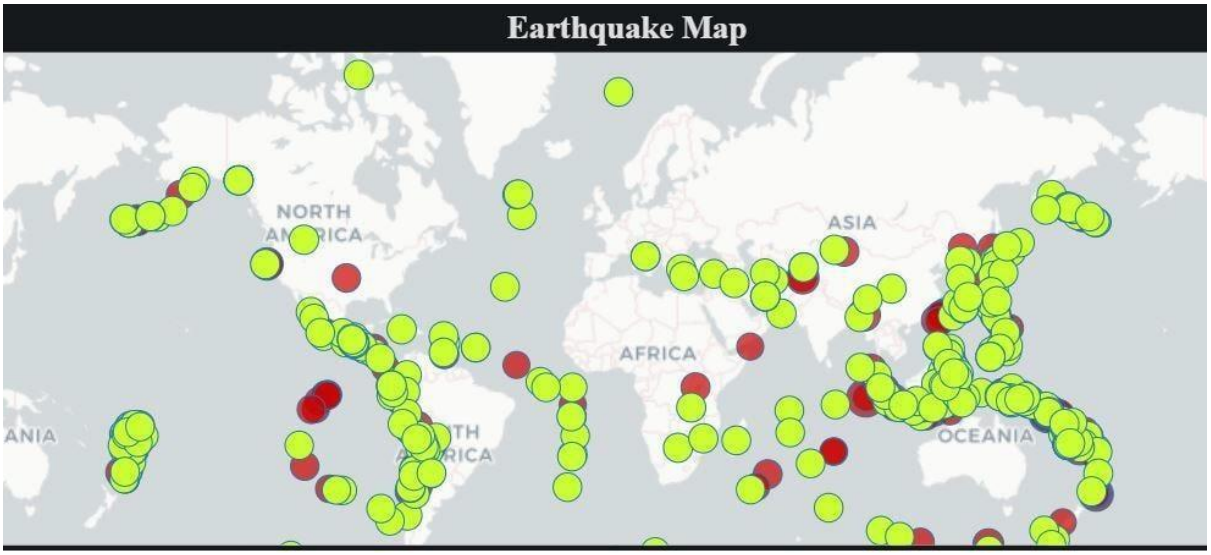
EARTHQUAKE PREDICTION MODEL USING PYTHON

Batch Member:
911721106009:GUHAN RAJ R

Phase 5 Submission Document
Project Title: Earth Quake Prediction

Dataset Link: https://www.kaggle.com/datasets/usgs/earthquake-database



Earthquake Prediction Using Python

- Earthquake prediction is a challenging and crucial field of study that aims to anticipate the occurrence of seismic events, such as earthquakes and tremors, in specific geographic regions. Earthquakes, with their potential for widespread destruction and loss of life, have been a subject of concern for both scientists and society for many years. The ability to predict earthquakes could provide early warning systems and contribute to improved disaster preparedness and response.

- However, it's important to note that earthquake prediction is an incredibly complex and evolving field. While there have been significant advancements in understanding the processes leading to earthquakes, predicting the precise time, location, and magnitude of future seismic events remains an ongoing scientific challenge. The reasons for this complexity include:

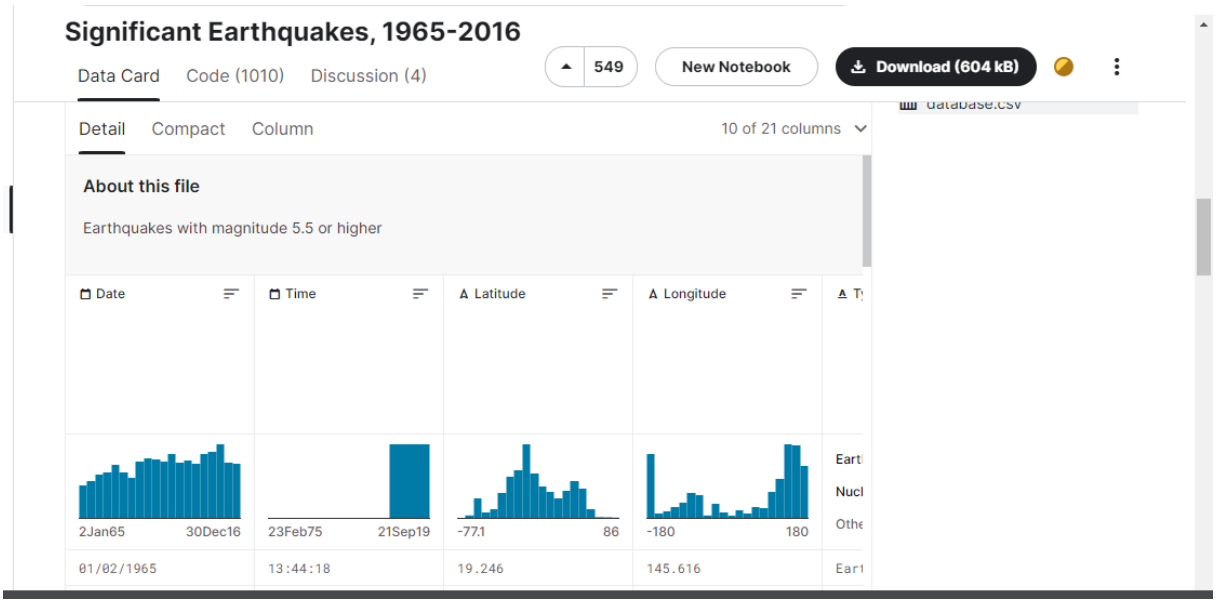
- Earthquakes are the result of complex interactions between tectonic plates, geological structures, and a range of other physical factors. Predicting the specific conditions that lead to a seismic event is a multidisciplinary endeavor.

- The temporal and spatial patterns of earthquakes are highly variable. Predicting when and where an earthquake will occur with precision is complicated due to these variations.

- The historical data on earthquakes is limited, and there is often insufficient information to draw definitive conclusions about future events.

- Ongoing research and advancements in technology are essential to improving prediction capabilities. Scientists continually develop and refine models and methodologies to enhance earthquake prediction.

DATASET



Building an earthquake prediction model requires access to a dataset that includes historical earthquake data and related features.

ANALYSE AND VISUALIZE EARTHQUAKE DATA IN PYTHON

Earthquake is a natural phenomenon whose occurrence predictability is still a hot topic in academia. This is because of the destructive power it holds. In this article, we'll learn how to analyze and visualize earthquake data with Python and Matplotlib.

Importing Libraries and Dataset

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

Pandas – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.

Matplotlib/Seaborn – This library is used to draw visualizations.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import warnings
warnings
```

filterwa Now, let's load the dataset into the panda's data frame for easy analysis

```
df = pd.read_csv('dataset.csv')
df.head()
```

OUTPUT:

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

The dataset we are using here contains data for the following columns:

- Origin time of the Earthquake
- Latitude and the longitude of the location.
- Depth – This means how much depth below the earth's level the earthquake started.
- The magnitude of the earthquake.

From the above description of the dataset, we can conclude that:

The maximum magnitude of the Earthquake is 7.

The maximum depth at which the earthquake started is 471 km below the ground.

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:

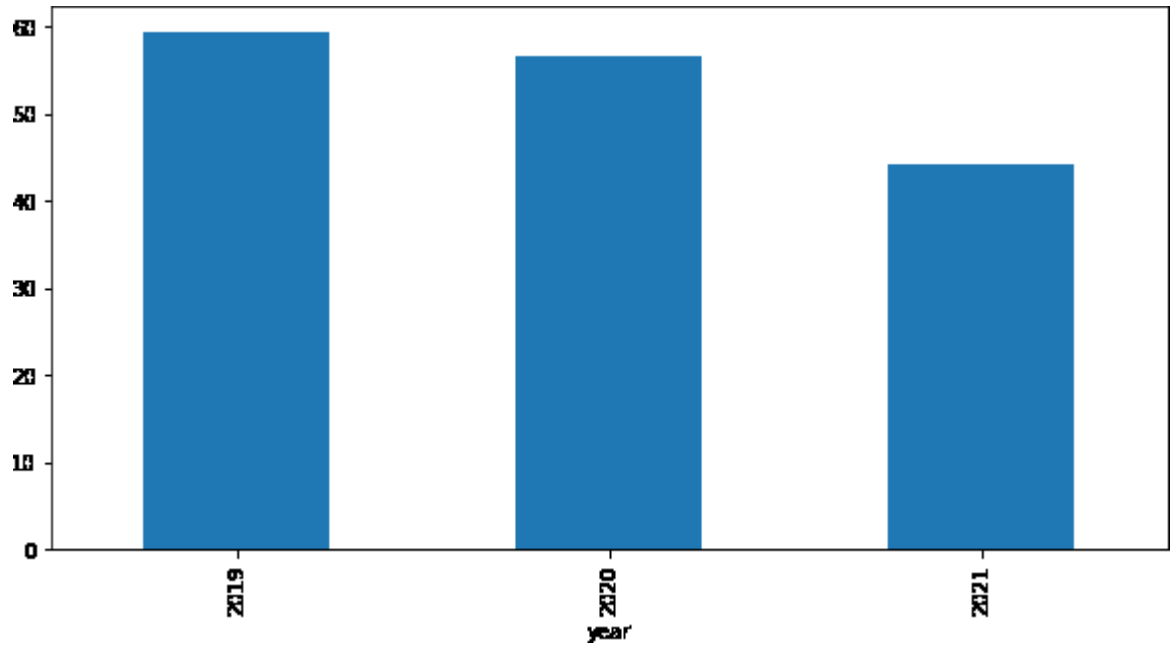
	Latitude	Longitude	Depth	Magnitude	Location	Date	Time
0	29.06	77.42	5.0	2.5	53km NNE of New Delhi, India	2021-07-31	09:43:23
1	19.93	72.92	5.0	2.4	91km W of Nashik, Maharashtra, India	2021-07-30	23:04:57
2	31.50	74.37	33.0	3.4	49km WSW of Amritsar, Punjab, India	2021-07-30	21:31:10
3	28.34	76.23	5.0	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

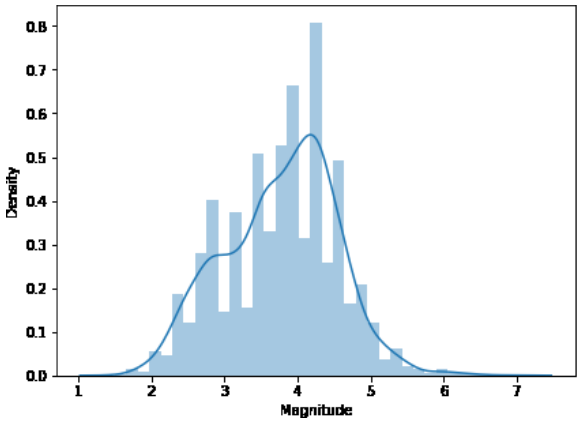
Exploratory Data Analysis

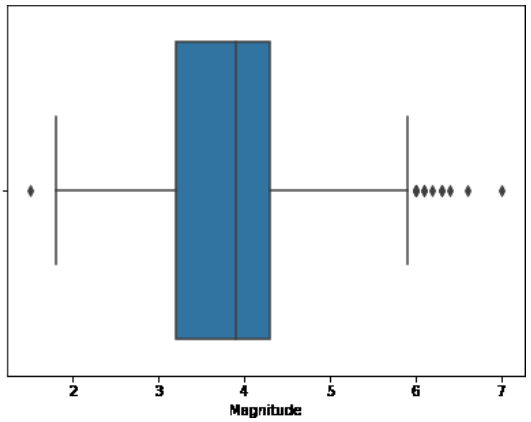
EDA is an approach to analyzing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations.

```
plt.figure(figsize=(10, 5))
x = df.groupby('year').mean()['Depth']
x.plot.bar()
plt.show()
```

OUTPUT





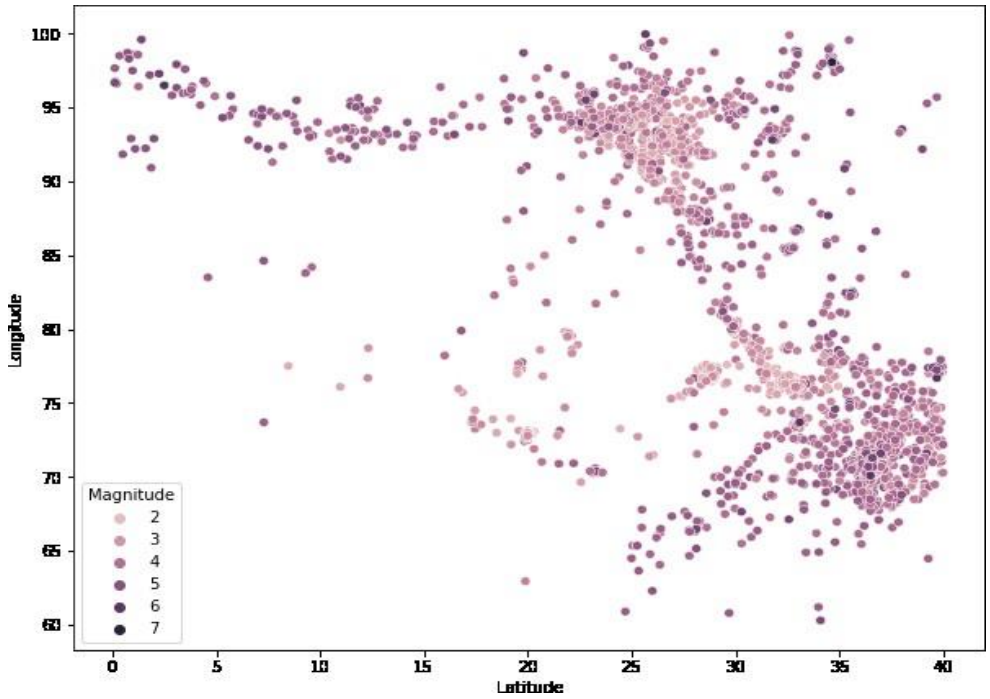



```
plt.subplots(figsize=(15, 5))

plt.subplot(1, 2, 1)
sb.distplot(df['Magnitude'])

plt.subplot(1, 2, 2)
sb.boxplot(df['Magnitude'])
```

```
plt.show()
```

```
plt.figure(figsize=(10, 8))
sb.scatterplot(data=df,
               x='Latitude',
               y='Longitude',
               hue='Magnitude')
plt.show()
```

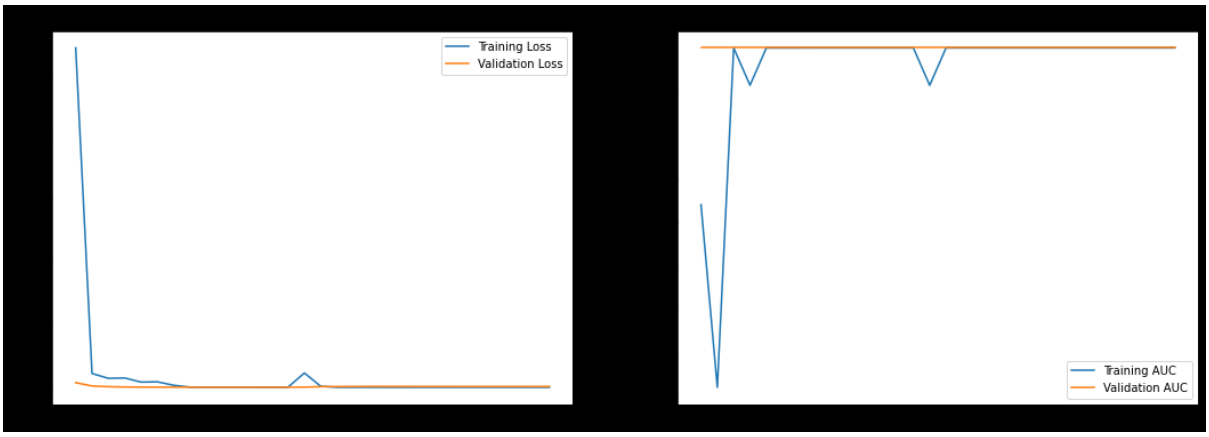
OUTPUT



RESULT

```
linkcode
plt.figure(figsize=(18, 6))
epochs_range = range(epochs)
train_loss, val_loss = history.history['loss'], history.history['val_loss']
train_auc, val_auc = history.history['auc'], history.history['val_auc']
plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")
plt.legend()
plt.title("Loss Over Time")
```

```
plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_auc, label="Training AUC")
plt.plot(epochs_range, val_auc, label="Validation AUC")
plt.legend()
plt.title("AUC Over Time")
plt.show()
```



Load the Dataset

- Use Python and data manipulation libraries like pandas to load the dataset into a DataFrame. Here's an example code snippet:

```
import pandas as pd

# Load the dataset (replace 'your_data.csv' with your dataset file)
df = pd.read_csv('your_data.csv')
```

Data Preprocessing

- Data preprocessing is crucial to prepare the dataset for machine learning. Common preprocessing steps include:

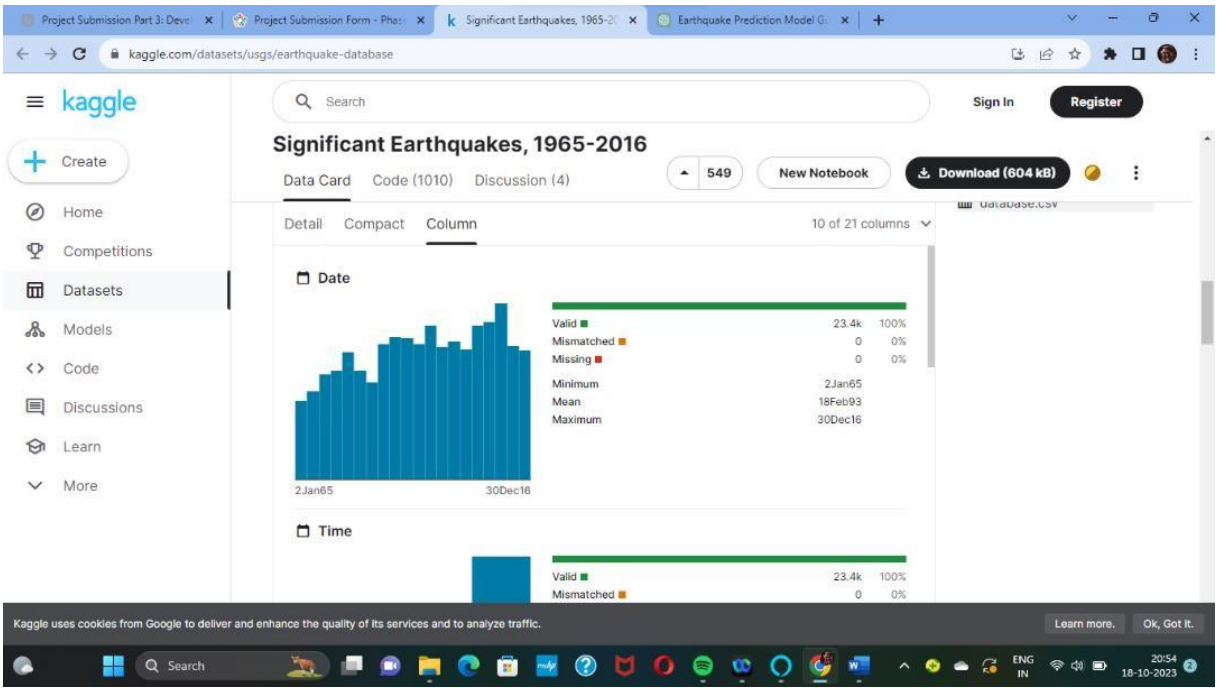
- Handling missing values: Decide how to handle missing data (e.g., impute, remove, or use advanced techniques).

- Feature selection: Choose relevant features for your prediction model.

- Feature engineering: Create new features or transform existing ones to better represent the data.

- Data normalization or standardization: Scale the features if needed.

- Handling categorical data: Encode categorical variables into numerical values (e.g., one-hot encoding)



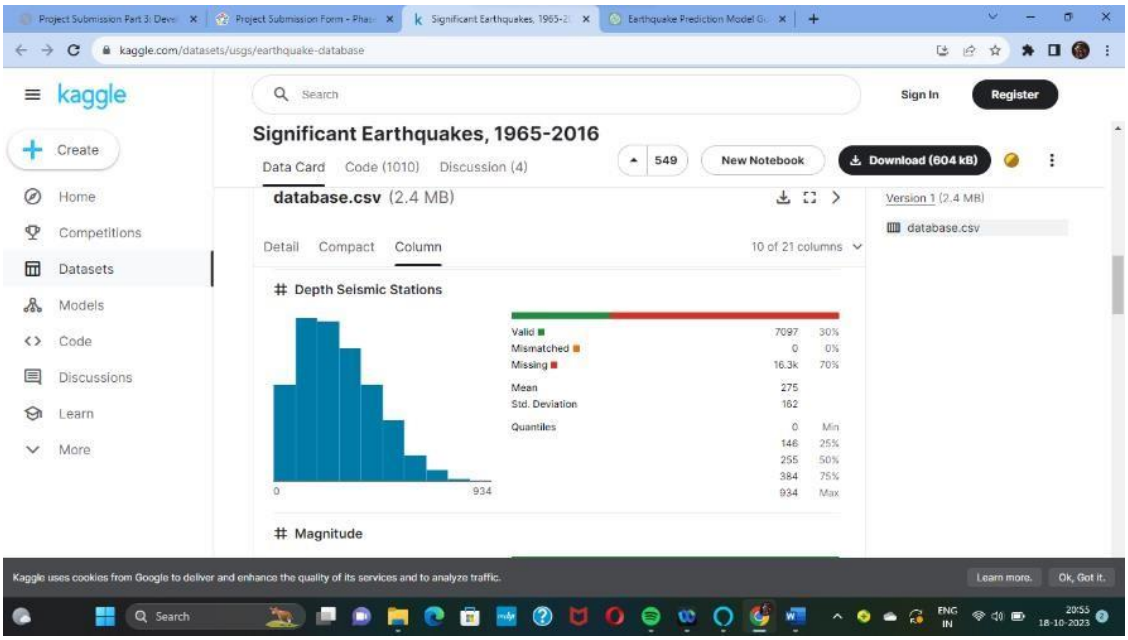
Exploratory Data Analysis (EDA)

- Analyze the dataset to gain insights into the data. You can create visualizations and summary statistics to better understand the distribution of data and relationships between variables.

Split Data into Training and Testing Sets

- To evaluate the performance of your model, split the dataset into training and testing sets. The typical split is 70-80% for training and 20-30% for testing.

```python
from sklearn.model_selection import train_test_split

X = df.drop(columns=['target_column']) # Features
y = df['target_column']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```




```

## Feature Scaling

- Depending on the choice of algorithms (e.g., neural networks, support vector machines), you may need to scale or normalize your data.

## Build and Train Your Model

- Choose a machine learning or deep learning model suitable for your problem (e.g., Random Forest, SVM, Neural Network).

- Train the model on the training data.
```

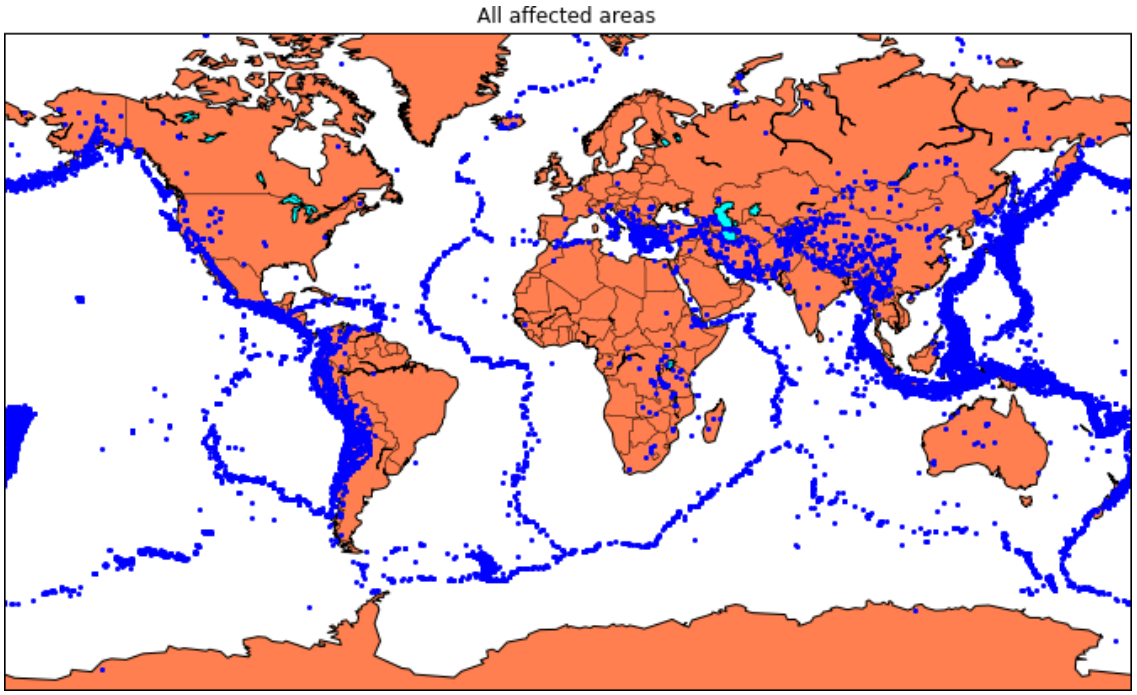
Visualization

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

In [8]:

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,ur
crnrlon=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()
```



- Recurrent Neural Networks (RNNs) for sequential data.

- Transformers for natural language processing tasks.

6. Data Preparation for Deep Learning:

- Preprocess data according to the requirements of your chosen deep learning architecture. This may include tokenization, sequence padding, or image resizing.

7. Implementing Deep Learning Models:

- Implement and train deep learning models using frameworks like TensorFlow or PyTorch. Experiment with different model architectures and hyperparameters.

8. Ensemble with Deep Learning:

- Combine the predictions of your ensemble models and deep learning models. You can use techniques like:

- Voting: Combining predictions by majority vote.

- Weighted Averaging: Assigning different weights to model predictions.

- Stacking: Using an additional meta-learner to learn how to combine the predictions effectively.

9. Hyperparameter Tuning:

- Fine-tune the hyperparameters of both ensemble and deep learning models using techniques like grid search or Bayesian optimization.

10. Evaluation and Metrics:

- Evaluate the performance of your models using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, or mean squared error, depending on the nature of your problem.

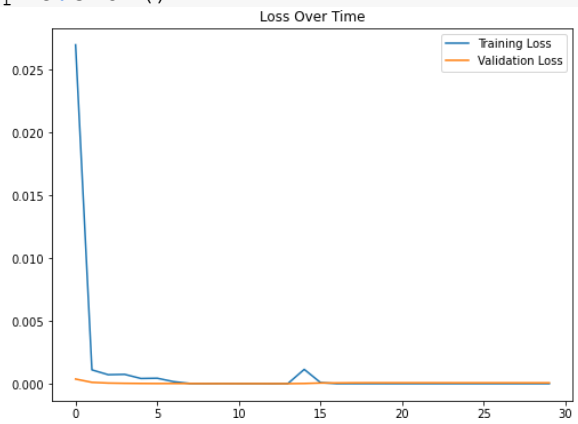
```
    #Add the response to the context.
    context.append({'role':'assistant', 'content':f"{response}"})

    #Undate the panels to shjow the conversation.
    panels.append(
        pn.Row('User:', pn.pane.Markdown(prompt, width=600)))
    panels.append(
        pn.Row('Assistant:', pn.pane.Markdown(response, width=600,
style={'background-color': '#F6F6F6'})))

    return pn.Column(*panels)

pn.extension()

panels = []

client_prompt = pn.widgets.TextInput(value="Hi", placeholder='Enter text here…')
button_conversation = pn.widgets.Button(name="talk")

interactive_conversation = pn.bind(add_prompts_conversation, button_conversation)

dashboard = pn.Column(
    client_prompt,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=300),
)

dashboard
```

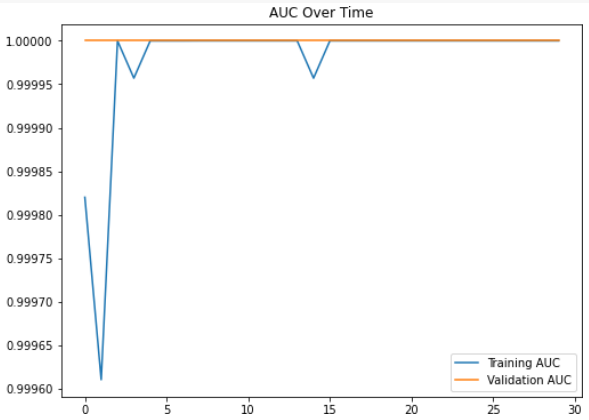
Results

In [40]:

linkcode

```
plt.figure(figsize=(18, 6))

epochs_range = range(epochs)
train_loss, val_loss = history.history['loss'], history.history['val_loss']
train_auc, val_auc = history.history['auc'], history.history['val_auc']

plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")
plt.legend()
plt.title("Loss Over Time")
plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_auc, label="Training AUC")
plt.plot(epochs_range, val_auc, label="Validation AUC")
plt.legend()
plt.title("AUC Over Time")

plt.show()
```



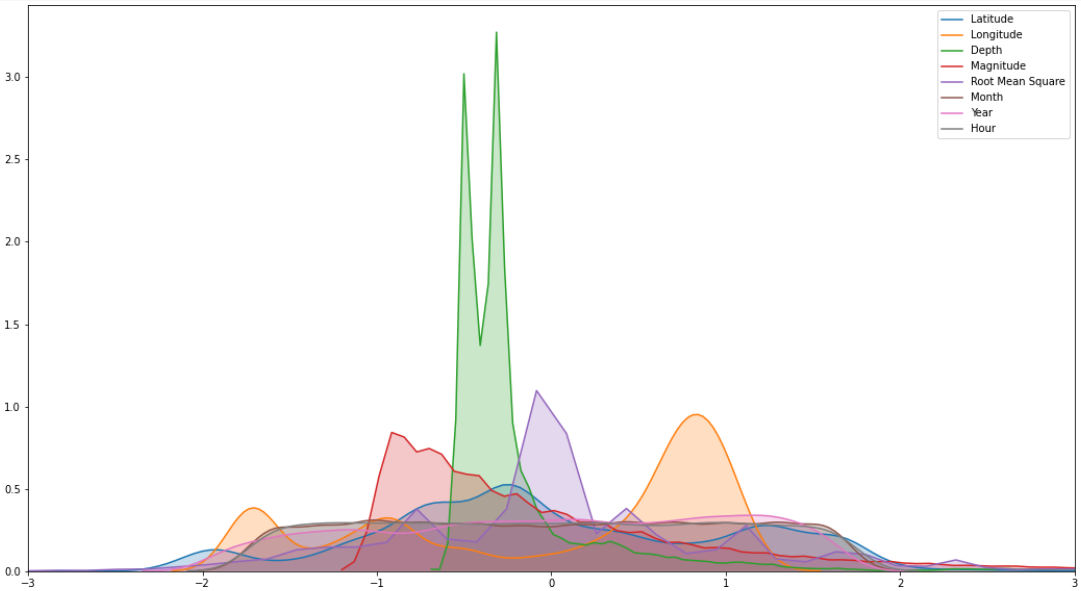


PROGRAM :

```
messages=[
        {"role": "system", "content": "You are an OrderBot in a fastfood
restaurant."},
        {"role": "user", "content": "I have only 10 dollars, what can I order?"},
        {"role": "assistant", "content": "We have the fast menu for 7 dollars."},
        {"role": "user", "content": "Perfect! Give me one! "}


    ]
import openai

#creamos el contexto
context =[
{'role':'system', 'content':"""Actua como el camarero de un restaurante de comida
rapida \
    pregunta al cliente que desea y ofrecele las cosas del menu. \
    En el menu hay: \
    Bocadillo fuet 6
    Bocadillo jamon 7
    Agua 2
    """}]

#Le pasamos el contexto a OpenAI y recogemos su respuesta.
mensajes = context
respuesta = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=mensajes)

#enseñamos la respuesta al usuario y pedimos una entrada nueva.
print(respuesta.choices[0].message["content"])

#añadimos la respuesta al pool de mensajes
mensajes.append(respuesta)

#añadimos una segunda linea del usuario.
mensajes.append({'role':'user', 'content':'un agua por favor'})

#Volvemos a llamar al modelo con las dos lineas añadidas.
respuesta = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=mensajes)

import openai
import panel as pn

#obtener la key
from mykeys import openai_api_key
openai.api_key=openai_api_key

def add_prompts_conversation(_):
    #Get the value introduced by the user
    prompt = client_prompt.value_input
    client_prompt.value = ''

    #Append to the context the User prompt.
    context.append({'role':'user', 'content':f"{prompt}"})

    #Get the response.
    response = continue_conversation(context)
```

Modeling and Training

```
inputs = tf.keras.Input(shape=(104,))
x = tf.keras.layers.Dense(64, activation='relu')(inputs)
x = tf.keras.layers.Dense(64, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, outputs)
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[tf.keras.metrics.AUC(name='auc')]
)


batch_size = 32
epochs = 30

history = model.fit(
    X_train,
    y_train,
 validation_split=0.2,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=[tf.keras.callbacks.ReduceLROnPlateau()],
    verbose=0
```



Splitting the Data

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

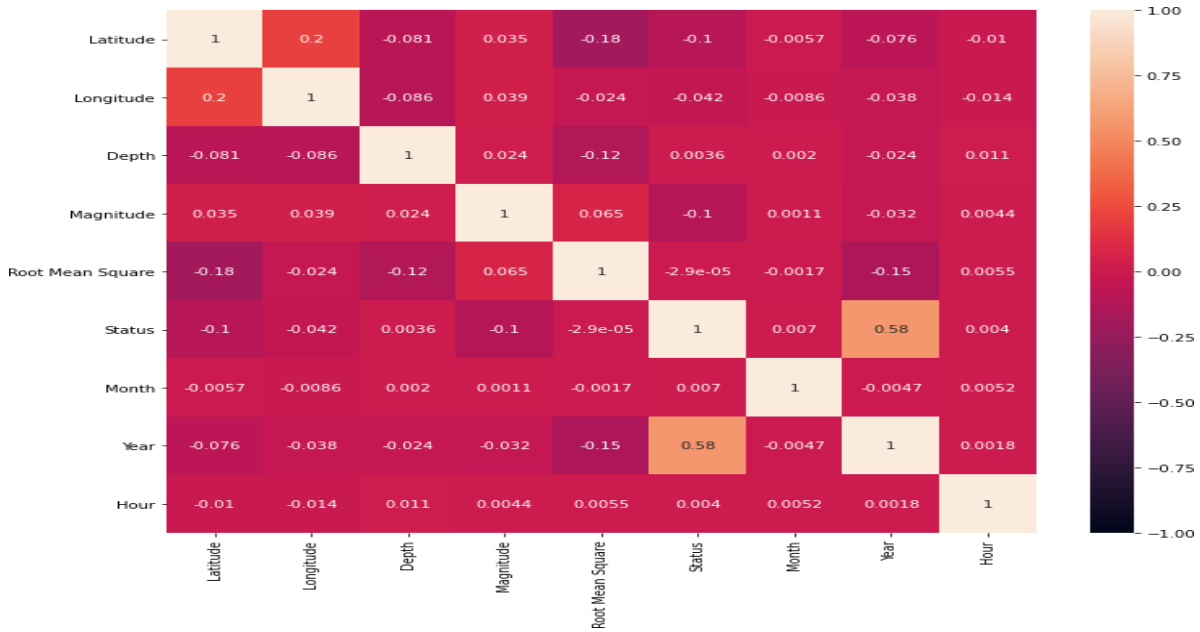
linkcode

```
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)
```

Neural Network model

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

```
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
```



Encoding

```
data['Type'].unique()
```

Out[30]:

```
array(['Earthquake', 'Nuclear Explosion', 'Explosion', 'Rock Burst'],
      dtype=object)
```

In [31]:

```
def onehot_encode(df, columns, prefixes):
    df = df.copy()
    for column, prefix in zip(columns, prefixes):
        dummies = pd.get_dummies(df[column], prefix=prefix)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(column, axis=1)
    return df
```

Splitting and Scaling

```
y = data.loc[:, 'Status']
X = data.drop('Status', axis=1)
scaler = StandardScaler()

X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=56)
```