Project Report

Data Science

Introduction

The aim of this project is to predict earthquake magnitudes using machine learning algorithms. Earthquakes are natural disasters that can cause significant damage and loss of life. By accurately predicting earthquake magnitudes, we can better prepare for and mitigate the impact of these events. In this project, we utilize data science techniques and machine learning models to predict earthquake magnitudes based on various features such as time, location, and type.

Problem Statement

The problem addressed in this project is the prediction of earthquake magnitudes. Given a dataset containing information about past earthquakes, including their location, time, and other relevant features, the goal is to build machine learning models that can accurately predict the magnitude of future earthquakes. This prediction can help in disaster preparedness and response efforts.

Scope

The scope of this project includes:
- ✓ **Data collection**: Obtaining earthquake data from reliable sources.
- ✓ **Data preprocessing**: Cleaning the data, handling missing values, and encoding categorical variables.
- ✓ **Feature engineering**: Extracting relevant features from the dataset.
- ✓ **Model selection**: Choosing appropriate machine learning algorithms for prediction.
- ✓ **Model evaluation**: Assessing the performance of the models using metrics such as RMSE, R^2, classification reports, and accuracy scores.
- ✓ **Visualization**: Visualizing the data and model predictions using plots and graphs.
- ✓ **User interface development**: Creating a graphical user interface (GUI) for users to interact with the prediction system.

Objective

The main objective of this project is to develop a predictive model for earthquake magnitudes using machine learning techniques. Specifically, the goals are to:
- ✓ Preprocess the earthquake dataset to make it suitable for modeling.
- ✓ Train multiple machine learning models on the preprocessed data.
- ✓ Evaluate the performance of the models using appropriate metrics.
- ✓ Visualize the data and model predictions.
- ✓ Develop a user-friendly interface for predicting earthquake magnitudes.

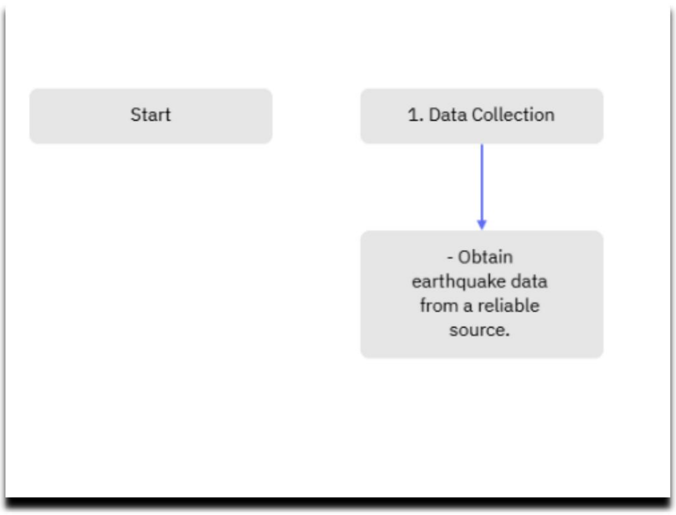
Methodology

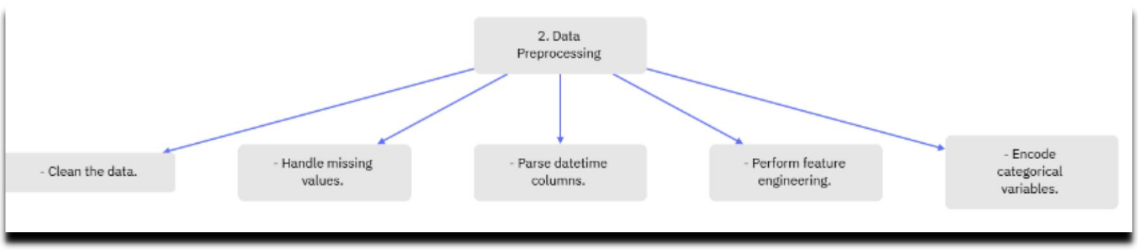
The methodology adopted for this project involves the following steps:

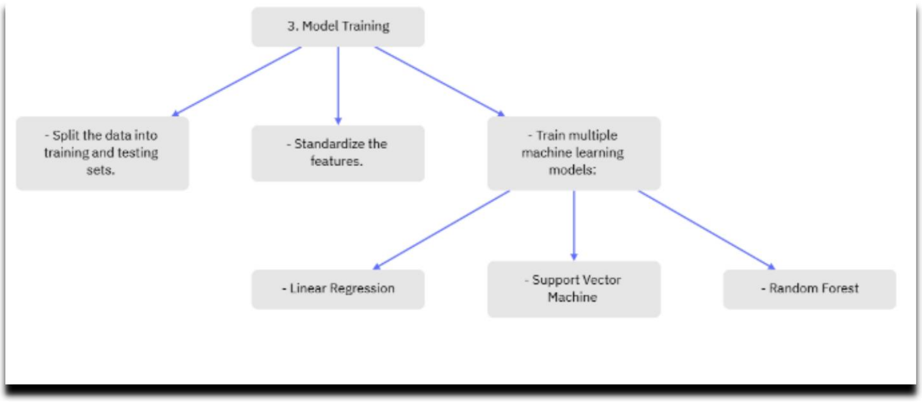
- ✓ Data collection: Obtain earthquake data from a reliable source.
- ✓ Data preprocessing: Clean the data, handle missing values, parse datetime columns, perform feature engineering, and encode categorical variables.
- ✓ Model training: Split the data into training and testing sets, standardize the features, and train multiple machine learning models such as Linear Regression, Support Vector Machine, and Random Forest.
- ✓ Model evaluation: Evaluate the performance of the models using metrics such as RMSE, R^2, classification reports, accuracy scores, and confusion matrices.
- ✓ Visualization: Visualize the data, actual vs. predicted magnitudes, residuals, feature importance, scatter plots, residual analysis, and prediction intervals.
- ✓ User interface development: Develop a GUI using tkinter to allow users to upload a dataset, preprocess the data, and make predictions using the trained models.

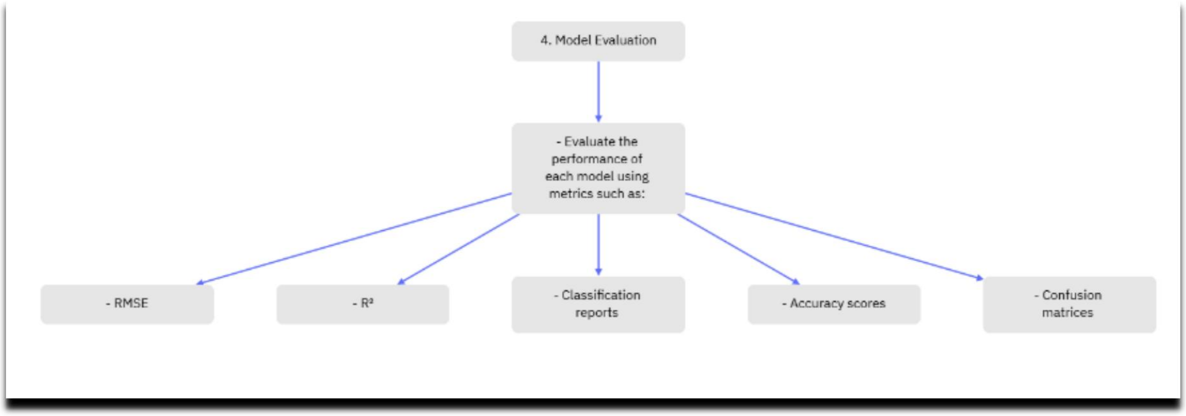
Flowchart

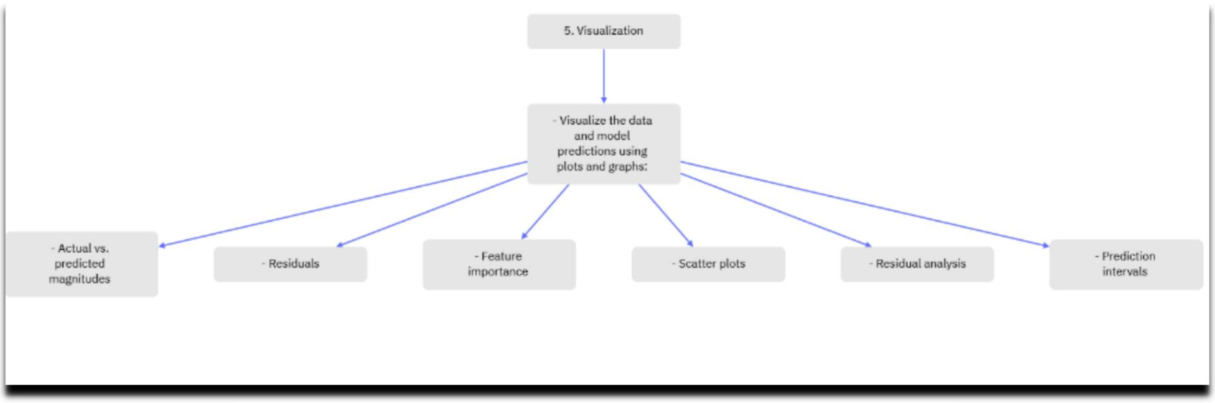
The flowchart illustrating the workflow of the project is as follows:

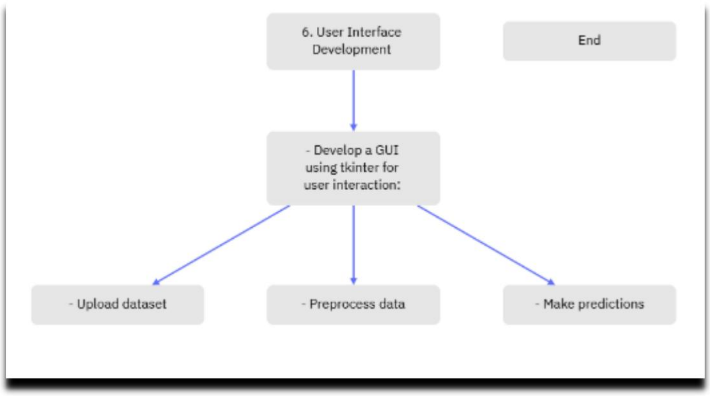












Code

The Python code used for data preprocessing, model training, evaluation, visualization, and GUI development is provided below.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, classification_report,
accuracy_score, confusion_matrix
import joblib
import matplotlib.pyplot as plt


# Load the dataset
data = pd.read_csv(r'C:\Users\hp\Downloads\database.csv')

# Check the first few rows to understand the structure and formats
print("Initial Data Head:\n", data.head())

# Parse Date and Time columns with the correct format
data['Datetime'] = pd.to_datetime(data['Date'] + ' ' + data['Time'],
format='%m/%d/%Y %H:%M:%S', errors='coerce')

# Drop rows with invalid datetime parsing
data.dropna(subset=['Datetime'], inplace=True)

# Check the data after parsing the datetime
print("Data after parsing Datetime:\n", data.head())
print("Number of valid rows:", len(data))

# Feature engineering
data['Year'] = data['Datetime'].dt.year
data['Month'] = data['Datetime'].dt.month
data['Day'] = data['Datetime'].dt.day
data['Hour'] = data['Datetime'].dt.hour

# Drop original Date and Time columns
data.drop(columns=['Date', 'Time', 'Datetime'], inplace=True)

# Handle missing values (impute with mean for simplicity)
data.fillna(data.mean(numeric_only=True), inplace=True)

# Encode categorical variables
le = LabelEncoder()
data['Type'] = le.fit_transform(data['Type'])
data['Magnitude Type'] = le.fit_transform(data['Magnitude Type'])

# Define features and target variable
X = data.drop(columns=['Magnitude'])
y = data['Magnitude']

# Check for empty datasets
```

```
print("Features head:\n", X.head())
print("Target head:\n", y.head())
print("Number of samples:", len(X))

# Split data into training and testing sets
if len(X) > 0:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
else:
    print("Error: The dataset is empty after preprocessing.")

# Continue if data is not empty
if len(X) > 0:
    # Standardize the features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Multiple Linear Regression
    lr = LinearRegression()
    lr.fit(X_train_scaled, y_train)
    y_pred_lr = lr.predict(X_test_scaled)

    # Support Vector Machine
    svm = SVR()
    svm.fit(X_train_scaled, y_train)
    y_pred_svm = svm.predict(X_test_scaled)

    # Random Forest
    rf = RandomForestRegressor()
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)

    # Evaluate models
    def evaluate_model(y_true, y_pred, model_name):
        rmse = np.sqrt(mean_squared_error(y_true, y_pred))
        r2 = r2_score(y_true, y_pred)
        print(f"{model_name} - RMSE: {rmse:.2f}, R²: {r2:.2f}")

    evaluate_model(y_test, y_pred_lr, "Multiple Linear Regression")
    evaluate_model(y_test, y_pred_svm, "Support Vector Machine")
    evaluate_model(y_test, y_pred_rf, "Random Forest")

    # Function to discretize magnitudes into classes
    def discretize_magnitudes(magnitudes):
        bins = [0, 4, 6, 8, 10]
        labels = ['Low', 'Medium', 'High', 'Very High']
        return pd.cut(magnitudes, bins=bins, labels=labels, right=False)

    # Discretize actual magnitudes
    y_test_class = discretize_magnitudes(y_test)

    # Discretize predicted magnitudes
```

```
    y_pred_rf_class = discretize_magnitudes(y_pred_rf)
    y_pred_lr_class = discretize_magnitudes(y_pred_lr)
    y_pred_svm_class = discretize_magnitudes(y_pred_svm)

    # Classification Report and Accuracy for Random Forest
    print("Random Forest Classification Report:")
    print(classification_report(y_test_class, y_pred_rf_class))
    print("Random Forest Accuracy:", accuracy_score(y_test_class, y_pred_rf_class))

    # Confusion Matrix for Random Forest
    print("Random Forest Confusion Matrix:")
    print(confusion_matrix(y_test_class, y_pred_rf_class))

    # Classification Report and Accuracy for Linear Regression
    print("\nMultiple Linear Regression Classification Report:")
    print(classification_report(y_test_class, y_pred_lr_class))
    print("Multiple Linear Regression Accuracy:", accuracy_score(y_test_class,
y_pred_lr_class))

    # Confusion Matrix for Linear Regression
    print("Multiple Linear Regression Confusion Matrix:")
    print(confusion_matrix(y_test_class, y_pred_lr_class))

    # Classification Report and Accuracy for Support Vector Machine
    print("\nSupport Vector Machine Classification Report:")
    print(classification_report(y_test_class, y_pred_svm_class))
    print("Support Vector Machine Accuracy:", accuracy_score(y_test_class,
y_pred_svm_class))

    # Confusion Matrix for Support Vector Machine
    print("Support Vector Machine Confusion Matrix:")
    print(confusion_matrix(y_test_class, y_pred_svm_class))

    # Plotting
    import matplotlib.pyplot as plt

    # Actual vs Predicted plot
    plt.figure(figsize=(14, 7))
    plt.plot(y_test.values, label='Actual')
    plt.plot(y_pred_lr, label='Predicted - LR')
    plt.plot(y_pred_svm, label='Predicted - SVM')
    plt.plot(y_pred_rf, label='Predicted - RF')
    plt.legend()
    plt.title('Actual vs Predicted')
    plt.show()

    # Residual plots
    plt.figure(figsize=(14, 7))
    plt.scatter(y_pred_lr, y_test - y_pred_lr, label='LR Residuals')
    plt.scatter(y_pred_svm, y_test - y_pred_svm, label='SVM Residuals')
    plt.scatter(y_pred_rf, y_test - y_pred_rf, label='RF Residuals')
    plt.axhline(y=0, color='r', linestyle='--')
```

```
plt.legend()
plt.title('Residuals Plot')
plt.show()

# Feature Importance Plot
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10, 5))
plt.title("Feature Importance")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
plt.tight_layout()
plt.show()

# Scatter Plot
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title('Actual vs Predicted Magnitude')
plt.show()

# Residual Analysis Plot
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_rf, y_test - y_pred_rf, alpha=0.5)
plt.axhline(0, color='r', linestyle='--')
plt.xlabel('Predicted Magnitude')
plt.ylabel('Residuals')
plt.title('Residual Analysis Plot')
plt.show()

# Prediction Interval Plot
pred_std = np.std(y_test - y_pred_rf)
plt.figure(figsize=(10, 6))
plt.plot(y_test.values, 'b-', label='Actual Magnitude')
plt.plot(y_pred_rf, 'r-', label='Predicted Magnitude')
plt.fill_between(range(len(y_pred_rf)), y_pred_rf - 1.96 * pred_std, y_pred_rf + 1.96 *
pred_std, color='gray', alpha=0.2, label='95% Prediction Interval')
plt.xlabel('Index')
plt.ylabel('Magnitude')
plt.title('Prediction Interval Plot')
plt.legend()
plt.show()

def plot_cap_curve(y_true, y_pred, model_name):
    # Reset index to ensure alignment
    y_true = y_true.reset_index(drop=True)
    y_pred = pd.Series(y_pred).reset_index(drop=True)

    total = len(y_true)
    total_positive = np.sum(y_true)
    total_negative = total - total_positive
```

```
        sorted_indices = np.argsort(y_pred)
        sorted_true = y_true[sorted_indices]

        cum_positive_rate = np.cumsum(sorted_true) / total_positive
        cum_total_rate = np.arange(1, total + 1) / total

        plt.plot(cum_total_rate, cum_positive_rate, label=f'{model_name} Model')

        plt.xlabel('Proportion of Data')
        plt.ylabel('Proportion of Positive Predictions')
        plt.title('Cumulative Accuracy Profile (CAP) Curve')
        plt.legend()


    plt.figure(figsize=(10, 6))
    plot_cap_curve(y_test, y_pred_rf, "Random Forest")
    plot_cap_curve(y_test, y_pred_lr, "Logistic Regression")
    plot_cap_curve(y_test, y_pred_svm, "Support Vector Machine")
    plt.plot([0, 1], [0, 1], 'k--', label='Random Model')
    plt.legend()
    plt.show()


else:
    print("Error: The dataset is empty after preprocessing. No data to train the models.")
```

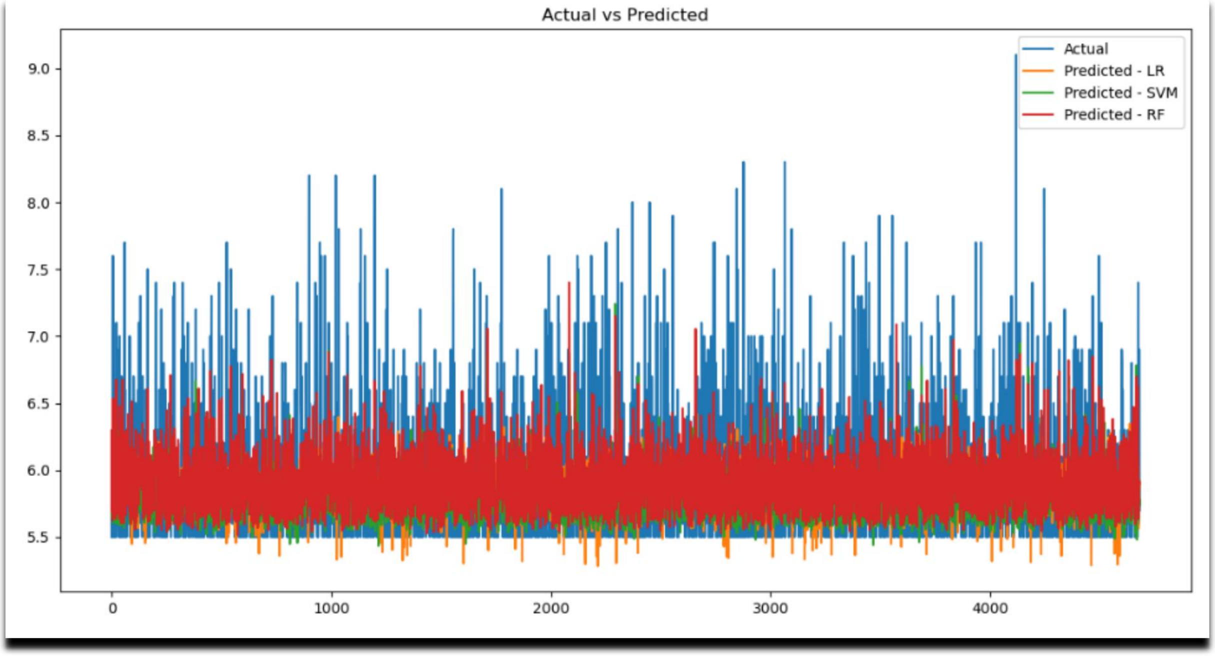
```
Initial Data Head:
        Date      Time  Latitude  Longitude        Type  Depth  Depth Error  \
0  1/2/1965  13:44:18    19.246    145.616  Earthquake  131.6          NaN
1  1/4/1965  11:29:49     1.863    127.352  Earthquake   80.0          NaN
2  1/5/1965  18:05:58   -20.579   -173.972  Earthquake   20.0          NaN
3  1/8/1965  18:49:43   -59.076    -23.557  Earthquake   15.0          NaN
4  1/9/1965  13:32:50    11.938    126.427  Earthquake   15.0          NaN

   Depth Seismic Stations  Magnitude Magnitude Type
0                     NaN        6.0             MW
1                     NaN        5.8             MW
2                     NaN        6.2             MW
3                     NaN        5.8             MW
4                     NaN        5.8             MW
Data after parsing Datetime:
        Date      Time  Latitude  Longitude        Type  Depth  Depth Error  \
0  1/2/1965  13:44:18    19.246    145.616  Earthquake  131.6          NaN
1  1/4/1965  11:29:49     1.863    127.352  Earthquake   80.0          NaN
2  1/5/1965  18:05:58   -20.579   -173.972  Earthquake   20.0          NaN
3  1/8/1965  18:49:43   -59.076    -23.557  Earthquake   15.0          NaN
4  1/9/1965  13:32:50    11.938    126.427  Earthquake   15.0          NaN

   Depth Seismic Stations  Magnitude Magnitude Type            Datetime
0                     NaN        6.0             MW 1965-01-02 13:44:18
1                     NaN        5.8             MW 1965-01-04 11:29:49
2                     NaN        6.2             MW 1965-01-05 18:05:58
3                     NaN        5.8             MW 1965-01-08 18:49:43
4                     NaN        5.8             MW 1965-01-09 13:32:50
Number of valid rows: 23409
```

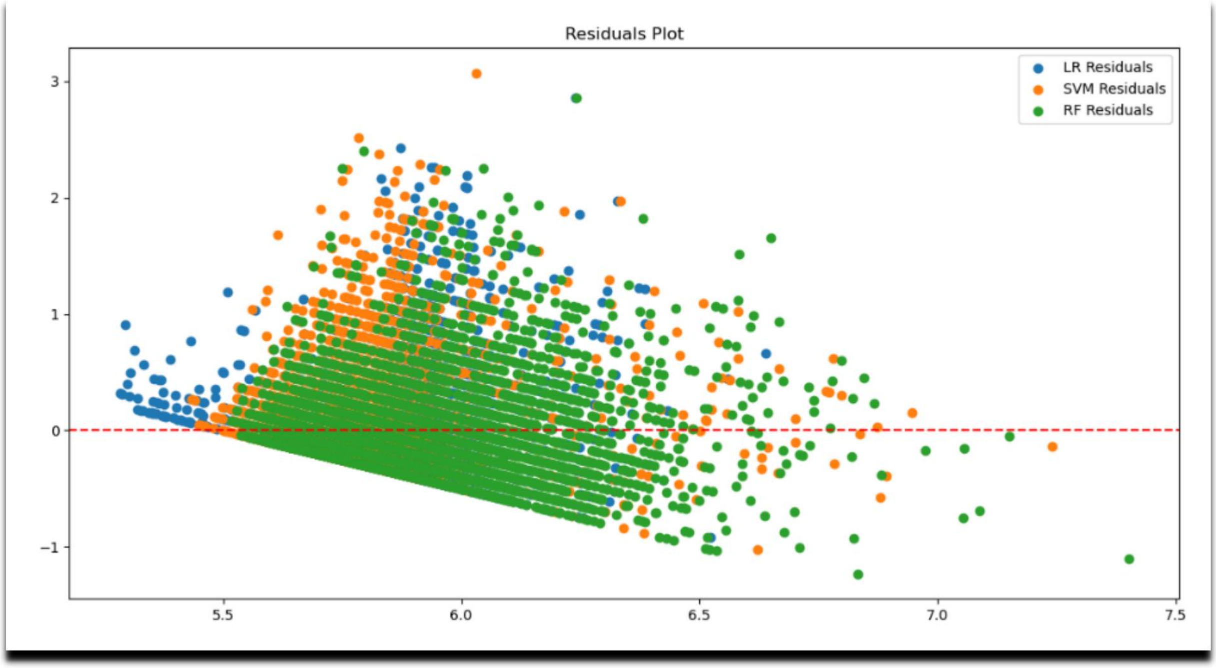
```
Features head:
    Latitude  Longitude  Type  Depth  Depth Error  Depth Seismic Stations  \
0     19.246    145.616     0  131.6     4.991118              275.362176
1      1.863    127.352     0   80.0     4.991118              275.362176
2    -20.579   -173.972     0   20.0     4.991118              275.362176
3    -59.076    -23.557     0   15.0     4.991118              275.362176
4     11.938    126.427     0   15.0     4.991118              275.362176

   Magnitude Type  Year  Month  Day  Hour
0               5  1965      1    2    13
1               5  1965      1    4    11
2               5  1965      1    5    18
3               5  1965      1    8    18
4               5  1965      1    9    13
Target head:
 0    6.0
1    5.8
2    6.2
3    5.8
4    5.8
Name: Magnitude, dtype: float64
Number of samples: 23409
Multiple Linear Regression - RMSE: 0.41, R²: 0.11
Support Vector Machine - RMSE: 0.40, R²: 0.13
Random Forest - RMSE: 0.38, R²: 0.20
Random Forest Classification Report:
```

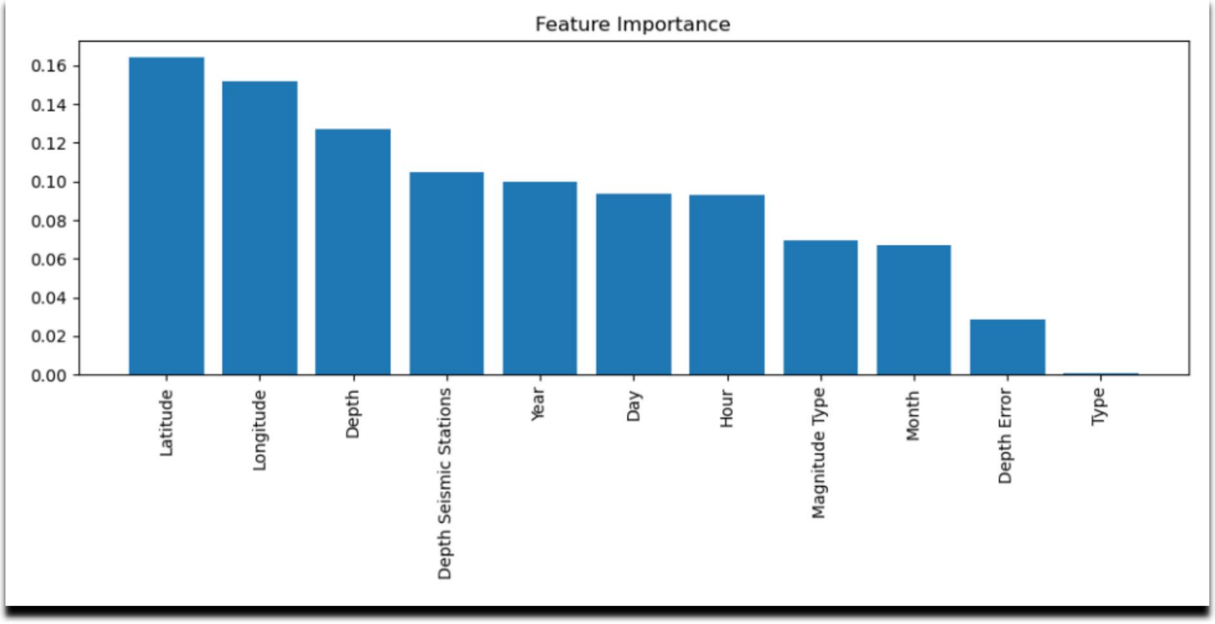
```
              precision    recall  f1-score   support

        High       0.54      0.52      0.53      1433
      Medium       0.79      0.80      0.80      3238
   Very High       0.00      0.00      0.00        11

    accuracy                           0.71      4682
   macro avg       0.44      0.44      0.44      4682
weighted avg       0.71      0.71      0.71      4682

Random Forest Accuracy: 0.714438274241777
Random Forest Confusion Matrix:
[[ 741  692    0]
 [ 634 2604    0]
 [   8    3    0]]

Multiple Linear Regression Classification Report:
```

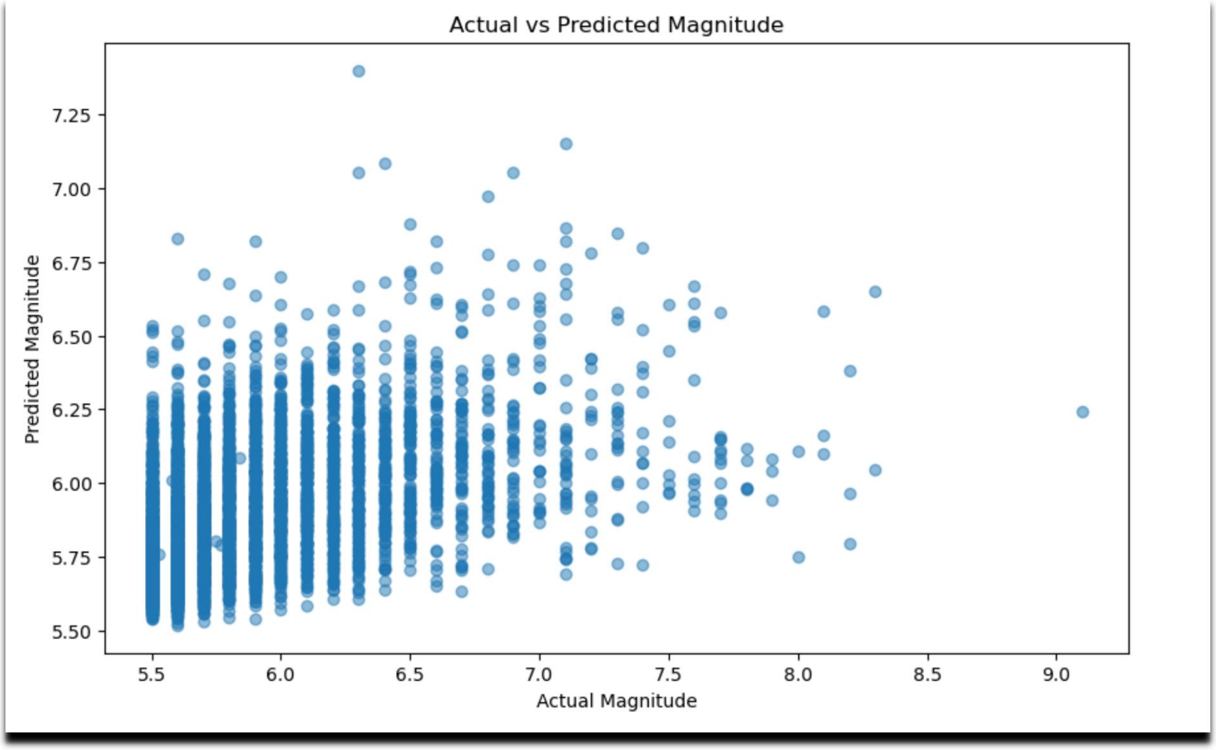
```
              precision    recall  f1-score   support

        High       0.50      0.22      0.31      1433
      Medium       0.72      0.90      0.80      3238
   Very High       0.00      0.00      0.00        11

    accuracy                           0.69      4682
   macro avg       0.41      0.37      0.37      4682
weighted avg       0.65      0.69      0.65      4682

Multiple Linear Regression Accuracy: 0.6928662964545066
Multiple Linear Regression Confusion Matrix:
[[ 315 1118    0]
 [ 309 2929    0]
 [   6    5    0]]

Support Vector Machine Classification Report:
```

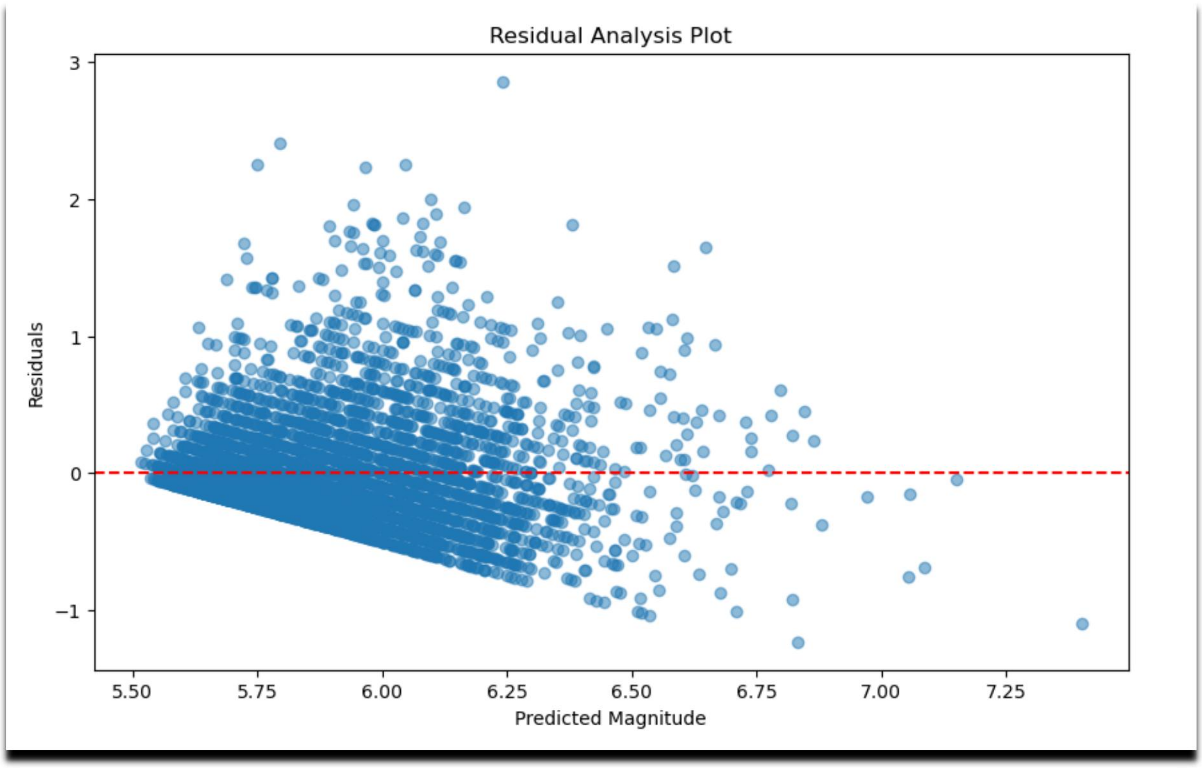
```
              precision    recall  f1-score   support

        High       0.68      0.14      0.24      1433
      Medium       0.72      0.97      0.83      3238
   Very High       0.00      0.00      0.00        11

    accuracy                           0.72      4682
   macro avg       0.47      0.37      0.35      4682
weighted avg       0.70      0.72      0.64      4682

Support Vector Machine Accuracy: 0.7150790260572405
Support Vector Machine Confusion Matrix:
[[ 204 1229    0]
 [  94 3144    0]
 [   3    8    0]]
```

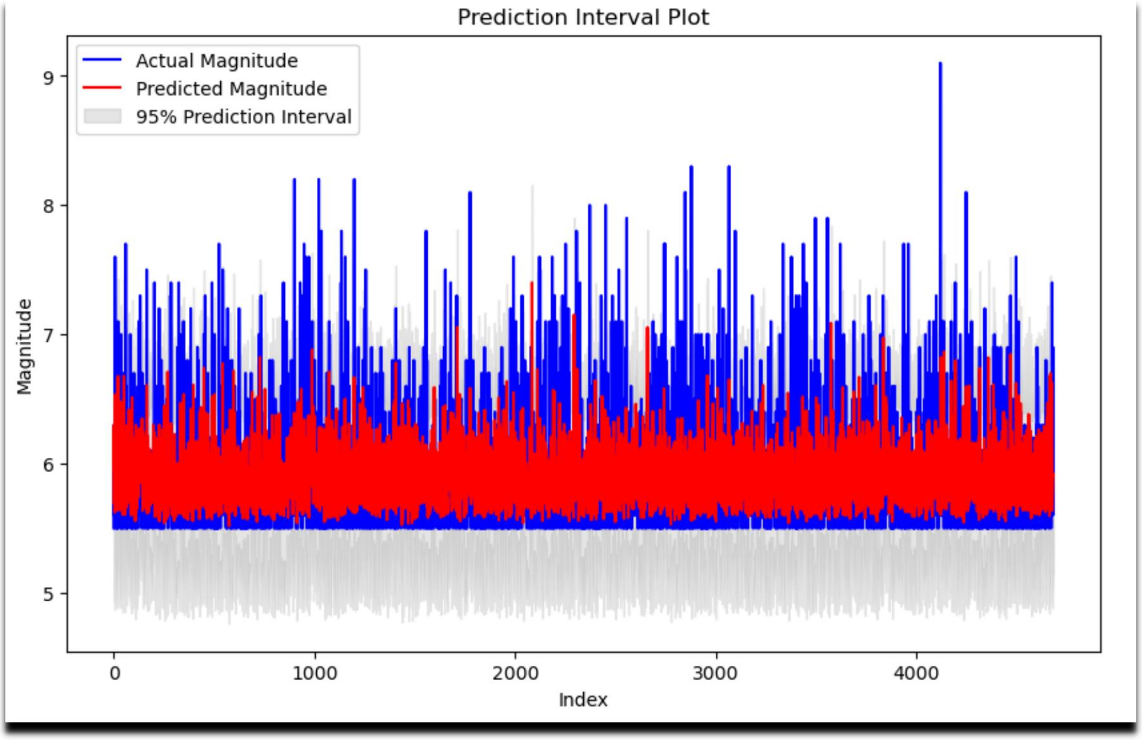


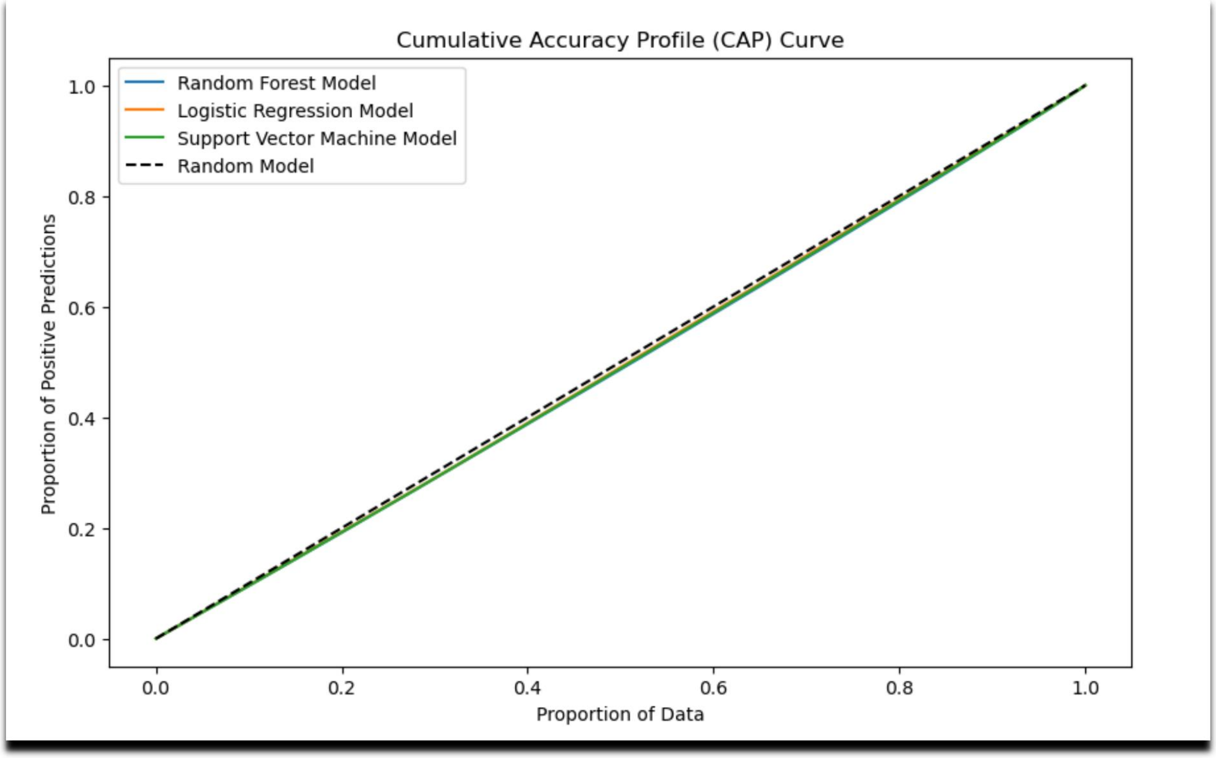



Feature Importance
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Latitude
Longitude
Depth
Depth Seismic Stations
Year
Day
Hour
Magnitude Type
Month
Depth Error
Type


Actual vs Predicted Magnitude
Predicted Magnitude
7.25
7.00
6.75
6.50
6.25
6.00
5.75
5.50
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
Actual Magnitude


Residual Analysis Plot
Residuals
Predicted Magnitude
3
2
1
0
−1
5.50
5.75
6.00
6.25
6.50
6.75
7.00
7.25


Prediction Interval Plot
Actual Magnitude
Predicted Magnitude
95% Prediction Interval
Magnitude
Index
9
8
7
6
5
0
1000
2000
3000
4000

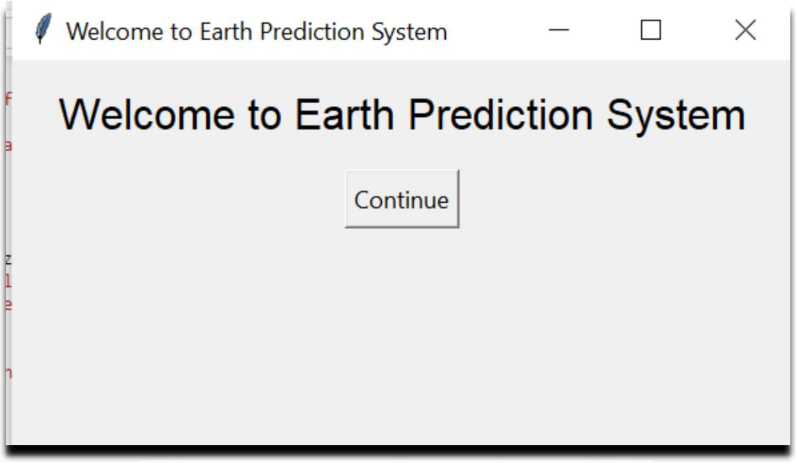


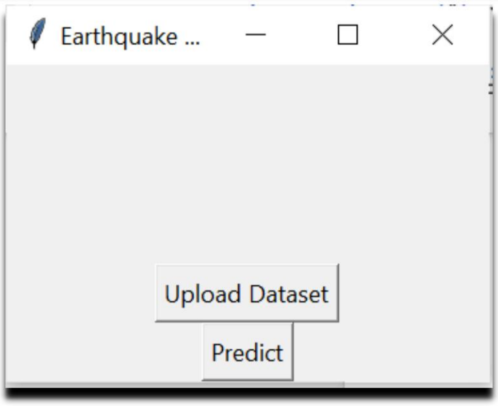

```
import tkinter as tk
from tkinter import filedialog, messagebox, Toplevel
import folium
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

class EarthquakePredictionUI:
    def __init__(self, master):
        self.master = master
        self.master.title("Welcome to Earth Prediction System")

        self.title_label = tk.Label(self.master, text="Welcome to Earth Prediction System",
font=("Helvetica", 16))
        self.title_label.pack(pady=20)

        self.continue_button = tk.Button(self.master, text="Continue",
command=self.show_prediction_page)
        self.continue_button.pack()

    def show_prediction_page(self):
        self.master.destroy()
        prediction_window = tk.Tk()
        PredictionPage(prediction_window)
        prediction_window.mainloop()

class PredictionPage:
    def __init__(self, master):
```

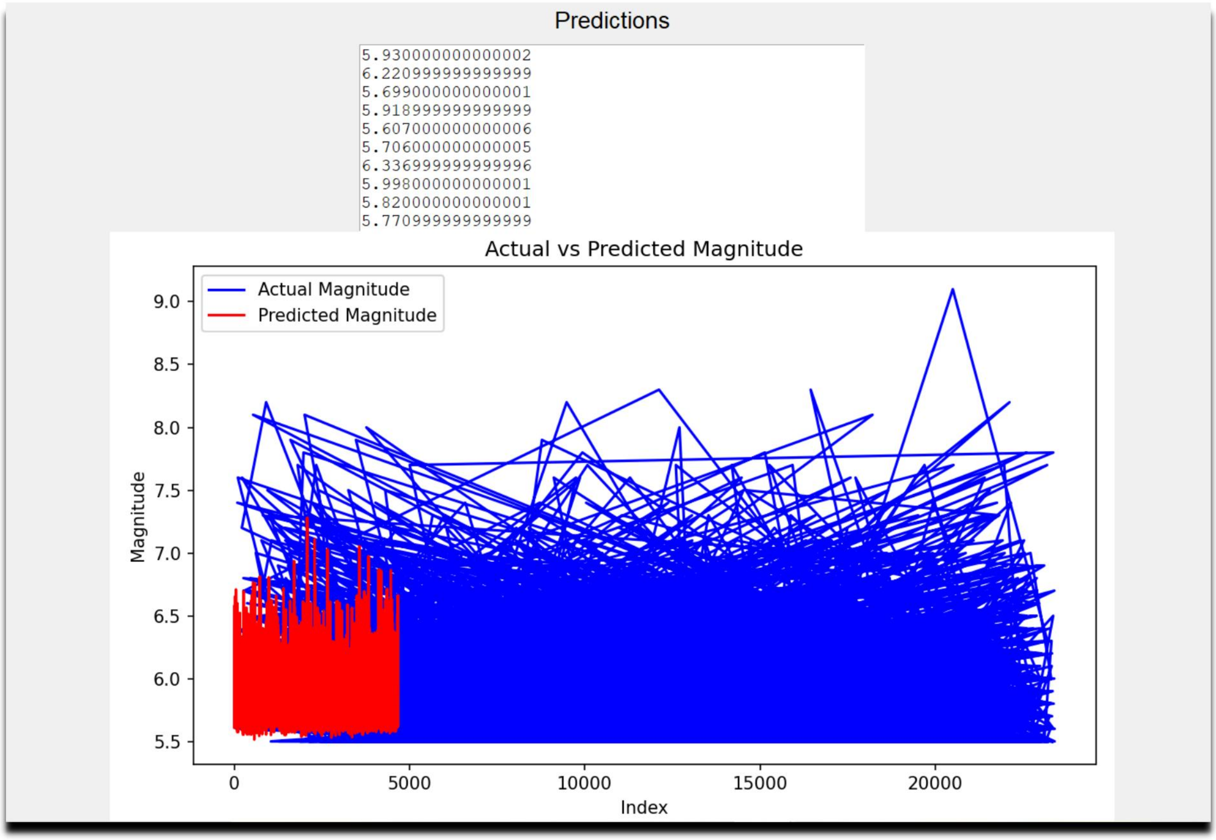
```
        self.master = master
        self.master.title("Earthquake Prediction")

        self.map_frame = tk.Frame(self.master)
        self.map_frame.pack(side="top", fill="both", expand=True)

        self.map = folium.Map(location=[0, 0], zoom_start=2)
        self.map.save("map.html")
        self.map_view = tk.Label(self.map_frame)
        self.map_view.pack()

        self.upload_button = tk.Button(self.master, text="Upload Dataset",
command=self.load_dataset)
        self.upload_button.pack()

        self.predict_button = tk.Button(self.master, text="Predict", command=self.predict)
        self.predict_button.pack()

    def load_dataset(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            try:
                self.data = pd.read_csv(file_path)
                messagebox.showinfo("Success", "Dataset loaded successfully!")
            except Exception as e:
                messagebox.showerror("Error", f"Error loading dataset: {str(e)}")

    def preprocess_data(self):
        if hasattr(self, 'data'):
            try:
                self.data['Datetime'] = pd.to_datetime(self.data['Date'] + ' ' + self.data['Time'],
format='%m/%d/%Y %H:%M:%S', errors='coerce')
                self.data.dropna(subset=['Datetime'], inplace=True)
                self.data['Year'] = self.data['Datetime'].dt.year
                self.data['Month'] = self.data['Datetime'].dt.month
                self.data['Day'] = self.data['Datetime'].dt.day
                self.data['Hour'] = self.data['Datetime'].dt.hour
                self.data.drop(columns=['Date', 'Time', 'Datetime'], inplace=True)
                self.data.fillna(self.data.mean(numeric_only=True), inplace=True)
                le = LabelEncoder()
                self.data['Type'] = le.fit_transform(self.data['Type'])
                self.data['Magnitude Type'] = le.fit_transform(self.data['Magnitude Type'])
                self.X = self.data.drop(columns=['Magnitude'])
                self.y = self.data['Magnitude']
                self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.X, self.y,
test_size=0.2, random_state=42)
                self.scaler = StandardScaler()
                self.X_train_scaled = self.scaler.fit_transform(self.X_train)
                self.X_test_scaled = self.scaler.transform(self.X_test)
                messagebox.showinfo("Success", "Data preprocessed successfully!")
            except Exception as e:
                messagebox.showerror("Error", f"Error preprocessing data: {str(e)}")
```

```
        else:
            messagebox.showerror("Error", "No dataset loaded!")

    def predict(self):
        self.preprocess_data()
        if hasattr(self, 'X_test_scaled'):
            try:
                rf = RandomForestRegressor()
                rf.fit(self.X_train_scaled, self.y_train)
                y_pred_rf = rf.predict(self.X_test_scaled)

                # Create a new window for displaying predictions and plots
                prediction_window = Toplevel(self.master)
                prediction_window.title("Predictions and Plots")

                # Display predictions
                predictions_label = tk.Label(prediction_window, text="Predictions",
font=("Helvetica", 14))
                predictions_label.pack(pady=10)

                # Show predictions in a text box
                predictions_text = tk.Text(prediction_window, height=10, width=50)
                for pred in y_pred_rf:
                    predictions_text.insert(tk.END, f"{pred}\n")
                predictions_text.pack()

                # Display plots
                self.display_plots(prediction_window, y_pred_rf)

            except Exception as e:
                messagebox.showerror("Error", f"Error predicting: {str(e)}")
        else:
            messagebox.showerror("Error", "Data preprocessing failed!")

    def display_plots(self, prediction_window, y_pred_rf):
        try:
            # Plot actual vs predicted
            fig_actual_pred = plt.figure(figsize=(8, 6))
            plt.plot(self.y_test, label='Actual Magnitude', color='blue')
            plt.plot(y_pred_rf, label='Predicted Magnitude', color='red')
            plt.xlabel('Index')
            plt.ylabel('Magnitude')
            plt.title('Actual vs Predicted Magnitude')
            plt.legend()
            plt.tight_layout()

            # Embed the plot in the Tkinter window
            canvas = FigureCanvasTkAgg(fig_actual_pred, master=prediction_window)
            canvas.draw()
            canvas.get_tk_widget().pack()

            # Button to return to main window
```

```
            return_button = tk.Button(prediction_window, text="Back to Main Window",
command=prediction_window.destroy)
            return_button.pack(pady=10)

        except Exception as e:
            messagebox.showerror("Error", f"Error displaying plots: {str(e)}")

if __name__ == "__main__":
    root = tk.Tk()
    EarthquakePredictionUI(root)
    root.mainloop()
```







Future Work

Some potential areas for future work include:

- ✓ Fine-tuning model hyperparameters to improve performance.
- ✓ Exploring additional features or data sources for better prediction accuracy.
- ✓ Enhancing the GUI with more features and functionalities.
- ✓ Deploying the prediction system as a web application or mobile app for wider accessibility.

Conclusion

In conclusion, this project demonstrates the application of data science techniques and machine learning models for earthquake magnitude prediction. Multiple models were trained and evaluated, and their performances were compared using various metrics. The developed GUI provides a user-friendly interface for predicting earthquake magnitudes based on user-provided data.