```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd

def TermProject(StressDrop, TotalNumber, alph, L):
    # Constants
    np.random.seed(42)
    k = 2

    # Import dataframe from files
    df = pd.read_csv('query.csv')
    print(df)

    # Define the spherical to Cartesian conversion
    def geodeticTocartesian(lat, lon, depth):
        R = 6371000  # Radius of the Earth in meters
        latRad = np.radians(lat)
        lonRad = np.radians(lon)
        Rreal = R - depth  # Adjust radius for depth

        x = Rreal * np.cos(latRad) * np.cos(lonRad)
        y = Rreal * np.cos(latRad) * np.sin(lonRad)
        z = Rreal * np.sin(latRad)
        return x, y, z

    # Generating the Data
    FracX = df['longitude'].to_numpy()
    FracZ = df['latitude'].to_numpy()
    FracY = df['depth'].to_numpy()

    # Convert to Cartesian coordinates using the spherical to Cartesian transformation
    CartesianX, CartesianY, CartesianZ = geodeticTocartesian(FracZ, FracX, FracY)

    rndm = (np.random.rand(TotalNumber) + (1 / (100 * 10000))) * 100
    EquivR = ((alph * k) / (StressDrop * np.pi) * (10**6 / rndm)**(3 / 2))**(1 / 3)
    Area = np.pi * EquivR**2 / alph
    Trend = 360 * np.random.rand(TotalNumber)
    Plunge = 90 * np.random.rand(TotalNumber)
    Strike = 360 * np.random.rand(TotalNumber)

    # Preamble
    A = np.vstack((CartesianX, CartesianY, CartesianZ, Area, EquivR, Trend, Plunge, Strike)).T

    # Calculating moment magnitude
    M0 = A[:, 3] * A[:, 4] * StressDrop / k
    Mw = 2 / 3 * (np.log10(M0) - 9.1)
    CritFracX, CritFracY, CritFracZ = A[:, 0], A[:, 1], A[:, 2]

    # Initialize containers for plots
    V_list = []
    B_value_list = []
    nbin_list = []
    P_list = []

    # Randomly Generate Spatial Coordinates within fracture bounds
    n2 = 0
    for i in range(L):
        # Generate random coordinates within the projected fracture bounds
        x_min, x_max = np.min(CritFracX), np.max(CritFracX)
        y_min, y_max = np.min(CritFracY), np.max(CritFracY)
        z_min, z_max = np.min(CritFracZ), np.max(CritFracZ)

        x = (x_max - x_min) * np.random.rand(2) + x_min
        y = (y_max - y_min) * np.random.rand(2) + y_min
        z = (z_max - z_min) * np.random.rand(2) + z_min

        # Combine into a 2x3 array
        RandTemp = np.column_stack((x, y, z))

        # Sort each column independently
        SortRand = np.sort(RandTemp, axis=0)

        # Find all faults within the random volume
        n1 = 0
        RandMw = []
        for j in range(len(Mw)):
            if (SortRand[0, 0] < CritFracX[j] < SortRand[1, 0] and
                SortRand[0, 1] < CritFracY[j] < SortRand[1, 1] and
                SortRand[0, 2] < CritFracZ[j] < SortRand[1, 2]):
```

```
                n1 += 1
                RandMw.append([CritFracX[j], CritFracY[j], CritFracZ[j], Mw[j]])

        n2 += 1
        if n1 >= 50:  # Ensure there are at least 50 fractures/events
            RandMw = np.array(RandMw)
            SortedMw = np.sort(RandMw[:, 3])
            enumerate = np.arange(len(SortedMw), 0, -1)
            P = np.polyfit(SortedMw, np.log10(enumerate), 1)
            V = np.ptp(SortRand[:, 0]) * np.ptp(SortRand[:, 1]) * np.ptp(SortRand[:, 2])
            nbin = len(enumerate)

            # Append to lists for later plotting
            V_list.append(V)
            B_value_list.append(np.abs(P[0]))
            nbin_list.append(nbin)
            P_list.append(P)

    # Plot the Gutenberg-Richter Relationship at intervals
    plt.figure(1)
    plt.semilogy(SortedMw, enumerate, linewidth=4)
    plt.plot(SortedMw, 10**(P[1] + P[0] * SortedMw), linewidth=4)
    plt.title('Real Gutenberg-Richter Relationship')
    plt.xlabel('Moment Magnitude')
    plt.ylabel('Number of Events')
    plt.legend(['Events in the Region', 'Gutenberg-Richter Fit'])
    plt.xlim([np.min(SortedMw) - np.ptp(SortedMw) * 1/8, np.max(SortedMw) + np.ptp(SortedMw) * 6/8])
    plt.text(SortedMw[-1], 10**(P[1] + P[0] * SortedMw[-1]),
             f'log(N) = {round(P[1], 2)} + {round(P[0], 2)}Mw', fontsize=15)
    plt.text(SortedMw[-1], 40, f'n={nbin}', fontsize=15)
    plt.text(SortedMw[-1], 20, f'Volume(m^3)={int(V)}', fontsize=15)
    plt.gca().tick_params(labelsize=15, width=4)

    # Plot B-Value Distribution with Volume
    plt.figure(2)
    plt.plot(V_list, B_value_list, '.', markersize=15)
    sorted_indices = np.argsort(V_list)
    sorted_V = np.array(V_list)[sorted_indices]
    sorted_B_value = np.array(B_value_list)[sorted_indices]
    window_size = 50
    moving_average = np.convolve(sorted_B_value, np.ones(window_size)/window_size, mode='valid')
    plt.plot(sorted_V[window_size-1:], moving_average, 'm', linewidth=1.2)
    plt.title('Real B-Value Distribution with Volume')
    plt.xlabel('Volume (m^3)')
    plt.ylabel('B-Value')
    plt.legend(['B-Value for each region', 'Moving Average (n=50)'])
    plt.gca().tick_params(labelsize=15, width=4)

    # Plot Earthquake Activity
    plt.figure(3)
    plt.plot(nbin_list, [P[1] for P in P_list], '.', markersize=15)
    def model(x, b0, b1, b2):
        return b0 + b1 * np.log10(b2 * x)
    beta, _ = curve_fit(model, nbin_list, 3 + np.array([P[1] for P in P_list]))
    plt.plot(np.sort(nbin_list), -3 + beta[0] + beta[1] * np.log10(beta[2] * np.sort(nbin_list)), linewidth=4)
    plt.title('Real Earthquake Activity')
    plt.xlabel('Number of Events in the Region')
    plt.ylabel('A-Value')
    plt.text(250, -.25, f'A = {round(-3 + beta[0], 1)} + {round(beta[1], 1)} log({round(abs(beta[2]), 1)}N)', fontsize=15)
    plt.gca().tick_params(labelsize=15, width=4)

    # Plot Number of Events vs. Volume
    plt.figure(4)
    plt.plot(V_list, nbin_list, '.', markersize=20)
    VolFit = np.polyfit(V_list, nbin_list, 1)
    V_list = np.array(V_list)
    plt.plot(V_list, VolFit[0] * V_list + VolFit[1], linewidth=4)
    plt.title('Real Volume vs Number of Events in Each Region')
    plt.xlabel('Volume (m^3)')
    plt.ylabel('Number of Events in Each Region')
    plt.gca().tick_params(labelsize=15, width=4)

    plt.show()

TermProject(StressDrop=10, TotalNumber=785, alph=0.7, L=5000)
```