# earthquake-prediction-1

November 20, 2024

[ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
```

[ ]:
```
initial_data = pd.read_csv("earth.csv")
```

[ ]:
```
print(initial_data.head())
```

```
                       time  latitude  longitude   depth  mag magType    nst  \
0  2024-11-12T14:31:59.659Z   -4.5205   152.4422  89.123  5.3     mww  126.0
1  2024-11-12T13:48:50.344Z   36.3155   141.2801  38.636  4.5      mb   58.0
2  2024-11-12T09:24:26.883Z    0.5833   126.1828  17.810  5.2     mww   81.0
3  2024-11-12T08:30:35.144Z   27.3495    88.3392  10.000  4.4      mb   40.0
4  2024-11-12T08:13:20.377Z   37.5859   135.4869 358.970  4.4      mb   97.0

     gap   dmin   rms  …                   updated  \
0   62.0  0.429  0.76  …  2024-11-12T14:56:49.850Z
1  137.0  2.487  0.53  …  2024-11-12T17:19:14.040Z
2   46.0  1.198  1.10  …  2024-11-12T09:39:46.040Z
3  145.0  2.753  0.83  …  2024-11-12T14:45:37.462Z
4   56.0  2.408  0.72  …  2024-11-12T08:49:19.040Z

                                  place        type horizontalError  \
0  27 km SE of Kokopo, Papua New Guinea  earthquake            7.91
1                62 km E of Ōarai, Japan  earthquake            9.05
2          135 km W of Ternate, Indonesia  earthquake            6.71
3            10 km NE of Gyalshing, India  earthquake           10.06
4            131 km WNW of Anamizu, Japan  earthquake            7.57

  depthError  magError  magNst    status  locationSource magSource
0      5.007     0.098    10.0  reviewed              us        us
1      7.721     0.079    50.0  reviewed              us        us
2      4.353     0.086    13.0  reviewed              us        us
3      1.922     0.098    30.0  reviewed              us        us
4      3.966     0.041   172.0  reviewed              us        us
```

```
[5 rows x 22 columns]
```

[ ]:
```python
timed_data= initial_data[['latitude', 'longitude', 'depth', 'mag', 'magType',
 ↪'gap', 'dmin','time']]
timed_data.head()
```

```
   latitude  longitude    depth  mag magType    gap   dmin  \
0   -4.5205   152.4422   89.123  5.3     mww   62.0  0.429
1   36.3155   141.2801   38.636  4.5      mb  137.0  2.487
2    0.5833   126.1828   17.810  5.2     mww   46.0  1.198
3   27.3495    88.3392   10.000  4.4      mb  145.0  2.753
4   37.5859   135.4869  358.970  4.4      mb   56.0  2.408

                       time
0  2024-11-12T14:31:59.659Z
1  2024-11-12T13:48:50.344Z
2  2024-11-12T09:24:26.883Z
3  2024-11-12T08:30:35.144Z
4  2024-11-12T08:13:20.377Z
```

[ ]:
```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
timed_data['magType_num'] = label_encoder.fit_transform(timed_data['magType'])

timed_data.head()
```

```
   latitude  longitude    depth  mag magType    gap   dmin  \
0   -4.5205   152.4422   89.123  5.3     mww   62.0  0.429
1   36.3155   141.2801   38.636  4.5      mb  137.0  2.487
2    0.5833   126.1828   17.810  5.2     mww   46.0  1.198
3   27.3495    88.3392   10.000  4.4      mb  145.0  2.753
4   37.5859   135.4869  358.970  4.4      mb   56.0  2.408

                       time  magType_num
0  2024-11-12T14:31:59.659Z           10
1  2024-11-12T13:48:50.344Z            1
2  2024-11-12T09:24:26.883Z           10
3  2024-11-12T08:30:35.144Z            1
4  2024-11-12T08:13:20.377Z            1
```

[ ]:
```python
timed_data= timed_data.drop(['magType'], axis=1)
print(timed_data.head())
```

```
   latitude  longitude    depth  mag    gap   dmin                      time  \
0   -4.5205   152.4422   89.123  5.3   62.0  0.429  2024-11-12T14:31:59.659Z
1   36.3155   141.2801   38.636  4.5  137.0  2.487  2024-11-12T13:48:50.344Z
2    0.5833   126.1828   17.810  5.2   46.0  1.198  2024-11-12T09:24:26.883Z
3   27.3495    88.3392   10.000  4.4  145.0  2.753  2024-11-12T08:30:35.144Z
```

```
4   37.5859   135.4869  358.970  4.4   56.0  2.408  2024-11-12T08:13:20.377Z

   magType_num
0           10
1            1
2           10
3            1
4            1
```

[ ]:
```
timed_data.shape
timed_data = timed_data.drop_duplicates()
```

[ ]:
```
timed_data= timed_data.dropna()
df= timed_data[['latitude', 'longitude', 'depth', 'mag', 'magType_num', 'gap',␣
 ↪'dmin']]
df.shape
```

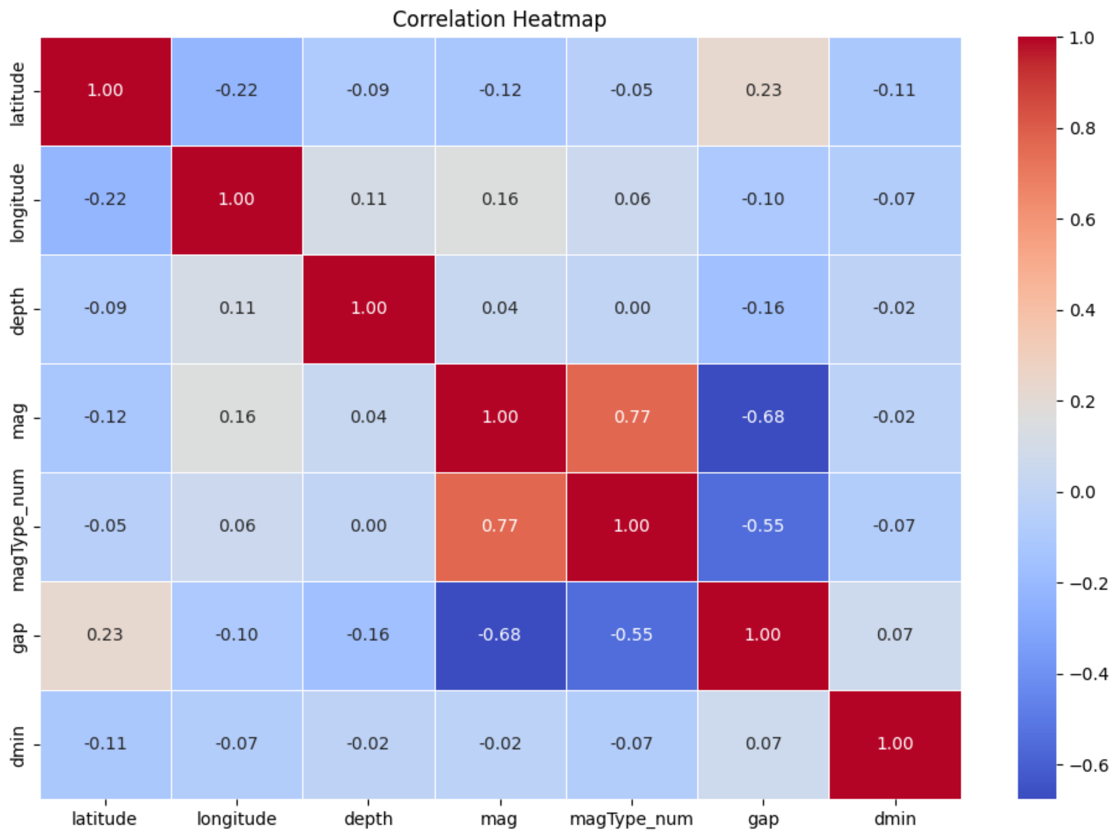
[ ]: (581, 7)

[ ]:
```
df.info()
fdata=df
```

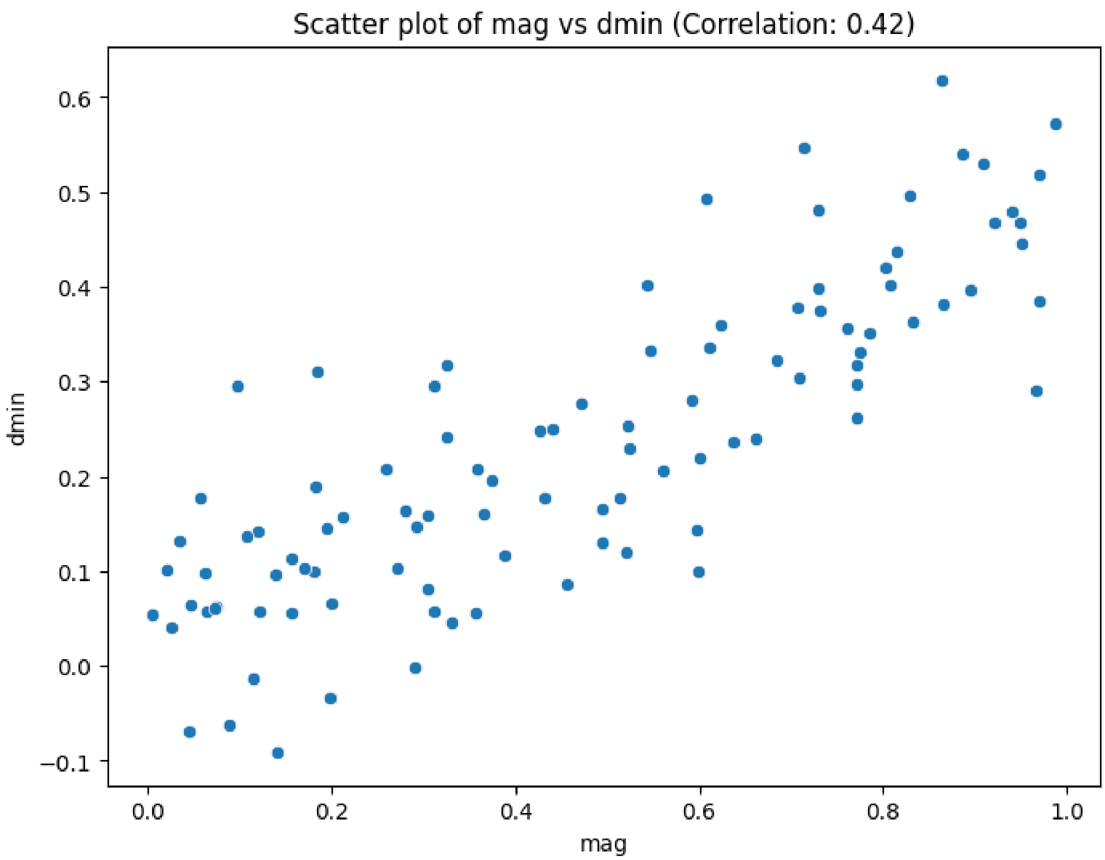
```
<class 'pandas.core.frame.DataFrame'>
Index: 581 entries, 0 to 761
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   latitude     581 non-null    float64
 1   longitude    581 non-null    float64
 2   depth        581 non-null    float64
 3   mag          581 non-null    float64
 4   magType_num  581 non-null    int64
 5   gap          581 non-null    float64
 6   dmin         581 non-null    float64
dtypes: float64(6), int64(1)
memory usage: 36.3 KB
```

[ ]:
```
import seaborn as sns

numerical_columns = fdata.select_dtypes(include=['float64', 'int64']).columns
correlation_matrix = fdata[numerical_columns].corr()

# Creating the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
 ↪linewidths=0.5)
plt.title('Correlation Heatmap')
```

```
plt.show()
```




[ ]:
```
np.random.seed(42)
mag = np.random.rand(100)
dmin = 0.5 * mag + np.random.normal(0, 0.1, 100)

data = pd.DataFrame({'mag': mag, 'dmin': dmin})
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data, x='mag', y='dmin')

# Add title and labels
plt.title('Scatter plot of mag vs dmin (Correlation: 0.42)')
plt.xlabel('mag')
plt.ylabel('dmin')

# Show the plot
plt.show()
```




```
[ ]: from sklearn.ensemble import RandomForestRegressor
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score, classification_report
```

```
[ ]: x= df[['latitude', 'longitude', 'depth', 'gap', 'dmin']]
     y= df['mag']
```

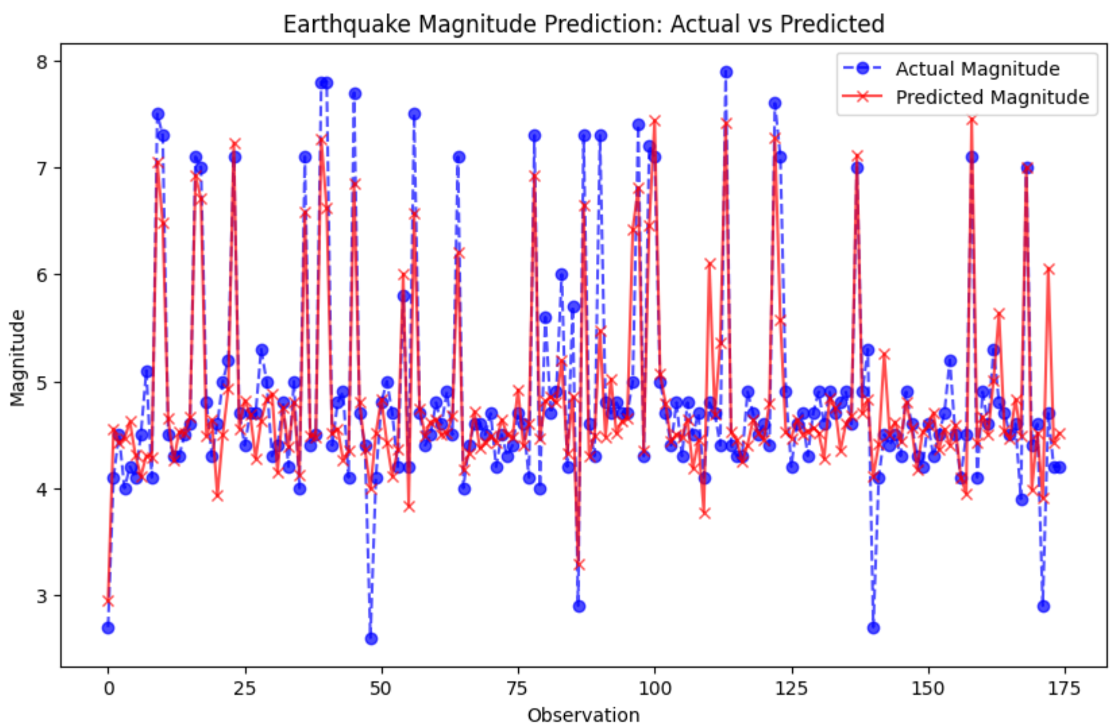
**RANDOM FOREST REGRESSION**

```
[ ]: from sklearn.ensemble import RandomForestRegressor
     from sklearn.metrics import mean_squared_error, r2_score
     from sklearn.model_selection import train_test_split
```

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
     ↪random_state=42)

     rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
     rf_regressor.fit(X_train, y_train)

     y_pred = rf_regressor.predict(X_test)
```

[ ]:
```
y_testArr= np.array(y_test)
```

[ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(y_testArr, label='Actual Magnitude', marker='o', color='b',␣
 ↪linestyle='--', alpha=0.7)
plt.plot(y_pred, label='Predicted Magnitude', marker='x', color='r',␣
 ↪linestyle='-', alpha=0.7)

plt.xlabel('Observation')
plt.ylabel('Magnitude')
plt.title('Earthquake Magnitude Prediction: Actual vs Predicted')
plt.legend()

plt.show()
```




[ ]:
```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse*100)
print("R^2 Score:", r2*100)
```

```
Mean Squared Error: 22.572230857142834
R^2 Score: 79.45543242957291
```

```
[ ]: import numpy as np
from sklearn.metrics import accuracy_score

# Define the bin edges (for example, 0-1, 1-2, ..., 9-10 for earthquake␣
 ↪magnitude)
bins = np.arange(0, 10, 2)  # Bins of size 1 for magnitudes between 0 and 10
labels = np.digitize(y_test, bins)  # Assigning labels for the true values
y_pred_binned = np.digitize(y_pred, bins)  # Assigning labels for the predicted␣
 ↪values

# Calculate accuracy (percentage of correct predictions in the correct bin)
accuracy = accuracy_score(labels, y_pred_binned)
print(f'Accuracy: {accuracy*100: .2f}')
```

```
Accuracy:  92.00
```

**SVM**

```
[ ]: from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.preprocessing import MinMaxScaler
```

```
[ ]: X = df.drop(columns=['mag'])  # Features
y = df['mag']  # Target variable (magnitude)
```

```
[ ]: scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Scale the features
scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))
```

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,␣
 ↪test_size=0.3, random_state=42)
```

```
[ ]: svm_model = SVR(kernel='rbf')
```

```
[ ]: svm_model.fit(X_train, y_train)
y_pred_scaled = svm_model.predict(X_test)
```

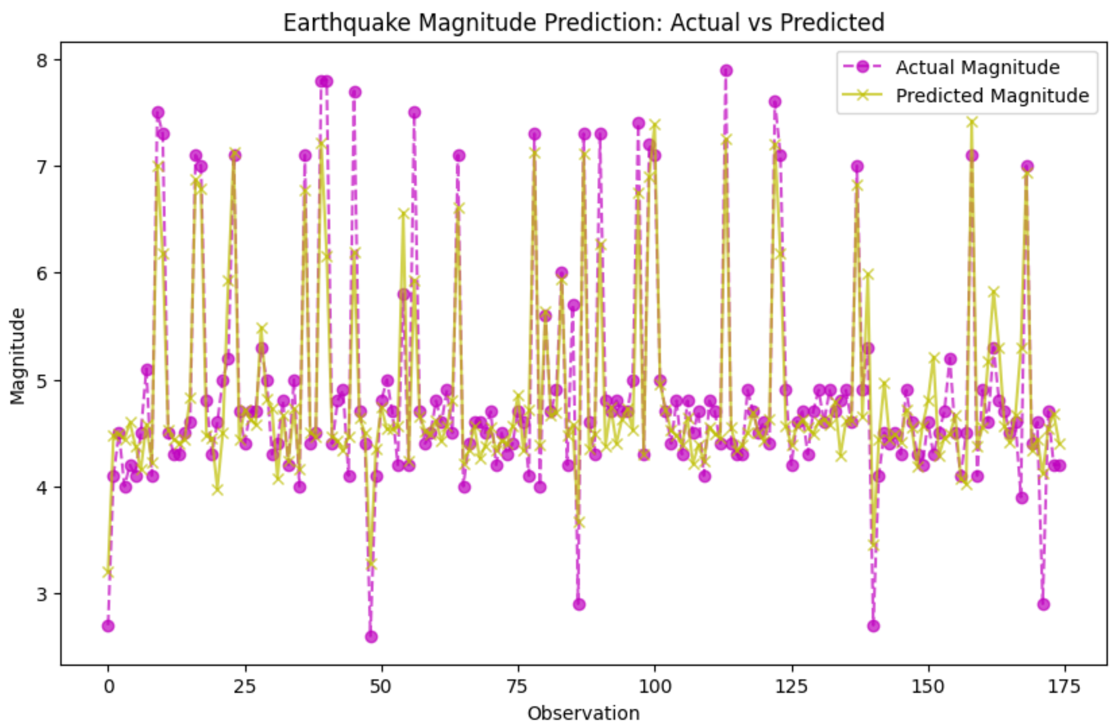
```
[ ]: y_pred_original = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1))
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1))
```

```
[ ]: mse = mean_squared_error(y_test_original, y_pred_original)
r2 = r2_score(y_test_original, y_pred_original)

print("Mean Squared Error:", mse*100)
print("R^2 Score:", r2*100)
```

```
Mean Squared Error: 18.272459190015784
R^2 Score: 83.36895564806977
```

[ ]:
```
y_testArr= np.array(y_test)
```

[ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(y_test_original, label='Actual Magnitude', marker='o', color='m',␣
 ↪linestyle='--', alpha=0.7)
plt.plot(y_pred_original, label='Predicted Magnitude', marker='x', color='y',␣
 ↪linestyle='-', alpha=0.7)
plt.xlabel('Observation')
plt.ylabel('Magnitude')
plt.title('Earthquake Magnitude Prediction: Actual vs Predicted')
plt.legend()
plt.show()
```




### EXTREME GRADIENT BOOSTING

[ ]:
```
import xgboost as xgb
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

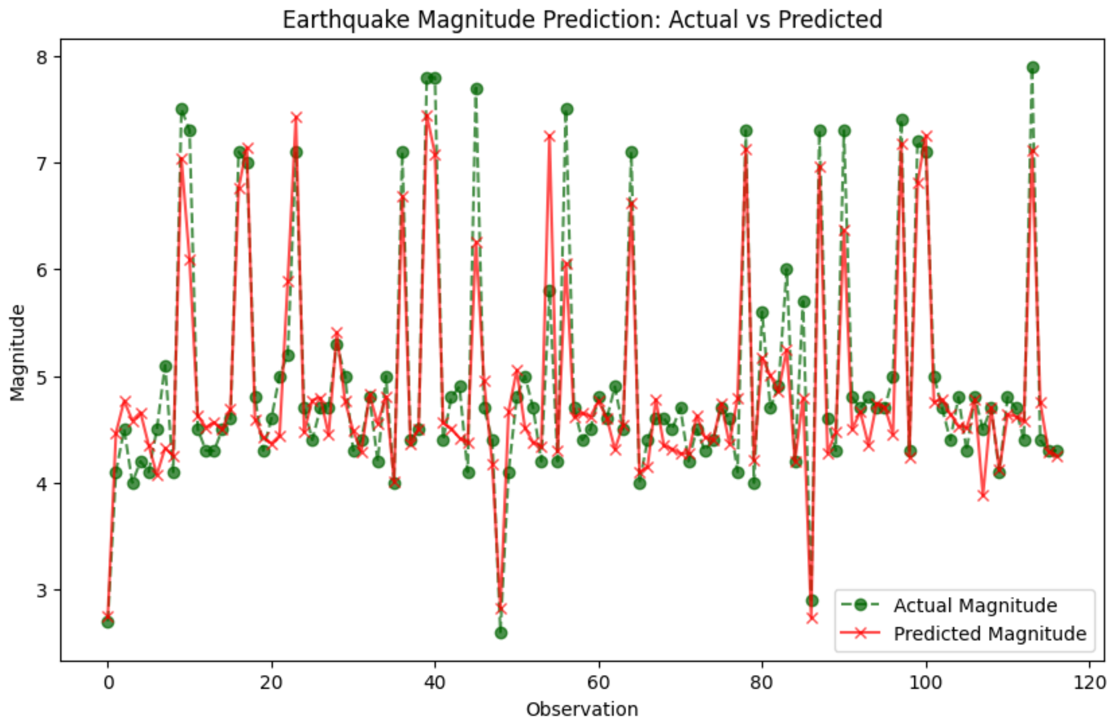
[ ]:
```
xg_reg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,␣
    ↪max_depth=6)
```

[ ]:
```
xg_reg.fit(X_train_scaled, y_train)
y_pred = xg_reg.predict(X_test_scaled)
```

[ ]:
```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse*100)
print("R^2 Score:", r2*100)
```

```
Mean Squared Error: 17.39545966030056
R^2 Score: 86.1940796346821
```

[ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(y_test.values, label='Actual Magnitude', marker='o',␣
    ↪color='darkgreen', linestyle='--', alpha=0.7)
plt.plot(y_pred, label='Predicted Magnitude', marker='x', color='r',␣
    ↪linestyle='-', alpha=0.7)
plt.xlabel('Observation')
plt.ylabel('Magnitude')
plt.title('Earthquake Magnitude Prediction: Actual vs Predicted')
plt.legend()
plt.show()
```

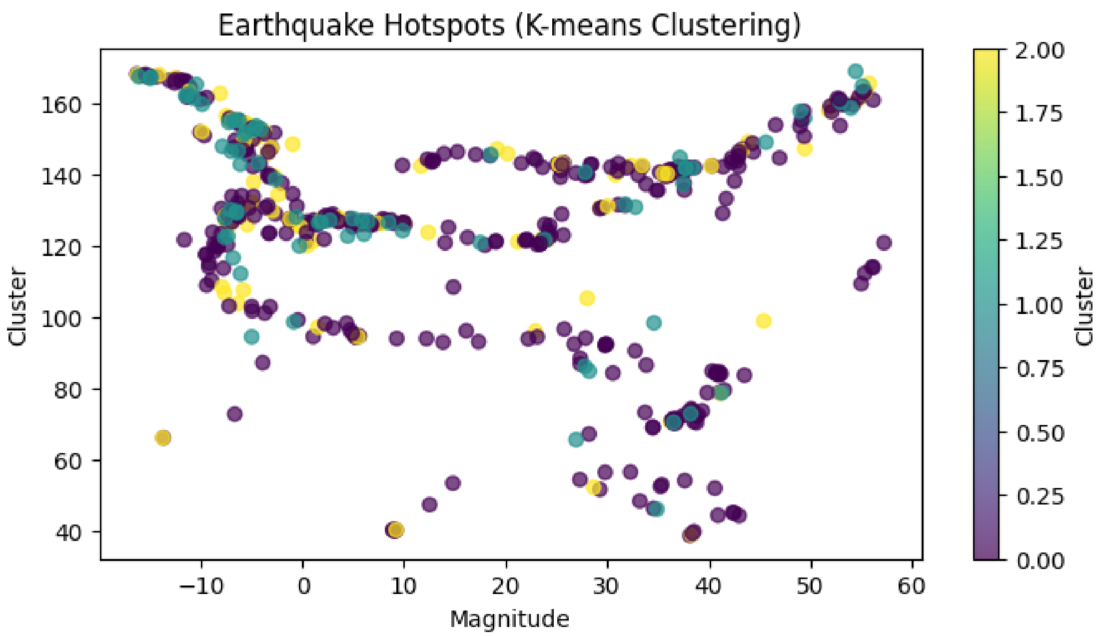



## EARTHQUAKE HOTSPOT ANALYSIS

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans, DBSCAN
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
features = df[['latitude', 'longitude', 'depth', 'mag', 'gap', 'dmin']]
mag_values= df[['mag']].values
scaler = StandardScaler()
features_scaled = scaler.fit_transform(mag_values)
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(features_scaled)
```

[ ]:
```
plt.figure(figsize=(8, 4))
plt.scatter(df['latitude'], df['longitude'], c=df['cluster'], cmap='viridis',␣
 ↪marker='o', alpha=0.7)
plt.colorbar(label='Cluster')
plt.title('Earthquake Hotspots (K-means Clustering)')
plt.xlabel('Magnitude')
plt.ylabel('Cluster')
plt.show()
```




[ ]:
```
import folium

# Create a base map
map = folium.Map(location=[df['latitude'].mean(), df['longitude'].mean()],␣
↪zoom_start=5)

# Add earthquake points
for index, row in df.iterrows():
    folium.CircleMarker(
        location=[row['latitude'], row['longitude']],
        radius=5,
        color='blue' if row['cluster'] == 0 else 'green' if row['cluster'] == 1␣
↪else 'red',
        fill=True,
        fill_color='blue' if row['cluster'] == 0 else 'green' if row['cluster']␣
↪== 1 else 'red',
        fill_opacity=0.6
    ).add_to(map)
#Blue: low-magnitude earthquakes
#Green: moderate-magnitude earthquakes
#Red: high-magnitude earthquakes

# Display the map
map
```

[ ]: <folium.folium.Map at 0x79bcb8681f90>

```
[ ]: cluster_centers = kmeans.cluster_centers_
     cluster_centers_original_scale = scaler.inverse_transform(cluster_centers)
```

**FEEDBACK NEURAL NETWORKS**

```
[ ]: import pandas as pd
     from sklearn.preprocessing import StandardScaler
     from sklearn.model_selection import train_test_split
     import torch
     from torch.utils.data import DataLoader, TensorDataset
```

```
[ ]: X = df[['latitude', 'longitude', 'mag', 'dmin', 'gap', 'depth']].values
     y = df['mag'].values
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
     X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
      ↪random_state=42)

     train_tensor = TensorDataset(torch.tensor(X_train, dtype=torch.float32), torch.
      ↪tensor(y_train, dtype=torch.float32))
     test_tensor = TensorDataset(torch.tensor(X_test, dtype=torch.float32), torch.
      ↪tensor(y_test, dtype=torch.float32))

     train_loader = DataLoader(train_tensor, batch_size=8, shuffle=True)
     test_loader = DataLoader(test_tensor, batch_size=8)
```

```
[ ]: import torch.nn as nn

     class EarthquakeModel(nn.Module):
         def __init__(self):
             super(EarthquakeModel, self).__init__()
             self.fc1 = nn.Linear(6, 64)  # Input layer (6 features)
             self.fc2 = nn.Linear(64, 32)  # Hidden layer
             self.fc3 = nn.Linear(32, 1)   # Output layer (1 value for magnitude)

         def forward(self, x):
             x = torch.relu(self.fc1(x))
             x = torch.relu(self.fc2(x))
             x = self.fc3(x)  # No activation at the output layer for regression
             return x

     # Instantiate the model
     model = EarthquakeModel()
```

```
[ ]: # Loss function and optimizer
     criterion = nn.MSELoss()
     optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

[ ]:
```python
# Training the model
num_epochs = 100

for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    for batch in train_loader:
        X_batch, y_batch = batch
        optimizer.zero_grad()  # Zero the gradients
        outputs = model(X_batch)  # Forward pass
        loss = criterion(outputs.view(-1), y_batch)  # Calculate loss
        loss.backward()  # Backward pass
        optimizer.step()  # Optimize weights

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

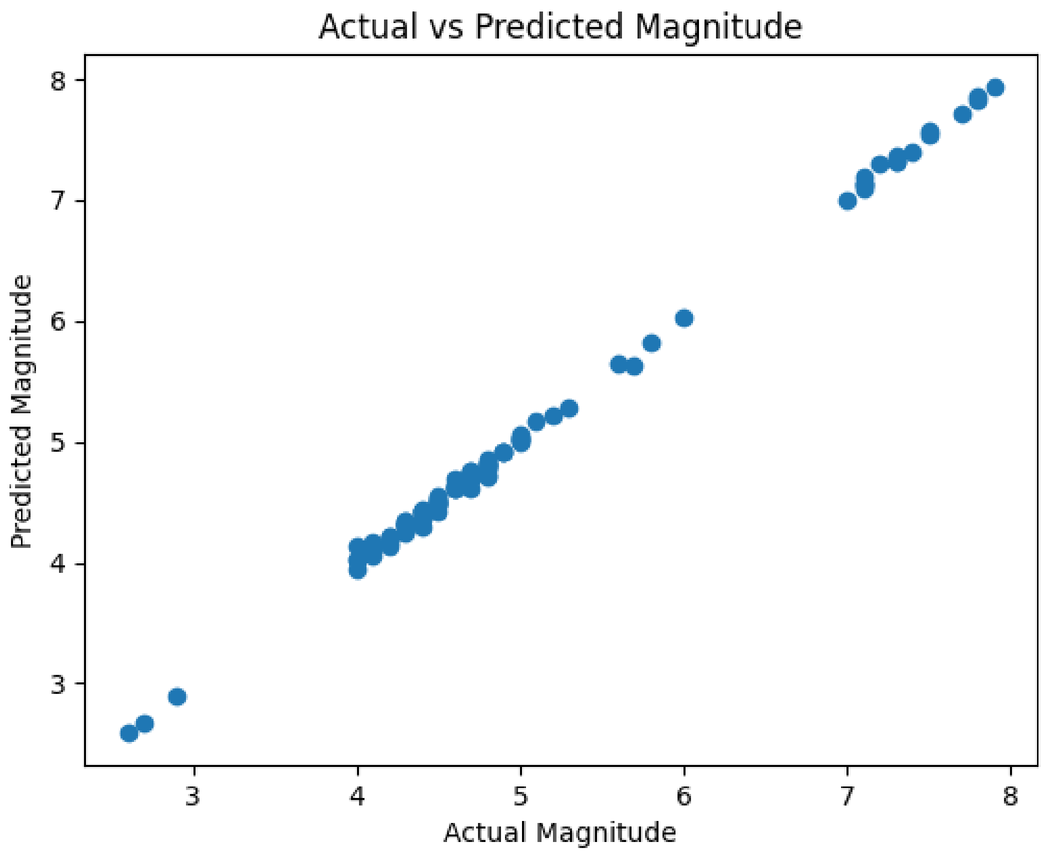
```
Epoch [10/100], Loss: 0.1445
Epoch [20/100], Loss: 0.0275
Epoch [30/100], Loss: 0.0026
Epoch [40/100], Loss: 0.0026
Epoch [50/100], Loss: 0.0034
Epoch [60/100], Loss: 0.0010
Epoch [70/100], Loss: 0.0029
Epoch [80/100], Loss: 0.0011
Epoch [90/100], Loss: 0.0012
Epoch [100/100], Loss: 0.0012
```

[ ]:
```python
# Evaluate the model
model.eval()  # Set model to evaluation mode
with torch.no_grad():
    predictions = model(torch.tensor(X_test, dtype=torch.float32))
    predictions = predictions.view(-1).numpy()

    # Calculate performance (e.g., Mean Squared Error on the test set)
    mse = ((predictions - y_test) ** 2).mean()
    print(f'Mean Squared Error: {mse*100:.2f}')
    ss_total = ((y_test - np.mean(y_test)) ** 2).sum()  # Total sum of squares
    ss_residual = ((y_test - predictions) ** 2).sum()  # Residual sum of squares
    r_squared = 1 - (ss_residual / ss_total)
    print(f'R-squared: {r_squared*100:.2f}')
```

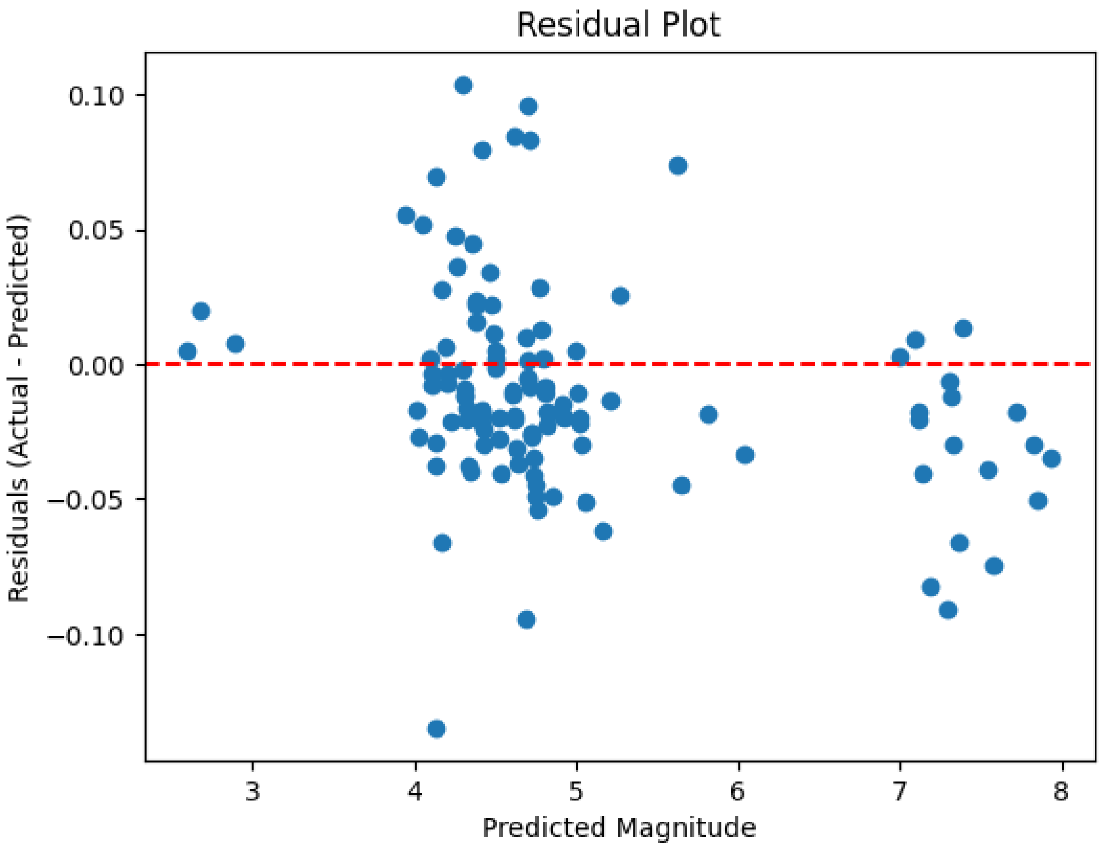
```
Mean Squared Error: 0.15
R-squared: 99.88
```

[ ]:
```python
import matplotlib.pyplot as plt
```

```python
# Plot actual vs predicted magnitudes
plt.scatter(y_test, predictions)
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title('Actual vs Predicted Magnitude')
plt.show()
```

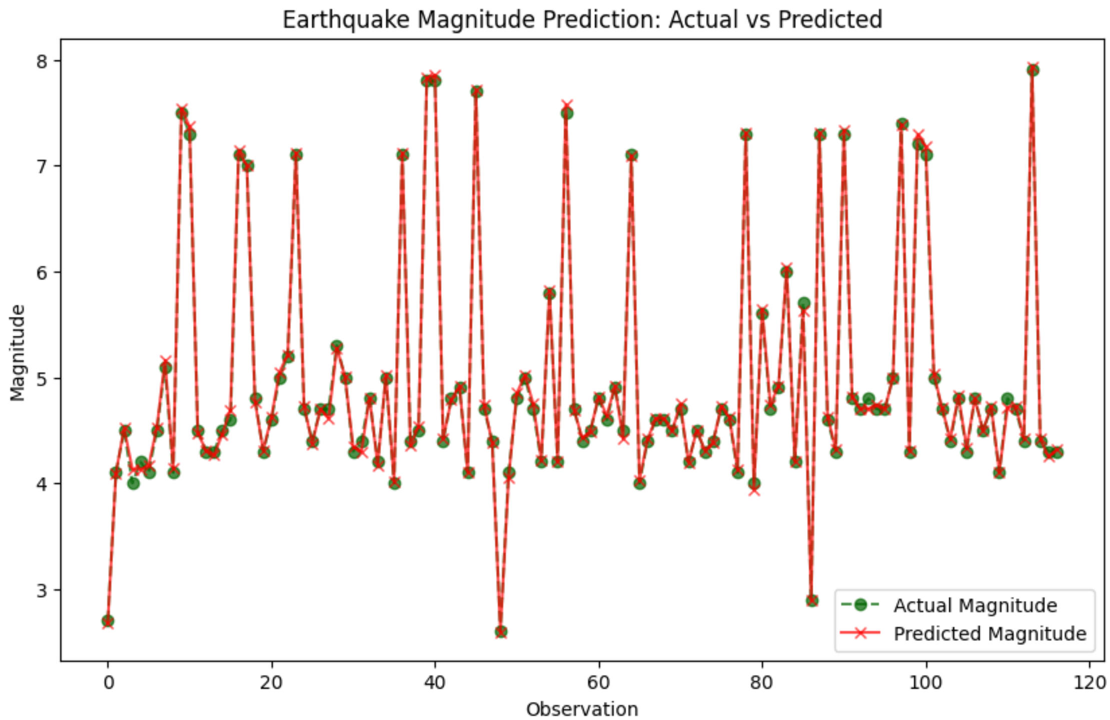



[ ]:
```python
residuals = y_test - predictions

# Plot residuals
plt.scatter(predictions, residuals)
plt.axhline(y=0, color='r', linestyle='--')  # Zero error line
plt.xlabel('Predicted Magnitude')
plt.ylabel('Residuals (Actual - Predicted)')
plt.title('Residual Plot')
plt.show()
```




[ ]:

```
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual Magnitude', marker='o', color='darkgreen',␣
 ↪linestyle='--', alpha=0.7)
plt.plot(predictions, label='Predicted Magnitude', marker='x', color='r',␣
 ↪linestyle='-', alpha=0.7)
plt.xlabel('Observation')
plt.ylabel('Magnitude')
plt.title('Earthquake Magnitude Prediction: Actual vs Predicted')
plt.legend()
plt.show()
```




# 1 LSTM

[ ]:
```
import pandas as pd

# Extracting relevant columns from the initial data
dfDate = pd.DataFrame(timed_data[['time', 'latitude', 'longitude', 'depth',␣
↪'mag', 'gap', 'dmin']])

# Convert 'time' column to datetime
dfDate['datetime'] = pd.to_datetime(initial_data['time'])

# Extract date, month, year, and optionally time
dfDate['date'] = dfDate['datetime'].dt.strftime('%Y-%m-%d')  # Keeps date as␣
↪'YYYY-MM-DD'
dfDate['month'] = dfDate['datetime'].dt.month
dfDate['year'] = dfDate['datetime'].dt.year
dfDate['time_of_day'] = dfDate['datetime'].dt.strftime('%H:%M:%S')  # Extracts␣
↪time

# Drop the original timestamp column if not needed
dfDate.drop(columns=['time', 'datetime'], inplace=True)

# Print the final DataFrame
```

```
dfDate.head()
```

[ ]:
```
   latitude  longitude    depth  mag    gap   dmin        date  month  year  \
0   -4.5205   152.4422   89.123  5.3   62.0  0.429  2024-11-12     11  2024
1   36.3155   141.2801   38.636  4.5  137.0  2.487  2024-11-12     11  2024
2    0.5833   126.1828   17.810  5.2   46.0  1.198  2024-11-12     11  2024
3   27.3495    88.3392   10.000  4.4  145.0  2.753  2024-11-12     11  2024
4   37.5859   135.4869  358.970  4.4   56.0  2.408  2024-11-12     11  2024

  time_of_day
0    14:31:59
1    13:48:50
2    09:24:26
3    08:30:35
4    08:13:20
```

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.optim as optim
```

[ ]:
```
df = pd.DataFrame(timed_data[['latitude', 'longitude', 'depth', 'mag', 'gap',␣
 ↪'dmin', 'time']])
df['datetime'] = pd.to_datetime(initial_data['time'])
df['year'] = df['datetime'].dt.year
df['month'] = df['datetime'].dt.month
df['day'] = df['datetime'].dt.day
df.drop(columns=['time', 'datetime'], inplace=True)
df.head()
```

[ ]:
```
   latitude  longitude    depth  mag    gap   dmin  year  month  day
0   -4.5205   152.4422   89.123  5.3   62.0  0.429  2024     11   12
1   36.3155   141.2801   38.636  4.5  137.0  2.487  2024     11   12
2    0.5833   126.1828   17.810  5.2   46.0  1.198  2024     11   12
3   27.3495    88.3392   10.000  4.4  145.0  2.753  2024     11   12
4   37.5859   135.4869  358.970  4.4   56.0  2.408  2024     11   12
```

[ ]:
```
features = ['latitude', 'longitude', 'depth', 'gap', 'dmin', 'day' ]
target = 'mag'

# Scale features and target
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[features + [target]])
```

```
# Convert to DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=features + [target])
```

[ ]:
```
def create_sequences(data, feature_columns, target_column, seq_length):
    sequences = []
    targets = []
    for i in range(len(data) - seq_length):
        seq = df[feature_columns].iloc[i:i + seq_length].values  # Only use␣
↪feature columns
        label = df[target_column].iloc[i + seq_length]  # Use the target column␣
↪for labels
        sequences.append(seq)
        targets.append(label)
    return np.array(sequences), np.array(targets)
```

[ ]:
```
seq_length = 10  # Number of previous timesteps to consider
X, y = create_sequences(scaled_df, feature_columns=features,␣
↪target_column=target, seq_length=seq_length)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
```

[ ]:
```
class EarthquakeLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size,␣
↪dropout=0.2):
        super(EarthquakeLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,␣
↪batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.lstm(x)  # LSTM outputs
        out = self.fc(out[:, -1, :])  # Fully connected layer on the last time␣
↪step
        return out
```

[ ]:
```
input_size = len(features)
hidden_size = 128  # Increased hidden size
num_layers = 3  # Increased number of layers
output_size = 1
```

```
model = EarthquakeLSTM(input_size, hidden_size, num_layers, output_size)
```

[ ]:
```
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0005)
```

[ ]:
```
num_epochs = 100
for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    # Backpropagation and optimization
    optimizer.zero_grad()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()


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

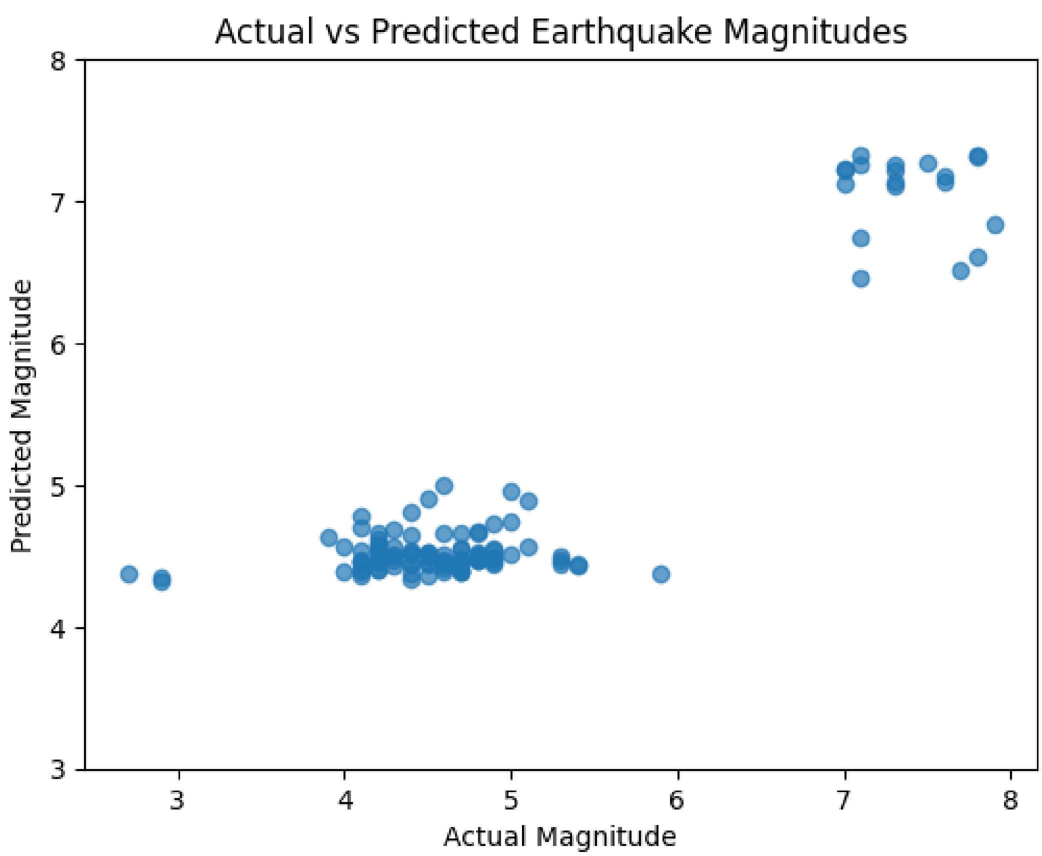
```
Epoch [10/100], Loss: 17.2436
Epoch [20/100], Loss: 2.8668
Epoch [30/100], Loss: 1.0277
Epoch [40/100], Loss: 0.9089
Epoch [50/100], Loss: 0.8981
Epoch [60/100], Loss: 0.8454
Epoch [70/100], Loss: 0.6729
Epoch [80/100], Loss: 0.4221
Epoch [90/100], Loss: 0.2793
Epoch [100/100], Loss: 0.2187
```

[ ]:
```
model.eval()  # Set model to evaluation mode
with torch.no_grad():
    predictions = model(X_test_tensor)
    test_loss = criterion(predictions, y_test_tensor)
    print(f'Mean Squared Error: {test_loss.item()*100:.2f}')

    y_test_numpy = y_test_tensor.numpy()
    predictions_numpy = predictions.numpy()

    # Calculate R-squared
    ss_total = ((y_test_numpy - np.mean(y_test_numpy)) ** 2).sum()  # Total sum␣
 ↪of squares
```

```
    ss_residual = ((y_test_numpy - predictions_numpy) ** 2).sum()  # Residual␣
↪sum of squares
    r_squared = 1 - (ss_residual / ss_total)
    print(f'R-squared: {r_squared*100:.2f}')
```

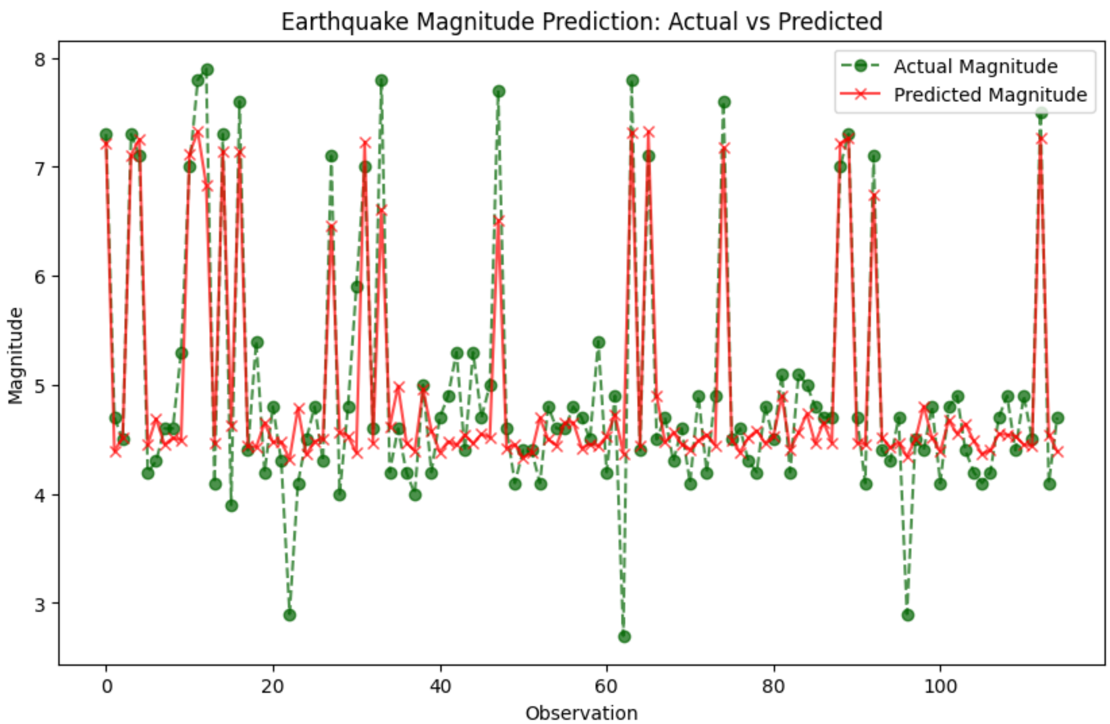
```
Mean Squared Error: 23.34
R-squared: 82.56
```

[ ]:
```
import matplotlib.pyplot as plt
plt.scatter(y_test, predictions.numpy(), alpha=0.7)
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title('Actual vs Predicted Earthquake Magnitudes')
plt.ylim(3, 8)  # Set y-axis range from 3 to 8
plt.show()
```




[ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual Magnitude', marker='o', color='darkgreen',␣
↪linestyle='--', alpha=0.7)
```

```
plt.plot(predictions, label='Predicted Magnitude', marker='x', color='r',␣
 ↪linestyle='-', alpha=0.7)
plt.xlabel('Observation')
plt.ylabel('Magnitude')
plt.title('Earthquake Magnitude Prediction: Actual vs Predicted')
plt.legend()
plt.show()
```




[ ]: