Building an earthquake prediction model using Python

Problem Definition:

The problem at hand is the development of an earthquake prediction model using Python to predict the occurrence of earthquakes before their arrival and taking preemptive and precautionary measures to reduce their impact.

Background:

In today’s world technology has developed to an advanced level where we are able to predict weather and are able to analyse the effects of many dangers and prevent them, we are still unable to detect the occurrence of earthquakes which is a major dissapointment. Predicting earthquakes is one of the great unsolved problems in the earth sciences. With the increase in the use of technology, many seismic monitoring stations have increased, so we can use machine learning and other data-driven methods to predict earthquakes.

Objective:

The primary objective is to build an earthquake prediction model using Python which is:

1. Able to detect whether an earthquake will occur in a specific area or not .
2. Developing a system that can provide early warnings to communities in earthquake-prone areas is crucial for minimising the impact of earthquakes. The model should be able to issue alerts as soon as possible after detecting seismic activity.
3. Determine the location and magnitude of an earthquake event. This information is essential for emergency response and can help in estimating the potential damage.
4. Integrate data from various sources, including seismometers, GPS sensors, and satellite imagery, to improve the accuracy of earthquake detection and prediction.
5. Ensure that the model operates in real-time, continuously monitoring seismic data and providing timely alerts.

Design Thinking:

1.Gather earthquake data from reliable sources. You can obtain earthquake data from sources like the USGS Earthquake Catalog or other geological institutions.

2.Clean and preprocess the earthquake data. This may involve:

* Removing duplicates or irrelevant columns.
* Handling missing data.
* Converting timestamps to a consistent format.
* Scaling or normalising numerical features.
* Encoding categorical features if necessary.
* Splitting the data into training, validation, and test sets.

3.Extract relevant features from the data that can help the model distinguish earthquake signals from noise. Feature engineering might involve:

* Time-based features (e.g., time of day, day of the week).
* Spatial features (e.g., location, distance to fault lines).
* Frequency domain features (e.g., Fourier transforms).
* Statistical features (e.g., mean, standard deviation).

4.Choose an appropriate machine learning or deep learning model for earthquake detection. Some common models include:

* Logistic Regression
* Random Forest
* Support Vector Machine (SVM)
* Convolutional Neural Network (CNN)
* Recurrent Neural Network (RNN)

5.Train the selected model on the training dataset using appropriate training techniques:

* Adjust hyperparameters (e.g., learning rate, batch size).
* Implement data augmentation if you have limited earthquake data.
* Monitor training progress and evaluate performance on the

validation set.

* Consider early stopping to prevent overfitting.

6.Evaluate the model's performance on the test dataset using relevant evaluation metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.

7.Fine-tune the model's hyperparameters using techniques like grid search or random search to optimise its performance.

8.Deploy the trained earthquake detector model to a production environment. This can be done using frameworks like Flask, Django, or cloud services like AWS, Azure, or Google Cloud.

9.Continuously monitor the model's performance in the production environment and update it as needed to adapt to changing data distributions or conditions.

10.If required, analyse the model's predictions to understand why it makes certain decisions. This can be crucial for building trust in the model's predictions.

11.Maintain thorough documentation of the entire process, including data sources, preprocessing steps, model architecture, and hyperparameters, for future reference and collaboration.

Outcome:

The specific outcome of your earthquake detector model will vary depending on the goals, resources, and data available for your project. A successful outcome would be a well performing model that contributes to earthquake monitoring and improves our understanding of seismic activity in a given region.

Success Metrics:

The choice of success metrics for your earthquake detector model will depend on our project's specific goals and requirements. Here are some common success metrics that we can consider are accuracy, precision, sensitivity, F1-score, ROC curve and AUC-ROC, AUC-PR, MAE, MSE, RMSE, specificity.

Stakeholders:

An earthquake detection model, when successfully designed and deployed, can benefit various stakeholders and serve a range of purposes. Some of the key beneficiaries of such a model are Public Safety Authorities, Civilian Populations, Infrastructure and Building Safety, Seismologists and Researches, Environmental Agencies, Insurance Companies, Government Agencies, and Commercial Interests.

Significance:

The significance of an earthquake detection model lies in its potential to address several critical issues related to earthquake monitoring, safety, and preparedness. The key aspects of its significance are

1. Early Warning and Public Safety.
2. Reducing Casualties and Injuries.
3. Protecting Infrastructure.
4. Improved Building and Infrastructure Design.
5. Resource Allocation.

Building an earthquake prediction model using Python

Phase 2:

Considering advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's performance.

Steps in implementing the model:

1.Gather the earthquake data from reliable sources. We can obtain earthquake data from the internet for India.

2.Clean and preprocess the earthquake data. This may involve removing duplicates or irrelevant columns, handling missing data,

converting timestamps to consistent format, scaling numerical features,

encoding categorical features if necessary, splitting the data into [training, validation, test sets].

3.Extracting relevant features from the data which help the model to distinguish earthquake signals from noise.

4.Choose an appropriate machine learning or deep learning model for earthquake detection.

5.Train the selected model on the training dataset using appropriate training techniques.

6.Evaluate the model's performance on the test dataset using relevant evaluation metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.

7.Fine-tune the model's hyperparameters using techniques like grid search or random search to optimise its performance.

8.Deploy the trained earthquake detector model to a production environment. This is done using frameworks like Flask or cloud services like AWS, Azure, or Google Cloud.

9.Continuously monitor the model's performance in the production environment and update it as needed to adapt to changing data distributions or conditions.

10.Maintain thorough documentation of the entire process, including data sources, preprocessing steps, model architecture, and hyperparameters, for future reference and collaboration.

Hyperparameter tuning:

Hyperparameter tuning involves systematically searching for the best combination of hyperparameters to optimise a model's performance. Hyperparameters are parameters that are not learned from the data but are set prior to training and can significantly impact a model's ability to learn from the data. Examples include learning rates, the number of hidden layers in a neural network, the depth of a decision tree, regularisation strength, batch size, and more.

**1. Importance of Hyperparameter Tuning:**

Hyperparameters play a crucial role in determining a model's performance. Poorly chosen hyperparameters can lead to issues like overfitting (model is too complex and fits the training data perfectly but performs poorly on new data) or underfitting (model is too simple to capture underlying patterns). Hyperparameter tuning seeks to find the best hyperparameters that strike a balance between model complexity and performance.

**2. Methods for Hyperparameter Tuning:**

There are several methods for hyperparameter tuning, but two of the most common approaches are:

**a. Grid Search:**

* In grid search, we specify a range of possible values for each hyperparameter.
* After that the algorithm systematically tests all combinations of these hyperparameters using cross-validation.
* It evaluates the model's performance (usually based on a chosen metric) for each combination.
* The combination that produces the best performance is selected as the optimal set of hyperparameters.

**b. Random Search:**

Random search is similar to grid search but more efficient in many cases. Instead of exhaustively trying all combinations, it randomly samples hyperparameters from predefined ranges. This approach can discover good hyperparameter combinations more quickly than grid search.

**3. Cross-Validation:**

Cross-validation is essential during hyperparameter tuning to estimate how well the model will perform on unseen data. Steps in k-fold cross-validation are

* Divide the dataset into k subsets or folds.
* The model is trained on k-1 folds and tested on the remaining fold.
* The process is repeated k times, each time with a different fold held out for testing.
* The average performance across all folds is used as an estimate of the model's performance.

**4. Performance Metrics:**

Choose appropriate performance metrics which align with the specific problem that we are trying to solve(.i.e.The earthquake prediction model). Common metrics include accuracy, precision, recall, F1-score, mean squared error, or area under the receiver operating characteristic curve (AUC-ROC).

**5. Iterative Process:**

It is an iterative process. We need to refine the search space, test different combinations for multiple times to find the best hyperparameters.

**6. Automated Hyperparameter Tuning:**

Automated hyperparameter tuning tools and libraries like scikit-learn's GridSearchCV, RandomizedSearchCV, or specialised libraries like Optuna, and hyperopt can simplify the process and make it more efficient.

**7. Domain Knowledge:**

Domain knowledge can be invaluable when selecting hyperparameters. Understanding the problem and the characteristics of the data can help us make informed choices about hyperparameter ranges and values.

**College Code :** 9508

**College Name:** Government College of Engineering, Tirunelveli.

Project Name:

Earthquake prediction model using python.

Phase 3 goal:

In this phase, we will be loading and preprocessing the dataset which was given in the previous phase submission.

Overview:

Loading and preprocessing a dataset for earthquake prediction using Python involves a series of steps to prepare your data for machine learning or data analysis tasks. Here's a detailed explanation of each step:

**1.Data Collection:**

The first step is to obtain earthquake data which is the dataset. The dataset is obtained from the previous phase submission and we can also obtain that dataset from the site given below:

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

**Code:**

*import requests*

*response = requests.get('https://www.kaggle.com/datasets/usgs/earthquake-databas e')*

*earthquake\_data = response.json()*

**2.Data Loading:** Once we have the earthquake data, we’ll need to load it into our Python environment. Popular libraries for data manipulation and analysis include Pandas and NumPy. We can use Pandas to create a DataFrame, which is a tabular data structure, to work with our earthquake data.

**Code:**

*import pandas as pd*

1. *Create a DataFrame from the earthquake data*

*earthquake\_df = pd.DataFrame(earthquake\_data['features'])*

**3.Data Exploration:** Before preprocessing, it's essential to explore our dataset to understand its structure and characteristics. Use Pandas and other visualisation libraries like Matplotlib or Seaborn to generate summary statistics, histograms, and plots.

**Code:**

1. *Display basic statistics about the dataset print(earthquake\_df.describe())*
2. *Visualise data*

*import matplotlib.pyplot as plt*

* *Example: Plot earthquake magnitudes plt.hist(earthquake\_df['properties.mag'], bins=20) plt.xlabel('Magnitude')*

*plt.ylabel('Frequency')*

*plt.title('Earthquake Magnitudes')*

*plt.show()*

**4.Data Preprocessing:** Data preprocessing is a crucial step to clean and prepare the dataset for machine learning. It may include the following:

* *Handling Missing Data: Remove or impute missing values, if any.*
* *Feature Selection: Select relevant features for earthquake prediction. Drop unnecessary columns.*
* *Data Scaling and Normalisation: Scale numerical features to have similar ranges. We can use libraries like Scikit-Learn for this.*
* *Categorical Encoding: If our dataset includes categorical variables, encode them using one-hot encoding or label encoding.*
* *Date and Time Features: Extract relevant information from date and time columns, e.g., year, month, day of the week.*
* *Feature Engineering: Create new features that may be useful for prediction, like distance from fault lines or historical earthquake data.*
* *Target Variable: Define and preprocess the target variable, which could be binary (e.g., earthquake occurred or not) or regression (e.g., earthquake magnitude).*

**Here's a simplified example(code):**

*from sklearn.preprocessing import StandardScaler*

* *Drop unnecessary columns*

*earthquake\_df = earthquake\_df.drop(['unnecessary\_column'], axis=1)*

* *Scale numerical features*

*scaler = StandardScaler()*

*earthquake\_df[['feature1', 'feature2']] = scaler.fit\_transform(earthquake\_df[['feature1', 'feature2']])*

**5.Data Splitting:** Divide the dataset into training and testing subsets. This is essential for evaluating the performance of the earthquake prediction model.The commonly used 'train\_test\_split' function from Scikit-Learn can help with this.

**Code:**

*from sklearn.model\_selection import train\_test\_split*

*X = earthquake\_df.drop('target\_column', axis=1)*

*y = earthquake\_df['target\_column']*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

**6.Feature Scaling:** Depending on the machine learning algorithm we plan to use, we may need to apply feature scaling to our dataset. For example, algorithms like Support Vector Machines and k-Nearest Neighbours often benefit from feature scaling.

**Code:**

*from sklearn.preprocessing import StandardScaler scaler = StandardScaler()*

*X\_train = scaler.fit\_transform(X\_train)*

*X\_test = scaler.transform(X\_test)*

**7.Model Building:** After preprocessing, we can proceed to build the earthquake prediction model using machine learning or deep learning techniques. This part of the process involves choosing an appropriate algorithm, training the model on the training data, and evaluating its performance on the testing data.

**Code:**

*from sklearn.ensemble import RandomForestClassifier*

1. *Create and train a model*

*model = RandomForestClassifier()*

*model.fit(X\_train, y\_train)*

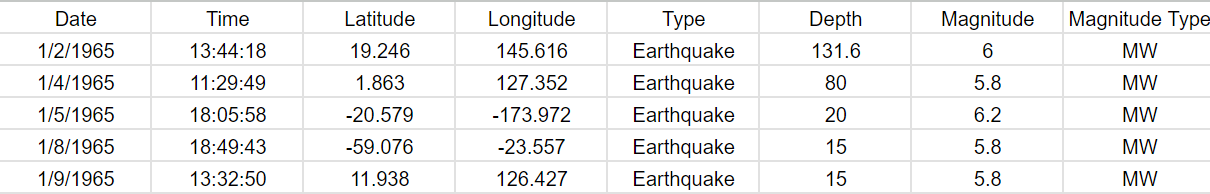
* *Evaluate the model*

*accuracy = model.score(X\_test, y\_test)*

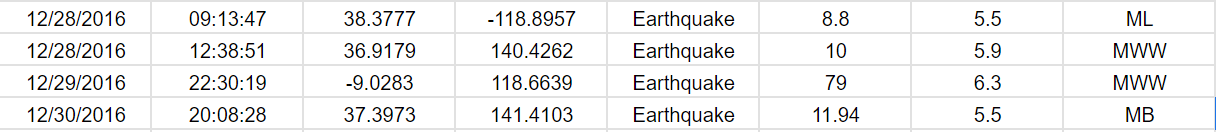
**8.Model Tuning and Evaluation:** Fine-tune the model, optimise hyperparameters, and evaluate its performance using appropriate metrics. This often involves cross-validation and can be an iterative process.

**9.Prediction:** Once the model is trained and evaluated, we can use it to make earthquake predictions on new, unseen data.

**Given Dataset:**



. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .



23,413 rows x 7 Columns

* *For Visualisation and preprocessing the data we will be using the pandas, numpy, matplotlib or seaborn.*
* *First install the above libraries with the help of the command prompt*
* *We have the code for the command prompt as follows:*

1.Open command prompt after installing python. 2.To install the libraries, type the following

-> py -m pip install pandas

-> py -m pip install numpy

-> py -m pip install matplotlib

-> py -m pip install seaborn

**Code:**

*For the code, follow along for the procedures,*

*Additionally, when typing the code at some parts we save the code in a stacking manner but in taking graphs as outputs we cut the ones we already finished.*

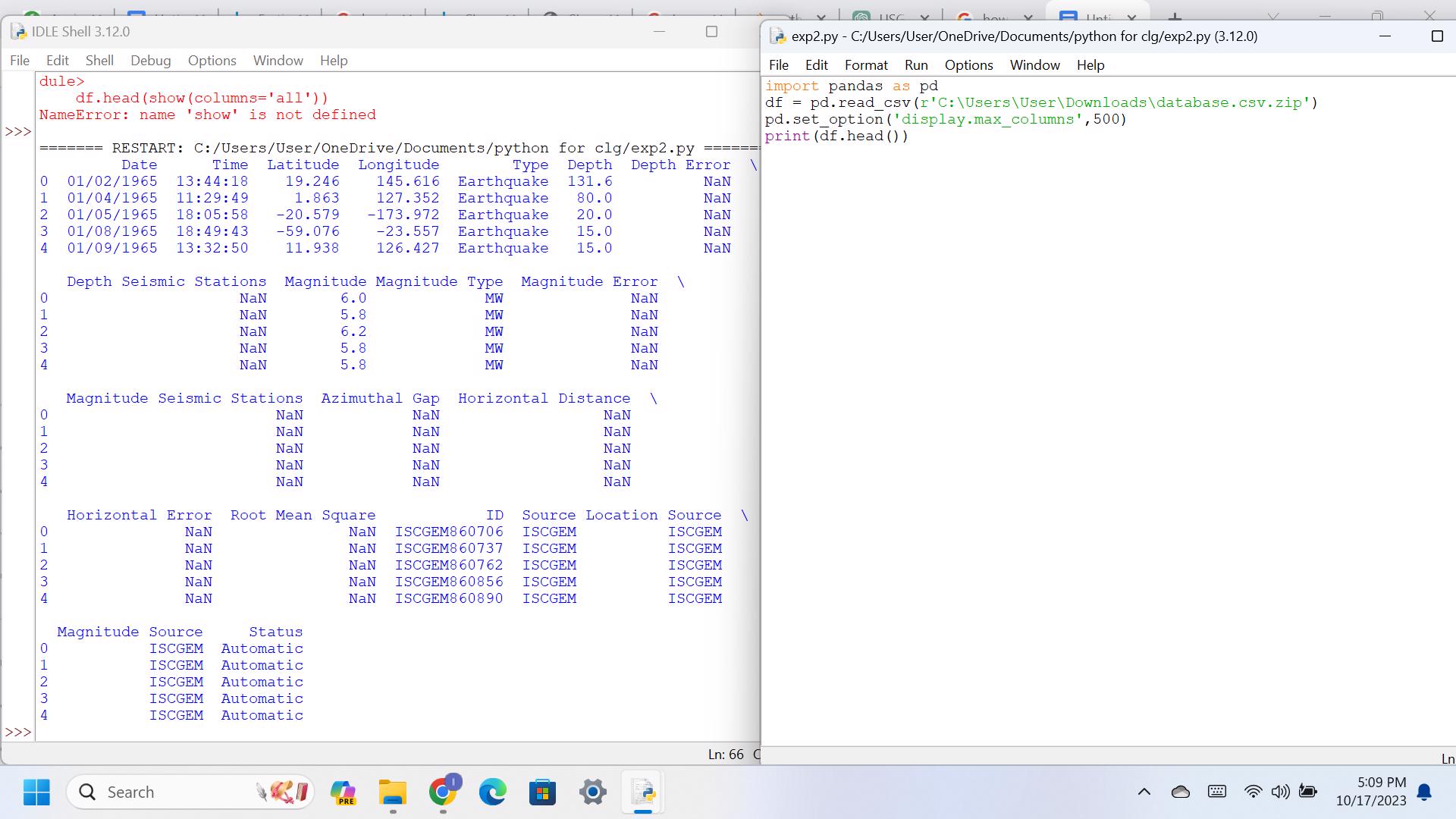
First set of code, /\* start \*/

import pandas as pd

df = pd.read\_csv(r'C:\Users\User\Downloads\database.csv.zip') pd.set\_option('display.max\_columns',500)

print(df.head())

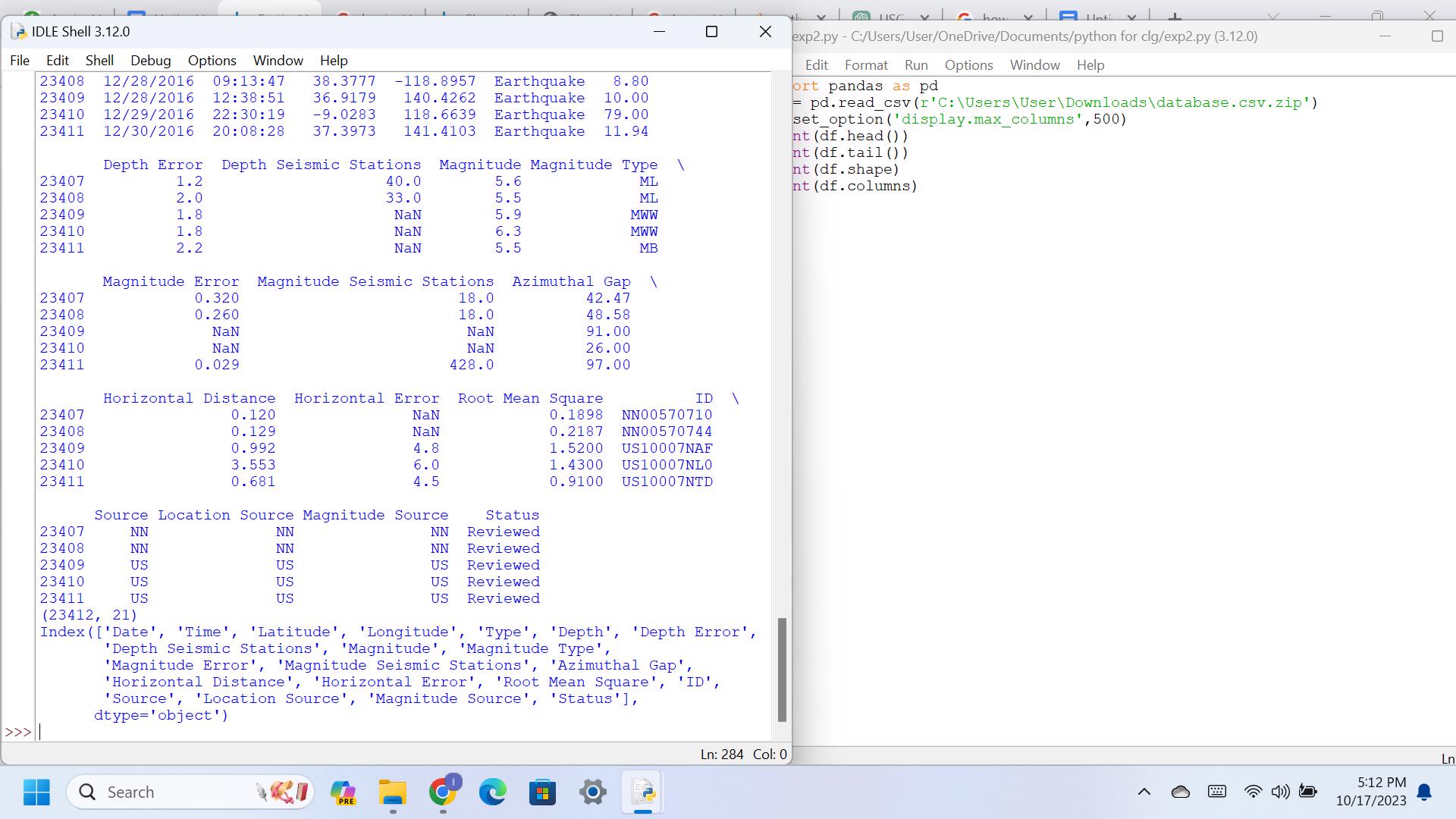
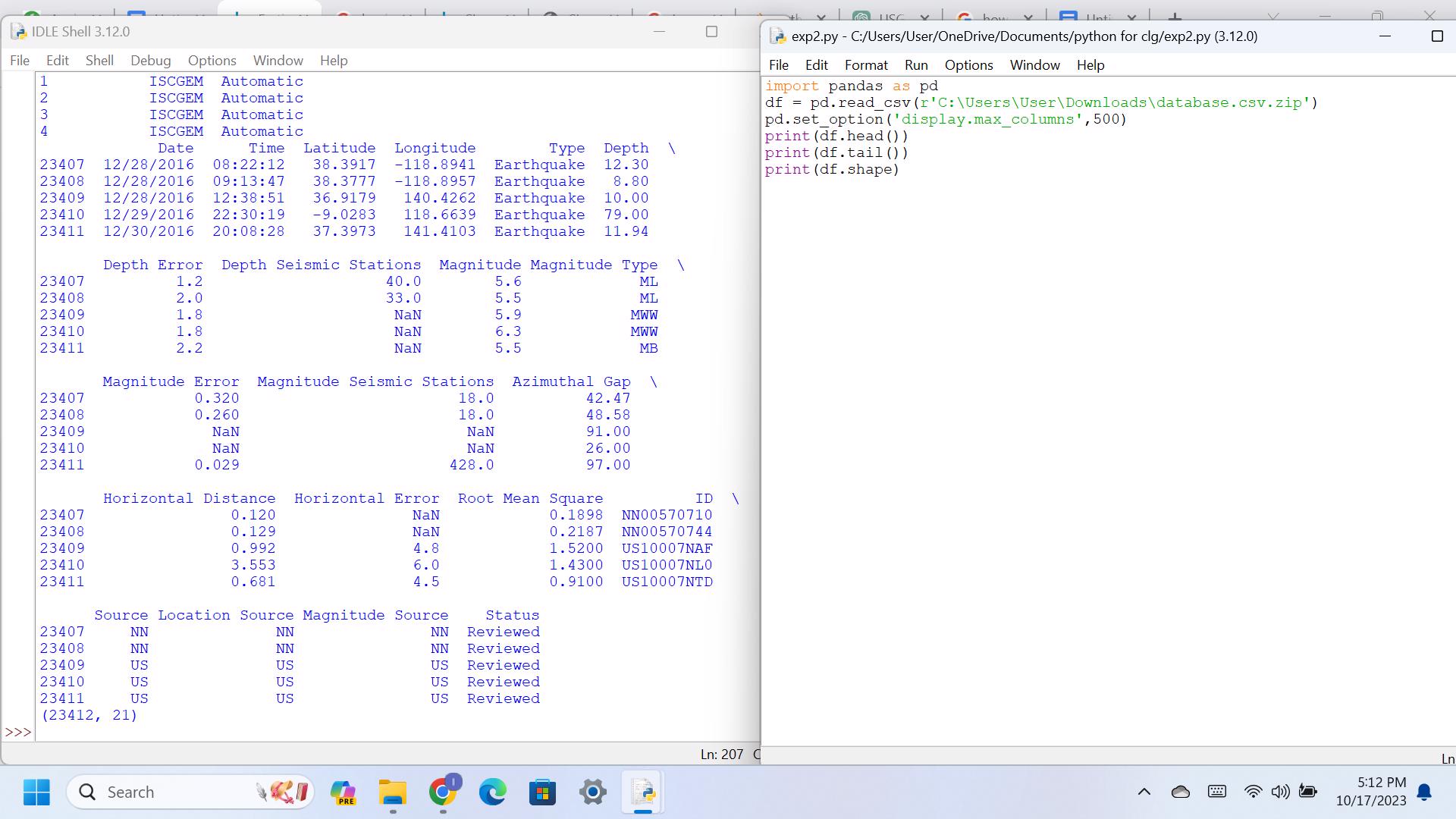
/\* end \*/ Output:



Second set of code, /\* Start \*/ print(df.tail()) print(df.shape) print(df.columns)

/\* end \*/

Output:



Third set of Code, /\* start \*/

print(df.dtypes)

df['Time'] = pd.to\_datetime(df['Time'], infer\_datetime\_format=True, utc=True, errors='ignore')

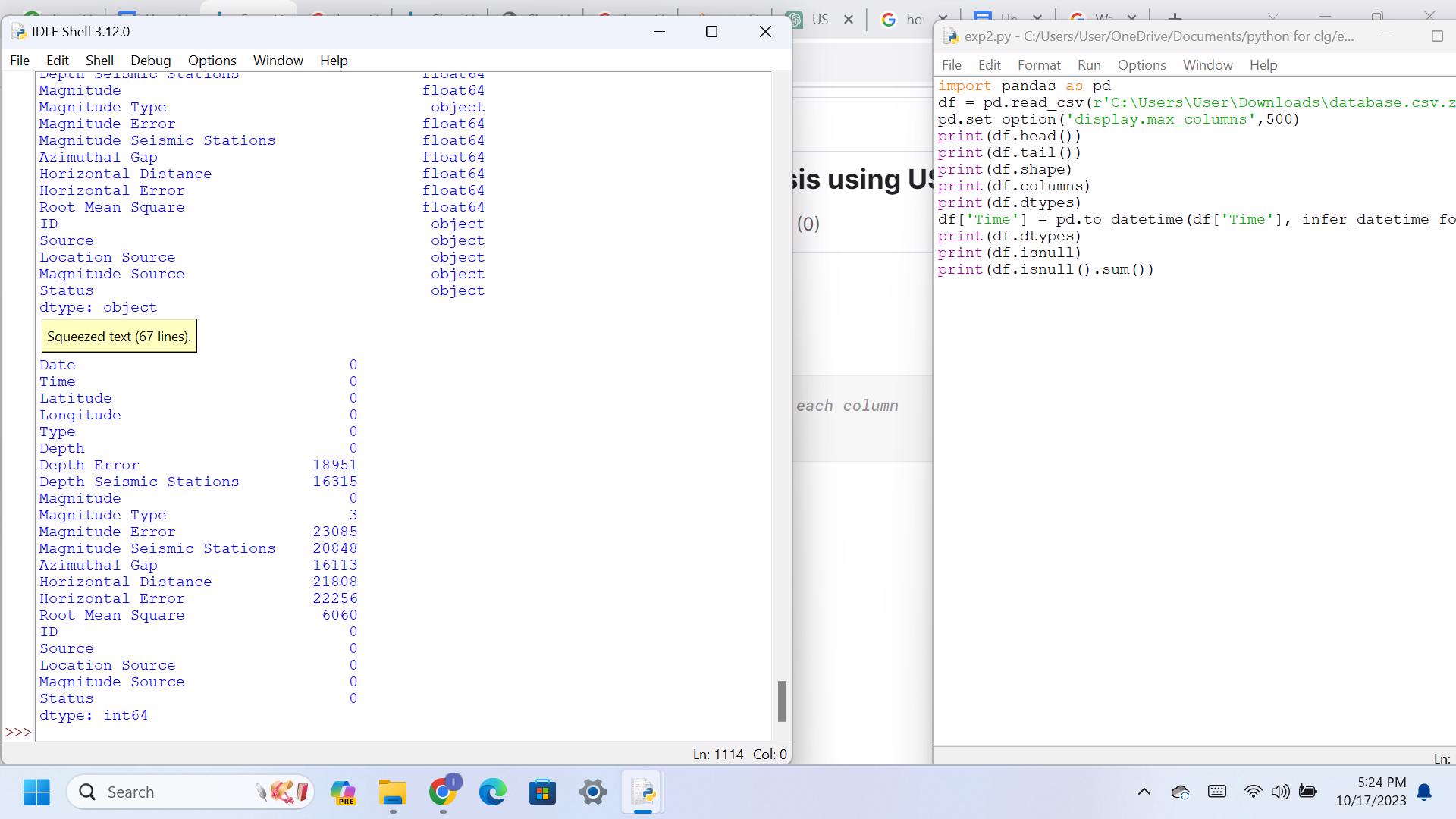
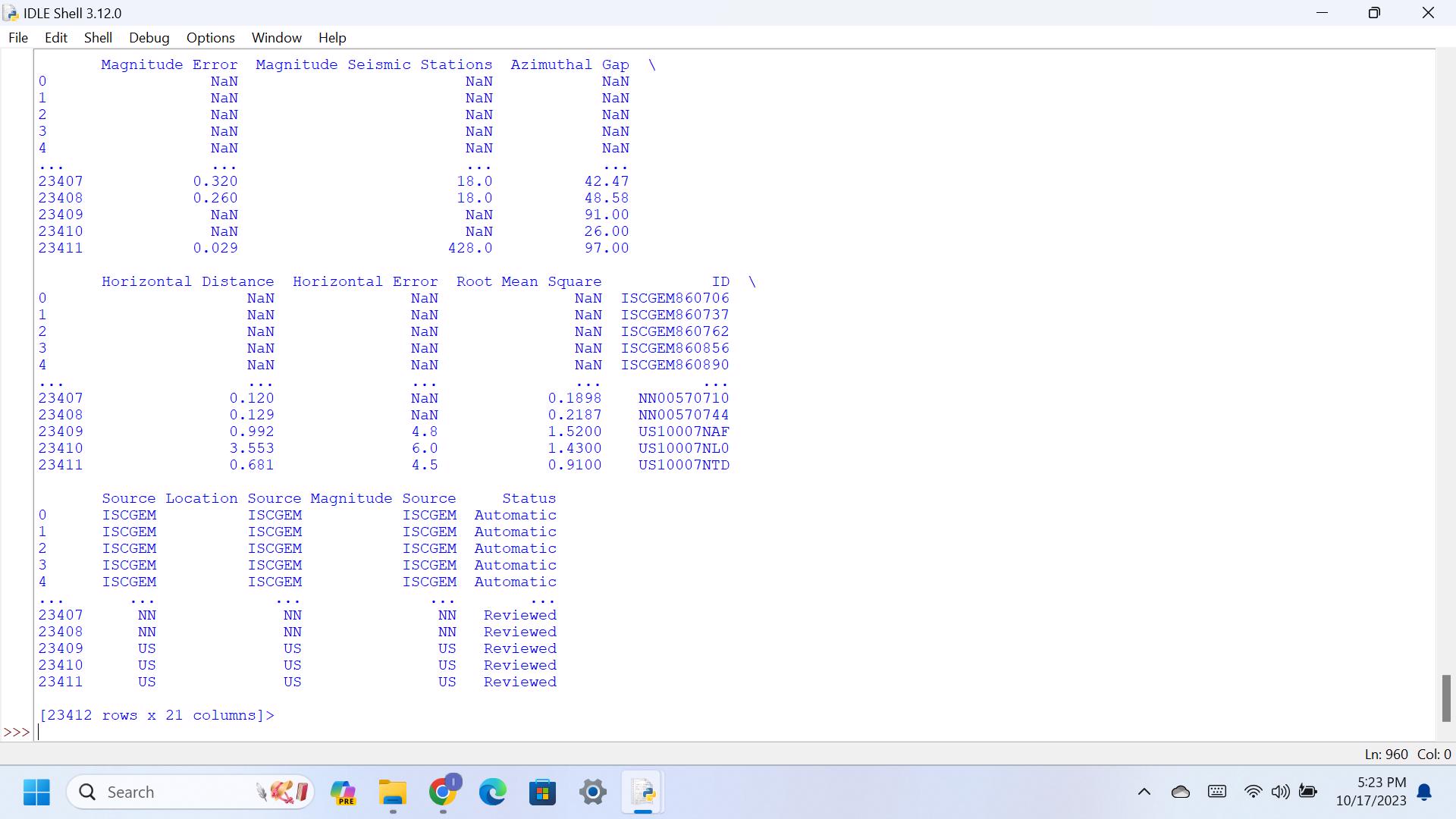
print(df.dtypes)

print(df.isnull)

/\* end \*/

*Here the Time is taken as a string so we convert the format and that is the second line of the above code.*

Output:



*Drop the columns with a large number of missing values as they may not contribute significantly to the analysis.*

Fourth set of Code,

/\* start \*/

if 'Depth Error' in df:

del df['Depth Error']

if 'Depth Seismic Stations' in df:

del df['Depth Seismic Stations']

if 'Magnitude Error' in df:

del df['Magnitude Error']

if 'Magnitude Seismic Stations' in df:

del df['Magnitude Seismic Stations'] if 'Azimuthal Gap' in df:

del df['Azimuthal Gap']

if 'Horizontal Distance' in df:

del df['Horizontal Distance'] if 'Horizontal Error' in df:

del df['Horizontal Error']

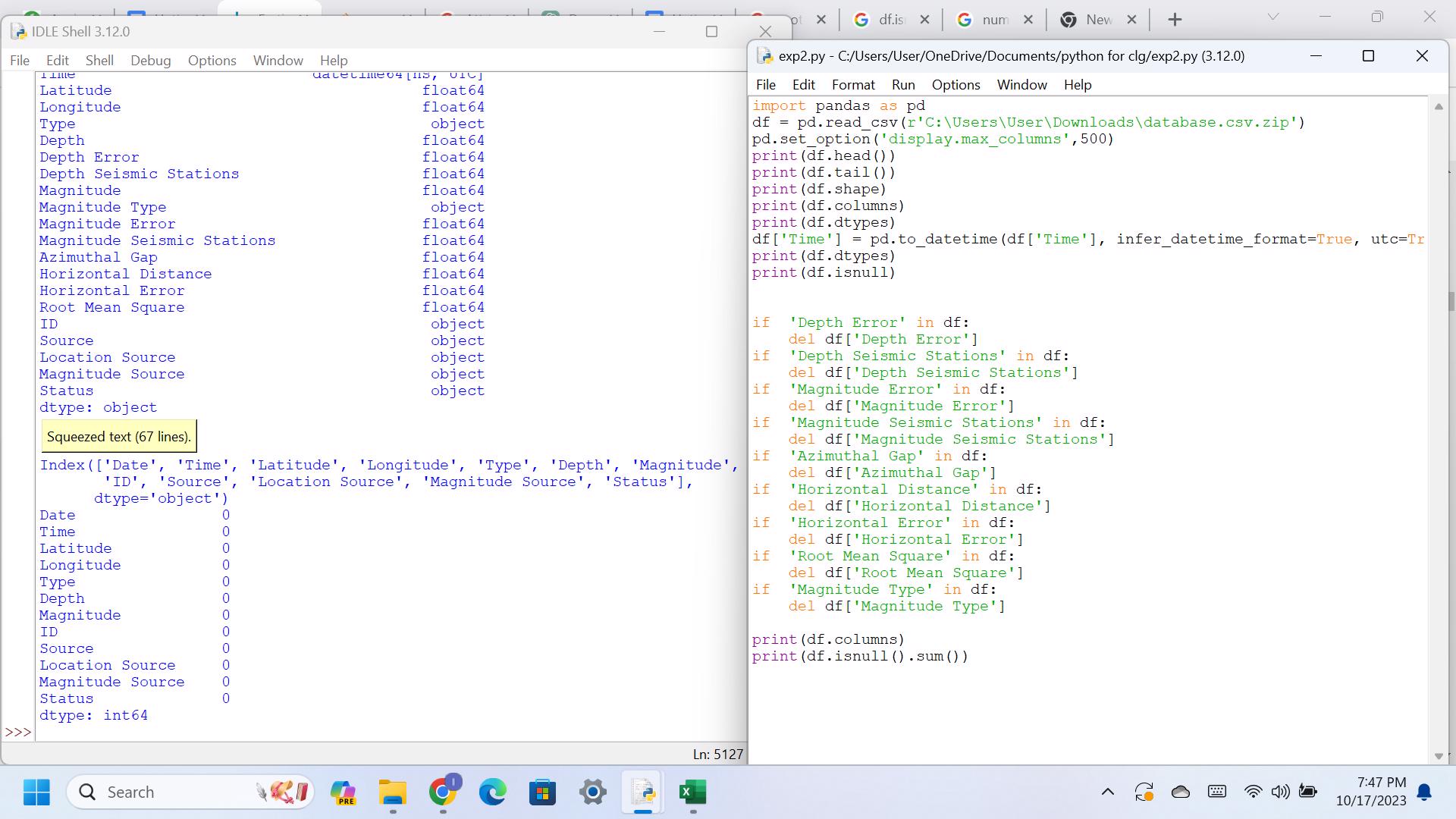
if 'Root Mean Square' in df:

del df['Root Mean Square'] if 'Magnitude Type' in df:

del df['Magnitude Type']

/\* end \*/

Output:



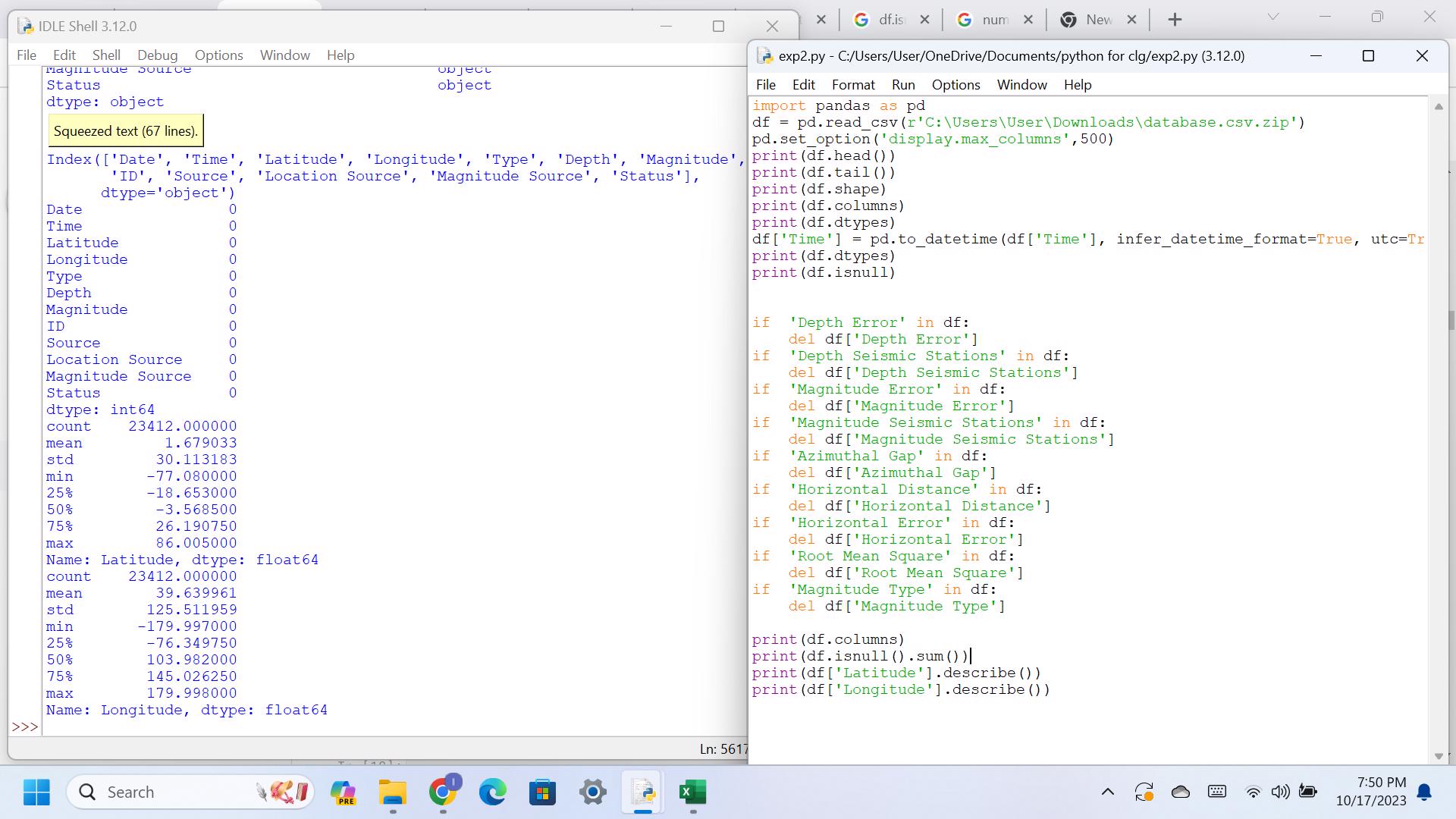
Fifth set of code,

/\* start \*/

print(df.columns) print(df.isnull().sum()) print(df['Latitude'].describe()) print(df['Longitude'].describe())

/\* end \*/

Output:



*Now we can check for exploratory data analysis (EDA) before processing to the analysis step.*

Univariate analysis:

Code:

/\* start \*/

sns.histplot(data=df, x='Magnitude', kde=True) plt.title('Histogram of Magnitude')

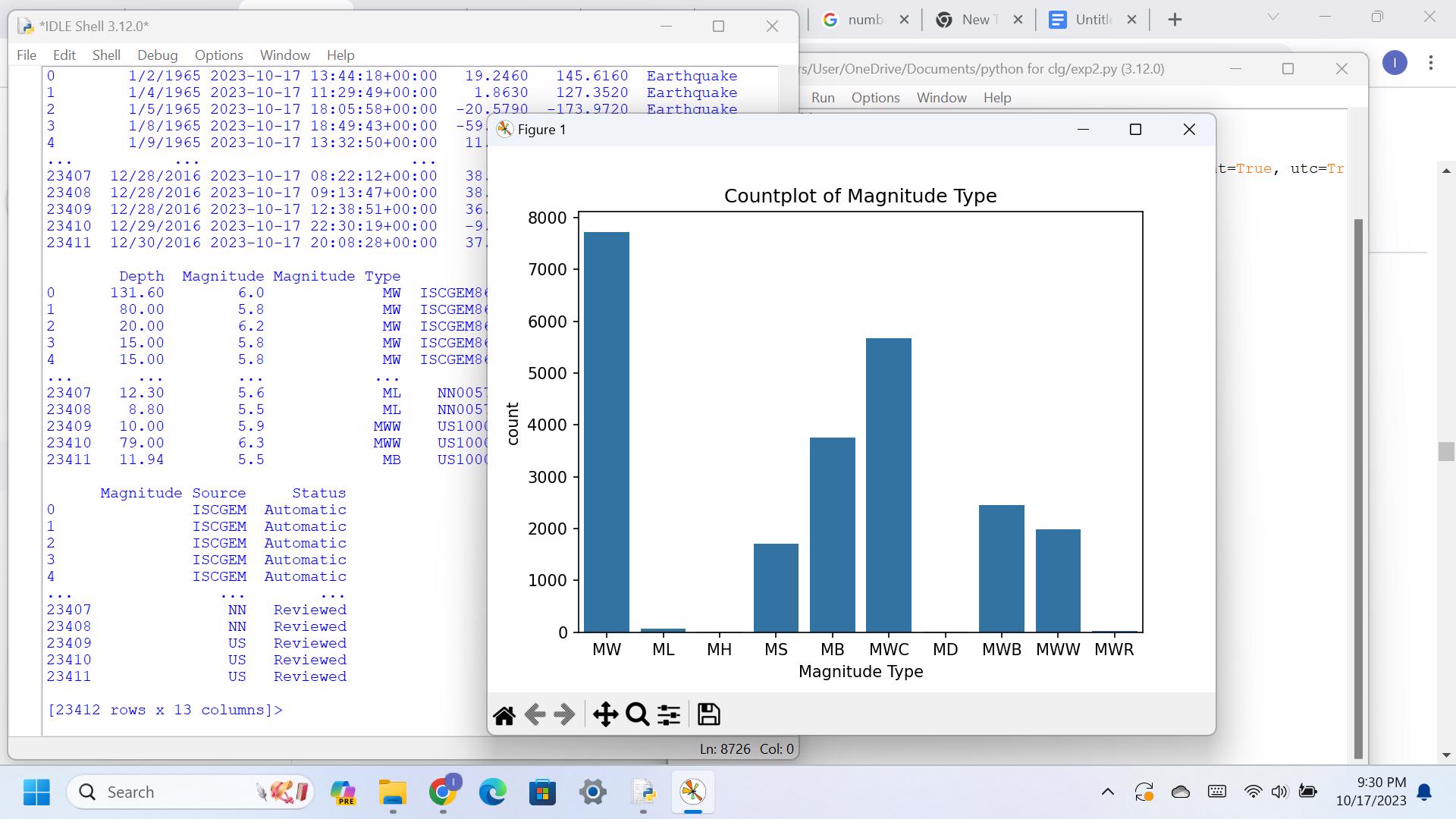
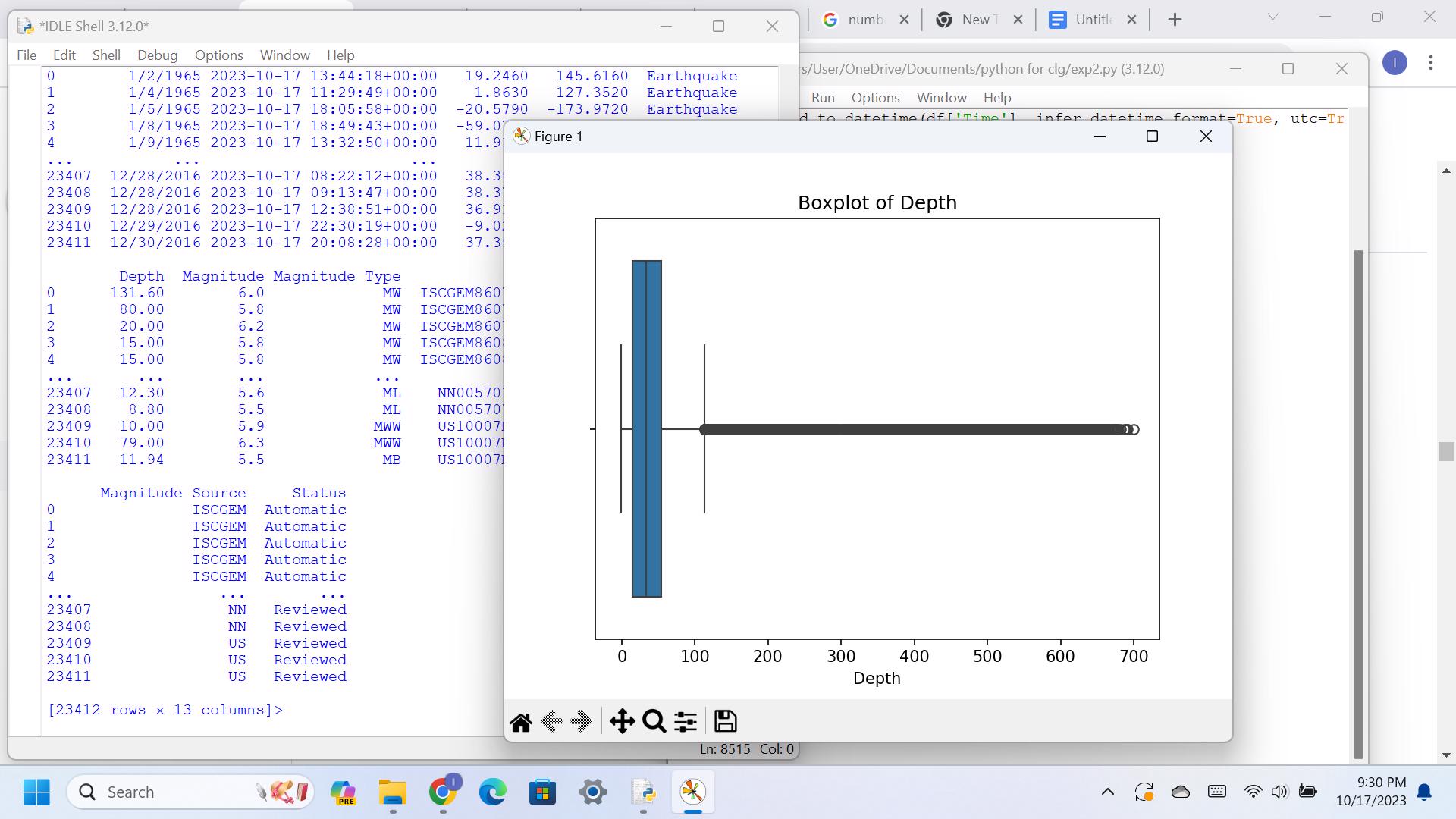
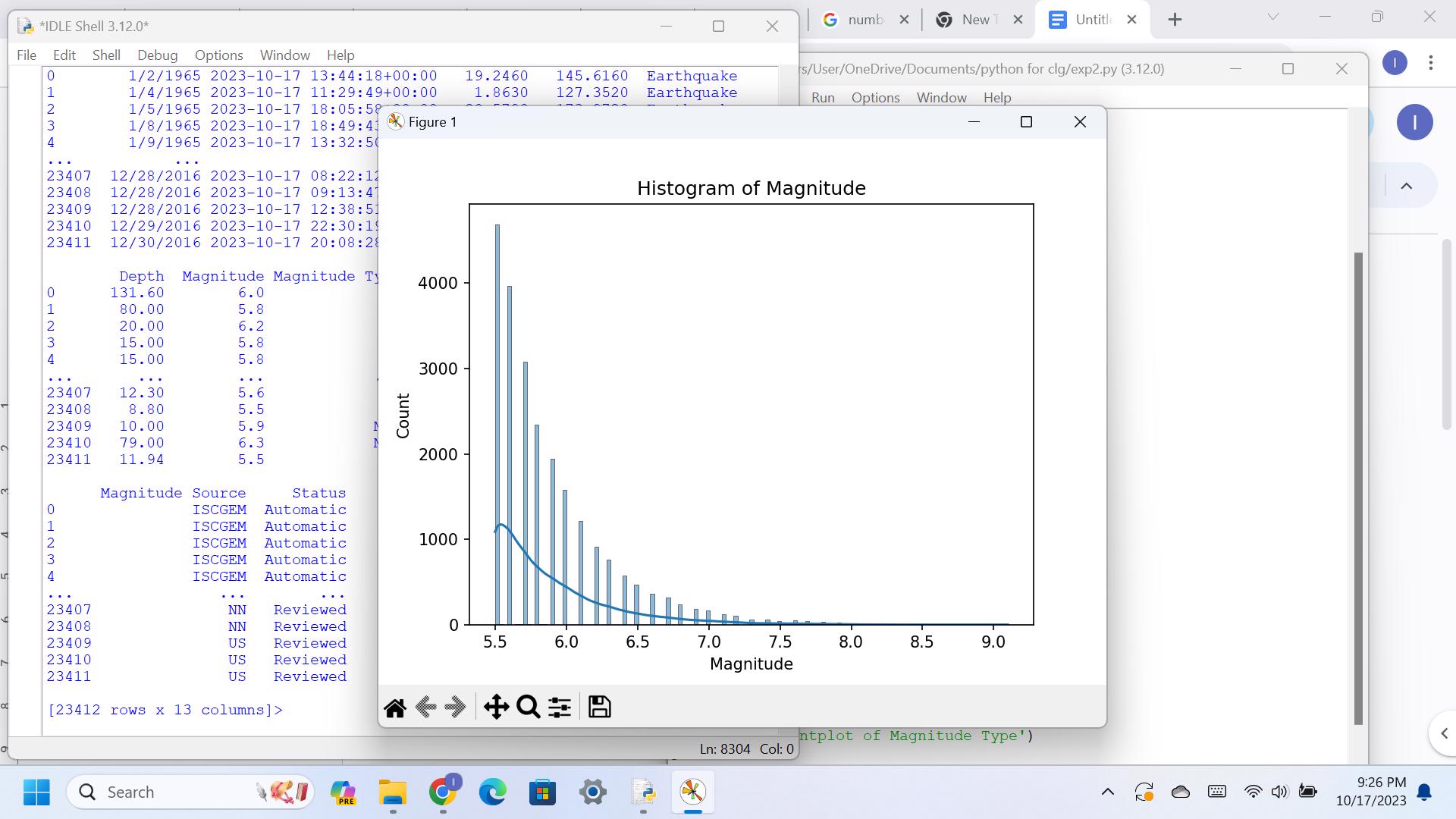
plt.show()

sns.boxplot(data=df, x='Depth')

plt.title('Boxplot of Depth') plt.show()

sns.countplot(data=df, x='Magnitude Type') plt.title('Countplot of Magnitude Type') plt.show()

Output:



Bivariate Analysis:

For bivariate analysis, we can look at the relationship between two variables in the dataset.

Code:

/\*start\*/

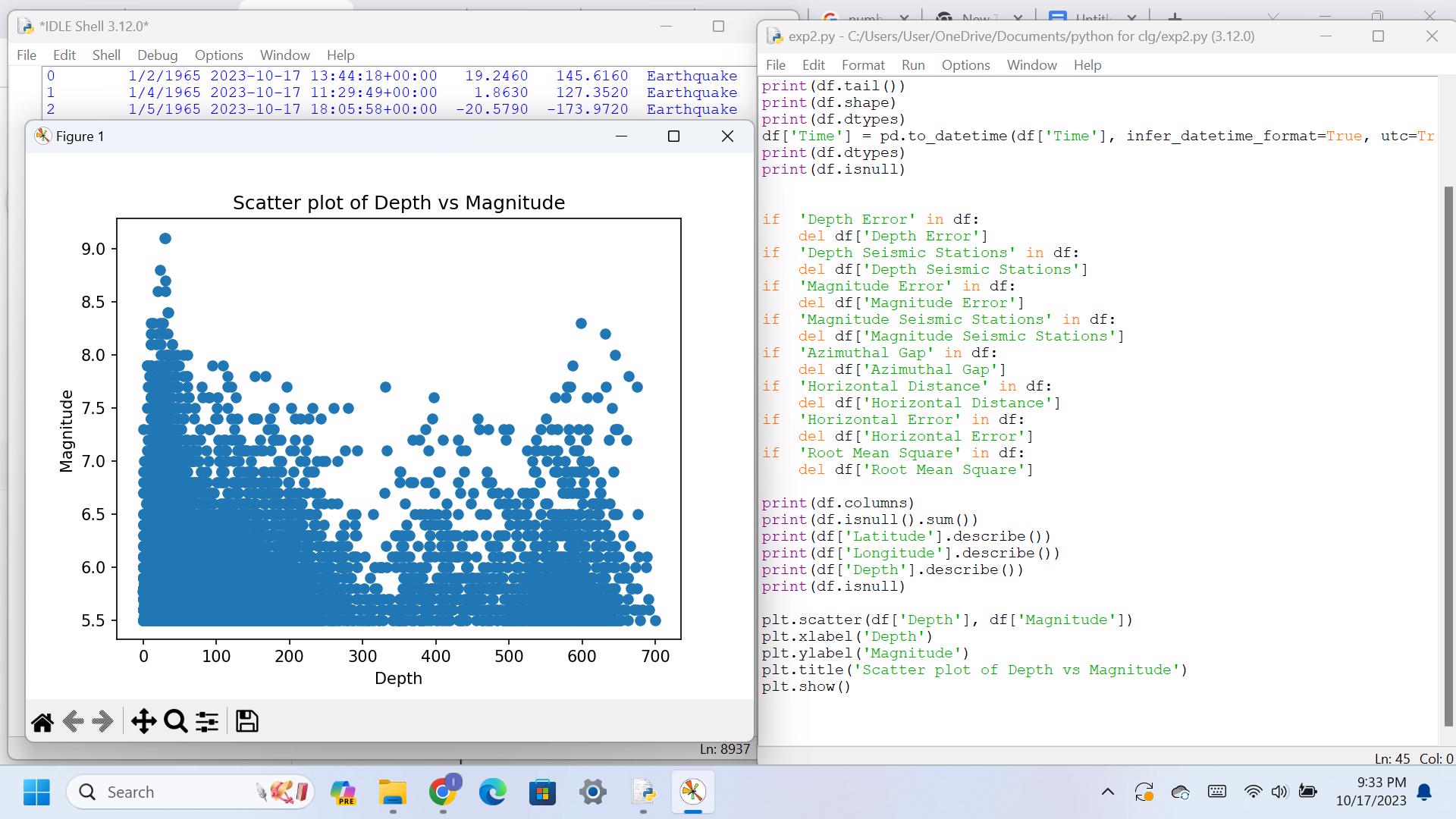
plt.scatter(df['Depth'], df['Magnitude']) plt.xlabel('Depth')

plt.ylabel('Magnitude')

plt.title('Scatter plot of Depth vs Magnitude') plt.show()

/\* end \*/

Output:



Code:

/\*start\*/

df.boxplot(column='Magnitude', by='Magnitude Type')

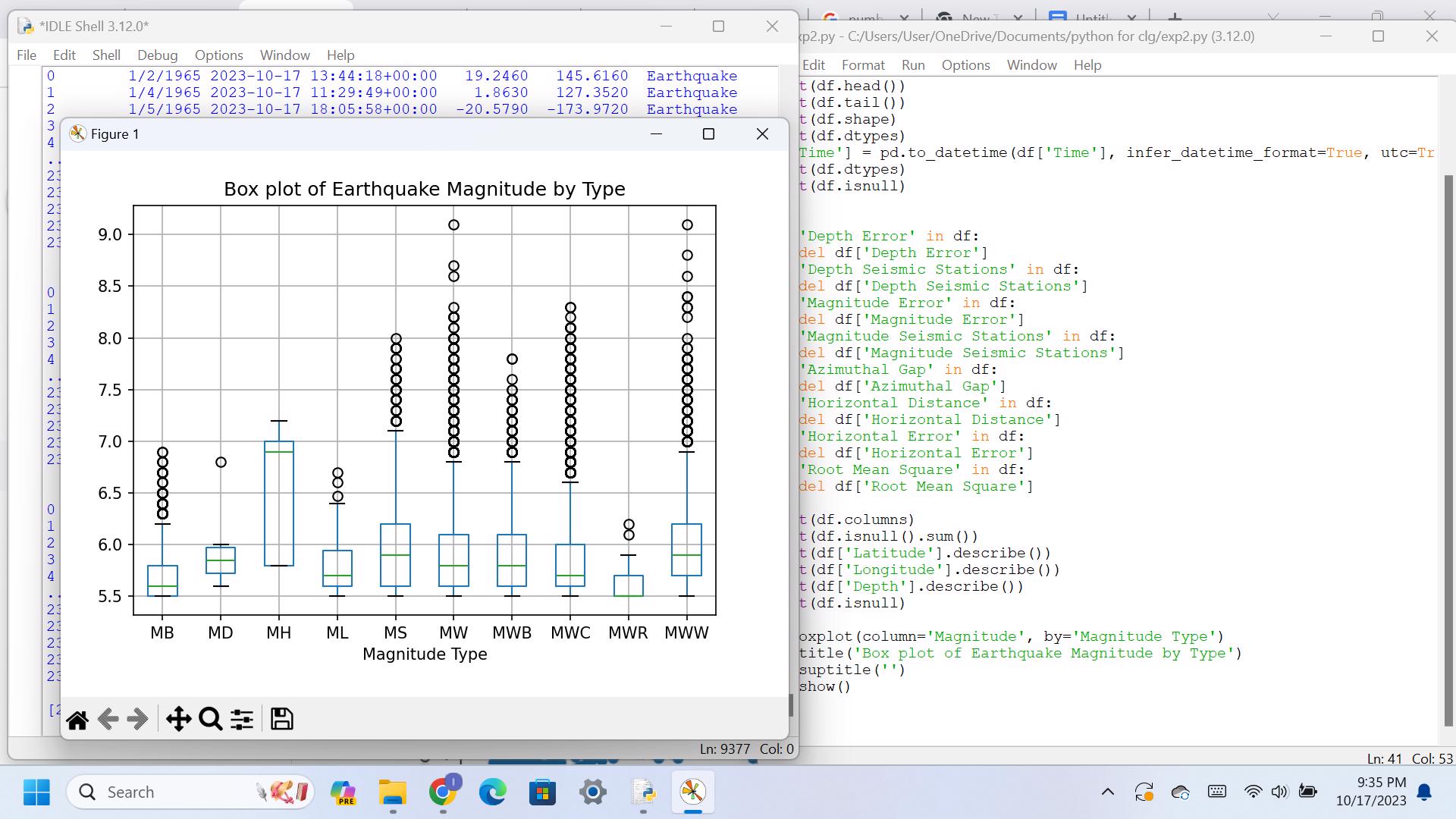
plt.title('Box plot of Earthquake Magnitude by Type')

plt.suptitle('')

plt.show()

/\* end \*/

Output:

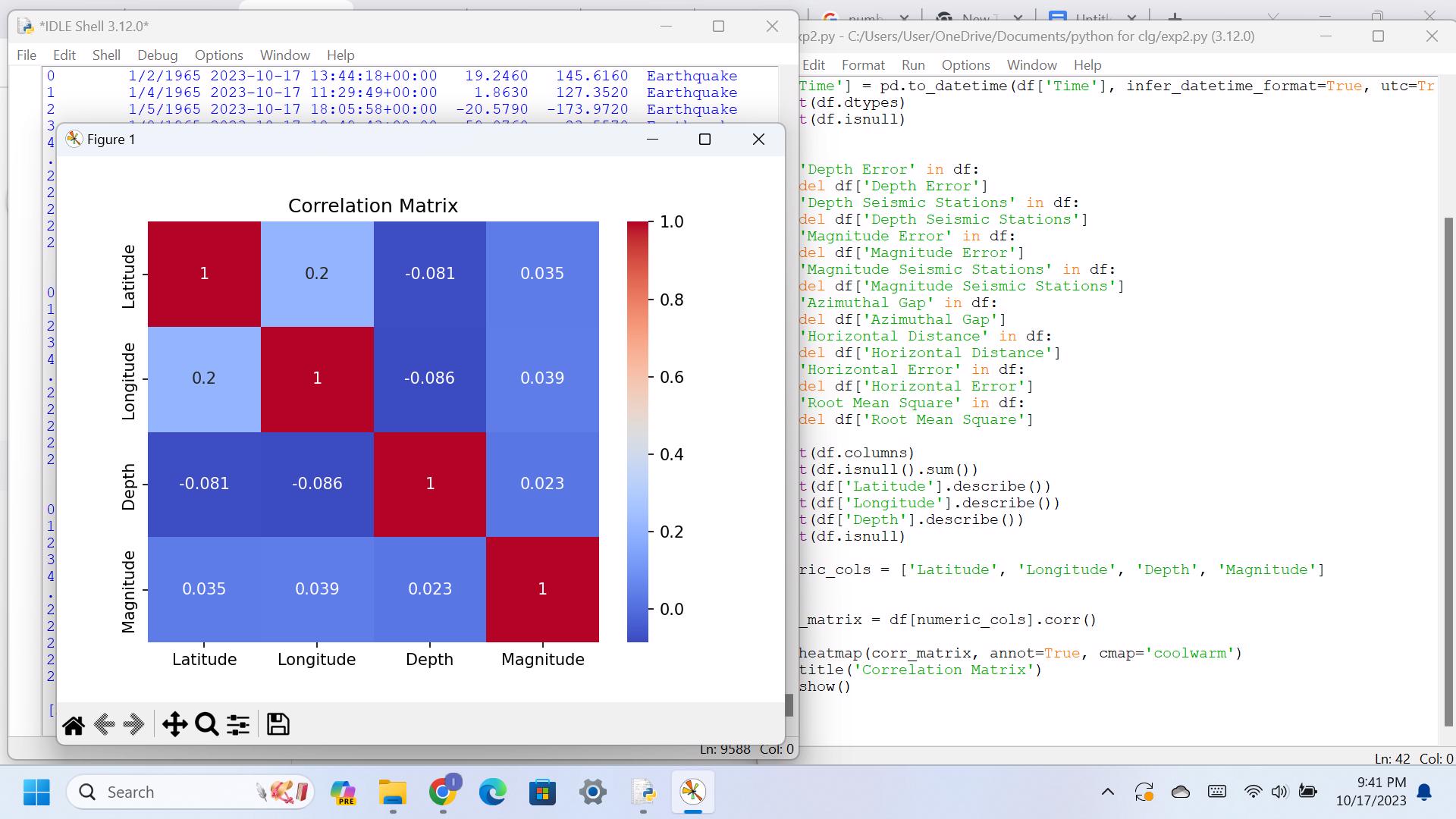


**Multivariate Analysis:**

Code:

/\* start \*/

numeric\_cols = ['Latitude', 'Longitude', 'Depth', 'Magnitude'] corr\_matrix = df[numeric\_cols].corr() sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm') plt.title('Correlation Matrix')



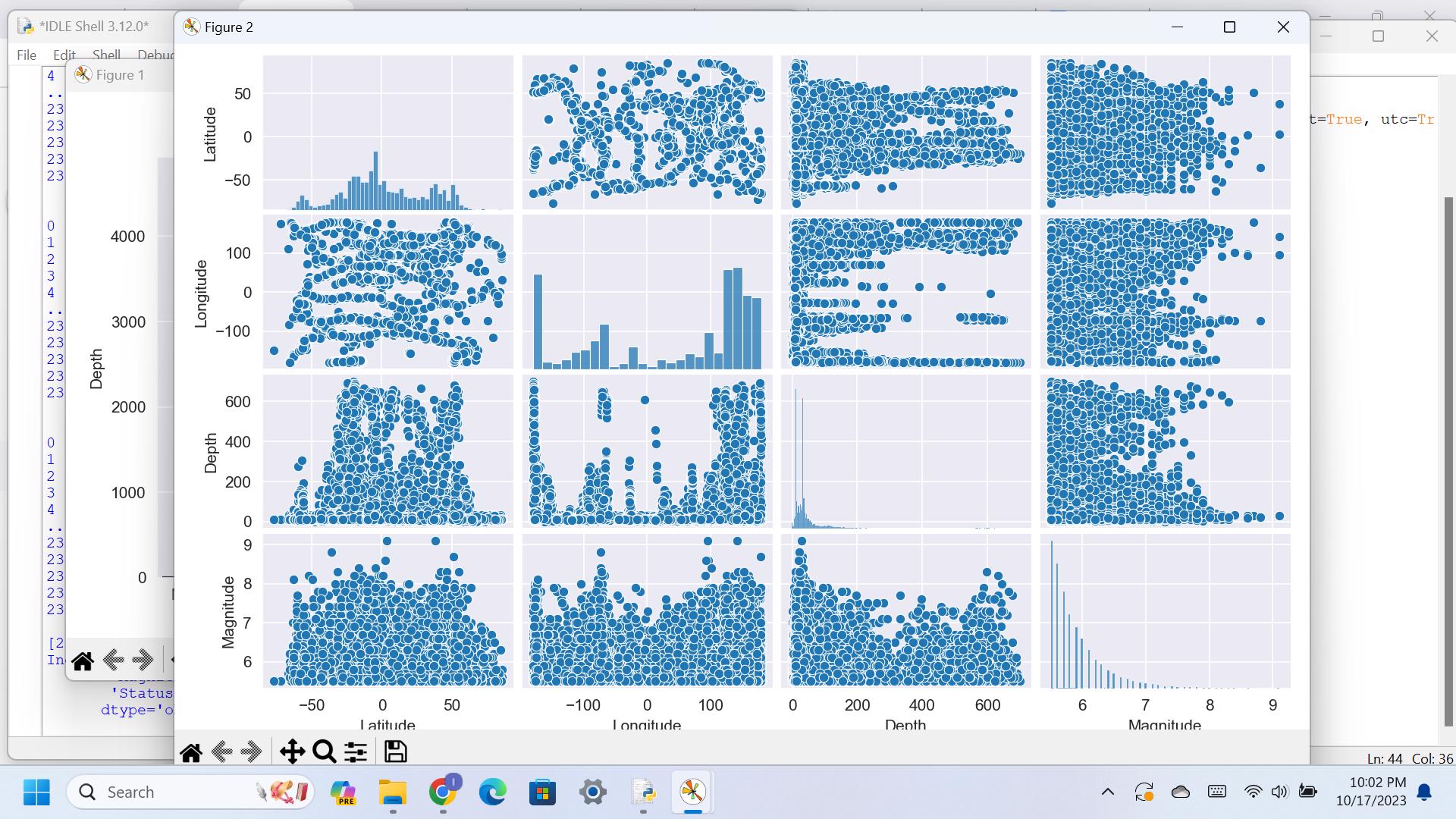
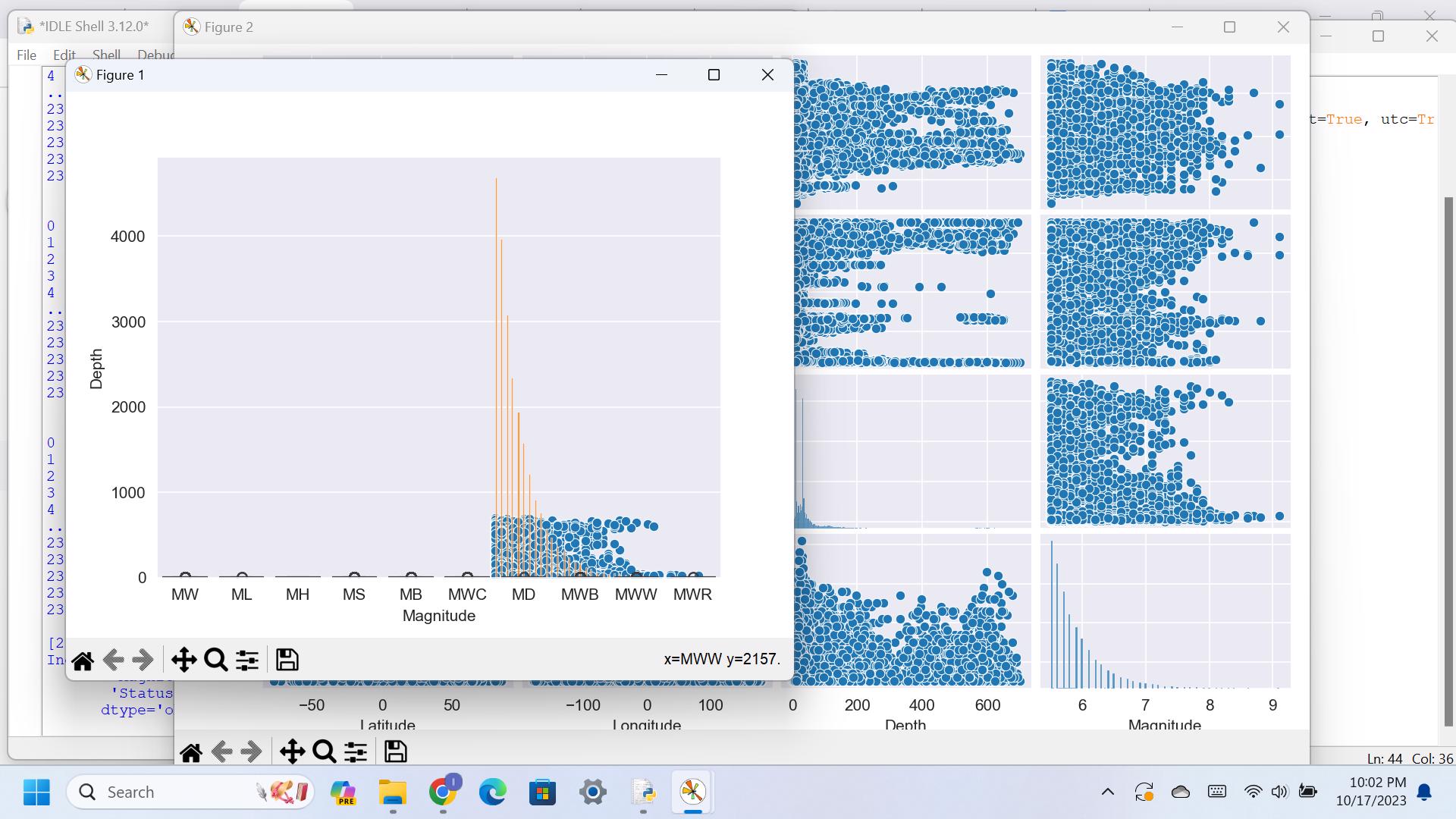
**Data Visualization:**

Code: /\*start\*/

sns.set\_style("darkgrid")

sns.scatterplot(data=df, x="Magnitude", y="Depth") sns.histplot(data=df, x="Magnitude") sns.boxplot(data=df, x="Magnitude Type", y="Magnitude")

sns.pairplot(df)



Earthquake Depth verses Magnitude: Code:

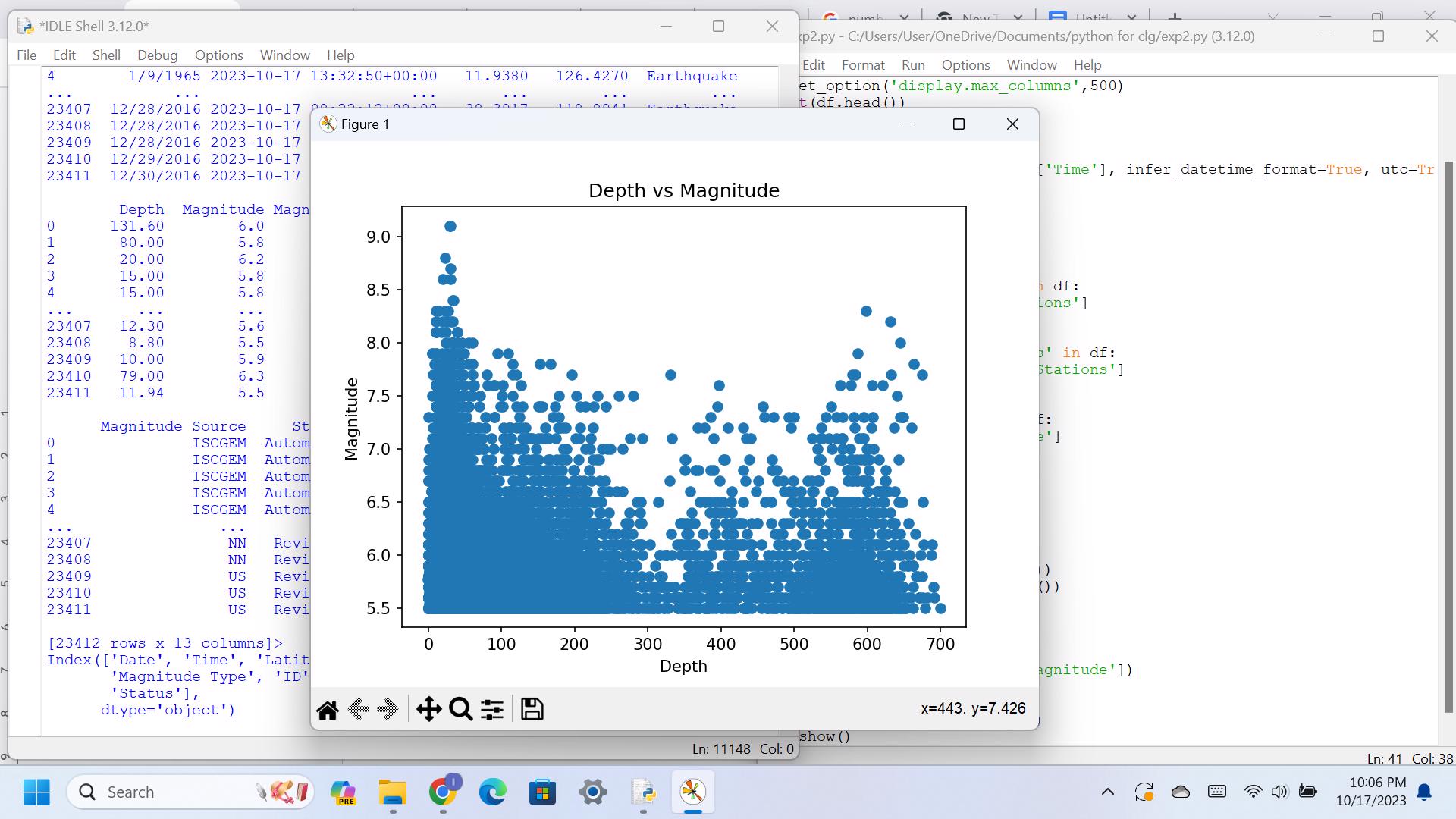
/\* Start \*/

plt.scatter(df['Depth'], df['Magnitude']) plt.xlabel('Depth') plt.ylabel('Magnitude')

plt.title('Depth vs Magnitude') plt.show()

/\*end\*/

Output:



Earthquake Magnitude verses Latitude: Code:

/\* start \*/

plt.scatter(df['Latitude'], df['Magnitude'], alpha=0.2) plt.xlabel('Latitude')

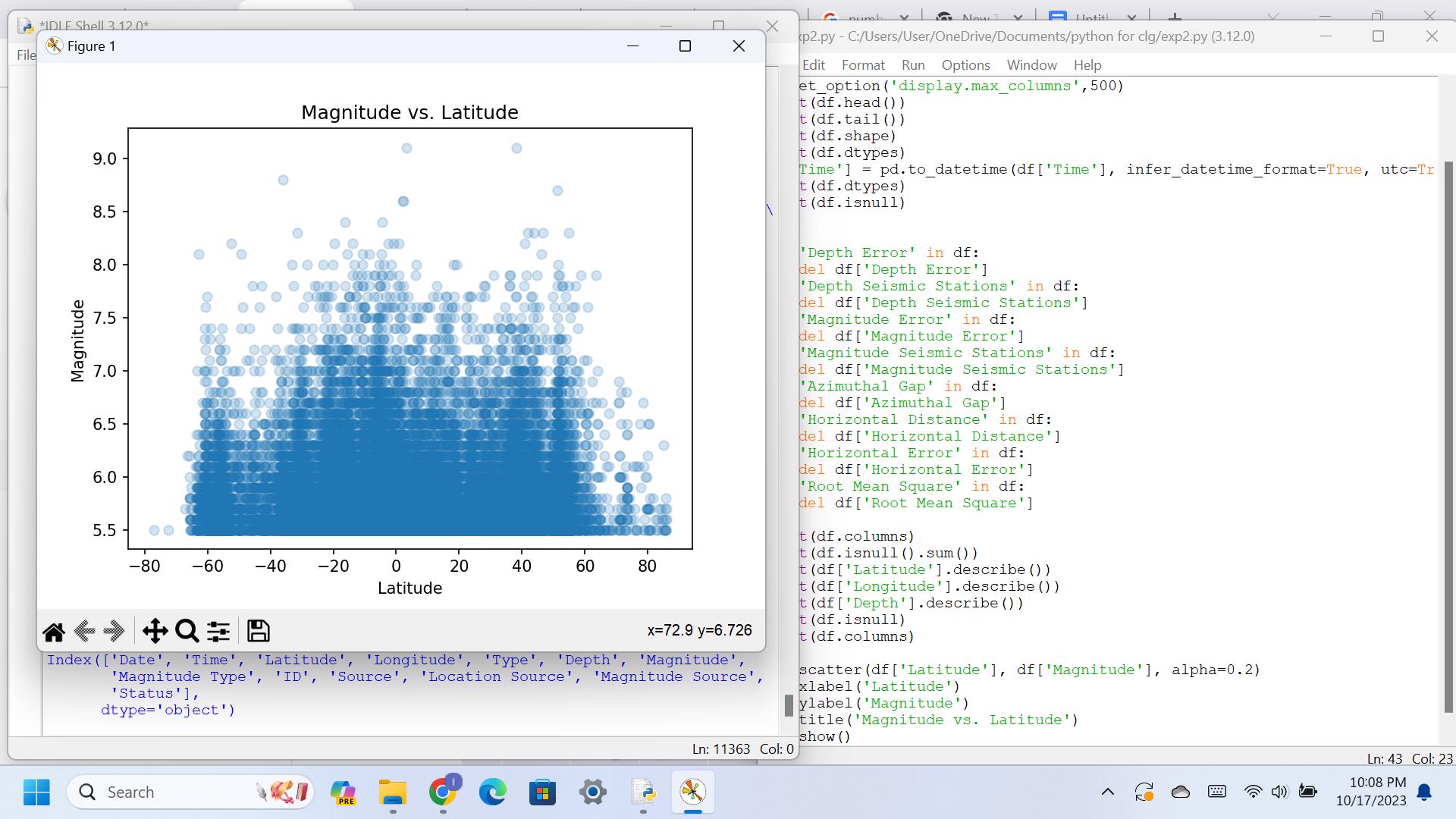
plt.ylabel('Magnitude')

plt.title('Magnitude vs. Latitude')

plt.show()

/\* end \*/

Output:



Earthquake Magnitude verses Longitude: Code:

/\* start \*/

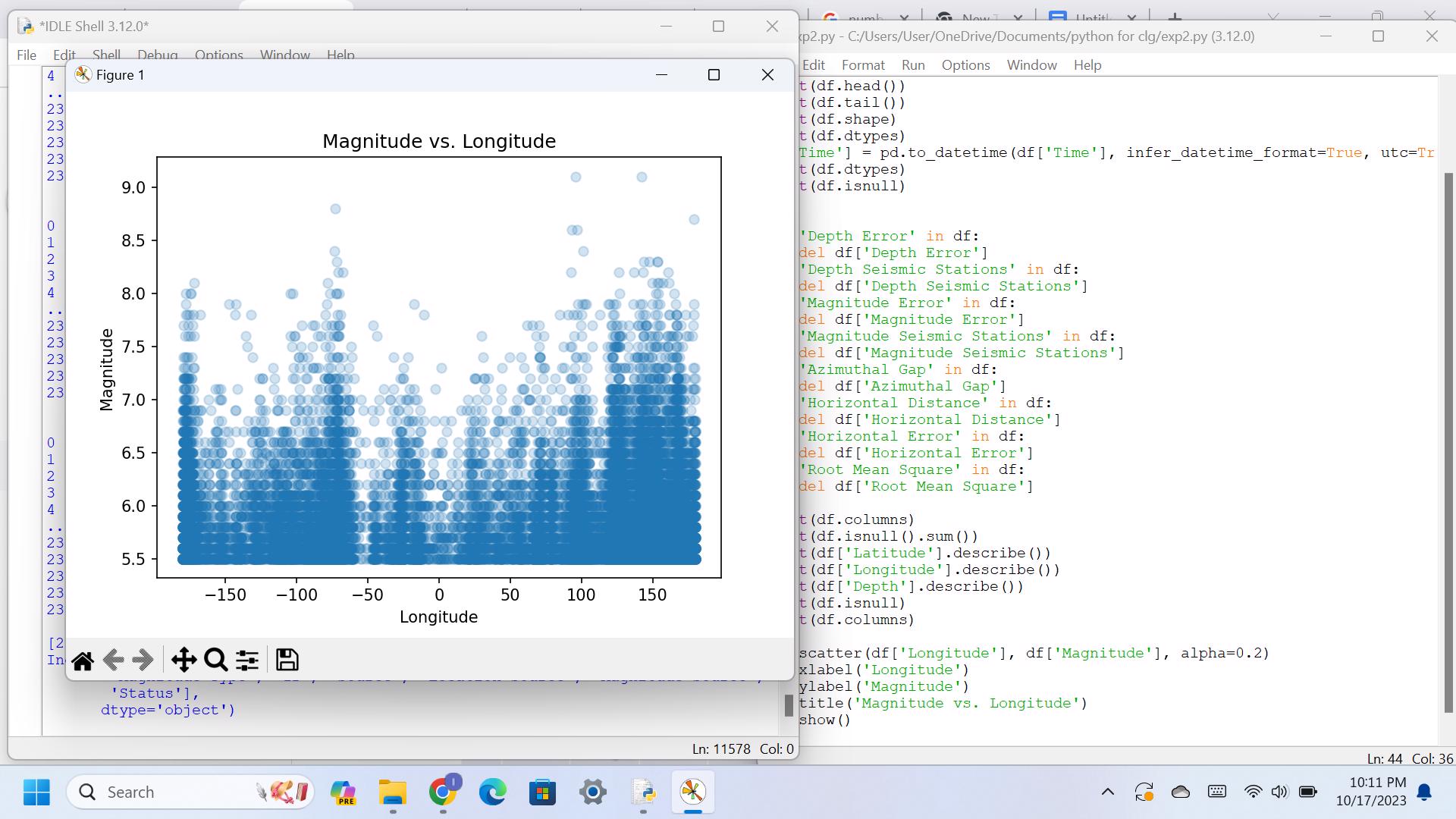
plt.scatter(df['Longitude'], df['Magnitude'], alpha=0.2)

plt.xlabel('Longitude') plt.ylabel('Magnitude')

plt.title('Magnitude vs. Longitude') plt.show()

/\* end \*/

Output:



**Conclusion:**

Based on our analysis of earthquake data collected from the USGS website, we have found several interesting insights.

-> We found that the depth of an earthquake is a major factor that contributes to the occurrence of earthquakes. As the depth of an earthquake decreases, the magnitude of the earthquake tends to increase.

->We also found that there is a relationship between the latitude and longitude of an earthquake and its magnitude. If latitude increases, then the density of magnitude decreases, and if longitude decreases, then the density of magnitude increases.

->Based on our findings, we can recommend that future earthquake prevention and preparation efforts should focus on developing early warning systems that can detect earthquakes at different depths and predicting their magnitudes accurately.

**College Code :** 9508

**College Name:** Government College of Engineering, Tirunelveli.

Project Name:

Earthquake prediction model using python.

Phase 4 goal:

To perform different activities like feature engineering, model training, evaluation as per the instructions in the project.

Overview:

In this phase, we will be performing different activities like feature engineering, model training, evaluation in detail. Here's a detailed explanation of each step:

1.Feature Engineering:

Feature engineering is the process of selecting and transforming the data that will be used as input for the earthquake prediction model. The choice of features is crucial as it directly impacts the model's ability to make accurate predictions.

**a) Data Collection:**

We begin by collecting historical earthquake data from reliable sources like the US Geological Survey (USGS) or other seismic monitoring agencies. This dataset should include information about past earthquakes, such as location, depth, magnitude, and time of occurrence.

**b) Feature Selection:**

Choose relevant features that may have a significant impact on earthquake prediction. Some common features include:

* Latitude and longitude of the earthquake's epicentre.
* Depth of the earthquake.
* Magnitude of the earthquake.
* Time and date of the earthquake.
* Geological features of the region, such as fault lines, plate boundaries, and soil types.
* Historical seismic activity in the region.

**c) Feature Transformation:**

We need to preprocess and transform the selected features to make them suitable for modelling. Common transformations include:

* Scaling features to have a standard range (e.g., using Min-Max scaling or Z-score normalisation).
* Encoding categorical features (e.g., one-hot encoding for regions).
* Extracting additional features from existing ones, such as extracting the day of the week or time of day from the timestamp.

**d) Feature Engineering Techniques:**

Consider more advanced techniques like Principal Component Analysis (PCA) or feature interaction engineering to discover hidden patterns and relationships in the data.

2.Model Training:

Model training involves using a machine learning algorithm to learn patterns and relationships in the data to make predictions. Here are the steps involved:

**a) Data Splitting:**

Divide the dataset into two subsets: training data, and test data. The training data is used to train the model, and the test data is used to evaluate the model's performance.

**b) Selecting an Algorithm:**

Choose an appropriate machine learning algorithm for the earthquake prediction task. Some common choices include:

* Regression models (e.g., linear regression, decision trees, random forests) for predicting earthquake magnitude.
* Classification models (e.g., logistic regression, support vector machines) for predicting earthquake occurrence (binary classification).

**c. Hyperparameter Tuning:**

Optimise the model's hyperparameters using techniques like grid search, random search, or Bayesian optimization. This step helps find the best configuration for the model.

**d. Model Training:**

Train the model on the training data using the selected algorithm and optimised hyperparameters.

3. Evaluation:

After training the model, we need to assess its performance to determine how well it can predict earthquakes. Evaluation involves several steps:

**a) Metrics:** Choose appropriate evaluation metrics based on the specific problem that we are trying to solve. For earthquake prediction, common metrics may include Mean Absolute Error (MAE) for regression tasks (e.g., magnitude prediction) and accuracy, precision, recall, F1-score, or ROC AUC for classification tasks (e.g., earthquake occurrence prediction).

**b) Validation Set Evaluation:**

Assess the model's performance on the validation dataset to ensure it's not overfitting the training data. This step helps us to fine-tune the model further if necessary.

**c) Test Set Evaluation:**

Evaluate the model on the test dataset to get a final estimate of its performance. This simulates how the model will perform on unseen earthquake data.

**d) Visualisation:**

Visualise the model's predictions and compare them to the actual earthquake data. This can help identify patterns and areas for improvement.

**e) Error Analysis:**

Analyse prediction errors to understand where the model struggles and potentially refine the feature engineering or modelling process.

**Code:**

First, we split the data by encoding the categorical variables and splitting the data into training and testing sets.

***Continue off from the code in the previous phase.***

*from sklearn.ensemble import RandomForestRegressor*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import mean\_squared\_error, r2\_score*

*X = df[['Latitude', 'Longitude', 'Depth']]*

*y = df['Magnitude']*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*rf = RandomForestRegressor(n\_estimators=100, random\_state=42)*

*rf.fit(X\_train, y\_train)*

*y\_pred = rf.predict(X\_test)*

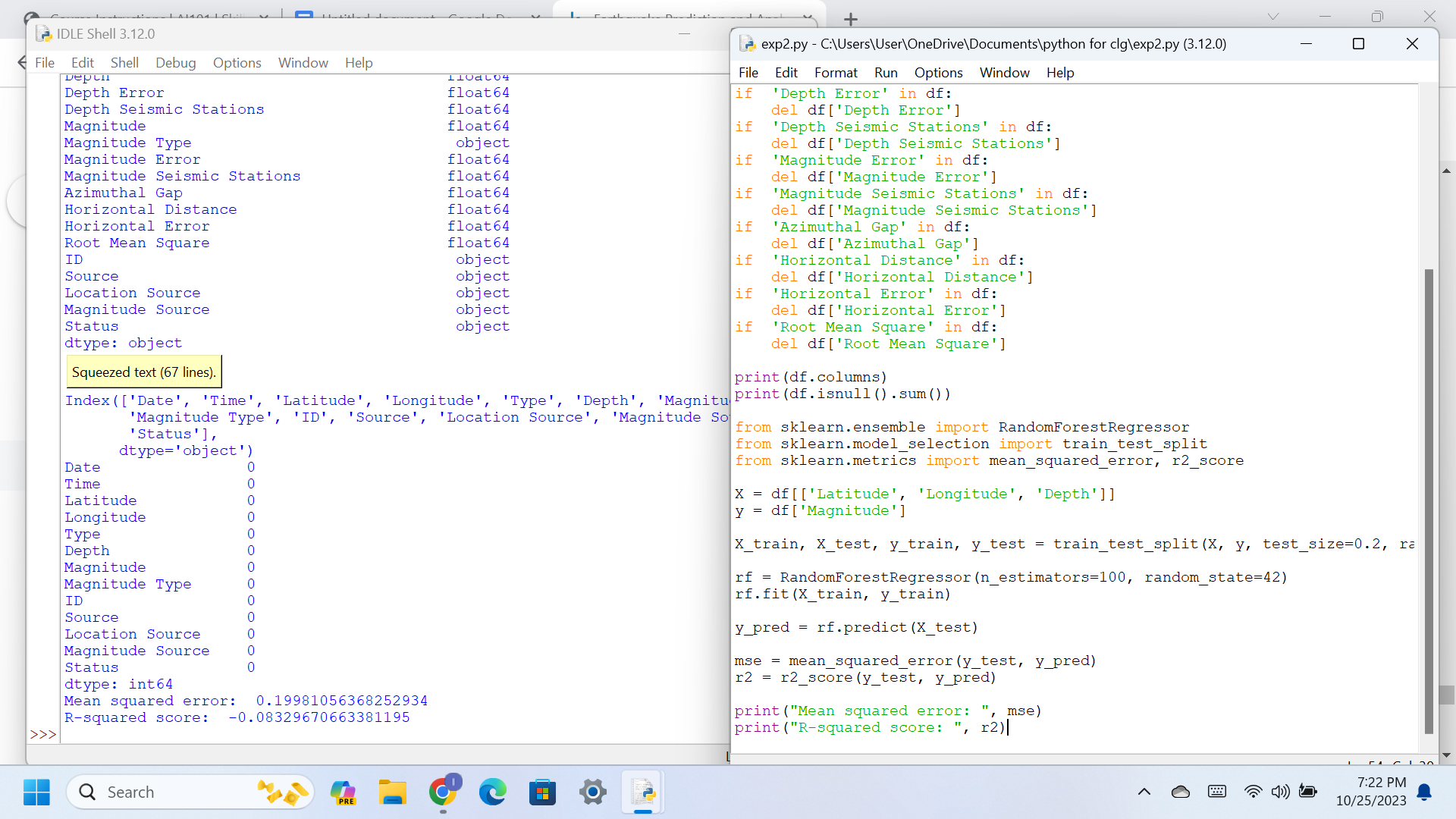
*mse = mean\_squared\_error(y\_test, y\_pred)*

*r2 = r2\_score(y\_test, y\_pred)*

*print("Mean squared error: ", mse)*

*print("R-squared score: ", r2)*

**Output:**



The above is the result of training the model. Now we use the trained model to predict the magnitude of future earthquakes and in this case we give a set of data.

***Now we test the trained model:***

*Latitude = 34.05*

*Longitude = -118.25*

*Depth = 10*

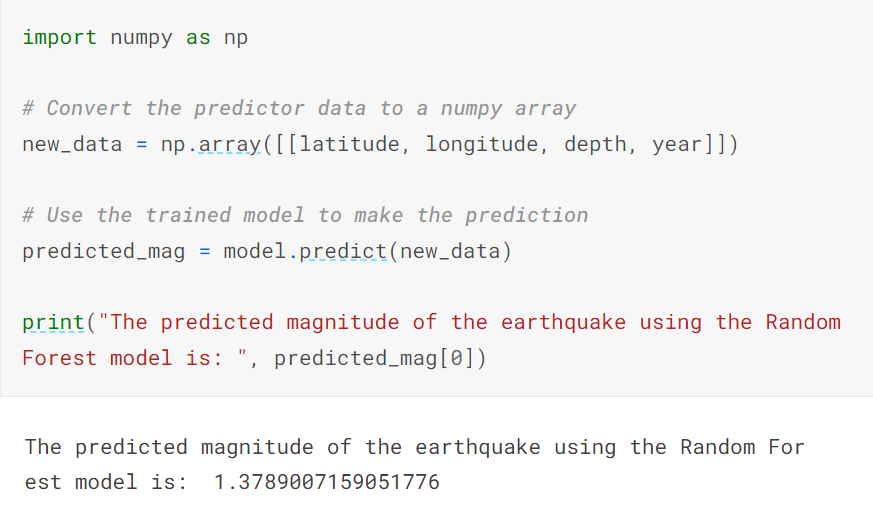
*import numpy as np*

*new\_data = np.array([[Latitude, Longitude, Depth]])*

*predicted\_mag = model.predict(new\_data)*

*print("The predicted magnitude of the earthquake using the Random Forest model is: ", predicted\_mag[0])*

**Output :**

****

For the above code, we install Keras and tensorflow with the help of command prompt.

We also include the following code also before the line where we use the predicted\_mag var to store the model.predict(new\_data)

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

**Conclusion:**

#### The predicted magnitude of the earthquake using the Random Forest model is 1.378900715905184. This means that given the input features of the model, the model predicted the magnitude of the earthquake to be 1.3789. However, it is important to keep in mind that this is just a prediction and there may be various other factors that could influence the actual magnitude of an earthquake.