4222-SURYA GROUP OF INSTITUTIONS

VIKARAVANDI -605 652

SUBJECT CODE-SB3001

COURSE NAME-EXPERIENCE BASED PRACTICAL LEARNING

PROJECT NAME:

EARTHQUAKE-PREDICTION- MODEL-USING-PYTHON

PREPARED BY:

K.ARUN

REG NO:422221106003

ECE DEPARTMENT

3RD-YEAR-5TH SEMESTER

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. Predicting earthquakes with high accuracy is a challenging task, and current scientific knowledge doesn't allow for precise short-term earthquake predictions. However, you can build models to analyze earthquake data and estimate earthquake probabilities in specific regions. This can be useful for assessing earthquake risk and preparedness. In this response, I'll provide a simplified example of creating an earthquake probability model using Python.

Step 1: Data Collection

You'll need earthquake data for your model. You can obtain earthquake data from sources like the USGS Earthquake Catalog or other relevant repositories.

Step 2: Data Preprocessing

Once you have the data, you'll need to preprocess it. This typically includes cleaning and transforming the data into a format suitable for modeling. Common preprocessing steps include filtering for relevant features and removing duplicates.

Step 3: Feature Engineering

Extract relevant features from your earthquake data. Features might include location, depth, magnitude, and time. You can also include geological and seismological data if available.

Step 4: Model Selection

You can use various machine learning models for this task, such as logistic regression, decision trees, random forests, or even deep learning models. In

this example, I'll use a simple logistic regression model for binary classification:

Step 5: Evaluation and Improvement

Evaluate your model using appropriate metrics and make improvements as necessary. You can try different models and hyperparameter tuning to enhance the model's performance.

Keep in mind that earthquake prediction is a complex and challenging problem. This example provides a basic framework, but real earthquake prediction models involve much more advanced and specialized techniques and domain expertise.

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

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

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()
```

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

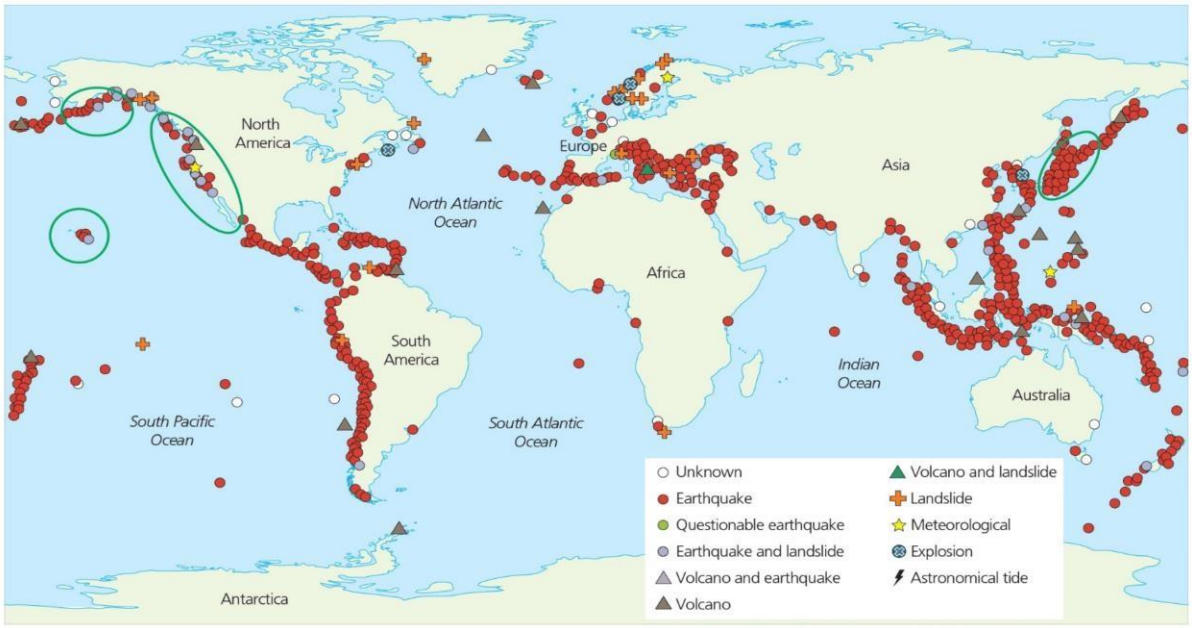
Visualization:

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.

In

[8]: 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]:
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.drawmapboundar
y()
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 ]]
```

DATA PREPROCESSING:

Before we can do any feature engineering, we need to preprocess the data to get it in a form suitable for analysis. The data we used in the course was a bit simpler than the competition data. For the Ames competition dataset, we'll need to:

Load the data from CSV files

Clean the data to fix any errors or inconsistencies

Encode the statistical data type (numeric, categorical)

Impute any missing values

SET OF HYPERPARAMETER:

1.n_estimators = number of trees in the foreset.

2.max_features = max number of features considered for splitting a node.

3.max_depth = max number of levels in each decision tree.

4.min_samples_split = min number of data points placed in a node before the node is split.

GRIDE SEARCH

A grid is a network of intersecting lines that forms a set of squares or rectangles like the image above. In grid search, each square in a grid has a combination of hyperparameters and the model has to train itself on each combination. For a clearer understanding, suppose that we want to train a Random Forest Classifier with the following set of hyperparameters.

DATA SOURCE:

n_estimators			
max_depth	(100, 20)	(150, 20)	(200, 20)
	(100, 30)	(150, 30)	(200, 30)
	(100, 40)	(150, 40)	(200, 40)

Implementation of Grid Search in Python

Scikit-learn library in Python provides us with an easy way to implement grid search in just a few lines of code. Have a look at the example below

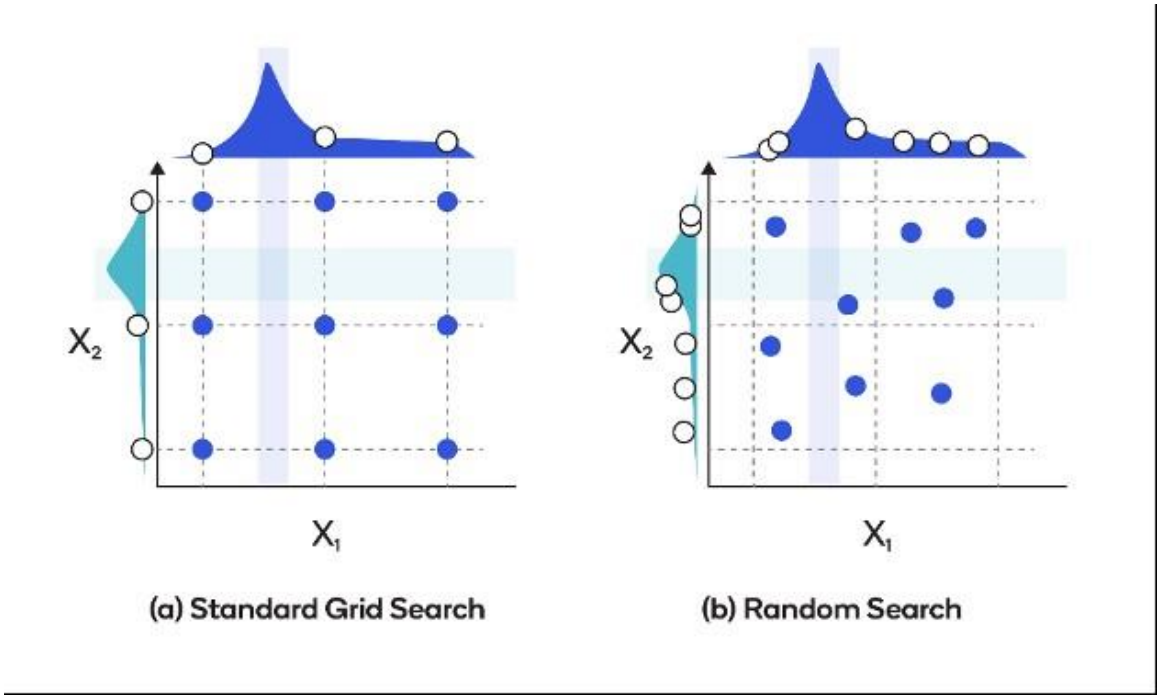
EXAMPLE:

```
from sklearn.model_selection import GridSearchCV model =

LogisticRegression() grid_vals = {'penalty': ['l1','l2'], 'C':

[0.001,0.01,0.1,1]} grid_lr = GridSearchCV(estimator=model,

param_grid=grid_vals, scoring='accuracy',                cv=6,

refit=True, return_train_score=True)


#Training and

Prediction

grid_lr.fit(X_train,

y_train) preds =

grid_lr.best__.predict(

X_test)
```

RANDOM SEARCH:

In grid search, it is required to define the range of values for each hyperparameter that needs to be tuned. In contrast, using a random search the distribution or range of values

is defined for each hyperparameter that needs to be tuned. The random search algorithm then samples combinations of hyperparameter values from these distributions and evaluates the model's performance. By iteratively sampling and evaluating a specified number of combinations, random search aims to identify the hyperparameter settings that yield the best performance.



RANDOM SEARCH IN PYTHON

```
from sklearn.model_selection import RandomizedSearchCV model =
RandomForestClassifier() param_vals = {'max_depth': [200, 500,
800, 1100], 'n_estimators': [100, 200, 300, 400],
        'learning_rate': [0.001, 0.01, 0.1, 1, 10]]} random_rf =
RandomizedSearchCV(estimator=model,
param_distributions=param_vals,                    n_iter=10,
scoring='accuracy', cv=5,                    refit=True, n_jobs=-1)
#Training and
prediction
random_rf.fit(X_train,
```

y_train) preds =

random_rf.best_estimat

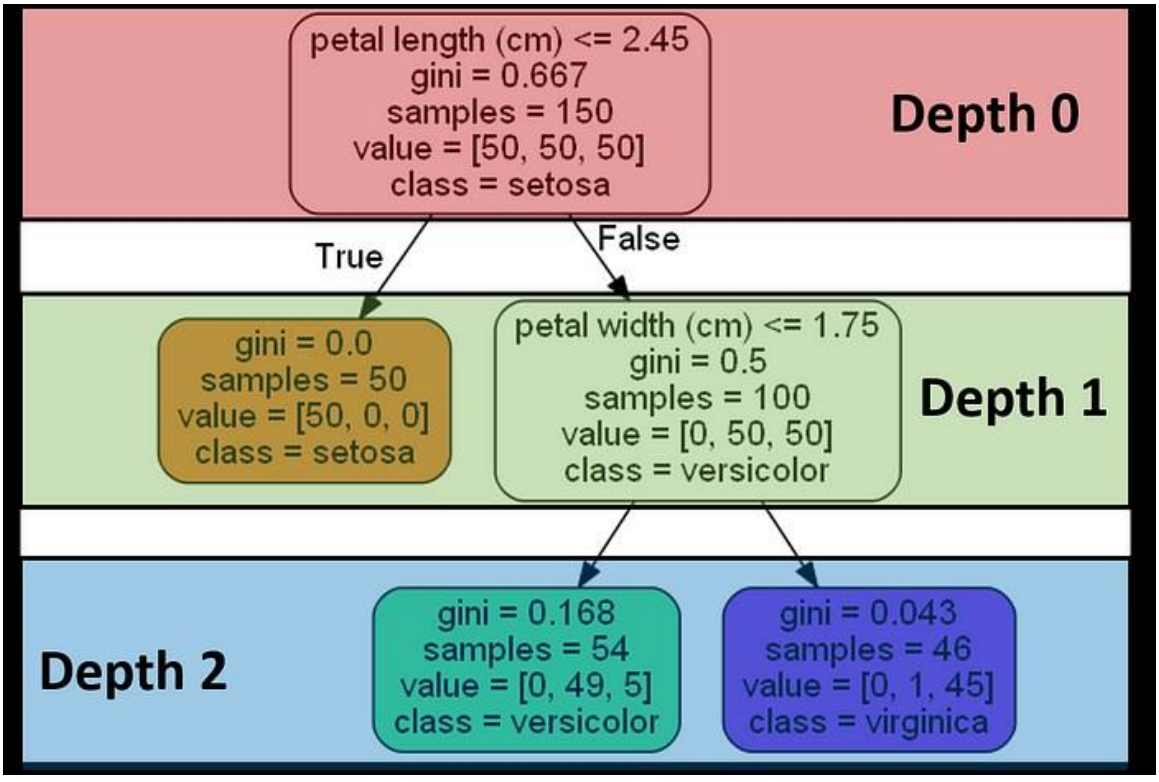
or_.predict(X_test)

Hyperparameter Tuning with Grid Search:

Grid search involves exhaustively searching a predefined set of hyperparameter values to find the combination that yields the best performance. A grid search systematically explores the hyperparameter space by creating a grid or a Cartesian product of all possible hyperparameter values and evaluating the model for each combination.

In grid search, it is required to define the range of values for each hyperparameter that needs to be tuned. The grid search algorithm then trains and evaluates the model using all possible combinations of these values. The performance metric, such as accuracy or mean squared error, is used to determine the best set of hyperparameters.

For example, consider a Random Forest classifier. The hyperparameters that can be tuned using grid search include the number of trees in the forest, the maximum depth of each tree, and the minimum number of samples required to split a node. By defining a grid of values for each hyperparameter, such as [100, 200, 300] for the number of trees and [5, 10, 15] for the maximum depth, grid search explores all possible combinations. The model is trained and evaluated for each combination, and the best-performing set of hyperparameters is selected.



Earthquake Prediction is a way of predicting the magnitude of an earthquake based on parameters such as longitude, latitude, depth, and duration magnitude, country, and depth using machine learning to give warnings of potentially damaging earthquakes early enough to allow appropriate response to the disaster, enabling people to minimize loss of life and property.

What is 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.

STEPS OF PREPROCESSING:

- Date parsing: Parsing date to dtype datetime64(ns).
- Time Parsing: Parsing time to dtype timedelta64.
- Adding Attributes: ” Date_Time ” and ” Days “.

DATA PREPROCESSING:

```
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')

time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms	...	updated
2023-02-14T21:31:52.124Z	60.828300	-151.841200	85.00	2.20	ml	NaN	NaN	NaN	1.6100	...	2023-02-14T21:35:21.982Z
2023-02-14T20:45:56.420Z	19.254333	-155.410828	31.32	2.27	ml	41.0	139.00	NaN	0.1500	...	2023-02-14T20:51:26.040Z
2023-02-14T20:45:12.919Z	38.146900	-117.982000	7.30	1.90	ml	11.0	110.46	0.02000	0.1385	...	2023-02-14T21:04:41.699Z
2023-02-14T20:43:53.796Z	63.898700	-148.655300	82.40	1.30	ml	NaN	NaN	NaN	0.5700	...	2023-02-14T20:46:28.820Z
2023-02-14T20:43:40.220Z	33.324167	-116.757167	12.42	0.89	ml	23.0	67.00	0.08796	0.1700	...	2023-02-14T21:22:42.029Z

updated	place	type	horizontalError	depthError	magError	magNst	status	locationSource	magSource
2023-02-14T21:35:21.982Z	33 km WNW of Nikiski, Alaska	earthquake	NaN	2.10	NaN	NaN	automatic	ak	ak
2023-02-14T20:51:26.040Z	9 km NE of Pāhala, Hawaii	earthquake	0.66	0.81	2.790	10.0	automatic	hv	hv
2023-02-14T21:04:41.699Z	Nevada	earthquake	NaN	1.30	0.210	9.0	reviewed	nn	nn
2023-02-14T20:46:28.820Z	15 km ENE of Healy, Alaska	earthquake	NaN	1.50	NaN	NaN	automatic	ak	ak
2023-02-14T21:22:42.029Z	9km N of Lake Henshaw, CA	earthquake	0.26	1.00	0.133	8.0	reviewed	ci	ci

DATA RANGE:

Check for data range: Check if the values fall within a reasonable range for the column they are in. For example, latitude values should be between -90 and 90, and longitude values should be between -180 and 180. In [16]:

```
# Check the range of values in the
latitude column
print(df['latitude'].describe()) count
10153.000000 mean
41.801063 std 18.817115
min -
64.428900
```

25%
34.710300
50%
38.832668 75%
58.244667 max
84.884100
Name: latitude,
dtype: float64 In
[17]:

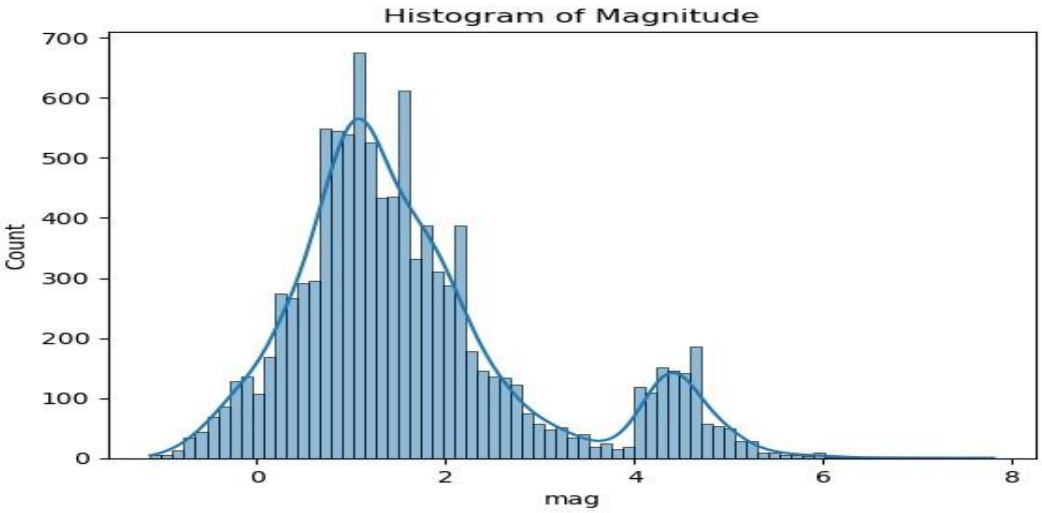
```
# Check the range of values in
the longitude column
print(df['longitude'].describe())
```

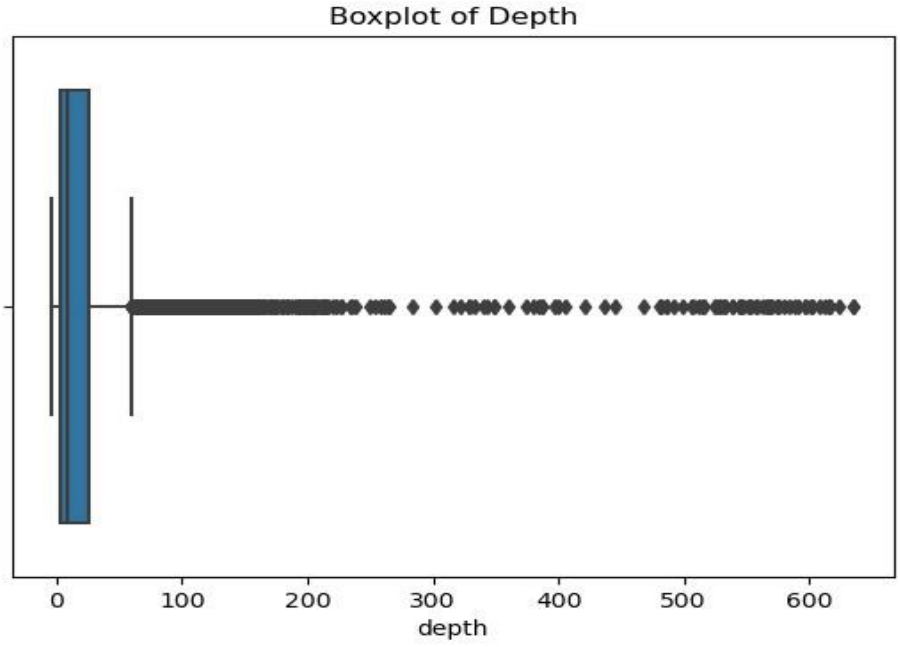
count 10153.000000 mean
-114.365448 std
68.809055 min -179.994000
25% -152.329700
50%
-
122.81
4835
75%
-
116.72
0167
max
179.98
4000
Name: longitude, dtype: float64

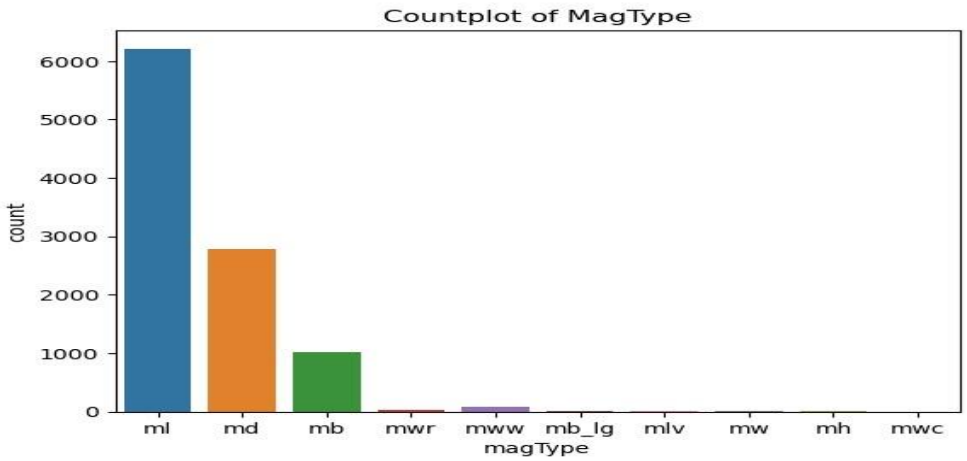
```
import seaborn as sns
import matplotlib.pyplot as plt

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

# Boxplot of depth
sns.boxplot(data=df, x='depth')
plt.title('Boxplot of Depth')
plt.show()

# Countplot of magType
sns.countplot(data=df, x='magType')
plt.title('Countplot of MagType')
plt.show()
```

The latitude and longitude columns have reasonable values with no apparent incorrect data. The magType column seems to have only 10 unique values, which seem reasonable for the type of data that is being analyzed.

UNIVARIATE ANALYSIS:

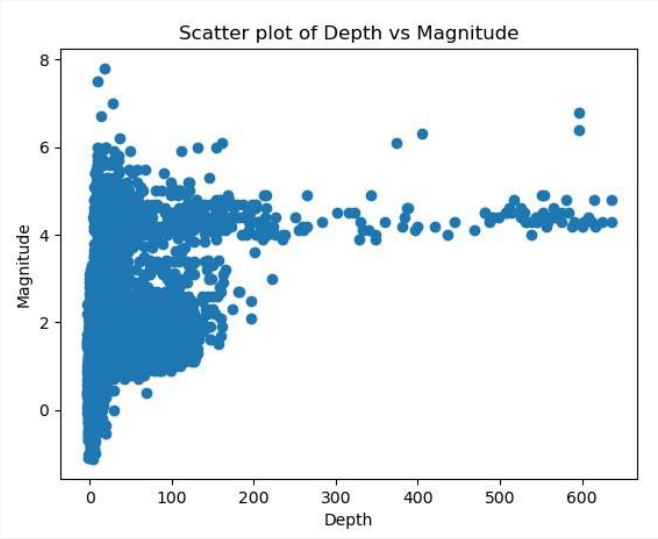






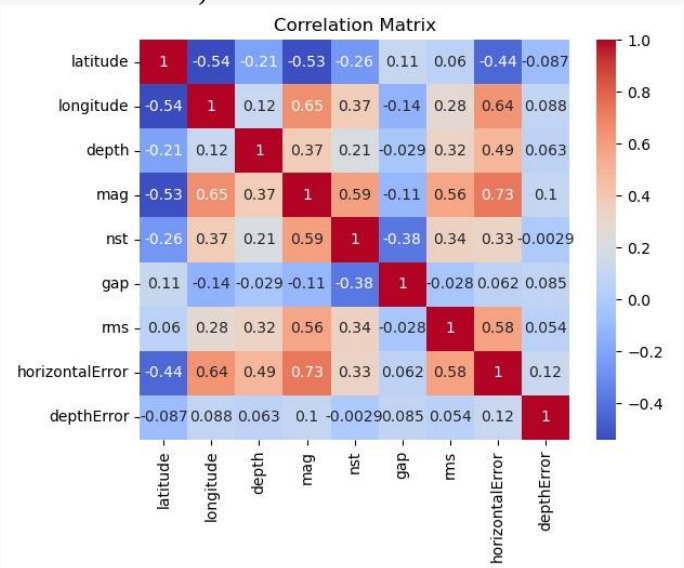

```
import matplotlib.pyplot as plt

# Scatter plot of depth vs magnitude
plt.scatter(df['depth'], df['mag'])
plt.xlabel('Depth')
plt.ylabel('Magnitude')
plt.title('Scatter plot of Depth vs Magnitude')
plt.show()

# Box plot of earthquake magnitude by type
df.boxplot(column='mag', by='type')
plt.title('Box plot of Earthquake Magnitude by Type')
plt.suptitle('')
```

BIVARIATE ANALYSIS:



MULTIVARIATE ANALYSIS:

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

# Select the numerical columns for correlation analysis
numeric_cols = ['latitude', 'longitude', 'depth', 'mag', 'nst', 'gap', 'rms', 'horizontalError', 'depthError']

# Create correlation matrix
corr_matrix = df[numeric_cols].corr()

# Plot heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
```

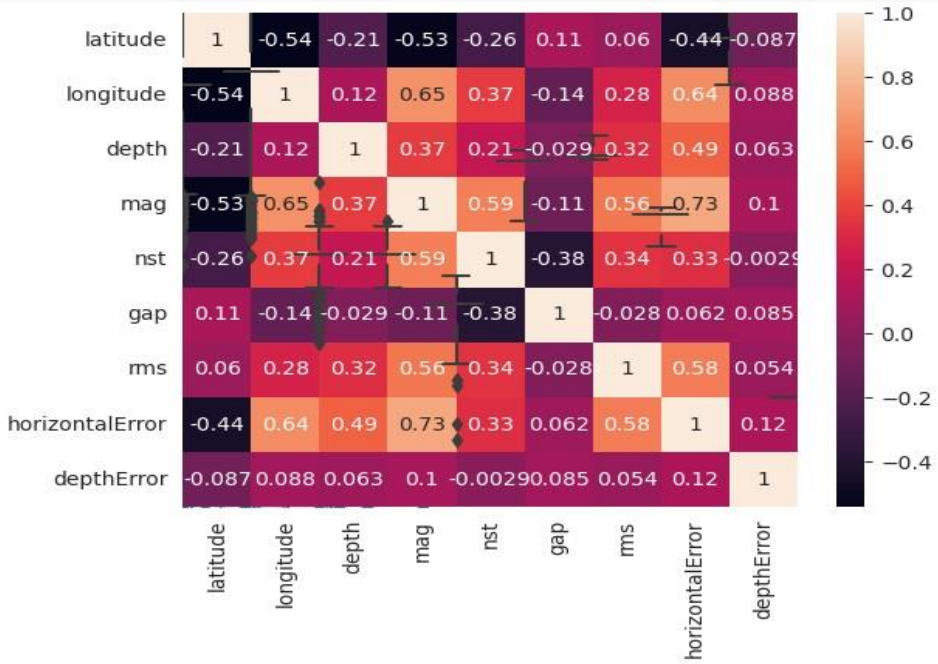



```
plt.show()
```

DATA VISUALIZATION:

```
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")
```

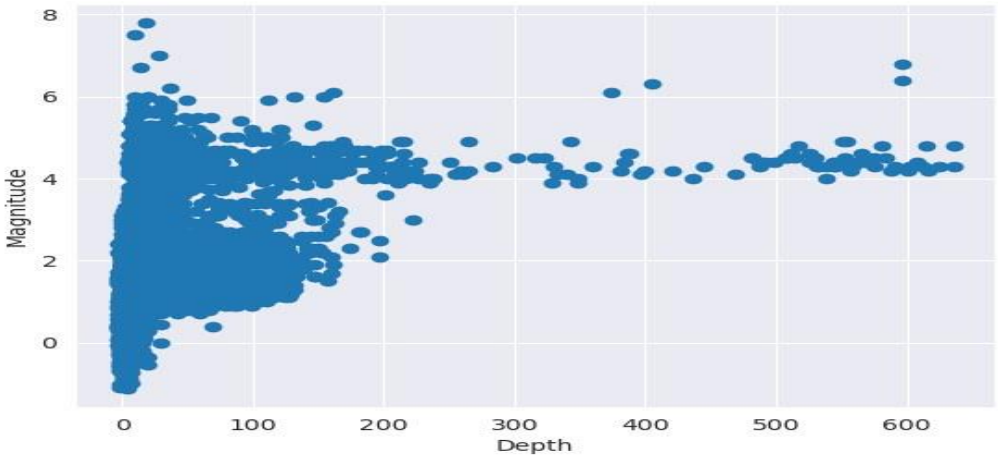
```
# Heatmap to show correlation between
variables corr = df.corr()
sns.heatmap(corr, annot=True)

# Pairplot to show scatterplots of all possible variable combinations
sns.pairplot(df)

# Show all visualizations
plt.show()
```



EARTHQUAKE PREDICTION ANALYSIS:

The correlation between the magnitude and other factors such as latitude, longitude, and time. We can perform a correlation analysis to see how these factors are related to the magnitude of earthquakes. This will help us understand which factors are most important in determining the magnitude of an earthquake.

```
import matplotlib.pyplot as plt

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




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

To building the project by performing different activities like feature engineering, model training, evaluation.

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 Machines

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_d
'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

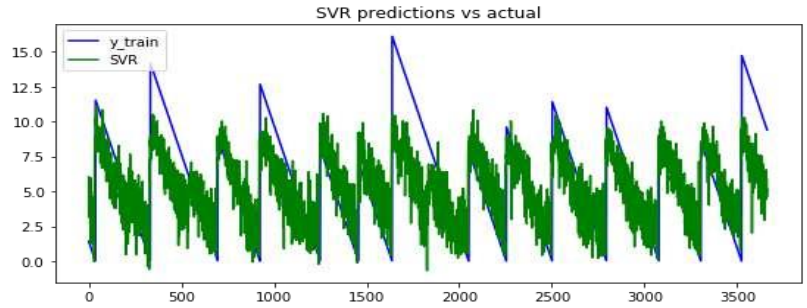
Learning Model Implementation:

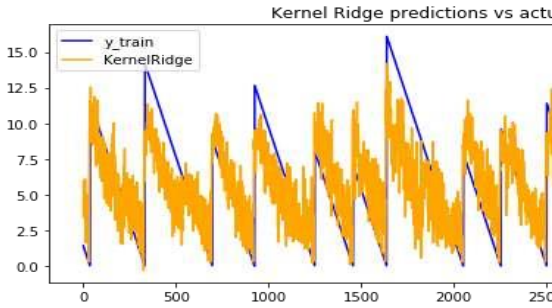
1. SUPPORT VECTOR MACHINES
2. KERNEL RIDGE

```
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_)
```

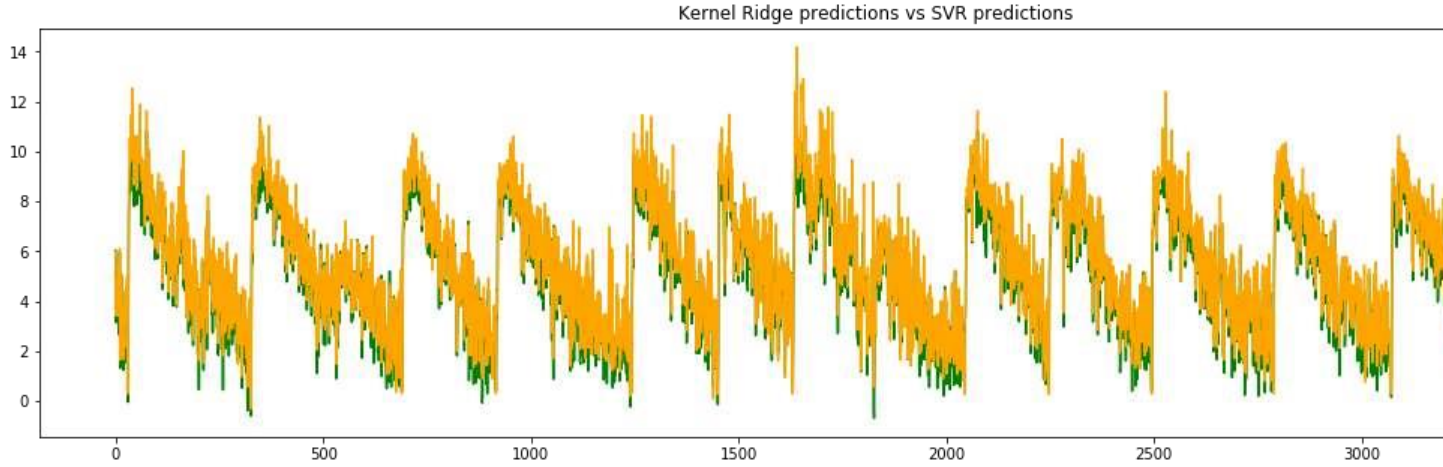
Graphical Analysis of Model Performance:

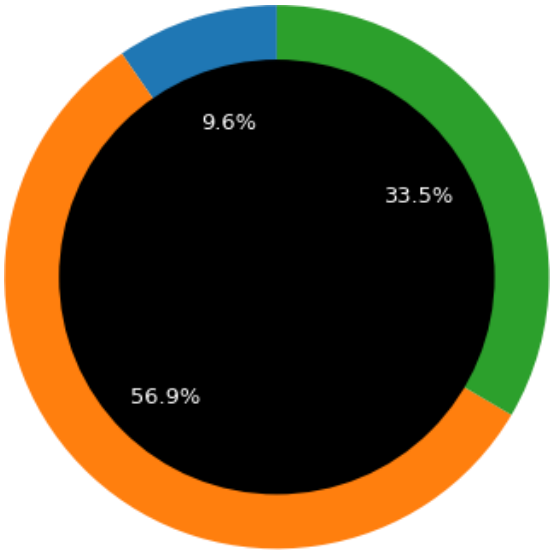
```
Graphical Analysis of Model Performance
```





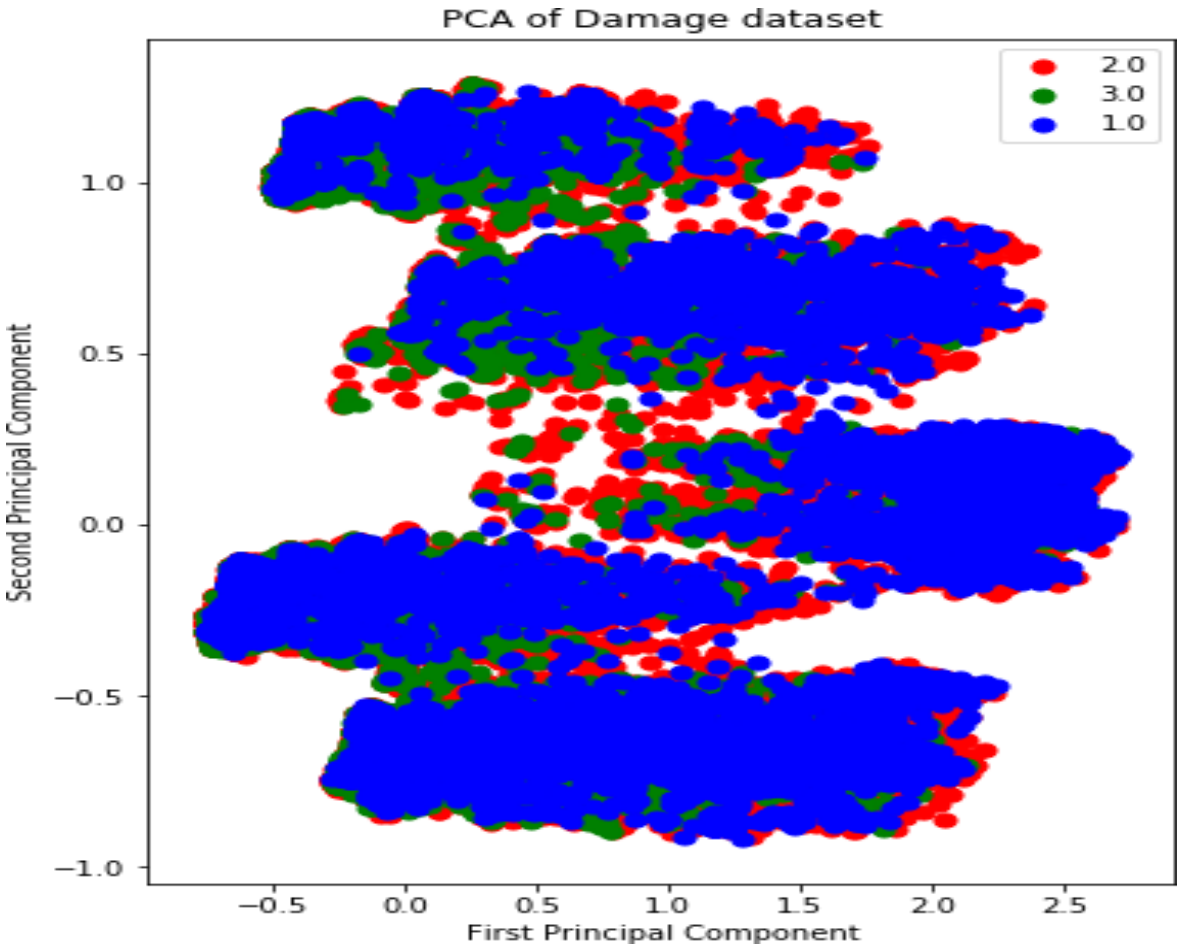
```
plt.figure(figsize=(20, 5)) plt.plot(y_pred1,
color='green', label='SVR') plt.plot(y_pred2,
color='orange',
label='KernelRidge')
plt.legend() plt.title('Kernel
Ridge predictions vs SVR
predictions') plt.show()
```





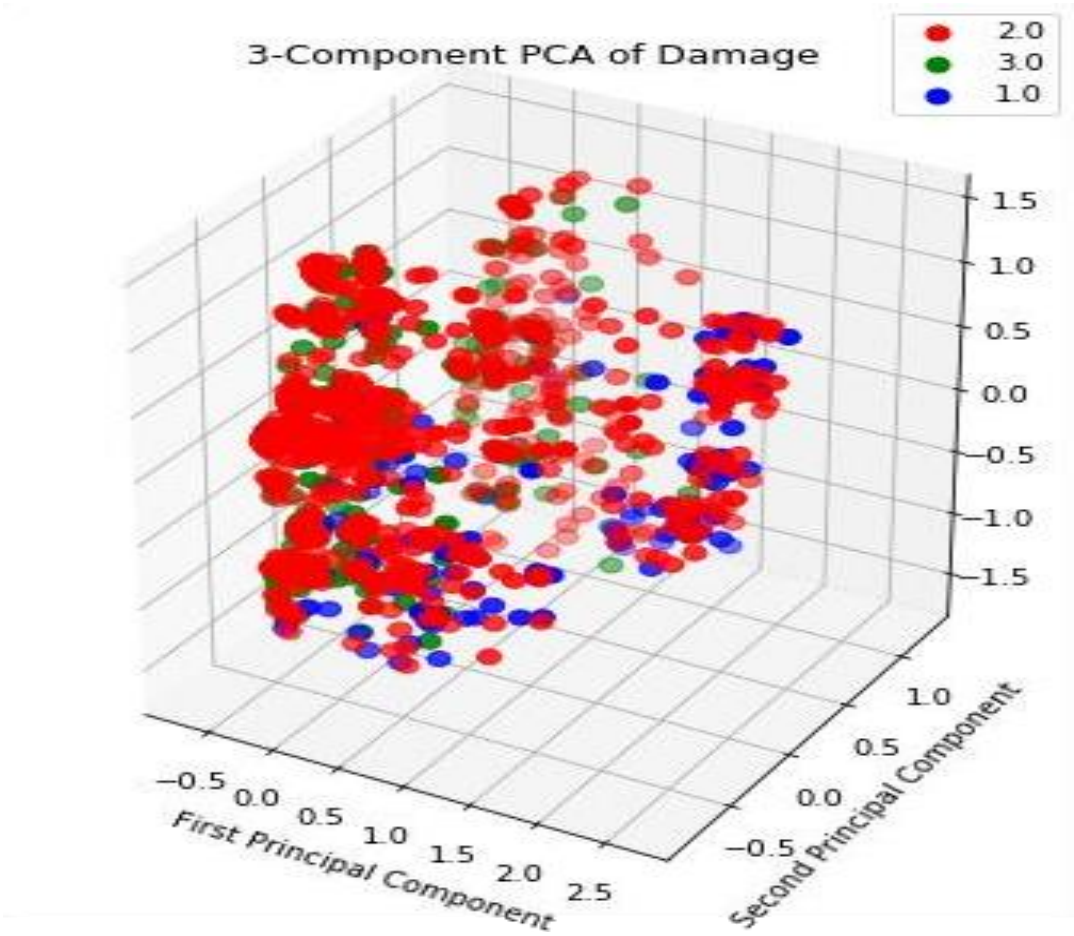

```
normalized_df=(df-df.min())/(df.max()-df.min()) df =
normalized_df


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')
```




```
X = df.drop("damage_grade", axis=1) y =
df["damage_grade"]
targets = df["damage_grade"].unique()
print(targets)
pca = PCA(n_components=3) X_r = pca.fit(X).transform(X) print(X_r.shape)


PCA_Df = pd.DataFrame(data = X_r[0:2000]
             , columns = ['principal component 1', 'principal component 2', 'principal component 3'])
colors = ['r', 'g', 'b'] fig =
plt.figure()
ax = fig.add_subplot(111, projection='3d') for
target, color in zip(targets,colors):
    indicesToKeep = df['damage_grade'] == target ax.scatter(PCA_Df.loc[indicesToKeep,
    'principal component 1']
              , PCA_Df.loc[indicesToKeep, 'principal component 2'], PCA_Df.loc[indicesToKeep,
'principal component 3'], c = color, s = 50) plt.legend((targets + .5) * 2)
plt.title('3-Component PCA of Damage ') plt.xlabel("First
Principal Component") plt.ylabel('Second Principal Component')


[0.5 1.  0. ]
```

(260601, 3)



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

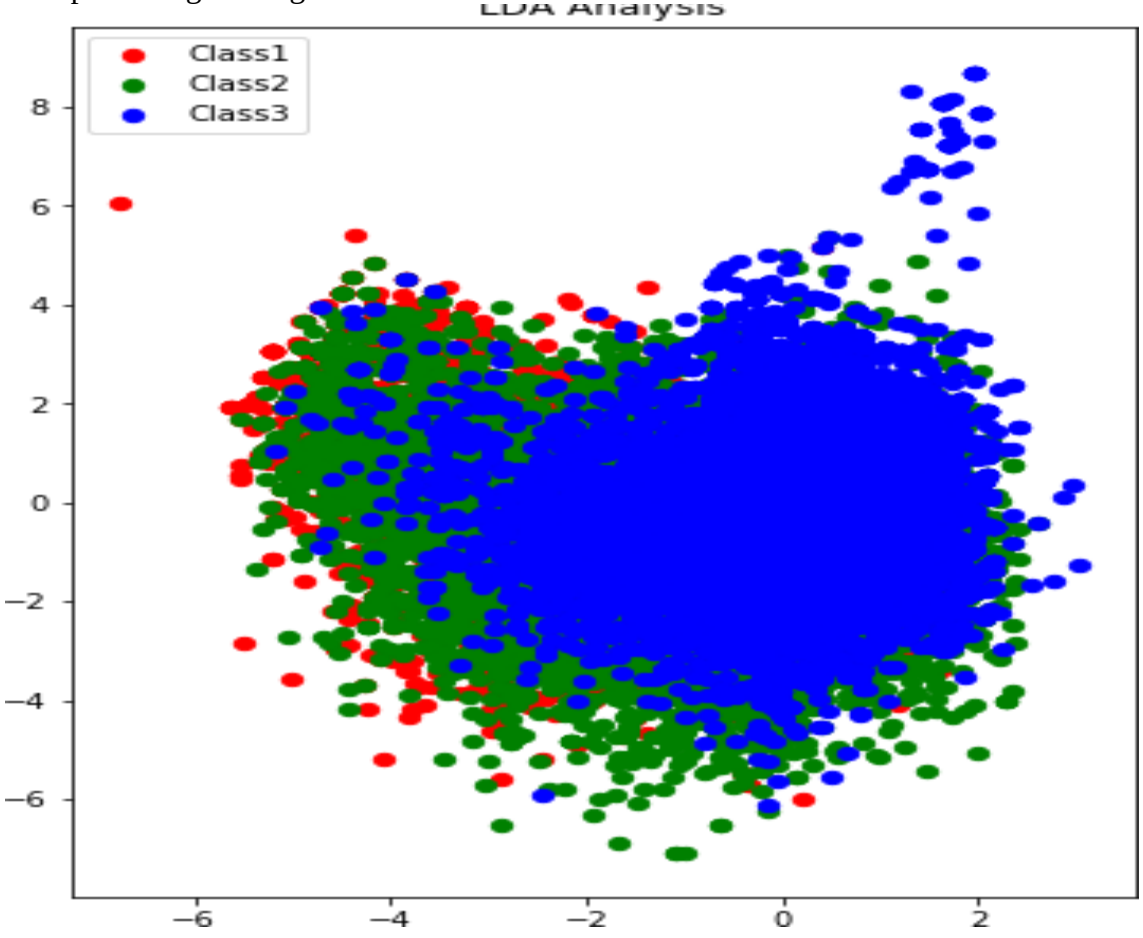
```
X = df.drop("damage_grade", axis=1).astype('int') y =
((df["damage_grade"] + 0.5) * 2).astype('int') lda =
LDA(n_components=2)
dmg_lda = lda.fit_transform(X, y)
print(dmg_lda)
l_x = dmg_lda[:,0] l_y =
dmg_lda[:,1]
cdict={1:'red',2:'green',3:'blue'}
labl={1:'Class1',2:'Class2',3:'Class3'} for l in
np.unique(y):
    ix=np.where(y==l)
    ax = plt.scatter(l_x[ix],l_y[ix],c=cdict[l],s=40, label=labl[l])
plt.title("LDA Analysis")
plt.legend()
```

```
/home/jordanrodrigues/anaconda3/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:388: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
```

```
[[-0.39785383 -1.70783617]
```

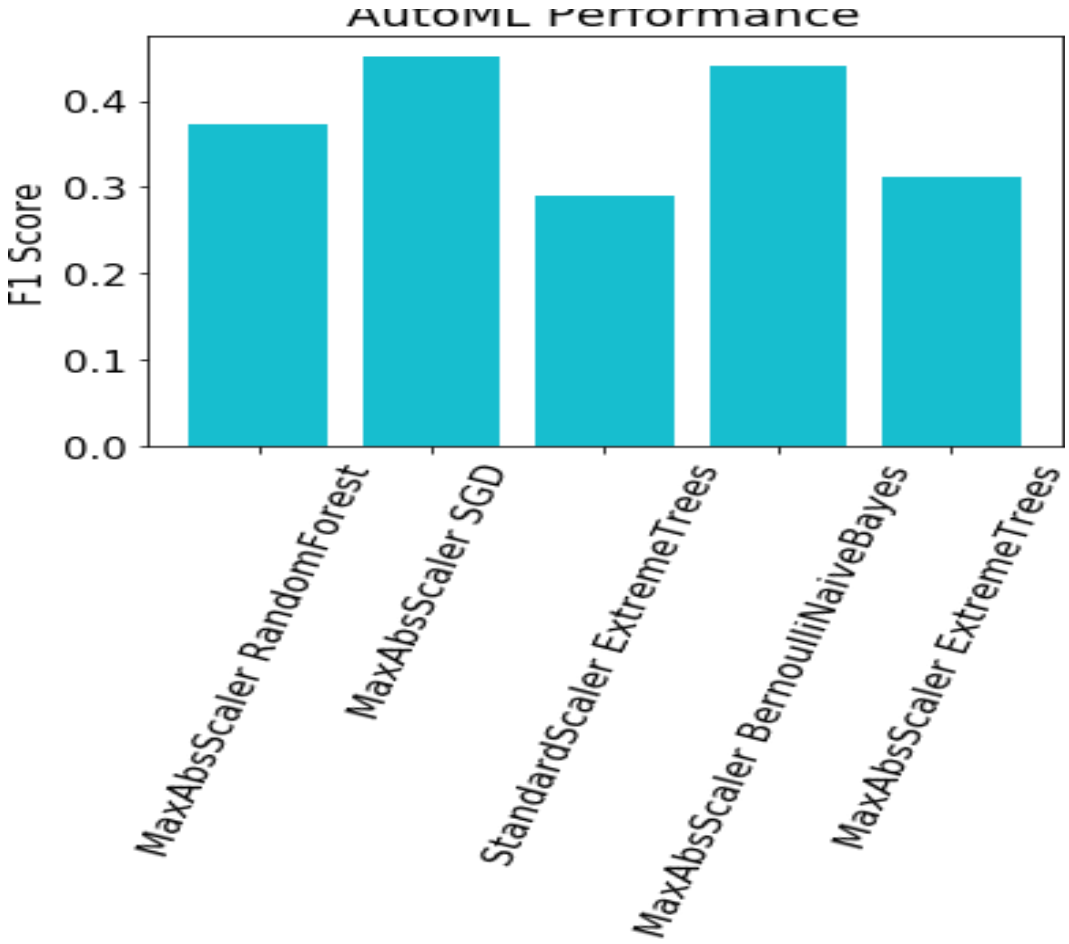
```
 [ 0.42992898 -0.02751219]
 [ 0.87874376  0.49585697]
 ...
 [ 0.61089644  0.2825365 ]
 [ 0.75307395  0.58296292]
 [ 1.70574956  7.67655413]]
```

<matplotlib.legend.Legend at 0x7f6fe9e9d908>




```
labels = ["MaxAbsScaler RandomForest", "MaxAbsScaler SGD", "StandardScaler
ExtremeTrees", "MaxAbsScaler BernoulliNaiveBayes", "MaxAbsScaler ExtremeTrees"]


values = [.3720, .4521, .2893, .4397,  .3116]


plt.bar(labels, values, color='tab:cyan')
plt.xticks(rotation=70)
plt.rcParams.update({'font.size': 15})
plt.title("AutoML Performance") plt.ylabel("F1
Score")
```

Text(0, 0.5, 'F1 Score')



CLEANING DATA:

DIAGNOSE DATA FOR CLEANING:

Missing Values:

Diagnosis: Check for columns with a significant number of missing values. Missing values can distort predictions and analysis.

Cleaning: Remove rows with missing values, fill missing values with mean or median, or use advanced imputation techniques based on the nature of the dataset.

Outliers:

Diagnosis: Identify values that deviate significantly from the rest of the data. Outliers can skew predictions and affect model performance.

Cleaning: Remove outliers based on statistical methods like IQR (Interquartile Range) or use domain knowledge to determine if they are valid data points.

Inconsistent Data Formats:

Diagnosis: Check for inconsistent formats in date, time, or geographical data. Consistent formats are essential for analysis and visualization.

Cleaning: Standardize formats across the dataset. For example, ensure all dates are in the same format and time zones are consistent.

Duplicate Data:

Diagnosis: Look for duplicate records that might have been entered into the dataset more than once.

Cleaning: Remove duplicate entries to maintain the integrity of the dataset.

Incorrect or Inaccurate Data.

FEATURE ENGINEERING:

Diagnosis: Explore the existing features and assess if new features can be derived to enhance the model's predictive power.

Cleaning: Create new features based on domain knowledge. For example, derive features like distance from fault lines, historical seismic activity, or geological features.

Imbalanced Data (if applicable):

Diagnosis: In classification tasks, check if the classes (e.g., earthquake occurrence vs. non-occurence) are imbalanced.

Cleaning: Balance the classes through techniques like oversampling, undersampling, or using algorithms that handle imbalanced data effectively.

Data Integrity Checks:

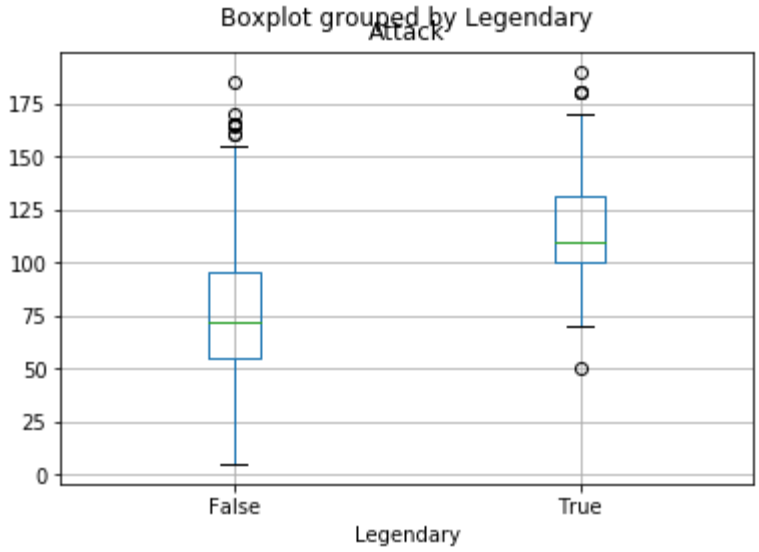
Diagnosis: Validate relationships between different columns. For instance, cross-verify location data with geographical databases.

Cleaning: Correct inconsistencies and ensure data integrity by validating relationships and dependencies within the dataset.

By diagnosing and addressing these issues, you can prepare a clean and reliable dataset for building an accurate earthquake prediction model. Remember that the specific cleaning steps may vary based on the characteristics of the dataset and the requirements of the prediction model.

```
import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt

import seaborn as sns  # visualization tool

from subprocess import check_output

#print(check_output(["ls", "../input"]).decode("utf8"))

data = pd.read_csv('../input/pokemon-challenge/pokemon.csv')

#data.info()
```

```
data.head()
```



TIDY DATA:

Tidy data is a concept introduced by statistician and data scientist Hadley Wickham. It pr ovides a standard way to organize and structure datasets to facilitate easier analysis and vi sualization. In tidy data:

1.Each variable forms a column

2. Each observation forms a row

3. Each type of observational unit forms a table

```
data_new = data.head()    # I only take 5 rows into new data
data_new
# lets melt
# id_vars = what we do not wish to melt
# value_vars = what we want to melt
melted = pd.melt(frame=data_new,id_vars = 'Name', value_vars= ['Attack','Defense'])
melted
```

PIVOTING DATA:

Pivoting data is a technique used to reorganize and reshape data in a table, usually to mak e it more suitable for analysis or visualization. This operation is common when dealing w ith spreadsheet software or data manipulation libraries like Pandas in Python. Pivoting all ows you to transform data from a long format (where different types of information are st ored in different rows) into a wide format (where information is organized into columns).

```
# Index is name
# I want to make that columns are variable
```

```
# Finally values in columns are value
melted.pivot(index = 'Name', columns = 'variable',values='value')
```

CONCATENATING DATA:

```
# Firstly lets create 2 data frame
data1 = data.head()
data2= data.tail()
conc_data_row = pd.concat([data1,data2],axis =0,ignore_index =True) # axis = 0 : adds dataframes in row
conc_data_row

data1 = data['Attack'].head()
data2= data['Defense'].head()
conc_data_col = pd.concat([data1,data2],axis =1) # axis = 0 : adds dataframes in row
conc_data_col
```

SEABORN:

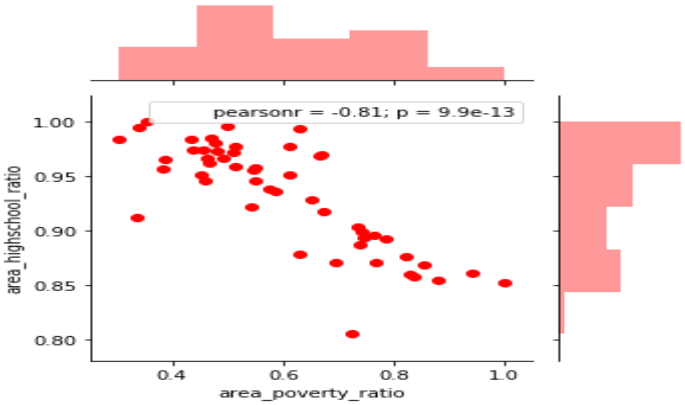
BAR PLOT:

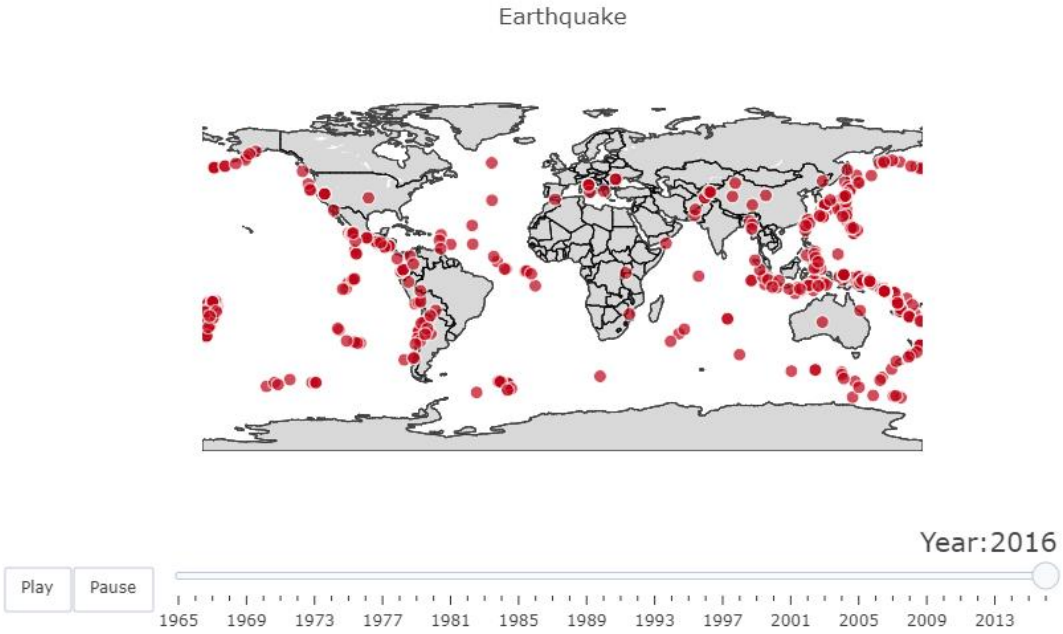
```
import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import seaborn as sns

import matplotlib.pyplot as plt

from collections import Counter

%matplotlib inline


percentage_people_below_poverty_level = pd.read_csv('../input/fatal-police-shootings-in-the-us/PercentagePeopleBelowPovertyLevel.csv', encoding="windows-1252")

kill = pd.read_csv('../input/fatal-police-shootings-in-the-us/PoliceKillingsUS.csv', encoding="windows-1252")

percent_over_25_completed_highSchool = pd.read_csv('../input/fatal-police-shootings-in-the-us/PercentOver25CompletedHighSchool.csv', encoding="windows-1252")

linkcode
```

percentage_people_below_poverty_level.head()

OUTPUT:

Text(0.5,1,'Poverty Rate Given States')





VISUALIZATION TOOLS:

Parallel Plots (Pandas):

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib_venn as venn
from math import pi
from pandas.tools.plotting import parallel_coordinates
import plotly.graph_objs as go
import plotly.plotly as py
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import warnings
warnings.filterwarnings("ignore")

linkcode
data = pd.read_csv('../input/iris/Iris.csv')
data = data.drop(['Id'],axis=1)
# Make the plot
plt.figure(figsize=(15,10))
parallel_coordinates(data, 'Species', colormap=plt.get_cmap("Set1"))
plt.title("Iris data class visualization according to features (setosa, versicolor, virginica)")
plt.xlabel("Features of data set")
plt.ylabel("cm")
plt.savefig('graph.png')
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:

1. Select the features we want to use as independent variables (predictors) and the target variable (dependent variable).
2. Split the data into training and testing sets.
3. Fit the model on the training set.
4. 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-squared score:
0.5632730346912534
```

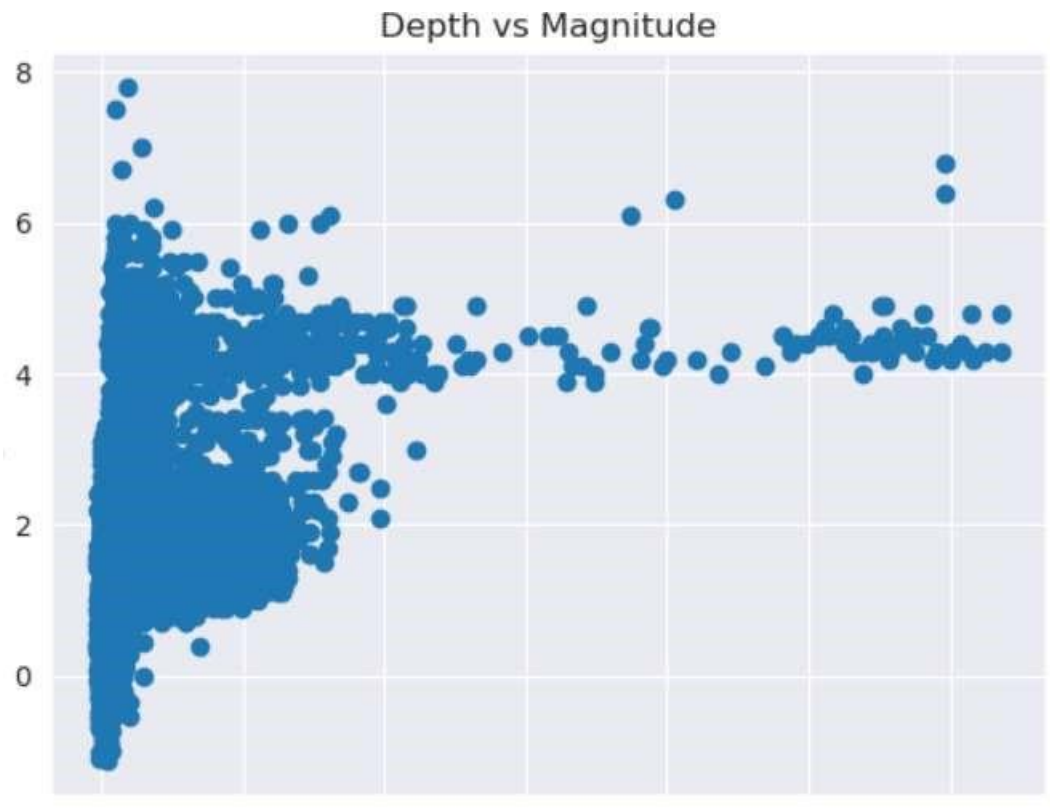
the mean squared error and the R-squared score will help us evaluate the performance of our regression model. The mean squared error represents the average squared difference between the predicted values and the actual values. A lower value indicates a better fit of the model. The R-squared score measures 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
```

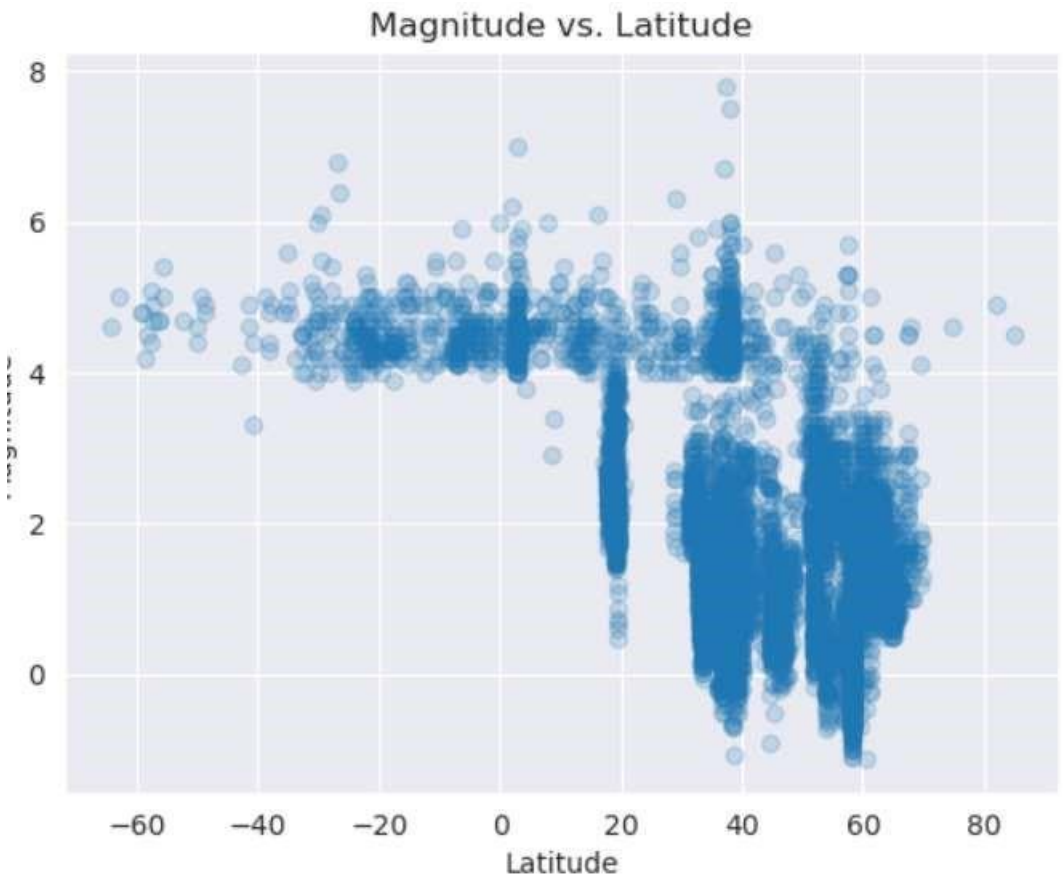
```
plt.scatter(df['depth'], df['mag']) plt.xlabel('Depth')
plt.ylabel('Magnitude') plt.title('Depth vs Magnitude')
plt.show()
```

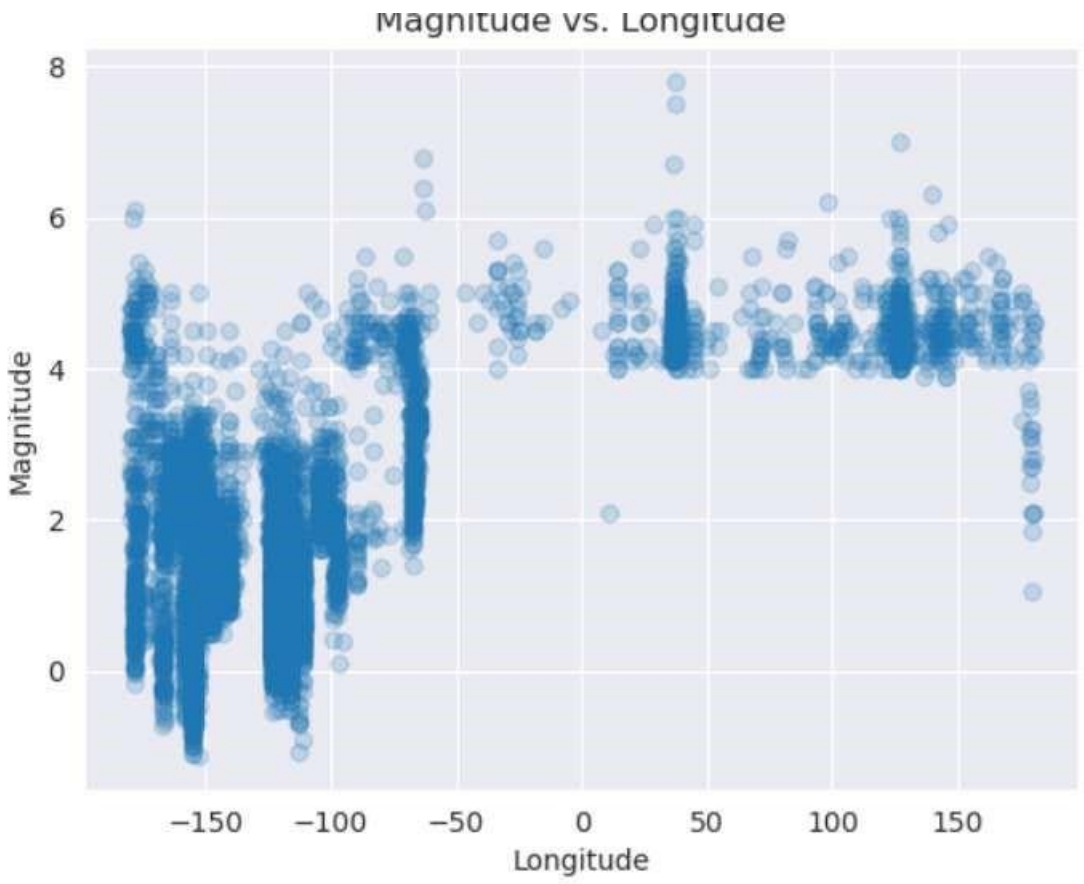


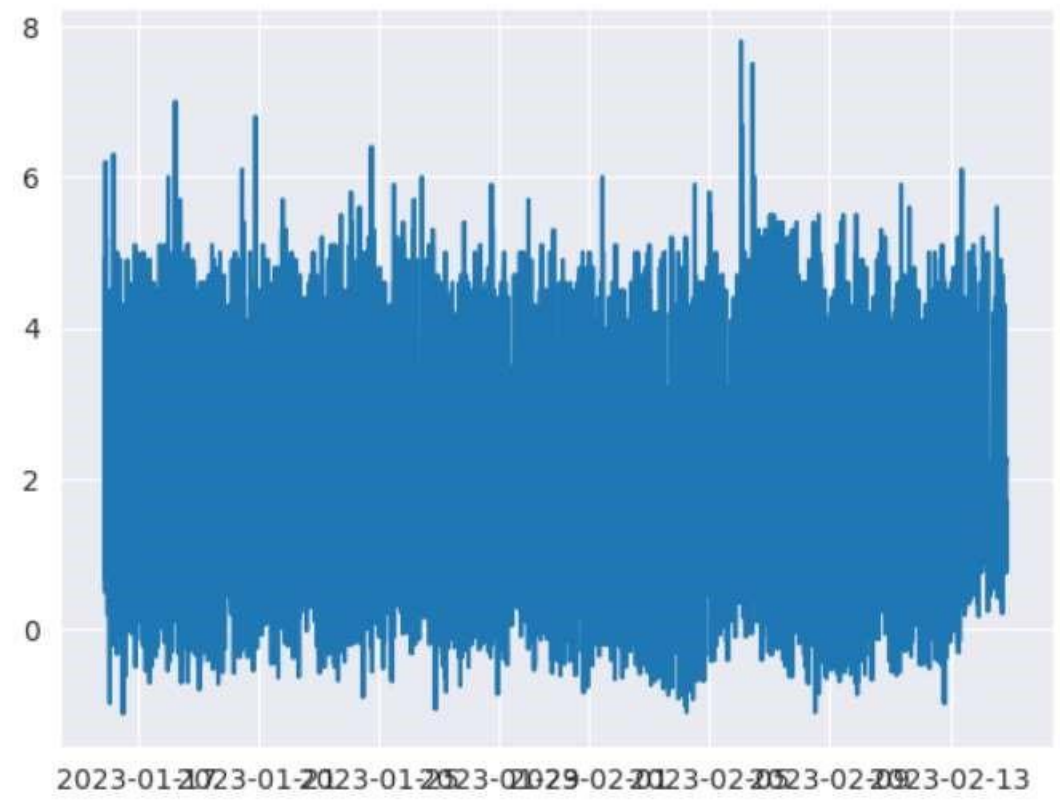
the scatterplot helps visualize the relationship between depth and magnitude. It appears that as the depth decreases, the occurrence of earthquakes with higher magnitudes increases, and as the depth increases, the occurrence of earthquakes with lower magnitudes increases.

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.

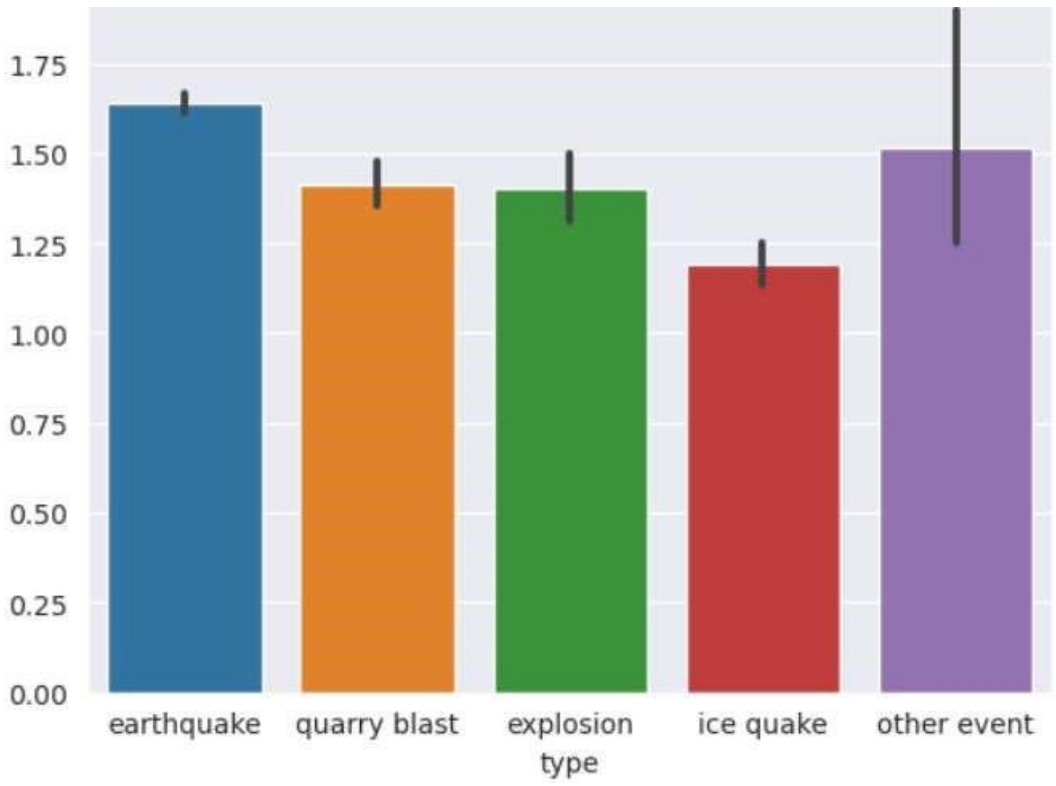
```
import matplotlib.pyplot as plt import seaborn as sns

# 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()
```



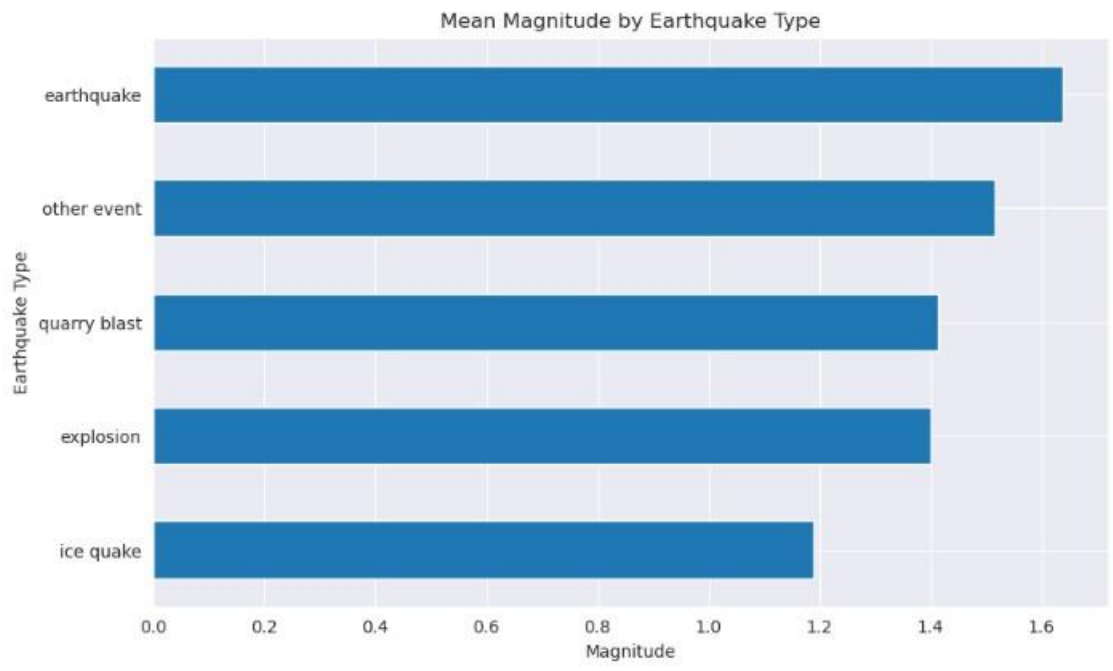




Bar chart of magnitude and type sns.barplot(data=df, x='type', y='mag') plt.show()




```
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 Io T 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.