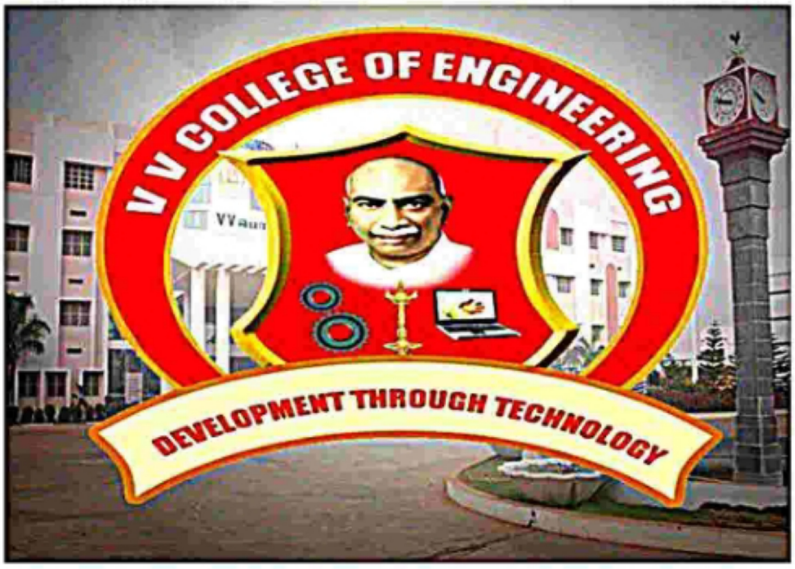


VV COLLEGE OF ENGINEERING

NALAIYA THIRAN PHASE 5 PROJECT-DOCUMENTATION AND SUBMISSION

NAME: P.HUDSHIYA

REG NO: 953421106016

DEPARTMENT: ECE

YEAR:III

BATCH:2021-2025

COURSE:ARTIFICIAL INTELLIGENCE

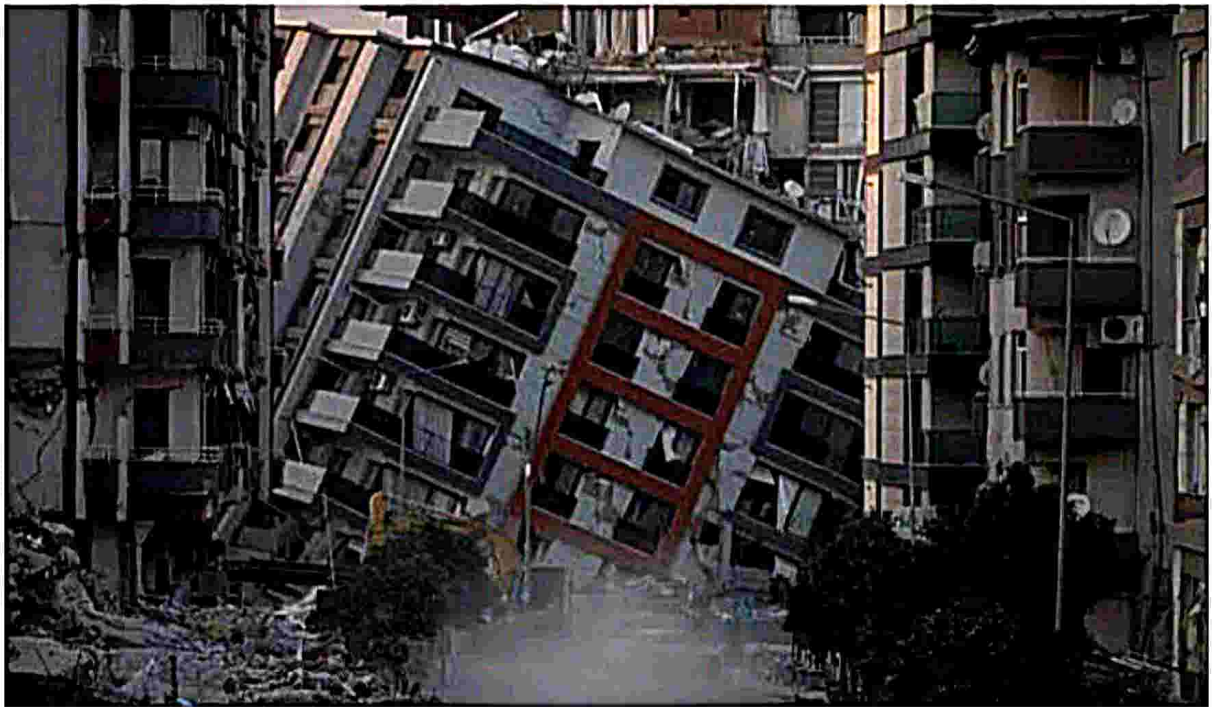
PROJECT TITLE: EARTHQUAKE PREDICTION USING MACHINE LEARNING PYTHON

ID:au953421106016

EARTHQUAKE PREDICTION MODEL USING MACHINE LEARNING-PYTHON

Objective:

An earthquake is the sudden release of strain energy in the Earth's crust, resulting in waves of shaking that radiate outwards from the earthquake source.



Problem Statement :

- An earthquake is a violent and abrupt shaking of the ground, caused by movement between tectonic plates along a fault line in the earth's crust.
- When the stored energy beneath the crust is suddenly released as an earthquake, the crust's response to the changing stress beneath it is not directly proportional.

- An earthquake prediction must define 3 elements:
  1) the date and time
  2) the location
  3) the magnitude
- In order to anticipate earthquakes, machine learning may be used to examine seismic data trends.
- Earthquakes can result in the ground shaking, soil liquefaction, landslides, fissures, avalanches, fires and tsunamis.
- The environmental effects of it are that including surface faulting, tectonic uplift and subsidence, tsunamis, soil liquefaction, ground resonance, landslides and ground failure, either directly linked to a quake source or provoked by the ground shaking.
- The main problem occur in earthquake occurs loss of natural,human and environment surrounding.It leads to disruption in economic activities.
- Seismologists study earthquakes by looking at the damage that was caused and by using seismometers. A seismometer is an instrument that records the shaking of the Earth's surface caused by seismic waves. The term seismograph usually refers to the combined seismometer and recording device.

Design thinking:

Earthquakes were once thought to result from supernatural forces in the prehistoric era. Aristotle was the

first to identify earthquakes as a natural occurrence and to provide some potential explanations for them in a truly scientific manner. One of nature's most destructive dangers is earthquakes. Strong earthquakes frequently have negative effects.

A lot of devastating earthquakes occasionally occur in nations like Japan, the USA, China, and nations in the middle and far east. Several major and medium-sized earthquakes have also occurred in India, which have resulted in significant property damage and fatalities. One of the most catastrophic earthquakes ever recorded occurred in Maharastra early on September 30, 1993. One of the main goals of researchers studying earthquake seismology is to develop effective predicting methods for the occurrence of the next severe earthquake event that may allow us to reduce the death toll and property damage.

STEPS TO IMPLEMENT:

- ✓ First create a dataset using kaggle website
- ✓ From 1990-2023 dataset
- ✓ https://www.kaggle.com/datasets/alessandrolobello/the-ultimate-earthquake-dataset-from-1990-2023
- ✓ I will start this task to create a model for earthquake prediction by importing the necessary python libraries:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

✓ Now let’s load and read the dataset.

✓ data = pd.read_csv("database.csv")
✓ data.columns
✓ Index([‘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’],

✓ Dtype=’object’)

✓ Now let’s see the main characteristics of earthquake data and create an object of these characteristics, namely, date, time, latitude, longitude, depth, magnitude:

✓

✓ Data = data[[‘Date’, ‘Time’, ‘Latitude’, ‘Longitude’, ‘Depth’, ‘Magnitude’]]

✓ Data.head()

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

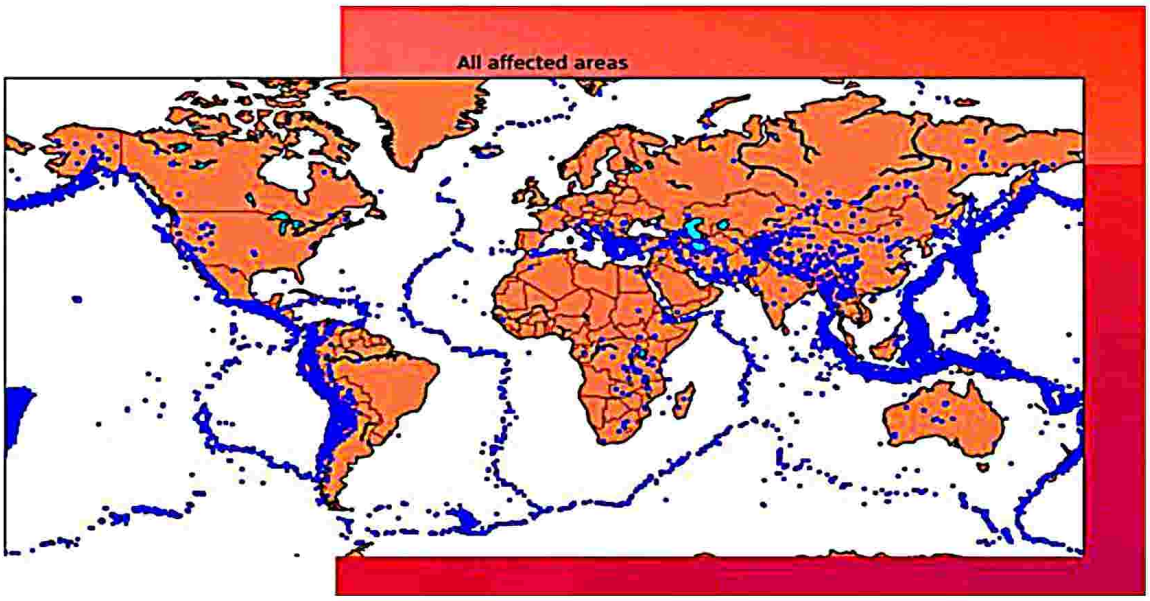
Since the data is random, so we need to scale it based on the model inputs. In this, we convert the given date and time to Unix time which is in seconds and a number. This can be easily used as an entry for the network we have built:

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

	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

Data visualation:



Now, before we create the earthquake prediction model, let's visualize the data on a world map that shows a clear representation of where the earthquake frequency will be more:

```
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()
```

Data splitting:

Now, to create the earthquake prediction model, we need to divide the data into Xs and ys which respectively will be entered into the model as inputs to receive the output from the model.

Here the inputs are TImestamp, Latitude and Longitude and the outputs are Magnitude and Depth. I'm going to split the xs and ys into train and test with validation. The training set contains 80% and the test set contains 20%:

```
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
```

Data Exploration:

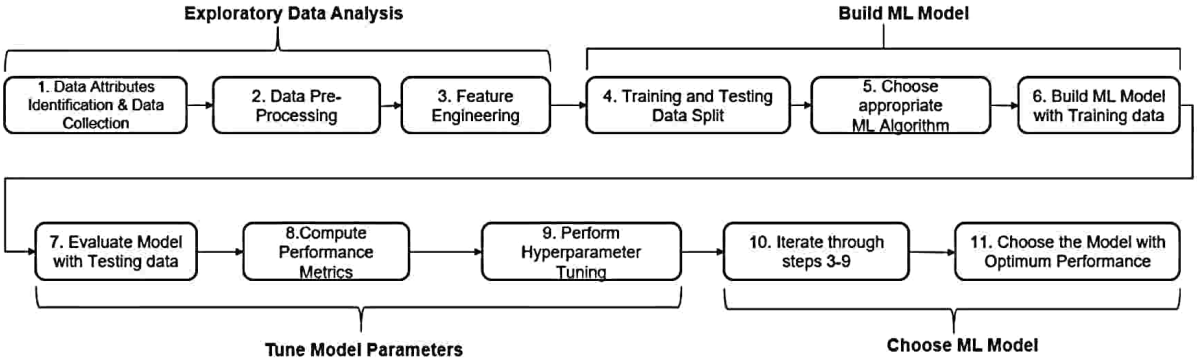
A Machine Learning project is as good as the foundation of data on which it is built. In order to perform well, machine learning data exploration models must ingest large quantities of data, and model accuracy will suffer if that data is not thoroughly explored first. Data exploration steps to follow before building a machine learning model include:

- Variable identification: define each variable and its role in the dataset
- Univariate analysis: for continuous variables, build box plots or histograms for each variable independently; for categorical variables, build bar charts to show the frequencies
- Bi-variable analysis - determine the interaction between variables by building visualization tools
- ~Continuous and Continuous: scatter plots
- ~Categorical and Categorical: stacked column chart
- ~Categorical and Continuous: boxplots combined with swarmplots
- Detect and treat missing values
- Detect and treat outliers

Model development:

Building an ML Model requires splitting of data into two sets, such as ‘training set’ and ‘testing set’ in the ratio of 80:20 or

70:30; A set of supervised (for labelled data) and



unsupervised (for unlabeled data) algorithms are available to choose from depending on the nature of input data and business outcome to predict. In case of labelled data, it is recommended to choose logistic regression algorithm if the outcome to predict is of binary (0/1) in nature; choose decision tree classifier (or) Random Forest® classifier (or) KNN if the outcome to predict is of multi-class (1/2/3/...) in nature; and choose linear regression algorithm (e.g., decision tree regressor, random forest regressor) if the outcome to predict is continuous in nature. The clustering (Unsupervised) algorithm is preferred to analyze unlabeled / unstructured (text) data (e.g., k-means clustering). Artificial Neural Network (ANN) algorithms are suggested to analyze other unstructured (image/voice) data types, such as Convolutional Neural Networks (CNN) for image recognition and Recurrent Neural Networks (RNN) for voice recognition and Natural Language Processing (NLP). The model is built using training dataset and make prediction using test dataset. Use of deep learning (neural networks) models is preferred over regression models (ML models) for better performance as these models introduce extra layer of non-linearity with the introduction of Activation Function (AF).

Earthquake prediction using machine learning - PYTHON

PHASE 2 - INNOVATION

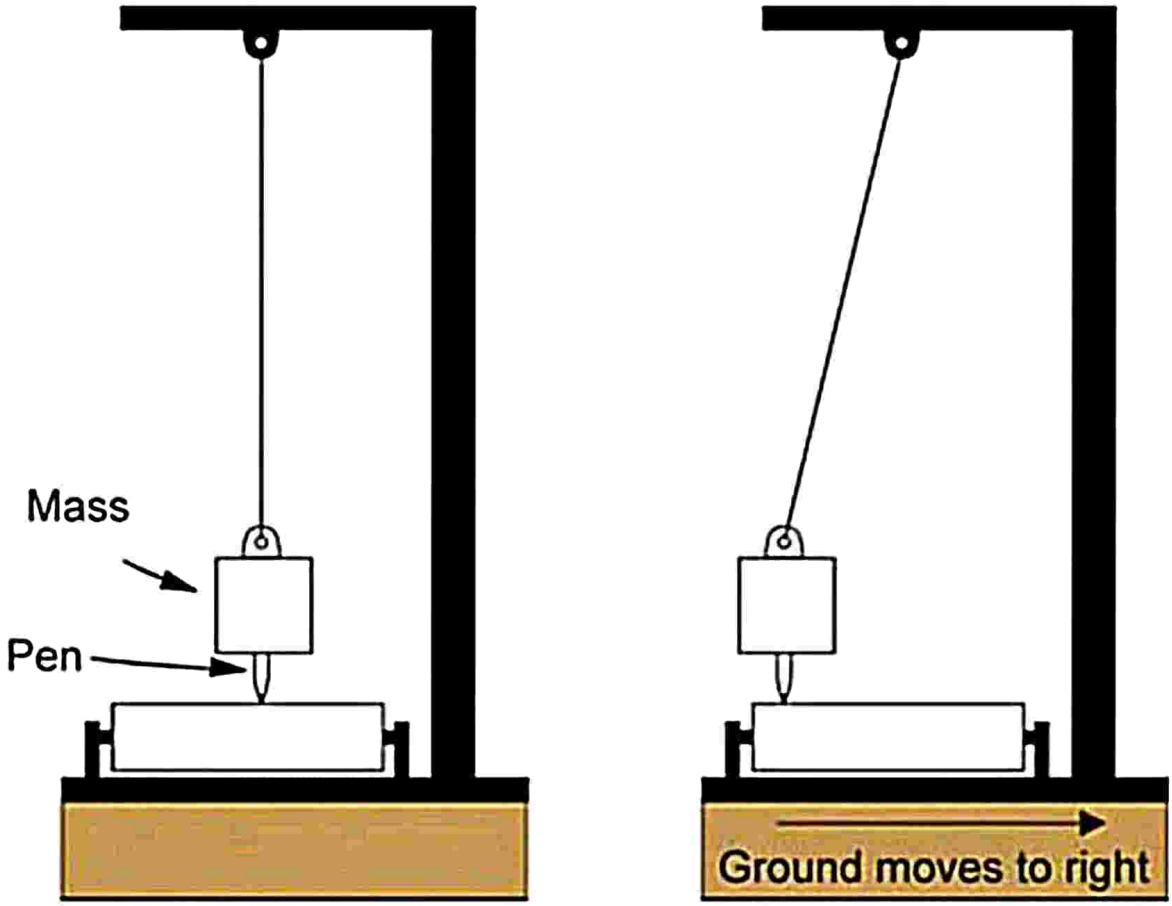
ABSTRACT :

Earthquake prediction has been a challenging research area, where a future occurrence of the devastating catastrophe is predicted. In this work, sixty seismic features are computed through employing seismological concepts, such as Gutenberg-Richter law, seismic rate changes, foreshock frequency, seismic energy release, total recurrence time. Further, Maximum Relevance and Minimum Redundancy (mRMR) criteria is applied to extract the relevant features. A Support Vector Regressor (SVR) and Hybrid Neural Network (HNN) based classification system is built to obtain the earthquake predictions. HNN is a step wise combination of three different Neural Networks, supported by Enhanced Particle Swarm Optimization (EPSO), to offer weight optimization at each layer. The newly computed seismic features in combination with SVR-HNN prediction system is applied on Hindukush, Chile and Southern California regions. The obtained numerical results show improved prediction performance for all the g regions, compared to previous prediction studies.

The following are the innovation tools used to predict the earthquake,

SEISMOMETER:

A seismogram is a record of the ground motions caused by seismic waves from an earthquake. A seismograph or seismometer is the measuring instrument that creates the seismogram. Almost all seismometers are based on the principle of inertia, that is, where a suspended mass tends to remain still when the ground moves.



Seismometers allow us to detect and measure earthquakes by converting vibrations due to seismic waves into electrical signals, which we can then display as seismograms on a computer screen. Seismologists study earthquakes and can use this data to determine where and how big a particular earthquake is.

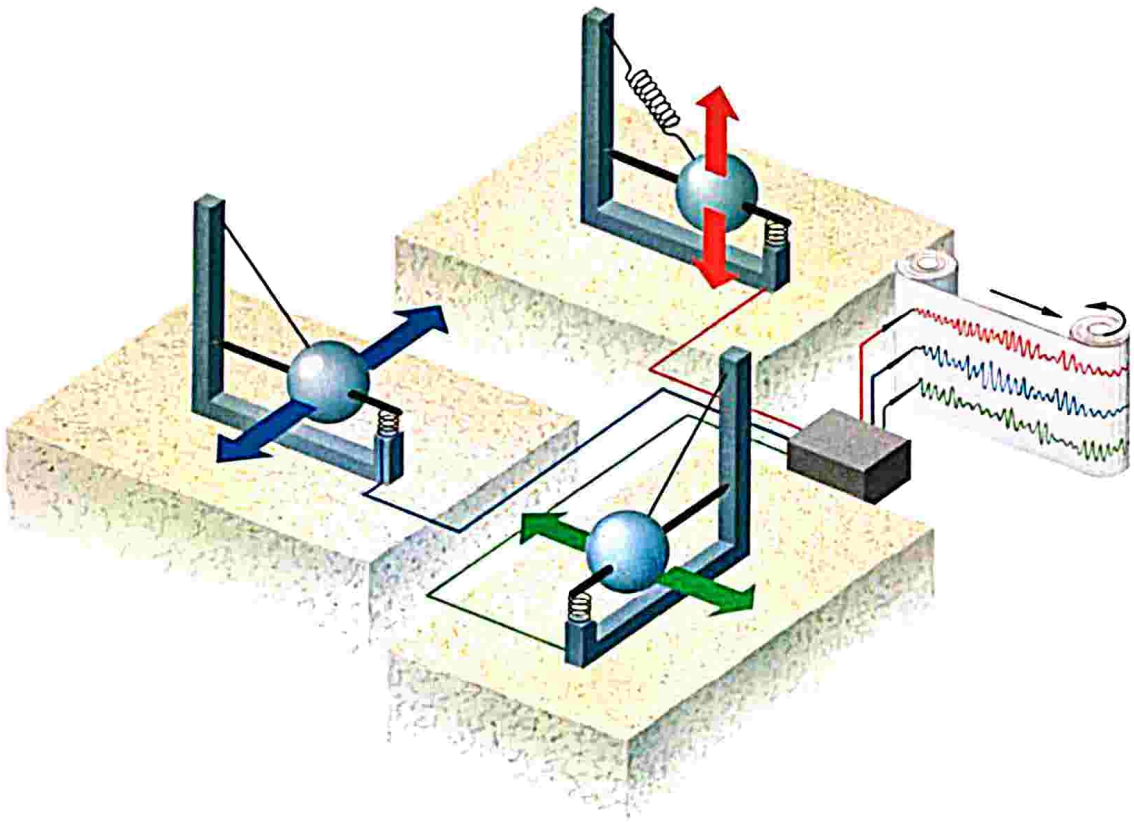
To record the actual motion of the ground in all three dimensions, seismologists need to use three separate sensors within the same

instrument. Each sensor records the vibrations in a different direction:

The Z component measures up/down motion

The E component measures east/west motion

The N component measures north-south motion



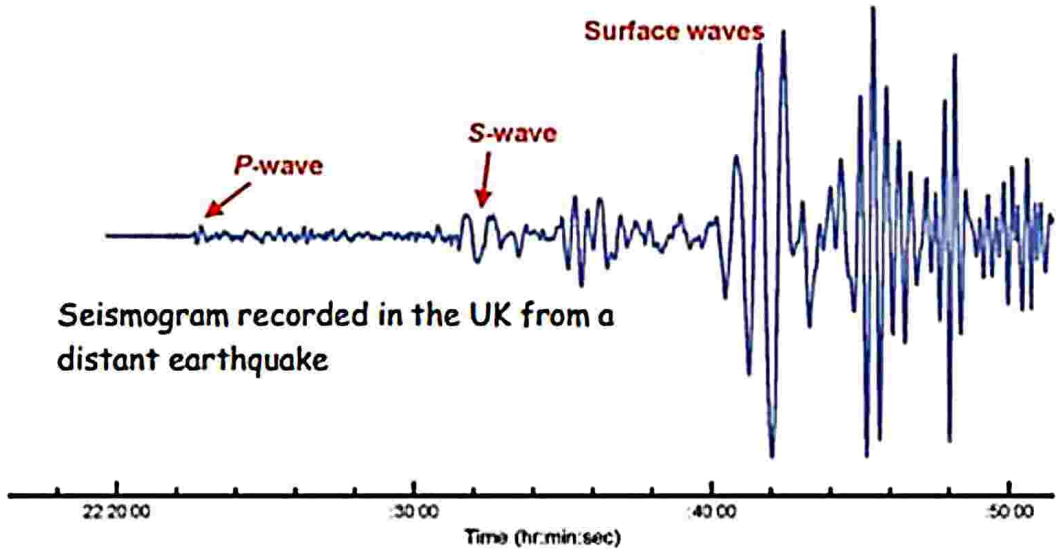
PARAMETERS:

Seismic waves

There are two basic types of seismic wave that travel through the body of the Earth: P-waves and S-waves. P-waves are longitudinal waves that consist of a series of compressions and dilations along the direction of travel. The P stands for primary because they travel the fastest. S-waves are transverse waves, whose motion is perpendicular to the direction of travel. The S stands for shear or secondary since they are slower than P-waves.

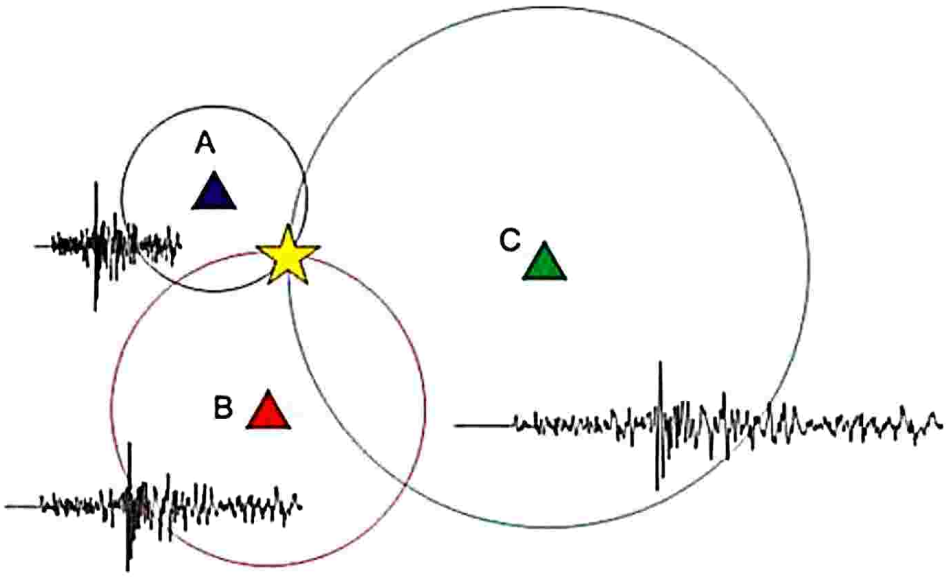
Where a free surface is present (like the Earth/air interface) these two types of motion can combine to form surface waves, which produce a type of shaking that causes buildings to fail and fall down. There are two types of surface waves: Rayleigh waves and Love waves. Rayleigh waves are generated by the interaction of P- and S-waves at the surface of the Earth, while Love waves are generated by interference of multiple shear waves. The ground motions from surface waves are often much larger than those motions from body waves.

Earthquakes generate different types of seismic waves and these travel at different speeds through the Earth. P-waves are fastest and are the first signal to arrive on a seismogram, followed by the slower S-wave, then the surface waves. The arrival times of the P- and S-waves at different seismometers are used to determine the location of the earthquake. Assuming that we know the relative speed of P- and S-waves, the time difference between the arrivals of the P- and S-waves determines the distance the earthquake is from the seismometer.



By looking at the seismograms from different recording stations, we can find out the epicentre of the earthquake. The signals arrive first at the closest station and last at the one furthest away. The time difference between the P- and S-waves tells us the distance the earthquake is from the seismometer. If we calculate the S minus P time to determine distance from the seismometer at three stations, we can work out where the epicentre of the earthquake is

Imagine A, B and C are three different seismometer stations at



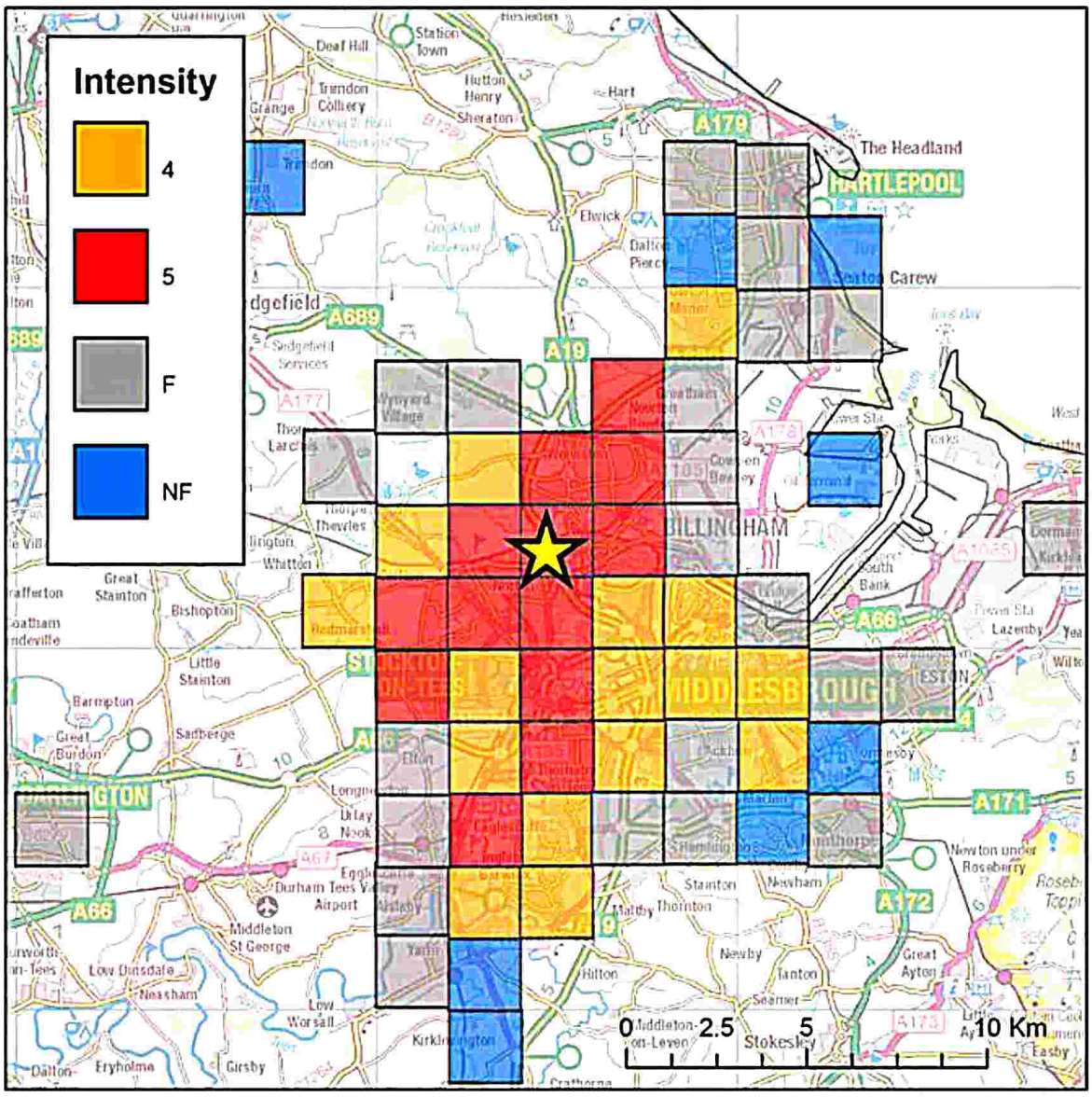
distant locations. Once we know the distance to an earthquake from three seismic stations, we can determine the location of the earthquake. Draw a circle around each station with a radius equal to its distance from the earthquake. The earthquake occurred at the point where all three circles intersect. BGS ©UKRI. All rights reserved.

Earthquake intensity

Intensity is a qualitative measure of the strength of shaking caused by an earthquake determined from the observed effects on people, objects and buildings. For a given earthquake, the intensity normally decreases with distance from the epicentre. There are a number of different intensity scales in use around the world that are all based on the shaking people experience and the effects it has on objects and buildings. It is also possible to estimate intensity from recordings of ground motions.

Macroseismic intensities (EMS) for the magnitude 3.1 ML earthquake on 23 January 2020, near Stockton-on-Tees, UK. The yellow star shows the earthquake epicentre. Intensities are calculated in 2 km grid squares from over 840 reports from people who felt the

earthquake. A minimum of five observations are needed in any grid square to calculate a value of intensity, otherwise the value is recorded as ‘Felt’, but no intensity is calculated (shown by grey squares). Blue squares indicate that reports from these locations suggest that the earthquake was not felt.

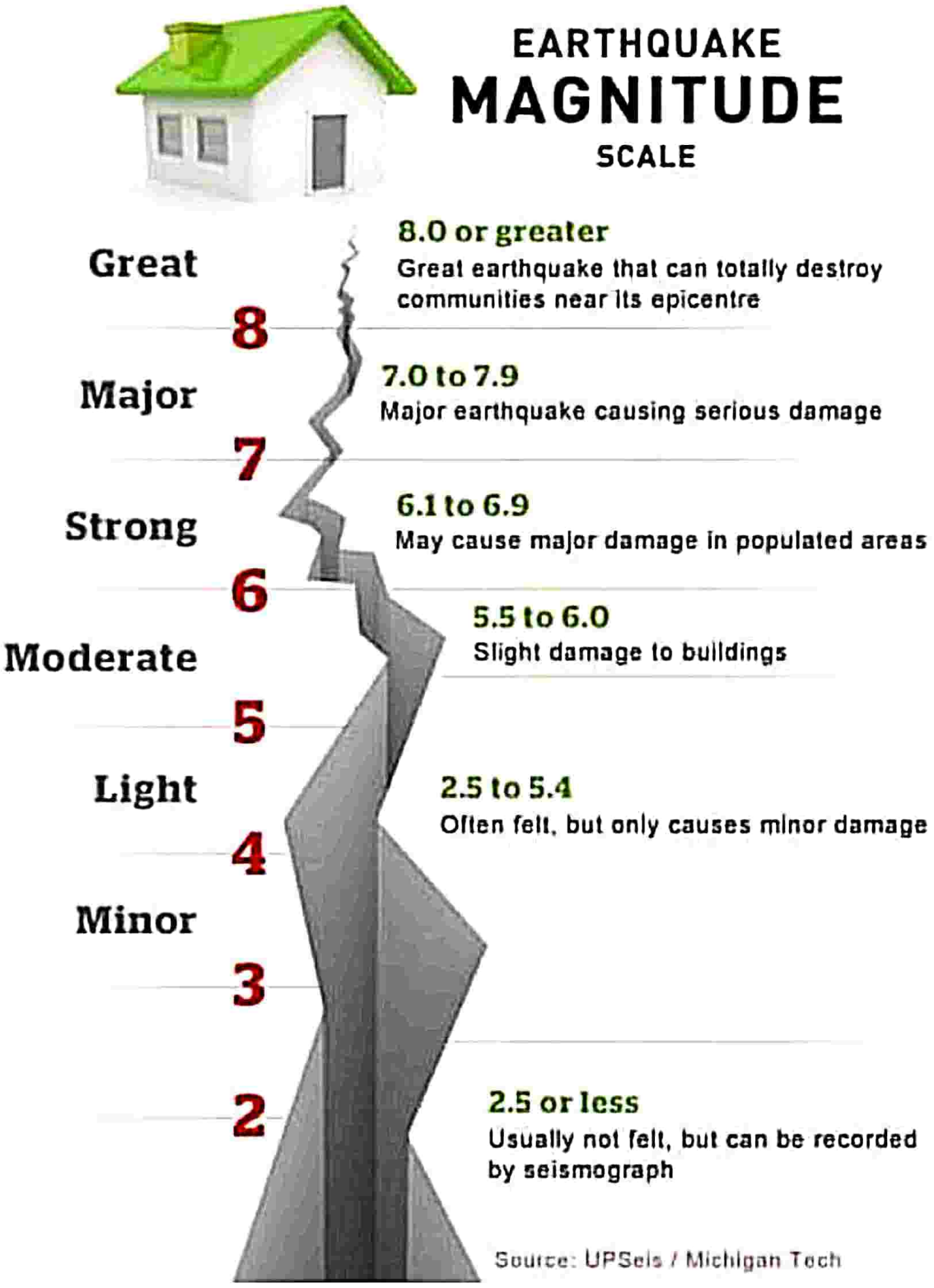


Earthquake magnitude

Magnitude is a measure of the amount of energy released during an earthquake and can be estimated from the

amplitude of ground motions recorded by seismometers. It is independent of distance from the epicentre.

A number of different magnitude scales have been developed based on the amplitude of different parts of the observed record of ground motion with specific corrections for distance. Earthquake magnitude scales are logarithmic, i.e. a one unit increase in magnitude corresponds to a tenfold increase in amplitude.



Moment magnitude

Nowadays, the most standard and reliable measure of earthquake size is moment magnitude (Mw), which is based on seismic 'moment'. Moment is related to the area of the earthquake fault rupture and the amount of slip on the rupture, as well as the strength of the rocks themselves. Richter's original magnitude scale underestimates the size of large events, so the constants used in the definition of Mw were chosen so that the magnitude numbers for Richter and moment magnitudes match for smaller events.

FEATURE ENGINEERING:

Feature Engineering helps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

```
Splitted = df['Origin Time'].str.split(' ', n=1,  Expand=True)
 Df['Date'] = splitted[0]
Df['Time'] = splitted[1].str[:-4]
 Df.drop('Origin Time',Axis=1   Inplace=True)
Df.head()
```

Output:

	Latitude	Longitude	Depth	Magnitude	Location	Date	Time
0	29.06	77.42	5.0	2.5	53km NNE of New Delhi, India	2021-07-31	09:43:23
1	19.93	72.92	5.0	2.4	91km W of Nashik, Maharashtra, India	2021-07-30	23:04:57
2	31.50	74.37	33.0	3.4	49km WSW of Amritsar, Punjab, India	2021-07-30	21:31:10
3	28.34	76.23	5.0	3.1	50km SW of Jhajjar, Haryana	2021-07-30	13:56:31
4	27.09	89.97	10.0	2.1	53km SE of Thimphu, Bhutan	2021-07-30	07:19:38

Now, let's divide the date column into the day, month, and year columns respectively.

```
Df['day'] = splitted[2].astype('int')
Df['month'] = splitted[1].astype('int')
Df['year'] = splitted[0].astype('int')
 Df.drop('Date', axis=1,  Inplace=True)
Df.head()
```

Output:

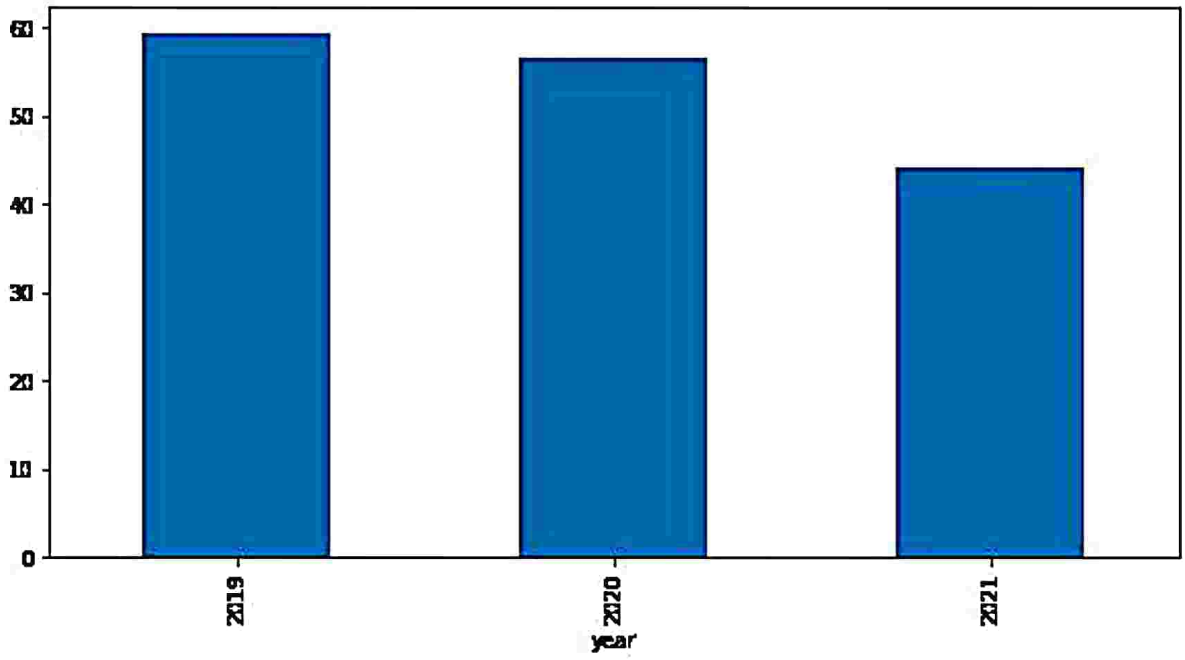
	Latitude	Longitude	Depth	Magnitude	Location	Time	day	month	year
0	29.06	77.42	5.0	2.5	53km NNE of New Delhi, India	09:43:23	31	7	2021
1	19.93	72.92	5.0	2.4	91km W of Nashik, Maharashtra, India	23:04:57	30	7	2021
2	31.50	74.37	33.0	3.4	49km WSW of Amritsar, Punjab, India	21:31:10	30	7	2021
3	28.34	76.23	5.0	3.1	50km SW of Jhajjar, Haryana	13:56:31	30	7	2021
4	27.09	89.97	10.0	2.1	53km SE of Thimphu, Bhutan	07:19:38	30	7	2021

Exploratory Data Analysis

EDA is an approach to analyzing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations

```
Plt.f
Plt.
```

```
plt.figure(figsize=(10, 5))
```




```
x = df.groupby('year').mean()['Depth']

x.plot.bar()

plt.show()
```

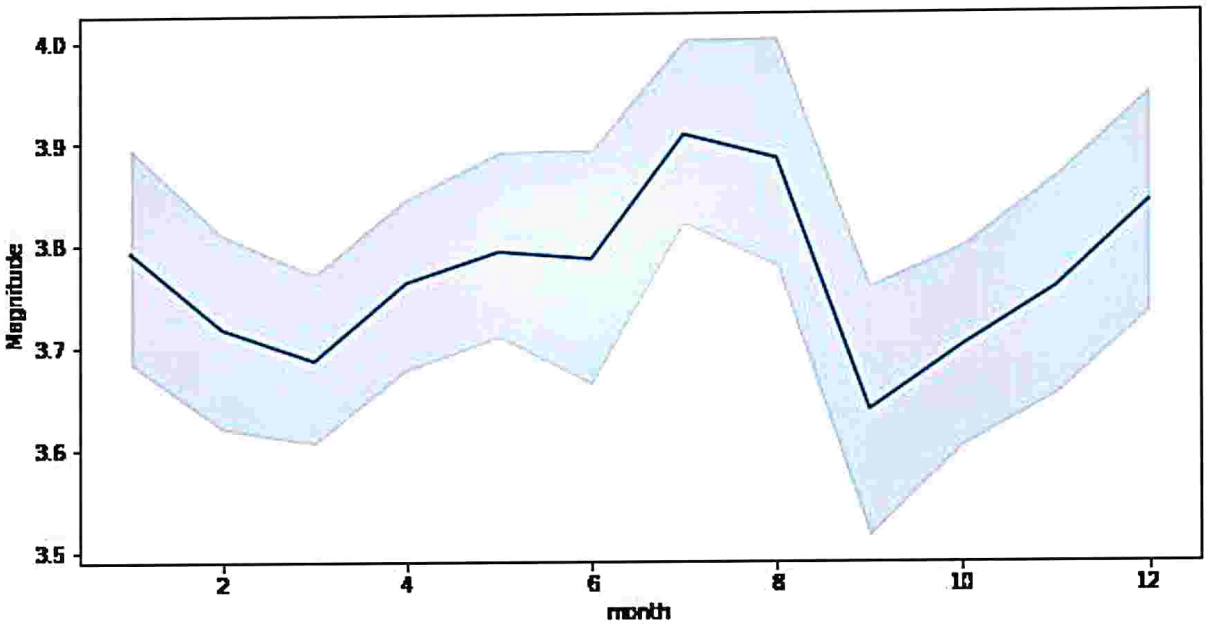
The depth from which earthquakes are starting is reducing with every passing year.

Plt.figure(figsize=(10, 5))

Sb.lineplot(data=df, X='month' Y='Magnitude')

Plt.show()

Output:



Here we can observe that the changes of an earthquake with higher magnitude are more observed during the season of monsoon.

Plt.subplots(figsize=(15, 5))

Plt.subplot(1, 2, 1)

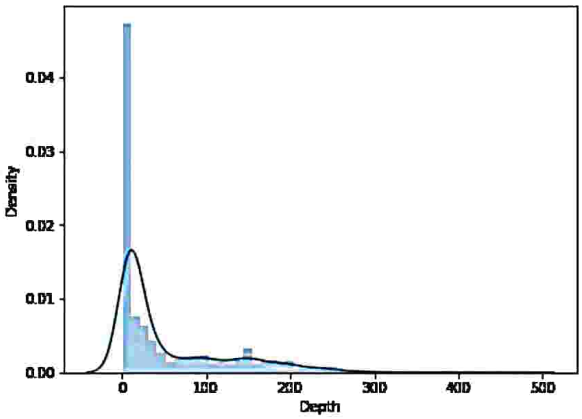
Sb.distplot(df['Depth'])

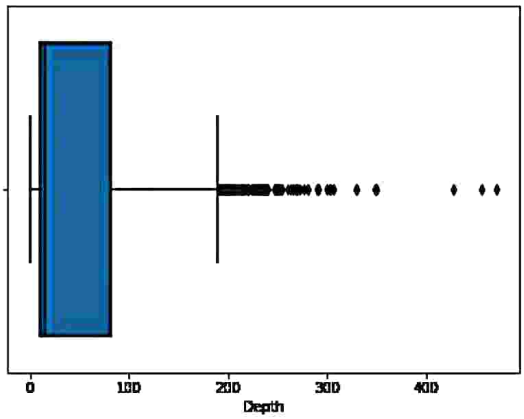
Plt.subplot(1, 2, 2)

Sb.boxplot(df['Depth'])

Plt.show()

Output:

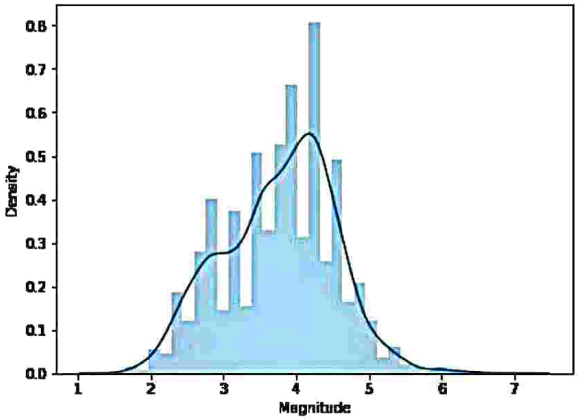


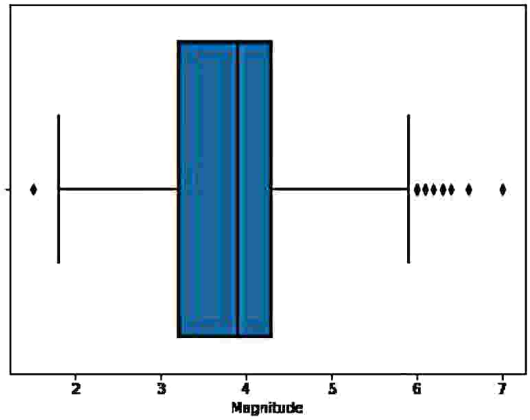


From the distribution graph, it is visible that there are some outliers that can be confirmed by using the boxplot. But the main point to observe here is that the distribution of the depth at which the earthquake rises is left-skewed.

```
Plt.subplots(figsize=(15, 5))

 Plt.subplot(1, 2, 1)

Sb.distplot(df['Magnitude'])

 Plt.subplot(1, 2, 2)

Sb.boxplot(df['Magnitude'])

 Plt.show()
```

Output:





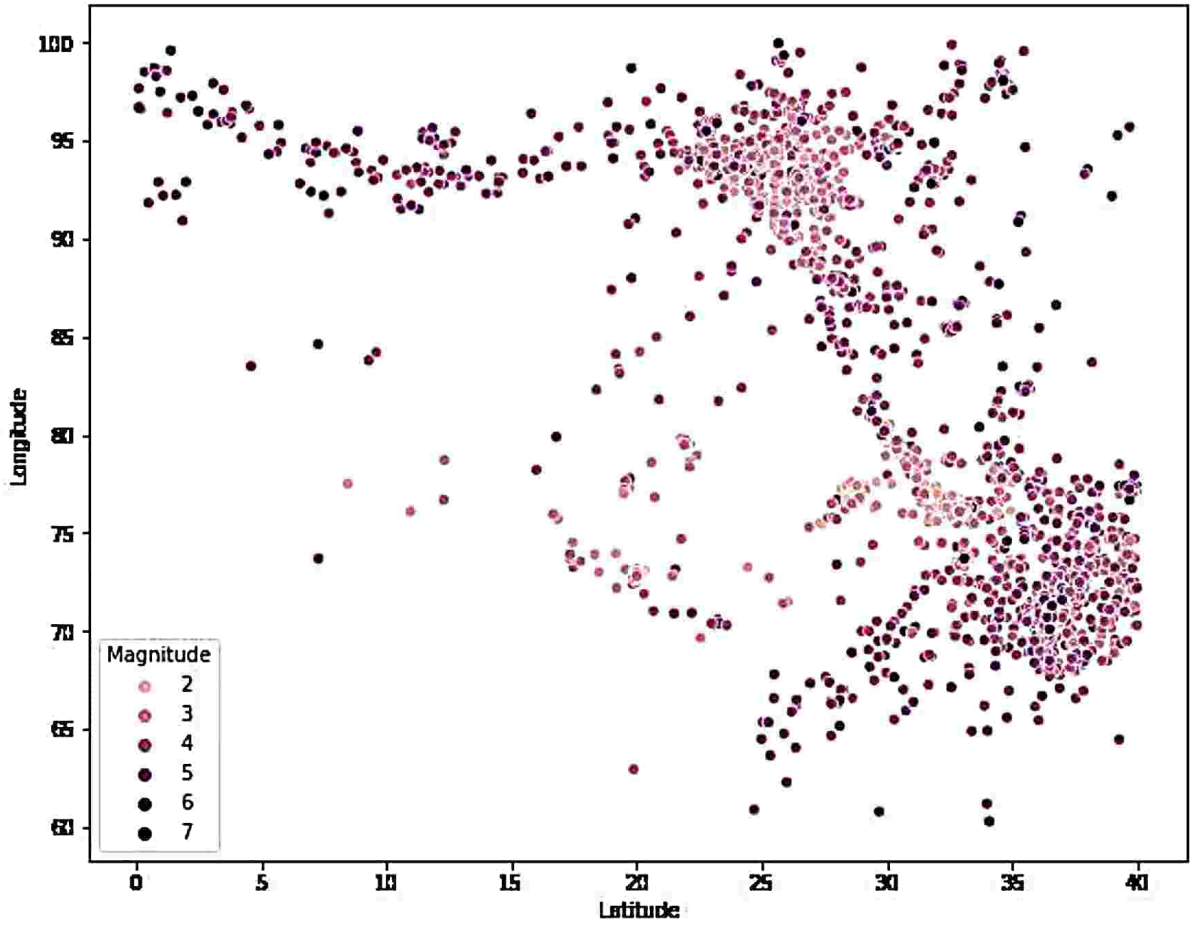
As we know that many natural phenomena follow a normal distribution and here we can observe that the magnitude of the earthquake also follows a normal distribution.

Plt.figure(figsize=(10, 8))

Sb.scatterplot(data=df, X='Latitude' Y='Longitude', Hue='Magnitude')

Plt.show()

Output:



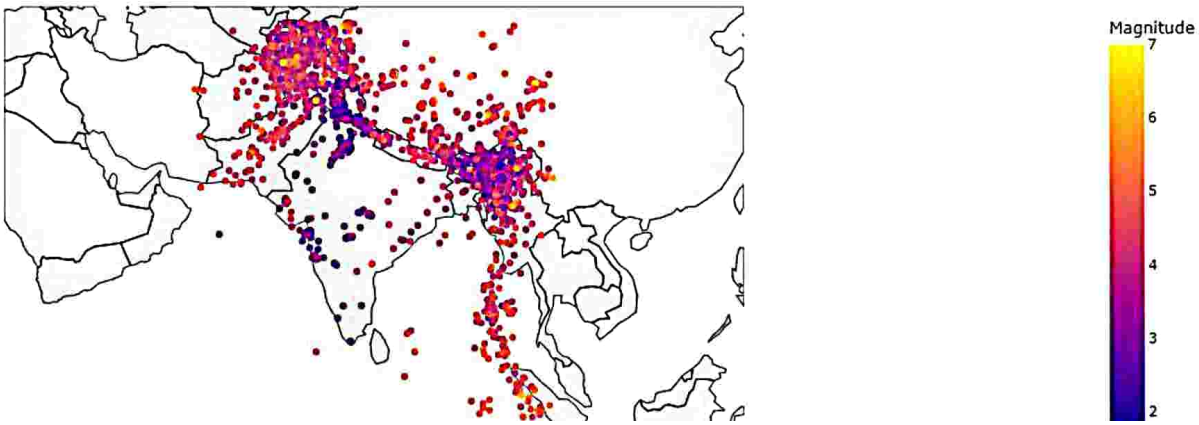
Now by using Plotly let's plot the latitude and the longitude data on the map to visualize which areas are more prone to earthquakes.

Import plotly.express as px

Import pandas as pd

```
Fig = px.scatter_geo(df, lat='Latitude',
                     Lon='Longitude',
                     Color="Magnitude",
                     Fitbounds='locations',
                     Scope='asia')
Fig.show()
```

Output:



Model Development Part 1

1.Data collection

Gather historical earthquake data, including location, magnitude, depth, and time of occurrence. Additionally, collect various geological and seismological data, such as fault line information, tectonic plate movements, and soil properties.

https://www.kaggle.com/datasets/alessandrolobello/the-ultimate-earthquake-dataset-from-1990-2023

This is the dataset link for this project from kaggle website

2.Data pre-processing

Clean and pre-process the data to handle missing values, outliers, and inconsistencies. You may need to aggregate data at different spatial and temporal scales.

1. Acquire the dataset

Acquiring the dataset is the first step in data pre-processing in machine learning. To build and develop Machine Learning models, you must first acquire the relevant dataset. This dataset will be comprised of data gathered from multiple and disparate sources which are then combined in a proper format to form a dataset. Dataset formats differ according to use cases. For instance, a business dataset will be entirely different from a medical dataset. While a business dataset will contain relevant industry and business data, a medical dataset will include healthcare-related data.

2. Import all the crucial libraries

Since Python is the most extensively used and also the most preferred library by Data Scientists around the world, we'll show you how to import Python libraries for data pre-processing in Machine Learning. Read more about Python libraries for Data Science here. The predefined Python libraries can perform specific data pre-processing jobs. Importing all the crucial libraries is the second step in data pre-processing in machine learning. The three core Python libraries used for this data pre-processing in Machine Learning are:

Numpy – Numpy is the fundamental package for scientific calculation in Python. Hence, it is used for inserting any type of mathematical operation in the code. Using Numpy, you can also add large multidimensional arrays and matrices in your code.

Pandas – Pandas is an excellent open-source Python library for data manipulation and analysis. It is extensively used for importing and managing the datasets. It packs in high-performance, easy-to-use data structures and data analysis tools for Python.

Matplotlib – Matplotlib is a Python 2D plotting library that is used to plot any type of charts in Python. It can deliver publication-quality figures in numerous hard copy formats and interactive environments across platforms (IPython shells, Jupyter notebook, web application servers, etc.).

3. Import the dataset

In this step, you need to import the dataset/s that you have gathered for the ML project at hand. Importing the dataset is one of the important steps in data pre-processing in machine learning. However, before you can import the dataset/s, you must set the current directory as the working directory. You can set the working directory in Spyder IDE in three simple steps:

Save your Python file in the directory containing the dataset.

Go to File Explorer option in Spyder IDE and choose the required directory.

Now, click on the F5 button or Run option to execute the file.

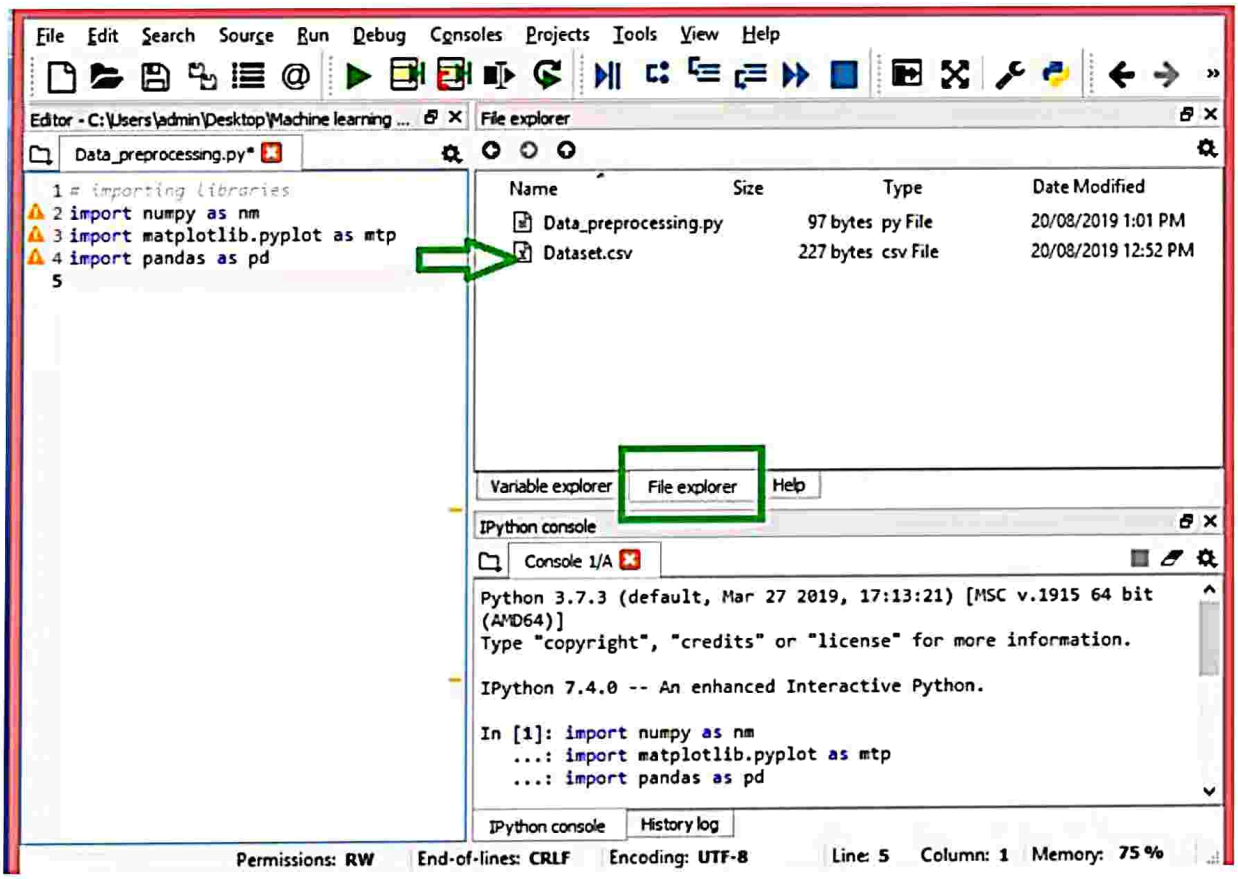
Once you've set the working directory containing the relevant dataset, you can import the dataset using the "read_csv()" function of the Pandas library. This function can read a CSV file (either locally or through a URL) and also perform various operations on it. The read_csv() is written as:

```
dataset= pd.read_csv('Dataset.csv')
```

In this line of code, "data_set" denotes the name of the variable wherein you stored the dataset. The function contains the name of the dataset as well. Once you execute this code, the dataset will be successfully imported.

During the dataset importing process, there's another essential thing you must do – extracting dependent and independent variables. For every Machine Learning model, it is necessary to

separate the independent variables (matrix of features) and dependent variables in a dataset.



To extract the independent variables, you can use “iloc[ ]” function of the Pandas library. This function can extract selected rows and columns from the dataset.

x= data_set.iloc[:,:-1].values

In the line of code above, the first colon(:) considers all the rows and the second colon(:) considers all the columns. The code contains “:-1” since you have to leave out the last column

containing the dependent variable. By executing this code, you will obtain the matrix of features, like this –

[['India' 38.0 68000.0]

['France' 43.0 45000.0]

['Germany' 30.0 54000.0]

['France' 48.0 65000.0]

['Germany' 40.0 nan]

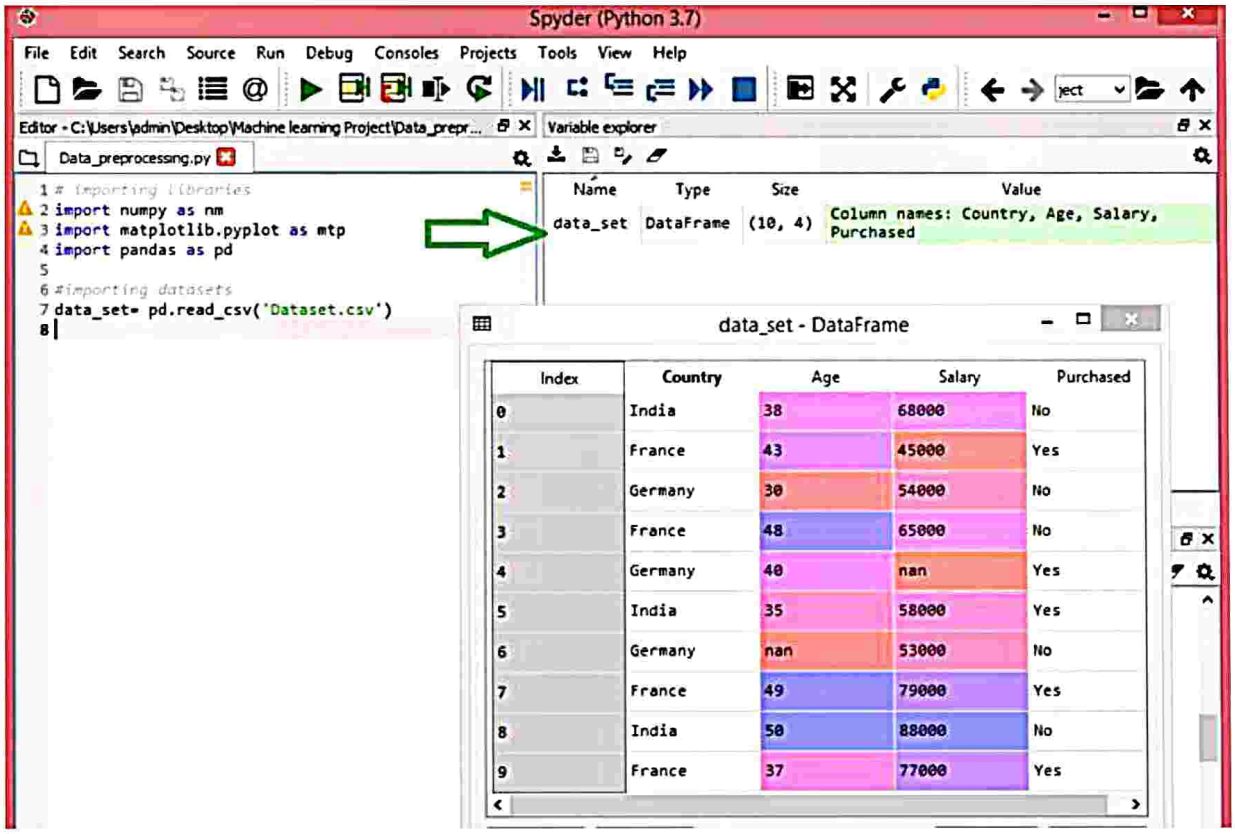
['India' 35.0 58000.0]

['Germany' nan 53000.0]

['France' 49.0 79000.0]

['India' 50.0 88000.0]

['France' 37.0 77000.0]]



You can use the “iloc[ ]” function to extract the dependent variable as well. Here’s how you write it:

Y= data_set.iloc[:,3].values

This line of code considers all the rows with the last column only. By executing the above code, you will get the array of dependent variables, like so –

Array([‘No’, ‘Yes’, ‘No’, ‘No’, ‘Yes’, ‘Yes’, ‘No’, ‘Yes’, ‘No’, ‘Yes’], Dtype=object)

3.Feature Engineering

Feature Engineering helps to derive some valuable features from the

existing ones. These extra features sometimes help in increasing the

performance of the model significantly and certainly help to gain deeper

insights into the data.

Splitted = df['Origin Time'].str.split(' ', n=1, Expand=True)

Df['Date'] = splitted[0]

Df['Time'] = splitted[1].str[:-4]

Df.drop('Origin Time',Axis=1 Inplace=True)

Df.head()

Output:

Now, let's divide the date column into the day, month, and year columns

respectively.

Df['day'] = splitted[2].astype('int')

Df['month'] = splitted[1].astype('int')

Df['year'] = splitted[0].astype('int')

Df.drop('Date', axis=1, Inplace=True)

Df.head()

Exploring data analysis

EDA is an approach to analyzing the data using visual techniques. It is

used to discover trends, and patterns, or to check assumptions with the

help of statistical summaries and graphical representations

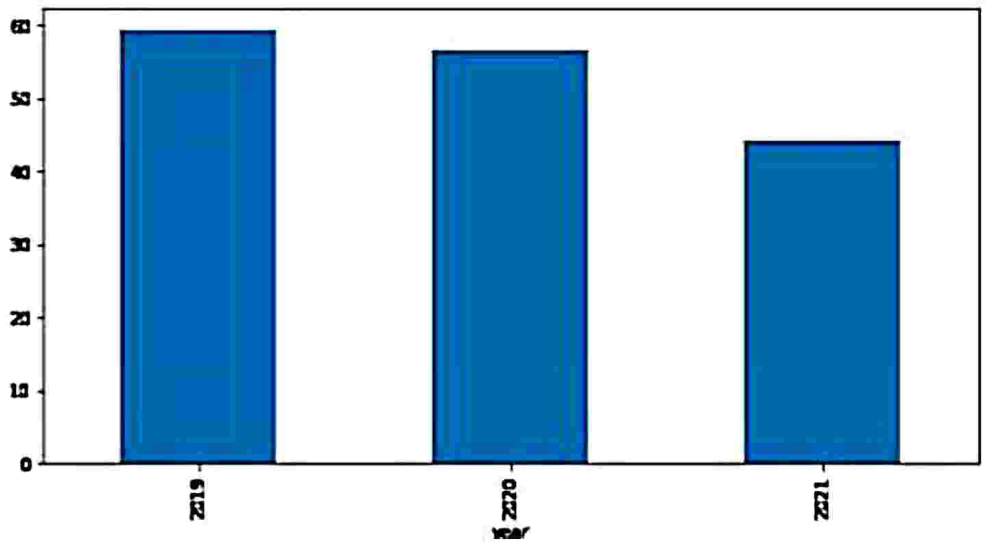
Plt.f Plt.

plt.figure(figsize=(10, 5))

x = df.groupby('year').mean()['Depth']

x.plot.bar()

plt.show()



4.Data splitting

Divide the dataset into training, validation, and test sets. Ensure that the temporal aspect is preserved, as earthquakes can exhibit temporal dependencies.

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%.

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)

(18727, 3) (4682, 3) (18727, 2) (4682, 3)

/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different 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.

```
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
module and should not be imported. It will be removed in a future
NumPy release.

  From numpy.core.umath_tests import inner1d

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]])

Reg.score(X_test, y_test)

Out:

0.8614799631765803
```

```
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)
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 ]])
Best_fit.score(X_test, y_test)
Out:
0.8749008584467053
```

5.Model selection

From the kaggle dataset we have to implement neural network modal

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.
```

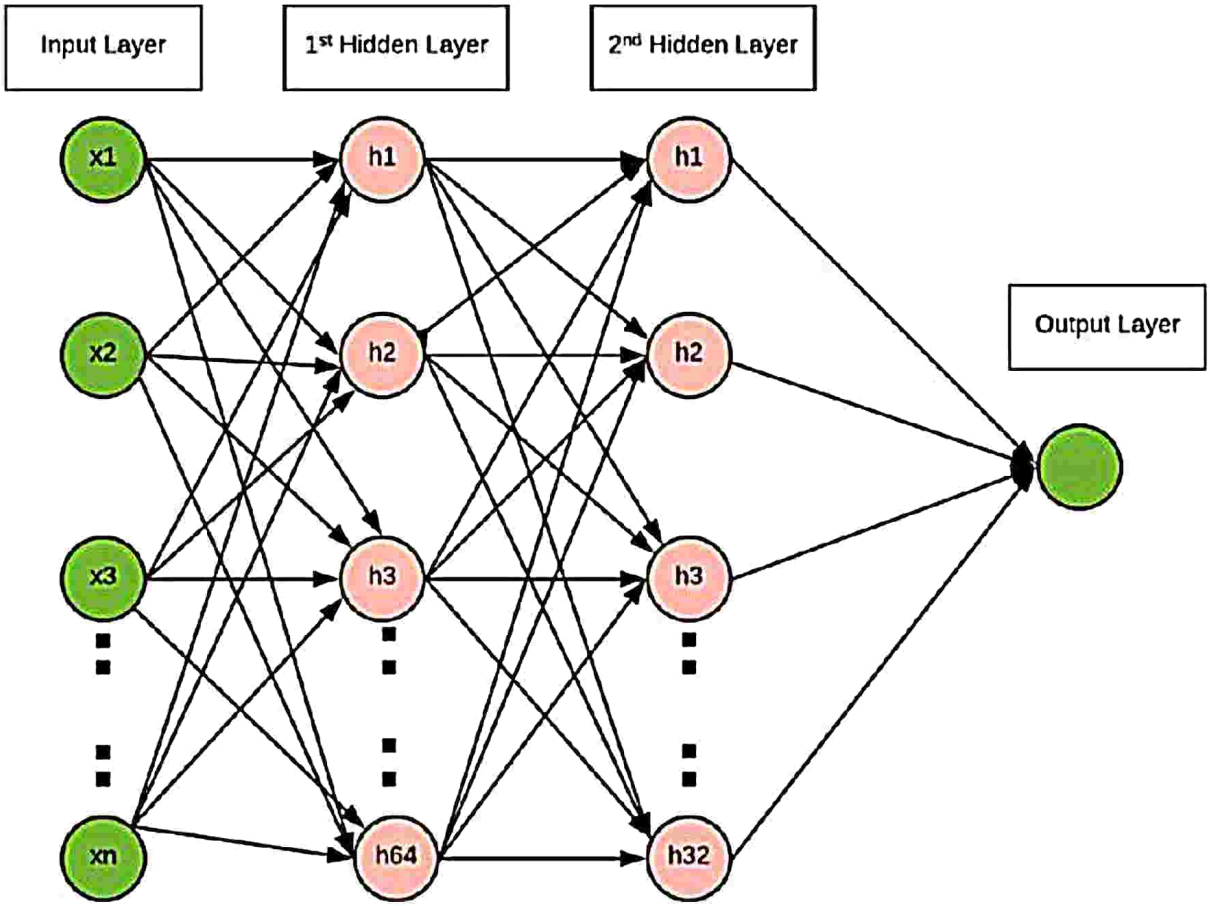
In this, we define the hyperparameters with two or more options to find the best fit.

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', 'expon
ential']
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', '
Nadam']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hinge']

param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs,
activation=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.

Neural Network:



6.Model Training

Train the selected model on the training dataset. Experiment with hyperparameter tuning to optimize model performance.

Step 1: Begin with existing data

Machine learning requires us to have existing data—not the data our application will use when we run it, but data to learn from. You need a lot of real data, in fact, the more the better. The more examples you provide, the better the computer should be able to learn. So just collect every scrap of data you have and dump it and voila! Right?

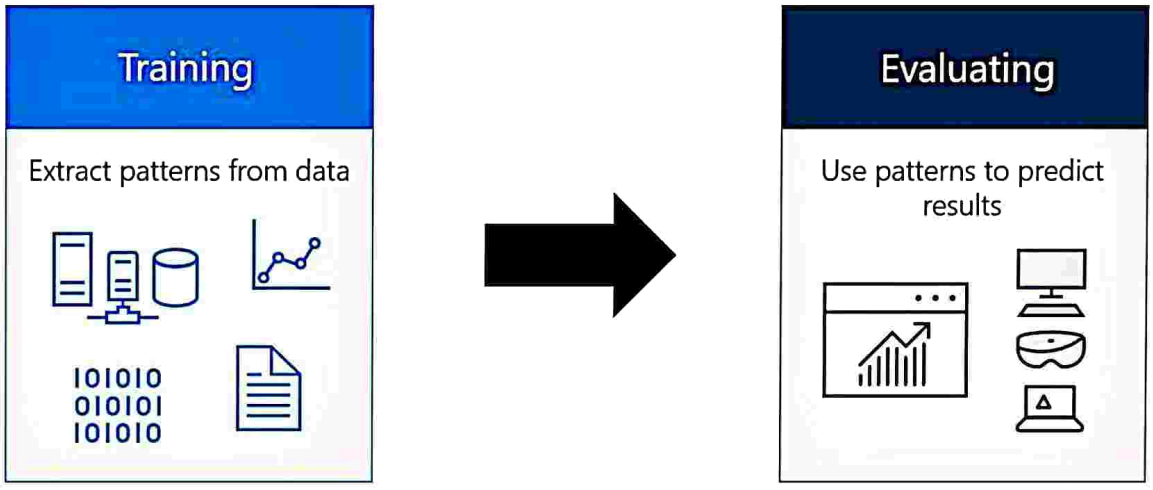
Wrong. In order to train the computer to understand what we want and what we don't want, you need to prepare, clean and label your data. Get rid of garbage entries, missing pieces of information, anything that's ambiguous or confusing. Filter your dataset down to only the information you're interested in right now. Without high quality data, machine learning does not work. So take your time and pay attention to detail.

Step 2: Analyze data to identify patterns

Unlike conventional software development where humans are responsible for interpreting large data sets, with machine learning, you apply a machine learning algorithm to the data. But don't think you're off the hook. Choosing the right algorithm, applying it, configuring it and testing it is where the human element comes back in.

There are several platforms to choose from both commercial and open source. Explore solutions from Microsoft, Google, Amazon, IBM or open source frameworks like TensorFlow, Torch and Caffe. They each have their own strengths and downsides, and each will interpret the same dataset a different way. Some are faster to train. Some are more configurable. Some allow for more visibility into the decision process. In order to make the right choice, you need to experiment with a few algorithms and test until you find the one that gives you the results most aligned to what you're trying to achieve with your data.

When it's all said and done, and you've successfully applied a machine learning algorithm to analyze your data and learn from it, you have a trained model.



Step 3: Make predictions

There is so much you can do with your newly trained model. You could import it into a software application you're building, deploy it into a web back end or upload and host it into a cloud service. Your trained model is now ready to take in new data and feed you predictions, aka results.

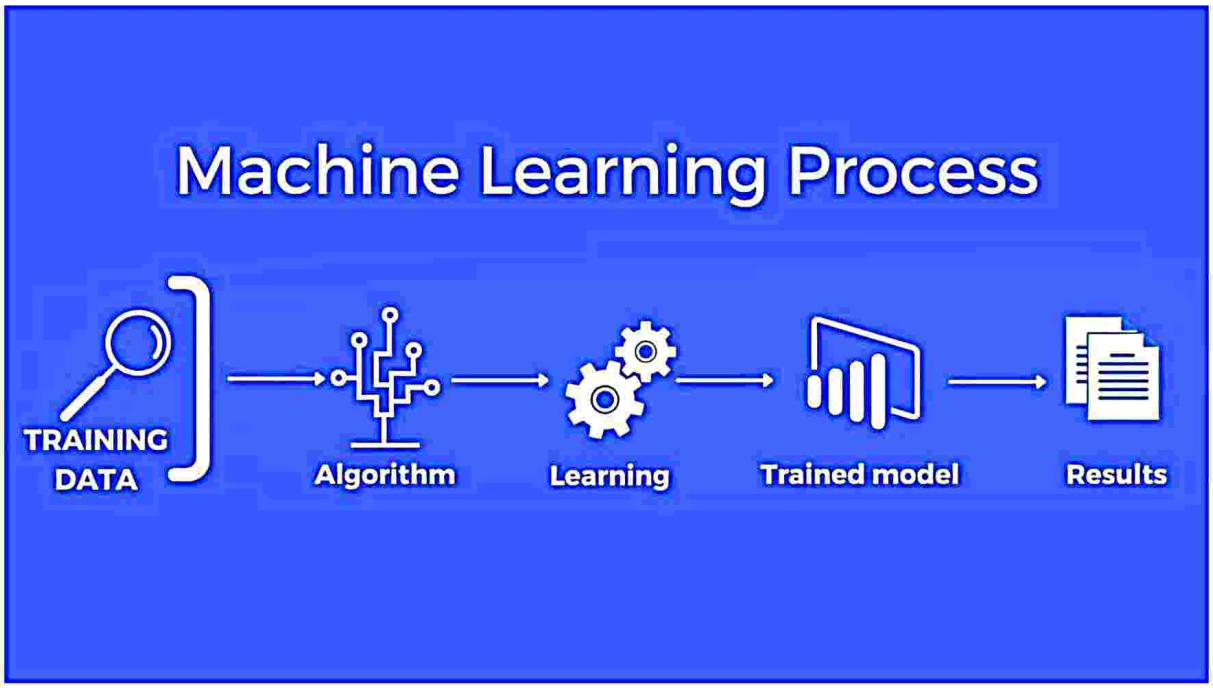
These results can look different depending on what kind of algorithm you go with. If you need to know what something is, go with a classification algorithm, which comes in two types. Binary classification categorizes data between two categories. Multi-class classification sorts data between—you guessed it—multiple categories.

When the result you're looking for is an actual number, you'll want to use a regression algorithm. Regression takes a lot of different data with different weights of importance and analyzes it with historical data to objectively provide an end result.

Both regression and classification are supervised types of algorithms, meaning you need to provide intentional data and direction for the computer to learn. There is also unsupervised algorithms which don't require labeled data or any guidance on the kind of result you're looking for.

One form of unsupervised algorithms is clustering. You use clustering when you want to understand the structure of your data. You provide a set of data and let the algorithm identify the categories within that set. On the other hand, anomaly is an unsupervised algorithm you can use when your data looks normal and uniform, and you want the algorithm to pull anything out of the ordinary that doesn't fit with the rest of the data.

Although supervised algorithms are more common, it's good to play around with each algorithm type and use case to better understand probability and practice splitting and training data in different ways. The more you toy with your data, the better your understanding of what machine learning can accomplish will become.



Data: The dataset you want to use must be well-structured, accurate. The data you use can be labeled or unlabeled. Unlabeled data are sample items — e.g. photos, videos, news articles — that don't need additional explanation, while labeled ones are augmented: unlabeled information is bundled and an explanation, with a tag, is added to them.

Algorithm: there are different types of algorithms that can be used (e.g. linear regression, logistic regression). Choosing the right algorithm is both a combination of business need, specification, experimentation and time available.

Model: A "model" is the output of a machine learning algorithm run on data. It represents the rules, numbers, and any other algorithm-specific data structures required to make predictions

Model Development Part 2

1.EVALUATION

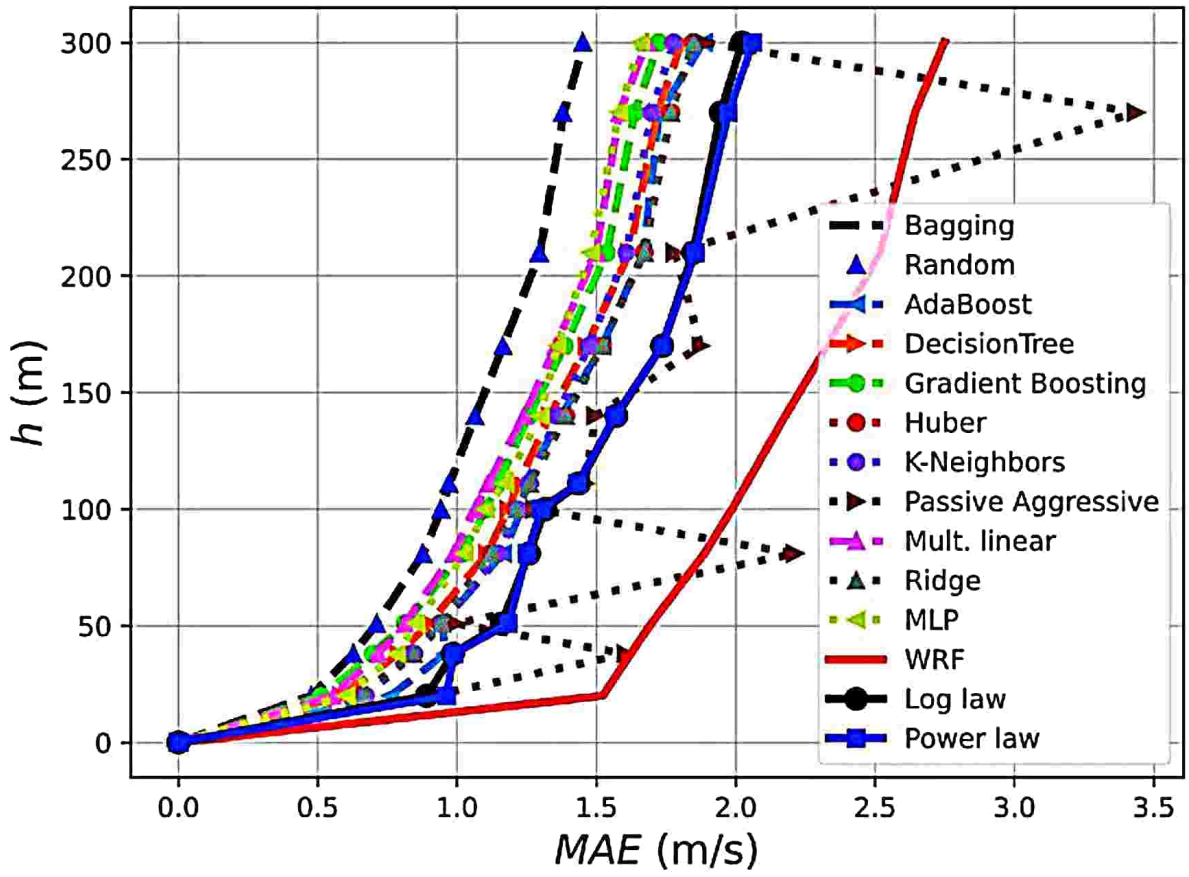
Assess the model's performance on the validation set. Common evaluation metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).

Model evaluation is the process that uses some metrics which help us to analyze the performance of the model. As we all know that model development is a multi-step process and a check should be kept on how well the model generalizes future predictions. Therefore evaluating a model plays a vital role so that we can judge the performance of our model. The evaluation also helps to analyze a model's key weaknesses. There are many metrics like Accuracy, Precision, Recall, F1 score, Area under Curve, Confusion Matrix, and Mean Square Error. Cross Validation is one technique that is followed during the training phase and it is a model evaluation technique as well.

Mean Absolute Error(MAE)

This is the simplest metric used to analyze the loss over the whole dataset. As we all know the error is basically the difference between the predicted and actual values. Therefore MAE is defined as the average of the errors calculated. Here we calculate the modulus of the error, perform the summation and then divide the result by the number of data points. It is a positive quantity and is not concerned about the direction. The formula of MAE is given by

$$MSE = \sum (y_{pred} - y_{actual})^2 / N$$



Mean Squared Error(MSE)

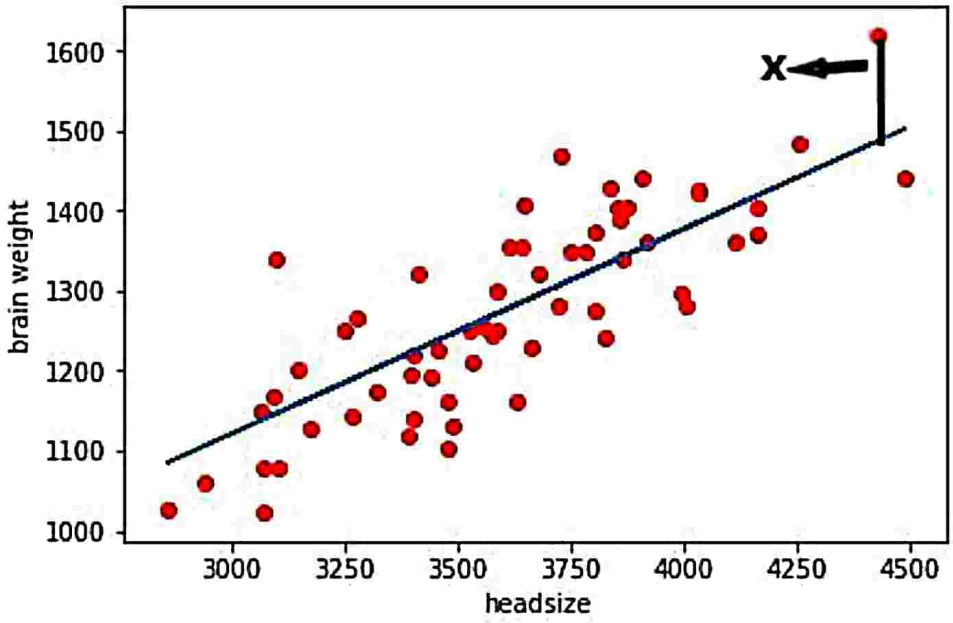
The most commonly used metric is Mean Square error or MSE. It is a function used to calculate the loss. We find the difference between the predicted values and the truth variable, square the result and then find the average over the whole dataset. MSE is always positive as we square the values. The small the MSE better is the performance of our model. The formula of MSE is given:

$$MSE = \sum (y_{pred} - y_{actual})^2 / N$$

Root Mean Squared Error(RMSE)

RMSE is a popular method and is the extended version of MSE(Mean Squared Error). This method is basically used to evaluate the performance of our model. It indicates how much the data points are spread around the best line. It is the standard deviation of the Mean squared error. A lower value means that the data point lies closer to the best fit line.

$$RMSE = \sqrt{(\sum (y_{pred} - y_{actual})^2 / N)}$$



2.HYPERPARAMETER TUNING

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters. However, there is another kind of parameter, known as ***Hyperparameters***, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyperparameters include:

1. The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization
2. The learning rate for training a neural network.
3. The C and sigma hyperparameters for support vector machines.
4. The k in k-nearest neighbors.

The aim of this article is to explore various strategies to tune hyperparameters for Machine learning models.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem.

GridSearchCV
In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.

As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of ***C=0.3 and Alpha=0.2***, the performance score comes out to be **0.726(Highest)**, therefore it is selected.

C	Alpha = 0.1	0.2	0.3	0.4
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

Program:

```
From sklearn.linear_model import LogisticRegression

From sklearn.model_selection import GridSearchCV

C_space = np.logspace(-5, 8, 15)

Param_grid = {‘C’: c_space}

Logreg = LogisticRegression()

Logreg_cv = GridSearchCV(logreg, param_grid, cv = 5)
```

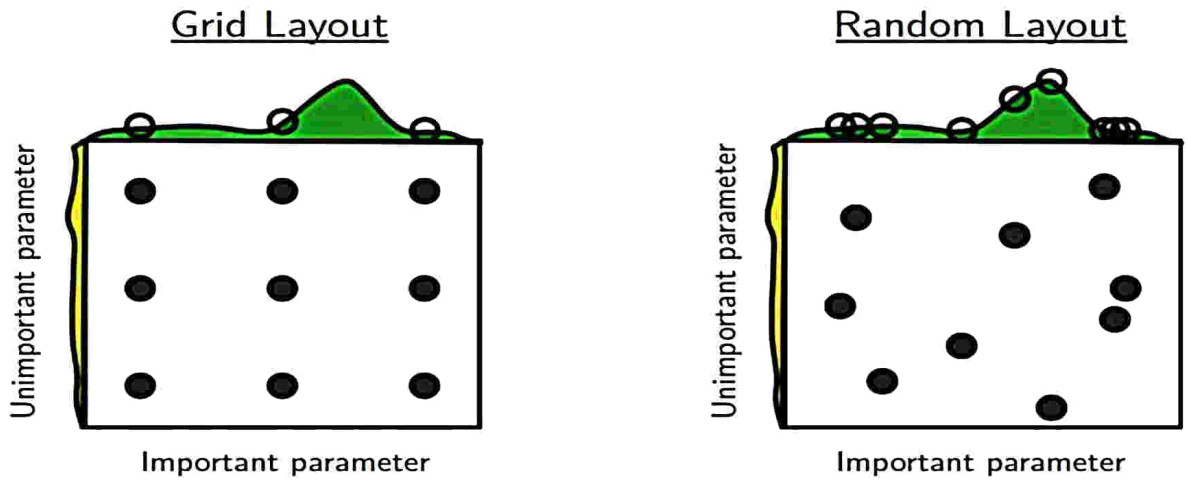
```
Logreg_cv.fit(X, y)

Print("Tuned Logistic Regression Parameters:
{}".format(logreg_cv.best_params_))

Print("Best score is {}".format(logreg_cv.best_score_))
```

RandomizedSearchCV

RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

Sample program:

```
>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.model_selection import
RandomizedSearchCV
>>> from scipy.stats import uniform
>>> iris = load_iris()
>>> logistic = LogisticRegression(solver='saga', tol=1e-2,
max_iter=200,random_state=0)
>>> distributions = dict(C=uniform(loc=0, scale=4),
penalty=['l2', 'l1'])
>>> clf = RandomizedSearchCV(logistic, distributions,
random_state=0)
>>> search = clf.fit(iris.data, iris.target)
>>> search.best_params_
{'C': 2..., 'penalty': 'l1'}
```



3.MODEL VALIDATION

Validation and testing begins with splitting your training dataset. The **"Valid-Test split"** is a technique to evaluate the performance of your **ML model**. You need to split the data because you don't want your model to over-learn from training data, to not perform well. But, most of all, you want to evaluate **how well your model is generalizing**.

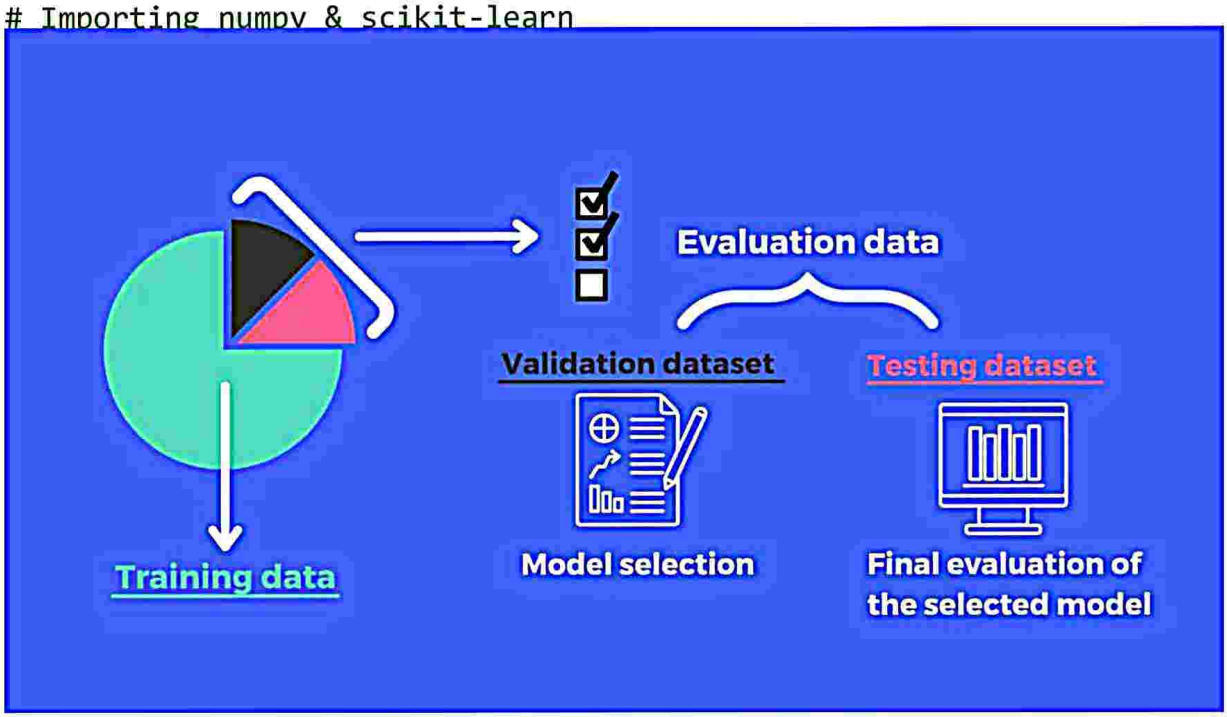
Hence, you held back from training dataset, **validation and testing** subsets for assessing your model in a meaningful way. Notice that a **typical split ratio** of data, between training, validation and testing sets is around . A brief explanation of the role of each of these dataset is below.

- **Validation dataset**: it is useful when it comes to **model selection**. The data included in this set will be used to find the optimal values for the parameters of the model under consideration. When you work with ML models, you typically need to **test multiple models** with different

parameters values for finding the optimal values that will give the best possible performance. Therefore, in order to **pick the best model** you must evaluate each of them.

- **Testing dataset**: when you have tuned the model by performing parameters optimisation, you should end up with the **final model**. The testing set is used to provide an **unbiased evaluation** of the performance of this model and ensure that it can generalise well to new, unseen data.

```
# Importing numpy & scikit-learn
```



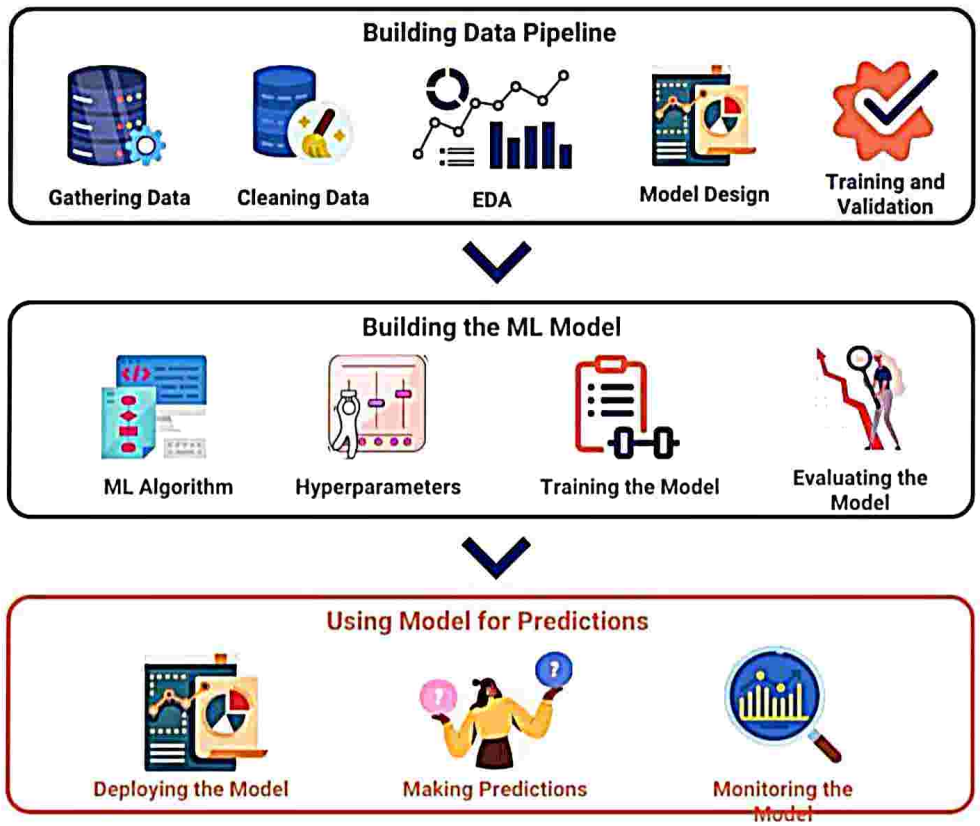
4.DEPLOYMENT

Machine learning model deployment is the process of placing a finished machine learning model into a live environment where it can be used for its intended purpose. Data science

models can be deployed in a wide range of environments, and they are often integrated with apps through an API so they can be accessed by end users.

While model deployment is the third stage of the data science lifecycle (manage, develop, deploy and monitor), every aspect of a model's creation is performed with deployment in mind.

Models are usually developed in an environment with carefully prepared data sets, where they are trained and tested. Most models created during the development stage do not meet desired objectives. Few models pass their test and those that do represent a sizable investment of resources. So moving a model into a dynamic environment can require a great deal of planning and preparation for the project to be successful.



5.MONITORING

Monitoring earthquake prediction is a critical task that involves the continuous collection and analysis of various data sources to detect potential seismic events. Here are some key aspects of monitoring earthquake prediction:

Seismic Activity Monitoring: Continuous monitoring of seismic activity using a network of seismometers and

accelerometers. These instruments detect ground motion and record seismic events, which are then analyzed to assess the likelihood of an earthquake.

Data Processing and Analysis: Collected seismic data is processed and analyzed to identify patterns and anomalies. Advanced algorithms and machine learning models can help detect precursors or changes in seismic activity that may indicate an impending earthquake.

Early Warning Systems: Some regions implement early warning systems that can provide a few seconds to minutes of advance notice before strong shaking from an earthquake reaches a specific location. These systems rely on real-time seismic data and communication infrastructure to alert the public and authorities.

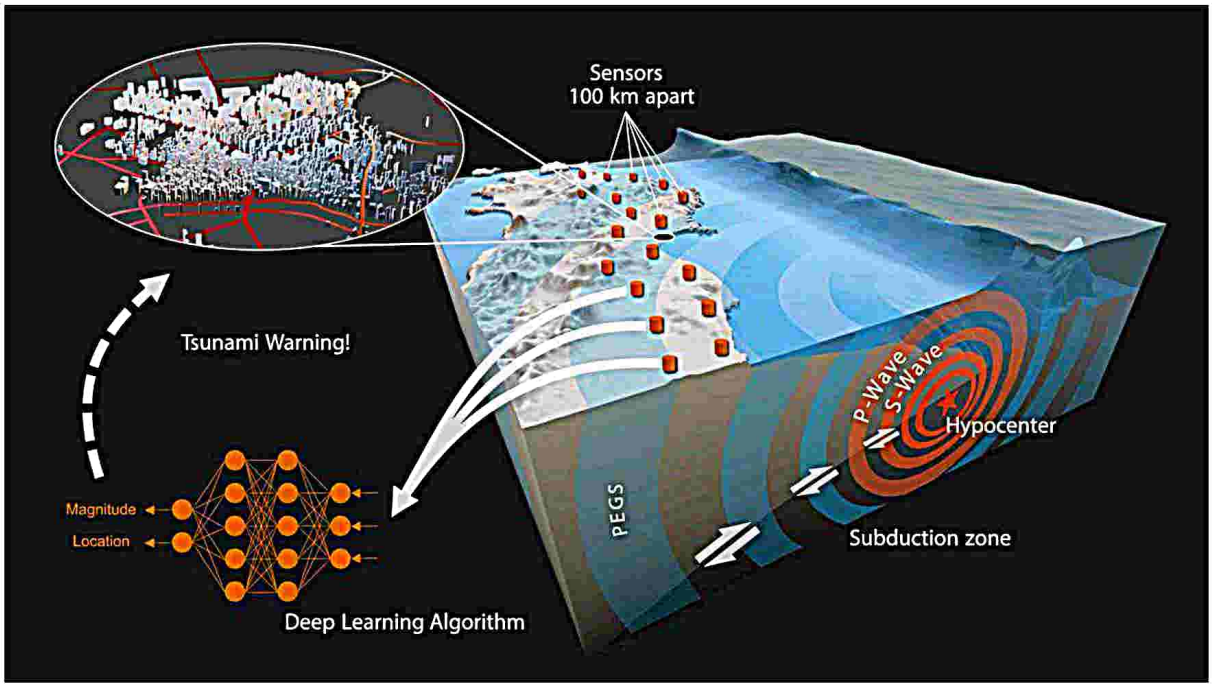
GPS and Ground Deformation Monitoring: Monitoring the displacement of the Earth's crust through GPS and ground deformation sensors can help identify tectonic stress accumulation that may lead to an earthquake.

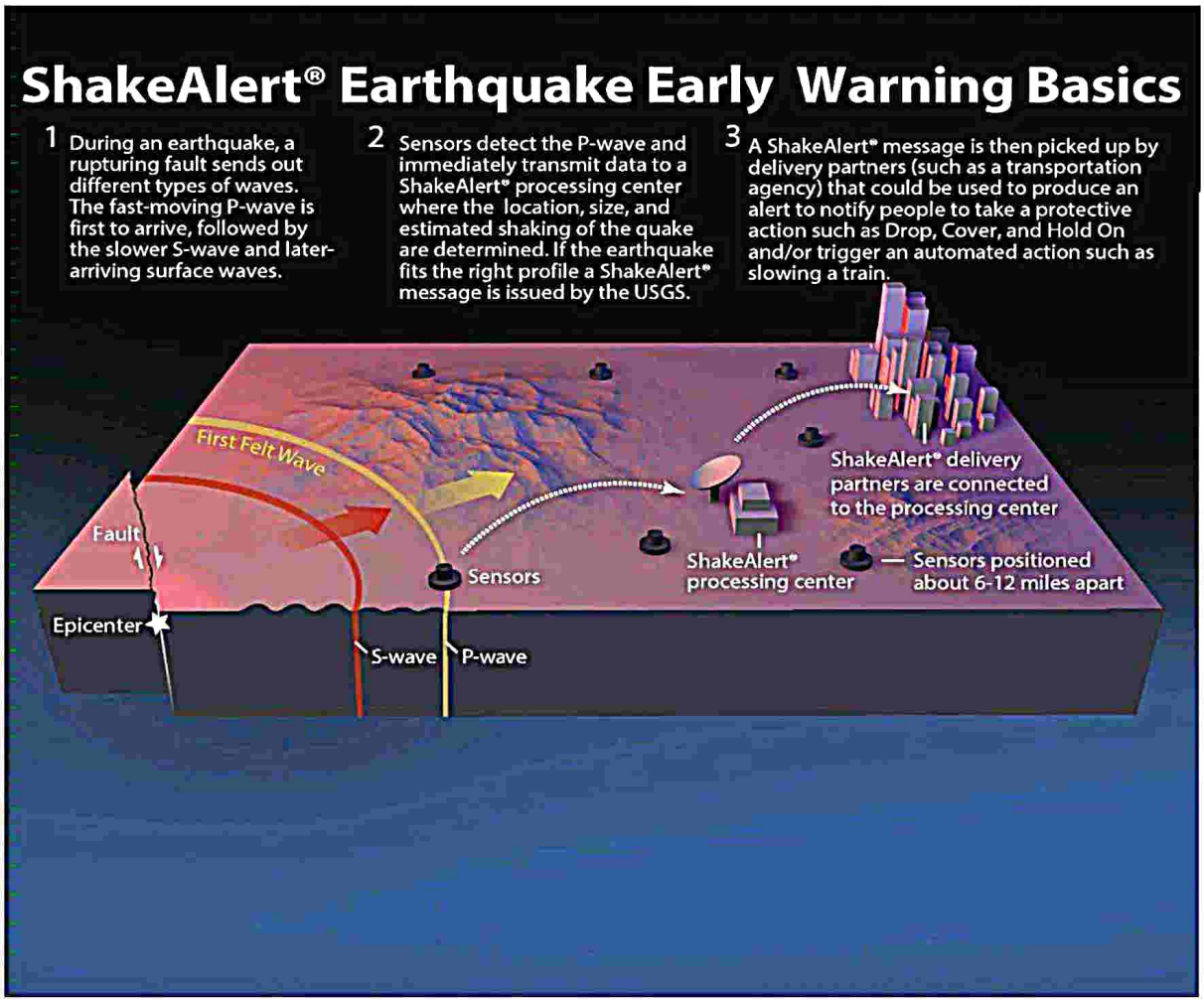
Monitoring Radon Gas and Other Precursors: Some researchers investigate unconventional precursors such as changes in radon gas emissions, groundwater levels, and animal behavior in an attempt to predict earthquakes

Satellite Imagery: Remote sensing techniques using satellites can be employed to monitor surface changes, fault movements, and other geological features that might be associated with seismic activity.

Global Seismic Networks: International cooperation through global seismic networks allows the sharing of seismic data and information about earthquakes in different regions, contributing to a better understanding of seismic patterns.

Public Awareness and Education: In earthquake-prone regions, public awareness and education programs are essential to inform and prepare the population for potential seismic events.





6.MAINTENANCE

Maintaining earthquake prediction and monitoring systems is essential to ensure their continued effectiveness in providing early warnings and risk assessments. Here are key aspects of maintaining earthquake prediction systems

Instrument Maintenance: Regularly calibrate and maintain seismometers, accelerometers, GPS devices, and other monitoring instruments. This ensures that data collection remains accurate and reliable.

Data Quality Assurance: Implement data quality control procedures to identify and address issues such as sensor malfunctions or data gaps promptly.

Network Reliability: Maintain a robust and redundant communication network for transmitting seismic data in real time. Redundancy is crucial to ensure data continuity during network failures.

Sensor Deployment and Upgrades: Periodically assess the location and density of monitoring sensors. If necessary, deploy additional sensors or upgrade existing ones to improve coverage and sensitivity.

Maintenance Scheduling: ML and DO Work Best Together