Earthquake prediction using python phase 5

Introduction

Earthquakes are natural disasters caused by the sudden release of energy in the Earth's crust, leading to seismic waves that can result in significant damage to structures and landscapes. Artificial Intelligence (AI) has emerged as a valuable tool in various aspects of earthquake-related research, prediction, and response. Here's an overview of how AI is contributing to our understanding and management of earthquakes:

1. Early Warning Systems: AI algorithms can analyze real-time seismic data to detect and predict the onset of earthquakes. These systems provide alerts to help mitigate damage and protect lives by giving people a few seconds or minutes of advance notice.

2. Seismic Image Analysis: AI can be used to process and interpret seismic imagery and data, helping seismologists better understand fault lines and earthquake dynamics, which is essential for predicting future events.

3. Building Resilience: AI-driven simulations can assess the structural integrity of buildings and infrastructure in earthquake-prone areas. This aids in designing structures that can withstand seismic forces.

4. Disaster Response: AI and machine learning are used to analyze satellite imagery and drone data to assess earthquake damage quickly. This information assists emergency responders in prioritizing their efforts.

5. Risk Assessment: AI models can calculate earthquake risk by analyzing historical seismic data, fault lines, and geological features, helping in insurance and urban planning.

6. Research and Modeling: AI accelerates the analysis of large datasets and complex models, making it easier for scientists to understand earthquake patterns and improve prediction methods.

In conclusion, AI is revolutionizing the field of earthquake research, prediction, and response by leveraging its ability to process vast amounts of data and identify patterns that were once hard to detect. This integration of AI technology is helping humanity become better prepared for and resilient to earthquakes.

Tools and software used

Earthquake prediction is a complex and challenging field, and it's important to note that reliable short-term earthquake prediction remains elusive. However, there are tools and software used in earthquake monitoring and research. Some of these include:

1. Seismometers: These instruments detect ground motion and are widely used to monitor and record earthquakes.

2. GPS and GNSS (Global Navigation Satellite System) receivers: They can measure the slow movement of tectonic plates, which can provide insights into potential seismic activity.

3. InSAR (Interferometric Synthetic Aperture Radar): This satellite-based technique can measure ground deformation over time, helping to identify areas at risk of earthquakes.

4. Earthquake Early Warning Systems: These systems use real-time seismic data to issue warnings seconds to minutes before strong shaking from an earthquake reaches a location. Examples include the ShakeAlert system in California.

5. Numerical Modeling Software: Researchers use software to simulate and model earthquake scenarios, helping to understand fault behavior and potential earthquake impacts.

6. GIS (Geographic Information Systems): GIS software is used to map and analyze earthquake data, as well as assess earthquake hazards and vulnerabilities.

7. Seismic Hazard Assessment Software: These tools calculate the probability of earthquakes of various magnitudes occurring in a specific region, often used in building code development and land-use planning.

8. Machine Learning and AI: These technologies are increasingly being used to analyze large datasets for earthquake prediction and early warning.

9. Open-source data platforms: Initiatives like the USGS Earthquake Hazards Program provide data and tools for earthquake research.

It's important to emphasize that while these tools and software are essential for earthquake monitoring and research, earthquake prediction, especially in terms of precise timing and location, remains a challenging scientific problem with limited success to date. Most efforts are focused on earthquake preparedness and early warning systems to mitigate the impact of earthquakes.

Problem Definition and Design Thinking

Problem Statement: Develop a model to estimate the probability of an earthquake occurring in a specific region within a certain time frame based on historical seismic data.

Key Objectives:
1. Predict the likelihood of an earthquake.
2. Provide information about the affected area and potential magnitude.
3. Continuously update and improve the model with new data.

Constraints and Considerations:
1. Earthquake prediction is inherently uncertain, and the goal is to provide probabilistic forecasts rather than precise predictions.
2. Availability of historical seismic data for model training.
3. Need for real-time data processing to update the model.

Design Thinking Process

1. **Empathize**:

- Understand the needs and concerns of stakeholders, such as government agencies, emergency responders, and the public.
- Gather insights from seismologists, data scientists, and domain experts.

2. **Define**:
- Clearly define the problem statement, objectives, and constraints.
- Set specific performance metrics (e.g., accuracy, precision, recall) for model evaluation.

3. **Ideate**:
- Brainstorm potential features and data sources that can be used for earthquake prediction.
- Explore different machine learning algorithms suitable for the problem.

4. **Prototype**:
- Develop a proof-of-concept model using Python and relevant libraries (e.g., scikit-learn, TensorFlow, PyTorch).
- Collect and preprocess historical seismic data for model training.
- Experiment with various machine learning algorithms, such as logistic regression, decision trees, or deep learning models.
- Implement real-time data streaming and processing for continuous updates.

5. **Test**:
- Evaluate the prototype model using appropriate metrics and cross-validation techniques.
- Gather feedback from experts and stakeholders to refine the model.

6. **Iterate**:
- Continuously refine the model based on feedback and new data.
- Explore advanced techniques like time-series analysis, anomaly detection, and ensemble methods.

7. **Implement**:
- Deploy the model in a production environment, considering scalability and performance.
- Develop a user-friendly interface or dashboard for stakeholders to access predictions.

8. **Monitor**:
- Establish a monitoring system to track the model's performance and provide timely updates.
- Continuously update the model with new seismic data.

9. **Feedback**:
- Gather feedback from users and stakeholders to make improvements over time.
- Stay informed about the latest research and advancements in earthquake prediction.

10. **Scale and Collaborate**:
- Collaborate with relevant authorities, researchers, and organizations to expand the model's coverage and effectiveness.
- Work on scaling the model's deployment to cover larger regions.

It's important to manage expectations when working on earthquake prediction models, as the science of earthquake prediction is still evolving, and precise predictions remain extremely challenging. Emphasize the importance of preparedness, early warning systems, and community education alongside prediction efforts.

Design into Innovation

1. **Data Collection**:
   - Gather earthquake-related data, including seismic, geospatial, and geological information.
   - Historical earthquake data and relevant features are crucial.

2. **Data Preprocessing**:
   - Handle missing values and outliers.
   - Normalize or standardize the data.
   - Split the dataset into training, validation, and test sets.

3. **Feature Engineering**:
   - Extract relevant features such as seismic activity, geological characteristics, and historical earthquake patterns.

4. **Ensemble Methods**:
   - Utilize ensemble techniques like Random Forest, Gradient Boosting, or AdaBoost for robust prediction.
   - Train multiple models and combine their predictions for improved accuracy.

5. **Deep Learning Architectures**:
   - Consider using deep neural networks, such as Convolutional Neural Networks (CNNs) for image data, or Recurrent Neural Networks (RNNs) for time series data.
   - Customize network architectures based on the nature of your data.

6. **Hyperparameter Tuning**:
   - Fine-tune hyperparameters using techniques like grid search or random search to optimize model performance.

7. **Evaluation Metrics**:
   - Use appropriate evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or custom metrics for earthquake prediction.

8. **Model Validation**:
   - Validate the models on the validation set to ensure they generalize well.

9. **Model Testing**:
   - Test the models on the test set to assess their real-world performance.

10. **Post-Processing**:
    - Apply post-processing techniques to further refine predictions, if needed.

11. **Visualization**:
    - Create visualizations to better understand the data and model predictions.

12. **Deployment**:
    - Deploy the model in a production environment for real-time earthquake prediction.

13. **Continual Learning**:
   - Continually update the model with new data and adapt it to changing patterns in seismic activity.

Remember that earthquake prediction is extremely challenging, and even the best models may have limitations. Collaboration with experts in the field is essential, and safety measures should always be in place regardless of predictive models.

Building loading and preprocessing the data set

Data Collection: Gather relevant data such as seismic wave patterns, geological information, historical earthquake occurrences, etc. This data can be sourced from geological surveys and seismic monitoring stations.
Preprocessing: Preprocess the data to remove noise and transform it into a usable format. This may include normalization, feature extraction, and filling or removing missing values.
Feature Engineering: Identify significant features or variables that are likely to have an impact on earthquake prediction. This may include the velocity of seismic waves, the depth of seismic activity, and the location of faults.
Model Selection: Select appropriate machine learning models for prediction. Common algorithms used in earthquake prediction include Support Vector Machines (SVM), Neural Networks, Random Forests, and Gradient Boosting.
Training: Split the data into training and testing sets, then train the selected model on the training data. Hyperparameter tuning and cross-validation can help in selecting the best model.
Prediction: Use the trained model to predict the likelihood of an earthquake in the given area. The output can be a binary classification (earthquake/no earthquake) or a continuous value representing the probability.
Evaluation: Evaluate the model using metrics like accuracy, precision, recall, and F1 score to assess its effectiveness in predicting earthquakes.
Deployment: If the model proves effective, it can be deployed in real-time systems to provide warnings and help in disaster preparedness.
Continuous Monitoring and Updating: Continuously monitor and update the model with new data to ensure its effectiveness in predicting earthquakes.
It’s worth mentioning that predicting earthquakes with high accuracy is still a challenging task. Machine learning models can provide insights and

predictions, but they must be used with caution and in conjunction with other scientific methods.

Since earthquake prediction is a critical application, it's essential to follow best practices in email communication when sharing this information. Ensure that the content is clear, concise, and well-formatted to minimize the likelihood of the email being classified as spam. Utilize trusted email servers and maintain a clean mailing list to further reduce the risk of your emails ending up in the spam folder.
Development

2.1 Dataset
In [2]:

```
data = pd.read_csv('../content.-Earthquake-Prediction/train.csv',
dtype={'acoustic_data': np.int16, 'time_to_failure': np.float64})
def calc_change_rate(x):
    change = (np.diff(x) / x[:-1]).values
    change = change[np.nonzero(change)[0]]
    change = change[~np.isnan(change)]
    change = change[change != -np.inf]
    change = change[change != np.inf]
    return np.mean(change)


def add_trend_feature(arr, abs_values=False):
    idx = np.array(range(len(arr)))
    if abs_values:
        arr = np.abs(arr)
    lr = LinearRegression()
    lr.fit(idx.reshape(-1, 1), arr)
    return lr.coef_[0]


def featuresx(x_data, X, segment):
```

```
    X.loc[segment, 'ave'] = x_data.mean()
    X.loc[segment, 'std'] = x_data.std()
    X.loc[segment, 'max'] = x_data.max()
    X.loc[segment, 'min'] = x_data.min()

    X.loc[segment, 'q01'] = np.quantile(x_data,0.01)
    X.loc[segment, 'q05'] = np.quantile(x_data,0.05)
    X.loc[segment, 'q95'] = np.quantile(x_data,0.95)
    X.loc[segment, 'q99'] = np.quantile(x_data,0.99)

    X.loc[segment, 'mean_change_abs'] = np.mean(np.diff(x_data))
    X.loc[segment, 'mean_change_rate'] = calc_change_rate(x_data)
    X.loc[segment, 'abs_max'] = np.abs(x_data).max()
    X.loc[segment, 'abs_min'] = np.abs(x_data).min()


    X.loc[segment, 'max_to_min'] = x_data.max() / np.abs(x_data.min())
    X.loc[segment, 'max_to_min_diff'] = x_data.max() - np.abs(x_data.min())
    X.loc[segment, 'count_big'] = len(x_data[np.abs(x_data) > 500])
    X.loc[segment, 'sum'] = x_data.sum()


    X.loc[segment, 'abs_trend'] = add_trend_feature(x_data,
abs_values=True)
    X.loc[segment, 'abs_mean'] = np.abs(x_data).mean()
    X.loc[segment, 'abs_std'] = np.abs(x_data).std()
    X.loc[segment, 'abs_median'] = np.median(np.abs(x_data))


    X.loc[segment, 'trend'] = add_trend_feature(x_data)
    X.loc[segment, 'mad'] = x_data.mad()
    X.loc[segment, 'kurt'] = x_data.kurtosis()
    X.loc[segment, 'skew'] = x_data.skew()
    X.loc[segment, 'med'] = x_data.median()
```

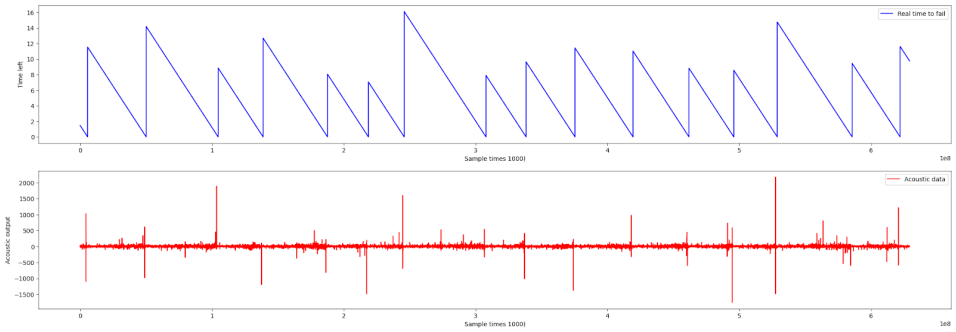
```
    X.loc[segment, 'abs_q95'] = np.quantile(np.abs(x_data),0.95)
    X.loc[segment, 'abs_q99'] = np.quantile(np.abs(x_data),0.99)
    X.loc[segment, 'F_test'], X.loc[segment, 'p_test'] =
stats.f_oneway(x_data[:30000],x_data[30000:60000],x_data[60000:90000],
x_data[90000:120000],x_data[120000:])
    X.loc[segment, 'av_change_abs'] = np.mean(np.diff(x_data))

    return X


def feature_extraction(data,segments):
    X = pd.DataFrame(index=range(segments), dtype=np.float64)
    y = pd.DataFrame(index=range(segments),
dtype=np.float64,columns=['time_to_failure'])
    for segment in tqdm(range(segments)):
        seg = data.iloc[segment*rows:segment*rows+rows]
        x_data = pd.Series(seg['acoustic_data'].values)
        y_data = seg['time_to_failure'].values[-1]


        y.loc[segment,'time_to_failure'] = y_data

        X = featuresx(x_data, X, segment)

    return X, y


from tqdm import tqdm
rows = 150_000
segments = int(np.floor(data.shape[0] / rows))


X,y = feature_extraction(data, segments)
```

```
y=y.values.flatten()
print(X.shape)
print(y.shape)
```
Output
```
0it [00:00, ?it/s](0, 0)
(0,)
```
2.1.1 Train-test split
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,shuffle=True, random_state=102)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```
Output
```
(3355, 30)
(839, 30)
(3355,)
(839,)
```

2.2 Data Exploration
```
n=1000
y1_acoustic_data=data['acoustic_data'][::n]
y2_time_to_failure=data['time_to_failure'][::n]
print(data[0:10*n:n])
```

```
#plotting the graphs of all the deta
fig, ax = plt.subplots(2, 1, figsize=(27,9))
ax[0].plot(y2_time_to_failure, color='b')
ax[0].legend(['Real time to fail'])
ax[0].set_ylabel('Time left')
ax[0].set_xlabel('Sample times 1000)')
ax[1].plot(y1_acoustic_data, color='r')
```

```
ax[1].legend(['Acoustic data'])
ax[1].set_ylabel('Acoustic output' )
ax[1].set_xlabel('Sample times 1000)')
      acoustic_data  time_to_failure
Output
0               12         1.469100
1000             6         1.469099
2000             3         1.469098
3000            29         1.469097
4000            13         1.469096
5000             6         1.468099
6000             9         1.468098
7000             0         1.468097
8000            -1         1.468096
9000             9         1.466999
Text(0.5, 0, 'Sample times 1000)')
```



2.3 Data Preparation

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

3. Training and Results

```
from sklearn.linear_model import LinearRegression
# define the model
model = LinearRegression()
```

```
# fit the model
model.fit(X_train, y_train)
# predict
predictions=model.predict(X_test)
#accuracy
score=model.score(X_test, y_test)
print(score)
mse = mean_squared_error(y_test, predictions)
print(mse)
```

Output

```
0.4534023803770155
7.017008736175641
#define and fit model
nsvm = NuSVR(kernel='linear')
nsvm.fit(X_train, y_train)

# predict
predictions=nsvm.predict(X_test)
# accuracy
score=nsvm.score(X_test, y_test)
print(score)
mse = mean_squared_error(y_test, predictions)
print(mse)
```

Output

```
0.44697377313180875
7.099536708456553
#define and fit model
svm = NuSVR(kernel='rbf')
svm.fit(X_train, y_train)

# predict
predictions=svm.predict(X_test)
# accuracy
score=svm.score(X_test, y_test)
```

```
print(score)
mse = mean_squared_error(y_test, predictions)
print(mse)
```
Output
0.4540831746662992
7.008268962521417
```
from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor(max_depth=3)
tree.fit(X_train, y_train)
predictions = tree.predict(X_test)
score=tree.score(X_test, y_test)
print(score)
mse = mean_squared_error(y_test, predictions)
print(mse)
```
Output
0.4421844511602614
7.161020170711448

4. Discussion and Conclusion
```
submission =
pd.read_csv('../input/LANL-Earthquake-Prediction/sample_submission.csv',
index_col='seg_id')
X_test = pd.DataFrame(dtype=np.float64, index=submission.index)

for seg_id in tqdm(X_test.index):
    seg = pd.read_csv('../content.Earthquake-Prediction/test/' + seg_id +
'.csv')

    x = pd.Series(seg['acoustic_data'].values)
    X_test = featuresx(x, X_test, seg_id)

print(x.shape)
print(X_test.shape)
```

```
X_test = scaler.transform(X_test)
submission['time_to_failure'] = svm.predict(X_test)
submission.to_csv('submission.csv')

print(submission)
```

Output

```
100%
2624/2624 [06:09<00:00, 6.53it/s]
(150000,)
(2624, 30)
          time_to_failure
seg_id
seg_00030f        4.342878
seg_0012b5        6.150087
seg_00184e        5.998249
seg_003339        7.985206
seg_0042cc        7.101499
...                    ...
seg_ff4236        5.610940
seg_ff7478        6.446504
seg_ff79d9        4.966563
seg_ffbd6a        2.149476
seg_ffe7cc        9.163199

[2624 rows x 1 columns]
```

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.filterwarnings('ignore')
```

Conclusion

We also learned to work with big data, which leads into not being able to plot everything or make a useful table to see what you doing. This had made the understanding of the deta much harder. Furthermore our results are all under 50% which is not good, but we did not do the project to

achieve a certain performance but to learn with the concept of machine learning.

Performing association analysis Generating insights

Development phase

Import the necessary libraries required for buidling the model and data analysis of the earthquakes.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv("/content/earthquake .csv")
data.head()
```

Output

Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ID	Source	Location Source	Magnitude Source	Status
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic

5 rows × 21 columns

```
data.columns
```

Output

```
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')
```

Figure out the main features from earthquake data and create a object of that features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.

```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```

Output

	Date	Time	Latitude	Longitude	Depth	Magnitude	
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0	
1	01/04/1965	11:29:49	1.863	127.352	80.0	5.8	
2	01/05/1965	18:05:58	-20.579	-173.972	20.0	6.2	
3	01/08/1965	18:49:43	-59.076	-23.557	15.0	5.8	
4	01/09/1965	13:32:50	11.938	126.427	15.0	5.8	

Here, the data is random we need to scale according to inputs to the model. In this, we convert given Date and Time to Unix time which is in seconds and a numeral. This can be easily used as input for the network we 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()
```

Output

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-157630542.0
1	1.863	127.352	80.0	5.8	-157465811.0
2	-20.579	-173.972	20.0	6.2	-157355642.0
3	-59.076	-23.557	15.0	5.8	-157093817.0
4	11.938	126.427	15.0	5.8	-157026430.0

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.

```
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()
```
Output

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

```
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)
```
Output

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

Here, we used the RandomForestRegressor model to predict reg.score(X_test, y_test) 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
```
Output
```
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)
```
Output

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)
```

Output

```
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)
```

Output

0.8749008584467053

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
```

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

```
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', 'exponential']
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
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
Output
Best: 0.957655 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons':
16, 'optimizer': 'SGD'}
0.333316 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'Adadelta'}
0.957655 (0.029957) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'SGD'}
0.645111 (0.456960) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'Adadelta'}

The best fit parameters are used for same model to compute the score with training data and testing data.
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(3,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='SGD', loss='squared_hinge', metrics=['accuracy'])

model.fit(X_train, y_train, batch_size=10, epochs=20, 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))
Output
4682/4682 [==============================] - 0s 39us/step
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

Advantage

Python can be a useful tool for studying earthquakes and seismic data due to its versatility and various libraries for scientific computing. Here are some advantages of using Python for earthquake-related tasks:

1. Data Analysis: Python offers powerful data analysis libraries like NumPy and Pandas, which are essential for handling and processing seismic data, such as earthquake magnitudes, depths, and locations.

2. Visualization: Libraries like Matplotlib and Seaborn enable the creation of informative plots and graphs, making it easier to visualize seismic data, earthquake epicenters, and historical earthquake patterns.

3. Machine Learning: Python has an extensive ecosystem for machine learning with libraries like Scikit-Learn and TensorFlow. Researchers can use these tools to develop earthquake prediction models or detect seismic patterns.

4. Geospatial Analysis: Python's Geospatial libraries (e.g., GDAL and Shapely) are beneficial for working with geographical data. You can analyze fault lines, plate boundaries, and map earthquake occurrences on geographic maps.

5. Data Access: Python has packages like ObsPy and FDSN that make it easier to access and retrieve seismic data from various sources, such as seismographic networks and earthquake databases.

6. Numerical Simulations: Researchers can use Python for numerical simulations of earthquake behavior and structural response. Libraries like SciPy can be helpful in solving partial differential equations relevant to seismic studies.

7. Integration with GIS: Python can be integrated with Geographic Information Systems (GIS) software like QGIS and ArcGIS, allowing for a more comprehensive analysis of earthquake-related data in a spatial context.

8. Community and Resources: Python has a large and active user community, which means you can find a wealth of resources, tutorials, and open-source projects related to earthquake research and analysis.

9. Customization: Python allows for the development of custom scripts and tools tailored to specific earthquake research needs, giving researchers greater flexibility.

10. Cross-Platform Compatibility: Python is available on multiple platforms (Windows, macOS, Linux) and is widely used in scientific research, making it accessible to a broad audience.

disadvantage

To analyze the disadvantages of earthquakes using Python, you can perform various tasks like data visualization, seismic hazard assessment, or risk analysis. Here's a simple example of how you might analyze earthquake data using Python:

1. **Data Collection**: Obtain earthquake data from sources like the USGS (United States Geological Survey) or a relevant dataset. You can use libraries like `pandas` to import and manipulate the data.

2. **Data Visualization**: Use Python libraries like `matplotlib` or `seaborn` to create visualizations that illustrate the disadvantages of earthquakes. For example, you can create maps showing earthquake-prone areas, time series plots of earthquake frequency, or histograms of earthquake magnitudes.

3. **Seismic Hazard Assessment**: Python can be used to calculate seismic hazard. The OpenQuake engine is a widely used open-source tool for this purpose. You can use Python to set up and run seismic hazard assessments using OpenQuake or similar tools.

4. **Risk Analysis**: Perform risk analysis by combining earthquake hazard data with information on buildings, infrastructure, and population. This can be done using Python libraries for probabilistic modeling, such as `numpy` and `scipy`. You can also use machine learning models for risk assessment.

Here's a basic example of Python code to plot earthquake data from a CSV file using `matplotlib`:

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load earthquake data from a CSV file
earthquake_data = pd.read_csv('earthquake_data.csv')

# Create a histogram of earthquake magnitudes
plt.hist(earthquake_data['Magnitude'], bins=20, color='blue', alpha=0.7)
plt.xlabel('Magnitude')
plt.ylabel('Frequency')
plt.title('Earthquake Magnitude Distribution')
plt.show()
```

Keep in mind that earthquake analysis is a complex field, and Python can be used for various aspects of it. Depending on your specific goals, you may need to employ more advanced techniques and libraries.

Conclusion

To draw conclusions about earthquakes using Python, you can analyze earthquake data to understand patterns, magnitudes, and locations. Here's a simplified example of how you can draw conclusions using Python:

1. Data Collection: Obtain earthquake data from reliable sources like the USGS Earthquake API.

2. Data Preprocessing: Clean and preprocess the data, removing duplicates and irrelevant information.

3. Data Visualization: Use libraries like Matplotlib or Seaborn to create visualizations like scatter plots, histograms, or heatmaps to explore the data.

4. Statistical Analysis: Calculate descriptive statistics, such as mean, median, and standard deviation of earthquake magnitudes, depths, and locations.

5. Geographic Analysis: Utilize geospatial libraries like Folium or Basemap to create maps showing earthquake locations and their magnitudes.

6. Time Series Analysis: Analyze how earthquake frequency or magnitude has changed over time using time series analysis techniques.

7. Machine Learning: Apply machine learning models to predict earthquake magnitudes or locations based on historical data.

8. Hypothesis Testing: Perform hypothesis tests to draw statistical conclusions about earthquake patterns or associations with other factors.

9. Reporting: Summarize your findings and conclusions in a clear and concise report or presentation.

Here's a simple Python code snippet to get you started with earthquake data analysis:

```python
import pandas as pd
import matplotlib.pyplot as plt

# Sample earthquake data (replace with your data source)
data = pd.read_csv("earthquake_data.csv")
```

```
# Data preprocessing and analysis
# (You can perform data preprocessing, visualization, and analysis here)

# Example: Creating a histogram of earthquake magnitudes
plt.hist(data['magnitude'], bins=20)
plt.xlabel('Magnitude')
plt.ylabel('Frequency')
plt.title('Histogram of Earthquake Magnitudes')
plt.show()
```

Remember that earthquake analysis can be quite complex, and the level of detail and sophistication in your analysis will depend on your specific goals and the data available.