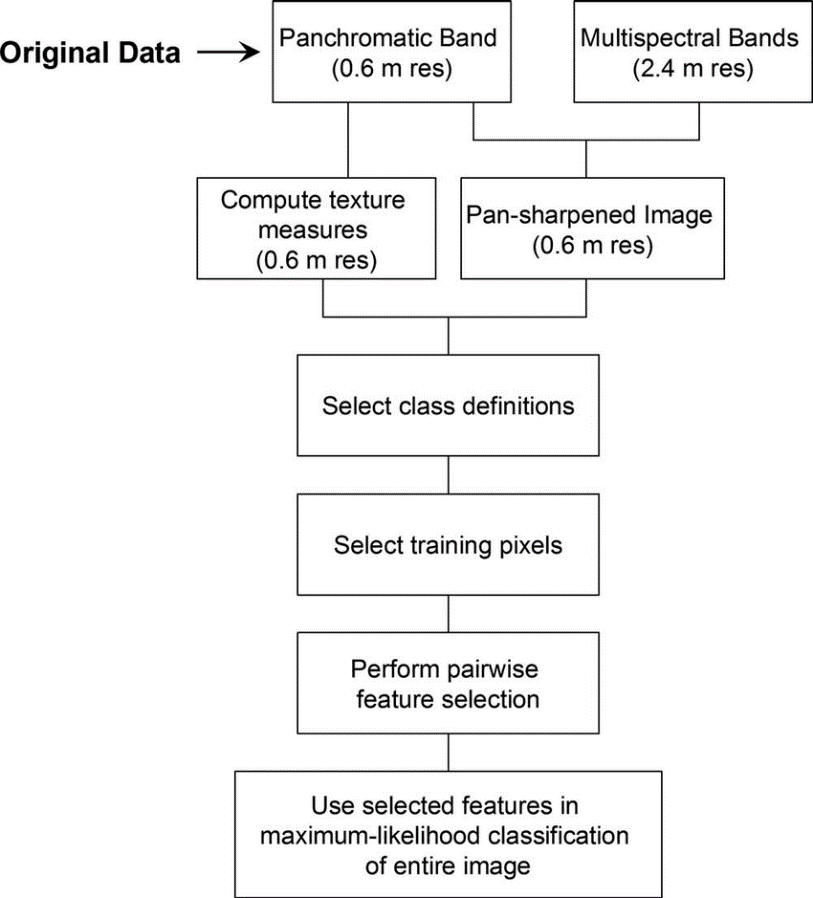
**Developing an Earthquake Prediction Model using python**

**Introduction:**

This document outlines a project plan for developing an earthquake prediction model using a Kaggle dataset. The primary goal of this project is to explore and understand key features of earthquake data, visualize the data on a world map, pre-process the data for model training, and build a neural network model for predicting earthquake magnitudes based on provided features



**Data Preprocessing:**

import pandas as pd

data = pd.read\_csv("earthquake\_data.csv")

print(data.head()

print(data.describe())

print(data.isnull().sum())

print(data.dtypes)

* **import pandas as pd:** This line imports the Pandas library and assigns it the alias 'pd.' This is a common convention when working with Pandas, making it easier to reference Pandas functions and methods.
* **data = pd.read\_csv("earthquake\_data.csv"):** This line reads the data from the CSV file "earthquake\_data.csv" and stores it in a Pandas DataFrame named 'data.' In this line, you're using the pd.read\_csv() function to read the CSV file and create a DataFrame, which is a table-like data structure in Pandas.
* **print(data.head()):** This line prints the first 5 rows of the DataFrame 'data' using the .head() method. The .head() method is used to display the top rows of the DataFrame to get an overview of the data.
* **print(data.describe()):** This line prints summary statistics of the numeric columns in the DataFrame using the .describe() method. It includes statistics like count, mean, standard deviation, minimum, and maximum values for each numeric column. This is helpful for getting a quick overview of the data's central tendencies and spread.
* **print(data.isnull().sum()):** This line prints the number of missing values in each column of the DataFrame using the .isnull().sum() method. It counts the number of NaN (Not-a-Number) values in each column, helping you identify which columns have missing data.
* **print(data.dtypes):** This line prints the data types of each column in the DataFrame using the .dtypes attribute. It shows whether each column contains integers, floats, strings, or other data types. This is helpful for understanding the data types of the features in the dataset.

**Data Visualization:**

Data visualization means using pictures, like charts and graphs, to show information from data. It helps people understand data better by making it easier to see patterns, trends, and important details. For example, a bar chart can show how different things compare, and a line chart can show how things change over time. Data visualization is like telling a story with pictures to make data more interesting and easier to use for decision-making.

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(8, 4))

sns.histplot(data['magnitude'], kde=True)

plt.title('Distribution of Earthquake Magnitudes')

plt.xlabel('Magnitude')

plt.show()

* **import matplotlib.pyplot as plt:** This line imports the Matplotlib library, which is a popular library for creating data visualizations, and assigns it the alias 'plt.' You can use 'plt' to access Matplotlib's functions and methods.
* **import seaborn as sns:** This line imports the Seaborn library, which is built on top of Matplotlib and provides a higher-level interface for creating attractive statistical visualizations.
* **plt.figure(figsize=(8, 4)):** This line creates a new figure (plot) with a specified figure size of 8 inches in width and 4 inches in height. This is done using Matplotlib's figure() function. Setting the figure size helps control the dimensions of the resulting plot.
* **sns.histplot(data['magnitude'], kde=True):** This line uses Seaborn's histplot() function to create a histogram plot of the 'magnitude' column from your DataFrame 'data.' The 'kde=True' argument indicates that a kernel density estimate (KDE) will be overlaid on top of the histogram, providing a smooth representation of the data's distribution.
* **plt.title('Distribution of Earthquake Magnitudes'):** This line sets the title of the plot to 'Distribution of Earthquake Magnitudes' using Matplotlib's title() function. This title provides context for the plot.
* **plt.xlabel('Magnitude'):** This line sets the label for the x-axis to 'Magnitude' using Matplotlib's xlabel() function. This label provides a description of what the x-axis represents.
* **plt.show():** This line displays the plot. In Matplotlib, this is how you actually render and show the plot on your screen.

data = data.dropna()

print(data.isnull().sum())

* **data = data.dropna():** This line of code is using the .dropna() method to remove rows with missing values (NaN or None) from the DataFrame 'data.' When you call this method without any arguments, it removes any row that contains at least one missing value. So, after this line of code, 'data' will no longer contain rows with missing data.
* **print(data.isnull().sum()):** This line prints the count of missing values in each column of the 'data' DataFrame using the .isnull().sum() method. It calculates how many missing values there are in each column after the rows with missing values have been removed. By printing this information, you can verify how many missing values remain in the DataFrame's columns after the data cleaning step.

m = folium.Map(location=[0, 0], zoom\_start=2)

for i, row in data.iterrows():

folium.CircleMarker(location=[row['latitude'], row['longitude']],

radius=2,

color='blue',

fill=True,

fill\_color='blue').add\_to(m)

m.save('earthquake\_map.html')

* **m = folium.Map(location=[0, 0], zoom\_start=2):** This line initializes a Folium map object named 'm.' It specifies the initial map location at latitude 0 and longitude 0 (the equator and the prime meridian) and sets the initial zoom level to 2. This line creates a map centered on these coordinates with an initial zoom level of 2.
* **for i, row in data.iterrows():** This for loop iterates through the rows of a DataFrame 'data' using the data.iterrows() method. 'i' is the index of the current row, and 'row' is a Pandas Series containing the data in that row.
* **folium.CircleMarker(location=[row['latitude'], row['longitude']], radius=2, color='blue', fill=True, fill\_color='blue').add\_to(m):** Within the loop, this code creates a blue circle marker on the map for each earthquake data point. It uses the latitude and longitude values from the current row to determine the marker's position. The 'radius' sets the size of the circle, 'color' determines the outline color (in this case, blue), and 'fill\_color' sets the color of the circle's interior. Finally, add\_to(m) adds each circle marker to the map 'm.'
* **m.save('earthquake\_map.html'):** After adding all the circle markers to the map, this line saves the map as an HTML file named 'earthquake\_map.html.' The resulting HTML file can be opened in a web browser to interact with the earthquake data on the map.

**Conclusion:**

To make a good earthquake prediction model, start with good data that's accurate and useful. Clean up the data by fixing any mistakes, pick the most important information, and make new facts that can help predict earthquakes. Put all the numbers in a format that's easy for the model to understand, and turn any words into numbers too. Divide the data into parts for the model to learn from and test on. To work with the data, use tools like Pandas and make pictures to help you understand it better. These early steps are super important because they set the stage for building a reliable earthquake prediction model.