SeqPlotter: Python package for sequence data analysis and visualization

SeqPlotter is a Python package developed to streamline the visualization and analysis of biological sequence data. It provides seamlessly integration into your Python workflow, with robust data handling capabilities, intuitive visualization, and useful analysis features. From parsing raw sequence files to creating stunning plots and charts.

1. Working with DNA sequences

1.1. Importing DNA module

In [1]:
```
# Importing module
from seqplotter.nucl import DNA
```

1.2. Creating DNA objects

```
>>> dna_seq1 = DNA("seqID", "DNA_sequence")
```

Output: retruns a DNA object with seqid and seq pair

In [2]:
```
# creating DNA object with seqID and sequence pair
seq1 = DNA("seq1", "GTGTTTTGACTAATAATTGGTCAAGCCTA")
```

1.3. Sequence length

```
>>> dna_seq1.length()
```

Output: returns sequence length (int) in bp

In [3]:
```
# print sequence length
seq1.length()
```

Out[3]:
```
29
```

1.4. Sequence composition

```
>>> dna_seq1.comp()
```

Output: returns a dict object with sequence composition i.e. count of each nucleotides

In [4]:
```
# print sequence composition
seq1.comp()
```

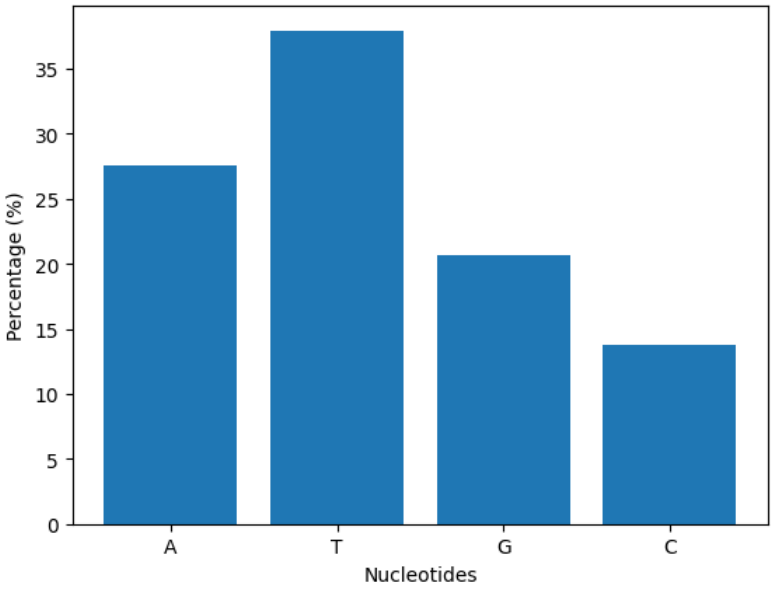
Out[4]:
```
{'A': 27.586206896551722,
 'T': 37.93103448275862,
 'G': 20.689655172413794,
 'C': 13.793103448275861}
```

1.5 Sequence composition barplot

```
>>> seq1_comp = dna_seq1.comp()
>>> DNA.barplot(seq1_comp)
```

Output: generate barplot showing sequence composition

In [5]:
```
# plot sequence composition
comp = seq1.comp()
DNA.barplot(comp)
```



1.6. Sequence slicing

```
>>> # returns nucleotide for the given position
>>> dna_seq1.slice(pos)

>>> # returns nucleotides between start to end position
>>> dna_seq1.slice(start_pos, end_pos)
```

Output: returns a string that contain the sliced sequence

In [6]:
```
# slice particular position
seq1.slice(2)
```

Out[6]: 'T'

In [7]:
```
# slice with start and end index
seq1.slice(2,5)
```

Out[7]: 'TGTT'

1.7. GC content

```
>>> dna_seq1.gc_percent()
```

Output: returns GC percent (string)

In [8]:
```
# GC percent
seq1.gc_percent()
```

Out[8]: '34.48'

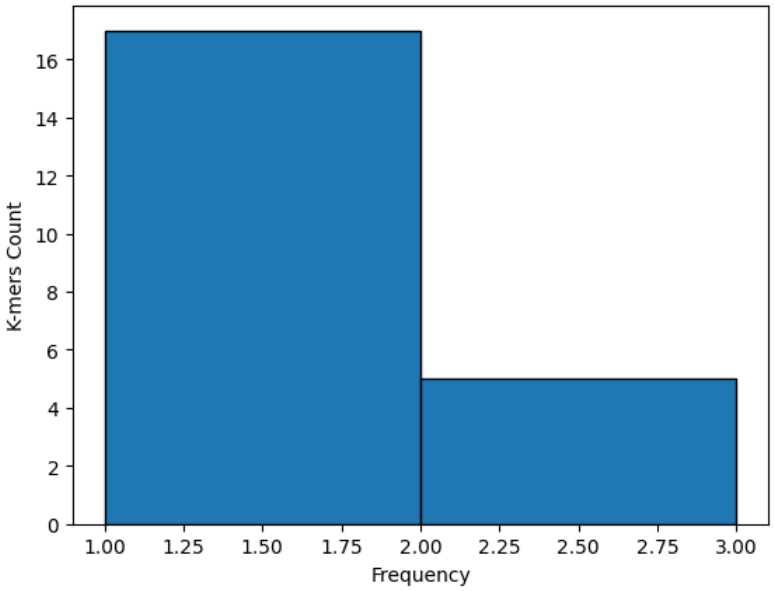
1.8. K-mer abundance plot

```
# generated k-mer plot with default kmer size (k=3)
>>> dna_seq1..kmer_abundance_plot()

# generate k-mer plot with k=6
>>> dna_seq1..kmer_abundance_plot(6)
```

Output: generate k-mer plot (histogram)

In [9]:
```
# kmer abundance plot
seq1.kmer_abundance_plot()
```



1.9. Readling FASTA file (DNA Sequence)

```
>>> fasta_records = DNA.read_fasta("/path/to/fasta/file")
```

Output: returns a list containing DNA object(s)

In [10…
```
# reading a FASTA file
data = DNA.read_fasta("seq.fasta")
```

1.10. Using head() and tail() functions

```
>>> # return summary of first 5 sequence (default)
>>> DNA.head()

>>> # return summary of first 10 sequence
>>> DNA.head(10)

>>> # return summary of last 5 sequence (default)
>>> DNA.tail()

>>> # return summary of last 10 sequence
>>> DNA.tail(10)
```

Output: print head and tail summary of parsed FASTA file

In [11…
```
# print first five records (by default)
DNA.head()
```

```
seqID   seq
seq1:   [CTTTTCTTTAGTGTTTTGACTAATAATTGGTCAAGCCTACCATTACAAACTATGTTCCATTACCAGTACA]
seq2:   [CTTTTCTTTAGTGTTTTGACTAATAATTGGTCAAGCCTACCATTACAAACTATGTTCCATTACCAGTACA]
seq3:   [CTTTTCTTTAGTGTTTTGACTAATAATTGGTCAAGCCTACCATTACAAACTATGTTCCATTACCAGTACA]
seq4:   [CTTTTCTTTAGTGTTTTGACTAATAATTGGAAACTATGTTCCATTACCAGTACA]
seq5:   [TCACAGAAGAGCAAGAGGCTCTTGTAGTGAAGTCTTGGAGTGTCATGAAGAAAAACTCAGCTGAATTAGG]
```

In [12…
```
# print first two records
DNA.head(2)
```

```
seqID   seq
seq1:   [CTTTTCTTTAGTGTTTTGACTAATAATTGGTCAAGCCTACCATTACAAACTATGTTCCATTACCAGTACA]
seq2:   [CTTTTCTTTAGTGTTTTGACTAATAATTGGTCAAGCCTACCATTACAAACTATGTTCCATTACCAGTACA]
```

In [13…
```
# print last five records (by default)
DNA.tail()
```

```
seqID   seq
seq5:   [TCACAGAAGAGCAAGAGGCTCTTGTAGTGAAGTCTTGGAGTGTCATGAAGAAAAACTCAGCTGAATTAGG]
seq6:   [TCTCAAACTCTTCATCAAGTAAGTAATGATCCCATTGATCTCTCTCTATTTTCTTTTTATGTATATAGCA]
seq7:   [TGAGATATGAACTACTATTTTGAACTGTAGGATCTTTGAGATTGCACCAACAACGA]
seq8:   [TTCTTGAGAGACTCACCAATTCCTGCTGAGCAAAATCCAAAGCTCAAGCCTCACGCAATGTCTGT]
seq9:   [TCATGGTAATAATCAATATCAAATAACATGATTTT]
```

In [14…
```
# print last six records
DNA.tail(3)
```

```
seqID   seq
```

```
seq7:   [TGAGATATGAACTACTATTTTGAACTGTAGGATCTTTGAGATTGCACCAACAACGA]
seq8:   [TTCTTGAGAGACTCACCAATTCCTGCTGAGCAAAATCCAAAGCTCAAGCCTCACGCAATGTCTGT]
seq9:   [TCATGGTAATAATCAATATCAAATAACATGATTTT]
```

1.11. Count records

```
>>> DNA.count(fasta_records)
```

Output: returns sequence count (int) i.e. number of records

In [15…
```
# print record counts
DNA.count(data)
```

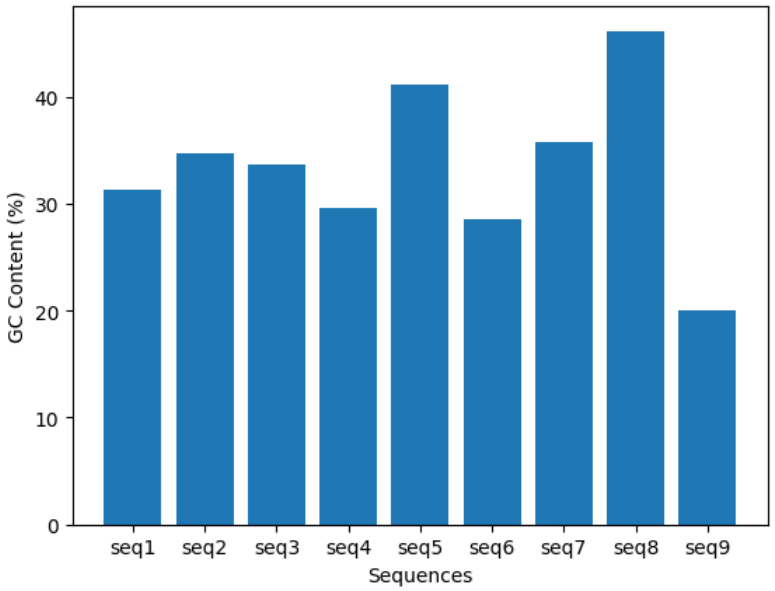
Out[15…
```
9
```

1.12. GC percent per sequence plot

```
>>> DNA.gc_plot(fasta_records)
```

Output: generate barplot showing per sequence GC percent

In [16…
```
# GC barplot
DNA.gc_plot(data)
```

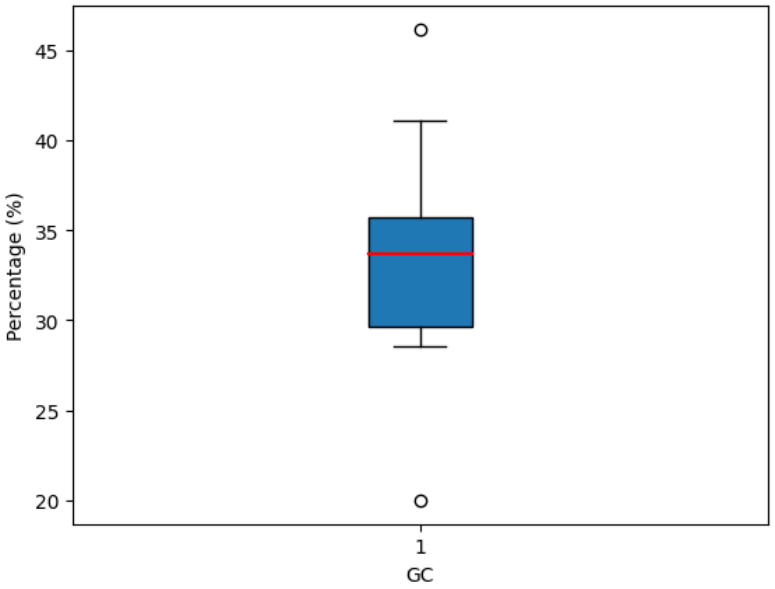


1.13. Boxplot showing GC distribution

```
>>> DNA.gc_distribution_plot(fasta_records)
```

Output: generate boxplot showing overall GC distribution across all the sequences

In [17…
```
# GC distribution plot
DNA.gc_distribution_plot(data)
```

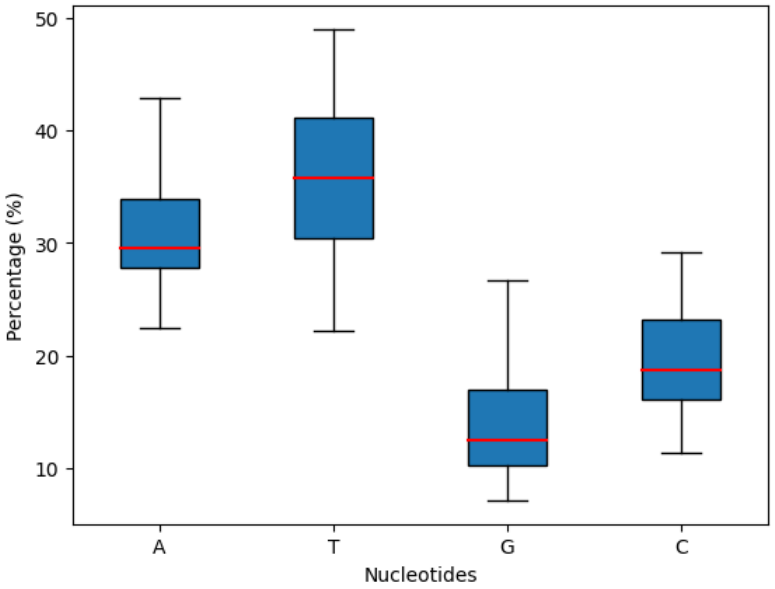


1.14. Boxplot showing nucleotides distribution

```
>>> DNA.nt_distribution_plot(fasta_records)
```

Output: generate boxplot showing overall nucleotides distribution across all the sequences

In [18…
```
# Nucleotides distribution plot
DNA.nt_distribution_plot(data)
```

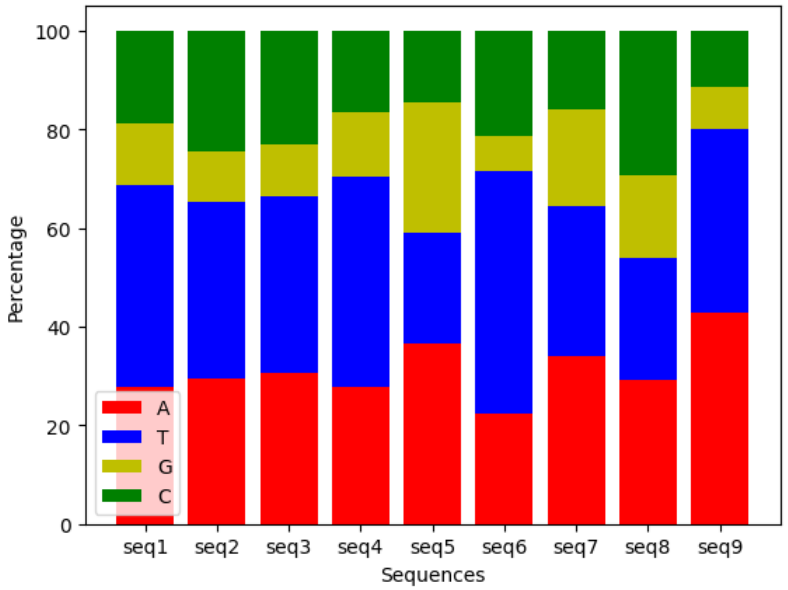


1.15. Stacked barplot showing per sequence base composition

```
>>> DNA.per_sequence_comp(fasta_records)
```

Output: generate stacked barplot showing per sequence nucleotide composition

In [19…
```
# Per sequence nucleotides composition
DNA.per_sequence_comp(data)
```

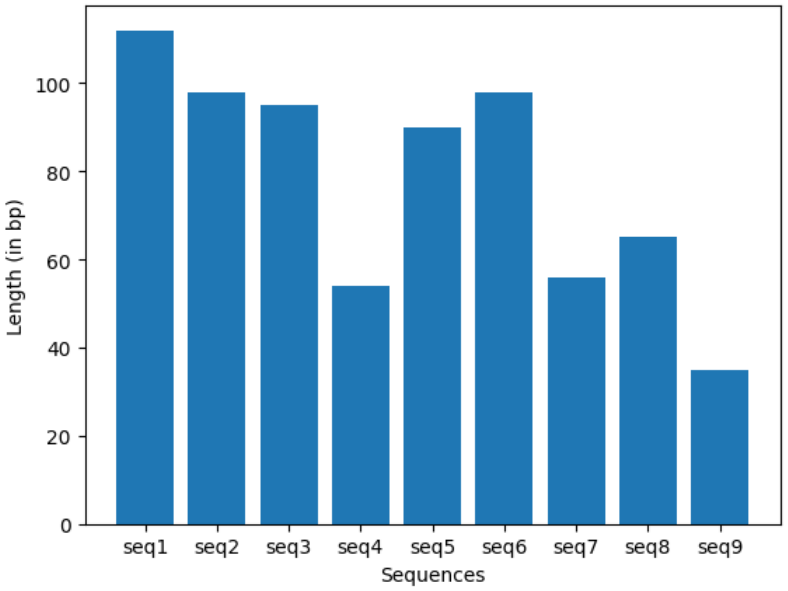


1.16. Barplot showing per sequence length

```
>>> DNA.length_plot(fasta_records)
```

Output: generate barplot showing per sequence length distribution

In [20…
```
# Per sequence length
DNA.length_plot(data)
```

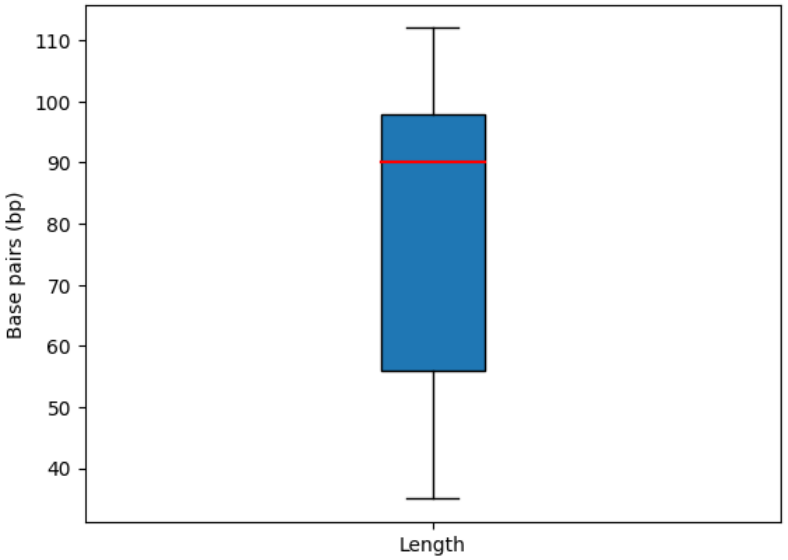


1.17. Boxplot showing overall length distribution

```
>>> DNA.length_distribution_plot(fasta_records)
```

Output: generate boxplot showing overall length distribution across all the sequences

In [21…
```
# Boxplot showing length distribution
DNA.length_distribution_plot(data)
```



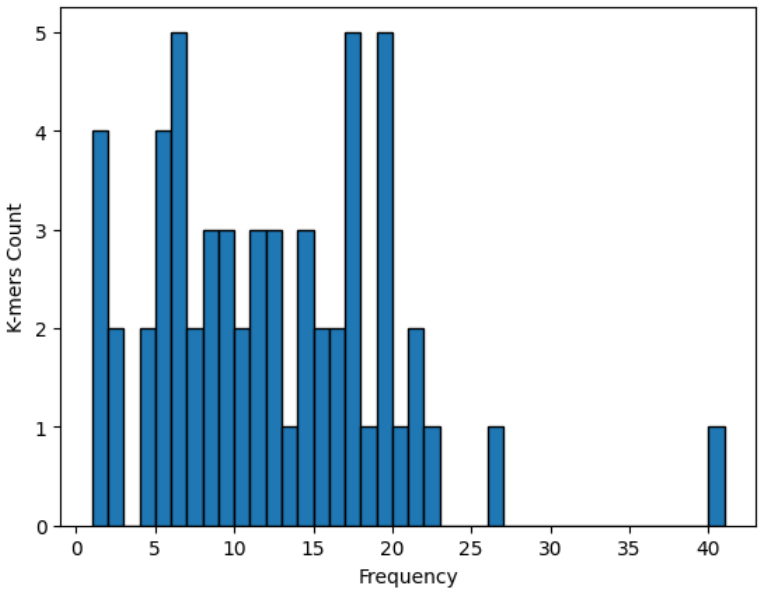
1.18. Overall k-mer abundance pattern

```
# default k-mer size (k=3)
>>> DNA.combined_kmer_abundance_plot(fasta_records)

# kmer size (k=7)
>>> DNA.combined_kmer_abundance_plot(fasta_records, 7)
```

Output: generate combined k-mer abundance plot (histogram) for all the sequence

In [22…
```
# Combined kmers distribution
DNA.combined_kmer_abundance_plot(data)
```



1.19. Converting sequence object to feature matrix

```
>>> from seqplotter_analysis import seq2feature
>>> feature_matrix = seq2feature(fasta_records) # with default options (k=3, min_sample=1)

>>> feature_matrix = seq2feature(fasta_records, 6, 2) # with (k=6, min_sample=2)
```

Options:

- k: kmer size
- min_sample: threshold of considering a k-mer as feature

Output: returns a dict object containing following keys ("matrix", "sample", "kmers"), where "kmers" are columns and "sample" are rows

< *dict{"matrix": numpy_array, "sample": seqID_list, "kmers": kmers_list}* >

In [23…
```
# Importing module
from seqplotter_analysis import seq2feature

# Converting sequence to feature matrix
feature_matrix = seq2feature(data)

# Visualizing feature matrix
import pandas as pd
pd.DataFrame(feature_matrix["matrix"], index=feature_matrix["sample"], columns=feature_matrix["kmers"])
```

Out[23…

	CTT	TTT	TTC	TCT	TTA	TAG	AGT	GTG	TGT	GTT	...	GAA	AGA	GAG	GCA	AGG	GCT	CTC	CTG	TGC	ACG
seq1	4	10	3	2	5	2	4	2	3	3	...	0	0	0	0	0	0	0	0	0	0
seq2	2	5	3	1	4	1	2	1	3	3	...	0	0	0	0	0	0	0	0	0	0
seq3	2	5	3	1	4	1	3	1	3	3	...	0	0	0	0	0	0	0	0	0	0
seq4	2	5	2	1	2	1	2	1	2	2	...	1	0	0	0	0	0	0	0	0	0
seq5	2	0	0	2	2	3	3	2	2	0	...	7	4	3	1	3	3	3	2	0	0
seq6	3	10	4	12	2	1	2	0	1	0	...	0	0	0	1	0	0	8	0	0	0
seq7	1	3	0	1	0	1	0	0	1	0	...	2	2	2	1	1	0	0	1	1	1
seq8	1	0	2	2	0	0	0	0	2	0	...	0	2	3	2	0	2	3	3	1	1
seq9	0	2	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

9 rows × 54 columns

1.20. Principal component analysis

```
>>> from seqplotter_analysis import PCA
>>> pca_matrix = PCA(feature_matrix) # return pca matrix with first 2 PCs

>>> pca_matrix = PCA(feature_matrix, 3) # return pca matrix with first 3 PCs
```

Output: returns a dict object containing following keys ("matrix", "sample", "components"), where "components" are columns and "sample" are rows

< dict{"matrix": numpy_array, "sample": seqID_list, "components": PCA_components} >

In [24…
```
# Importing module
from seqplotter_analysis import PCA

# Performing PCA
pca_matrix = PCA(feature_matrix)

# Visualizing PCA matrix
import pandas as pd
pd.DataFrame(pca_matrix["matrix"], index=pca_matrix["sample"], columns=pca_matrix["components"])
```

Out[24…

	PC-1	PC-2
seq1	9.629055	2.243708
seq2	5.947428	3.774627
seq3	6.225862	4.230590
seq4	0.776820	1.919865
seq5	-9.639273	0.927017
seq6	3.503045	-15.066248
seq7	-4.161328	0.626651
seq8	-7.607692	0.437185
seq9	-4.673917	0.906603

2. Working with Protein sequences

2.1. Importing PROT module

In [25…
```
# Importing module
```

```
from seqplotter.prot import PROT
```

2.2 Creating PROT object

```
>>> prot_seq1 = PROT("seqID", "protein_sequence")
```

Output: retruns a DNA object with seqid and seq pair

In [26…
```
# Creating Protein object with seqID and sequence pair
seq1 = PROT("seq1", "VDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAV")
```

2.3. Sequence length

```
>>> prot_seq1.length()
```

Output: returns sequence length (int) in bp

In [27…
```
# print sequence length
seq1.length()
```

Out[27… 35

2.4. Sequence composition

```
>>> prot_seq1.comp()
```

Output: returns a dict object with sequence composition i.e. count of each amino acids

In [28…
```
# print sequence composition
seq1.comp()
```

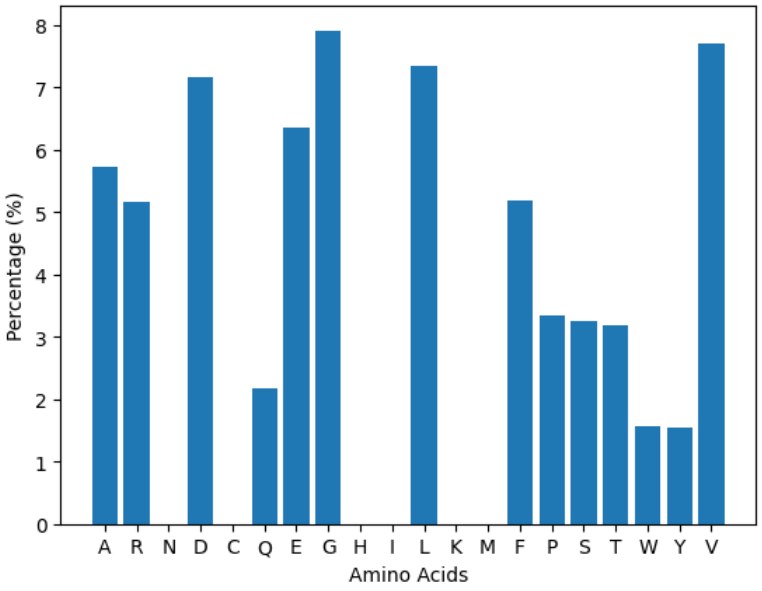
Out[28…
```
{'A': 5.714285714285714,
 'R': 5.166051660516605,
 'N': 0.0,
 'D': 7.163266076755573,
 'C': 0.0,
 'Q': 2.1718543403148107,
 'E': 6.353851345091472,
 'G': 7.909936022986237,
 'H': 0.0,
 'I': 0.0,
 'L': 7.342245635482005,
 'K': 0.0,
 'M': 0.0,
 'F': 5.188382310331203,
 'P': 3.3327849177003355,
 'S': 3.260373880823663,
 'T': 3.1947334484658114,
 'W': 1.567453020447904,
 'Y': 1.5536341292308162,
 'V': 7.701922950367426}
```

2.5. Sequence composition plot

```
>>> prot_comp = prot_seq1.comp()
>>> PROT.barplot(prot_comp)
```

Output: generate barplot showing sequence composition

In [29…
```
# plot sequence composition
comp = seq1.comp()
PROT.barplot(comp)
```



2.6. Molecular weight in KDa

```
>>> prot_seq1.mol_weight()
```

Output: return molecular weight (int) of a protein sequence in KDa

In [30…
```
# molecular weight
seq1.mol_weight()
```

Out[30…
```
3.87
```

2.7. Sequence slicing

```
>>> # returns amino acid for the given position
>>> prot_seq1.slice(pos)

>>> # returns amino acids between start to end position
>>> prot_seq1.slice(start_pos, end_pos)
```

Output: returns a string that contain the sliced sequence

In [31…
```
# slice particular position
seq1.slice(6)
```

Out[31…
```
'G'
```

In [32…
```
# slice with start and end index
seq1.slice(4,8)
```

Out[32…
```
'VGGEA'
```

2.8. Reading FASTA file (Protein sequence)

```
>>> fasta_records = PROT.read_fasta("/path/to/fasta/file")
```

Output: returns a list containing PROT object(s)

In [33…
```
# reading a FASTA file
data = PROT.read_fasta("prot_seq.fasta")
```

2.9. Using head() and tail() functions

```
>>> # return summary of first 5 sequence (default)
>>> PROT.head()

>>> # return summary of first 10 sequence
>>> PROT.head(10)

>>> # return summary of last 5 sequence (default)
>>> PROT.tail()
```

```
>>> # return summary of last 10 sequence
>>> PROT.tail(10)
```

Output: print head and tail summary of parsed FASTA file

In [34…
```
# Head
PROT.head(5)
```

```
seqID   seq
seq1:   [MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG]
seq2:   [MLPFWKRLLYAAVIAGALVGADAQFWKTAGTAGSIQDSVKHYNRNEPKFPIDDSYDIVDSAGVARGDLPP]
seq3:   [TGQKGDRGEPGLNGLPGNPGQKGEPGRAGATGKPGLLGPPGPPGGGRGTPGPPGPKGPRGYVGAPGPQGL]
seq4:   [LPGATGEPGKPALCDLSLIEPLKGDKGYPGAPGAKGVQGFKGAEGLPGIPGPKGEFGFKGEKGLSGAPGN]
```

In [35…
```
# Tail
PROT.tail()
```

```
seqID   seq
seq1:   [MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG]
seq2:   [MLPFWKRLLYAAVIAGALVGADAQFWKTAGTAGSIQDSVKHYNRNEPKFPIDDSYDIVDSAGVARGDLPP]
seq3:   [TGQKGDRGEPGLNGLPGNPGQKGEPGRAGATGKPGLLGPPGPPGGGRGTPGPPGPKGPRGYVGAPGPQGL]
seq4:   [LPGATGEPGKPALCDLSLIEPLKGDKGYPGAPGAKGVQGFKGAEGLPGIPGPKGEFGFKGEKGLSGAPGN]
```

2.10. Count records

```
>>> PROT.count(fasta_records)
```

Output: returns sequence count (int) i.e. number of records

In [36…
```
# Sequence count
PROT.count(data)
```

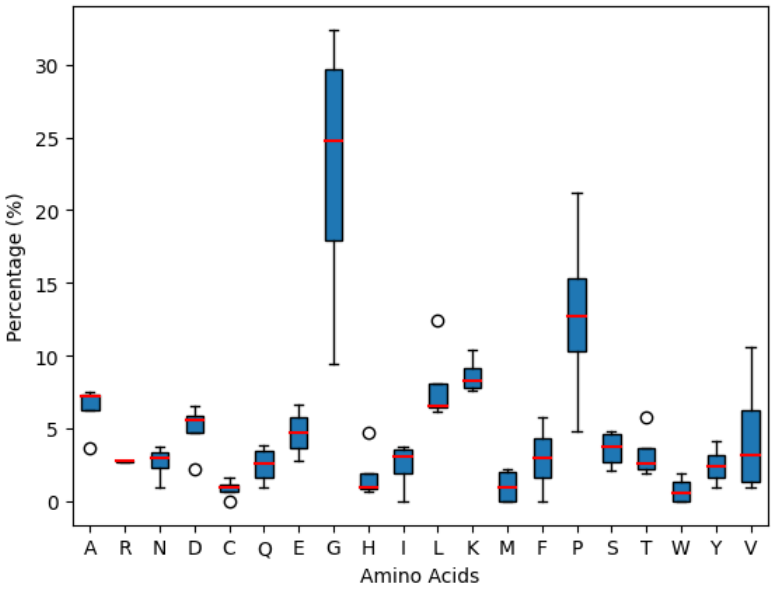
Out[36… 4

2.11. Amino acids distribution plot

```
>>> PROT.aa_distribution_plot(fasta_records)
```

Output: generate boxplot showing overall amino acids distribution across all the sequences

In [37…
```
# Boxplot aa distribution plot
PROT.aa_distribution_plot(data)
```

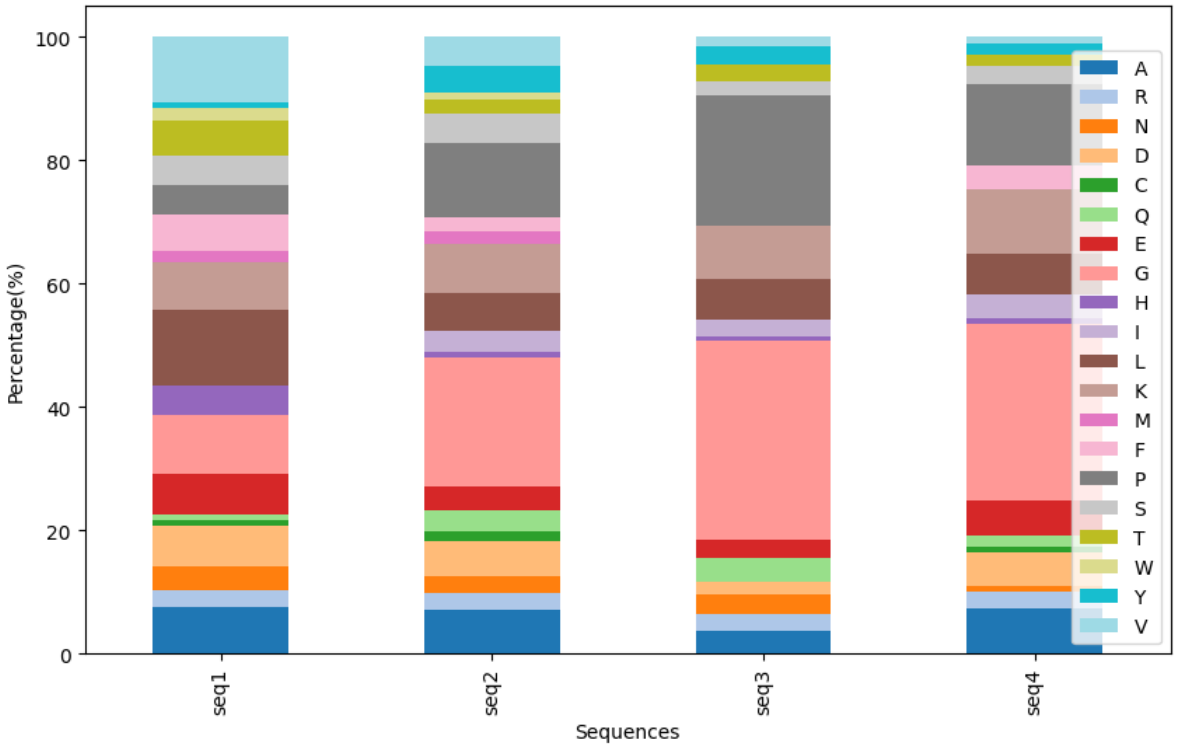


2.12. Stacked barplot showing per sequence amino acid composition

```
>>> PROT.per_sequence_comp(fasta_records)
```

Output: generate stacked barplot showing per sequence amino acids composition

In [38…
```
# Per sequence nucleotides composition
PROT.per_sequence_comp(data)
```

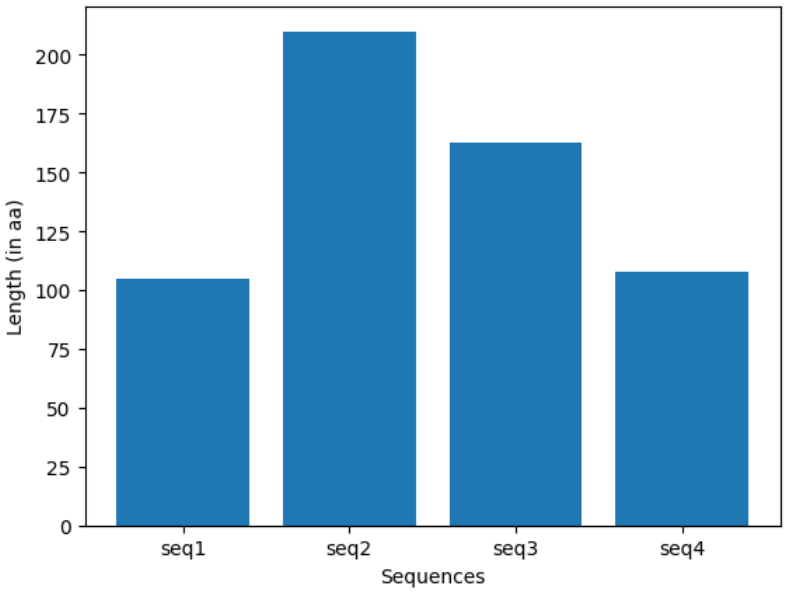


2.13. Barplot showing per sequence length

```
PROT.length_plot(fasta_records)
```

Output: generate barplot showing per sequence length distribution

In [39…
```
# Per sequence length
PROT.length_plot(data)
```

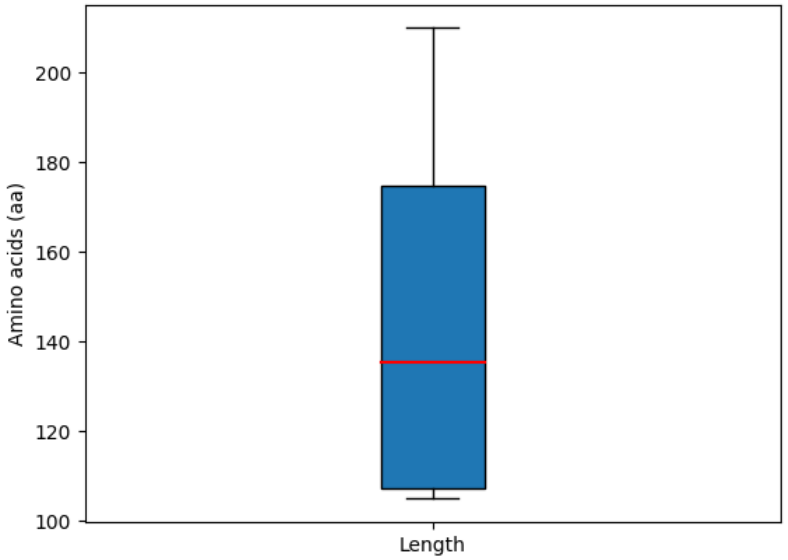


2.14. Boxplot showing overall length distribution

```
>>> PROT.length_distribution_plot(fasta_records)
```

Output: generate boxplot showing overall length distribution across all the sequences

In [40…
```
PROT.length_distribution_plot(data)
```



2.15. Converting sequence object to feature matrix

```
>>> from seqplotter_analysis import seq2feature
>>> feature_matrix = seq2feature(fasta_records) # with default options (k=3, min_sample=1)

>>> feature_matrix = seq2feature(fasta_records, 6, 2) # with (k=6, min_sample=2)
```

Options:

- k: kmer size
- min_sample: threshold of considering a k-mer as feature

Output: returns a dict object containing following keys ("matrix", "sample", "kmers"), where "kmers" are columns and "sample" are rows

< *dict{"matrix": numpy_array, "sample": seqID_list, "kmers": kmers_list}* >

In [41…
```
# Importing module
from seqplotter_analysis import seq2feature

# Converting sequence to feature matrix
feature_matrix = seq2feature(data)

# Visualizing feature matrix
import pandas as pd
pd.DataFrame(feature_matrix["matrix"], index=feature_matrix["sample"], columns=feature_matrix["kmers"])
```

Out[41…

	RLL	GDL	DLS	GNP	GKK	DGL	LKG	KGT	FWK	AGA	...	DAG	PGY	GYP	YPG	GAK	AKG	GFK	FKG	GRD	RDG
seq1	1	1	1	1	1	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	0
seq2	1	1	0	1	0	0	1	0	2	1	...	0	0	0	0	0	0	0	0	0	0
seq3	0	0	0	1	1	1	1	1	0	1	...	2	2	1	1	0	0	0	0	0	0
seq4	0	0	1	0	0	0	1	0	0	0	...	0	0	2	2	2	2	2	2	2	2

4 rows × 84 columns

2.16 Principal component analysis

```
>>> from seqplotter_analysis import PCA
>>> pca_matrix = PCA(feature_matrix) # return pca matrix with first 2 PCs

>>> pca_matrix = PCA(feature_matrix, 3) # return pca matrix with first 3 PCs
```

Output: returns a dict object containing following keys ("matrix", "sample", "components"), where "components" are columns and "sample" are rows

< *dict{"matrix": numpy_array, "sample": seqID_list, "components": PCA_components}* >

In [42…
```
# Importing module
from seqplotter_analysis import PCA

# Performing PCA
```

```
pca_matrix = PCA(feature_matrix)

# Visualizing PCA matrix
import pandas as pd
pd.DataFrame(pca_matrix["matrix"], index=pca_matrix["sample"], columns=pca_matrix["components"])
```

Out[42…

	PC-1	PC-2
seq1	5.069106	-3.616282
seq2	2.496495	-2.811089
seq3	-11.398262	-0.046961
seq4	3.832661	6.474332