AI Earthquake Prediction System Project Report

Team Members:5

Team Leader:Parthasarathy.S

Team Members:

- Kiruba.K
- Parthasarathy.S
- Parthasarathy.M
- Mahendran.T
- sivenesan.N

Problem Statement:

- Earthquakes are natural disasters that can cause widespread destruction and loss of life. Predicting earthquakes accurately can significantly reduce their impact by enabling early warnings, preparation, and timely response. However, earthquakes are complex phenomena, and their prediction is challenging due to the uncertainties involved. A predictive model using AI and machine learning could potentially aid in forecasting earthquakes, allowing authorities to take preventive measures and potentially save lives and resources.

Design Thinking:

Empathize: * Understand the impact of earthquakes on communities, the significance of early warning systems, and the challenges faced by seismologists in predicting earthquakes accurately.

Define: * Clearly define the problem and its scope. Determine the key factors affecting earthquake prediction, including seismic data, historical patterns, geographic features, and various other parameters that influence seismic activity.

Ideate: * Generate ideas and strategies for creating an AI-based earthquake prediction model. This involves brainstorming on various machine learning algorithms, data sources, feature engineering, and potential variables that could contribute to accurate predictions.

Prototype: * Develop a prototype model using Python that leverages machine learning algorithms to analyze seismic data and predict earthquakes. The model should be based on historical earthquake data and features derived from seismic readings.

Test: * Evaluate the prototype model's accuracy, precision, and recall by comparing its predictions against known earthquake events. Refine the model based on the test results and iterate to enhance its predictive capabilities.

Implement: * Develop a user-friendly interface to deliver predictions in real-time. Collaborate with relevant authorities to integrate the model into existing early warning systems or disaster management protocols.

Phases of Development:

Phase development:

Data Collection: * Gather seismic data from various sources, including seismographs, satellites, and geological surveys. This data may contain information on seismic activity, fault lines, geographic features, and historical earthquake records.

Data Preprocessing: * Clean the data, handle missing values, normalize or scale features, and prepare the dataset for model training. This phase involves data exploration and feature selection to identify the most relevant parameters for prediction.

Feature Engineering: * Create new features or transform existing ones that can improve the model's predictive capability. This might involve extracting statistical features from seismic signals, considering geographical attributes, or incorporating historical seismic patterns.

Model Selection: * Choose appropriate machine learning algorithms such as Random Forest, Support Vector Machines, Neural Networks, or Gradient Boosting, and train these models using the preprocessed data.

Model Evaluation: * Evaluate the models using metrics like accuracy, precision, recall, and F1 score. Use techniques such as cross-validation to ensure the model's robustness.

Model Optimization: * Fine-tune the model by adjusting hyperparameters, exploring ensemble methods, or applying regularization techniques to enhance its predictive performance.

Deployment: * Create an application or system that integrates the trained model, allowing for real-time predictions and alerts. Ensure scalability and reliability of the system for continuous monitoring and updates.

Monitoring and Maintenance: * Continuously monitor the model's performance and make necessary updates or improvements based on new data and feedback. Regular maintenance ensures the model stays effective and reliable over time.

Further Elaboration:

Data Collection and Processing:

- Sources of Data: Gather data from seismic stations, geological surveys, satellite observations, and historical earthquake records. This might include information about seismic waves, fault lines, tectonic plate movement, and various geological attributes. Data Cleaning and Integration: Address missing values, outliers, and inconsistencies in the data. Integrate diverse datasets, ensuring compatibility for analysis. Feature Engineering:

Feature Extraction: * Extract relevant features from the seismic data, which could involve statistical characteristics of seismic signals, geographic properties, historical earthquake patterns, or spectral analysis. Dimensionality Reduction: Implement techniques like PCA (Principal Component Analysis) or feature selection methods to reduce the dimensionality of data while preserving crucial information. Model Development:

Algorithm Selection: * Experiment with various machine learning algorithms such as Random Forest, Support Vector Machines (SVM), Long Short-Term Memory (LSTM) networks, or Convolutional Neural Networks (CNN). Model Training: Split the data into training and validation sets. Train the chosen models on the training data and fine-tune them using validation sets. Evaluation and Validation:

Cross-Validation: * Employ techniques like k-fold cross-validation to evaluate the models' performance on different subsets of data. Performance Metrics: Assess the model's accuracy, precision,recall, F1 score, and area under the ROC curve to understand its effectiveness. Model Optimization:

Hyperparameter Tuning: * Optimize model parameters to enhance performance using methods like grid search, random search, or Bayesian optimization. Ensemble Methods: Consider ensemble techniques to combine predictions from multiple models, improving accuracy and robustness.

Real-time Implementation: **User Interface Development**: * Create an intuitive user interface for accessing predictions or integrating the model into existing earthquake monitoring systems.

API Integration: * Develop an API that allows other systems to interact with the prediction model, enabling real-time alerts and updates. Deployment and Continuous Improvement:

Deployment Strategy: * Choose a suitable deployment environment considering scalability and accessibility. Feedback Loop: Gather feedback on model predictions and performance, incorporating it into continuous model improvement.

Ethical and Regulatory Considerations: * Ensure transparency and accountability in model predictions, emphasizing the limitations and uncertainties involved in earthquake forecasting. Adhere to ethical standards and privacy protocols in data handling and model deployment.

Collaboration and Knowledge Sharing:

- Collaborate with scientists, seismologists, and disaster management authorities to validate the model's predictions and integrate it into real-world applications. Share knowledge, methodologies, and findings with the scientific community for further research and development. Long-term Maintenance:
- Regularly update the model with new data to adapt to changing seismic patterns and technological advancements. Maintain and support the system to ensure its functionality over an extended period.

##Import Dataset

[ ]:
```
import pandas as pd
from google.colab import data_table
data_table.enable_dataframe_formatter()

# Read CSV file with space delimiter
df = pd.read_csv('/content/Earthquake_Data.csv', delimiter=r'\s+')

# Print the first 5 rows of the data frame
display(df)
```

```
      Date(YYYY/MM/DD)         Time  Latitude  Longitude  Depth   Mag Magt  \
0           1966/07/01  09:41:21.82   35.9463  -120.4700  12.26  3.20   Mx
1           1966/07/02  12:08:34.25   35.7867  -120.3265   8.99  3.70   Mx
2           1966/07/02  12:16:14.95   35.7928  -120.3353   9.88  3.40   Mx
3           1966/07/02  12:25:06.12   35.7970  -120.3282   9.09  3.10   Mx
4           1966/07/05  18:54:54.36   35.9223  -120.4585   7.86  3.10   Mx
...                ...          ...       ...        ...    ...   ...  ...
18025       2007/12/19  12:14:09.62   34.1438  -116.9822   7.03  4.06   ML
18026       2007/12/21  12:14:56.45   37.3078  -121.6735   8.47  3.08   ML
```

```
18027         2007/12/23 21:43:43.54    37.2127  -117.8230  10.00   3.54     ML
18028         2007/12/28 01:59:42.40    36.5292  -121.1133   5.99   3.04     ML
18029         2007/12/28 23:20:28.12    38.7710  -122.7370   2.34   3.40     Mw

       Nst  Gap  Clo   RMS   SRC    EventID
0        7  171   20  0.02  NCSN  -4540462
1        8   86    3  0.04  NCSN  -4540520
2        8   89    2  0.03  NCSN  -4540521
3        8  101    3  0.08  NCSN  -4540522
4        9  161   14  0.04  NCSN  -4540594
...    ...  ...  ...   ...   ...       ...
18025   10   73   14  0.08  NCSN  40207706
18026  114   45    5  0.12  NCSN  51192926
18027   45  176   40  0.07  NCSN  51193070
18028   70   45    4  0.06  NCSN  51193343
18029   49   37    1  0.07  NCSN  51193419

[18030 rows x 13 columns]
```

##Preprocessing No preprocessing required because the data is already clean and structured. We just have to change the column names to meaningful names.

```
[ ]: new_column_names = ["Date(YYYY/MM/DD)",      "Time(UTC)",   "Latitude(deg)",
     "Longitude(deg)", "Depth(km)", "Magnitude(ergs)",
                          "Magnitude_type",  "No_of_Stations",  "Gap",  "Close",  "RMS",
     "SRC", "EventID"]

     df.columns = new_column_names
     ts = pd.to_datetime(df["Date(YYYY/MM/DD)"] + " " + df["Time(UTC)"])
     df = df.drop(["Date(YYYY/MM/DD)", "Time(UTC)"], axis=1)
     df.index = ts
     display(df)
```

```
                          Latitude(deg)  Longitude(deg)  Depth(km)  \
1966-07-01 09:41:21.820        35.9463       -120.4700      12.26
1966-07-02 12:08:34.250        35.7867       -120.3265       8.99
1966-07-02 12:16:14.950        35.7928       -120.3353       9.88
1966-07-02 12:25:06.120        35.7970       -120.3282       9.09
1966-07-05 18:54:54.360        35.9223       -120.4585       7.86
...                                ...             ...        ...
2007-12-19 12:14:09.620        34.1438       -116.9822       7.03
2007-12-21 12:14:56.450        37.3078       -121.6735       8.47
2007-12-23 21:43:43.540        37.2127       -117.8230      10.00
2007-12-28 01:59:42.400        36.5292       -121.1133       5.99
2007-12-28 23:20:28.120        38.7710       -122.7370       2.34

                         Magnitude(ergs) Magnitude_type  No_of_Stations  Gap  \
1966-07-01 09:41:21.820             3.20             Mx               7  171
```

```
1966-07-02  12:08:34.250           3.70           Mx           8   86
1966-07-02  12:16:14.950           3.40           Mx           8   89
1966-07-02  12:25:06.120           3.10           Mx           8  101
1966-07-05  18:54:54.360           3.10           Mx           9  161
...                                 ...     ...          ...  ...
2007-12-19  12:14:09.620           4.06           ML          10   73
2007-12-21  12:14:56.450           3.08           ML         114   45
2007-12-23  21:43:43.540           3.54           ML          45  176
2007-12-28  01:59:42.400           3.04           ML          70   45
2007-12-28  23:20:28.120           3.40           Mw          49   37

                         Close   RMS   SRC   EventID
1966-07-01  09:41:21.820    20  0.02  NCSN  -4540462
1966-07-02  12:08:34.250     3  0.04  NCSN  -4540520
1966-07-02  12:16:14.950     2  0.03  NCSN  -4540521
1966-07-02  12:25:06.120     3  0.08  NCSN  -4540522
1966-07-05  18:54:54.360    14  0.04  NCSN  -4540594
...                        ...   ...   ...       ...
2007-12-19  12:14:09.620    14  0.08  NCSN  40207706
2007-12-21  12:14:56.450     5  0.12  NCSN  51192926
2007-12-23  21:43:43.540    40  0.07  NCSN  51193070
2007-12-28  01:59:42.400     4  0.06  NCSN  51193343
2007-12-28  23:20:28.120     1  0.07  NCSN  51193419

[18030 rows x 11 columns]
```

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

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 18030 entries, 1966-07-01 09:41:21.820000 to 2007-12-28 23:20:28.120000
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Latitude(deg)     18030 non-null  float64
 1   Longitude(deg)    18030 non-null  float64
 2   Depth(km)         18030 non-null  float64
 3   Magnitude(ergs)   18030 non-null  float64
 4   Magnitude_type    18030 non-null  object
 5   No_of_Stations    18030 non-null  int64
 6   Gap               18030 non-null  int64
 7   Close             18030 non-null  int64
 8   RMS               18030 non-null  float64
 9   SRC               18030 non-null  object
 10  EventID           18030 non-null  int64
dtypes: float64(5), int64(4), object(2)
memory usage: 1.7+ MB
```

##Export Preprocessed dataset Export the data into xlsx file

```
[ ]: file_name = 'Earthquake_data_processed.xlsx'

     # saving the excel
     df.to_excel(file_name)
     print('DataFrame is written to Excel File successfully.')
```

DataFrame is written to Excel File successfully.

```
[ ]: import warnings
     warnings.filterwarnings('ignore')
```

##Partition the data into Training and Testing data

```
[ ]: from sklearn.model_selection import train_test_split

     # Select relevant columns
     X = df[['Latitude(deg)', 'Longitude(deg)', 'Depth(km)', 'No_of_Stations']]
     y = df['Magnitude(ergs)']

     # Split 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=0)
```

##Linear regression

Loading the model and fitting it with training data

```
[ ]: from sklearn.linear_model import LinearRegression

     # Train the linear regression model
     regressor = LinearRegression()
     regressor.fit(X_train, y_train)
```

```
[ ]: LinearRegression()
```

Predict the testing data

Find the predicted values and evaluate it using metrics of linear regression

```
[ ]: from sklearn.metrics import r2_score, mean_squared_error

     scores= {"Model name": ["Linear regression", "SVM", "Random Forest"], "mse":␣
      ↪[], "R^2": []}

     # Predict on the testing set
     y_pred = regressor.predict(X_test)

     # Compute R^2 and MSE
```

```
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

scores['mse'].append(mse)
scores['R^2'].append(r2)

print("R^2: {:.2f}, MSE: {:.2f}".format(r2, mse))
```

```
R^2: 0.03, MSE: 0.18
```

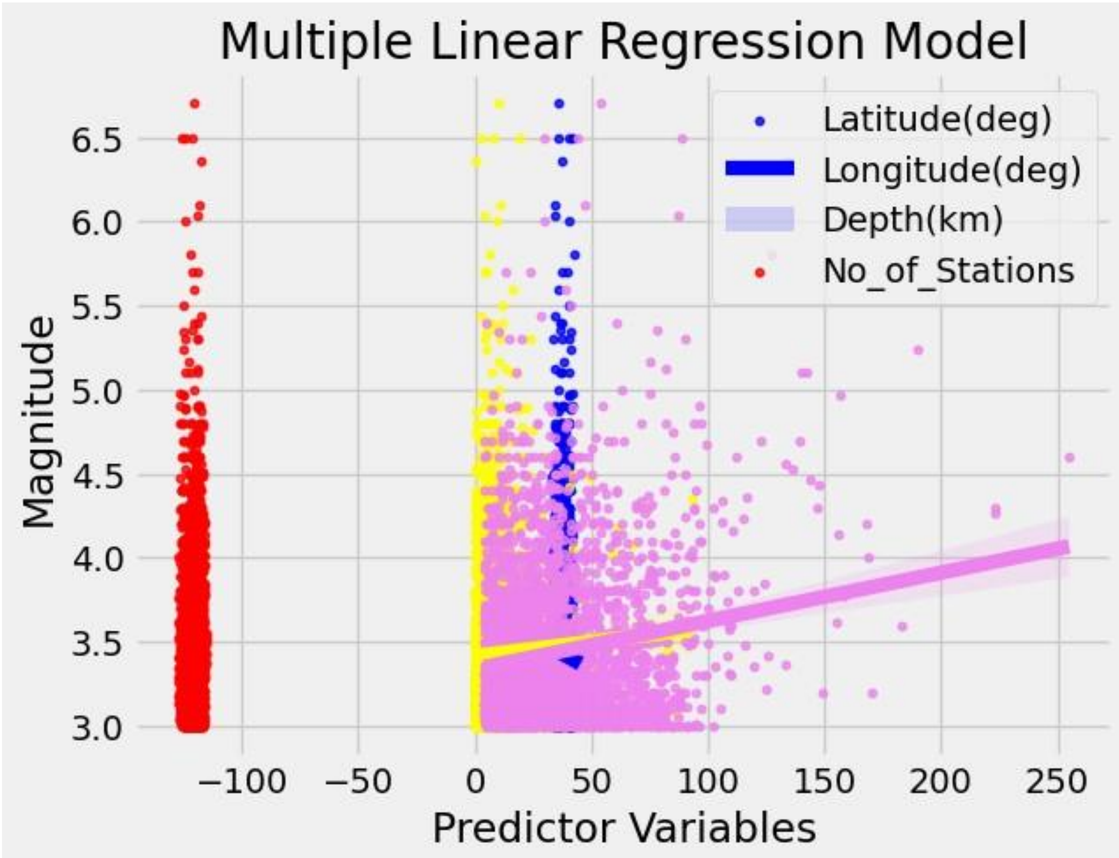
Predict for new data

[ ]:
```
# Predict on new data
new_data = [[33.89, -118.40, 16.17, 11], [37.77, -122.42, 8.05, 14]]
new_pred = regressor.predict(new_data)
print("New predictions:", new_pred)
```

```
New predictions: [3.447483   3.33027751]
```

Plot multiple linear regression model

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the regression line
sns.regplot(x=X_test['Latitude(deg)'], y=y_test, color='blue', scatter_kws={'s':
  10})
sns.regplot(x=X_test['Longitude(deg)'], y=y_test, color='red', scatter_kws={'s':
  10})
sns.regplot(x=X_test['Depth(km)'], y=y_test, color='yellow', scatter_kws={'s':
  10})
sns.regplot(x=X_test['No_of_Stations'], y=y_test, color='violet',
  scatter_kws={'s': 10})
plt.legend(labels=['Latitude(deg)', 'Longitude(deg)', 'Depth(km)',
  'No_of_Stations'])
plt.xlabel('Predictor Variables')
plt.ylabel('Magnitude')
plt.title('Multiple Linear Regression Model')
plt.show()
```



##SVM

Loading the model and fitting it with training data

[ ]:
```
from sklearn.svm import SVR

# Select a subset of the training data
subset_size = 500
X_train_subset = X_train[:subset_size]
y_train_subset = y_train[:subset_size]

# Create an SVM model
svm = SVR(kernel='rbf', C=1e3, gamma=0.1)

# Train the SVM model on the subset of data
svm.fit(X_train_subset, y_train_subset)

# Evaluate the model on the test set
score = svm.score(X_test, y_test)
print("Test score:", score)
```

```
Test score: -1.9212973747969442
```

Predict the testing data

Find the predicted values and evaluate it using metrics like MSE, r2

```
[ ]: # Predict on the testing set
y_pred_svm = svm.predict(X_test)

# Compute R^2 and MSE
r2_svm = r2_score(y_test, y_pred_svm)
mse_svm = mean_squared_error(y_test, y_pred_svm)

scores['mse'].append(mse_svm)
scores['R^2'].append(r2_svm)

print("SVM R^2: {:.2f}, MSE: {:.2f}".format(r2_svm, mse_svm))
```

```
SVM R^2: -1.92, MSE: 0.53
```

Predict for new data

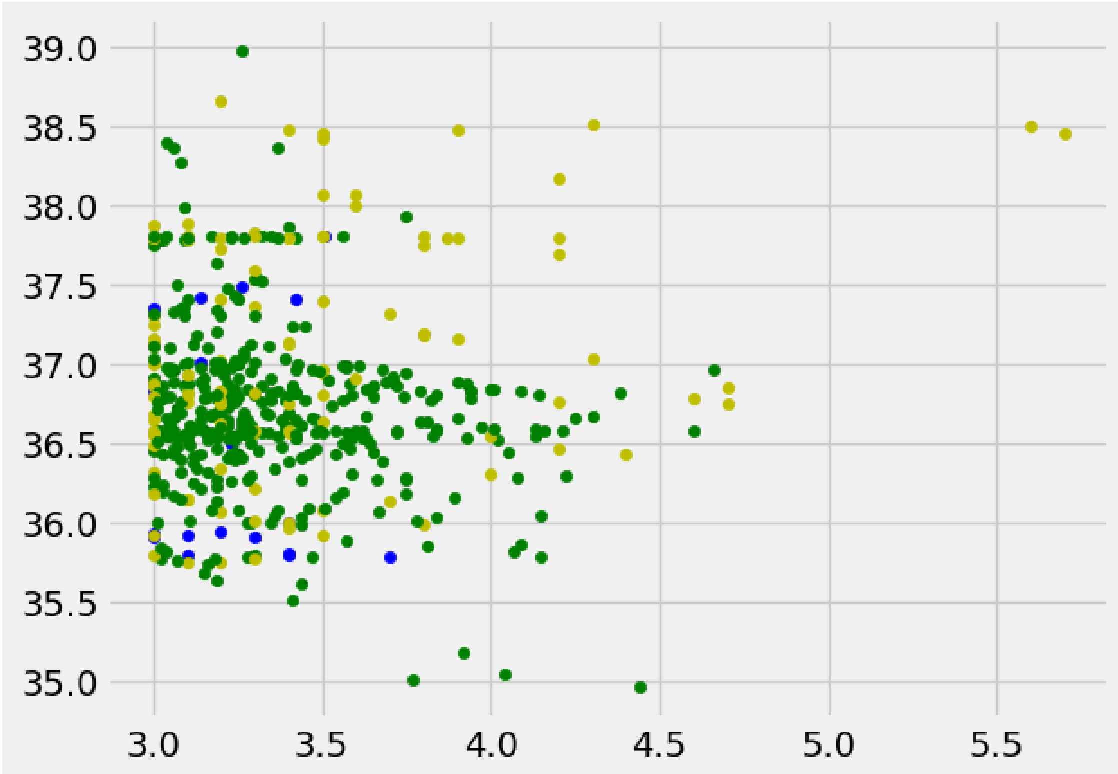
```
[ ]: # Predict on new data
new_pred_svm = svm.predict(new_data)
print("New SVM predictions:", new_pred_svm)
```

```
New SVM predictions: [3.57401976 3.03496212]
```

Plot model

```
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
from sklearn.svm import SVC

style.use('fivethirtyeight')

# create mesh grids
def make_meshgrid(x, y, h =.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

# plot the contours
def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
```

```
    return out

# color = ['y', 'b', 'g', 'k']

subset_size = 500

# modify the column names based on the dataset
features = df[['Magnitude(ergs)','Latitude(deg)']][:subset_size].values
classes = df['Magnitude_type'][:subset_size].values

# create 3 svm with rbf kernels
svm1 = SVC(kernel ='rbf')
svm2 = SVC(kernel ='rbf')
svm3 = SVC(kernel ='rbf')
svm4 = SVC(kernel ='rbf')

# fit each svm's
svm1.fit(features, (classes=='ML').astype(int))
svm2.fit(features, (classes=='Mx').astype(int))
svm3.fit(features, (classes=='Md').astype(int))

fig, ax = plt.subplots()
X0, X1 = features[:, 0], features[:, 1]
xx, yy = make_meshgrid(X0, X1)

# plot the contours
'''
plot_contours(ax, svm1, xx, yy, cmap = plt.get_cmap('hot'), alpha = 0.8)
plot_contours(ax, svm2, xx, yy, cmap = plt.get_cmap('hot'), alpha = 0.3)
plot_contours(ax, svm3, xx, yy, cmap = plt.get_cmap('hot'), alpha = 0.5)
'''
color = ['y', 'b', 'g', 'k', 'm']



for i in range(subset_size):
    if classes[i] == 'ML':
        plt.scatter(features[i][0], features[i][1], s = 20, c = color[0])
    elif classes[i] == 'Mx':
        plt.scatter(features[i][0], features[i][1], s = 20, c = color[1])
    elif classes[i] == 'Md':
        plt.scatter(features[i][0], features[i][1], s = 20, c = color[2])
    else:
        plt.scatter(features[i][0], features[i][1], s = 20, c = color[4])
plt.show()
```




```
[ ]: print(df.columns)
     df['Magnitude_type'].unique()
```

```
Index(['Latitude(deg)', 'Longitude(deg)', 'Depth(km)', 'Magnitude(ergs)',
       'Magnitude_type', 'No_of_Stations', 'Gap', 'Close', 'RMS', 'SRC',
       'EventID'],
      dtype='object')
```

```
[ ]: array(['Mx', 'ML', 'Md', 'Mw'], dtype=object)
```

##Naive Bayes

> **Note: Naive bayes is used for strings and numbers(categorically) it can be used for classification so it can be either 1 or 0 nothing in between like 0.5 (regression). Even if we force naive bayes and tweak it a little bit for regression the result is disappointing; A team experimented with this and achieve not so good results.**

This code is just for predicting categorical data magnitude type with Naive Bayes

```
[ ]: import pandas as pd
     import numpy as np
     from sklearn.naive_bayes import GaussianNB
     from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Read CSV file with space delimiter
df = pd.read_csv('/content/Earthquake_Data.csv', delimiter=r'\s+')

new_column_names = ["Date(YYYY/MM/DD)",     "Time(UTC)",  "Latitude(deg)",␣
↪"Longitude(deg)", "Depth(km)", "Magnitude",
                   "Magnitude_Category", "No_of_Stations", "Gap", "Close",␣
↪"RMS", "SRC", "EventID"]

df.columns = new_column_names

# Convert magnitude column to categorical data
df['Magnitude_Category'] = pd.cut(df['Magnitude'], bins=[0, 5, 6, 7, np.inf],␣
↪labels=['Minor', 'Moderate', 'Strong', 'Major'])

# Encode Magnitude Category
le = LabelEncoder()
df['Magnitude_Category_Encoded']  =  le.fit_transform(df['Magnitude_Category'])

# Normalize latitude and longitude values
scaler = MinMaxScaler()
df[['Latitude(deg)', 'Longitude(deg)']] = scaler.
↪fit_transform(df[['Latitude(deg)', 'Longitude(deg)']])

# Select features
X = df[['Latitude(deg)', 'Longitude(deg)', 'No_of_Stations']]
y = df['Magnitude_Category_Encoded']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Train the Gaussian Naive Bayes model on the training data
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

[ ]: GaussianNB()

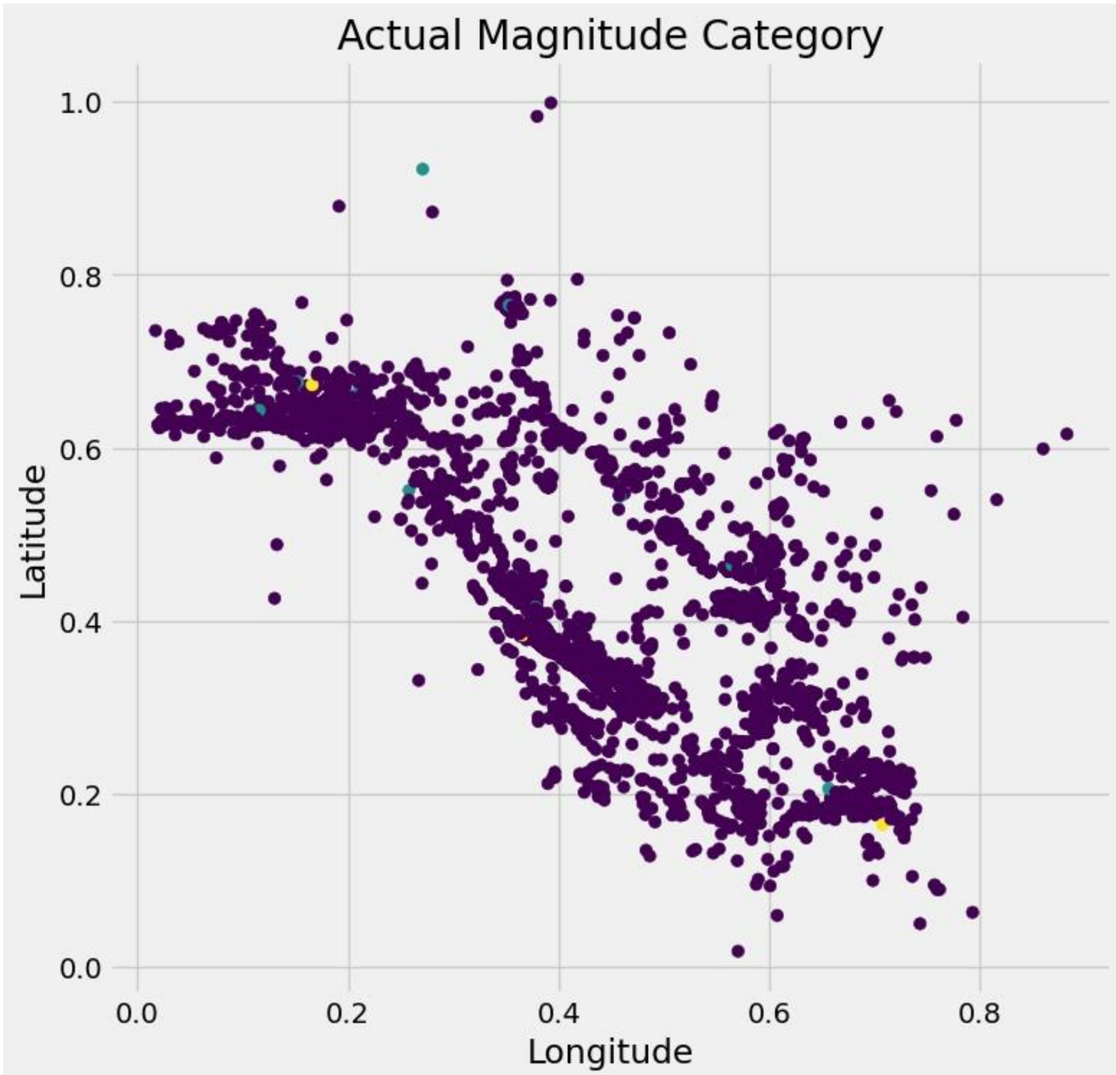
[ ]:
```
# Use the trained model to make predictions on the testing data
y_pred = gnb.predict(X_test)
```

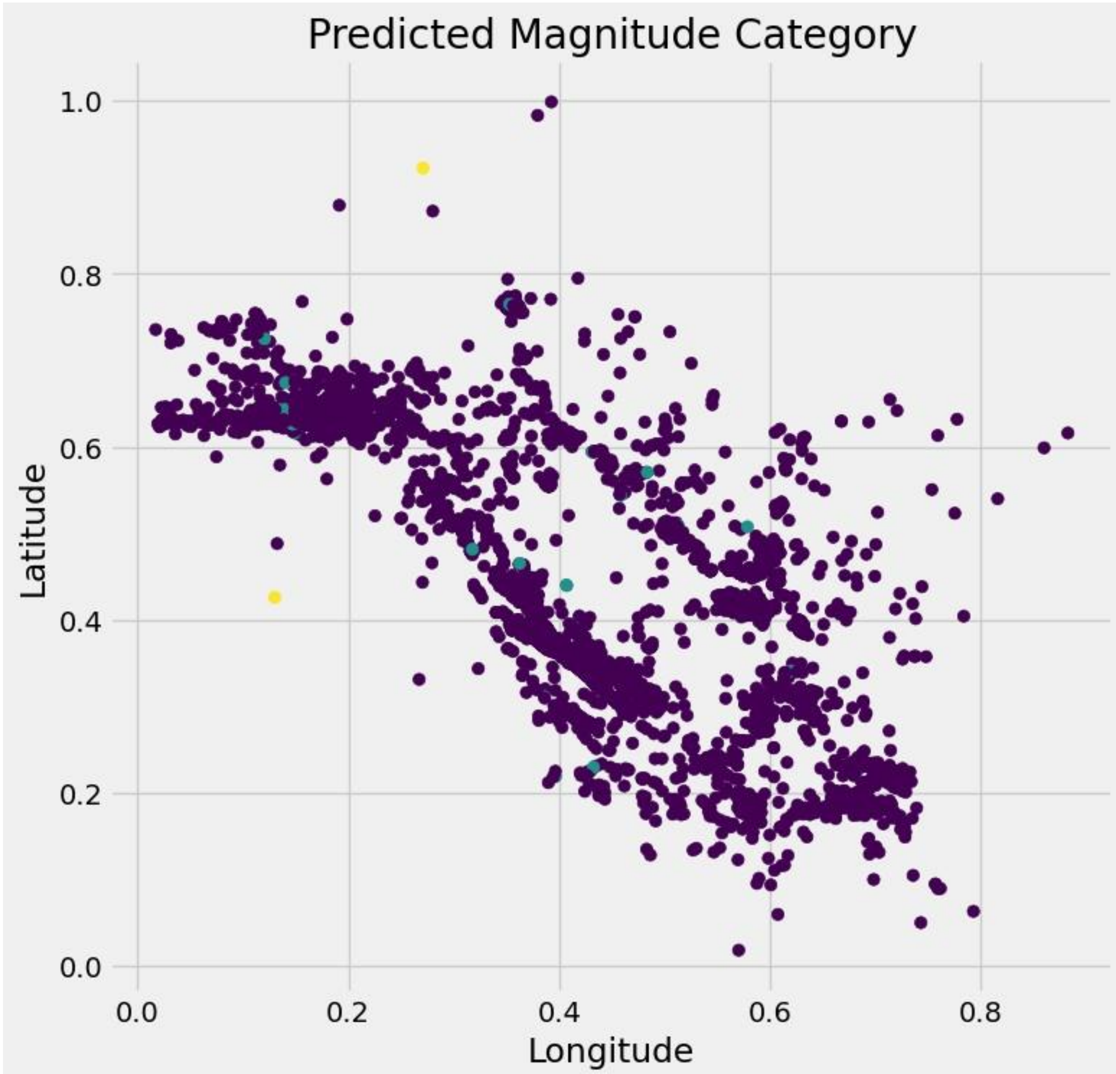
[ ]:
```python
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Calculate and print the confusion matrix and classification report
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', cm)

cr = classification_report(y_test, y_pred, labels=[0, 1, 2, 3],
    target_names=['Minor', 'Moderate', 'Strong', 'Major'])
print('Classification Report:\n', cr)
```

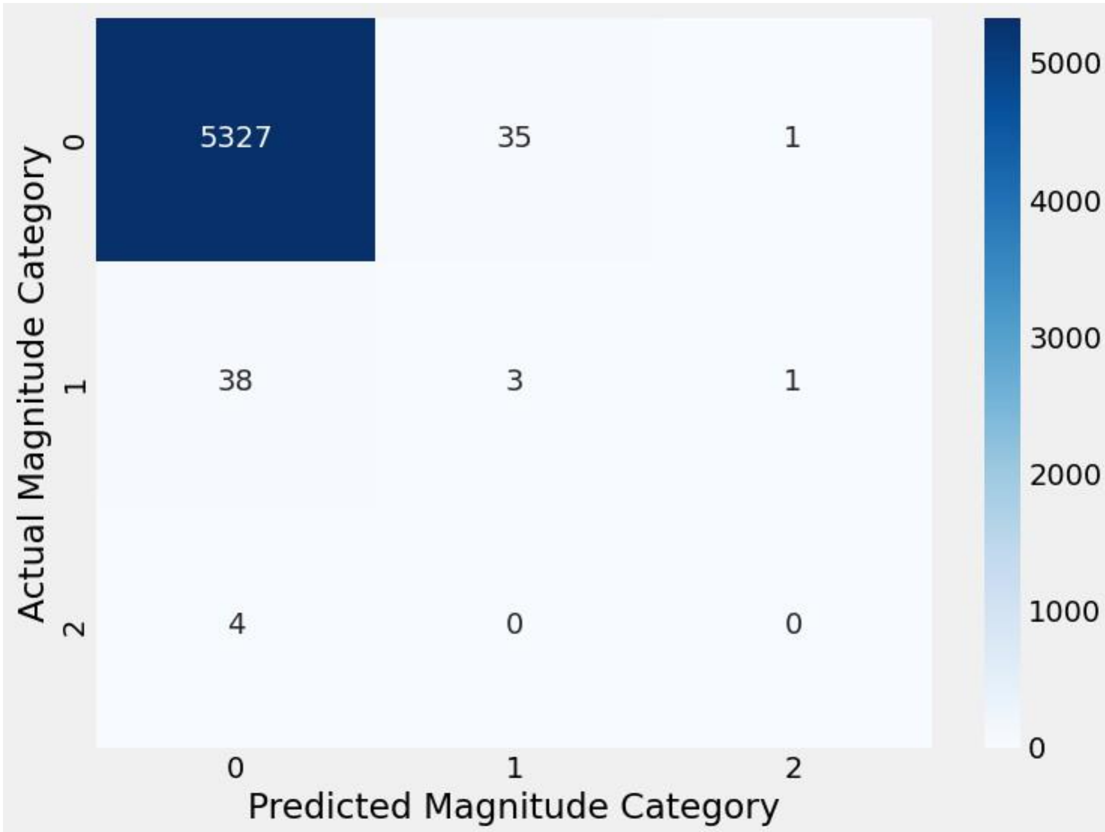
```
Accuracy: 0.9853947125161767
Confusion Matrix:
 [[5327   35    1]
 [  38    3    1]
 [   4    0    0]]
Classification Report:
               precision    recall  f1-score   support

       Minor       0.00      0.00      0.00         0
    Moderate       0.99      0.99      0.99      5363
      Strong       0.08      0.07      0.07        42
       Major       0.00      0.00      0.00         4

   micro avg       0.99      0.99      0.99      5409
   macro avg       0.27      0.27      0.27      5409
weighted avg       0.98      0.99      0.98      5409
```

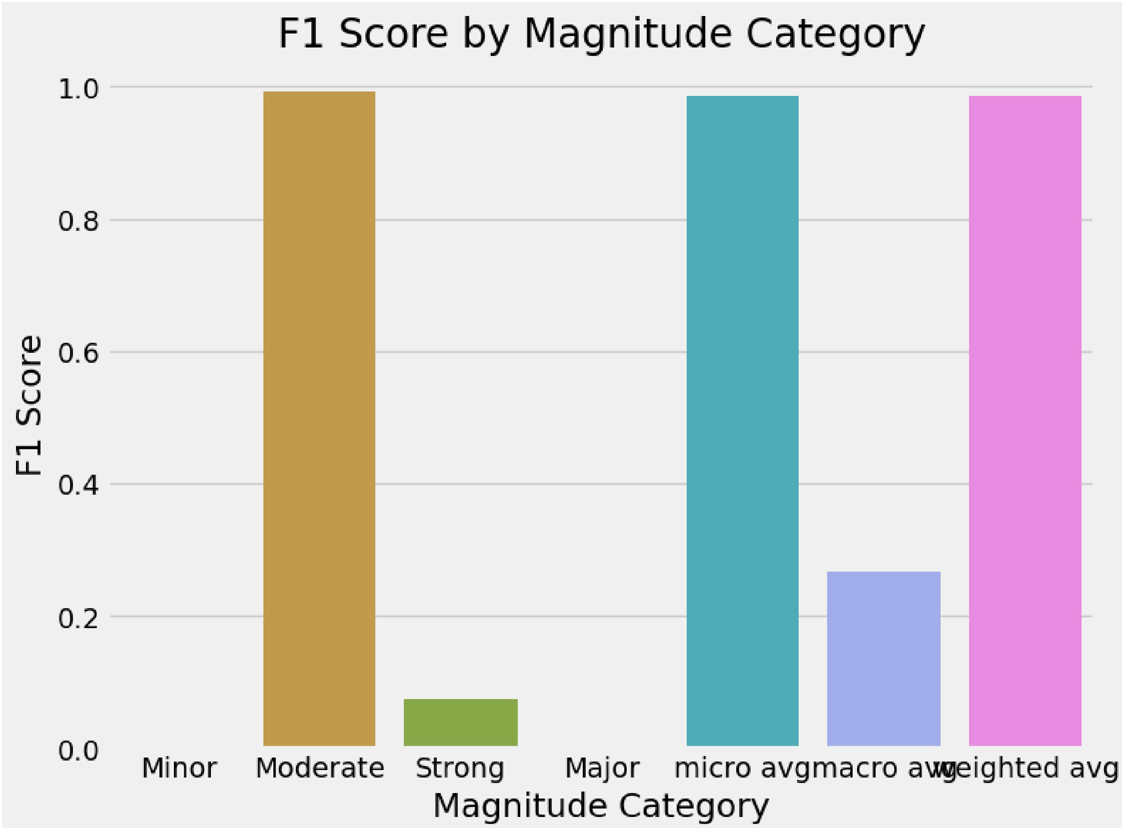
[ ]:
```python
# Create a scatter plot of actual vs predicted values
plt.figure(figsize=(8, 8))
plt.scatter(X_test['Longitude(deg)'], X_test['Latitude(deg)'], c=y_test,
    cmap='viridis')
plt.title('Actual Magnitude Category')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
print(" ")
plt.figure(figsize=(8, 8))
plt.scatter(X_test['Longitude(deg)'], X_test['Latitude(deg)'], c=y_pred,
    cmap='viridis')
plt.title('Predicted Magnitude Category')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
print(" ")
```

```
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted Magnitude Category')
plt.ylabel('Actual Magnitude Category')
plt.show()
print(" ")

cr = classification_report(y_test, y_pred, labels=[0, 1, 2, 3],
  target_names=['Minor', 'Moderate', 'Strong', 'Major'], output_dict=True)
# Convert classification report dictionary to DataFrame
cr_df = pd.DataFrame(cr).transpose()

# Create bar plot of classification report scores
plt.figure(figsize=(8, 6))
sns.barplot(x=cr_df.index, y=cr_df['f1-score'])
plt.xlabel('Magnitude Category')
plt.ylabel('F1 Score')
plt.title('F1 Score by Magnitude Category')
plt.show()
print(" ")
```


Actual Magnitude Category
Latitude
Longitude
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8


Predicted Magnitude Category
Latitude
Longitude
0.0
0.2
0.4
0.6
0.8
1.0


5327
35
1
38
3
1
4
0
0
Actual Magnitude Category
Predicted Magnitude Category
0
1
2
0
1000
2000
3000
4000
5000



##Random Forest

Loading the model and fitting it with training data

```
[ ]: from sklearn.ensemble import RandomForestRegressor

# Initialize a random forest regressor with 100 trees
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the regressor to the training data
rf.fit(X_train, y_train)
```

```
[ ]: RandomForestRegressor(random_state=42)
```

Predict the testing data and evaluate it

Find the predicted values and evaluate it using metrics like MSE, r2

```
[ ]: # Predict the target variable on the test data
y_pred = rf.predict(X_test)
```

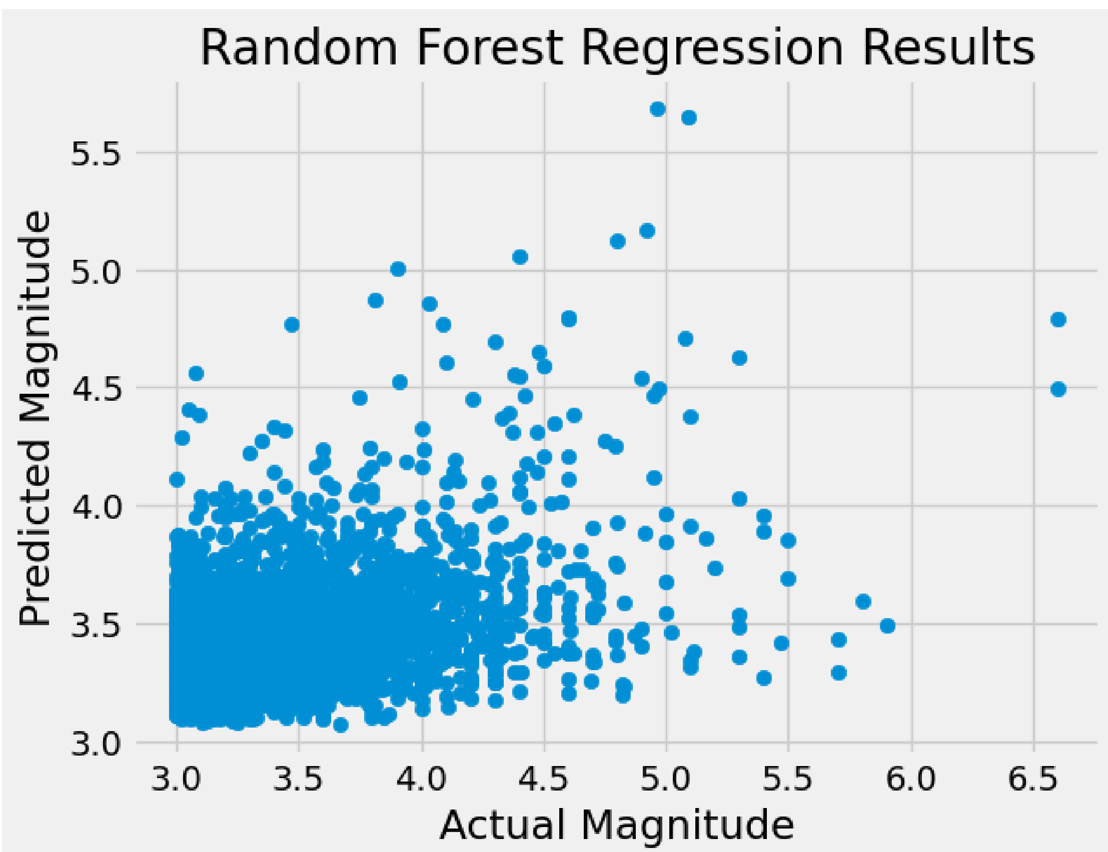
```
# Evaluate the performance of the model using mean squared error and R^2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

scores['mse'].append(mse)
scores['R^2'].append(r2)

print('Mean Squared Error: ', mse)
print('R^2 Score: ', r2)
```

```
Mean Squared Error:  0.15599116006378258
R^2 Score:  0.1428805732295345
```

Plot model

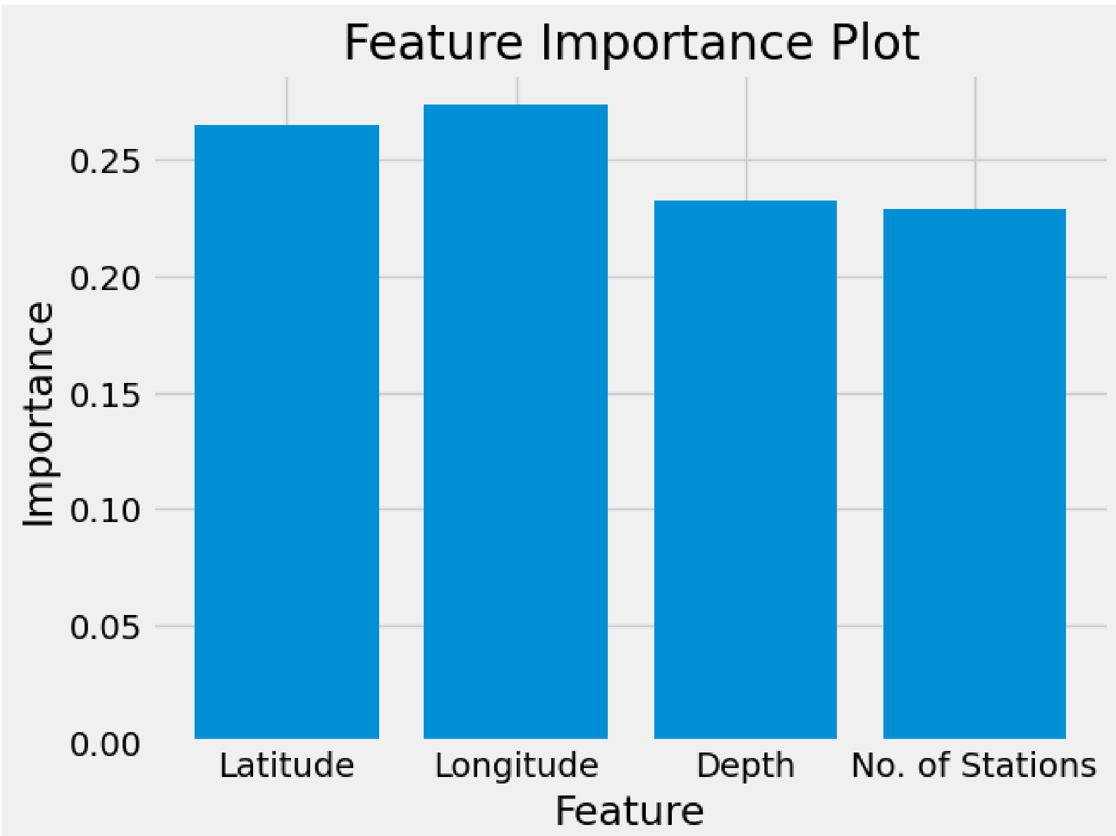
Scatter plot

[ ]:
```
# Plot the predicted and actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title('Random Forest Regression Results')
plt.show()
```



Feature Importance This plot shows the importance of each feature in the model. You can create a feature importance plot using the feature_importances_ attribute of the random forest model.

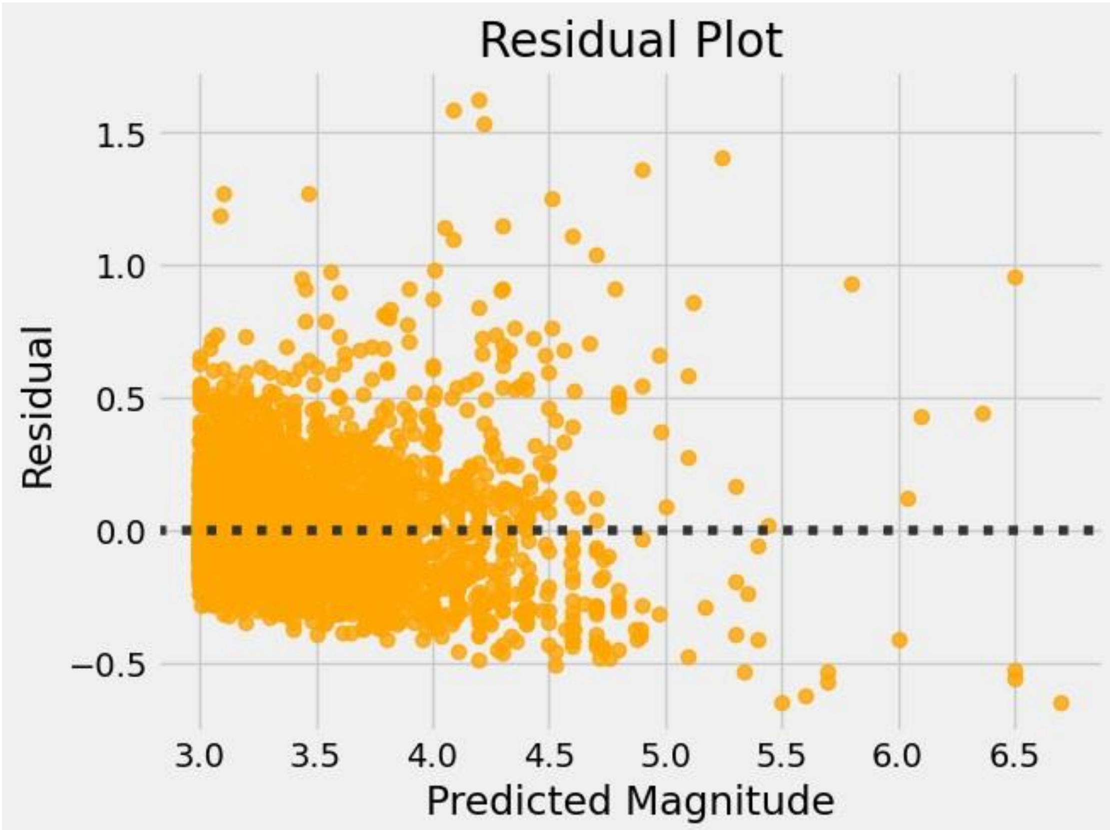
[ ]:
```
importances = rf.feature_importances_
features = ['Latitude', 'Longitude', 'Depth', 'No. of Stations']
plt.bar(features, importances)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance Plot')
plt.show()
```



Residual Plot A residual plot shows the difference between the actual values and the predicted values. You can create a residual plot using the residplot() function from the seaborn library.

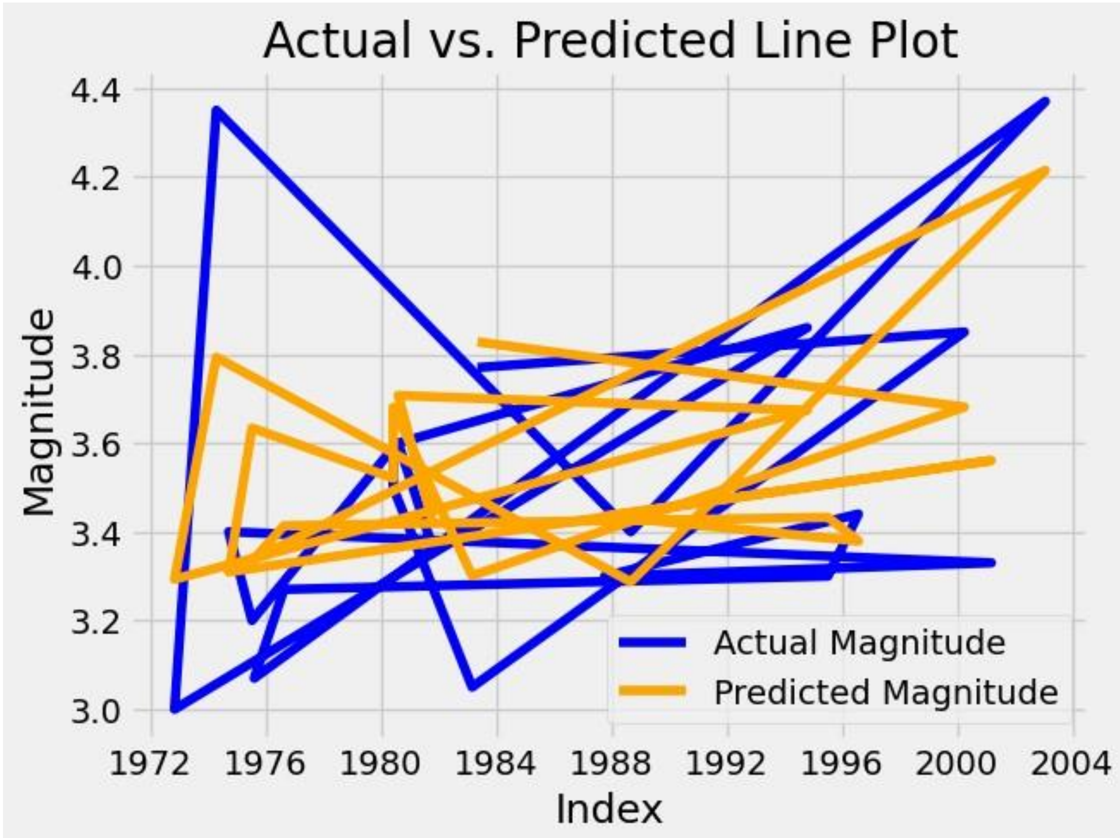
[ ]:
```
import seaborn as sns
sns.residplot(x= y_test, y =y_pred, color='orange')
plt.xlabel('Predicted Magnitude')
plt.ylabel('Residual')
plt.title('Residual  Plot')
```

```
plt.show()
```



Actual vs. Predicted Line Plot Actual vs. Predicted Line Plot: A line plot can be used to show the trend of the actual and predicted values over time (if the data is time-series). You can create a line plot using the plot() function.

```
[ ]: plt.plot(y_test.index[:20], y_test[:20], color='blue', label='Actual Magnitude')
     plt.plot(y_test.index[:20], y_pred[:20], color='orange', label='Predicted
      Magnitude')
     plt.xlabel('Index')
     plt.ylabel('Magnitude')
     plt.title('Actual vs. Predicted Line Plot')
     plt.legend()
     plt.show()
```



Concluding the accurate model

[ ]:
```
scores_df = pd.DataFrame(scores)
display(scores_df)
```

```
          Model name       mse       R^2
0  Linear regression  0.175628  0.034983
1                SVM  0.531661 -1.921297
2      Random Forest  0.155991  0.142881
```


```
          Model name       mse       R^2
0  Linear regression  0.175628  0.034983
1                SVM  0.531661 -1.921297
2      Random Forest  0.155991  0.142881
```


[ ]:
```
scores_df[scores_df["mse"] == scores_df["mse"].min()]
```

[ ]:
```
      Model name       mse       R^2
2  Random Forest  0.155991  0.142881
```

[ ]:
```
scores_df[scores_df["R^2"] == scores_df["R^2"].max()]
```

```
[ ]:      Model name       mse       R^2
 2  Random Forest  0.155991  0.142881
```

From the above result we can conclude that random forest is the most accurate model for predicting the magnitude of Earthquake compared to all other models used in this project.

** Requirement To be Installed To run program Source Code**

- Numpy
- Pandas
- Seaborn
- Scikit_learn
- matplotlib

Required Files are provided in github Click Here

To Know More about Project goto Github Readme File in github......