Final Submission

INTRODUCTION:

Earthquake is phenomenon which happens due to the movement of tectonic plates. Since this is a natural event we can’t foretell the happening of earthquake. But by measuring some factors we can predict the earthquake as much as possible.

DATASET:

We have given the dataset which has details of city, date, latitude, longitude, time. With these details we can analyses why earthquake happened in these areas

Also we have the depth and the magnitude of the earthquake. So we can understand the intensity of the earthquake with these measurements.

Dataset link: **https://www.kaggle.com/datasets/usgs/earthquake-database**

NEED:

An earthquake prediction model is a valuable tool for several reasons:

1. Early Warning: One of the primary purposes of earthquake prediction models is to provide early warning to communities and authorities. Predicting earthquakes, even with a limited lead time, can save lives and reduce property damage by allowing people to take precautionary measures, such as evacuating buildings or moving to safer locations.

2. Mitigation and Preparedness: Knowing where and when earthquakes are likely to occur enables governments and communities to prepare better. This includes strengthening infrastructure, implementing building codes and construction standards that can withstand earthquakes, and establishing emergency response plans.

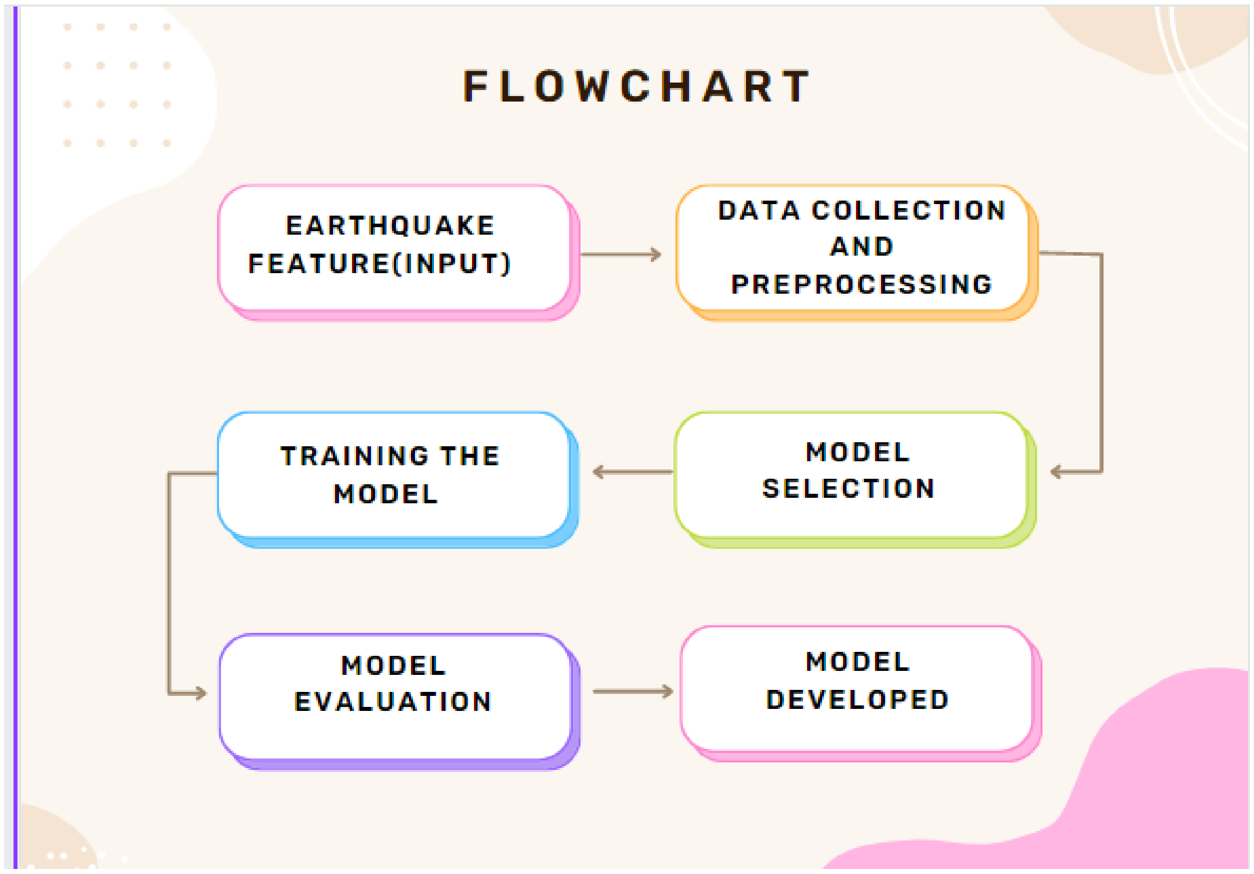
3. Resource Allocation: Earthquake prediction can help allocate resources more efficiently. For example, emergency services can be prepositioned in areas prone to earthquakes, improving response times and the effectiveness of relief efforts.

4. Public Awareness: Public awareness of earthquake risk can lead to more informed decisions about where to live and work, as well as how to secure homes and businesses against seismic hazards.

5. Scientific Understanding: Developing earthquake prediction models contributes to a deeper understanding of the Earth's geology and the dynamics of tectonic plates. This knowledge is valuable for academic research and can lead to advancements in seismology and geophysics.

6. Infrastructure Planning: Urban planners and engineers can use earthquake prediction models to inform the design and construction of critical infrastructure, such as bridges, dams, and hospitals, to make them more resilient to seismic activity.

DESIGN THINKING:



1. Data Exploration:

Import the earthquake dataset from Kaggle into your preferred data analysis environment (e.g., Python with libraries like Pandas and Matplotlib).

Explore the dataset to understand its structure and contents. Check for missing values, data types, and outliers.

Identify the key features you want to use for prediction, such as latitude, longitude, and time.

2. Data Visualization:

Use data visualization tools (e.g., Matplotlib, Seaborn, or Plotly) to create plots and graphs to gain insights from the data.

Plot earthquake occurrences on a world map to visualize their distribution globally. You can use tools like Folium or Plotly for interactive maps.

3. Data Preprocessing:

Prepare the data for training and testing. This includes feature scaling, handling missing values (if any), and encoding categorical variables (if necessary).

Split the dataset into a training set and a testing set. Typically, an 80/20 or 70/30 split is used, with the majority for training.

4. Neural Network Model:

Build a neural network model using a deep learning framework like TensorFlow or PyTorch. For predicting earthquake magnitudes, you can design a regression neural network.

Define the architecture of your neural network, including the number of layers and neurons, activation functions, and loss function.

Compile the model with an appropriate optimizer and evaluation metric.

5. Model Training:

Train your neural network model on the training data. Monitor the training process for metrics like loss and validation loss.

Experiment with different hyperparameters, such as learning rate and batch size, to optimize your model's performance.

6. Model Evaluation:

Evaluate your trained model on the testing data using appropriate evaluation metrics for regression tasks. Common metrics include Mean Absolute Error (MAE) and Mean Squared Error (MSE).

7. FineTuning and Optimization:

If the model's performance is not satisfactory, consider finetuning the architecture, hyperparameters, or using more advanced techniques like recurrent neural networks (RNNs) if timeseries data is involved.

8. Deployment (Optional):

If you intend to use this model for realtime prediction or deployment, create an API or interface to make predictions based on new data.

Phases of Development :

In **first phase** we defined our project and try to explain our point of view and how we solve the problem. Also we gave an overall path to develop a model.

In **second phase** we given a little bit detailed definition of the project and given the type of model used to develop an earthquake prediction model by us.

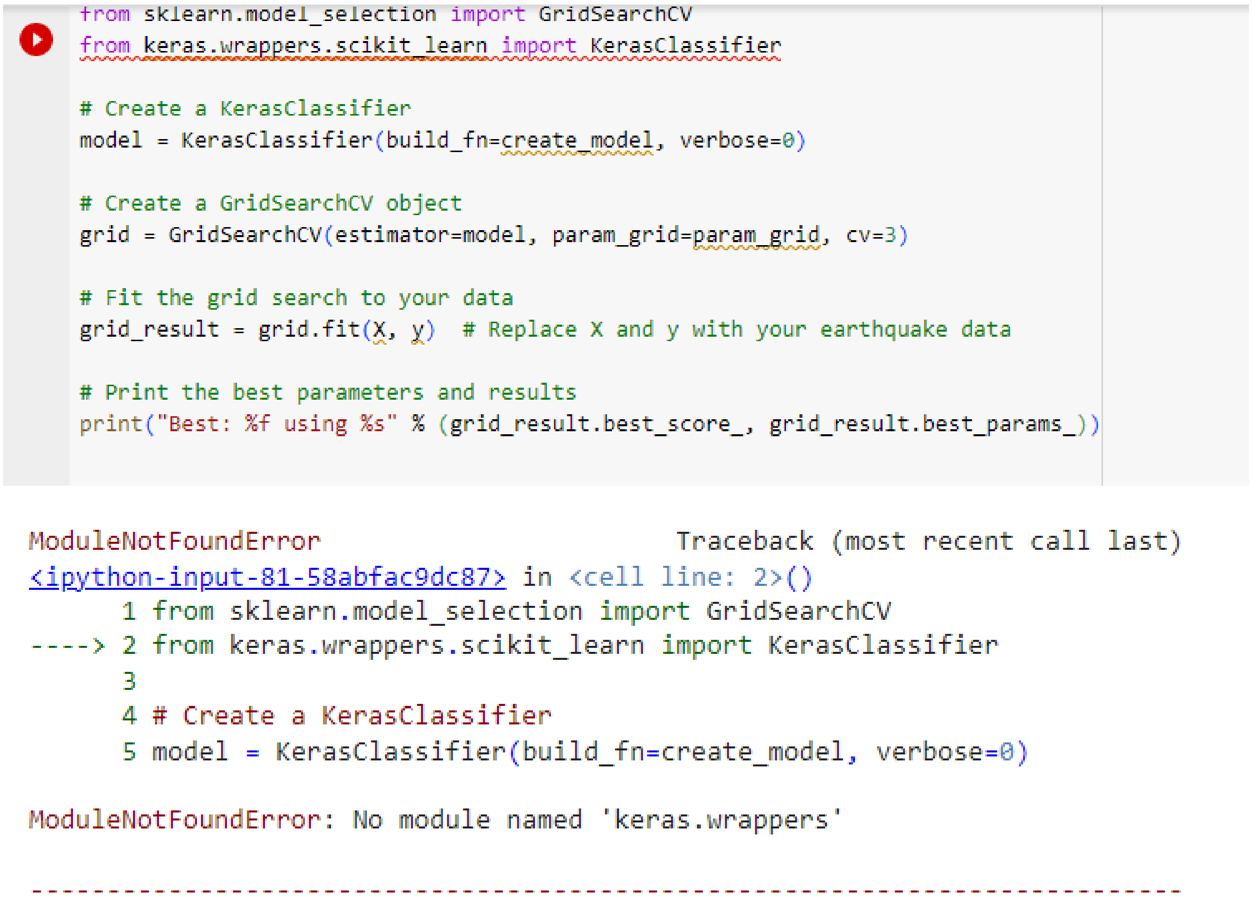
In **third phase** we preprocess the data by cleaning it. We taken the only part of data needed for our model and checked for the missing values but luckily we don’t have any missing values. Also we visualized the data in the world map and did some analysis on the data.

In **fourth phase** we again visualized the data in the world map and divided the dataset into four parts to train the model and with another part to test the model and to check the accuracy score.

At last we will train the model and we are going to test model and calculate the accuracy score.

Training the model:

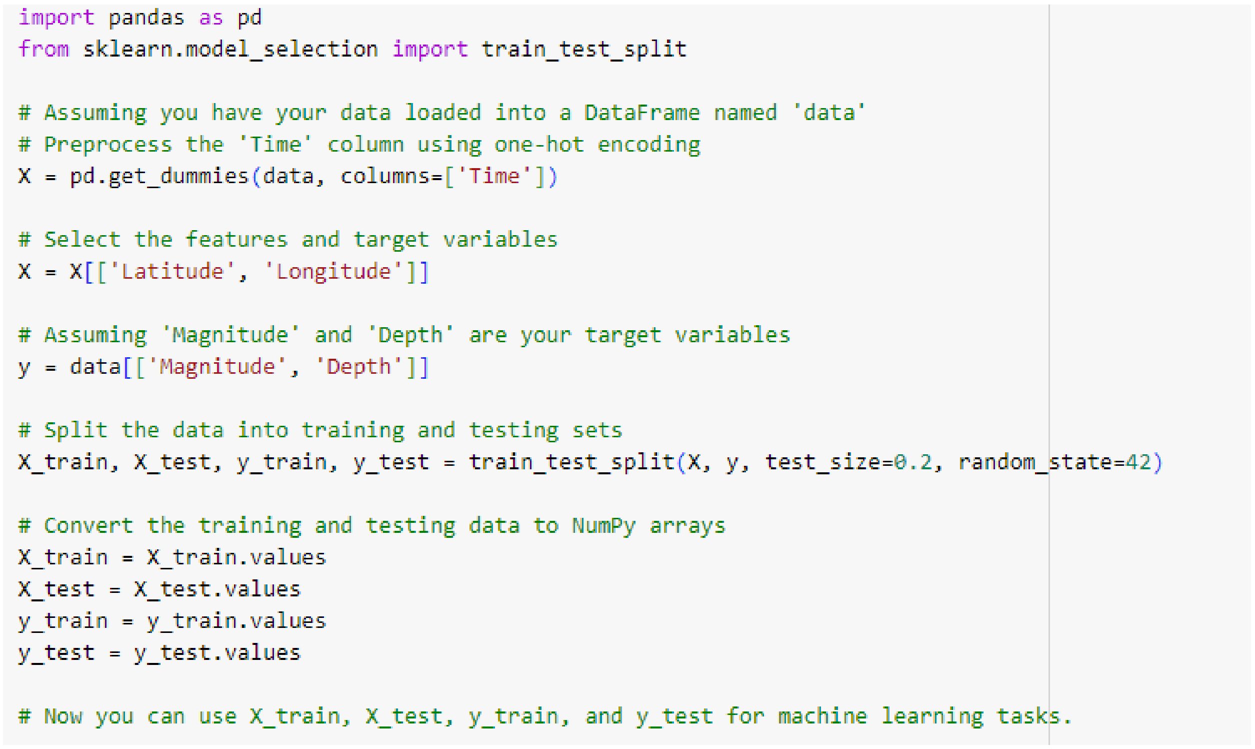
As per the plan ,Now we need to train the model.The most widely used is Ann.Also in the design thinking we specified we will use Artificial Neural Network.But there is problem in google colab which can't the module kensar.wrapper.



```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

# Create a KerasClassifier
model = KerasClassifier(build_fn=create_model, verbose=0)

# Create a GridSearchCV object
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)

# Fit the grid search to your data
grid_result = grid.fit(X, y)  # Replace X and y with your earthquake data

# Print the best parameters and results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-81-58abfac9dc87> in <cell line: 2>()
      1 from sklearn.model_selection import GridSearchCV
----> 2 from keras.wrappers.scikit_learn import KerasClassifier
      3 
      4 # Create a KerasClassifier
      5 model = KerasClassifier(build_fn=create_model, verbose=0)

ModuleNotFoundError: No module named 'keras.wrappers'

---------------------------------------------------------------------------
```

It also recommended that the keras was shifted to the TensorFlow. We also tried that but in vain it also returned error .So we tried pytorch.So for now we again separate the data into four parts so that we can feed them into model developed by pytorch.



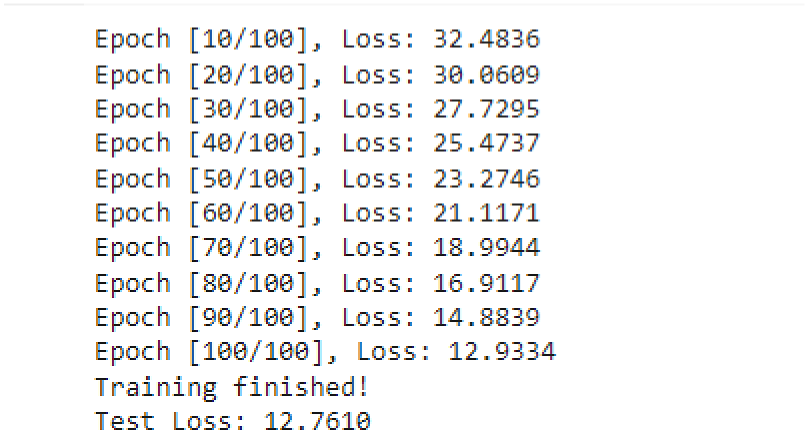
```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming you have your data loaded into a DataFrame named 'data'
# Preprocess the 'Time' column using one-hot encoding
X = pd.get_dummies(data, columns=['Time'])

# Select the features and target variables
X = X[['Latitude', 'Longitude']]

# Assuming 'Magnitude' and 'Depth' are your target variables
y = data[['Magnitude', 'Depth']]

# Split the 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)

# Convert the training and testing data to NumPy arrays
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values

# Now you can use X_train, X_test, y_train, and y_test for machine learning tasks.
```

Code for model:

```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load your earthquake data (replace 'your_data.csv' with your actual data file)
data = pd.read_csv('database.csv')

# Split the data into features (X) and targets (y)
X = data[['Latitude', 'Longitude', 'Depth']]
y = data[['Magnitude']]

# Split the 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)

# Standardize the input features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Define a neural network model
class EarthquakeModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Initialize the model
input_size = X_train.shape[1]
output_size = 1
hidden_size = 64
model = EarthquakeModel(input_size, hidden_size, output_size)

# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    inputs = torch.tensor(X_train, dtype=torch.float32)
    targets = torch.tensor(y_train.values, dtype=torch.float32)

    # Forward pass
    outputs = model(inputs)
    loss = criterion(outputs, targets)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')

    print('Training finished!')

    # Evaluate the model on the test data
    with torch.no_grad():
       test_inputs = torch.tensor(X_test, dtype=torch.float32)
       test_targets = torch.tensor(y_test.values, dtype=torch.float32)
       test_outputs = model(test_inputs)
       test_loss = criterion(test_outputs, test_targets)

    print(f'Test Loss: {test_loss.item():.4f}')
```

OUTPUT:



```
Epoch [10/100], Loss: 32.4836
Epoch [20/100], Loss: 30.0609
Epoch [30/100], Loss: 27.7295
Epoch [40/100], Loss: 25.4737
Epoch [50/100], Loss: 23.2746
Epoch [60/100], Loss: 21.1171
Epoch [70/100], Loss: 18.9944
Epoch [80/100], Loss: 16.9117
Epoch [90/100], Loss: 14.8839
Epoch [100/100], Loss: 12.9334
Training finished!
Test Loss: 12.7610
```

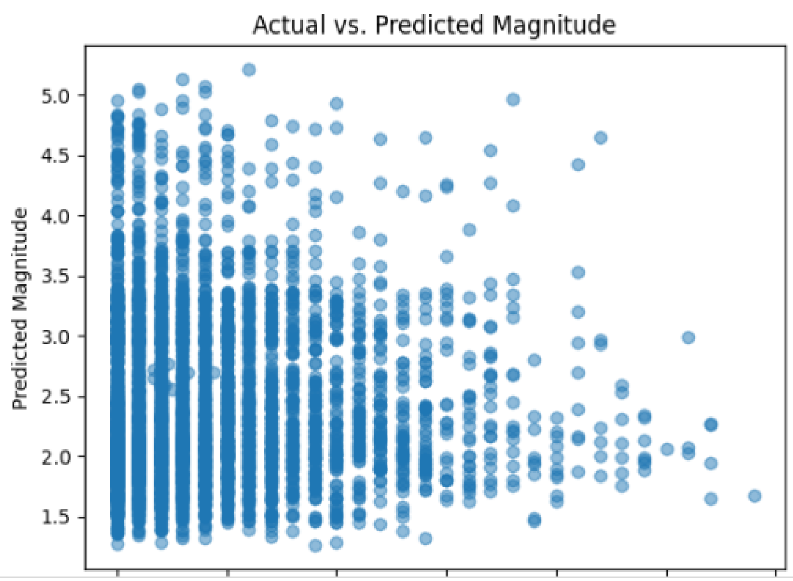
Model evalution:

Now we need to evalute the model we developed.We use the following code to check the evalution.

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Calculate metrics
mae = mean_absolute_error(y_test, test_outputs)
mse = mean_squared_error(y_test, test_outputs)
rmse = mean_squared_error(y_test, test_outputs, squared=False)  # RMSE
r2 = r2_score(y_test, test_outputs)

print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'R-squared (R2): {r2:.4f}')

# Coded by austin
```

```
Mean Absolute Error (MAE): 3.4830
Mean Squared Error (MSE): 12.7610
Root Mean Squared Error (RMSE): 3.5723
R-squared (R2): -68.1853
```

```python
plt.scatter(y_test, test_outputs, alpha=0.5)
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("Actual vs. Predicted Magnitude")
plt.show()
```



✓ Connected to Python 3 Google Compute Engine backend

The scores provided are metrics commonly used to evaluate the performance of regression models, such as the earthquake magnitude prediction model you've built. Let's break down what each of these metrics means and how to interpret them:

Mean Absolute Error (MAE): 3.4830

- MAE represents the average absolute difference between the predicted values and the actual values.
- In your context, it means that, on average, your model's predictions for earthquake magnitude are off by approximately 3.4830 units.

Mean Squared Error (MSE): 12.7610

- MSE is the average of the squared differences between predicted and actual values.
- It tends to amplify larger errors, making it more sensitive to outliers.
- Your MSE of 12.7610 indicates that, on average, the squared errors between predictions and actual values are around 12.7610.

Root Mean Squared Error (RMSE): 3.5723

- RMSE is the square root of the MSE and is expressed in the same units as the target variable.
- It provides a similar measure of error as the MAE but penalizes larger errors more heavily due to the square root.
- An RMSE of 3.5723 means that, on average, your model's predictions are off by approximately 3.5723 units.

R-squared (R2): -68.1853

- R-squared, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable (earthquake magnitude) that is predictable from the independent variables (your model's predictions).
- An R-squared value can range from 0 to 1, with higher values indicating a better fit. However, it can also be negative when the model fits worse than a horizontal line.
- An R2 of -68.1853 suggests that your model doesn't fit the data well, and its predictions are worse than a horizontal line.

In summary, the MAE, MSE, and RMSE scores provide a measure of the error or the difference between your model's predictions and the actual earthquake magnitudes. Lower values for these metrics indicate better model performance.

The R-squared score, however, indicates the goodness of fit of your model. A negative R2 suggests that your model may not be a good fit for the data, and it's performing worse than a simple horizontal line.

It's important to further analyze your model and dataset to identify potential issues and improve model performance. This could involve feature engineering, model tuning, or considering different algorithms or approaches. So let us change the hyper-parameter tuning to check whether we can improve the scores.

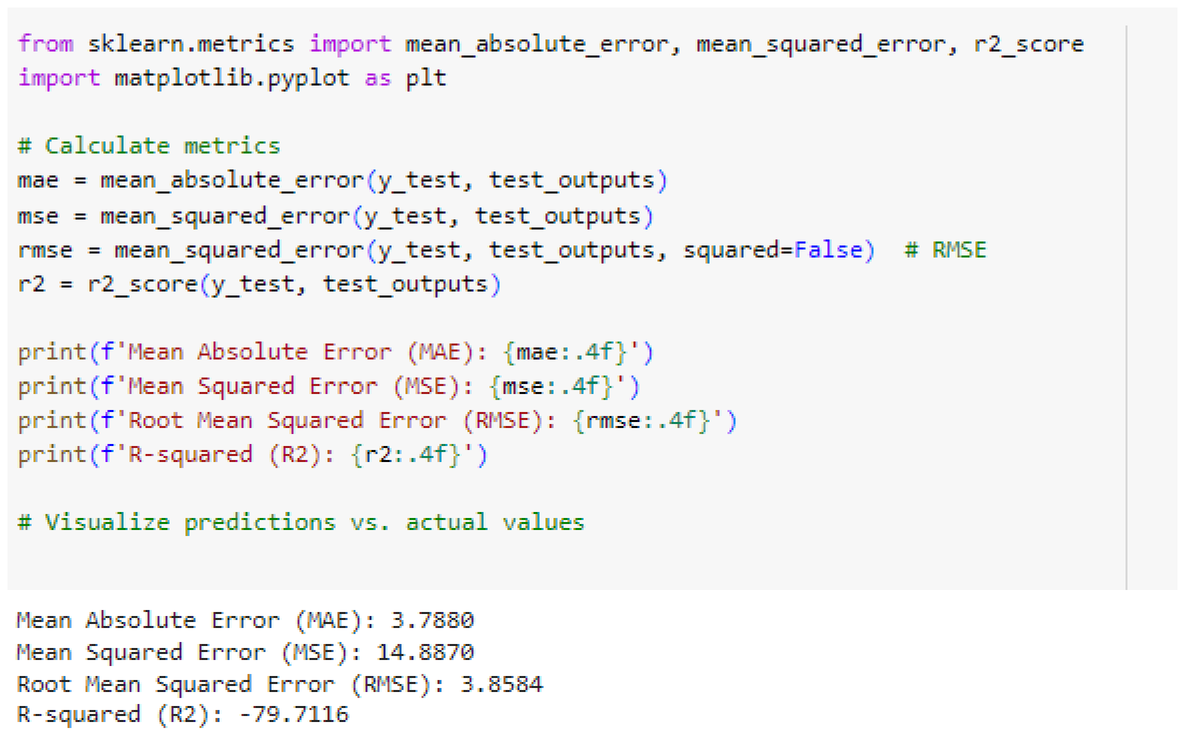
Hyper-parameter tuning:

Code:

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import GridSearchCV

# Define your PyTorch model
class EarthquakeModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Load and preprocess your data as you did before
# Create the model instance
input_size = X_train.shape[1]
output_size = 1
hidden_size = 64
model = EarthquakeModel(input_size, hidden_size, output_size)

# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    inputs = torch.tensor(X_train, dtype=torch.float32)
    targets = torch.tensor(y_train.values, dtype=torch.float32)

    # Forward pass
    outputs = model(inputs)
    loss = criterion(outputs, targets)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')

print('Training finished!')

# Evaluate the model on the test data
with torch.no_grad():
    test_inputs = torch.tensor(X_test, dtype=torch.float32)
    test_targets = torch.tensor(y_test.values, dtype=torch.float32)
    test_outputs = model(test_inputs)
    test_loss = criterion(test_outputs, test_targets)
```

print(f'Test Loss: {test_loss.item():.4f}')

After the changes let us check the scores again.



```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Calculate metrics
mae = mean_absolute_error(y_test, test_outputs)
mse = mean_squared_error(y_test, test_outputs)
rmse = mean_squared_error(y_test, test_outputs, squared=False)  # RMSE
r2 = r2_score(y_test, test_outputs)

print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'R-squared (R2): {r2:.4f}')

# Visualize predictions vs. actual values
```

```
Mean Absolute Error (MAE): 3.7880
Mean Squared Error (MSE): 14.8870
Root Mean Squared Error (RMSE): 3.8584
R-squared (R2): -79.7116
```

The metrics you've provided after evaluation indicate that there hasn't been a significant improvement in the model's performance. In fact, the values for MAE, MSE, RMSE, and R-squared have slightly increased, which typically means the model's predictions are further from the actual values.So let stop the tuning here.

Innovation techniques during development:

In your specific case, using PyTorch for your earthquake magnitude prediction model is perfectly fine. The impact on your project will likely be minimal. PyTorch offers a more user-friendly experience for building and experimenting with models. If you're comfortable with it and your project requirements are met, there's no need to switch to TensorFlow.

In summary, both frameworks have their strengths, and the choice between them often comes down to personal preference and specific project requirements. PyTorch is an excellent choice for research, experimentation, and building custom models, as you've done in your earthquake prediction project.

Conclusion:

Thus we developed a earthquake prediction model to check whether there will be a occurance of earthquake or not in created and evaluated in the pytorch.

Python file -https://colab.research.google.com/drive/1I-XykVefOZVGO9rV-ktkkPlxLMupnlWJ?usp=sharing