EARTH QUAKE PREDICTION USING PYTHON

TEAM MEMBER

621121104012: GOWRI.K

Phase 5 Submission

Project Title*: *Earth Quake Prediction

Topic: In this section we will document the complete project and prepare if for submission.

Earthquake Prediction

Introduction

- Earthquake prediction is a complex and challenging field of study that aims to forecast when and where earthquakes might occur.
- While it's important to note that accurate short-term earthquake prediction remains elusive, researchers use various data analysis techniques to assess seismic hazard and understand earthquake patterns.
- Python, a versatile programming language, is widely used for earthquake prediction research due to its extensive libraries and data analysis capabilities.

In this introduction, we'll explore the basic steps and tools involved in earthquake prediction using Python.

1. Data Collection:

The foundation of earthquake prediction is data. Earthquake researchers collect seismic data, including ground motion records, fault movement data, and historical earthquake information. Databases like the United States Geological Survey (USGS) provide access to this data.

2. Data Preprocessing:

Once the data is collected, it needs to be preprocessed to remove noise, correct errors, and make it suitable for analysis. Python libraries like NumPy and pandas are commonly used for data preprocessing.

3. Feature Extraction:

Researchers extract relevant features from the seismic data, such as peak ground acceleration, frequency content, and ground motion duration. These features help in understanding earthquake characteristics.

4. Machine Learning:

Machine learning techniques are applied to the feature data to build predictive models. Supervised learning models, such as decision trees, support vector machines, and neural networks, can be used to identify patterns and correlations in the data.

5. Time Series Analysis:

Time series analysis is crucial for identifying seismic trends and patterns. Libraries like SciPy and status models in Python are helpful for time series analysis.

6. Geospatial Analysis:

Geographic information systems (GIS) and geospatial libraries like

Geo Pandas and Base Map are used to analyze the spatial distribution of earthquakes. Researchers create maps and spatial models to identify high-risk areas.

7. Statistical Methods:

Statistical tests and methods are applied to assess the likelihood of earthquakes occurring in a specific region. Probabilistic seismic hazard assessment (PSHA) is a widely used approach.

8. Deep Learning:

Deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are used for advanced seismic pattern recognition and prediction.

9. Visualization:

Data visualization is essential for presenting results and insights. Python libraries like Matplotlib, Seaborn, and Plot help in creating informative charts and graphs.

10. Collaboration and Communication:

Collaboration with seismologists and earthquake experts is crucial to validate predictions and models. Researchers often create reports, articles, and presentations to communicate their findings to the scientific community and the public.

It's important to emphasize that earthquake prediction is an ongoing research field with significant uncertainties. While Python and data analysis techniques can help in understanding seismic patterns and improving our preparedness for earthquakes, short-term earthquake prediction remains a complex challenge.

Researchers continue to work towards better understanding earthquake mechanisms and improving the accuracy of predictions. Python and its rich ecosystem of libraries are valuable tools in this ongoing effort.

Design thinking and present in form of documents

Abstract:

The prediction of earthquakes remains a significant challenge, with potentially catastrophic consequences. This abstract introduces a design thinking approach for the development of an Earthquake Prediction Model using Python, emphasizing user-centricity, innovation, and problem-solving. The approach is structured into modular phases, each guided by design thinking principles.

Data Source:

In this phase, the research team engages with stakeholders, including seismologists, disaster management authorities, and the general public, to gain deep insights into their concerns, information needs, and the impact of earthquakes on their lives.

Feature Exploration:

A well-defined problem statement and clear project objectives are established. This phase includes identifying the target regions for earthquake prediction and the desired prediction timeframes.

Visualization:

Creative brainstorming sessions are conducted to explore innovative ideas for building an Earthquake Prediction Model. This phase encourages considering various data sources, machine learning algorithms, and predictive features.

Data Splitting:

A preliminary prototype of the Earthquake Prediction Model is developed using historical seismic data. This prototype serves as a proof of concept, allowing stakeholders to interact with and provide feedback on the system's functionality and usability.

The prototype is tested with stakeholders, including seismologists, emergency response teams, and the public. User feedback is collected to identify model inaccuracies, usability issues, or other concerns. Iterative improvements are made based on this feedback.

Model Development:

- After refining the prototype, a full-scale Earthquake Prediction Model is developed using Python. The model incorporates a wide range of data sources, including seismic data, geological features, and meteorological information, and employs advanced machine learning techniques.

- The fully developed model is deployed into a real-time monitoring system that continuously assesses incoming seismic data. Alerts are generated and communicated to relevant authorities and the public when potential earthquake events are detected.

Training and Evaluation:

- Continuous monitoring of the model's performance is carried out, assessing prediction accuracy and user satisfaction. Regular updates and model retraining are scheduled to adapt to evolving seismic patterns and data characteristics.

- The design thinking process is iterative, and this module encourages the incorporation of user feedback and emerging technologies to continually evolve the Earthquake Prediction Model. New data sources, improved algorithms, and enhanced alerting mechanisms are explored.

This design thinking approach for developing an Earthquake Prediction Model using Python places user needs and real-world impact at the forefront. By integrating design thinking principles into each module, the development process becomes more empathetic and user-centric, ensuring that the prediction system addresses the specific concerns and requirements of stakeholders while continuously evolving to improve earthquake prediction capabilities.

```
import numpy as np

import pandas as pd
import matplotlib. pyplot as plt

import os
print(os.listdir("../input"))
```

output:

```
['database.csv']
```

Output:

Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type
01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW
01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW
01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW
01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW
01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW

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

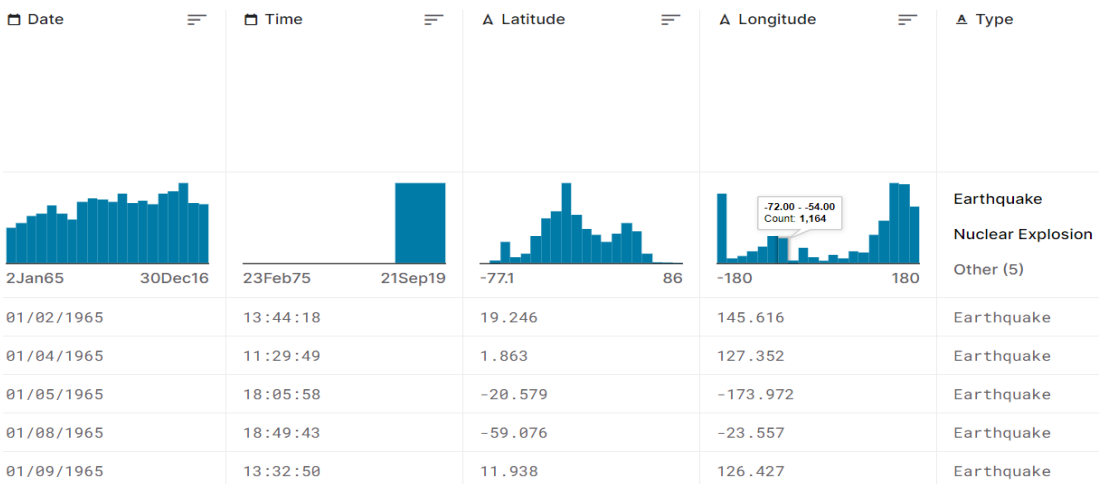
output:

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

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlo
n=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)

from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

```
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]])
```



Date	Time	Latitude	Longitude	Type
2Jan65 – 30Dec16	23Feb75 – 21Sep19	-77.1 – 86	-180 – 180 (-72.00 - -54.00 Count: 1,164)	Earthquake / Nuclear Explosion / Other (5)
01/02/1965	13:44:18	19.246	145.616	Earthquake
01/04/1965	11:29:49	1.863	127.352	Earthquake
01/05/1965	18:05:58	-20.579	-173.972	Earthquake
01/08/1965	18:49:43	-59.076	-23.557	Earthquake
01/09/1965	13:32:50	11.938	126.427	Earthquake

2. Hyperparameter tuning and Feature engineering

Hyperparameter Tuning:

Hyperparameters are parameters that are not learned by the model during training but are set prior to training. Optimizing these hyperparameters can significantly impact a model's performance. Here's how you can approach hyperparameter tuning.

Grid Search:

In grid search, you define a range of hyperparameter values, and the algorithm evaluates the model's performance for all possible combinations of these values. This can be computationally expensive but is thorough.

Random Search:

Random search selects hyperparameter values randomly from predefined ranges. It's less computationally intensive than grid search and can often find good hyperparameters faster.

Bayesian Optimization:

Bayesian optimization is a more sophisticated approach that uses probabilistic models to find the best set of hyperparameters. It's efficient and can often outperform grid or random search.

Automated Hyperparameter Tuning:

You can also use automated machine learning libraries such as TPOT, Auto-Google to automate the hyperparameter tuning process.

```
# Data manipulation
import pandas as pd
import numpy as np

# Modeling
import lightgbm as lgb

# Splitting data
from sklearn.model_selection import train_test_split
```

```
N_FOLDS = 5
MAX_EVALS = 5
```

```
features = pd.read_csv('../input/home-credit-default-risk/application_train.csv')

# Sample 16000 rows (10000 for training, 6000 for testing)
features = features.sample(n = 16000, random_state = 42)

# Only numeric features
features = features.select_dtypes('number')

# Extract the labels
labels = np.array(features['TARGET'].astype(np.int32)).reshape((-1, ))
features = features.drop(columns = ['TARGET', 'SK_ID_CURR'])

# Split into training and testing data
train_features, test_features, train_labels, test_labels = train_test_split(featu
res, labels, test_size = 6000, random_state = 50)
```

```
print("Training features shape: ", train_features.shape)
print("Testing features shape: ", test_features.shape)
Training features shape:  (10000, 104)
Testing features shape:  (6000, 104)
```

```
Create a training and testing dataset
train_set = lgb.Dataset(data = train_features, label = train_labels)
test_set = lgb.Dataset(data = test_features, label = test_labels)
```

```
Training features shape:  (10000, 104)
Testing features shape:  (6000, 104)
```

```
# Get default hyperparameters
model = lgb.LGBMClassifier()
default_params = model.get_params()

# Remove the number of estimators because we set this to 10000 in the cv call
del default_params['n_estimators']

# Cross validation with early stopping
cv_results = lgb.cv(default_params, train_set, num_boost_round = 10000, early_sto
pping_rounds = 100,
                    metrics = 'auc', nfold = N_FOLDS, seed = 42)
```

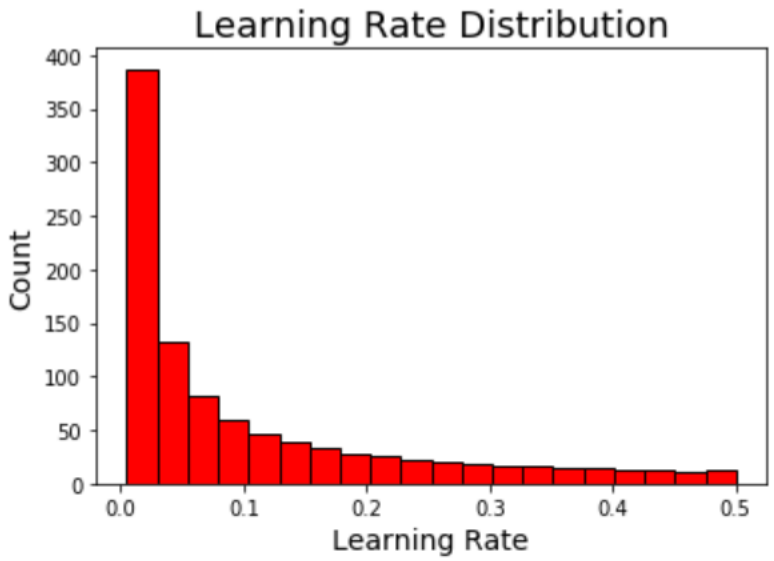
```
import random

random.seed(50)
```

```
# Randomly sample a boosting type
boosting_type = random.sample(param_grid['boosting_type'], 1)[0]

# Set subsample depending on boosting type
subsample = 1.0 if boosting_type == 'goss' else random.sample(param_grid['subsamp
le'], 1)[0]

print('Boosting type: ', boosting_type)
print('Subsample ratio: ', subsample)
```

```
Boosting type:  goss
Subsample ratio:  1.0
```

```
Boosting type:  goss
Subsample ratio:  1.0
```

```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# Learning rate histogram
plt.hist(param_grid['learning_rate'], bins = 20, color = 'r', edgecolor = 'k');
plt.xlabel('Learning Rate', size = 14); plt.ylabel('Count', size = 14); plt.title
('Learning Rate Distribution', size = 18);
```



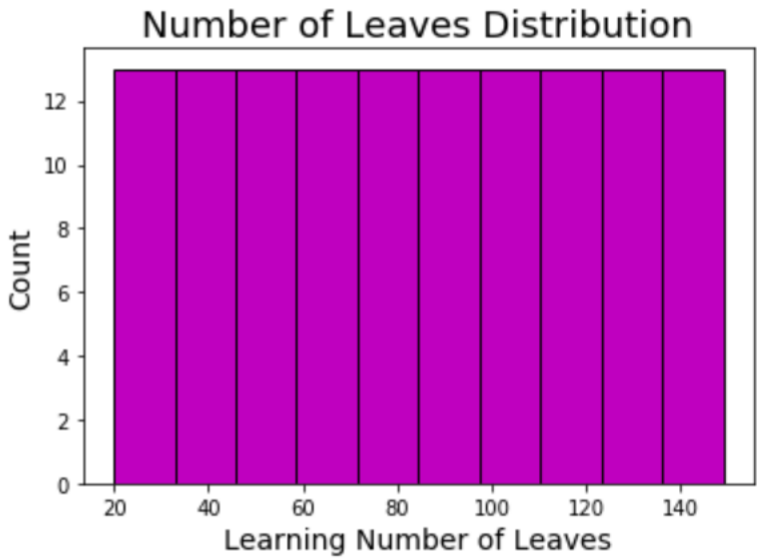

```
a = 0
b = 0

# Check number of values in each category
```

```
for x in param_grid['learning_rate']:
    # Check values
    if x >= 0.005 and x < 0.05:
        a += 1
    elif x >= 0.05 and x < 0.5:
        b += 1

print('There are {} values between 0.005 and 0.05'.format(a))
print('There are {} values between 0.05 and 0.5'.format(b))
```

```
There are 499 values between 0.005 and 0.05
There are 499 values between 0.05 and 0.5
```

```
a = 0
b = 0

# Check number of values in each category
for x in param_grid['learning_rate']:
    # Check values
    if x >= 0.005 and x < 0.05:
        a += 1
    elif x >= 0.05 and x < 0.5:
        b += 1

print('There are {} values between 0.005 and 0.05'.format(a))
print('There are {} values between 0.05 and 0.5'.format(b))
```

```
There are 499 values between 0.005 and 0.05
There are 499 values between 0.05 and 0.5
```



Feature Engineering:

Feature engineering involves creating new features from the existing dataset or transforming existing features to provide more relevant information to

the model. Here are some techniques for feature engineering:

Feature engineering involves creating new features from the existing dataset or transforming existing features to provide more relevant information to the model. Here are some techniques for feature engineering:

Feature Selection:

Identify and keep only the most relevant features, removing any irrelevant or redundant ones. This can help reduce dimensionality and improve model performance.

Feature Scaling:

Ensure that all features are on a similar scale, especially when using models sensitive to feature scales like gradient boosting or support vector machines

Feature Extraction:

Techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used to extract relevant information from high-dimensional data.

One-Hot Encoding:

Convert categorical variables into binary vectors (one-hot encoding) so that machine learning algorithms can work with them effectively.

Feature Creation:

Sometimes, creating new features based on domain knowledge can be beneficial. For example: deriving a "customer loyalty score" from transaction history data.

Handling Missing Data:

Decide on a strategy to handle missing data, which could involve imputation techniques like mean imputation, median imputation, or more advanced methods like K-nearest neighbor imputation.

Remember to validate the performance of your model using techniques like cross-validation to ensure that the improvements from hyperparameter tuning and feature engineering are statistically significant and generalize well to unseen data. It's also essential to iterate and experiment with different approaches to find the best combination of hyperparameters and feature engineering techniques for your specific problem.

```
import pandas as pd
import seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.ticker import MaxNLocator


from scipy import interp
import math
from scipy.stats import norm
from scipy import stats


import warnings
warnings.filterwarnings('ignore') # Disabling warnings for clearer outputs.
pd.options.display.max_columns = 50 # Pandas option to increase max number of columns
to display.
plt.style.use('ggplot') # Setting default plot style.



# Read train and test data from csv files for visualization:

v_train = pd.read_csv('/kaggle/input/titanic/train.csv')
v_test = pd.read_csv('/kaggle/input/titanic/test.csv')

idx = len(v_train)
```

In [3]:

```
linkcode
# Checking train and test sets
display(v_train.sample(3))
display(v_test.sample(3))
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
743	744	0	3	McNamee, Mr. Neal	male	24.0	1	0	376566	16.1000	NaN	S
547	548	1	2	Padro y Manent, Mr. Julian	male	NaN	0	0	SC/PARIS 2146	13.8625	NaN	C
820	821	1	1	Hays, Mrs.	female	52.0	1	1	12749	93.5000	B69	S

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
143	1035	2	Beauchamp, Mr. Henry James	male	28.0	0	0	244358	26.00	NaN	S
57	949	3	Abelseth, Mr. Olaus Jorgensen	male	25.0	0	0	348122	7.65	F G63	S
34	926	1	Mock, Mr. Philipp Edmund	male	30.0	1	0	13236	57.75	C78	C

```
# Merging visualization datasets.

v_train.drop('PassengerId', axis=1, inplace=True)
v_test.drop('PassengerId', axis=1, inplace=True)
```

```
v_merged = pd.concat([v_train, v_test], sort=False).reset_index(drop=True)
```

In [5]:

```
# Checking merged shape:

display(v_merged.shape)
```

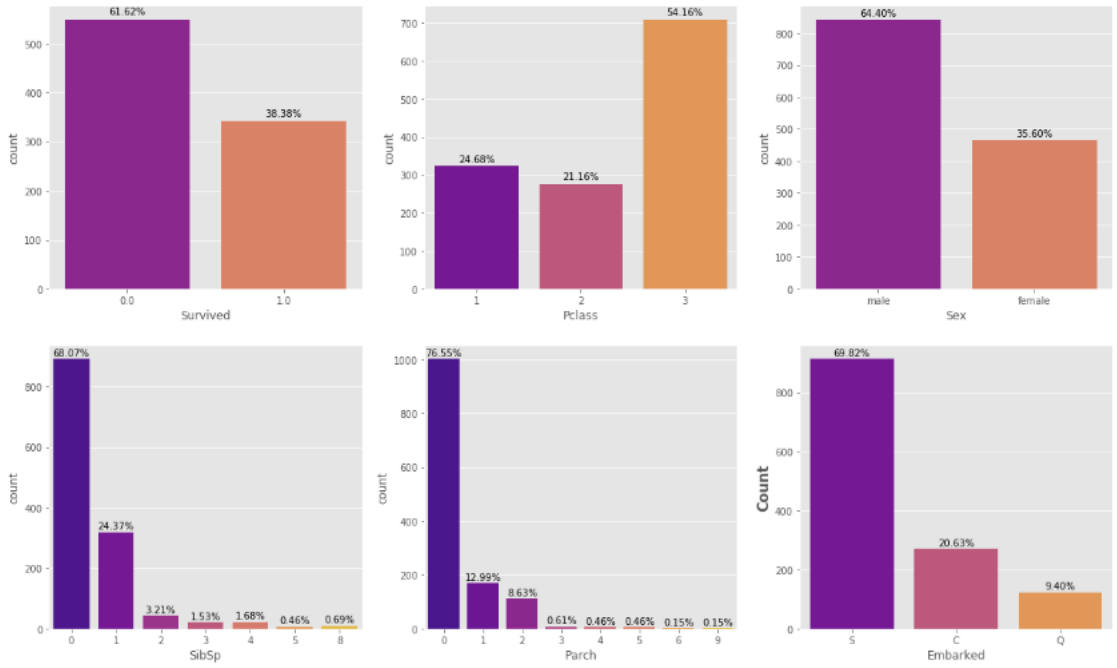
```
(1309, 11)
```

In [6]:

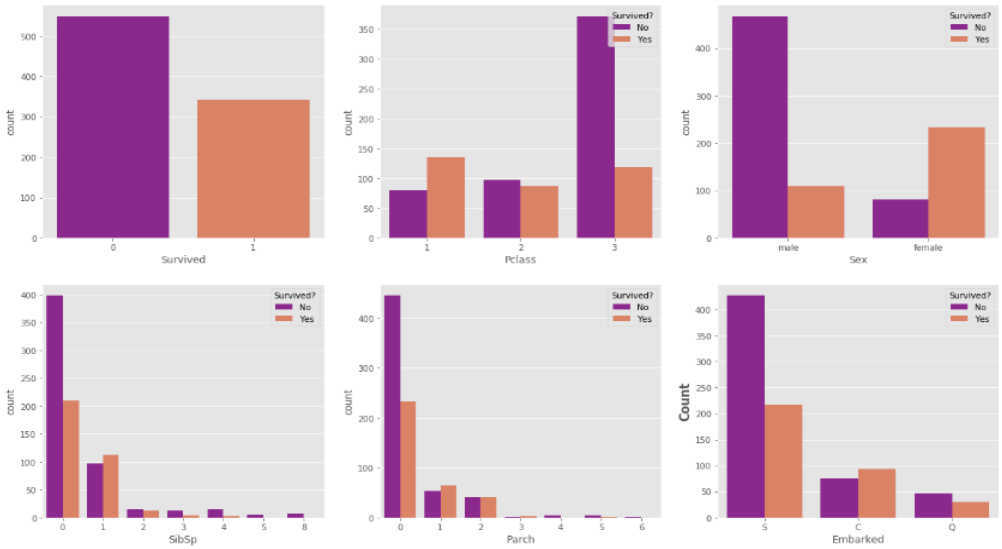
```
# Checking features and target columns:

display(v_merged.columns)
```

```
Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

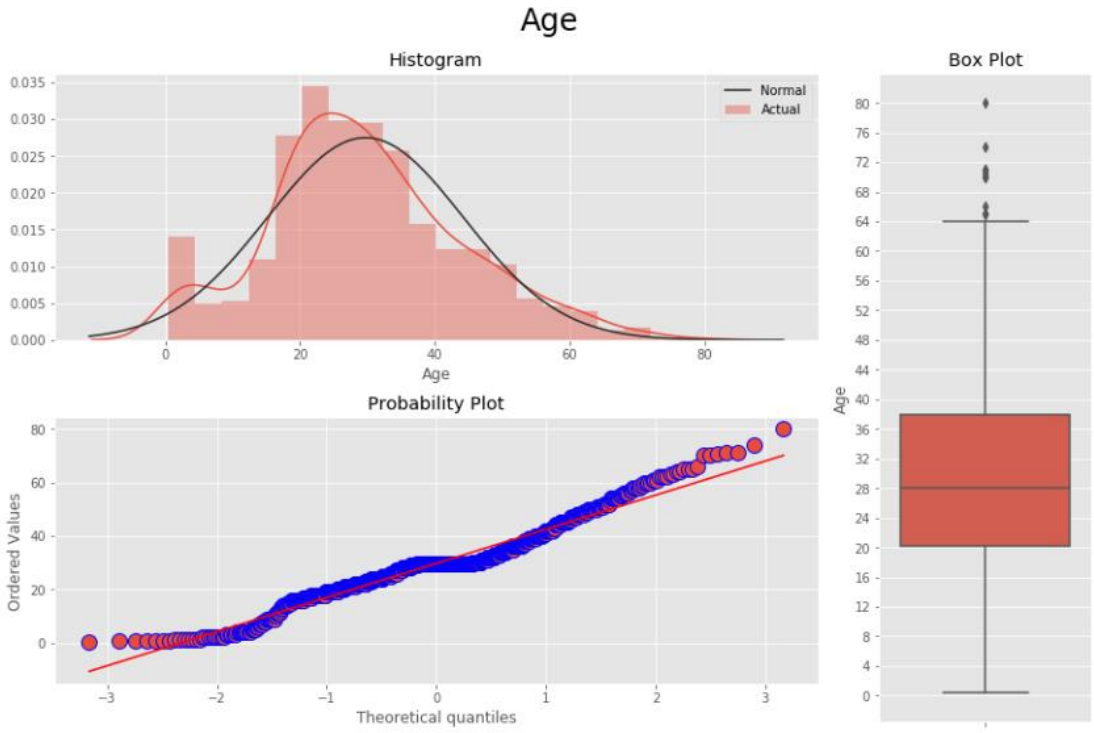
```
plotFrequency(cats)
```

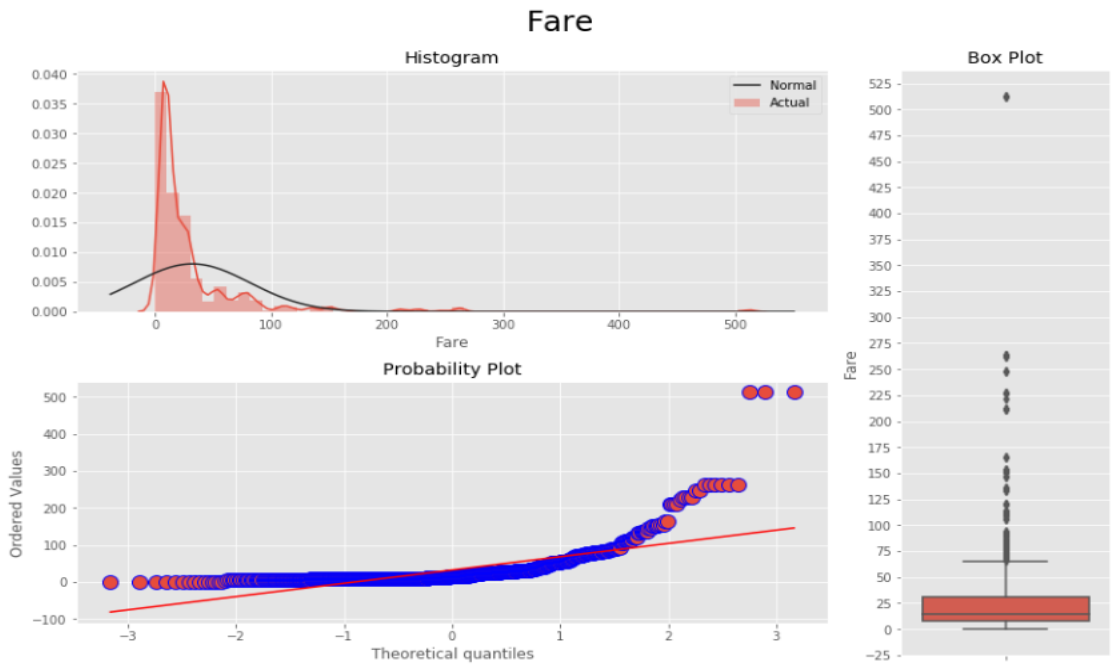



```
plotsurvival(cats, v_train)
```




```
plot_3chart(v_train, 'Age')
plot_3chart(v_train, 'Fare')
```





Preprocess

Abstract:

An earthquake is shaking of the surface of the Earth, which caused as the result of movable plate boundary interactions Earthquakes are measured using remarks from seismometers with Richter magnitude scale.

Ground rupture, Landslides, Soil liquefaction and Tsunami are the main effects created by earthquakes. Today's earthquake warning systems used to provide regional notification of an earthquake in progress. Many methods have been already developed for predicting the time and place in which earthquakes will occur.

Program:

In [1]:

```
import pandas as pd
```

Preprocessing steps:

In [2]:

```
df = pd.read_csv
```

In [3]:

```
df.head()
```

out [3]:

	time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms
0	2023-02-14T21:31:52.124Z	60.828300	-151.841200	85.00	2.20	ml	NaN	NaN	NaN	1.6100
1	2023-02-14T20:45:56.420Z	19.254333	-155.410828	31.32	2.27	ml	41.0	139.00	NaN	0.1500
2	2023-02-14T20:45:12.919Z	38.146900	-117.982000	7.30	1.90	ml	11.0	110.46	0.02000	0.1385
3	2023-02-14T20:43:53.796Z	63.898700	-148.655300	82.40	1.30	ml	NaN	NaN	NaN	0.5700
4	2023-02-14T20:43:40.220Z	33.324167	-116.757167	12.42	0.89	ml	23.0	67.00	0.08796	0.1700

In [4]:

```
df.tail()
```

Out [4]:

	time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms
10148	2023-01-15T21:45:41.552Z	41.507200	19.986200	10.000	4.10	mb	32.0	40.0	0.1850	0.66
10149	2023-01-15T21:42:53.540Z	33.323167	-116.900333	19.470	1.28	ml	42.0	39.0	0.1282	0.16
10150	2023-01-15T21:42:37.119Z	62.524300	-149.442100	50.900	1.10	ml	NaN	NaN	NaN	0.48
10151	2023-01-15T21:39:04.248Z	63.160500	-150.495600	112.900	1.70	ml	NaN	NaN	NaN	0.40
10152	2023-01-15T21:37:21.608Z	41.451300	19.999500	16.295	4.90	mb	105.0	37.0	0.1460	0.74

In [5]:

```
df.shape
```

```
Out [5]:
        (10153, 22)
In [6]:
       df.columns
Out[6]:
Index (['time', 'latitude', 'longitude', 'depth', 'mag', 'magType', 'nst',
      'gap', 'dmin', 'rms', 'net', 'id', 'updated', 'place', 'type',
      'horizontalError', 'depthError', 'magError', 'magNst', 'status',
      'locationSource', 'magSource'],
     dtype='object')

In[7]:
df.dtypes
Out [7]:
time             object
latitude         float64
longitude        float64
depth            float64
mag              float64
magType          object
nst              float64
gap              float64
dmin             float64
rms              float64
```

In [8]:

```
df['time'] = pd.to_datetime(df['time'])
```

In [9]:

```
df.dtypes
```

Out [9]:

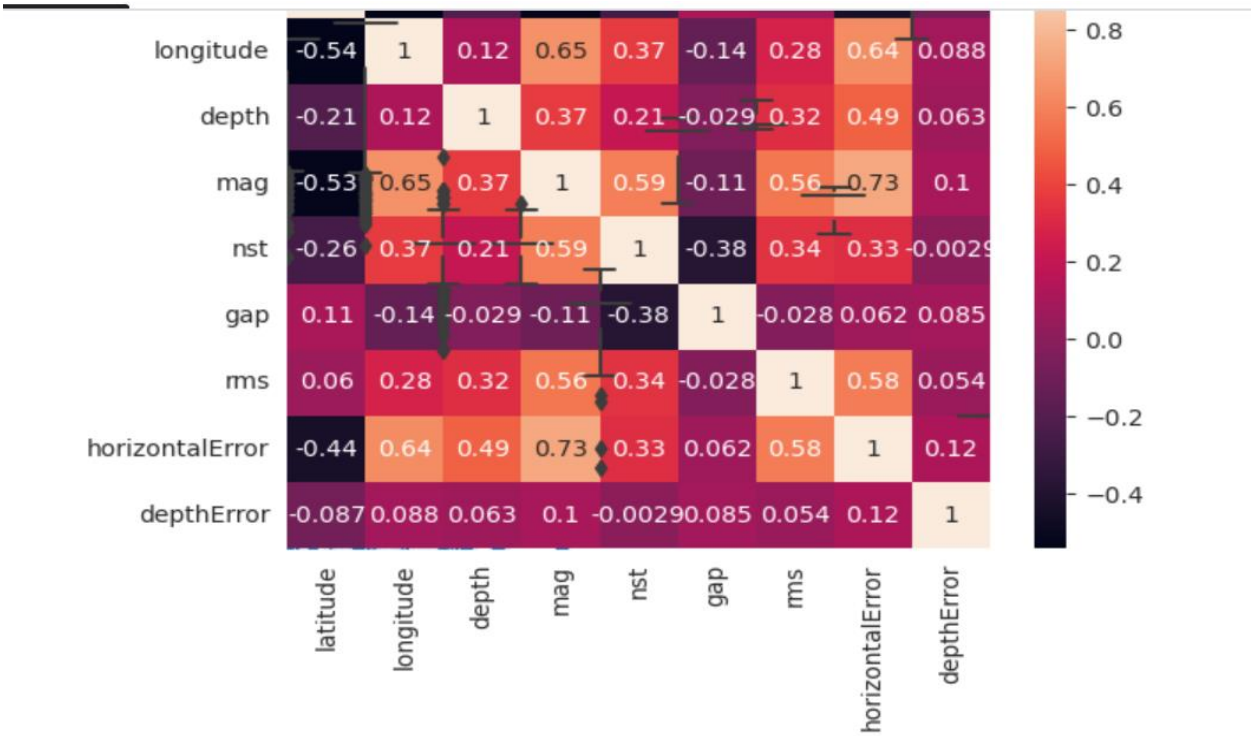
```
time             datetime64[ns, UTC]
latitude         float64
longitude        float64
depth            float64
mag              float64
magType          object
nst              float64
gap              float64
dmin             float64
rms              float64
```

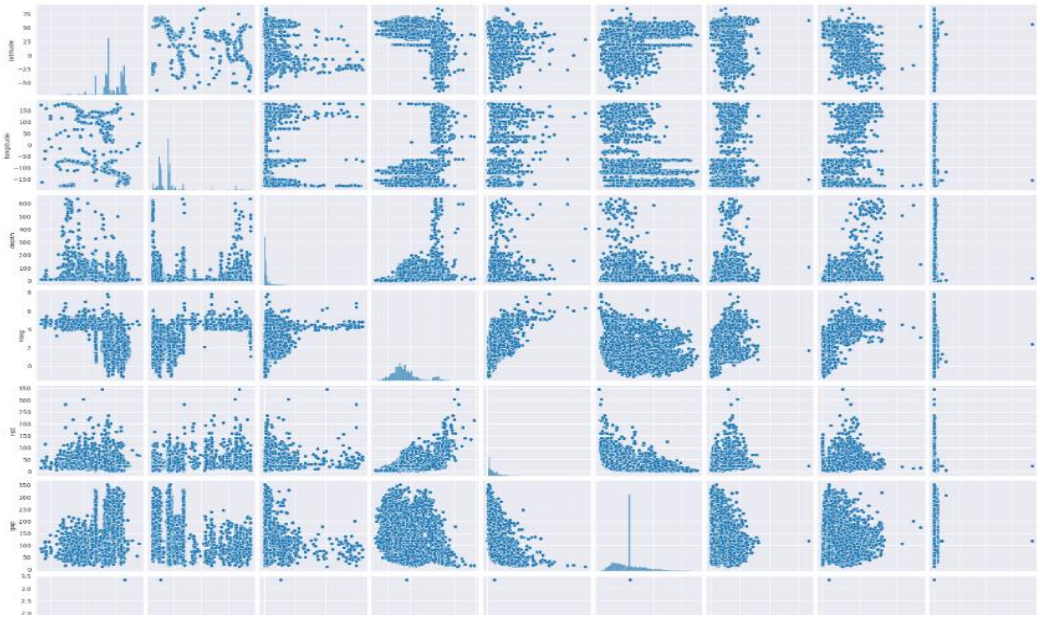
Data Visualization:

In [10]:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Set style for all visualizations
sns.set_style("darkgrid")
# Scatter plot to show relationship between magnitude and depth
sns.scatterplot(data=df, x="mag", y="depth")
# Bar plot to show distribution of magnitudes
```

```
sns.histplot(data=df, x="mag")
# Box plot to show distribution of magnitudes by type
sns.boxplot(data=df, x="magType", y="mag")
# Heatmap to show correlation between variables
corr = df.corr()
sns.heatmap(corr, annot=True)
# Pairplot to show scatterplots of all possible variable combinations
sns.pairplot(df)
# Show all visualizations
plt.show()
```





Statistical Analysis to test hypothesis.

In [11]:

```
from scipy.stats import ttest_ind
# creating a new 'region' column by extracting region from 'place' column
df['region'] = df['place'].str.extract(',\s(.*$)')
# grouping by 'region' and calculating mean of 'mag' column for each group
mean_mag_by_region = df.groupby('region')['mag'].mean()
# separating the two groups based on the 'mag' column
group1 = df[df['mag'] < mean_mag_by_region.mean()]
group2 = df[df['mag'] >= mean_mag_by_region.mean()]
# performing independent t-test between the two groups
t_stat, p_val = ttest_ind(group1['mag'], group2['mag'], equal_var=False)
```

```
print("T-test statistic: ", t_stat)

print("P-value: ", p_val)
```

T-test statistic: -210.99984844737443

P-value: 0.0

Earthquake Prediction Analysis:

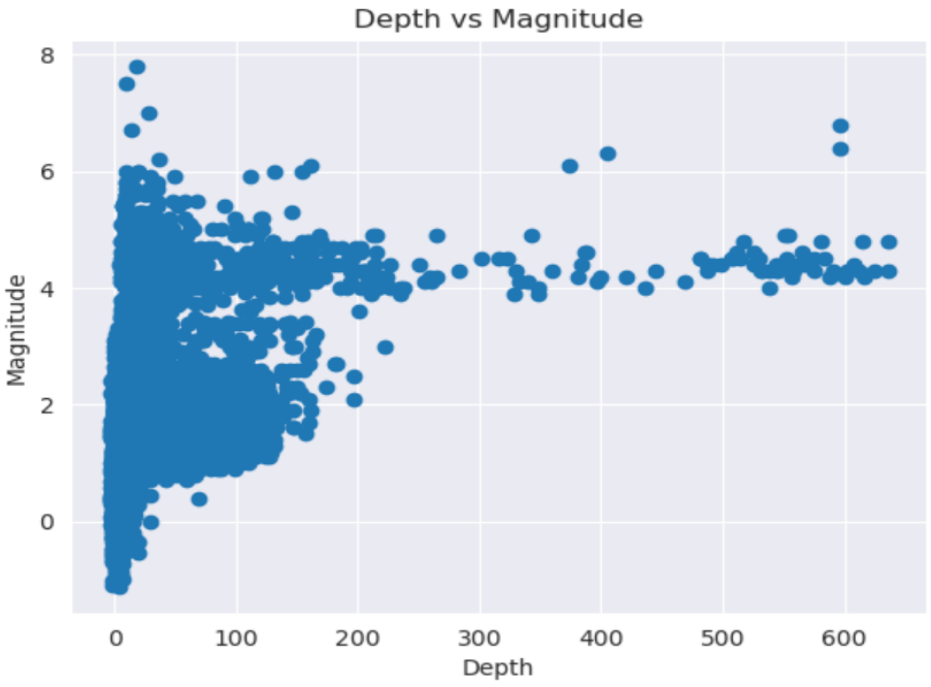
```
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

# Select the features we want to use

X = df[['latitude', 'longitude', 'depth', 'gap']]

y = df['mag']

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fit the model on the training set

model = LinearRegression()

model.fit(X_train, y_train)

# Evaluate the model on the testing set

y_pred = model.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)
```

Mean squared error: 0.7318282185361162

R-squared score: 0.5632730346912534

In[12]:

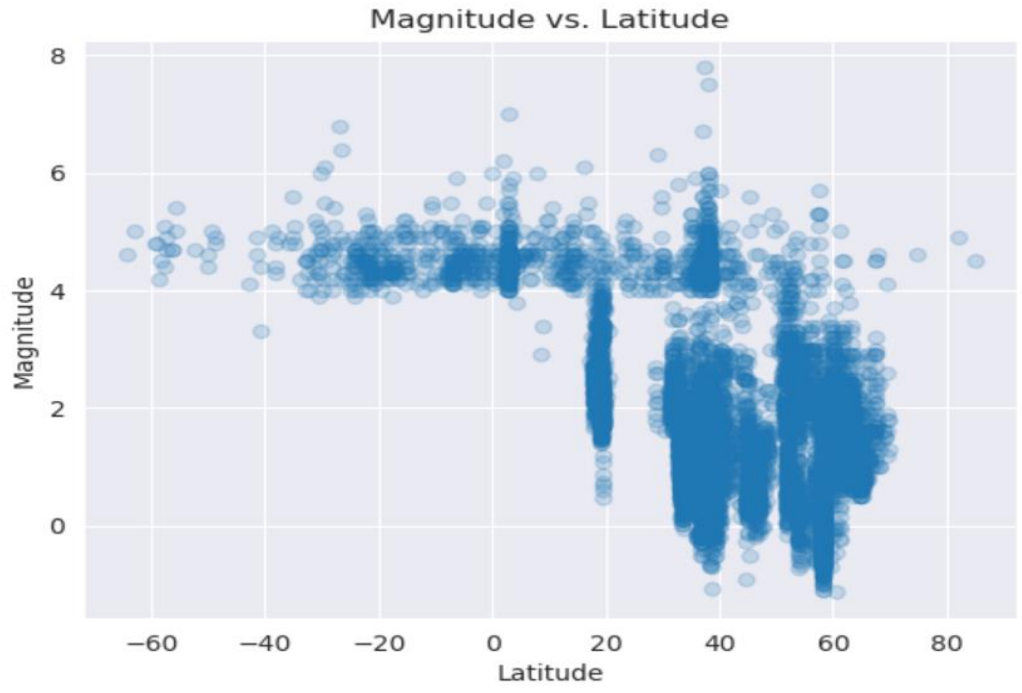
```
import matplotlib.pyplot as plt
plt.scatter(df['depth'], df['mag'])
plt.xlabel('Depth')
plt.ylabel('Magnitude')
plt.title('Depth vs Magnitude')
plt.show()
```

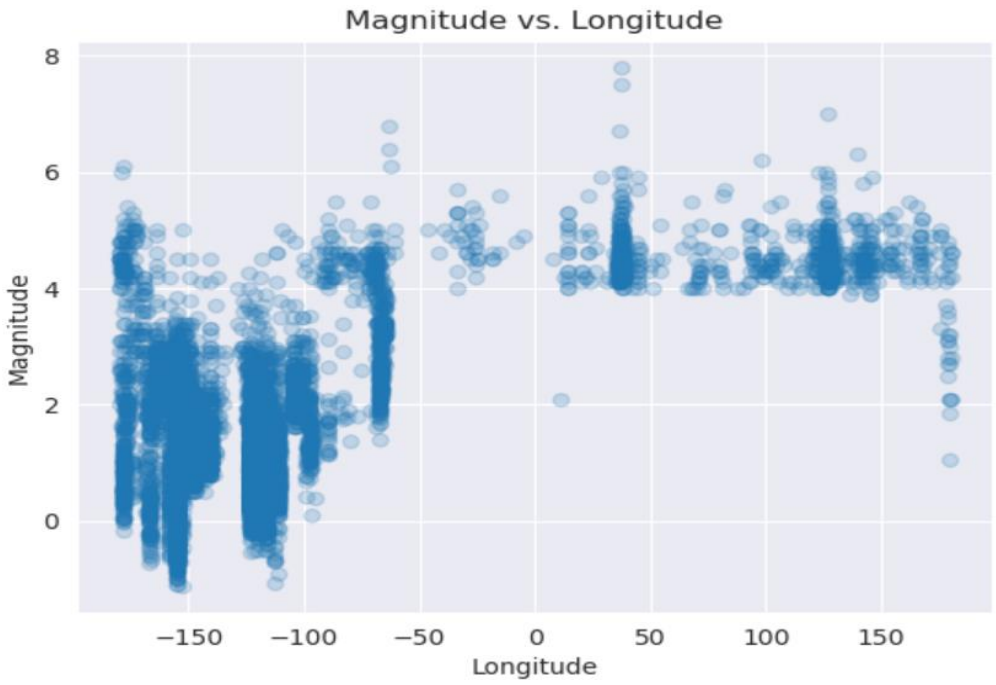


In[13]:

```
import matplotlib.pyplot as plt
import seaborn as sns
# Scatter plot of magnitude vs. latitude
```

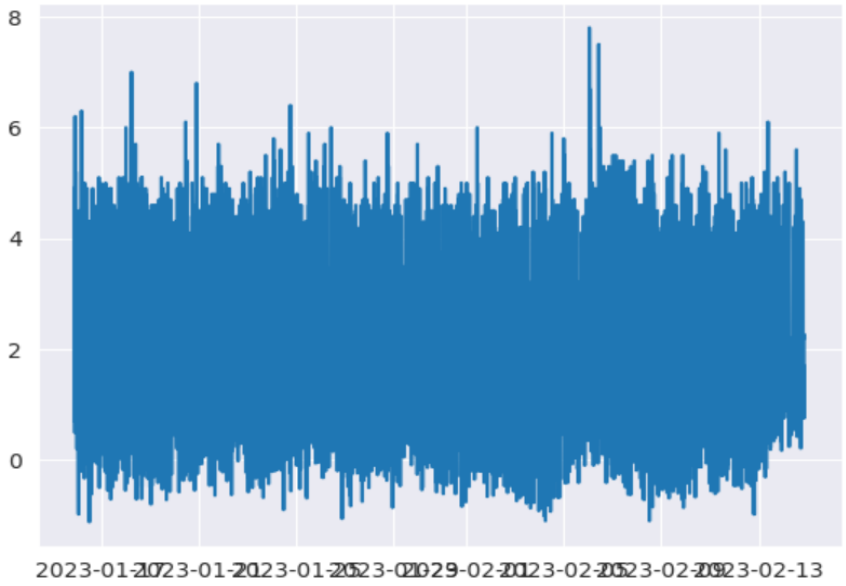
```
plt.scatter(df['latitude'], df['mag'], alpha=0.2)
plt.xlabel('Latitude')
plt.ylabel('Magnitude')
plt.title('Magnitude vs. Latitude')
plt.show()
# Scatter plot of magnitude vs. longitude
plt.scatter(df['longitude'], df['mag'], alpha=0.2)
plt.xlabel('Longitude')
plt.ylabel('Magnitude')
plt.title('Magnitude vs. Longitude')
plt.show()
```





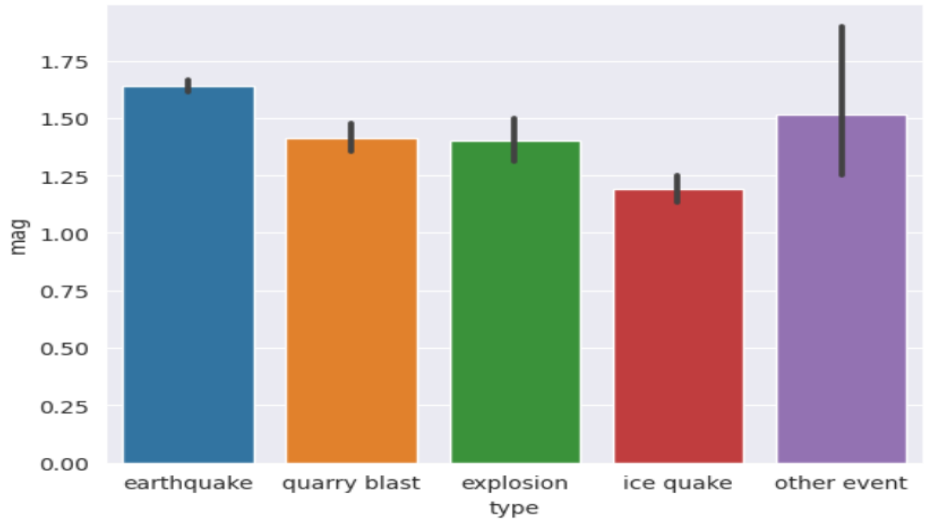
In [14]:

```
# Line plot of magnitude and time
plt.plot(df['time'], df['mag'])
plt.show()
```



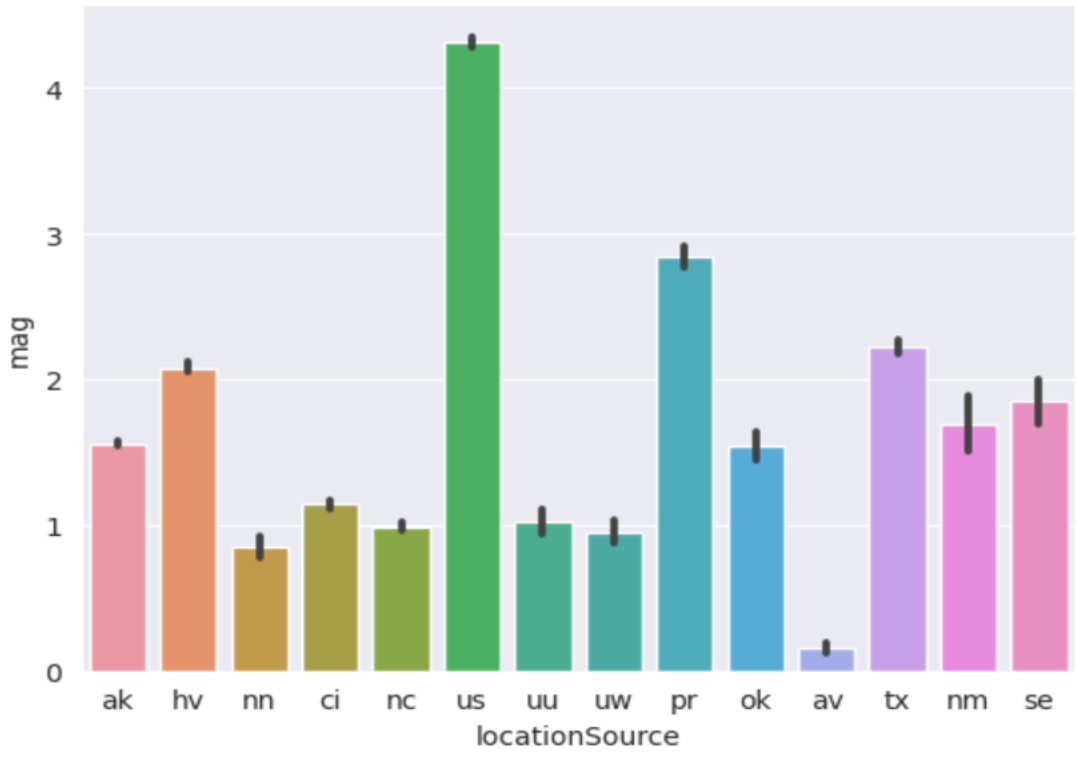
In [15]:

```
sns.barplot(data=df, x='type', y='mag')
plt.show()
```



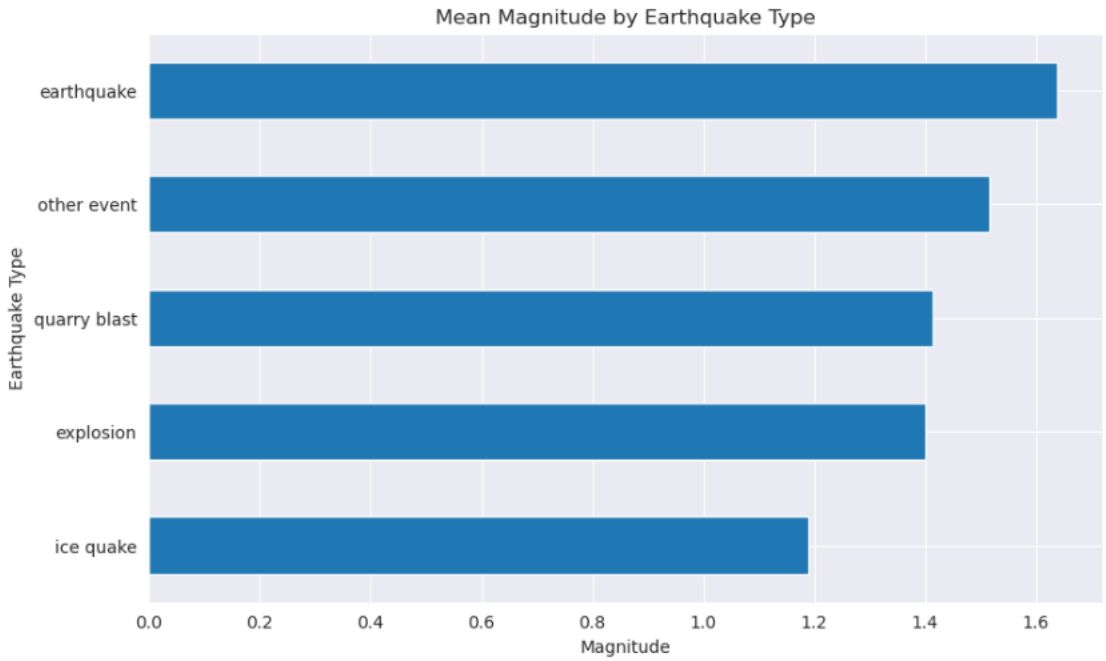
In [16]:

```
sns.barplot(data=df, x='locationSource', y='mag')
plt.show()
```



In [17]:

```
# Create a bar chart of the mean magnitude for each earthquake type
mean_mag_by_type = df.groupby('type')['mag'].mean().sort_values()
mean_mag_by_type.plot(kind='barh', figsize=(10,6))
plt.title('Mean Magnitude by Earthquake Type')
plt.xlabel('Magnitude')
plt.ylabel('Earthquake Type')
plt.show()
```



In [18]:

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

# Prepare the input and target variables for the model

X = df[['latitude', 'longitude', 'depth']]

y = df['mag']


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Random Forest model and fit it to the training data

rf = RandomForestRegressor(n_estimators=100, random_state=42)

rf.fit(X_train, y_train)
```

```
# Use the model to make predictions on the test data
y_pred = rf.predict(X_test)
# Evaluate the model's performance
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)
```

Mean squared error: 0.23816798930658079

R-squared score: 0.8578704939642665

```
latitude = 34.05
longitude = -118.25
depth = 10
year= 2023
import numpy as np
# Convert the predictor data to a numpy array
new_data = np.array([[latitude, longitude, depth, year]])
# Use the trained model to make the prediction
predicted_mag = model.predict(new_data)
print("The predicted magnitude of the earthquake using the Random Forest model is: ", predicted_mag[0])
```

The predicted magnitude of the earthquake using the Random Forest model is: 1.3789007159051776

Feature engineering, model training, evaluation

Earthquake prediction is a complex and challenging task, and while it's not possible to predict earthquakes with high accuracy, you can still work on earthquake-related projects, such as earthquake risk assessment or aftershock forecasting. Here's a high-level overview of how you can perform different activities like feature engineering, model training, and evaluation using Python.

1. Data Collection and Preprocessing:

- Obtain earthquake-related data from sources like USGS Earthquake API or various datasets available online.
- Clean the data, handle missing values, and remove outliers if necessary.

2. Feature Engineering:

- Create meaningful features that could potentially capture patterns in the data, such as:
- Geographic features: latitude, longitude, depth, etc.
- Time-based features: date, time of the day, month, etc.
- Seismic features: magnitude, depth, etc.
- Calculate statistical features like mean, median, standard deviation, etc., for various time windows.

3. Data Visualization:

- Visualize the data to gain insights and understand the relationships between different features.
- Use libraries like Matplotlib, Seaborn, or Plot for creating visualizations.

4. Model Training:

- Choose appropriate machine learning models for earthquake prediction, such as:
- Support Vector Machines (SVM)

- Random Forest
- Gradient Boosting Machines
- Split the data into training and testing sets using techniques like k-fold cross-validation.
- Train the models using the training data.

5. Model Evaluation:

- Evaluate the performance of the models using appropriate metrics, such as:
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared value
- Compare the performance of different models to select the best one.

6. Hyperparameter Tuning:

- Fine-tune the hyperparameters of the chosen model to improve its performance.
- Use techniques like grid search or random search to find the optimal set of hyperparameters.

7. Testing and Deployment:

- Test the final model with unseen data to ensure its generalization capability.
- Deploy the model in a suitable environment, depending on the specific requirements of the application.

Example Code (Data Collection and Preprocessing):

```
import pandas as pd


# Load data
data = pd.read_csv('earthquake_data.csv')
```

```
# Data preprocessing
# Handle missing values
data = data.dropna()

# Remove outliers
# You can use statistical techniques to identify and remove outliers

# Feature Engineering
# Create new features
data['date'] = pd.to_datetime(data['timestamp'], unit='s')
data['month'] = data['date'].dt.month
data['hour'] = data['date'].dt.hour

# Visualize the data
# Use libraries like Matplotlib, Seaborn, or Plotly for visualization

# Model Training
# Split the data into features and target variable
X = data[['latitude', 'longitude', 'depth', 'magnitude', 'month', 'hour']]
y = data['target_variable']

# Train the model
# Choose an appropriate model like Random Forest Regressor or Gradient Boosting Regressor

# Model Evaluation
# Evaluate the performance of the model using suitable metrics
```

Hyperparameter Tuning

Fine-tune the hyperparameters using techniques like Grid Search CV or Randomized Search CV

Testing and Deployment

Test the final model on unseen data and deploy it in a suitable environment

Make sure to choose the appropriate libraries and techniques based on the specific requirements of your project. Also, consider the ethical implications and limitations associated with earthquake prediction.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
print(os.listdir("../input"))
```

```
['database.csv']
```

In [2]:

```
linkcode
data = pd.read_csv("../input/database.csv")
data.head()
```

Out[2]:

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

In [3]:

```
data.columns
```

Out[3]:

```
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')
```

In [4]:

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

Out[4]:

	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

In [5]:

```
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')
```

In [6]:

```
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

In [7]:

```
linkcode
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08
4	11.938	126.427	15.0	5.8	-1.57026e+08

Visualization:

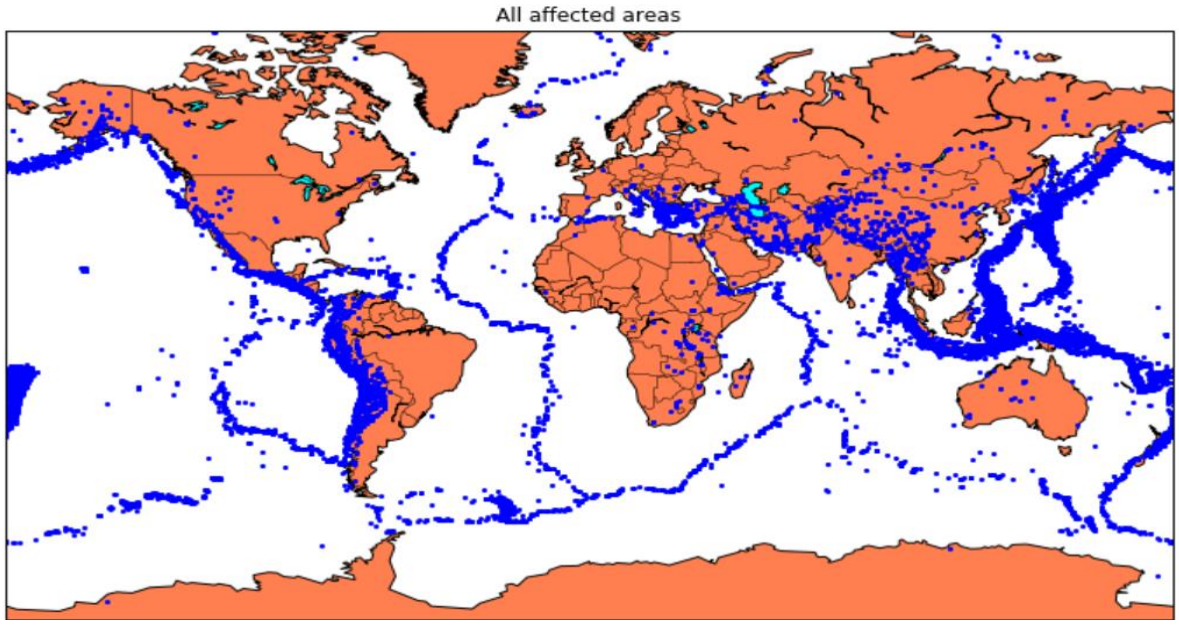
All the earthquakes from the database in visualized on to the world map clear representation of the locations where frequency of the earthquake will be more.

In [8]:

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=18
0,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)
```

In [9]:

```
linkcode
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()
```



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%.

In [10]:

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
```

In [11]:

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

In [12]:

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

Out[12]:

```
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]])
```

In [13]:

```
reg.score(X_test, y_test)
```

Out[13]:

```
0.8614799631765803
```

In [14]:

```
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)
```

Out[14]:

```
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 ]])
```

In [15]:

```
best_fit.score(X_test, y_test)
```

Out[15]:

```
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.

In [16]:

```
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 [17]:

```
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=a
ctivation, optimizer=optimizer, loss=loss)
```

In [18]:

```
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))
```

In [19]:

```
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'])
```

In [20]:

linkcode

```
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_data=(X_t
est, y_test))

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'])
```

In [20]:

```
linkcode
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_data=(X_t
est, y_test))
```

```
<keras.callbacks.History at 0x7ff0a8db8cc0>
```

In [21]:

```
linkcode
model.save('earthquake.h5')
```

Advantage

Predicting earthquakes is a complex and challenging task, and achieving accurate short-term earthquake predictions remains elusive. However, using Python for earthquake prediction research offers several advantages:

1.Data Analysis and Visualization:

Python has a rich ecosystem of libraries for data analysis and visualization. Libraries like NumPy, pandas, Matplotlib, and Seaborn make it easy to process seismic data, identify patterns, and create informative visualizations to gain insights.

2. Machine Learning and Deep Learning:

Python's extensive machine learning libraries, including scikit-learn, TensorFlow, and enable researchers to develop predictive models. Deep learning techniques can be applied to analyze complex patterns in seismic data.

3. Flexibility and Open Source:

Python is an open-source language, which means that researchers have access to a vast community of contributors and a wide range of packages specifically

tailored for earthquake research. It's easy to adapt and extend existing tools to suit specific research needs.

4.Interdisciplinary Collaboration:

Python is a widely used programming language in various scientific fields. It facilitates collaboration between seismologists, geophysicists, data scientists, and machine learning experts, enabling cross-disciplinary research efforts.

5.Geospatial Analysis:

Python libraries such as Geo Pandas and Base map enable researchers to perform geospatial analysis, which is crucial for understanding the spatial distribution of earthquakes and identifying high-risk areas.

6.Efficient Prototyping:

Python is known for its ease of use and rapid development capabilities. Researchers can quickly prototype and test new earthquake prediction models, making it easier to experiment with different approaches.

7. Community Support:

Python has a large and active community of users and developers. This means that researchers can find online resources, tutorials, and support to help with their earthquake prediction projects.

8. Reproducibility:

Python's code is easily shareable and reproducible, which is vital for scientific research. Researchers can share their code, data, and methodologies with others, making it easier to verify and build upon their work.

9. Cross-Platform Compatibility:

Python is cross-platform and can run on various operating systems, which makes it accessible to researchers using different platforms.

10. Access to Data Sources:

Python can be used to access and fetch earthquake-related data from various sources, including government agencies and research institutions, which is crucial for any prediction effort.

Disadvantage:

Using Python for earthquake prediction research offers several advantages, as mentioned in the previous response. However, it's also important to be aware of certain disadvantages and challenges associated with this approach:

1.Limited Predictive Power:

One of the primary disadvantages is the inherent difficulty in accurately predicting earthquakes, especially in the short term. The complex and dynamic nature of seismic events makes it challenging to develop highly accurate prediction models, and Python does not overcome the fundamental limitations of earthquake prediction itself.

2. Data Quality and Availability:

The accuracy of any predictive model relies heavily on the quality and quantity of available data. In some regions, seismic data may be limited or not easily accessible, which can hinder the development of robust prediction models.

3. Model Complexity:

Developing accurate earthquake prediction models often involves complex and computationally intensive algorithms. Python, while versatile, may not always provide the optimal performance needed for processing and analyzing large datasets and running intricate simulations.

4. Data Preprocessing Challenges:

Cleaning and preprocessing seismic data can be a time-consuming and challenging task. Python can help with data preprocessing, but the specific challenges associated with geological and geophysical data can be complex.

5. Model Evaluation:

Accurately assessing the performance of earthquake prediction models can be challenging. Evaluating models is crucial for determining their effectiveness, but it can be difficult to define suitable evaluation metrics for predicting rare and high-impact events like earthquakes.

6. Ethical Considerations:

Earthquake prediction can have significant societal and ethical implications. False alarms or inaccurate predictions can lead to unnecessary panic and evacuations, potentially eroding public trust in the predictions. The ethical implications of using Python for this research should be carefully considered.

7. Resource Intensity:

Developing and running predictive models can be computationally intensive, requiring substantial computational resources. Researchers may need access to high-performance computing clusters or cloud resources, which can be expensive.

8. Interdisciplinary Challenges:

Earthquake prediction research often requires collaboration between experts from various fields, including seismology, geophysics, data science, and machine learning. Bridging the gap between these disciplines can be challenging.

9. Uncertainty and Risk Communication:

Even if a predictive model shows promise, conveying the level of uncertainty to the public, policymakers, and other stakeholders is a challenging task. Misunderstandings and miscommunications can lead to unintended consequences.

10. Continuing Research:

Earthquake prediction remains an active area of research with many unknowns. Python and its libraries are essential tools for research, but they do not eliminate the inherent uncertainties and difficulties in this field.

While Python is a powerful tool for earthquake prediction research, it's essential to recognize its limitations and the broader challenges associated with earthquake prediction as a scientific endeavor. Researchers should approach this field with caution and acknowledge the complexity and limitations of the task at hand.

BENEFITS:

Predicting earthquakes with high accuracy is a challenging problem, and it's important to understand that current scientific knowledge and technology do not allow for precise earthquake prediction. However, there are ways to monitor seismic activity and assess earthquake risk using Python and other tools, which can provide some benefits. Here are some of the potential benefits of using Python for earthquake monitoring and risk assessment:

1. Data Analysis:

Python is a powerful tool for data analysis and manipulation. You can use libraries like NumPy, Pandas, and Matplotlib to process and visualize seismic data, which can help in understanding historical earthquake patterns and trends.

2. Machine Learning:

Python has a rich ecosystem of machine learning libraries, such as Scikit-Learn and TensorFlow, which can be applied to seismic data for predictive modeling. While predicting the exact time and location of earthquakes is challenging, machine learning can help in assessing earthquake risk and identifying potential seismic hotspots.

3. Real-time Monitoring:

Python can be used to develop real-time monitoring systems that collect and analyze data from seismic sensors. This can help in providing early warning systems for earthquakes, which can save lives and reduce damage in some cases.

4. Risk Assessment:

Python can be used to create models that assess earthquake risk based on historical data, geological factors, and other relevant information. This can be valuable for urban planning, construction, and disaster preparedness.

5. Visualization:

Python's data visualization libraries make it easier to create maps, graphs, and visual representations of earthquake data. This can aid in conveying information to the public, researchers, and decision-makers.

6. Opensource Community:

Python is open-source and has a large and active community of developers and researchers. This means that you can find a wealth of resources, code libraries, and tools related to earthquake monitoring and prediction.

7. Integration with Data Sources:

Python can be used to collect and integrate data from various sources, including seismic sensors, GPS data, and geological databases, making it easier to perform comprehensive analyses.

It's important to note that while Python can be a valuable tool in earthquake monitoring and risk assessment, predicting earthquakes with high precision remains a complex and ongoing scientific challenge. Seismic activity is influenced by many factors, and our ability to predict specific events is limited. Nevertheless, Python can assist in improving our understanding of earthquake patterns and enhancing preparedness and mitigation efforts.

Conclusion:

In conclusion, using Python for earthquake prediction and monitoring has several benefits, but it's important to temper expectations and understand the limitations in this field. In practice, earthquake prediction efforts primarily focus on assessing risk and understanding long-term patterns rather than pinpointing exact time and location of future events. Python's capabilities are valuable for enhancing earthquake research, preparedness, and mitigation, but it's essential to manage expectations and work within the constraints of our current scientific knowledge and technology.

PREPARED BY:

GOWRI.K