Earthquake Prediction

It is well known that if a disaster has happened in a region, it is likely to happen there again. Some regions really have frequent earthquakes, but this is just a comparative quantity compared to other regions. So, predicting the earthquake with Date and Time, Latitude and Longitude from previous data is not a trend which follows like other things, it is natural occuring.

Import the necessary libraries required for buidling the model and data analysis of the earthquakes.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
print(os.listdir("../input"))
```

```
['database.csv']
```

Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.

In [2]:

```
data = pd.read_csv("../input/database.csv")
data.head()
```

Out[2]:

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ID	Source	Location Source	Magnitude Source	Status
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ID	Source	Location Source	Magnitude Source	Status
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic

In [3]:

```
data.columns
```

Out[3]:

```
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Er
ror',
       'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',
       'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
       'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID'
,
       'Source', 'Location Source', 'Magnitude Source', 'Status'],
      dtype='object')
```

Figure out the main features from earthquake data and create a object of that features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.

In [4]:

```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```

Out[4]:

	Date	Time	Latitude	Longitude	Depth	Magnitude
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0
1	01/04/1965	11:29:49	1.863	127.352	80.0	5.8
2	01/05/1965	18:05:58	-20.579	-173.972	20.0	6.2
3	01/08/1965	18:49:43	-59.076	-23.557	15.0	5.8
4	01/09/1965	13:32:50	11.938	126.427	15.0	5.8

Here, the data is random we need to scale according to inputs to the model. In this, we convert given Date and Time to Unix time which is in seconds and a numeral. This can be easily used as input for the network we built.

In [5]:

```
import datetime
import time

timestamp = []
for d, t in zip(data['Date'], data['Time']):
    try:
        ts = datetime.datetime.strptime(d+' '+t, '%m/%d/%Y %H:%M:%S')
        timestamp.append(time.mktime(ts.timetuple()))
    except ValueError:
        # print('ValueError')
        timestamp.append('ValueError')
```

In [6]:

```
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

In [7]:

```
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

Out[7]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08
4	11.938	126.427	15.0	5.8	-1.57026e+08

Visualization

Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

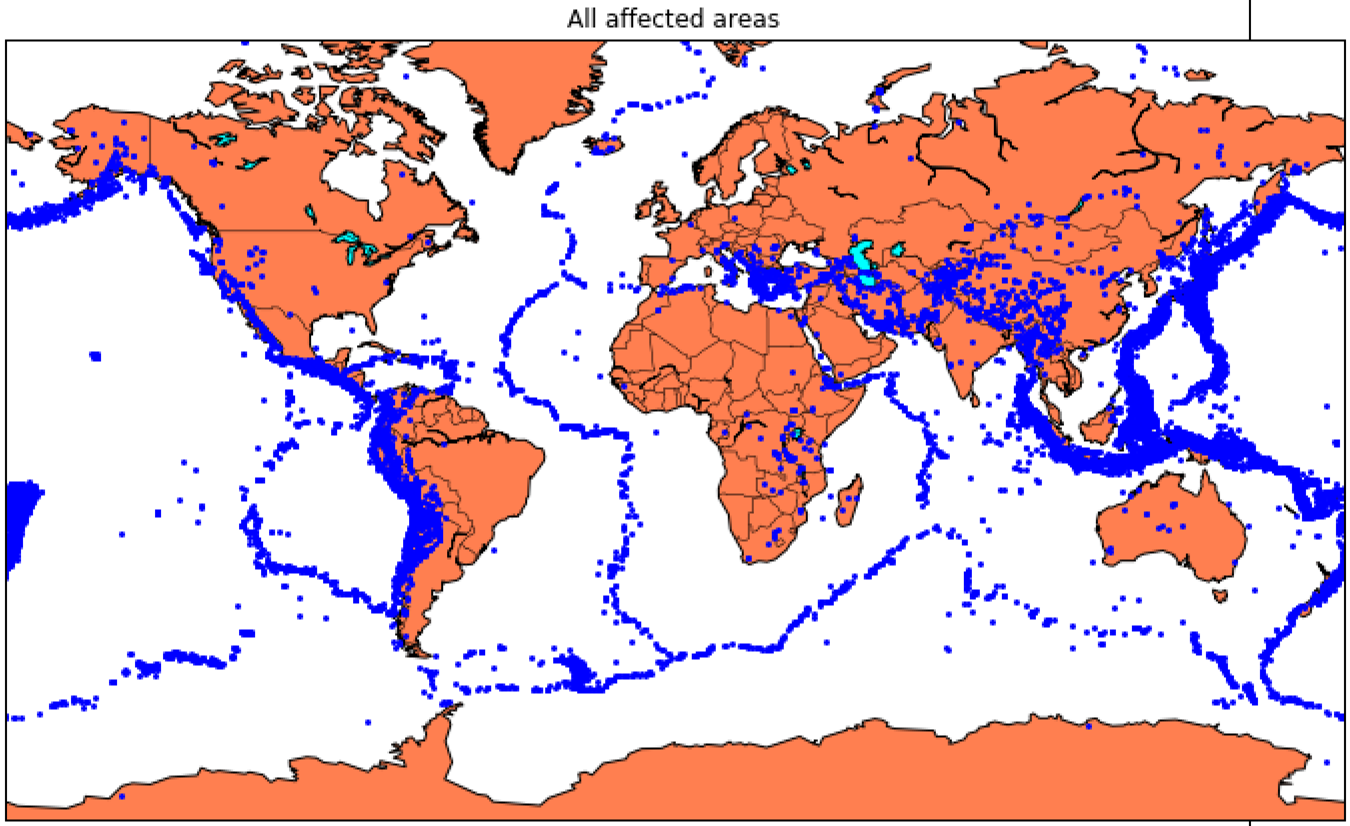
In [8]:

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urc
rnrlon=180,lat_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
            #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)
```

In [9]:

```
fig = plt.figure(figsize=(12,10))
plt.title("All affected areas")
m.plot(x, y, "o", markersize = 2, color = 'blue')
m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py:17
04: MatplotlibDeprecationWarning: The axesPatch function was deprecated in
 version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py:17
07: MatplotlibDeprecationWarning: The axesPatch function was deprecated in
 version 2.1. Use Axes.patch instead.
  if limb is not ax.axesPatch:
```



Splitting the Data

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and

Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

In [10]:

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

In [11]:

```
from sklearn.cross_validation import train_test_split

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

```
(18727, 3) (4682, 3) (18727, 2) (4682, 3)
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: Dep
recationWarning: This module was deprecated in version 0.18 in favor of th
e model_selection module into which all the refactored classes and functio
ns are moved. Also note that the interface of the new CV iterators are dif
ferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Here, we used the RandomForestRegressor model to predict the outputs, we see the strange prediction from this with score above 80% which can be assumed to be best fit but not due to its predicted values.

In [12]:

```
from sklearn.ensemble import RandomForestRegressor

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

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py
:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy modul
e and should not be imported. It will be removed in a future NumPy release
.
  from numpy.core.umath_tests import inner1d
```

Out[12]:

```
array([[  5.96,  50.97],
       [  5.88,  37.8 ],
       [  5.97,  37.6 ],
       ...,
       [  6.42,  19.9 ],
       [   5.73,  591.55],
       [   5.68,   33.61]])
```

In [13]:

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

Out[13]:

```
0.8614799631765803
```

In [14]:

```
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)
```

Out[14]:

```
array([[   5.8888 ,   43.532  ],
       [   5.8232 ,   31.71656],
       [   6.0034 ,   39.3312 ],
       ...,
       [   6.3066 ,   23.9292 ],
       [   5.9138 ,  592.151  ],
       [   5.7866 ,   38.9384 ]])
```

In [15]:

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

Out[15]:

```
0.8749008584467053
```

Neural Network model

In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.

In [16]:

```
from keras.models import Sequential
from keras.layers import Dense

def create_model(neurons, activation, optimizer, loss):
    model = Sequential()
    model.add(Dense(neurons, activation=activation, input_shape=(3,)))
    model.add(Dense(neurons, activation=activation))
    model.add(Dense(2, activation='softmax'))

    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

    return model
```

```
Using TensorFlow backend.
```

In this, we define the hyperparameters with two or more options to find the best fit.

In [17]:

```
from keras.wrappers.scikit_learn import KerasClassifier

model = KerasClassifier(build_fn=create_model, verbose=0)

# neurons = [16, 64, 128, 256]
neurons = [16]
# batch_size = [10, 20, 50, 100]
batch_size = [10]
epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'exponent
ial']
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nad
am']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hinge']

param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, act
ivation=activation, optimizer=optimizer, loss=loss)
```

Here, we find the best fit of the above model and get the mean test score and standard deviation of the best fit model.

In [18]:

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)
```

```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_param
s_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.957655 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10
, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.333316 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epo
chs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epo
chs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.957655 (0.029957) with: {'activation': 'relu', 'batch_size': 10, 'epochs
': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.645111 (0.456960) with: {'activation': 'relu', 'batch_size': 10, 'epochs
': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

The best fit parameters are used for same model to compute the score with training data and testing data.

In [19]:

```
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(3,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='SGD', loss='squared_hinge', metrics=['accuracy'])
```

In [20]:

```
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_
data=(X_test, y_test))
```

```
Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 6s 330us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 6s 320us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 6s 320us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 4/20
```

```
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 5/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 6s 323us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 8/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 9/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 10/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 11/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 12/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 13/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 14/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 15/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [==============================] - 6s 323us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 19/20
```

```
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
                                                                   Out[20]:

<keras.callbacks.History at 0x7ff0a8db8cc0>
```

```
                                                                   In [21]:
[test_loss, test_acc] = model.evaluate(X_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(tes
t_loss, test_acc))
```

```
4682/4682 [==============================] - 0s 39us/step
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9
241777017858995
```

We see that the above model performs better but it also has lot of noise (loss) which can be neglected for prediction and use it for furthur prediction.

The above model is saved for furthur prediction.

```
                                                                   In [22]:
model.save('earthquake.h5')
```

More

Introduction

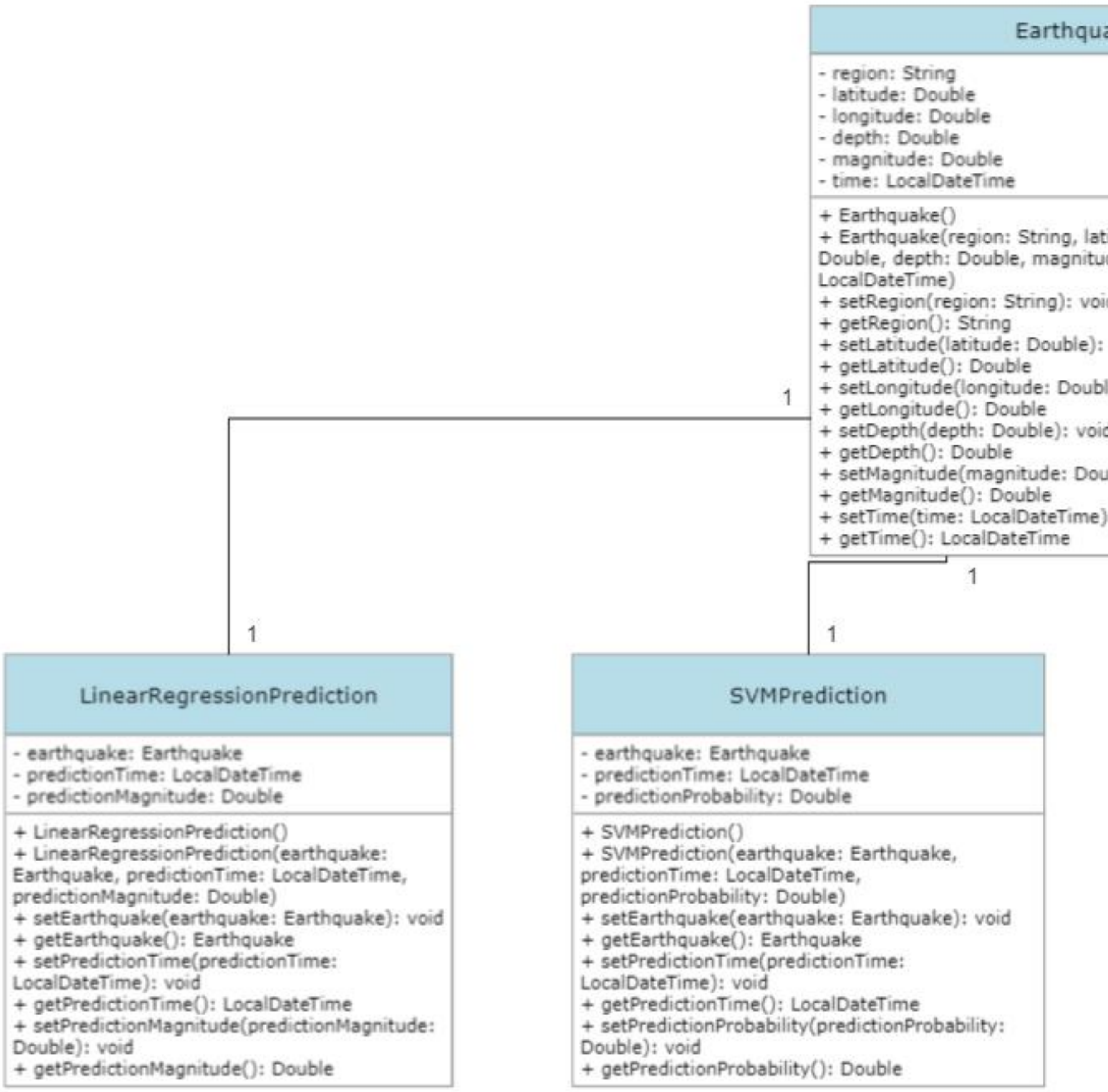
The SOCR Earthquake Dataset can be used to build machine learning models to predict earthquakes or to better understand earthquake patterns and characteristics. Here are a few possible ways machine learning models can be used with this dataset:

1. Earthquake prediction: You can use this dataset to build a model that predicts when and where an earthquake might occur based on past earthquake data. You could use techniques such as time series analysis, clustering, or classification to identify patterns in the data and make predictions.

2. Magnitude prediction: You can use this dataset to build a model that predicts the magnitude of an earthquake based on other factors such as location, depth, or the number of seismic stations that recorded the earthquake. You could use regression techniques to build this model.
3. Risk assessment: You can use this dataset to identify areas that are at higher risk of earthquakes based on historical earthquake data. You could use clustering or classification techniques to identify patterns in the data and identify areas with similar characteristics.
4. Anomaly detection: You can use this dataset to detect anomalies or outliers in the data, which could represent earthquakes that are unusual or unexpected. You could use techniques such as clustering or classification to identify patterns in the data and detect anomalies.
5. Data visualization: You can use this dataset to create visualizations of earthquake data, which could help you identify patterns and relationships in the data. You could use techniques such as scatter plots, heat maps, or geographic information systems (GIS) to visualize the data.

These are just a few examples of the many ways that machine learning models can be used with the SOCR Earthquake Dataset. The specific approach you take will depend on your research question and the goals of your analysis. In this project we focus mainly on Earthquake prediction and Magnitude prediction.

Class diagram


Earthqua
- region: String
- latitude: Double
- longitude: Double
- depth: Double
- magnitude: Double
- time: LocalDateTime
+ Earthquake()
+ Earthquake(region: String, lat
Double, depth: Double, magnitu
LocalDateTime)
+ setRegion(region: String): voi
+ getRegion(): String
+ setLatitude(latitude: Double):
+ getLatitude(): Double
+ setLongitude(longitude: Doubl
+ getLongitude(): Double
+ setDepth(depth: Double): voic
+ getDepth(): Double
+ setMagnitude(magnitude: Dou
+ getMagnitude(): Double
+ setTime(time: LocalDateTime)
+ getTime(): LocalDateTime
1
1
1
1
LinearRegressionPrediction
- earthquake: Earthquake
- predictionTime: LocalDateTime
- predictionMagnitude: Double
+ LinearRegressionPrediction()
+ LinearRegressionPrediction(earthquake: Earthquake, predictionTime: LocalDateTime, predictionMagnitude: Double)
+ setEarthquake(earthquake: Earthquake): void
+ getEarthquake(): Earthquake
+ setPredictionTime(predictionTime: LocalDateTime): void
+ getPredictionTime(): LocalDateTime
+ setPredictionMagnitude(predictionMagnitude: Double): void
+ getPredictionMagnitude(): Double
SVMPrediction
- earthquake: Earthquake
- predictionTime: LocalDateTime
- predictionProbability: Double
+ SVMPrediction()
+ SVMPrediction(earthquake: Earthquake, predictionTime: LocalDateTime, predictionProbability: Double)
+ setEarthquake(earthquake: Earthquake): void
+ getEarthquake(): Earthquake
+ setPredictionTime(predictionTime: LocalDateTime): void
+ getPredictionTime(): LocalDateTime
+ setPredictionProbability(predictionProbability: Double): void
+ getPredictionProbability(): Double

Data visualization

Software used: **Tableau**

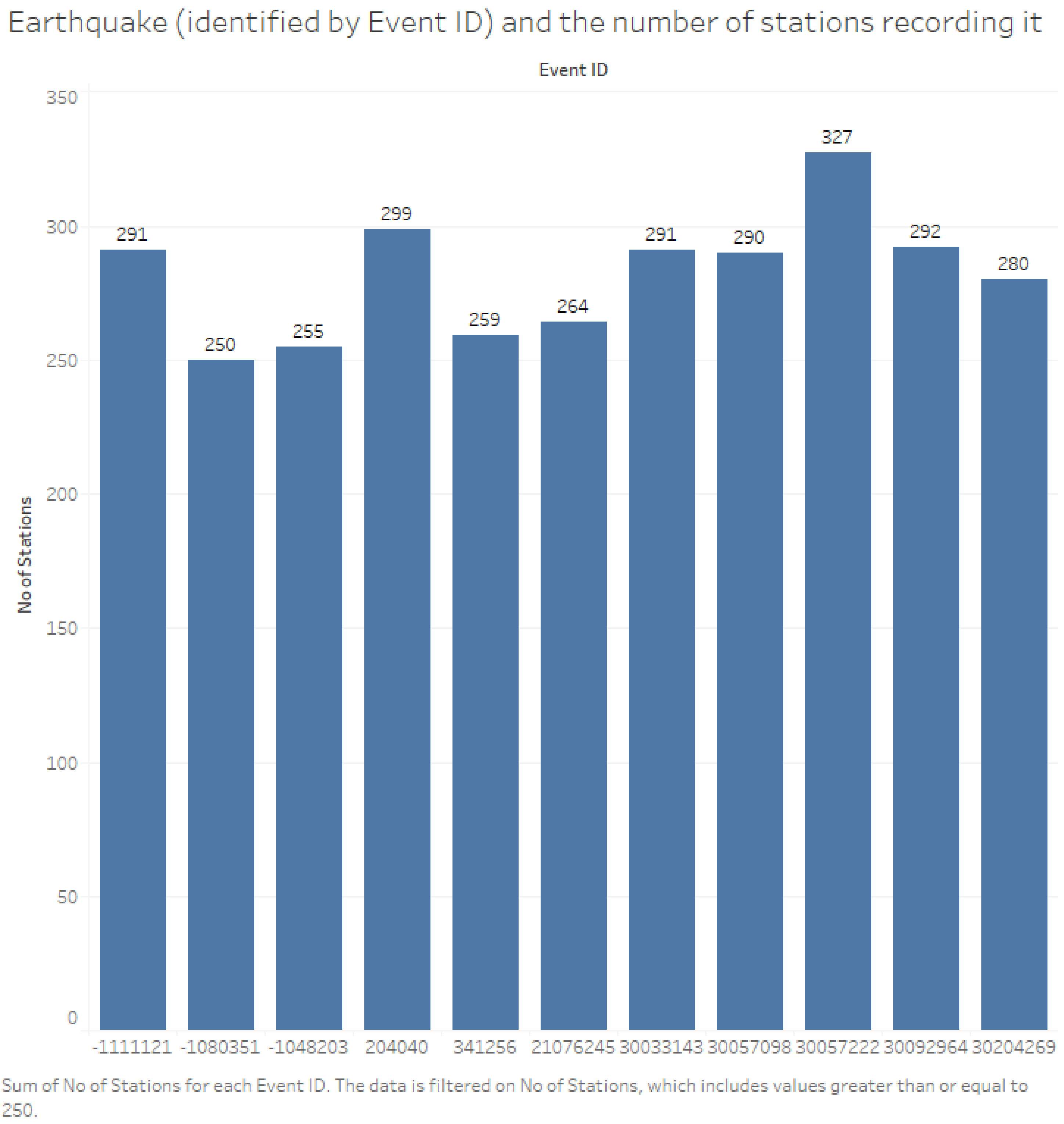
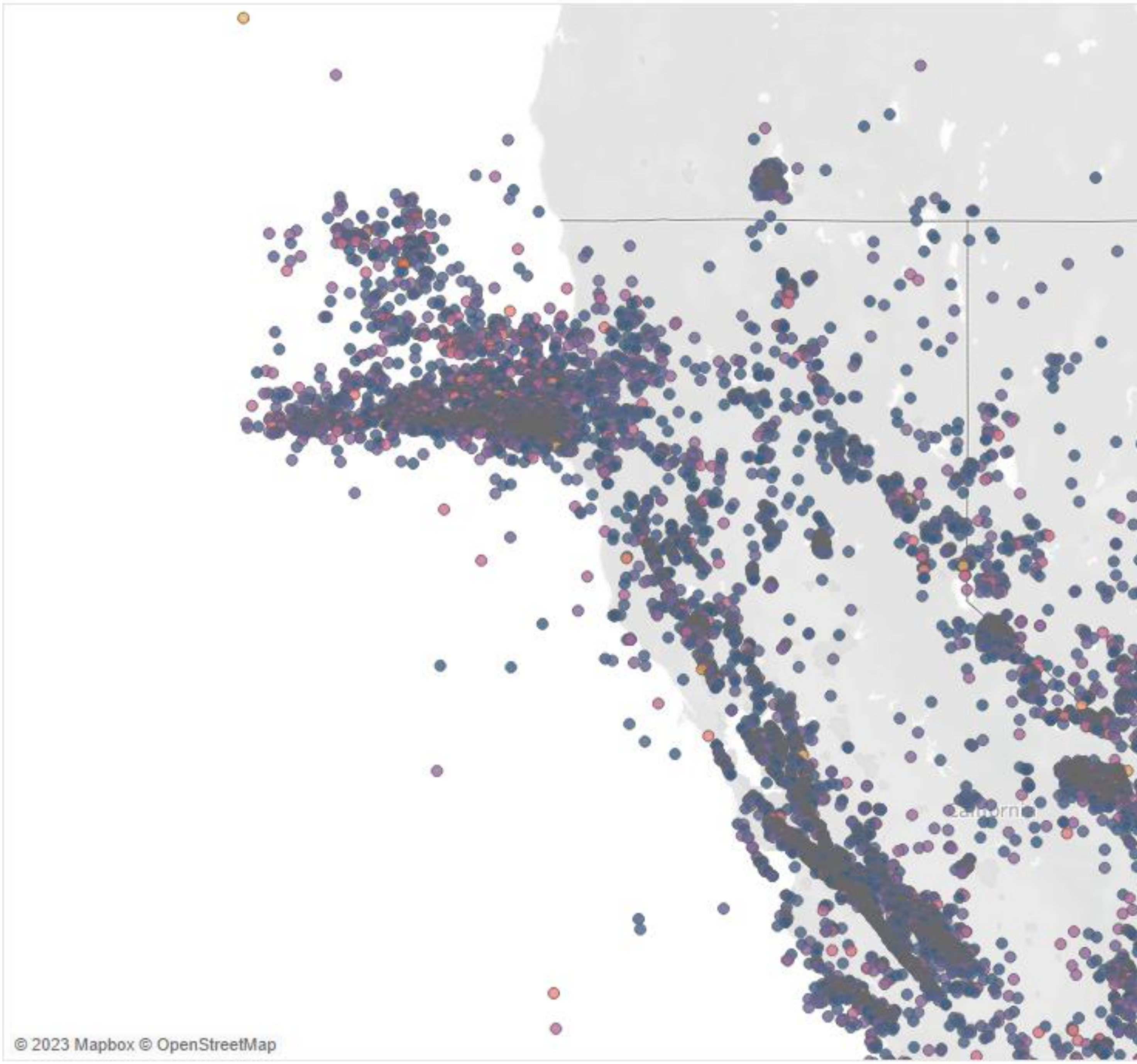


Figure 1

Earthquake (identified by Event ID) and the number of stations recording it

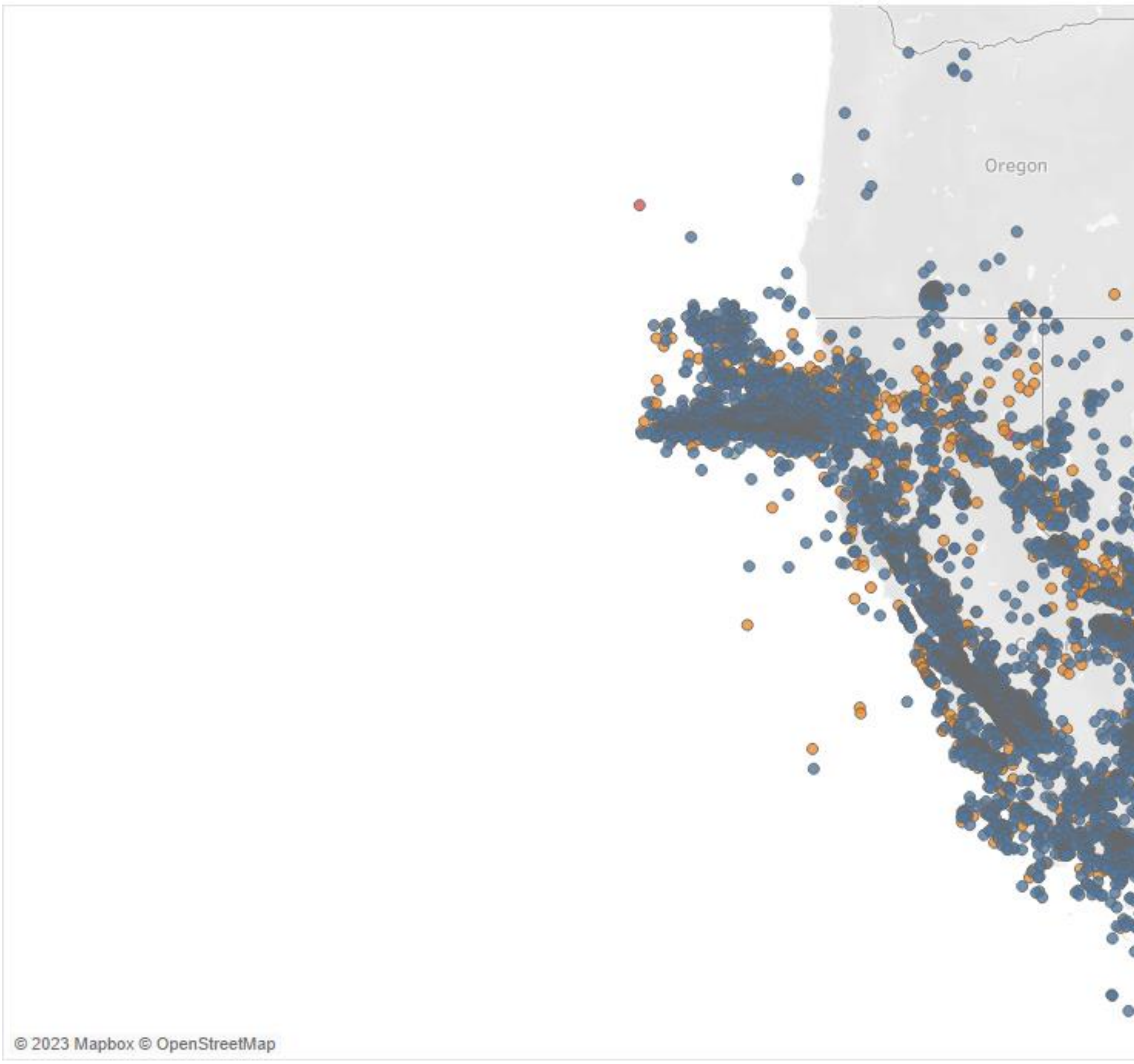
Earthquake based on its magnitude



Map based on Longitude (generated) and Latitude (generated). Color shows sum of Magnitude(ergs). Details are shown for Latitu

Figure 2
Earthquake based on its magnitude

Earthquake based on its magnitude type



Map based on Longitude (generated) and Latitude (generated). Color shows details about Magnitude type. Details are shown for L

Figure 3

EEarthquake based on its magnitude type

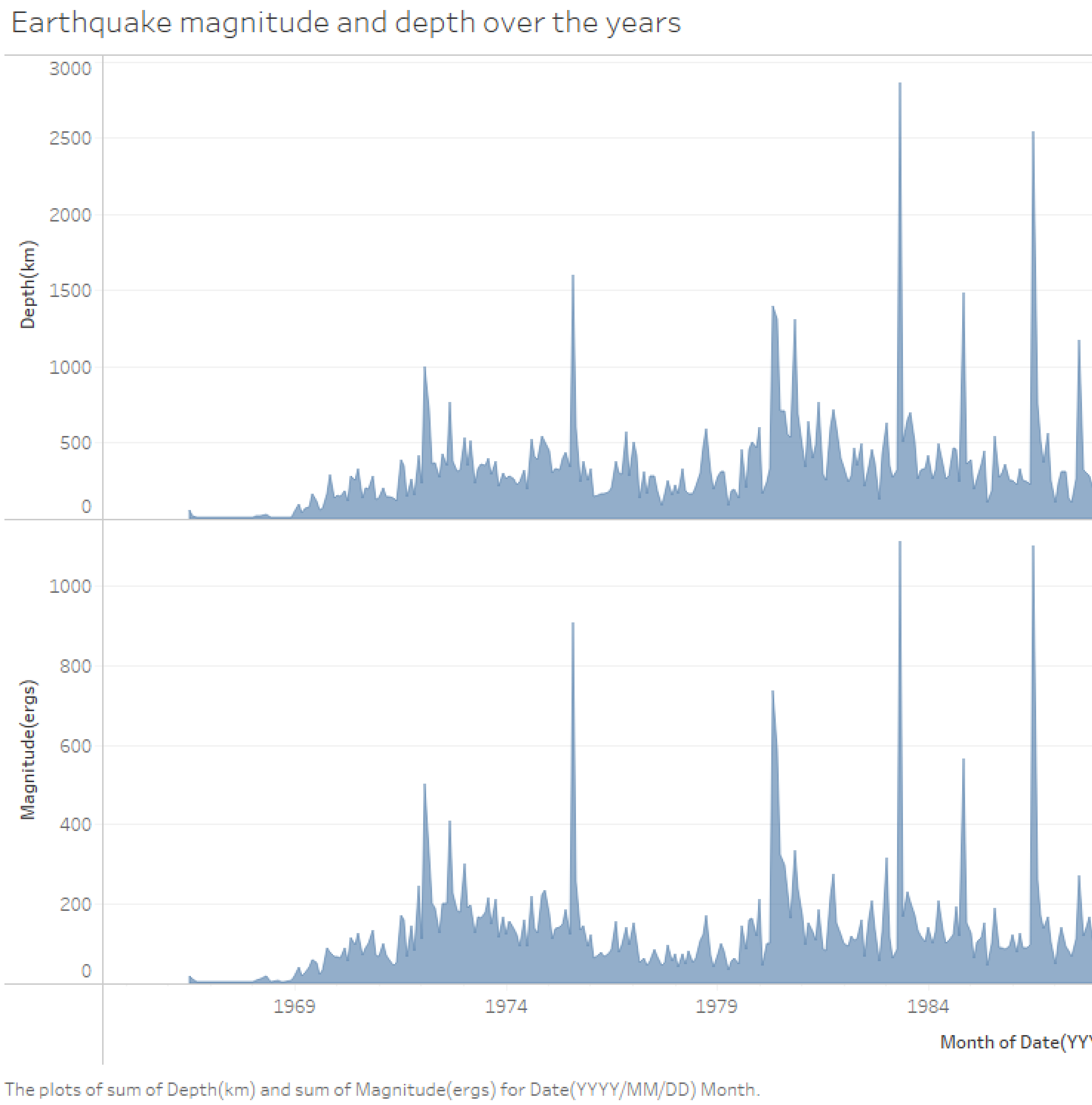


Figure 4
Earthquake magnitude and depth over the years

Implementation

We will use four models in this project:

1. Linear regression
2. Support Vector Machine(SVM)
3. NaiveBayes
4. Random Forest

Linear Regression

Linear regression is a type of supervised machine learning algorithm that is used to model the linear relationship between a dependent variable (in this case, earthquake magnitude) and one or more independent variables (in this case, latitude, longitude, depth, and the number of seismic stations that recorded the earthquake).

The basic idea behind linear regression is to find the line of best fit through the data that minimizes the sum of the squared residuals (the difference between the predicted and actual values of the dependent variable). The coefficients of the line of best fit are estimated using a method called ordinary least squares, which involves minimizing the sum of the squared residuals with respect to the coefficients.

In this situation, we have used multiple linear regression to model the relationship between earthquake magnitude and latitude, longitude, depth, and the number of seismic stations that recorded the earthquake. The multiple linear regression model assumes that there is a linear relationship between the dependent variable (magnitude) and each of the independent variables (latitude, longitude, depth, and number of seismic stations), and that the relationship is additive (i.e., the effect of each independent variable on the dependent variable is independent of the other independent variables).

Once the model has been fit to the data, we can use it to predict the magnitude of a new earthquake given its latitude, longitude, depth, and the number of seismic stations that recorded it. This can be useful for earthquake monitoring and early warning systems, as well as for understanding the underlying causes of earthquakes and improving our ability to predict them in the future.

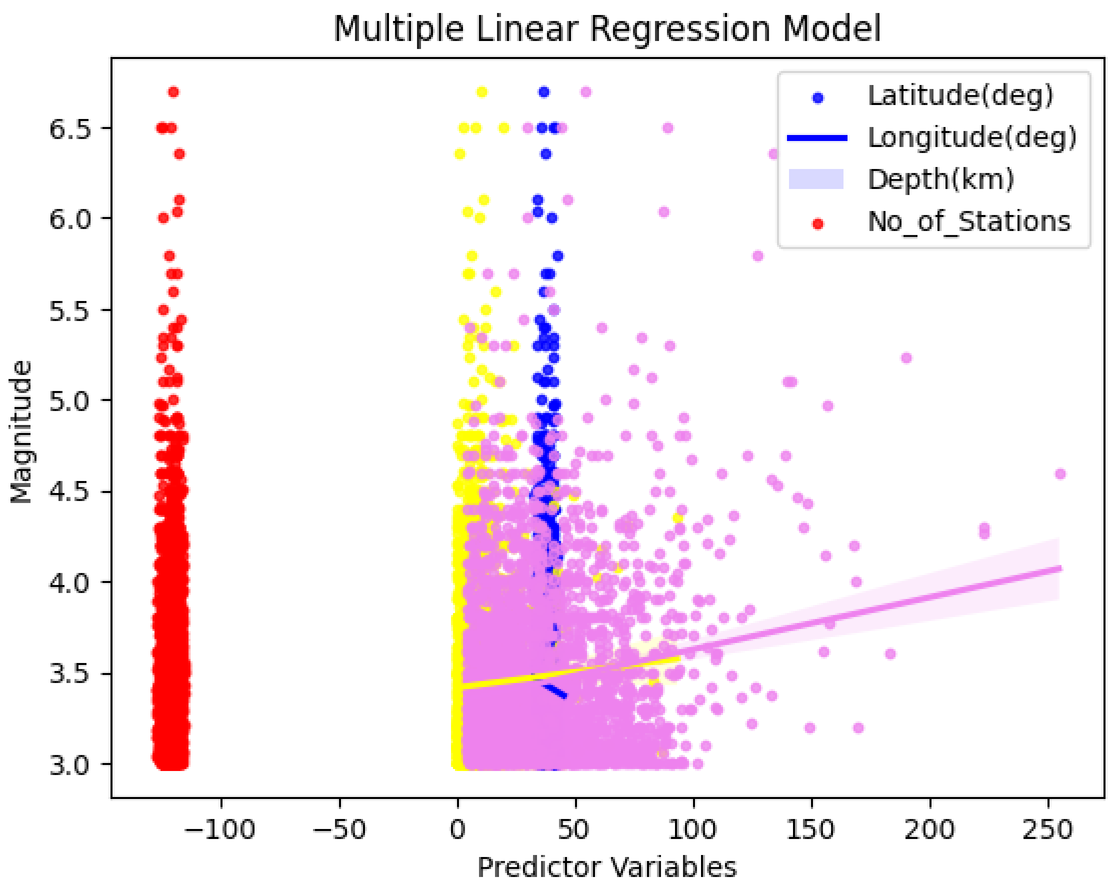


Figure 5

Multiple linear regression plot using seaborn library(python)

The linear regression equation used in our multiple linear regression model for earthquake magnitude prediction with latitude, longitude, depth, and number of seismic stations as independent variables can be written as:

Magnitude = -0.6028 * Latitude + 1.2012 * Longitude - 0.0008 * Depth + 0.0239 * No_of_stations + 0.1573

Where:

- Magnitude is the dependent variable, representing the magnitude of the earthquake
- Latitude, Longitude, Depth, and No_of_stations are the independent variables
- The coefficients (-0.6028, 1.2012, -0.0008, and 0.0239) represent the slopes of the regression line for each independent variable
- The intercept (0.1573) represents the predicted magnitude when all independent variables are zero.

- This equation allows us to predict the magnitude of an earthquake based on its latitude, longitude, depth, and the number of seismic stations that recorded it. By plugging in the values of the independent variables for a given earthquake, we can obtain an estimate of its magnitude.

The results we obtained from the linear regression model were as follows:

- Mean squared error (MSE): 0.17562
- R-squared (R2) score: 0.03498

SVM

Support Vector Machines (SVM) is a type of supervised machine learning algorithm that can be used for both regression and classification tasks. The basic idea behind SVM is to find the best boundary that separates the data into different classes or predicts a continuous output variable (in this case, earthquake magnitude).

In SVM, the data points are mapped to a higher-dimensional space where the boundary can be easily determined. The best boundary is the one that maximizes the margin, which is the distance between the boundary and the closest data points from each class. This boundary is called the "hyperplane."

For regression tasks, SVM uses a similar approach but instead of a hyperplane, it finds a line (or curve in higher dimensions) that best fits the data while maximizing the margin. This line is the "support vector regression line."

SVM can handle both linear and non-linear data by using different kernels that transform the data into a higher-dimensional space. Some commonly used kernels include linear, polynomial, and radial basis function (RBF) kernels.

Once the SVM model has been trained on the data, it can be used to predict the magnitude of a new earthquake given its features (latitude, longitude, depth, and number of seismic stations). This can be useful for predicting the magnitude of earthquakes in real-time and for better understanding the factors that contribute to earthquake occurrence.

Figure 6
SVM plot using matplotlib.pyplot library(python)

The predicted values from SVM model when evaluated using mse and r2 metrics:

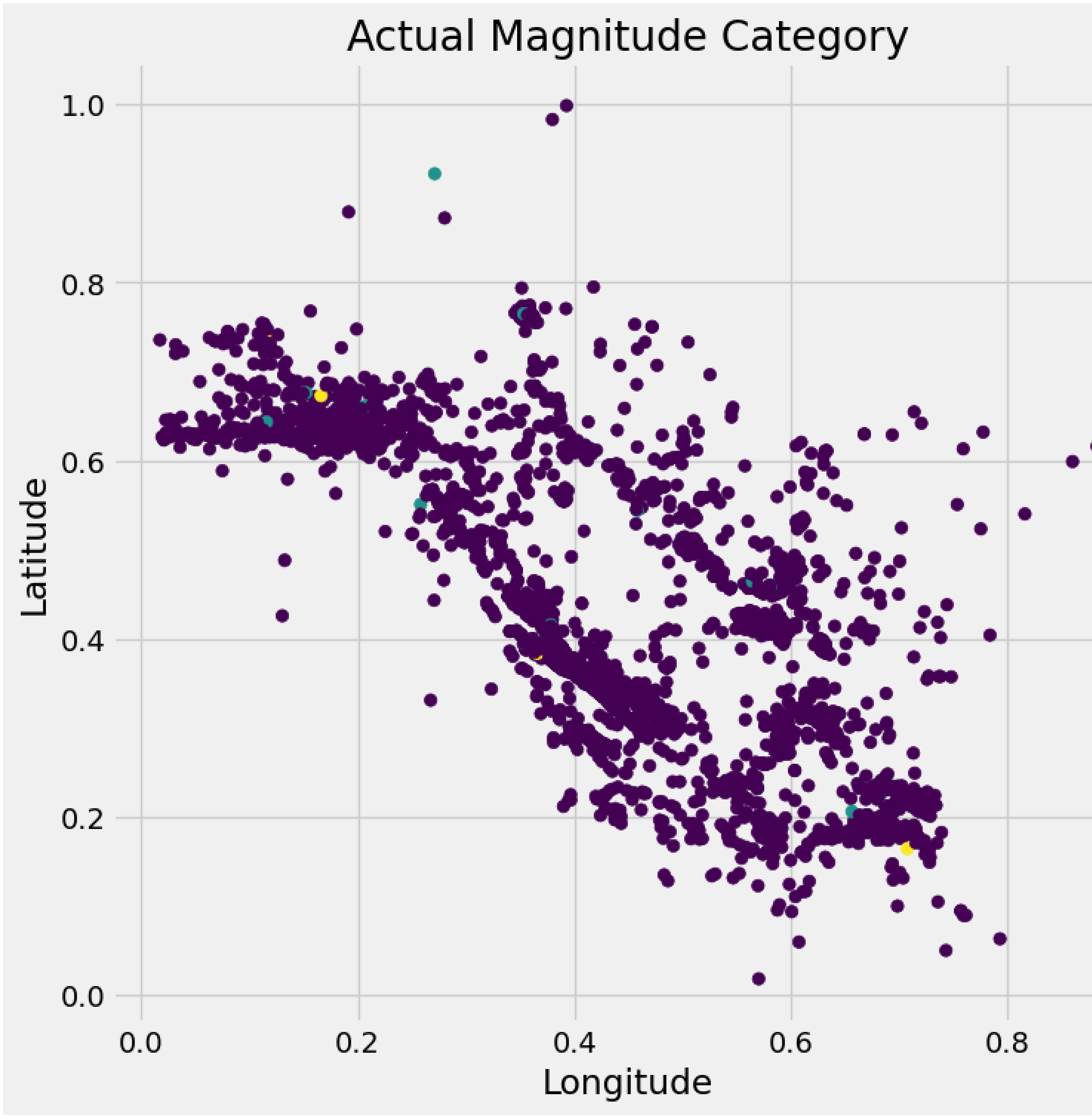
- Mean squared error (MSE): 0.53166
- R-squared (R2) score: -1.92129

Naive Bayes

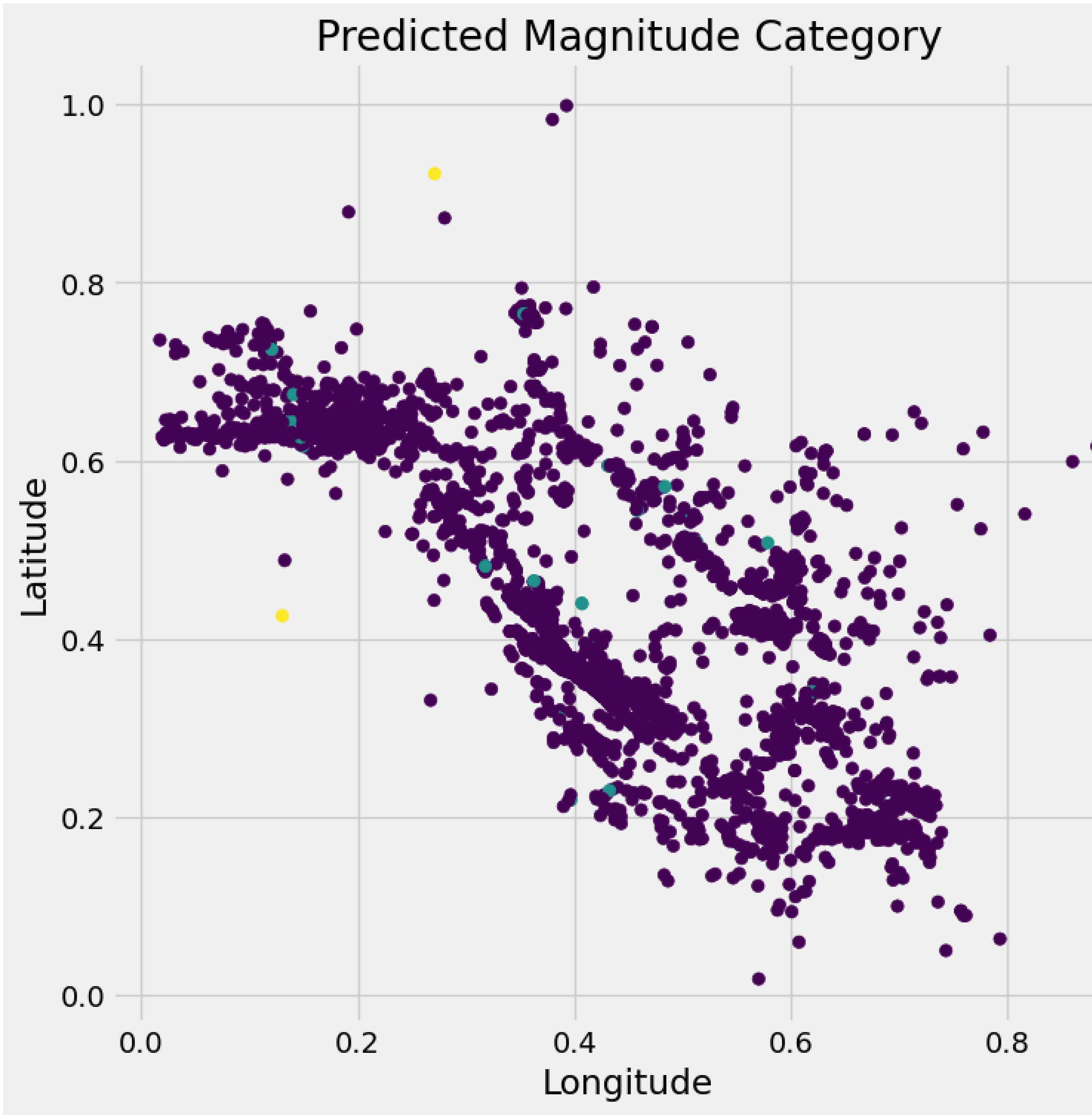
In statistics, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features (see Bayes classifier). They are among the simplest Bayesian network models,[1] but coupled with kernel density estimation, they can achieve high accuracy levels.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression,[3]: 718 which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the code, we used the Naive Bayes classifier to predict the magnitude of earthquakes based on their latitude, longitude and number of monitoring stations. We split the data into training and testing sets, trained the Naive Bayes model on the training data, and evaluated its performance on the test data using the accuracy score, confusion matrix and classification report



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



Figure

Actual vs Predicted

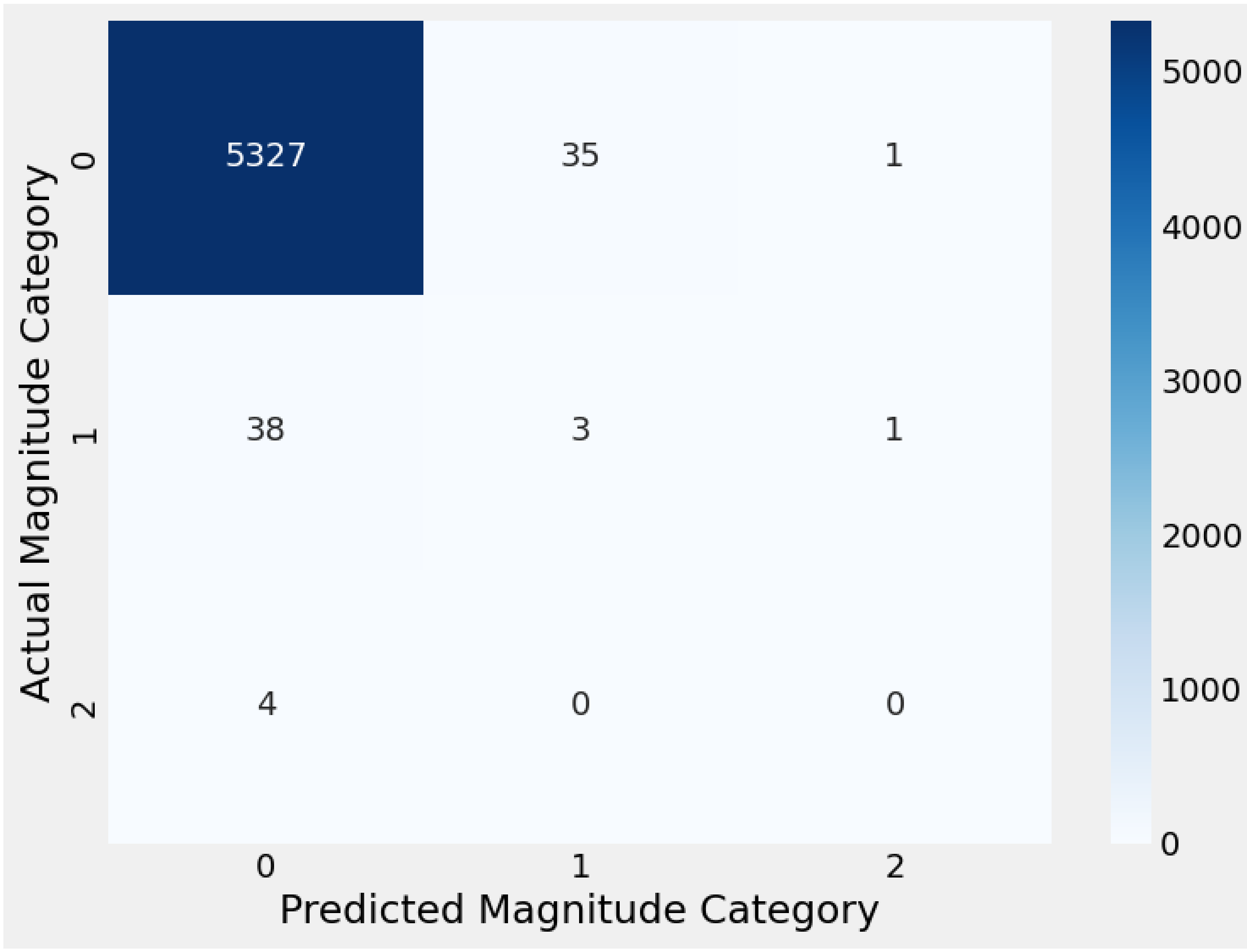


Figure
Heatmap of Confusion Matrix

- Accuracy: 0.9853947125161767
- Confusion Matrix: [[5327 35 1] [38 3 1] [4 0 0]]

Random Forest

Random forest is a machine learning algorithm that is used for both classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to create a more accurate and robust model.

The basic idea behind random forest is to create multiple decision trees, each trained on a subset of the data and a random subset of the features. Each tree makes a prediction, and the final prediction is the average (for regression) or the mode (for classification) of the individual tree predictions. By creating many trees and taking their average, random forest can reduce the impact of overfitting and improve the accuracy and stability of the model.

In the code we provided earlier, we used the random forest algorithm to predict the magnitude of earthquakes based on their latitude, longitude, depth, and number of monitoring stations. We split the data into training and testing sets, trained the random forest model on the training data, and evaluated its performance on the test data using the mean squared error (MSE) and R-squared (R2) score.

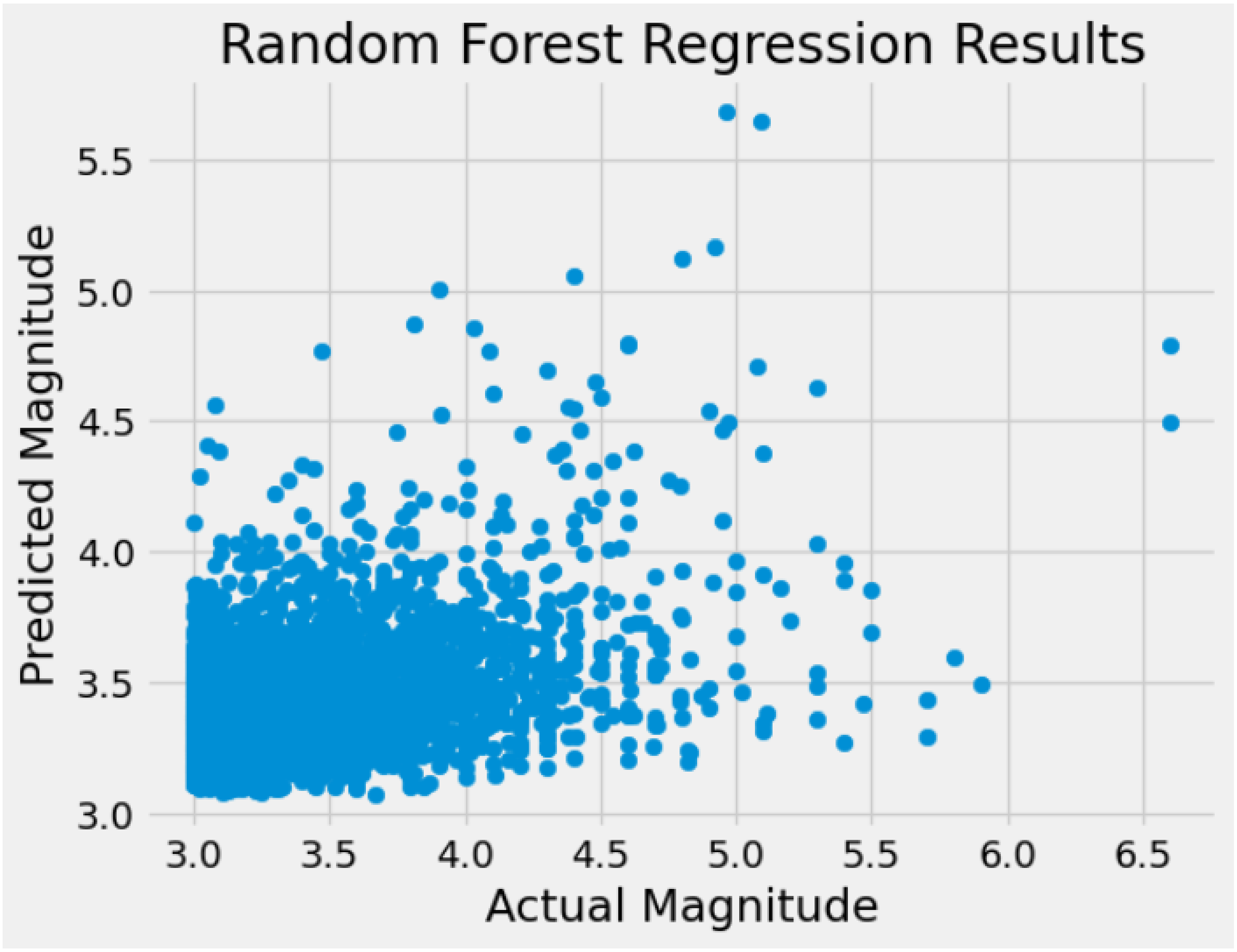


Figure 7

Actual vs Predicted

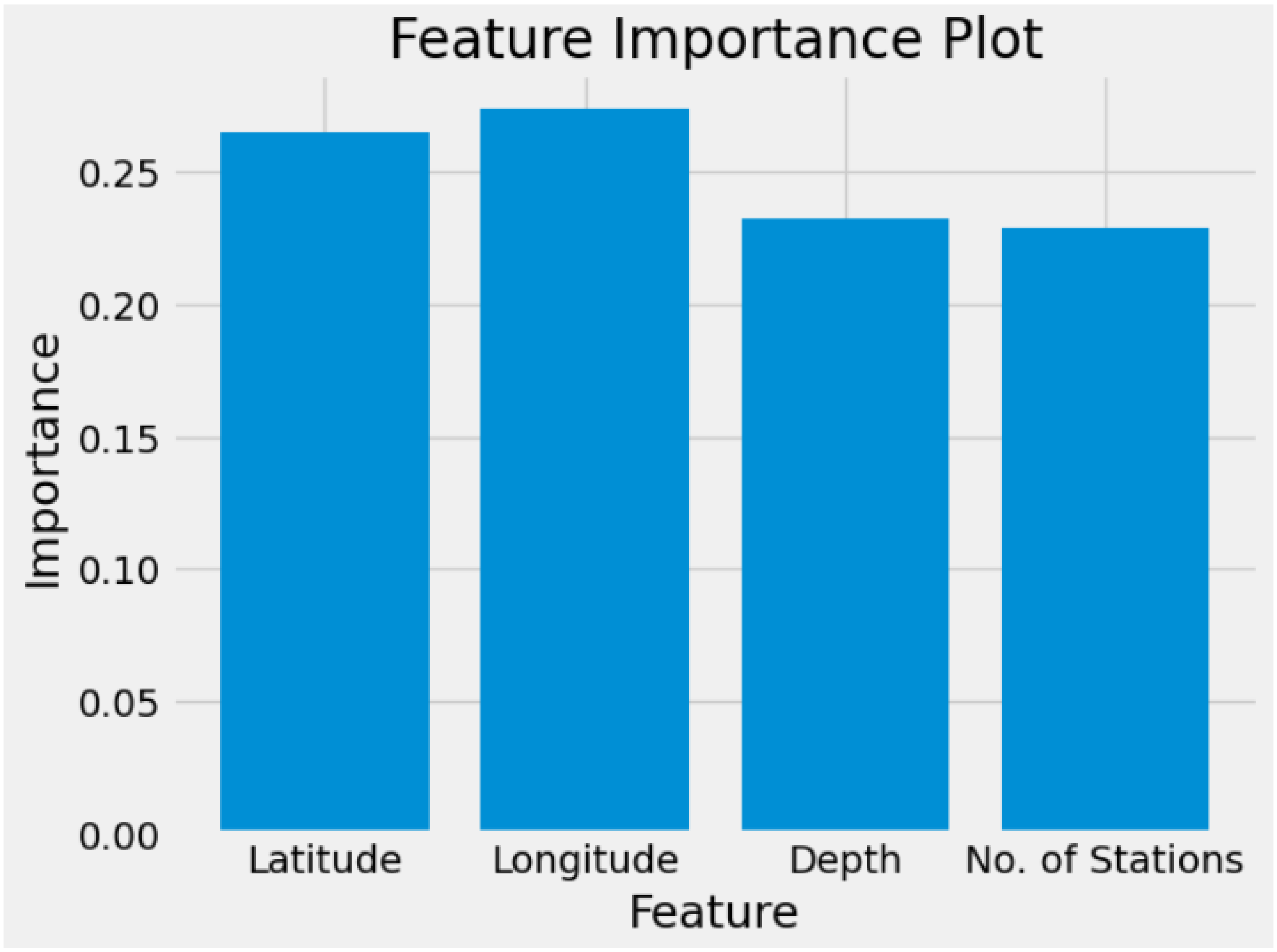


Figure 8

Feature Importance Plot

The results we obtained from the random forest model were as follows:

- Mean squared error (MSE): 0.15599
- R-squared (R2) score: 0.14288

These results indicate that the random forest model was able to accurately predict the magnitude of earthquakes based on the given features. The low MSE and high R2 score indicate that the model was making accurate predictions, and was able to explain a large proportion of the variance in the target variable.

Overall, the random forest algorithm is a powerful tool for machine learning tasks, and can be used in a variety of applications, including finance, healthcare, and image recognition

Conclusion

When comparing two models, both the mean squared error (MSE) and R-squared (R2) score can be used to evaluate the performance of the models.

In general, a model with a lower MSE and a higher R2 score is considered a better model. This is because the MSE measures the average difference between the predicted and actual values, and a lower MSE indicates that the model is making more accurate predictions. The R2 score measures the proportion of the variance in the target variable that is explained by the model, and a higher R2 score indicates that the model is able to explain more of the variability in the target variable.

From the results of this project 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.

However, it's important to keep in mind that the relative importance of MSE and R2 score may vary depending on the specific problem and the context in which the models are being used. For example, in some cases, minimizing the MSE may be more important than maximizing the R2 score, or vice versa. It's also possible that one model may perform better on one metric and worse on another, so it's important to consider both metrics together when evaluating the performance of the models.