SURYA GROUP OF INSTITUTIONS

VIKRAVANDI-605652

# NAAN MUDHALVAN PROJECT

# EARTHQUAKE PREDICTION MODEL USING PYTHON

## PHASE-4

## DEVELOPMENT PART-2

PREPARED BY

NAME:S.RAGAVENDRAN

REG.NO:422221106306

DEPT/YEAR: ECE/3RD

## INTRODUCTION:

Earthquake pose significant threats to human lives and infrastructure.Predicting these natural disasters can save lives and minimize damage.In this project , we will explore the development of an earthquake prediction model using python, a powerful and versatile programming language. We will leverage the data analysis,machine learning techniques,and geographic visualization to gain insights into earthquake patterns.

## 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.

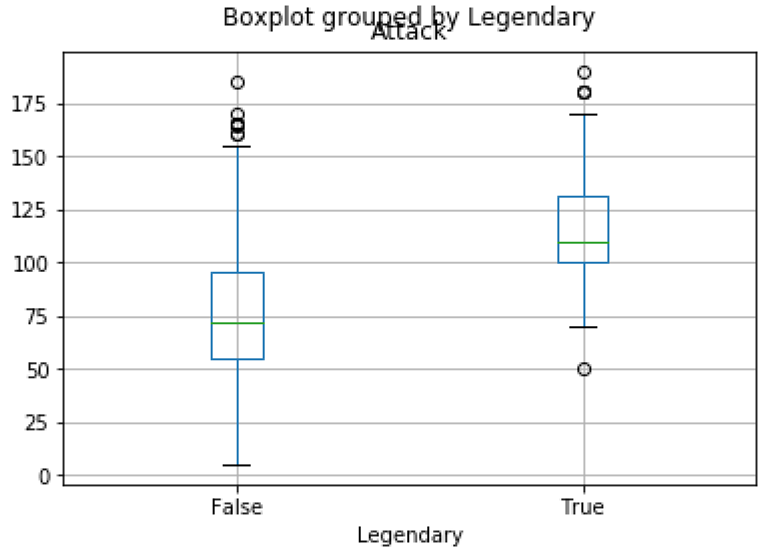
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns  # visualization tool
from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8"))
data = pd.read_csv('../input/pokemon-challenge/pokemon.csv')
#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 standard 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 suitable for analysis or visualization. This operation is common when dealing with spreadsheet software or data manipulation libraries like Pandas in Python. Pivoting allows you to transform data from a long format (where different types of information are stored 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 data
frames 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:

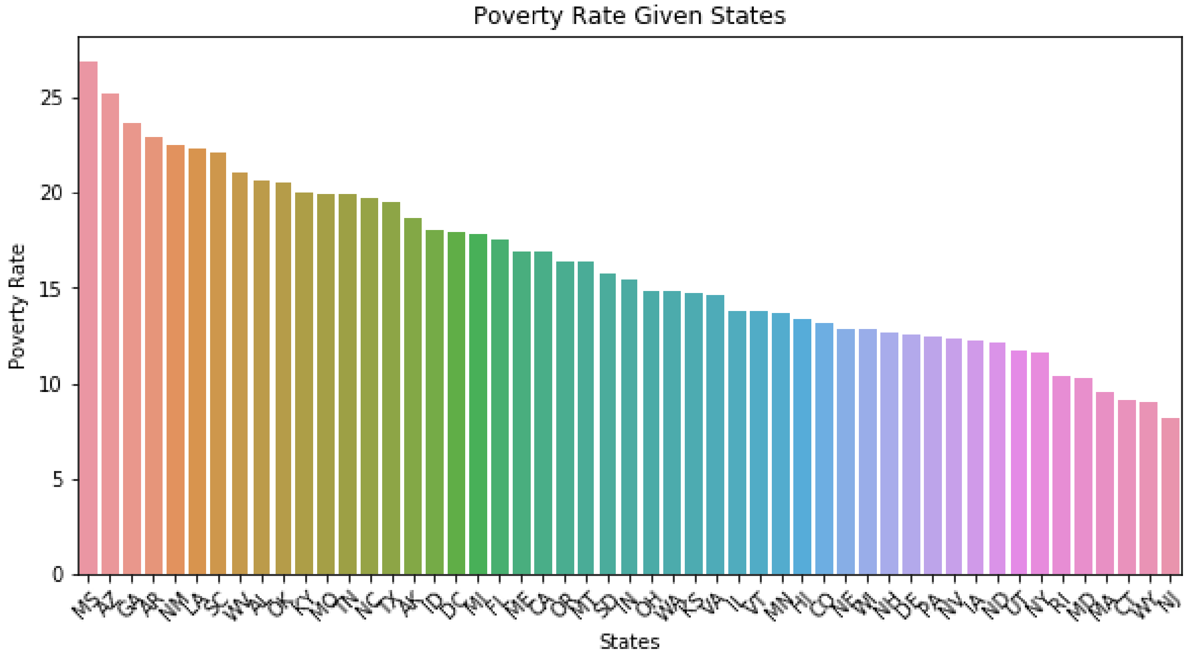
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline

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')
```

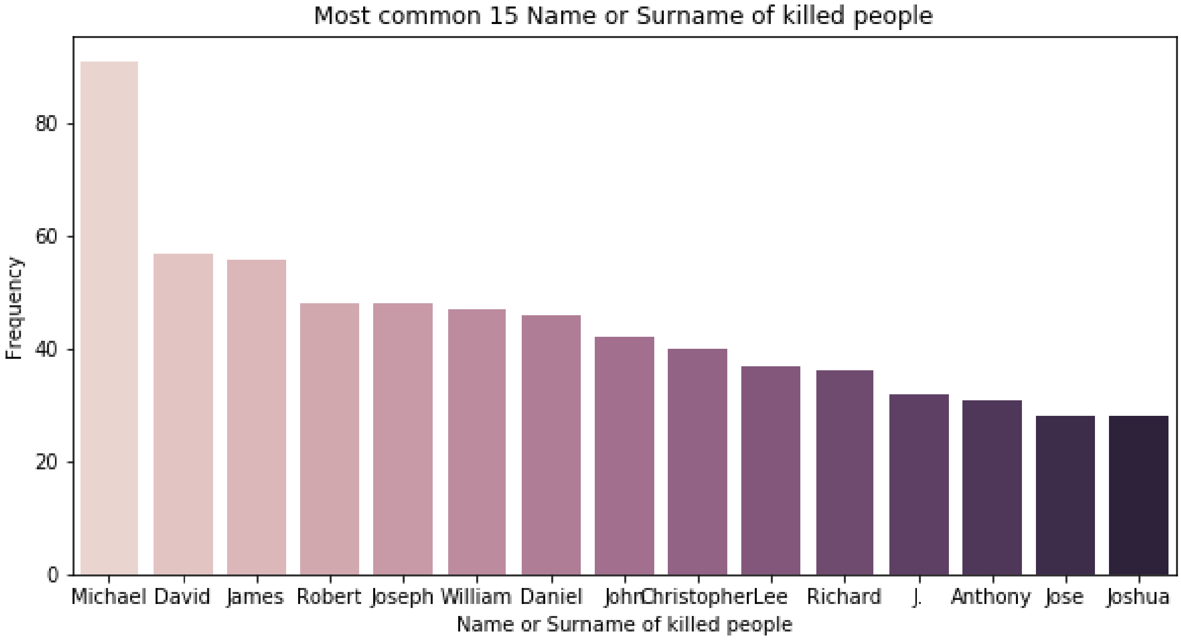


```
kill.head()
# Most common 15 Name or Surname of killed people
```

```
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 = Tr
ue)
percent_over_25_completed_highSchool.percent_completed_hs = percent_over_25_complet
ed_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['Ge
ographic 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:

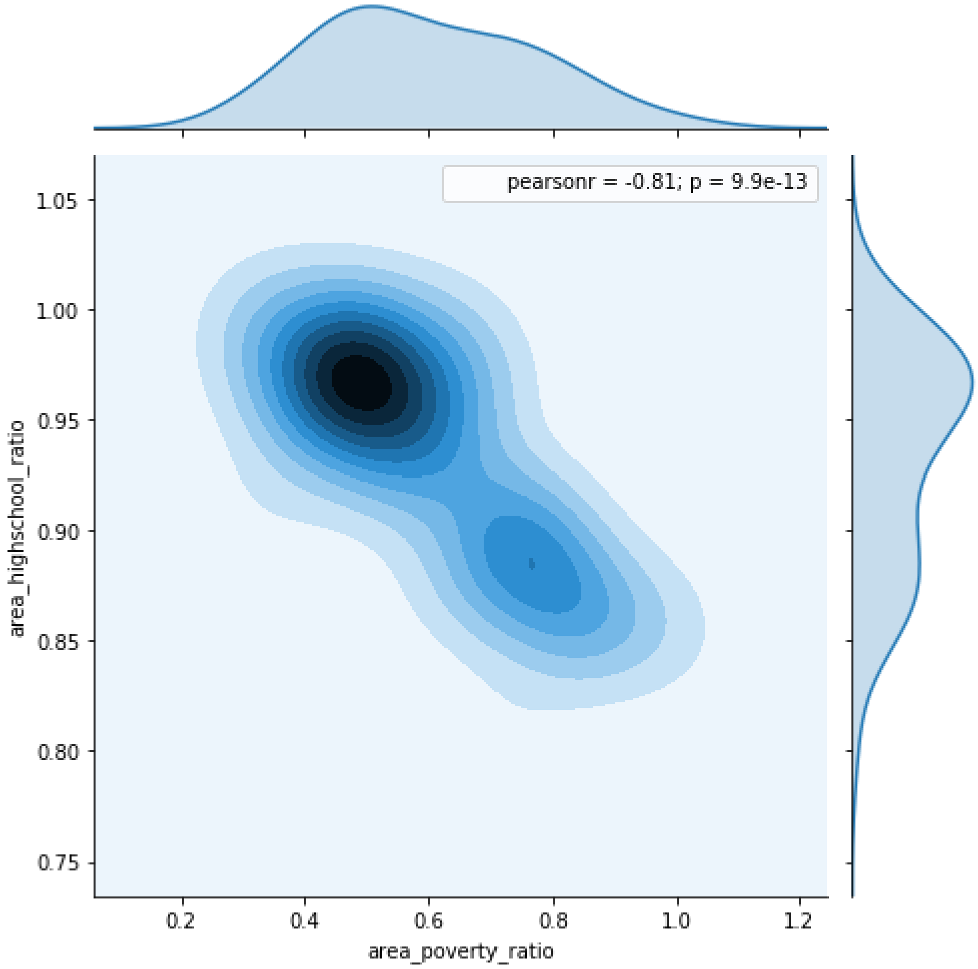
A point plot is a type of data visualization that displays individual data points along with the summary statistics. It is particularly A joint plot is a data visualization technique in statistics that combines multiple plots to show the relationship between two variables. It typically includes a scatter 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

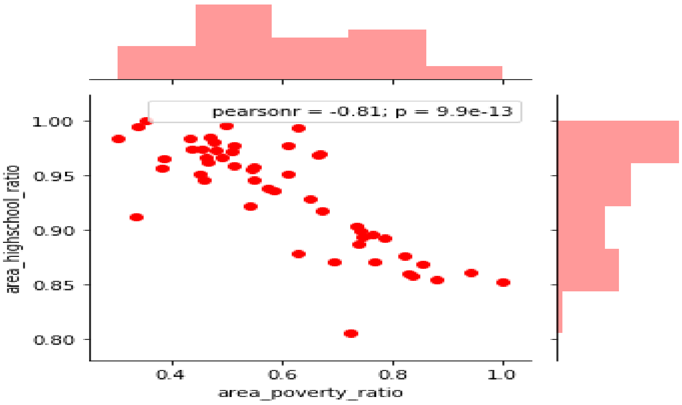
2.Histograms (or KDE)

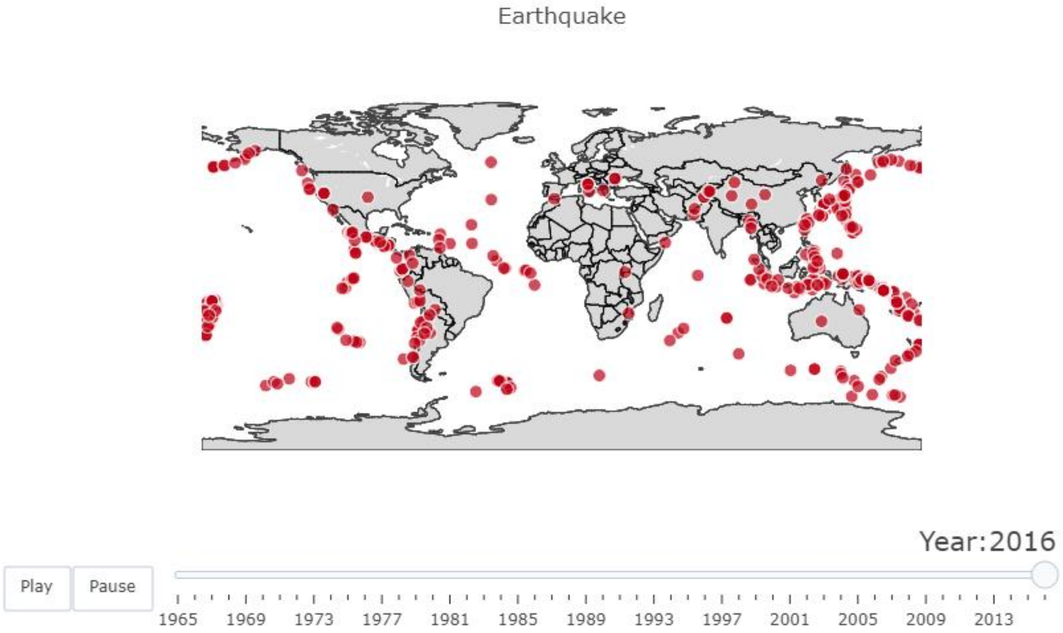
3.Correlation Coefficient

```
# Visualization of high school graduation rate vs Poverty rate of each state with different style 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, color="r")
```





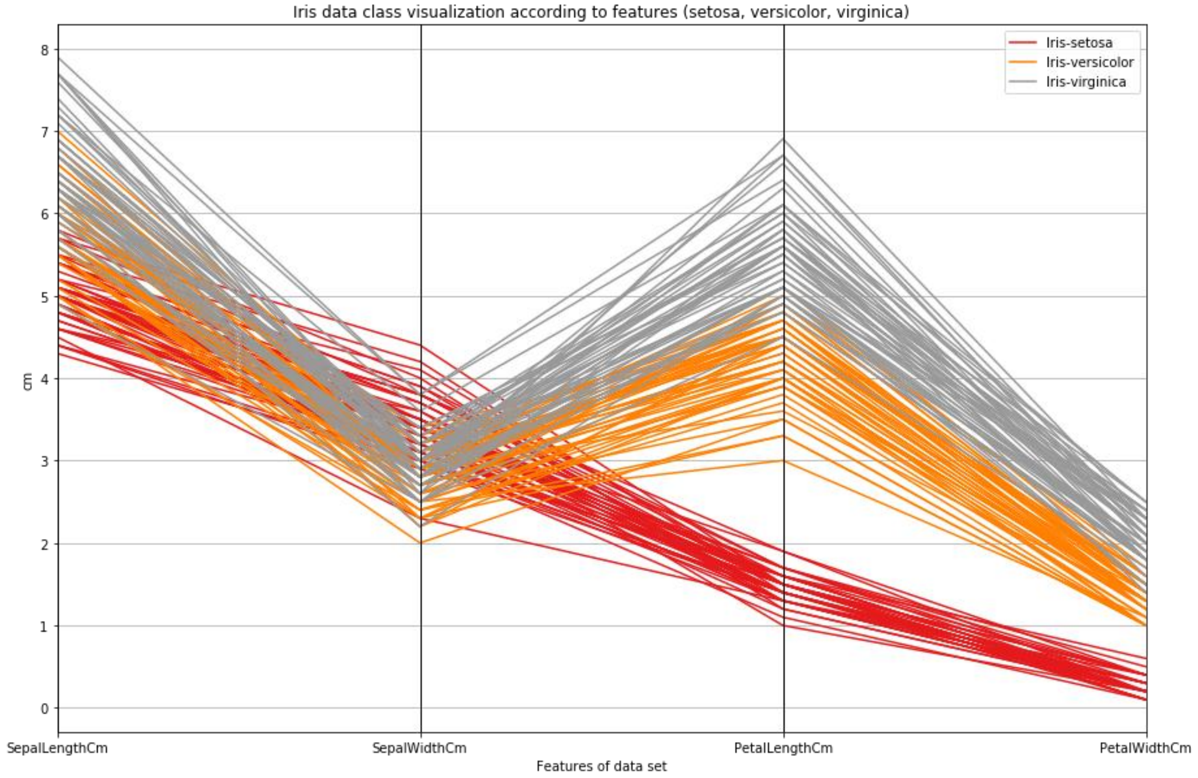
## VISUALIZATION TOOLS:

### 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()
```

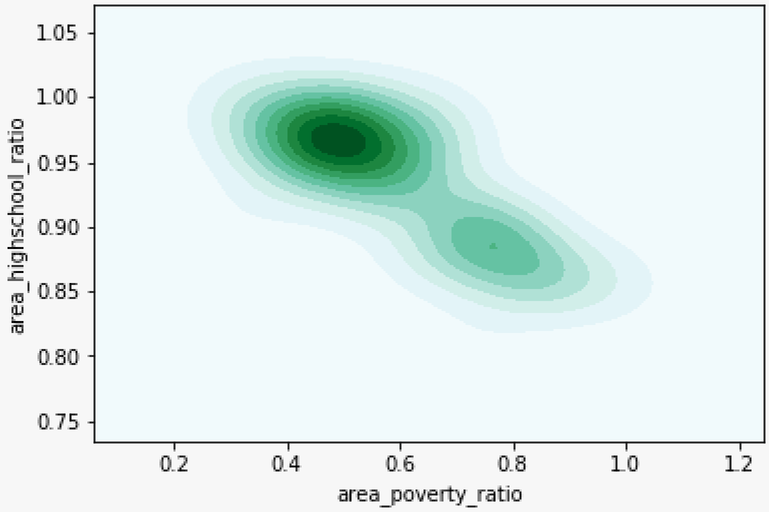


## TESTING WITH ASSERTS:

```
# 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 variable. 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 the realm of earthquake prediction, robust data visualization plays a pivotal role in enhancing our understanding, aiding accurate predictions, and enabling effective communication of findings. Through the comprehensive analysis and visualization of seismic data, we can draw valuable insights that are crucial for early warning systems, disaster preparedness, and scientific research.