4222-SURYA GROUP OF INSTITUTIONS

VIKARAVANDI -605 652

NAAN MUDHALAVAN PROJECT

SUBJECT CODE: SB3001

COURSE NAME: EXPERIENCE BASED PRACTICAL LEARNING

EARTHQUAKE PREDICTION MODEL USING PYTHON

PREPARED BY:

M.GAYATHRI

REG NO:422221106008

ECE DEPARTMENT

EARTHQUAKE PREDICTION MODEL USING PYTHON

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 SOURCE:

```
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()
```

Out[4]:

Sino	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

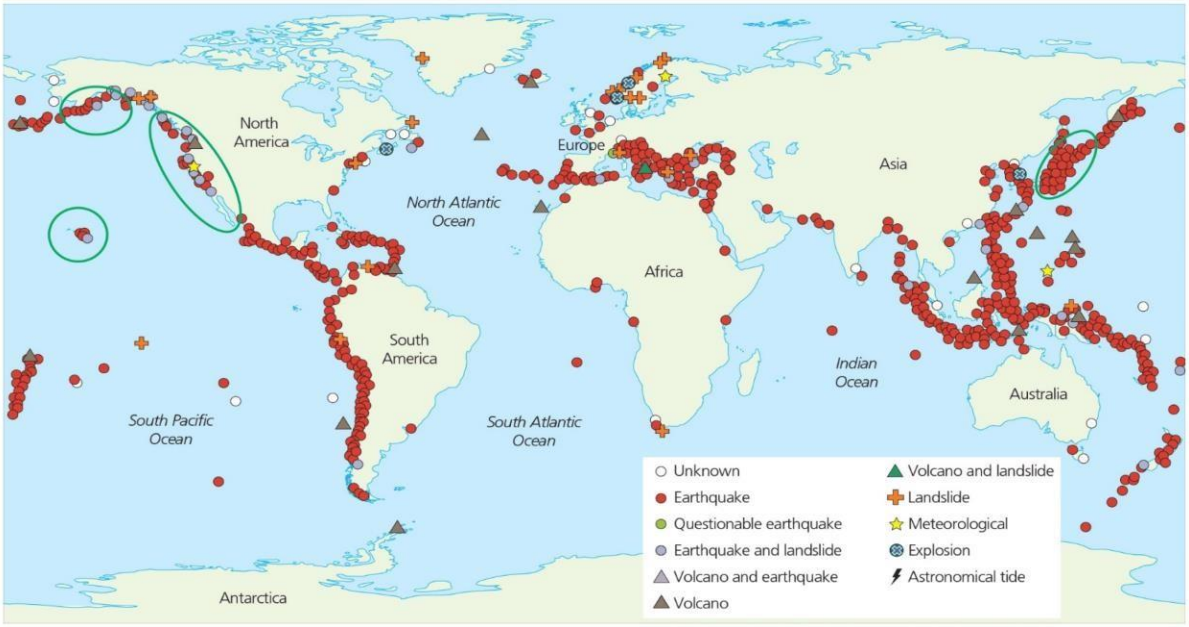
VISUALIZATION:

Here, all the earthquakes from the database in visualized on to the world map which shows clear representationof 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.drawmapboundary()
m.drawcountrie
s()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 repeatablefrom

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

```
from keras.wrappers.scikit_learn import KerasClassifier

model = KerasClassifier(build_fn=create_model, verbose=0)

# param_grid = {
#     "neurons": [16, 64],
#     "batch_size": [10, 20],
#     "epochs": [10],
#     "activation": ['sigmoid', 'relu'],
#     "optimizer": ['SGD', 'Adadelta'],
#     "loss": ['squared_hinge']
# }
param_grid = {
    "neurons": [16],
    "batch_size": [10, 20],
    "epochs": [10],
    "activation": ['sigmoid', 'relu'],
    "optimizer": ['SGD', 'Adadelta'],
    "loss": ['squared_hinge']
}


X_train = np.asarray(X_train).astype(np.float32)
y_train = np.asarray(y_train).astype(np.float32)
X_test = np.asarray(X_test).astype(np.float32)
y_test = np.asarray(y_test).astype(np.float32)


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

best_params = grid_result.best_params_
best_params
```

```
2023-02-12 14:30:16.688729: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 4. Tune using inter_op_parallelism_threads for best performance.
```

2023-02-12 14:30:16.721324: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 4. Tune using inter_op_parallelism_threads for best performance.
2023-02-12 14:30:16.733601: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 4. Tune using inter_op_parallelism_threads for best performance.
2023-02-12 14:30:16.761165: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 4. Tune using inter_op_parallelism_threads for best performance.
2023-02-12 14:30:17.151828: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2023-02-12 14:30:17.151828: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2023-02-12 14:30:17.151827: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2023-02-12 14:30:17.164576: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2023-02-12 14:34:25.381389: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
2023-02-12 14:34:25.461923: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)

{'activation': 'sigmoid',
 'batch_size': 20,
 'epochs': 10,
 'loss': 'squared_hinge',
 'neurons': 16,
 'optimizer': 'SGD'}

FEATURE ENGINEERING

datas

	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source	Magnitude Source	Status
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic
...	...	...	...	...	...	...	...	...	...	...	...	...	...
23404	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN	NN	Reviewed
23405	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN	NN	Reviewed
23406	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US	US	Reviewed
23407	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US	US	Reviewed
23408	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US	US	Reviewed

```
data['Month'] = data['Date'].apply(lambda x: x[0:2])
```

```
data['Year'] = data['Date'].apply(lambda x: x[-4:])

data = data.drop('Date', axis=1)


data['Month'] = data['Month'].astype(np.int)


data[data['Year'].str.contains('Z')]
```

	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source	Magnitude Source	Status	Month	Year
3378	1975-02-23T02:58:41.000Z	8.017	124.075	Earthquake	623.0	5.6	MB	1.022784	US	US	US	Reviewed	19	000Z
7510	1985-04-28T02:53:41.530Z	-32.998	-71.766	Earthquake	33.0	5.6	MW	1.300000	US	US	HRV	Reviewed	19	530Z
20647	2011-03-13T02:23:34.520Z	36.344	142.344	Earthquake	10.1	5.8	MWC	1.060000	US	US	GCMT	Reviewed	20	520Z

```
invalid_year_indices = data[data['Year'].str.contains('Z')].index

data = data.drop(invalid_year_indices, axis=0).reset_index(drop=True)


data['Year'] = data['Year'].astype(np.int)


data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))

data = data.drop('Time', axis=1)
```

	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source	Magnitude Source	Status	Month	Year	Hour
0	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	13
1	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	11
2	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	18
3	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	18
4	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23401	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN	NN	Reviewed	12	2016	8
23402	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN	NN	Reviewed	12	2016	9
23403	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US	US	Reviewed	12	2016	12
23404	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US	US	Reviewed	12	2016	22
23405	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US	US	Reviewed	12	2016	20

```
data['Status'].unique()
```

```
array(['Automatic', 'Reviewed'], dtype=object)
```

```
data['Status'] = data['Status'].apply(lambda x: 1 if x == 'Reviewed' else 0)
```

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()

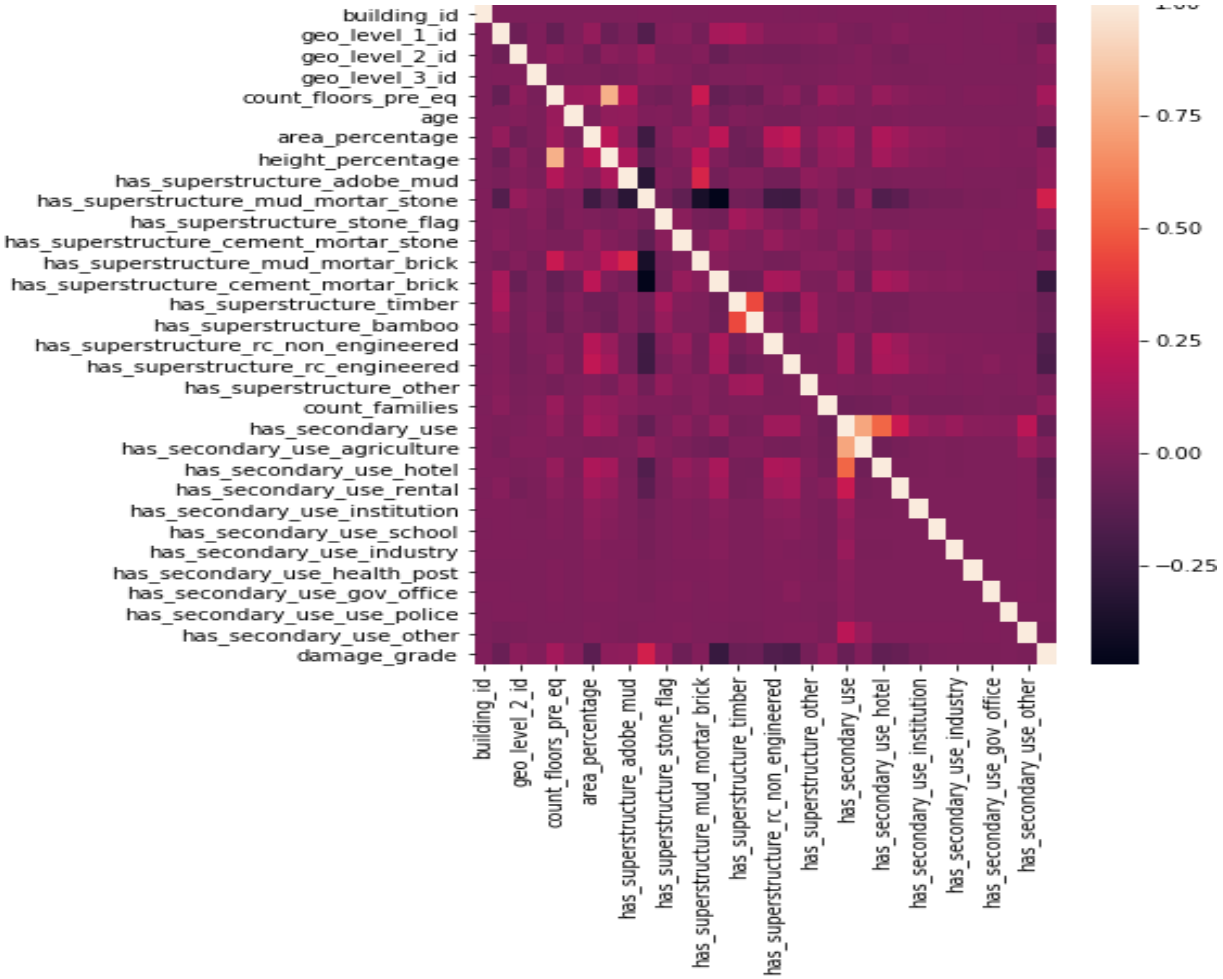
SINO	building_id	damage_grade
1	**47748**	2
2	**212102**	3
3	**60133**	3
4	**34181**	2
5	**25045**	2

df["damage_grade"] = labels["damage_grade"] df.head()

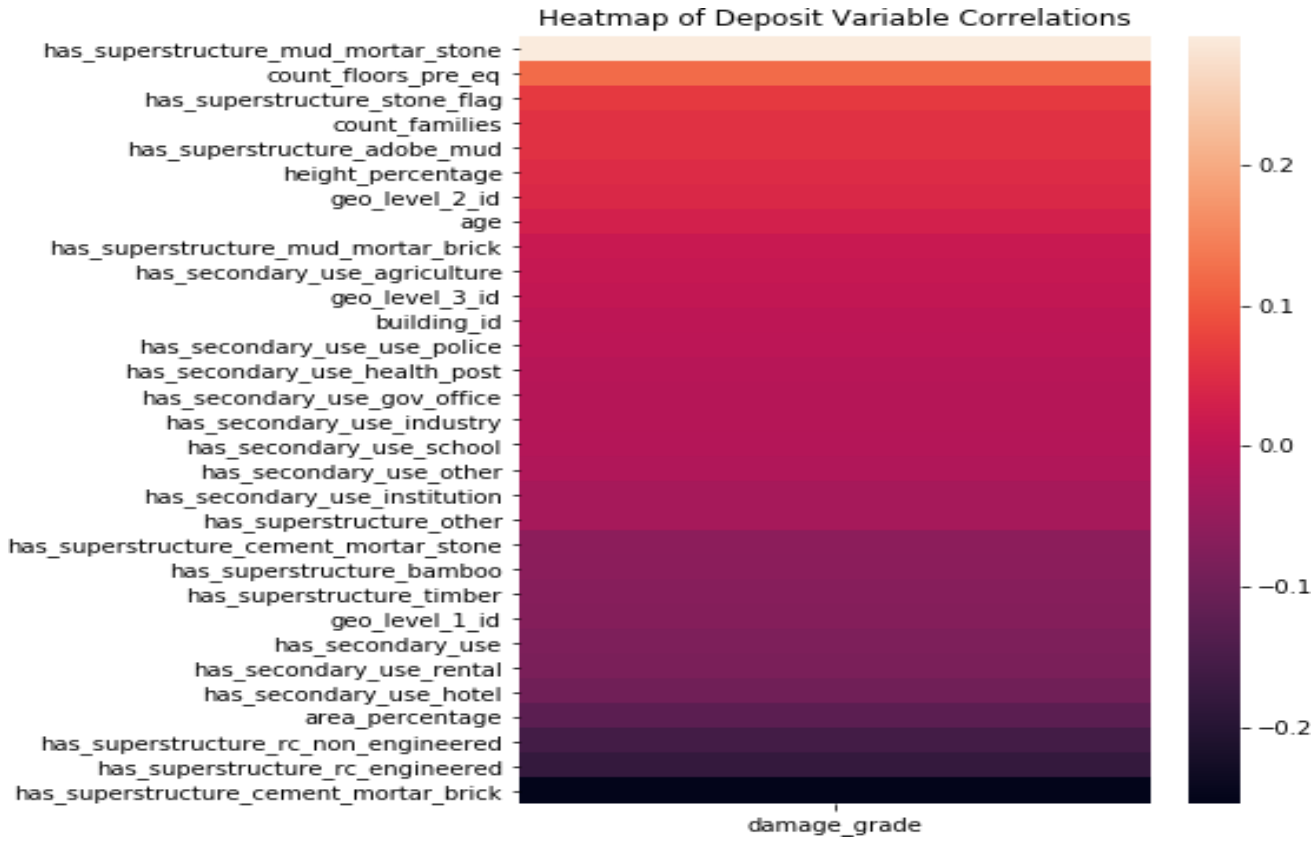
SINO	building_id	building_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	area_percentage	height_percentage
47748	4	30	266	1224	1	25	15	2
212102	8	17	409	12182	2	0	13	7
60133	12	17	716	7056	2	5	12	6
34181	16	4	651	105	2	80	5	4
25045	17	3	1387	3909	5	40	5	10

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

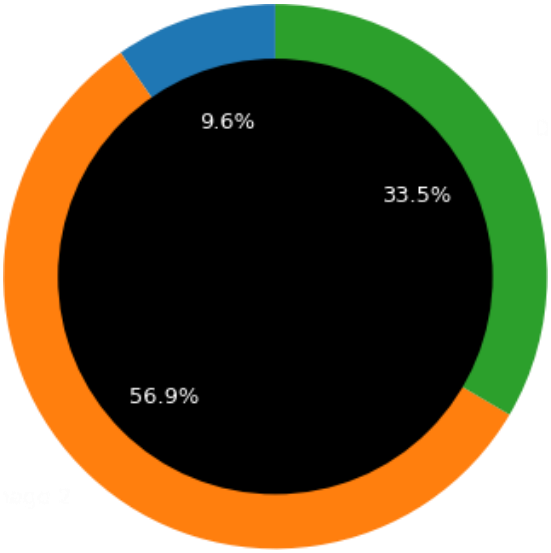
plt.rcParams["figure.figsize"] = (6,8) sns.heatmap(df.corr())
<matplotlib.axes._subplots.AxesS



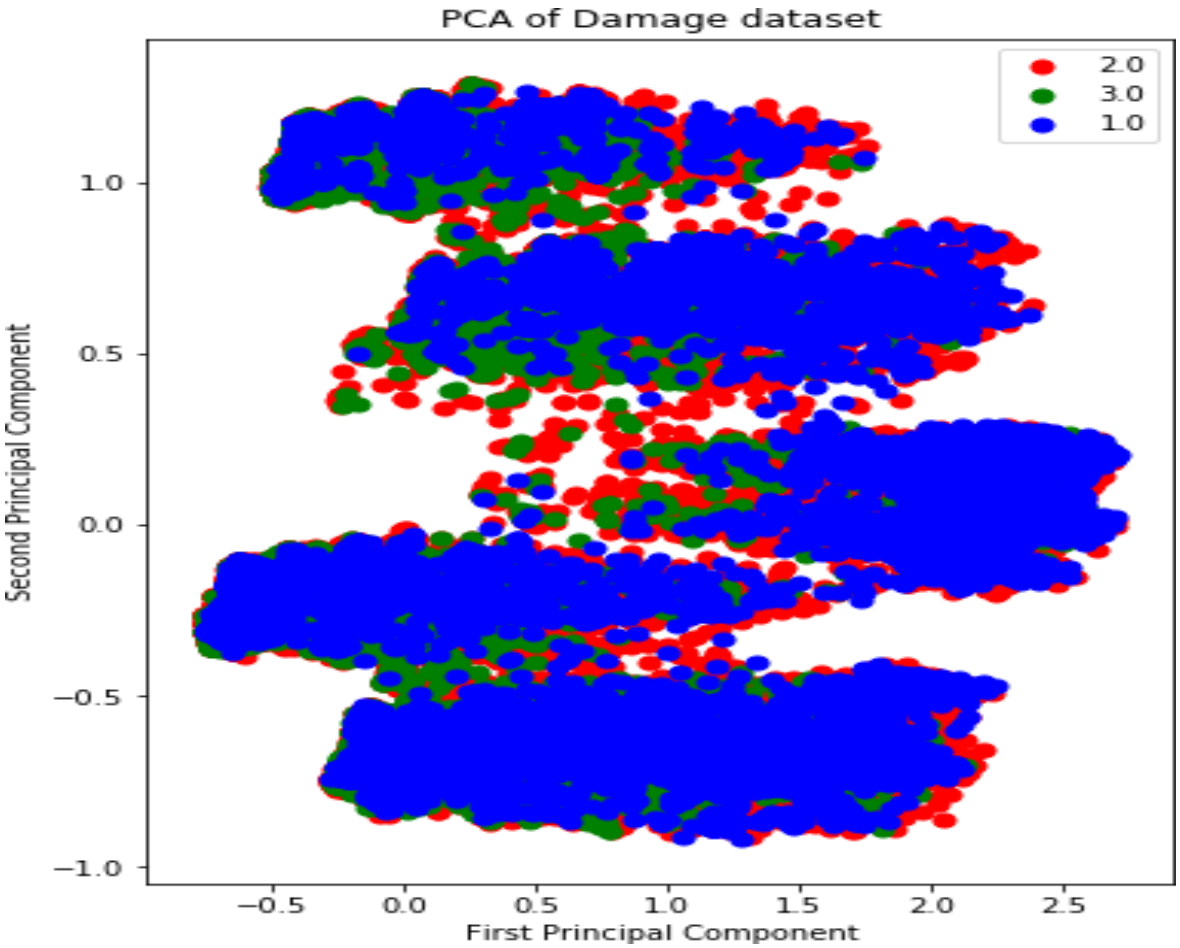
```
'damage_grade' correlation_matrix = df.corr()
 def plot_deposit_correlations(data): '''
     Isolates the deposit columns of the correlation matrix and visualize it. '''
     deposit_correlation_column =
 pd.DataFrame(correlation_matrix[DEPOSIT_COLUMN].drop(DEPOSIT_COLU MN))
     deposit_correlation_column =
 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')
```



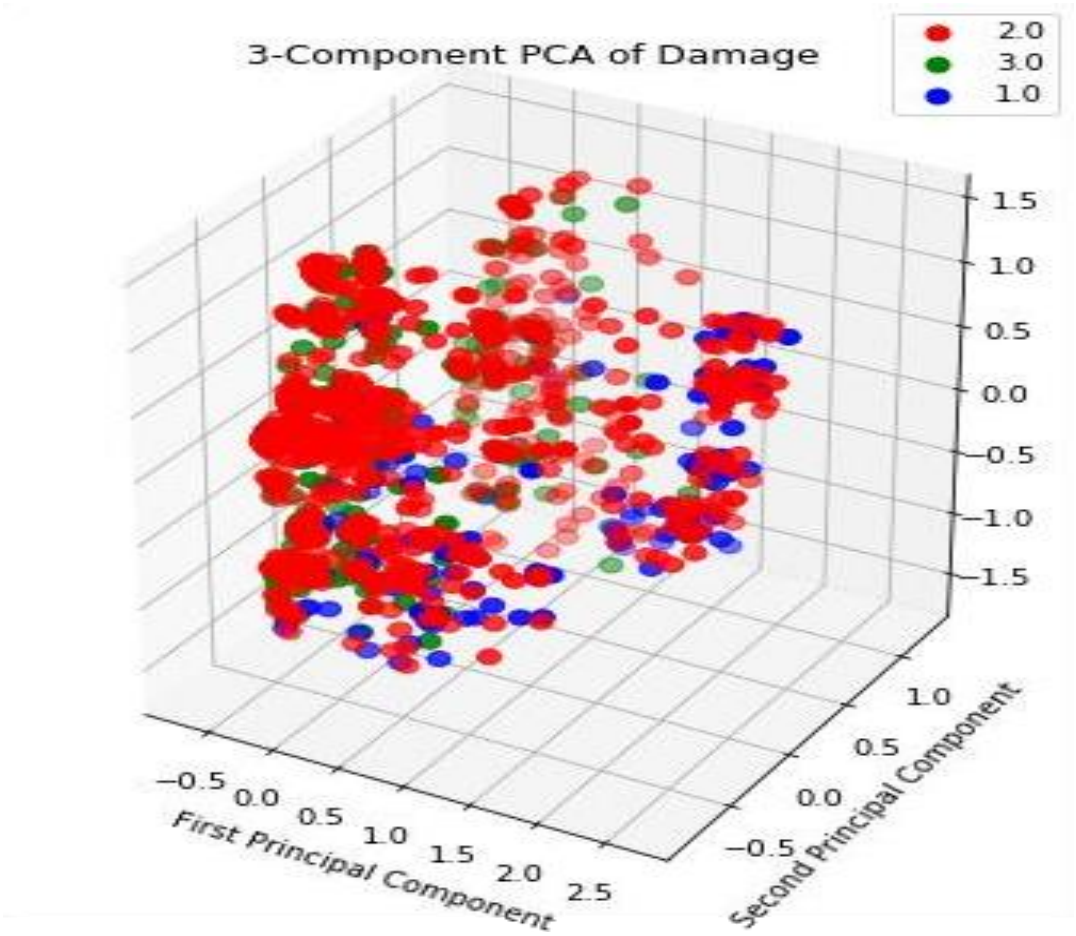

```
X = df.drop("damage_grade", axis=1) y =
df["damage_grade"]
targets = df["damage_grade"].unique()
print(targets)
```

```
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')
```

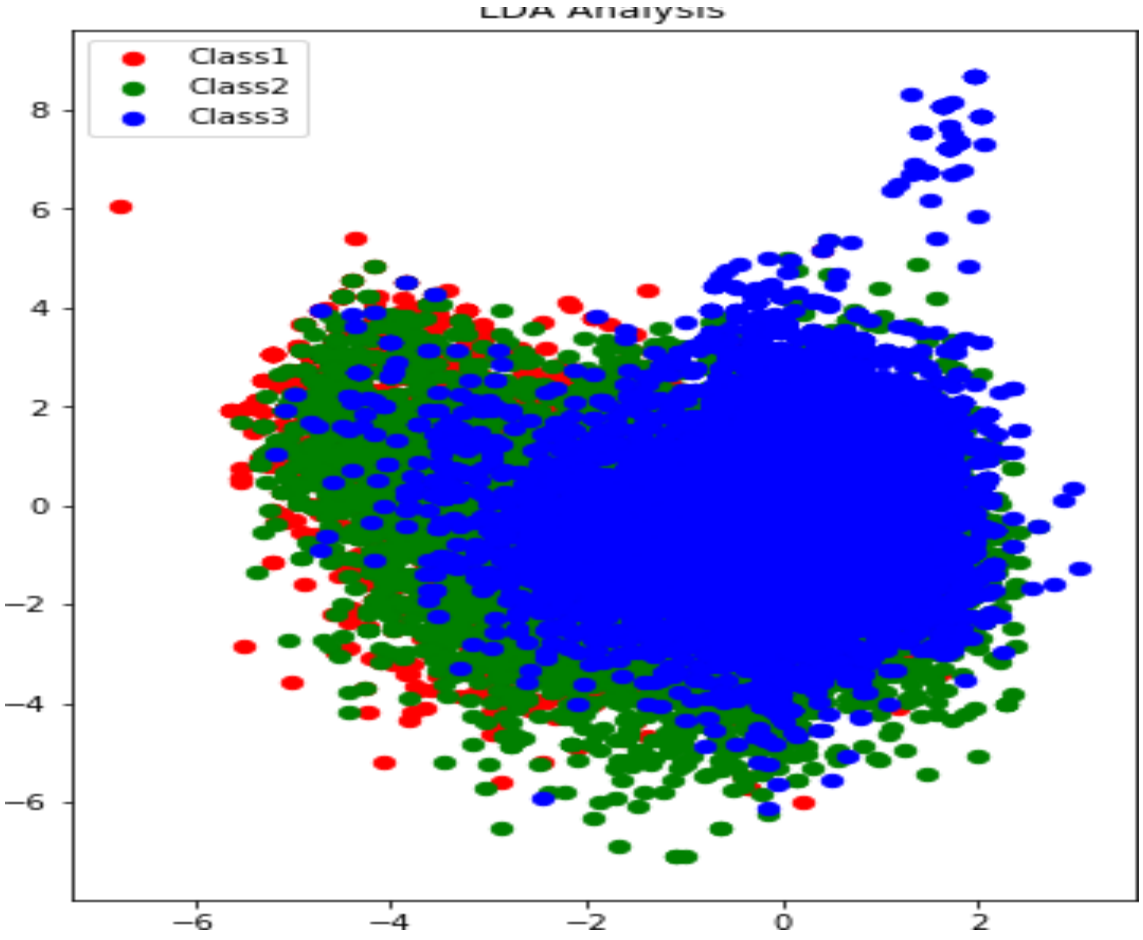



```
X = df.drop("damage_grade", axis=1).astype('int') y =
((df["damage_grade"] + 0.5) * 2).astype('int') lda =
LDA(n_components=2)
dmg_lda = lda.fit_transform(X, y)
print(dmg_lda)
l_x = dmg_lda[:,0] l_y =
dmg_lda[:,1]
cdict={1:'red',2:'green',3:'blue'}
labl={1:'Class1',2:'Class2',3:'Class3'} for l in
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>
```

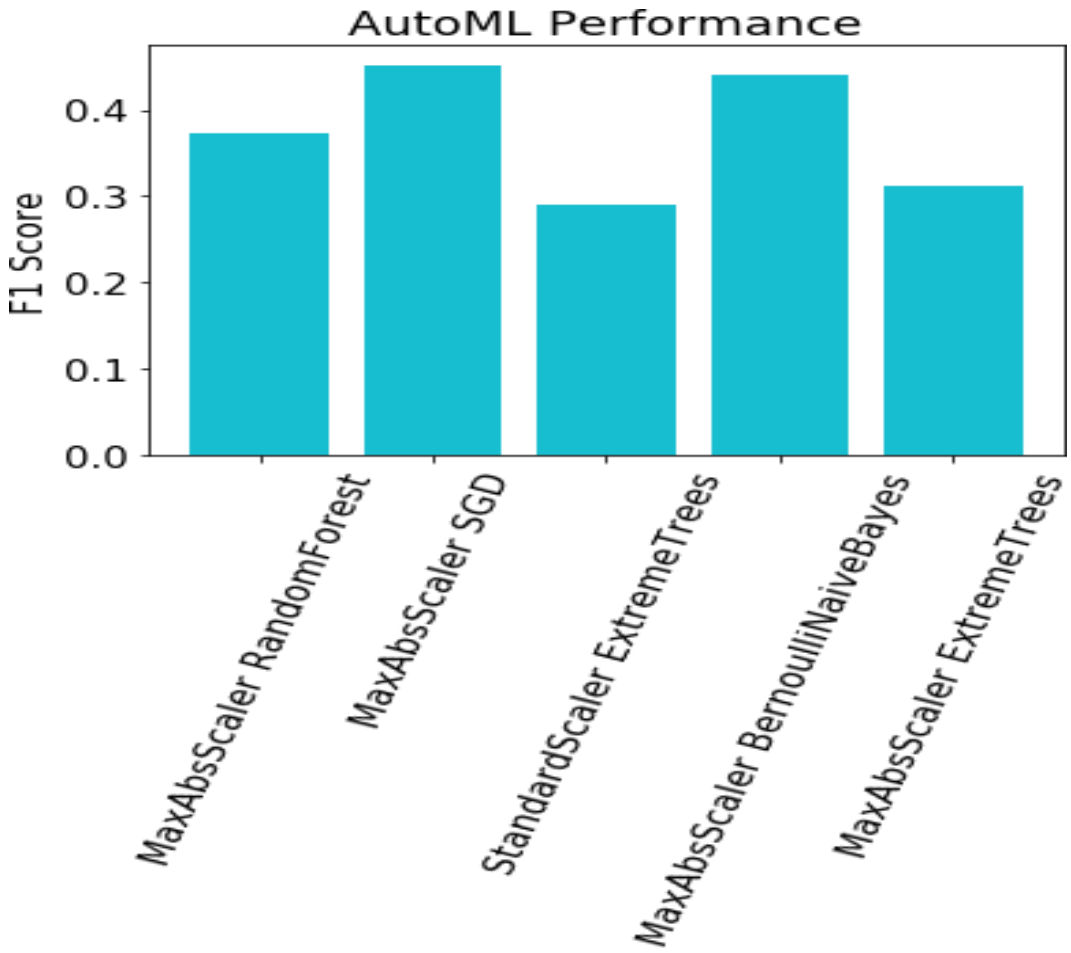


labels = ["MaxAbsScaler RandomForest", "MaxAbsScaler SGD", "StandardScaler ExtremeTrees", "MaxAbsScaler BernoulliNaiveBayes", "MaxAbsScaler ExtremeTrees"]

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.

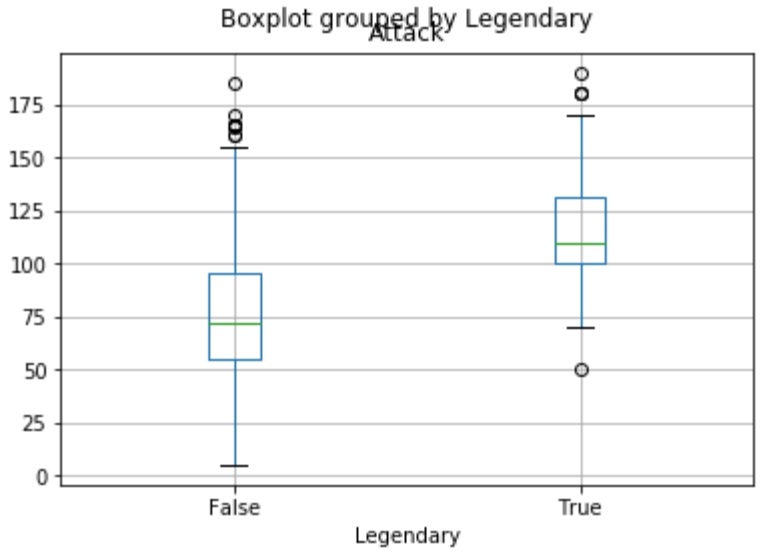
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns  # visualization tool
from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8"))
data = pd.read_csv('../input/pokemon-challenge/pokemon.csv')
#data.info()
data.head()
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65
1	2	Ivysaur	Grass	Poison	60	62	63	80	80
2	3	Venusaur	Grass	Poison	80	82	83	100	100
3	4	Mega Venusaur	Grass	Poison	80	100	123	122	120
4	5	Charmander	Fire	NaN	39	52	43	60	50

VISUAL EXPLOTARY DATA ANALYSIS:

```
data.boxplot(column='Attack',by = 'Legendary')
# For example: compare attack of pokemons that are legendary  or not
# Black line at top is max
# Blue line at top is 75%
# Red line is median (50%)
# Blue line at bottom is 25%
# Black line at bottom is min
# There are no outliers
```

<matplotlib.axes._subplots.AxesSubplot at 0x7eb7c84999e8>



TIDY DATA:

Tidy data is a concept introduced by statistician and data scientist Hadley Wickham. It provides a standa rd way to organize and structure datasets to facilitate easier analysis and visualization. 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 make it more suita ble for analysis or visualization. This operation is common when dealing with spreadsheet software or d ata manipulation libraries like Pandas in Python. Pivoting allows you to transform data from a long form at (where different types of information are stored in different rows) into a wide format (where informati on 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 r ow
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:

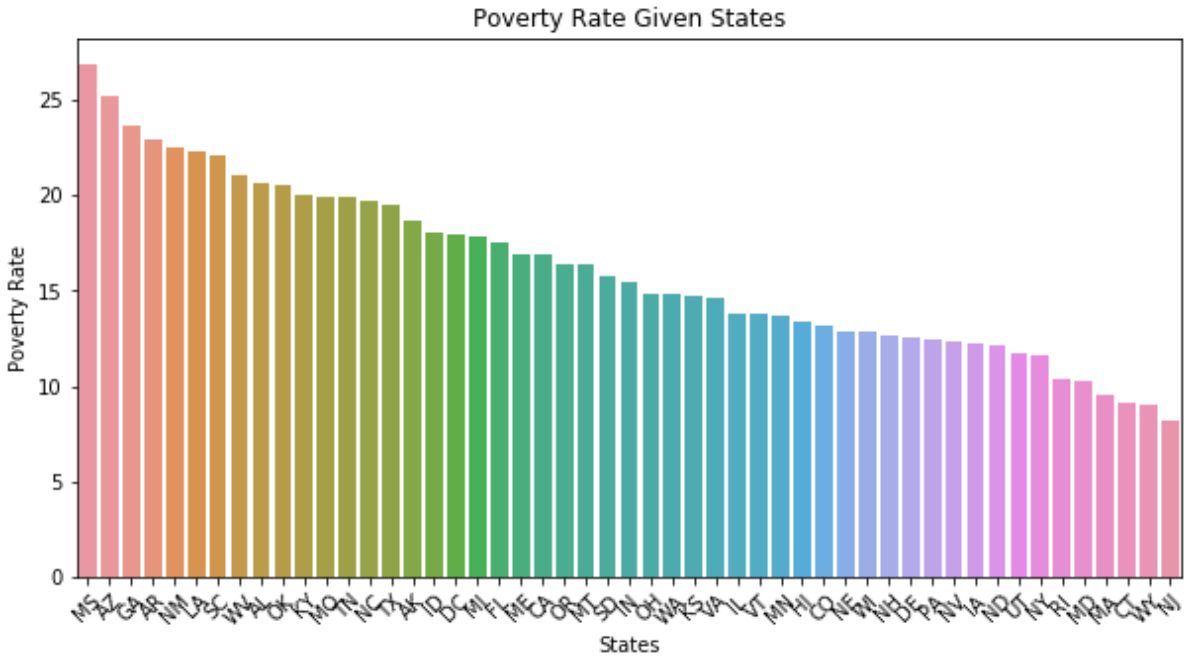
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline

percentage_people_below_poverty_level = pd.read_csv('../input/fatal-police-shootings-in-the-us/Percent agePeopleBelowPovertyLevel.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/Percent Over25CompletedHighSchool.csv', encoding="windows-1252")
```

linkcode

```
percentage_people_below_poverty_level.head()
```

OUTPUT:

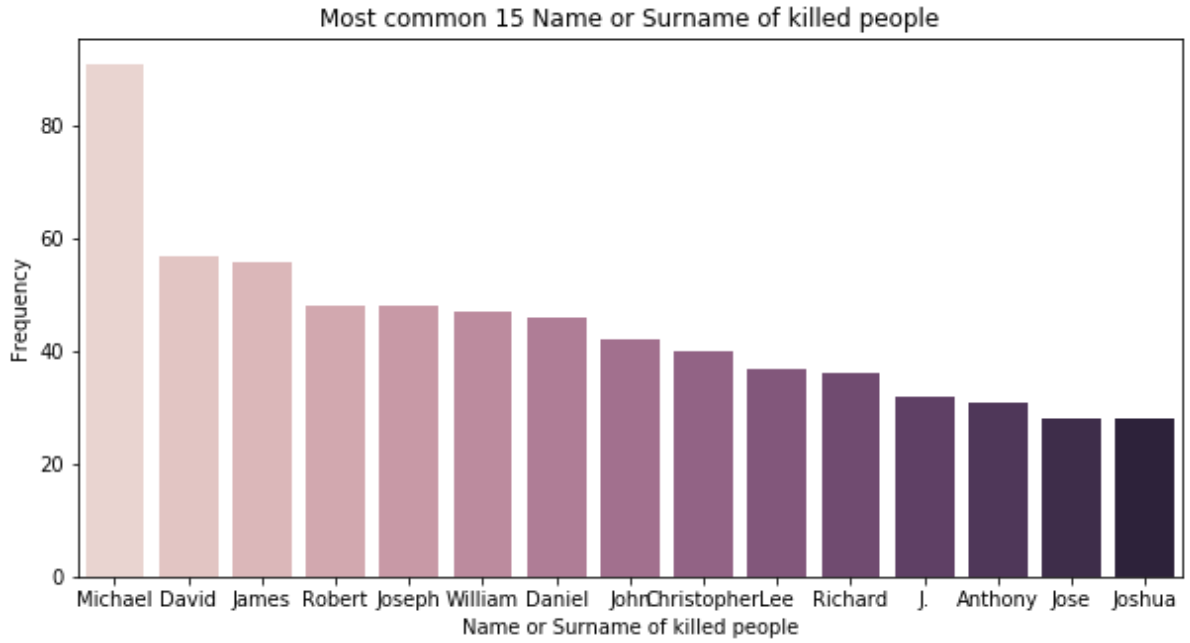
```
Text(0.5,1,'Poverty Rate Given States')
```




```
kill.head()
# Most common 15 Name or Surname of killed people
separate = kill.name[kill.name != 'TK TK'].str.split()
a,b = zip(*separate)
name_list = a+b
name_count = Counter(name_list)
most_common_names = name_count.most_common(15)
x,y = zip(*most_common_names)
x,y = list(x),list(y)
#
plt.figure(figsize=(10,5))
ax= sns.barplot(x=x, y=y,palette = sns.cubehelix_palette(len(x)))
plt.xlabel('Name or Surname of killed people')
plt.ylabel('Frequency')
plt.title('Most common 15 Name or Surname of killed people')
```

OUTPUT:

Text(0.5,1,'Most common 15 Name or Surname of killed people')



POINT PLOT:

A point plot is a type of data visualization that displays individual data points along with the summary statistics. It is particularly useful for comparing values of a categorical variable for different subgroups. Point plots are similar to scatter plots but are specifically used for categorical data.

In a point plot:
X-axis: Represents different categories or subgroups of the data.
Y-axis: Represents the values being compared.
Data Points: Individual data points are plotted for each category or subgroup**.**
Central Mark: Often, a line or a marker at the center of the data points represents the average or median value.
Error Bars (Optional): Error bars can be added to indicate the variability or uncertainty in the data.
Point plots are effective for comparing the central tendency (mean, median, etc.) of the data points for different categories. They provide a clear visualization of how the values vary across different subgroups.

```
percent_over_25_completed_highSchool.percent_completed_hs.replace(['-'],0.0,inplace = True)
percent_over_25_completed_highSchool.percent_completed_hs = percent_over_25_completed_highSchool.percent_completed_hs.astype(float)
area_list = list(percent_over_25_completed_highSchool['Geographic Area'].unique())
area_highschool = []
for i in area_list:
    x = percent_over_25_completed_highSchool[percent_over_25_completed_highSchool['Geographic Area']==i]
    area_highschool_rate = sum(x.percent_completed_hs)/len(x)
    area_highschool.append(area_highschool_rate)
# sorting
data = pd.DataFrame({'area_list': area_list,'area_highschool_ratio':area_highschool})
new_index = (data['area_highschool_ratio'].sort_values(ascending=True)).index.values
```

```
sorted_data2 = data.reindex(new_index)
# high school graduation rate vs Poverty rate of each state
sorted_data['area_poverty_ratio'] = sorted_data['area_poverty_ratio']/max( sorted_data['area_
poverty_ratio'])
sorted_data2['area_highschool_ratio'] = sorted_data2['area_highschool_ratio']/max( sorted_da
ta2['area_highschool_ratio'])
data = pd.concat([sorted_data,sorted_data2['area_highschool_ratio']],axis=1)
data.sort_values('area_poverty_ratio',inplace=True)

# visualize
f,ax1 = plt.subplots(figsize =(10,5))
sns.pointplot(x='area_list',y='area_poverty_ratio',data=data,color='lime',alpha=0.8)
sns.pointplot(x='area_list',y='area_highschool_ratio',data=data,color='red',alpha=0.8)
plt.text(40,0.6,'high school graduate ratio',color='red',fontsize = 17,style = 'italic')
plt.text(40,0.55,'poverty ratio',color='lime',fontsize = 18,style = 'italic')
plt.xlabel('States',fontsize = 15,color='blue')
plt.ylabel('Values',fontsize = 15,color='blue')
plt.title('High School Graduate  VS  Poverty Rate',fontsize = 20,color='blue')
plt.grid()
```

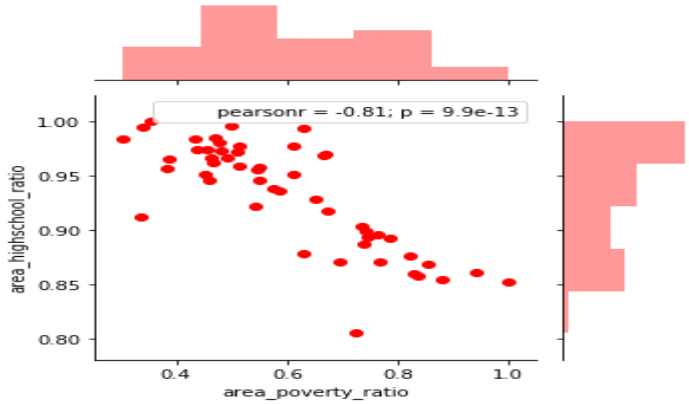
JOINT PLOT:

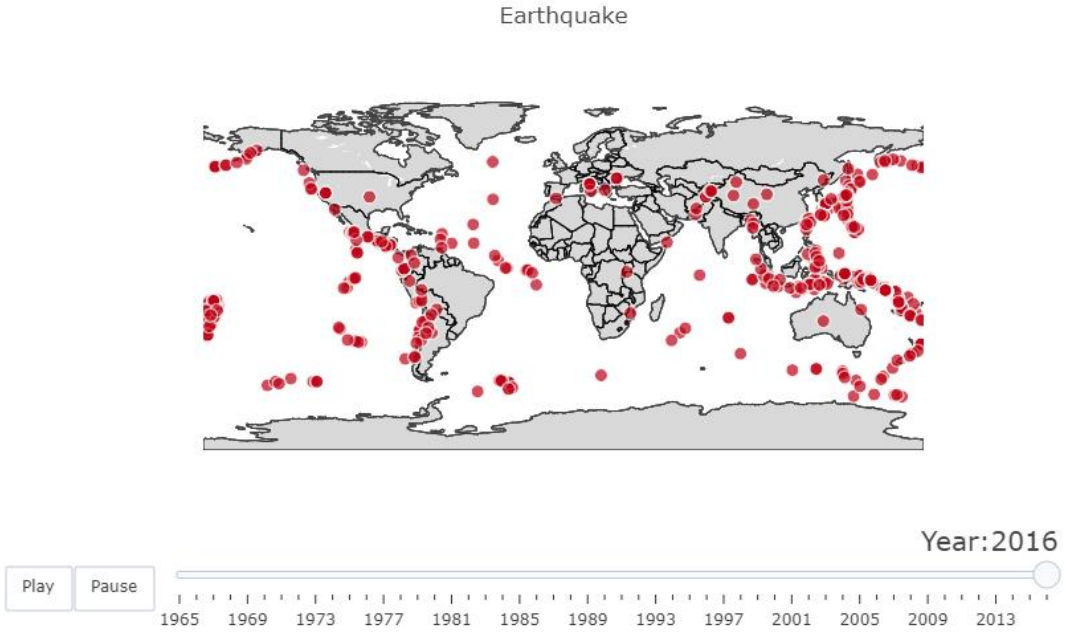
A point plot is a type of data visualization that displays individual data points along with the summary statistics. It is particularly A joint plot is a data visualization technique in statistics that combines multiple plots to show the relationship between two variables. It typically includes a scatter plot to represent the individual data points of the two variables, histograms or kernel density estimates (KDE) along the axes to show the distributions of each variable separately, and a correlation coefficient to quantify the relationship between the variables.

1.Scatter Plot
2.Histograms (or KDE)
3.Correlation Coefficient

```
# Visualization of high school graduation rate vs Poverty rate of each state with different styl
e of seaborn code
# joint kernel density
# pearsonr= if it is 1, there is positive correlation and if it is, -1 there is negative correlation.
# If it is zero, there is no correlation between variables
# Show the joint distribution using kernel density estimation
g = sns.jointplot(data.area_poverty_ratio, data.area_highschool_ratio, kind="kde", size=7)
plt.savefig('graph.png')
plt.show()
```

```
# you can change parameters of joint plot
# kind : { "scatter" | "reg" | "resid" | "kde" | "hex" }
# Different usage of parameters but same plot with previous one
g = sns.jointplot("area_poverty_ratio", "area_highschool_ratio", data=data,size=5, ratio=3, co
lor="r")
```



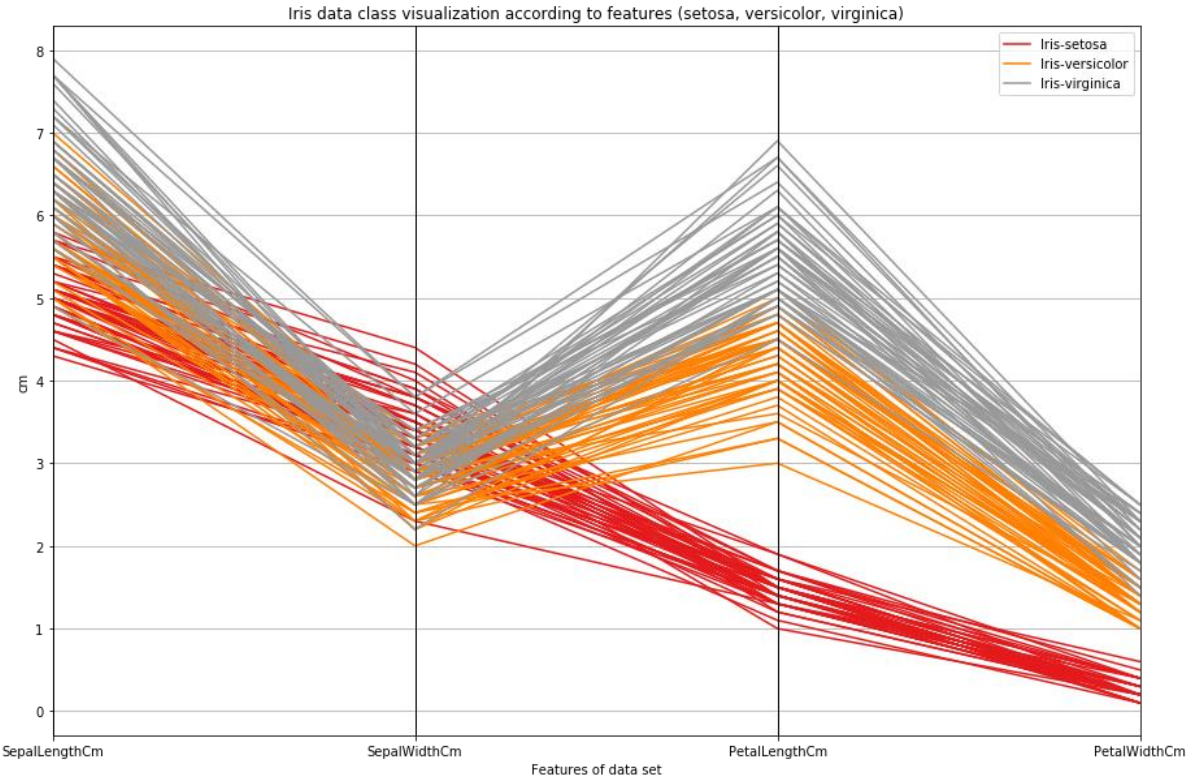


VISUALIZATION TOOLS:

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()



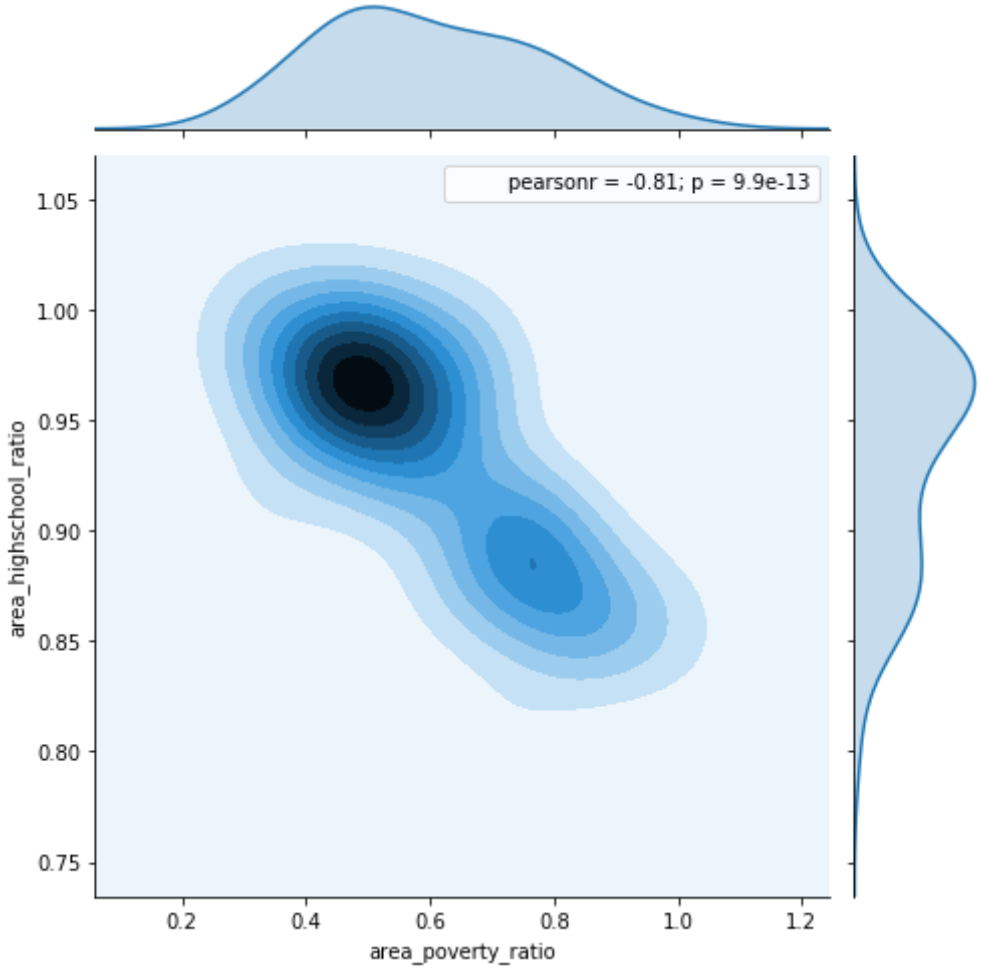
TESTING WITH ASSERTS:

 # Lets chech Type 2
data["Type 2"].value_counts(dropna =False)
 # As you can see, there are 386 NAN value

NaN 386
Flying 97
Ground 35
Poison 34
Psychic 33
Fighting 26
Grass 25
Fairy 23
Steel 22
Dark 20

```
Dragon      18
Rock        14
Water       14
Ice         14
Ghost       14
Fire        12
Electric     6
Normal       4
Bug          3
Name: Type 2, dtype: int64
```

```
 # Lets drop nan values
data1=data.copy()   # also we will use data to fill missing value so I assign it to data1 variable
data1["Type 2"].dropna(inplace = True)  # inplace = True means we do not assign it to new v
ariable. Changes automatically assigned to data
assert  data1['Type 2'].notnull().all() # returns nothing because we drop nan values
data1["Type 2"].fillna('empty',inplace = True) # ıstersen empty ıle de doldurabılırız
# # With assert statement we can check a lot of thing. For example
# assert data.columns[1] == 'Name'
# assert data.Speed.dtypes == np.int
```



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.