```
import numpy as np
import tkinter as tk
from tkinter import ttk
import matplotlib.pyplot as plt
from tkinter import messagebox

def compute_shear_rates(Q, Lp, Sp, L, Rp, Phi, k, Sor, Rw):
    mu = 0.001  # viscosity in Pa.s (for water)

    Ap = np.pi * Rp ** 2
    qp = Q / (Lp * Sp)
    shear_rate_perforation = 4 * qp / Ap

    dp_dr = (8 * mu * qp) / (np.pi * k * Rw ** 4)  # pressure
gradient
    distances = np.linspace(0.01, 100, 200)  # starting from 0.01 to
avoid division by zero
    shear_away = np.abs(-k * dp_dr / distances)  # Taking absolute
value

    return shear_rate_perforation, distances, shear_away

class ShearRateCalculator(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Shear Rate Calculator")
        self.geometry("800x600")
        self.create_widgets()

    def create_widgets(self):
        # Entry fields
        labels = ["Injection rate into the well (m3/day):",
"Wellbore radius (inch):", "Perf zone thickness (ft):",
                  "Perforation shot density (SPF):", "Perf tunnel
length (inch):",
                  "Perf radius (inch):", "Porosity:", "K (md):",
"Sor:"]
        self.entries = {}
        self.conversion_labels = {}
        for i, label in enumerate(labels):
            ttk.Label(self, text=label).grid(row=i*2, column=0,
padx=5, pady=5, sticky="e")
            entry_var = tk.StringVar()
            entry = ttk.Entry(self, textvariable=entry_var)
            entry.grid(row=i*2, column=1, padx=5, pady=5)
            self.entries[label] = entry

            # Add conversion labels and traces for live update
            conversion_label = ttk.Label(self, text="")
            conversion_label.grid(row=i*2 + 1, column=1, sticky="w")
            self.conversion_labels[label] = conversion_label
            entry_var.trace("w", lambda *args, label=label,
var=entry_var: self.update_conversion(label, var))
```

```
        # Submit button
        self.submit_button = ttk.Button(self, text="Calculate",
command=self.on_submit)
        self.submit_button.grid(row=len(labels)*2, column=0,
columnspan=2, pady=20)

        # Display shear rate in one perforation label
        self.shear_rate_label = ttk.Label(self, text="")
        self.shear_rate_label.grid(row=len(labels)*2 + 1, column=0,
columnspan=2)

    def update_conversion(self, label, var):
        val = var.get()
        try:
            if label == "Injection rate into the well (m3/day):" and
val:
                bbl_val = float(val) * 6.28981

self.conversion_labels[label].config(text=f"Equivalent to
{bbl_val:.2f} bbl/day")
            elif label == "Perforation shot density (SPF):" and val:
                per_meter_val = float(val) * 3.281

self.conversion_labels[label].config(text=f"Equivalent to
{per_meter_val:.2f} shots/m")
            elif label == "Perf tunnel length (inch):" and val:
                cm_val = float(val) * 2.54

self.conversion_labels[label].config(text=f"Equivalent to
{cm_val:.2f} cm")
            elif label == "Perf radius (inch):" and val:
                cm_val = float(val) * 2.54

self.conversion_labels[label].config(text=f"Equivalent to
{cm_val:.2f} cm")
            elif label == "Wellbore radius (inch):" and val:
                cm_val = float(val) * 2.54

self.conversion_labels[label].config(text=f"Equivalent to
{cm_val:.2f} cm")
            elif label == "Perf zone thickness (ft):" and val:
                meter_val = float(val) * 0.3048

self.conversion_labels[label].config(text=f"Equivalent to
{meter_val:.2f} meters")
        except ValueError:
            # Handle non-numeric input
            self.conversion_labels[label].config(text="Invalid
input")

    def on_submit(self):
        try:
        # Retrieve values from the entry widgets and convert units
where necessary
```

```
            Q = float(self.entries["Injection rate into the well
(m3/day):"].get())
            Lp = float(self.entries["Perf zone thickness
(ft):"].get()) * 0.3048  # feet to meters
            Sp = float(self.entries["Perforation shot density
(SPF):"].get())
            L = float(self.entries["Perf tunnel length
(inch):"].get()) * 0.0254  # inches to meters
            Rp = float(self.entries["Perf radius (inch):"].get()) *
0.0254  # inches to meters
            Phi = float(self.entries["Porosity:"].get())
            k = float(self.entries["K (md):"].get()) * 9.869233e-13
# conversion to m^2
            Sor = float(self.entries["Sor:"].get())
            Rw = float(self.entries["Wellbore radius
(inch):"].get()) * 0.0254  # inches to meters

        # Plot shear rate in perforations for varying injection
rates
            injection_rates = np.linspace(Q, Q*10, 10)
            shear_rates_perforations = [compute_shear_rates(q, Lp,
Sp, L, Rp, Phi, k, Sor, Rw)[0] for q in injection_rates]

        except ValueError:
            # This will handle conversion issues like if a user
enters text instead of a number
            tk.messagebox.showerror("Invalid Input", "Please enter
valid numbers in all fields.")
        except Exception as e:
            # A general catch for other potential issues
            tk.messagebox.showerror("Error", str(e))

        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

        ax1.plot(injection_rates, shear_rates_perforations, 'o-',
color='blue', label='Shear Rate in Perforations')
        ax1.set_yscale('log')
        ax1.set_title("Shear Rate in Perforations vs Injection
Rate")
        ax1.set_xlabel("Injection Rate (m3/day)")
        ax1.set_ylabel("Shear Rate (s^-1)")
        ax1.grid(True, which='both', linestyle='--', linewidth=0.5)
        ax1.legend(loc='best')

        # Shear rate away from wellbore for the given injection rate
        shear_rate_perforation, distances, shear_away =
compute_shear_rates(Q, Lp, Sp, L, Rp, Phi, k, Sor, Rw)
        self.shear_rate_label.config(text=f"Shear rate in one
perforation: {shear_rate_perforation:.2f} s^-1")

        ax2.plot(distances, shear_away, 'o-', color='red',
label='Shear Rate away from Wellbore')
        if not all(val <= 0 for val in shear_away):
            ax2.set_yscale('log')
```

```
            ax2.set_title("Shear Rate vs. Distance from Wellbore")
            ax2.set_xlabel("Distance (m)")
            ax2.set_ylabel("Shear Rate (s^-1)")
            ax2.grid(True, which='both', linestyle='--',
linewidth=0.5)
            ax2.legend(loc='best')

        fig.suptitle('Shear Rate Analysis')
        plt.tight_layout()
        plt.show()

if __name__ == "__main__":
    app = ShearRateCalculator()
    app.mainloop()
def run():
    app = ShearRateCalculator()
    app.mainloop()
```