EARTHQUAKE PREDICTION MODEL USING PYTHON

AI_PHASE 4 DEVELOPMENT PART 2

NAME: SATHISHKUMAR M

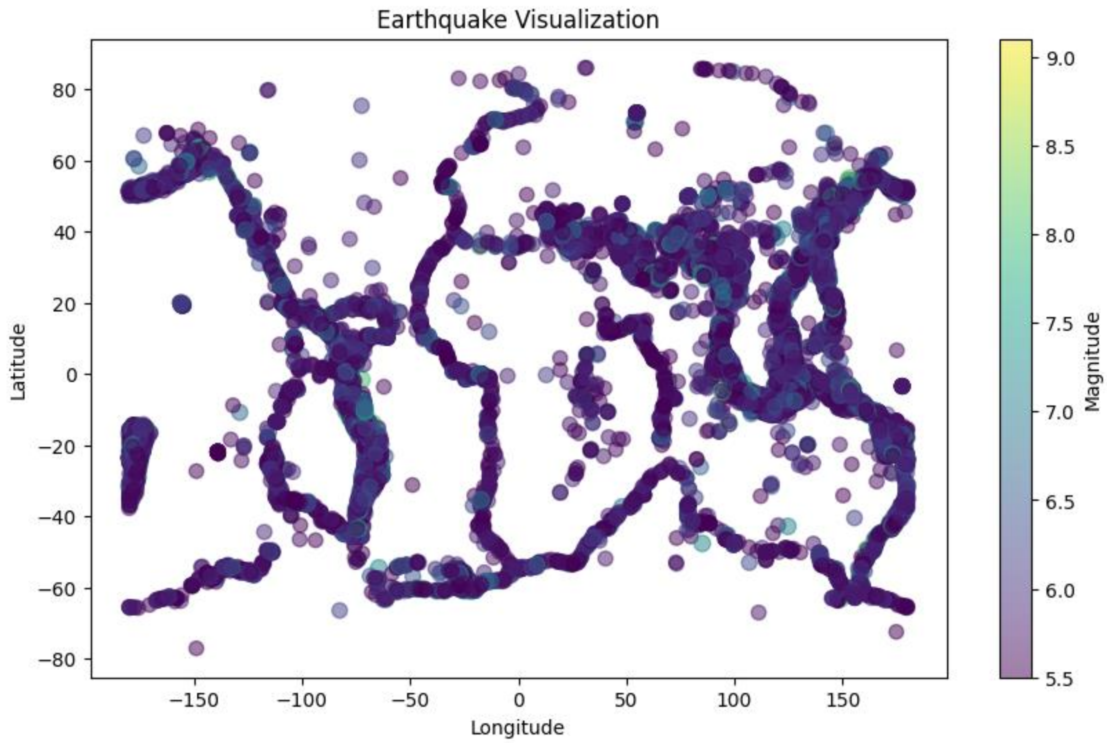
REG.NO: 610821205307

VISUALIZATION

```
# Extract latitude, longitude, and magnitude columns
latitude = data['Latitude']
longitude = data['Longitude']
magnitude = data['Magnitude']

# Create a scatter plot to visualize earthquakes on a map
plt.figure(figsize=(10, 6))
plt.scatter(longitude, latitude, c=magnitude, cmap='viridis', s=magnitude *
10, alpha=0.5)
plt.colorbar(label='Magnitude')
plt.title('Earthquake Visualization')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

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



FEATURE ENGINEERING

```
import numpy as np
import pandas as pd

# Synthetic seismic data (replace this with your real data)
data = pd.DataFrame({
    'time': np.arange(0, 100, 0.1),
    'acceleration': np.random.rand(1000),
    # Add more columns for other sensor data if available
})

# Define functions for feature engineering
def basic_statistics(data):
    # Calculate basic statistical features
    features = {
        'mean': data['acceleration'].mean(),
        'std_dev': data['acceleration'].std(),
        'min': data['acceleration'].min(),
        'max': data['acceleration'].max(),
    }
    return features

def time_domain_features(data):
    # Calculate time domain features
    # For example, root mean square (RMS) amplitude
```

```python
    rms = np.sqrt(np.mean(data['acceleration']**2))
    return {'RMS_amplitude': rms}

def frequency_domain_features(data):
    # Calculate frequency domain features using Fourier transform
    fft_result = np.fft.fft(data['acceleration'])
    # Extract amplitude and frequency information
    amplitude = np.abs(fft_result)
    frequency = np.fft.fftfreq(len(fft_result))
    # Find the dominant frequency component
    dominant_frequency = frequency[np.argmax(amplitude)]
    return {'dominant_frequency': dominant_frequency}

# Apply feature engineering functions to your data
statistical_features = basic_statistics(data)
time_domain_features = time_domain_features(data)
frequency_domain_features = frequency_domain_features(data)

# Combining all features into a single feature vector
feature_vector = {**statistical_features, **time_domain_features,
**frequency_domain_features}

# Your feature vector is now ready for use in training your earthquake
prediction model
print(feature_vector)
```

```
{'mean': 0.49201126044541615, 'std_dev': 0.29111111319763416, 'min':
0.0012840627090102696, 'max': 0.9997457913830645, 'RMS_amplitude':
0.5716082705420084, 'dominant_frequency': 0.0}
```

MODEL TRAINING

```python
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Splitting 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)

# Creating a Random Forest regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Training the model on the training data
model.fit(X_train, y_train)
```

```
# Making predictions on the test data
y_pred = model.predict(X_test)

# Evaluating the model's performance (for regression tasks)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 968.4488974862994
```

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, X_test.shape)
```

```
(18727, 3) (4682, 3) (18727, 2) (4682, 3)
```

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

```
array([[  5.865,  42.024],
       [  5.826,  33.09 ],
       [  6.082,  39.741],
       ...,
       [  6.306,  23.059],
       [  5.96 , 592.283],
       [  5.808,  38.222]])
```

```
reg.score(X_test, y_test)
```

```
0.3926671400442392
```

```
from sklearn.model_selection import GridSearchCV

parameters = {'n_estimators':[10, 20, 50, 100, 200, 500]}

grid_obj = GridSearchCV(reg, parameters)
grid_fit = grid_obj.fit(X_train, y_train)
best_fit = grid_fit.best_estimator_
best_fit.predict(X_test)
```

```
array([[  5.886 ,  43.031 ],
       [  5.82  ,  31.3982],
       [  6.0124,  39.5216],
       ...,
       [  6.294 ,  22.9908],
```

```
       [  5.9218, 592.385 ],
       [  5.7894,  39.2764]])
```

NEURAL NETWORK MODEL

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Generate synthetic seismic and geological data (replace with real data)
n_samples = 1000
n_features = 10

X = np.random.rand(n_samples, n_features)
y = np.random.randint(2, size=n_samples)

 # Binary labels (0: no earthquake, 1: earthquake)

# Feature engineering and preprocessing (replace with actual preprocessing
steps)
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Splitting 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)

# Building a basic feedforward neural network model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Training the model
model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))
```

MODEL EVALUATION

```
# Evaluating the model on the test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

```
Epoch 1/50
25/25 [==============================] - 1s 13ms/step - loss: 0.7015 -
accuracy: 0.5000 - val_loss: 0.7059 - val_accuracy: 0.4400
Epoch 2/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6850 -
accuracy: 0.5425 - val_loss: 0.7107 - val_accuracy: 0.4600
Epoch 3/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6771 -
accuracy: 0.5813 - val_loss: 0.7109 - val_accuracy: 0.4550
Epoch 4/50
25/25 [==============================] - 0s 8ms/step - loss: 0.6713 -
accuracy: 0.5738 - val_loss: 0.7097 - val_accuracy: 0.4600
Epoch 5/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6646 -
accuracy: 0.6075 - val_loss: 0.7112 - val_accuracy: 0.4400
Epoch 6/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6595 -
accuracy: 0.5938 - val_loss: 0.7147 - val_accuracy: 0.4550
Epoch 7/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6534 -
accuracy: 0.6162 - val_loss: 0.7126 - val_accuracy: 0.4500
Epoch 8/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6470 -
accuracy: 0.6250 - val_loss: 0.7141 - val_accuracy: 0.4400
Epoch 9/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6420 -
accuracy: 0.6313 - val_loss: 0.7181 - val_accuracy: 0.4650
Epoch 10/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6358 -
accuracy: 0.6313 - val_loss: 0.7191 - val_accuracy: 0.4650
Epoch 11/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6259 -
accuracy: 0.6662 - val_loss: 0.7170 - val_accuracy: 0.4850
Epoch 12/50
25/25 [==============================] - 0s 5ms/step - loss: 0.6199 -
accuracy: 0.6712 - val_loss: 0.7249 - val_accuracy: 0.4850
Epoch 13/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6113 -
accuracy: 0.6900 - val_loss: 0.7219 - val_accuracy: 0.4950
Epoch 14/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6047 -
accuracy: 0.7025 - val_loss: 0.7189 - val_accuracy: 0.5000
Epoch 15/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5968 -
accuracy: 0.7075 - val_loss: 0.7280 - val_accuracy: 0.4850
Epoch 16/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5878 -
accuracy: 0.7262 - val_loss: 0.7263 - val_accuracy: 0.4700
Epoch 17/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5785 -
```

```
accuracy: 0.7250 - val_loss: 0.7319 - val_accuracy: 0.4800
Epoch 18/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5697 -
accuracy: 0.7362 - val_loss: 0.7417 - val_accuracy: 0.4650
Epoch 19/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5598 -
accuracy: 0.7538 - val_loss: 0.7333 - val_accuracy: 0.4900
Epoch 20/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5501 -
accuracy: 0.7650 - val_loss: 0.7402 - val_accuracy: 0.4650
Epoch 21/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5408 -
accuracy: 0.7638 - val_loss: 0.7440 - val_accuracy: 0.4650
Epoch 22/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5300 -
accuracy: 0.7763 - val_loss: 0.7530 - val_accuracy: 0.4700
Epoch 23/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5202 -
accuracy: 0.7800 - val_loss: 0.7539 - val_accuracy: 0.4950
Epoch 24/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5113 -
accuracy: 0.8000 - val_loss: 0.7617 - val_accuracy: 0.4850
Epoch 25/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5043 -
accuracy: 0.7900 - val_loss: 0.7582 - val_accuracy: 0.4850
Epoch 26/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4944 -
accuracy: 0.7937 - val_loss: 0.7634 - val_accuracy: 0.4950
Epoch 27/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4820 -
accuracy: 0.8175 - val_loss: 0.7699 - val_accuracy: 0.4800
Epoch 28/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4719 -
accuracy: 0.8150 - val_loss: 0.7832 - val_accuracy: 0.4750
Epoch 29/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4603 -
accuracy: 0.8263 - val_loss: 0.7937 - val_accuracy: 0.4800
Epoch 30/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4600 -
accuracy: 0.8250 - val_loss: 0.7819 - val_accuracy: 0.5100
Epoch 31/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4462 -
accuracy: 0.8225 - val_loss: 0.8214 - val_accuracy: 0.4800
Epoch 32/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4316 -
accuracy: 0.8450 - val_loss: 0.8068 - val_accuracy: 0.4800
Epoch 33/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4133 -
accuracy: 0.8587 - val_loss: 0.8268 - val_accuracy: 0.4600
Epoch 34/50
```

```
25/25 [==============================] - 0s 4ms/step - loss: 0.4042 -
accuracy: 0.8675 - val_loss: 0.8109 - val_accuracy: 0.4750
Epoch 35/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3955 -
accuracy: 0.8775 - val_loss: 0.8383 - val_accuracy: 0.4850
Epoch 36/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3863 -
accuracy: 0.8900 - val_loss: 0.8515 - val_accuracy: 0.5000
Epoch 37/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3807 -
accuracy: 0.8725 - val_loss: 0.8455 - val_accuracy: 0.5000
Epoch 38/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3745 -
accuracy: 0.8863 - val_loss: 0.8584 - val_accuracy: 0.5000
Epoch 39/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3564 -
accuracy: 0.9000 - val_loss: 0.8744 - val_accuracy: 0.4750
Epoch 40/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3484 -
accuracy: 0.9137 - val_loss: 0.8772 - val_accuracy: 0.4850
Epoch 41/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3338 -
accuracy: 0.9000 - val_loss: 0.8788 - val_accuracy: 0.5050
Epoch 42/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3304 -
accuracy: 0.9000 - val_loss: 0.9459 - val_accuracy: 0.4600
Epoch 43/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3270 -
accuracy: 0.8988 - val_loss: 0.9118 - val_accuracy: 0.4950
Epoch 44/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3149 -
accuracy: 0.9275 - val_loss: 0.9395 - val_accuracy: 0.4400
Epoch 45/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3010 -
accuracy: 0.9237 - val_loss: 0.9541 - val_accuracy: 0.4600
Epoch 46/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2908 -
accuracy: 0.9262 - val_loss: 0.9331 - val_accuracy: 0.4650
Epoch 47/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2813 -
accuracy: 0.9350 - val_loss: 0.9438 - val_accuracy: 0.4900
Epoch 48/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2721 -
accuracy: 0.9425 - val_loss: 0.9695 - val_accuracy: 0.4850
Epoch 49/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2704 -
accuracy: 0.9337 - val_loss: 1.0072 - val_accuracy: 0.4450
Epoch 50/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2602 -
accuracy: 0.9425 - val_loss: 0.9908 - val_accuracy: 0.4700
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.9908 - accuracy:
0.4700
Test Loss: 0.9907826781272888, Test Accuracy: 0.4699999988079071
```