniques make it a strong choice for tackling large-scale and complex classification problems, including earthquake intensity prediction.
- **LightGBM Classifier:** Similar to the XGBoost classifier, LightGBM is also used to classify the earthquake intensity level. LightGBM is known for its fast training speed, lower memory usage, and better scalability with large datasets, which makes it a great alternative to XGBoost. It also helps improve the model's performance and handles large-scale data more efficiently.

Both models are trained using the preprocessed and augmented dataset, which incorporates geographical features and synthetic high-magnitude data to enhance the model's accuracy. To address class imbalance between the intensity levels, the classifier models use sample weights, ensuring that each intensity level is appropriately represented during training.









### 3. Model Evaluation and Performance

To measure the models' effectiveness, several evaluation metrics are applied:

- **Mean Squared Error (MSE)** for the regressor: Measures the difference between predicted and actual magnitudes, helping to evaluate the model's accuracy. Lower MSE indicates better predictive accuracy.
- **Accuracy Score** for the classifier: Evaluates how well the classifier predicts earthquake intensity categories. An accuracy score of 100% indicates perfect classification.
- **Confusion Matrix**: Provides more detailed insight into classifier performance, highlighting the number of correct and incorrect predictions for each intensity class.

The system's models are evaluated and fine-tuned periodically, ensuring they remain robust and continue to provide accurate predictions.

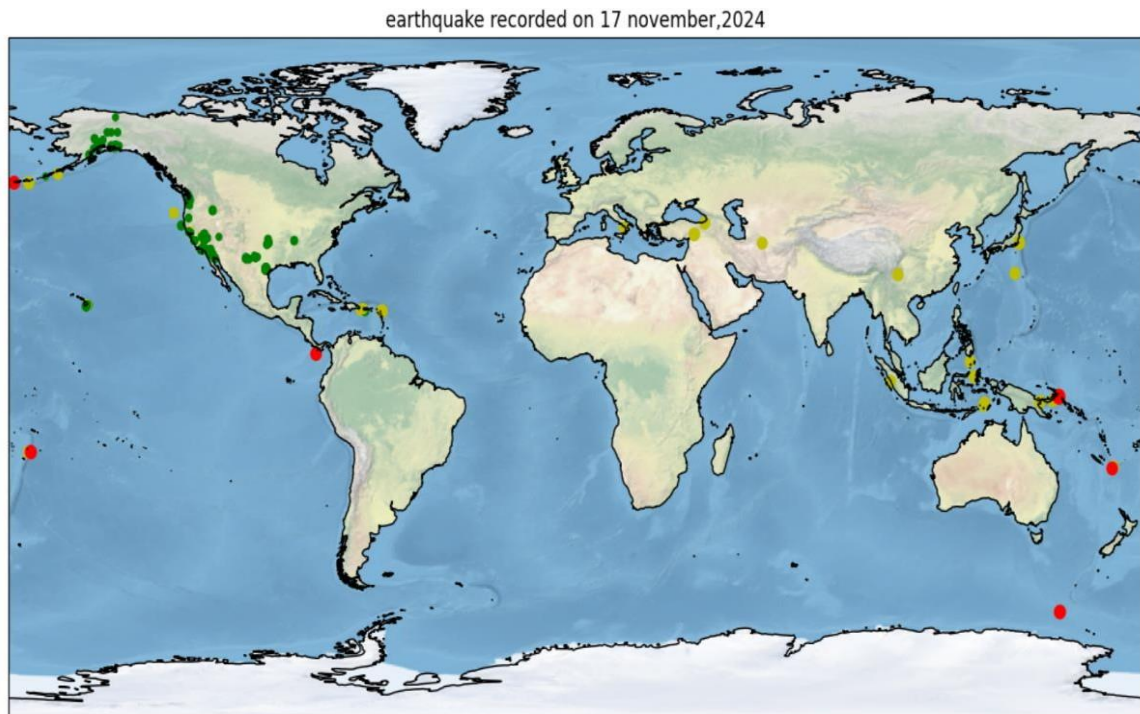
## 4. Real-Time Data Visualization

The system also supports real-time earthquake data visualization, allowing users to see seismic activity geographically and temporally. This is achieved by combining **Cartopy** for creating map projections with **Matplotlib** for plotting the data points.

### Visualization Steps:

- **Geographical Map:** A map with coastlines and country borders provides context, helping users visualize the global distribution of earthquakes. Cartopy is used to project the data on a world map.
- **Filtering Data by Date:** Users can filter earthquake data by specific time periods, allowing for more targeted analysis of seismic events within a selected timeframe.
- **Color-Coded Earthquake Points:** Earthquakes are displayed as color-coded points on the map, with the color indicating intensity (green for low, yellow for medium, and red for high intensity). The size of the points corresponds to the magnitude, with larger points representing stronger earthquakes.

- **Interactive Map:** Users can zoom in and out on the map to focus on specific regions, making it easier to analyze localized seismic activity.

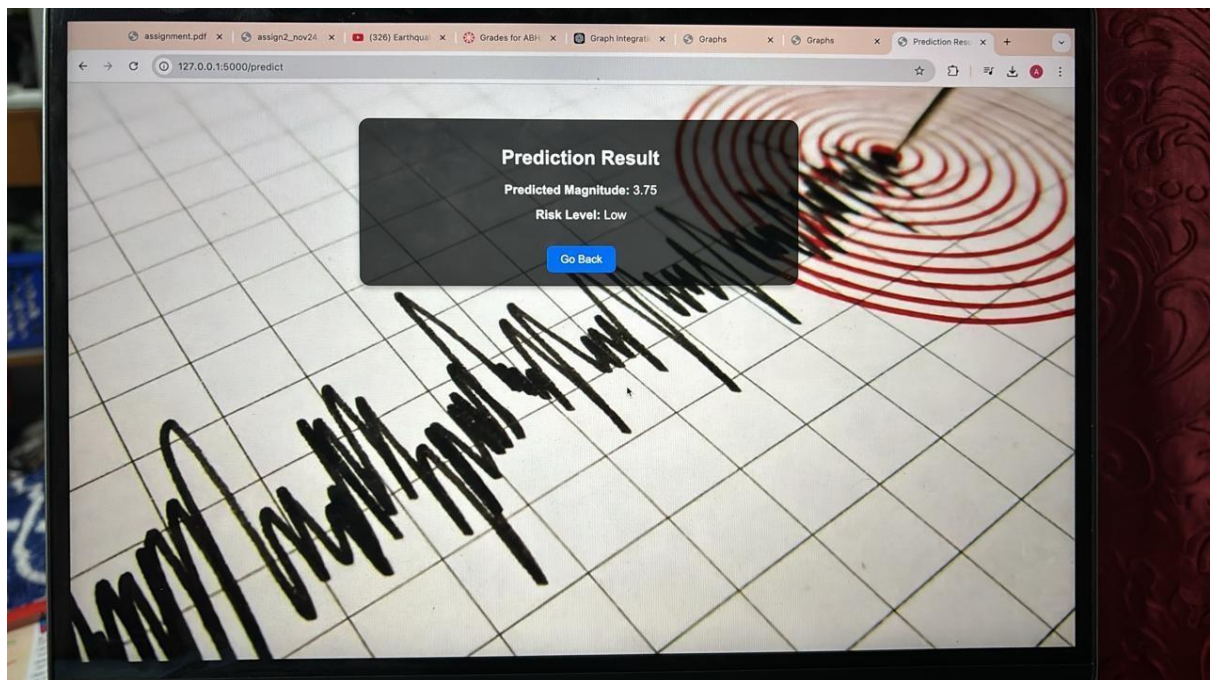
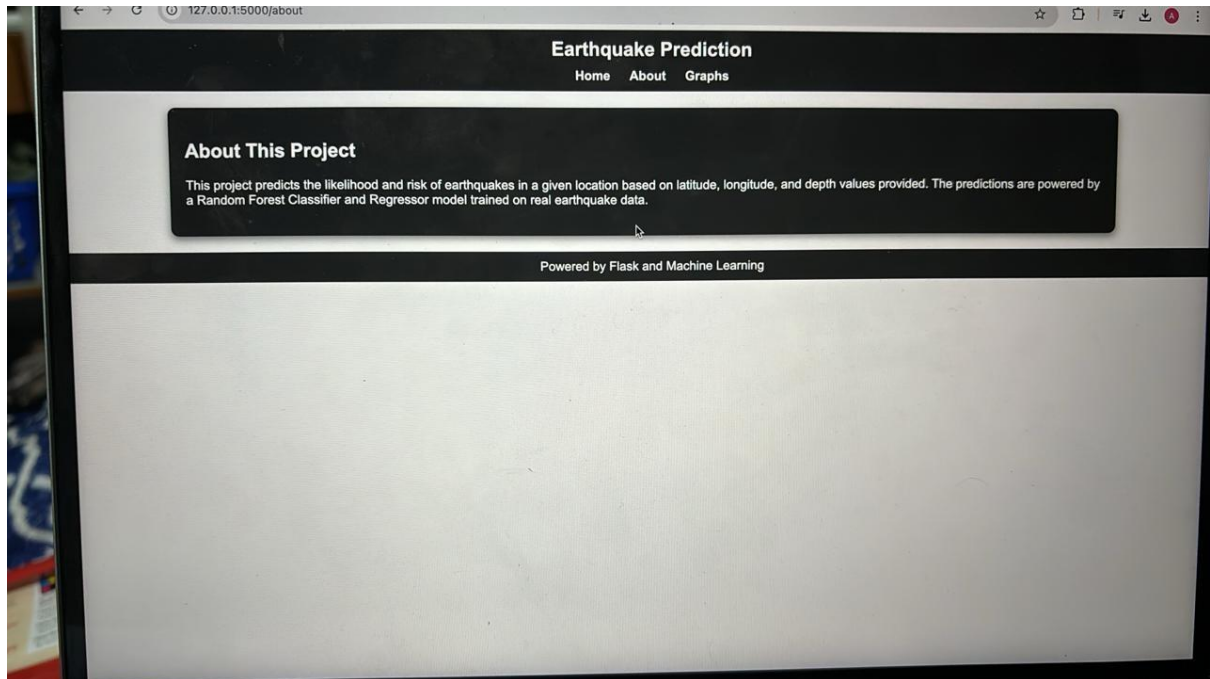


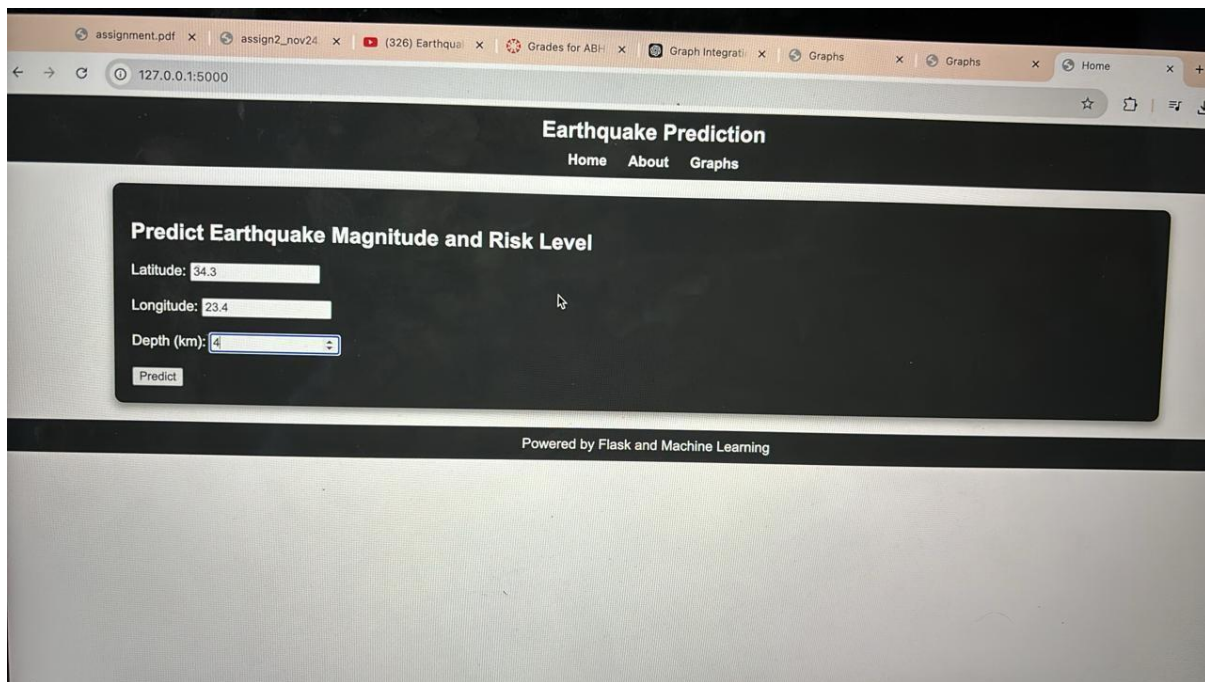
## 5. Web Application for User Interaction

The project also includes a **Flask-based web application**, providing an interactive interface for users to engage with the system. Key features include:

- **Prediction Form:** Users can input the latitude, longitude, and depth of an earthquake to predict its magnitude and intensity. The system then displays the prediction results.
- **Visualization Dashboard:** A dashboard provides users with access to the confusion matrix, feature importance graphs, and real-time earthquake maps. These visual tools give users a quick overview of model performance and seismic activity globally.
- **Interactive Results:** After entering earthquake data, users can view the predicted magnitude and intensity, as well as

additional insights drawn from historical data, all through the web interface.





## 6. Model Deployment and Continuous Learning

Once trained, the models (XGBoost Regressor and XGBoost Classifier) are serialized using **Pickle**, allowing them to be deployed and used in real-time predictions. The models are loaded within the web application, which processes user input and generates predictions on-demand.

The system is also designed for **continuous learning**, where it can periodically update its models by incorporating fresh earthquake data, ensuring ongoing accuracy and adaptation to changing seismic patterns.

## 7. Benefits and Applications

This integrated system provides numerous advantages, including:

- **Disaster Preparedness:** By predicting earthquake intensity, authorities can better prepare for potential impacts. Early warnings can be issued, and emergency response plans can be activated.



- **Seismic Research:** Researchers can use the system to analyze earthquake trends and investigate the relationship between geographical features and earthquake intensity.
- **Public Awareness:** The visualization tools help non-experts understand seismic risks and enhance public knowledge about earthquake preparedness.

## 8. Future Enhancements and Developments

While the system is functional, there are several areas for potential improvement:

- **Real-Time Earthquake Data Integration:** The system can be extended to include live earthquake data feeds, providing immediate predictions and visualizations for newly recorded seismic events.
- **More Advanced Features:** Adding additional features like fault lines, tectonic plate boundaries, and geological data can further enhance prediction accuracy and deepen the system's analysis.
- **Improved Modeling Techniques:** Incorporating deep learning models or hybrid approaches could further refine predictions, especially for large-magnitude earthquakes.

## Source code

```
import matplotlib
matplotlib.use('Agg') # Ensure non-interactive backend for plotting
from flask import Flask, render_template, request, url_for
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.metrics import confusion_matrix, accuracy_score

# Initialize Flask app
app = Flask(__name__)

# Load trained models and scaler
try:
    with open('scaler.pkl', 'rb') as scaler_file:
        scaler = pickle.load(scaler_file)

    with open('xgb_classifier.pkl', 'rb') as xgb_classifier_file:
        xgb_classifier = pickle.load(xgb_classifier_file)

    with open('xgb_regressor.pkl', 'rb') as xgb_regressor_file:
        xgb_regressor = pickle.load(xgb_regressor_file)

    with open('classifier.pkl', 'rb') as rf_classifier_file: # Load Random Forest classifier
        classifier = pickle.load(rf_classifier_file)

except FileNotFoundError as e:
    print(f"Error loading model: {e}")
    exit(1)

# File paths for temporary plot images
RF_CM_PATH = "static/confusion_matrix_rf.png"
RF_BAR_PATH = "static/bar_graph_rf.png"
XGB_CM_PATH = "static/confusion_matrix_xgb.png"
XGB_BAR_PATH = "static/bar_graph_xgb.png"

# Home route
@app.route('/')
def home():
    return render_template('homepage.html')
```

```
# Route for rendering graphs
@app.route('/graphs')
def graphs():
    try:
        # Load dataset
        data = pd.read_csv('dataset.csv')
        X = data[['Latitude', 'Longitude', 'Depth']]
        y = pd.Categorical(data['Magnitude']).apply(
            lambda x: "Low" if x < 4 else "Medium" if x < 6 else "High"
        ).codes

        # Scale features (function) transform: Any
        X_scaled = scaler.transform(X)

        ## --- RANDOM FOREST ANALYSIS ---
        # Random Forest Predictions
        y_rf_pred = classifier.predict(X) # Using Random Forest without scaling
        rf_accuracy = accuracy_score(y, y_rf_pred) * 100

        # Random Forest Confusion Matrix
        cm_rf = confusion_matrix(y, y_rf_pred)
        plt.figure(figsize=(4, 3))
        sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=["Low", "Medium", "High"],
                    yticklabels=["Low", "Medium", "High"])
        plt.title('Random Forest Confusion Matrix')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.tight_layout()
        plt.savefig(RF_CM_PATH)
        plt.close()

        # Random Forest Feature Importance Bar Graph
        feature_importances_rf = pd.Series(classifier.feature_importances_, index=X.columns)
        plt.figure(figsize=(5, 3))
        feature_importances_rf.sort_values().plot(kind='barh', color='skyblue')
        plt.title('Random Forest Feature Importances')
        plt.xlabel('Importance')
        plt.ylabel('Features')
        plt.tight_layout()
        plt.savefig(RF_BAR_PATH)
```

```

plt.close()

## --- XGBOOST ANALYSIS ---
# XGBoost Predictions
y_xgb_pred = xgb_classifier.predict(X_scaled)
xgb_accuracy = accuracy_score(y, y_xgb_pred) * 100

# XGBoost Confusion Matrix
cm_xgb = confusion_matrix(y, y_xgb_pred)
plt.figure(figsize=(4, 3))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Oranges', xticklabels=["Low", "Medium", "High"],
            yticklabels=["Low", "Medium", "High"])
plt.title('XGBoost Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.savefig(XGB_CM_PATH)
plt.close()

# XGBoost Feature Importance Bar Graph
feature_importances_xgb = pd.Series(xgb_classifier.feature_importances_, index=X.columns)
plt.figure(figsize=(5, 3))
feature_importances_xgb.sort_values().plot(kind='barh', color='coral')
plt.title('XGBoost Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.tight_layout()
plt.savefig(XGB_BAR_PATH)
plt.close()

# Render the graphs page with metrics and file paths for the graphs
return render_template(
    'graphs.html',
    rf_accuracy=rf_accuracy,
    xgb_accuracy=xgb_accuracy,
    rf_cm_url=url_for('static', filename='confusion_matrix_rf.png'),
    rf_bar_url=url_for('static', filename='bar_graph_rf.png'),
    xgb_cm_url=url_for('static', filename='confusion_matrix_xgb.png'),
    xgb_bar_url=url_for('static', filename='bar_graph_xgb.png')
)

```

```

except Exception as e:
    return f"Error generating graphs: {str(e)}"

# Route for prediction form and results
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        try:
            # Parse input data from the form
            latitude = float(request.form['latitude'])
            longitude = float(request.form['longitude'])
            depth = float(request.form['depth'])

            # Prepare input for prediction
            input_data = pd.DataFrame([[latitude, longitude, depth]], columns=['Latitude', 'Longitude', 'Depth'])
            scaled_data = scaler.transform(input_data)

            # Perform predictions
            predicted_magnitude = xgb_regressor.predict(scaled_data)[0] # Predict magnitude
            predicted_intensity_code = xgb_classifier.predict(scaled_data)[0]
            predicted_intensity = ['Low', 'Medium', 'High'][int(predicted_intensity_code)]

            # Render results
            return render_template(
                'prediction_result.html',
                magnitude=round(predicted_magnitude, 2),
                intensity=predicted_intensity
            )
        except Exception as e:
            return f"Error: {str(e)}"

    # Render the prediction form if request method is GET
    return render_template('prediction.html')

if __name__ == '__main__':
    app.run(debug=False, host='0.0.0.0', port=5000)

```

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import mean_squared_error, accuracy_score, confusion_matrix
import pickle
from xgboost import XGBRegressor, XGBClassifier

# Load the dataset
data = pd.read_csv('dataset.csv')

# Features (Latitude, Longitude, Depth) and target (Magnitude)
X = data[['Latitude', 'Longitude', 'Depth']]
y_magnitude = data['Magnitude'] # Target column for regression

# Categorize Magnitude into Intensity Levels for Classification
def categorize_intensity(magnitude):
    """
    Categorize magnitude into intensity levels based on updated thresholds.
    """
    if magnitude < 4.5: # Adjusted threshold for "Low"
        return "Low"
    elif 4.5 <= magnitude < 6.5: # Adjusted range for "Medium"
        return "Medium"
    else:
        return "High" # "High" for anything 6.5 and above

# Apply categorization
data['Intensity'] = data['Magnitude'].apply(categorize_intensity)
y_intensity = data['Intensity']

# Print label distribution for debugging and validation
print("Original Label Distribution:")
print(data['Intensity'].value_counts())

# Augment data for high-magnitude scenarios
high_magnitude_samples = pd.DataFrame({
    'Latitude': np.random.uniform(30, 50, 50),
    'Longitude': np.random.uniform(-130, -100, 50),
    'Depth': np.random.uniform(10, 50, 50)
})

```

```

# Combine augmented data with original dataset
augmented_data = pd.concat([data, high_magnitude_samples], ignore_index=True)

# Update features and target
X = augmented_data[['Latitude', 'Longitude', 'Depth']]
y_magnitude = augmented_data['Magnitude']
y_intensity = augmented_data['Intensity'].apply(categorize_intensity)

# Print updated label distribution for validation
print("Augmented Label Distribution:")
print(y_intensity.value_counts())

# Encode intensity levels for classification
y_intensity_encoded = pd.Categorical(y_intensity).codes

# Normalize features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Split data for regressor and classifier
X_train, X_test, y_magnitude_train, y_magnitude_test = train_test_split(
    X_scaled, y_magnitude, test_size=0.2, random_state=42
)
X_train_cls, X_test_cls, y_intensity_train, y_intensity_test = train_test_split(
    X_scaled, y_intensity_encoded, test_size=0.2, random_state=42
)

# Address class imbalance for the classifier
sample_weights = compute_sample_weight("balanced", y_intensity_train)

# Train XGBoost Regressor
xgb_regressor = XGBRegressor(n_estimators=200, learning_rate=0.1, max_depth=6, random_state=42)
xgb_regressor.fit(X_train, y_magnitude_train)

# Evaluate Regressor
y_magnitude_pred = xgb_regressor.predict(X_test)
mse = mean_squared_error(y_magnitude_test, y_magnitude_pred)
print(f"Mean Squared Error (Regressor): {mse:.2f}")

```



```
# Train XGBoost Regressor
xgb_regressor = XGBRegressor(n_estimators=200, learning_rate=0.1, max_depth=6, random_state=42)
xgb_regressor.fit(X_train, y_magnitude_train)

# Evaluate Regressor
y_magnitude_pred = xgb_regressor.predict(X_test)
mse = mean_squared_error(y_magnitude_test, y_magnitude_pred)
print(f"Mean Squared Error (Regressor): {mse:.2f}")

# Train XGBoost Classifier
xgb_classifier = XGBClassifier(n_estimators=200, learning_rate=0.1, max_depth=6, random_state=42)
xgb_classifier.fit(X_train_cls, y_intensity_train, sample_weight=sample_weights)

# Evaluate Classifier
y_intensity_pred = xgb_classifier.predict(X_test_cls)
accuracy = accuracy_score(y_intensity_test, y_intensity_pred)
print(f"Accuracy (Classifier): {accuracy * 100:.2f}%")

# Generate confusion matrix
conf_matrix = confusion_matrix(y_intensity_test, y_intensity_pred)
print("Confusion Matrix (Classifier):")
print(conf_matrix)

# Save models and scaler
with open('xgb_regressor.pkl', 'wb') as file:
    pickle.dump(xgb_regressor, file)

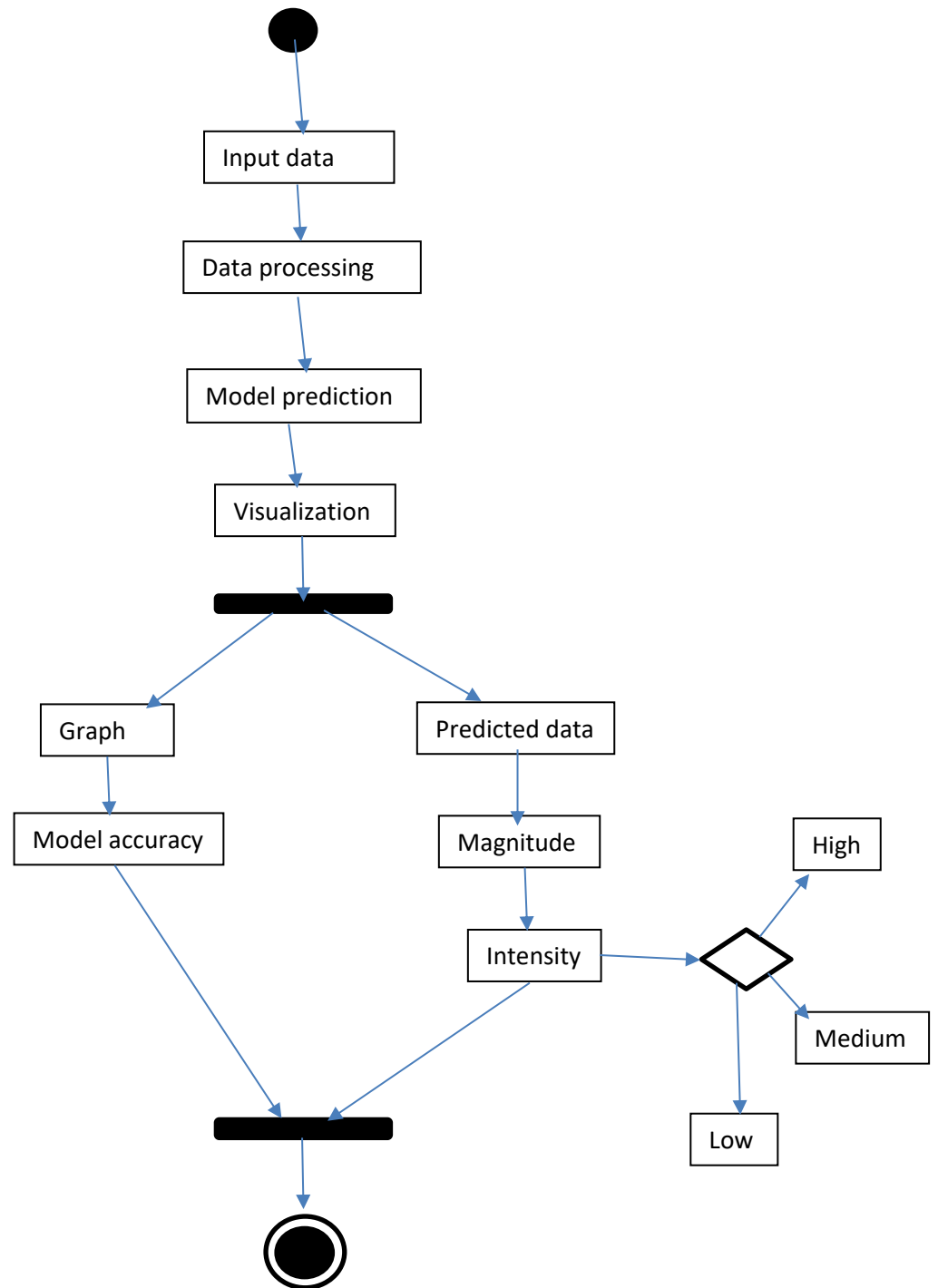
with open('xgb_classifier.pkl', 'wb') as file:
    pickle.dump(xgb_classifier, file)

with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)

print("Models and scaler saved successfully.")
```



## Architecture Diagram



## Conclusion

The **Earthquake Intensity Prediction System** is an innovative and comprehensive tool that merges machine learning prediction techniques with interactive data visualization. By predicting earthquake intensities and visualizing seismic activity globally, the system helps authorities and researchers understand and manage seismic risks. It offers valuable insights for disaster preparedness, earthquake research, and public education, ensuring that users are better equipped to handle the challenges posed by seismic events.

## References

1. [https://www.researchgate.net/publication/379711505\\_Forecasting\\_Earthquake\\_Using\\_Machine\\_Learning](https://www.researchgate.net/publication/379711505_Forecasting_Earthquake_Using_Machine_Learning)
2. [https://www.researchgate.net/publication/380073427\\_Earthquake\\_Magnitude\\_Prediction\\_Using\\_Machine\\_Learning\\_Techniques](https://www.researchgate.net/publication/380073427_Earthquake_Magnitude_Prediction_Using_Machine_Learning_Techniques)
3. <https://www.annualreviews.org/content/journals/10.1146/annurev-earth-071822-100323>
4. <https://www.sciencedirect.com/science/article/abs/pii/S2211675320300361>