Decoding Protein-Protein Interactions: A Model Performance Comparison

ABSTRACT

Protein-protein interactions (PPIs) play a crucial role in various biological processes, influencing cell signaling, metabolic pathways, and disease mechanisms. Understanding and classifying these interactions into positive and negative categories can significantly impact drug discovery and systems biology research. This study explores the classification of PPIs using multiple machine learning models, including Logistic Regression, Decision Tree, Random Forest, and XGBoost. The dataset was sourced from Kaggle, containing separate files for positive and negative protein interactions. The primary objective was to evaluate model performance in distinguishing between these two classes.

The dataset underwent preprocessing, including data normalization, feature selection, and train-test splitting. Logistic Regression, Decision Tree, and Random Forest achieved a perfect classification accuracy of 100%, with precision, recall, and F1-score of 1.00 for both classes. Such performance, however, raises concerns about potential overfitting, suggesting a need for further validation on external datasets. Conversely, the XGBoost model performed poorly, with an accuracy of only 50% and failed to classify negative interactions properly. This study highlights the effectiveness of classical machine learning models in classifying PPIs while underlining the need for robust validation techniques to ensure real-world applicability. Future work should focus on deep learning approaches and feature engineering to enhance classification robustness.

Keywords: Protein-Protein Interactions, Machine Learning, Logistic Regression, Random Forest, Overfitting, XGBoost

Introduction

1.1 Proteins: The Fundamental Building Blocks of Life Proteins are macromolecules essential for life, performing a vast array of functions within biological systems. These biomolecules are composed of amino acids linked by peptide bonds and are responsible

for numerous structural, enzymatic, and regulatory functions in cells (Nelson & Cox, 2021). Proteins are involved in catalyzing biochemical reactions, maintaining cellular structures, transporting molecules, and facilitating communication between cells. They are indispensable in nearly all physiological processes, from metabolism and immunity to growth and development (Berg et al., 2015).

Proteins exhibit remarkable diversity in structure and function, making them crucial for sustaining life. Their role extends beyond basic cellular functions to complex biological processes such as signal transduction, immune responses, and DNA replication. The study of protein interactions, particularly protein-protein interactions (PPIs), has gained significance in biotechnology, molecular biology, and drug discovery, as understanding these interactions provides insights into cellular mechanisms and disease pathology (Alberts et al., 2019).

1.2 Structure and Composition of Proteins Proteins are composed of 20 standard amino acids, each possessing a central carbon atom (α-carbon), an amino group (-NH_2), a carboxyl group (-COOH), and a distinctive side chain (R-group) that determines the amino acid's properties (Voet & Voet, 2018). The sequence and arrangement of these amino acids dictate the protein's final structure and function. Protein structures are classified into four levels:

Primary Structure: The linear sequence of amino acids in a polypeptide chain, determined by genetic coding. Secondary Structure: Localized folding patterns, such as α-helices and β-sheets, stabilized by hydrogen bonds. Tertiary Structure: The three-dimensional conformation formed by interactions between secondary structures, including hydrophobic interactions, disulfide bonds, and van der Waals forces. Quaternary Structure: The assembly of multiple polypeptide chains (subunits) into a functional protein complex (Nelson & Cox, 2021). 1.3 Properties of Proteins Proteins exhibit several fundamental properties that define their structure, function, and stability in biological systems:

1.3.1 Solubility and Hydrophobicity The solubility of proteins depends on the nature of their amino acid composition. Hydrophilic proteins dissolve easily in aqueous environments, while hydrophobic proteins tend to aggregate or integrate into lipid membranes. The hydrophobic effect plays a crucial role in protein folding and stability, ensuring that nonpolar amino acid residues remain buried inside the protein structure while polar residues interact with the aqueous environment (Branden & Tooze, 2012).

1.3.2 Denaturation and Stability Proteins are sensitive to changes in temperature, pH, and chemical environments. Denaturation refers to the loss of a protein's functional three-dimensional structure due to external stressors such as heat, acids, or detergents. Denatured proteins often lose their biological activity, but some can regain functionality through renaturation if the denaturing agent is removed (Nelson & Cox, 2021).

1.3.3 Catalytic Activity One of the most vital functions of proteins is acting as enzymes to catalyze biochemical reactions. Enzymatic proteins lower activation energy, accelerating metabolic processes essential for life. For example, DNA polymerase facilitates DNA replication, while amylase aids in carbohydrate digestion (Berg et al., 2015).

1.3.4 Specificity and Binding Affinity Proteins exhibit a high degree of specificity for their biological targets. This property is crucial for protein-protein interactions, enzyme-substrate recognition, and antigen-antibody interactions. The binding affinity of proteins is determined by molecular complementarity, hydrogen bonding, electrostatic interactions, and van der Waals forces (Voet & Voet, 2018).

1.3.5 Structural and Mechanical Properties Some proteins serve as structural components within cells and tissues. For instance, collagen provides tensile strength to connective tissues, keratin forms protective structures such as hair and nails, and actin and myosin facilitate muscle contraction. These proteins contribute to the mechanical integrity and mobility of organisms (Alberts et al., 2019).

1.3.6 Role in Cellular Communication Proteins are critical in cell signaling and communication. Hormonal proteins such as insulin regulate glucose metabolism, while receptor proteins like G-protein coupled receptors (GPCRs) mediate responses to extracellular signals. These functions are vital for maintaining homeostasis and coordinating cellular activities (Nelson & Cox, 2021).

1.4 Importance of Protein-Protein Interactions (PPIs) Protein-protein interactions are essential for virtually all biological functions, from enzyme-substrate binding to immune system responses. These interactions determine cellular responses, signaling pathways, and metabolic regulation. Disruptions in PPIs have been linked to various diseases, including cancer, neurodegenerative disorders, and infectious diseases (Zhou et al., 2022).

Understanding PPIs helps in drug design and therapeutic development, as targeting specific protein interactions can modulate disease progression. Machine learning models have become powerful tools in identifying and classifying PPIs, providing insights into protein function and interaction networks (Sureja, 2023). This study focuses on classifying positive and negative PPIs using machine learning approaches to enhance our understanding of biological interactions and their implications in biotechnology and medicine.

Problem Statement

Proteins play a crucial role in biological processes, interacting with each other to regulate cellular functions. Protein-protein interactions (PPIs) are fundamental to many physiological and pathological mechanisms, making their identification and classification essential for advancements in biotechnology, drug discovery, and disease research. However, distinguishing between positive PPIs (functionally relevant interactions) and negative PPIs (non-interacting protein pairs) remains a significant challenge due to the complexity of protein structures and interaction dynamics.

Traditional experimental methods such as yeast two-hybrid screening and co-immunoprecipitation are time-consuming, expensive, and often produce false positives or negatives. As an alternative, machine learning models provide an efficient computational approach to classify PPIs based on large datasets. In this study, a dataset containing positive and negative PPIs was analyzed using machine learning models—Logistic Regression, Decision Tree, Random Forest, and XGBoost—to determine the most effective model for classification.

The results indicate that while Logistic Regression, Decision Tree, and Random Forest achieved 100% accuracy, potentially overfitting, XGBoost failed to classify class 1 properly. This raises concerns about the generalizability of models and highlights the need for further validation on unseen datasets. Addressing these challenges will improve the accuracy and reliability of computational PPI classification in biological research.

library used

pandas is a powerful data manipulation and analysis library in Python. It provides data structures like DataFrame and Series to efficiently handle and manipulate large datasets, and it offers tools for merging, reshaping, selecting, and cleaning data. numpy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays.matplotlib is a widely-used plotting library for Python. pyplot is a module in matplotlib used for creating static, animated, and interactive visualizations in Python.seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.itertools is a standard Python library providing functions that create iterators for efficient looping. It includes functions for creating permutations, combinations, and Cartesian products.Bio is part of the Biopython library, which provides tools for biological computation. It includes modules for reading and writing sequence data, working with bioinformatics file formats, and conducting various sequence-based analyses.warnings is a Python module used for issuing warning messages. It allows the

programmer to control how warnings are displayed and to suppress them if necessary.This module from Biopython's SeqUtils package provides tools for computing various properties of proteins, such as molecular weight, amino acid composition, and hydrophobicity.sklearn.model_selection is part of the Scikit-learn library, providing utilities for splitting datasets into training and test sets, performing cross-validation, and tuning hyperparameters.sklearn.linear_model provides a set of linear models, such as linear regression and logistic regression, used for modeling data with linear relationships.sklearn.tree contains decision tree algorithms used for classification and regression tasks. It includes tools for growing decision trees, pruning them, and visualizing the tree.sklearn.ensemble provides ensemble methods like Random Forests and Gradient Boosting, which combine multiple models to improve performance and accuracy.sklearn.preprocessing provides techniques for scaling, normalizing, and transforming data before using it in machine learning models, such as StandardScaler and MinMaxScaler.xgboost is an optimized gradient boosting library designed for speed and performance. It is used for building decision tree-based models and is highly effective for classification and regression tasks.

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import Bio
import warnings
from Bio.SeqUtils import ProtParam
warnings.filterwarnings("ignore")
```

Data Collection and Cleaning

The dataset for this project was collected from two reliable sources:

PPI Dataset Overview

This dataset provides a collection of protein-protein interaction (PPI) data, which is essential for understanding how proteins interact with each other to carry out cellular functions. The dataset includes interaction information between proteins from various species, making it useful for exploring biological networks and predicting protein behavior in various conditions.

Data Source:

The dataset can be accessed at the following link: PPI Dataset on Kaggle

- **protein_sequences_1**: The amino acid sequence of the first protein in the pair.
- **protein_sequences_2**: The amino acid sequence of the second protein in the pair.
- **proteins**: A combined representation of both protein sequences, typically formatted as a tuple or list indicating the two proteins being compared.
- **Sequence_Length_Difference**: The difference in length between the two protein sequences (i.e., the absolute value of the length difference between `protein_sequences_1` and `protein_sequences_2`).
- **Average_Hydrophobicity**: The average hydrophobicity of the protein sequences. Hydrophobicity refers to the tendency of amino acids to avoid water and interact with each other in the protein's interior.
- **Charge_Polarity_Ratio**: The ratio of charged amino acids (positively and negatively charged) to neutral amino acids in the protein sequences. It gives insight into the protein's electrostatic properties.
- **Aliphatic_Index**: A measure of the protein's stability, calculated based on the aliphatic (non-polar) amino acids. A higher aliphatic index indicates greater stability at higher temperatures.
- **Sequence_Similarity_Score**: A measure of the similarity between the two protein sequences, typically based on alignment algorithms (e.g., BLAST, Smith-Waterman).
- **Hydrogen_Bond_Ratio**: The ratio of hydrogen bonds formed in the protein structure, indicating its structural stability and potential for protein-protein interaction.
- **Cysteine_Count**: The number of cysteine residues in the protein sequences. Cysteine is important for forming disulfide bonds that contribute to protein stability.
- **Amino_Acid_Diversity**: A measure of the diversity of amino acids present in the protein sequences. Higher diversity might indicate structural or functional flexibility.
- **Avg_Molecular_Weight**: The average molecular weight of the amino acids in the protein sequence. It reflects the size of the protein.
- **pH_Stability_Index**: A measure of the protein's stability under various pH conditions. This index can give insight into whether the protein is more stable in acidic or basic environments.

- **Aromaticity**: The proportion of aromatic amino acids (like phenylalanine, tyrosine, and tryptophan) in the protein sequence. Aromatic amino acids are often involved in protein stability and ligand binding.
- **Helix_Percentage**: The percentage of the protein's structure that is composed of alpha helices. This gives an indication of the secondary structure.
- **Turn_Percentage**: The percentage of the protein's structure that forms turns (i.e., irregular secondary structures).
- **Sheet_Percentage**: The percentage of the protein's structure that is made up of beta sheets. It is another component of the protein's secondary structure.
- **protein1_stretch**: A measure of how much of the first protein's sequence stretches or aligns with regions of high similarity to other proteins, which can indicate regions of evolutionary significance.
- **protein2_stretch**: Similar to `protein1_stretch`, but for the second protein in the pair. It shows how much of the second protein aligns with other proteins.

In [15]:
```
amino_acids_posi = pd.read_csv(r"C:\Users\manoj\Downloads\Protein Protein Interaction Dataset\positive_protein_sequenc
```

In [16]:
```
amino_acids_neg = pd.read_csv(r"C:\Users\manoj\Downloads\Protein Protein Interaction Dataset\negative_protein_sequence
```

In [17]:
```
amino_acids_posi.head()
amino_acids_posi.head()
```

Out[17]:

	protein_sequences_1	protein_sequences_2
0	MESSKKMDSPGALQTNPPLKLHTDRSAGTPVFVPEQGGYKEKFVKT...	MARPHPWWLCVLGTLVGLSATPAPKSCPERHYWAQGKLCCQMCEPG...
1	MVMSSYMVNSKYVDPKFPPCEEYLQGGYLGEQGADYYGGGAQGADF...	MAENVVEPGPPSAKRPKLSSPALSASASDGTDFGSLFDLEHDLPDE...
2	MNRHLWKSQLCEMVQPSGGPAADQDVLGEESPLGKPAMLHLPSEQG...	MEGGRRARVVIESKRNFFLGAFPTPFPAEHVELGRLGDSETAMVPG...
3	MAPPSTREPRVLSATSATKSDGEMVLPGFPDADSFVKFALGSVVAV...	MLFYSFFKSLVGKDVVVELKNDLSICGTLHSVDQYLNIKLTDISVT...
4	MQSGPRPPLPAPGLALALTLTMLARLASAASFFGENHLEVPVATAL...	MQTIKCVVVGDGAVGKTCLLISYTTNKFPSEYVPTVFDNYAVTVMI...

In [18]:
```
amino_acids_posi.tail()
amino_acids_posi.tail()
```

Out[18]:

	protein_sequences_1	protein_sequences_2
36625	METQFRRGGLGCSPASIKRKKKREDSGDFGLQVSTMFSEDDFQSTE...	MFYGTHFIMSPPTKSKLKRQSQLLSSMLSRTLSYKYRDLDSTFSSL...
36626	MDAKARNCLLQHREALEKDIKTSYIMDHMISDGFLTISEEEKVRNE...	MSQSNRELVVDFLSYKLSQKGYSWSQFSDVEENRTEAPEGTESEME...
36627	MSLLCVRVKRAKFQGSPDKFNTYVTLKVQNVKSTTVAVRGDQPSWE...	MADTIFGSGNDQWVCPNDRQLALRAKLQTGWSVHTYQTEKQRRKQH...
36628	MERRRITSAARRSYVSSGEMMVGGLAPGRRLGPGTRLSLARMPPPL...	MDLHKQWENTETNWHKEKMELLDQFDNERKEWESQWKIMQKKIEEL...
36629	METPLDVLSRAASLVHADDEKREAALRGEPRMQTLPVASALSSHRT...	MERMSDSADKPIDNDAEGVWSPDIEQSFQEALAIYPPCGRRKIILS...

In [19]:
```
amino_acids_neg.head()
amino_acids_neg.head()
```

Out[19]:

	protein_sequences_1	protein_sequences_2
0	MSVEMDSSSFIQFDVPEYSSTVLSQLNELRLQGKLCDIIVHIQGQP...	MGDTFIRHIALLGFEKRFVPSQHYVYMFLVKWQDLSEKVVYRRFTE...
1	MPITRMRMRPWLEMQINSNQIPGLIWINKEEMIFQIPWKHAAKHGW...	MTMPVNGAHKDADLWSSHDKMLAQPLKDSDVEVYNIIKKESNRQRV...
2	MLCVRGARLKRELDATATVLANRQDESEQSRKRLIEQSREFKKNTP...	MRLTLLCCTWREERMGEEGSELPVCASCGQRIYDGQYLQALNADWH...
3	MDALESLLDEVALEGLDGLCLPALWSRLETRVPPFPLPLEPCTQEF...	MERLQKQPLTSPGSVSPSRDSSVPGSPSSIVAKMDNQVLGYKDLAA...
4	MALSRGLPRELAEAVAGGRVLVVGAGGIGCELLKNLVLTGFSHIDL...	MVVMNSLRVILQASPGKLLWRKFQIPRFMPARPCSLYTCTYKTRNR...

In [20]:
```
amino_acids_neg.tail()
amino_acids_neg.tail()
```

Out[20]:

	protein_sequences_1	protein_sequences_2
36475	MERFVVTAPPARNRSKTALYVTPLDRVTEFGGELHEDGGKLFCTSC...	MLGMIKNSLFGSVETWPWQVLSKGDKEEVAYEERACEGGKFATVEV...
36476	MAAGKSGGSAGEITFLEALARSESKRDGGFKNNWSFDHEEESEGDT...	MSSASGLRRGHPAGGEENMTETDAFYKREMFDPAEKYKMDHRRRGI...
36477	MQAQQYQQQRRKFAAAFLAFIFILAAVDTAEAGKKEKPEKKVKKSD...	MPRGRKSRRRRNARAAEENRNNRKIQASEASETPMAASVVASTPED...
36478	MNQPQRMAPVGTDKELSDLLDFSMMFPLPVTNGKGRPASLAGAQFG...	MVPALRYLVGACGRARGRFAGGSPGACGFASGRPRPLCGGSRSAST...
36479	MSATGPISNYYVDSLISHDNEDLLASRFPATGAHPAAARPSGLVPD...	MSDYENDDECWSVLEGFRVTLTSVIDPSRITPYLRQCKVLNPDDEE...

In [21]:
```
amino_acids_posi.columns
```

Out[21]:
```
Index(['protein_sequences_1', 'protein_sequences_2'], dtype='object')
```

In [22]:
```
amino_acids_neg.columns
```

Out[22]:
```
Index(['protein_sequences_1', 'protein_sequences_2'], dtype='object')
```

In [23]:
```
amino_acids_posi["proteins"] = 1
amino_acids_neg ["proteins"] = 0
protein_df = pd.concat([amino_acids_posi,amino_acids_neg ], axis=0, ignore_index=True)
```

In [24]:
```
protein_df
```

Out[24]:

	protein_sequences_1	protein_sequences_2
0	MESSKKMDSPGALQTNPPLKLHTDRSAGTPVFVPEQGGYKEKFVKT...	MARPHPWWLCVLGTLVGLSATPAPKSCPERHYWAQGKLCCQMCEPG...
1	MVMSSYMVNSKYVDPKFPPCEEYLQGGYLGEQGADYYGGGAQGADF...	MAENVVEPGPPSAKRPKLSSPALSASASDGTDFGSLFDLEHDLPDE...
2	MNRHLWKSQLCEMVQPSGGPAADQDVLGEESPLGKPAMLHLPSEQG...	MEGGRRARVVIESKRNFFLGAFPTPFPAEHVELGRLGDSETAMVPG...
3	MAPPSTREPRVLSATSATKSDGEMVLPGFPDADSFVKFALGSVVAV...	MLFYSFFKSLVGKDVVVELKNDLSICGTLHSVDQYLNIKLTDISVT...
4	MQSGPRPPLPAPGLALALTLTMLARLASAASFFGENHLEVPVATAL...	MQTIKCVVVGDGAVGKTCLLISYTTNKFPSEYVPTVFDNYAVTVMI...
...	...	...
73105	MERFVVTAPPARNRSKTALYVTPLDRVTEFGGELHEDGGKLFCTSC...	MLGMIKNSLFGSVETWPWQVLSKGDKEEVAYEERACEGGKFATVEV...
73106	MAAGKSGGSAGEITFLEALARSESKRDGGFKNNWSFDHEEESEGDT...	MSSASGLRRGHPAGGEENMTETDAFYKREMFDPAEKYKMDHRRRGI...
73107	MQAQQYQQQRRKFAAAFLAFIFILAAVDTAEAGKKEKPEKKVKKSD...	MPRGRKSRRRRNARAAEENRNNRKIQASEASETPMAASVVASTPED...
73108	MNQPQRMAPVGTDKELSDLLDFSMMFPLPVTNGKGRPASLAGAQFG...	MVPALRYLVGACGRARGRFAGGSPGACGFASGRPRPLCGGSRSAST...
73109	MSATGPISNYYVDSLISHDNEDLLASRFPATGAHPAAARPSGLVPD...	MSDYENDDECWSVLEGFRVTLTSVIDPSRITPYLRQCKVLNPDDEE...

73110 rows × 3 columns

*** Since both datasets have the same column names , concatenate them vertically**

*** axis = 0 tells adding more rows below via axis = 1 tells adding columns**

***ignore_index=True shows Resets index after merging (If you don’t need to keep the original row indices)**

***ignore_index=False shows Keeps the original index values (If you want to retain original index information)**

In [26]:
```
protein_df.shape
protein_df.shape[0]
```

Out[26]: 73110

no.of rows = 73110

In [28]:
```
protein_df.shape[1]
```

Out[28]: 3

no.of columns = 3

exploratory data analysis (EDA)

There are 20 amino acids that make up proteins. These amino acids are used by all living things to make proteins.

Essential amino acids

histidine, isoleucine, leucine, lysine, methionine, phenylalanine, threonine, tryptophan, and valine.

Non-essential amino acids.

The other 11 amino acids that the body can synthesize Alanine,Arginine Asparagine, Aspartic acid ,Cysteine, Glutamic acid,Glutamine,Glycine,Proline,Serine,Tyrosine.

In [32]:
```
amino_acids = {"A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P",
               "Q", "R", "S", "T", "V", "W", "Y"}
```

Standard Amino Acids

Single-Letter Code	Amino Acid Name
A	Alanine

Single-Letter Code	Amino Acid Name
C	Cysteine
D	Aspartic Acid
E	Glutamic Acid
F	Phenylalanine
G	Glycine
H	Histidine
I	Isoleucine
K	Lysine
L	Leucine
M	Methionine
N	Asparagine
P	Proline
Q	Glutamine
R	Arginine
S	Serine
T	Threonine
V	Valine
W	Tryptophan
Y	Tyrosine

In [34]:
```
def count_non_standard(sequence):
    """Check for non-standard amino acids in a protein sequence."""
```

```
    sequence_set = set(sequence)
    non_standard_amino_acids = sequence_set - amino_acids
    if non_standard_amino_acids:
        print(f"Non-standard amino acids in sequence: {non_standard_amino_acids}")
    return len(non_standard_amino_acids) > 0
```

This function checks if a given protein sequence contains any non-standard amino acids, which are amino acids that are not part of the 20 standard ones. It first converts the sequence into a set, allowing it to identify all unique amino acids present. Then, it compares this set with the standard amino acids to find any that do not belong to the standard list. If any non-standard amino acids are found, they are printed. Finally, the function returns True if non-standard amino acids are present, otherwise, it returns False

In [36]:
```
non_standard_count = sum(count_non_standard(seq) for seq in protein_df['protein_sequences_1'])

print("Number of sequences with non-standard amino acids:", non_standard_count)
```

```
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Number of sequences with non-standard amino acids: 3
```

In [37]:
```
count_non_standard2 = sum(count_non_standard(seq) for seq in protein_df['protein_sequences_2'])

print("Number of sequences with non-standard amino acids:", count_non_standard2)
```

```
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
```

```
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
```

```
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
```

```
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
```

```
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Non-standard amino acids in sequence: {'U'}
Number of sequences with non-standard amino acids: 162
```

The amino acid represented by "U" is Selenocysteine.Selenocysteine is a non-standard amino acid that contains selenium instead of sulfur. It's the 21st amino acid and is encoded by the codon UGA, which normally functions as a stop codon.It's chemically similar to cysteine, but it provides a selective advantage for particular redox enzymatic functions.

In [39]:
```python
# Define amino acid properties
hydrophobicity = {
    'A': 1.8, 'C': 2.5, 'D': -3.5, 'E': -3.5, 'F': 2.8, 'G': -0.4,
    'H': -3.2, 'I': 4.5, 'K': -3.9, 'L': 3.8, 'M': 1.9, 'N': -3.5,
    'P': -1.6, 'Q': -3.5, 'R': -4.5, 'S': -0.8, 'T': -0.7, 'V': 4.2,
    'W': -0.9, 'Y': -1.3
}

positive_charged = {'R', 'K', 'H'}
negative_charged = {'D', 'E'}
aliphatic_residues = {'A', 'V', 'I', 'L'}
hydrogen_bond_donors_acceptors = {'S', 'T', 'Y', 'H'}
ph_sensitive_residues = {'H', 'D', 'E', 'K', 'R'}

# Function to compute features
def compute_features(seq1, seq2):
    if not seq1 or not seq2:
        return [0] * 13  # Return zeroed features if any sequence is missing
```

```
    # Sequence length difference
    seq_length_diff = abs(len(seq1) - len(seq2))

    # Average hydrophobicity
    avg_hydro = (sum(hydrophobicity.get(aa, 0) for aa in seq1) / len(seq1) +
                 sum(hydrophobicity.get(aa, 0) for aa in seq2) / len(seq2)) / 2

    # Charge polarity ratio
    pos1 = sum(1 for aa in seq1 if aa in positive_charged)
    neg1 = sum(1 for aa in seq1 if aa in negative_charged)
    pos2 = sum(1 for aa in seq2 if aa in positive_charged)
    neg2 = sum(1 for aa in seq2 if aa in negative_charged)
    charge_ratio = (pos1 + pos2) / (neg1 + neg2 + 1e-6)  # Avoid division by zero

    # Aliphatic index
    aliphatic_index = (sum(1 for aa in seq1 if aa in aliphatic_residues) +
                       sum(1 for aa in seq2 if aa in aliphatic_residues)) / (len(seq1) + len(seq2) + 1e-6)

    # Sequence similarity score
    min_length = min(len(seq1), len(seq2))
    similarity_score = sum(1 for i in range(min_length) if seq1[i] == seq2[i]) / (min_length + 1e-6)

    # Hydrogen bond donor/acceptor ratio
    hbond_ratio = (sum(1 for aa in seq1 if aa in hydrogen_bond_donors_acceptors) +
                   sum(1 for aa in seq2 if aa in hydrogen_bond_donors_acceptors)) / (len(seq1) + len(seq2) + 1e-6)

    # Cysteine count
    cys_count = seq1.count('C') + seq2.count('C')

    # Amino acid diversity score
    diversity_score = len(set(seq1 + seq2))

    # Average molecular weight
    mw1 = ProtParam.ProteinAnalysis(seq1).molecular_weight() / len(seq1)
    mw2 = ProtParam.ProteinAnalysis(seq2).molecular_weight() / len(seq2)
    avg_molecular_weight = (mw1 + mw2) / 2

    # pH stability index
```

```
    ph_sensitive_count = sum(1 for aa in seq1 if aa in ph_sensitive_residues) + \
                         sum(1 for aa in seq2 if aa in ph_sensitive_residues)

    # Aromaticity feature
    aromaticity = (ProtParam.ProteinAnalysis(seq1).aromaticity() +
                   ProtParam.ProteinAnalysis(seq2).aromaticity()) / 2

    # Secondary structure percentages (Helix, Turn, Sheet)
    sec_struct1 = ProtParam.ProteinAnalysis(seq1).secondary_structure_fraction()
    sec_struct2 = ProtParam.ProteinAnalysis(seq2).secondary_structure_fraction()
    helix_percent = (sec_struct1[0] + sec_struct2[0]) / 2
    turn_percent = (sec_struct1[1] + sec_struct2[1]) / 2
    sheet_percent = (sec_struct1[2] + sec_struct2[2]) / 2

    return [seq_length_diff, avg_hydro, charge_ratio, aliphatic_index, similarity_score,
            hbond_ratio, cys_count, diversity_score, avg_molecular_weight, ph_sensitive_count,
            aromaticity, helix_percent, turn_percent, sheet_percent]

# Apply the function to the dataset
feature_columns = ["Sequence_Length_Difference", "Average_Hydrophobicity", "Charge_Polarity_Ratio",
                   "Aliphatic_Index", "Sequence_Similarity_Score", "Hydrogen_Bond_Ratio", "Cysteine_Count",
                   "Amino_Acid_Diversity", "Avg_Molecular_Weight", "pH_Stability_Index",
                   "Aromaticity", "Helix_Percentage", "Turn_Percentage", "Sheet_Percentage"]
# Compute features for each row
protein_df[feature_columns] = protein_df.apply(
    lambda row: pd.Series(compute_features(row["protein_sequences_1"], row["protein_sequences_2"])), axis=1
)

# Adding sequence stretch
protein_df["protein1_stretch"] = protein_df["protein_sequences_1"].str.len()
protein_df["protein2_stretch"] = protein_df["protein_sequences_2"].str.len()

# Display the updated dataframe
print(protein_df.head())
```

```
                                protein_sequences_1  \
0  MESSKKMDSPGALQTNPPLKLHTDRSAGTPVFVPEQGGYKEKFVKT...
1  MVMSSYMVNSKYVDPKFPPCEEYLQGGYLGEQGADYYGGGAQGADF...
2  MNRHLWKSQLCEMVQPSGGPAADQDVLGEESPLGKPAMLHLPSEQG...
3  MAPPSTREPRVLSATSATKSDGEMVLPGFPDADSFVKFALGSVVAV...
4  MQSGPRPPLPAPGLALALTLTMLARLASAASFFGENHLEVPVATAL...

                                protein_sequences_2  proteins  \
0  MARPHPWWLCVLGTLVGLSATPAPKSCPERHYWAQGKLCCQMCEPG...         1
1  MAENVVEPGPPSAKRPKLSSPALSASASDGTDFGSLFDLEHDLPDE...         1
2  MEGGRRARVVIESKRNFFLGAFPTPFPAEHVELGRLGDSETAMVPG...         1
3  MLFYSFFKSLVGKDVVVELKNDLSICGTLHSVDQYLNIKLTDISVT...         1
4  MQTIKCVVVGDGAVGKTCLLISYTTNKFPSEYVPTVFDNYAVTVMI...         1

   Sequence_Length_Difference  Average_Hydrophobicity  Charge_Polarity_Ratio  \
0                       308.0               -0.486667               1.408163
1                      2159.0               -0.795597               1.595960
2                       119.0               -0.679034               0.925000
3                       790.0               -0.359423               1.238095
4                      2131.0               -0.157130               0.950178

   Aliphatic_Index  Sequence_Similarity_Score  Hydrogen_Bond_Ratio  \
0         0.240338                   0.053846             0.194444
1         0.208692                   0.062745             0.182840
2         0.266458                   0.081146             0.142111
3         0.272449                   0.084211             0.161224
4         0.322324                   0.083770             0.179467

   Cysteine_Count  Amino_Acid_Diversity  Avg_Molecular_Weight  \
0            49.0                  20.0            112.838286
1            55.0                  20.0            109.390215
2            24.0                  20.0            114.981923
3            16.0                  20.0            113.989214
4            18.0                  20.0            109.597951

   pH_Stability_Index  Aromaticity  Helix_Percentage  Turn_Percentage  \
0               236.0     0.067186          0.303629         0.279076
1               514.0     0.066018          0.265286         0.371210
```

```
2              308.0     0.057335          0.399857          0.230763
3              282.0     0.083348          0.330211          0.263039
4              548.0     0.074831          0.314328          0.282699

   Sheet_Percentage  protein1_stretch  protein2_stretch
0          0.290588               568               260
1          0.242115               255              2414
2          0.284633               419               538
3          0.362890               885                95
4          0.381837              2322               191
```

In [40]:
```
protein_df.head()
```

Out[40]:

	protein_sequences_1	protein_sequences_2	prote
0	MESSKKMDSPGALQTNPPLKLHTDRSAGTPVFVPEQGGYKEKFVKT...	MARPHPWWLCVLGTLVGLSATPAPKSCPERHYWAQGKLCCQMCEPG...	
1	MVMSSYMVNSKYVDPKFPPCEEYLQGGYLGEQGADYYGGGAQGADF...	MAENVVEPGPPSAKRPKLSSPALSASASDGTDFGSLFDLEHDLPDE...	
2	MNRHLWKSQLCEMVQPSGGPAADQDVLGEESPLGKPAMLHLPSEQG...	MEGGRRARVVIESKRNFFLGAFPTPFPAEHVELGRLGDSETAMVPG...	
3	MAPPSTREPRVLSATSATKSDGEMVLPGFPDADSFVKFALGSVVAV...	MLFYSFFKSLVGKDVVVELKNDLSICGTLHSVDQYLNIKLTDISVT...	
4	MQSGPRPPLPAPGLALALTLTMLARLASAASFFGENHLEVPVATAL...	MQTIKCVVVGDGAVGKTCLLISYTTNKFPSEYVPTVFDNYAVTVMI...	

In [41]:
```
protein_df.columns
```

Out[41]:
```
Index(['protein_sequences_1', 'protein_sequences_2', 'proteins',
       'Sequence_Length_Difference', 'Average_Hydrophobicity',
       'Charge_Polarity_Ratio', 'Aliphatic_Index', 'Sequence_Similarity_Score',
       'Hydrogen_Bond_Ratio', 'Cysteine_Count', 'Amino_Acid_Diversity',
       'Avg_Molecular_Weight', 'pH_Stability_Index', 'Aromaticity',
       'Helix_Percentage', 'Turn_Percentage', 'Sheet_Percentage',
       'protein1_stretch', 'protein2_stretch'],
      dtype='object')
```

In [42]:
```
from Bio.SeqUtils.ProtParam import ProteinAnalysis
```

In [43]:
```
def aa_comp(seq):
    amino_acid_composition = {}
    amino_acids = list("ACDEFGHIKLMNPQRSTVWYU")  # Standard amino acids + Selenocysteine (U)

    for aa in amino_acids:
        amino_acid_composition[aa] = round(float(seq.count(aa)) / len(seq) * 100, 3)

    return amino_acid_composition

protein_df["aa_composition"] = protein_df["protein_sequences_1"].apply(aa_comp)
protein_df["aa_composition"] = protein_df["protein_sequences_2"].apply(aa_comp)

print(protein_df.head())
```

	protein_sequences_1
0	MESSKKMDSPGALQTNPPLKLHTDRSAGTPVFVPEQGGYKEKFVKT...
1	MVMSSYMVNSKYVDPKFPPCEEYLQGGYLGEQGADYYGGGAQGADF...
2	MNRHLWKSQLCEMVQPSGGPAADQDVLGEESPLGKPAMLHLPSEQG...
3	MAPPSTREPRVLSATSATKSDGEMVLPGFPDADSFVKFALGSVVAV...
4	MQSGPRPPLPAPGLALALTLTMLARLASAASFFGENHLEVPVATAL...

	protein_sequences_2	proteins
0	MARPHPWWLCVLGTLVGLSATPAPKSCPERHYWAQGKLCCQMCEPG...	1
1	MAENVVEPGPPSAKRPKLSSPALSASASDGTDFGSLFDLEHDLPDE...	1
2	MEGGRRARVVIESKRNFFLGAFPTPFPAEHVELGRLGDSETAMVPG...	1
3	MLFYSFFKSLVGKDVVVELKNDLSICGTLHSVDQYLNIKLTDISVT...	1
4	MQTIKCVVVGDGAVGKTCLLISYTTNKFPSEYVPTVFDNYAVTVMI...	1

	Sequence_Length_Difference	Average_Hydrophobicity	Charge_Polarity_Ratio
0	308.0	-0.486667	1.408163
1	2159.0	-0.795597	1.595960
2	119.0	-0.679034	0.925000
3	790.0	-0.359423	1.238095
4	2131.0	-0.157130	0.950178

	Aliphatic_Index	Sequence_Similarity_Score	Hydrogen_Bond_Ratio
0	0.240338	0.053846	0.194444
1	0.208692	0.062745	0.182840
2	0.266458	0.081146	0.142111
3	0.272449	0.084211	0.161224
4	0.322324	0.083770	0.179467

	Cysteine_Count	Amino_Acid_Diversity	Avg_Molecular_Weight
0	49.0	20.0	112.838286
1	55.0	20.0	109.390215
2	24.0	20.0	114.981923
3	16.0	20.0	113.989214
4	18.0	20.0	109.597951

	pH_Stability_Index	Aromaticity	Helix_Percentage	Turn_Percentage
0	236.0	0.067186	0.303629	0.279076
1	514.0	0.066018	0.265286	0.371210

```
2              308.0     0.057335          0.399857          0.230763
3              282.0     0.083348          0.330211          0.263039
4              548.0     0.074831          0.314328          0.282699

   Sheet_Percentage  protein1_stretch  protein2_stretch  \
0          0.290588               568               260
1          0.242115               255              2414
2          0.284633               419               538
3          0.362890               885                95
4          0.381837              2322               191

                                      aa_composition
0  {'A': 7.308, 'C': 8.077, 'D': 3.462, 'E': 6.15...
1  {'A': 6.297, 'C': 2.113, 'D': 3.19, 'E': 4.184...
2  {'A': 3.717, 'C': 2.416, 'D': 6.877, 'E': 9.29...
3  {'A': 4.211, 'C': 2.105, 'D': 8.421, 'E': 4.21...
4  {'A': 5.759, 'C': 3.141, 'D': 5.759, 'E': 6.80...
```

In [44]:
```
protein_df.describe()
```

Out[44]:

	proteins	Sequence_Length_Difference	Average_Hydrophobicity	Charge_Polarity_Ratio	Aliphatic_Index	Sequence_Simil
count	73110.000000	73110.000000	73110.000000	73110.000000	73110.000000	73
mean	0.501026	468.117262	-0.475682	1.193942	0.266040	
std	0.500002	799.562678	0.229246	0.288368	0.032653	
min	0.000000	0.000000	-1.800824	0.316901	0.051839	
25%	0.000000	121.000000	-0.604008	1.033898	0.245877	
50%	1.000000	278.000000	-0.466046	1.150697	0.266931	
75%	1.000000	564.000000	-0.336713	1.298104	0.287009	
max	1.000000	33274.000000	0.949566	9.666666	0.446602	

In [45]:
```
protein_df.head()
```

Out[45]:

	protein_sequences_1	protein_sequences_2	prote
0	MESSKKMDSPGALQTNPPLKLHTDRSAGTPVFVPEQGGYKEKFVKT...	MARPHPWWLCVLGTLVGLSATPAPKSCPERHYWAQGKLCCQMCEPG...	
1	MVMSSYMVNSKYVDPKFPPCEEYLQGGYLGEQGADYYGGGAQGADF...	MAENVVEPGPPSAKRPKLSSPALSASASDGTDFGSLFDLEHDLPDE...	
2	MNRHLWKSQLCEMVQPSGGPAADQDVLGEESPLGKPAMLHLPSEQG...	MEGGRRARVVIESKRNFFLGAFPTPFPAEHVELGRLGDSETAMVPG...	
3	MAPPSTREPRVLSATSATKSDGEMVLPGFPDADSFVKFALGSVVAV...	MLFYSFFKSLVGKDVVVELKNDLSICGTLHSVDQYLNIKLTDISVT...	
4	MQSGPRPPLPAPGLALALTLTMLARLASAASFFGENHLEVPVATAL...	MQTIKCVVVGDGAVGKTCLLISYTTNKFPSEYVPTVFDNYAVTVMI...	

In [46]:
```
protein_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73110 entries, 0 to 73109
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   protein_sequences_1         73110 non-null  object
 1   protein_sequences_2         73110 non-null  object
 2   proteins                    73110 non-null  int64
 3   Sequence_Length_Difference  73110 non-null  float64
 4   Average_Hydrophobicity      73110 non-null  float64
 5   Charge_Polarity_Ratio       73110 non-null  float64
 6   Aliphatic_Index             73110 non-null  float64
 7   Sequence_Similarity_Score   73110 non-null  float64
 8   Hydrogen_Bond_Ratio         73110 non-null  float64
 9   Cysteine_Count              73110 non-null  float64
 10  Amino_Acid_Diversity        73110 non-null  float64
 11  Avg_Molecular_Weight        73110 non-null  float64
 12  pH_Stability_Index          73110 non-null  float64
 13  Aromaticity                 73110 non-null  float64
 14  Helix_Percentage            73110 non-null  float64
 15  Turn_Percentage             73110 non-null  float64
 16  Sheet_Percentage            73110 non-null  float64
 17  protein1_stretch            73110 non-null  int64
 18  protein2_stretch            73110 non-null  int64
 19  aa_composition              73110 non-null  object
dtypes: float64(14), int64(3), object(3)
memory usage: 11.2+ MB
```

In [47]:
```
protein_df.columns
```

Out[47]:
```
Index(['protein_sequences_1', 'protein_sequences_2', 'proteins',
       'Sequence_Length_Difference', 'Average_Hydrophobicity',
       'Charge_Polarity_Ratio', 'Aliphatic_Index', 'Sequence_Similarity_Score',
       'Hydrogen_Bond_Ratio', 'Cysteine_Count', 'Amino_Acid_Diversity',
       'Avg_Molecular_Weight', 'pH_Stability_Index', 'Aromaticity',
       'Helix_Percentage', 'Turn_Percentage', 'Sheet_Percentage',
       'protein1_stretch', 'protein2_stretch', 'aa_composition'],
      dtype='object')
```

In this dataset, label encoding should be applied only to categorical (non-numeric) features. The columns protein_sequences_1 and protein_sequences_2 contain protein sequences, which are not categorical values but rather biological sequences.All other columns with int64 and float64 data types are already numerical and do not require encoding

In [49]:
```
cols = protein_df.columns.tolist()
cols.remove('protein_sequences_1')
cols.remove('protein_sequences_2')
cols.remove('proteins')
```

In [50]:
```
protein_df.columns
```

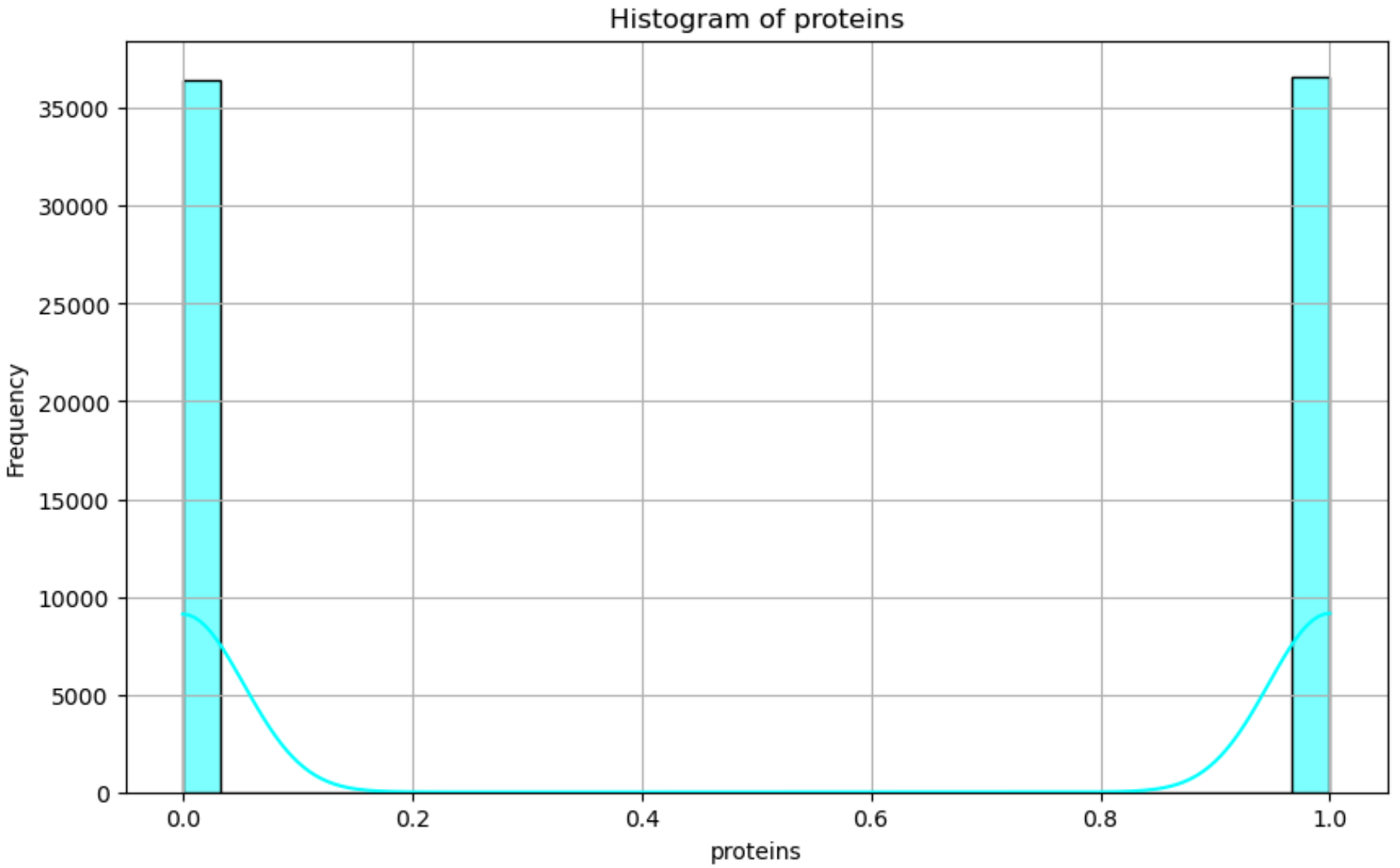
Out[50]:
```
Index(['protein_sequences_1', 'protein_sequences_2', 'proteins',
       'Sequence_Length_Difference', 'Average_Hydrophobicity',
       'Charge_Polarity_Ratio', 'Aliphatic_Index', 'Sequence_Similarity_Score',
       'Hydrogen_Bond_Ratio', 'Cysteine_Count', 'Amino_Acid_Diversity',
       'Avg_Molecular_Weight', 'pH_Stability_Index', 'Aromaticity',
       'Helix_Percentage', 'Turn_Percentage', 'Sheet_Percentage',
       'protein1_stretch', 'protein2_stretch', 'aa_composition'],
      dtype='object')
```

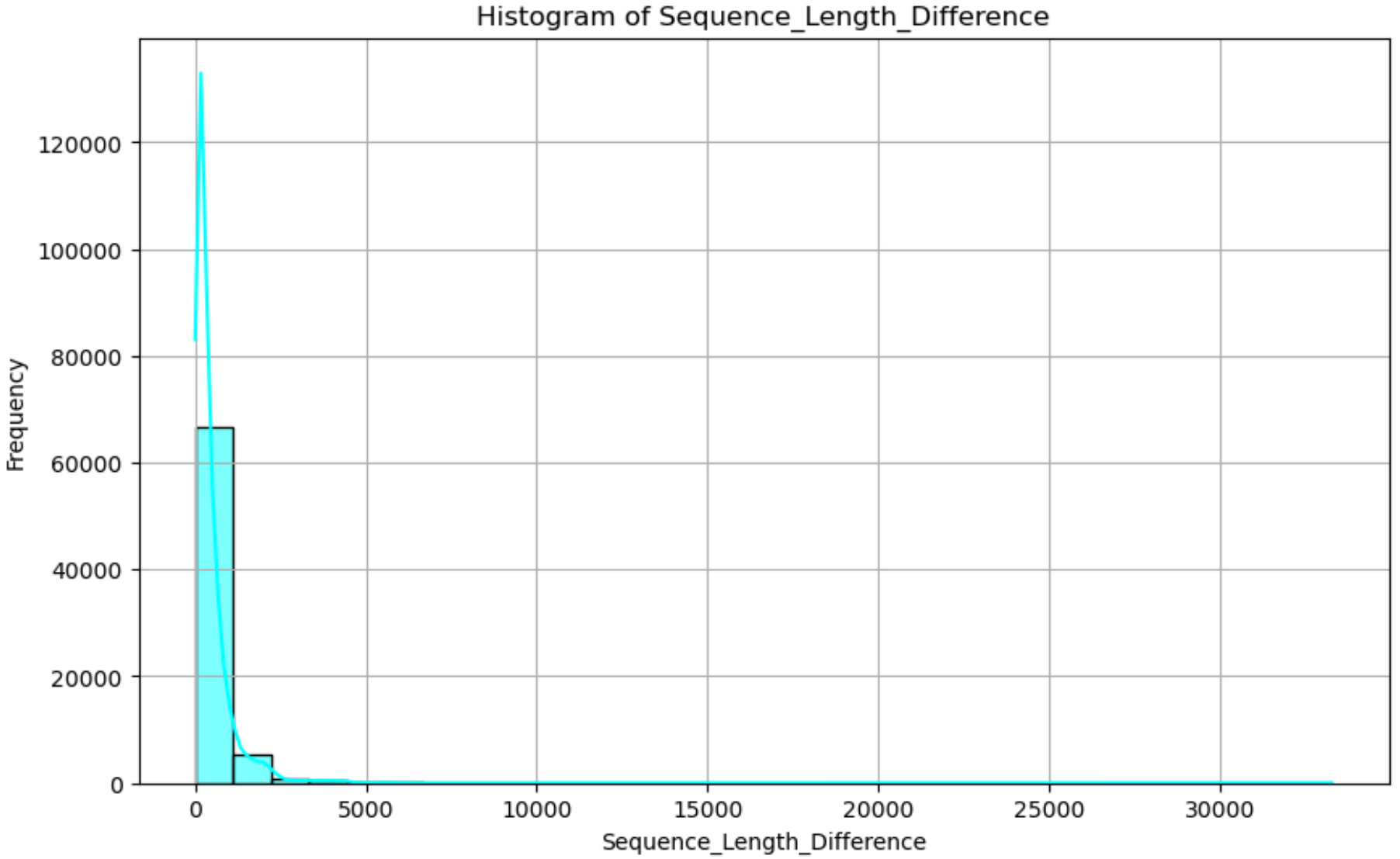
After removing the protein_sequences_1, protein_sequences_2, and proteins columns from the protein_df dataframe, the dataset now consists entirely of numerical features (int64 and float64). These features are already in a format suitable for machine learning models, so label encoding is no longer necessary. Label encoding is typically used to convert categorical variables into numerical representations, but since all remaining features are numerical, no categorical encoding is required. This step ensures that the dataset is clean and ready for further preprocessing, such as scaling or feature selection, before training a machine learning model.

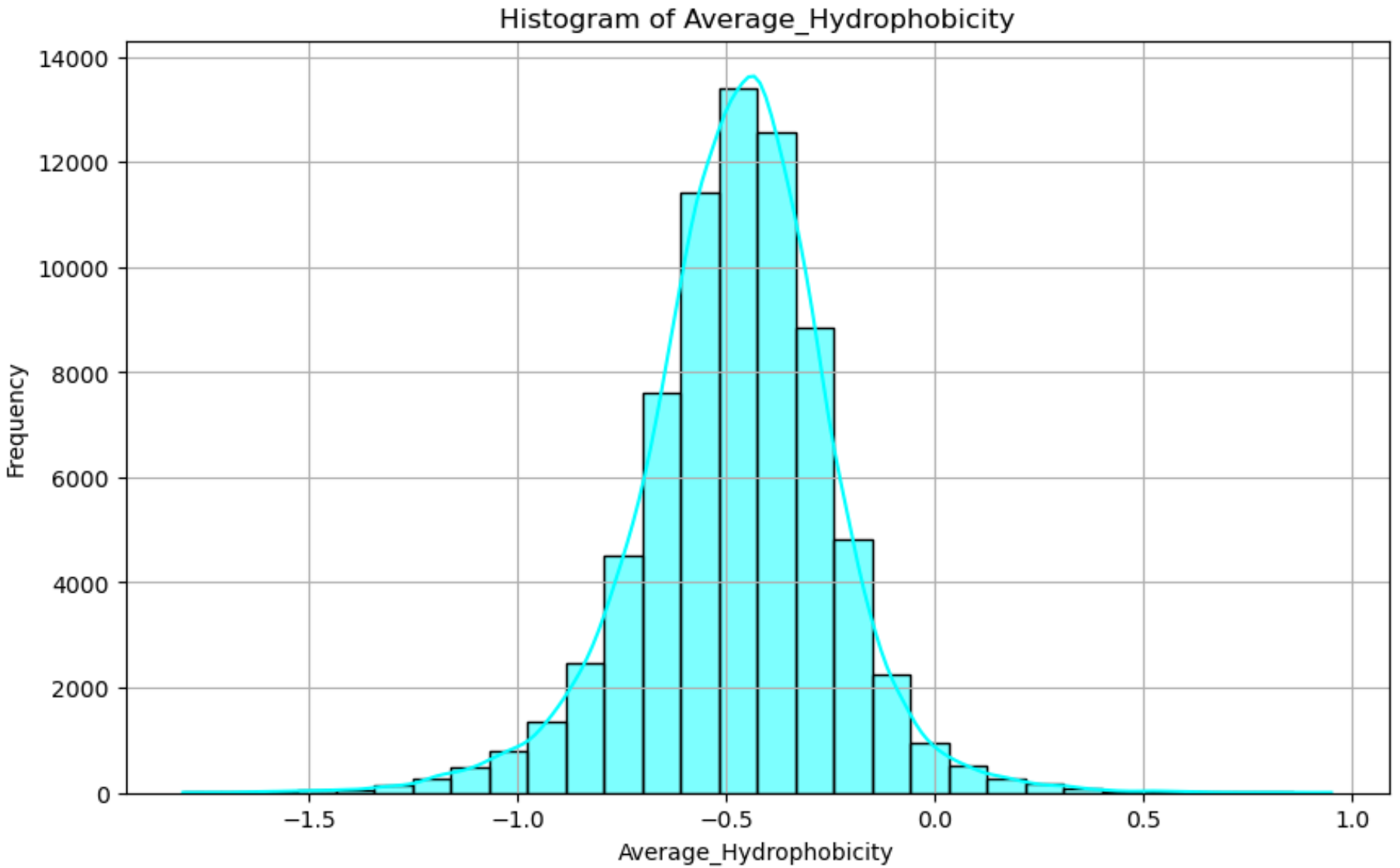
DATA VISUALIZATION

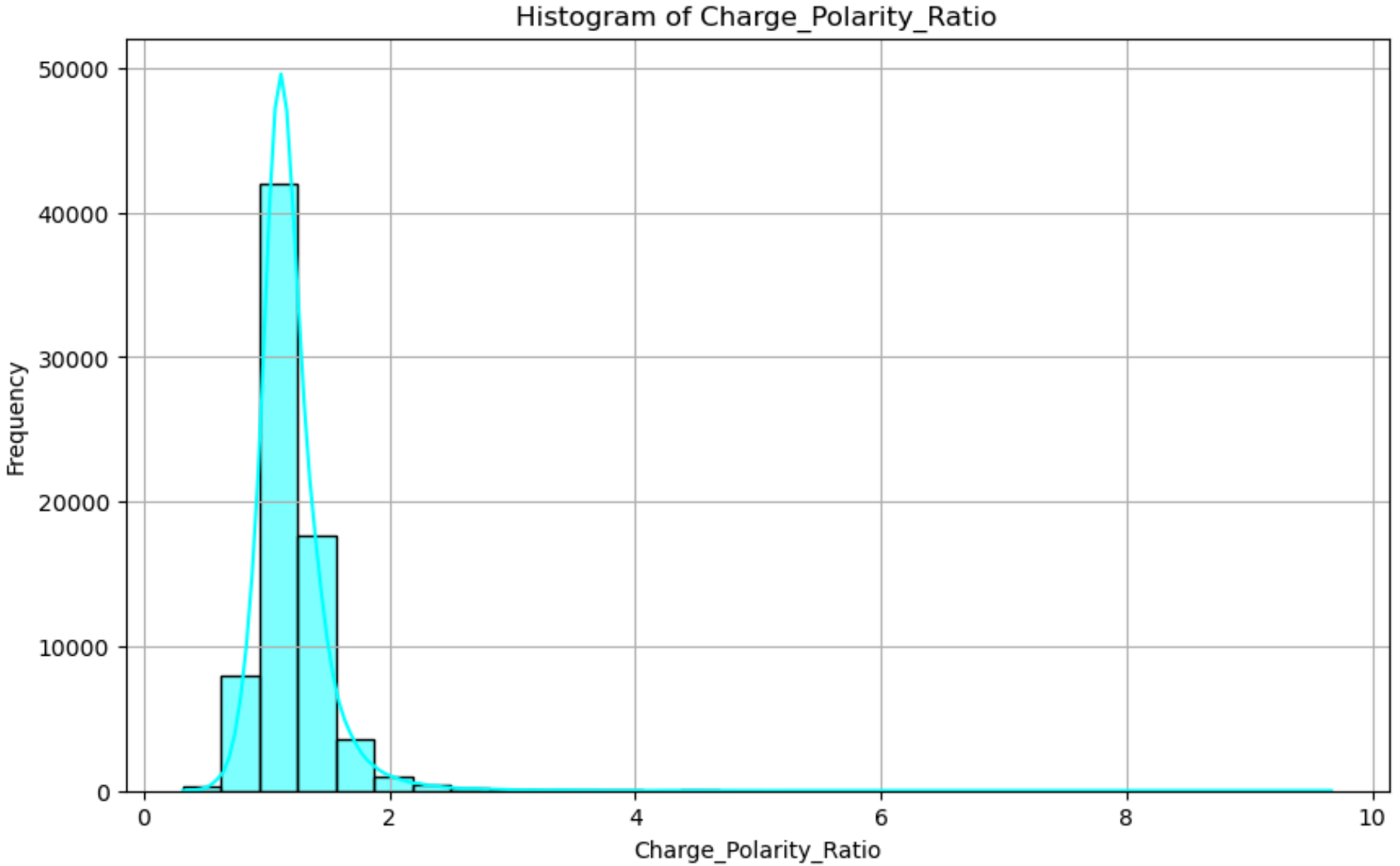
In [53]:
```
Num_Col = protein_df.select_dtypes(include=['int64', 'float64']).columns
for col in Num_Col:
    plt.figure(figsize=(10, 6))
```

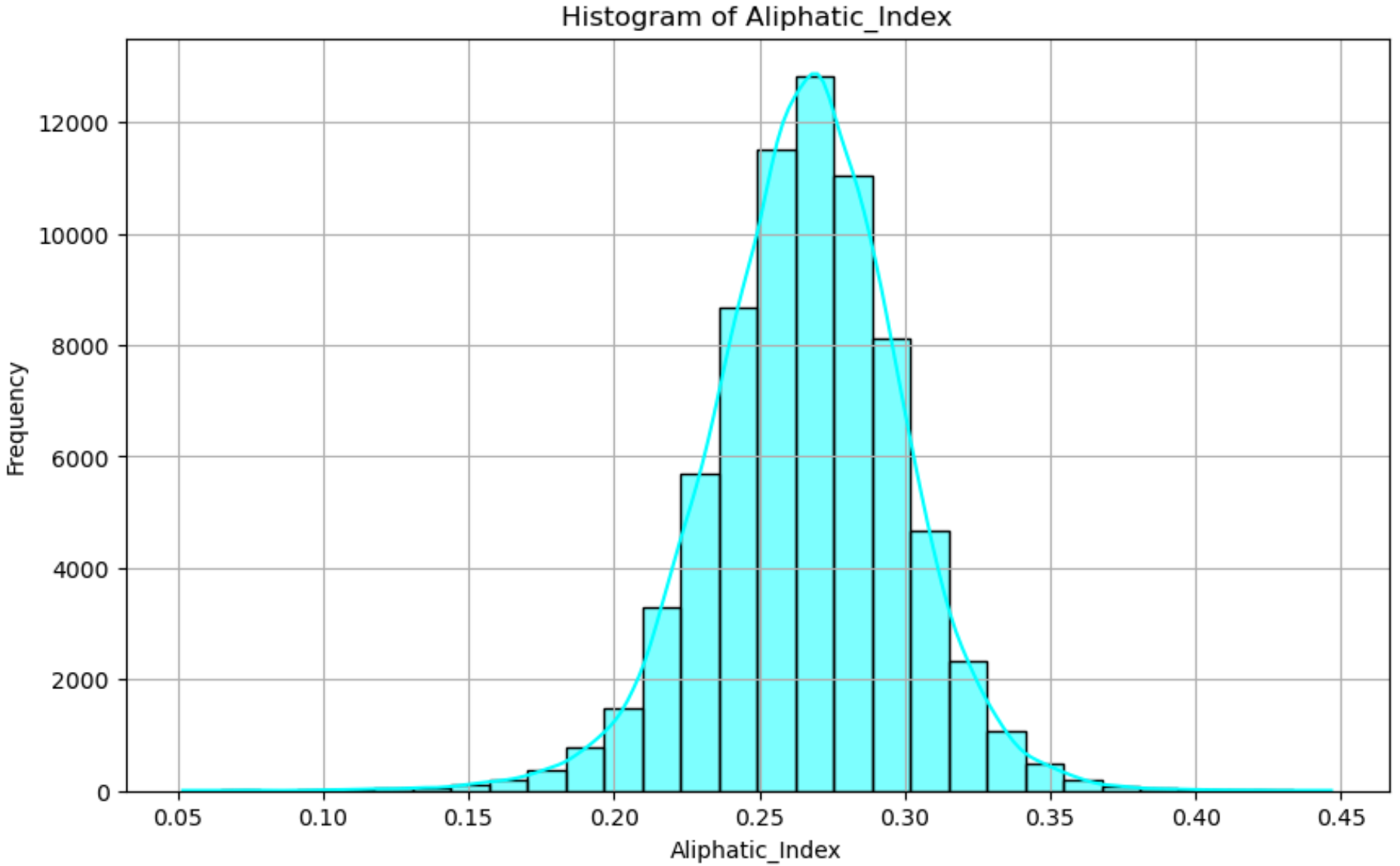
```
    sns.histplot(protein_df[col], bins=30, kde=True, color='Cyan')
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

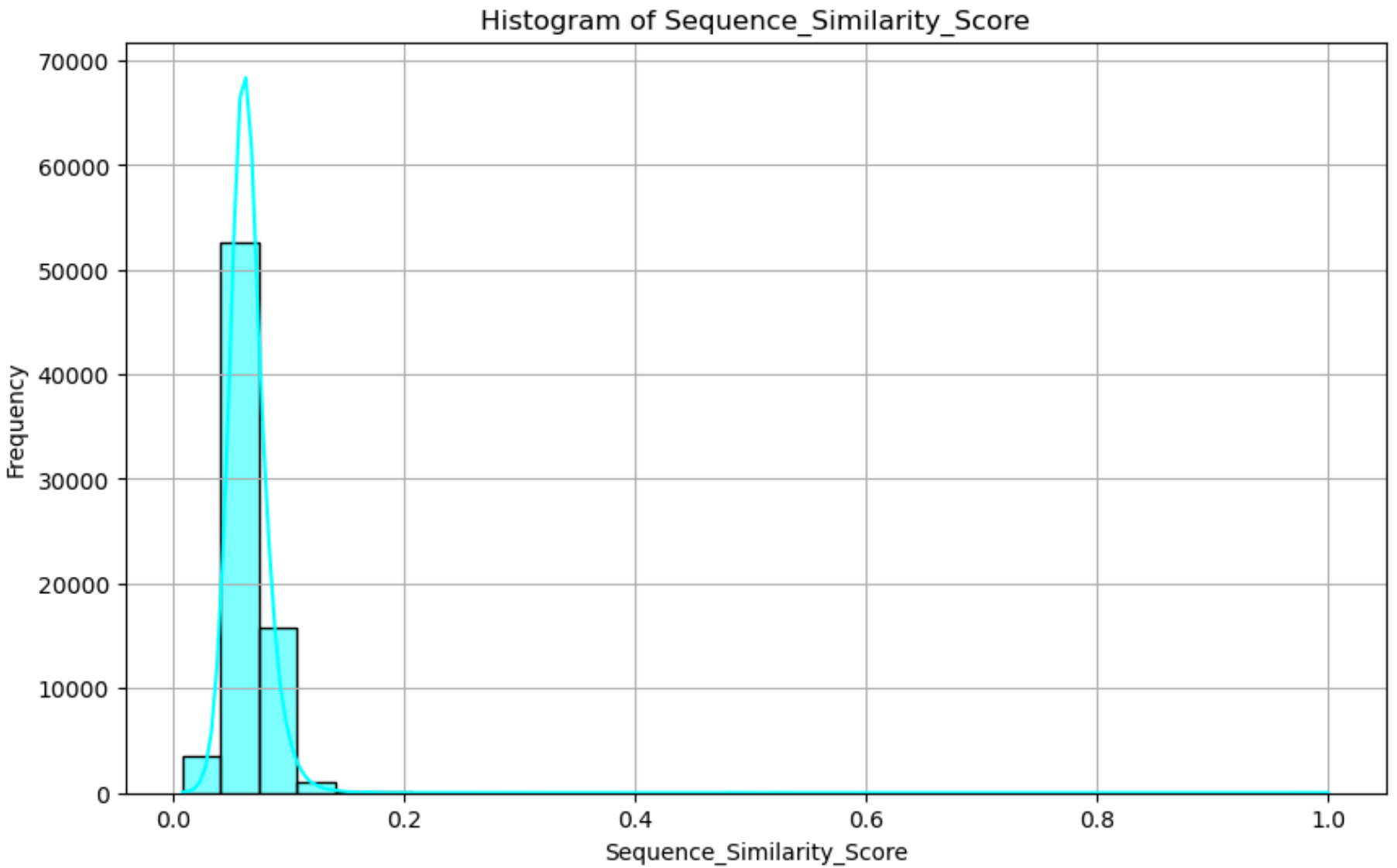


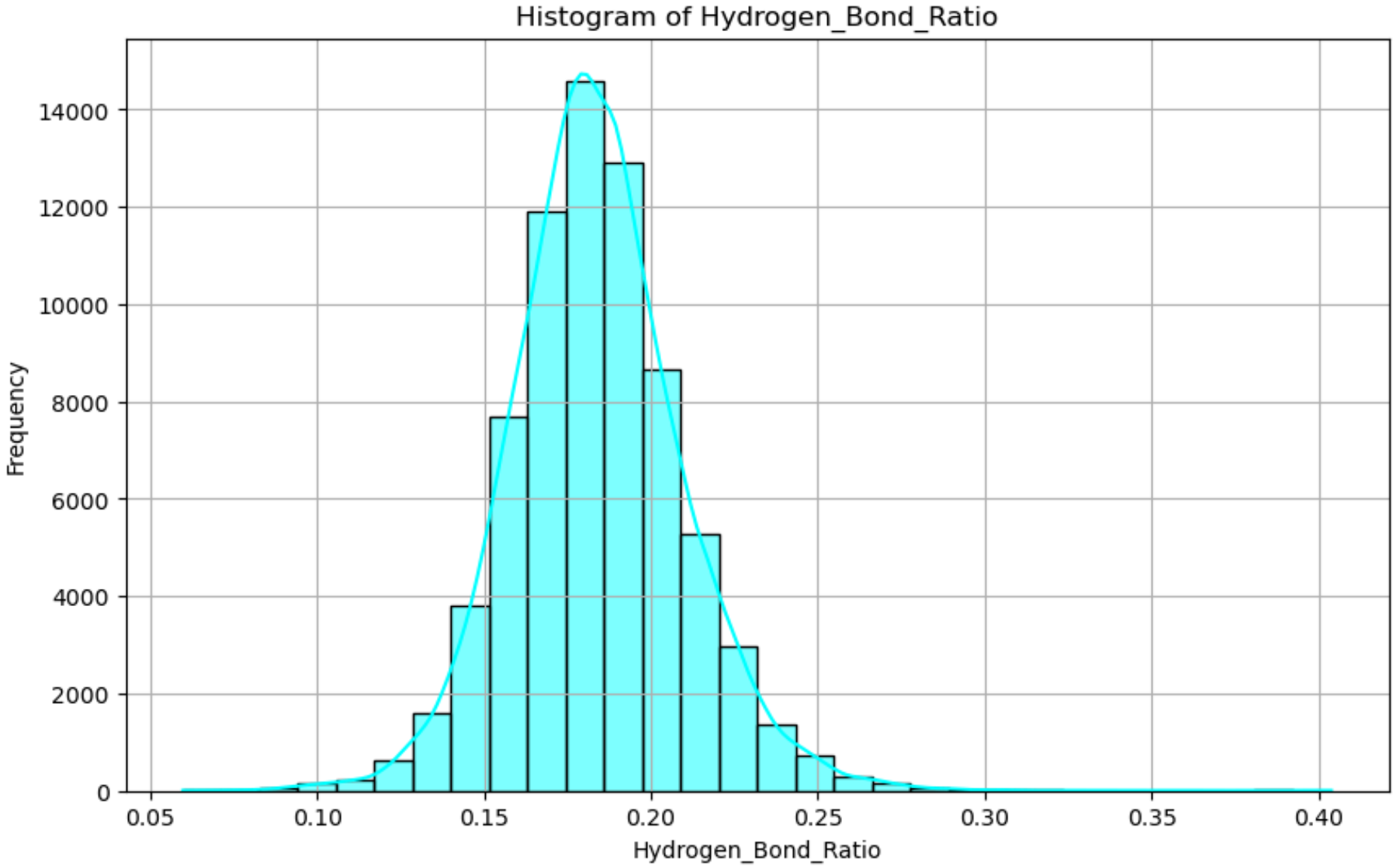

Histogram of Sequence_Length_Difference
Frequency
120000
100000
80000
60000
40000
20000
0
0
5000
10000
15000
20000
25000
30000
Sequence_Length_Difference

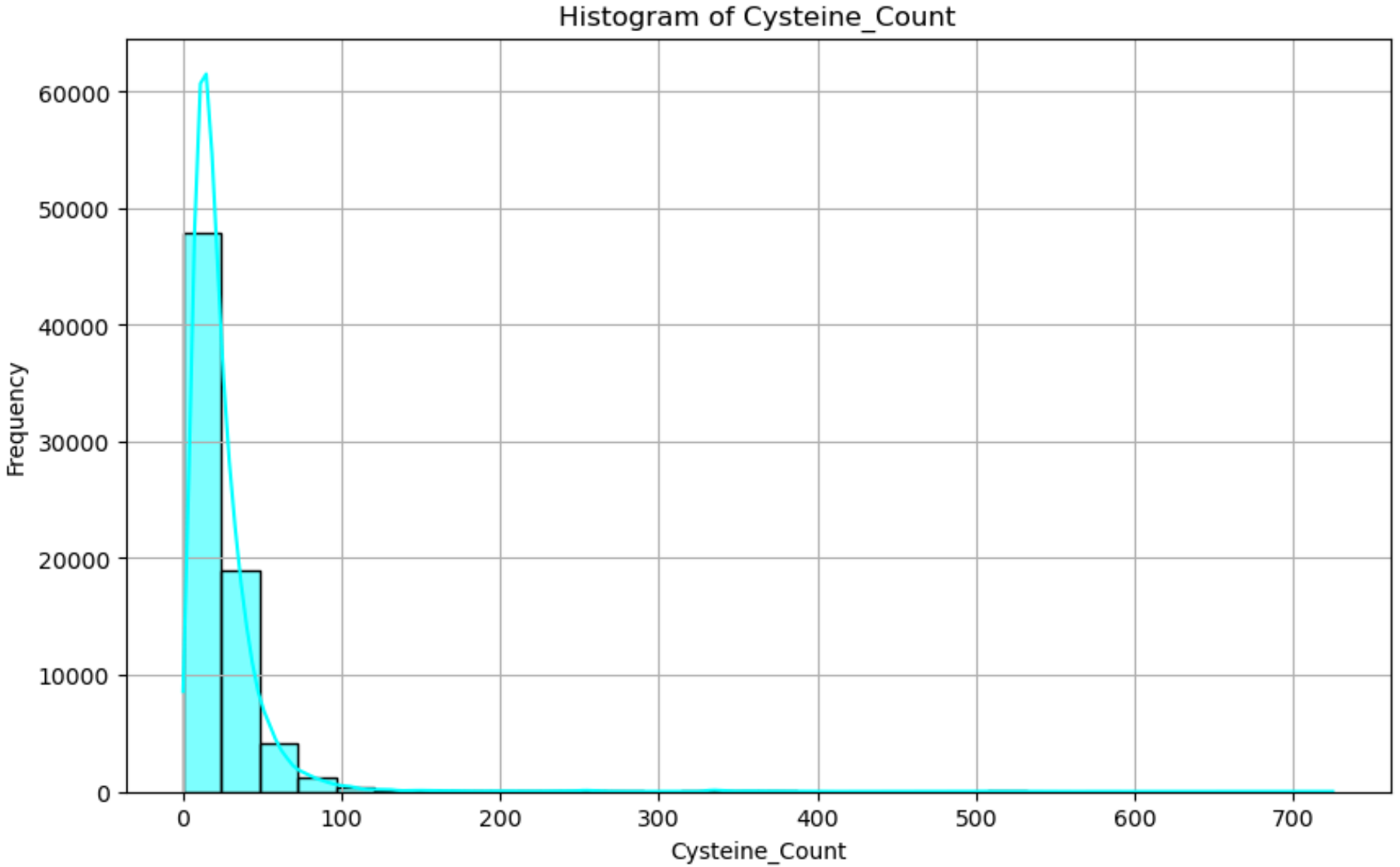

Histogram of Average_Hydrophobicity
Frequency
14000
12000
10000
8000
6000
4000
2000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Average_Hydrophobicity

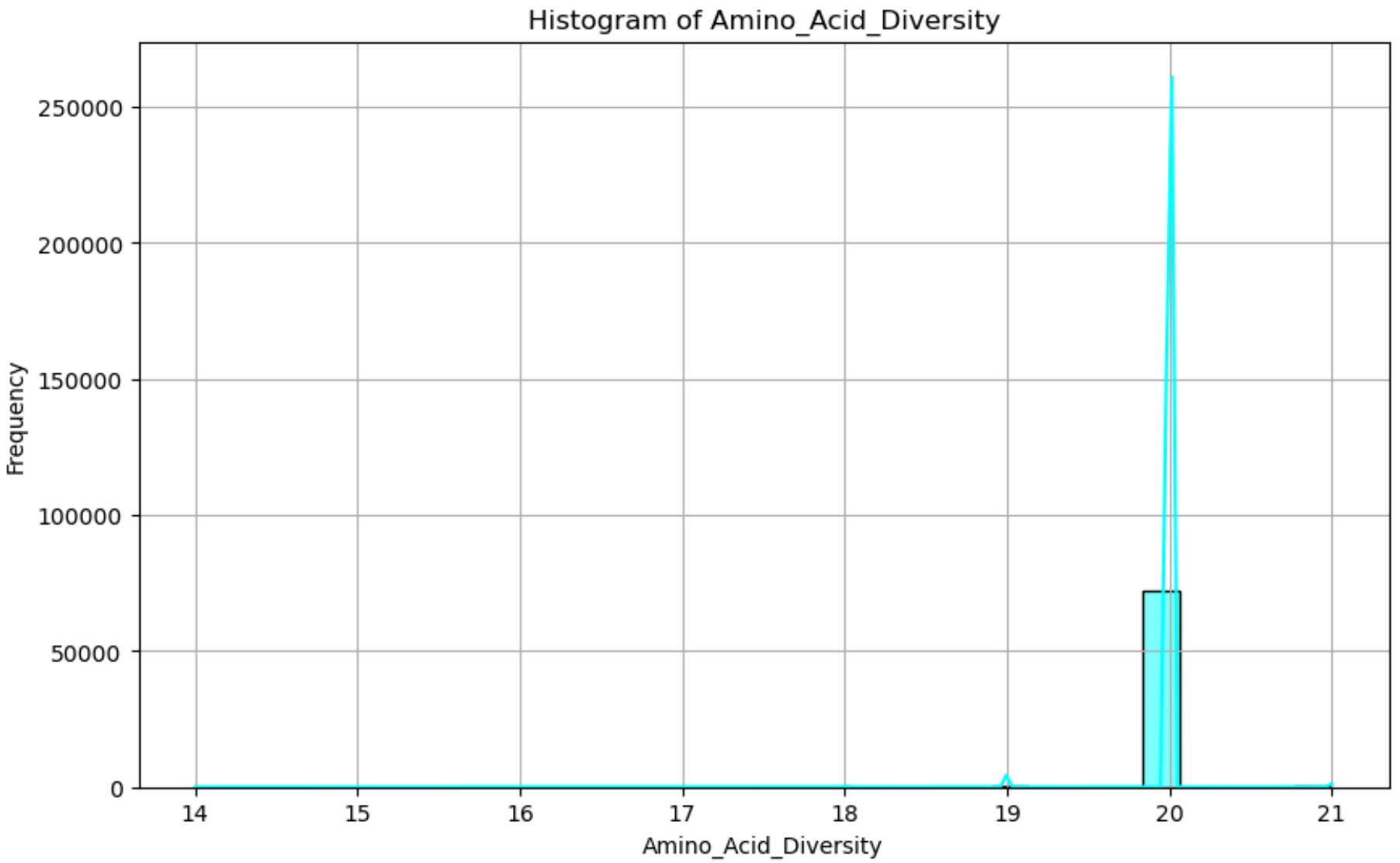

Histogram of Charge_Polarity_Ratio
Frequency
Charge_Polarity_Ratio
0
10000
20000
30000
40000
50000
0
2
4
6
8
10

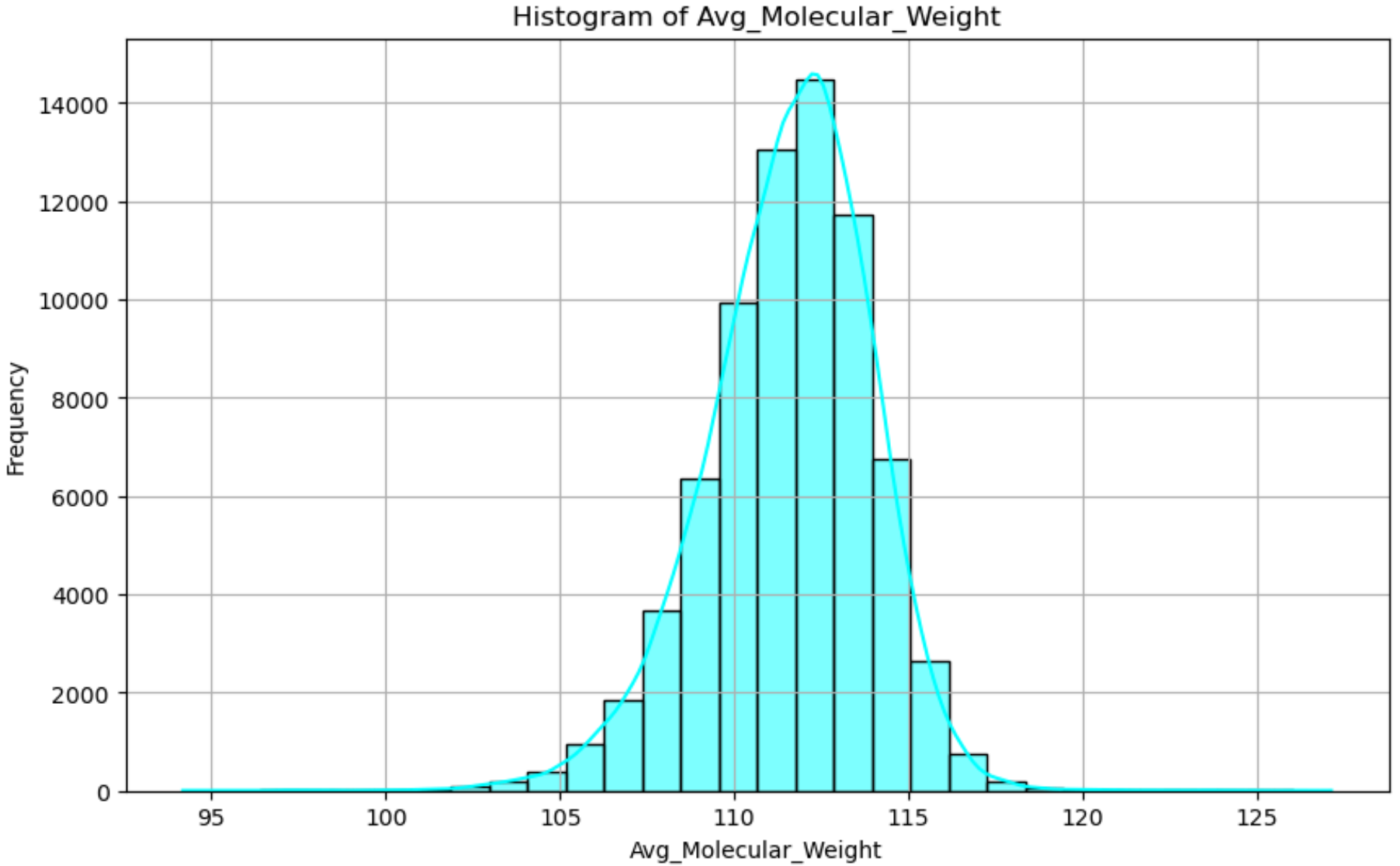

Histogram of Aliphatic_Index
Frequency
Aliphatic_Index
0
2000
4000
6000
8000
10000
12000
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45

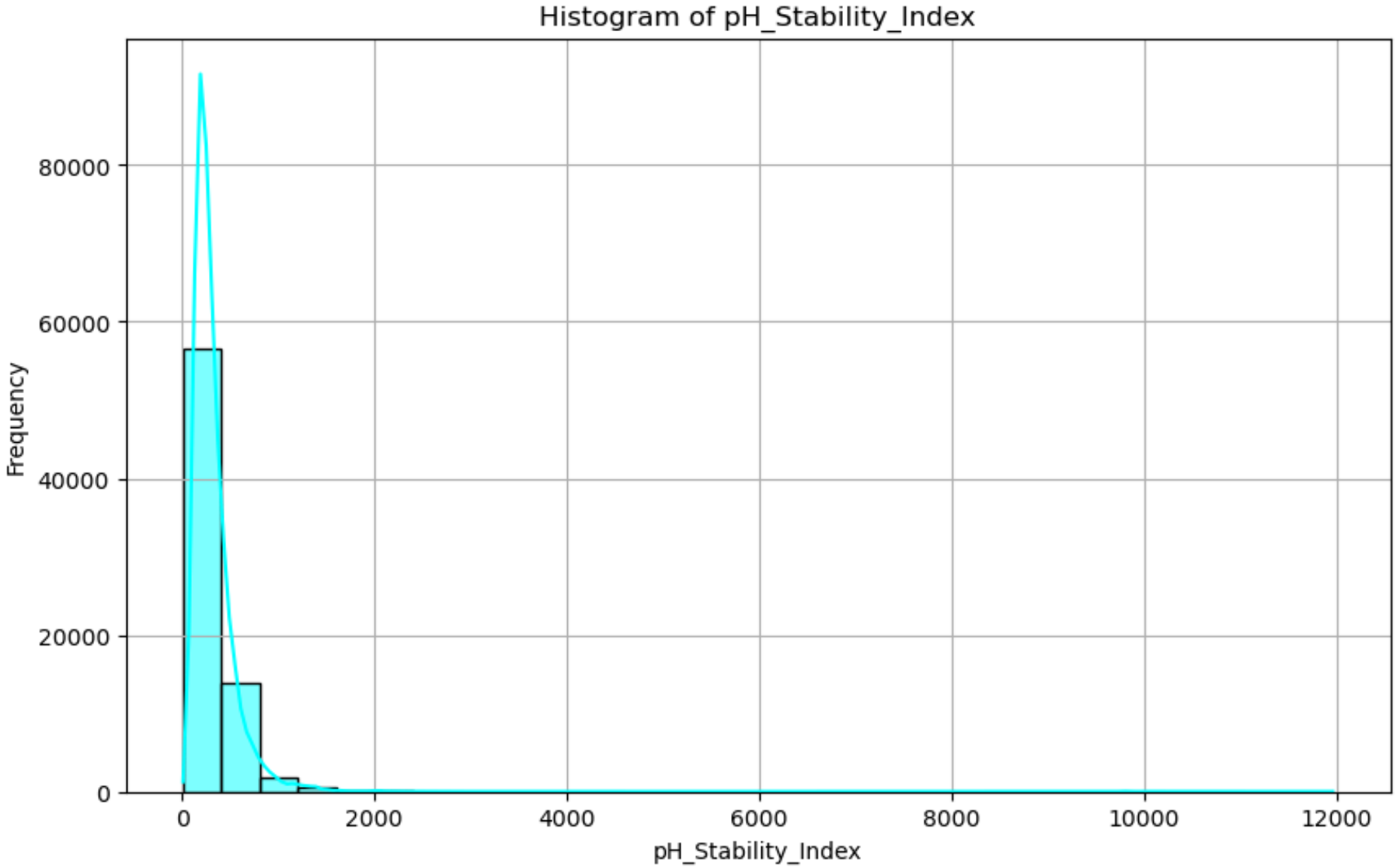

Histogram of Sequence_Similarity_Score
Frequency
70000
60000
50000
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
Sequence_Similarity_Score

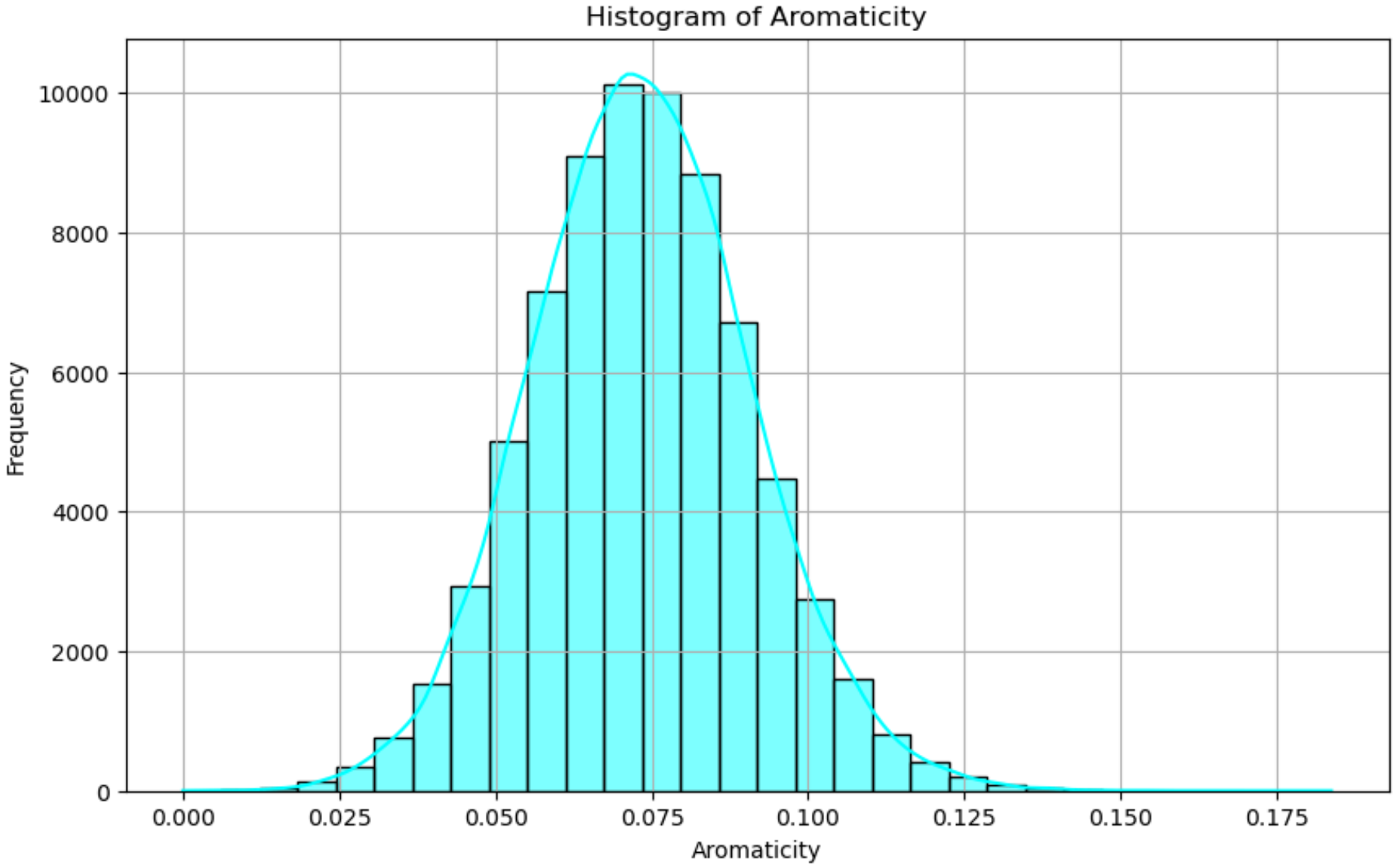

Histogram of Hydrogen_Bond_Ratio
Frequency
0
2000
4000
6000
8000
10000
12000
14000
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Hydrogen_Bond_Ratio

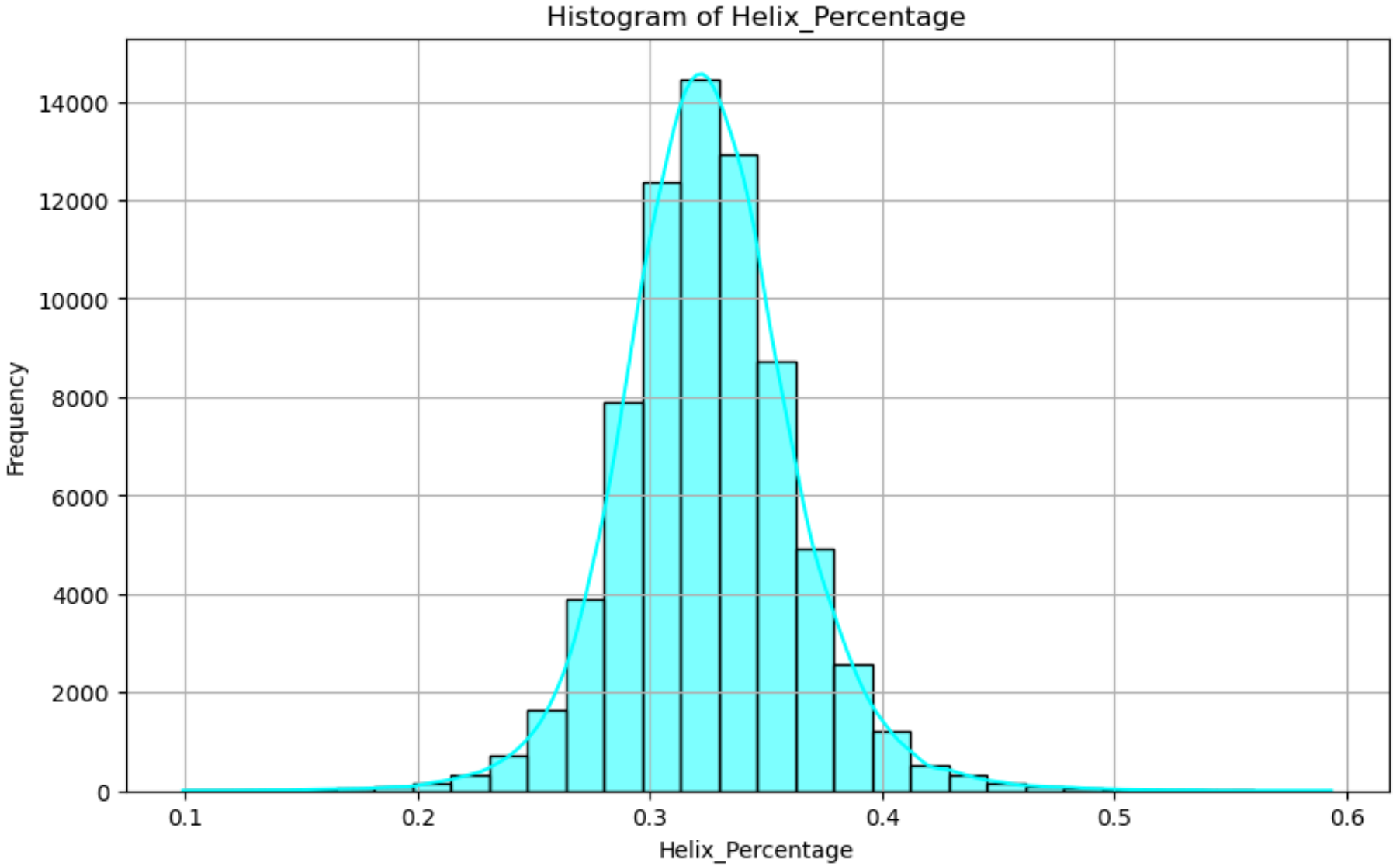

Histogram of Cysteine_Count
Frequency
Cysteine_Count
0
10000
20000
30000
40000
50000
60000
0
100
200
300
400
500
600
700

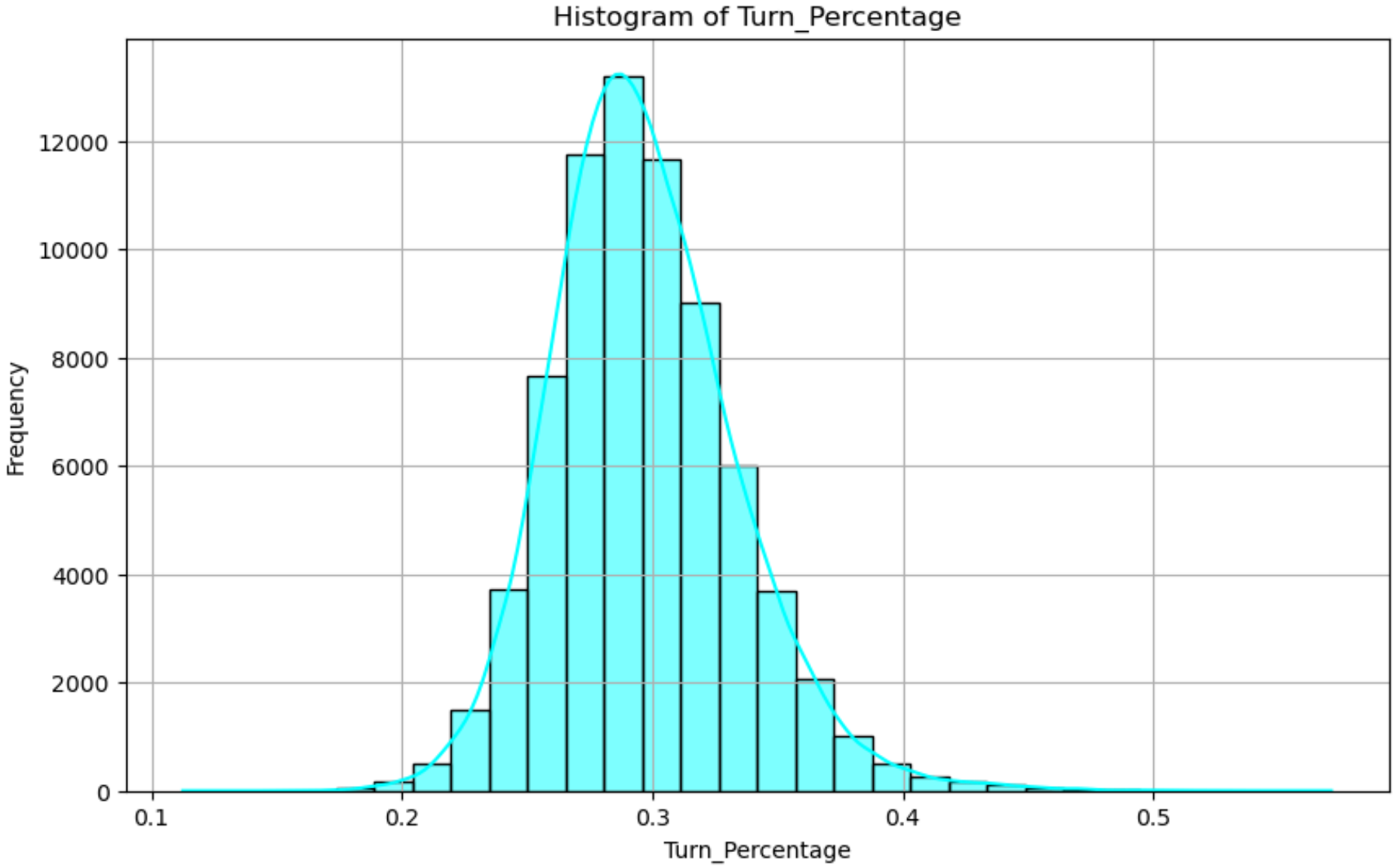

Histogram of Amino_Acid_Diversity
Frequency
250000
200000
150000
100000
50000
0
14
15
16
17
18
19
20
21
Amino_Acid_Diversity

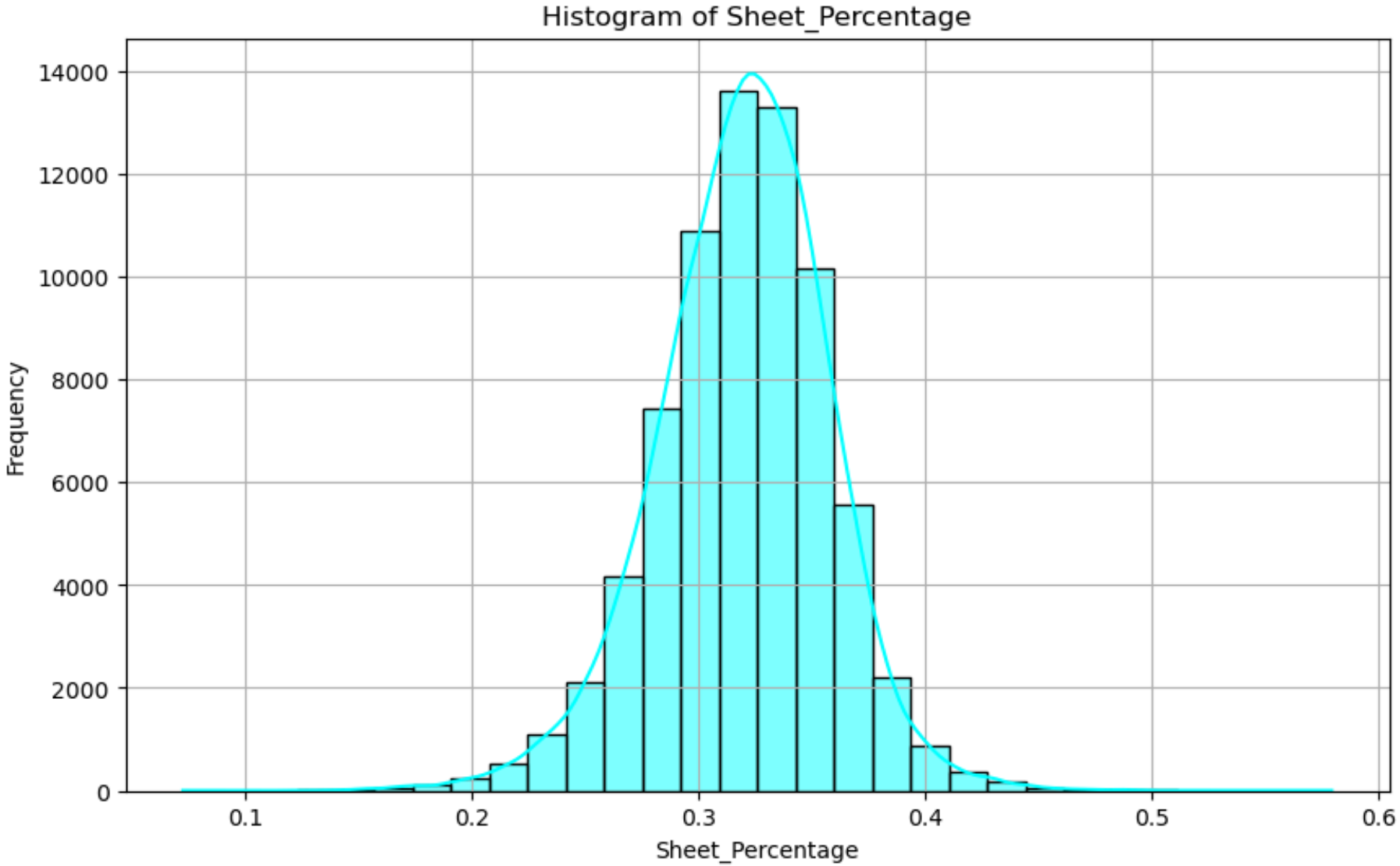

Histogram of Avg_Molecular_Weight
Frequency
0
2000
4000
6000
8000
10000
12000
14000
95
100
105
110
115
120
125
Avg_Molecular_Weight

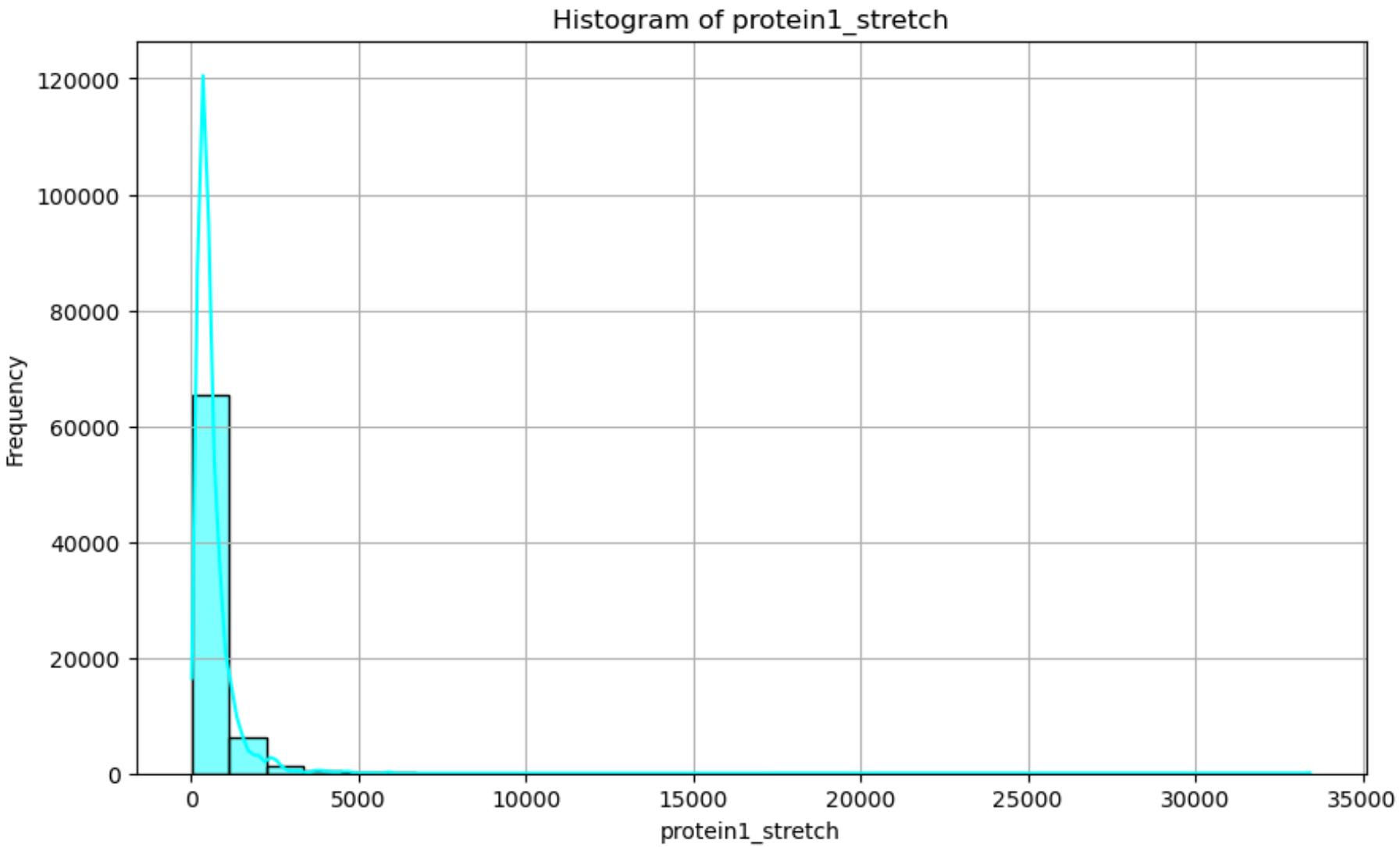

Histogram of pH_Stability_Index
Frequency
80000
60000
40000
20000
0
0
2000
4000
6000
8000
10000
12000
pH_Stability_Index

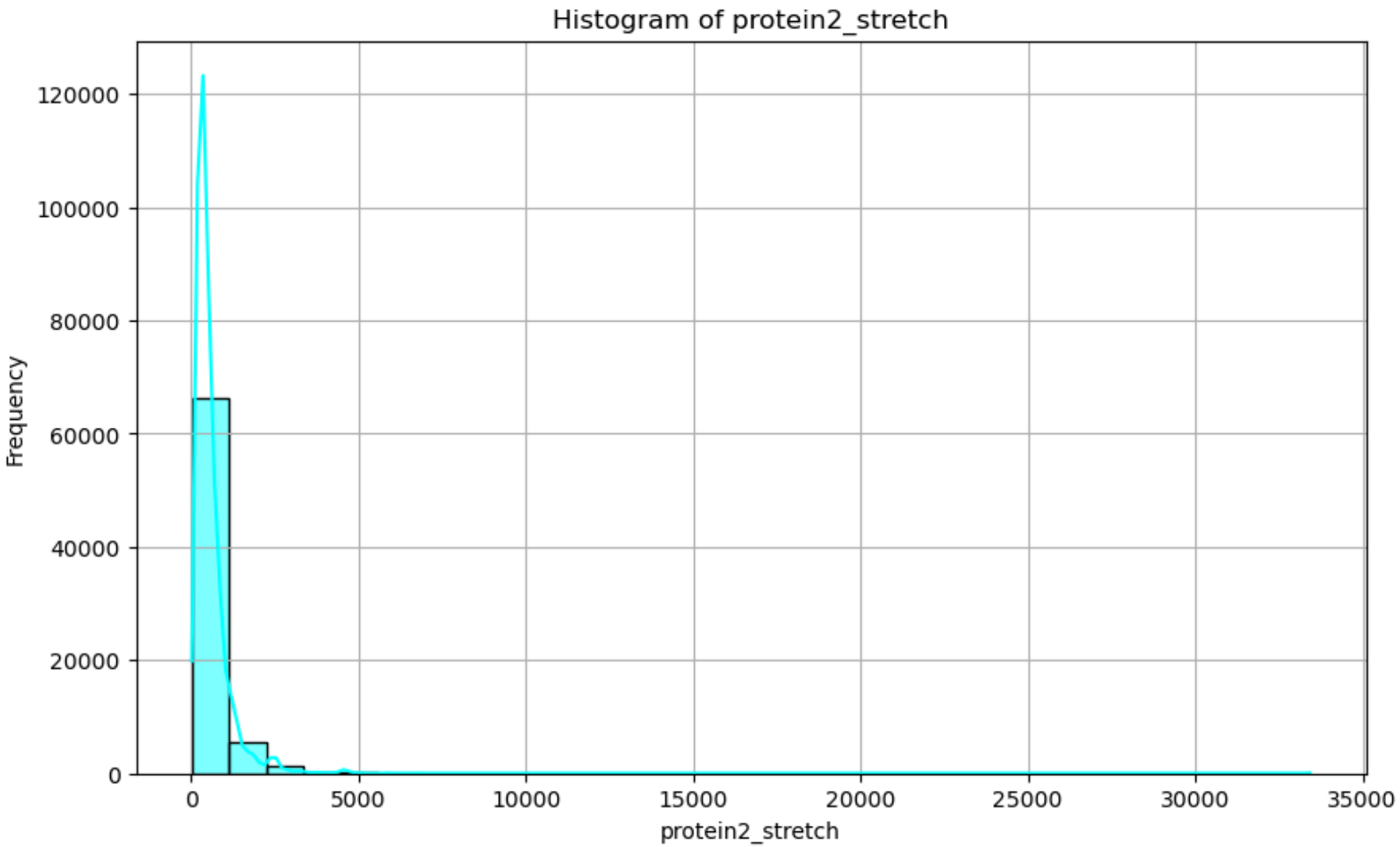

Histogram of Aromaticity
Frequency
10000
8000
6000
4000
2000
0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Aromaticity


Histogram of Helix_Percentage
Frequency
14000
12000
10000
8000
6000
4000
2000
0
0.1
0.2
0.3
0.4
0.5
0.6
Helix_Percentage


Histogram of Turn_Percentage
Frequency
Turn_Percentage
0
2000
4000
6000
8000
10000
12000
0.1
0.2
0.3
0.4
0.5

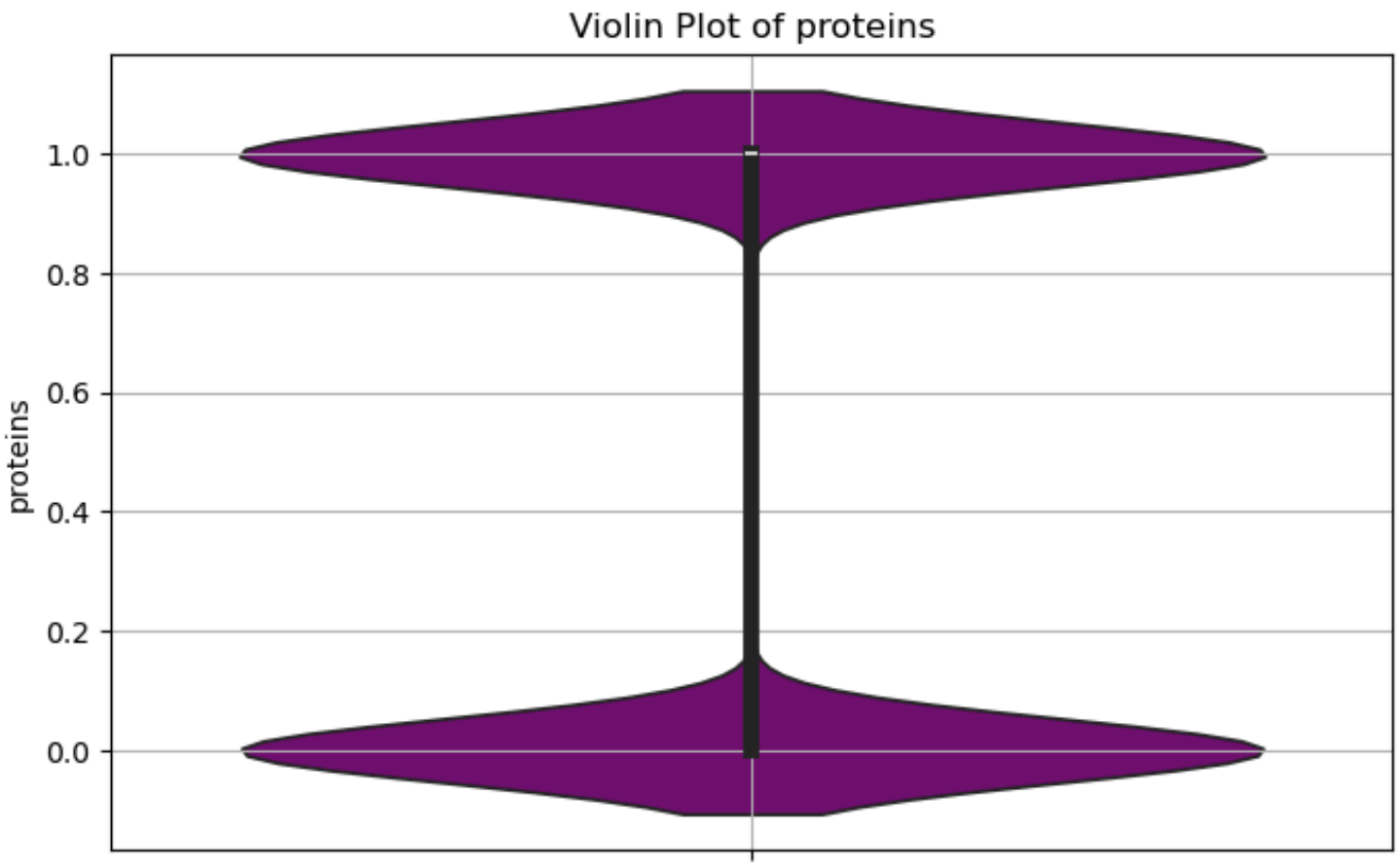

Histogram of Sheet_Percentage
Frequency
Sheet_Percentage
0
2000
4000
6000
8000
10000
12000
14000
0.1
0.2
0.3
0.4
0.5
0.6

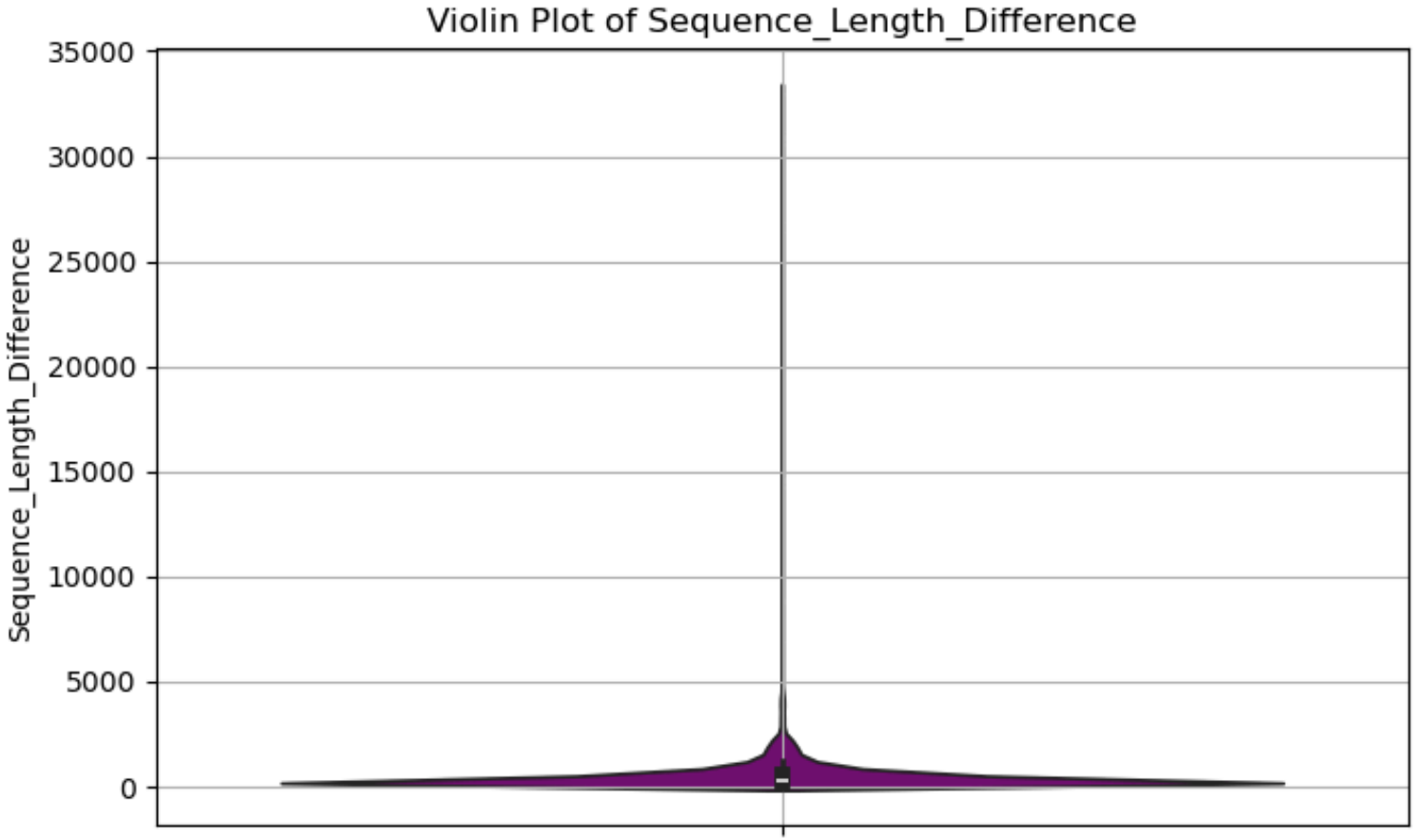

Histogram of protein1_stretch
Frequency
120000
100000
80000
60000
40000
20000
0
0
5000
10000
15000
20000
25000
30000
35000
protein1_stretch

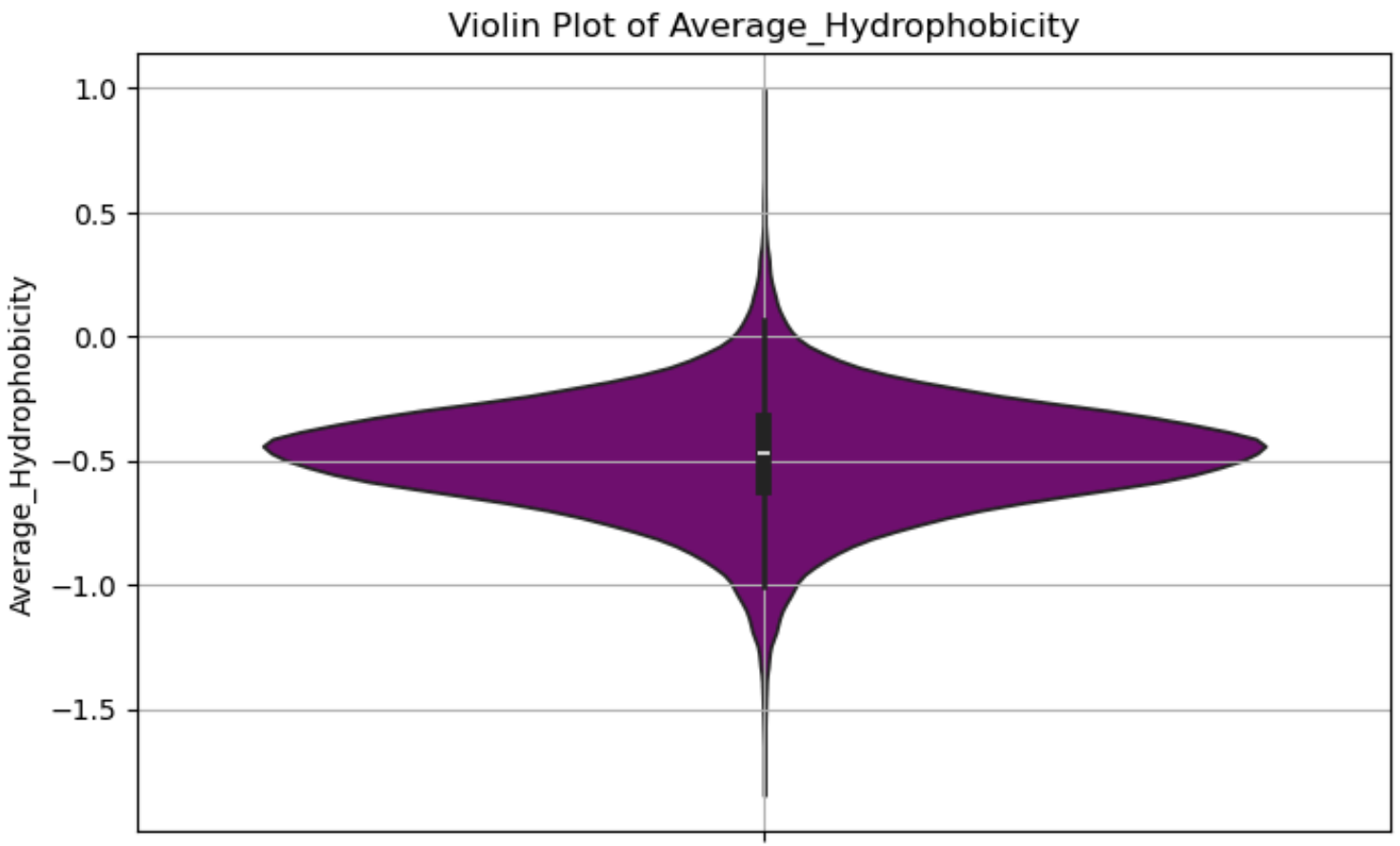


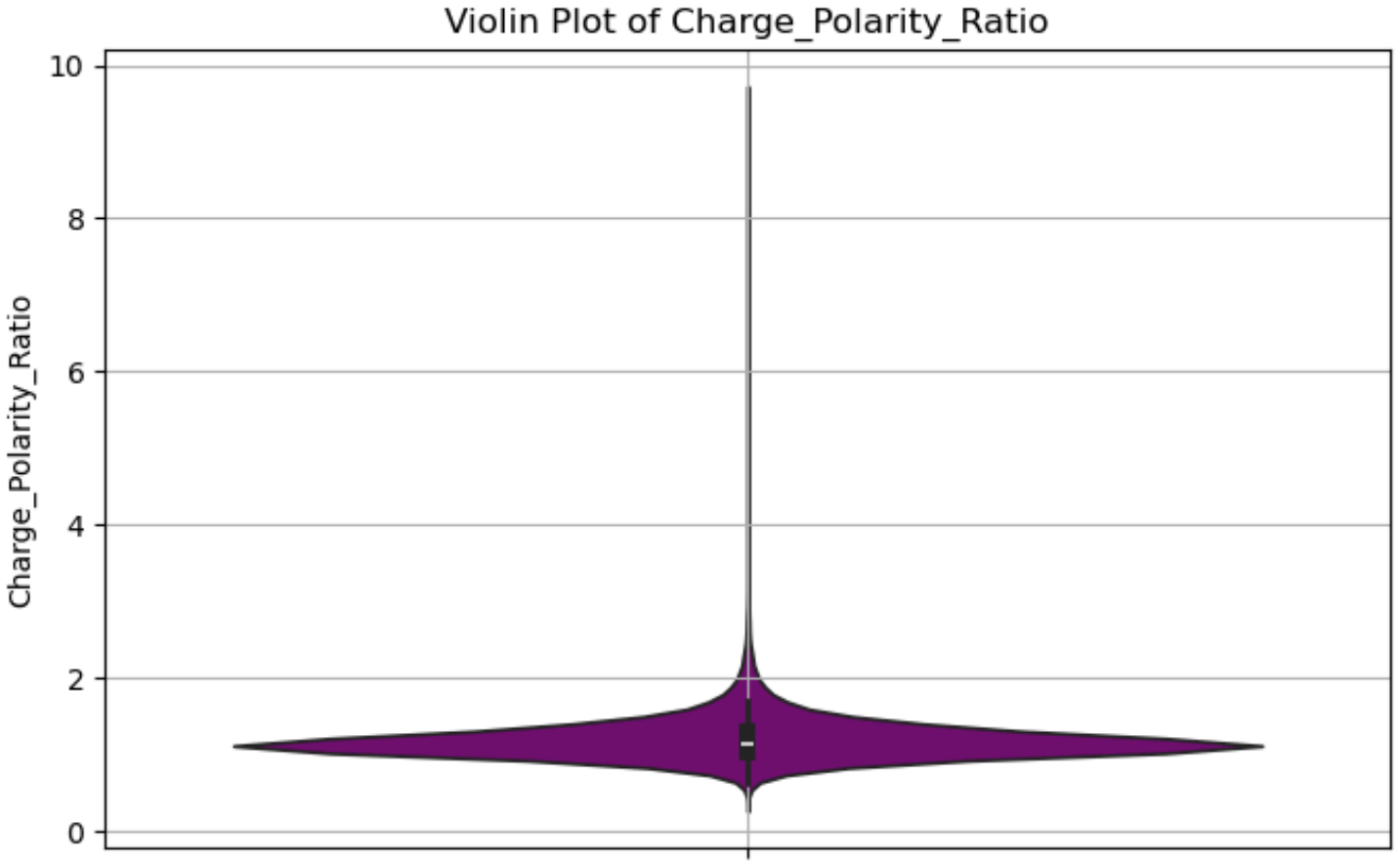
In [54]:
```
Num_Col = protein_df.select_dtypes(include=['int64','float64']).columns
for col in Num_Col:
    plt.figure(figsize=(8, 5))
    sns.violinplot(y=protein_df[col], color='purple')
    plt.title(f'Violin Plot of {col}')
    plt.ylabel(col)
```

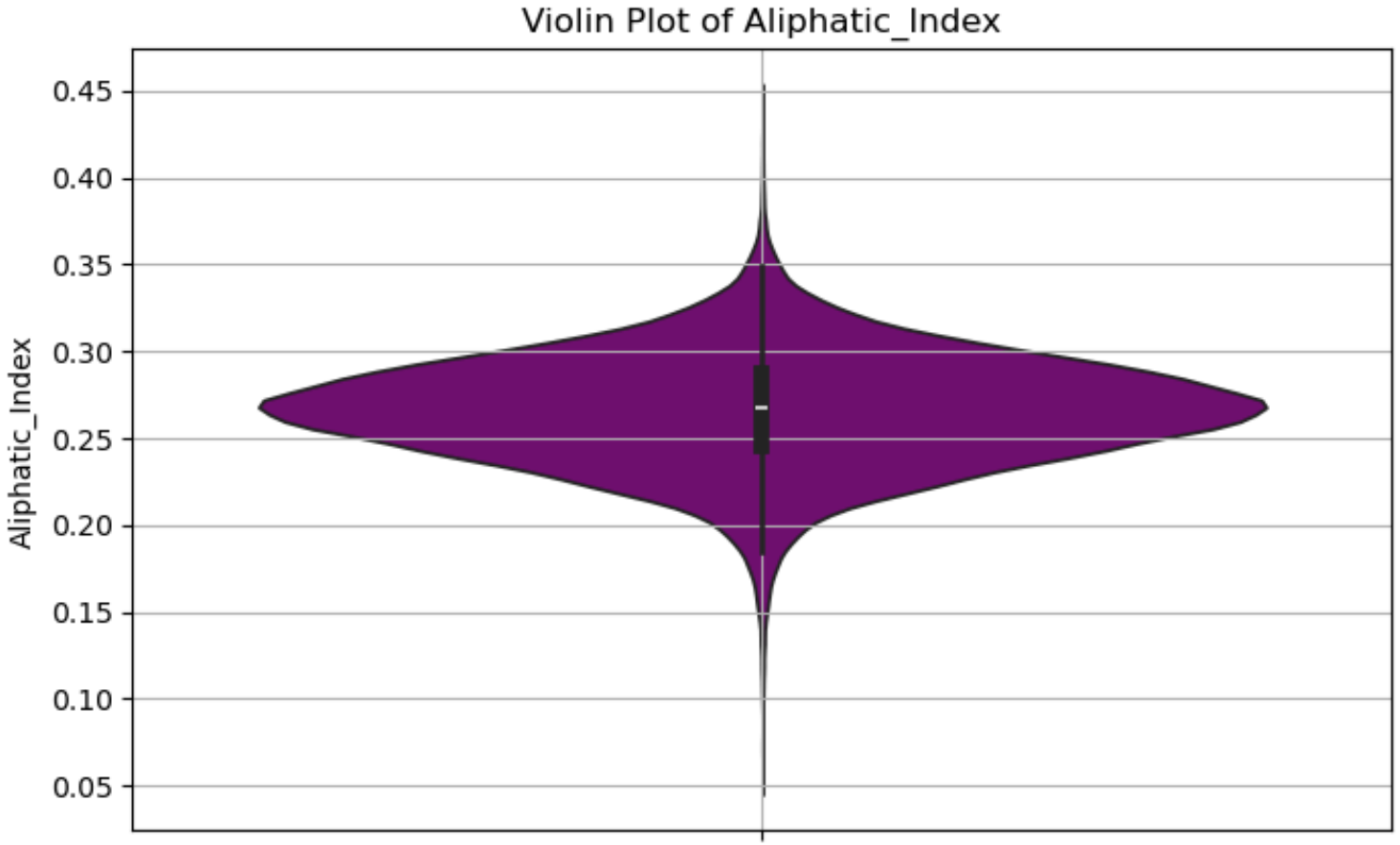
```
    plt.grid(True)
    plt.show()
```

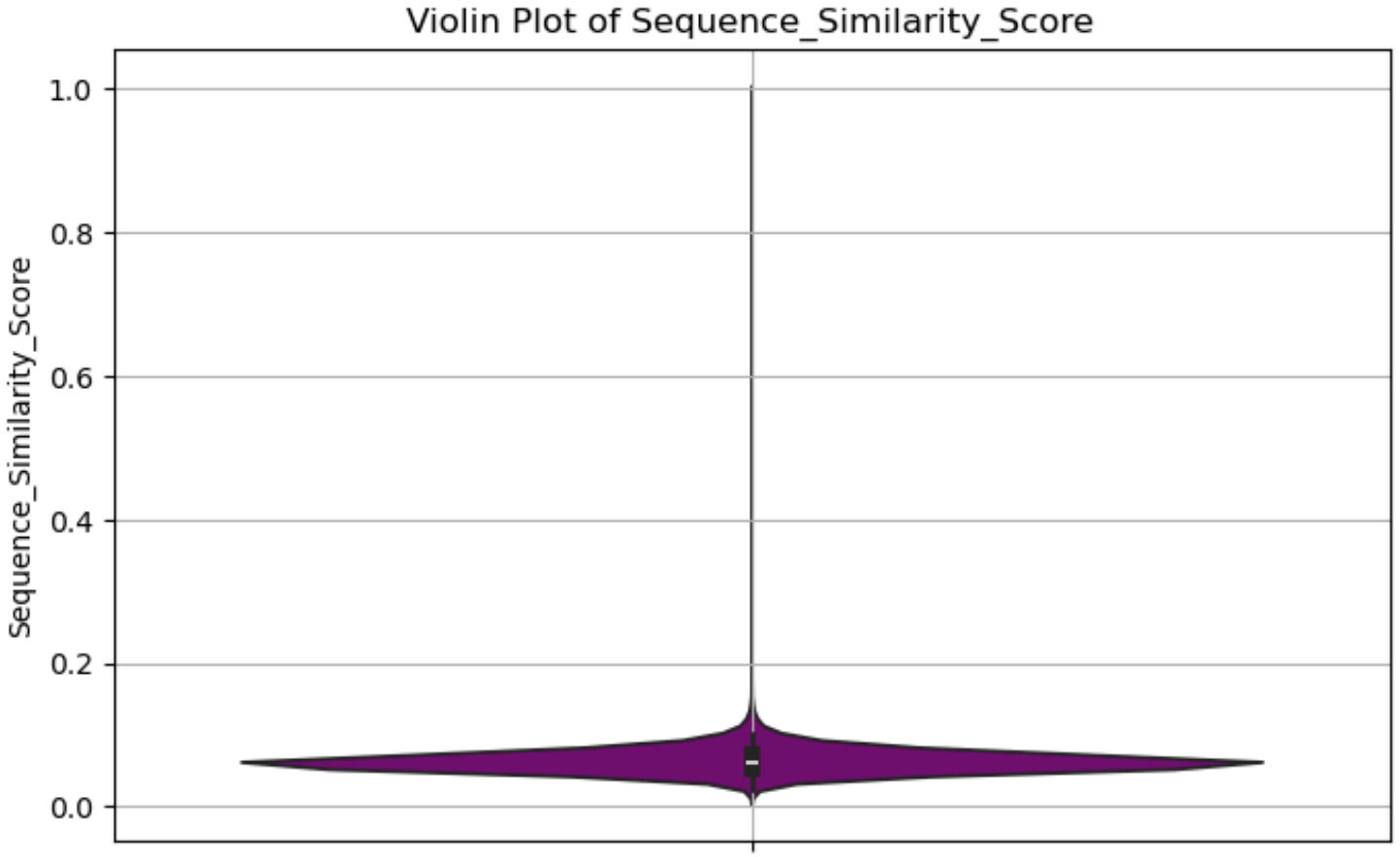


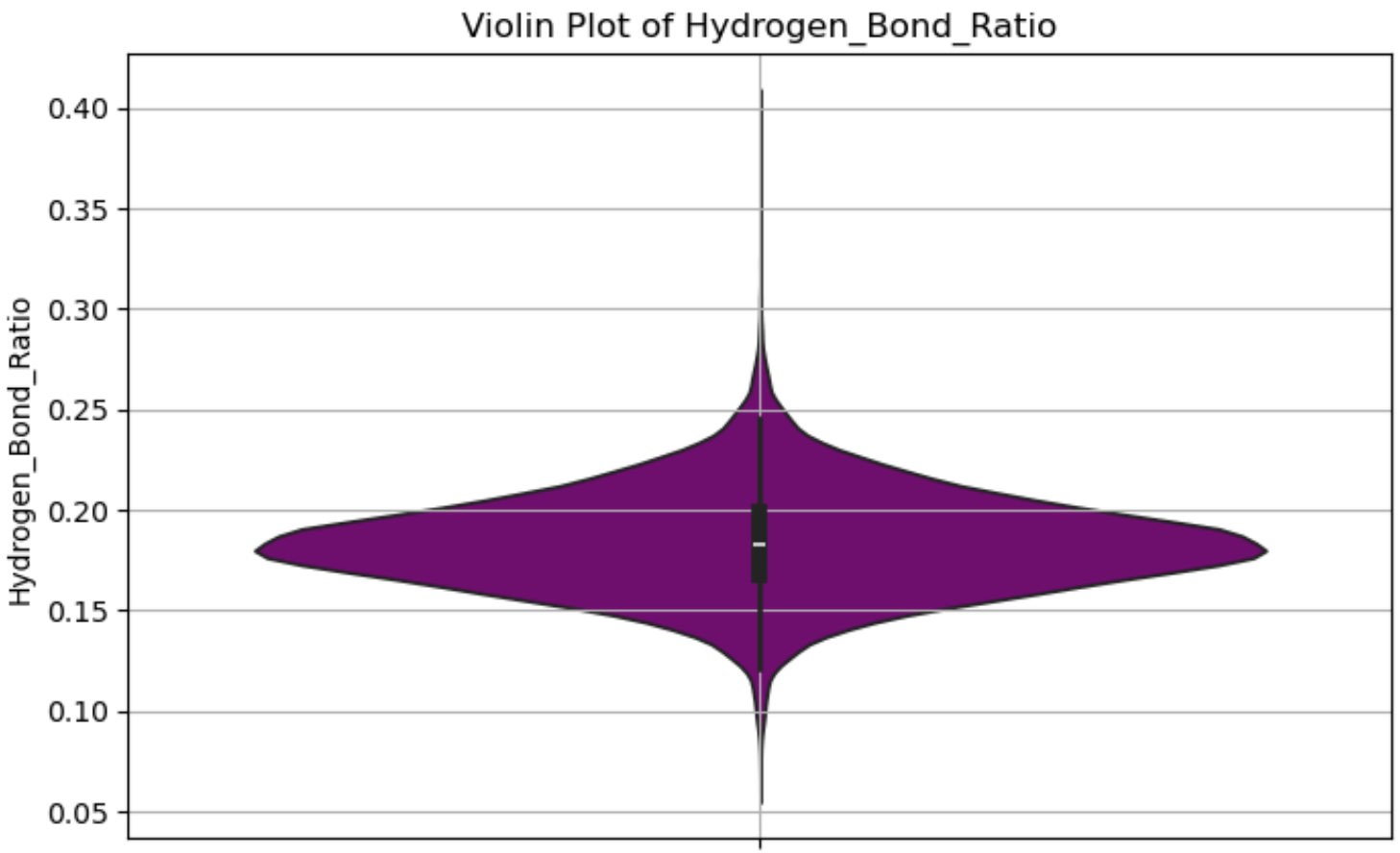

Violin Plot of Sequence_Length_Difference
Sequence_Length_Difference
35000
30000
25000
20000
15000
10000
5000
0

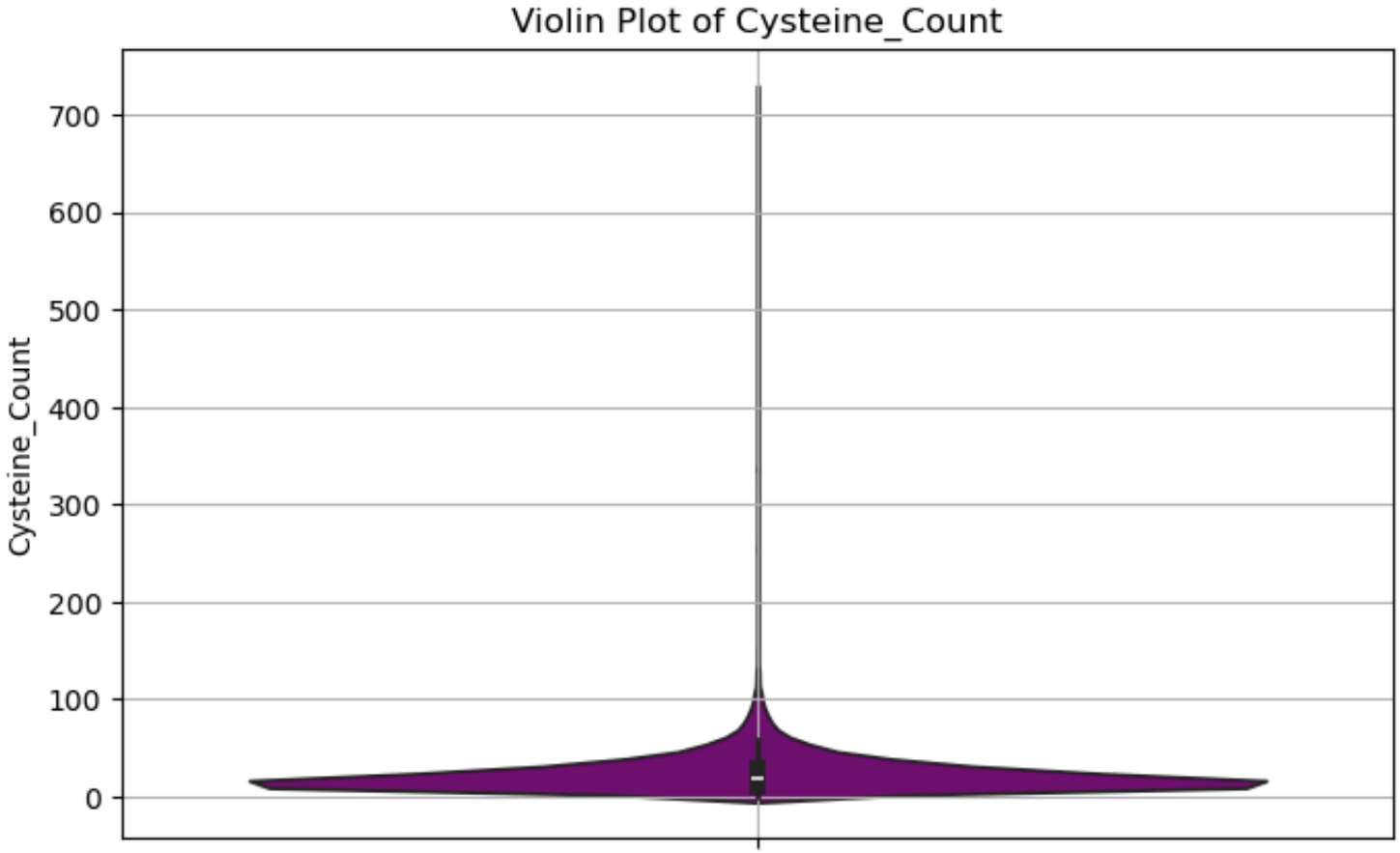

Violin Plot of Average_Hydrophobicity
Average_Hydrophobicity
1.0
0.5
0.0
−0.5
−1.0
−1.5

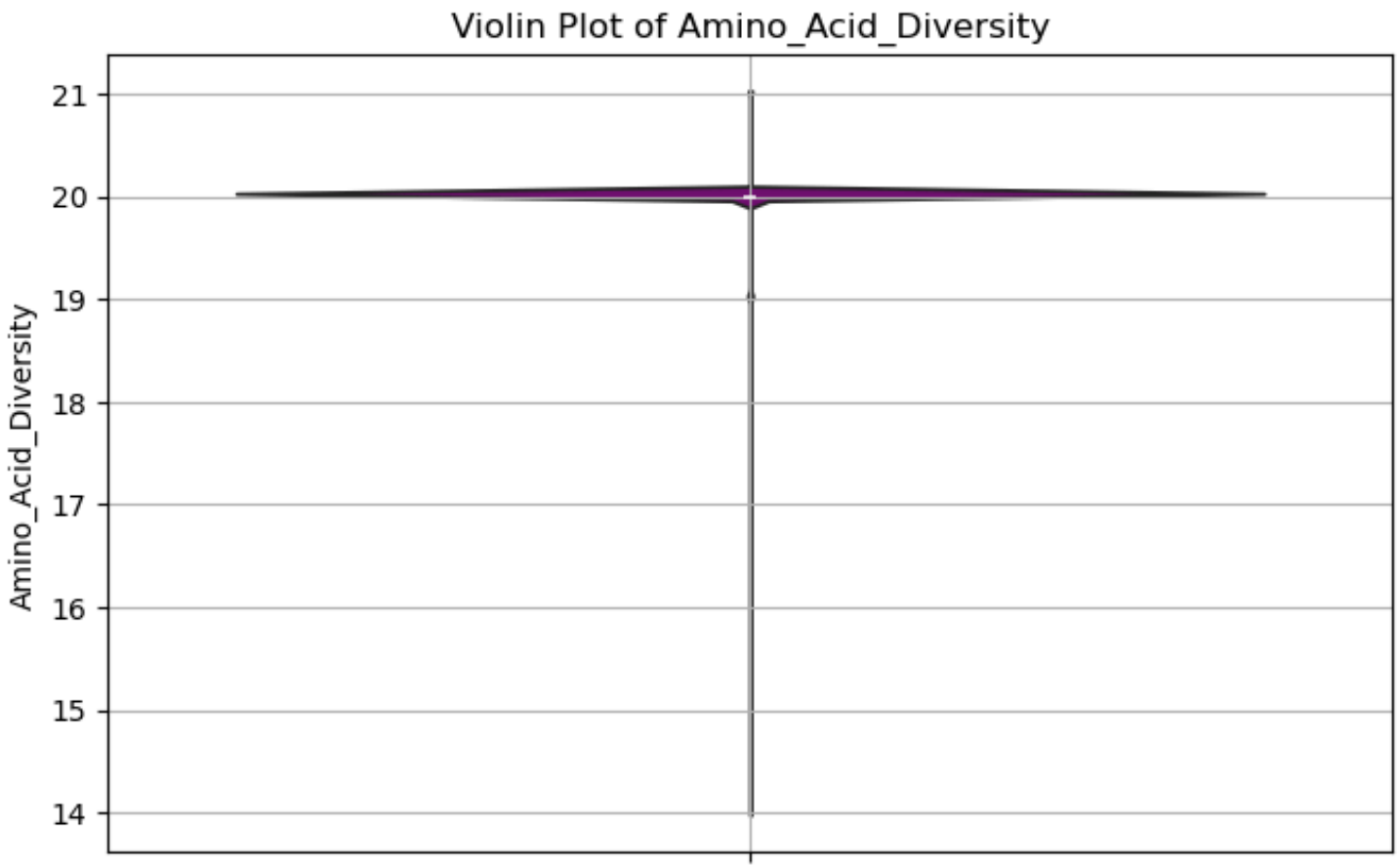

Violin Plot of Charge_Polarity_Ratio
Charge_Polarity_Ratio
10
8
6
4
2
0

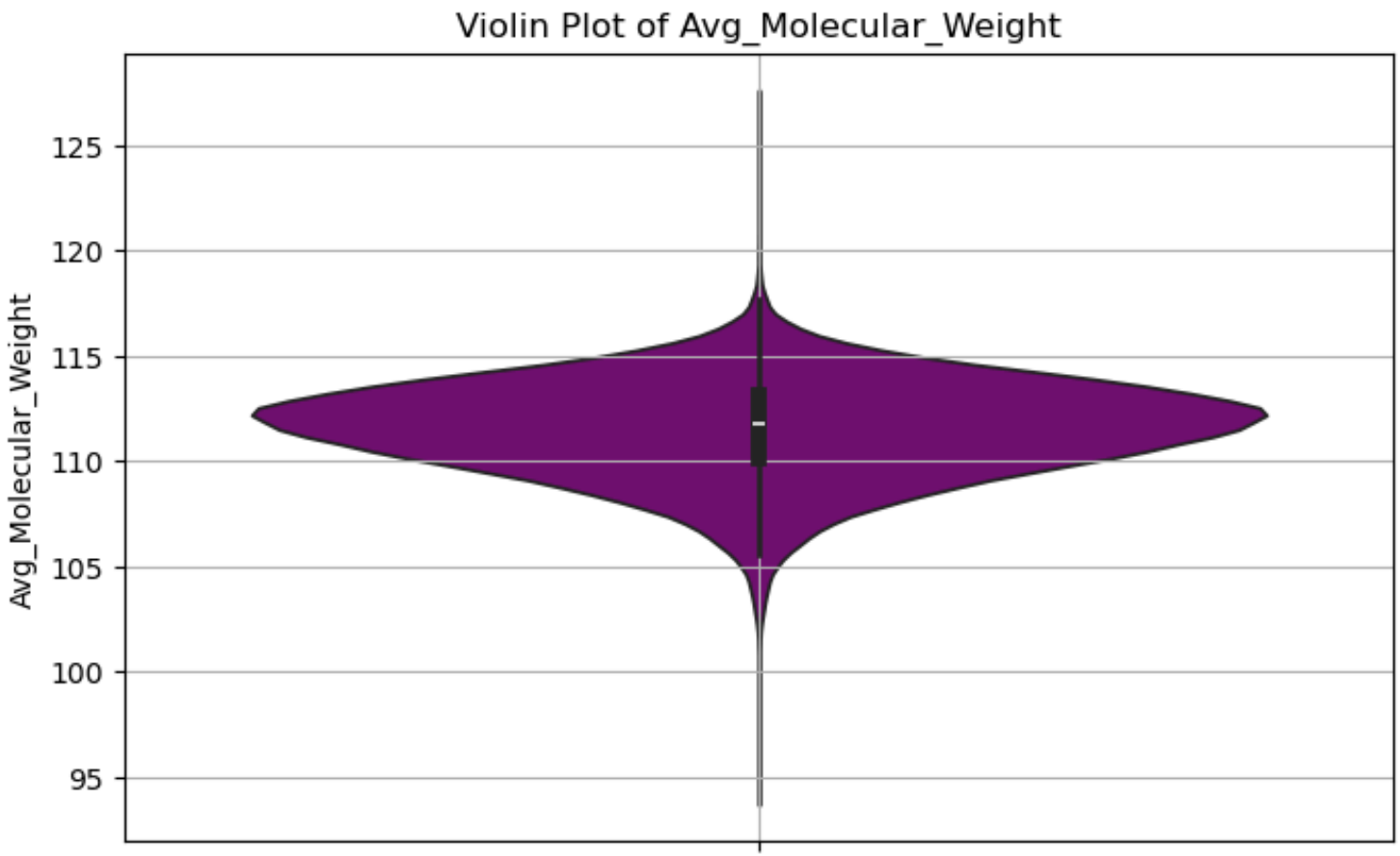

Violin Plot of Aliphatic_Index
Aliphatic_Index
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05

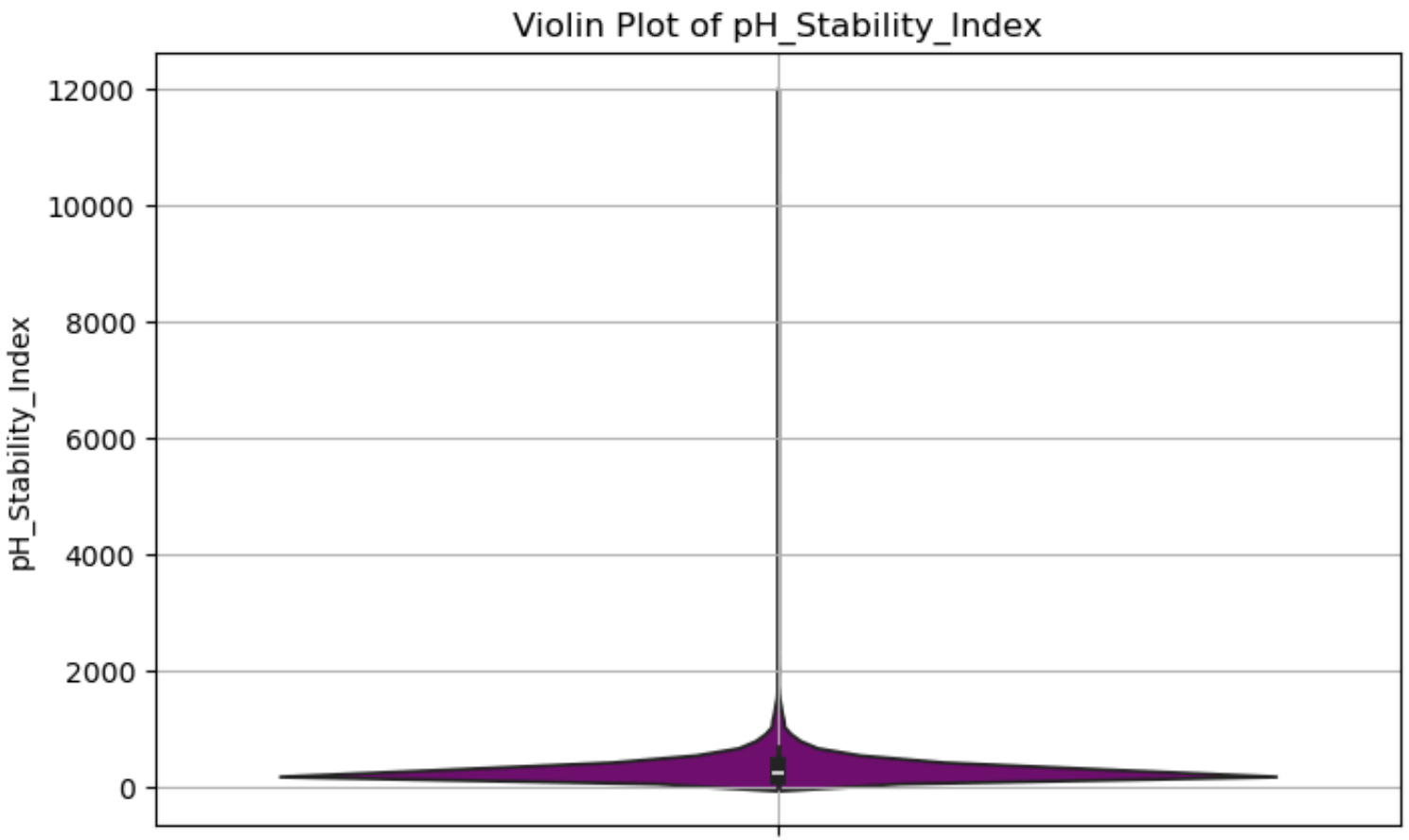

Violin Plot of Sequence_Similarity_Score
Sequence_Similarity_Score
1.0
0.8
0.6
0.4
0.2
0.0

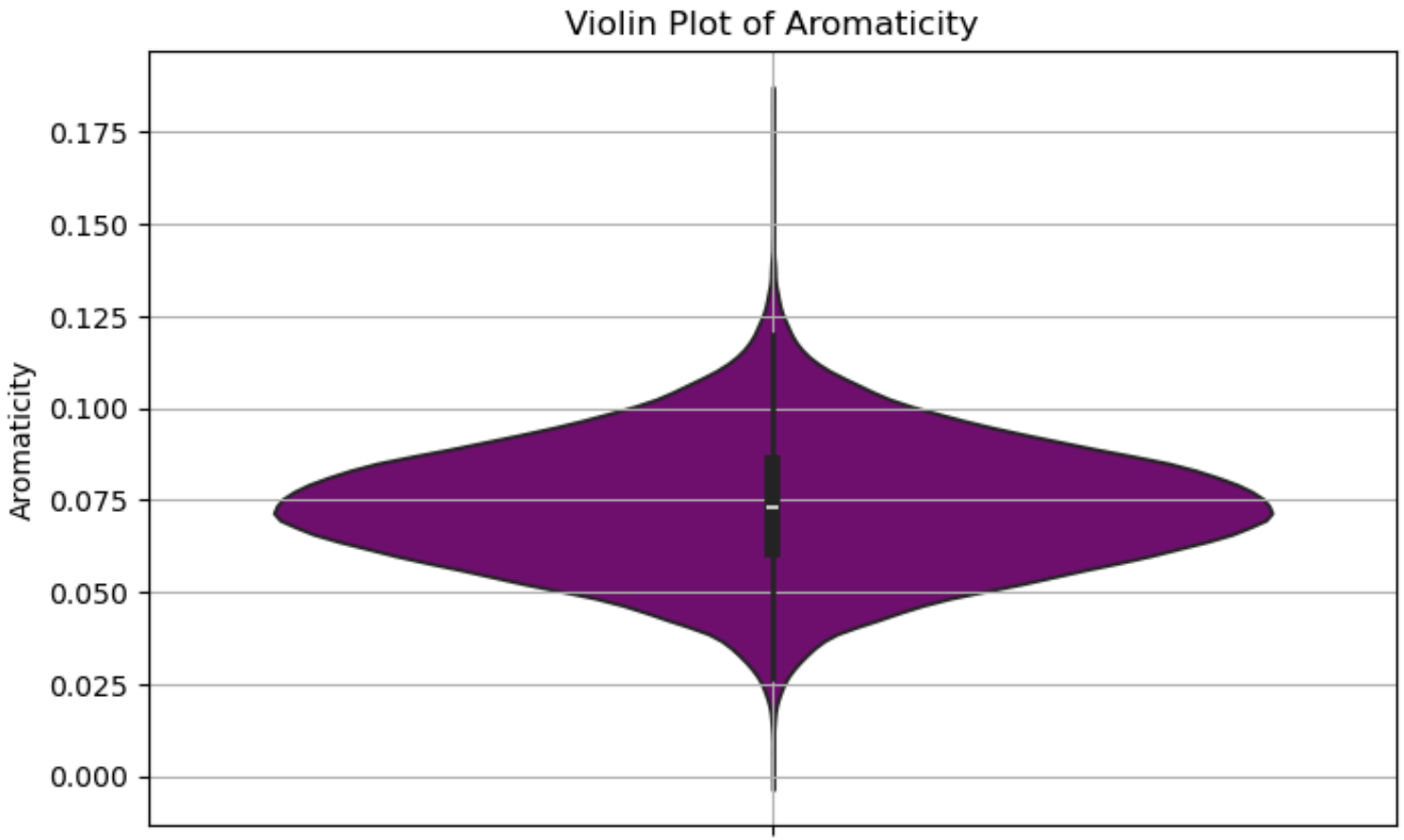

Violin Plot of Hydrogen_Bond_Ratio
Hydrogen_Bond_Ratio
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05

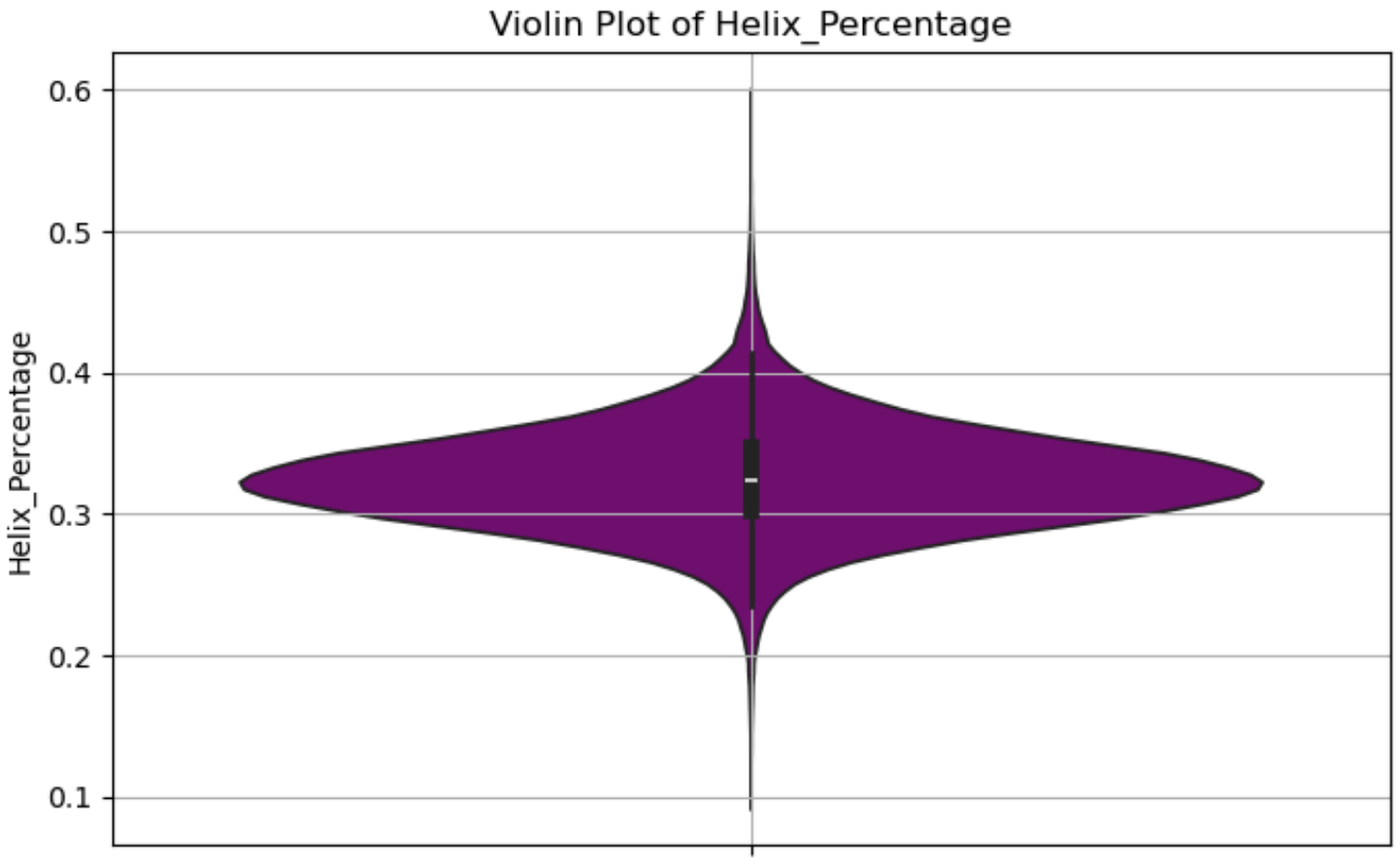

Violin Plot of Cysteine_Count
Cysteine_Count
700
600
500
400
300
200
100
0

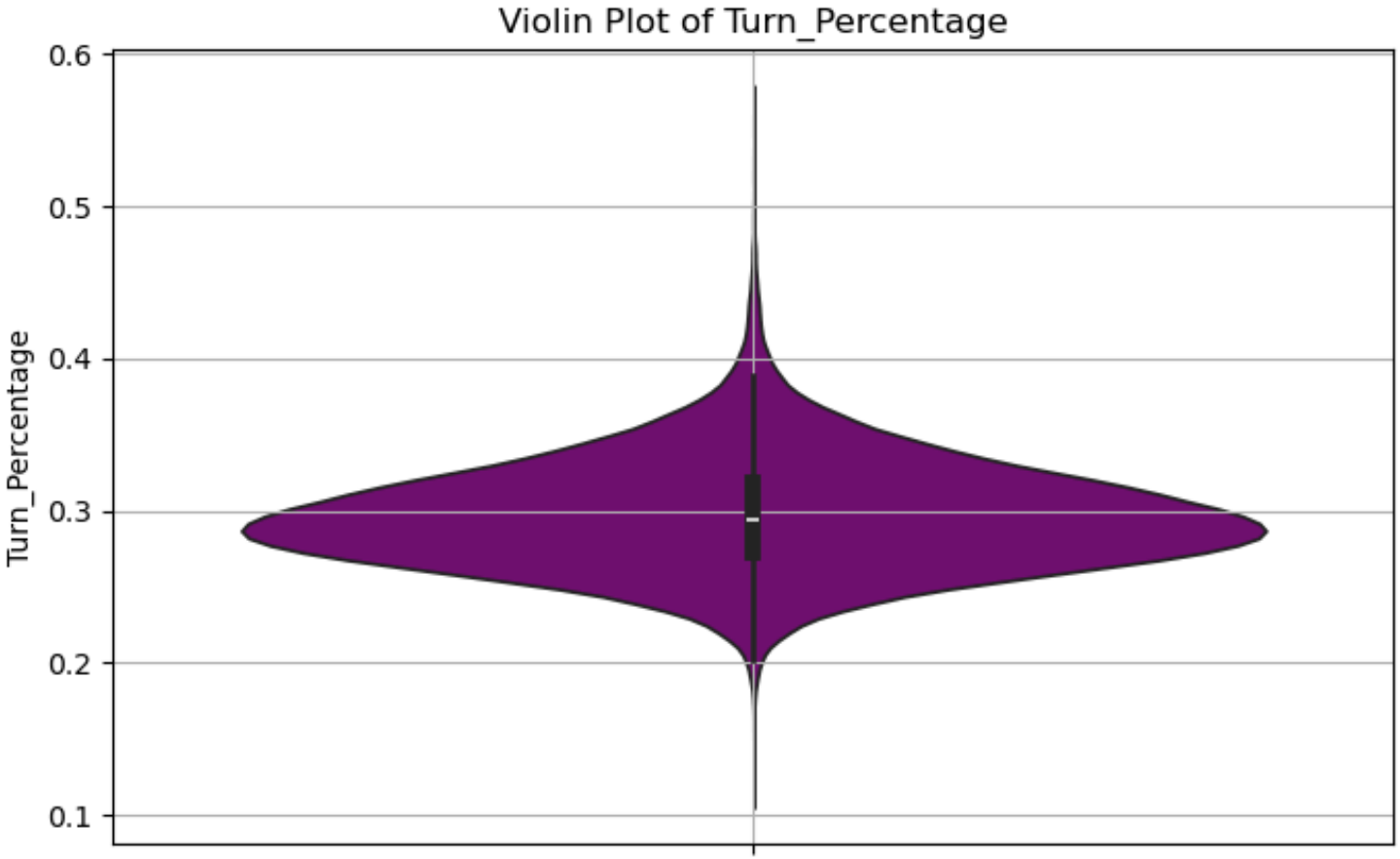

Violin Plot of Amino_Acid_Diversity
Amino_Acid_Diversity
21
20
19
18
17
16
15
14

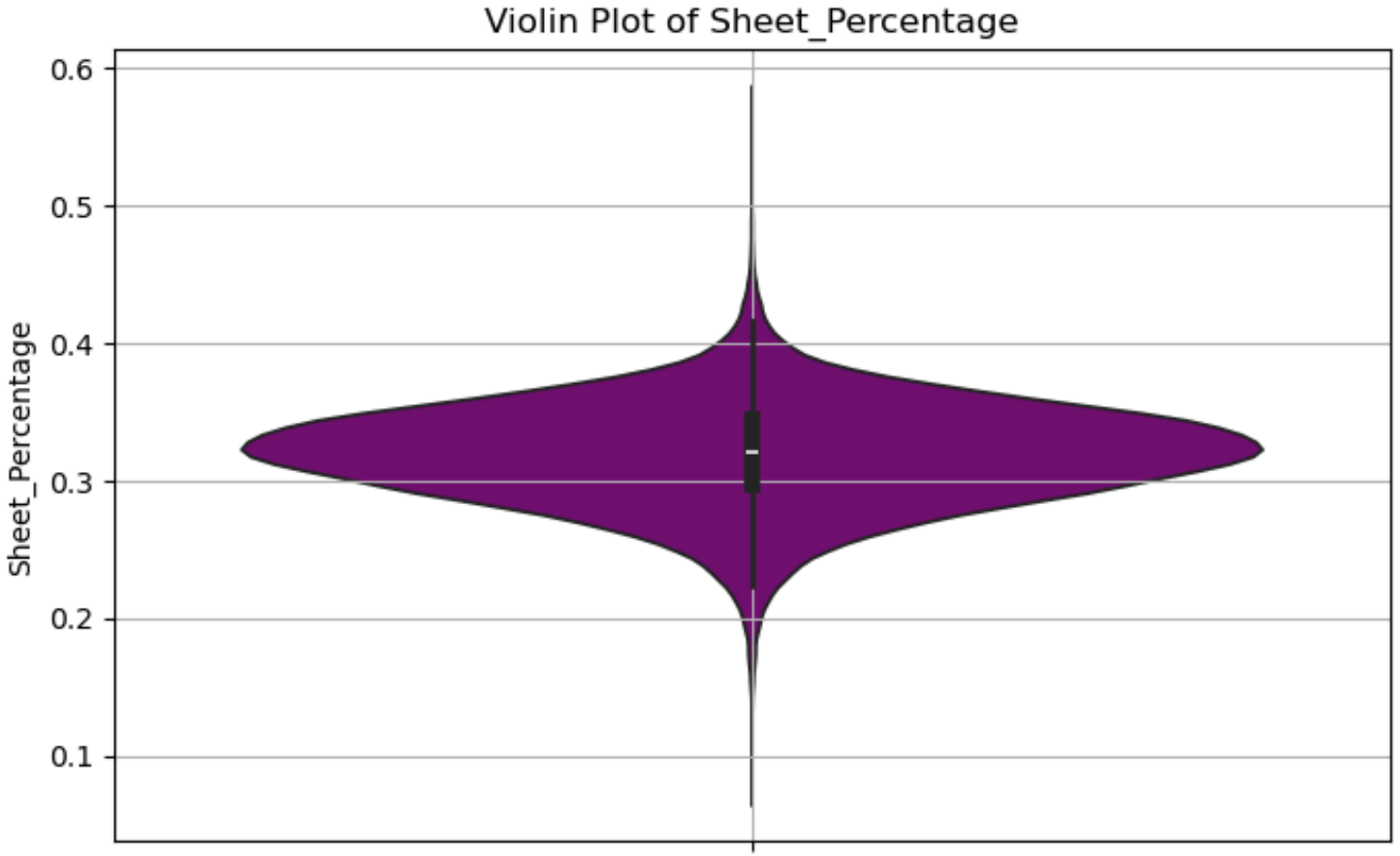

Violin Plot of Avg_Molecular_Weight
Avg_Molecular_Weight
125
120
115
110
105
100
95

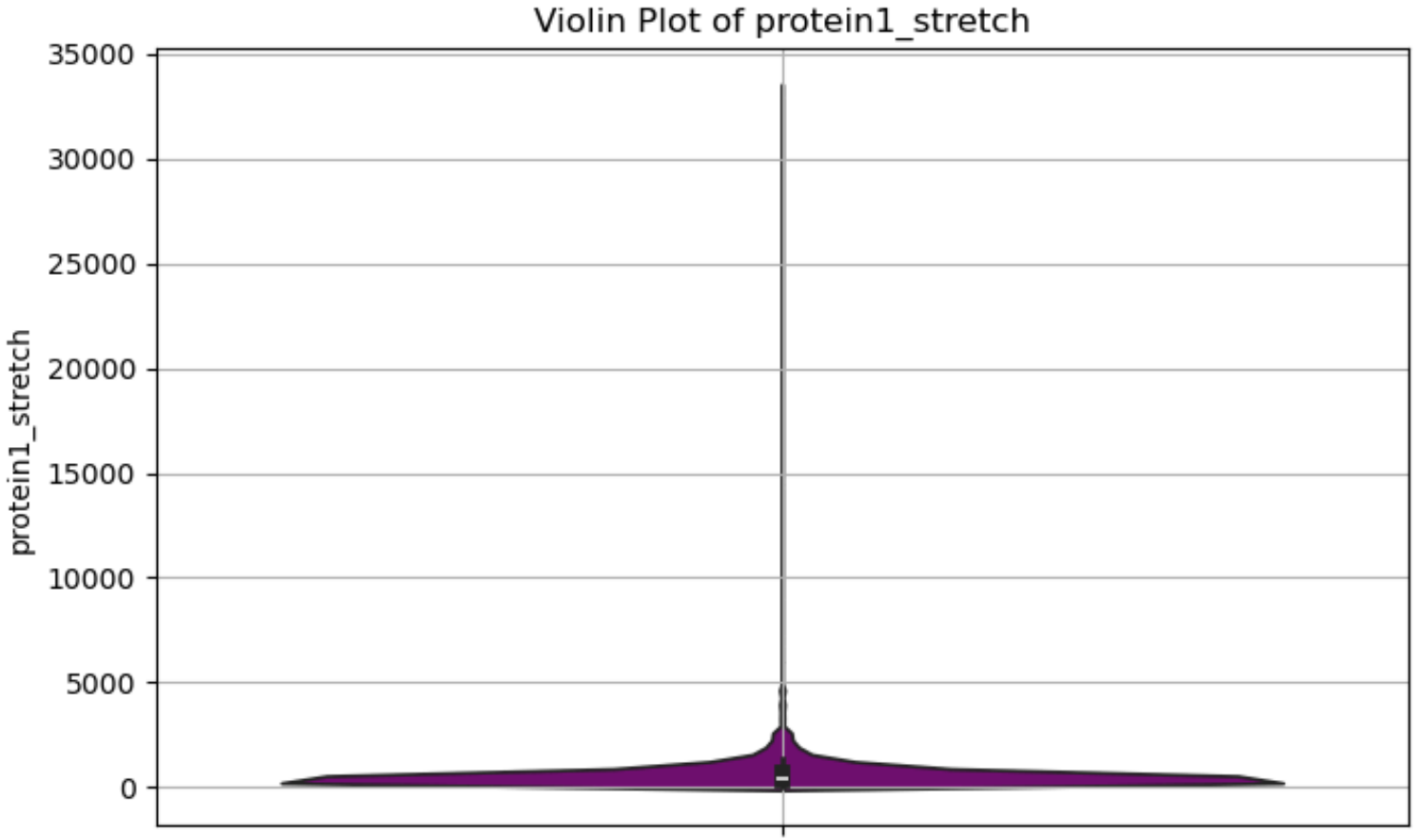

Violin Plot of pH_Stability_Index
pH_Stability_Index
12000
10000
8000
6000
4000
2000
0

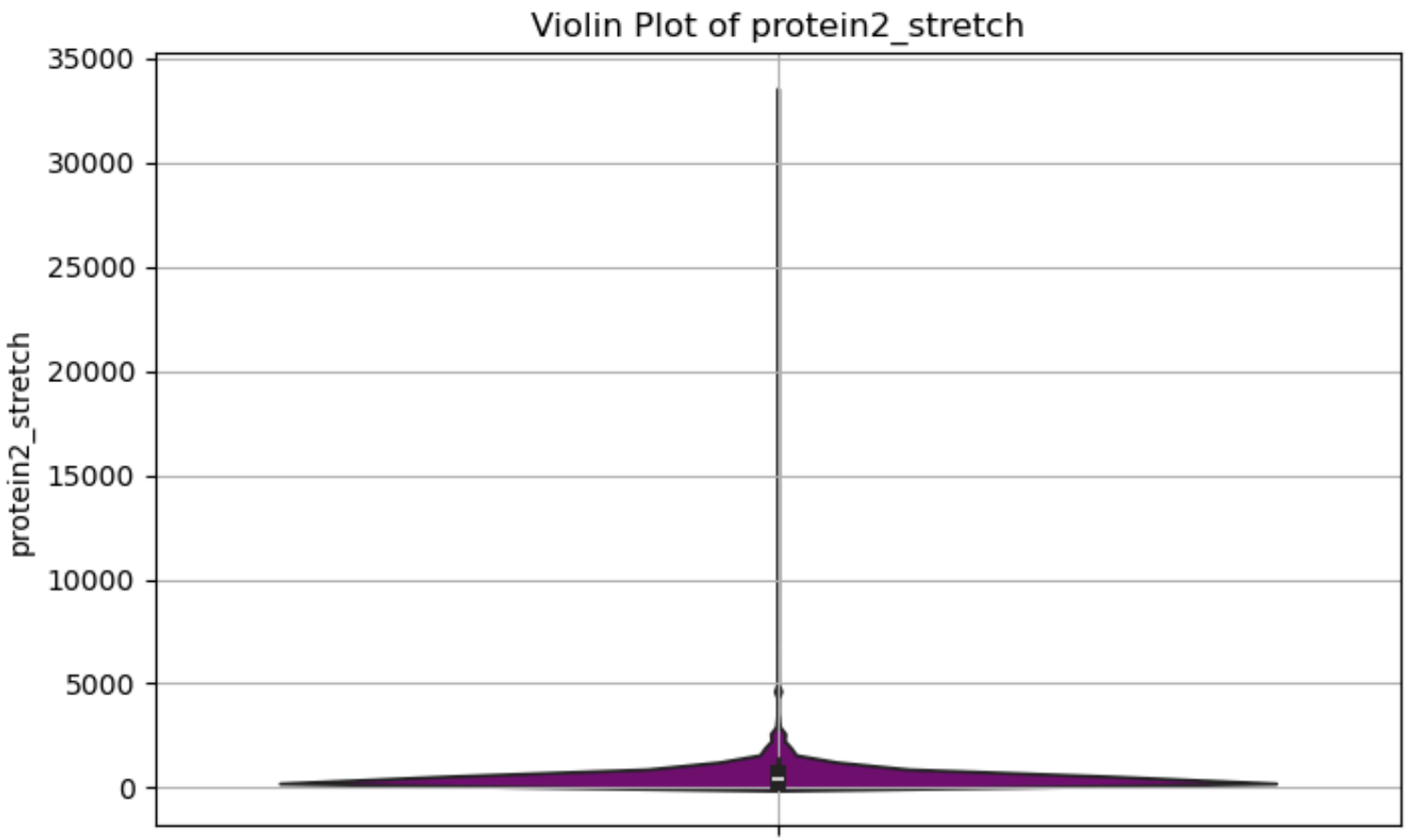

Violin Plot of Aromaticity
Aromaticity
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000


Violin Plot of Helix_Percentage
Helix_Percentage
0.6
0.5
0.4
0.3
0.2
0.1

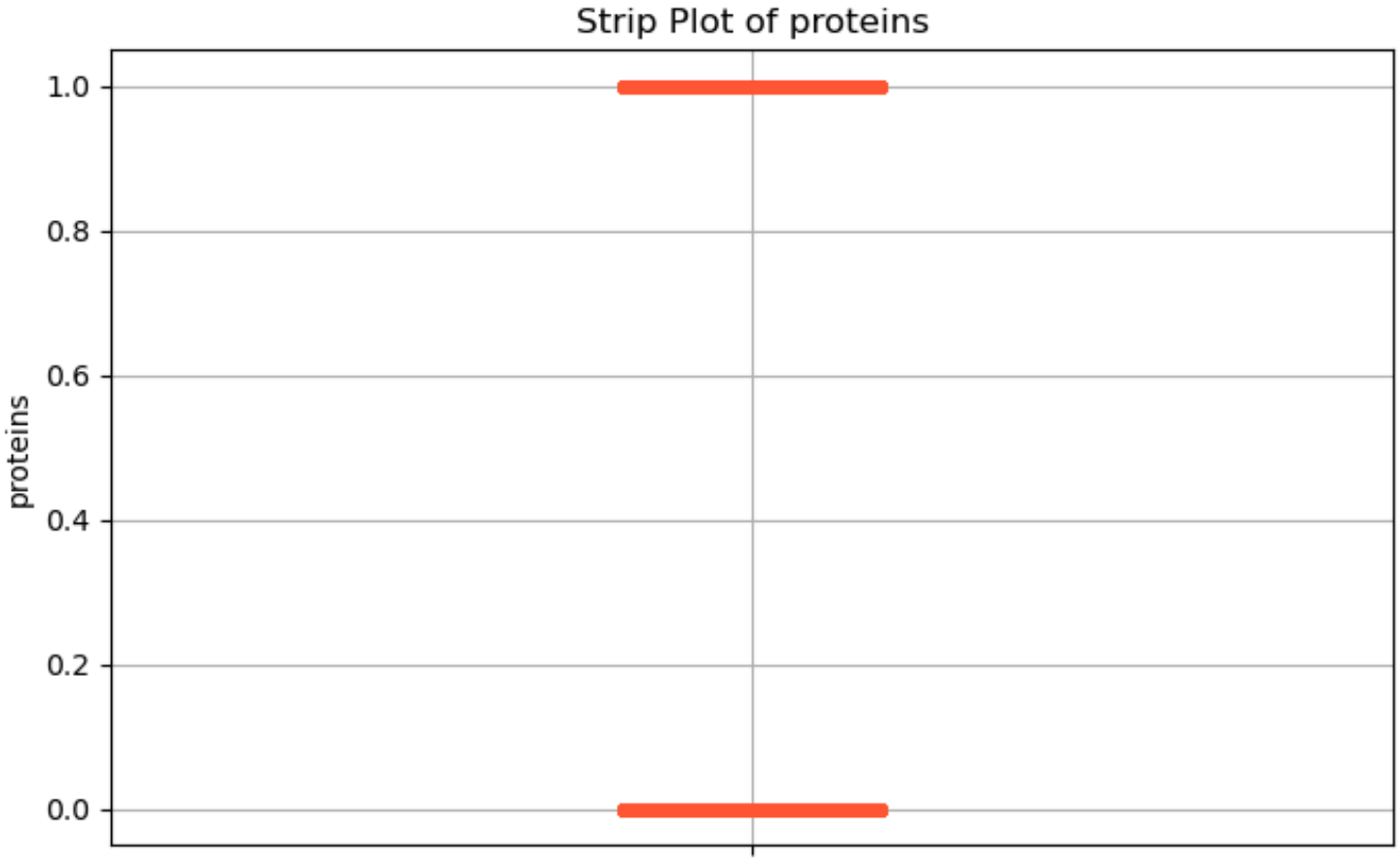

Violin Plot of Turn_Percentage
Turn_Percentage
0.6
0.5
0.4
0.3
0.2
0.1

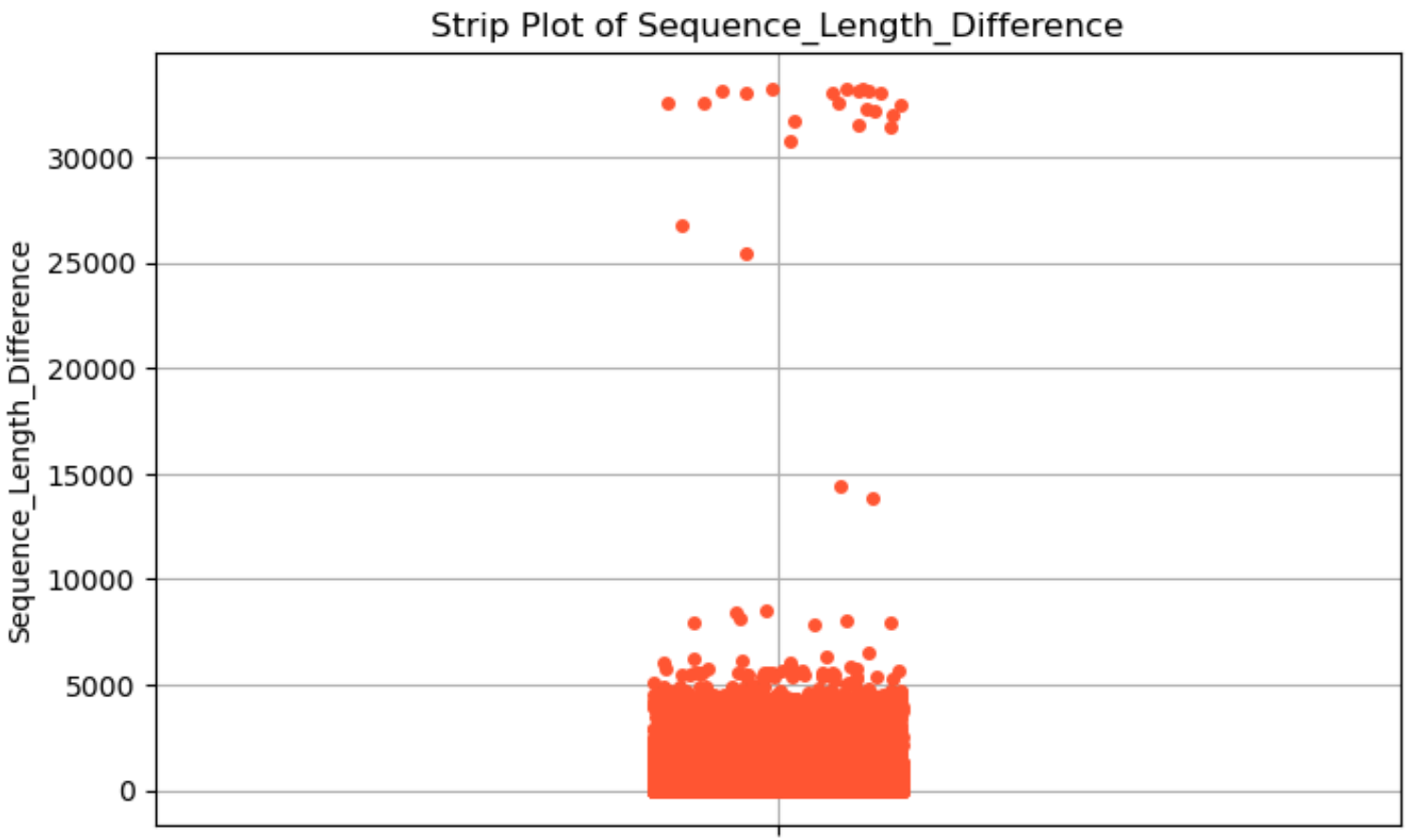

Violin Plot of Sheet_Percentage
Sheet_Percentage
0.6
0.5
0.4
0.3
0.2
0.1

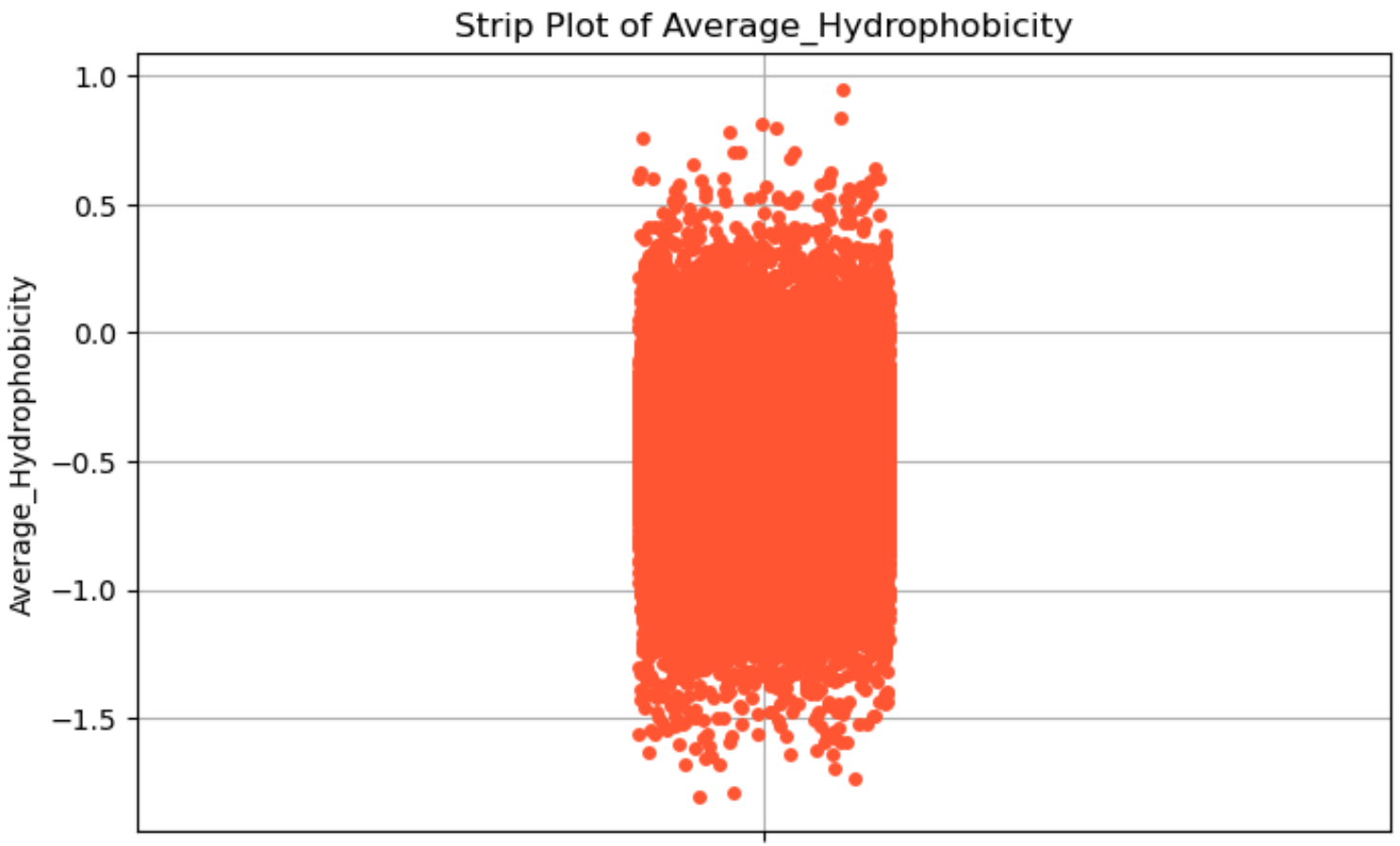

Violin Plot of protein1_stretch
protein1_stretch
35000
30000
25000
20000
15000
10000
5000
0

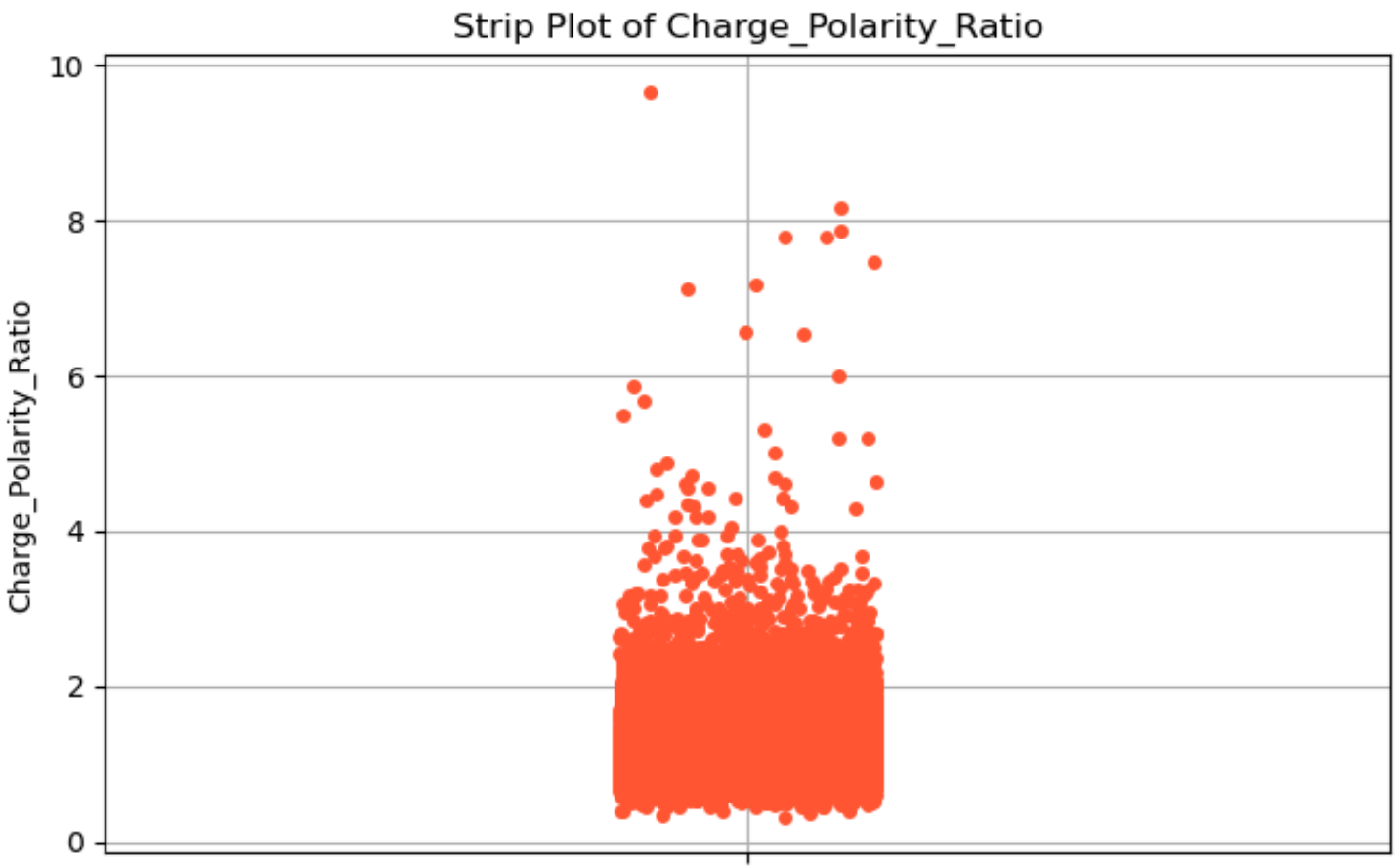


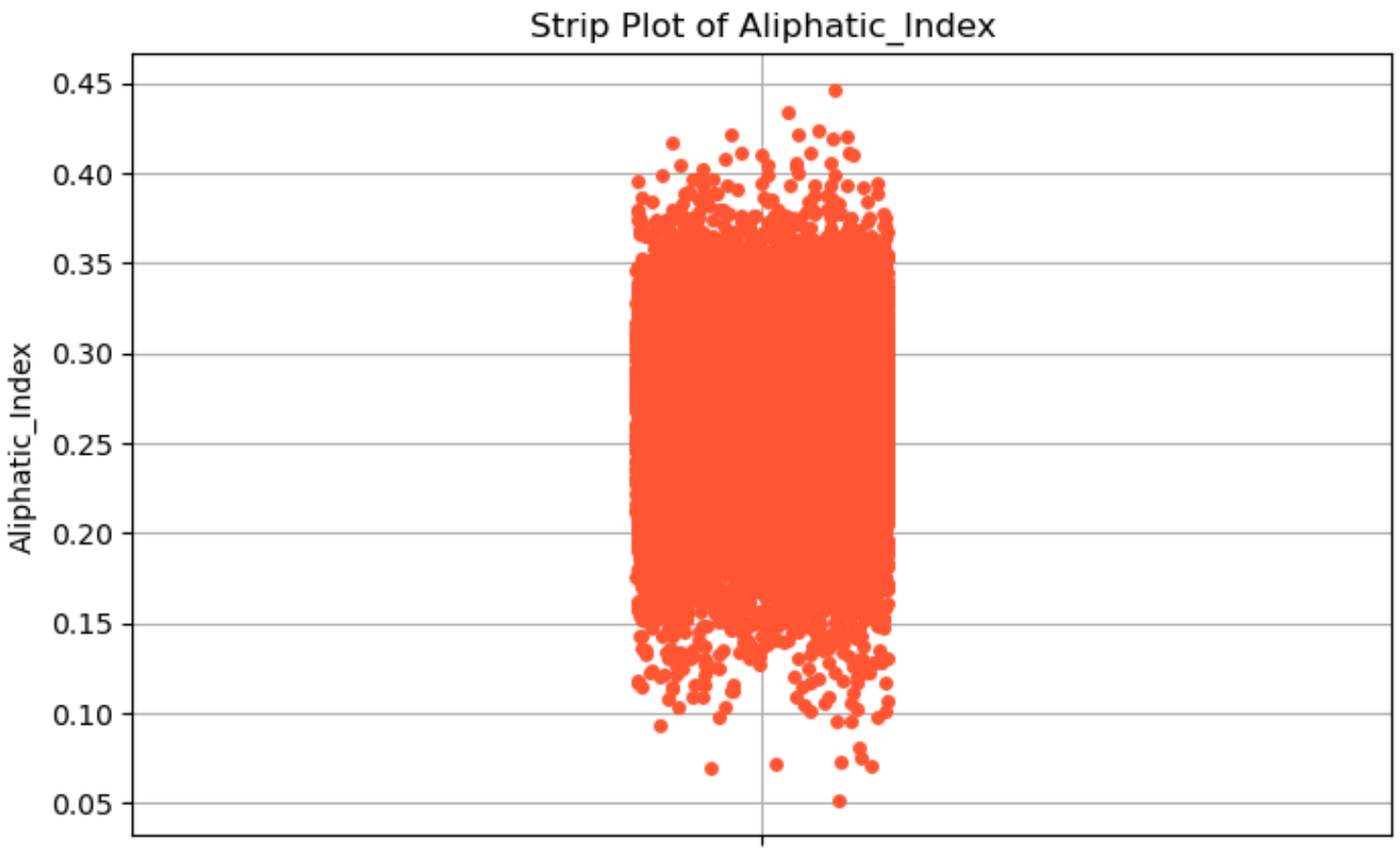
In [55]:
```
Num_Col = protein_df.select_dtypes(include=['int64','float64']).columns
for col in Num_Col:
    plt.figure(figsize=(8, 5))
    sns.stripplot(y=protein_df[col], color='#FF5733', jitter=True)
    plt.title(f'Strip Plot of {col}')
    plt.ylabel(col)
    plt.grid(True)
    plt.show()
```


Strip Plot of proteins
proteins
1.0
0.8
0.6
0.4
0.2
0.0

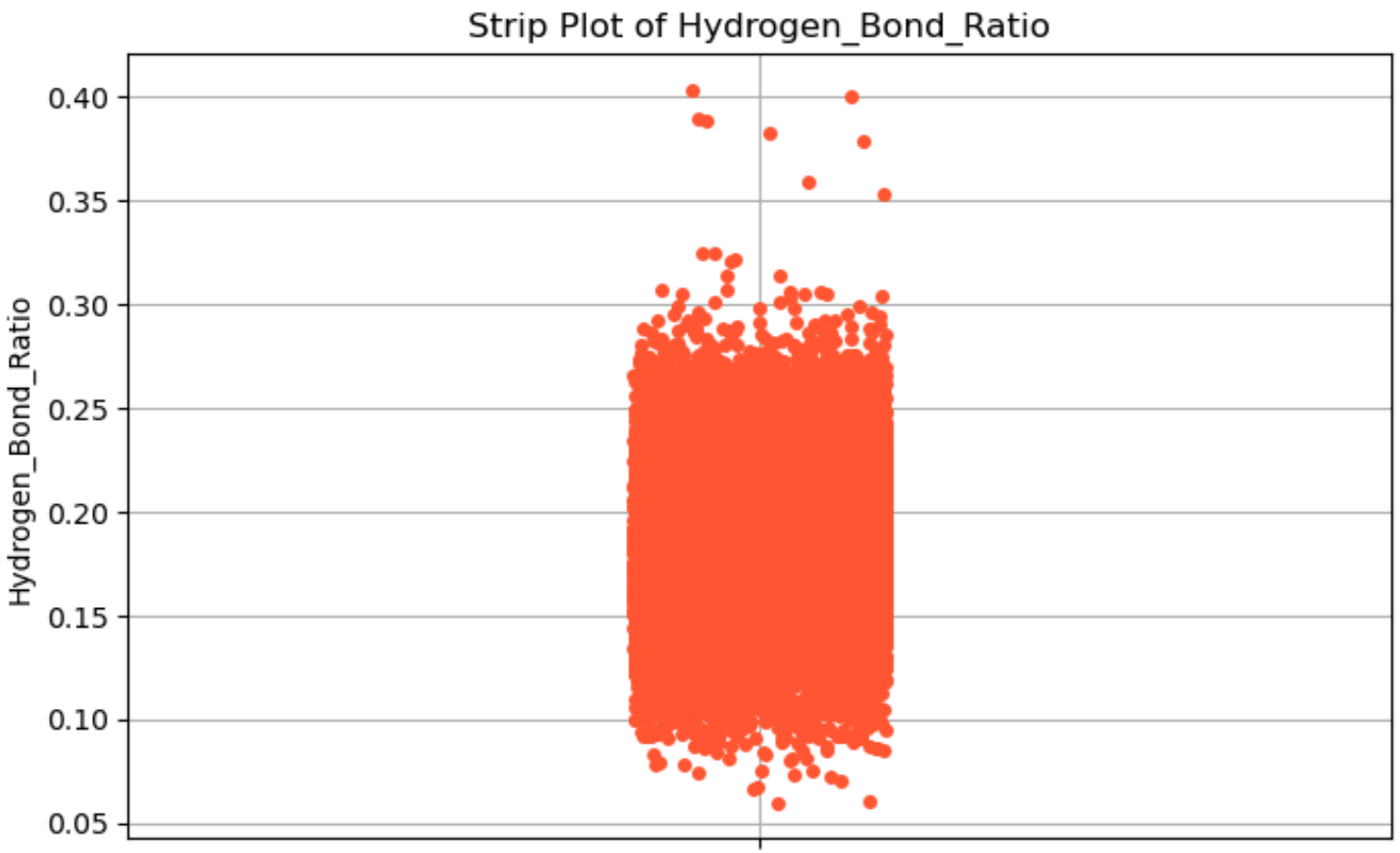

Strip Plot of Sequence_Length_Difference
Sequence_Length_Difference
30000
25000
20000
15000
10000
5000
0

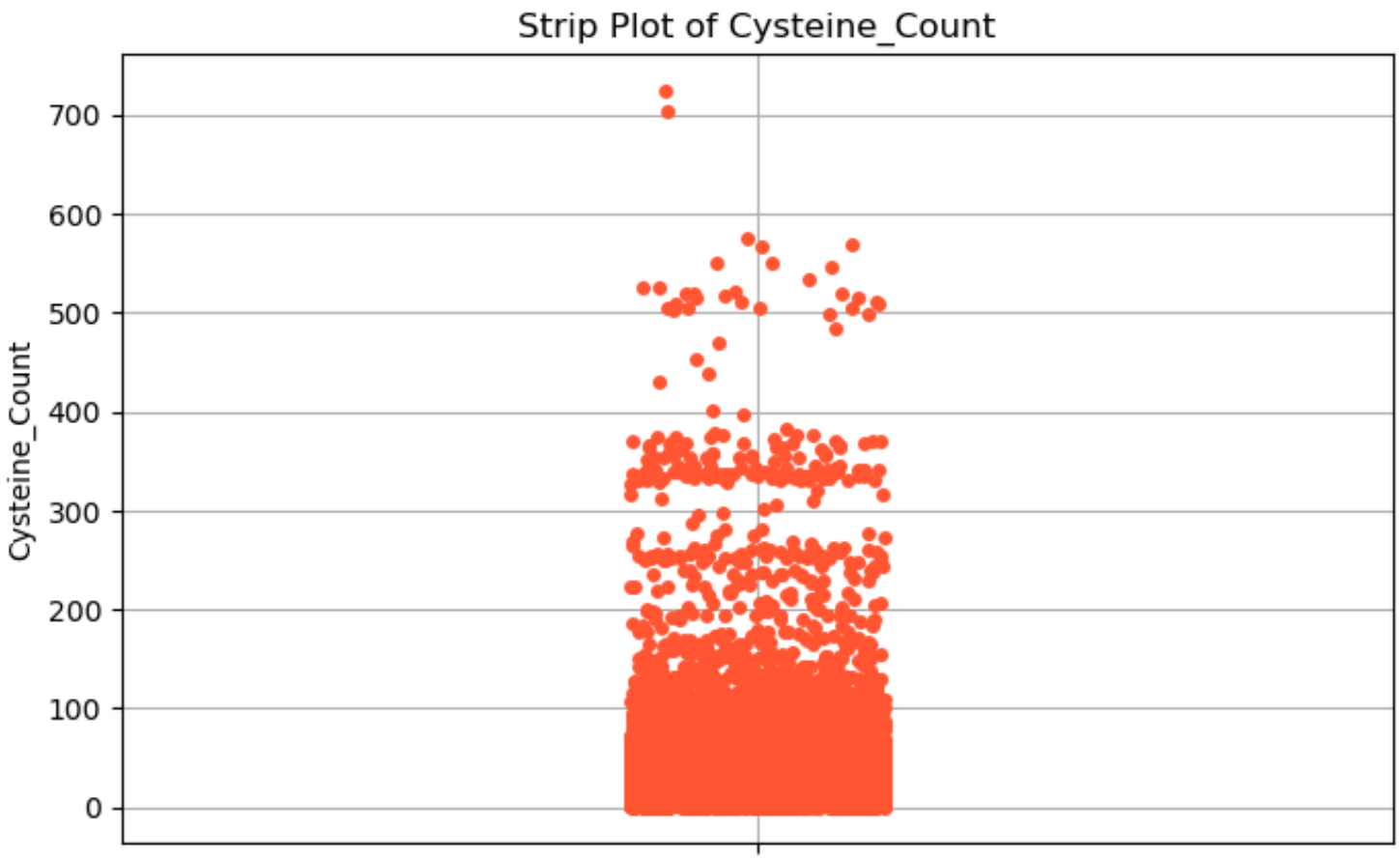

Strip Plot of Average_Hydrophobicity
Average_Hydrophobicity
1.0
0.5
0.0
−0.5
−1.0
−1.5

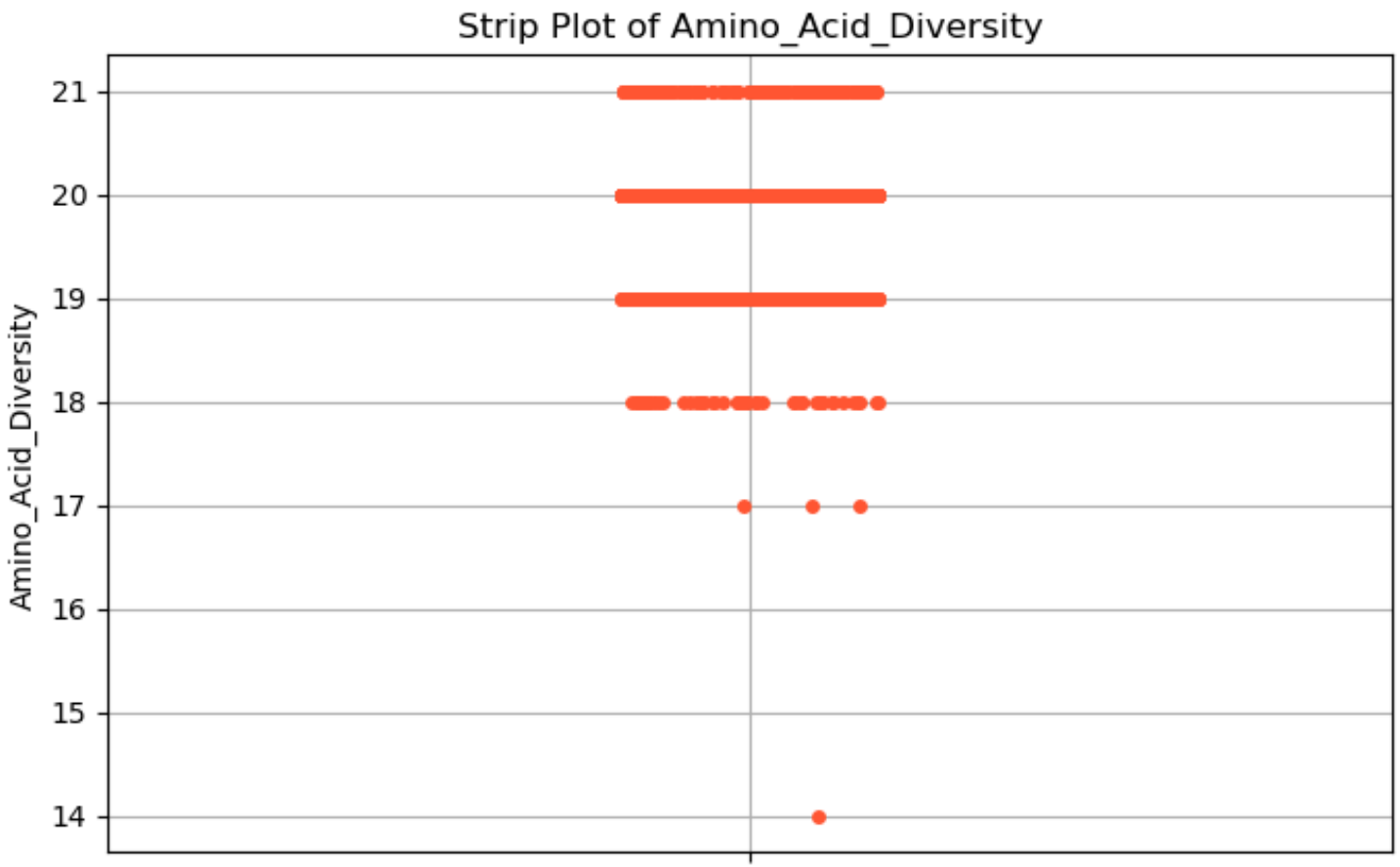

Strip Plot of Charge_Polarity_Ratio
Charge_Polarity_Ratio
10
8
6
4
2
0

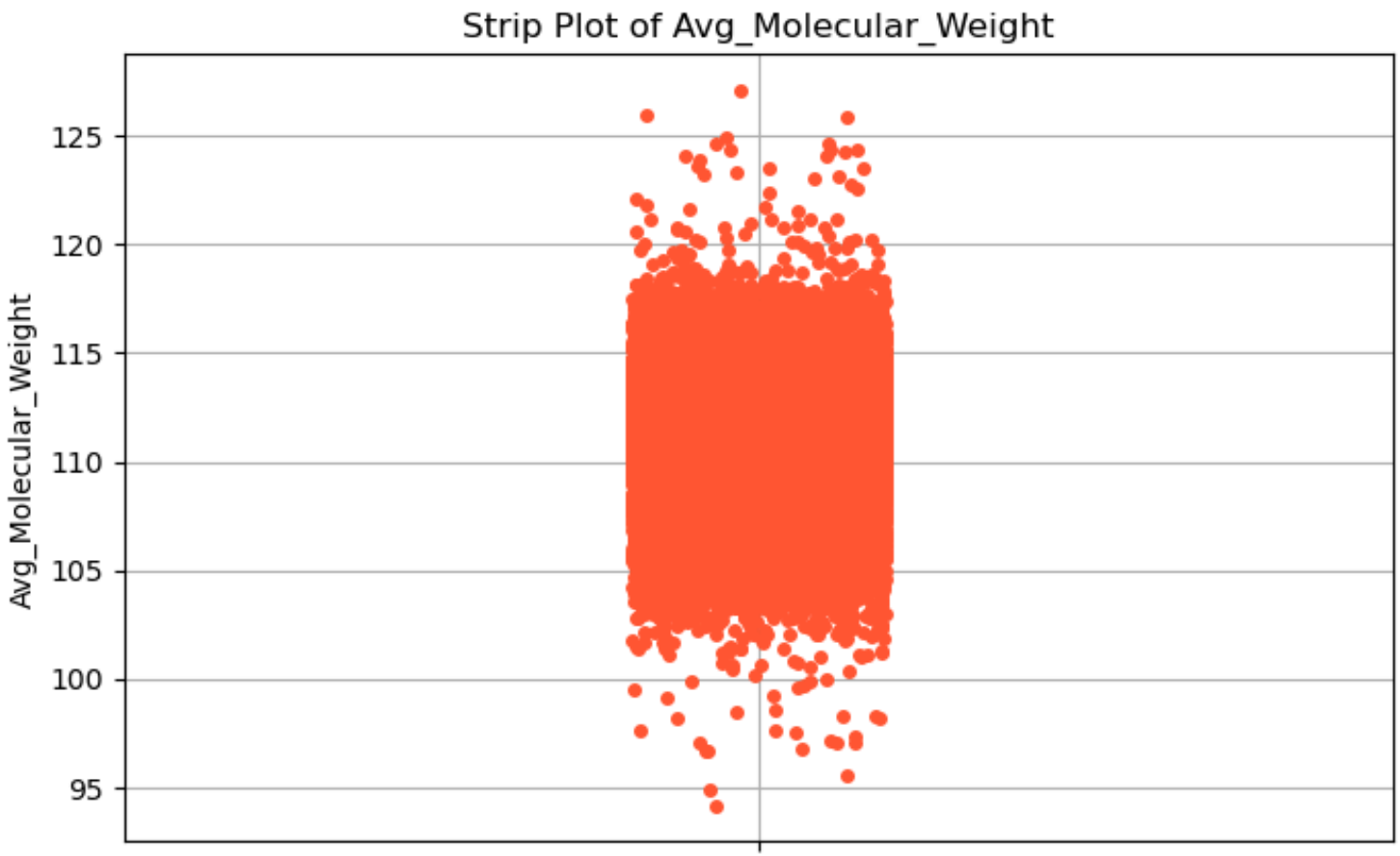

Strip Plot of Aliphatic_Index
Aliphatic_Index
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05

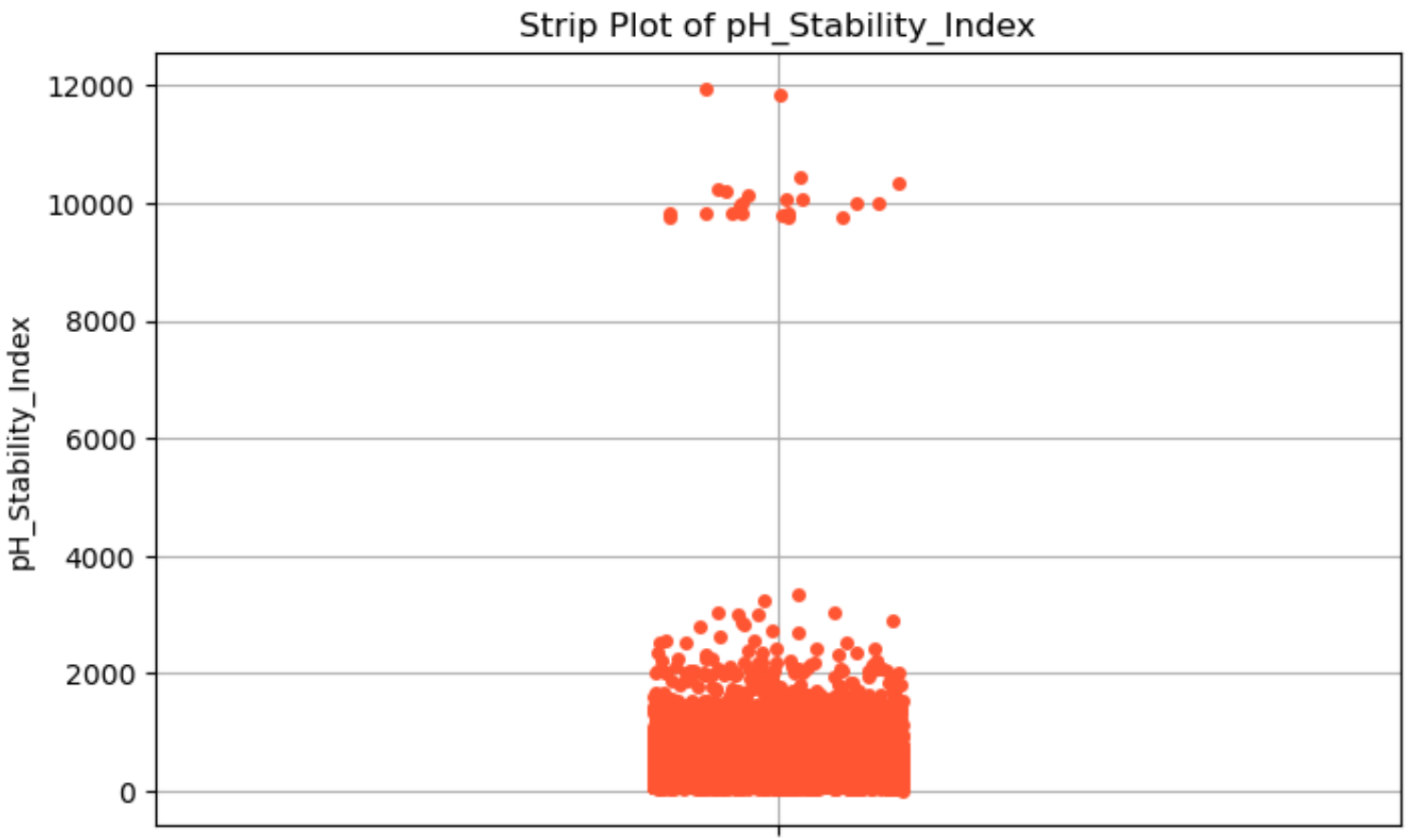

Strip Plot of Sequence_Similarity_Score
Sequence_Similarity_Score
1.0
0.8
0.6
0.4
0.2
0.0

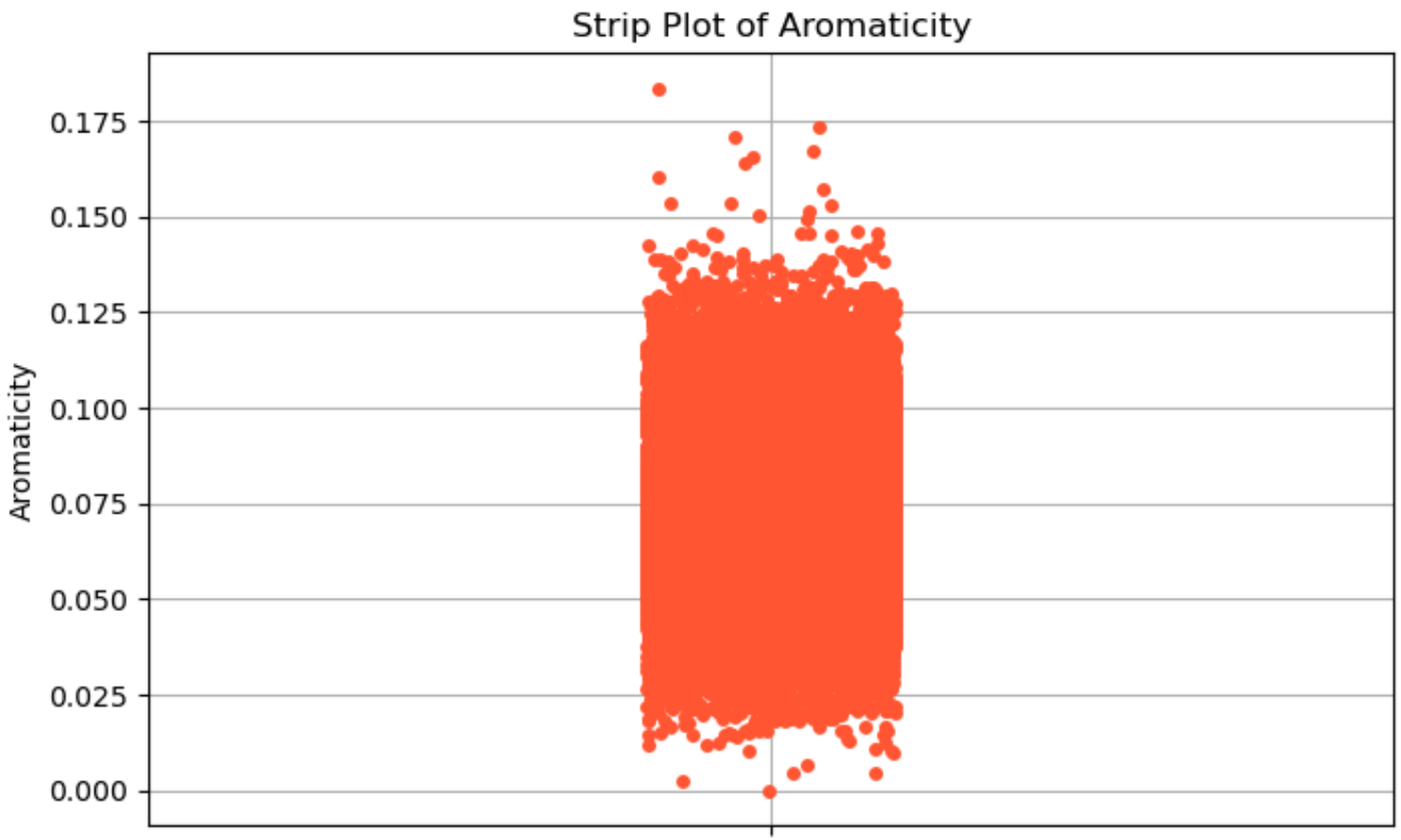

Strip Plot of Hydrogen_Bond_Ratio
Hydrogen_Bond_Ratio
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05

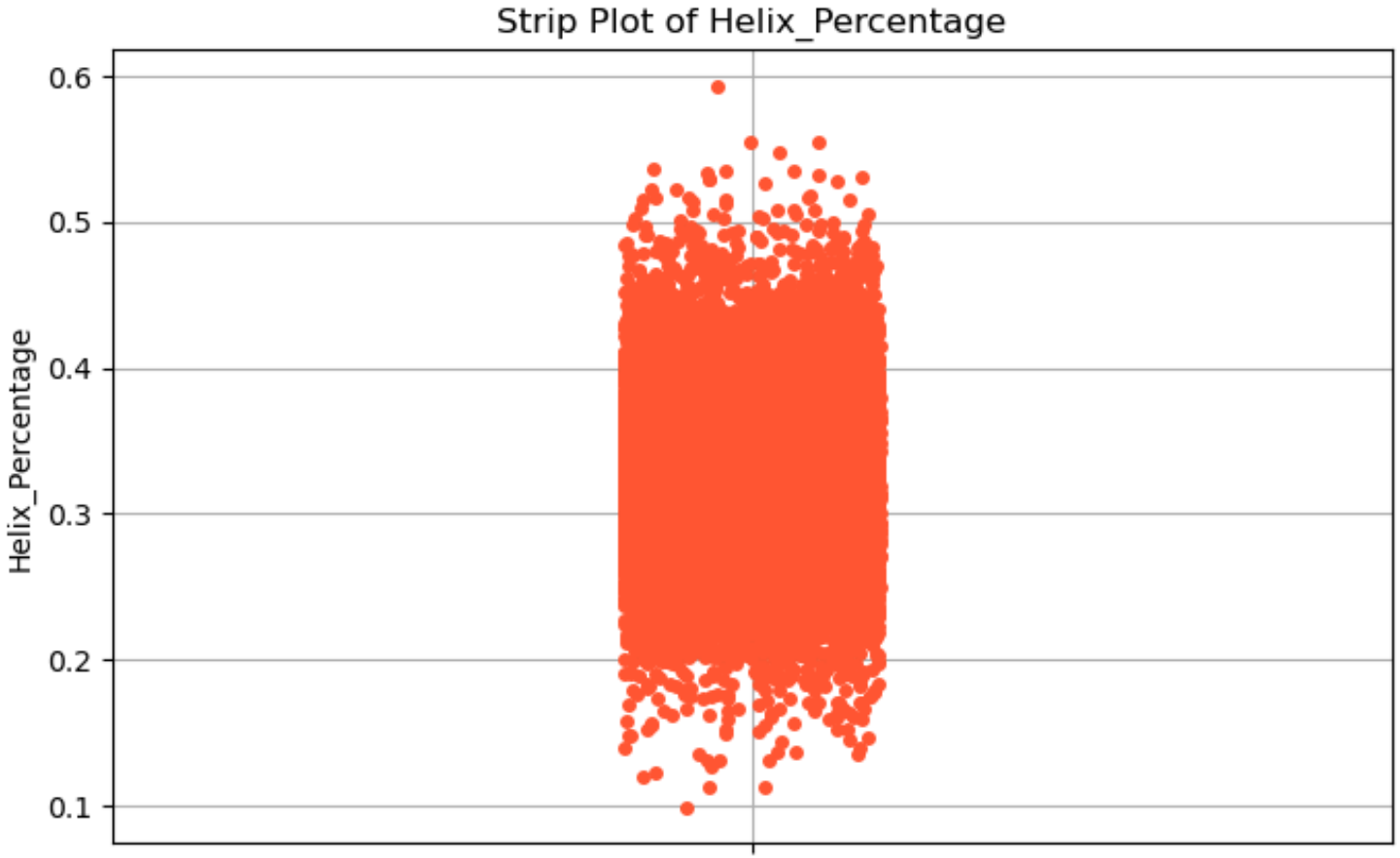

Strip Plot of Cysteine_Count
Cysteine_Count
700
600
500
400
300
200
100
0

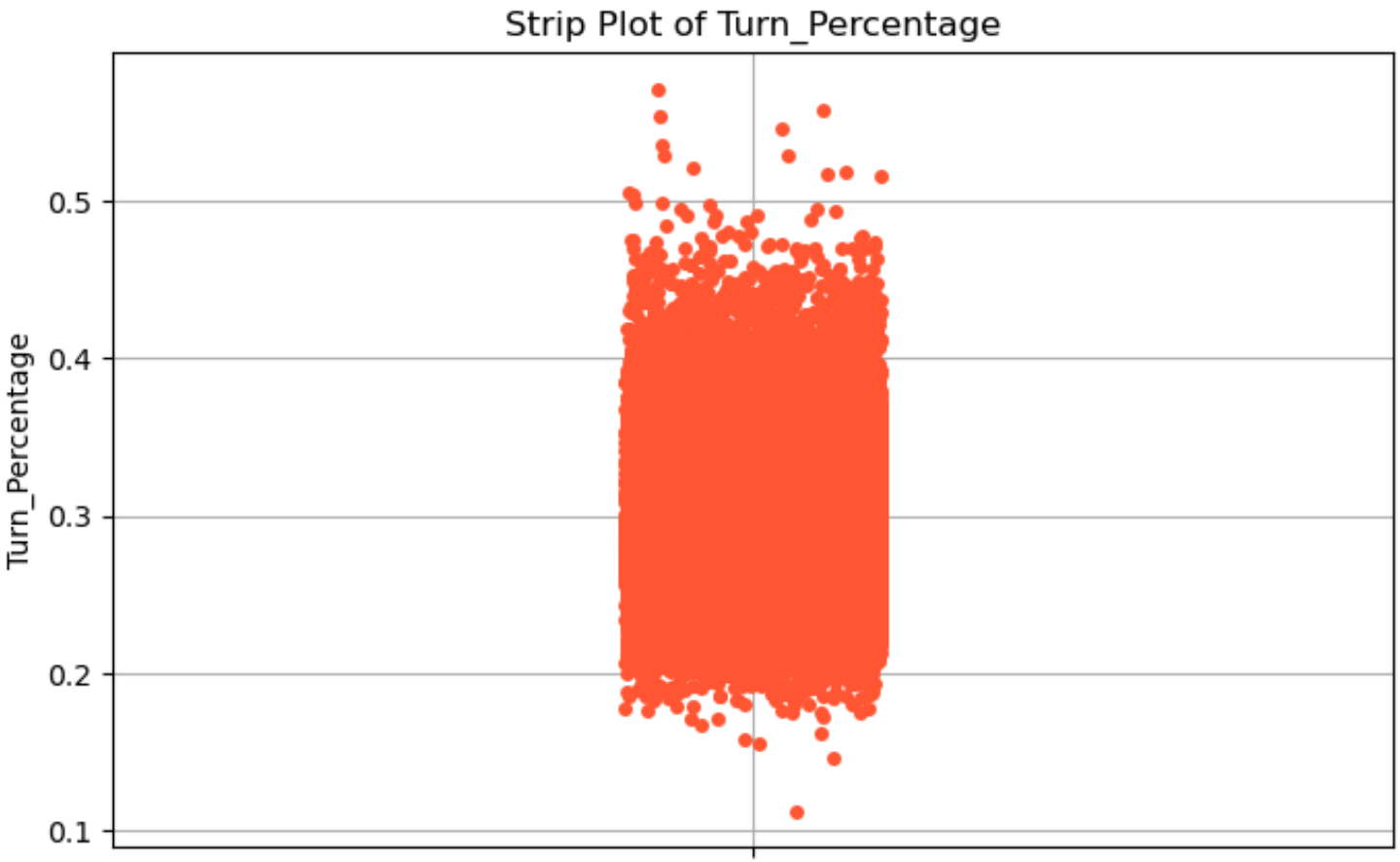

Strip Plot of Amino_Acid_Diversity
Amino_Acid_Diversity
21
20
19
18
17
16
15
14

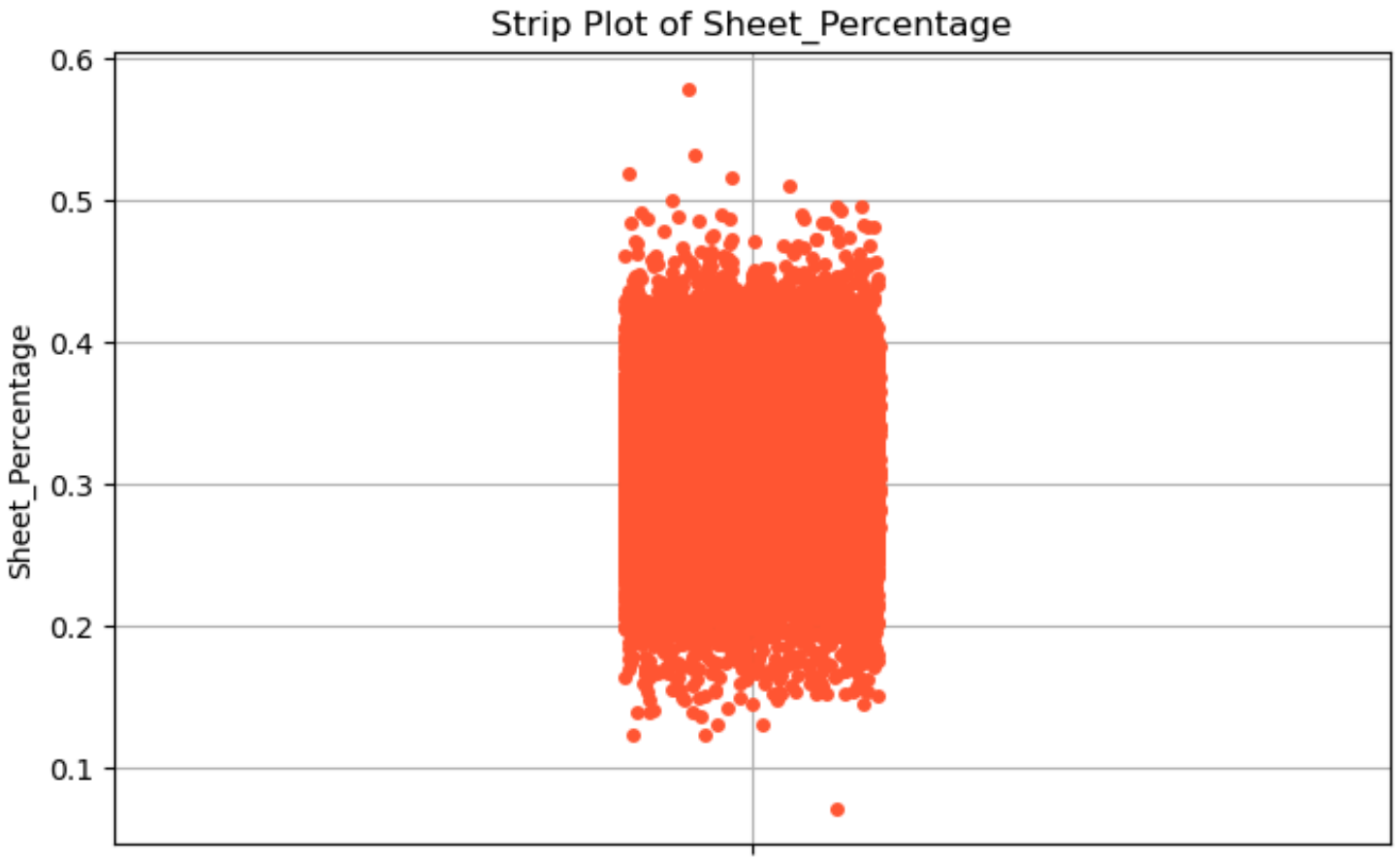

Strip Plot of Avg_Molecular_Weight
Avg_Molecular_Weight
125
120
115
110
105
100
95

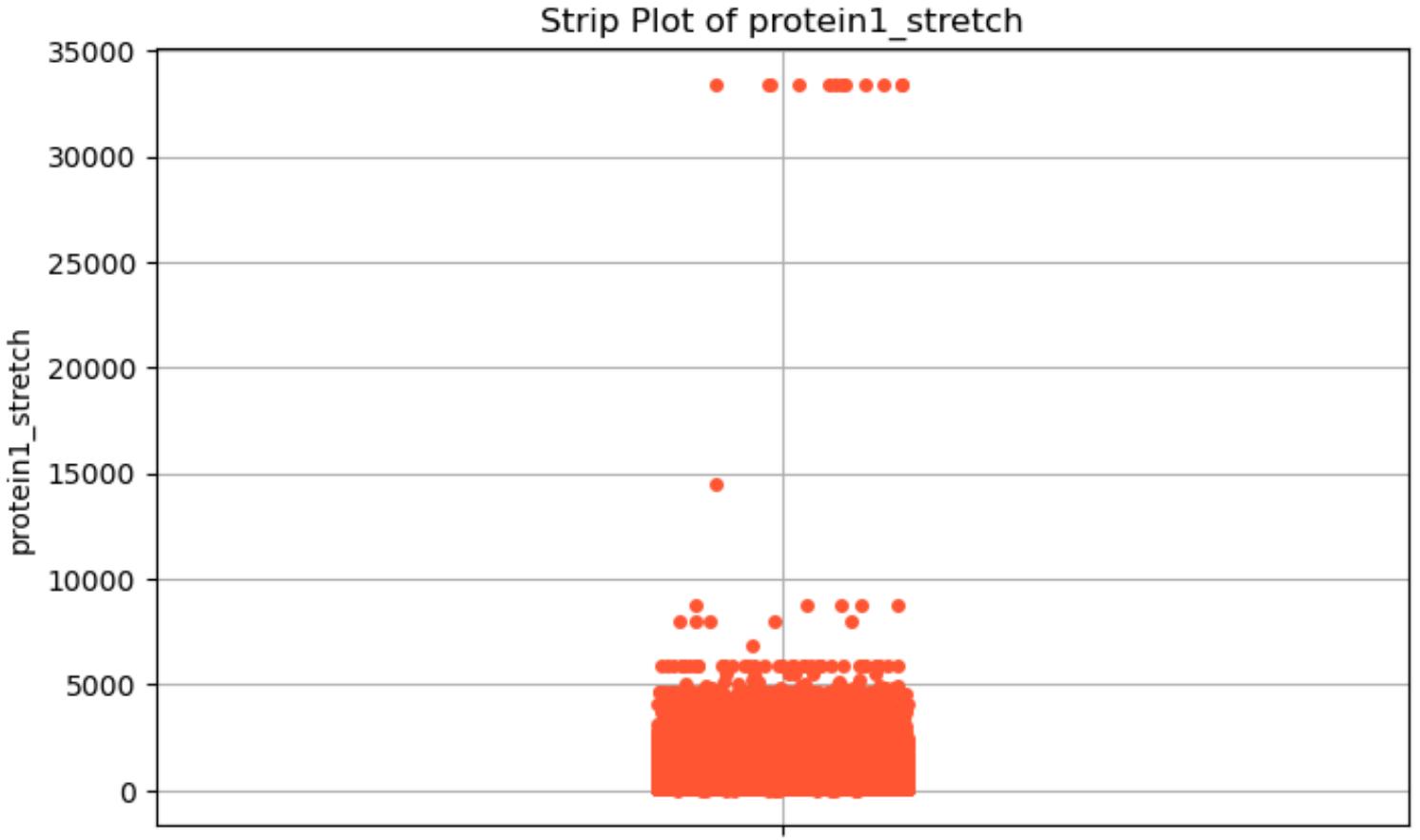

Strip Plot of pH_Stability_Index
pH_Stability_Index
12000
10000
8000
6000
4000
2000
0

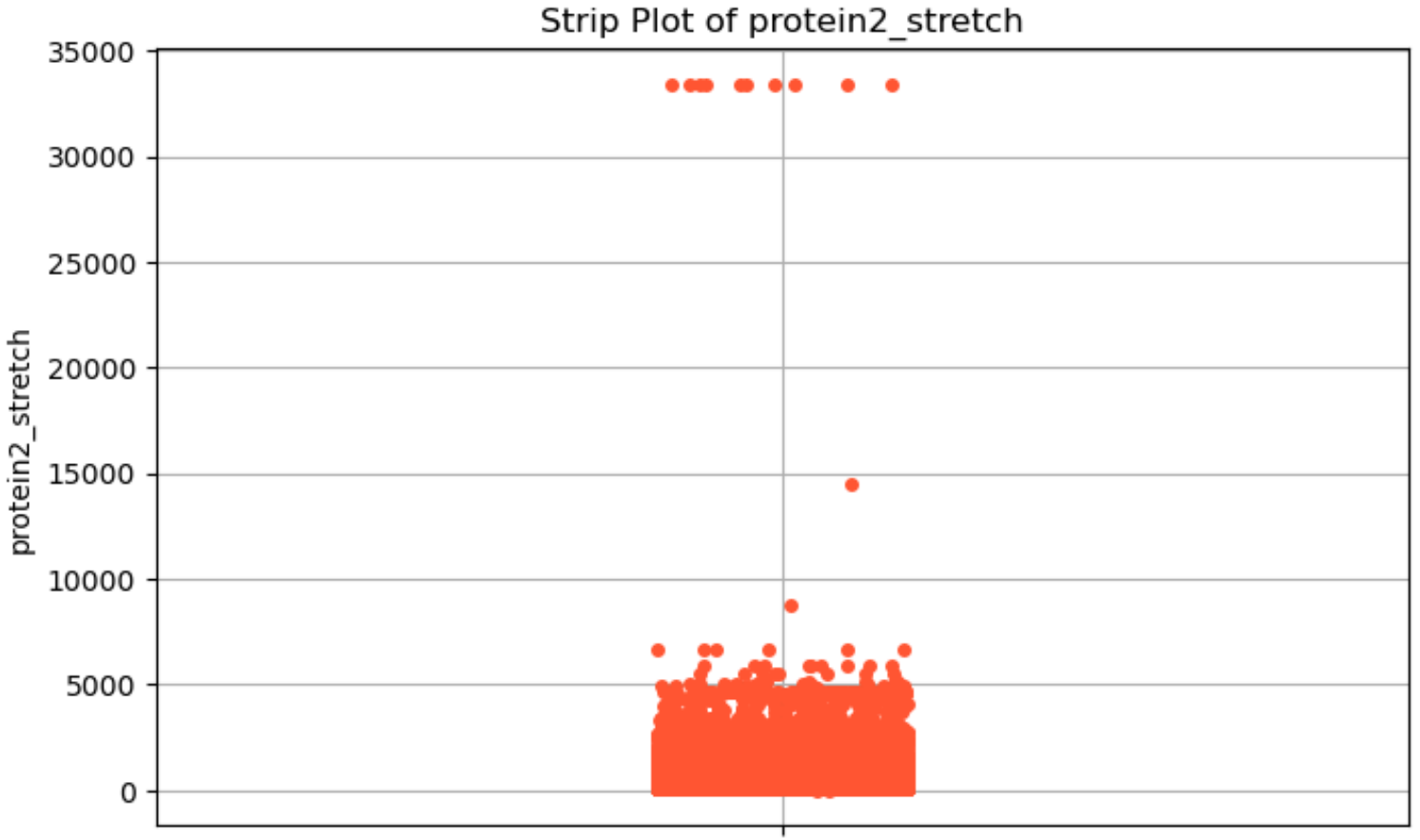

Strip Plot of Aromaticity
Aromaticity
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000


Strip Plot of Helix_Percentage
Helix_Percentage
0.6
0.5
0.4
0.3
0.2
0.1

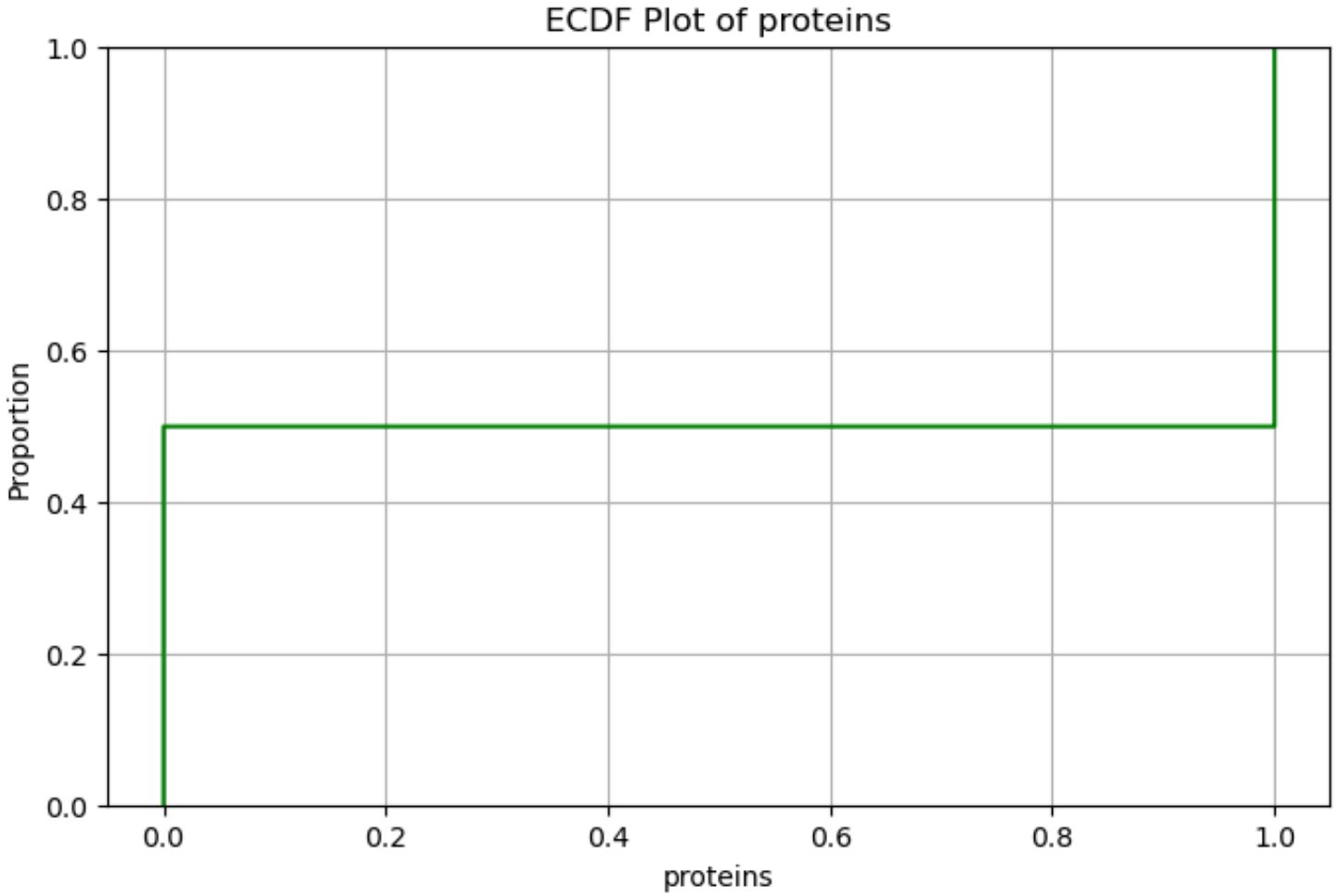

Strip Plot of Turn_Percentage
Turn_Percentage
0.5
0.4
0.3
0.2
0.1

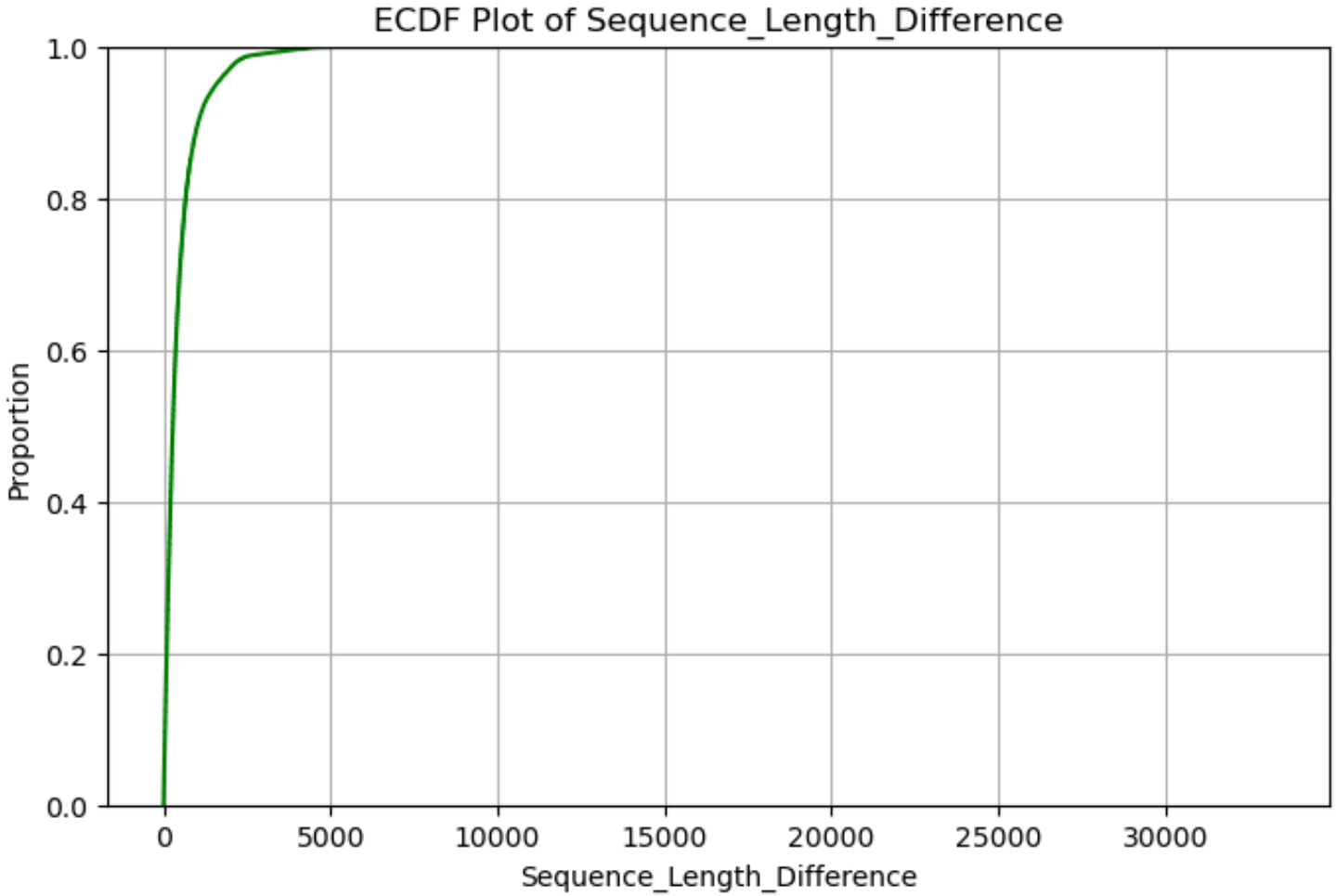

Strip Plot of Sheet_Percentage
Sheet_Percentage
0.6
0.5
0.4
0.3
0.2
0.1

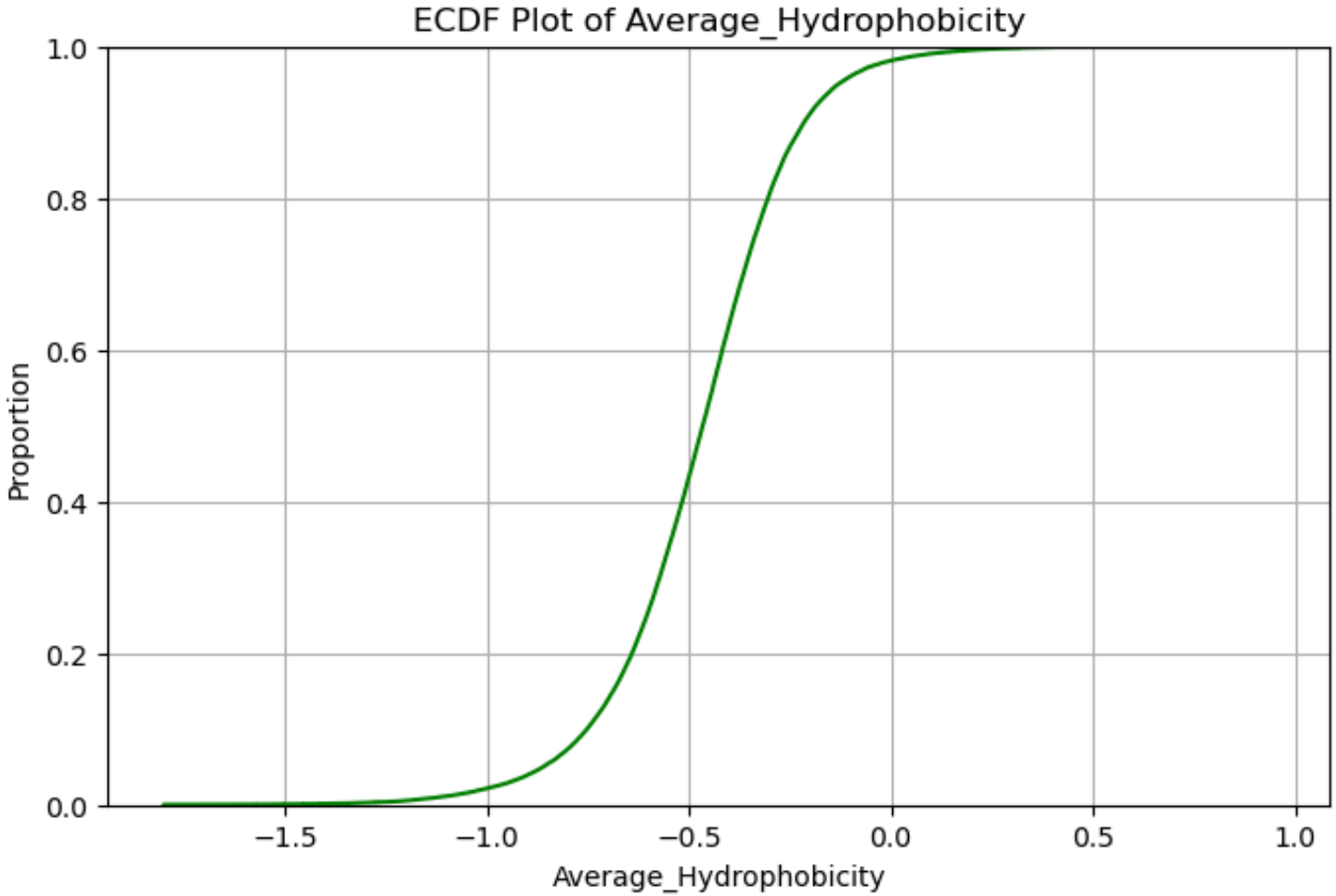

Strip Plot of protein1_stretch
protein1_stretch
35000
30000
25000
20000
15000
10000
5000
0

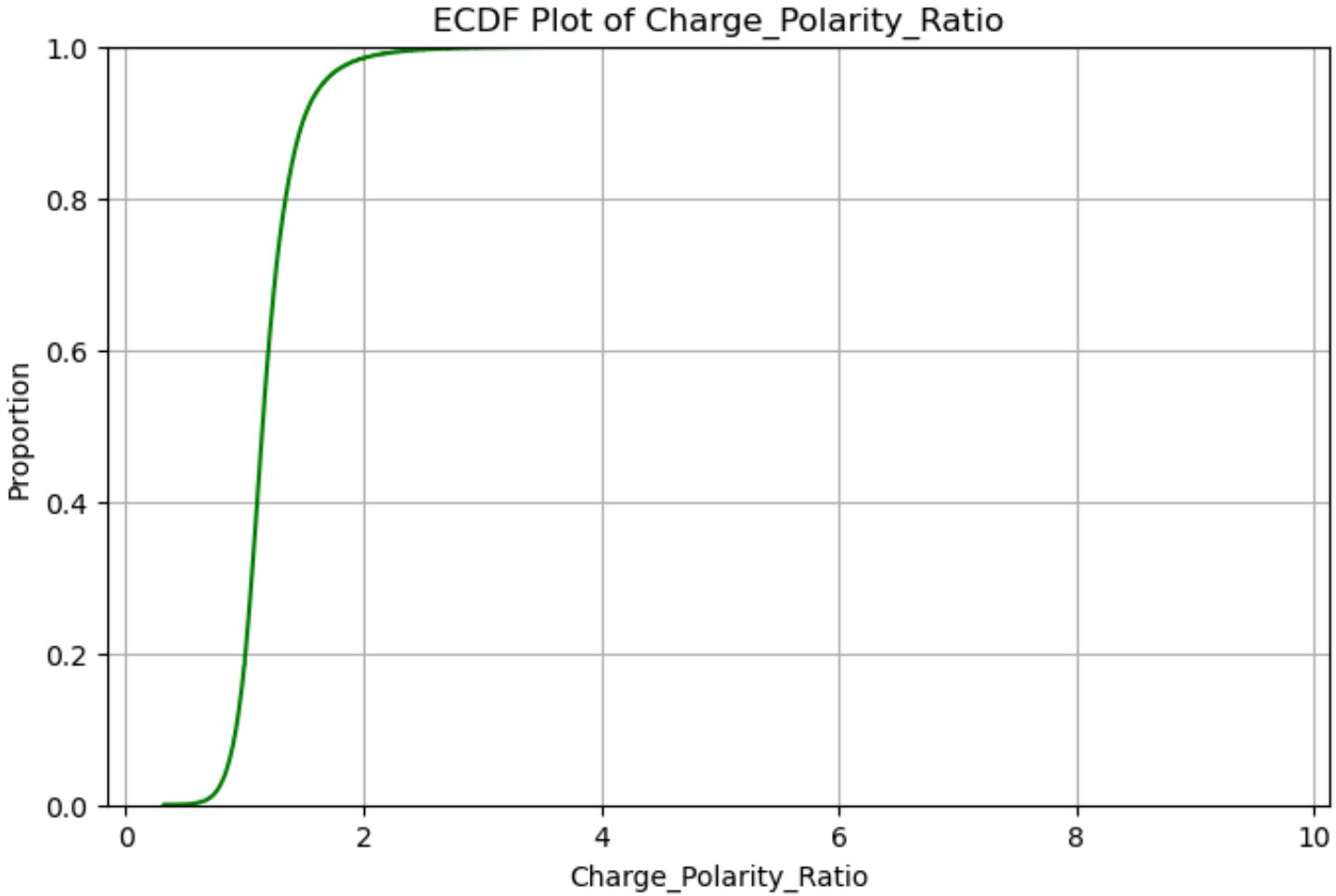


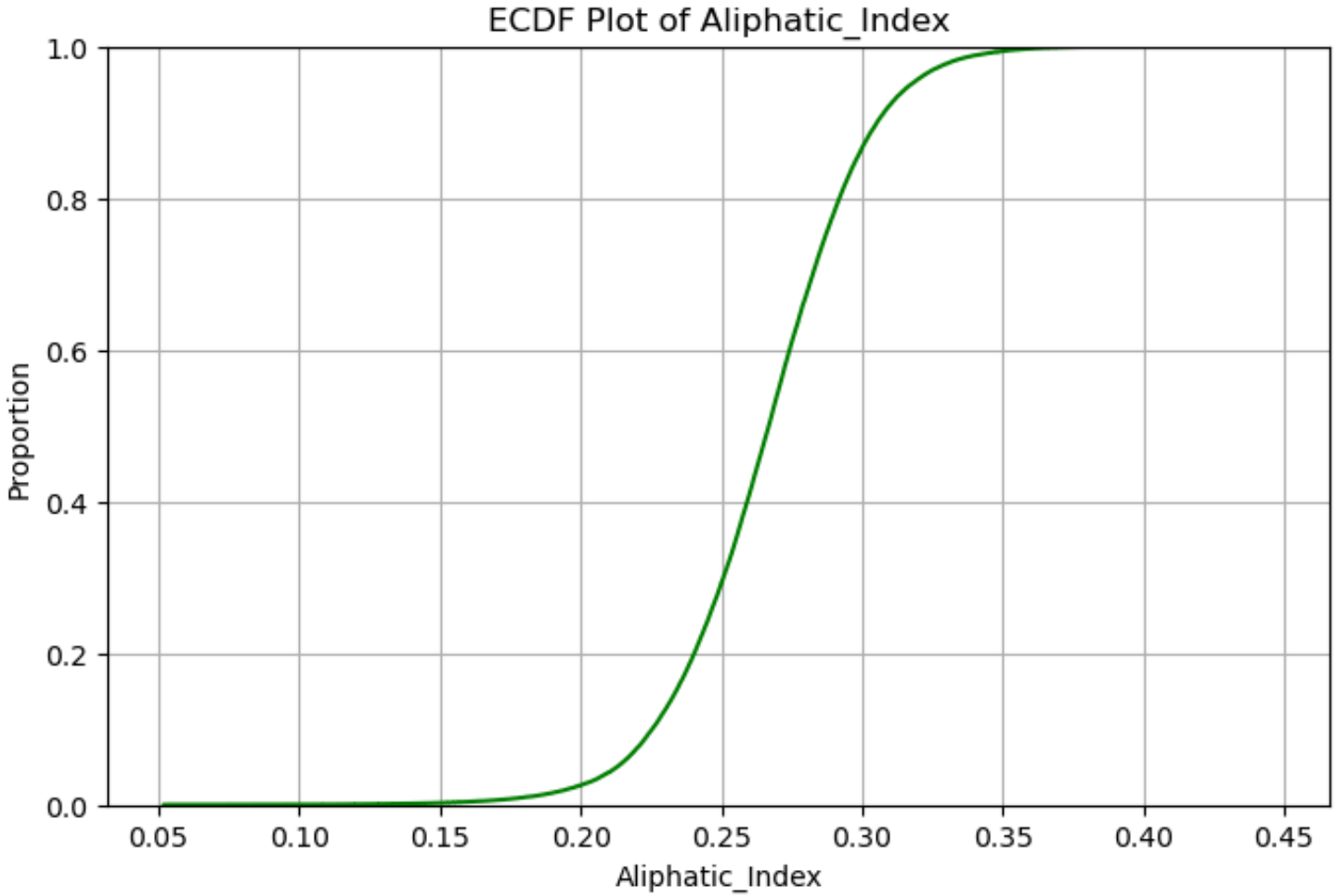
In [56]:
```
Num_Col = protein_df.select_dtypes(include=['int64', 'float64']).columns
for col in Num_Col:
    plt.figure(figsize=(8, 5))
    sns.ecdfplot(protein_df[col], color='Green')
    plt.title(f'ECDF Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Proportion')
    plt.grid(True)
    plt.show()
```

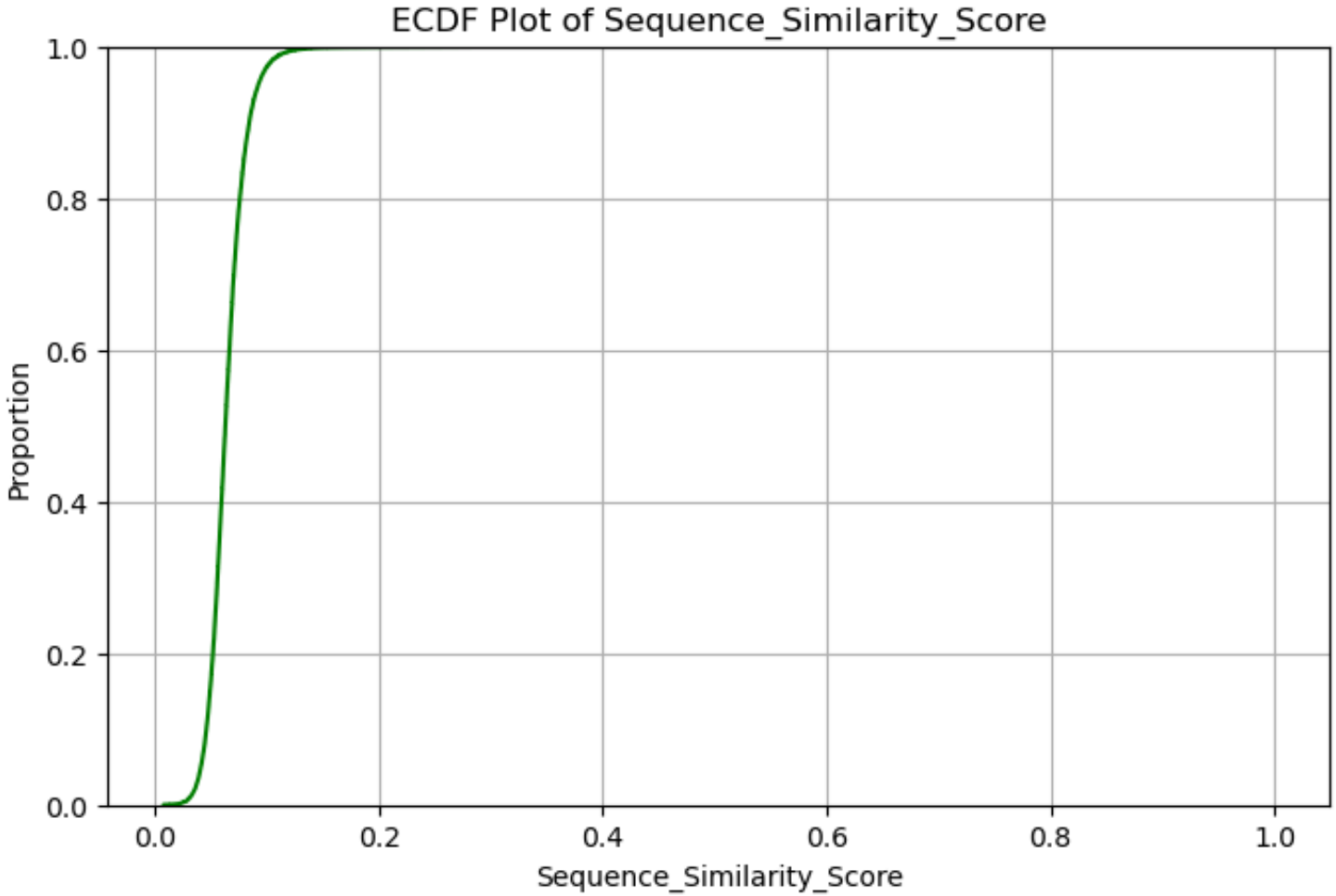

ECDF Plot of proteins
Proportion
proteins
0.0
0.2
0.4
0.6
0.8
1.0

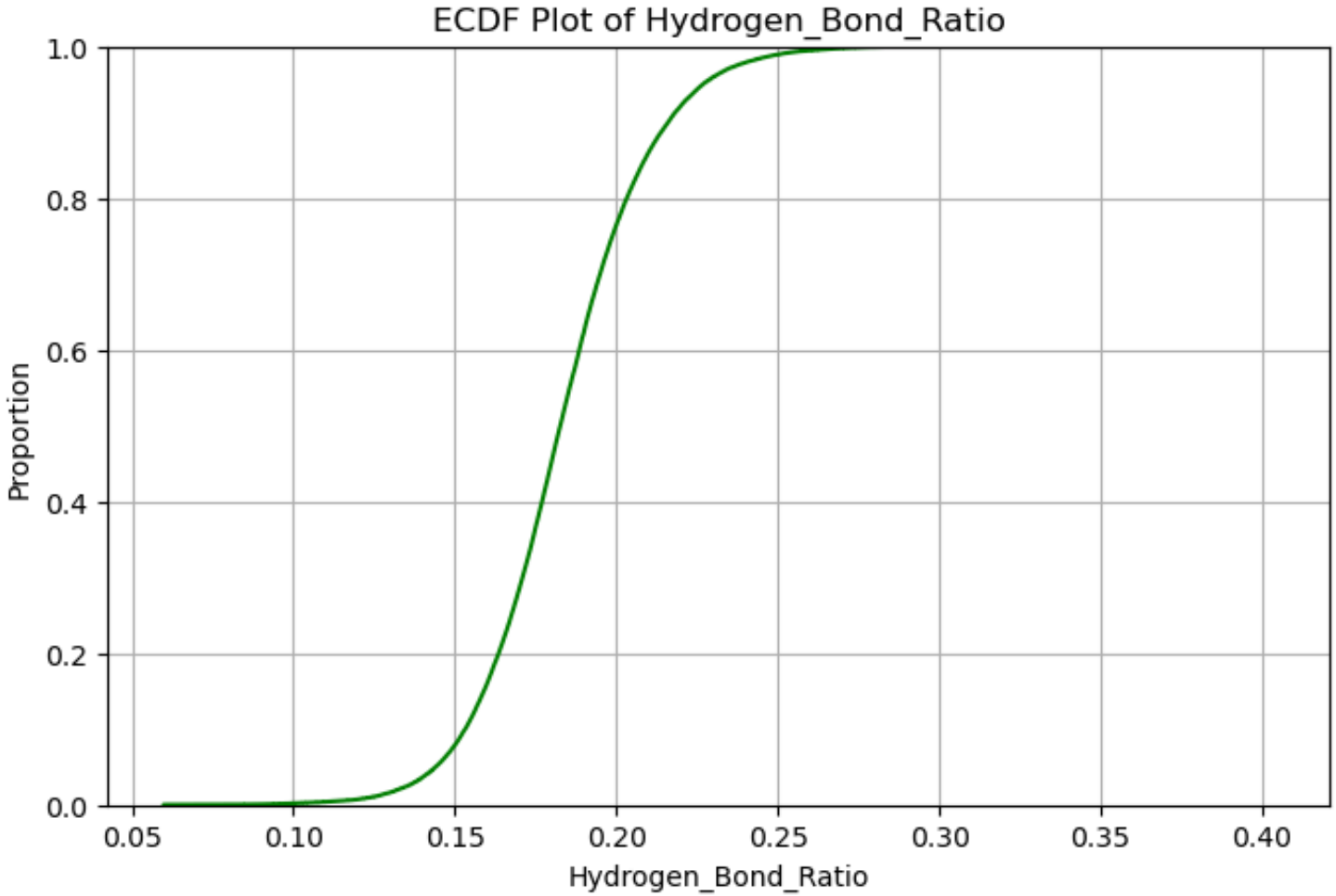

ECDF Plot of Sequence_Length_Difference
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000
Sequence_Length_Difference

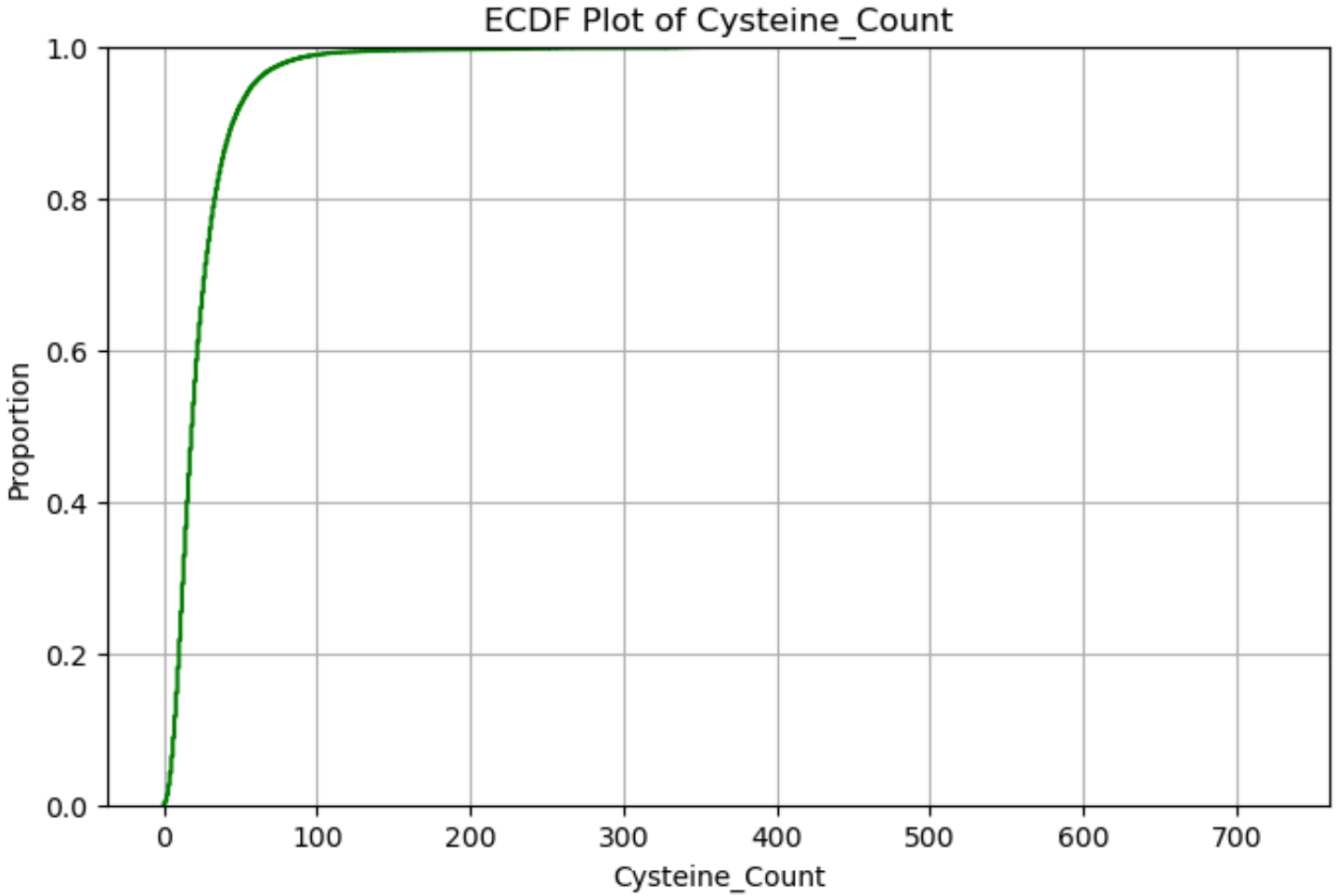

ECDF Plot of Average_Hydrophobicity
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Average_Hydrophobicity

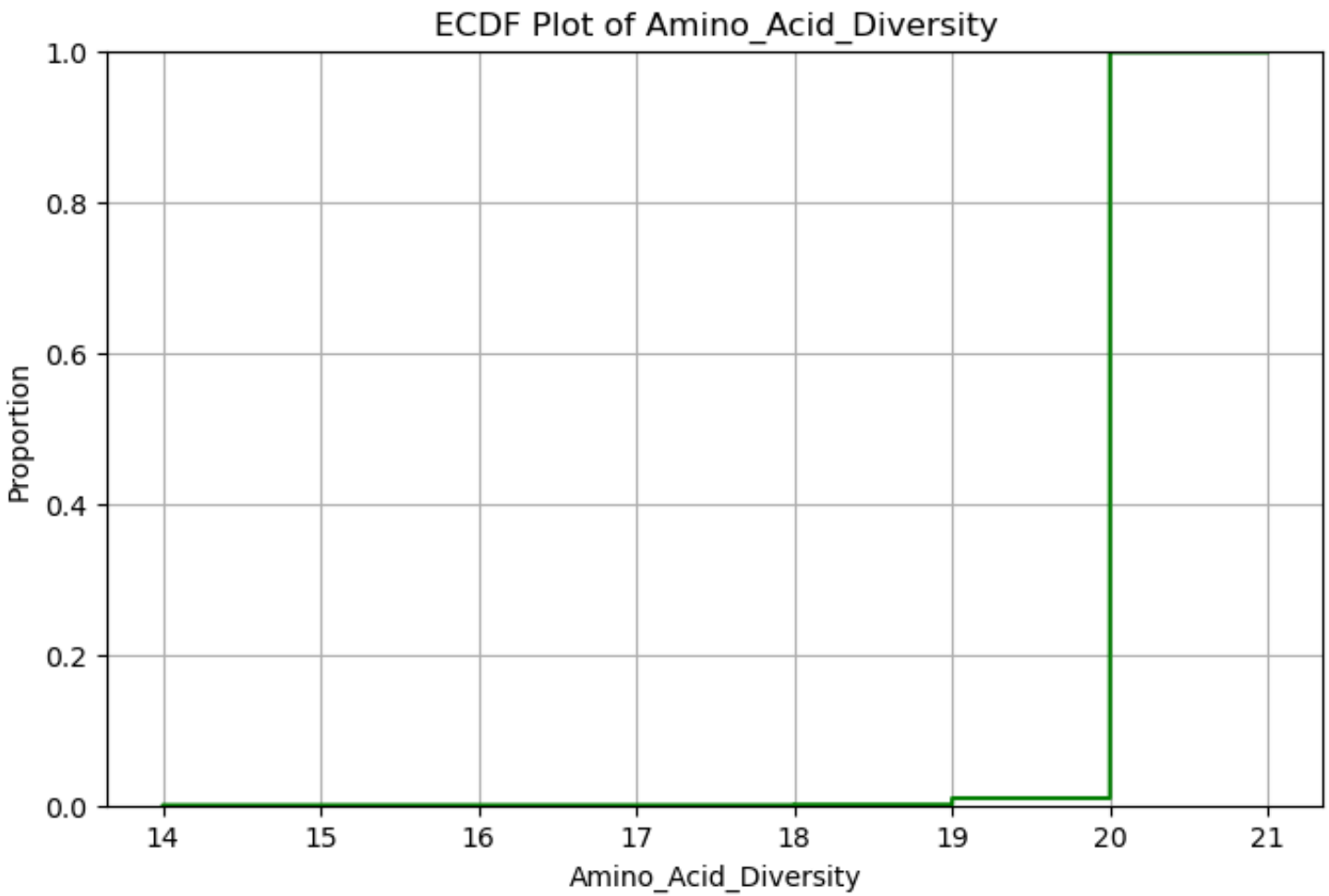

ECDF Plot of Charge_Polarity_Ratio
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Charge_Polarity_Ratio

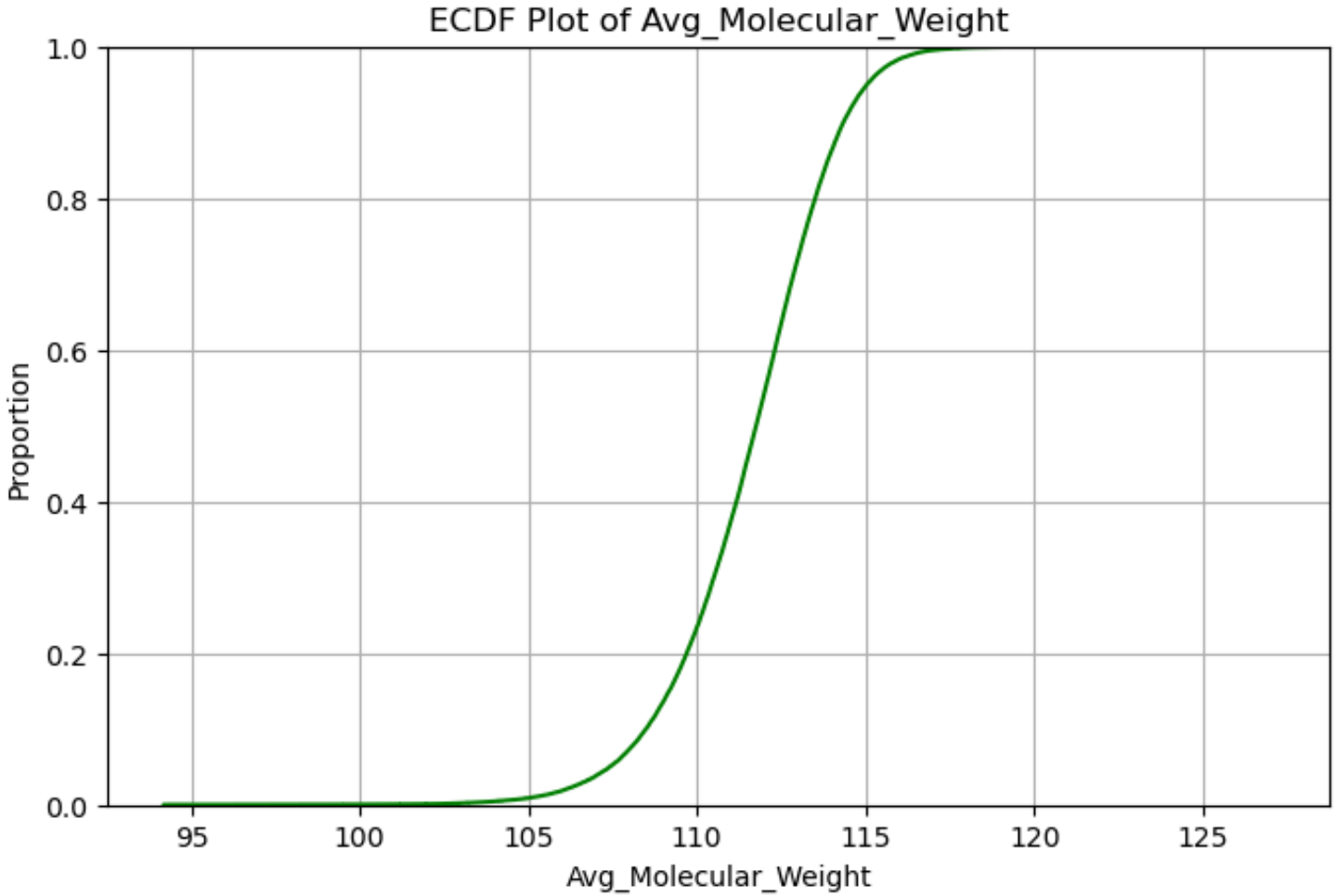

ECDF Plot of Aliphatic_Index
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
Aliphatic_Index

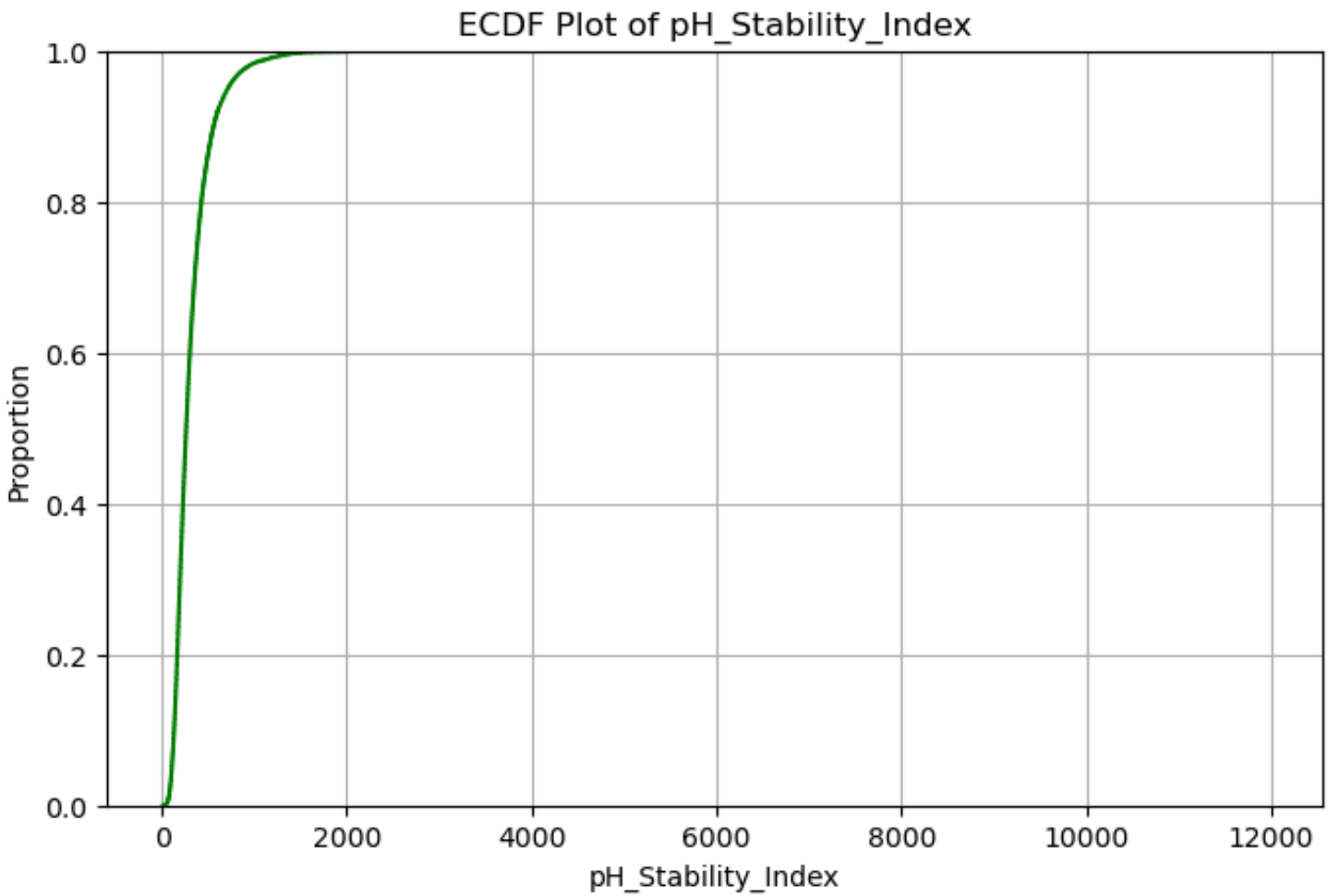

ECDF Plot of Sequence_Similarity_Score
Proportion
Sequence_Similarity_Score
0.0
0.2
0.4
0.6
0.8
1.0

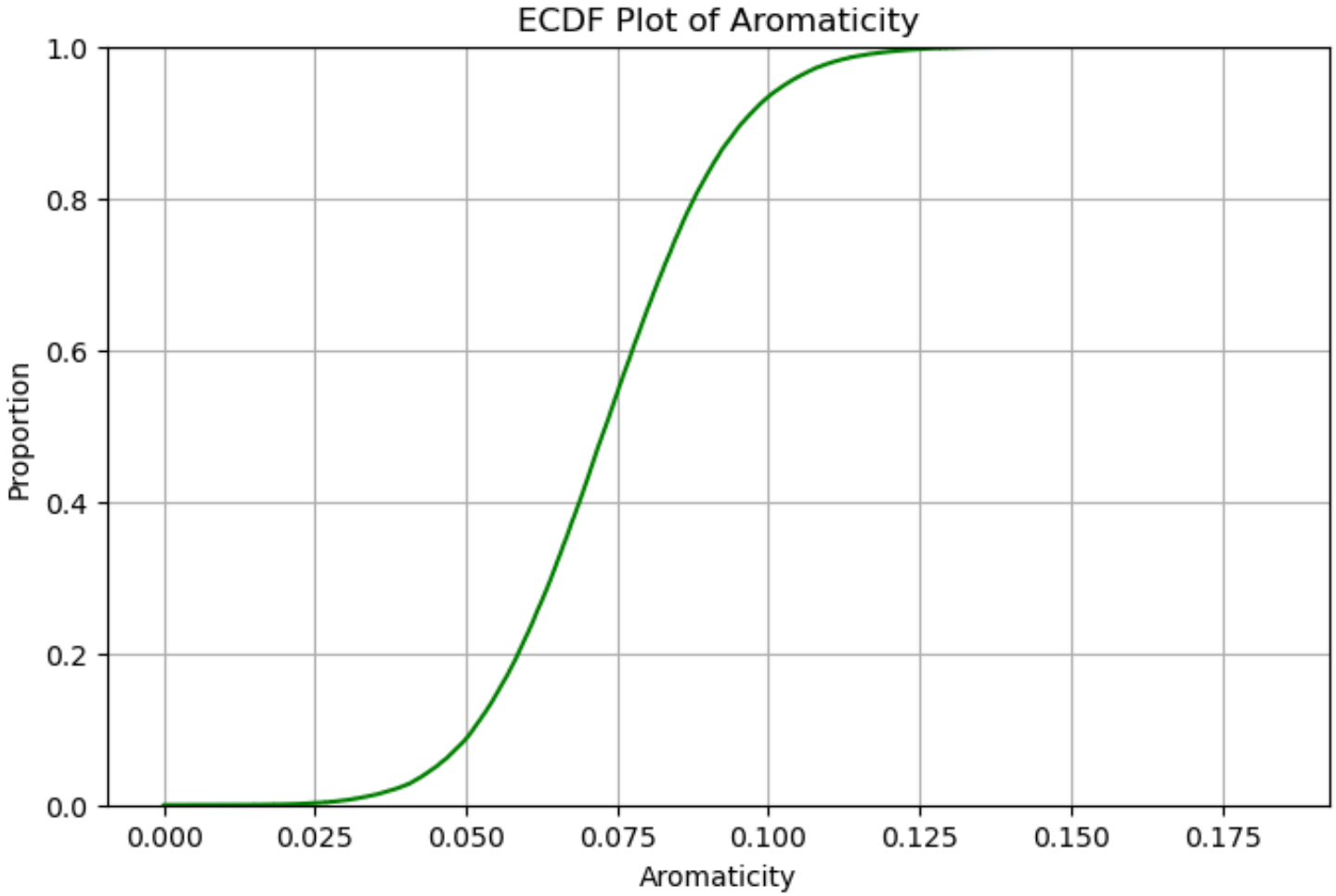

ECDF Plot of Hydrogen_Bond_Ratio
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Hydrogen_Bond_Ratio

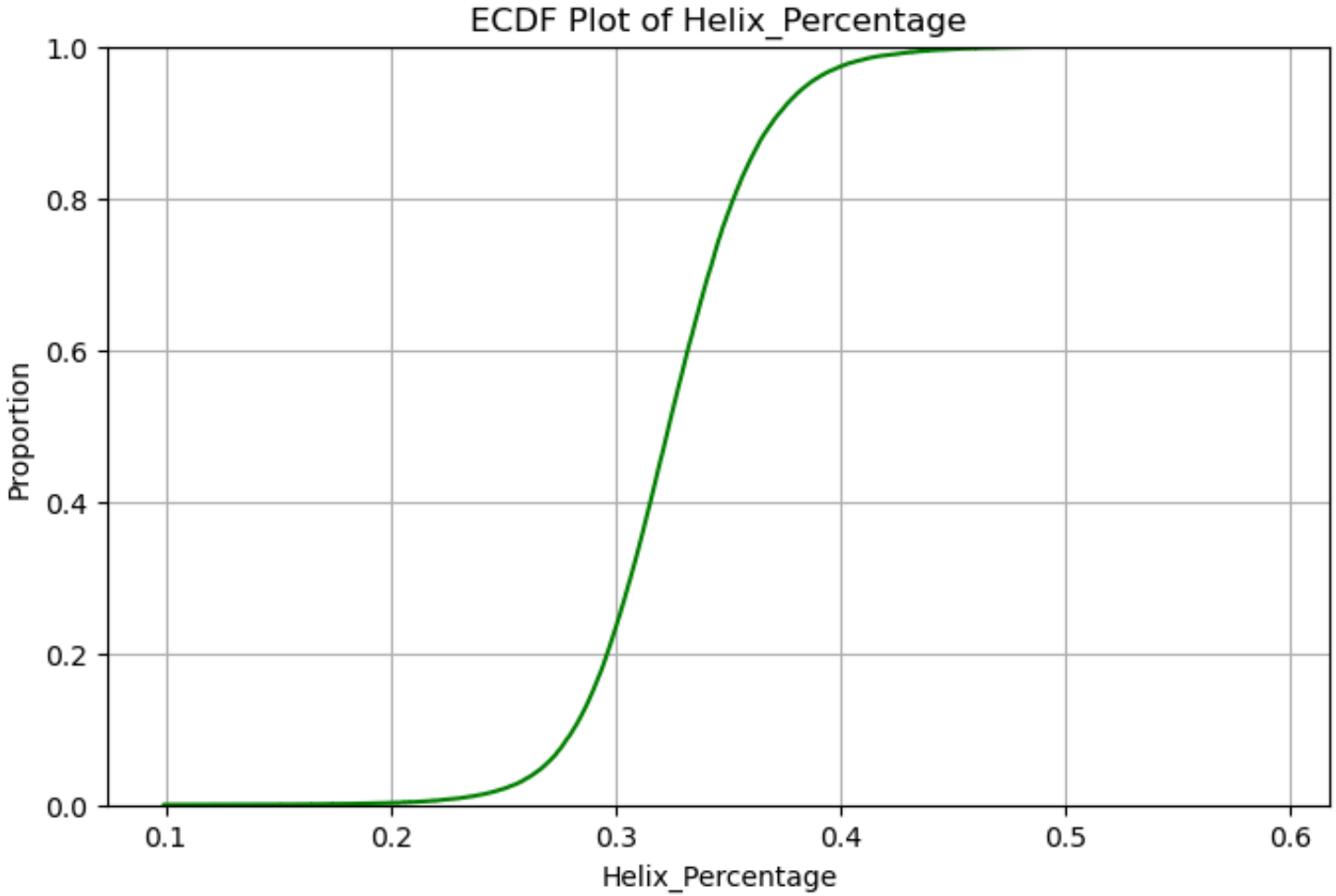

ECDF Plot of Cysteine_Count
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0
100
200
300
400
500
600
700
Cysteine_Count

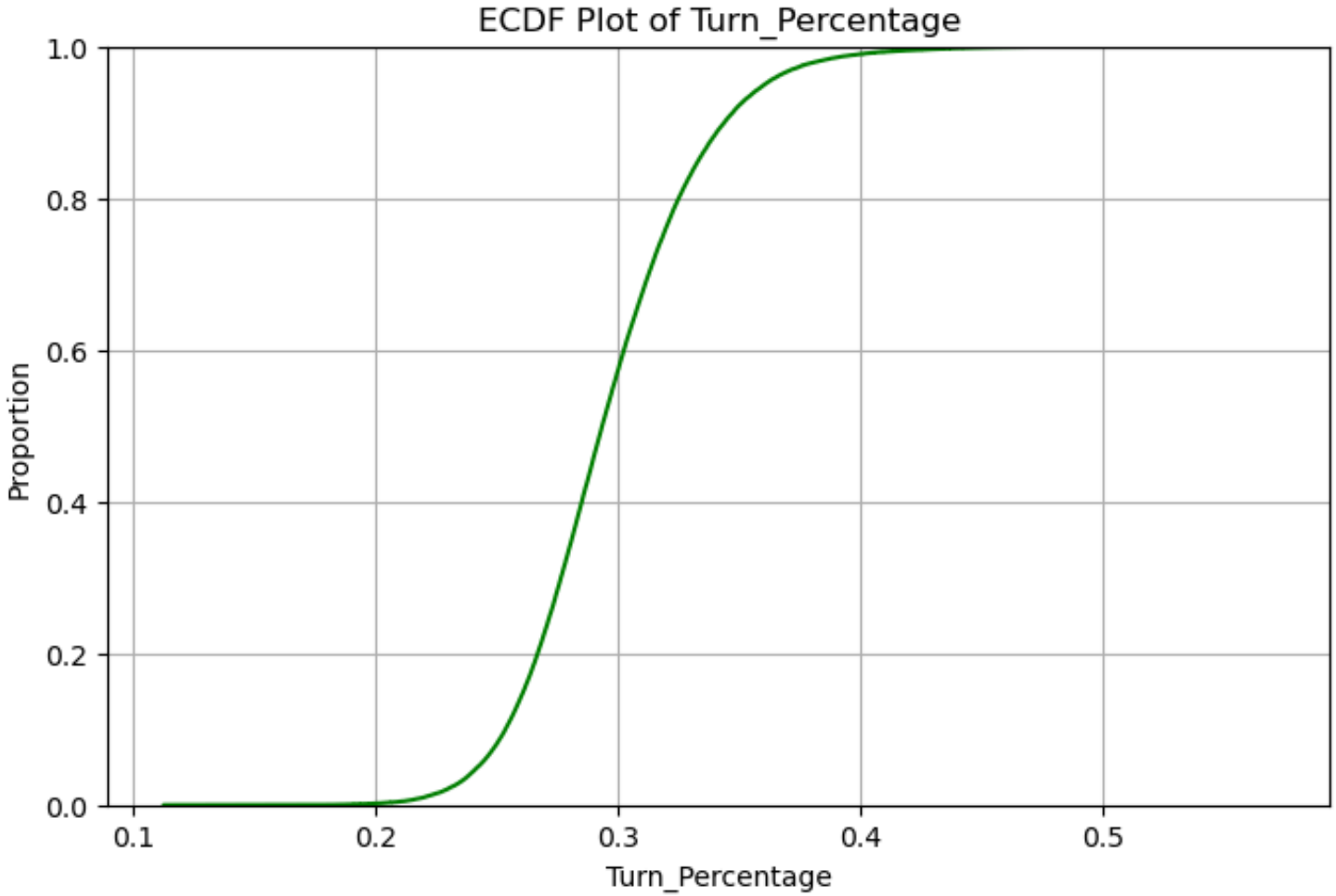

ECDF Plot of Amino_Acid_Diversity
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
14
15
16
17
18
19
20
21
Amino_Acid_Diversity

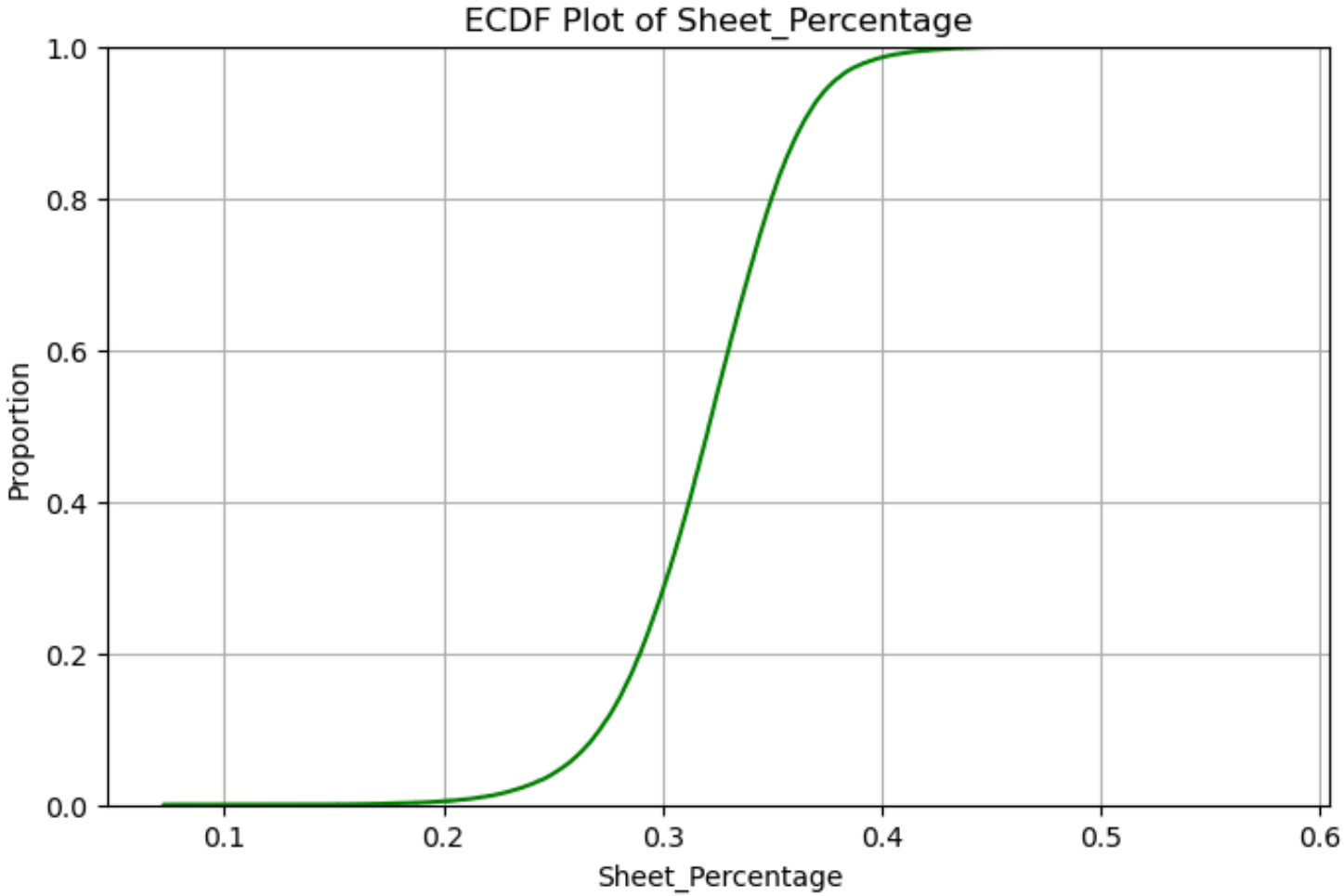

ECDF Plot of Avg_Molecular_Weight
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
95
100
105
110
115
120
125
Avg_Molecular_Weight

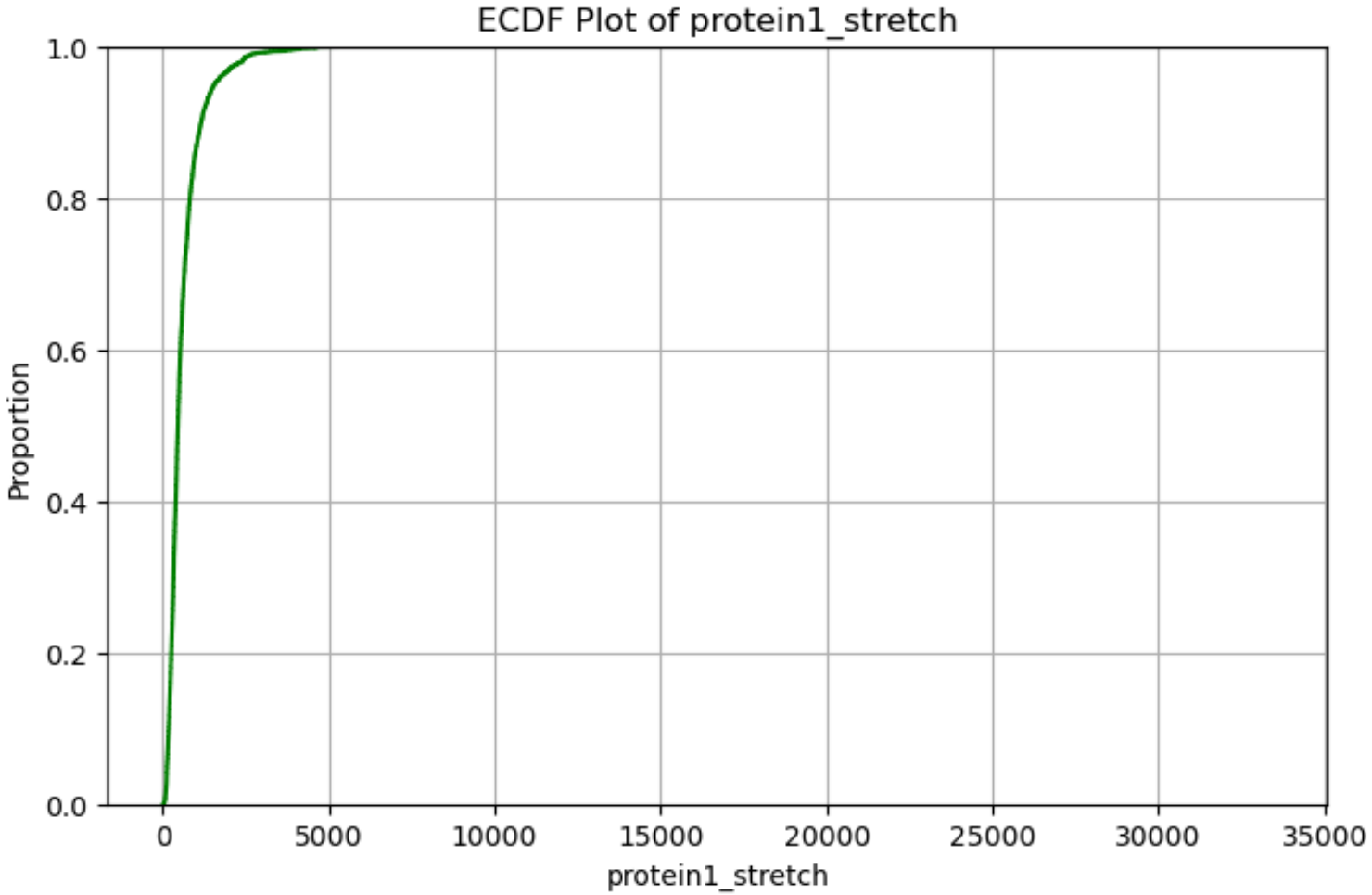

ECDF Plot of pH_Stability_Index
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
12000
pH_Stability_Index

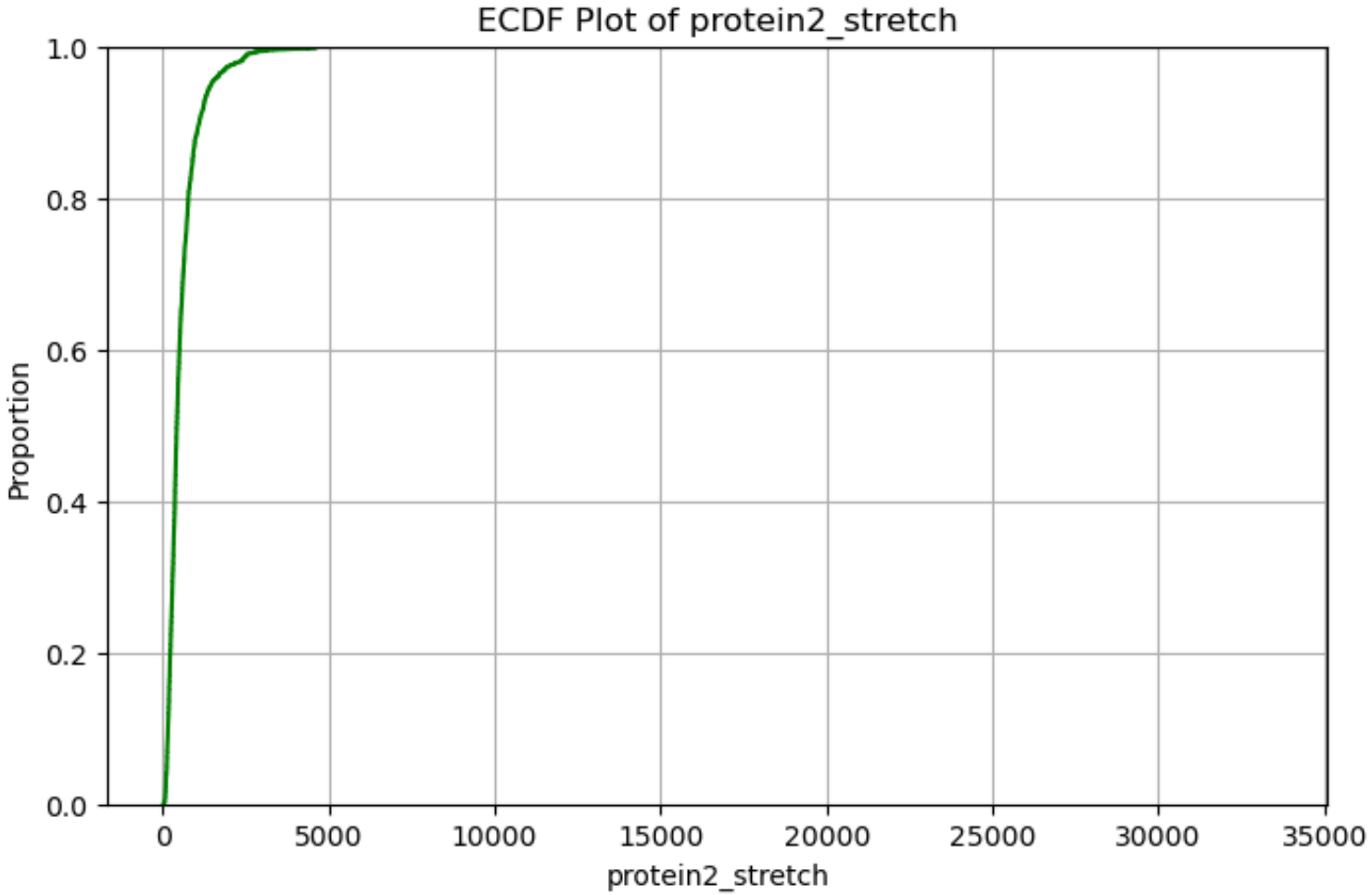

ECDF Plot of Aromaticity
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Aromaticity


ECDF Plot of Helix_Percentage
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Helix_Percentage

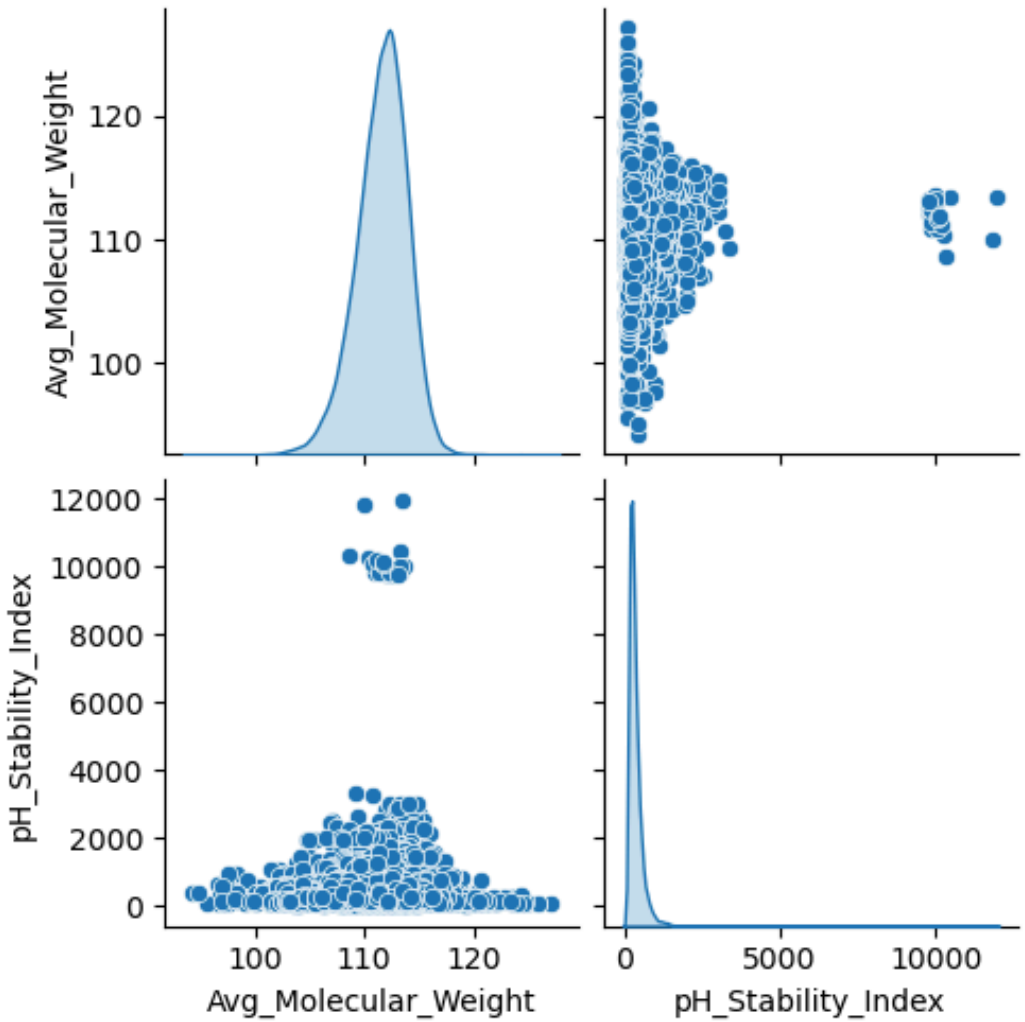

ECDF Plot of Turn_Percentage
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.1
0.2
0.3
0.4
0.5
Turn_Percentage


ECDF Plot of Sheet_Percentage
Proportion
Sheet_Percentage
0.0
0.2
0.4
0.6
0.8
1.0
0.1
0.2
0.3
0.4
0.5
0.6

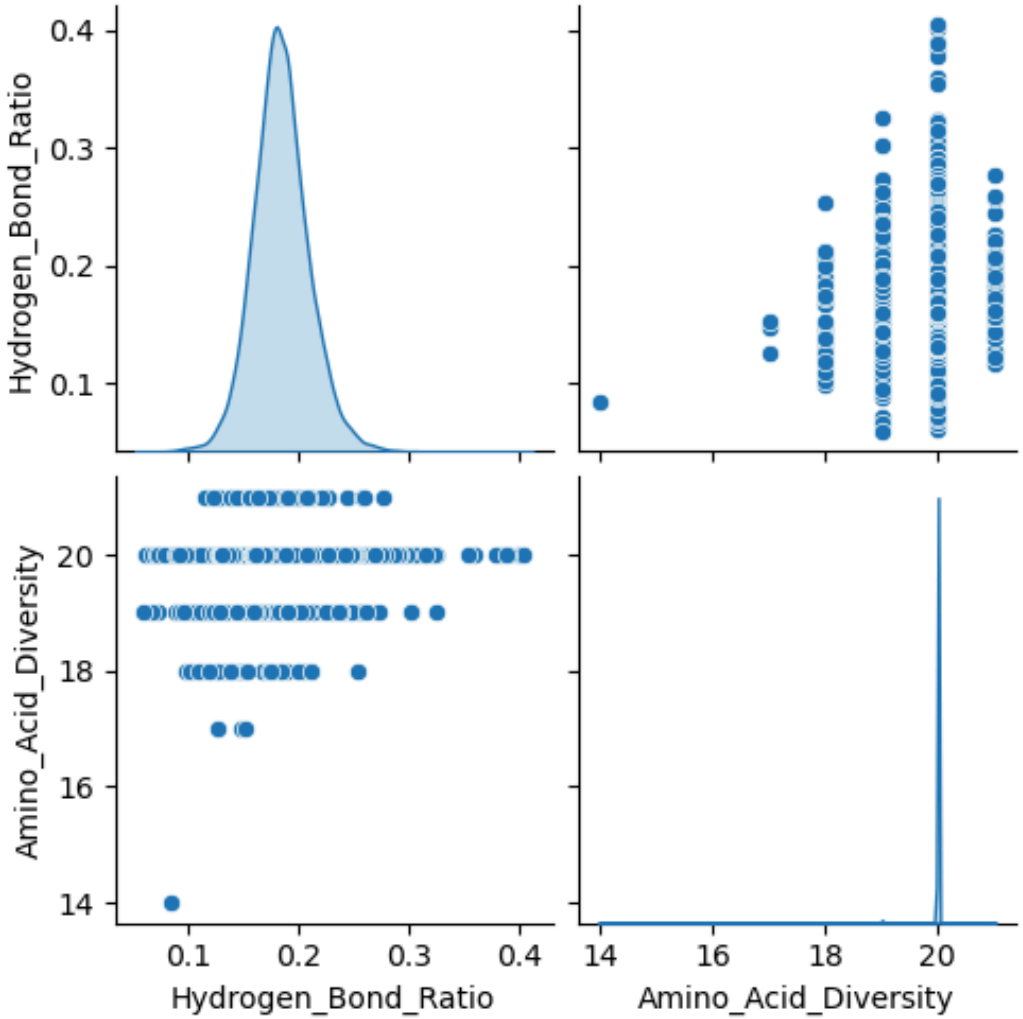

ECDF Plot of protein1_stretch
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000
35000
protein1_stretch



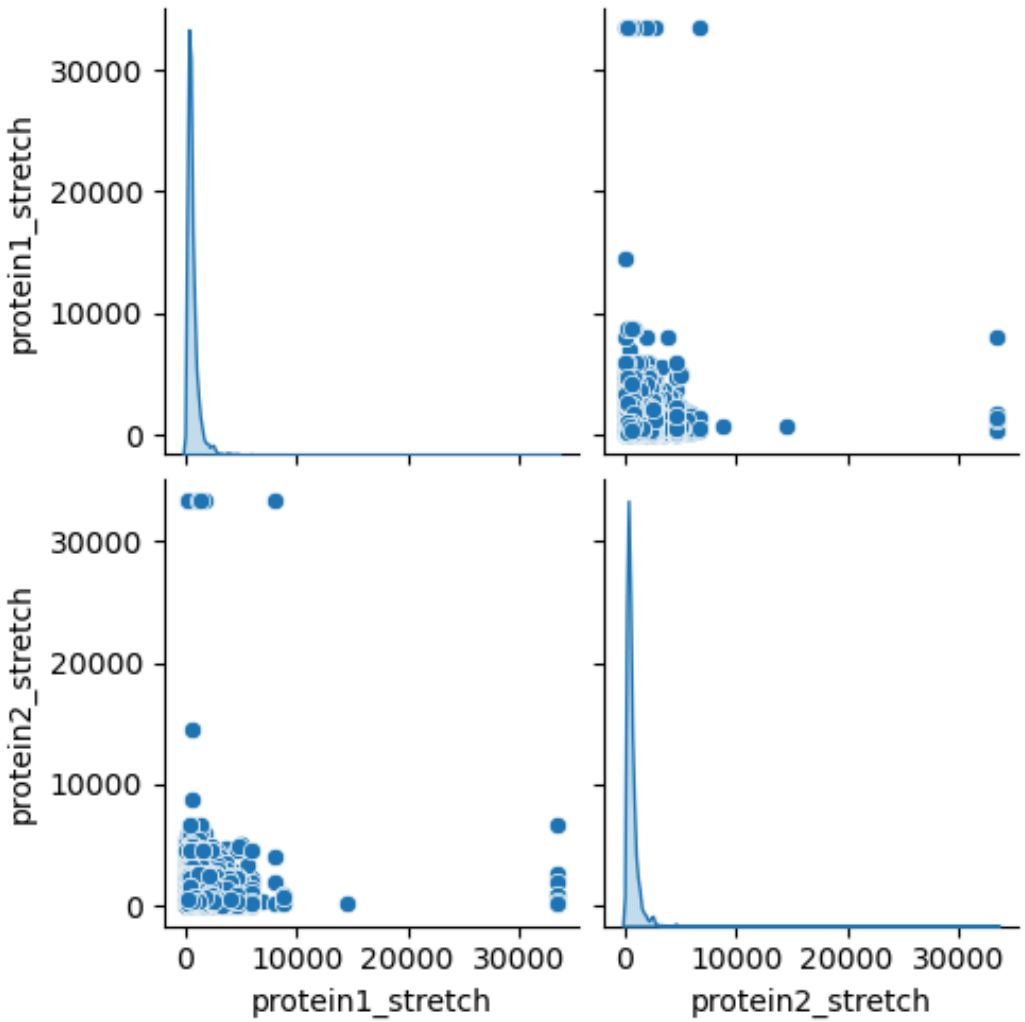
In [57]:
```
sns.pairplot(protein_df[["Avg_Molecular_Weight", "pH_Stability_Index"]],
             diag_kind="kde", palette="husl")
plt.show()
```



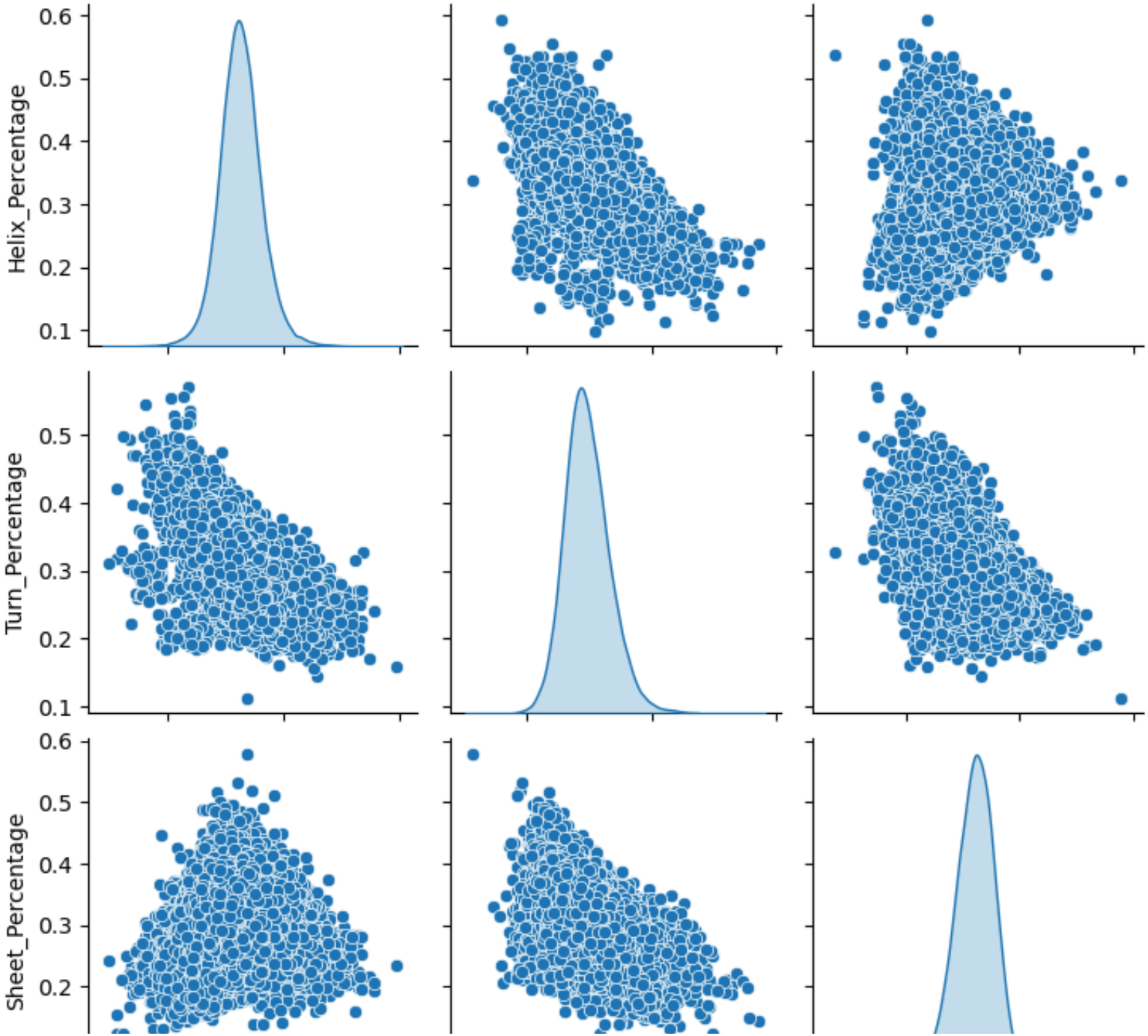
In [58]:
```
sns.pairplot(protein_df[["Hydrogen_Bond_Ratio", "Amino_Acid_Diversity"]],
             diag_kind="kde", palette="Set2")
plt.show()
```

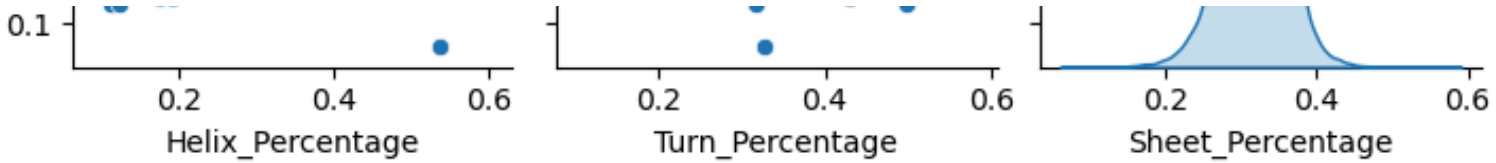


In [59]:
```
sns.pairplot(protein_df[["protein1_stretch", "protein2_stretch"]],
             diag_kind="kde", palette="rocket")
plt.show()
```



In [60]:
```
custom_palette = {0: "#E63946", 1: "#E63946"}  # Blue for 0, Orange for 1
sns.pairplot(protein_df[["Helix_Percentage", "Turn_Percentage", "Sheet_Percentage"]],
             diag_kind="kde",palette="husl")
plt.show()
```


Helix_Percentage
Turn_Percentage
Sheet_Percentage
0.6
0.5
0.4
0.3
0.2
0.1



MODEL BUILDING

In [62]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [63]:
```
cols = protein_df.select_dtypes(include=['number']).columns
corr_matrix = protein_df[cols].corr()
corr_matrix
```

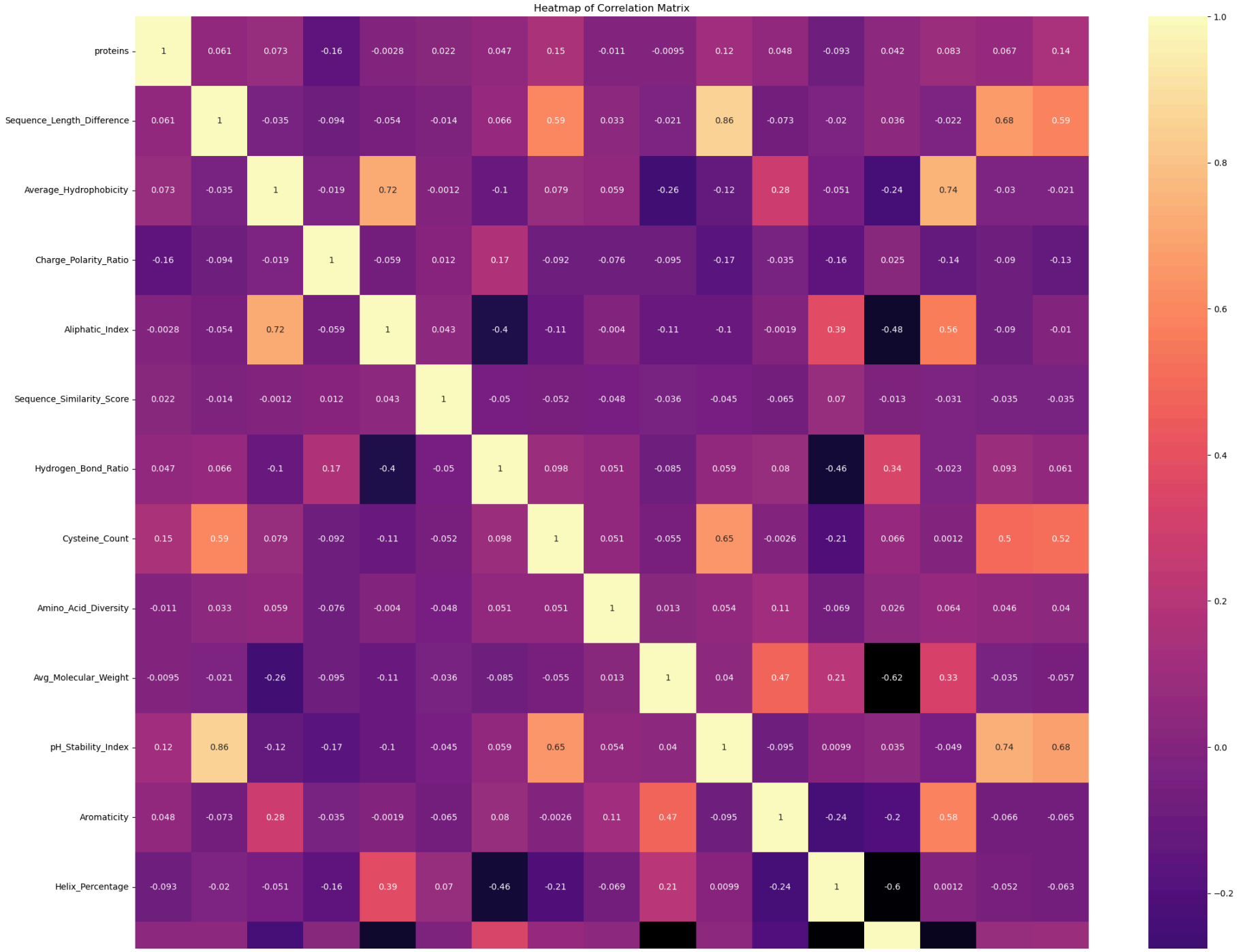
Out[63]:

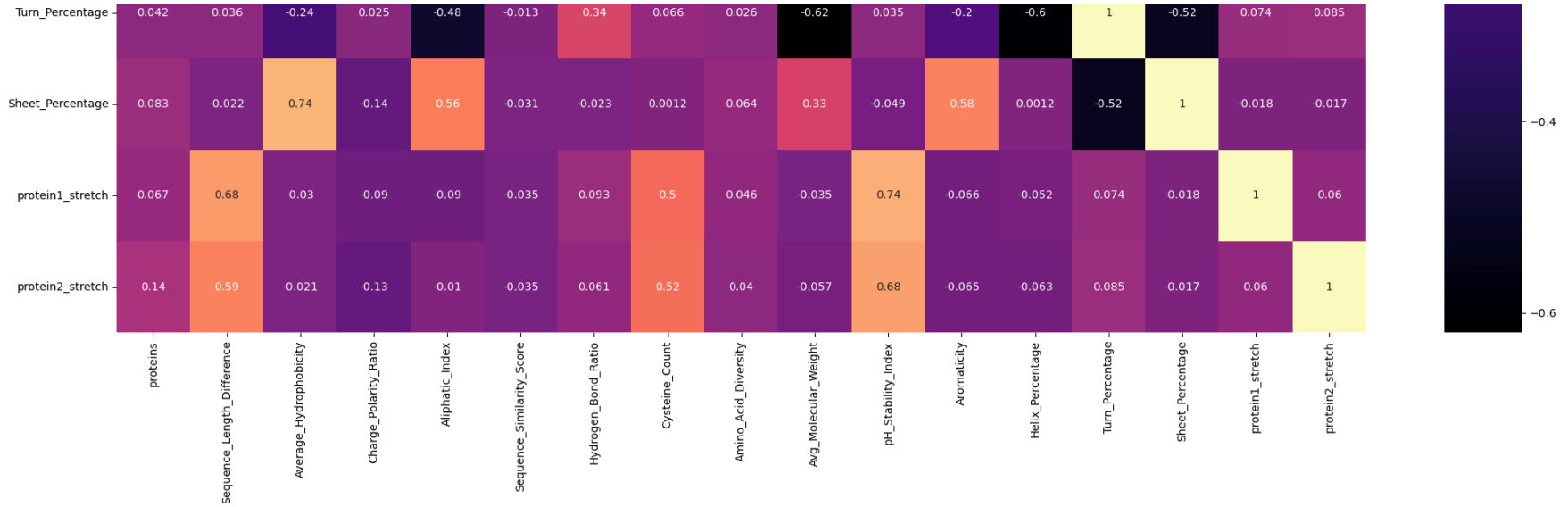
	proteins	Sequence_Length_Difference	Average_Hydrophobicity	Charge_Polarity_Ratio	Aliphatic_Inde
proteins	1.000000	0.061016	0.072984	-0.156650	-0.00279
Sequence_Length_Difference	0.061016	1.000000	-0.034510	-0.093735	-0.05396
Average_Hydrophobicity	0.072984	-0.034510	1.000000	-0.019454	0.71832
Charge_Polarity_Ratio	-0.156650	-0.093735	-0.019454	1.000000	-0.05917
Aliphatic_Index	-0.002795	-0.053960	0.718324	-0.059176	1.00000
Sequence_Similarity_Score	0.021979	-0.014060	-0.001213	0.012260	0.04278
Hydrogen_Bond_Ratio	0.046796	0.066431	-0.102017	0.170913	-0.39676
Cysteine_Count	0.147338	0.588675	0.078561	-0.091678	-0.10855
Amino_Acid_Diversity	-0.011072	0.032512	0.058655	-0.076359	-0.00402
Avg_Molecular_Weight	-0.009466	-0.021370	-0.258000	-0.094727	-0.11173
pH_Stability_Index	0.118005	0.859859	-0.121202	-0.171048	-0.09958
Aromaticity	0.048261	-0.073359	0.284683	-0.034636	-0.00194
Helix_Percentage	-0.093312	-0.019879	-0.050522	-0.157778	0.38505
Turn_Percentage	0.042034	0.035657	-0.235723	0.024700	-0.48078
Sheet_Percentage	0.083036	-0.022047	0.742536	-0.137558	0.55898
protein1_stretch	0.067054	0.675026	-0.030416	-0.090448	-0.08981
protein2_stretch	0.141524	0.586511	-0.020753	-0.134280	-0.01020

In [64]:
```
corr_matrix.shape
```

Out[64]:
```
(17, 17)
```

In [72]:
```
# Drop non-numeric columns
protein_df_clean = protein_df.select_dtypes(include=[float, int])

correlation_matrix = protein_df_clean.corr()

plt.figure(figsize=(25,25))
sns.heatmap(correlation_matrix, annot=True, cmap='magma')
plt.title('Heatmap of Correlation Matrix')
plt.show()
```


Heatmap of Correlation Matrix
proteins 1 0.061 0.073 -0.16 -0.0028 0.022 0.047 0.15 -0.011 -0.0095 0.12 0.048 -0.093 0.042 0.083 0.067 0.14
Sequence_Length_Difference 0.061 1 -0.035 -0.094 -0.054 -0.014 0.066 0.59 0.033 -0.021 0.86 -0.073 -0.02 0.036 -0.022 0.68 0.59
Average_Hydrophobicity 0.073 -0.035 1 -0.019 0.72 -0.0012 -0.1 0.079 0.059 -0.26 -0.12 0.28 -0.051 -0.24 0.74 -0.03 -0.021
Charge_Polarity_Ratio -0.16 -0.094 -0.019 1 -0.059 0.012 0.17 -0.092 -0.076 -0.095 -0.17 -0.035 -0.16 0.025 -0.14 -0.09 -0.13
Aliphatic_Index -0.0028 -0.054 0.72 -0.059 1 0.043 -0.4 -0.11 -0.004 -0.11 -0.1 -0.0019 0.39 -0.48 0.56 -0.09 -0.01
Sequence_Similarity_Score 0.022 -0.014 -0.0012 0.012 0.043 1 -0.05 -0.052 -0.048 -0.036 -0.045 -0.065 0.07 -0.013 -0.031 -0.035 -0.035
Hydrogen_Bond_Ratio 0.047 0.066 -0.1 0.17 -0.4 -0.05 1 0.098 0.051 -0.085 0.059 0.08 -0.46 0.34 -0.023 0.093 0.061
Cysteine_Count 0.15 0.59 0.079 -0.092 -0.11 -0.052 0.098 1 0.051 -0.055 0.65 -0.0026 -0.21 0.066 0.0012 0.5 0.52
Amino_Acid_Diversity -0.011 0.033 0.059 -0.076 -0.004 -0.048 0.051 0.051 1 0.013 0.054 0.11 -0.069 0.026 0.064 0.046 0.04
Avg_Molecular_Weight -0.0095 -0.021 -0.26 -0.095 -0.11 -0.036 -0.085 -0.055 0.013 1 0.04 0.47 0.21 -0.62 0.33 -0.035 -0.057
pH_Stability_Index 0.12 0.86 -0.12 -0.17 -0.1 -0.045 0.059 0.65 0.054 0.04 1 -0.095 0.0099 0.035 -0.049 0.74 0.68
Aromaticity 0.048 -0.073 0.28 -0.035 -0.0019 -0.065 0.08 -0.0026 0.11 0.47 -0.095 1 -0.24 -0.2 0.58 -0.066 -0.065
Helix_Percentage -0.093 -0.02 -0.051 -0.16 0.39 0.07 -0.46 -0.21 -0.069 0.21 0.0099 -0.24 1 -0.6 0.0012 -0.052 -0.063
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



In [74]:
```
X = protein_df[cols].values
y = protein_df['proteins'].values
```

In [76]:
```
print(y.shape)
print(X.shape)
```

```
(73110,)
(73110, 17)
```

In [78]:
```
from sklearn.model_selection import train_test_split, cross_val_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [80]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier


# Assuming X and y are your features and labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Define models
lr = LogisticRegression()
DT = DecisionTreeClassifier()
rnf = RandomForestClassifier()

# Perform cross-validation
lr_scores = cross_val_score(lr, X_train, y_train, cv=5)
DT_scores = cross_val_score(DT, X_train, y_train, cv=5)
rnf_scores = cross_val_score(rnf, X_train, y_train, cv=5)

# Print results
print("Logistic Regression CV Accuracy:", lr_scores.mean())
print("Decision Tree CV Accuracy:", DT_scores.mean())
print("Random Forest CV Accuracy:", rnf_scores.mean())
```

```
Logistic Regression CV Accuracy: 1.0
Decision Tree CV Accuracy: 1.0
Random Forest CV Accuracy: 1.0
```

In [81]:
```
corr_matrix = pd.DataFrame(X_train).corr()
print(corr_matrix)
```

	0	1	2	3	4	5	6	\
0	1.000000	0.062277	0.071931	-0.156315	-0.005646	0.020335	0.047489	
1	0.062277	1.000000	-0.034414	-0.095446	-0.052974	-0.011366	0.067092	
2	0.071931	-0.034414	1.000000	-0.017652	0.719688	-0.004359	-0.101559	
3	-0.156315	-0.095446	-0.017652	1.000000	-0.058167	0.014464	0.174352	
4	-0.005646	-0.052974	0.719688	-0.058167	1.000000	0.040424	-0.396010	
5	0.020335	-0.011366	-0.004359	0.014464	0.040424	1.000000	-0.046363	
6	0.047489	0.067092	-0.101559	0.174352	-0.396010	-0.046363	1.000000	
7	0.148300	0.590440	0.079393	-0.092610	-0.107190	-0.051740	0.101349	
8	-0.011119	0.033432	0.054224	-0.075045	-0.005324	-0.044014	0.051062	
9	-0.007625	-0.020877	-0.257975	-0.094433	-0.112479	-0.042836	-0.083913	
10	0.119447	0.858837	-0.120625	-0.173513	-0.099617	-0.046066	0.060477	
11	0.049994	-0.073679	0.285552	-0.033402	0.000459	-0.071649	0.080209	
12	-0.093084	-0.021968	-0.048847	-0.162490	0.383682	0.066683	-0.456959	
13	0.040759	0.035189	-0.236967	0.024525	-0.480712	-0.007835	0.334509	
14	0.084207	-0.022070	0.743256	-0.135954	0.559681	-0.036603	-0.022312	
15	0.068848	0.671871	-0.027836	-0.092553	-0.087334	-0.034034	0.094429	
16	0.141396	0.586918	-0.020920	-0.134863	-0.011168	-0.036526	0.062743	

	7	8	9	10	11	12	13	\
0	0.148300	-0.011119	-0.007625	0.119447	0.049994	-0.093084	0.040759	
1	0.590440	0.033432	-0.020877	0.858837	-0.073679	-0.021968	0.035189	
2	0.079393	0.054224	-0.257975	-0.120625	0.285552	-0.048847	-0.236967	
3	-0.092610	-0.075045	-0.094433	-0.173513	-0.033402	-0.162490	0.024525	
4	-0.107190	-0.005324	-0.112479	-0.099617	0.000459	0.383682	-0.480712	
5	-0.051740	-0.044014	-0.042836	-0.046066	-0.071649	0.066683	-0.007835	
6	0.101349	0.051062	-0.083913	0.060477	0.080209	-0.456959	0.334509	
7	1.000000	0.051698	-0.055196	0.650684	-0.002787	-0.208449	0.066512	
8	0.051698	1.000000	0.009830	0.054844	0.103030	-0.067099	0.028745	
9	-0.055196	0.009830	1.000000	0.039686	0.467806	0.209205	-0.620749	
10	0.650684	0.054844	0.039686	1.000000	-0.095528	0.007466	0.036242	
11	-0.002787	0.103030	0.467806	-0.095528	1.000000	-0.231719	-0.199663	
12	-0.208449	-0.067099	0.209205	0.007466	-0.231719	1.000000	-0.600324	
13	0.066512	0.028745	-0.620749	0.036242	-0.199663	-0.600324	1.000000	
14	0.001884	0.059016	0.324582	-0.049039	0.581215	0.002858	-0.517110	
15	0.506950	0.047197	-0.034846	0.734684	-0.065493	-0.053366	0.073521	
16	0.518399	0.040794	-0.057775	0.686331	-0.066070	-0.064583	0.086537	

```
          14        15        16
0   0.084207  0.068848  0.141396
1  -0.022070  0.671871  0.586918
2   0.743256 -0.027836 -0.020920
3  -0.135954 -0.092553 -0.134863
4   0.559681 -0.087334 -0.011168
5  -0.036603 -0.034034 -0.036526
6  -0.022312  0.094429  0.062743
7   0.001884  0.506950  0.518399
8   0.059016  0.047197  0.040794
9   0.324582 -0.034846 -0.057775
10 -0.049039  0.734684  0.686331
11  0.581215 -0.065493 -0.066070
12  0.002858 -0.053366 -0.064583
13 -0.517110  0.073521  0.086537
14  1.000000 -0.015953 -0.018279
15 -0.015953  1.000000  0.059349
16 -0.018279  0.059349  1.000000
```

In [84]:
```
print(X_train)
print(X_test)
```

```
[[ 0.00000000e+00  4.40000000e+01 -2.78383776e-01 ...  3.33836052e-01
   3.61000000e+02  4.05000000e+02]
 [ 1.00000000e+00  7.90000000e+01 -3.89098104e-01 ...  3.33924306e-01
   6.76000000e+02  5.97000000e+02]
 [ 0.00000000e+00  4.06000000e+02 -9.41478697e-01 ...  2.92875045e-01
   7.98000000e+02  3.92000000e+02]
 ...
 [ 0.00000000e+00  4.62000000e+02 -4.17795591e-01 ...  3.23785556e-01
   1.23500000e+03  7.73000000e+02]
 [ 1.00000000e+00  8.90000000e+01 -9.00530088e-01 ...  2.67023993e-01
   3.53000000e+02  2.64000000e+02]
 [ 1.00000000e+00  2.27000000e+02 -4.64428737e-01 ...  3.29159901e-01
   1.27300000e+03  1.04600000e+03]]
[[ 1.00000000e+00  6.53000000e+02 -2.88519121e-01 ...  3.32332000e-01
   1.64000000e+02  8.17000000e+02]
 [ 1.00000000e+00  1.55600000e+03 -5.82686967e-01 ...  3.26853775e-01
   2.47200000e+03  9.16000000e+02]
 [ 0.00000000e+00  4.44000000e+02 -2.90261417e-01 ...  3.31370029e-01
   5.22000000e+02  9.66000000e+02]
 ...
 [ 0.00000000e+00  3.75000000e+02 -7.26355143e-01 ...  2.93788834e-01
   1.66000000e+02  5.41000000e+02]
 [ 1.00000000e+00  6.80000000e+01 -4.95696775e-01 ...  3.08812917e-01
   4.82000000e+02  4.14000000e+02]
 [ 0.00000000e+00  8.00000000e+00 -4.41946198e-01 ...  3.24646584e-01
   3.73000000e+02  3.81000000e+02]]
```

MODEL EVALUATION

In [87]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

LogisticRegression

In [90]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[90]:
```
LogisticRegression()
```

In [92]:
```
y_pred_lr = lr.predict(X_test)
y_pred_lr
```

Out[92]:
```
array([1, 1, 0, ..., 0, 1, 0], dtype=int64)
```

In [94]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7251
           1       1.00      1.00      1.00      7371

    accuracy                           1.00     14622
   macro avg       1.00      1.00      1.00     14622
weighted avg       1.00      1.00      1.00     14622
```

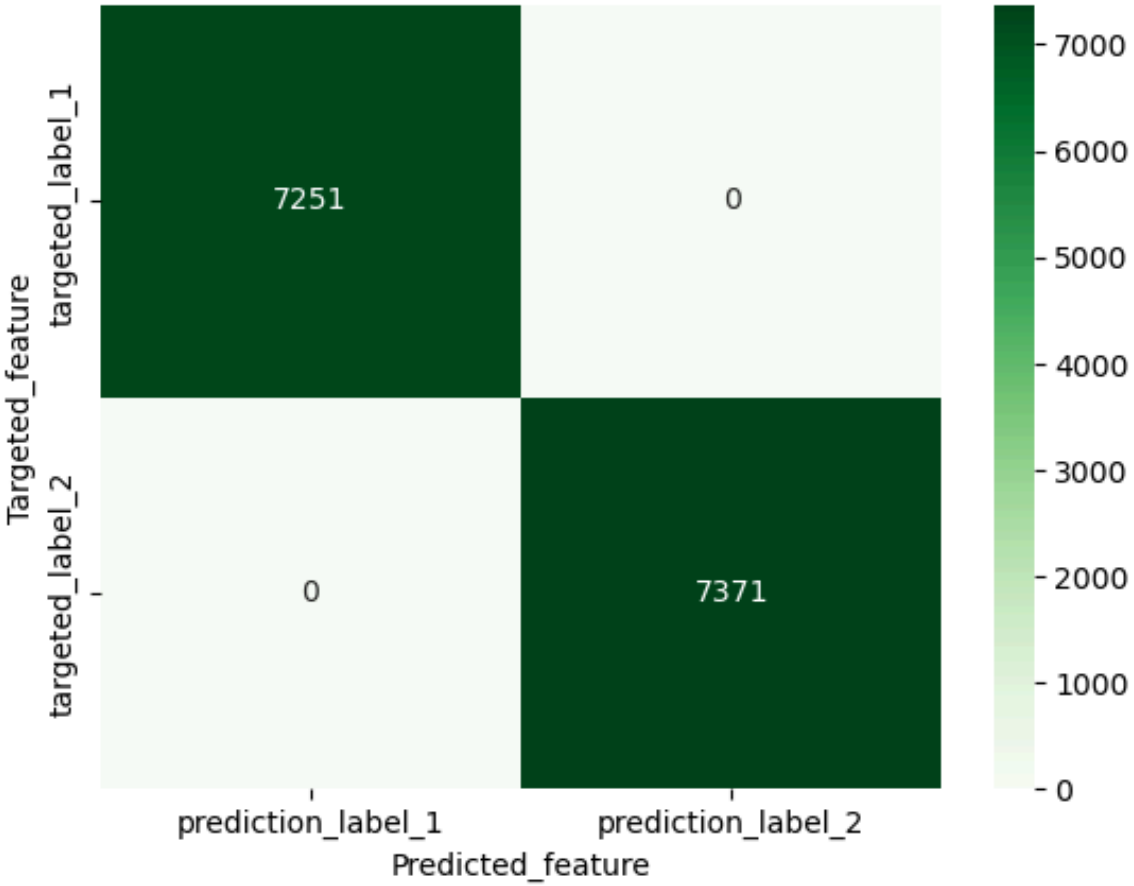
In [96]:
```
accuracy = accuracy_score(y_test, y_pred_lr)
print(f"Logistic Regression Model Accuracy: {accuracy * 100:.2f}%")
```

```
Logistic Regression Model Accuracy: 100.00%
```

In [98]:
```
y_train_pred = lr.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 1.0
```

In [100…
```
cm = confusion_matrix(y_test, y_pred_lr)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="Greens")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



DecisionTreeClassifier

In [103…
```
DT = DecisionTreeClassifier()
DT.fit(X_train,y_train)
```

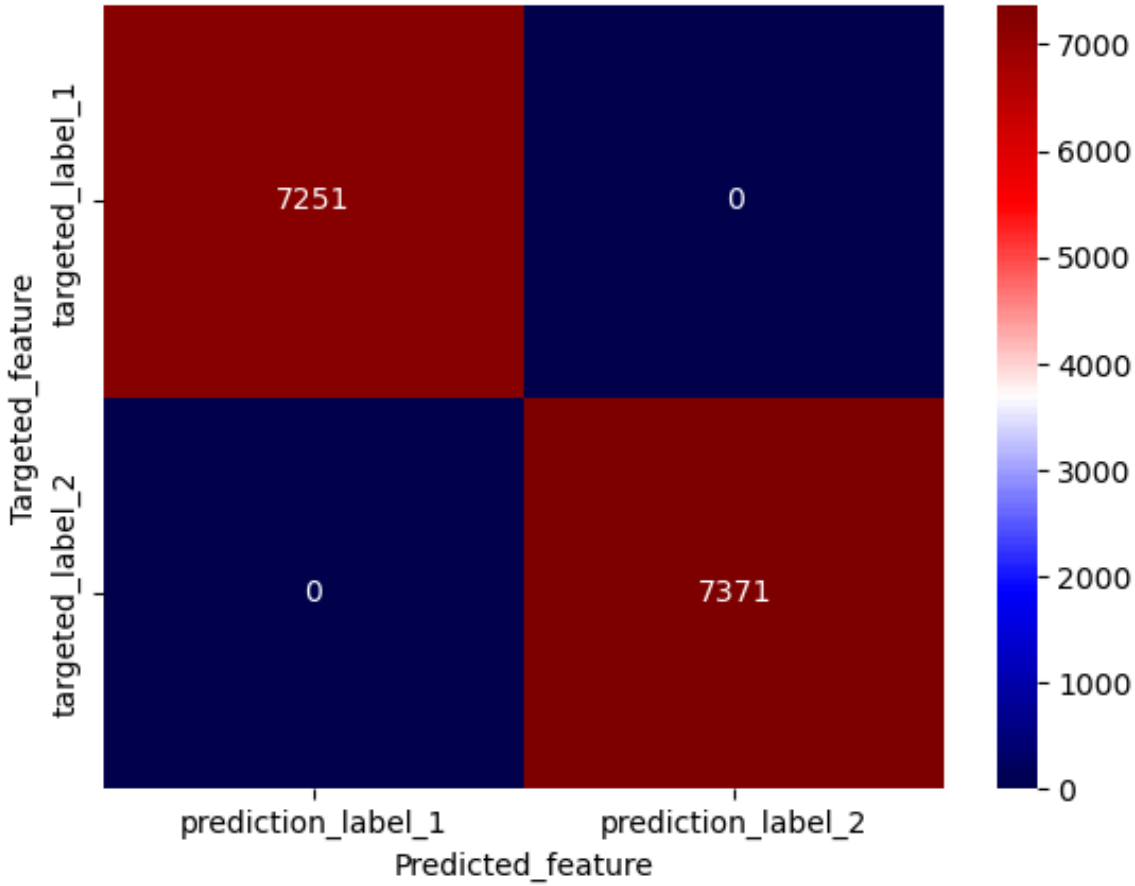
Out[103…
```
DecisionTreeClassifier()
```

In [105…
```
y_pred_DT = DT.predict(X_test)
y_pred_DT
```

Out[105…
```
array([1, 1, 0, ..., 0, 1, 0], dtype=int64)
```

In [107…
```
print(classification_report(y_test,y_pred_DT))



cm = confusion_matrix(y_test, y_pred_DT)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="seismic")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7251
           1       1.00      1.00      1.00      7371

    accuracy                           1.00     14622
   macro avg       1.00      1.00      1.00     14622
weighted avg       1.00      1.00      1.00     14622
```



In [108…
```
accuracy = accuracy_score(y_test, y_pred_DT)
print(f"DecisionTreeClassifier Model Accuracy: {accuracy * 100:.2f}%")
y_train_pred = DT.predict(X_train)
```

```
DecisionTreeClassifier Model Accuracy: 100.00%
```

In [111…
```
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 1.0
```

RandomForestClassifier

In [114…
```
rnf = RandomForestClassifier()
```

In [116…
```
rnf.fit(X_train,y_train)
```

Out[116…
RandomForestClassifier

RandomForestClassifier()

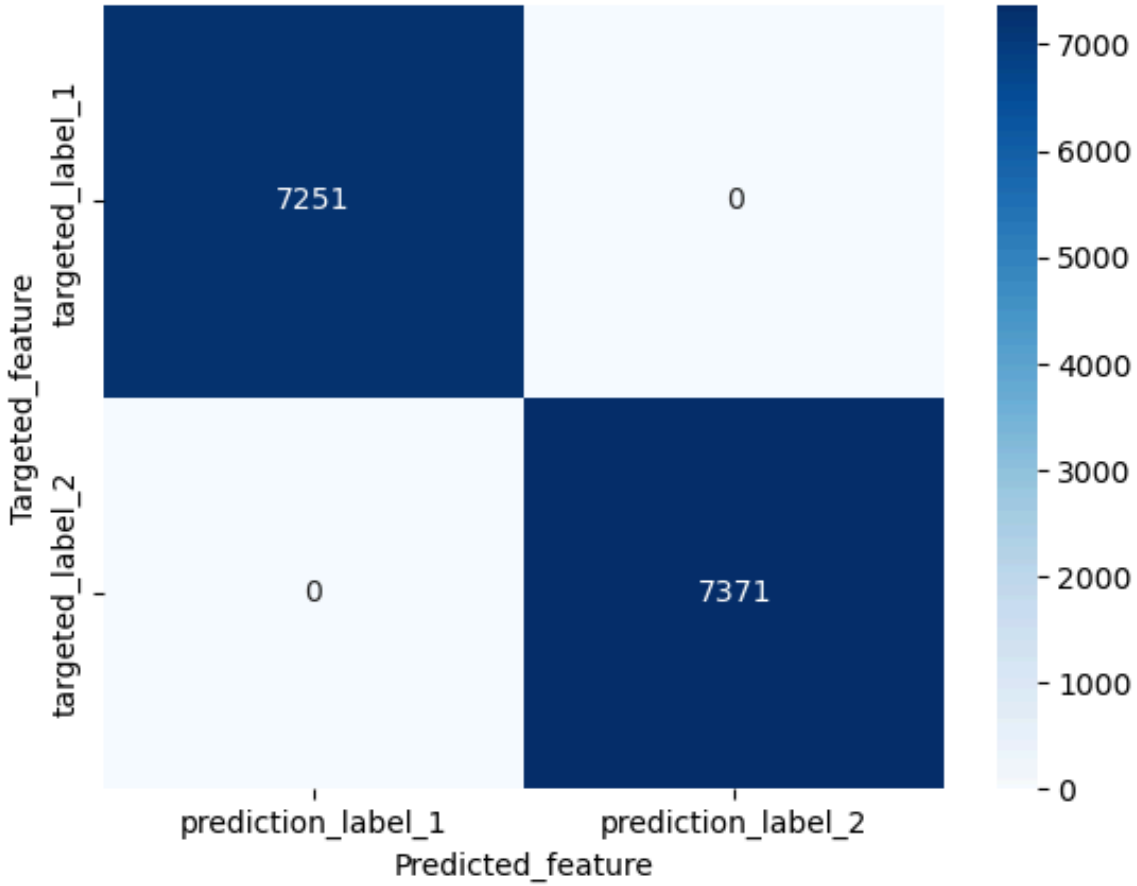
In [117…
```
y_pred_rnf = rnf.predict(X_test)
y_pred_rnf
```

Out[117…
```
array([1, 1, 0, ..., 0, 1, 0], dtype=int64)
```

In [118…
```
print(classification_report(y_test,y_pred_rnf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7251
           1       1.00      1.00      1.00      7371

    accuracy                           1.00     14622
   macro avg       1.00      1.00      1.00     14622
weighted avg       1.00      1.00      1.00     14622
```

In [119…
```
cm = confusion_matrix(y_test, y_pred_rnf)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap= "Blues")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



In [120…
```
y_train_pred = rnf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 1.0
```

In [121…
```
accuracy = accuracy_score(y_test, y_pred_rnf)
print(f"RandomForestClassifier Model Accuracy: {accuracy * 100:.2f}%")
y_train_pred = DT.predict(X_train)
```

```
RandomForestClassifier Model Accuracy: 100.00%
```

XGBClassifier

In [123…
```
from xgboost import XGBClassifier
```

In [124…
```
xg = XGBClassifier()
xg.fit(X_train,y_train)
```

Out[124…
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [125…
```
y_pred_xg = xg.predict(X_test)
y_pred_xg
```

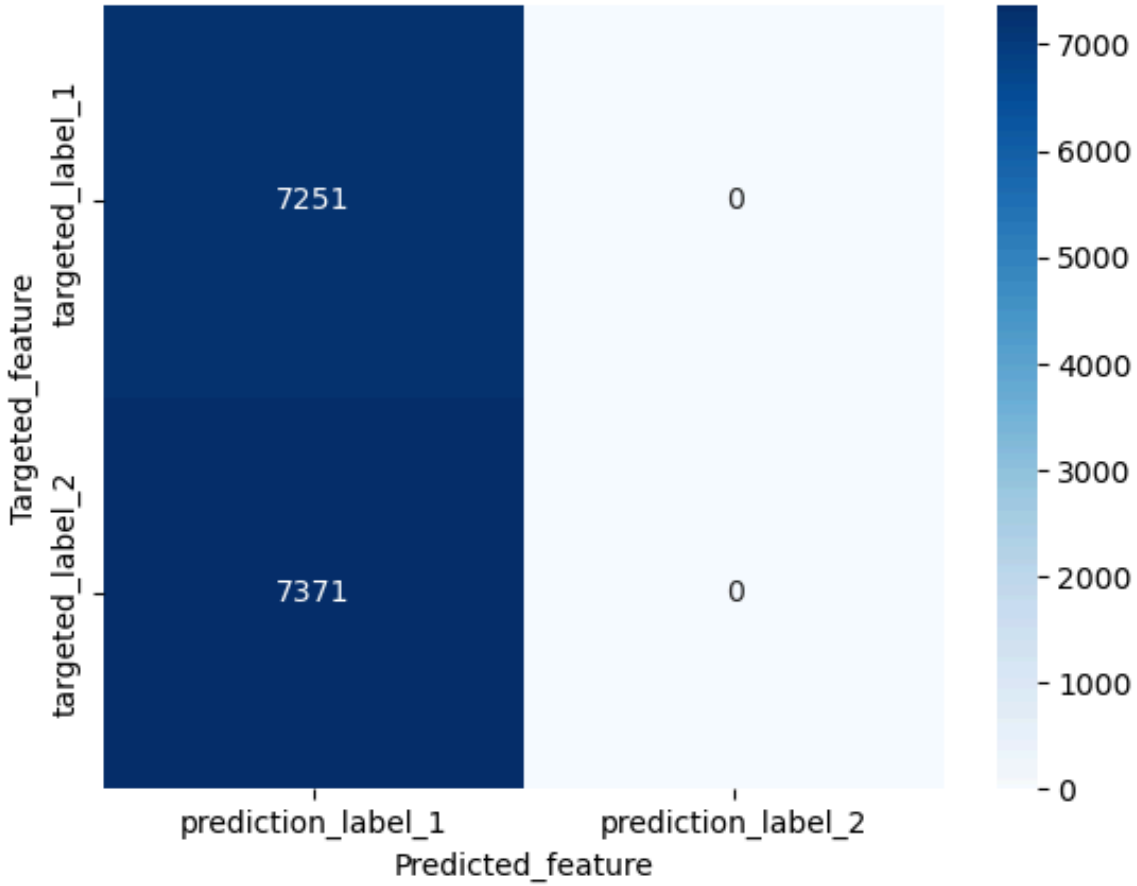
Out[125…
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [126…
```
print(classification_report(y_test,y_pred_xg))
```

	precision	recall	f1-score	support
0	0.50	1.00	0.66	7251
1	0.00	0.00	0.00	7371
accuracy			0.50	14622
macro avg	0.25	0.50	0.33	14622
weighted avg	0.25	0.50	0.33	14622

In [127…

```
cm = confusion_matrix(y_test, y_pred_xg)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap= "Blues")
ax.xaxis.set_ticklabels(['prediction_label_1', 'prediction_label_2'])
ax.yaxis.set_ticklabels(['targeted_label_1', 'targeted_label_2'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("Targeted_feature")
plt.show()
```



In [128…

```
accuracy = accuracy_score(y_test, y_pred_xg)
print(f"XGBClassifier Model Accuracy: {accuracy * 100:.2f}%")
y_train_pred = DT.predict(X_train)
```

```
XGBClassifier Model Accuracy: 49.59%
```

Conclusion

In this study, multiple machine learning models were evaluated for their classification performance. Logistic Regression, Decision Tree, and Random Forest classifiers all achieved 100% accuracy, precision, recall, and F1-score. While these results indicate excellent

model performance, they may also suggest overfitting, which occurs when a model learns the training data too well and fails to generalize to new, unseen data (Kuhn & Johnson, 2020). Overfitting can be identified by evaluating the models on an independent test set or using cross-validation techniques (James et al., 2021).were evaluated for classifying positive and negative protein-protein interactions (PPI) using a dataset of 84.2 MB. The Logistic Regression, Decision Tree, and Random Forest models all achieved 100% accuracy, precision, recall, and F1-score. However, such perfect performance raises concerns of potential overfitting

Random Forest, an ensemble learning method, generally reduces overfitting compared to Decision Trees by averaging multiple trees (Breiman, 2001). However, achieving 100% accuracy still raises concerns, making it necessary to assess feature importance and check for data leakage (Hastie, Tibshirani, & Friedman, 2017).

Conversely, the XGBoost model (XGBClassifier) showed poor performance, with an accuracy of 50% and a failure to classify class 1 correctly. This may be due to improper hyperparameter tuning, class imbalance, or insufficient data for learning (Chen & Guestrin, 2016). Further optimization techniques, such as adjusting learning rate, increasing the number of estimators, or applying SMOTE for handling class imbalance, could improve XGBoost's performance (Lemaitre, Nogueira, & Aridas, 2017).

To ensure robust model evaluation, future studies should perform k-fold cross-validation, test the models on an independent dataset, and apply regularization techniques to avoid overfitting. Given the results, Random Forest appears to be the most stable model, but further testing is required to confirm its real-world applicability. This study demonstrates the potential of machine learning-based PPI classification but also emphasizes the importance of addressing overfitting, data imbalance, and feature selection for real-world applicability in bioinformatics and biotechnology. However, such perfect performance raises concerns of potential overfitting, where models might perform well on the training data but fail to generalize to unseen data

To address this, future work should incorporate k-fold cross-validation, use independent test sets, and apply regularization techniques. These steps will help mitigate overfitting and ensure that the models' performance is robust and generalizable. Random Forest, as an ensemble method, showed promise in preventing overfitting compared to Decision Trees, but further validation is necessary to confirm its real-world applicability in biological systems. Additionally, the poor performance of the XGBoost model (50% accuracy) suggests the need for further optimization, such as better Future studies with larger datasets and more refined techniques will be essential to verify the results and enhance model robustness for real-world use.hyperparameter tuning, handling class imbalance, and exploring feature engineering techniques to improve its performance.

References

Cleveland Clinic. (n.d.). *Amino acids*. Cleveland Clinic. https://my.clevelandclinic.org/health/articles/22243-amino-acids

National Center for Biotechnology Information. (n.d.). *Amino acids*. National Institutes of Health. https://www.ncbi.nlm.nih.gov/books/NBK557845/

WebMD. (n.d.). *Foods high in amino acids*. WebMD. https://www.webmd.com/diet/foods-high-in-amino-acids

ScienceDirect. (n.d.). *Selenocysteine*. ScienceDirect. https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/selenocysteine

PubMed. (2005). *The role of selenocysteine in human health*. https://pubmed.ncbi.nlm.nih.gov/15967579/

Sureja, S. (2023). *PPI dataset*. Kaggle. https://www.kaggle.com/datasets/spandansureja/ppi-dataset/data

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., & Walter, P. (2019). *Molecular biology of the cell* (6th ed.). Garland Science.

Berg, J. M., Tymoczko, J. L., Gatto, G. J., & Stryer, L. (2015). *Biochemistry* (8th ed.). W. H. Freeman.

Branden, C., & Tooze, J. (2012). *Introduction to protein structure* (2nd ed.). Garland Science.

Nelson, D. L., & Cox, M. M. (2021). *Lehninger principles of biochemistry* (8th ed.). W. H. Freeman.

McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56. https://doi.org/10.25080/Majora-92bf1922-00a

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Waskom, M. L., Botvinnik, O., O'Kane, D., Hobson, P., Seabold, S., & Mueller, A. (2020). seaborn: Statistical data visualization. *Journal of Open Source Software, 5*(51), 2236. https://doi.org/10.21105/joss.02236

Python Software Foundation. (2020). *itertools — Functions creating iterators for efficient looping*. https://docs.python.org/3/library/itertools.html

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., ... & De Hoon, M. J. L. (2009). Biopython: Freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics, 25*(11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163

Python Software Foundation. (2020). *warnings — Issue warning messages*. https://docs.python.org/3/library/warnings.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785