4222-SURYA GROUP OF INSTITUTIONS
VIKARAVANDI -605 652

NAAN MUDHALVAN PROJECT

SUBJECT CODE - SB3001
Coures Name –Experience Based Practical Learning

EARTHQUAKE PREDICTION USING MODEL PYTHON

PREPARED BY:
S.SUBASRI
REG NO:422221106310
ECE DEPARTMENT

EARTHQUAKE-PREDICTION-USINGPYTHON:

INTRODUCTION:

Earthquake prediction is a challenging and complex task that is still an active area of research. It is a way to predict the magnitude of earthquake based on parameters such as longitude, latitude, depth, and duration magnitude, country. These approaches are based on the analysis of seismic data, historical earthquake data, and other relevant factors. People used to minimize loss of life and property.

ML MODELS USED:

• Linear Regression

• Decision Tree

• K-Nearest Neighbors

STEPS TAKEN:

• Data source

• Feature exploration

• Visualization

• Data splitting

• Training and evaluation

DATA SOUCE:

```
import numpy as npimport pandas as pd
 import matplotlib.pyplot as plt
import os
 print(os.listdir("../input"))
```

SI NO	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

FEATURE EXPLORATION:

```
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.

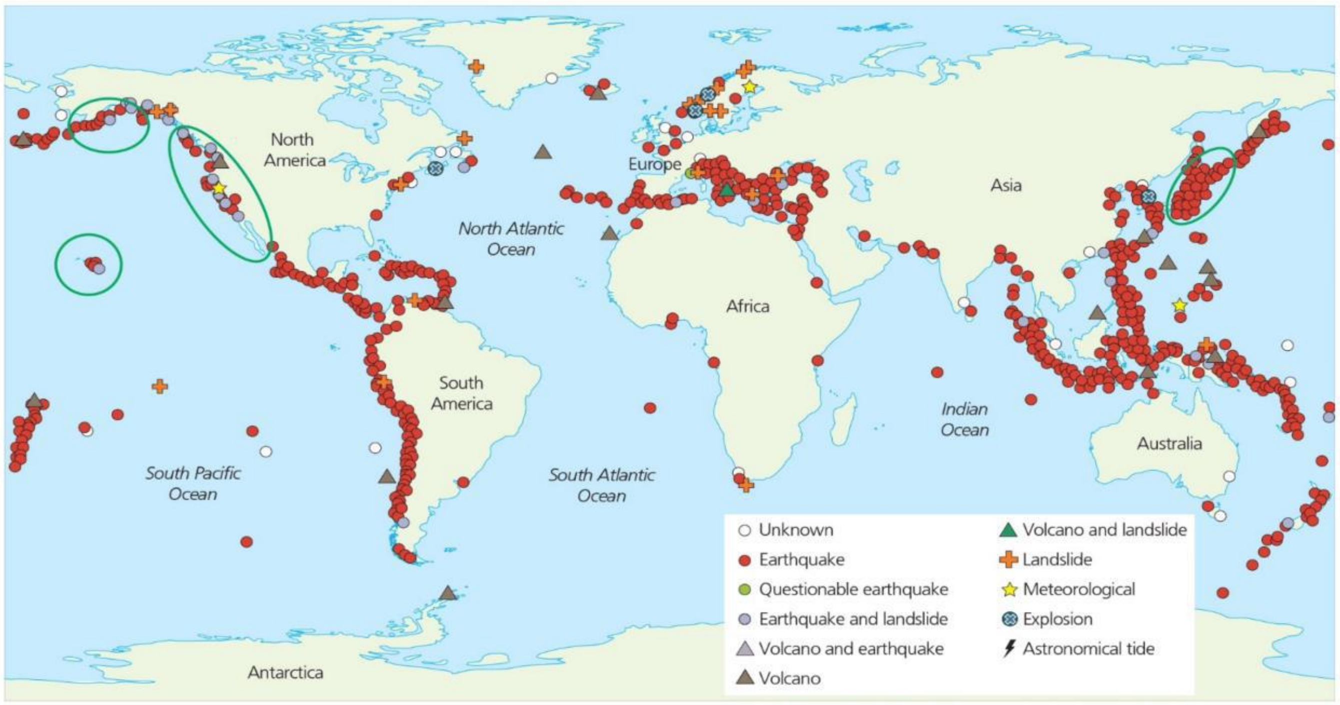
```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```

SI NO	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

Visualization:

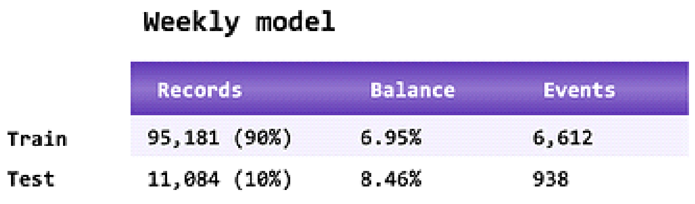
Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

```
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()
```



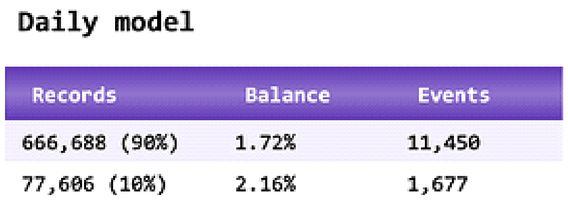
Data splitting:

The data split was 90% train and 10% test.



Weekly model

	Records	Balance	Events
Train	95,181 (90%)	6.95%	6,612
Test	11,084 (10%)	8.46%	938



Daily model

Records	Balance	Events
666,688 (90%)	1.72%	11,450
77,606 (10%)	2.16%	1,677

TRAINING AND EVALUATION

```
# demonstrate that the train-test split procedure is repeatable
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_blobs(n_samples=100)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize first 5 rows
print(X_train[:5, :])
# split again, and we should see the same split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize first 5 rows
print(X_train[:5, :])
```

```
[[-2.54341511 4.98947608]
[ 5.65996724 -8.50997751]
[-2.5072835 10.06155749]
[ 6.92679558 -5.91095498]
[ 6.01313957 -7.7749444 ]]

[[-2.54341511 4.98947608]
[ 5.65996724 -8.50997751]
[-2.5072835 10.06155749]
[ 6.92679558 -5.91095498]
[ 6.01313957 -7.7749444 ]]
```

INNOVATION:

In this phase, we can explore innovative advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's performance and also used ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.

HYPERPARAMETER TUNING:

As we know that there are parameters that are internally learned from the given dataset and derived from the dataset, they are represented in making predictions, classification and etc., These are so-called **Model Parameters,** and they are varying with respect to the nature of the data we couldn't control this since it depends on the data. Like '**m**' and '**C**' in linear equation, which is the value of coefficients learned from the given dataset.Some set of parameters that are used to control the behaviour of the model/algorithm and adjustable in order to obtain an improvised model with optimal performance is so-called hyperparameter.

STEPS FOR EARTHQUAKE PREPROCESSING:

1. **Data Collection:**

Gather raw seismic data from various seismometers and networks.

2. **Data Conversion:**

Convert data from different formats into a standard format for consistency.

3. **Noise Removal:**

Remove noise from the data to enhance the signal-to-noise ratio. This can involve using filters or statistical methods to identify and remove unwanted noise.

4. **Instrument Response Removal:**

Correct for the instrument's response to seismic waves. Seismic instruments often have a specific frequency response that needs to be removed to obtain accurate earthquake signals.

5. **Data Segmentation:**

Divide the continuous data into smaller segments for analysis. Common segments include hours or days.

6. **Event Detection:**

Use algorithms to detect earthquake events within the segmented data. These algorithms can identify patterns associated with seismic events.

7. **Event Association:**

Identify which seismic signals belong to the same earthquake event. This can involve clustering nearby seismic events in both time and space.

8. **Phase Picking:**

Identify the arrival times of seismic waves (P, S, etc.) for each event. This is crucial for locating the epicenter and determining the magnitude.

9. **Hypocenter Location:**

Calculate the earthquake's hypocenter (latitude, longitude, and depth) using the phase arrival times from different seismometers. Various methods, like triangulation, are used for this purpose.

10. **Magnitude Estimation:**

Determine the earthquake's magnitude using the amplitude of seismic waves. Common magnitude scales include Richter scale and moment magnitude scale.

11. **Data Integration:**

Integrate the earthquake location, magnitude, and other relevant information into a database or a suitable format for further analysis and visualization.

12. Quality Control:

Perform quality checks at various stages of the preprocessing to ensure the accuracy and reliability of the results.

```
import pandas as pd import
numpy as np

import matplotlib.pyplot as
plt import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA from mpl_toolkits.mplot3d import Axes3D

df = pd.read_csv("data/train_values.csv")
```

SI NO	building_id	geo_level_1_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	area_percentage	height_percentage
0	802906	6	487	12198	2	30	6	5
1	28830	8	900	2812	2	10	8	7
2	94947	21	363	8973	2	10	5	5
3	590882	22	418	10694	2	10	6	5
4	201944	11	131	1488	3	30	8	9

```
Index(['building_id', 'geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id',
       'count_floors_pre_eq', 'age', 'area_percentage', 'height_percentage',
       'land_surface_condition', 'foundation_type', 'roof_type', 'ground_floor_type',
       'other_floor_type', 'position', 'plan_configuration',
       'has_superstructure_adobe_mud',
       'has_superstructure_mud_mortar_stone', 'has_superstructure_stone_flag',
      'has_superstructure_cement_mortar_stone', 'has_superstructure_mud_mortar_brick',
      'has_superstructure_cement_mortar_brick', 'has_superstructure_timber',
      'has_superstructure_bamboo', 'has_superstructure_rc_non_engineered',
      'has_superstructure_rc_engineered', 'has_superstructure_other',
      'legal_ownership_status', 'count_families', 'has_secondary_use',
      'has_secondary_use_agriculture', 'has_secondary_use_hotel',
      'has_secondary_use_rental', 'has_secondary_use_institution',
    'has_secondary_use_school', 'has_secondary_use_industry',
    'has_secondary_use_health_post', 'has_secondary_use_gov_office',
    'has_secondary_use_use_police', 'has_secondary_use_other'], dtype='object')
```

```
labels = pd.read_csv("data/train_labels.csv")
labels.head()
```

SINO	**building_id**	**damage_grade**
1	802906	3
2	28830	2
3	94947	3
4	590882	2
5	201944	3

```
df = df.sort_values("building_id")
labels = labels.sort_values("building_id")
df.head()
```

SINO	**building_id**	**geo_level_1_id**	**geo_level_2_id**	**geo_level_3_id**	**count_floors_pre_eq**	**age**	**area_percentage**	**height_percentage**
47748	4	30	266	1224	1	25	5	2
212102	8	17	409	12182	2	0	13	7
60133	12	17	716	7056	2	5	12	6
34181	16	4	651	105	2	80	5	4
25045	17	3	1387	3909	5	40	5	10

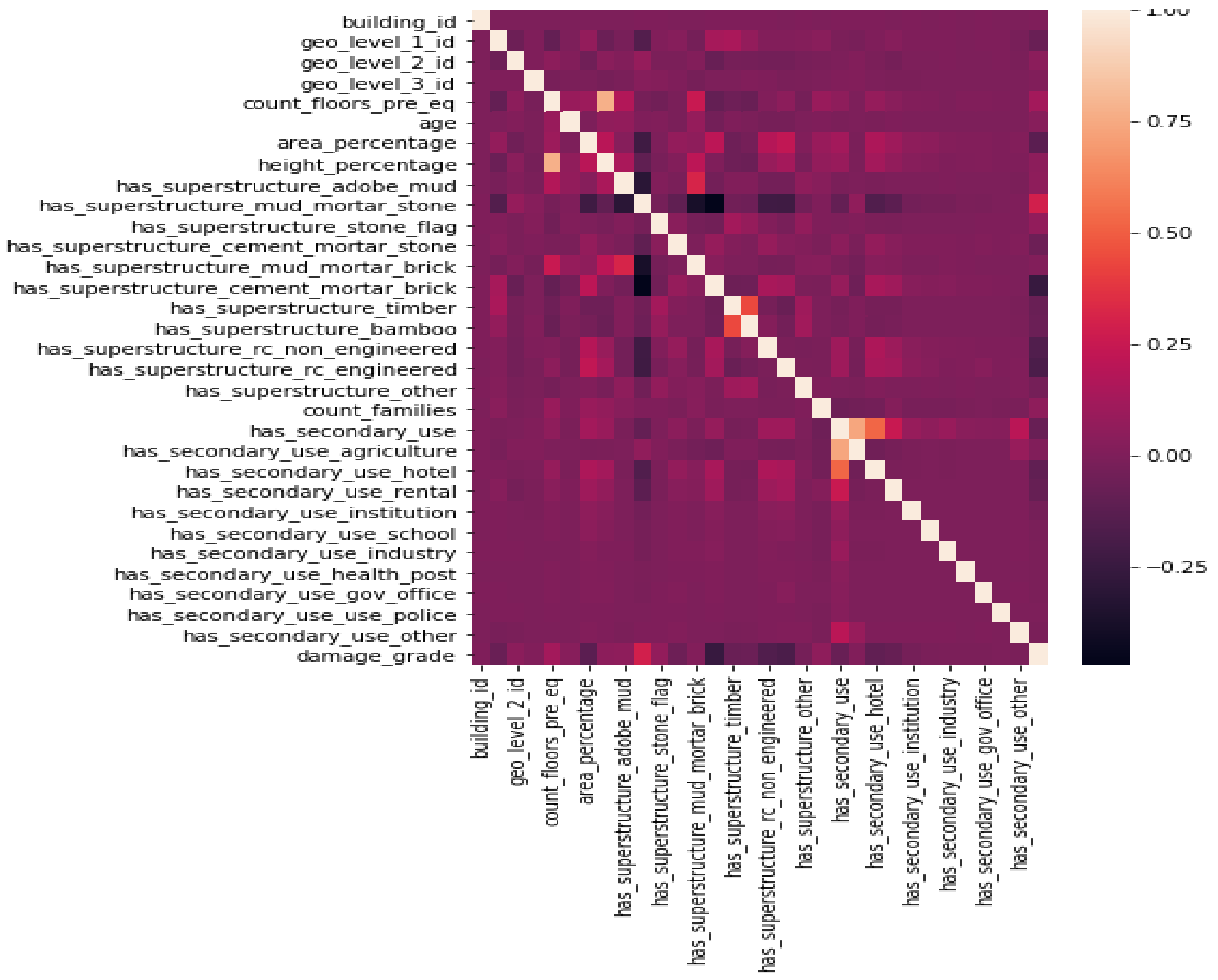
```
labels.head()
```

SINO	**building_id**	**damage_grade**
1	**47748**	2
2	**212102**	3
3	**60133**	3
4	**34181**	2
5	**25045**	2

SINO	buildin g_id	buildin g_id	geo_lev el_2_id	geo_lev el_3_id	count_f loors_p re_eq	age	area_pe rcentag e	height_ percent age
47748	4	30	266	1224	1	25	15	2
212102	8	17	409	12182	2	0	13	7
60133	12	17	716	7056	2	5	12	6
34181	16	4	651	105	2	80	5	4
25045	17	3	1387	3909	5	40	5	10

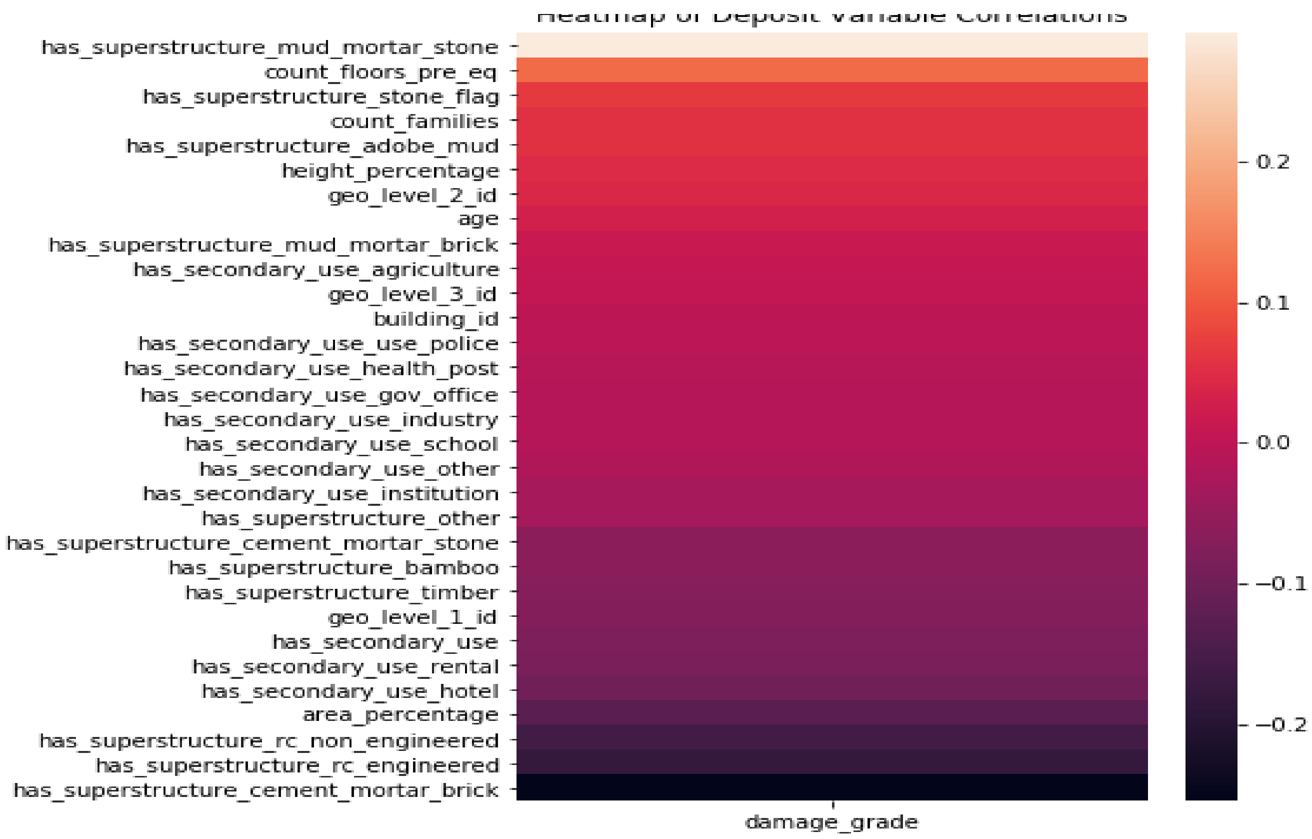
df.to_csv("data/labeled_train.csv")

plt.rcParams["figure.figsize"] = (6,8) sns.heatmap(df.corr())
<matplotlib.axes._subplots.AxesS



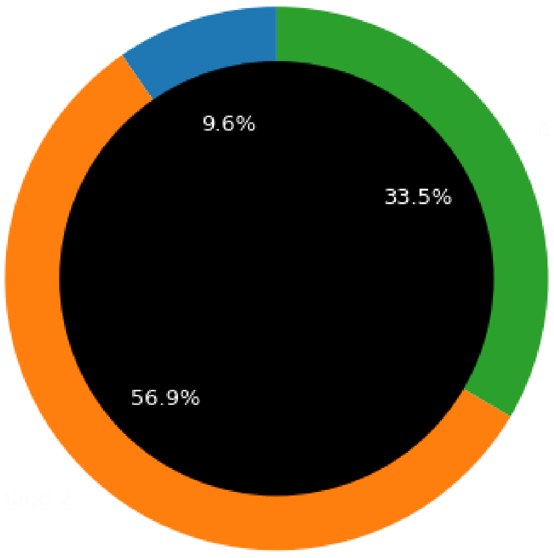
'damage_grade' correlation_matrix = df.corr()
def plot_deposit_correlations(data): '''
Isolates the deposit columns of the correlation matrix and visualize it. '''

```
    deposit_correlation_column =
pd.DataFrame(correlation_matrix[DEPOSIT_COLUMN].drop(DEPOSIT_COLU MN))
    deposit_correlation_column =
deposit_correlation_column.sort_values(by=DEPOSIT_COLUMN, ascending=False)
    sns.heatmap(deposit_correlation_column) plt.title('Heatmap
    of Deposit Variable Correlations')
plot_deposit_correlations(df)
```



```
categorical_vars = ["land_surface_condition", "roof_type", "ground_floor_type",
"other_floor_type", "position", "plan_configuration", "legal_ownership_status"] for var in
categorical_vars:
    df = fuck_naman(df, var)


df.to_csv('data/labeled_train.csv') df["damage_grade"]


2    148259
3     87218
1     25124
Name: damage_grade, dtype: int64

# Pie chart

labels = ['Damage 1', 'Damage 2', 'Damage 3']
sizes = [25124, 148259, 87218,]
```

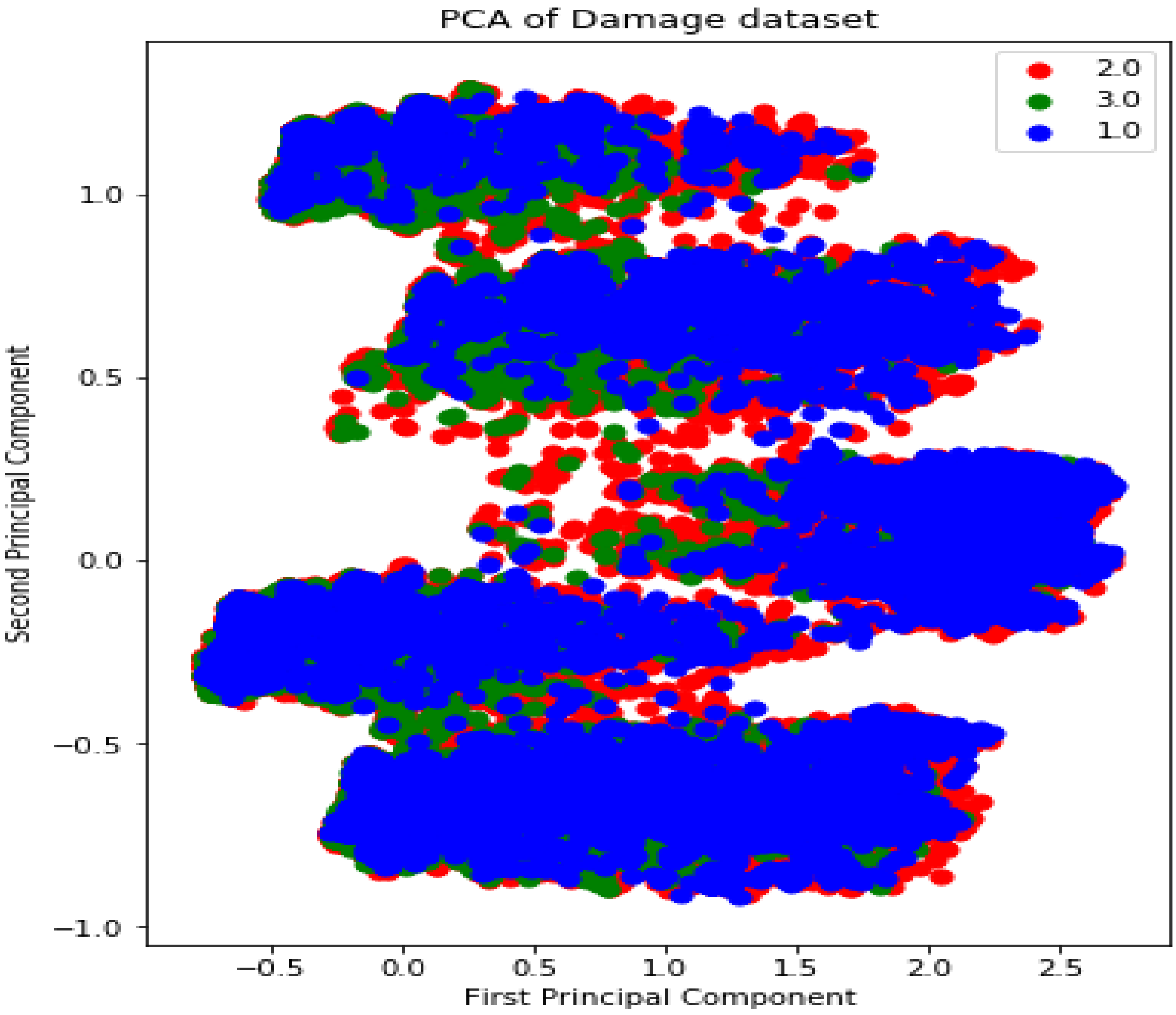
```
# only "explode" the 2nd slice (i.e. 'Hogs') explode = (0,
0, 0)
fig1, ax1 = plt.subplots()
patches, texts, autotexts = ax1.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
for text in texts:
    text.set_color('white')
    text.set_size(13)
for autotext in autotexts:
    autotext.set_color('white')
    autotext.set_size(13)
#draw circle
centre_circle = plt.Circle((0,0),0.80,fc='black') fig =
plt.gcf() fig.gca().add_artist(centre_circle)
# Equal aspect ratio ensures that pie is drawn as a circle ax1.axis('equal')
plt.tight_layout() plt.show()
value_counts()
```



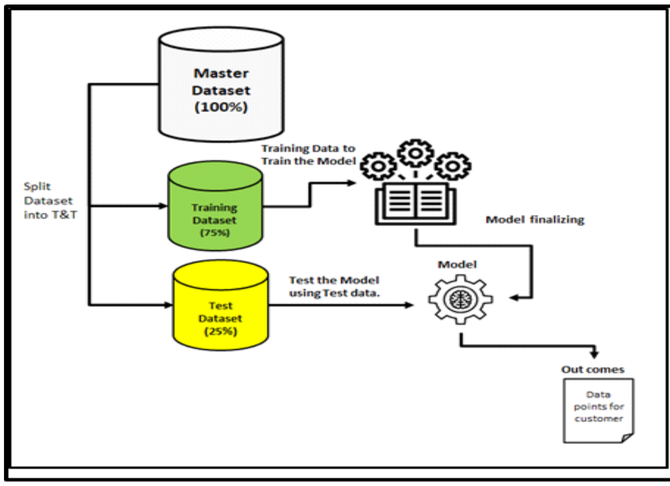
```
normalized_df=(df-df.min())/(df.max()-df.min()) df =
normalized_df


X = df.drop("damage_grade", axis=1) y =
df["damage_grade"]
targets   =   df["damage_grade"].unique()
print(targets)
pca = PCA(n_components=2) X_r
=          pca.fit(X).transform(X)
print(X_r.shape)
PCA_Df = pd.DataFrame(data = X_r
          , columns = ['principal component 1', 'principal component 2'])
print(PCA_Df.head())
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors): indicesToKeep =
    df['damage_grade'] == target
    plt.scatter(PCA_Df.loc[indicesToKeep, 'principal component 1']
              , PCA_Df.loc[indicesToKeep, 'principal component 2'], c = color, s = 50)
plt.legend((targets + .5) * 2)
plt.title('PCA of Damage dataset')

plt.xlabel("First Principal Component") plt.ylabel('Second
Principal Component')
```

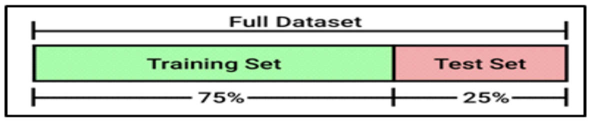
```
[0.5 1.  0. ]
(260601, 2)
   principal component 1  principal component 2
0               0.029283              -0.653984
1              -0.605595              -0.171097
2              -0.417904               1.041256
3              -0.719406              -0.306778
4              -0.102610              -0.187304
```

```
Text(0, 0.5, 'Second Principal Component')
```



HYPERPARAMETER LIFE CYCLE:





Hyperparameter Space

As we know that there is a list of HPs for any selected algorithm(s) and our job is to figure out the best combination of HPs and to get the optimal results by tweaking them strategically, this process will be providing us with the platform for **Hyperparameter Space** and this combination leads to provide the best optimal results, no doubt in that but finding this combo is not so easy, we have to search throughout the space. Here every combination of selected HP value is said to be the "**MODEL**" and have to evaluate the same on the spot. For this reason, there are two generic approaches to search effectively in the HP space are **GridSearch CV** and **RandomSearch CV.** Here CV denotes **Cross-Validation.**

DATA LEAKAGE:

Well! Now quickly will understand what is Data leakage in ML, this is mainly due to not following some of the recommended best practices during the Data Science/Machine Learning life cycle.

The resulting

is Data Leakage, that's fine, what is the issue here, after successful testing with perfect accuracy followed by training the model then the model has been planned to move into production. At this moment ALL Is Well.

Still, the actual/real-time data is applied to this model in the production environment, you will get poor scores. By this time, you may think that why did this happen and how to fix this. This is all because of the data that we split data into training and testing subsets. During the training the model has the knowledge of data, which the model is trying to predict, this results in inaccurate and bad prediction outcomes after the model is deployed into production.

Causes of Data Leakage

Data Pre-processing

The major root cause is doing all EDA processes before splitting the dataset into test and train

<!--[endif]-->Doing straightforward normalizing or rescaling on a given dataset

<!--[endif]-->Performing Min/Max values of a feature

<!--[endif]-->Handling missing values without reserving the test and train

<!--[endif]-->Removing outliers and Anomaly on a given dataset

<!--[endif]-->Applying standard scaler, scalin

Data Leakage:

Well! Now quickly will understand what is Data leakage in ML, this is mainly due to not following some of the recommended best practices during the Data Science/Machine Learning life cycle. The resulting is Data Leakage, that's fine, what is the issue here, after successful testing with perfect accuracy followed by training the model then the model has been planned to move into production. At this moment ALL Is Well. Still, the actual/real-time data is applied to this model in the production environment, you will get poor scores. By this time, you may think that why did this happen and how to fix this. This is all because of the data that we split data into training and testing subsets. During the training the model has the knowledge of data, which the model is trying to predict, this results in inaccurate and bad prediction outcomes after the model is deployed into production.

Causes of Data Leakage
Data Pre-processing
The major root cause is doing all EDA processes before splitting the dataset into test and train
<!--[endif]-->Doing straightforward normalizing or rescaling on a given dataset
<!--[endif]-->Performing Min/Max values of a feature
<!--[endif]-->Handling missing values without reserving the test and train
<!--[endif]-->Removing outliers and Anomaly on a given dataset
<!--[endif]-->Applying standard scaler, scaling, assert normal distribution on the full dataset
Bottom line is, we should avoid doing anything to our training dataset that involves having knowledge of the test dataset. So that our model will perform in production as a generalised model.
will go through the available Hyperparameters across the various algorithms and how we could implement all these factors and impact the model.

Steps to Perform Hyperparameter Tuning

Select the right type of model.
<!--[endif]-->Review the list of parameters of the model and build the HP space
<!--[endif]-->Finding the methods for searching the hyperparameter space
<!--[endif]-->Applying the cross-validation scheme approach

GRIDE SEARCH:

The **Grid Search** one that we have discussed above usually increases the complexity in terms of the computation flow, So sometimes GS is considered inefficient since it attempts all the combinations of given hyperparameters. But the **Randomized Search** is used to train the models based on random hyperparameters and combinations. obviously, the number of training models are small column than grid search.
In simple terms, In Random Search, in a given grid, the list of hyperparameters are trained and test our model on a random combination of given hyperparameters.
Getting RandomForestClassifier object for my operation.
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import randint as sp_randint
Assigning my Train and Test spilt to my RandomForestClassifier object

```
# build a RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50)
```

Specifying the list of parameters and distributions

```
param_dist = {"max_depth": [3, None],
"max_features": sp_randint(1, 11),
"min_samples_split": sp_randint(2, 11),
"min_samples_leaf": sp_randint(1, 11),
"bootstrap": [True, False],
"criterion": ["gini", "entropy"]}
```

Defining the sample, distributions and cross-validation

```
samples = 8 # number of random samples
randomCV = RandomizedSearchCV(clf, param_distributions=param_dist,
n_iter=samples,cv=3)
```

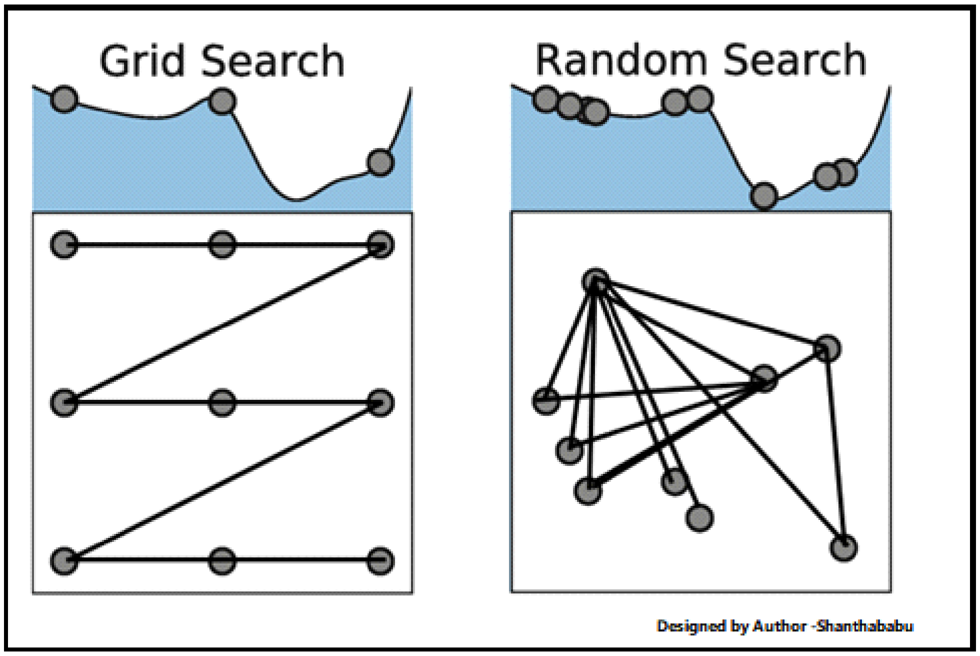
All parameters are set and, let's do the fit model

```
randomCV.fit(X, y)
print(randomCV.best_params_)
```

Output
```
{'bootstrap': False, 'criterion': 'gini', 'max_depth': 3, 'max_features': 3,
'min_samples_leaf': 7, 'min_samples_split': 8}
```
As per the Cross-Validation process, will figure out the mean and get the results
```
randomCV.cv_results_['mean_test_score']
```
Output
```
array([0.73828125, 0.69010417, 0.7578125 , 0.75911458, 0.73828125,
```
nan, nan, 0.7421875])

Best accuracy from training
```
print(randomCV.score(X_test,y_test))
```
Output
```
0.8744588744588745
```
You may have a question, now which technique is best to go. The straight answer is RandomSearshCV, let's

The blow pictorial representation would give you the best understanding of GridSearchCV and RandomSearshCV



STEP TAKEN:

1. Splitting the Datasets
2. Building ML models And Data Analysis
3. Visualization
4. Prediction

1.SPLITTING THE DATASETS:

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

2.Building ML models And Data Analysis:

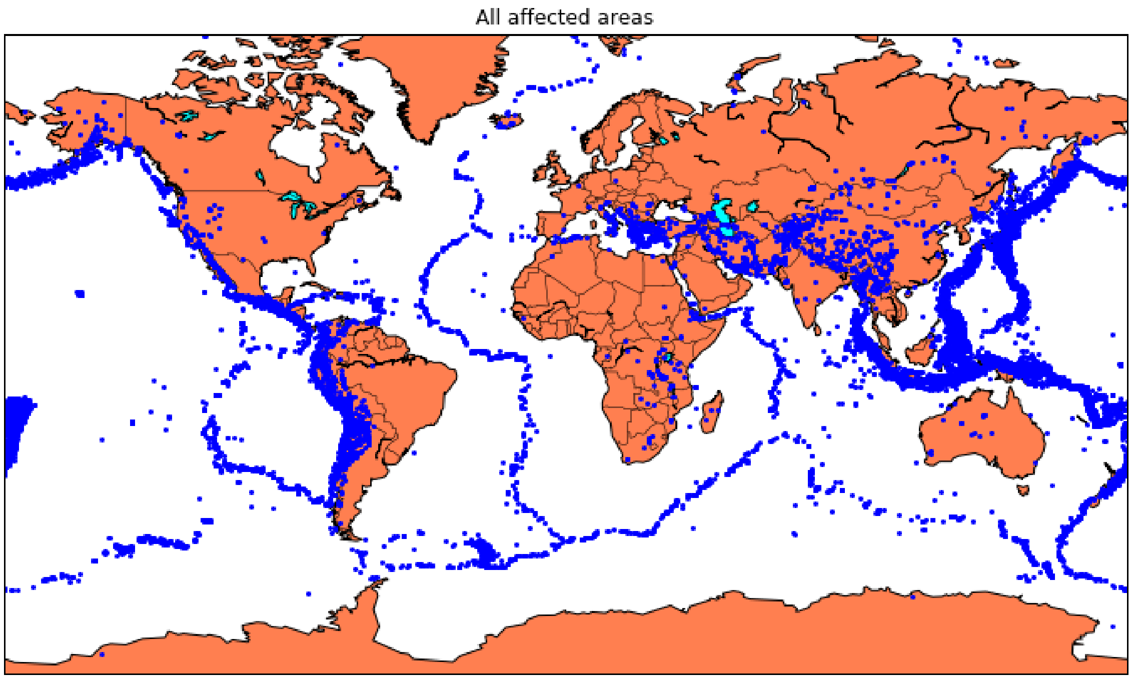
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

	Date	Time	Latitude	Langitude	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

3.Visualization:

```
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)
```

```
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()
```



4.Prediction:

```
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()
```

	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

Data Description:

The acoustic_data input signal is used to predict the time remaining before the next laboratory earthquake (time_to_failure).The training data is a single, continuous segment of experimental data. The test data consists of a folder containing many small segments. The data within each test file is continuous, but the test files do not represent a continuous segment of the experiment; thus, the predictions cannot be assumed to follow the same regular pattern seen in the training file.

Model Building:

The following predictive models will be built:

1. KNN 2.Random Forest 3.Gradient Boosted Machine

The model which best predicts the damage grade on new data given the input features. Within this section, a pre-processing pipeline will be set up to prepare the data for training. A test dataset will be extracted from the main data to give us the opportunity to assess how the models perform on completely new data. The models will be evaluated on the test data using the evaulation metric F1.

Creating a Test Dataset:

A test set is created to help evaluate the performance of predictive models which will be developed in this section. It provides the opportunity to assess how the model performs on unseen data. The test set will account for 20% of the observations, choosen at random but stratified around the target variable to ensure that the proportions are the same in the training and testing data.

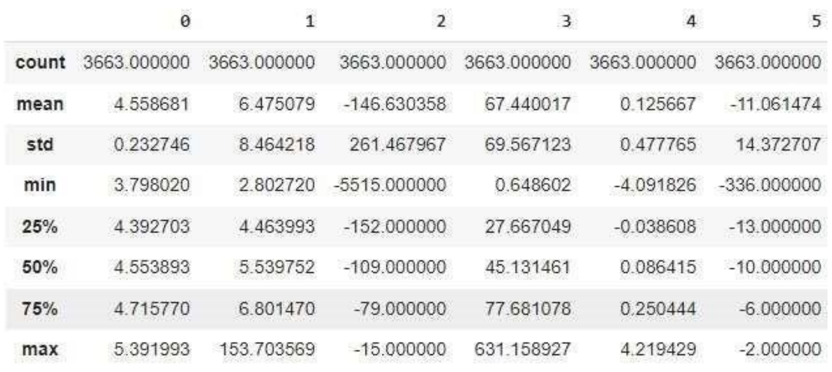
The table below shows that this has been achieved and we cann see that the training and test dataframes have very similar proportions of observations for each damage grade

Exploratory Data Analysis:

```
train = pd.read_csv('train.csv', nrows=6000000, dtype={'acoustic_data' : np.int16, 'time_to_failure':np.float64})
train.head(10)
```

Feature Engineering:

```
def gen_features(X):
strain=[] strain.append(X.mean()) strain.append(X.std()) strain.append(X.min())
strain.append(X.kurtosis()) strain.append(X.skew()) strain.append(np.quantile(X, 0.01))
return
pd.Series(strain) train = pd.read_csv('train.csv', iterator=True,
chunksize=150_000, dtype={'acoustic_data': np.int16, 'time_to_failure':
np.float64})
X_train = pd.DataFrame() y_train = pd.Series() for df in train:
ch = gen_features(df['acoustic_data'])
X_train = X_train.append(ch, ignore_index=True) y_train =
y_train.append(pd.Series(df['time_to_failure'].values[-1]))
```



	0	1	2	3	4	5
count	3663.000000	3663.000000	3663.000000	3663.000000	3663.000000	3663.000000
mean	4.558681	6.475079	-146.630358	67.440017	0.125667	-11.061474
std	0.232746	8.464218	261.467967	69.567123	0.477765	14.372707
min	3.798020	2.802720	-5515.000000	0.648602	-4.091826	-336.000000
25%	4.392703	4.463993	-152.000000	27.667049	-0.038608	-13.000000
50%	4.553893	5.539752	-109.000000	45.131461	0.086415	-10.000000
75%	4.715770	6.801470	-79.000000	77.681078	0.250444	-6.000000
max	5.391993	153.703569	-15.000000	631.158927	4.219429	-2.000000

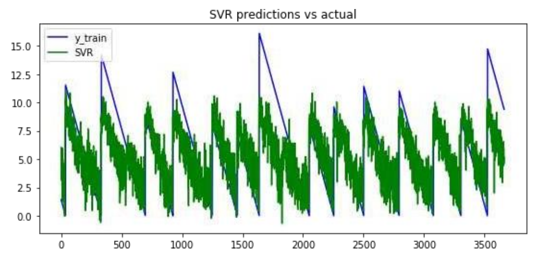
Learning Model Implementation:

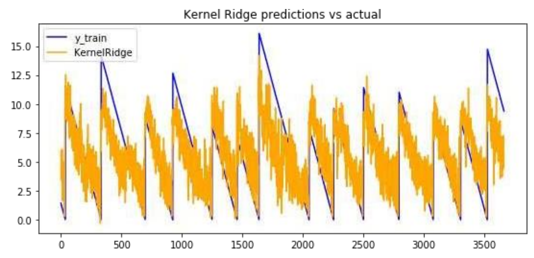
1. SUPPORT VECTOR MACHINES

```
scaler = StandardScaler() scaler.fit(X_train) X_train_scaled =
scaler.transform(X_train) parameters =[{'gamma': [0.001, 0.005,
0.01, 0.02, 0.05, 0.1],
               'C': [0.1, 0.2, 0.25, 0.5, 1, 1.5, 2]}]
#'nu': [0.75, 0.8, 0.85, 0.9, 0.95, 0.97]}]   reg1 =
GridSearchCV(SVR(kernel='rbf', tol=0.01), parameters, cv=5,
scoring='neg_mean_absolute_error') reg1.fit(X_train_scaled, y_train.values.flatten()) y_pred1 =
reg1.predict(X_train_scaled) print("Best CV score:
{:.4f}".format(reg1.best_score_)) print(reg1.best_params_)


  parameters = [{'gamma': np.linspace(0.001, 0.1,10),
               'alpha': [0.005, 0.01, 0.02, 0.05, 0.1]}]   reg2 =
GridSearchCV(KernelRidge(kernel='rbf'), parameters, cv=5,
scoring='neg_mean_absolute_error') reg2.fit(X_train_scaled, y_train.values.flatten()) y_pred2 =
reg2.predict(X_train_scaled) print("Best CV score:
{:.4f}".format(reg2.best_score_)) print(reg2.best_params_)
```

Graphical Analysis of Model Performance:

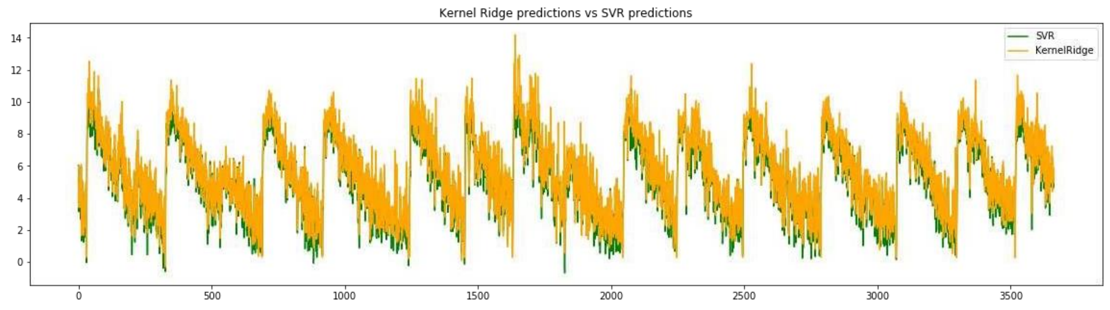
```
Graphical Analysis of Model Performance
```





```
plt.figure(figsize=(20, 5)) plt.plot(y_pred1,
color='green', label='SVR') plt.plot(y_pred2,
```

```
color='orange', label='KernelRidge') plt.legend() plt.title('KernelRidge predictions vs
SVR predictions') plt.show()
```



Earthquake Prediction Analysis

Research Question: What are the main factors that contribute to the occurrence of earthquakes?

Based on our result and dataset, we can perform further analysis to identify the main factors that contribute to the occurrence of earthquakes. We can use techniques such as regression analysis and correlation analysis to identify the factors that are most strongly associated with earthquake occurrence. Additionally, we can use machine learning algorithms such as decision trees or random forests to identify the most important features in predicting earthquakes. Here's an outline of the steps we can take for regression analysis:

Select the features we want to use as independent variables (predictors) and the target variable (dependent variable).

Split the data into training and testing sets.

Fit the model on the training set.

Evaluate the model on the testing set.

For the target variable, we can use the 'mag' column since that represents the magnitude of the earthquake.

For the independent variables, we can use some combination of the other columns in the dataset, such as 'latitude', 'longitude', 'depth', and 'gap'. We can experiment with different combinations to see which ones give the best results.

For the regression model, we can use a linear regression model.

This code in the below selects the 'latitude', 'longitude', 'depth', and 'gap' columns as the independent variables, and the 'mag' column as the target variable. It then splits the data into training and testing sets, fits a linear regression model on the training set, and evaluates the model on the testing set using mean squared error and R-squared score.

```
port 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-
squaredscore: 0.5632730346912534
```

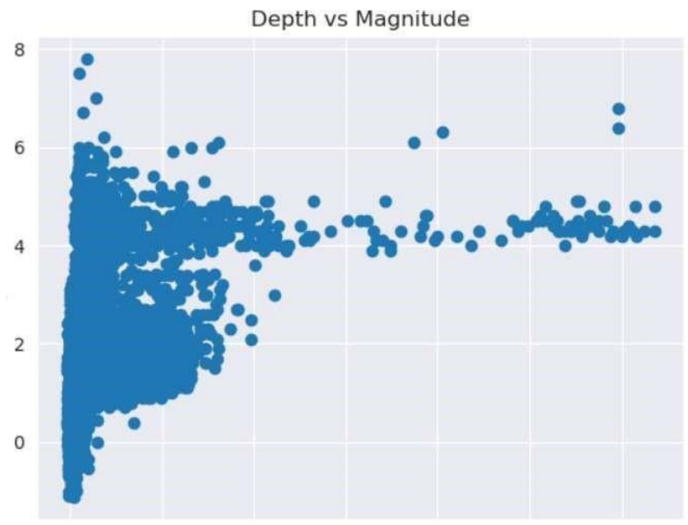
the mean squared error and the R-squared score will help us evaluate the performance of our regression model. The mean squared error represents the average squared difference between the predicted values and the actual values. A lower value indicates a better fit of the model. The R-squared score measures how well the model fits the data, with values closer to 1 indicating a better fit.

Based on the output, the mean squared error is 0.73, which is a relatively low value indicating a good fit. The R-squared score is 0.56, which indicates that our model explains about 56% of the variability in the data, which is not too bad but leaves room for improvement.

Next, we can visualize the relationship between the earthquake magnitude and the depth of the earthquake using a scatter plot to better understand the relationship between these variables.

```
import matplotlib.pyplot as plt
```

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

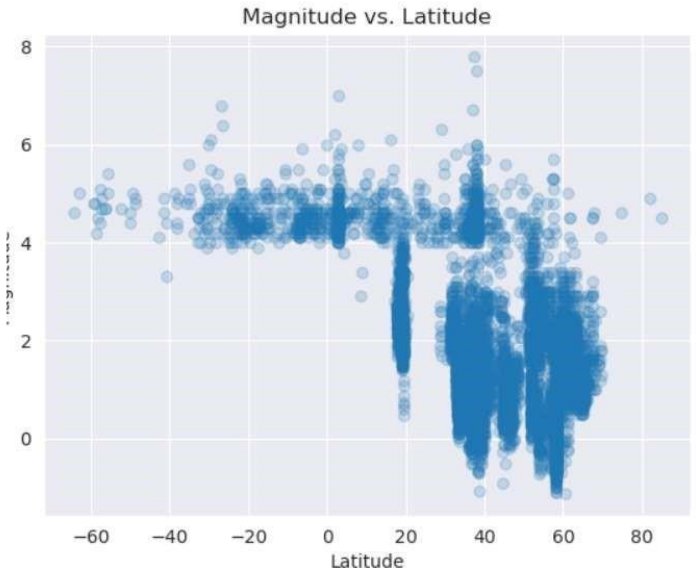


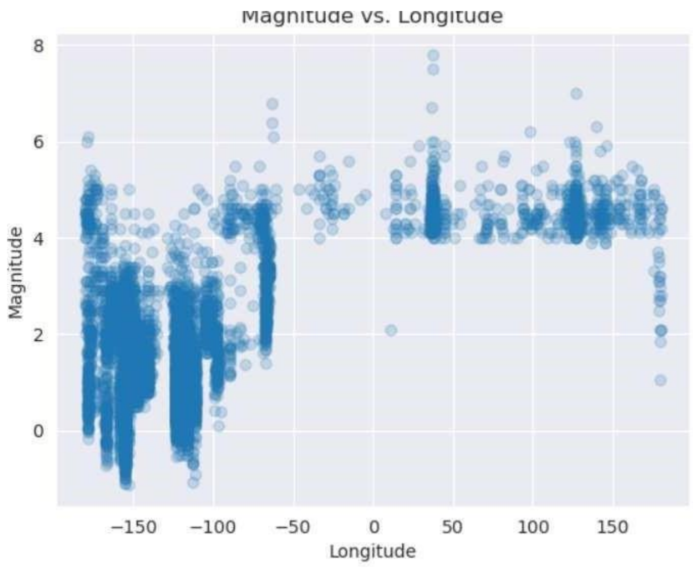
the scatterplot helps visualize the relationship between depth and magnitude. It appears that as the depth decreases, the occurrence of earthquakes with higher magnitudes increases, and as the depth increases, the occurrence of earthquakes with lower magnitudes increases.

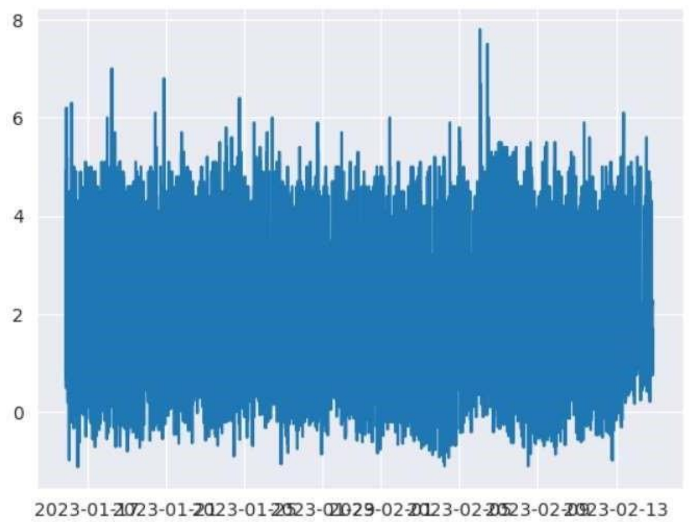
This suggests that the depth of an earthquake is an important factor that contributes to the occurrence of earthquakes, and could potentially be used in earthquake prediction models.

```
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()
```

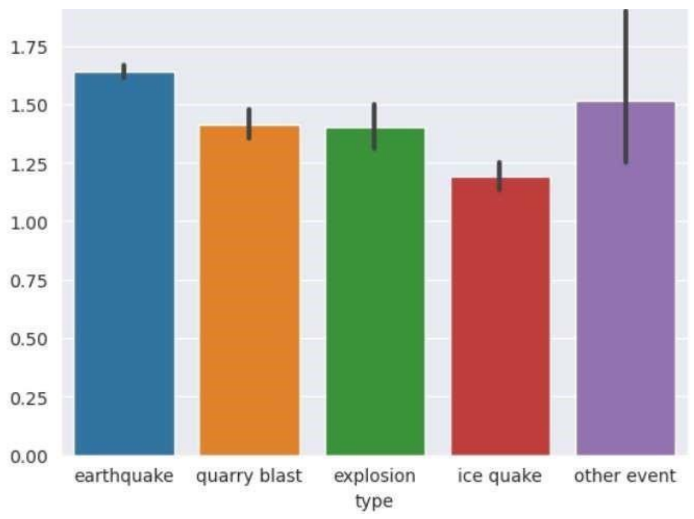
```
# Scatter plot of magnitude vs. longitude
plt.scatter(df['longitude'], df['mag'], alpha=0.2)
plt.xlabel('Longitude') plt.ylabel('Magnitude')
plt.title('Magnitude vs. Longitude') plt.show()
```



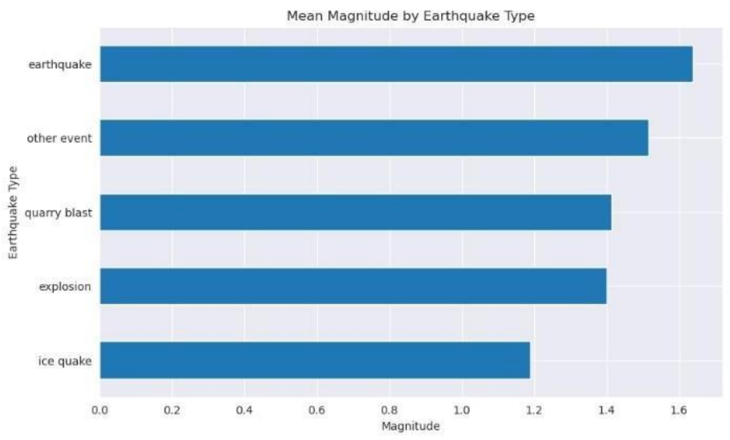




Bar chart of magnitude and type
```
sns.barplot(data=df, x='type', y='mag')plt.show()
```



```
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()
```



CONCLUSION:

In conclusion, developing an accurate earthquake prediction model remains a complex and ongoing challenge. While significant progress has been made in understanding seismic patterns and precursor signals, the inherent unpredictability of earthquakes makes it difficult to create a foolproof prediction system. Continued research, collaboration between scientists and data analysts, and advancements in technology are crucial to improving the reliability of earthquake prediction models. Although we may not yet have a perfect solution, the pursuit of this knowledge is vital for enhancing our ability to mitigate the potential impact of earthquakes on vulnerable communities.

FUTURE WORK:

The future work in the field of earthquake prediction systems involves a multidisciplinary approach and continuous technological advancements. Here are some key areas for future research and development in earthquake prediction systems:

1. Advanced Sensor Technologies: Develop more sensitive and cost-effective sensor technologies to detect subtle changes in the Earth's crust. Advancements in sensor networks, including IoT devices and satellite technology, can enhance data collection capabilities.

2. Data Integration and Fusion: Integrate data from various sources, such as seismic sensors, GPS, satellite imagery, and social media, using advanced data fusion techniques. Combining different data types can provide a comprehensive understanding of seismic activities and improve prediction accuracy.

3. Machine Learning and AI: Explore advanced machine learning algorithms, including deep learning, reinforcement learning, and neural networks, to analyze complex patterns in large-scale seismic data. AI techniques can assist in detecting subtle precursors and predicting earthquake occurrences more accurately.

4. Real-Time Data Processing: Develop real-time data processing systems capable of handling vast amounts of data quickly and efficiently. Implement algorithms for real-time analysis, enabling timely earthquake alerts and warnings to be disseminated to the affected regions.

5. Earthquake Early Warning (EEW) Systems: Enhance existing EEW systems to provide faster and more reliable warnings to people and infrastructure in earthquake-prone areas. Improve the communication infrastructure and develop user-friendly mobile applications for delivering real-time alerts to the public.

6. Uncertainty Quantification: Research methods to quantify and communicate uncertainties associated with earthquake predictions. Understanding the reliability and confidence levels of predictions is crucial for decision-making and risk assessment.

7. Community Engagement: Involve local communities in earthquake monitoring efforts. Citizen science initiatives and community-based sensor networks can provide valuable data and enhance the coverage of seismic monitoring in regions with limited resources.

8. High-Performance Computing .