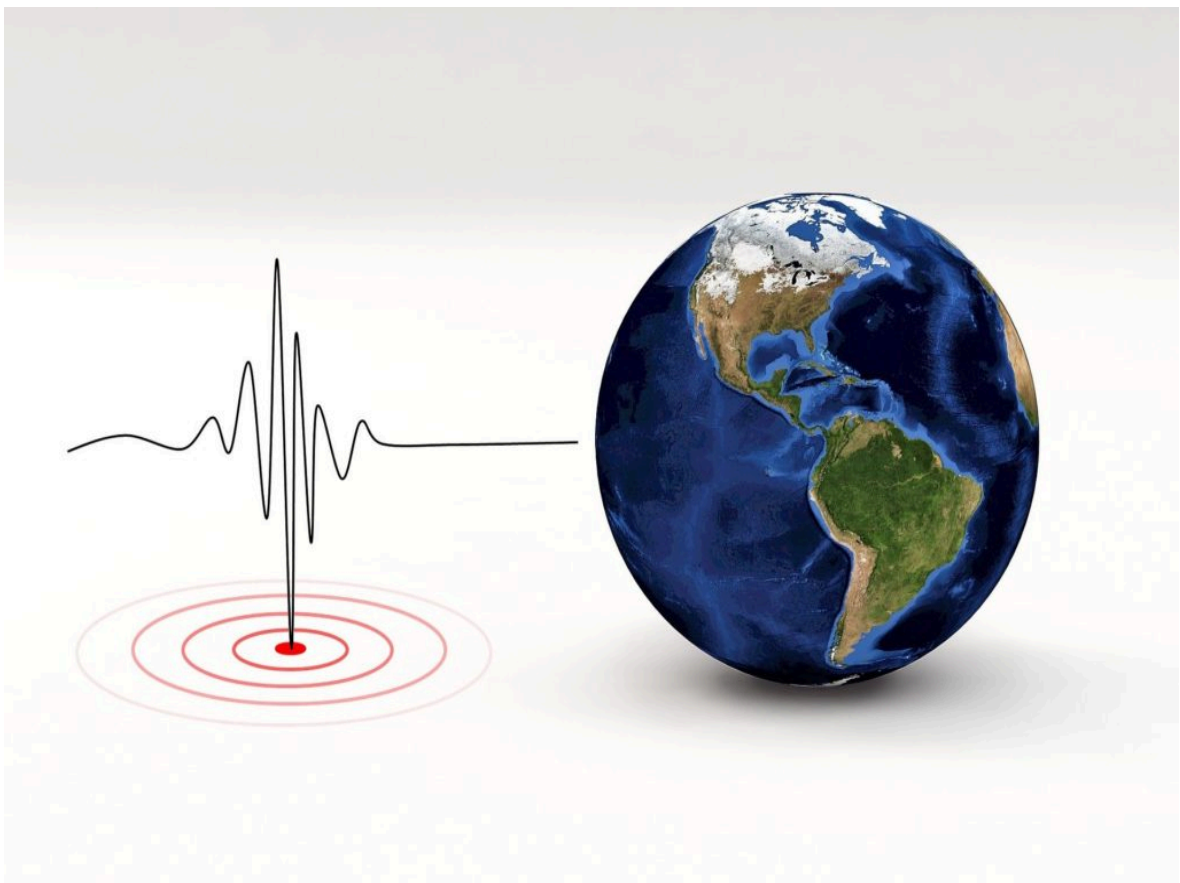


EARTHQUAKE PREDICTION MODEL USING PYTHON

Project Title : Earthquake Prediction

Phase 3 : Development Part 1

Topic : Start building the earthquake prediction model by Loading and pre-processing the dataset.



Earthquake Prediction

INTRODUCTION :

- Earthquake prediction remains a highly challenging and complex field of study. As of my last knowledge update in September 2021, there is no reliable method for accurately predicting the exact time, location, and magnitude of an earthquake with a high degree of precision.
- Earthquakes are the result of complex interactions between tectonic plates beneath the Earth's surface, and while scientists have made significant progress in understanding the Earth's geology and seismology, there are still many uncertainties.
- Earthquakes are inherently difficult to predict because they result from the buildup and sudden release of stress along geological fault lines. Scientific research continues in this field, and new technologies and methods are continually being developed to improve our understanding of earthquakes and enhance early warning systems.
- For the most up-to-date information on earthquake prediction, it's advisable to consult seismological organisations and research institutions, as developments in this field may have occurred after my last knowledge update in September 2021.
- This prediction can be classified into short-term, intermediate and long-term prediction based on the duration of the time scale. This problem is nigh impossible to solve and as a result in 1997, Geller et al. [5] concluded that predicting earthquakes is impossible . Over time research has shown that it is not completely impossible to predict earthquakes.

- In the last two decades, countless studies have been conducted on the topic of earthquake prediction. Predicting an earthquake involves stating the exact time, magnitude and location of an upcoming earthquake.

DATA LINK : <https://www.kaggle.com/datasets/usgs/earthquake-database>

GIVEN DATASET :

date	depth	mag	place	latitude	longitude	depth_avg_22	depth_avg_15	depth_avg_7	mag_avg_22	mag_avg_15	mag_avg_7	mag_outcome
2020-07-14	6.70	1.58	Oklahoma	36.171483	-97.718347	6.717727	6.560000	7.100000	1.352273	1.271333	1.357143	1.338571
2020-07-14	7.55	2.07	Oklahoma	36.171483	-97.718347	6.730000	6.682667	7.132857	1.372727	1.334667	1.527143	1.535714
2020-07-14	7.39	1.89	Oklahoma	36.171483	-97.718347	6.747727	6.708667	6.940000	1.396818	1.377333	1.570000	1.335714
2020-07-15	7.75	1.48	Oklahoma	36.171483	-97.718347	6.834545	6.764000	6.848571	1.383182	1.388667	1.581429	1.251429
2020-07-15	7.81	1.50	Oklahoma	36.171483	-97.718347	6.841364	6.854667	6.964286	1.404545	1.385333	1.602857	1.291429

NECESSARY STEPS TO FOLLOW :

1.Import libraries :

Start by importing necessary libraries:

Program :

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

```
import os
print(os.listdir("../input"))
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

2. Load the Dataset :

Load the dataset into a Panda Dataframe. You can find earthquake prediction dataset in CSV format, now you can explore and work with those dataset.

Program :

```
data = pd.read_csv("database.csv")
pd.read()
```

3. Exploratory Data Analysis (EDA) :

Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics, and visualising it to identify patterns.

Program :

```
# Check for missing values
print(df.isnull().sum())

# Explore statistics
print(df.describe())

# Visualise the data (e.g., histograms, scatter plots,
etc.)
```

4. Feature Engineering :

Depending on your dataset, you may need to create new features or transform existing ones. This can involve one-hot encoding categorical

variables, handling date/time data, or scaling numerical features.

Program :

```
#Example :One-hot encoding for categorical variables
df =pd.get_dummies(df,columns=['Depth','Magnitude'])
```

5. Split the Data :

Split your dataset into training and testing sets. This helps you evaluate your model's performance later.

Program :

```
X =final_data[['Timestamp','Latitude','Longitude']]
y =final_data[['Magnitude','Depth']]
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,random_state=42)
print(X_train.shape,X_test.shape,y_train.shape,
X_test.shape)
```

Importance of loading and processing dataset :

- Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for earthquake prediction models, as earthquake datasets are often complex.
- By loading and preprocessing the dataset, we ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

Challenges involved in loading and preprocessing a earthquake prediction dataset :

There are a number of challenges involved in loading and preprocessing a earthquake prediction dataset, including:

- **Handling missing values :**

Earthquake prediction datasets often contain missing values, which can be due to a variety of factors, such as human error or incomplete data Collection. Common methods for handling missing values include dropping the rows with missing values, imputing the missing values with the mean or median of the feature, or using a more sophisticated method such as multiple imputation.

- **Encoding categorical variables :**

The choice of encoding method depends on the nature of your categorical variables and the specific requirements of your earthquake prediction model. You should consider the type of data, cardinality of the categories, and the potential impact on model performance when selecting an encoding method.

- **Scaling the features :**

It is often helpful to scale the features before training a machine learning model. This can help to improve the performance of the model and make it more robust to outliers. There are a variety of ways to scale the features, such as min-max scaling and standard Scaling.

- **Splitting the dataset into training and testing sets :**

Once the data has been pre-processed, we need to split the dataset into training and testing sets. The training set will be used to train the model, and the testing set will be used to evaluate the performance of the model on unseen data. It is important to split the dataset in a way that is representative of the real world distribution of the data.

How to overcome the challenges of loading and preprocessing a Earthquake prediction :

There are a number of things that can be done to overcome the challenges of loading and preprocessing a earthquake prediction dataset including:

- **Use a data preprocessing library :**

There are a number of libraries available that can help with data preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling the features.

- **Carefully consider the specific needs of your model :**

The best way to preprocess the data will depend on the specific machine learning algorithm that you are using. It is important to carefully consider the requirements of the algorithm and to preprocess the data in a way that is compatible with the algorithm.

➤ **Validate the preprocessed data :**

It is important to validate the preprocessed data to ensure that it is in a format that can be used by the machine learning algorithm and that it is of high quality. This can be done by inspecting the data visually or by using statistical methods.

1. Loading the dataset :

- Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.
- The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used. However, there are some general steps that are common to most machine learning frameworks:

a. Identify the dataset :

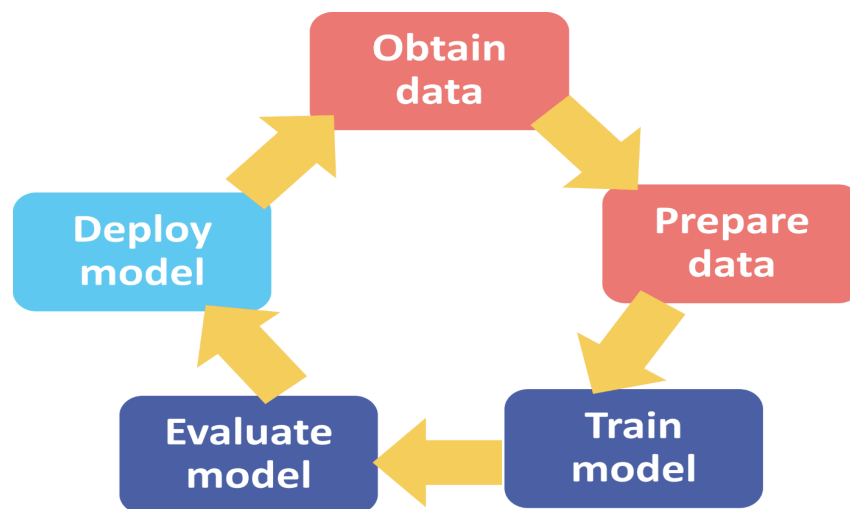
The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage Service.

b. Load the dataset :

Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c. Preprocess the dataset :

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.



Here, how to load a dataset using machine learning in Python.

[#importing the required libraries](#)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import
r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
```

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xg
%matplotlib inline import warnings
warnings.filterwarnings("ignore")

```

Reading the csv dataset

```

from google.colab import files
upload=files.upload()
eq = pd.read_csv("archive (1).zip")
eq.head()

```

The screenshot shows a Google Colab notebook with the following code and output:

```

import warnings
[1] warnings.filterwarnings("ignore")

[2] from google.colab import files
    upload=files.upload()

Choose Files archive (1).zip
• archive (1).zip(application/x-zip-compressed) - 604367 bytes, last modified: 10/23/2023 - 100% done
Saving archive (1).zip to archive (1).zip

eq = pd.read_csv("archive (1).zip")
eq.head()

```

The output displays the first 5 rows of the dataset, which contains earthquake data. The columns are: Date, Time, Latitude, Longitude, Type, Depth, Depth Error, Depth Seismic Stations, Magnitude, Magnitude Type, Magnitude Seismic Stations, Azimuthal Gap, Horizontal Distance, Horizontal Error, Root Mean Square, and ISC.

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ISC
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN	ISC
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	ISC
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN	ISC
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	ISC
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	ISC

5 rows x 21 columns

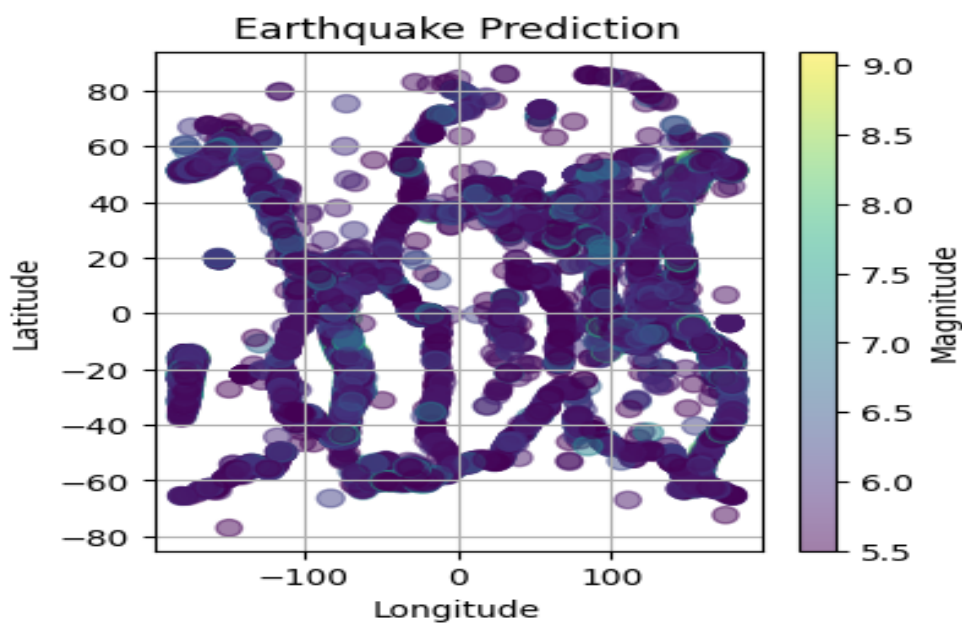
#Basic plots using Matplotlib(Visualisation)

Extract latitude, longitude, and magnitude data

```

latitude = eq['Latitude']
longitude = eq['Longitude']
magnitude = eq['Magnitude']
plt.figure(figsize=(4, 4))
plt.scatter(longitude, latitude, c=magnitude,
            cmap='viridis', s=magnitude * 10, alpha=0.5)
plt.title("Earthquake Prediction")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.colorbar(label="Magnitude")
plt.grid(True)
plt.show()

```

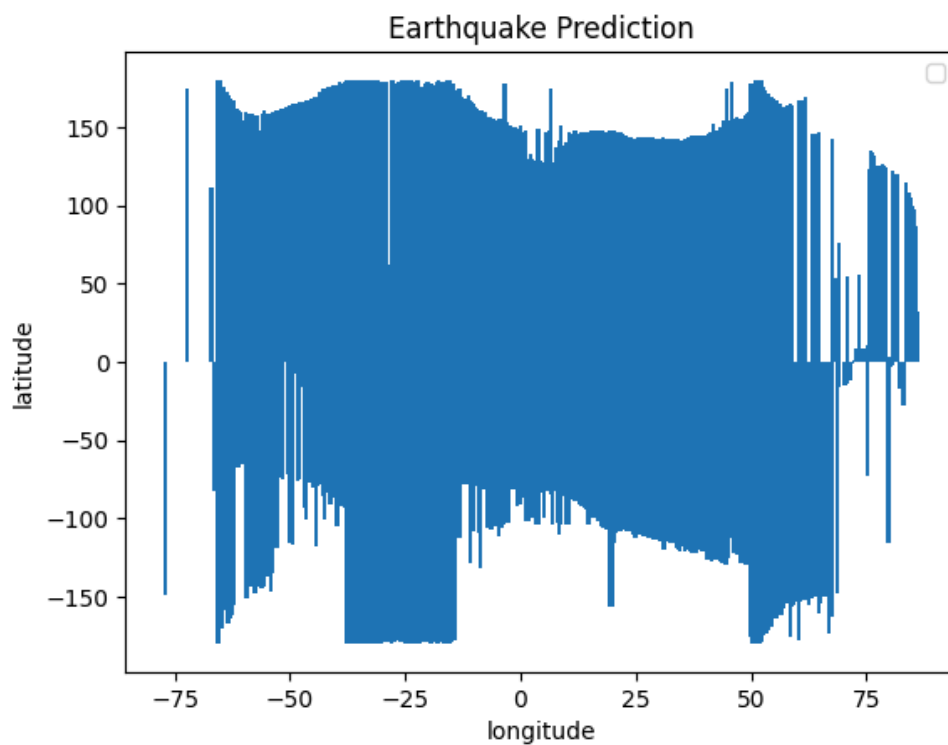


```

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
latitude = eq['Latitude']
longitude = eq['Longitude']
ax.bar(latitude, longitude)

```

```
ax.set_ylabel('latitude')
ax.set_xlabel('longitude')
ax.set_title('Earthquake Prediction')
ax.legend()
plt.show()
```

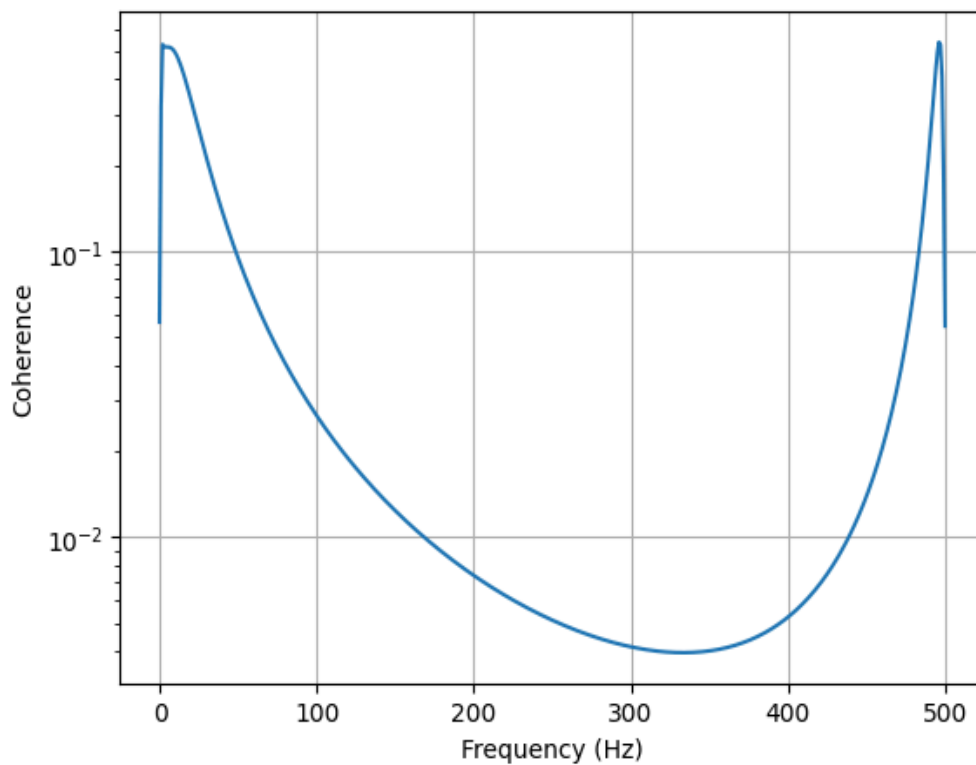


```
import numpy as np
from scipy.signal import coherence
import matplotlib.pyplot as plt
fs = 1000 # Sample rate (Hz)
t = np.arange(0, 10, 1/fs)
freq1 = 5 # Frequency of the first signal (Hz)
freq2 = 3 # Frequency of the second signal (Hz)
signal1 = np.sin(2 * np.pi * freq1 * t)
```

```

signal2 = np.sin(2 * np.pi * freq2 * t)
f, Cxy = coherence(signal1, signal2, fs=fs, nperseg=1024)
plt.figure()
plt.semilogy(f, Cxy)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Coherence')
plt.grid()
plt.show()

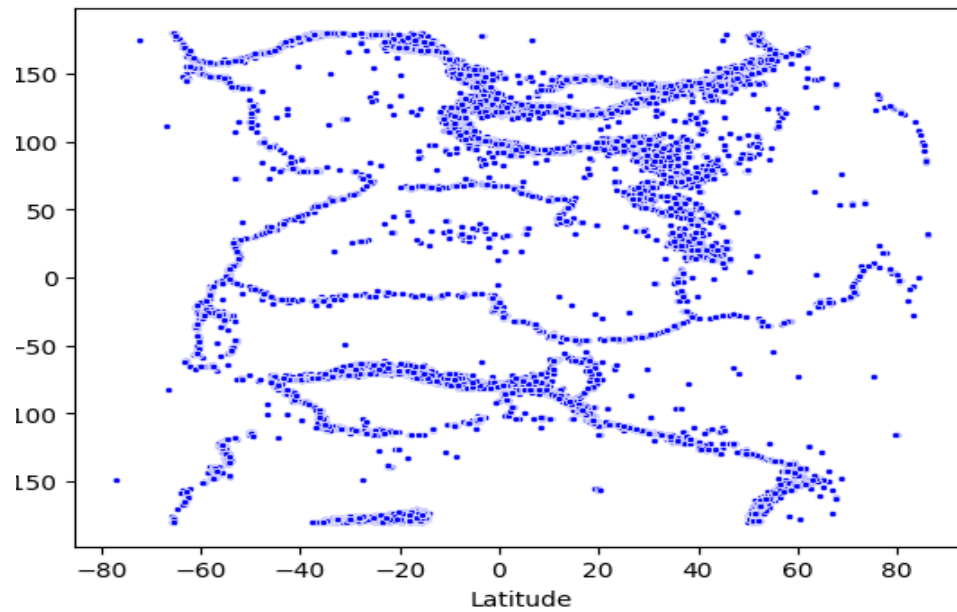
```



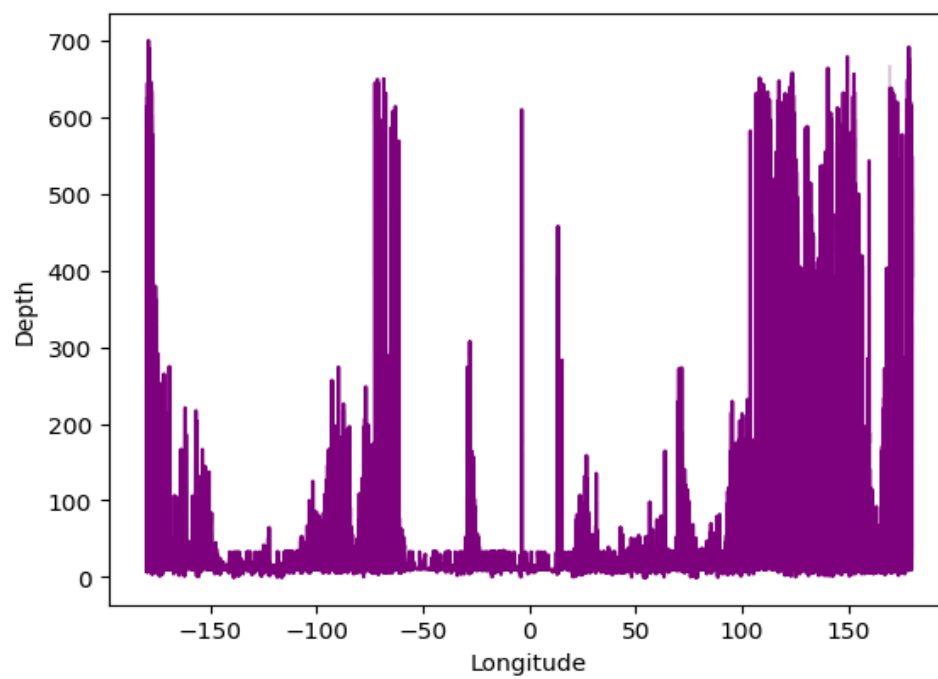
```

sns.scatterplot(data=eq, x='Latitude', y='Longitude', color='
blue', marker='.')

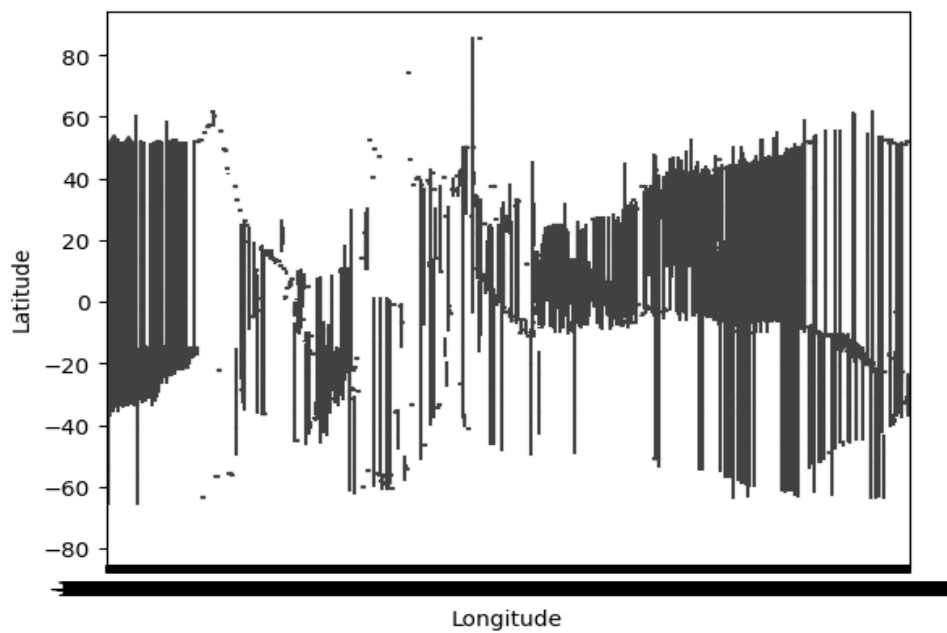
```



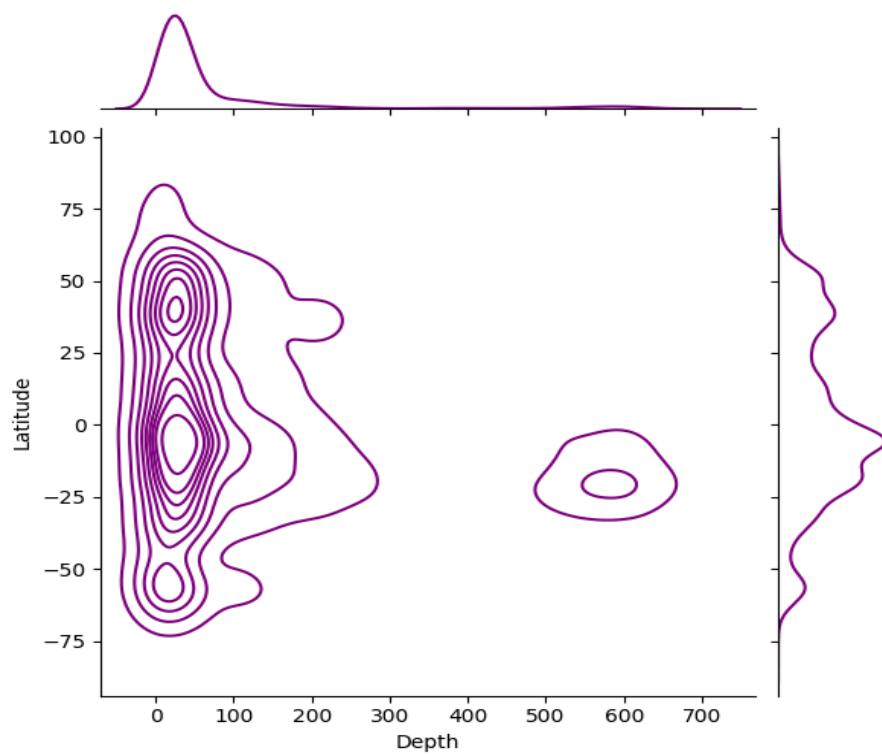
```
sns.lineplot(data=eq,x='Longitude',y='Depth',color='purple')
```



```
sns.boxplot(data=eq,x='Longitude',y='Latitude')
```



```
sns.jointplot(x="Depth", y="Latitude",  
data=eq,kind="kde",color='purple');
```



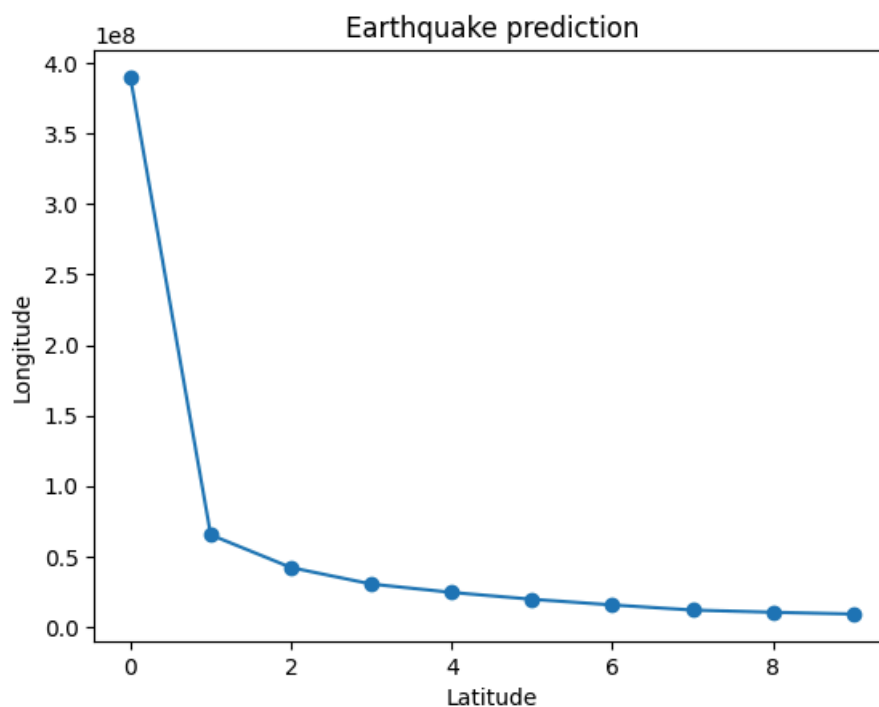
```

import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x = eq['Latitude']
y = eq['Longitude']
data = list(zip(x, y))

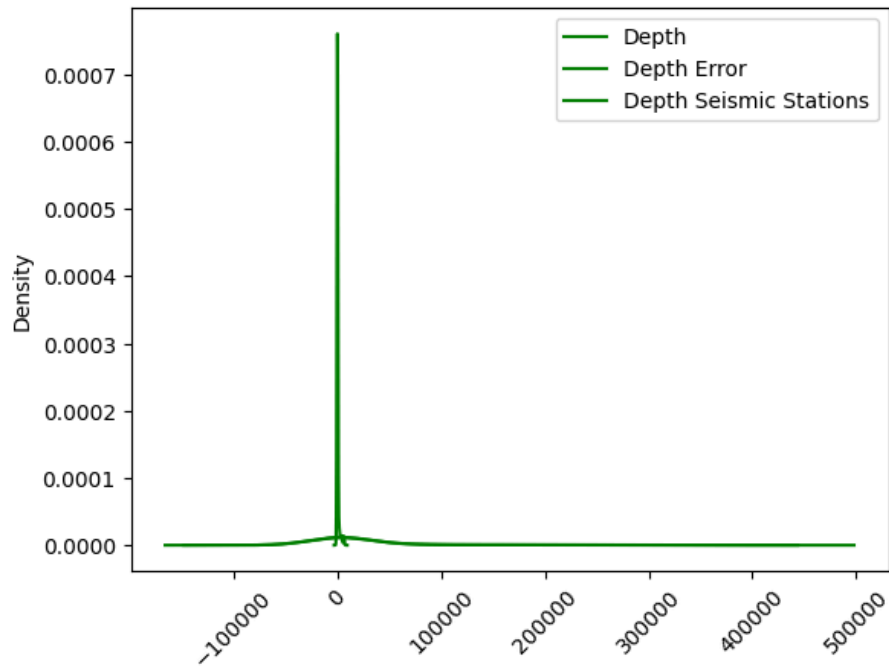
inertias = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, n_init=10)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(inertias, marker='o')
plt.title('Earthquake prediction')
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.show()

```




```
eq[[x for x in eq.columns if "Depth" in x]
+ ["Magnitude"]].groupby("Magnitude").sum().plot(kind="kde"
, rot=45,color='green');
```



```
import tensorflow as tf
from tensorflow import keras

# Load the data
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()

# Preprocess the data
x_train = x_train.reshape((x_train.shape[0], 28 *
28)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28 *
28)).astype('float32') / 255
y_train = keras.utils.to_categorical(y_train)
```

```

y_test = keras.utils.to_categorical(y_test)

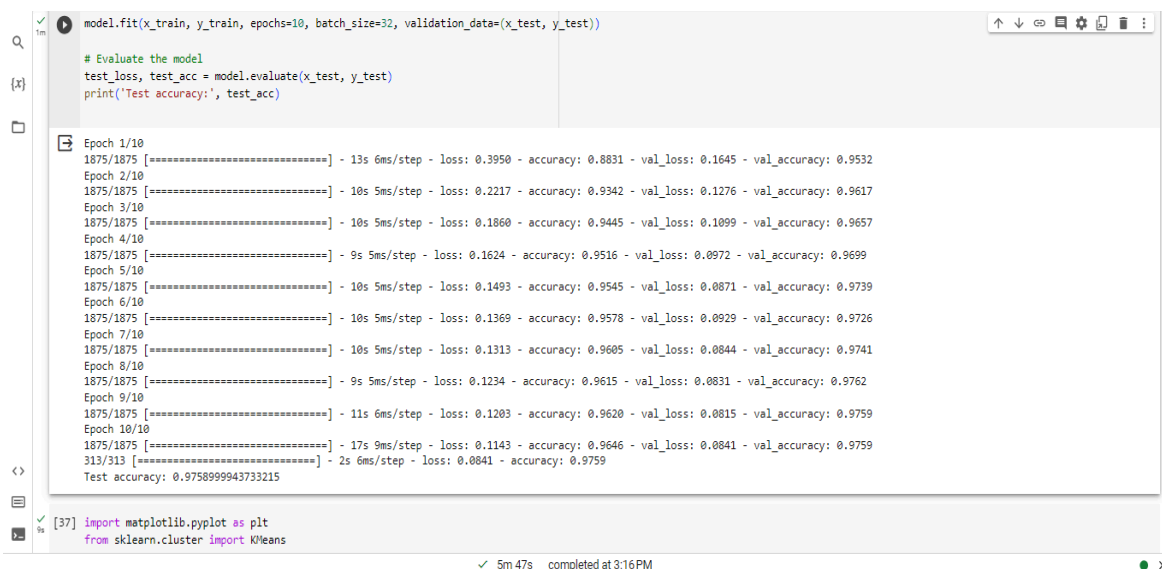
# Define the model architecture
model = keras.models.Sequential([
    keras.layers.Dense(128, activation='relu',
input_shape=(28 * 28,)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)

```



```

model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)

```

```

Epoch 1/10
1875/1875 [=====] - 13s 6ms/step - loss: 0.3950 - accuracy: 0.8831 - val_loss: 0.1645 - val_accuracy: 0.9532
Epoch 2/10
1875/1875 [=====] - 10s 5ms/step - loss: 0.2217 - accuracy: 0.9342 - val_loss: 0.1276 - val_accuracy: 0.9617
Epoch 3/10
1875/1875 [=====] - 10s 5ms/step - loss: 0.1860 - accuracy: 0.9445 - val_loss: 0.1099 - val_accuracy: 0.9657
Epoch 4/10
1875/1875 [=====] - 9s 5ms/step - loss: 0.1624 - accuracy: 0.9516 - val_loss: 0.0972 - val_accuracy: 0.9699
Epoch 5/10
1875/1875 [=====] - 10s 5ms/step - loss: 0.1493 - accuracy: 0.9545 - val_loss: 0.0871 - val_accuracy: 0.9739
Epoch 6/10
1875/1875 [=====] - 10s 5ms/step - loss: 0.1369 - accuracy: 0.9578 - val_loss: 0.0929 - val_accuracy: 0.9726
Epoch 7/10
1875/1875 [=====] - 10s 5ms/step - loss: 0.1313 - accuracy: 0.9605 - val_loss: 0.0844 - val_accuracy: 0.9741
Epoch 8/10
1875/1875 [=====] - 9s 5ms/step - loss: 0.1234 - accuracy: 0.9615 - val_loss: 0.0831 - val_accuracy: 0.9762
Epoch 9/10
1875/1875 [=====] - 11s 6ms/step - loss: 0.1203 - accuracy: 0.9620 - val_loss: 0.0815 - val_accuracy: 0.9759
Epoch 10/10
1875/1875 [=====] - 17s 9ms/step - loss: 0.1143 - accuracy: 0.9646 - val_loss: 0.0841 - val_accuracy: 0.9759
313/313 [=====] - 2s 6ms/step - loss: 0.0841 - accuracy: 0.9759
Test accuracy: 0.975899943733215

```

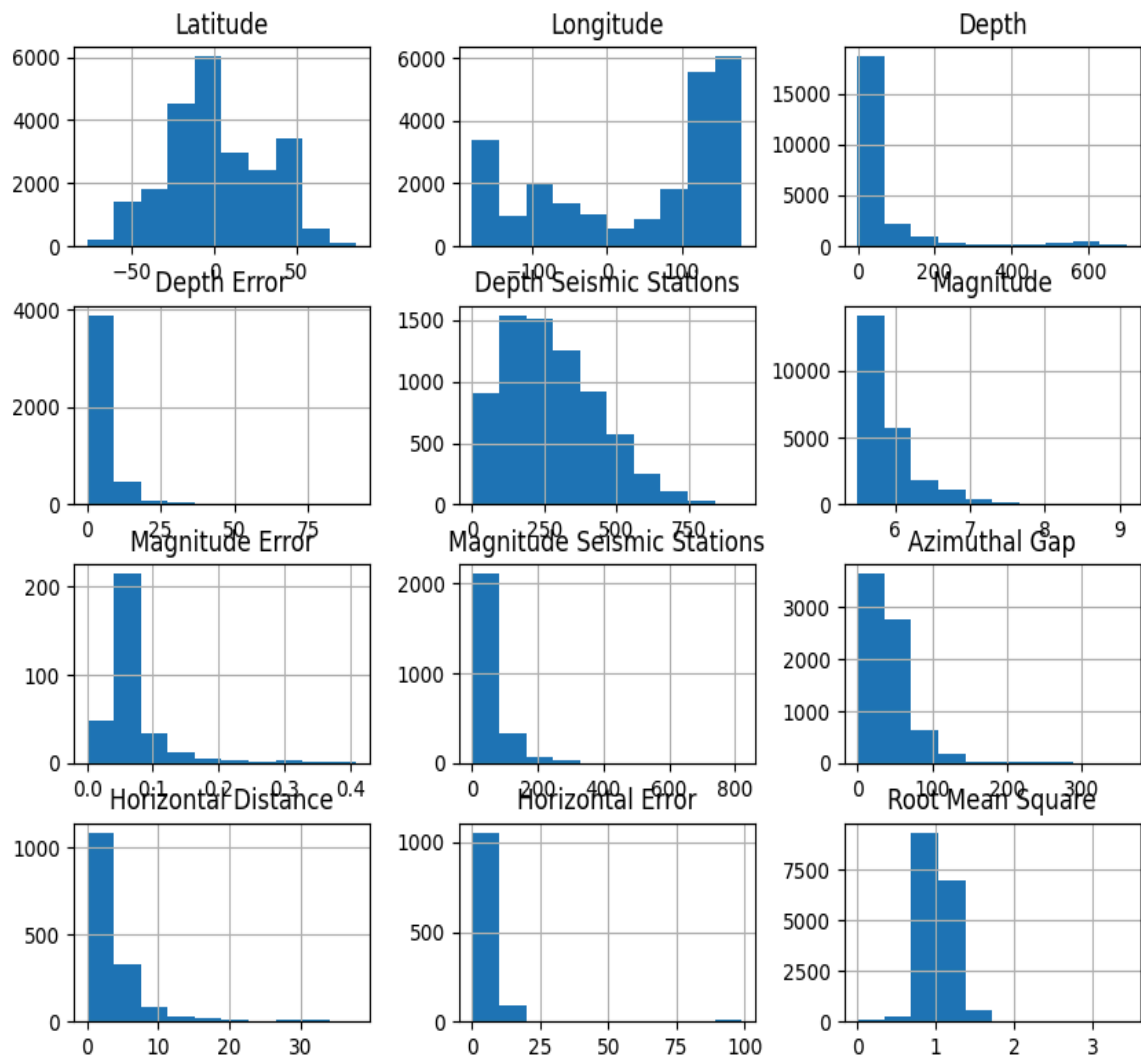
```

[37] import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

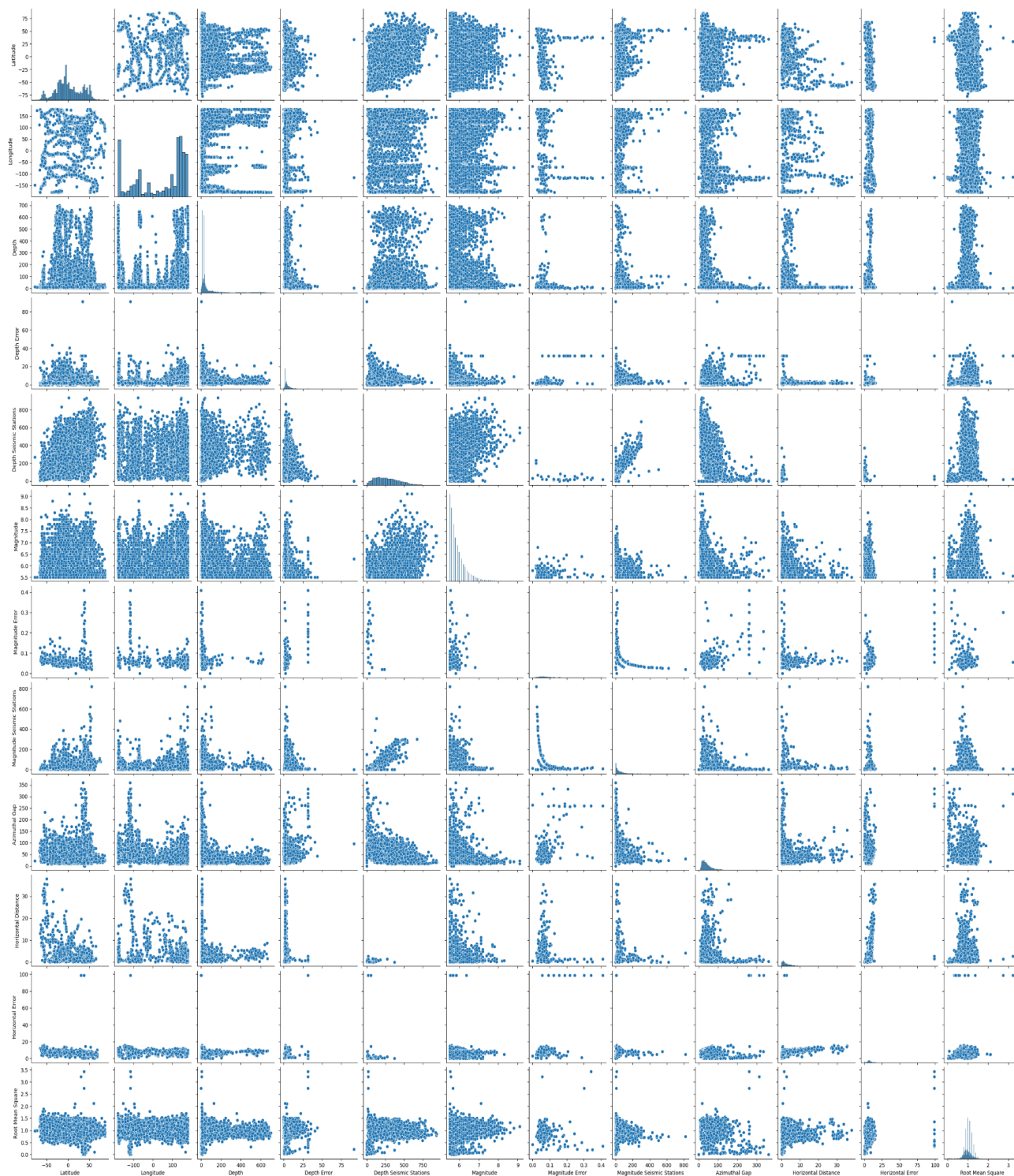
```

5m 47s completed at 3:16 PM

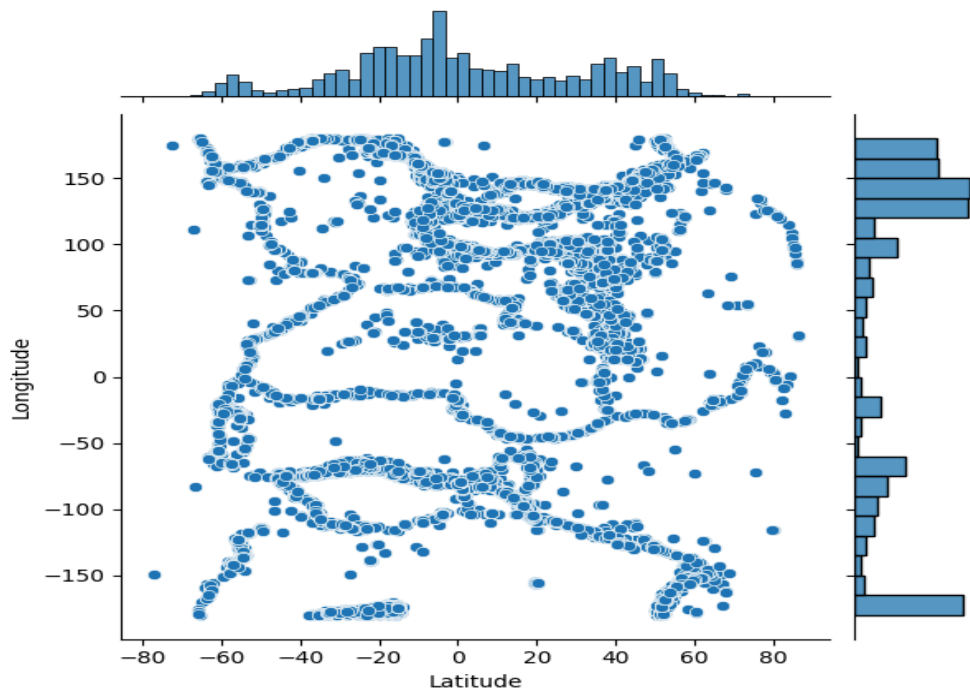
```
eq.hist(figsize=(10,8))
```



```
sns.pairplot(eq, hue="Depth")
```



```
sns.jointplot(eq,x='Latitude',y='Longitude')
```



```
eq.isnull().sum()
```

Latitude

```
eq.isnull().sum()
```

Date	0
Time	0
Latitude	0
Longitude	0
Type	0
Depth	0
Depth Error	18951
Depth Seismic Stations	16315
Magnitude	0
Magnitude Type	3
Magnitude Error	23085
Magnitude Seismic Stations	28048
Azimuthal Gap	16113
Horizontal Distance	21088
Horizontal Error	22256
Root Mean Square	6060
ID	0
Source	0
Location Source	0
Magnitude Source	0
Status	0
dtype: int64	

Some common data preprocessing tasks include:

1. Data cleaning:

This involves identifying and correcting errors and inconsistencies in the data. For example, this may involve removing duplicate records, correcting typos, and filling in missing values.

2. Data transformation:

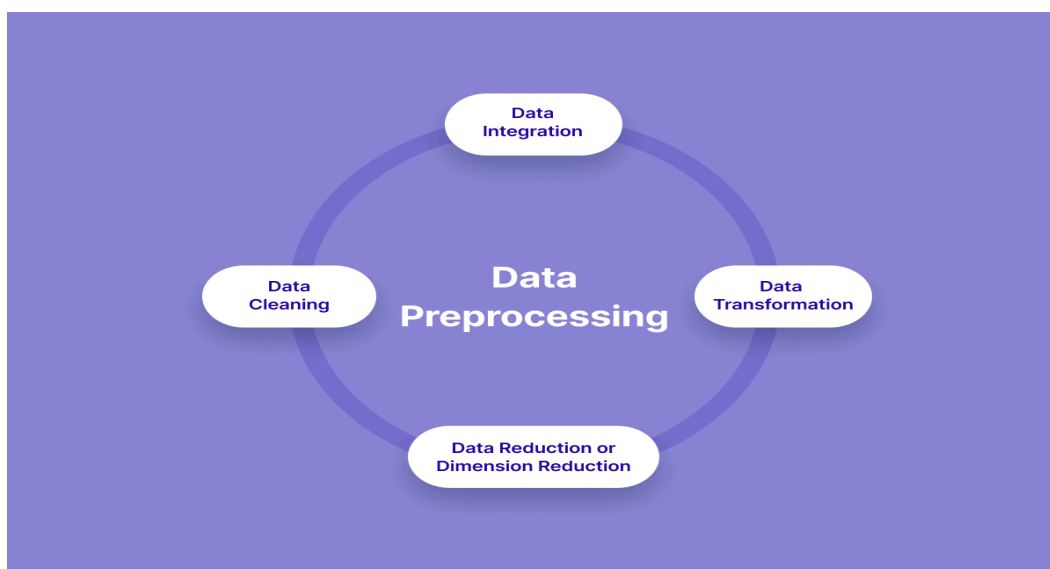
This involves converting the data into a format that is suitable for the analysis task. For example, this may involve converting categorical data to numerical data, or scaling the data to a suitable range.

3. Feature engineering:

This involves creating new features from the existing data. For example, this may involve creating features that represent interactions between variables, or features that represent summary statistics of the data.

4. Data integration:

This involves combining data from multiple sources into a single dataset. This may involve resolving inconsistencies in the data, such as different data formats or different variable names.

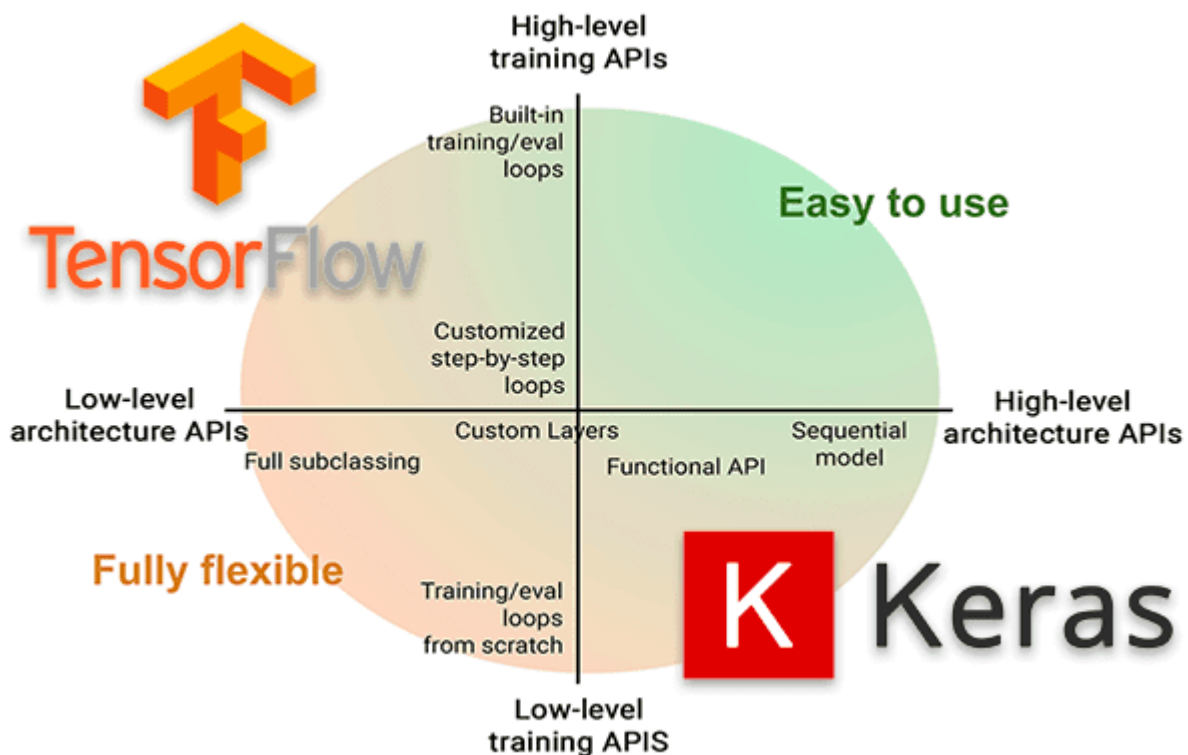


TensorFlow :

TensorFlow is an open-source platform for machine learning and a symbolic math library that is used for machine learning applications.

Keras :

It is an Open Source Neural Network library that runs on top of Theano or Tensorflow. It is designed to be fast and easy for the user to use. It is a useful library to construct any deep learning algorithm of whatever choice we want.



Convolutional Neural Network (CNN):

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence enables a computer to understand and interpret the visual data.

When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network there are three types of layers:

1. Input Layer :

It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).

2. Hidden Layer :

The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.

3. Output Layer :

The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

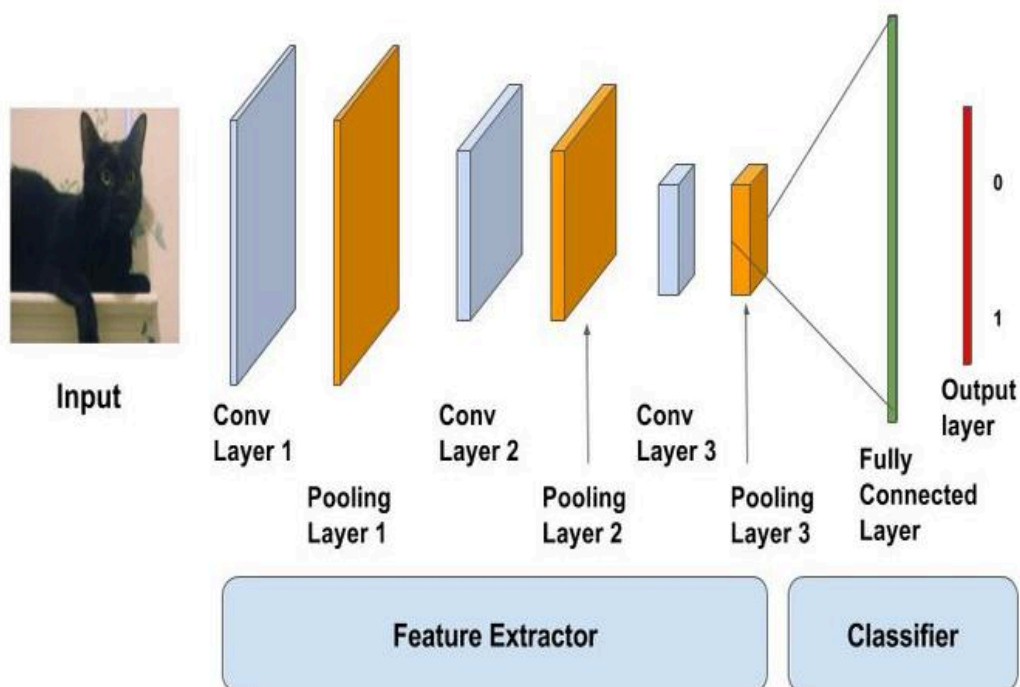
The data is fed into the model and output from each layer is obtained from the

above step is called **feedforward**, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing.

After that, we backpropagate into the model by calculating the derivatives. This step is called **Backpropagation** which basically is used to minimize the loss.

CNN architecture :

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

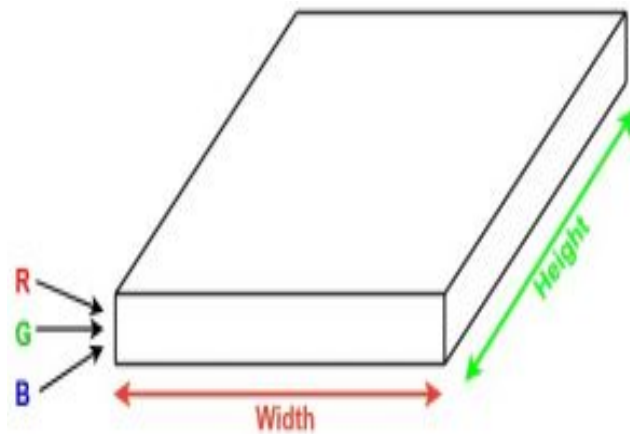


The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through

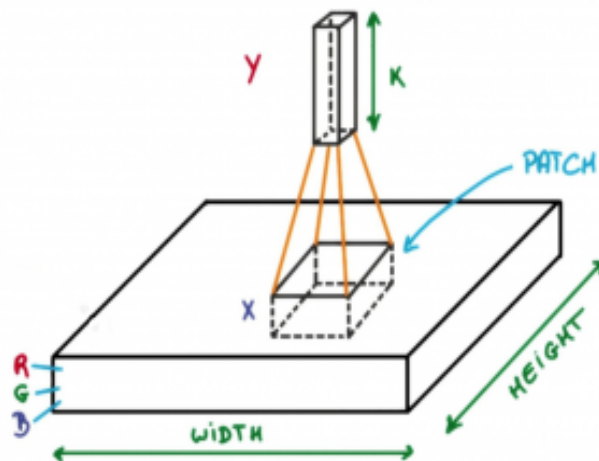
backpropagation and gradient descent.

How Convolutional Layers works ?

Convolution Neural Networks or covers are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called **Convolution**. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.



Now let's talk about a bit of mathematics that is involved in the whole convolution process.

- Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimensions $34 \times 34 \times 3$. The possible size of filters can be $a \times a \times 3$, where 'a' can be anything like 3, 5, or 7 but smaller as compared to the image dimension.
- During the forward pass, we slide each filter across the whole input volume step by step where each step is called **stride** (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume.
- As we slide our filters we'll get a 2-D output for each filter and we'll stack them together as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

EXAMPLE PROGRAM:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from itertools import product
plt.rc('figure', autolayout=True)
plt.rc('image', cmap='magma')
kernel =tf.constant([[[-1,-1,-1],
                        [-1,8,-1],
                        [-1,-1,-1]],])
image =tf.io.read_file('Sampleimg.jpg')
image =tf.io.decode_jpeg(image,channels=1)
image =tf.image.resize(image,size=[300, 300])
img =tf.squeeze(image).numpy()
plt.figure(figsize=(5,5))
plt.imshow(img,cmap='gray')
plt.axis('off')
plt.title('Original Gray Scale image')
plt.show();
image =tf.image.convert_image_dtype(image, dtype
=tf.float32)
image =tf.expand_dims(image, axis=0)
kernel =tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel =tf.cast(kernel, dtype=tf.float32)
conv_fn =tf.nn.conv2d

image_filter = conv_fn( input=image, filters=kernel,
strides=1,# or (1,1)
padding= 'SAME',)
```

```

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(tf.squeeze(image_filter))
plt.axis('off')
plt.title('Convolution')
relu_fn =tf.nn.relu
image_detect =relu_fn(image_filter)
plt.subplot(1,3,2)
plt.imshow(tf.squeeze(image_condense))
tf.squeeze(image_detect))
plt.axis('off')
plt.title('Activation')
pool =tf.nn.pool
image_condense =pool(input= image_detect,
                      window_shape=(2,2) ,
                      pooling_type='MAX' ,
                      strides=(2,2) ,
                      padding='SAME' ,)

plt.subplot(1,3,3)
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title('Pooling')
plt.show()

```

Advantages of Convolutional Neural Networks (CNNs):

1. Good at detecting patterns and features in images, videos, and audio signals.
2. Robust to translation, rotation, and scaling invariance.
3. End-to-end training, no need for manual feature extraction.

4. Can handle large amounts of data and achieve high accuracy.

Disadvantages of Convolutional Neural Networks (CNNs):

1. Computationally expensive to train and require a lot of memory.
2. Can be prone to overfitting if not enough data or proper regularization is used.
3. Requires large amounts of labeled data.
4. Interpretability is limited, it's hard to understand what the network has learned.

OpenCV :

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

Applications of OpenCV:

There are lots of applications which are solved using OpenCV, some of them are listed below :

- face recognition
- Automated inspection and surveillance
- number of people – count (foot traffic in a mall, etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anomaly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching

- Video/image search and retrieval
- Robot and driver-less car navigation and control
- object recognition
- Medical image analysis
- Movies – 3D structure from motion
- TV Channels advertisement recognition

OpenCV Functionality :

- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular
- stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA acceleration (gpu)

CONCLUSION:

In the quest to build a Earthquake prediction model, we have embarked on critical journey that begins with loading and preprocessing the dataset. We have traversed through essential steps, starting with importing the necessary libraries to facilitate data manipulation and analysis. Understanding the data's structure, characteristics, and any potential issues through exploratory data analysis (EDA), CNN, Keras, OpenCV and Tensorflow is essential for informed decision-making.

Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning Algorithms. With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a Earthquake prediction model. Earthquake prediction is yet an immature science and has

not yet been completely successful. In this project we can try to find patterns from previous data and based on those patterns try to predict the magnitude of earthquake that can occur.

I conclude that this project has taught me a lot about machine learning And It is great of how these technologies can help in real life applications.