####### Required library #######

pip install biopython

####### Required modules #######

import re

import random

from Bio.Seq import Seq

import pandas as pd

import plotly.graph\_objects as go

from jinja2 import Template

import webbrowser

####### Prompt user to enter CDS sequence #######

def get\_cds\_sequence():

print("Enter the CDS sequence or paste below:")

cds\_input = input("Enter CDS sequence: ").strip()

if cds\_input:

return cds\_input.upper()

else:

print("No sequence provided. Exiting...")

exit()

####### Find PAM and Target sequence sequences #######

def find\_pam\_and\_Target sequence(sequence, pam\_regex):

pam\_matches = []

for match in pam\_regex.finditer(sequence):

pam\_start = match.start()

if pam\_start >= 20:

Target sequence\_seq = sequence[pam\_start - 20:pam\_start]

pam\_seq = sequence[pam\_start:pam\_start + 3]

pam\_matches.append({

"start": pam\_start - 20,

"Target sequence": Target sequence\_seq,

"pam": pam\_seq,

"gc\_content": (Target sequence\_seq.count("G") + Target sequence\_seq.count("C")) / len(Target sequence\_seq) \* 100,

"self\_complementarity": sum([1 for i in range(len(Target sequence\_seq)) if Target sequence\_seq[i] == Target sequence\_seq[-(i + 1)]]),

"efficiency": 100 - abs(50 - ((Target sequence\_seq.count("G") + Target sequence\_seq.count("C")) / len(Target sequence\_seq) \* 100))

})

return pam\_matches

####### Design primers and calculate their properties #######

def design\_primers(sequence, Target sequence\_start, Target sequence\_seq, buffer\_range=(50, 150)):

buffer = random.randint(\*buffer\_range)

left\_primer\_start = max(0, Target sequence\_start - buffer - 20)

right\_primer\_start = Target sequence\_start + len(Target sequence\_seq) + buffer

if left\_primer\_start + 20 > len(sequence) or right\_primer\_start + 20 > len(sequence):

return None, None, None, None, None

left\_primer = sequence[left\_primer\_start:left\_primer\_start + 20]

right\_primer = sequence[right\_primer\_start:right\_primer\_start + 20]

left\_tm = calculate\_tm\_advanced(left\_primer)

right\_tm = calculate\_tm\_advanced(right\_primer)

fragment\_length = calculate\_fragment\_length(left\_primer\_start, right\_primer\_start, sequence)

if fragment\_length <= 0:

return None, None, None, None, None

return left\_primer, right\_primer, left\_tm, right\_tm, fragment\_length

####### Calculate fragment length #######

def calculate\_fragment\_length(left\_start, right\_start, sequence):

if left\_start >= len(sequence) or right\_start >= len(sequence) or left\_start > right\_start:

return None

return right\_start - left\_start

####### Calculate Tm for primers #######

def calculate\_tm\_advanced(primer):

at = primer.count("A") + primer.count("T")

gc = primer.count("G") + primer.count("C")

if at + gc == 0:

return 0

return round(64.9 + 41 \* (gc - 16.4) / (at + gc), 2)

####### Create advanced single-axis interactive graph #######

def create\_single\_axis\_graph(sequence, pam\_matches):

fig = go.Figure()

for match in pam\_matches:

Target sequence\_color = f"rgba({255 - int(match['efficiency'] \* 2.55)}, {int(match['efficiency'] \* 2.55)}, 0, 1)"

# Target sequence region

fig.add\_trace(go.Scatter(

x=list(range(match["start"], match["start"] + 20)),

y=[1] \* 20,

mode='lines+markers',

name=f'Target sequence: {match["Target sequence"]}',

line=dict(color=Target sequence\_color, width=4),

hovertemplate=(

f"<b>Target sequence:</b> {match['Target sequence']}<br>"

f"<b>GC Content:</b> {match['gc\_content']:.2f}%<br>"

f"<b>Efficiency:</b> {match['efficiency']:.2f}<br>"

f"<b>Position:</b> {match['start']}<br>"

f"<b>Left Primer:</b> {match.get('left\_primer', 'N/A')}<br>"

f"<b>Right Primer:</b> {match.get('right\_primer', 'N/A')}<br>"

f"<b>Fragment Length:</b> {match.get('fragment\_length', 'N/A')}<br>"

f"<b>Left Primer Tm:</b> {match.get('left\_tm', 'N/A')} °C<br>"

f"<b>Right Primer Tm:</b> {match.get('right\_tm', 'N/A')} °C<br>"

)

))

# PAM region

fig.add\_trace(go.Scatter(

x=list(range(match["start"] + 20, match["start"] + 23)),

y=[1] \* 3,

mode='lines+markers',

name='PAM Region',

line=dict(color='red', width=3),

hovertemplate=(

f"<b>PAM:</b> {match['pam']}<br>"

f"<b>Position:</b> {match['start'] + 20}<br>"

)

))

# Layout settings

fig.update\_layout(

title="Single Axis Visualization of Target sequence, PAM, and Primer Regions",

xaxis=dict(title="Gene Position", rangeslider=dict(visible=True)),

yaxis=dict(visible=False), # Hide Y-axis for single axis

legend=dict(title="Regions"),

height=400,

hoverlabel=dict(bgcolor="white", font\_size=12, font\_family="Arial")

)

return fig

####### Generate HTML report #######

def generate\_html\_report(sequence, pam\_matches, graph\_html):

template = Template("""

<!DOCTYPE html>

<html>

<head>

<title>CRISPR Analysis Report</title>

<style>

body { font-family: Arial, sans-serif; margin: 20px; }

table { border-collapse: collapse; width: 100%; margin-top: 20px; }

th, td { border: 1px solid #ddd; text-align: left; padding: 8px; }

th { background-color: #f4f4f4; }

</style>

</head>

<body>

<h1>CRISPR Analysis Report</h1>

<h2>CDS Sequence</h2>

<div style="white-space: pre-wrap;">{{ sequence }}</div>

<h2>Target sequence, PAM, and Primer Details</h2>

<table>

<tr>

<th>Start Position</th>

<th>Target sequence Sequence</th>

<th>PAM Sequence</th>

<th>GC Content (%)</th>

<th>Efficiency</th>

<th>Left Primer</th>

<th>Right Primer</th>

<th>Fragment Length (bp)</th>

<th>Left Primer Tm</th>

<th>Right Primer Tm</th>

</tr>

{% for match in pam\_matches %}

<tr>

<td>{{ match.start }}</td>

<td>{{ match.Target sequence }}</td>

<td>{{ match.pam }}</td>

<td>{{ match.gc\_content }}</td>

<td>{{ match.efficiency }}</td>

<td>{{ match.left\_primer }}</td>

<td>{{ match.right\_primer }}</td>

<td>{{ match.fragment\_length }}</td>

<td>{{ match.left\_tm }}</td>

<td>{{ match.right\_tm }}</td>

</tr>

{% endfor %}

</table>

<h2>Interactive Graph</h2>

{{ graph\_html | safe }}

</body>

</html>

""")

return template.render(sequence=sequence, pam\_matches=pam\_matches, graph\_html=graph\_html)

####### Save results to Excel #######

def save\_to\_excel(pam\_matches, filename="crispr\_pam\_results.xlsx"):

df = pd.DataFrame(pam\_matches)

df.to\_excel(filename, index=False)

print(f"PAM and Target sequence results saved to {filename}")

####### Main execution #######

sequence = get\_cds\_sequence()

pam\_regex = re.compile(r"(?=(.GG))") # Default PAM for SpCas9: NGG

pam\_matches = find\_pam\_and\_Target sequence(sequence, pam\_regex)

# Add primer details and filter valid matches

filtered\_pam\_matches = []

for match in pam\_matches:

primers = design\_primers(sequence, match["start"], match["Target sequence"])

if primers[0] is not None: # Include only valid primer matches

match.update({

"left\_primer": primers[0],

"right\_primer": primers[1],

"left\_tm": primers[2],

"right\_tm": primers[3],

"fragment\_length": primers[4]

})

filtered\_pam\_matches.append(match)

####### Create advanced single-axis interactive graph #######

graph = create\_single\_axis\_graph(sequence, filtered\_pam\_matches)

graph\_html = graph.to\_html(full\_html=False)

####### Generate HTML report #######

html\_report = generate\_html\_report(sequence, filtered\_pam\_matches, graph\_html)

####### Save HTML report #######

html\_filename = "crispr\_analysis\_report.html"

with open(html\_filename, "w") as f:

f.write(html\_report)

print(f"HTML report saved to {html\_filename}")

webbrowser.open(html\_filename)

####### Save to Excel #######

save\_to\_excel(filtered\_pam\_matches)