📚 Detailed Project Documentation

1. Project Overview

The Earthquake Prediction project leverages **machine learning techniques** to predict the **magnitude of earthquakes** based on seismic data. Using a **Random Forest Regressor**, this project demonstrates the use of **supervised learning** for regression tasks in the context of geophysical data analysis. The goal of this project is to predict the magnitude of future earthquakes based on historical data, which can assist in preparedness efforts for regions prone to seismic activity.

This project consists of multiple components:

1. **Data Collection and Preprocessing**: Collection of raw seismic data, followed by preprocessing to clean and normalize the data for training.
2. **Model Training**: Training a machine learning model (Random Forest Regressor) to predict earthquake magnitude.
3. **Model Evaluation**: Using **Mean Squared Error (MSE)** and **R^2 score** to evaluate the model's prediction accuracy.
4. **Model Deployment**: Deploying the trained model as an API via **FastAPI** and providing a user-friendly interface with **Streamlit** for easy interaction.

The primary output is the **predicted magnitude** of future earthquakes based on input features such as **depth, latitude, longitude**, and **number of reporting stations**.

2. Data Source & Description

The dataset used for this project comes from publicly available earthquake datasets. A typical dataset might contain the following features:

- **Source**: Data obtained from reliable earthquake monitoring agencies such as the **United States Geological Survey (USGS)** or public datasets from **Kaggle**.
- **File Format**: The dataset is provided in CSV format, which is ideal for easy manipulation and analysis in Python.

Key Features of the Dataset:

- **depth** (float): Depth of the earthquake in kilometers.
- **latitude** (float): Latitude where the earthquake occurred.
- **longitude** (float): Longitude where the earthquake occurred.
- **nst** (integer): The number of reporting stations that recorded the earthquake.
- **mag** (float): The magnitude of the earthquake, which is the target variable we aim to predict.

3. Problem Definition

The main goal of this project is to **predict earthquake magnitudes** based on historical data using a **Random Forest Regressor** model. This is a **regression** problem because the target variable (magnitude) is continuous.

4. Data Preprocessing

The quality of input data plays a critical role in the performance of machine learning models. The preprocessing steps carried out include:

1. **Handling Missing Values**: If there are any missing values in the dataset, we handle them by either:
    - Imputing missing values using techniques like **mean imputation** (for numerical data).
    - Removing rows or columns with excessive missing data if they can't be reasonably filled.
2. **Feature Scaling**: Features such as depth, latitude, and longitude are numerical, and it's important to **standardize** these variables to ensure that they are on the same scale. This is done using **StandardScaler**, which normalizes data to have a mean of 0 and a standard deviation of 1.
3. **Encoding Categorical Features**: If there are categorical variables (not in this case, but for generality), they are encoded into numeric representations using techniques such as **One-Hot Encoding** or **Label Encoding**.
4. **Splitting the Dataset**: The data is divided into two sets:
    - **Training Set** (80%): Used to train the model.
    - **Testing Set** (20%): Used to evaluate the model's performance.
5. **Outlier Detection**: We analyze outliers in numerical data and either remove or transform them to avoid negatively impacting the model's performance.

5. Model Training

The model training process involves the following steps:

1. **Choosing the Model**: We use a **Random Forest Regressor**, which is an ensemble model consisting of multiple decision trees. Random Forest is a powerful model for regression tasks because:
    - It handles both linear and non-linear relationships well.
    - It is robust to overfitting, especially when there are many features.
    - It can provide feature importance, which helps understand the relationship between features and the target variable.
2. **Training the Model**:
    - We train the **Random Forest model** on the training dataset using the `fit()` method.
    - The **hyperparameters** of the model (like the number of trees in the forest) are tuned to get the best possible performance.
3. **Cross-Validation**: We use **cross-validation** to ensure that the model generalizes well to unseen data. This helps avoid overfitting and provides a better estimate of model performance.
4. **Hyperparameter Tuning**: Parameters such as the **number of trees** (`n_estimators`) and **maximum depth of trees** are tuned using grid search or random search to find the optimal combination for our model.

6. Model Evaluation

Once the model is trained, it is evaluated using the **test data**:

1. **Mean Squared Error (MSE)**: This metric computes the average squared difference between actual and predicted values. A lower MSE indicates better model performance.

   Formula:

   $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

   Where y_i is the true value and $\hat{y}_i$ is the predicted value.

2. **R² Score**: The R^2 score measures how well the model explains the variability in the target variable. The value of R^2 lies between 0 and 1, where 1 means perfect predictions and 0 means the model does not explain the data well.

   Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

   Where $\bar{y}_i$ is the mean of actual values.

3. **Model Comparison**: We compare the performance of our trained model against a baseline model (such as a simple mean prediction) to assess the model's value.

7. Model Deployment

Once the model is trained and evaluated, it is deployed for making predictions in a production setting. The deployment includes:

1. **FastAPI**:
   - A REST API is created using **FastAPI** to expose the trained model. The API accepts input data (depth, latitude, longitude, number of stations) and returns the predicted earthquake magnitude.
   - The API is served locally or can be hosted on cloud services.
2. **Streamlit App**:
   - A **Streamlit** app is developed for users to input data through an intuitive interface. The app takes user input (such as depth, latitude, and longitude) and displays the predicted magnitude in real time.