# Project title - Earthquake Prediction Model using Python

## Phase 4 – Development Part-2

Colab Notebook link: Click here

## Data Visualization:

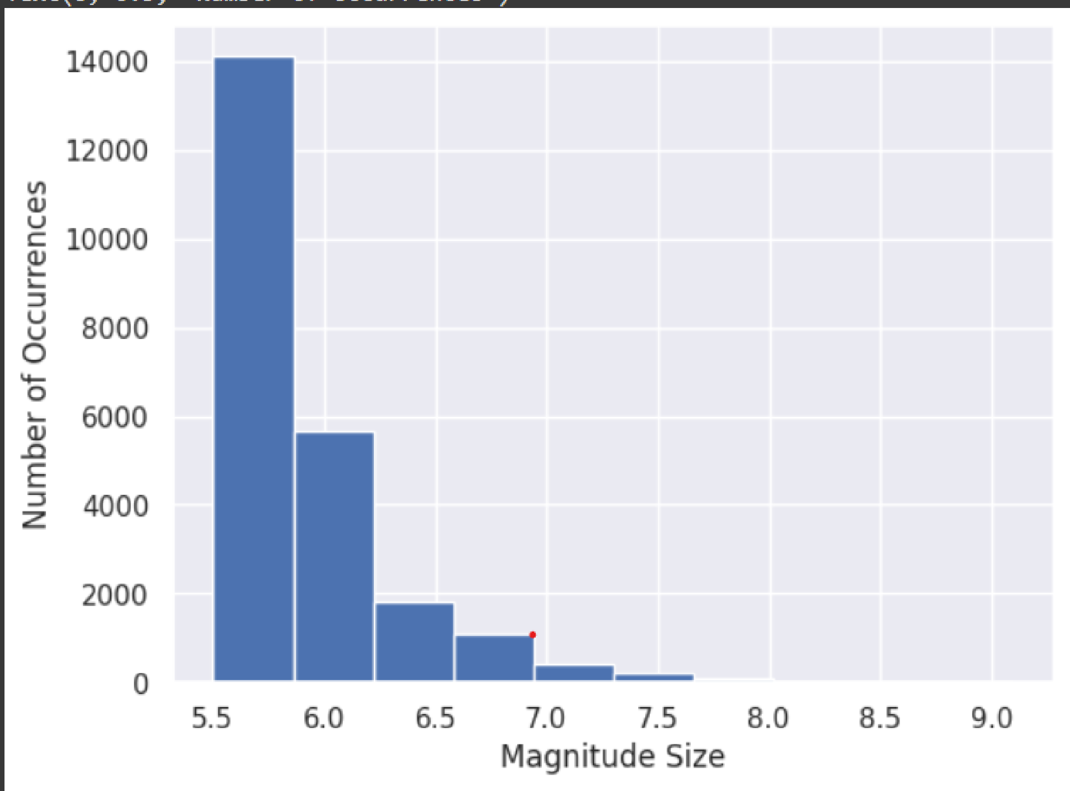
- As a first step, the necessary packages for data visualization are imported.

```
[126] import matplotlib.pyplot as plt
      from mpl_toolkits.basemap import Basemap
      import seaborn as sns
      sns.set(style="darkgrid")
```

- A histogram of "Magnitude Size" vs "Number of Earthquake Occurrences" was made.

```
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')
```

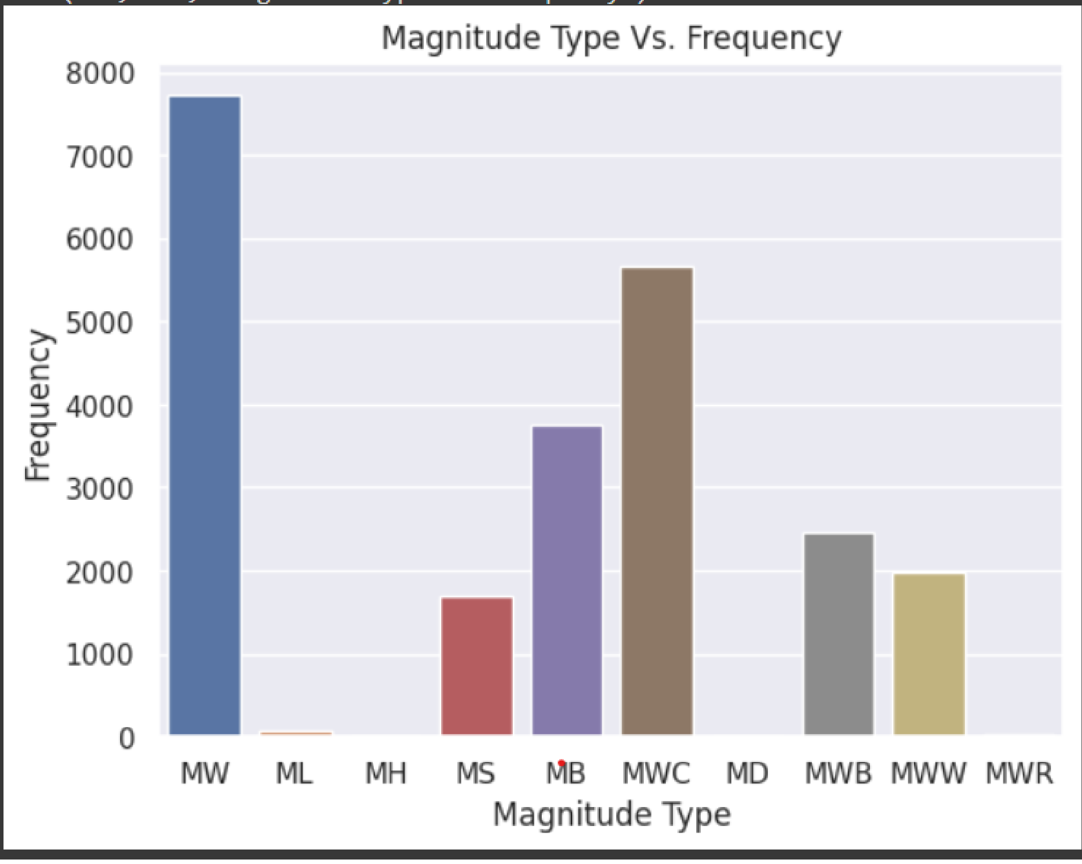
```
Text(0, 0.5, 'Number of Occurrences')
```



➢ A countplot was drawn against “Magnitude Type” and “Frequency”.

```
[132] sns.countplot(x="Magnitude Type",data = data)
      plt.ylabel("Frequency")
      plt.title("Magnitude Type Vs. Frequency")
```

```
Text(0.5, 1.0, 'Magnitude Type Vs. Frequency')
```

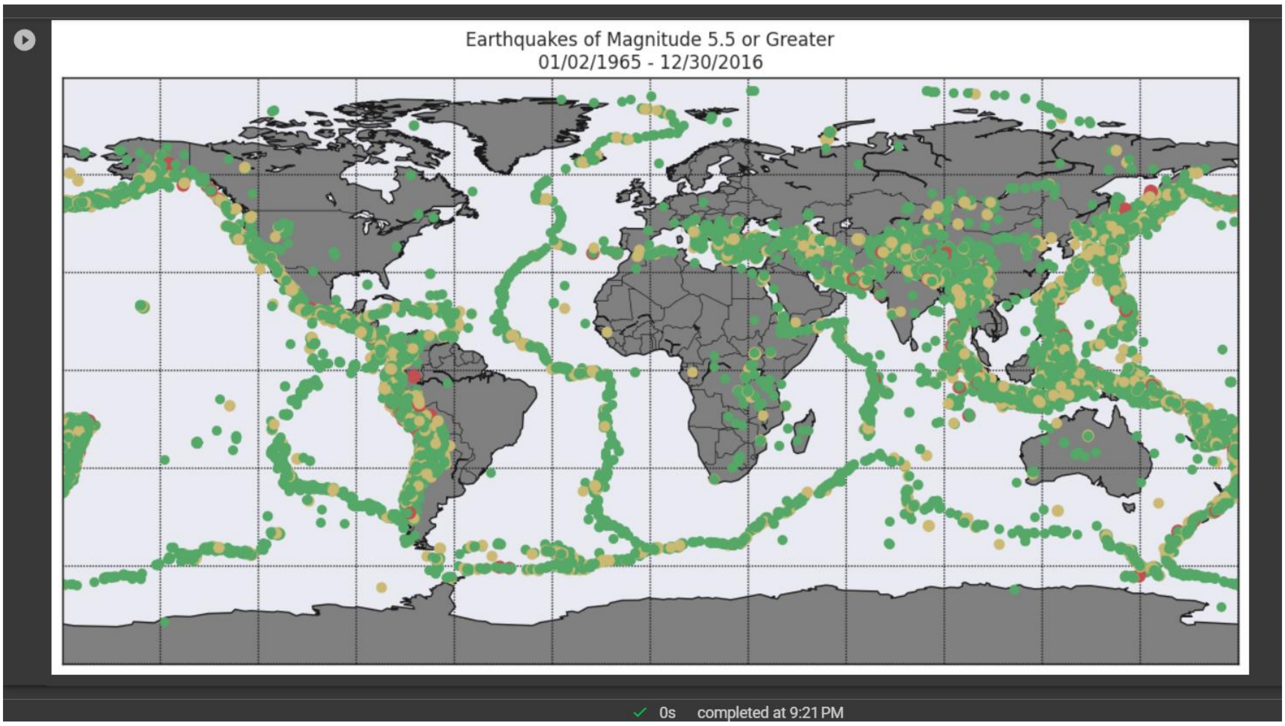



## Visualization on World Map:

- ➢ The Basemap toolkit of the matplotlib module is used for visualizing the impact caused by the earthquake all over the globe in terms of their magnitude.
- ➢ The projection called “cyl” – “Cylindrical Equidistant” is used.
- ➢ Green markers are used to indicate earthquakes with minimal magnitude.
- ➢ Yellow markers indicate average earthquakes.
- ➢ Red markers indicate the most malignant form of earthquake.

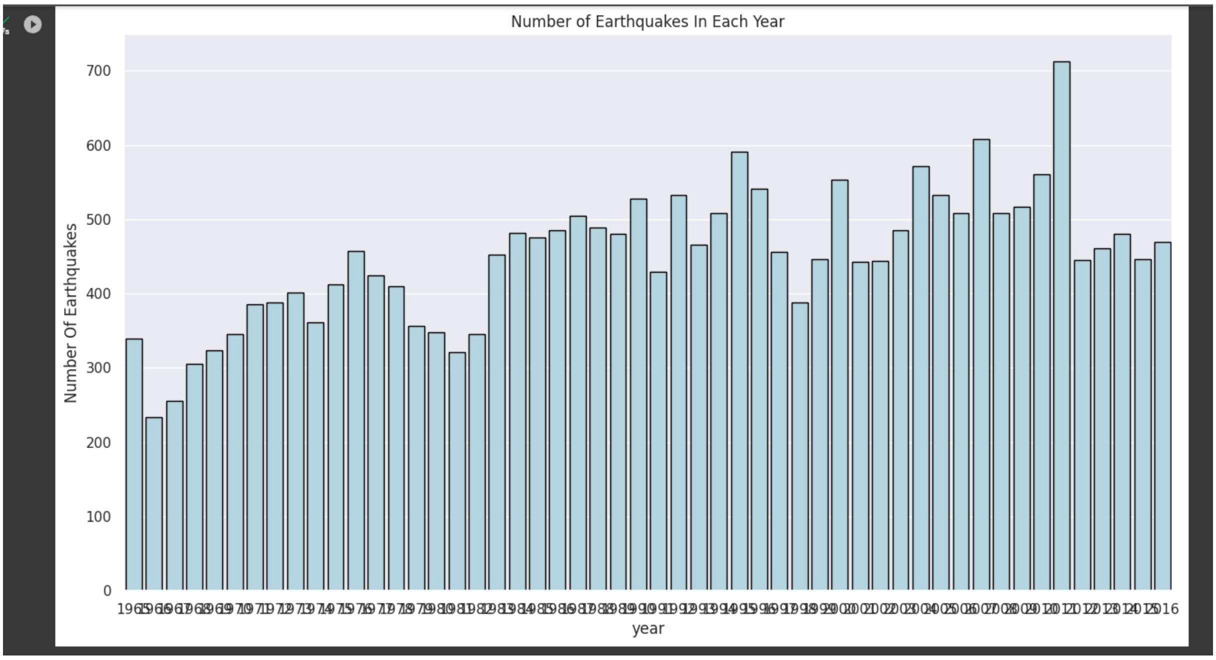
```
def get_marker_color(magnitude):
    if magnitude < 6.2:
        return ('go')
    elif magnitude < 7.5:
        return ('yo')
    else:
        return ('ro')

plt.figure(figsize=(14,10),edgecolor='w')
```

```python
plt.figure(figsize=(14,10),edgecolor='w')

map = Basemap(projection='cyl', llcrnrlat=-90, urcrnrlat=90,
            llcrnrlon=-180, urcrnrlon=180,)
map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color = 'gray')
map.drawmapboundary()
map.drawmeridians(np.arange(0, 360, 30))
map.drawparallels(np.arange(-90, 90, 30))

# Reading the longitude, latitude and magnitude values from the dataset
lons = data['Longitude'].values
lats = data['Latitude'].values
magnitudes = data['Magnitude'].values
timestrings = data['Date'].tolist()

min_marker_size = 0.5
for lon, lat, mag in zip(lons, lats, magnitudes):
    x,y = map(lon, lat)
    msize = mag
    marker_string = get_marker_color(mag)
    map.plot(x, y, marker_string, markersize=msize)

title_string = "Earthquakes of Magnitude 5.5 or Greater\n"
title_string += "%s - %s" % (timestrings[0][:10], timestrings[-1][:10])
plt.title(title_string)

plt.show()
```



## Countplot showing number of earthquake occurences during each year since 1960s:

```python
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['year'] = data['date'].apply(lambda x: str(x).split('-')[0])
plt.figure(figsize=(15, 8))
sns.set(font_scale=1.0)
sns.countplot(x="year", data=data,color="lightblue",edgecolor="black")
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each Year')
```




## Model Building:

- In have build two models for predicting the occurrence of earthquake using the pre-trained datas.
- One is built using “Logistic Regression” and the other is done using “Neural Networks”

## Logistic Regression Model:

- The training and testing datasets are split in the following manner.

```python
[135] import sklearn
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
x = df[['Latitude', 'Longitude', 'Timestamp']]
y = df[['Magnitude']]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
print(x_train.shape,x_test.shape)

(17421, 3) (5808, 3)
```

## Evaluating the Logistic Regression Model:

- Accuracy_score is used as an evaluation metrics.
- This model showed an accuracy of about 93%

```
[136] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
      log=LogisticRegression()
      model=log.fit(x_train,y_train)
      y_pred=log.predict(x_test)
      print("Accuracy is:",(metrics.accuracy_score(y_test,y_pred))*100)

      Accuracy is: 93.01190988664084
```

## Neural Network Model:

- The hyperparameter tuning for selecting the optimized parameters for our neural network model is done using the GridSearchCV methodology.

Neural Network Model

```
import sklearn
from sklearn.model_selection import train_test_split, GridSearchCV

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
print(x_train.shape, x_test.shape, y_train.shape, x_test.shape)
from keras.models import Sequential
from keras.layers import Dense

# 3 dense layers, 16, 16, 2 nodes each

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
from keras.wrappers.scikit_learn import KerasClassifier

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

param_grid = {
    "neurons": [16, 64],
    "batch_size": [10, 20],
    "epochs": [10],
    "activation": ['sigmoid', 'relu'],
    "optimizer": ['SGD', 'Adadelta'],
    "loss": ['squared_hinge']
}

(16260, 3) (6969, 3) (16260, 1) (6969, 3)
```

```
[138] x_train = np.asarray(x_train).astype(np.float32)
      y_train = np.asarray(y_train).astype(np.float32)
      x_test = np.asarray(x_test).astype(np.float32)
      y_test = np.asarray(y_test).astype(np.float32)
```

## GridSearchCV for finding best fits:

GridSearchCV is used for finding the best parameters for tuning the model's performance

```
[139] grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
      grid_result = grid.fit(x_train, y_train)

      best_params = grid_result.best_params_
      best_params

      {'activation': 'relu',
       'batch_size': 10,
       'epochs': 10,
       'loss': 'squared_hinge',
       'neurons': 16,
       'optimizer': 'SGD'}
```

## Training the neural network model using *best_params* from the GridSearchCV and Model Evaluation:

- The neural network almost encountered zero loss and the accuracy is nearly 99%

```
[140] model = Sequential()
      model.add(Dense(16, activation=best_params['activation'], input_shape=(3,)))
      model.add(Dense(16, activation=best_params['activation']))
      model.add(Dense(2, activation='softmax'))

      model.compile(optimizer=best_params['optimizer'], loss=best_params['loss'], metrics=['accuracy'])
      model.fit(x_train, y_train, batch_size=best_params['batch_size'], epochs=best_params['epochs'], verbose=1, validation_data=(x_test, y_test))

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

```
Epoch 1/10
1626/1626 [==============================] - 16s 9ms/step - loss: nan - accuracy: 0.9900 - val_loss: nan - val_accuracy: 0.9918
Epoch 2/10
1626/1626 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 3/10
1626/1626 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 4/10
1626/1626 [==============================] - 8s 5ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 5/10
1626/1626 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 6/10
1626/1626 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 7/10
1626/1626 [==============================] - 6s 4ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 8/10
1626/1626 [==============================] - 6s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 9/10
1626/1626 [==============================] - 4s 2ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 10/10
1626/1626 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
218/218 [==============================] - 1s 2ms/step - loss: nan - accuracy: 0.9918
Evaluation result on Test Data : Loss = nan, accuracy = 0.9918209314346313
```