## Import Dataset

In [48]:

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** | **Nst** | **Gap** | **Clo** | **RMS** | **SRC** | **EventID** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1966/07/01 | 09:41:21.82 | 35.9463 | -120.4700 | 12.26 | 3.20 | Mx | 7 | 171 | 20 | 0.02 | NCSN | -4540462 |
| **1** | 1966/07/02 | 12:08:34.25 | 35.7867 | -120.3265 | 8.99 | 3.70 | Mx | 8 | 86 | 3 | 0.04 | NCSN | -4540520 |
| **2** | 1966/07/02 | 12:16:14.95 | 35.7928 | -120.3353 | 9.88 | 3.40 | Mx | 8 | 89 | 2 | 0.03 | NCSN | -4540521 |
| **3** | 1966/07/02 | 12:25:06.12 | 35.7970 | -120.3282 | 9.09 | 3.10 | Mx | 8 | 101 | 3 | 0.08 | NCSN | -4540522 |
| **4** | 1966/07/05 | 18:54:54.36 | 35.9223 | -120.4585 | 7.86 | 3.10 | Mx | 9 | 161 | 14 | 0.04 | NCSN | -4540594 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **18025** | 2007/12/19 | 12:14:09.62 | 34.1438 | -116.9822 | 7.03 | 4.06 | ML | 10 | 73 | 14 | 0.08 | NCSN | 40207706 |
| **18026** | 2007/12/21 | 12:14:56.45 | 37.3078 | -121.6735 | 8.47 | 3.08 | ML | 114 | 45 | 5 | 0.12 | NCSN | 51192926 |
| **18027** | 2007/12/23 | 21:43:43.54 | 37.2127 | -117.8230 | 10.00 | 3.54 | ML | 45 | 176 | 40 | 0.07 | NCSN | 51193070 |
| **18028** | 2007/12/28 | 01:59:42.40 | 36.5292 | -121.1133 | 5.99 | 3.04 | ML | 70 | 45 | 4 | 0.06 | NCSN | 51193343 |
| **18029** | 2007/12/28 | 23:20:28.12 | 38.7710 | -122.7370 | 2.34 | 3.40 | Mw | 49 | 37 | 1 | 0.07 | NCSN | 51193419 |

18030 rows × 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.

In [49]:

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)** | **Magnitude(ergs)** | **Magnitude\_type** | **No\_of\_Stations** | **Gap** | **Close** | **RMS** | **SRC** | **EventID** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1966-07-01 09:41:21.820** | 35.9463 | -120.4700 | 12.26 | 3.20 | Mx | 7 | 171 | 20 | 0.02 | NCSN | -4540462 |
| **1966-07-02 12:08:34.250** | 35.7867 | -120.3265 | 8.99 | 3.70 | Mx | 8 | 86 | 3 | 0.04 | NCSN | -4540520 |
| **1966-07-02 12:16:14.950** | 35.7928 | -120.3353 | 9.88 | 3.40 | Mx | 8 | 89 | 2 | 0.03 | NCSN | -4540521 |
| **1966-07-02 12:25:06.120** | 35.7970 | -120.3282 | 9.09 | 3.10 | Mx | 8 | 101 | 3 | 0.08 | NCSN | -4540522 |
| **1966-07-05 18:54:54.360** | 35.9223 | -120.4585 | 7.86 | 3.10 | Mx | 9 | 161 | 14 | 0.04 | NCSN | -4540594 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2007-12-19 12:14:09.620** | 34.1438 | -116.9822 | 7.03 | 4.06 | ML | 10 | 73 | 14 | 0.08 | NCSN | 40207706 |
| **2007-12-21 12:14:56.450** | 37.3078 | -121.6735 | 8.47 | 3.08 | ML | 114 | 45 | 5 | 0.12 | NCSN | 51192926 |
| **2007-12-23 21:43:43.540** | 37.2127 | -117.8230 | 10.00 | 3.54 | ML | 45 | 176 | 40 | 0.07 | NCSN | 51193070 |
| **2007-12-28 01:59:42.400** | 36.5292 | -121.1133 | 5.99 | 3.04 | ML | 70 | 45 | 4 | 0.06 | NCSN | 51193343 |
| **2007-12-28 23:20:28.120** | 38.7710 | -122.7370 | 2.34 | 3.40 | Mw | 49 | 37 | 1 | 0.07 | NCSN | 51193419 |

18030 rows × 11 columns

In [79]:

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

In [64]:

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.

In [65]:

import warnings

warnings.filterwarnings('ignore')

## Partition the data into Training and Testing data

In [66]:

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

In [74]:

from sklearn.linear\_model import LinearRegression

*# Train the linear regression model*

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

Out[74]:

LinearRegression()

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

### Predict the testing data

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

In [75]:

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

In [70]:

*# 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

In [34]:

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})

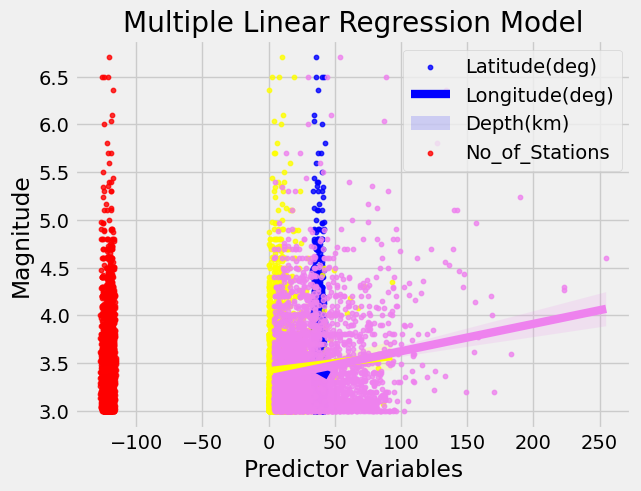
plt.legend(labels=['Latitude(deg)', 'Longitude(deg)', 'Depth(km)', 'No\_of\_Stations'])

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

In [76]:

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

In [77]:

*# 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

In [37]:

*# 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

In [38]:

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])

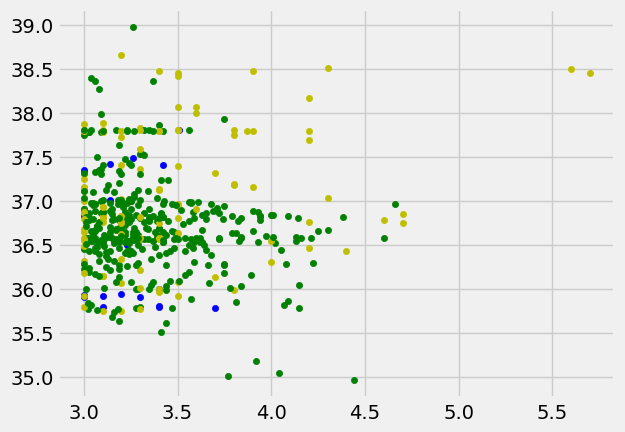
elif classes[i] == 'Md':

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()



In [39]:

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')

Out[39]:

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**

In [40]:

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)

Out[40]:

GaussianNB()

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [44]:

*# Use the trained model to make predictions on the testing data*

y\_pred = gnb.predict(X\_test)

In [46]:

*# 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

In [47]:

*# 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'])

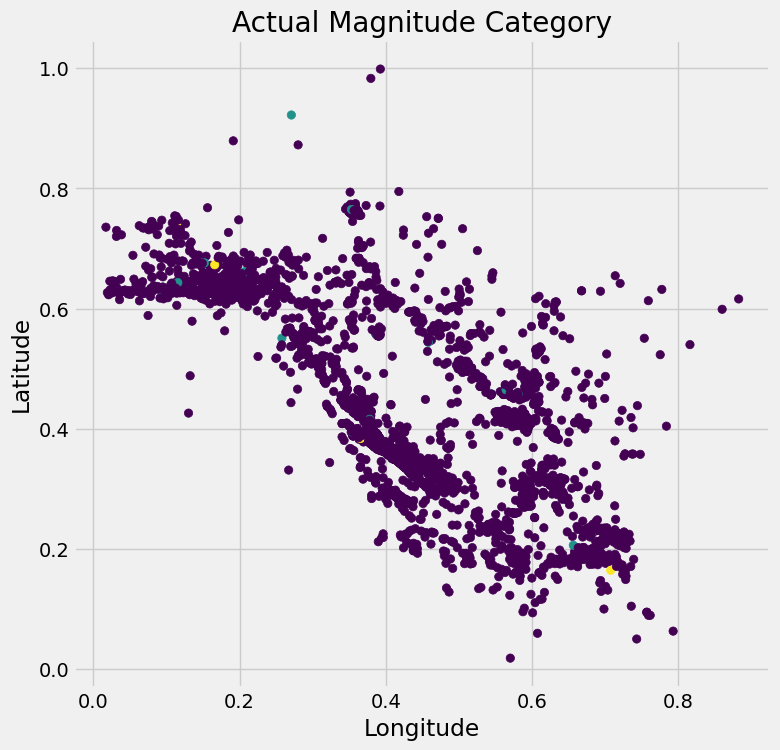
plt.xlabel('Magnitude Category')

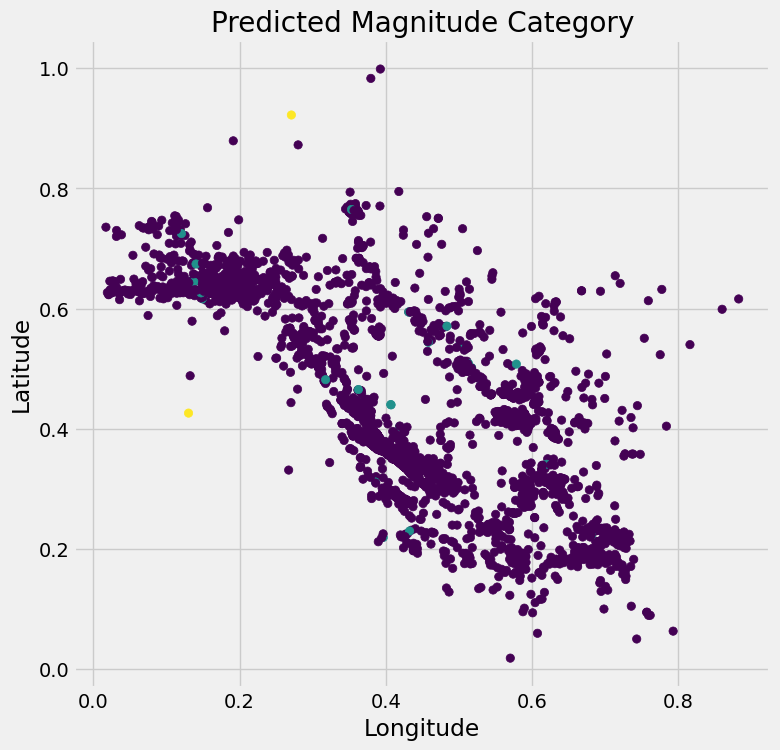
plt.ylabel('F1 Score')

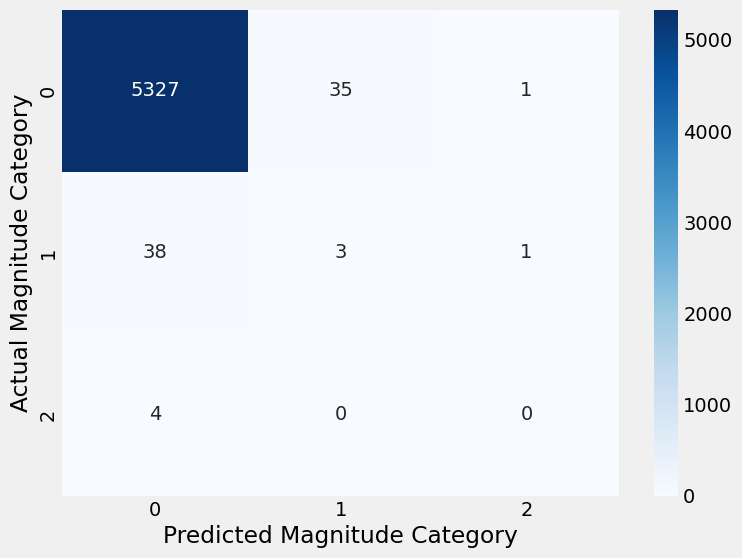
plt.title('F1 Score by Magnitude Category')

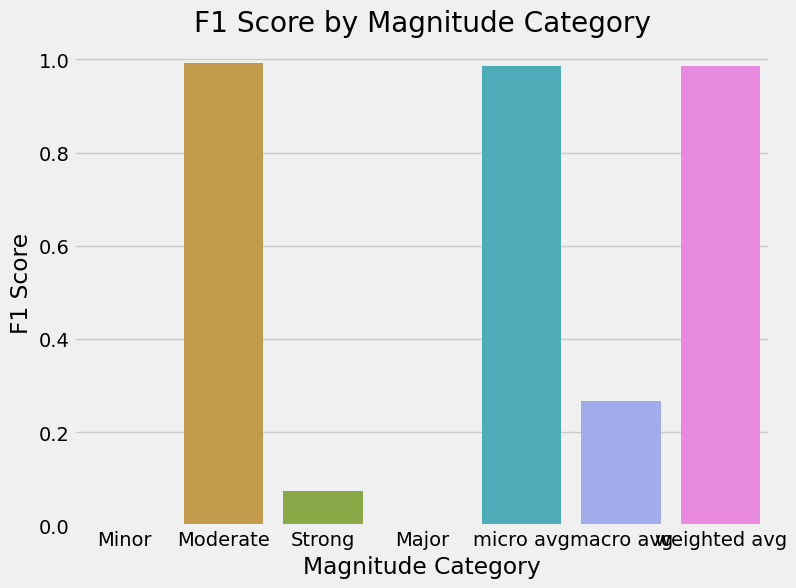
plt.show()

print(" ")









## Random Forest

### Loading the model and fitting it with training data

In [80]:

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)

Out[80]:

RandomForestRegressor(random\_state=42)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

### Predict the testing data and evaluate it

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

In [89]:

*# 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**

In [82]:

*# Plot the predicted and actual values*

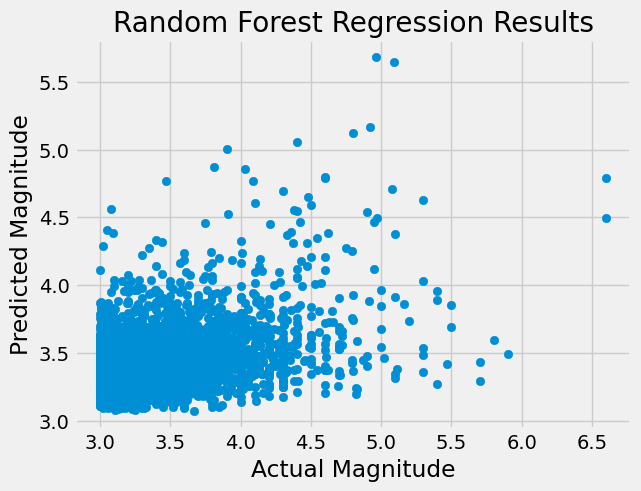
plt.scatter(y\_test, y\_pred)

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.

In [17]:

importances = rf.feature\_importances\_

features = ['Latitude', 'Longitude', 'Depth', 'No. of Stations']

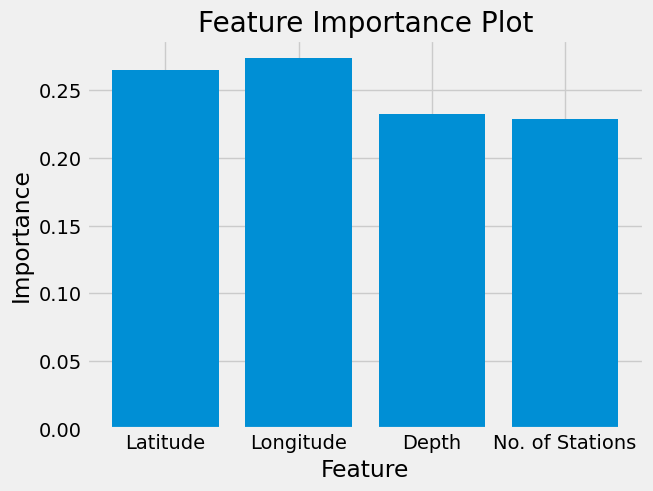
plt.bar(features, importances)

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.

In [19]:

import seaborn as sns

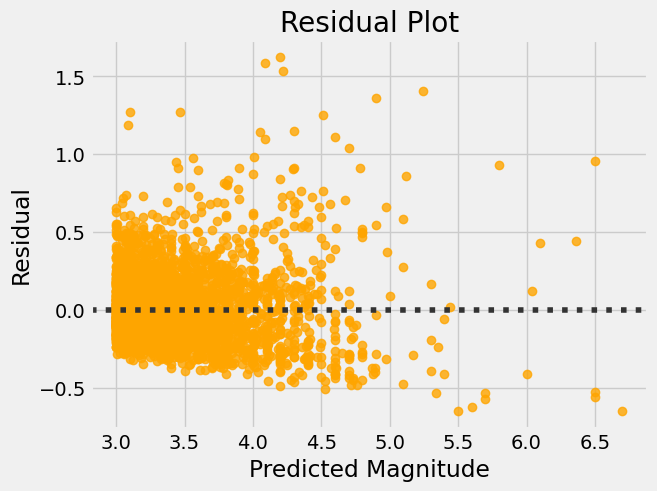
sns.residplot(x= y\_test, y =y\_pred, color='orange')

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.

In [63]:

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')

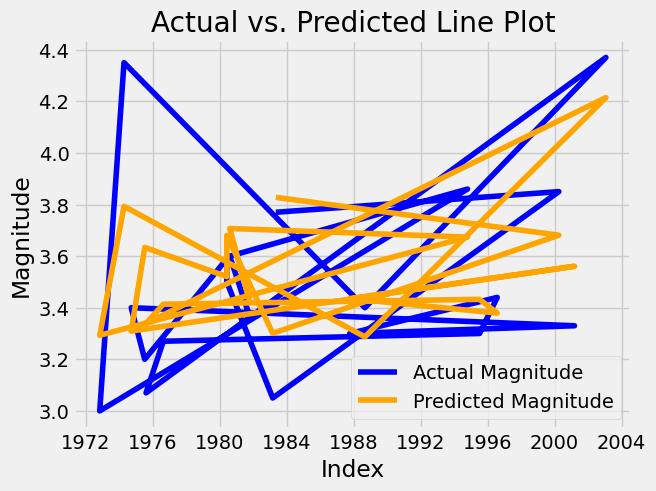
plt.xlabel('Index')

plt.ylabel('Magnitude')

plt.title('Actual vs. Predicted Line Plot')

plt.legend()

plt.show()



## Concluding the accurate model

In [96]:

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 |

In [102]:

scores\_df[scores\_df["mse"] == scores\_df["mse"].min()]

Out[102]:

|  | **Model name** | **mse** | **R^2** |
| --- | --- | --- | --- |
| **2** | Random Forest | 0.155991 | 0.142881 |

In [103]:

scores\_df[scores\_df["R^2"] == scores\_df["R^2"].max()]

Out[103]:

|  | **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.