Promoter Project

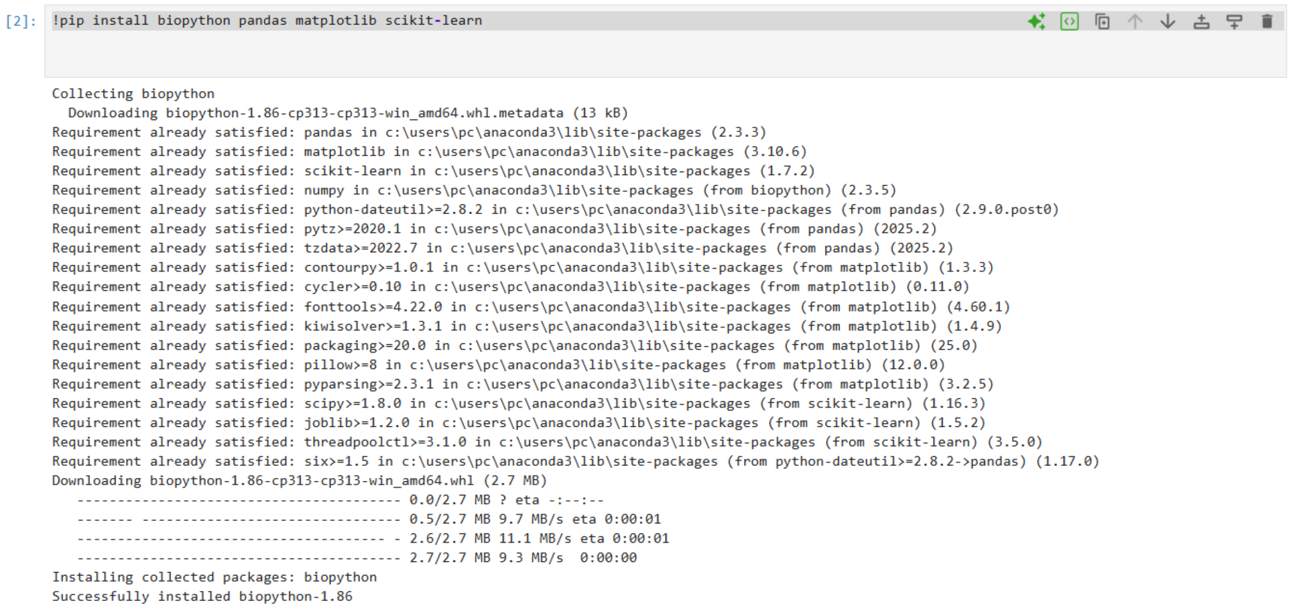
1.Introduction

This project focuses on the exploratory analysis of a labelled DNA sequence dataset (promoters. Data) to investigate simple, interpretable sequence features that distinguish promoter from non-promoter regions. Using basic bioinformatics techniques, the analysis examines sequence length, GC content, and trinucleotide (3-mer) composition, with a particular emphasis on comparing promoter (+) and non-promoter (−) classes. These features provide an accessible starting point for understanding how short-range nucleotide patterns may relate to regulatory function.

The dataset consists of fixed-length DNA sequences from *Escherichia coli*, where each record includes a binary class label, a sequence identifier, and a nucleotide string composed of A, C, G, and T. In addition to computing summary statistics, the repository visualizes the top 10 most frequent 3-mers for each class using bar plots, enabling a clear comparison of compositional biases between promoters and non-promoters. The data used in this project is publicly available from Kaggle and serves as a practical example of feature-based sequence analysis in computational biology.

2. Python code – analyses.

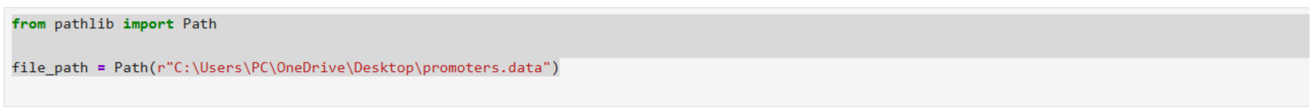
1.Installing Biopython



```
[2]: !pip install biopython pandas matplotlib scikit-learn
```

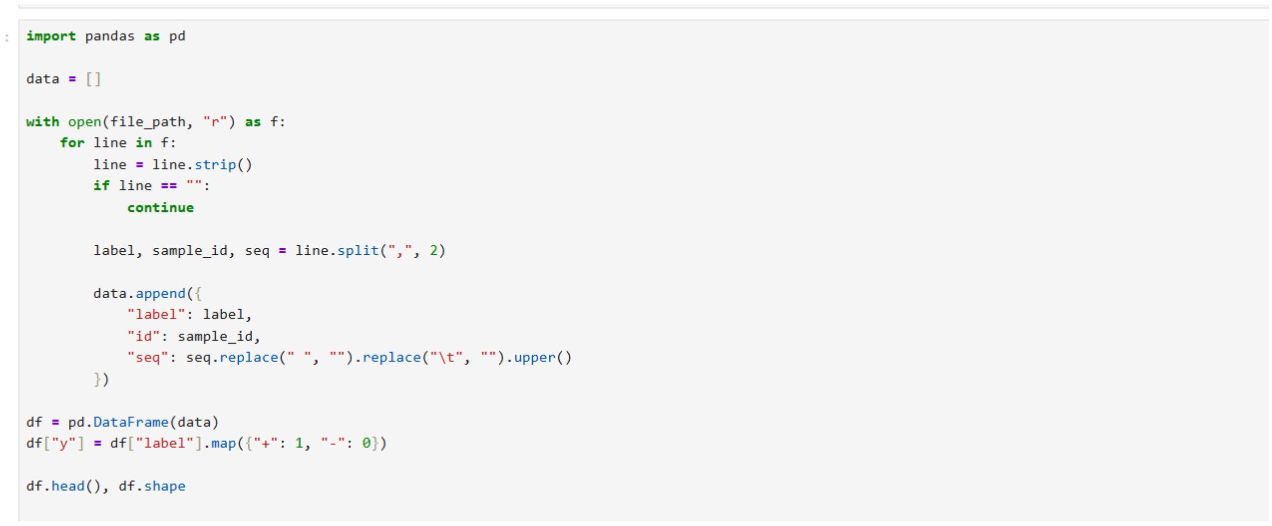
```
Collecting biopython
  Downloading biopython-1.86-cp313-cp313-win_amd64.whl.metadata (13 kB)
Requirement already satisfied: pandas in c:\users\pc\anaconda3\lib\site-packages (2.3.3)
Requirement already satisfied: matplotlib in c:\users\pc\anaconda3\lib\site-packages (3.10.6)
Requirement already satisfied: scikit-learn in c:\users\pc\anaconda3\lib\site-packages (1.7.2)
Requirement already satisfied: numpy in c:\users\pc\anaconda3\lib\site-packages (from biopython) (2.3.5)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\pc\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\pc\anaconda3\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\pc\anaconda3\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (12.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\pc\anaconda3\lib\site-packages (from matplotlib) (3.2.5)
Requirement already satisfied: scipy>=1.8.0 in c:\users\pc\anaconda3\lib\site-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\pc\anaconda3\lib\site-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\pc\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\users\pc\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading biopython-1.86-cp313-cp313-win_amd64.whl (2.7 MB)
   ---------------------------------------- 0.0/2.7 MB ? eta -:--:--
   ------- -------------------------------- 0.5/2.7 MB 9.7 MB/s eta 0:00:01
   -------------------------------------- - 2.6/2.7 MB 11.1 MB/s eta 0:00:01
   ---------------------------------------- 2.7/2.7 MB 9.3 MB/s  0:00:00
Installing collected packages: biopython
Successfully installed biopython-1.86
```

2. This code imports a tool to safely handle file paths and creates a reference to the promoters.data file on your computer.



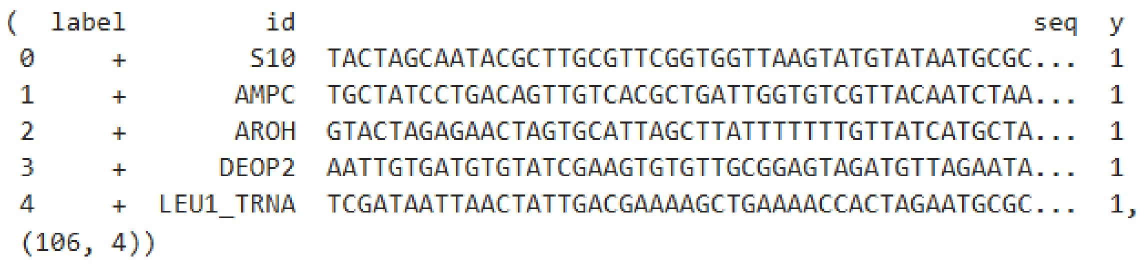
```python
from pathlib import Path

file_path = Path(r"C:\Users\PC\OneDrive\Desktop\promoters.data")
```

3. This code reads a DNA dataset from a text file, cleans each sequence, and converts it into a Pandas table (DataFrame) ready for analysis or machine learning.



```python
import pandas as pd

data = []

with open(file_path, "r") as f:
    for line in f:
        line = line.strip()
        if line == "":
            continue

        label, sample_id, seq = line.split(",", 2)

        data.append({
            "label": label,
            "id": sample_id,
            "seq": seq.replace(" ", "").replace("\t", "").upper()
        })

df = pd.DataFrame(data)
df["y"] = df["label"].map({"+": 1, "-": 0})

df.head(), df.shape
```

Result:

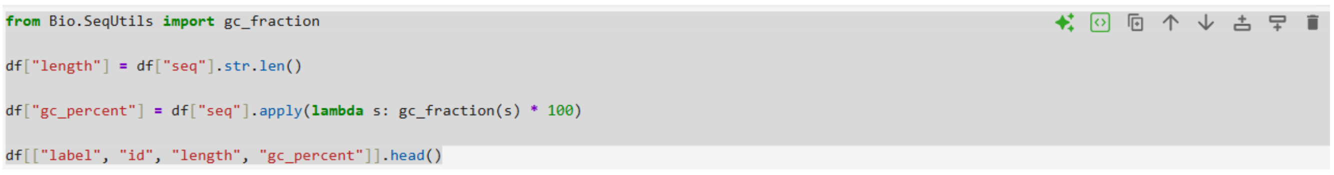


```
(  label         id                                                seq  y
 0     +        S10  TACTAGCAATACGCTTGCGTTCGGTGGTTAAGTATGTATAATGCGC...  1
 1     +       AMPC  TGCTATCCTGACAGTTGTCACGCTGATTGGTGTCGTTACAATCTAA...  1
 2     +       AROH  GTACTAGAGAACTAGTGCATTAGCTTATTTTTTTGTTATCATGCTA...  1
 3     +      DEOP2  AATTGTGATGTGTATCGAAGTGTGTTGCGGAGTAGATGTTAGAATA...  1
 4     +  LEU1_TRNA  TCGATAATTAACTATTGACGAAAAGCTGAAAACCACTAGAATGCGC...  1,
 (106, 4))
```

Interpretation:

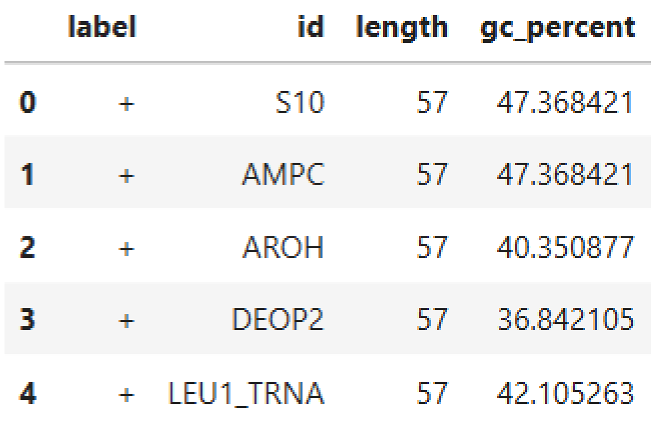
This table shows the first five DNA sequences from the dataset after preprocessing. Each row represents one sequence sample, including its class label (+ for promoter), a unique identifier, the DNA sequence, and the binary target variable (y = 1 for promoter). The dataset contains 106 sequences in total with 4 columns, confirming that the data were successfully loaded and structured for downstream bioinformatics analysis.

4. This code calculates basic biological features for each DNA sequence in the dataset: the sequence length and the GC percentage, and then displays the results.



```python
from Bio.SeqUtils import gc_fraction

df["length"] = df["seq"].str.len()

df["gc_percent"] = df["seq"].apply(lambda s: gc_fraction(s) * 100)

df[["label", "id", "length", "gc_percent"]].head()
```

Result:



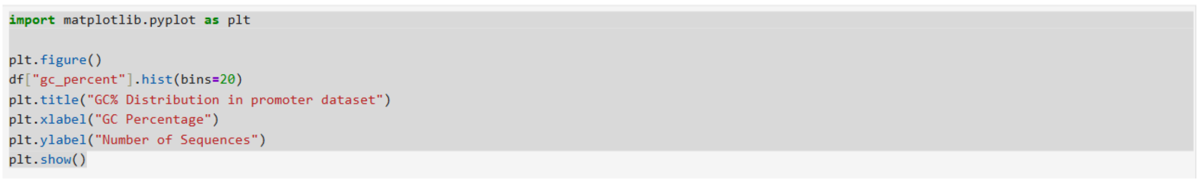
	label	id	length	gc_percent
0	+	S10	57	47.368421
1	+	AMPC	57	47.368421
2	+	AROH	57	40.350877
3	+	DEOP2	57	36.842105
4	+	LEU1_TRNA	57	42.105263

Interpretation:

All sequences shown have a uniform length of 57 nucleotides, which is consistent with typical promoter datasets where fixed-length regions upstream of genes are analysed. The GC content varies across sequences, ranging approximately from 36% to 47%, indicating differences in nucleotide composition between promoter regions. This variability in GC percentage suggests potential biological diversity in promoter architecture, as promoter regions are often enriched in AT-rich motifs rather than high GC content.

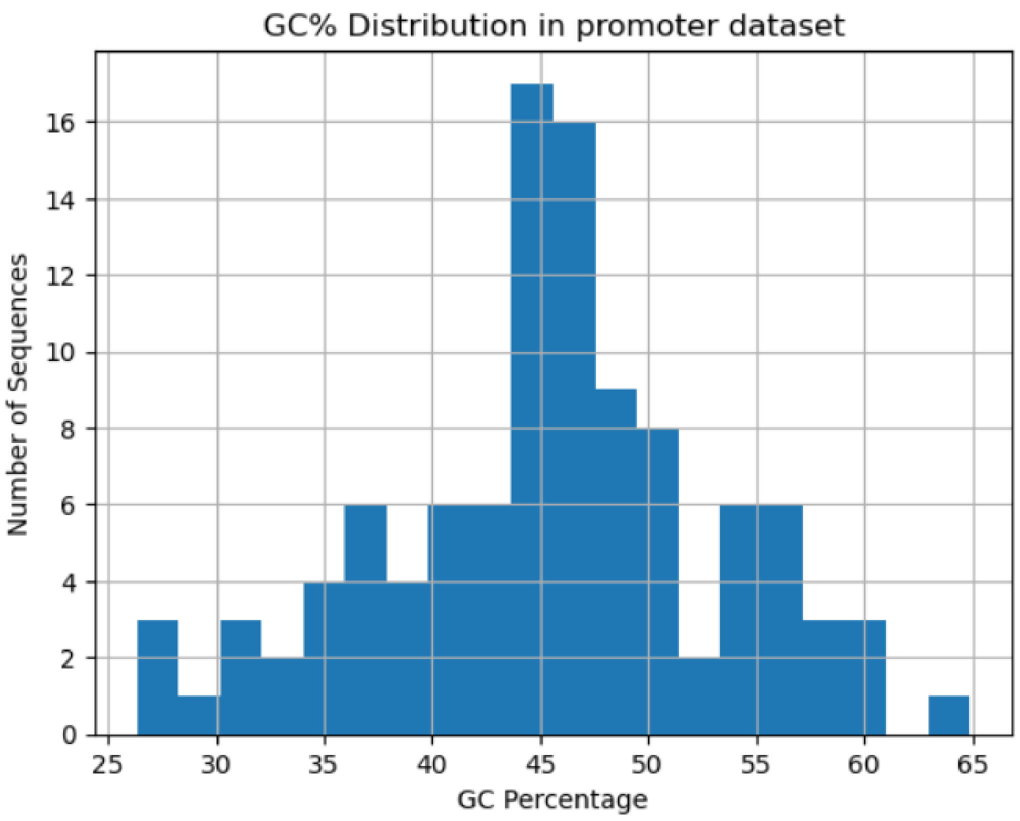
5. Creating a histogram.

This code plots a histogram of GC percentages for all DNA sequences in the dataset. It shows how many sequences fall into different GC-content ranges, helping you see the overall GC distribution.



```python
import matplotlib.pyplot as plt

plt.figure()
df["gc_percent"].hist(bins=20)
plt.title("GC% Distribution in promoter dataset")
plt.xlabel("GC Percentage")
plt.ylabel("Number of Sequences")
plt.show()
```

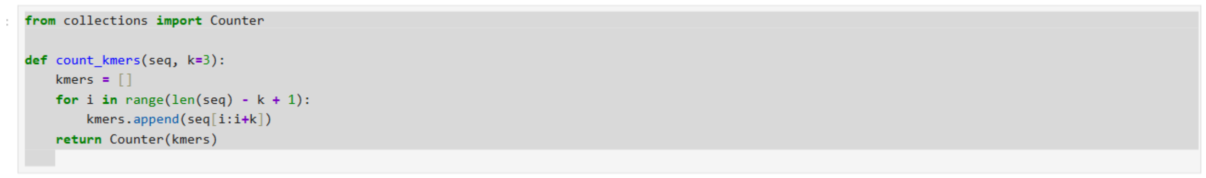
Result:



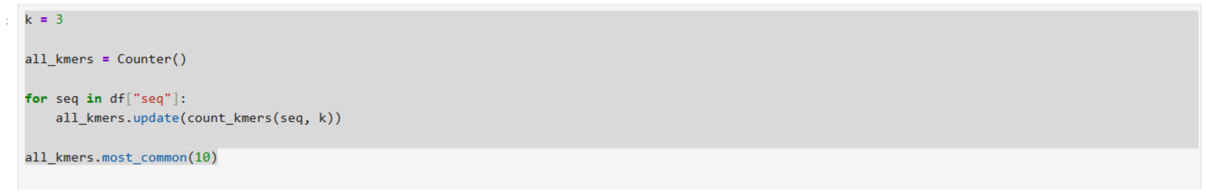
Interpretation:

Most sequences have a moderate GC content centred around ~45–50%, indicating a typical GC-balanced promoter dataset.
There are fewer AT-rich (<35%) and GC-rich (>60%) sequences, suggesting these are less common outliers.

6.This code counts all 3-letter DNA patterns across every sequence and reports the 10 most frequent ones.

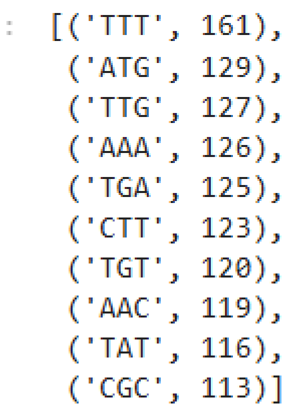


```python
from collections import Counter

def count_kmers(seq, k=3):
    kmers = []
    for i in range(len(seq) - k + 1):
        kmers.append(seq[i:i+k])
    return Counter(kmers)
```



```python
k = 3

all_kmers = Counter()

for seq in df["seq"]:
    all_kmers.update(count_kmers(seq, k))

all_kmers.most_common(10)
```

Result:

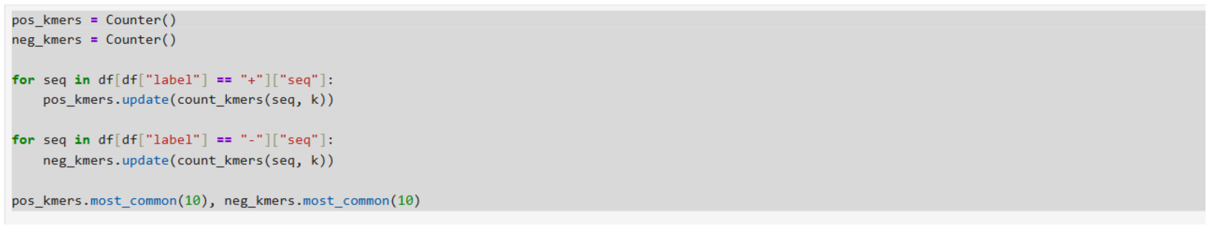


```
: [('TTT', 161),
   ('ATG', 129),
   ('TTG', 127),
   ('AAA', 126),
   ('TGA', 125),
   ('CTT', 123),
   ('TGT', 120),
   ('AAC', 119),
   ('TAT', 116),
   ('CGC', 113)]
```

Interpretation:

These are the 10 most frequent 3-mers across all DNA sequences in your dataset. They are dominated by simple AT-rich patterns (TTT, AAA, TAT) and common biological motifs like ATG (start codon) and TGA (stop codon), suggesting the dataset contains non-random, biologically meaningful sequence structure rather than random DNA.

7. This code separates DNA sequences into promoter (+) and non-promoter (−) groups and counts 3-mers in each group independently.
It then returns the 10 most frequent 3-mers for promoters and for non-promoters, allowing you to compare their sequence patterns.



```
pos_kmers = Counter()
neg_kmers = Counter()

for seq in df[df["label"] == "+"]["seq"]:
    pos_kmers.update(count_kmers(seq, k))

for seq in df[df["label"] == "-"]["seq"]:
    neg_kmers.update(count_kmers(seq, k))

pos_kmers.most_common(10), neg_kmers.most_common(10)
```

Result:

```
([('AAA', 111),
  ('TTT', 108),
  ('TTA', 75),
  ('TAT', 75),
  ('AAT', 74),
  ('ATA', 72),
  ('TAA', 70),
  ('TGT', 69),
  ('CGC', 65),
  ('TTG', 65)],
 [('ATG', 77),
  ('GAG', 75),
  ('TGA', 73),
  ('ACG', 69),
  ('TCA', 69),
  ('CTC', 65),
  ('AAC', 63),
  ('TTG', 62),
  ('GAC', 62),
  ('AGA', 61)])
```

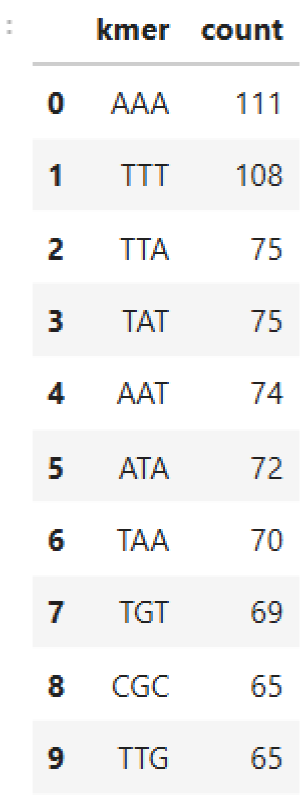
Interpretation:

These results show different dominant 3-mers in the two classes (likely non-promoter vs promoter sequences).
One group is strongly AT-rich (AAA, TTT, TTA, TAT, ATA), while the other contains more GC-mixed and functional motifs (ATG, TGA, ACG, GAG), indicating that promoter and non-promoter regions have distinct sequence patterns that can be used for classification.

8. This code extracts the 10 most frequent 3-mers from the promoter (+) sequences and stores them in a list.
It then converts that list into a Pandas DataFrame with columns for the k-mer and its frequency, making the results easy to view or plot.

```
top_pos = pos_kmers.most_common(10)
df_top_kmers = pd.DataFrame(top_pos, columns=["kmer", "count"])
df_top_kmers
```

Result:

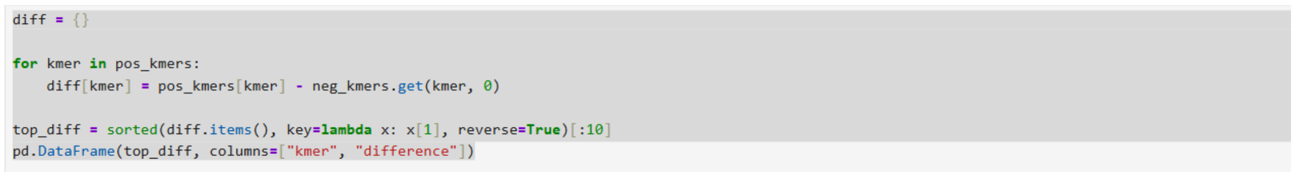


	kmer	count
0	AAA	111
1	TTT	108
2	TTA	75
3	TAT	75
4	AAT	74
5	ATA	72
6	TAA	70
7	TGT	69
8	CGC	65
9	TTG	65

Interpretation:

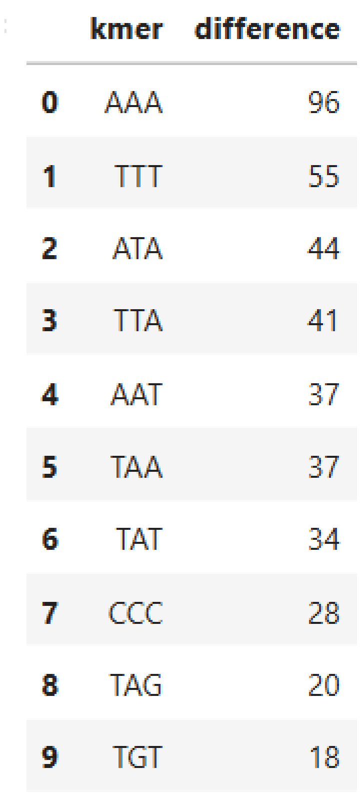
This table shows the 10 most frequent 3-mers in one class of sequences (likely non-promoters).
The dominance of AT-rich motifs (AAA, TTT, TTA, TAT, ATA) indicates these sequences are AT-biased and lack strong regulatory motifs, which is typical for non-promoter regions.

9. This code computes how much more frequent each 3-mer is in promoter (+) sequences compared to non-promoter (−) sequences.
It then selects the 10 k-mers with the largest positive differences and displays them in a Data Frame, highlighting motifs most enriched in promoters.



```
diff = {}

for kmer in pos_kmers:
    diff[kmer] = pos_kmers[kmer] - neg_kmers.get(kmer, 0)

top_diff = sorted(diff.items(), key=lambda x: x[1], reverse=True)[:10]
pd.DataFrame(top_diff, columns=["kmer", "difference"])
```

Result:

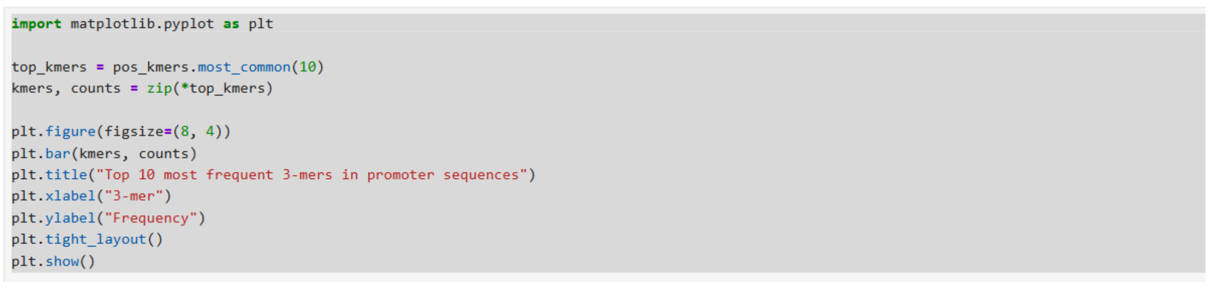


	kmer	difference
0	AAA	96
1	TTT	55
2	ATA	44
3	TTA	41
4	AAT	37
5	TAA	37
6	TAT	34
7	CCC	28
8	TAG	20
9	TGT	18

Interpretation:

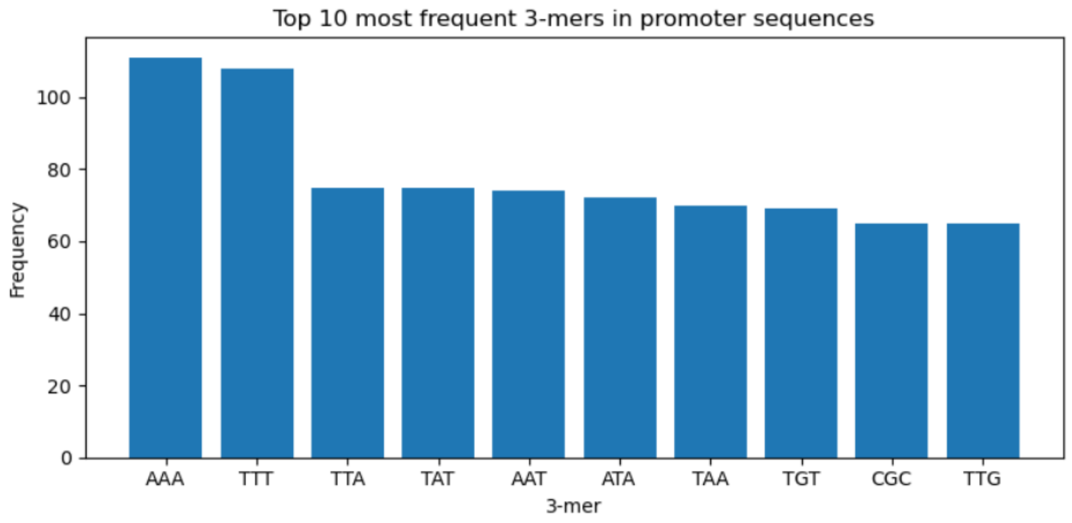
This table shows the 3-mers with the largest frequency differences between promoter (+) and non-promoter (−) sequences.
High values (e.g. AAA, TTT, ATA) indicate motifs that are much more enriched in one class, making them strong discriminative features for promoter classification.

10. This code takes the 10 most frequent 3-mers from promoter (+) sequences and visualizes them as a bar chart.
Each bar represents a 3-mer and its frequency, making it easy to see which sequence motifs are most common in promoters.



```python
import matplotlib.pyplot as plt

top_kmers = pos_kmers.most_common(10)
kmers, counts = zip(*top_kmers)

plt.figure(figsize=(8, 4))
plt.bar(kmers, counts)
plt.title("Top 10 most frequent 3-mers in promoter sequences")
plt.xlabel("3-mer")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

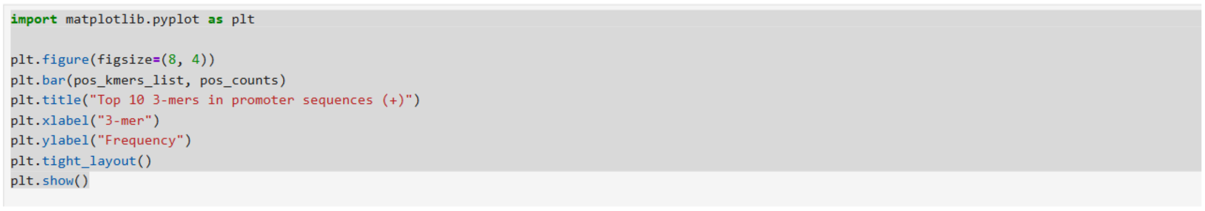
Result:



Interpretation:

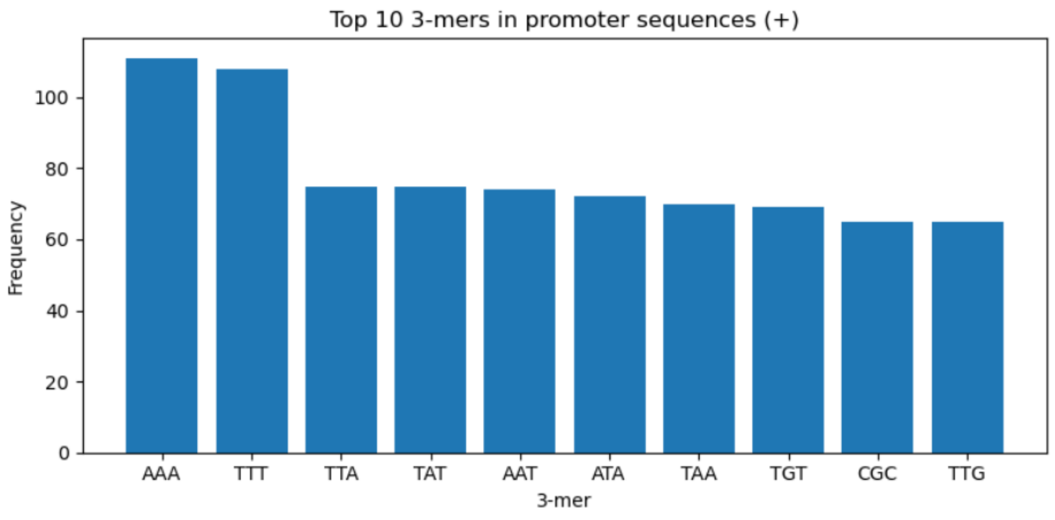
This bar chart shows the 10 most frequent 3-mers in promoter (+) sequences, with AAA and TTT dominating, indicating a strong AT-rich composition.
Less frequent but notable motifs (e.g. TGT, CGC) suggest additional sequence structure beyond simple AT bias, which may help distinguish promoters from non-promoters.

11. This code creates a bar chart showing how often each of the top 3-mers appears in promoter (+) sequences.
It sets the figure size, labels, and title, then displays the plot so you can visually compare 3-mer frequencies.



```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
plt.bar(pos_kmers_list, pos_counts)
plt.title("Top 10 3-mers in promoter sequences (+)")
plt.xlabel("3-mer")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

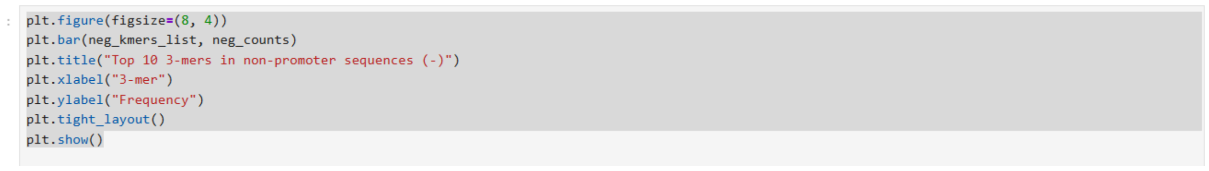
Result:



Interpretation:

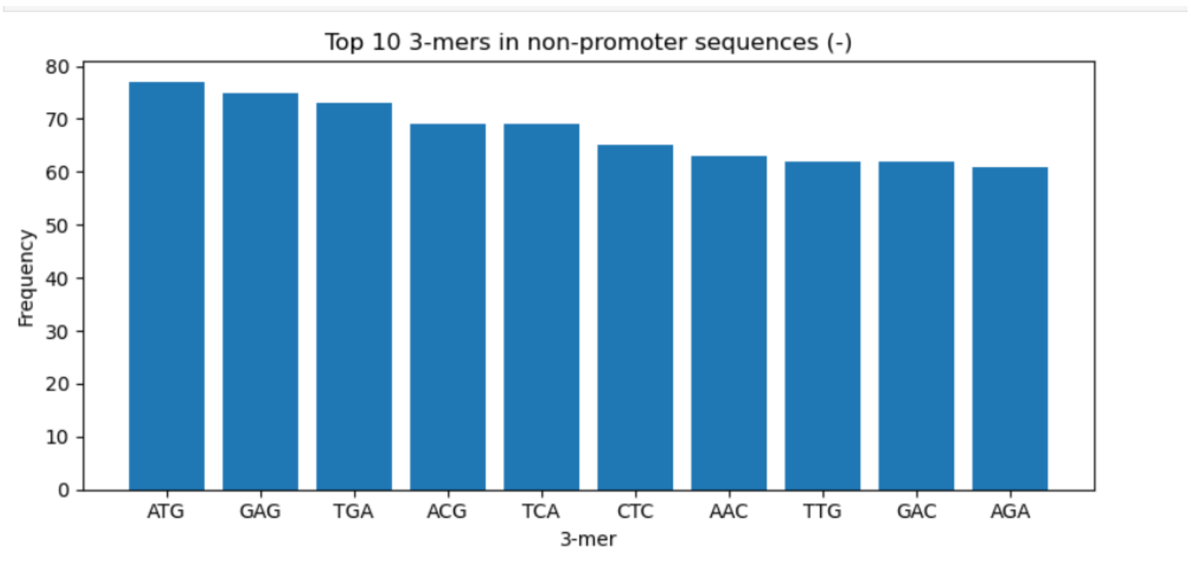
This plot shows the most frequent 3-mers in promoter (+) sequences, with AAA and TTT appearing far more often than others, indicating a strong AT-rich character of promoters. The presence of less frequent motifs like CGC suggests additional, more specific sequence patterns beyond simple AT repeats.

12. This code creates and displays a bar chart of the 10 most frequent 3-mers in non-promoter (−) sequences.
It labels the axes and title so you can visually compare how often each 3-mer appears in non-promoter regions.



```
plt.figure(figsize=(8, 4))
plt.bar(neg_kmers_list, neg_counts)
plt.title("Top 10 3-mers in non-promoter sequences (-)")
plt.xlabel("3-mer")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

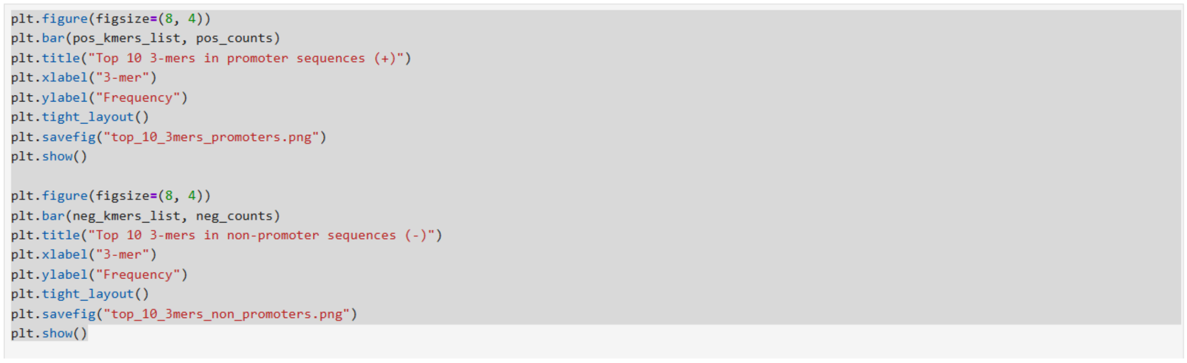
Result:



Interpretation:

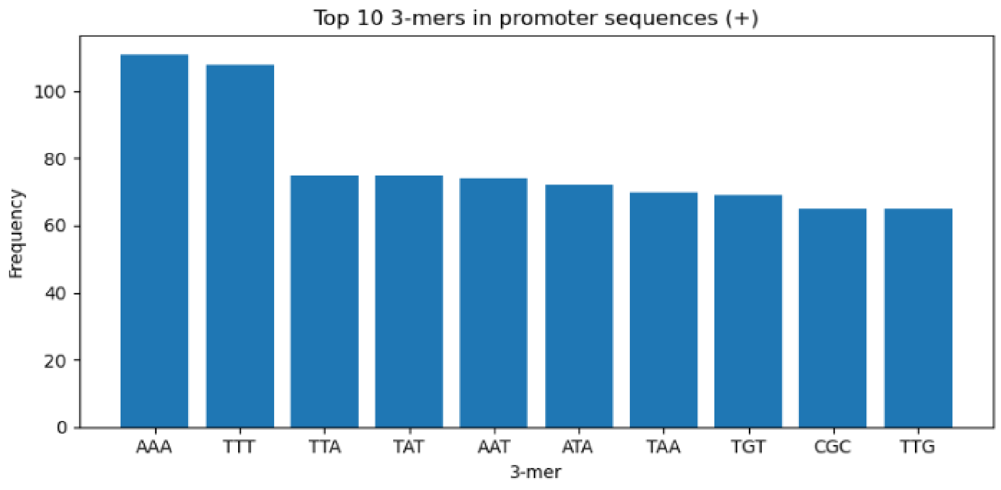
This plot shows the most frequent 3-mers in non-promoter (−) sequences, which are more GC-mixed and diverse compared to promoters.
Motifs like ATG, TGA, ACG, GAG suggest coding-like or background genomic patterns, distinguishing non-promoter regions from the AT-rich promoter sequences.

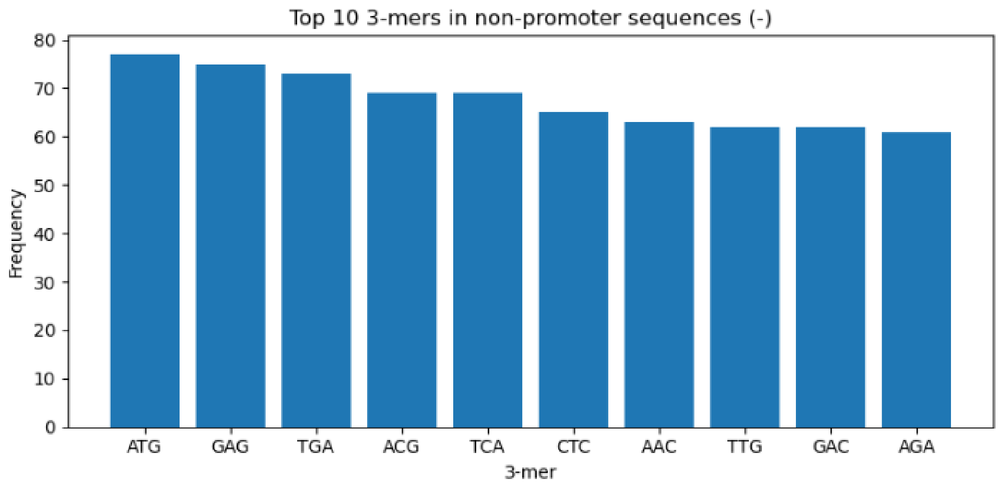
13. This code creates two separate bar charts showing the top 10 most frequent 3-mers in promoter (+) and non-promoter (−) sequences.
Each plot is saved as a PNG image file (top_10_3mers_promoters.png and top_10_3mers_non_promoters.png) and also displayed on screen.



```
plt.figure(figsize=(8, 4))
plt.bar(pos_kmers_list, pos_counts)
plt.title("Top 10 3-mers in promoter sequences (+)")
plt.xlabel("3-mer")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("top_10_3mers_promoters.png")
plt.show()

plt.figure(figsize=(8, 4))
plt.bar(neg_kmers_list, neg_counts)
plt.title("Top 10 3-mers in non-promoter sequences (-)")
plt.xlabel("3-mer")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("top_10_3mers_non_promoters.png")
plt.show()
```

Result:





Interpretation:

These two plots compare the top 10 most frequent 3-mers in promoter (+) and non-promoter (−) sequences.
Promoters are clearly AT-rich (dominated by AAA, TTT, TTA, TAT), while non-promoters show more GC-mixed and coding-like motifs (ATG, TGA, ACG, GAG), indicating distinct sequence patterns between the two classes.