K-Mer Analysis Tool

### Code

|  |
| --- |
| import tkinter as tk  from tkinter import filedialog, messagebox, ttk  from collections import Counter  from Bio.Phylo.TreeConstruction import DistanceCalculator, DistanceTreeConstructor  from Bio import Phylo  from Bio.Align import MultipleSeqAlignment  from Bio.Seq import Seq  from Bio.SeqRecord import SeqRecord  import matplotlib.pyplot as plt  import seaborn as sns  import pandas as pd  # For data export  import numpy as np  class KmerAnalysisTool:      def \_\_init\_\_(self, root):          self.root = root          self.root.title("K-mer Analysis Tool")          self.root.geometry("900x700")          self.root.configure(bg="#e8f1f2")          # Title Label          title\_label = tk.Label(root, text="K-mer Analysis Tool", font=("Arial", 24, "bold"), bg="#007BFF", fg="white")          title\_label.pack(pady=10, fill=tk.X)          # Input Frame          input\_frame = tk.Frame(root, bg="#e8f1f2", pady=10)          input\_frame.pack(fill=tk.X, padx=20)          tk.Label(input\_frame, text="Enter DNA Sequence:", font=("Arial", 12), bg="#e8f1f2").grid(row=0, column=0, sticky="w", pady=5)          self.sequence\_entry = tk.Text(input\_frame, height=5, width=60, font=("Arial", 10), relief=tk.GROOVE, borderwidth=2)          self.sequence\_entry.grid(row=1, column=0, columnspan=2, pady=5)          upload\_button = ttk.Button(input\_frame, text="Upload FASTA File", command=self.upload\_fasta)          upload\_button.grid(row=2, column=0, pady=10, sticky="w")          tk.Label(input\_frame, text="K-mer Size:", font=("Arial", 12), bg="#e8f1f2").grid(row=3, column=0, sticky="w", pady=5)          self.kmer\_size\_entry = ttk.Entry(input\_frame, width=10)          self.kmer\_size\_entry.grid(row=3, column=1, sticky="w")          # Buttons Frame          button\_frame = tk.Frame(root, bg="#e8f1f2")          button\_frame.pack(pady=10)          ttk.Button(button\_frame, text="Perform K-mer Analysis", command=self.perform\_analysis).grid(row=0, column=0, padx=10, pady=5)          ttk.Button(button\_frame, text="Detect AMR Genes", command=self.detect\_amr\_genes).grid(row=0, column=1, padx=10, pady=5)          ttk.Button(button\_frame, text="Generate Phylogenetic Tree", command=self.generate\_phylogenetic\_tree).grid(row=0, column=2, padx=10, pady=5)          ttk.Button(button\_frame, text="Generate Heatmap", command=self.generate\_heatmap).grid(row=0, column=3, padx=10, pady=5)          ttk.Button(button\_frame, text="Export Data", command=self.export\_data).grid(row=0, column=4, padx=10, pady=5)          # Output Frame          output\_frame = tk.Frame(root, bg="#e8f1f2")          output\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)          tk.Label(output\_frame, text="Output:", font=("Arial", 12, "bold"), bg="#e8f1f2").pack(anchor="w", pady=5)          self.output\_text = tk.Text(output\_frame, height=10, font=("Arial", 10), relief=tk.GROOVE, borderwidth=2)          self.output\_text.pack(fill=tk.BOTH, expand=True)      def upload\_fasta(self):          filepath = filedialog.askopenfilename(filetypes=[("FASTA files", "\*.fasta"), ("All files", "\*.\*")])          if filepath:              with open(filepath, "r") as file:                  lines = file.readlines()                  sequence = "".join(line.strip() for line in lines if not line.startswith(">"))                  self.sequence\_entry.delete("1.0", tk.END)                  self.sequence\_entry.insert("1.0", sequence)      def perform\_analysis(self):          sequence = self.sequence\_entry.get("1.0", tk.END).strip().upper()          try:              k = int(self.kmer\_size\_entry.get())              if k <= 0 or k > len(sequence):                  raise ValueError          except ValueError:              messagebox.showerror("Error", "Invalid k-mer size.")              return          self.kmer\_counts = self.count\_kmers(sequence, k)          self.output\_text.delete("1.0", tk.END)          self.output\_text.insert("1.0", f"K-mer Counts (k={k}):\n")          for kmer, count in self.kmer\_counts.items():              self.output\_text.insert(tk.END, f"{kmer}: {count}\n")      def count\_kmers(self, sequence, k):          return Counter(sequence[i:i + k] for i in range(len(sequence) - k + 1))      def generate\_heatmap(self):          if not hasattr(self, 'kmer\_counts'):              messagebox.showerror("Error", "No k-mer data to generate heatmap.")              return          kmer\_list = list(self.kmer\_counts.keys())          frequencies = list(self.kmer\_counts.values())          heatmap\_data = np.array([frequencies])          sns.heatmap(heatmap\_data, annot=True, fmt="d", cmap="coolwarm", xticklabels=kmer\_list, yticklabels=["Frequency"])          plt.title("K-mer Frequencies")          plt.xlabel("K-mers")          plt.ylabel("Frequency")          plt.show()      def detect\_amr\_genes(self):          sequence = self.sequence\_entry.get("1.0", tk.END).strip().upper()          amr\_genes = ["bla", "tet", "erm", "sul", "aac"]          amr\_kmers = ["AACCGGTT", "TTGGCCAA"]          detected\_genes = [gene for gene in amr\_genes if gene.upper() in sequence]          detected\_kmers = [kmer for kmer in amr\_kmers if kmer.upper() in sequence]          self.output\_text.delete("1.0", tk.END)          if detected\_genes or detected\_kmers:              self.output\_text.insert(tk.END, "Detected AMR Genes and K-mers:\n")              if detected\_genes:                  self.output\_text.insert(tk.END, f"Genes: {', '.join(detected\_genes)}\n")              if detected\_kmers:                  self.output\_text.insert(tk.END, f"K-mers: {', '.join(detected\_kmers)}\n")          else:              self.output\_text.insert(tk.END, "No AMR genes or k-mers detected.\n")      def generate\_phylogenetic\_tree(self):          try:              alignment = MultipleSeqAlignment([                  SeqRecord(Seq("ATCGTACGAT"), id="Seq1"),                  SeqRecord(Seq("ATCGTTCGAT"), id="Seq2"),                  SeqRecord(Seq("ATCGTACCAT"), id="Seq3"),              ])              calculator = DistanceCalculator("identity")              distance\_matrix = calculator.get\_distance(alignment)              constructor = DistanceTreeConstructor()              tree = constructor.upgma(distance\_matrix)              Phylo.draw(tree)          except Exception as e:              messagebox.showerror("Error", f"Failed to generate phylogenetic tree: {str(e)}")      def export\_data(self):          if not hasattr(self, 'kmer\_counts') or not self.kmer\_counts:              messagebox.showerror("Error", "No analysis data to export.")              return          data = {"K-mer": list(self.kmer\_counts.keys()), "Count": list(self.kmer\_counts.values())}          df = pd.DataFrame(data)          save\_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV Files", "\*.csv"), ("All Files", "\*.\*")])          if save\_path:              df.to\_csv(save\_path, index=False)              messagebox.showinfo("Success", f"Data exported successfully to {save\_path}")  if \_\_name\_\_ == "\_\_main\_\_":      root = tk.Tk()      app = KmerAnalysisTool(root)      root.mainloop() |

**Interface**

A screenshot of a computer

Description automatically generated

**DNA sequence =** ("ATGCGTACGTAGCTAGCTAGCTAGCTGACGATCGTAGCTAGCTAGCTGACTGATCGTAGCTAGCTAG\n" "GATCGTAGCTAGCTAGCTGACGATCGTAGCTAGCTAGCTGACGATCGTAGCTAGCTAGCTGACGATC\n" "GCTAGCTAGCTAGCTGACTAGCTAGCTGATCGTAGCTAGCTAGCTGACGATCGTAGCTAGCTAGCTA\n" "GCTAGCTGACGATCGTAGCTAGCTAGCTGACTGATCGTAGCTAGCTAGCTGACGATCGTAGCTAGCT" )

## Output

A screenshot of a computer

Description automatically generated

