Python Code of Seismic Vulnerability of Rock Tunnels in India and Adjacent Countries

```python
from PIL import Image, ImageTk
from geopy import distance
import matplotlib.pyplot as plt
import subprocess
from pandas import read_excel  # Add this line
import math
from tkinter import *
from tkinter import messagebox
import sys
import datetime

# Add a counter to check if it's the first time compute_tasks is called
compute_tasks_counter = 0
compute_tasks_counter_dsha = 0

def compute_tasks_tunnel(active_sheet, location1, RMR, OD, pga_list):
    global compute_tasks_counter
    if compute_tasks_counter == 0:
        text_to_print = """ The Seismic Damages of Rock Tunnels are
Classified as Damages to Lining, Portal, and Invert. The sub-categories of
those damages are:
            Lining Damage (1):
            Collapse of a Tunnel/Sheared-off Lining (A11)
            Grade 1 and 2 -Lining Dislocation (B11 and B12 respectively)
            Grade 1 and 2- Lining Spalling (C11 and C12 respectively)
            Lining Deformation (D11)
            Grade-1, 2, 3 and 4 Lining cracks (E11, E12, E13 and E14
respectively)

            Portal Damage(2):
            Grade-1 Portal/ Slope Damage (A21)
            Grade-2,3,4, and 5 Portal Damage (B21,B22,B23 and B24
respectively)

            Invert Damage(3):
            Grade 1,2 and 3-Invert Damage (A31, A32, and A33 respectively)
            N- Indicates No Damage Scenario
            The amount of damage for each category and the accessibility of
the tunnel for each damage class are elaborated at the bottom of this
report"""
        print(

'**************************************************************************
***************************************************************************
************************')
        print(text_to_print)
        print(

'**************************************************************************
***************************************************************************
************************')
        print(
            "The Results of Seismic Vulnerability of Tunnel for a given
lat, long are in the form of \n "
            "Results: {Fault ID, Fault Name, Fault Layer, Source-Site
Distance (Lat, Long), Total Fault Length, Predicted Magnitude, PGA, Damage
Class, Probable Damages to Tunnel}\n")
        compute_tasks_counter += 1  # Update the counter
```

```
    # start_time = time.time()
    seismic_faults = {}
    fault_ct = 0
    threshold_dist = 250
    current_site = 'x'
    row_no = 1

    for index, row in active_sheet.iterrows():
        location2 = (row.iloc[14], row.iloc[15])  # Assuming latitude is in
column 14 and longitude is in column 15

        if row_no == 1:
            row_no += 1
        elif row_no == 2:
            if current_site != row.iloc[1]:  # Assuming 'Fault ID' is in
column 1
                previous_least_dist = threshold_dist
                current_site = int(row.iloc[1])  # Convert 'Fault ID' to
integer if necessary
                dist_value = round(distance.distance(location1,
location2).km, 2)
                if dist_value < threshold_dist and dist_value <
previous_least_dist:
                    previous_least_dist = dist_value
                    seismic_faults[current_site] = [row.iloc[1],
row.iloc[2], row.iloc[7], dist_value, location2, row.iloc[17],
row.iloc[18]]
            else:
                dist_value = round(distance.distance(location1,
location2).km, 2)  # Recalculate distance
                if dist_value < threshold_dist and dist_value <
previous_least_dist:
                    previous_least_dist = dist_value
                    seismic_faults[current_site] = [row.iloc[1],
row.iloc[2], row.iloc[7], dist_value, location2,
                                                    row.iloc[17],
row.iloc[18]]

    for key, value in seismic_faults.items():
        log_pga = 0.56 * value[6] - 0.0031 * value[3] - math.log10(
            value[3] + 0.0055 * 10 ** (0.5 * value[6])) + 0.26 + 0.37
        PGA = round((10 ** log_pga) * 0.00102, 2)
        pga_list.append((PGA, key, value))  # Store PGA, key, and value in
the list


def save_to_file(content, file_name):
    with open(file_name, 'w') as file:
        file.write(content)


def open_file_with_default_editor(file_name):
    subprocess.Popen(["notepad.exe", file_name], shell=True)


def retrieve_tunnel():
    # Get the current date and time
    current_time = datetime.datetime.now()

    # Format the date and time to include in the filename
    formatted_time = current_time.strftime("%Y-%m-%d_%H-%M-%S")
```

```
    # Create the filename with the formatted date and time
    filename = f"Tunnel_Report_{formatted_time}.txt"

    with open(filename, "w+", encoding="utf-8") as f:
        sys.stdout = f
        rmr = 0
        od = 0

        # print("Submitted Data")
        print("The Generated Report is as follows:")
        latitude = float(textbox1.get("1.0", "end-1c"))
        longitude = float(textbox2.get("1.0", "end-1c"))
        RMR = int(textbox3.get("1.0", "end-1c"))
        OD = float(textbox4.get("1.0", "end-1c"))
        Lining = lining_var.get()
        Shape = shape_var.get()


        print("latitude is ", latitude)
        print("longitude is ", longitude)
        print("RMR is ",RMR)
        print("OD is ", OD, "m")
        print("Lining Type is ", Lining,                 "(The order of
criticality of lining type is UL>TL>ML>CL>RCL)")
        print("Shape is ", Shape,                          "(The order of
criticality of shape is RE>AH>D>HS>OV>CR)")

        if RMR > -1 and RMR < 101:
            rmr = 1
        else:
            messagebox.showerror("Incorrect Input", "Please enter RMR
between 1 and 100")

        if OD > 0 and OD < 4001:
            od = 1
        else:
            messagebox.showerror("Incorrect Input", "Please enter OD
between 1 and 4000")

        if rmr == 1 and od == 1:
            location1 = (latitude, longitude)

            active_sheet1 = read_excel("Active_faults_Order.xlsx")
            active_sheet2 = read_excel("Fault_Tectonic_order.xlsx")
            active_sheet3 = read_excel("Thrust_Tectonic_order.xlsx")
            active_sheet4 = read_excel("Himatibet_faults_Order.xlsx")
            active_sheet5 = read_excel("Iran_faults_ order.xlsx")

            pga_list = []  # Initialize the list to store PGA values

            compute_tasks_tunnel(active_sheet1, location1, RMR, OD,
pga_list)
            compute_tasks_tunnel(active_sheet2, location1, RMR, OD,
pga_list)
            compute_tasks_tunnel(active_sheet3, location1, RMR, OD,
pga_list)
            compute_tasks_tunnel(active_sheet4, location1, RMR, OD,
pga_list)
            compute_tasks_tunnel(active_sheet5, location1, RMR, OD,
pga_list)
```

```
            # Sort the list based on PGA values from highest to lowest
            sorted_pga_list = sorted(pga_list, key=lambda x: x[0],
reverse=True)

            # Initialize counters for each damage class
            damage_counts = {
                "Extremely High": 0,
                "Very High": 0,
                "High": 0,
                "Moderate": 0,
                "Low": 0,
                "Very Low": 0,
                "Extremely Low": 0
            }

            # Print the sorted results
            for pga, key, value in sorted_pga_list:
                print(

"**************************************************************************
***************************************************************************
******************\n")
                print(f"Results: {value}")
                # print(
                # "Key: {}, Value: {}, Source-Site Distance (Lat, Long): {}
km, Total Fault Length: {}, Predicted Magnitude: {}, PGA: {:.2f}, Damage
Class: {}, Probable Damages to Tunnel: {}".format(
                #  key, value[0], round(value[3], 2), value[4], value[5],
round(PGA, 2), "Not Calculated",
                # "Not Calculated"))
                print("The PGA is ", round(pga, 2), "\n")
                # print("The PGA is ", pga, "\n")

                if pga > 1.70 and (1 <= RMR <= 40) and (1 <= OD <= 4000):
                    print("Damage Class: Extremely High and Probable
Damages : (Lining = A11)  ")
                    damage_counts["Extremely High"] += 1

                elif (pga > 1.70) and (41 <= RMR <= 60) and (1 <= OD <=
4000):
                    print(
                        "Damage Class: Very High and Probable Damages :
(Lining = B11), (Portal = A21), (Invert = A31) ")
                    damage_counts["Very High"] += 1

                elif (pga > 1.70) and (61 <= RMR <= 80) and (1 <= OD <=
4000):
                    print(
                        "Damage Class: High and Probable Damages : (Lining
= B12, C11, E11), (Portal = B21), (Invert = A32) ")
                    damage_counts["High"] += 1

                elif (pga > 1.70) and (81 <= RMR <= 100) and (1 <= OD <=
4000):
                    print(
                        "Damage Class: Moderate and Probable Damages :
(Lining = C12, D11, E12), (Portal = B22), (Invert = A33) ")
                    damage_counts["Moderate"] += 1
```

```
                elif (1.11 <= pga <= 1.70) and (1 <= RMR <= 40) and (1 <=
OD <= 4000):
                    print(
                        "Damage Class: Very High and Probable Damages :
(Lining = B11), (Portal = A21), (Invert = A31) ")
                    damage_counts["Very High"] += 1

                elif (1.11 <= pga <= 1.70) and (41 <= RMR <= 60) and (1 <=
OD <= 4000):
                    print(
                        "Damage Class: High and Probable Damages : (Lining
= B12, C11, E11), (Portal = B21), (Invert = A32) ")
                    damage_counts["High"] += 1

                elif (1.11 <= pga <= 1.70) and (61 <= RMR <= 80) and (1 <=
OD <= 4000):
                    print(
                        "Damage Class: Moderate and Probable Damages :
(Lining = C12, D11, E12), (Portal = B22), (Invert = A33) ")
                    damage_counts["Moderate"] += 1

                elif (1.11 <= pga <= 1.70) and (81 <= RMR <= 100) and (1 <=
OD <= 4000):
                    print(
                        "Damage Class: Low and Probable Damages : (Lining =
E13), (Portal = B23), (Invert = N) ")
                    damage_counts["Low"] += 1



                elif (0.91 <= pga <= 1.10) and (1 <= RMR <= 60) and (1 <=
OD <= 800):
                    print(
                        "Damage Class: High and Probable Damages : (Lining
= B12, C11, E11), (Portal = B21), (Invert = A32) ")
                    damage_counts["High"] += 1

                elif (0.91 <= pga <= 1.10) and (1 <= RMR <= 60) and (801 <=
OD <= 4000):
                    print(
                        "Damage Class: Low and Probable Damages : (Lining =
E13), (Portal = B23), (Invert = N) ")
                    damage_counts["Low"] += 1

                elif (0.91 <= pga <= 1.10) and (61 <= RMR <= 80) and (1 <=
OD <= 200):
                    print(
                        "Damage Class: Moderate and Probable Damages :
(Lining = C12, D11, E12), (Portal = B22), (Invert = A33) ")
                    damage_counts["Moderate"] += 1

                elif (0.91 <= pga <= 1.10) and (61 <= RMR <= 80) and (201
<= OD <= 800):
                    print(
                        "Damage Class: Low and Probable Damages : (Lining =
E13), (Portal = B23), (Invert = N) ")
                    damage_counts["Low"] += 1

                elif (0.91 <= pga <= 1.10) and (61 <= RMR <= 80) and (801
<= OD <= 4000):
```

```
                    print(
                        "Damage Class: Very Low and Probable Damages :
(Lining = E14), (Portal = B24), (Invert = N) ")
                    damage_counts["Very Low"] += 1

                elif (0.91 <= pga <= 1.10) and (81 <= RMR <= 100) and (1 <=
OD <= 4000):
                    print(
                        "Damage Class: Very Low and Probable Damages :
(Lining = E14), (Portal = B24), (Invert = N) ")
                    damage_counts["Very Low"] += 1



                elif (0.41 <= pga <= 0.90) and (1 <= RMR <= 60) and (1 <=
OD <= 500):
                    print(
                        "Damage Class: Moderate and Probable Damages :
(Lining = C12, D11, E12), (Portal = B22), (Invert = A33) ")
                    damage_counts["Moderate"] += 1

                elif (0.41 <= pga <= 0.90) and (1 <= RMR <= 60) and (501 <=
OD <= 4000):
                    print(
                        "Damage Class: Low and Probable Damages : (Lining =
E13), (Portal = B23), (Invert = N) ")
                    damage_counts["Low"] += 1

                elif (0.41 <= pga <= 0.90) and (61 <= RMR <= 100) and (1 <=
OD <= 250):
                    print(
                        "Damage Class: Low and Probable Damages : (Lining =
E13), (Portal = B23), (Invert = N) ")
                    damage_counts["Low"] += 1

                elif (0.41 <= pga <= 0.90) and (61 <= RMR <= 100) and (251
<= OD <= 4000):
                    print(
                        "Damage Class: Very Low and Probable Damages :
(Lining = E14), (Portal = B24), (Invert = N) ")
                    damage_counts["Very Low"] += 1



                elif (0.11 <= pga <= 0.40) and (1 <= RMR <= 80) and (1 <=
OD <= 400):
                    print("Damage Class: Low and Probable Damages : (Lining
= E13), (Portal = B23), (Invert = N) ")
                    damage_counts["Low"] += 1

                elif (0.11 <= pga <= 0.40) and (81 <= RMR <= 100) and (1 <=
OD <= 400):
                    print("Damage Class: Very Low and Probable Damages :
(Lining = E14), (Portal = B24), (Invert = N) ")
                    damage_counts["Very Low"] += 1

                elif (0.11 <= pga <= 0.40) and (1 <= RMR <= 100) and (401
<= OD <= 4000):
                    print("Damage Class: Very Low and Probable Damages :
(Lining = E14), (Portal = B24), (Invert = N) ")
                    damage_counts["Very Low"] += 1
```

```
                elif (0.01 <= pga <= 0.10) and (1 <= RMR <= 100) and (1 <=
OD <= 800):
                    print("Damage Class: Very Low and Probable Damages :
(Lining = E14), (Portal = B24), (Invert = N) ")
                    damage_counts["Very Low"] += 1

                elif (0.01 <= pga <= 0.10) and (1 <= RMR <= 100) and (801
<= OD <= 4000):
                    print("Damage Class: Extremely Low and Probable Damages
: N ")
                    damage_counts["Extremely Low"] += 1



                elif pga < 0.01 and (1 <= RMR <= 100) and (1 <= OD <=
4000):
                    print("Damage Class: Extremely Low and Probable Damages
: N ")
                    damage_counts["Extremely Low"] += 1


            messagebox.showinfo("Success",
                                "The Predicted Seismic Damages of Tunnel at
that Site are Successfully Executed and saved in {0}".format(
                                    filename))
            print(
"***************************************************************************
***************************************************************************
******************\n")
            # Print the counts of each damage class
            print("\nCount of each Damage Class:")
            for damage_class, count in damage_counts.items():
                print(f"{damage_class}: {count}")

            # Create a bar graph
            damage_classes = list(damage_counts.keys())
            counts = list(damage_counts.values())

            plt.figure(figsize=(9, 7))
            bars = plt.bar(damage_classes, counts, color='darkblue')

            # Annotate each bar with its count
            for bar, count in zip(bars, counts):
                plt.text(bar.get_x() + bar.get_width() / 2,
bar.get_height() + 0.05, count,
                         ha='center', va='bottom', fontsize=16)

            plt.gca().spines['right'].set_visible(False)
            plt.gca().spines['top'].set_visible(False)
            total_faults = sum(counts)
            plt.legend([f'Total faults: {total_faults}'], loc='center',
bbox_to_anchor=(1.25, 1))

            plt.xlabel('Damage Class', fontname='Times New Roman',
fontsize=14, fontweight='bold')
            plt.ylabel('Fault Count', fontname='Times New Roman',
fontsize=14, fontweight='bold')
```

```
            #plt.title('Damage Class Counts')
            plt.xticks(fontname='Times New Roman', rotation=45)
            plt.yticks(fontname='Times New Roman')
            plt.tight_layout()

            # Save the bar graph as an image
            graph_filename = f"Tunnel_Report_Graph_{formatted_time}.png"
            plt.savefig(graph_filename)

            # Print the filename of the graph alongside the text report
            print("Bar graph saved as:", graph_filename)

            print(
"***************************************************************************
***************************************************************************
******************\n")

            text_to_print2 = """ The Seismic Damages of Rock Tunnels are
Classified as Damages to Lining, Portal, and Invert.
                The sub-categories of those damages and the amount of
damage are elaborated as follows:

                Lining Damage (1):

                    Collapse of a Tunnel/Sheared-off Lining (A11)- Tunnels
intersecting the faults are sheared off or dislocated up to 3-4 m
vertically and horizontally.
                         Major sections of lining or roof cave in and
completely fall and collapse. It leads to the collapse of a tunnel

                    Grade 1 -Lining Dislocation (B11)- The lining is
sheared off or dislocated up to 2 m vertically and horizontally.
                        A section of lining or roof caves in and completely
falls and collapses.
                        Will either lead to collapse or heavy
rehabilitation of the tunnel.

                    Grade 2 -Lining Dislocation (B12)-The lining is
dislocated up to 0.8 m horizontally and slightly shifted vertically.
                        A portion of the roof caves in and falls. This will
lead to heavy rehabilitation of the tunnel.

                    Grade 1 -Lining Spalling (C11)- A large-scale spalling
characterized by the detachment of lining or steel reinforcements, severe
falling of lining,
                            exposed Reinforcement, distortion, and
surrounding rock collapse. The area of crushed lining is >2 m2(square
meter). Requires heavy rehabilitation of tunnel.

                    Grade 2- Lining Spalling (C12)- A small-scale spalling
characterized by the detachment of lining or steel reinforcements, moderate
to slight falling of lining,
                        and exposed reinforcement. The area of crushed
lining is <2 m2(square meter). Requires moderate rehabilitation of the
tunnel.

                    Lining Deformation (D11)- The lining is deformed up to
a maximum of 25 cm horizontally. The crown and sidewall of the tunnel are
deformed or
                        laterally shifted sideways from the original
```

position. Requires moderate rehabilitation of the tunnel.

Grade-1 Lining cracks (E11)- Cracks extend up to l >15 m, w >35 mm (l-Length of Crack, w- Width of Crack). Requires just a slight rehabilitation of the tunnel.

Grade-2 Lining cracks (E12)- Cracks extend up to l: 5-15 m, w: 5-35 mm (l-Length of Crack, w- Width of Crack). Requires just a slight rehabilitation of the tunnel.

Grade-3 Lining cracks (E13)- Cracks extend up to l <5 m, w <5 mm (l-Length of Crack, w- Width of Crack). Requires just a slight rehabilitation of the tunnel.

Grade-4 Lining cracks (E14)- Cracks extend up to l <1 m, w <3 mm (l- Length of Crack, w- Width of Crack). Requires just a slight rehabilitation of the tunnel.

Portal Damage(2):

Grade-1 Portal/ Slope Damage (A21)- Characterized by portal/slope collapse, head wall fractured by heavy rockfalls, and completely buried tunnel portal.

This will lead to either collapse or heavy rehabilitation of the tunnel.

Grade-2 Portal Damage (B21)- Characterized by severe failure of a slope and rock fall, most of the tunnel portal is buried, and portal wall cracking.

Severe damage to the portal such as loosening of the curved head-wall of the portal, and structure puncture.

Requires heavy rehabilitation of the tunnel.

Grade-3 Portal Damage (B22)– Characterized by moderate rock fall, massive gravel, or large stones piled up/overlaying of rock and soil deposits in front of a portal.

Portal cracks of continuous ring-shaped (Width of Crack >15 mm). Requires moderate rehabilitation of the tunnel.

Grade-4 Portal Damage (B23)- Characterized by small-scale rock fall, sparse gravel, or small stones piled up in front of a tunnel,

and portal cracks (Width of Crack < 15 mm). Requires just a slight rehabilitation of the tunnel.

Grade-5 Portal Damage (B24)- Characterized by overhead raveling of loose rock that will fall on a tunnel or slight chipping of rock chunks from a slope or

slight cracks in the portal. Requires slight rehabilitation of the tunnel.

Invert Damage(3):

Grade 1-Invert Damage (A31)- Characterized by severe invert upheaval (uplift height-hu < 150 cm), invert will dislocate laterally.

Severe extended cracks (l ≥ 20 m, w ≥ 10 cm where, l -Length of Crack, w- Width of Crack) at the tunnel bottom.

This will lead to either collapse or heavy reinforcement of the tunnel.

Grade 2-Invert Damage (A32)- Characterized by moderate invert upheaval (uplift height- hu < 50 cm) and

moderate-scale cracks (l < 20 m, w < 10 cm, where,

```
l -Length of Crack, w- Width of Crack) at the tunnel bottom.
                        Requires heavy rehabilitation of the tunnel.

                    Grade 3-Invert Damage (A33)- Characterized by slight
invert upheaval (uplift height- hu < 20 cm) or
                        slight cracks (l < 5 m, w < 5 cm, where, l -Length
of Crack, w- Width of Crack) at the tunnel bottom.
                        It requires moderate rehabilitation of the tunnel.

                N- Indicates No Damage Scenario, where the damage doesn't
take place


                The Accessibility of Tunnel for respective damage classes
are:
                AT1- Cannot be operable after the seismic event: for EH and
VH damage classes
                AT2- Operable after a seismic event with many precautions
and caution boards: for H damage class
                AT3- Operable after the seismic event without regulations
and caution boards: for M damage class
                AT4- A tunnel is immediately operable after a seismic
event:  for L and VL damage classes
                AT5 - The tunnel is operable and remains undamaged by
seismic eventS: for EL damage class """
            # print(text_to_print2)
            # Add the content of text_to_print2
            f.write("\n" + "=" * 80 + "\n")
            f.write(text_to_print2 + "\n")

        # Open the text file using the default text editor
        open_file_with_default_editor(filename)

        # Open the PNG file using the default image viewer
        subprocess.Popen(["start", graph_filename], shell=True)

def compute_tasks_dsha(active_sheet, location1, pga_list):
    global compute_tasks_counter_dsha
    if compute_tasks_counter_dsha == 0:
        text_to_print = "The Deterministic Seismic Hazard Analysis (DSHA)
of the site is as follows:"
        # print(
        #
'***************************************************************************
***************************************************************************
************************')
        print(text_to_print)
        print(

'***************************************************************************
***************************************************************************
*******************')
        print(
            "The Deterministic Seismic Hazard Analysis (DSHA) report of the
site for a given lat, long is in the form of \n "
            "Results: {Fault ID, Fault Name, Fault Layer, Source-Site
Distance (Lat, Long), Total Fault Length, Predicted Magnitude, (The PGA is
' ') }\n")
        compute_tasks_counter_dsha += 1  # Update the counter

    # start_time = time.time()
```

```
    seismic_faults = {}
    fault_ct = 0
    threshold_dist = 250
    current_site = 'x'
    row_no = 1

    for index, row in active_sheet.iterrows():
        location2 = (row.iloc[14], row.iloc[15]) # Assuming latitude is in
column 14 and longitude is in column 15

        if row_no == 1:
            row_no += 1
        elif row_no == 2:
            if current_site != row.iloc[1]:  # Assuming 'Fault ID' is in
column 1
                previous_least_dist = threshold_dist
                current_site = int(row.iloc[1])  # Convert 'Fault ID' to
integer if necessary
                dist_value = round(distance.distance(location1,
location2).km, 2)
                if dist_value < threshold_dist and dist_value <
previous_least_dist:
                    previous_least_dist = dist_value
                    seismic_faults[current_site] = [row.iloc[1],
row.iloc[2], row.iloc[7], dist_value, location2, row.iloc[17],
row.iloc[18]]
            else:
                dist_value = round(distance.distance(location1,
location2).km, 2)  # Recalculate distance
                if dist_value < threshold_dist and dist_value <
previous_least_dist:
                    previous_least_dist = dist_value
                    seismic_faults[current_site] = [row.iloc[1],
row.iloc[2], row.iloc[7], dist_value, location2, row.iloc[17],
row.iloc[18]]

    for key, value in seismic_faults.items():
        log_pga = 0.56 * value[6] - 0.0031 * value[3] - math.log10(
            value[3] + 0.0055 * 10 ** (0.5 * value[6])) + 0.26 + 0.37
        PGA = round((10 ** log_pga) * 0.00102, 2)
        pga_list.append((PGA, key, value))  # Store PGA, key, and value in
the list


def retrieve_dsha():
    # Get the current date and time
    current_time = datetime.datetime.now()

    # Format the date and time to include in the filename
    formatted_time = current_time.strftime("%Y-%m-%d_%H-%M-%S")

    # Create the filename with the formatted date and time
    filename = f"DSHA_Report_{formatted_time}.txt"

    with open(filename, "w+") as f:
        sys.stdout = f
        rmr = 1
        od = 1

        # print("Submitted Data")
        # print("The Generated Report is as follows:")
```

```
        latitude = float(textbox1.get("1.0", "end-1c"))
        longitude = float(textbox2.get("1.0", "end-1c"))

        print("latitude is ", latitude)
        print("longitude is ", longitude)

        if rmr == 1 and od == 1:
            location1 = (latitude, longitude)

            active_sheet1 = read_excel("Active_faults_Order.xlsx")
            active_sheet2 = read_excel("Fault_Tectonic_order.xlsx")
            active_sheet3 = read_excel("Thrust_Tectonic_order.xlsx")
            active_sheet4 = read_excel("Himatibet_faults_Order.xlsx")   #
Add this line
            active_sheet5 = read_excel("Iran_faults_ order.xlsx")

            pga_list = []  # Initialize the list to store PGA values

            compute_tasks_dsha(active_sheet1, location1, pga_list)
            compute_tasks_dsha(active_sheet2, location1, pga_list)
            compute_tasks_dsha(active_sheet3, location1, pga_list)
            compute_tasks_dsha(active_sheet4, location1, pga_list)
            compute_tasks_dsha(active_sheet5, location1, pga_list)

            # Sort the list based on PGA values from highest to lowest
            sorted_pga_list = sorted(pga_list, key=lambda x: x[0],
reverse=True)

            # Initialize counters for each damage class
            PGA_Range_counts = {
                ">1.50g": 0,
                "1.01 to 1.50g": 0,
                "0.76 to 1.00g": 0,
                "0.51 to 0.75g": 0,
                "0.26 to 0.50g": 0,
                "0.11 to 0.25g": 0,
                "0.01 to 0.10g": 0,
                "<0.01g": 0
            }
            # Print the sorted results
            for pga, key, value in sorted_pga_list:
                print(

"**************************************************************************
***************************************************************************
******************\n")
                print(f"Results: {value}")
                print("The PGA is ", round(pga, 2), "\n")
                # print("The PGA is ", pga, "\n")
                if pga > 1.50:
                    PGA_Range_counts [">1.50g"] += 1

                elif (1.01 <= pga <= 1.50) :
                    PGA_Range_counts ["1.01 to 1.50g"] += 1

                elif (0.76 <= pga <= 1.00) :
                    PGA_Range_counts ["0.76 to 1.00g"] += 1

                elif (0.51 <= pga <= 0.75) :
                    PGA_Range_counts ["0.51 to 0.75g"] += 1
```

```
                elif (0.26 <= pga <= 0.50) :
                    PGA_Range_counts ["0.26 to 0.50g"] += 1

                elif (0.11 <= pga <= 0.25) :
                    PGA_Range_counts ["0.11 to 0.25g"] += 1

                elif (0.01 <= pga <= 0.10) :
                    PGA_Range_counts ["0.01 to 0.10g"] += 1

                elif ( pga < 0.01):
                    PGA_Range_counts ["<0.01g"] += 1

            messagebox.showinfo("Success",
                                "The Deterministic Seismic Hazard Analysis
(DSHA) of the Site is Successfully Executed and saved in {0}".format(
                                    filename))
            print(

"***************************************************************************
***************************************************************************
******************\n")
            # Print the counts of each damage class
            print("\nCount of each PGA Class:")
            for PGA_class in PGA_Range_counts :
                print(f"{PGA_class}: {PGA_Range_counts [PGA_class]}")

            # Create a bar graph
            PGA_classes = list(PGA_Range_counts.keys())
            counts = list(PGA_Range_counts.values())

            plt.figure(figsize=(9, 7))
            bars1 = plt.bar(PGA_classes, counts, color='darkblue')

            # Annotate each bar with its count
            for bar, count in zip(bars1, counts):
                plt.text(bar.get_x() + bar.get_width() / 2,
bar.get_height() + 0.05, count,
                         ha='center', va='bottom', fontsize=16)

            plt.gca().spines['right'].set_visible(False)
            plt.gca().spines['top'].set_visible(False)
            total_faults = sum(counts)
            plt.legend([f'Total faults: {total_faults}'], loc='upper
right', bbox_to_anchor=(1.25, 1))

            plt.xlabel('PGA Ranges (in g)', fontname='Times New Roman',
fontsize=14, fontweight='bold')
            plt.ylabel('Fault Count', fontname='Times New Roman',
fontsize=14, fontweight='bold')
            #plt.title('Damage Class Counts')
            plt.xticks(fontname='Times New Roman', rotation=45)
            plt.yticks(fontname='Times New Roman')
            plt.tight_layout()

            # Save the bar graph as an image
            graph_filename = f"DSHA_Report_Graph_{formatted_time}.png"
            plt.savefig(graph_filename)

            # Print the filename of the graph alongside the text report
            print("Bar graph saved as:", graph_filename)
```

```
            # Open the text file using the default text editor
            open_file_with_default_editor(filename)

            # Open the PNG file using the default image viewer
            subprocess.Popen(["start", graph_filename], shell=True)


original_stdout = sys.stdout

import tkinter as tk

def toggle_parameters():
    if report_var.get() == "Tunnel Vulnerability Report":
        textbox3.config(state=NORMAL)  # Enable RMR
        textbox4.config(state=NORMAL)  # Enable Overburden
        for rb in lining_radios:
            rb.config(state=NORMAL)  # Enable Lining Radios
        for rb in shape_radios:
            rb.config(state=NORMAL)  # Enable Shape Radios
        a3.config(fg="black")  # Set RMR label color to black
        a4.config(fg="black")  # Set Overburden label color to black
        a5.config(fg="black")  # Set Lining label color to black
        a6.config(fg="black")  # Set Shape label color to black
        submitbutton.config(state=NORMAL, fg="green")  # Enable and set
color for Generate Report button
        submitbutton_dsha.config(state=DISABLED, fg="grey")  # Disable and
set color for Generate DSHA Report button
    else:
        textbox3.config(state=DISABLED)  # Disable RMR
        textbox4.config(state=DISABLED)  # Disable Overburden
        for rb in lining_radios:
            rb.config(state=DISABLED)  # Disable Lining Radios
        for rb in shape_radios:
            rb.config(state=DISABLED)  # Disable Shape Radios
        a3.config(fg="grey")  # Set RMR label color to grey
        a4.config(fg="grey")  # Set Overburden label color to grey
        a5.config(fg="grey")  # Set Lining label color to grey
        a6.config(fg="grey")  # Set Shape label color to grey
        submitbutton.config(state=DISABLED, fg="grey")  # Disable and set
color for Generate Report button
        submitbutton_dsha.config(state=NORMAL, fg="green")  # Enable and
set color for Generate DSHA Report button


def toggle_generate_button():
    if report_var.get() == "Tunnel Vulnerability Report":
        submitbutton_dsha.config(state=NORMAL, fg="green")  # Enable and
set color for Generate DSHA Report button
    else:
        submitbutton.config(state=NORMAL, fg="green")  # Enable and set
color for Generate Report button


def show_information():
    information_text = (
        "This tool serves two main purposes:\n\n"
        "1. Seismic Vulnerability of Rock Tunnels (SVRT):\n"
        "- Evaluate the SVRT using Reddy and Singh's approach.\n"
        "- Predicts the Damage class and probable damages to tunnel before
an Earthquake.\n"
        "- Assumes tunnels are not designed for seismicity.\n"
```

```
        "- Can perform for locations in India and adjacent countries.\n"
        "- Analysis performed within a threshold distance of 250 km.\n"
        "- PGA is calculated using the attenuation relationship of Kanno et
al. (2006) .\n\n"
        "2. Deterministic Seismic Hazard Analysis (DSHA):\n"
        "- Assess the deterministic seismic hazard of sites.\n"
        "- Can perform for locations in India and adjacent countries.\n"
        "- Analysis performed within a threshold distance of 250 km.\n"
        "- PGA is calculated using the attenuation relationship of Kanno et
al. (2006).\n\n"
        "You can generate reports and bar graphs for SVRT or DSHA as
needed."
    )
    messagebox.showinfo("Tool Information", information_text)

import tkinter as tk

def close_window():
    root.destroy()

def on_closing():
    pass
root = tk.Tk()
root.title("Seismic Vulnerability of Rock Tunnels in India and Adjacent
Countries")
root.geometry("1250x620")
root.resizable(width=False, height=False)


bg_image2 = tk.PhotoImage(file=r"IIT_Roorkee_Logo_11.png")
bg_label2 = tk.Label(root, image=bg_image2)
bg_label2.place(relx=1, rely=1, anchor="se")

# Radio buttons for report selection
report_var = StringVar()
report_var.set("Tunnel Vulnerability Report")  # Default selection
report_radios = [
    Radiobutton(root, text="Deterministic Seismic Hazard Analysis (DSHA)",
variable=report_var, value="DSHA Report",
                command=toggle_parameters, font=("calibri", 18, "bold")),
    Radiobutton(root, text=" Seismic Vulnerability of Rock Tunnel (SVRT)",
variable=report_var, value="Tunnel Vulnerability Report",
                command=toggle_parameters, font=("calibri", 18, "bold"))
]

for idx, radio in enumerate(report_radios):
    radio.place(x=20 + idx * 500, y=5)

a1 = Label(text="Latitude", font="calibri 16 bold")
a1.place(x=20, y=60)
textbox1 = Text(height=2, width=30)
textbox1.place(x=180, y=60)

a2 = Label(text="Longitude", font="calibri 16 bold")
a2.place(x=20, y=110)
textbox2 = Text(height=2, width=30)
textbox2.place(x=180, y=110)

a3 = Label(text="Rock Mass Rating(RMR)", font="calibri 16 bold")
a3.place(x=520, y=60)
textbox3 = Text(height=2, width=30)
```

```
textbox3.place(x=780, y=60)

a4 = Label(text="Overburden (in m)", font="calibri 16 bold")
a4.place(x=520, y=110)
textbox4 = Text(height=2, width=30)
textbox4.place(x=780, y=110)

# Radio buttons for Lining options
a5 = Label(text="Lining Type", font="calibri 16 bold")
a5.place(x=522, y=160)

lining_var = StringVar()
lining_options = ["UL", "TL", "ML", "CL", "RCL", "Others"]
lining_radios = []

x_coordinate = 770
y_coordinate = 170

for option in lining_options:
    rb = Radiobutton(root, text=option, variable=lining_var, value=option)
    rb.place(x=x_coordinate, y=y_coordinate)
    lining_radios.append(rb)
    x_coordinate += 50

# Radio buttons for Shape options
a6 = Label(text="Shape", font="calibri 16 bold")
a6.place(x=522, y=210)

shape_var = StringVar()
shape_options = ["HS", "CR", "RE", "OV", "D","AH", "Others"]
shape_radios = []

x_coordinate = 770
y_coordinate = 220

for option in shape_options:
    rb = Radiobutton(root, text=option, variable=shape_var, value=option)
    rb.place(x=x_coordinate, y=y_coordinate)
    shape_radios.append(rb)
    x_coordinate += 50

custom_font = ("Times New Roman bold", 11)
foreground_color = "black"
background_color = "white"

# Create and configure labels with the specified text, font, and colors
a77 = Label(root, text="Note:", font=custom_font, fg=foreground_color,
bg=background_color)
a77.place(x=15, y=172)

a72 = Label(root, text="Please enter Latitude and Longitude in Decimal
Degrees format only", font=custom_font,
            fg=foreground_color, bg=background_color)
a72.place(x=15, y=192)

a72 = Label(root, text="For example: 29.99219444, 78.82808888 is in Decimal
Degrees format.", font=custom_font,
            fg=foreground_color, bg=background_color)
a72.place(x=15, y=212)

a74 = Label(root, text="29°59'31.9\"N, 78°49'41.1\"E is in degrees,
```

```
minutes, and seconds format ", font=custom_font,
            fg=foreground_color, bg=background_color)
a74.place(x=15, y=232)

a771 = Label(root, text="Note:", font=custom_font, fg=foreground_color,
bg=background_color)
a771.place(x=502, y=272)

a7 = Label(root, text="For Lining Type: UL- Unlined; TL- Timber Lined",
font=custom_font, fg=foreground_color,
           bg=background_color)
a7.place(x=502, y=292)

a8 = Label(root, text="ML- Masonry Lined (Brick/Brick+Stone/Brick+Plain
concrete)", font=custom_font,
           fg=foreground_color, bg=background_color)
a8.place(x=502, y=312)

a9 = Label(root, text="CL- Concrete/Reinforced Concrete only with primary
lining as major support ", font=custom_font,
           fg=foreground_color, bg=background_color)
a9.place(x=502, y=332)

a91 = Label(root, text="RCL- Reinforced Concrete with primary+secondary
lining+support system", font=custom_font,
            fg=foreground_color, bg=background_color)
a91.place(x=502, y=352)

a71 = Label(root, text="Note: ", font=custom_font, fg=foreground_color,
bg=background_color)
a71.place(x=1022, y=260)

a81 = Label(root, text="For Shape of Tunnel:", font=custom_font,
fg=foreground_color, bg=background_color)
a81.place(x=1022, y=280)

a811 = Label(root, text="HS-Horseshoe, RE- Rectangle", font=custom_font,
fg=foreground_color, bg=background_color)
a811.place(x=1022, y=300)

a83 = Label(root, text="D- D shaped, OV- Ovoid", font=custom_font,
fg=foreground_color, bg=background_color)
a83.place(x=1022, y=320)

a841 = Label(root, text="CR- Circular", font=custom_font,
fg=foreground_color, bg=background_color)
a841.place(x=1022, y=340)

a841 = Label(root, text="AH- Arched with Horizontal roof",
font=custom_font, fg=foreground_color, bg=background_color)
a841.place(x=1022, y=360)

a29 = Label(root, text="Developed by: A. Dinesh Reddy, Dr.Aditya Singh ",
bg="black", fg="white",
            font="Times 16 bold italic")
a29.place(x=10, y=550)

a29 = Label(root, text="Department of Civil Engineering, IIT Roorkee ",
bg="black", fg="white",
            font="Times 16 bold italic")
a29.place(x=10, y=580)
```

```
# Configure labels, fonts, and colors as before...
# Default label colors are set to black
a3.config(fg="black")
a4.config(fg="black")
a5.config(fg="black")
a6.config(fg="black")

submitbutton = Button(text="Generate SVRT Report", height=1, width=20,
bg="black", fg="green",
                      font="comicsansms 18 bold",
                      borderwidth=8, command=lambda: retrieve_tunnel())
submitbutton.place(x=700, y=395)

submitbutton_dsha = Button(text="Generate DSHA Report", height=1, width=20,
bg="black", fg="green",
                           font="comicsansms 18 bold", borderwidth=8,
command=lambda: retrieve_dsha())
submitbutton_dsha.place(x=100, y=395)

info_button = Button(text="The Tool Info", command=show_information)
info_button.place(x=1120, y=10)

buttonclose = Button(root, text="Close",bg="black", fg="green",font="Times
16 bold", command=close_window)
buttonclose.pack(side = "bottom") # to close application
root.protocol("WM_DELETE_WINDOW", on_closing)
mainloop()
```