EARTHQUAKE PREDICTION MODEL USING PYTHON

TEAM MEMBER
311121205050-SHERINE P

PHASE 1- DOCUMENT SUBMISSION

Project: ***Earthquake Prediction Model using Python***

OBJECTIVE:

Create a precise and dependable earthquake prediction model with Python to reduce seismic risks and enhance public safety. Leveraging advanced data analysis and machine learning, this project seeks to provide early earthquake warnings and preparedness strategies, ultimately safeguarding communities from potential disasters.

Phase 1: *Data Preprocessing and Feature Engineering*

1.DATA SOURCE

A good data source for earthquake prediction using python should be Accurate, Complete, Covering the geographic area of interest, Accessible.

2.DATA PREPROCESSING

Data preprocessing is a pivotal aspect of the "Earthquake Prediction Model using Python" project. This process encompasses key steps, including duplicate removal to ensure data integrity, handling missing values judiciously, encoding categorical variables for machine learning compatibility, and normalizing data scales for enhanced model performance. These measures collectively pave the way for a reliable and accurate earthquake prediction model, vital for improving public safety and mitigating risks.

a) **Duplicate Removal:**

Duplicate removal is the initial step in data preprocessing. It involves identifying and eliminating redundant data points to ensure that our dataset is free from any replicated information. This helps prevent biases in our analysis and ensures the accuracy of the model.

b) **Handling Missing Values:**

- ***Mean Imputation:*** This involves replacing missing values in numerical features with the mean value of that feature.
- ***Median Imputation:*** Alternatively, missing values can be replaced with the median value of the respective numerical feature. These imputation techniques help preserve the dataset's significance while addressing missing data issues.

c) **Categorical Variable Encoding:**
   - ➢ ***One-Hot Encoding:*** One-hot encoding transforms categorical variables into binary vectors, creating separate binary columns for each category.
   - ➢ ***Label Encoding:*** Label encoding assigns a unique integer to each category, converting them into numeric values interpretable by machine learning algorithms.

d) **Data Normalization:**
   - ➢ ***Standardization:*** Standardization scales numerical features to have a mean of zero and a standard deviation of one, making it suitable for algorithms sensitive to feature scales.
   - ➢ ***Min-Max Scaling:*** Min-max scaling constraints data within a specific range, typically between 0 and 1, while preserving the relationships between data points. These normalization techniques ensure that the model's performance is not influenced by variations in feature scales, contributing to a more robust earthquake prediction model that enhances public safety by reducing seismic risks

PYTHON PROGRAM:

Import Dataset:

```
from google.colab import files
uploaded = files.upload()
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("Earthquake_Data.csv", delimiter=r'\s+')

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

Preprocessing:

```
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)

df.info()
```

Exporting preprocessed data:

```
file_name = 'Earthquake_data_processed.xlsx'

# saving the excel
df.to_excel(file_name)
print('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)
```

OUTPUT:

index	Date(YYYY/MM/DD)	Time	Latitude	Longitude	Depth	Mag	Magt	Nst	Gap	Clo	RMS	SRC	EventID
0	1966/07/01	09:41:21.82	35.9463	-120.47	12.26	3.2	Mx	7	171	20	0.02	NCSN	-4540462
1	1966/07/02	12:08:34.25	35.7867	-120.3265	8.99	3.7	Mx	8	86	3	0.04	NCSN	-4540520

2	1966/07/02	12:16:14.95	35.7928	-120.3353	9.88	3.4	Mx	8	89	2	0.03	NCSN	-45405521
3	1966/07/02	12:25:06.12	35.797	-120.3282	9.09	3.1	Mx	8	101	3	0.08	NCSN	-45405522
4	1966/07/05	18:54:54.36	35.9223	-120.4585	7.86	3.1	Mx	9	161	14	0.04	NCSN	-45405594
5	1966/07/27	08:12:00.26	35.9103	-120.4397	8.02	3.0	Mx	10	158	12	0.02	NCSN	-45408537
6	1966/08/03	12:39:05.79	35.8137	-120.3527	6.59	3.4	Mx	10	131	2	0.05	NCSN	-45408891
7	1966/08/07	17:03:24.14	35.938	-120.4568	11.76	3.0	Mx	11	153	19	0.04	NCSN	-45409522
8	1966/08/19	22:51:20.04	35.914	-120.4272	1.67	3.3	Mx	6	165	11	0.1	NCSN	-45409669
9	1966/09/07	00:20:52.12	36.0032	-120.0317	10.61	3.4	Mx	13	258	27	0.14	NCSN	-45410446
10	1968/01/12	22:19:10.35	36.6453	-121.2497	6.84	3.0	ML	14	155	2	0.07	NCSN	-10013556

Processed data:

index	Latitude(deg)	Longitude(deg)	Depth(km)	Magnitude(ergs)	Magnitude_type	No_of_Stations	Gap	Close	RMS	SRC	EventID
1966-07-01 09:41:21.820000	35.9463	-120.47	12.26	3.2	Mx	7	171	20	0.02	NCSN	-4540462
1966-07-02 12:08:34.250000	35.7867	-120.3265	8.99	3.7	Mx	8	86	3	0.04	NCSN	-4540520

1966-07-02 12:16:14.950000	35.7928	-120.3353	9.88	3.4	Mx	8	89	2	0.03	NCSN	-4540521
1966-07-02 12:25:06.120000	35.797	-120.3282	9.09	3.1	Mx	8	101	3	0.08	NCSN	-4540522
1966-07-05 18:54:54.360000	35.9223	-120.4585	7.86	3.1	Mx	9	161	14	0.04	NCSN	-4540594

<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)

3.FEATURE SELECTION:

Feature Selection is the process of identifying and selecting the most relevant features from a dataset for a given machine learning task. The goal of feature selection is to improve the performance of the machine learning model by reducing the number of features and eliminating irrelevant or redundant features.

There are a variety of feature selection techniques. Some of the most common techniques include:

- **Correlation-based feature selection**: It is a fundamental technique employed to identify and retain the most influential variables from our dataset. This method evaluates the strength and direction of linear relationships between each feature and the target variable, which, in our case, is earthquake occurrence.

- **Information gain-based feature selection:** It is a pivotal technique employed to discern the most informative attributes from our dataset. This method harnesses information theory metrics to assess the importance of each feature concerning its ability to contribute valuable insights into earthquake prediction. Information gain quantifies how much knowledge a particular feature can provide about the occurrence of seismic events, making it a powerful tool for feature selection.

- **Recursive feature elimination (RFE):** It is a pivotal technique employed to discern the most informative attributes from our dataset. This method harnesses information theory metrics to assess the importance of each feature concerning its ability to contribute valuable insights into earthquake prediction. Information gain quantifies how much knowledge a particular feature can provide about the occurrence of seismic events, making it a powerful tool for feature selection.

- **Model selection:** is a pivotal step that determines the algorithm best suited to capture the complex patterns within seismic data. We explore various modeling techniques to identify the most suitable one for our predictive task. Linear regression, a foundational approach, provides insight into the basic relationships between variables and serves as a benchmark for our analysis**.**.Some of the most common algorithms include:

- **Linear regression:** linear regression stands as a foundational modeling technique that aids us in understanding and establishing basic relationships between various attributes and seismic event occurrences. This straightforward yet powerful method is invaluable for providing an initial baseline model. Linear regression assumes a linear connection between predictor variables and the target, allowing us to quantify the influence of each feature on earthquake predictions.

- **Random forest regressor:** The Random Forest Regressor emerges as a robust and versatile modeling technique. This ensemble algorithm leverages the power of decision trees to capture intricate, nonlinear relationships within seismic data. By aggregating multiple decision trees, Random Forest enhances predictive accuracy and reduces overfitting, making it particularly well-suited for complex and noisy datasets.

- **Gradient boosting regressor:** The Gradient Boosting Regressor stands out as a powerful and adaptive modeling technique. This ensemble algorithm employs a sequential learning approach, iteratively improving the model's predictive accuracy by minimizing errors from previous iterations. This makes it highly effective in capturing complex, nonlinear relationships within seismic data.

4. MODEL SELECTION:

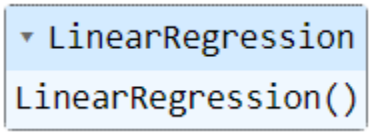
Choose machine learning algorithms suitable for regression tasks. Common models for predicting earthquakes include Linear regression , SVM, Naive Bayes ,random forest.

Experiment with multiple algorithms to determine which one provides the best performance for your specific dataset. You can also consider ensemble methods.

PYTHON PROGRAM - 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)
```



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
```

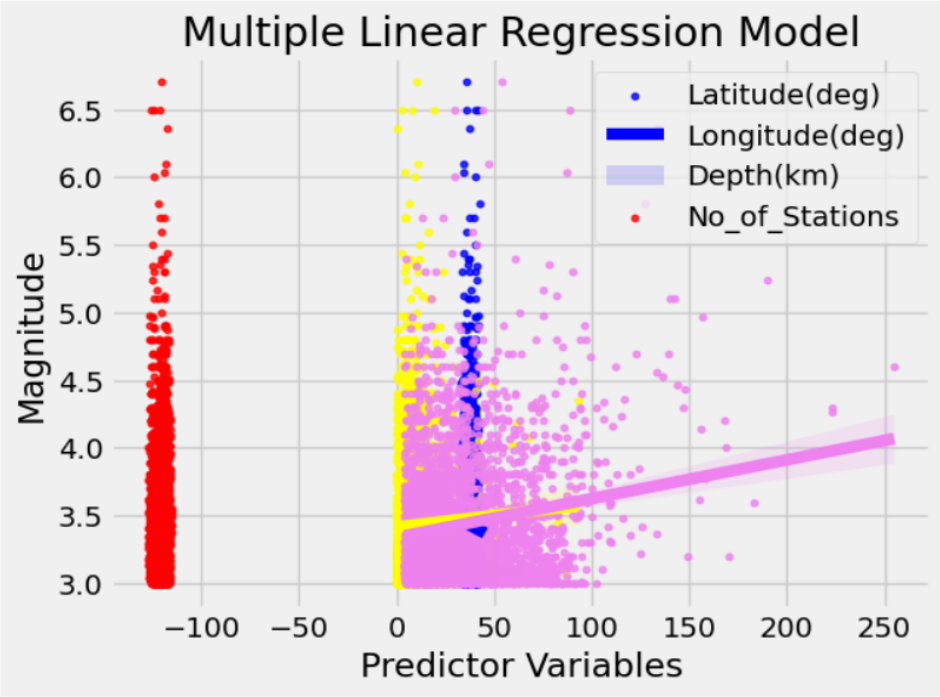
Predict for new data

```
# 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()
```



PYTHON PROGRAM - SUPPORT VECTOR MACHINE:

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)
```

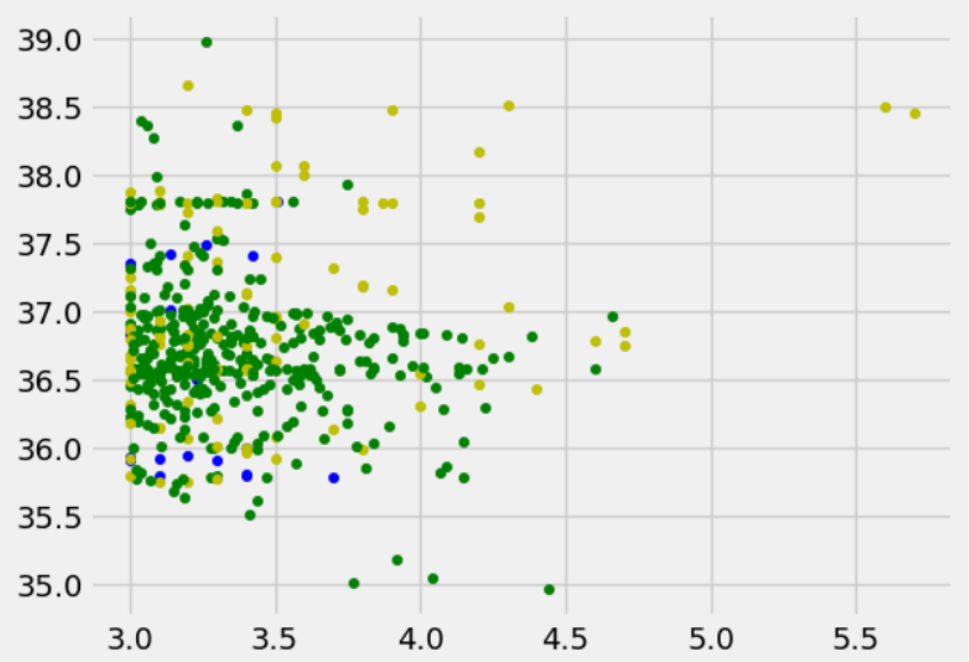
```
New SVM predictions: [3.57401976 3.03496212]
```

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()

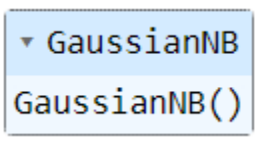
```
Index(['Latitude(deg)', 'Longitude(deg)', 'Depth(km)', 'Magnitude(ergs)',
       'Magnitude_type', 'No_of_Stations', 'Gap', 'Close', 'RMS', 'SRC',
       'EventID'],
      dtype='object')
array(['Mx', 'ML', 'Md', 'Mw'], dtype=object)
```

PYTHON PROGRAM - 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 achieved not so good results.

```
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)
```




```
# Use the trained model to make predictions on the testing data
y_pred = gnb.predict(X_test)
# Calculate the accuracy of the model
```

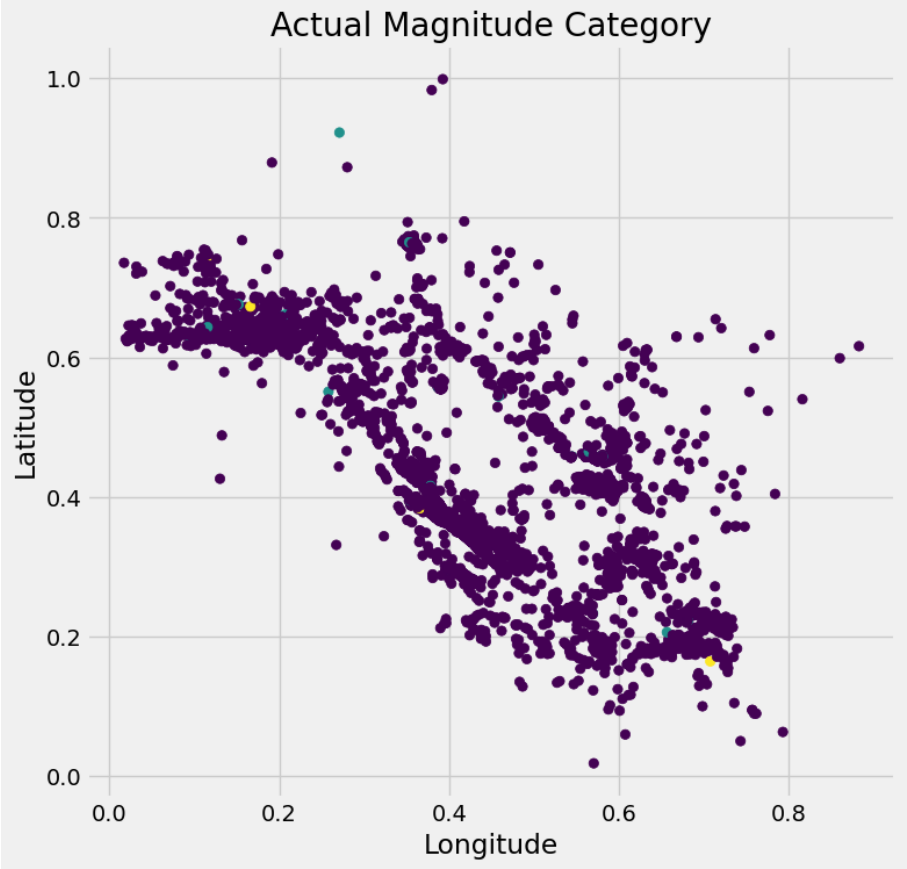
```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# 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)
```

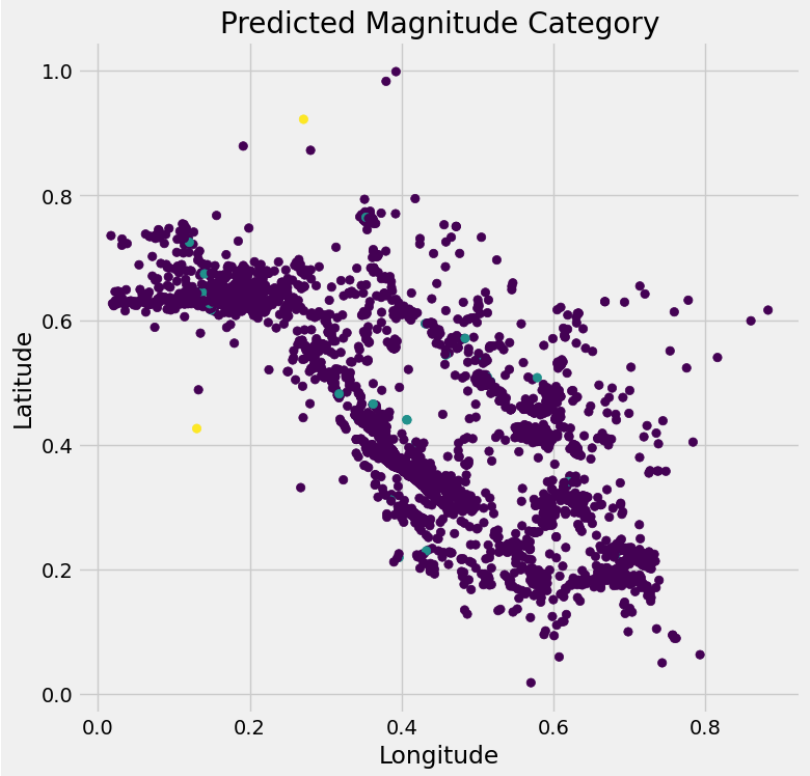
o/p:

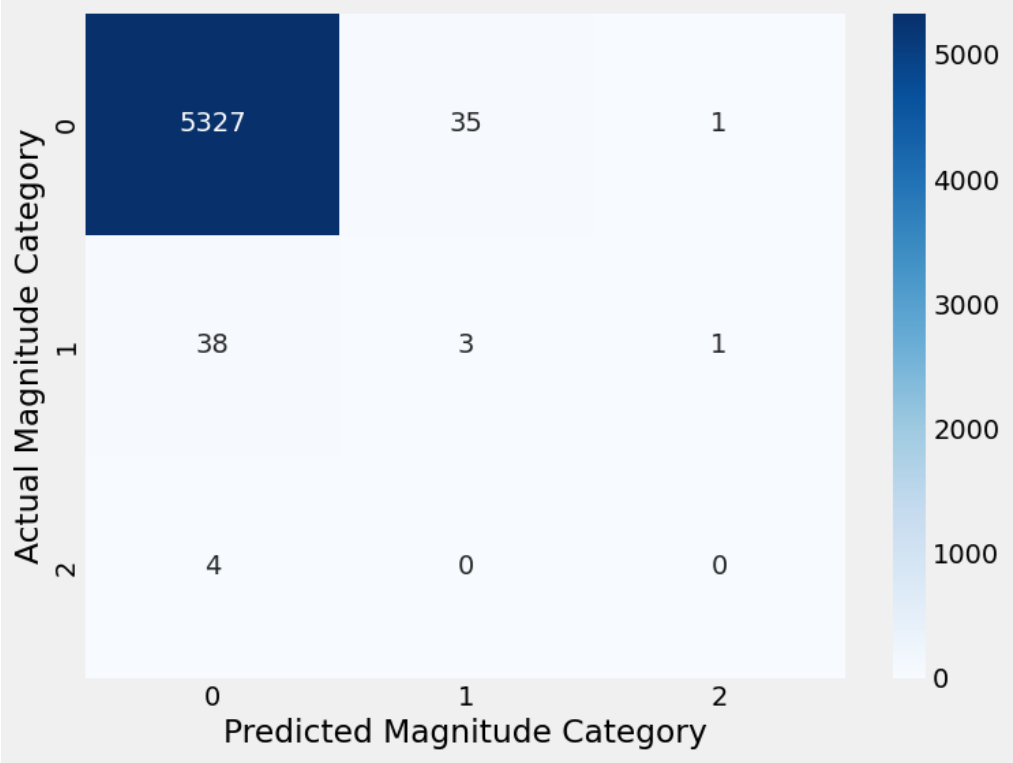
```
Accuracy: 0.9853947125161767
Confusion Matrix:
 [[5327   35    1]
 [  38    3    1]
 [   4    0    0]]
Classification Report:
              precision    recall  f1-score   support

       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
```

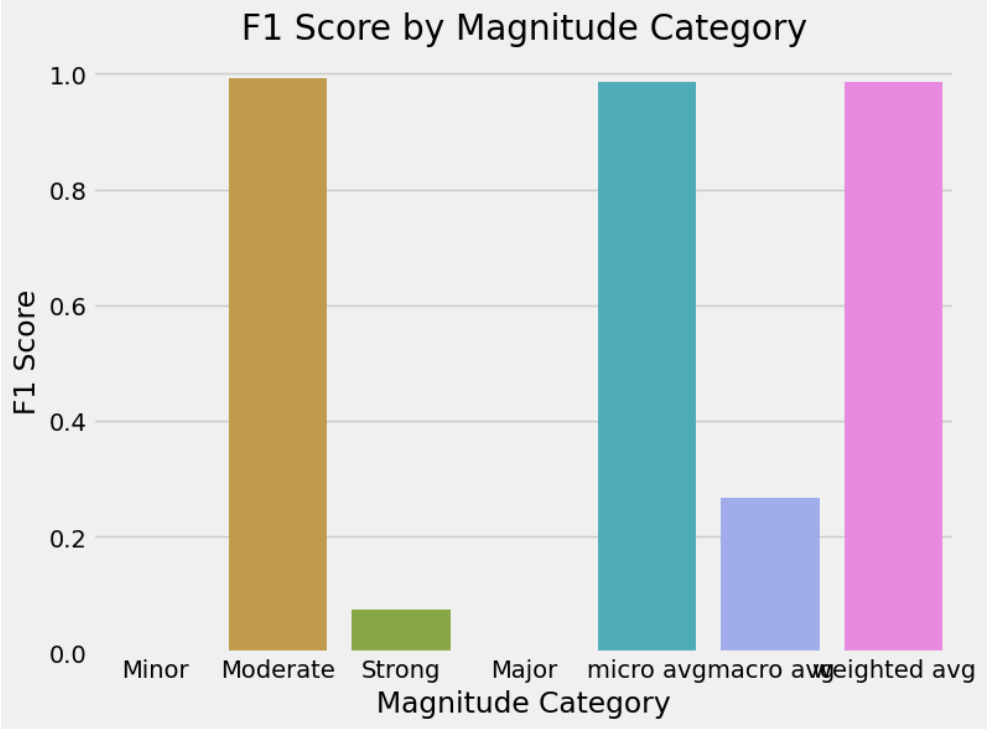
```
# Create a scatter plot of actual vs predicted values
plt.figure(figsize=(8, 8))
plt.scatter(X_test['Longitude(deg)'], X_test['Latitude(deg)'], c=y_test, cmap='viridis')
plt.title('Actual Magnitude Category')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
print(" ")
plt.figure(figsize=(8,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




Predicted 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

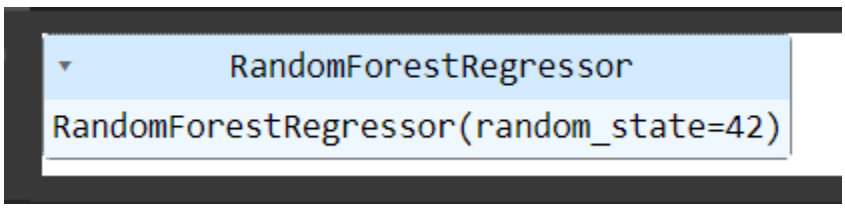

Actual Magnitude Category
Predicted Magnitude Category
5327
35
1
38
3
1
4
0
0
0
1
2
5000
4000
3000
2000
1000
0



PYTHON PROGRAM-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)
```



Predict the testing data and evaluate it

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

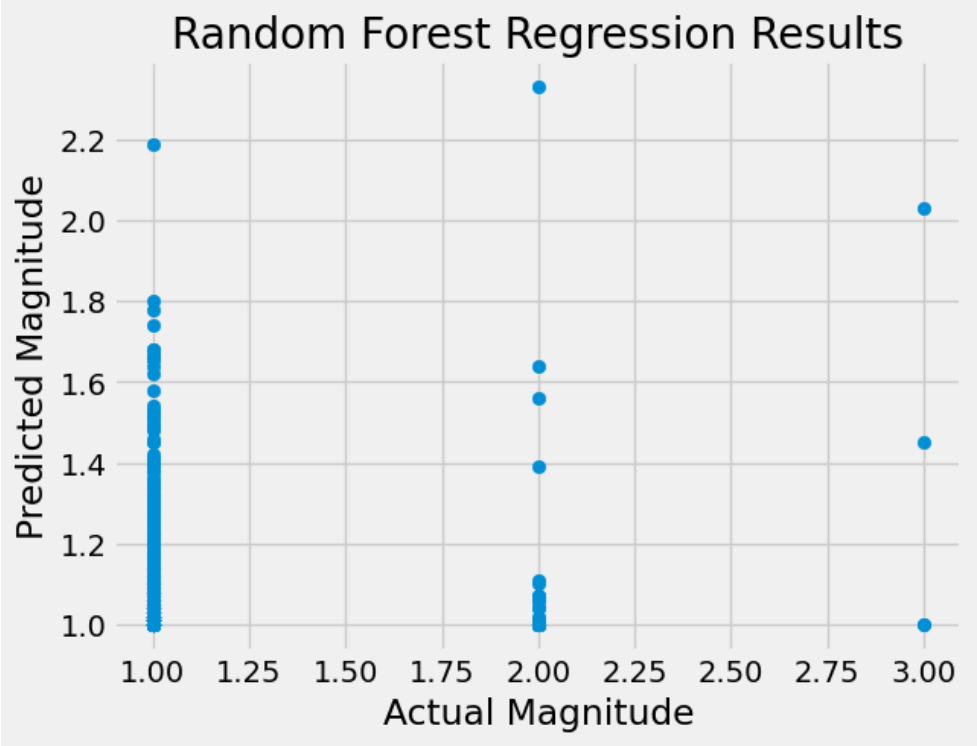
```
# Predict the target variable on the test data
y_pred = rf.predict(X_test)

# 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:  0.012586078757626l8
R^2 Score:  -0.18318898696107633
```

Plot model

Scatter plot

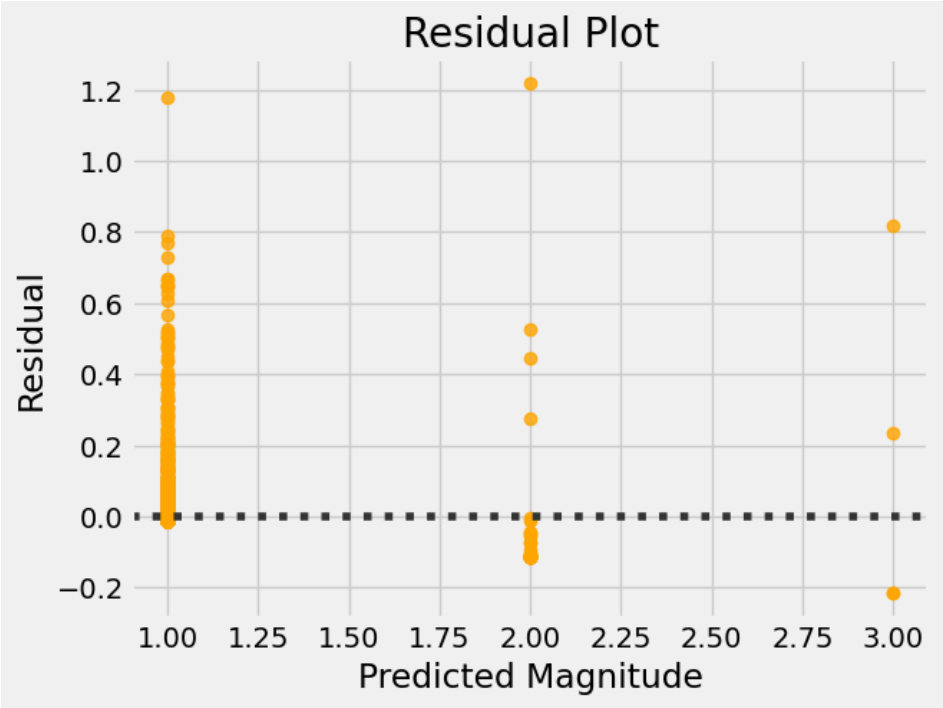
```
# Plot the predicted and actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title('Random Forest Regression Results')
plt.show()
```




```
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()
```

Residual Plot

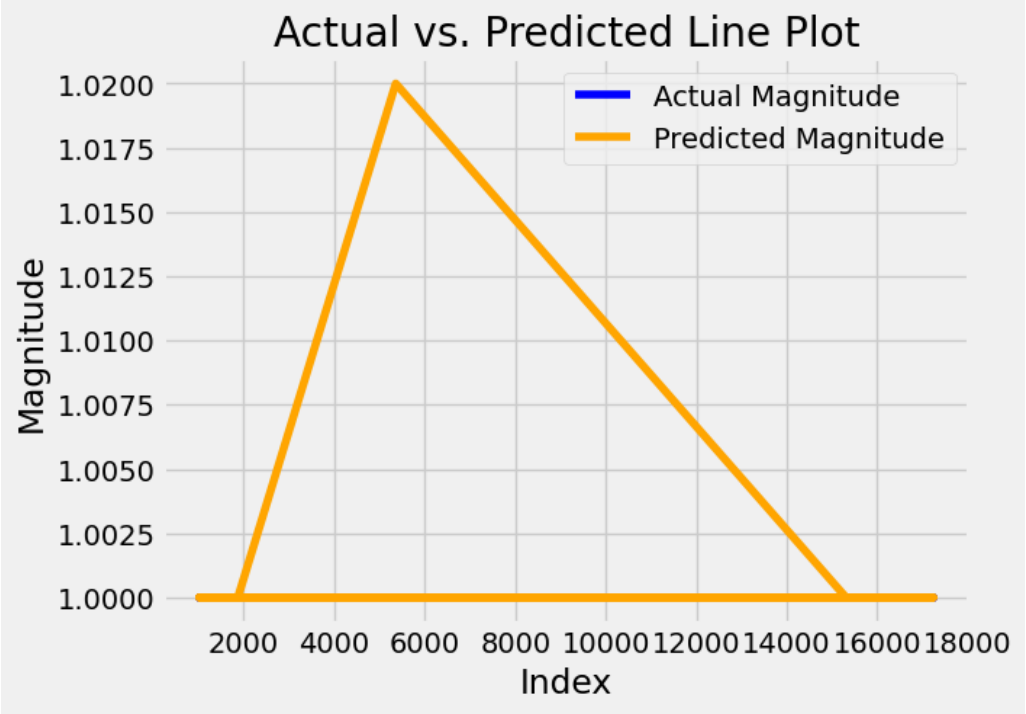
A residual plot shows the difference between the actual values and the predicted values. You can create a residual plot using the residplot() function from the seaborn library.




```
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()
```

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.


Actual vs. Predicted Line Plot
Actual Magnitude
Predicted Magnitude
Magnitude
1.0200
1.0175
1.0150
1.0125
1.0100
1.0075
1.0050
1.0025
1.0000
2000
4000
6000
8000
10000
12000
14000
16000
18000
Index

CONCLUSION:

Concluding the accurate model:

scores_df = pd.DataFrame(scores)

display(scores_df)

1 to 3 of 3 entries Filter

index	Model name	mse	R^2
0	Linear regression	0.18794191864127402	0.02351355576513192
1	SVM	0.698141478003235	-2.6273211125904736
2	Random Forest	0.01258607875762618	-0.18318898696107633

Show 25 per page

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

1 entry Filter

index	Model name	mse	R^2
2	Random Forest	0.01258607875762618	-0.18318898696107633

Show 25 per page

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

1 entry Filter

index	Model name	mse	R^2
0	Linear regression	0.18794191864127402	0.02351355576513192

Show 25 per page

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.