STAR LION COLLEGE OF ENGINEERING AND TECHNOLODGY

Program : Earthquake Prediction model using python
Name : S. Bharaniga
Date : 26.10.2023
Register No : 822021104004
Nanmuthalvan ID :au822021104004

Earthquake Detection Model Using Python

Abstract:

We attempt to automatically detect earthquake events in distributed acoustic sensing (DAS) data via.

A supervised learning approach. Detecting earthquakes with different magnitudes could potentially.

Provide the ability of predicting major catastrophic events**.**

Introduction:

Distributed acoustic sensing (DAS) is an emerging technology used to record seismic data that employs.

Fiber optic cables as a probing system. By measuring the backscattered energy of a pulsing laser transmit.

Ted down a fiber optic cable, it is possible to measure the strain rate occurring within different sections.

Of the cable [1]. DAS recording systems have been shown to measure data comparable with conventional.

Geophones [2] and have been successfully used in exploration and earthquake seismology settings [3, 4].

Dataset And Preprocessing:

The fiber optic cable is deployed in Stanford's telecommunication tunnels in a double loop pattern.

Every 8 meters of cable acts as a receiver and records vibration at a sampling rate of 50Hz, creating a data.

Matrix of 300 channels distributed in space, each continuously recording cable strain since September.

2016. The array generates contiguous time series that conveniently lend themselves to image processing.

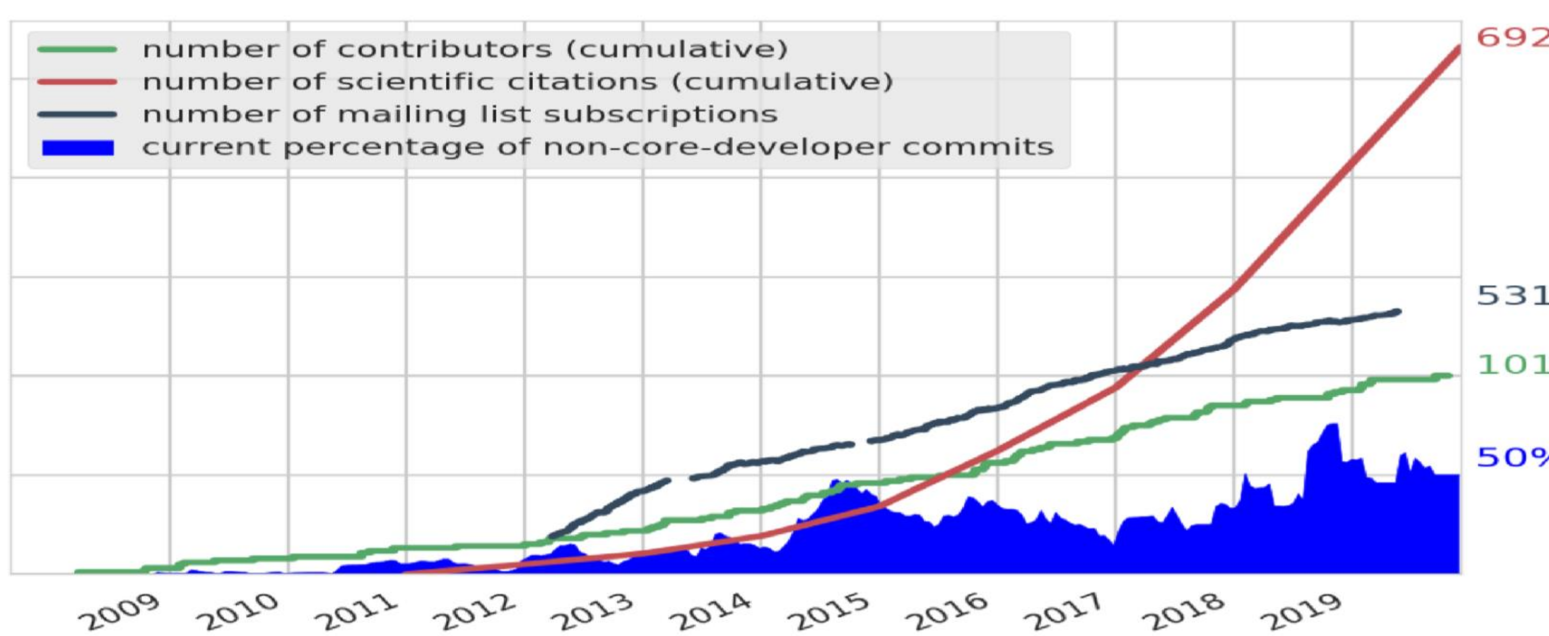
Labeling The Data :

We labeled the various categories by using complementary data from other sources. For the cars.

Proceeded with a methodology similar to [9], where a clustering algorithm (K-means) was applied.

Data in the continuous wavelet domain (CWT) and was able to separate different types of seismic signals.

By projecting the data over the array's geometry, a human supervisor can easily hand pick



the clusters.

Corresponding to traffic noise.

Results and Discussion:

In order to understand how much each class is separable in the single amplitude feature space, in Figure.

4a we show the amplitude as a function of class number. We clearly notice that the three classes entirely

Overlap, thus they are not linearly separable. In addition, Figure 4b displays the normalized histogramOf the amplitudes for each class.

Feature Extraction:

In the context of ML-based earthquake detection, amplitude and frequency are the two key pieces of information among different statistics of the accelerometer signal.

IQR (Interquartile Range): IQR is the interquartile range $Q3$

$-Q1$

Of the 3 component vector sum VS

;

$VS=X2+Y2+Z2$−−−−−−−−−−−√

(1)where X, Y, and Z are the acceleration components.

CAV

(Cumulative Absolute Velocity): CAV

Feature is the cumulative measure of the VS

In the time window and is calculated as

$CAV=\int s0|VS(t)|dt$

(2)where s is the total time period of the feature window in seconds, and t is the time. In this work, we used a twosecond feature window.

ZC

(Zero-Crossing): ZC

Is the maximum zero-crossing rate of X, Y, and Z component and the zero-crossing rate of component X can be calculated as:

$ZCX=1N-1\sum t=1N-11\mathbb{R}<0(XtXt-1)$

(3)where N is the total length of the signal X and $1\mathbb{R}<0$

Is indicator function.

IQR and CAV are the amplitude features, while ZC is the frequency feature, and these are proposed in [6,30].

These features detect earthquakes and can discriminate non-earthquake data, but through exhaustive experimental evaluations and also its implementation in the static environment as given in our previous work, we found that in a noisy environment (noisy sensors or external events), its performance can be degraded.

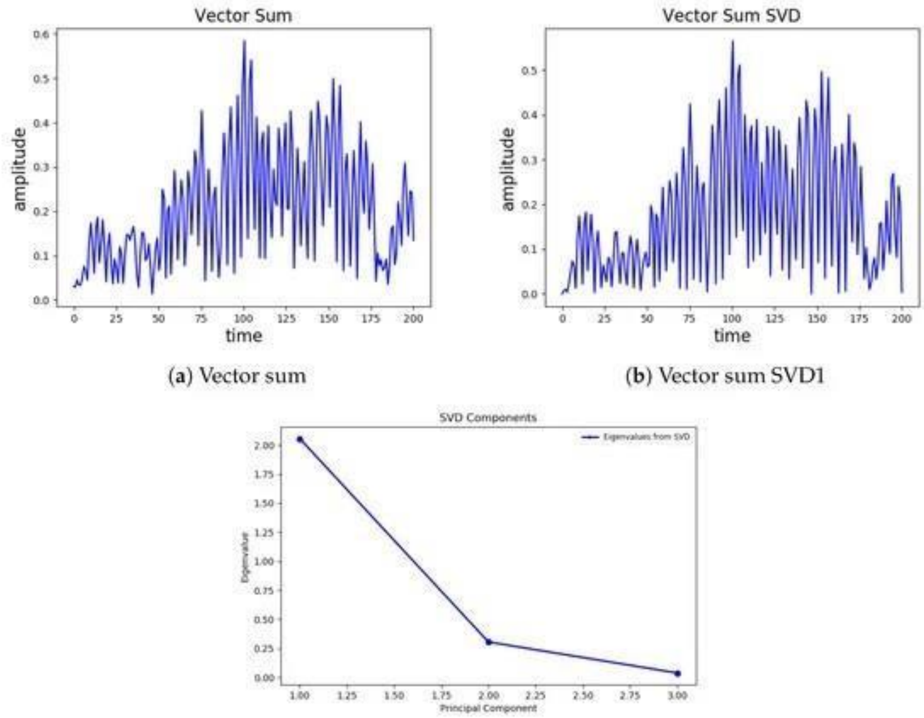
Where, A is an M x N matrix, where M represents two-second points, i.e., 200, and N is 3. SVD provides three new vectors $UMxM$, $SMxN$, $and\ VTNxN$

, which, if linearly combined, give back the approximated original vector; where U is the set of singular vectors with singular values in vector S, VT

Is the primary direction. The new vectors are ordered, and the first vector explains most of the original acceleration amplitude and frequency information, as shown in Figure 3. Figure 3a depict almost the same structure; therefore, we select the first vector as a primary vector U[:,0]

From the given SVD's, along with the first value S[0]

Of S, which is a scaling factor (give amplitude information of the given vector). We extracted the following three additional features.



(a) Vector sum

(b) Vector sum SVD1

(c) Scree plot

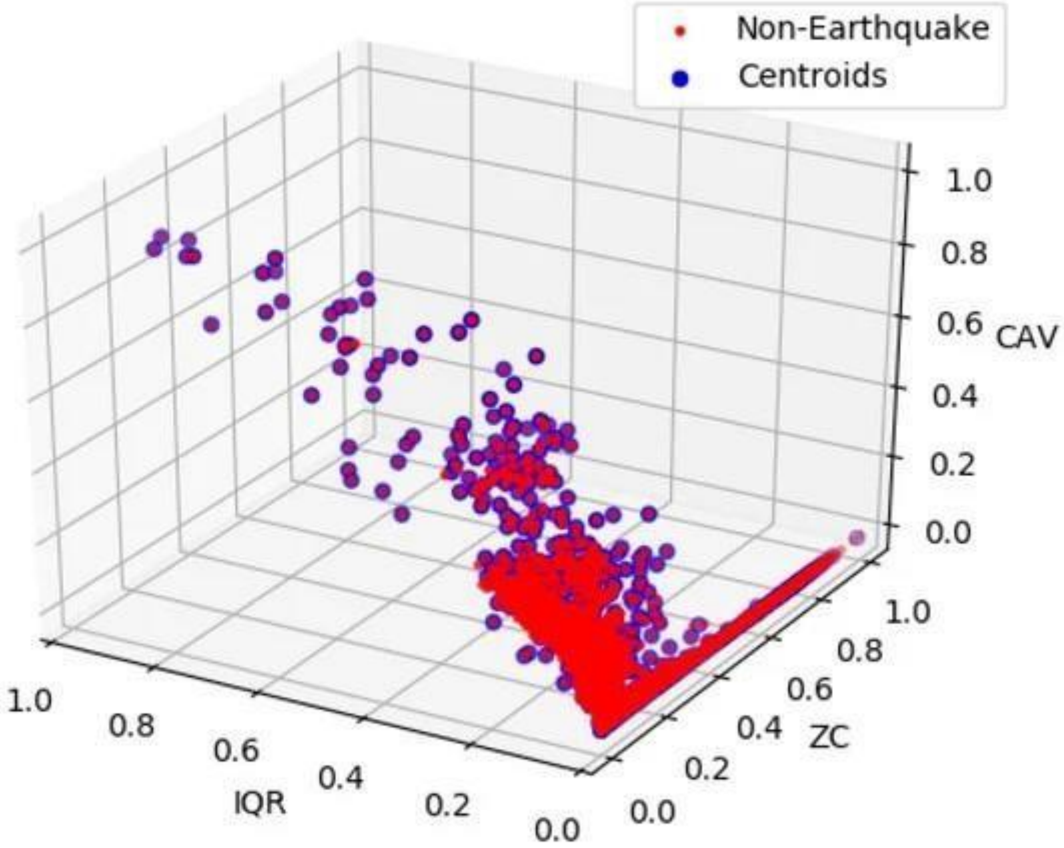
A two-second window of the strongest portion of the earthquake; (a) The vector sum of X, Y, and Z; (b) vector sum of the primary vector of SVD (center); (c) scree plot of the three components.

Pre-Processing:

In our methodology, the pre-processing involved balancing the dataset and scaling the features to range from 0 to 1. Balancing is required because the imbalanced datasets greatly affect the performance of the machine learning model [34]. In our case, the non-earthquake dataset (noise and human activities) is much larger than the earthquake dataset.

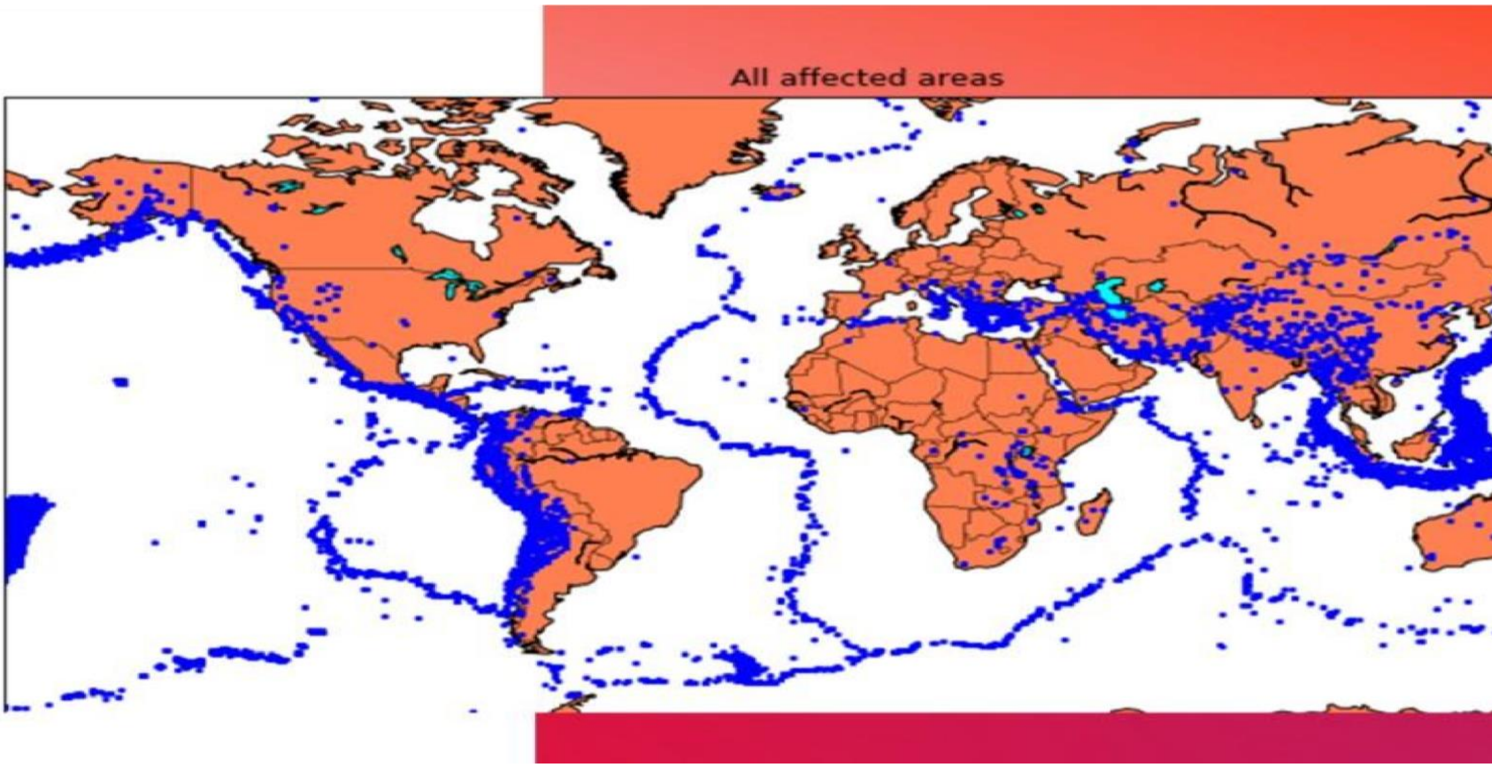
From mpl_toolkits.basemap import Basemap

```
M = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=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)


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()
```



Non-Earthquake
Centroids
CAV
1.0
0.8
0.6
0.4
0.2
0.0
IQR
1.0
0.8
0.6
0.4
0.2
0.0
ZC
0.0
0.2
0.4
0.6
0.8
1.0

Output:



EXTRACT, TRANSFORM, LOAD (ETL) :

Once we have successfully running PySpark in Jupyter Notebook now we can load the dataset from the local directory.

```
In [3]: # Load the dataset
        df_load = spark.read.csv(r"C:\Users\Intel X Nvidia\Downloads\database.csv", header=True)
        # Preview df_load
        df_load.take(1)

Out[3]: [Row(Date='01/02/1965', Time='13:44:18', Latitude='19.246', Longitude='145.616', Type='Earthquake', Depth='131.6', Depth Error=
        None, Depth Seismic Stations=None, Magnitude='6', Magnitude Type='MW', Magnitude Error=None, Magnitude Seismic Stations=None, A
        zimuthal Gap=None, Horizontal Distance=None, Horizontal Error=None, Root Mean Square=None, ID='ISCGEM860706', Source='ISCGEM',
        Location Source='ISCGEM', Magnitude Source='ISCGEM', Status='Automatic')]
```

After we load the dataset we can preview the column using df.take() this function help to show us the specific row. In the output, we can see the dataset contains many columns hence we don’t need to use all the columns. So we drop column we don’t need using df.drop().

```
In [4]: # Drop fields we don't need from df_load
        lst_dropped_columns = ['Depth Error', 'Time', 'Depth Seismic Stations','Magnitude Error','Magnitude Seismic Stations','Azimuthal
            'Root Mean Square','Source','Location Source','Magnitude Source','Status']

        df_load = df_load.drop(*lst_dropped_columns)
        # Preview df_load
        df_load.show(5)
```

```
+----------+--------+---------+----------+-----+---------+--------------+------------+
|      Date|Latitude|Longitude|      Type|Depth|Magnitude|Magnitude Type|          ID|
+----------+--------+---------+----------+-----+---------+--------------+------------+
|01/02/1965|  19.246|  145.616|Earthquake|131.6|        6|            MW|ISCGEM860706|
|01/04/1965|   1.863|  127.352|Earthquake|   80|      5.8|            MW|ISCGEM860737|
|01/05/1965| -20.579| -173.972|Earthquake|   20|      6.2|            MW|ISCGEM860762|
|01/08/1965| -59.076|  -23.557|Earthquake|   15|      5.8|            MW|ISCGEM860856|
|01/09/1965|  11.938|  126.427|Earthquake|   15|      5.8|            MW|ISCGEM860890|
+----------+--------+---------+----------+-----+---------+--------------+------------+
only showing top 5 rows
```

Now we can see only the columns that we need to operate, the dataset is much cleaner now. After we sorting the column now the thing we need to do is append the “Year” column into the dataframe. Before we add it to the dataframe we need to convert the type of “Date” column into “the timestamp” because the original type of “Date” is an “object” which is “object” type cannot be extracted. So we can simply do this:

```
In [5]: # Create a year field and add it to the dataframe
        df_load = df_load.withColumn('Year', year(to_timestamp('Date', 'dd/MM/yyyy')))
        # Preview df_load
        df_load.show(5)
```

```
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
|      Date|Latitude|Longitude|      Type|Depth|Magnitude|Magnitude Type|          ID|Year|
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
|01/02/1965|  19.246|  145.616|Earthquake|131.6|        6|            MW|ISCGEM860706|1965|
|01/04/1965|   1.863|  127.352|Earthquake|   80|      5.8|            MW|ISCGEM860737|1965|
|01/05/1965| -20.579| -173.972|Earthquake|   20|      6.2|            MW|ISCGEM860762|1965|
|01/08/1965| -59.076|  -23.557|Earthquake|   15|      5.8|            MW|ISCGEM860856|1965|
|01/09/1965|  11.938|  126.427|Earthquake|   15|      5.8|            MW|ISCGEM860890|1965|
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
only showing top 5 rows
```

After we add the “Year” column into the dataframe now we can count how many quakes occurred in each year. We can use groupBy() and count():

```
In [6]: # Build the quakes frequency dataframe using the year field and counts for each year
        df_quake_freq = df_load.groupBy('Year').count().withColumnRenamed('count', 'Counts')
        # Preview df_quake_freq
        df_quake_freq.show(5)

        +----+------+
        |Year|Counts|
        +----+------+
        |1990|   196|
        |1975|   150|
        |1977|   148|
        |2003|   187|
        |2007|   211|
        +----+------+
        only showing top 5 rows
```

Based on the dataframe we can see that the year column is not sorted sequentially, later we can handle this.

After we count the quakes based on a year now we can check the type of every data in a column like this:

As we can see from the output most of the type of the co

```
In [7]: # Preview df_load schema
        df_load.printSchema()

        root
         |-- Date: string (nullable = true)
         |-- Latitude: string (nullable = true)
         |-- Longitude: string (nullable = true)
         |-- Type: string (nullable = true)
         |-- Depth: string (nullable = true)
         |-- Magnitude: string (nullable = true)
         |-- Magnitude Type: string (nullable = true)
         |-- ID: string (nullable = true)
         |-- Year: integer (nullable = true)
```

As we can see from the output most of the type of the column is a string which is cannot be joined. For that, we need to convert some columns we need from strings into numeric types using cast().

In [8]:
```
# Cast some fields from string into numeric types
df_load = df_load.withColumn('Latitude', df_load['Latitude'].cast(DoubleType()))\
    .withColumn('Longitude', df_load['Longitude'].cast(DoubleType()))\
    .withColumn('Depth', df_load['Depth'].cast(DoubleType()))\
    .withColumn('Magnitude', df_load['Magnitude'].cast(DoubleType()))

# Preview of df_load
df_load.show(5)
```

```
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
|      Date|Latitude|Longitude|      Type|Depth|Magnitude|Magnitude Type|          ID|Year|
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
|01/02/1965|  19.246|  145.616|Earthquake|131.6|      6.0|            MW|ISCGEM860706|1965|
|01/04/1965|   1.863|  127.352|Earthquake| 80.0|      5.8|            MW|ISCGEM860737|1965|
|01/05/1965| -20.579| -173.972|Earthquake| 20.0|      6.2|            MW|ISCGEM860762|1965|
|01/08/1965| -59.076|  -23.557|Earthquake| 15.0|      5.8|            MW|ISCGEM860856|1965|
|01/09/1965|  11.938|  126.427|Earthquake| 15.0|      5.8|            MW|ISCGEM860890|1965|
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
only showing top 5 rows
```

In [9]:
```
# Preview df_load schema
df_load.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)
 |-- Type: string (nullable = true)
 |-- Depth: double (nullable = true)
 |-- Magnitude: double (nullable = true)
 |-- Magnitude Type: string (nullable = true)
 |-- ID: string (nullable = true)
 |-- Year: integer (nullable = true)
```

After we converting the string columns into numeric now we can join the df_max and the df_avg into a new variable called df_quake_freq.

In [10]:
```
# Create avg magnitude and max magnitude fields and add to df_quake_freq
df_max = df_load.groupBy('Year').max('Magnitude').withColumnRenamed('max(Magnitude)', 'Max_Magnitude')
df_avg = df_load.groupBy('Year').avg('Magnitude').withColumnRenamed('avg(Magnitude)', 'Avg_Magnitude')
```

In [11]:
```
# Join df_max, and df_avg to df_quake_freq
df_quake_freq = df_quake_freq.join(df_avg, ['Year']).join(df_max, ['Year'])
df_quake_freq = df_quake_freq.orderBy(asc('Year'))
df_quake_freq.show(5)
```

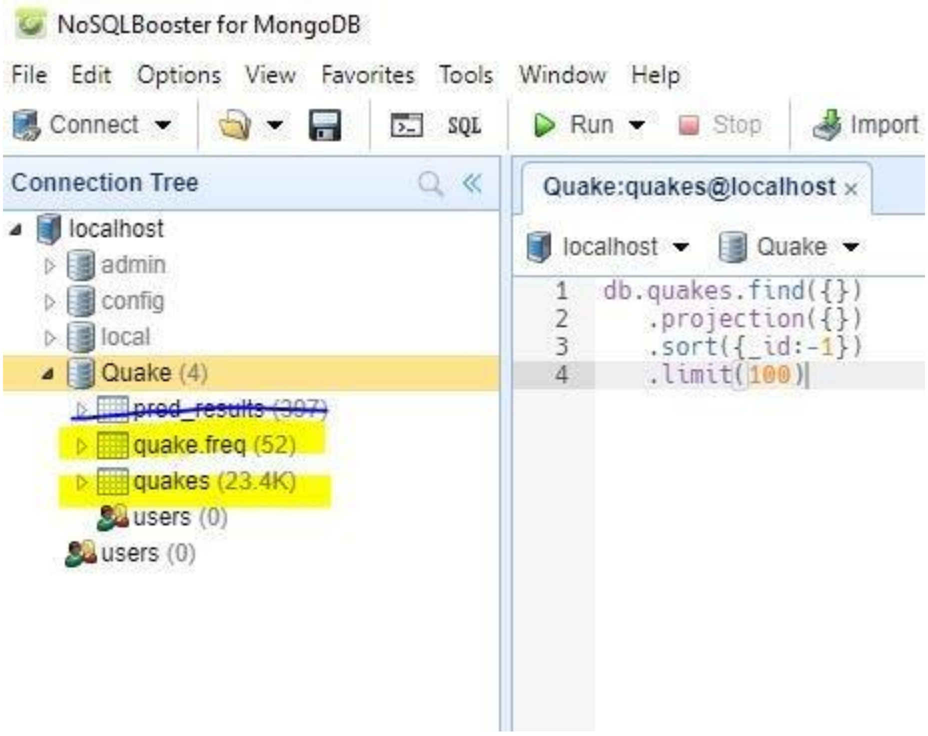
```
+----+------+-----------------+-------------+
|Year|Counts|    Avg_Magnitude|Max_Magnitude|
+----+------+-----------------+-------------+
|1965|   156| 6.009615384615388|          8.7|
|1966|    98| 6.060714285714285|          7.7|
|1967|   103|5.962135922330985|          7.2|
|1968|   106| 6.070754716981133|          7.6|
|1969|   114| 6.015789473684214|          7.5|
+----+------+-----------------+-------------+
only showing top 5 rows
```

```
In [12]: # Remove nulls
         df_load.dropna()
         df_quake_freq.dropna()

Out[12]: DataFrame[Year: int, Counts: bigint, Avg_Magnitude: double, Max_Magnitude: double]

In [13]: # Preview dataframes
         df_load.show(5)

+----------+--------+---------+----------+-----+---------+--------------+------------+----+
|      Date|Latitude|Longitude|      Type|Depth|Magnitude|Magnitude Type|          ID|Year|
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
|01/02/1965|  19.246|  145.616|Earthquake|131.6|      6.0|            MW|ISCGEM860706|1965|
|01/04/1965|   1.863|  127.352|Earthquake| 80.0|      5.8|            MW|ISCGEM860737|1965|
|01/05/1965| -20.579| -173.972|Earthquake| 20.0|      6.2|            MW|ISCGEM860762|1965|
|01/08/1965| -59.076|  -23.557|Earthquake| 15.0|      5.8|            MW|ISCGEM860856|1965|
|01/09/1965|  11.938|  126.427|Earthquake| 15.0|      5.8|            MW|ISCGEM860890|1965|
+----------+--------+---------+----------+-----+---------+--------------+------------+----+
only showing top 5 rows

In [14]: df_quake_freq.show(5)

+----+------+-----------------+-------------+
|Year|Counts|    Avg_Magnitude|Max_Magnitude|
+----+------+-----------------+-------------+
|1965|   156| 6.009615384615388|          8.7|
|1966|    98| 6.060714285714285|          7.7|
|1967|   103|5.9621359223300985|          7.2|
|1968|   106| 6.070754716981133|          7.6|
|1969|   114| 6.015789473684214|          7.5|
+----+------+-----------------+-------------+
only showing top 5 rows
```

After we removed the nulls now the data is ready to use, the next thing we need to do is save the data into MongoDB.

```
In [15]: # Build the tables/collections in mongodb
         # Write df_load to mongodb
         df_load.write.format('mongo')\
             .mode('overwrite')\
             .option('spark.mongodb.output.uri', 'mongodb://127.0.0.1:27017/Quake.quakes').save()

In [16]: # Write df_quake_freq to mongodb
         df_quake_freq.write.format('mongo')\
             .mode('overwrite')\
             .option('spark.mongodb.output.uri', 'mongodb://127.0.0.1:27017/Quake.quake.freq' ).save()
```

To make sure the dataframe is saved properly into MongoDB, we can open the NoSQLBooster and expand the Quake database, if you see this, it means the data is already stored successfully, ignore pred_results in this step.



After the training data is already saved the next thing we can do is load the test data. You can download the test data from [here](). The file name is query.csv, its contents are the same as the training data but the difference is its scope only for 2017.

Okay, once you have downloaded the file you now can load the test and training data with the Jupyter Notebook.

```
In [18]: # Load the test data file into a dataframe
         df_test = spark.read.csv(r"C:\Users\Intel X Nvidia\Downloads\query.csv", header=True)
         # Preview df_test
         df_test.take(1)

Out[18]: [Row(time='2017-01-02T00:13:06.300Z', latitude='-36.0365', longitude='51.9288', depth='10', mag='5.7', magType='mwb', nst=None,
         gap='26', dmin='14.685', rms='1.37', net='us', id='us10007p5d', updated='2017-03-27T23:53:17.040Z', place='Southwest Indian Rid
         ge', type='earthquake', horizontalError='10.3', depthError='1.7', magError='0.068', magNst='21', status='reviewed', locationSou
         rce='us', magSource='us')]
```

```
In [19]: # Load the training data from mongo into a dataframe
         df_train = spark.read.format('mongo')\
             .option('spark.mongodb.input.uri', 'mongodb://127.0.0.1:27017/Quake.quakes').load()

         # Preview df_train
         df_train.show(5)

+----------+-----+------------+--------+---------+---------+--------------+----------+----+--------------------+
|      Date|Depth|          ID|Latitude|Longitude|Magnitude|Magnitude Type|      Type|Year|                 _id|
+----------+-----+------------+--------+---------+---------+--------------+----------+----+--------------------+
|01/02/1965|131.6|ISCGEM860706|  19.246|  145.616|      6.0|            MW|Earthquake|1965|[6139c28b50a0ee1a...|
|01/04/1965| 80.0|ISCGEM860737|   1.863|  127.352|      5.8|            MW|Earthquake|1965|[6139c28b50a0ee1a...|
|01/05/1965| 20.0|ISCGEM860762| -20.579| -173.972|      6.2|            MW|Earthquake|1965|[6139c28b50a0ee1a...|
|01/08/1965| 15.0|ISCGEM860856| -59.076|  -23.557|      5.8|            MW|Earthquake|1965|[6139c28b50a0ee1a...|
|01/09/1965| 15.0|ISCGEM860890|  11.938|  126.427|      5.8|            MW|Earthquake|1965|[6139c28b50a0ee1a...|
+----------+-----+------------+--------+---------+---------+--------------+----------+----+--------------------+
only showing top 5 rows
```

select the columns we wanted and rename After test and training data are loaded, the next thing we can do is them.

In [20]:
```
# Select fields we will use and discard fields we don't need
df_test_clean = df_test['time', 'latitude', 'longitude', 'mag', 'depth']
# Preview df_test_clean
df_test_clean.show(5)
```

```
+--------------------+--------+---------+---+------+
|                time|latitude|longitude|mag| depth|
+--------------------+--------+---------+---+------+
|2017-01-02T00:13:...|-36.0365|  51.9288|5.7|    10|
|2017-01-02T13:13:...|  -4.895| -76.3675|5.9|   106|
|2017-01-02T13:14:...|-23.2513| 179.2383|6.3|551.62|
|2017-01-03T09:09:...| 24.0151|  92.0177|5.7|    32|
|2017-01-03T21:19:...|-43.3527| -74.5017|5.5| 10.26|
+--------------------+--------+---------+---+------+
only showing top 5 rows
```

In [21]:
```
# Rename fields
df_test_clean = df_test_clean.withColumnRenamed('time', 'Date')\
    .withColumnRenamed('latitude', 'Latitude')\
    .withColumnRenamed('longitude', 'Longitude')\
    .withColumnRenamed('mag', 'Magnitude')\
    .withColumnRenamed('depth', 'Depth')

# Preview df_test_clean
df_test_clean.show(5)
```

```
+--------------------+--------+---------+---------+------+
|                Date|Latitude|Longitude|Magnitude| Depth|
+--------------------+--------+---------+---------+------+
|2017-01-02T00:13:...|-36.0365|  51.9288|      5.7|    10|
|2017-01-02T13:13:...|  -4.895| -76.3675|      5.9|   106|
|2017-01-02T13:14:...|-23.2513| 179.2383|      6.3|551.62|
|2017-01-03T09:09:...| 24.0151|  92.0177|      5.7|    32|
|2017-01-03T21:19:...|-43.3527| -74.5017|      5.5| 10.26|
+--------------------+--------+---------+---------+------+
only showing top 5 rows
```

As you can see how we are doing the same set of processes as we did previously to the training data, so now we check and convert the type of fields in test data from the string into numeric.

```
In [22]: # Preview Schema
         df_test_clean.printSchema()

         root
          |-- Date: string (nullable = true)
          |-- Latitude: string (nullable = true)
          |-- Longitude: string (nullable = true)
          |-- Magnitude: string (nullable = true)
          |-- Depth: string (nullable = true)

In [23]: # Cast some string fields into numeric fields
         df_test_clean = df_test_clean.withColumn('Latitude', df_test_clean['Latitude'].cast(DoubleType()))\
             .withColumn('Longitude', df_test_clean['Longitude'].cast(DoubleType()))\
             .withColumn('Depth', df_test_clean['Depth'].cast(DoubleType()))\
             .withColumn('Magnitude', df_test_clean['Magnitude'].cast(DoubleType()))

In [24]: df_test_clean.printSchema()

         root
          |-- Date: string (nullable = true)
          |-- Latitude: double (nullable = true)
          |-- Longitude: double (nullable = true)
          |-- Magnitude: double (nullable = true)
          |-- Depth: double (nullable = true)
```

After all the columns we need are converted to numeric, now we can create a training and testing dataframe, and remove all the missing values within using dropna().

```
In [25]: # Create training and testing dataframes
         df_testing = df_test_clean['Latitude', 'Longitude', 'Magnitude', 'Depth']
         df_training = df_train['Latitude', 'Longitude', 'Magnitude', 'Depth']
```

```
In [26]: # Preview df_training
         df_training.show(5)
```

```
+--------+---------+---------+-----+
|Latitude|Longitude|Magnitude|Depth|
+--------+---------+---------+-----+
|  19.246|  145.616|      6.0|131.6|
|   1.863|  127.352|      5.8| 80.0|
| -20.579| -173.972|      6.2| 20.0|
| -59.076|  -23.557|      5.8| 15.0|
|  11.938|  126.427|      5.8| 15.0|
+--------+---------+---------+-----+
only showing top 5 rows
```

```
In [27]: # Preview df_testing
         df_testing.show(5)
```

```
+--------+---------+---------+------+
|Latitude|Longitude|Magnitude| Depth|
+--------+---------+---------+------+
|-36.0365|  51.9288|      5.7|  10.0|
|  -4.895| -76.3675|      5.9| 106.0|
|-23.2513| 179.2383|      6.3|551.62|
| 24.0151|  92.0177|      5.7|  32.0|
|-43.3527| -74.5017|      5.5| 10.26|
+--------+---------+---------+------+
only showing top 5 rows
```

```
In [28]: # Drop record with null values from out dataframes
         df_testing = df_testing.dropna()
         df_training = df_training.dropna()
```

Once we have removed all the nulls and the dataframe is tidy now we can move to the machine learning session.

MACHINE LEARNING :

Now we move to the machine learning session, in this process, we will import some necessary libraries to create the model.

```
In [29]: from pyspark.ml import Pipeline
         from pyspark.ml.regression import RandomForestRegressor
         from pyspark.ml.feature import VectorAssembler
         from pyspark.ml.evaluation import RegressionEvaluator
```

After we imported the libraries we needed we can create the model.

```
In [30]: # Select feature to parse into our model and then create the feature vector
         assembler = VectorAssembler(inputCols=['Latitude', 'Longitude', 'Depth'], outputCol='features')

         # Create the model
         model_reg = RandomForestRegressor(featuresCol='features', labelCol='Magnitude')

         # Chain the assembler with the model in a pipeline
         pipeline = Pipeline(stages=[assembler, model_reg])

         # Train the Model
         model = pipeline.fit(df_training)

         # Make the prediction
         pred_results = model.transform(df_testing)

In [31]: # Preview pred_results dataframe
         pred_results.show(5)

         +--------+---------+---------+------+--------------------+-----------------+
         |Latitude|Longitude|Magnitude| Depth|            features|       prediction|
         +--------+---------+---------+------+--------------------+-----------------+
         |-36.0365|  51.9288|      5.7|  10.0|[-36.0365,51.9288...|5.845803808668242|
         |  -4.895| -76.3675|      5.9| 106.0|[-4.895,-76.3675,...|5.882302317310106|
         |-23.2513| 179.2383|      6.3|551.62|[-23.2513,179.238...|5.905875451821726|
         | 24.0151|  92.0177|      5.7|  32.0|[24.0151,92.0177,...|5.881770835768791|
         |-43.3527| -74.5017|      5.5| 10.26|[-43.3527,-74.501...|5.954785795157244|
         +--------+---------+---------+------+--------------------+-----------------+
         only showing top 5 rows
```

As we can see from the syntax above to make a prediction we need to aggregate latitude, longitude, and depth data into one vector and stored it into a new column called features. After that, the results of the prediction are stored automatically in the prediction column. We can compare the magnitude prediction with the magnitude from the test data, the difference is tolerable. To verify this model is reliable we need to test the accuracy using RMSE. If the RMSE is below 0.5 it means the model is a good fit and we can use it to predict.

In [32]:
```
# Evaluate the model
# RMSE should be less than 0.5 for the model to be useful
evaluator = RegressionEvaluator(labelCol='Magnitude', predictionCol='prediction', metricName='rmse')
rmse = evaluator.evaluate(pred_results)
print('Root Mean Squared Error (RMSE) on test data = %g' % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 0.402274
```

After we calculate the RMSE the result is 0.402274 which means the model is a good fit and reliable.

Now the next thing we can do is creating a dataset for prediction, drop the column we don't need, and rename some columns.

In [34]:
```
# Create the prediction dataset
df_pred_results = pred_results['Latitude', 'Longitude', 'prediction']

# Rename the prediction field
df_pred_results = pred_results.withColumnRenamed('prediction', 'Pred_Magnitude')

# Add more columns to our prediction dataset
df_pred_results = df_pred_results.withColumn('Year', lit(2017))\
    .withColumn('RMSE', lit(rmse))

# Discard column that we don't need ('features' is vector so it will cause error when load into mongodb)
columns_to_drop = ['Magnitude', 'Depth', 'features']
df_pred_results = df_pred_results.drop(*columns_to_drop)

# Preview df_pred_results
df_pred_results.show(5)
```

```
+--------+---------+-----------------+----+-------------------+
|Latitude|Longitude|   Pred_Magnitude|Year|               RMSE|
+--------+---------+-----------------+----+-------------------+
|-36.0365|  51.9288|5.845803808668242|2017|0.40227436189606913|
|  -4.895| -76.3675|5.882302317310106|2017|0.40227436189606913|
|-23.2513| 179.2383|5.905875451821726|2017|0.40227436189606913|
| 24.0151|  92.0177|5.881770835768791|2017|0.40227436189606913|
|-43.3527| -74.5017|5.954785795157244|2017|0.40227436189606913|
+--------+---------+-----------------+----+-------------------+
only showing top 5 rows
```

DATA VISUALIZATION :

Now is the interesting part because we can see our model through plots. Before we start to create the plot we need to import some libraries. One of the libraries is Bokeh which is an important part to visualize the model.

```
In [38]: import pandas as pd
         from bokeh.io import output_notebook, output_file
         from bokeh.plotting import figure, show, ColumnDataSource
         from bokeh.models.tools import HoverTool
         import math
         from math import pi
         from bokeh.palettes import Category20c
         from bokeh.transform import cumsum
         from bokeh.tile_providers import CARTODBPOSITRON, get_provider, Vendors
         from bokeh.themes import built_in_themes
         from bokeh.io import curdoc
         from pymongo import MongoClient
         import warnings
         warnings.filterwarnings('ignore')
         from pyspark.sql.functions import desc
```

Import libraries to visualize data

After the libraries are imported, the next thing we can do is create a custom read function. This part is important to read data from MongoDB.

```
In [39]: # Create a custom read function to read data from mongodb into a dataframe
def read_mongo(host='127.0.0.1', port=27017, username=None, password=None, db='Quake', collection='pred_results'):

    mongo_uri = 'mongodb://{}:{}/{}.{}'.format(host, port, db, collection)

    # Connect to mongodb
    conn = MongoClient(mongo_uri)
    db = conn[db]

    # Select all records from the collection
    cursor = db[collection].find()

    # Create the dataframe
    df = pd.DataFrame(list(cursor))

    # Delete the _id field
    del df['_id']

    return df
```

```
In [40]: # Load the datasets from mongodb
df_quakes = read_mongo(collection='quakes')
df_quake_freq = read_mongo(collection='quake.freq')
df_quake_pred = read_mongo(collection='pred_results')
```

Custom read function to read data from MongoDB

Then, we extract the data from 2016.

```
In [41]: df_quakes_2016 = df_quakes[df_quakes['Year'] == 2016]
# Preview df_quakes_2016
df_quakes_2016.head()
```

Out[41]:

	Date	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	ID	Year
22943	01/01/2016	-50.5575	139.4489	Earthquake	10.00	6.3	MWW	US10004ANT	2016.0
22944	01/01/2016	-28.6278	-177.2810	Earthquake	34.00	5.8	MWW	US10004AQY	2016.0
22945	01/02/2016	44.8069	129.9406	Earthquake	585.47	5.8	MWW	US10004ATB	2016.0
22946	01/03/2016	24.8036	93.6505	Earthquake	55.00	6.7	MWW	US10004B2N	2016.0
22947	01/05/2016	30.6132	132.7337	Earthquake	4.71	5.8	MWW	US10004BEN	2016.0

Data from 2016

Then type output_notebook to verify the BokehJS is loaded in Jupyter Notebook.

Okay, after we create the dataset from 2016 the next thing we can do now is creating a function to style our plot. To style the plots you can simply do this:

```
In [42]: # Show plots embedded in Jupyter Notebook
         output_notebook()
```

BokehJS 2.3.3 successfully loaded.

```
In [43]: # Create custom style function to style our plots
         def style(p):
             # Title
             p.title.align = 'center'
             p.title.text_font_size = '20pt'
             p.title.text_font = 'serif'

             # Axis titles
             p.xaxis.axis_label_text_font_size = '14pt'
             p.xaxis.axis_label_text_font_style = 'bold'
             p.yaxis.axis_label_text_font_size = '14pt'
             p.yaxis.axis_label_text_font_style = 'bold'

             # Tick labels
             p.xaxis.major_label_text_font_size = '12pt'
             p.yaxis.major_label_text_font_size = '12pt'

             # Plot the legend in the top left corner
             p.legend.location = 'top_left'

             return p
```

Styling plots

After we created a custom style function, now we can create the Geo Map plot. Using Geo Map we can see our model applied on the earth map. The syntax is quite long so I will quote the code below:

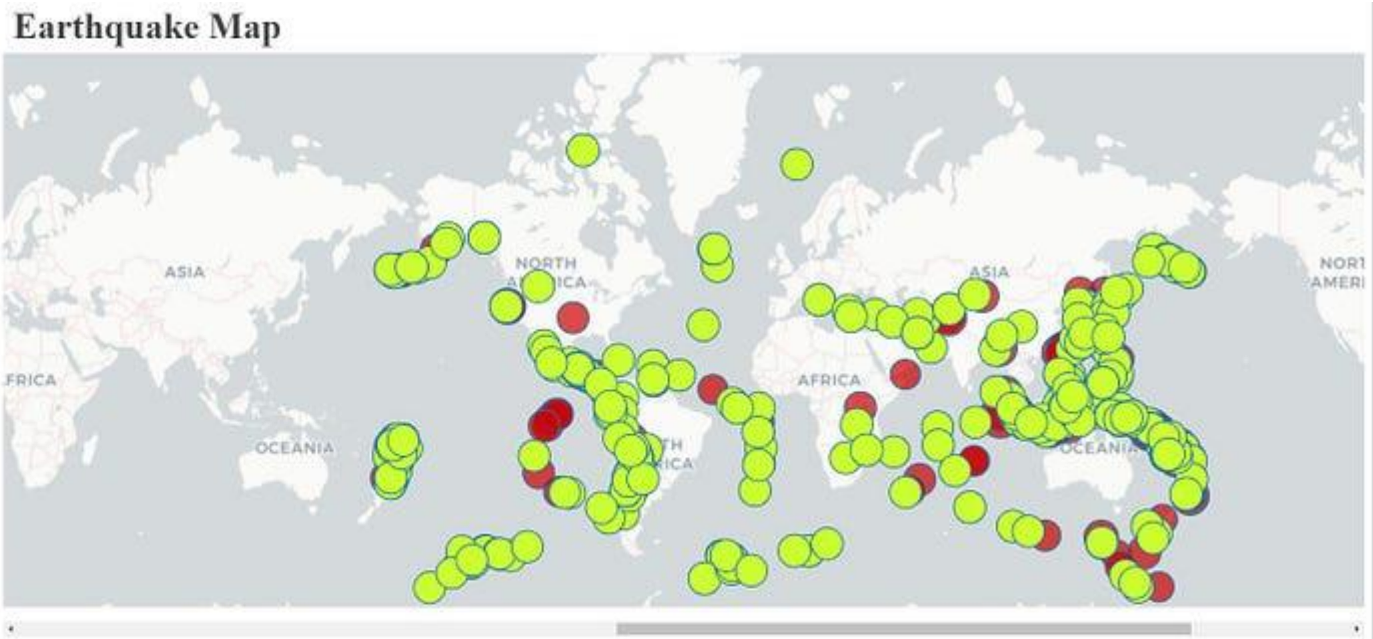
```
# Create the Geo Map plot
def plotMap():
 lat = df_quakes_2016['Latitude'].values.tolist()
 lon = df_quakes_2016['Longitude'].values.tolist()

 pred_lat = df_quake_pred['Latitude'].values.tolist()
```

```
pred_lon = df_quake_pred['Longitude'].values.tolist()

lst_lat = []
lst_lon = []
lst_pred_lat = []
lst_pred_lon = []

i=0
j=0

# Convert lat and lon values into merc_projection format
for i in range (len(lon)):
r_major = 6378137.000
x = r_major * math.radians(lon[i])
scale = x/lon[i]
y = 180.0/math.pi * math.log(math.tan(math.pi/4.0 +
lat[i] * (math.pi/180.0)/2.0)) * scale

lst_lon.append(x)
lst_lat.append(y)
i += 1

# Convert predicted lat and long values into merc_projection
format
for j in range (len(pred_lon)):
r_major = 6378137.000
x = r_major * math.radians(pred_lon[j])
scale = x/pred_lon[j]
y = 180.0/math.pi * math.log(math.tan(math.pi/4.0 +
pred_lat[j] * (math.pi/180.0)/2.0)) * scale

lst_pred_lon.append(x)
lst_pred_lat.append(y)
j += 1

df_quakes_2016['coords_x'] = lst_lat
df_quakes_2016['coords_y'] = lst_lon
df_quake_pred['coords_x'] = lst_pred_lat
df_quake_pred['coords_y'] = lst_pred_lon

# Scale the circles
df_quakes_2016['Mag_Size'] = df_quakes_2016['Magnitude'] * 4
df_quake_pred['Mag_Size'] = df_quake_pred['Pred_Magnitude'] * 4

# Create datasources for our ColumnDataSource object
lats = df_quakes_2016['coords_x'].tolist()
longs = df_quakes_2016['coords_y'].tolist()
mags = df_quakes_2016['Magnitude'].tolist()
years = df_quakes_2016['Year'].tolist()
mag_size = df_quakes_2016['Mag_Size'].tolist()
```

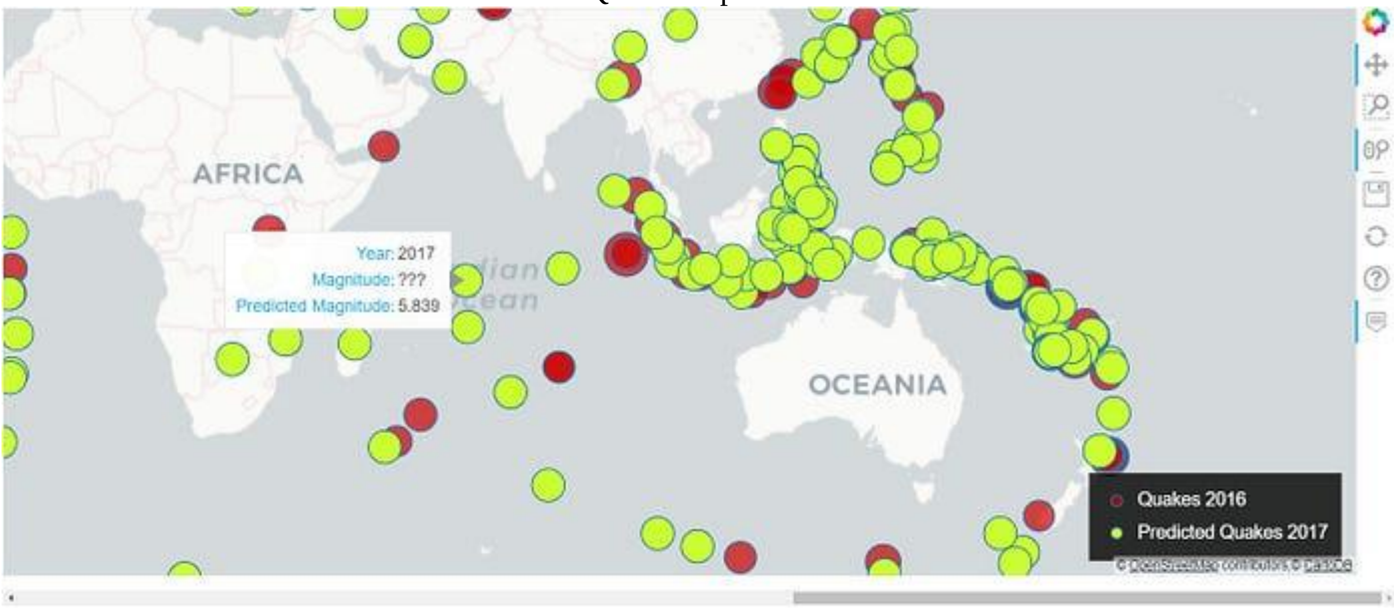
```
pred_lats = df_quake_pred['coords_x'].tolist()
pred_longs = df_quake_pred['coords_y'].tolist()
pred_mags = df_quake_pred['Pred_Magnitude'].tolist()
pred_year = df_quake_pred['Year'].tolist()
pred_mag_size = df_quake_pred['Mag_Size'].tolist()

# Create column datasource
cds = ColumnDataSource(
data=dict(
lat=lats,
lon=longs,
mag=mags,
year=years,
mag_s=mag_size
)
)

pred_cds = ColumnDataSource(
data=dict(
pred_lat=pred_lats,
pred_long=pred_longs,
pred_mag=pred_mags,
year=pred_year,
pred_mag_s=pred_mag_size
)
)

# Tooltips
TOOLTIPS = [
("Year", " @year"),
("Magnitude", " @mag"),
("Predicted Magnitude", " @pred_mag")
]

# Create figure
p = figure(title = 'Earthquake Map',
plot_width=2300, plot_height=450,
x_range=(-2000000, 6000000),
y_range=(-1000000, 7000000),
tooltips=TOOLTIPS)

p.circle(x='lon', y='lat', size='mag_s', fill_color='#cc0000',
fill_alpha=0.7,
source=cds, legend='Quakes 2016')

# Add circles for our predicted earthquakes
p.circle(x='pred_long', y='pred_lat', size='pred_mag_s',
fill_color='#ccff33', fill_alpha=7.0,
source=pred_cds, legend='Predicted Quakes 2017')

tile_provider = get_provider(Vendors.CARTODBPOSITRON)
```

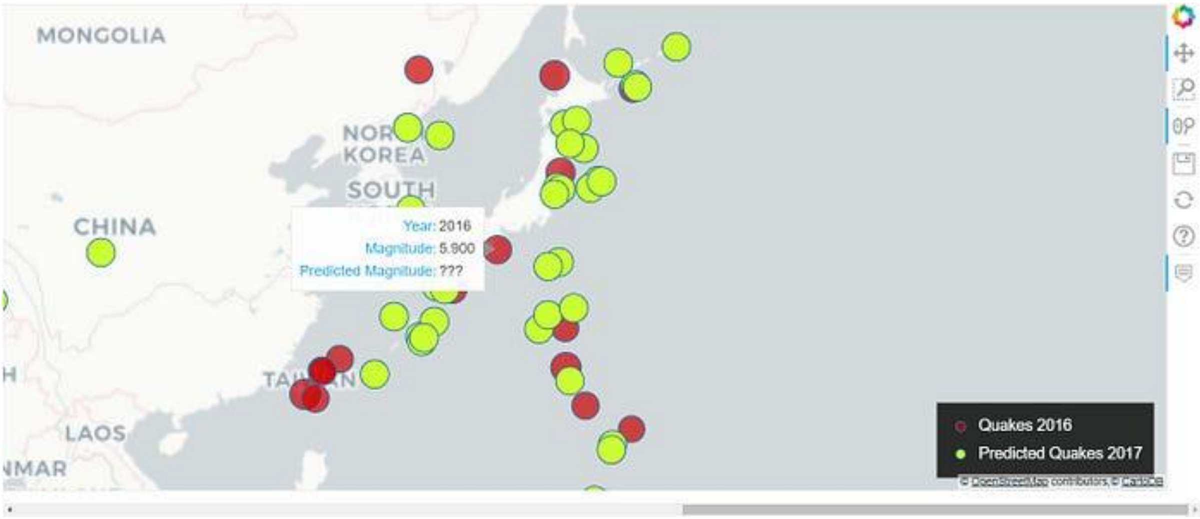
```
 p.add_tile(tile_provider)

 # Style the map plot
 # Title
 p.title.align='center'
 p.title.text_font_size='20pt'
 p.title.text_font='serif'

 # Legend
 p.legend.location='bottom_right'
 p.legend.background_fill_color='black'
 p.legend.background_fill_alpha=0.8
 p.legend.click_policy='hide'
 p.legend.label_text_color='white'
 p.xaxis.visible=False
 p.yaxis.visible=False
 p.axis.axis_label=None
 p.axis.visible=False
 p.grid.grid_line_color=None


 show(p)

plotMap()
```

After you running the syntax now you see the output of the map with the dots. The green dots represent the predicted quakes of 2017 and the red dots for quakes of the year 2016.


Earthquake Map
ASIA
OCEANIA
AFRICA
ASIA
OCEANIA

Quake Map

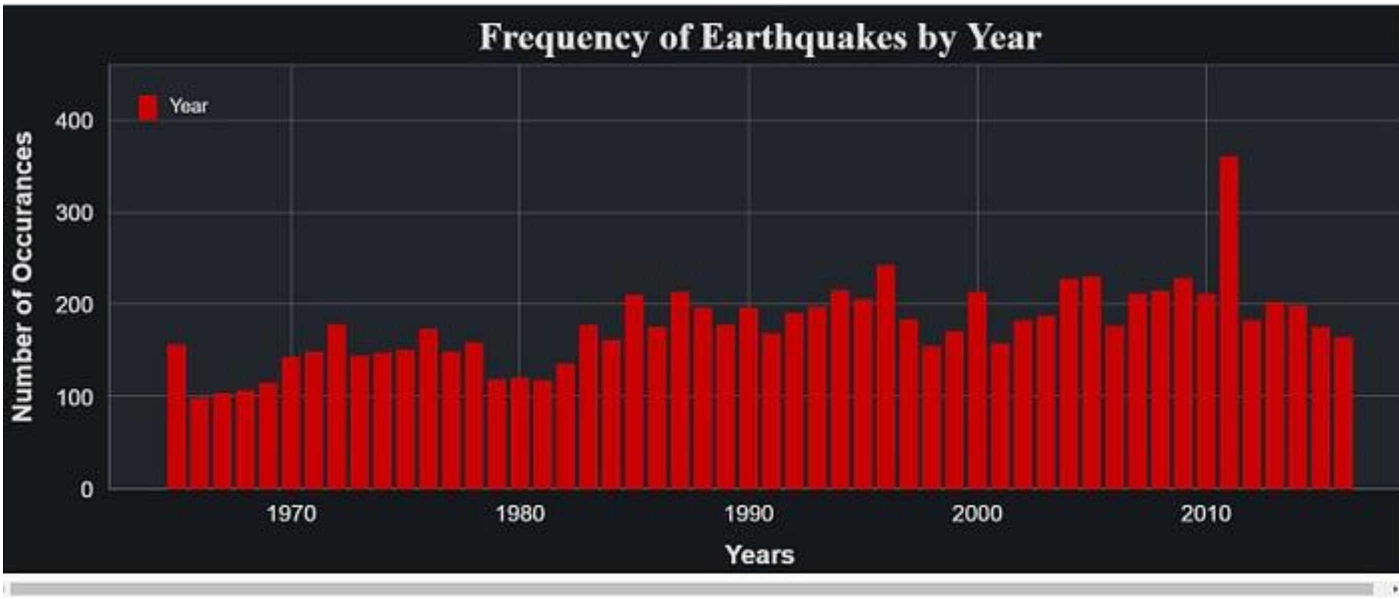

AFRICA
OCEANIA
Year: 2017
Magnitude: ???
Predicted Magnitude: 5.839
Quakes 2016
Predicted Quakes 2017



Okay after we creating the geo map, the next thing we can do is creating a bar chart. In this bar chart, we will visualize how the frequency of quakes every year.

```python
# Create the Bar Chart
def plotBar():
    # Load the datasource
    cds = ColumnDataSource(data=dict(
        yrs = df_quake_freq['Year'].values.tolist(),
        numQuakes = df_quake_freq['Counts'].values.tolist()
    ))

    # Tooltip
    TOOLTIPS =[
        ('Year', ' @yrs'),
        ('Number of earthquakes', ' @numQuakes')
    ]

    # Create a figure
    barChart = figure(title='Frequency of Earthquakes by Year',
                      plot_height=400,
                      plot_width=1150,
                      x_axis_label='Years',
                      y_axis_label='Number of Occurances',
                      x_minor_ticks=2,
                      y_range=(0, df_quake_freq['Counts'].max() +100),
                      toolbar_location=None,
                      tooltips=TOOLTIPS)

    # Create a vertical bar
    barChart.vbar(x='yrs', bottom=0, top='numQuakes',
                  color='#cc0000', width=0.75,
                  legend='Year', source=cds)

    # Style the bar chart
    barChart = style(barChart)

    show(barChart)

    return barChart

plotBar()
```

Creating a bar chart



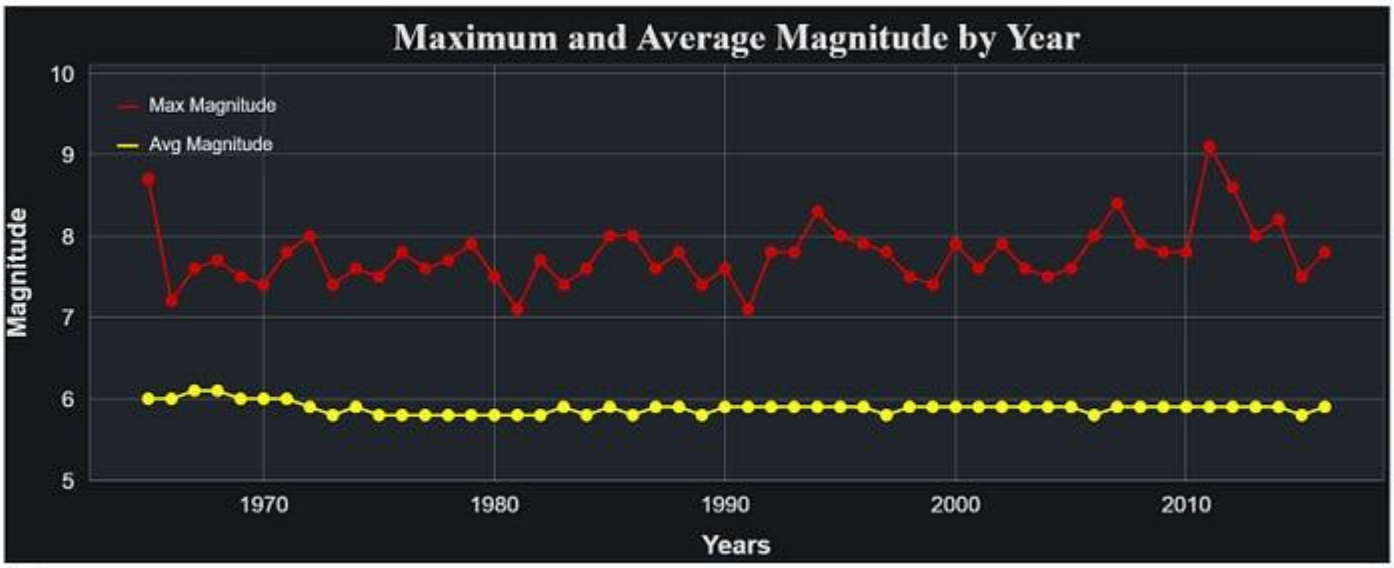
Frequency of Earthquakes by Year

The bar chart tells us how is the trend of earthquakes every year, based on the graphic we can see 2010 is the year of the highest number of earthquakes occurrences.

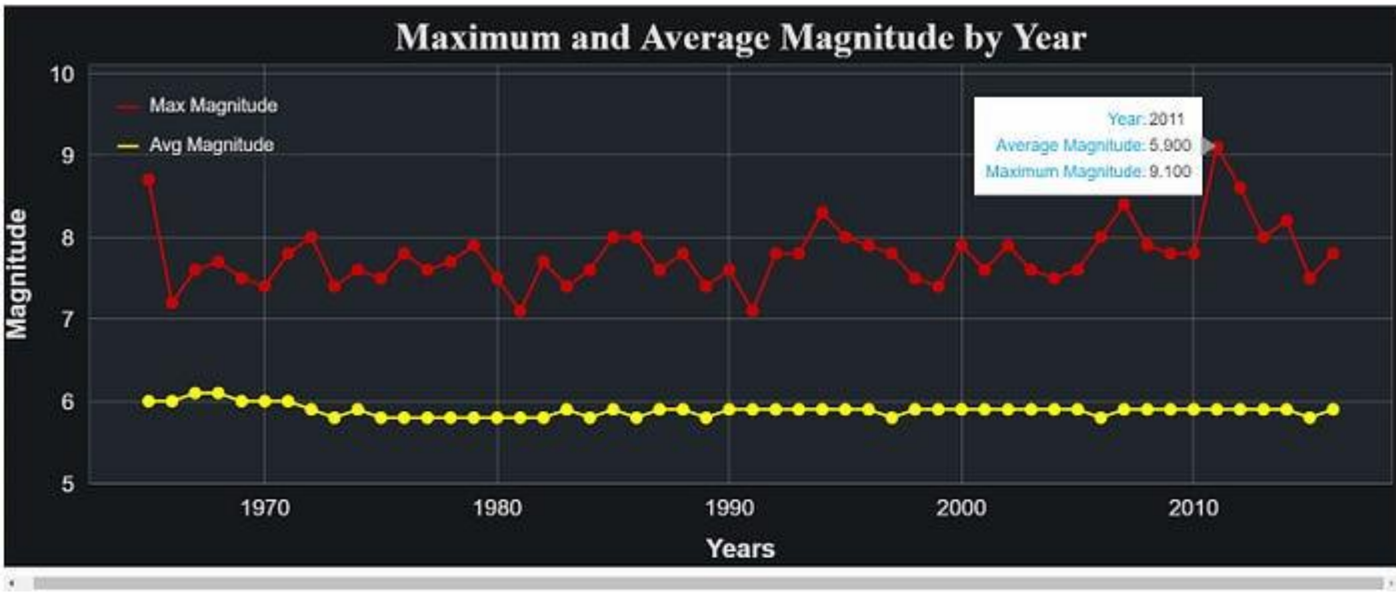
After we create the bar chart for quakes frequency, now we will create the line chart to know the trend of magnitude by year.

```python
In [46]: # Create a magnitude plot
def plotMagnitude():
    # Load the datasource
    cds = ColumnDataSource(data=dict(
        yrs = df_quake_freq['Year'].sort_values().values.tolist(),
        avg_mag = df_quake_freq['Avg_Magnitude'].round(1).values.tolist(),
        max_mag = df_quake_freq['Max_Magnitude'].values.tolist()
    ))

    # Tooltip
    TOOLTIPS = [
        ('Year', ' @yrs'),
        ('Average Magnitude', ' @avg_mag'),
        ('Maximum Magnitude', ' @max_mag')
    ]

    # Create the figure
    mp = figure(title='Maximum and Average Magnitude by Year',
                plot_width=1150, plot_height=400,
                x_axis_label='Years',
                y_axis_label='Magnitude',
                x_minor_ticks=2,
                y_range=(5, df_quake_freq['Max_Magnitude'].max() + 1),
                toolbar_location=None,
                tooltips=TOOLTIPS)

    # Max Magnitude
    mp.line(x='yrs', y='max_mag', color='#cc0000', line_width=2, legend='Max Magnitude', source=cds)
    mp.circle(x='yrs', y='max_mag', color='#cc0000', size=8, fill_color='#cc0000', source=cds)

    # Average Magnitude
    mp.line(x='yrs', y='avg_mag', color='yellow', line_width=2, legend='Avg Magnitude', source=cds)
    mp.circle(x='yrs', y='avg_mag', color='yellow', size=8, fill_color='yellow', source=cds)

    mp = style(mp)

    show(mp)

    return mp

plotMagnitude()
```

Creating a magnitude plot



Maximum and Average Magnitude Trend by Year



Highest Maximum Magnitude by Year

As we can see from the line chart we can know that in 2011 there was an earthquake with a magnitude 9.1 scale Richter if we investigated that magnitude is related to the 2011 Tōhoku earthquake and tsunami which is the highest magnitude between 1965 to 2016.