Configuration Manual

MSc Research Project

MSc Data Analytics

Pranav Ramakrishnan

Student ID: x23107979

School of Computing

National College of Ireland

Supervisor: Abdul Qayum

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. ……………Pranav Ramakrishnan…………………………………………………
Student ID:	………………………x23107979………………………………………………………….……
Programme:	………MSc Data Analytics………… **Year:** …2024-25………
Module:	…………………MSc Research Project……………………………………….………
Lecturer:	……………Abdul Qayum……………………………………….………
Submission Due Date:	……12/12/2024…………………………………………………………….………
Project Title:	………Advanced Earthquake Risk Reduction Through Machine Learning Enhanced Early Warning System…………………
Word Count:	…………1656………………… **Page Count:** ……12…..……..………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …… [illegible] ………………………………………………………………………………

Date: …………………12-12-2024………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pranav Ramakrishnan
Student ID: 23107979

1 Introduction

This manual describes a solution of setting and deploying an automated machine learning system to estimate the earthquake magnitude, the following section presents itself as an effective one in explaining concept insights. The purpose is to enhance Earthquake Early Warning, providing alerts in order to minimize loss of life and property damage.
Moreover, the system processes seismic data, eliminates all but the premier machine learning model among 19 options, and fine-tunes these models. They comprise data pre-processing, model training, testing and deploying the app in the form of an easy to use web application. The app provides predictions of magnitudes and their corresponding risks. This way guide enables the users of the system set to enable them to get better predictions of the earthquakes.

2 System Requirements

Component	**Minimum Specification**	**Recommended Specification**
Processor (CPU)	Dual-Core Processor (e.g., Intel i3 or AMD A4)	Quad-Core Processor (e.g., Intel i5/i7 or AMD Ryzen)
Memory (RAM)	4 GB	16GB
Storage	20GB free space	50 GB free space (preferably SSD)
Graphics (GPU)	Not Required	Dedicated GPU (e.g., NVIDIA GTX 1050 or higher) for deep learning models
Network	Standard Ethernet or Wi-Fi connection	High-speed Internet for faster data processing and model deployment

3 Software Requirements

3.1 Operating system

- System OS: Windows 11
- Processor: i5
- RAM: 8 GB

3.2 Development Environment

Integrated Development Environment (IDE)

Jupyter Notebook (via Anaconda) or Google Colab for interactive coding.

3.3 Additional tools and libraries

- **Python Version**: 3.11.10
- **Core Libraries**:
  - NumPy (>= 1.21.0)
  - Pandas (>= 1.3.0)
  - Scikit-learn (>= 1.0.0)
- **Visualization Tools**:
  - Matplotlib (>= 3.4.0)
  - Seaborn (>= 0.11.0)
  - GeoPandas(>=1.0.2)
- **Machine Learning Frameworks**:
  - PyCaret (>= 3.3.2)
  - XGBoost (>= 1.4.0)
  - LightGBM (>= 3.2.0)
  - CatBoost (>= 0.26)
- **Web App Development**:
  - Streamlit (>= 1.41.0)
- **Model Persistence**:
  - Pickle (standard library)

Other Requirements

- **Browser**: Modern web browser (e.g., Google Chrome, Mozilla Firefox) for Streamlit app visualization.

4. Software Installation

4.1 Installation for Jupyter Notebook

Step 1

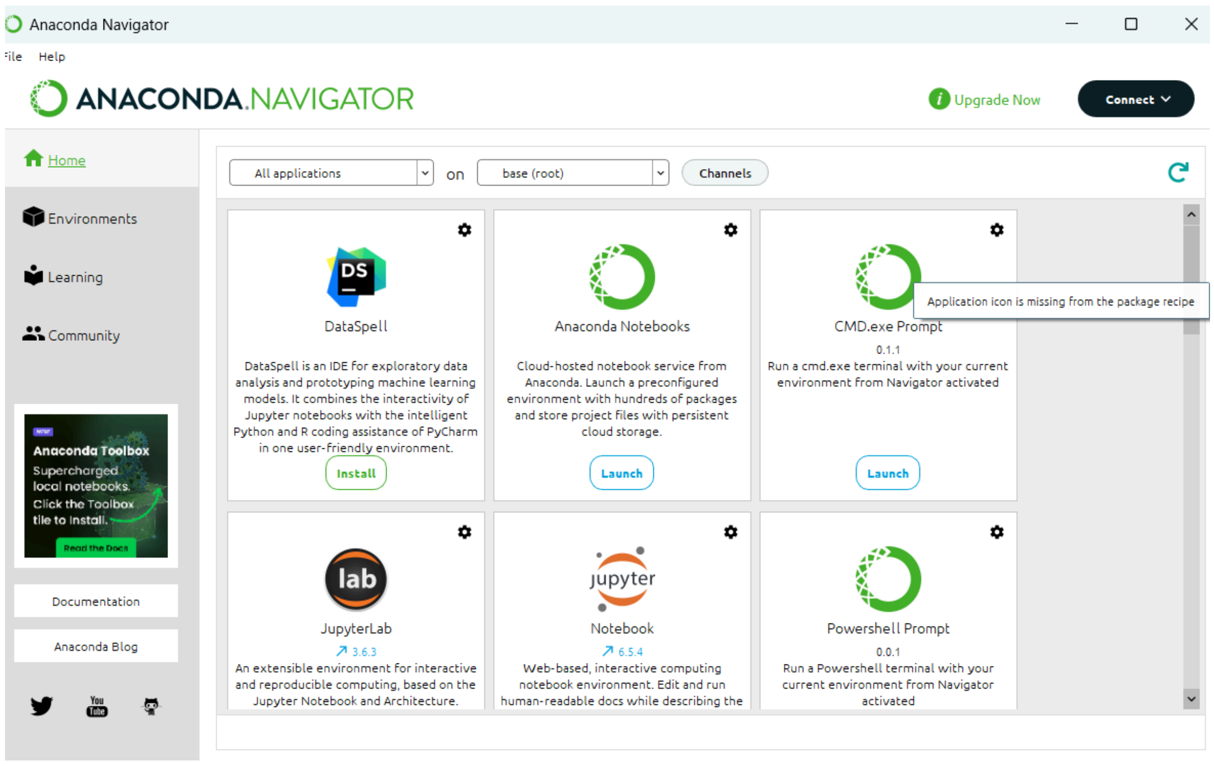
Install Anaconda Navigator from https://www.anaconda.com/download.

Step2

After Installing Anaconda Navigator . Search Anaconda Navigator on the Search bar.

Step 3

Open Anaconda Navigator .Its look like the following image. (fig 1)



DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Install

Anaconda Notebooks

Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

Launch

CMD.exe Prompt

0.1.1

Run a cmd.exe terminal with your current environment from Navigator activated

Launch

JupyterLab

↗ 3.6.3

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook

↗ 6.5.4

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the

Powershell Prompt

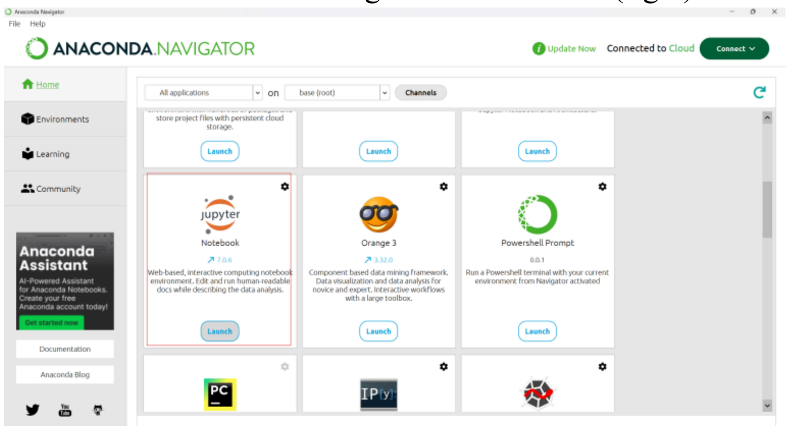
0.0.1

Run a Powershell terminal with your current environment from Navigator activated

Figure 1: Anaconda Navigator Home Page

Step 4

Search Jupyter Notebook on Anaconda Navigator and install it. (fig 2)



store project files with persistent cloud storage.

Launch

Launch

Launch

Notebook

↗ 7.0.6

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch

Orange 3

↗ 3.32.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

Launch

Powershell Prompt

0.0.1

Run a Powershell terminal with your current environment from Navigator activated

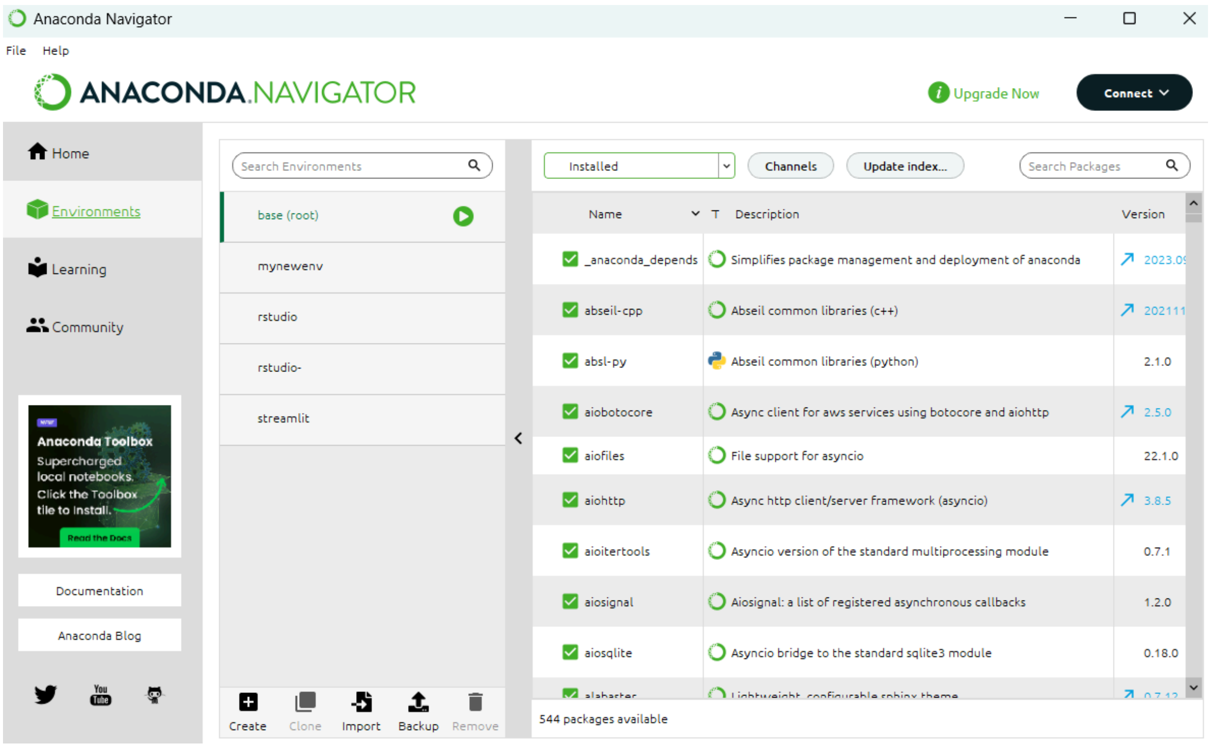
Launch

Figure 2: Jupyter notebook selection option

After Installing Jupyter Notebook we see the Jupyter Notebook installed on your local environment you access using typing jupyter notebook on search bar or you can use Command Prompt for accessing this jupyter notebook.

Step 5

create a Python Environment in Anaconda Navigator where Jupyter Notebook will run on this python Environment



Name	Description	Version
_anaconda_depends	Simplifies package management and deployment of anaconda	2023.0
abseil-cpp	Abseil common libraries (c++)	202111
absl-py	Abseil common libraries (python)	2.1.0
aiobotocore	Async client for aws services using botocore and aiohttp	2.5.0
aiofiles	File support for asyncio	22.1.0
aiohttp	Async http client/server framework (asyncio)	3.8.5
aioitertools	Asyncio version of the standard multiprocessing module	0.7.1
aiosignal	Aiosignal: a list of registered asynchronous callbacks	1.2.0
aiosqlite	Asyncio bridge to the standard sqlite3 module	0.18.0
alabaster	Lightweight, configurable sphinx theme	0.7.12

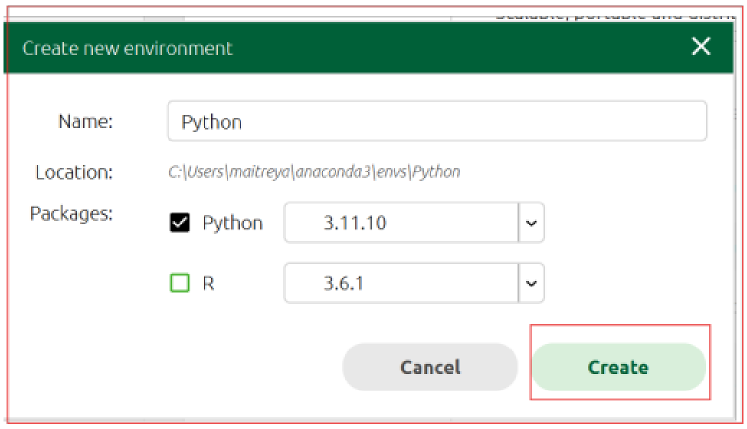
544 packages available

Figure 3: Jupyter notebook Environment page

Go to Environment Tab, which is on the left side of the Panel below the Home tab.After Selecting Environment Tab There is a Create Option which is shown just like above (fig3)Click on that option .

step 6

After Clicking on the ”CREATE ” option you will get the Following Figure 4 type on your screen .


Create new environment

Name: Python

Location: C:\Users\maitreya\anaconda3\envs\Python

Packages: ☑ Python 3.11.10

☐ R 3.6.1

Figure 4:Python Environment Creation

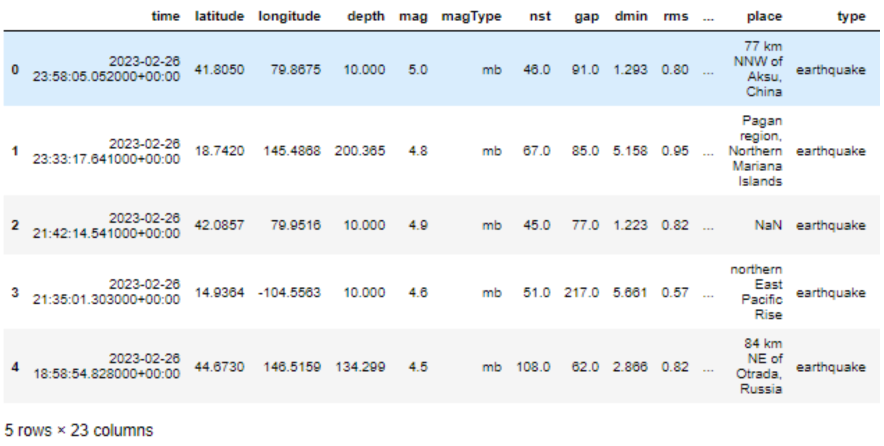
Select Python with 3.11.10 version and hit the create Tab shown in Figure 4

5. Data Preparation

5.1 Collection of data

The data is collected from kaggle.The name of the dataset is World Earthquake data from 1906-2022.The dataset contains the features such as magnitude,depth, latitude longitude and other relevant columns .This dataset have the earthquake data of different regions and years of more than 300000 rows.First five rows of the data set is in figure 5

https://www.kaggle.com/datasets/garrickhague/world-earthquake-data-from-1906-2022



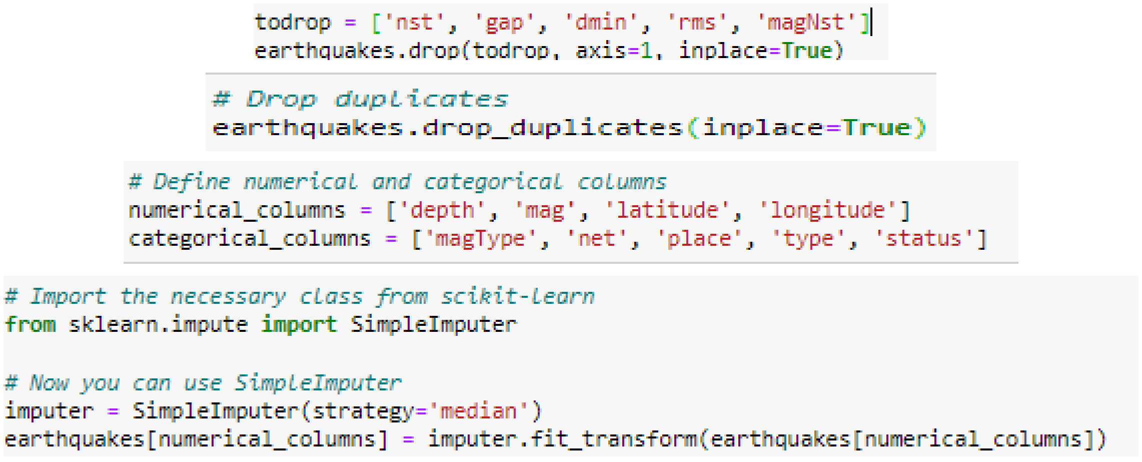
	time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms	...	place	type
0	2023-02-26 23:58:05.052000+00:00	41.8050	79.8675	10.000	5.0	mb	46.0	91.0	1.293	0.80	...	77 km NNW of Aksu, China	earthquake
1	2023-02-26 23:33:17.641000+00:00	18.7420	145.4868	200.365	4.8	mb	67.0	85.0	5.158	0.95	...	Pagan region, Northern Mariana Islands	earthquake
2	2023-02-26 21:42:14.541000+00:00	42.0857	79.9516	10.000	4.9	mb	45.0	77.0	1.223	0.82	...	NaN	earthquake
3	2023-02-26 21:35:01.303000+00:00	14.9364	-104.5563	10.000	4.6	mb	51.0	217.0	5.661	0.57	...	northern East Pacific Rise	earthquake
4	2023-02-26 18:58:54.828000+00:00	44.6730	146.5159	134.299	4.5	mb	108.0	62.0	2.866	0.82	...	84 km NE of Otrada, Russia	earthquake

5 rows × 23 columns

Figure 5:First 5 rows of the dataset

5.2 Data Cleaning

In figure 6 the first code dropped redundant or less critical columns to streamline the dataset. and the second code is for deleting the duplicate in the dataset and the third code make the missing values in the dataset to numerical values.The last code makes the all the missing values in the numerical columns are replaced with their respective medium values.



```
todrop = ['nst', 'gap', 'dmin', 'rms', 'magNst']
earthquakes.drop(todrop, axis=1, inplace=True)
```

```
# Drop duplicates
earthquakes.drop_duplicates(inplace=True)
```

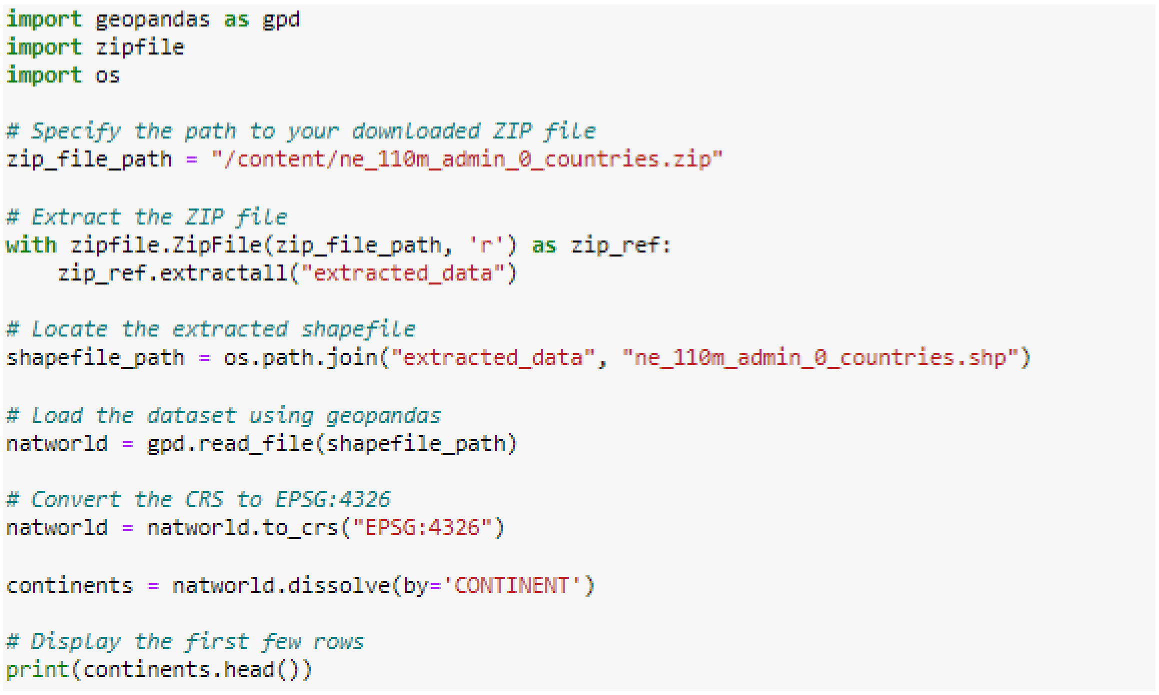
```
# Define numerical and categorical columns
numerical_columns = ['depth', 'mag', 'latitude', 'longitude']
categorical_columns = ['magType', 'net', 'place', 'type', 'status']
```

```
# Import the necessary class from scikit-learn
from sklearn.impute import SimpleImputer

# Now you can use SimpleImputer
imputer = SimpleImputer(strategy='median')
earthquakes[numerical_columns] = imputer.fit_transform(earthquakes[numerical_columns])
```

Figure 6: Data Cleaning

5.3 Exploratory Data Analysis

Using Geopandas 2 maps has been created with the help of an external zip file. The code used for that is in figure 7.The first map represents the magnitude level with the colour and the other map represents the depth level and histogram has been created represents the earthquake that happens over time.

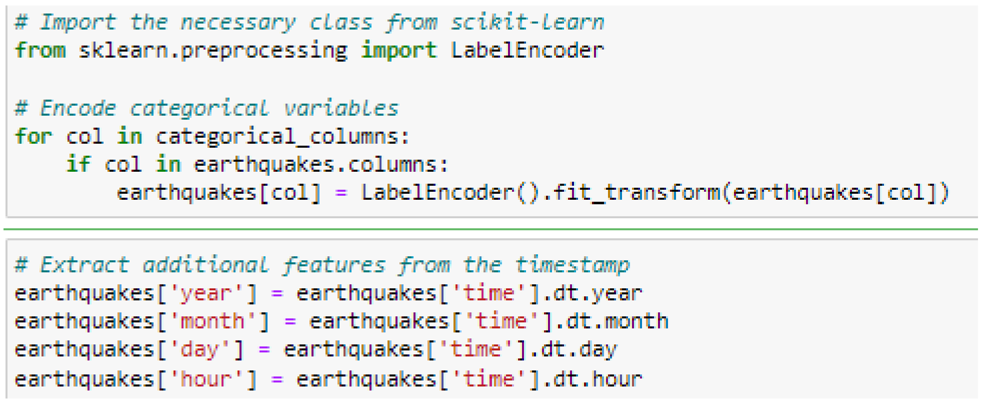


```python
import geopandas as gpd
import zipfile
import os

# Specify the path to your downloaded ZIP file
zip_file_path = "/content/ne_110m_admin_0_countries.zip"

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("extracted_data")

# Locate the extracted shapefile
shapefile_path = os.path.join("extracted_data", "ne_110m_admin_0_countries.shp")

# Load the dataset using geopandas
natworld = gpd.read_file(shapefile_path)

# Convert the CRS to EPSG:4326
natworld = natworld.to_crs("EPSG:4326")

continents = natworld.dissolve(by='CONTINENT')

# Display the first few rows
print(continents.head())
```

Figure 7: Geopandas

5.4 Data Transformation

Using the LabelEncoder all categorical columns are converted to numerical values, enabling compatibility with most machine learning algorithms.(fig8) and extract additional features from the time column for capturing the temporal patterns . Four new columns have been added; those are Year, Month,Day,Hour.These will be useful for time-based analysis or feature engineering.



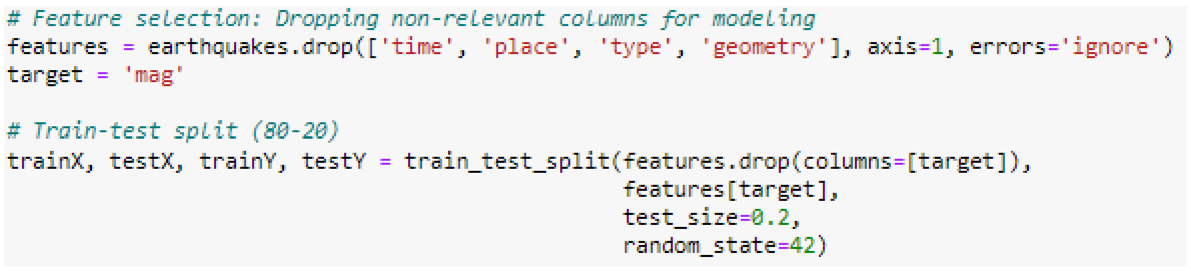
```python
# Import the necessary class from scikit-learn
from sklearn.preprocessing import LabelEncoder

# Encode categorical variables
for col in categorical_columns:
    if col in earthquakes.columns:
        earthquakes[col] = LabelEncoder().fit_transform(earthquakes[col])
```

```python
# Extract additional features from the timestamp
earthquakes['year'] = earthquakes['time'].dt.year
earthquakes['month'] = earthquakes['time'].dt.month
earthquakes['day'] = earthquakes['time'].dt.day
earthquakes['hour'] = earthquakes['time'].dt.hour
```

Figure 8: Data Transformation code

6 Feature selection and Data Splitting

By taking the “mag” as target variable and dropping the non relevant columns like time,place type and Geometry(fig9).The data has been splitted in to two 20 for testing and 80 for training

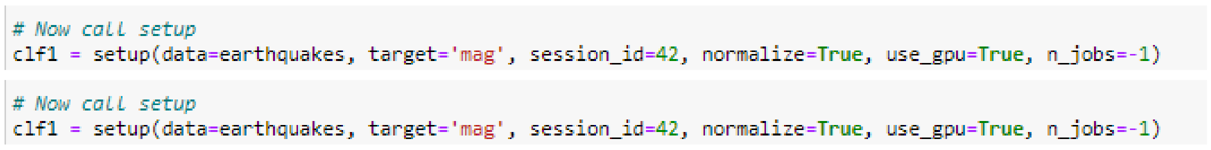


```
# Feature selection: Dropping non-relevant columns for modeling
features = earthquakes.drop(['time', 'place', 'type', 'geometry'], axis=1, errors='ignore')
target = 'mag'

# Train-test split (80-20)
trainX, testX, trainY, testY = train_test_split(features.drop(columns=[target]),
                                                features[target],
                                                test_size=0.2,
                                                random_state=42)
```

Figure 9: code for feature selection and data splitting

7 PyCaret setup

Using the code from figure 10 the PyCaret setup for Regression has been initialised and the best model of the regression has been compared



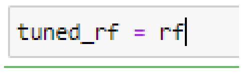
```
# Now call setup
clf1 = setup(data=earthquakes, target='mag', session_id=42, normalize=True, use_gpu=True, n_jobs=-1)
```

```
# Now call setup
clf1 = setup(data=earthquakes, target='mag', session_id=42, normalize=True, use_gpu=True, n_jobs=-1)
```

Figure 10 PyCaret Setup

7.1 Model tuning

From the compared models 8 best performing model has been selected and has been tuned individual.The code used for tung is in figure 11

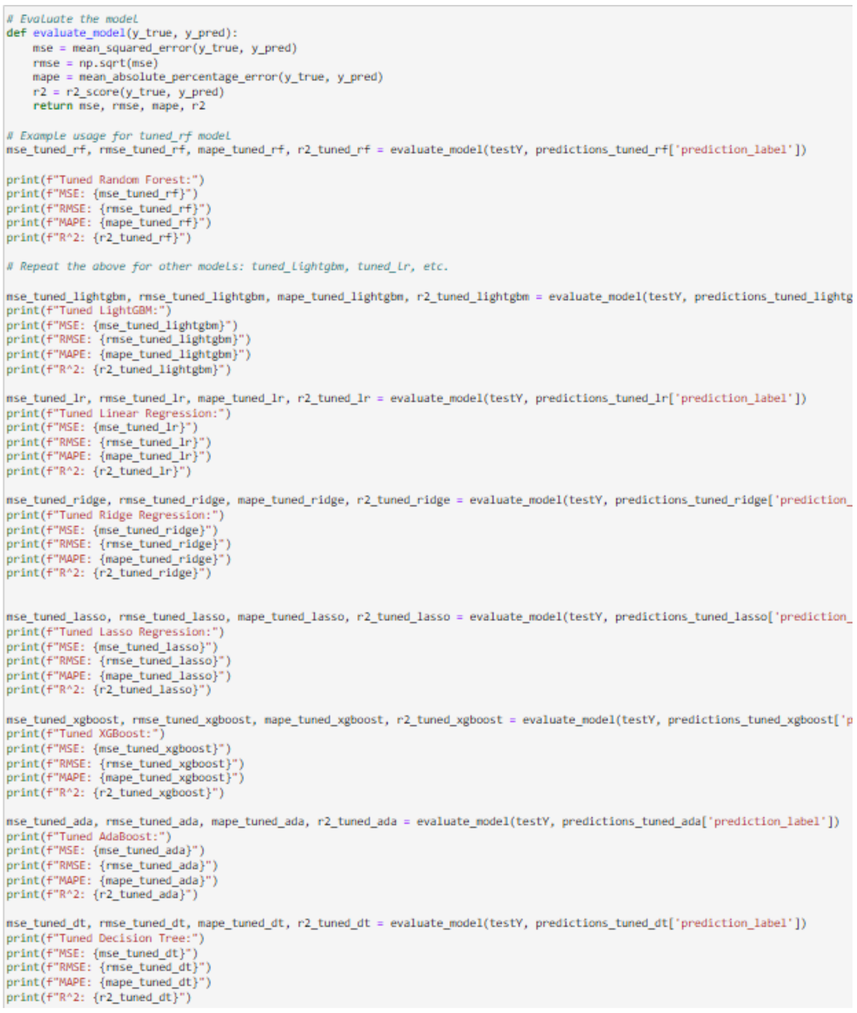


```
tuned_rf = rf
```

Figure 11 Tuning

7.2 Evaluate the models

Each selected model has been evaluated individually for finding the Mean square error,Root mean square error,Mean absolute percentage error and R-squared value.the code used is in the figure 12

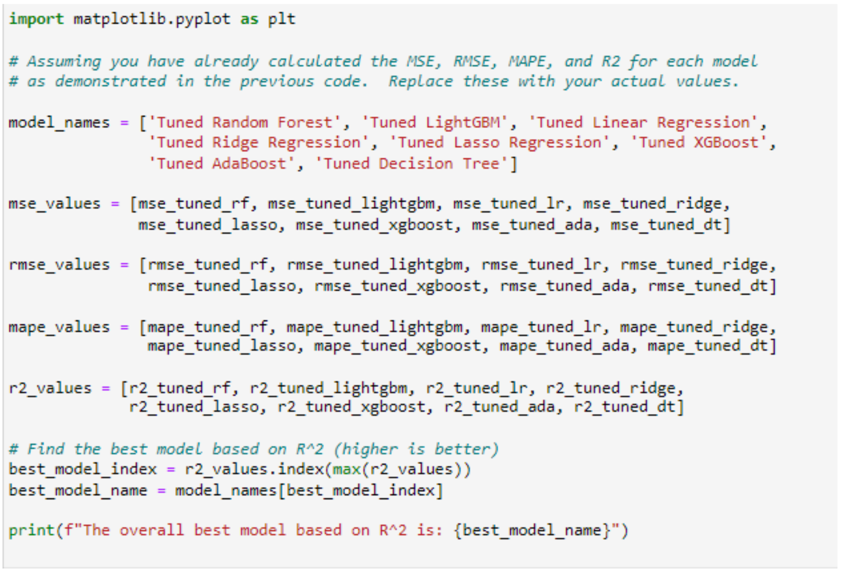


```python
# Evaluate the model
def evaluate_model(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, rmse, mape, r2

# Example usage for tuned_rf model
mse_tuned_rf, rmse_tuned_rf, mape_tuned_rf, r2_tuned_rf = evaluate_model(testY, predictions_tuned_rf['prediction_label'])

print(f"Tuned Random Forest:")
print(f"MSE: {mse_tuned_rf}")
print(f"RMSE: {rmse_tuned_rf}")
print(f"MAPE: {mape_tuned_rf}")
print(f"R^2: {r2_tuned_rf}")

# Repeat the above for other models: tuned_lightgbm, tuned_lr, etc.

mse_tuned_lightgbm, rmse_tuned_lightgbm, mape_tuned_lightgbm, r2_tuned_lightgbm = evaluate_model(testY, predictions_tuned_lightg
print(f"Tuned LightGBM:")
print(f"MSE: {mse_tuned_lightgbm}")
print(f"RMSE: {rmse_tuned_lightgbm}")
print(f"MAPE: {mape_tuned_lightgbm}")
print(f"R^2: {r2_tuned_lightgbm}")

mse_tuned_lr, rmse_tuned_lr, mape_tuned_lr, r2_tuned_lr = evaluate_model(testY, predictions_tuned_lr['prediction_label'])
print(f"Tuned Linear Regression:")
print(f"MSE: {mse_tuned_lr}")
print(f"RMSE: {rmse_tuned_lr}")
print(f"MAPE: {mape_tuned_lr}")
print(f"R^2: {r2_tuned_lr}")

mse_tuned_ridge, rmse_tuned_ridge, mape_tuned_ridge, r2_tuned_ridge = evaluate_model(testY, predictions_tuned_ridge['prediction_
print(f"Tuned Ridge Regression:")
print(f"MSE: {mse_tuned_ridge}")
print(f"RMSE: {rmse_tuned_ridge}")
print(f"MAPE: {mape_tuned_ridge}")
print(f"R^2: {r2_tuned_ridge}")


mse_tuned_lasso, rmse_tuned_lasso, mape_tuned_lasso, r2_tuned_lasso = evaluate_model(testY, predictions_tuned_lasso['prediction_
print(f"Tuned Lasso Regression:")
print(f"MSE: {mse_tuned_lasso}")
print(f"RMSE: {rmse_tuned_lasso}")
print(f"MAPE: {mape_tuned_lasso}")
print(f"R^2: {r2_tuned_lasso}")

mse_tuned_xgboost, rmse_tuned_xgboost, mape_tuned_xgboost, r2_tuned_xgboost = evaluate_model(testY, predictions_tuned_xgboost['p
print(f"Tuned XGBoost:")
print(f"MSE: {mse_tuned_xgboost}")
print(f"RMSE: {rmse_tuned_xgboost}")
print(f"MAPE: {mape_tuned_xgboost}")
print(f"R^2: {r2_tuned_xgboost}")

mse_tuned_ada, rmse_tuned_ada, mape_tuned_ada, r2_tuned_ada = evaluate_model(testY, predictions_tuned_ada['prediction_label'])
print(f"Tuned AdaBoost:")
print(f"MSE: {mse_tuned_ada}")
print(f"RMSE: {rmse_tuned_ada}")
print(f"MAPE: {mape_tuned_ada}")
print(f"R^2: {r2_tuned_ada}")

mse_tuned_dt, rmse_tuned_dt, mape_tuned_dt, r2_tuned_dt = evaluate_model(testY, predictions_tuned_dt['prediction_label'])
print(f"Tuned Decision Tree:")
print(f"MSE: {mse_tuned_dt}")
print(f"RMSE: {rmse_tuned_dt}")
print(f"MAPE: {mape_tuned_dt}")
print(f"R^2: {r2_tuned_dt}")
```

Figure 11: Evaluation of each model

8 Best model

From the evaluation the best model has been selected using this code in figure 12

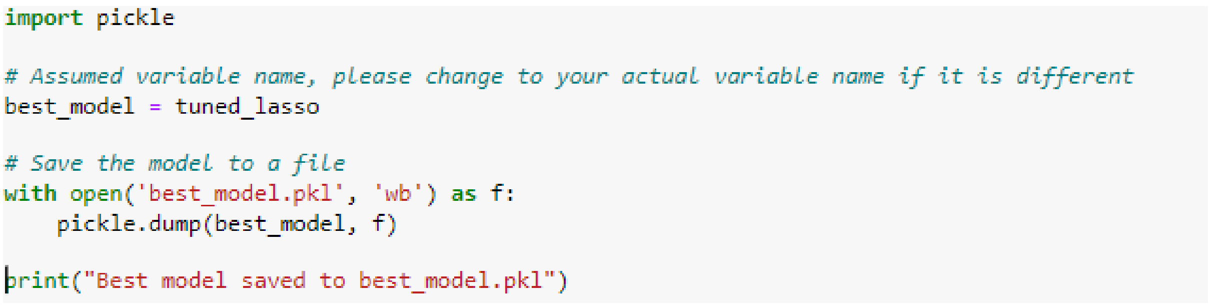


```python
import matplotlib.pyplot as plt

# Assuming you have already calculated the MSE, RMSE, MAPE, and R2 for each model
# as demonstrated in the previous code.  Replace these with your actual values.

model_names = ['Tuned Random Forest', 'Tuned LightGBM', 'Tuned Linear Regression',
               'Tuned Ridge Regression', 'Tuned Lasso Regression', 'Tuned XGBoost',
               'Tuned AdaBoost', 'Tuned Decision Tree']

mse_values = [mse_tuned_rf, mse_tuned_lightgbm, mse_tuned_lr, mse_tuned_ridge,
              mse_tuned_lasso, mse_tuned_xgboost, mse_tuned_ada, mse_tuned_dt]

rmse_values = [rmse_tuned_rf, rmse_tuned_lightgbm, rmse_tuned_lr, rmse_tuned_ridge,
               rmse_tuned_lasso, rmse_tuned_xgboost, rmse_tuned_ada, rmse_tuned_dt]

mape_values = [mape_tuned_rf, mape_tuned_lightgbm, mape_tuned_lr, mape_tuned_ridge,
               mape_tuned_lasso, mape_tuned_xgboost, mape_tuned_ada, mape_tuned_dt]

r2_values = [r2_tuned_rf, r2_tuned_lightgbm, r2_tuned_lr, r2_tuned_ridge,
             r2_tuned_lasso, r2_tuned_xgboost, r2_tuned_ada, r2_tuned_dt]

# Find the best model based on R^2 (higher is better)
best_model_index = r2_values.index(max(r2_values))
best_model_name = model_names[best_model_index]

print(f"The overall best model based on R^2 is: {best_model_name}")
```

Figure 12:Best model code

8.1 Saving the best model

The best model has been saved by a pickle file using pickle library.The code used for that is in figure 13

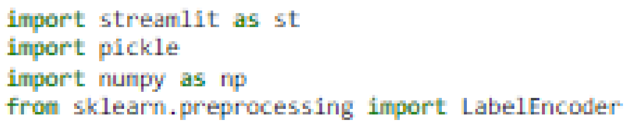


```python
import pickle

# Assumed variable name, please change to your actual variable name if it is different
best_model = tuned_lasso

# Save the model to a file
with open('best_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)

print("Best model saved to best_model.pkl")
```

Figure 13:Saving the best model

9 Deployment of web app

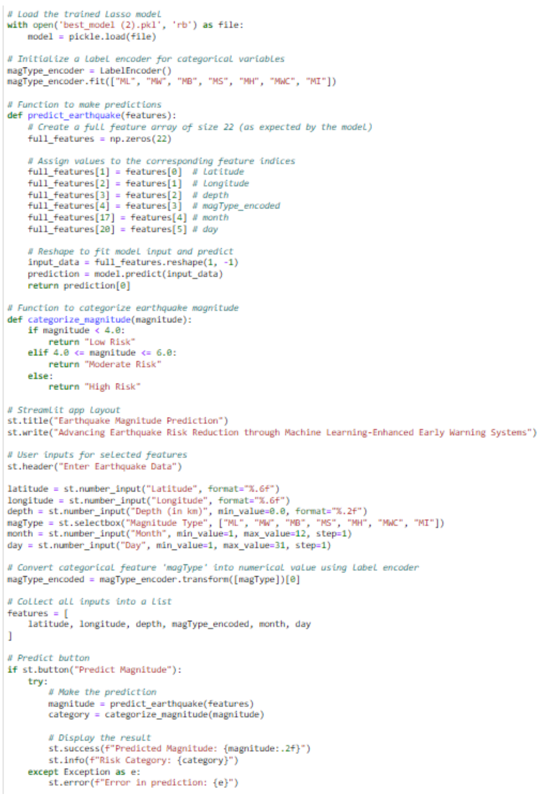
Import the liberates for the deployment of the app the stream for the deployment that we use is Stramlit.(fig 14)



```python
import streamlit as st
import pickle
import numpy as np
from sklearn.preprocessing import LabelEncoder
```

Figure 14:libraries for web app

Then the code for developing the web app using the best model in figure 15.

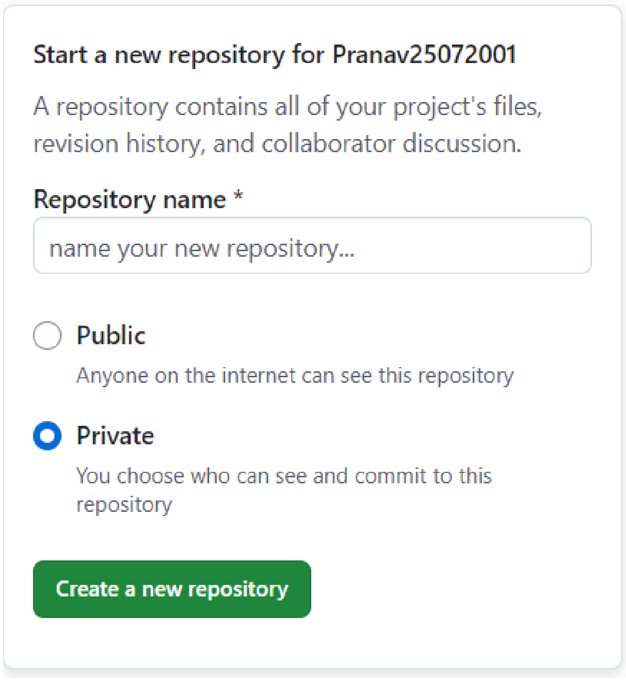


```python
# Load the trained Lasso model
with open('best_model (2).pkl', 'rb') as file:
    model = pickle.load(file)

# Initialize a label encoder for categorical variables
magType_encoder = LabelEncoder()
magType_encoder.fit(["ML", "MW", "MB", "MS", "MH", "MWC", "MI"])

# Function to make predictions
def predict_earthquake(features):
    # Create a full feature array of size 22 (as expected by the model)
    full_features = np.zeros(22)

    # Assign values to the corresponding feature indices
    full_features[1] = features[0]  # latitude
    full_features[2] = features[1]  # longitude
    full_features[3] = features[2]  # depth
    full_features[4] = features[3]  # magType_encoded
    full_features[17] = features[4] # month
    full_features[20] = features[5] # day

    # Reshape to fit model input and predict
    input_data = full_features.reshape(1, -1)
    prediction = model.predict(input_data)
    return prediction[0]

# Function to categorize earthquake magnitude
def categorize_magnitude(magnitude):
    if magnitude < 4.0:
        return "Low Risk"
    elif 4.0 <= magnitude <= 6.0:
        return "Moderate Risk"
    else:
        return "High Risk"

# Streamlit app layout
st.title("Earthquake Magnitude Prediction")
st.write("Advancing Earthquake Risk Reduction through Machine Learning-Enhanced Early Warning Systems")

# User inputs for selected features
st.header("Enter Earthquake Data")

latitude = st.number_input("Latitude", format="%.6f")
longitude = st.number_input("Longitude", format="%.6f")
depth = st.number_input("Depth (in km)", min_value=0.0, format="%.2f")
magType = st.selectbox("Magnitude Type", ["ML", "MW", "MB", "MS", "MH", "MWC", "MI"])
month = st.number_input("Month", min_value=1, max_value=12, step=1)
day = st.number_input("Day", min_value=1, max_value=31, step=1)

# Convert categorical feature 'magType' into numerical value using label encoder
magType_encoded = magType_encoder.transform([magType])[0]

# Collect all inputs into a list
features = [
    latitude, longitude, depth, magType_encoded, month, day
]

# Predict button
if st.button("Predict Magnitude"):
    try:
        # Make the prediction
        magnitude = predict_earthquake(features)
        category = categorize_magnitude(magnitude)

        # Display the result
        st.success(f"Predicted Magnitude: {magnitude:.2f}")
        st.info(f"Risk Category: {category}")
    except Exception as e:
        st.error(f"Error in prediction: {e}")
```

Figure 15:code of web app

9.1 Creating a repository in github

By clicking on the create repository with the name of it(fig16).A repository has been made.



Start a new repository for Pranav25072001

A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *

name your new repository...

Public

Anyone on the internet can see this repository

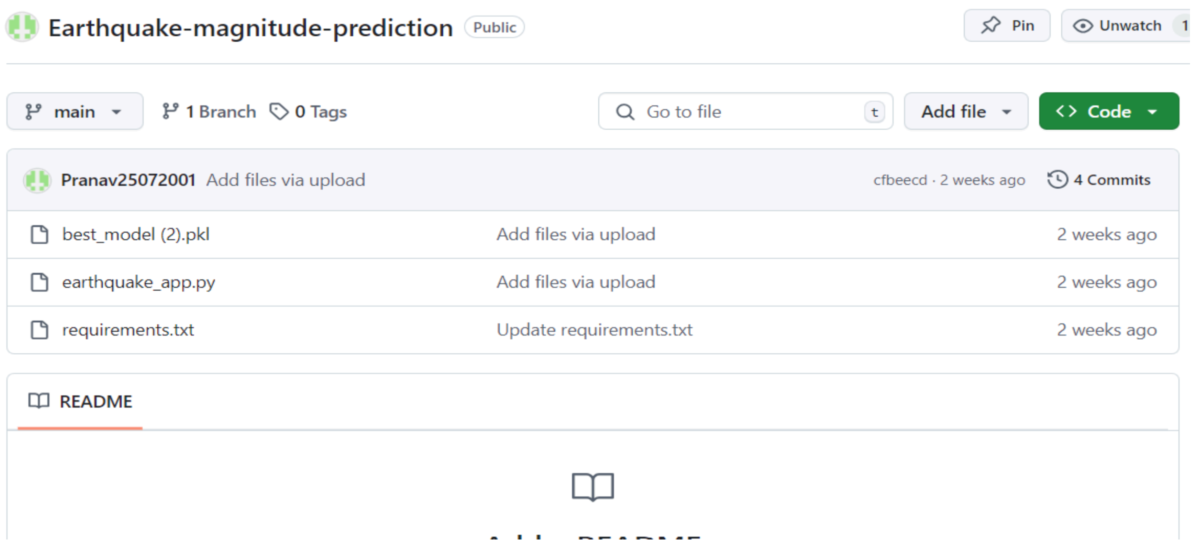
[✓ Private]

You choose who can see and commit to this repository

Create a new repository

Figure 16: Creating a repository

Add the required text file then the code of the web app(fig 15) the the best model to this repository(fig 17)



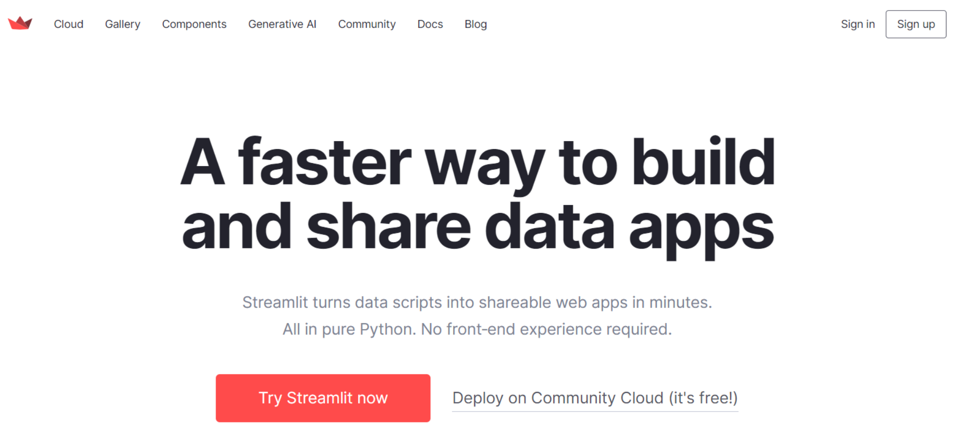
Earthquake-magnitude-prediction Public

Pranav25072001 Add files via upload cfbeecd · 2 weeks ago 4 Commits

best_model (2).pkl	Add files via upload	2 weeks ago
earthquake_app.py	Add files via upload	2 weeks ago
requirements.txt	Update requirements.txt	2 weeks ago

Figure 17: Adding the files to the repository

Open the streamlit in the browser the interface in figure 18 and on the ring side there will be a signup button.create the account using the github account.

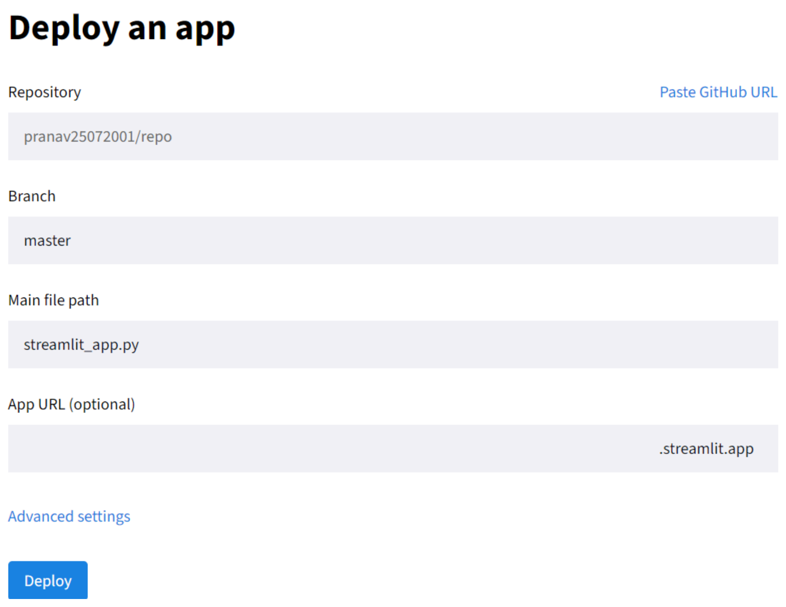


A faster way to build and share data apps

Streamlit turns data scripts into shareable web apps in minutes.
All in pure Python. No front-end experience required.

Figure 18:Interface of Streamlit

By clicking the create app after signup connect the repository and give the detail that is need.This has been shown in the figure 19. By clicking the deploy button the app will be created.


Deploy an app

Repository: pranav25072001/repo

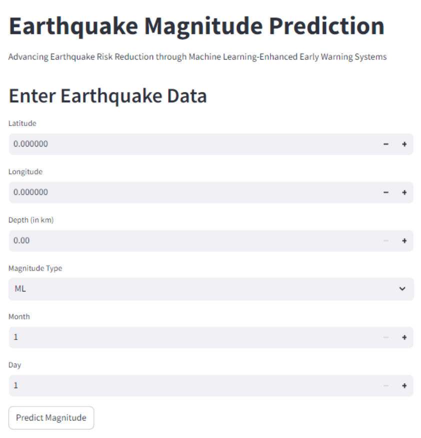
Branch: master

Main file path: streamlit_app.py

App URL (optional): .streamlit.app

Figure 19: Repository connection to Streamlit

The interface of the app is in figure 20


Earthquake Magnitude Prediction

Advancing Earthquake Risk Reduction through Machine Learning-Enhanced Early Warning Systems

Enter Earthquake Data

Latitude: 0.000000

Longitude: 0.000000

Depth (in km): 0.00

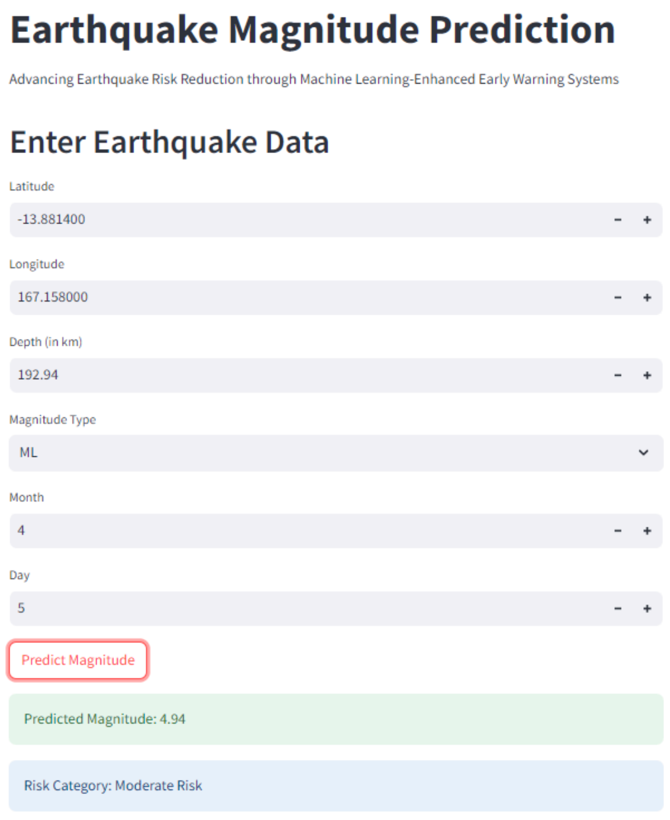
Magnitude Type: ML

Month: 1

Day: 1

Figure 20 The interface of Web app

The magnitude prediction of the web app is in figure 21.The output will look like this.


Earthquake Magnitude Prediction

Advancing Earthquake Risk Reduction through Machine Learning-Enhanced Early Warning Systems

Enter Earthquake Data

Latitude: -13.881400

Longitude: 167.158000

Depth (in km): 192.94

Magnitude Type: ML

Month: 4

Day: 5

Predicted Magnitude: 4.94

Risk Category: Moderate Risk

Figure 21: Magnitude prediction

References

Annamalai, R., Deena, S., Venkatakrishnan, R., Shankar, H., Harshitha, Y.S., Harini, K. and Reddy, M.N., 2023, December. Automating Machine Learning Model Development: An OperationalML Approach with PyCARET and Streamlit. In *2023 Innovations in Power and Advanced Computing Technologies (i-PACT)* (pp. 1-6). IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, pp.2825-2830.

Richards, T., 2021. *Getting Started with Streamlit for Data Science: Create and deploy Streamlit web applications from scratch in Python*. Packt Publishing Ltd.

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T. and Church, K., 2016. Introduction to Anaconda and Python: Installation and setup. *Quant. Methods Psychol*, *16*(5), pp.S3-S11.