capstone

April 23, 2025

0.1 Data import and Cleaning

[1]:
```python
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import Bio
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, roc_auc_score, roc_curve
from Bio.SeqUtils.ProtParam import ProteinAnalysis
import requests
from io import StringIO
from Bio import SeqIO

epitopes = pd.read_csv(r'/Users/tariq/Documents/capstone/data/
 ↪epitope_table_export_1740279588.csv')
assays = pd.read_csv(r'/Users/tariq/Documents/capstone/data/
 ↪tcell_table_export_1740279970.csv')

def fetch_full_sequence(url):
    if pd.notna(url):  # Check if the URL is not NaN
        url = f'{url}.fasta'
        try:
            response = requests.get(url)
            if response.status_code == 200:
                fasta_io = StringIO(response.text)
                records = list(SeqIO.parse(fasta_io, "fasta"))
                if records:  # Check if there are any records
                    return str(records[0].seq)
                else:
                    print("No records found in the FASTA file.")
        except requests.exceptions.RequestException as e:
            print(f"Request failed: {e}")
    return None
```

```
#epitopes['Full Sequence'] = epitopes['Epitope - Molecule Parent IRI'].
↪apply(fetch_full_sequence)
epitopes = pd.read_csv(r'/Users/tariq/Documents/capstone/data/epitope_full_seq.
↪csv')
```

```
/var/folders/5j/4p7c5_1x2fg18bk0nf74_hg40000gn/T/ipykernel_31212/1845155022.py:1
5: DtypeWarning: Columns (13,14,45,46,47,48,49,54,55,56,57,60,65,66,67,68,69,70,
71,72,73,74,75,76,77,78,79,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99
,100,101,102,105,106,107,108,109,110,111,112,113,115,120,123,128,132,134,135,141
,142,143,144,145,149,152) have mixed types. Specify dtype option on import or
set low_memory=False.
  assays = pd.read_csv(r'/Users/tariq/Documents/capstone/data/tcell_table_export
_1740279970.csv')
```

[2]:
```
# make all the column names snake case
epitopes.columns = epitopes.columns.str.lower()
assays.columns = assays.columns.str.lower()

# remove spaces from column names
epitopes.columns = epitopes.columns.str.replace(' ', '')
epitopes.columns = epitopes.columns.str.replace('-', ' ')
epitopes.columns = epitopes.columns.str.replace(' ', '_')

assays.columns = assays.columns.str.replace(' ', '')
assays.columns = assays.columns.str.replace('-', ' ')
assays.columns = assays.columns.str.replace(' ', '_')

epitopes = epitopes.filter(['epitope_objecttype', 'epitope_name',␣
↪'fullsequence'])
assays = assays.filter(['epitope_name', 'epitope_moluculeparent', 'host_name',␣
↪'host_mhcpresent', 'assay_method','assay_responsemeasured',␣
↪'assay_qualitativemeasurement', 'mhcrestriction_name',␣
↪'mhcrestriction_class', 'assayantigen_name'])

# map mhc name and class from the assays dataframe to a new column in the␣
↪epitopes dataframe based on epitope_name
mhc = assays.filter(['epitope_name', 'mhcrestriction_name',␣
↪'mhcrestriction_class'])
mhc = mhc.drop_duplicates(subset=['epitope_name'])
epitopes = epitopes.merge(mhc, on='epitope_name', how='left')
```

[3]:
```
epitopes.head()
```

[3]:
```
  epitope_objecttype epitope_name  \
0     Linear peptide    AAGIGILTV
1     Linear peptide   AAGIGILTVI
2     Linear peptide   ACDPHSGHFV
```

```
3     Linear peptide   ADLVGFLLLK
4     Linear peptide    ADVEFCLSL

                                           fullsequence mhcrestriction_name  \
0  MPREDAHFIYGYPKKGHGHSYTTAEEAAGIGILTVILGVLLLIGCW…                HLA-A2
1  MPREDAHFIYGYPKKGHGHSYTTAEEAAGIGILTVILGVLLLIGCW…           HLA-A*02:01
2                                                NaN                HLA-A2
3  MSLEQRSLHCKPEEALEAQQEALGLVCVQAATSSSSPLVLGTLEEV…           HLA-A*11:01
4  MLLAVLYCLLWSFQTSAGHFPRACVSSKNLMEKECCPPWSGDRSPC…           HLA-B*44:03

  mhcrestriction_class
0                    I
1                    I
2                    I
3                    I
4                    I
```

[4]:
```
epitopes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28681 entries, 0 to 28680
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   epitope_objecttype    28681 non-null  object
 1   epitope_name          28681 non-null  object
 2   fullsequence          7164 non-null   object
 3   mhcrestriction_name   17613 non-null  object
 4   mhcrestriction_class  17613 non-null  object
dtypes: object(5)
memory usage: 1.1+ MB
```

0.2 Feature Engineering

[5]:
```
epitopes['epitope_length'] = epitopes['epitope_name'].str.len()
```

[6]:
```
# Function to count amino acids in a peptide
def count_amino_acids(peptide):
    try:
        # Create a ProteinAnalysis object for the peptide
        analyzer = ProteinAnalysis(peptide)
        # Get amino acid counts and normalize to frequencies
        aa_count = analyzer.count_amino_acids()
        total_aa = sum(aa_count.values())
        aa_freq = {aa: count for aa, count in aa_count.items()}
        # Add the peptide itself to the results
        aa_freq['peptide'] = peptide
        return aa_freq
    except Exception as e:
        # Handle invalid peptides (e.g., with non-standard amino acids)
        result = {aa: 0 for aa in 'ACDEFGHIKLMNPQRSTVWY'}
        result['peptide'] = peptide
        return result

# Create analyzer function that will be used in the next cell
def analyzer(peptide):
    return count_amino_acids(peptide)

# Use both epitope name and peptide sequence in the DataFrame
epitope_composition_df = epitopes.apply(lambda row:␣
↪count_amino_acids(row['epitope_name']), axis=1).apply(pd.Series)
```

[7]:
```
epitope_composition_df.head()
```

[7]:
```
   A  C  D  E  F  G  H  I  K  L  …  N  P  Q  R  S  T  V  W  Y     peptide
0  2  0  0  0  0  2  0  2  0  1  …  0  0  0  0  0  1  1  0  0   AAGIGILTV
1  2  0  0  0  0  2  0  3  0  1  …  0  0  0  0  0  1  1  0  0  AAGIGILTVI
2  1  1  1  0  1  1  2  0  0  0  …  0  1  0  0  1  0  1  0  0  ACDPHSGHFV
3  1  0  1  0  1  1  0  0  1  4  …  0  0  0  0  0  0  1  0  0  ADLVGFLLLK
4  1  1  1  1  1  0  0  0  0  2  …  0  0  0  0  1  0  1  0  0   ADVEFCLSL

[5 rows x 21 columns]
```

[8]:
```
# Example DataFrame with a 'peptide' column
df = pd.DataFrame({
    'peptide': ['ACDEFGHIK', 'LMNPQRSTV', 'WYFP']
})

# Kyte-Doolittle hydrophobicity scale
kyte_doolittle = {
    'I': 4.5, 'V': 4.2, 'L': 3.8, 'F': 2.8, 'C': 2.5,
    'M': 1.9, 'A': 1.8, 'G': -0.4, 'T': -0.7, 'S': -0.8,
    'W': -0.9, 'Y': -1.3, 'P': -1.6, 'H': -3.2, 'E': -3.5,
    'Q': -3.5, 'D': -3.5, 'N': -3.5, 'K': -3.9, 'R': -4.5
}

def compute_avg_hydrophobicity(peptide):
    # Get hydrophobicity scores for each amino acid; default to 0 if missing
    scores = [kyte_doolittle.get(aa, 0) for aa in peptide]
    return sum(scores) / len(scores) if scores else 0

# Apply the function to the 'peptide' column to create a new column 'avg_hydro'
epitopes['epitope_avg_hydro'] = epitopes['epitope_name'].
↪apply(compute_avg_hydrophobicity)
```

[9]:
```python
# Import the molecular_weight function from Bio.SeqUtils



def calculate_molecular_weight(peptide):
    """Calculate the molecular weight of a peptide sequence using Biopython."""
    try:
        # ProteinAnalysis only works with standard amino acids
        protein = ProteinAnalysis(peptide)
        return protein.molecular_weight()
    except Exception as e:
        # Handle peptides with non-standard amino acids
        return None

# Apply the function to calculate molecular weight for each epitope
epitopes['molecular_weight'] = epitopes['epitope_name'].
 ↪apply(calculate_molecular_weight)
```

[10]:
```python
def calculate_aromaticity(peptide):
    """Calculate the aromaticity of a peptide sequence using Biopython."""
    try:
        # ProteinAnalysis only works with standard amino acids
        protein = ProteinAnalysis(peptide)
        return protein.aromaticity()
    except Exception as e:
        # Handle peptides with non-standard amino acids
        return None

# Apply the function to calculate molecular weight for each epitope
epitopes['aromaticity'] = epitopes['epitope_name'].apply(calculate_aromaticity)
```

[11]:
```python
def calculate_isoelectric_point(peptide):
    """Calculate the isoelectric point of a peptide sequence using Biopython."""
    try:
        # ProteinAnalysis only works with standard amino acids
        protein = ProteinAnalysis(peptide)
        return protein.isoelectric_point()
    except Exception as e:
        # Handle peptides with non-standard amino acids
        return None

# Apply the function to calculate molecular weight for each epitope
epitopes['isoelectric_point'] = epitopes['epitope_name'].
 ↪apply(calculate_isoelectric_point)
```

[12]:
```python
def calculate_instability(peptide):
    """Calculate the instability of a peptide sequence using Biopython."""
    try:
        # ProteinAnalysis only works with standard amino acids
        protein = ProteinAnalysis(peptide)
        return protein.instability_index()
    except Exception as e:
        # Handle peptides with non-standard amino acids
        return None

# Apply the function to calculate molecular weight for each epitope
epitopes['instability'] = epitopes['epitope_name'].apply(calculate_instability)
```

[13]:
```
def calculate_charge_at_pH7(peptide):
    """Calculate the charge of a peptide sequence at pH 7 using Biopython."""
    try:
        # ProteinAnalysis only works with standard amino acids
        protein = ProteinAnalysis(peptide)
        return protein.charge_at_pH(7)
    except Exception as e:
        # Handle peptides with non-standard amino acids
        return None

# Apply the function to calculate molecular weight for each epitope
epitopes['charge_at_pH7'] = epitopes['epitope_name'].
 ↪apply(calculate_charge_at_pH7)
```

[14]:
```
epitopes.head()
```

[14]:
```
  epitope_objecttype epitope_name  \
0     Linear peptide    AAGIGILTV
1     Linear peptide   AAGIGILTVI
2     Linear peptide   ACDPHSGHFV
3     Linear peptide   ADLVGFLLLK
4     Linear peptide    ADVEFCLSL

                                        fullsequence mhcrestriction_name  \
0  MPREDAHFIYGYPKKGHGHSYTTAEEAAGIGILTVILGVLLLIGCW...              HLA-A2
1  MPREDAHFIYGYPKKGHGHSYTTAEEAAGIGILTVILGVLLLIGCW...         HLA-A*02:01
2                                                NaN              HLA-A2
3  MSLEQRSLHCKPEEALEAQQEALGLVCVQAATSSSSPLVLGTLEEV...         HLA-A*11:01
4  MLLAVLYCLLWSFQTSAGHFPRACVSSKNLMEKECCPPWSGDRSPC...         HLA-B*44:03

  mhcrestriction_class  epitope_length  epitope_avg_hydro  molecular_weight  \
0                    I               9           2.122222          813.9814
1                    I              10           2.360000          927.1390
2                    I              10          -0.140000         1069.1507
3                    I              10           1.620000         1088.3394
4                    I               9           1.233333          996.1348
```

```
   aromaticity  isoelectric_point  instability  charge_at_pH7
0     0.000000           5.570017    11.422222      -0.204125
1     0.000000           5.570017    11.280000      -0.204125
2     0.100000           5.972266    61.830000      -1.038557
3     0.100000           5.880358   -16.470000      -0.204004
4     0.111111           4.050028    20.855556      -2.210095
```

0.3 Generation of Negative Samples

[15]:
```python
def generate_negatives(row):
    epitope = row["epitope_name"]
    full_seq = row["fullsequence"]
    mhc = row["mhcrestriction_name"]

    # Handle missing or empty sequences
    if pd.isnull(full_seq) or full_seq == "":
        return []

    epitope = str(epitope)
    full_seq = str(full_seq)
    ep_len = len(epitope)

    negatives = []
    for i in range(len(full_seq) - ep_len + 1):
        window = full_seq[i:i+ep_len]
        if window != epitope:
            negatives.append({"peptide": window, "mhc": mhc})
    return negatives

# Apply the function to each row
'''
negatives = pd.DataFrame()
negatives['negatives'] = epitopes.apply(generate_negatives, axis=1)
negatives = negatives[["negatives"]].explode("negatives").reset_index(drop=True)
negatives.dropna(subset=["negatives"], inplace=True)


# Remove duplicate peptide-mhc combinations
print(f"Shape before removing duplicates: {negatives.shape}")
negatives = negatives.drop_duplicates(subset=['negatives'])
print(f"Shape after removing duplicates: {negatives.shape}")

# Check for any remaining NaN values
print(f"Number of NaN values in negatives: {negatives['negatives'].isna().
 ↪sum()}")

# Extract peptide and mhc into separate columns
```

```
negatives['peptide'] = negatives['negatives'].apply(lambda x: x['peptide'])
negatives['mhc'] = negatives['negatives'].apply(lambda x: x['mhc'])

# Calculate features on the peptide column
negatives['peptide_length'] = negatives['peptide'].apply(len)
negatives['peptide_avg_hydro'] = negatives['peptide'].
 ↪apply(compute_avg_hydrophobicity)
negatives['molecular_weight'] = negatives['peptide'].
 ↪apply(calculate_molecular_weight)
negatives['aromaticity'] = negatives['peptide'].apply(calculate_aromaticity)
negatives['isoelectric_point'] = negatives['peptide'].
 ↪apply(calculate_isoelectric_point)
negatives['instability'] = negatives['peptide'].apply(calculate_instability)
negatives['charge_at_pH7'] = negatives['peptide'].apply(calculate_charge_at_pH7)

# Drop the original dictionary column if no longer needed
negatives.drop('negatives', axis=1, inplace=True)
'''
```

[15]:
```
'\nnegatives = pd.DataFrame()\nnegatives[\'negatives\'] =
epitopes.apply(generate_negatives, axis=1)\nnegatives = negatives[["negatives"]]
.explode("negatives").reset_index(drop=True)\nnegatives.dropna(subset=["negative
s"], inplace=True)\n\n\n# Remove duplicate peptide-mhc
combinations\nprint(f"Shape before removing duplicates:
{negatives.shape}")\nnegatives =
negatives.drop_duplicates(subset=[\'negatives\'])\nprint(f"Shape after removing
duplicates: {negatives.shape}")\n\n# Check for any remaining NaN
values\nprint(f"Number of NaN values in negatives:
{negatives[\'negatives\'].isna().sum()}")\n\n# Extract peptide and mhc into
separate columns\nnegatives[\'peptide\'] = negatives[\'negatives\'].apply(lambda
x: x[\'peptide\'])\nnegatives[\'mhc\'] = negatives[\'negatives\'].apply(lambda
x: x[\'mhc\'])\n\n# Calculate features on the peptide
column\nnegatives[\'peptide_length\'] =
negatives[\'peptide\'].apply(len)\nnegatives[\'peptide_avg_hydro\'] = negatives[
\'peptide\'].apply(compute_avg_hydrophobicity)\nnegatives[\'molecular_weight\']
= negatives[\'peptide\'].apply(calculate_molecular_weight)\nnegatives[\'aromatic
ity\'] = negatives[\'peptide\'].apply(calculate_aromaticity)\nnegatives[\'isoele
ctric_point\'] = negatives[\'peptide\'].apply(calculate_isoelectric_point)\nnega
tives[\'instability\'] = negatives[\'peptide\'].apply(calculate_instability)\nne
gatives[\'charge_at_pH7\'] =
negatives[\'peptide\'].apply(calculate_charge_at_pH7)\n\n# Drop the original
dictionary column if no longer needed\nnegatives.drop(\'negatives\', axis=1,
inplace=True)\n'
```

[16]:
```
negatives = pd.read_csv("data/negatives_MHC.csv")
```

```
/var/folders/5j/4p7c5_1x2fg18bk0nf74_hg40000gn/T/ipykernel_31212/1811011591.py:1
```

```
: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or
set low_memory=False.
  negatives = pd.read_csv("data/negatives_MHC.csv")
```

[17]:
```
nine_mers = epitopes[epitopes['epitope_length'] == 9]
nine_mers.to_csv("data/ninemer_epitopes.csv", index=False)
```

[18]:
```
ninemer_negatives = negatives[negatives['peptide_length'] == 9]
ninemer_negatives_trimmed = ninemer_negatives[:50000]
ninemer_negatives_trimmed.to_csv("data/ninemer_negatives_trimmed.csv",␣
 ↪index=False)
```

0.4 EDA

0.4.1 Data Summary

[19]:
```
epitopes.head()
```

[19]:
```
  epitope_objecttype epitope_name  \
0     Linear peptide    AAGIGILTV
1     Linear peptide   AAGIGILTVI
2     Linear peptide   ACDPHSGHFV
3     Linear peptide   ADLVGFLLLK
4     Linear peptide    ADVEFCLSL

                                         fullsequence mhcrestriction_name  \
0  MPREDAHFIYGYPKKGHGHSYTTAEEAAGIGILTVILGVLLLIGCW…              HLA-A2
1  MPREDAHFIYGYPKKGHGHSYTTAEEAAGIGILTVILGVLLLIGCW…         HLA-A*02:01
2                                                NaN              HLA-A2
3  MSLEQRSLHCKPEEALEAQQEALGLVCVQAATSSSSPLVLGTLEEV…         HLA-A*11:01
4  MLLAVLYCLLWSFQTSAGHFPRACVSSKNLMEKECCPPWSGDRSPC…         HLA-B*44:03

  mhcrestriction_class  epitope_length  epitope_avg_hydro  molecular_weight  \
0                    I               9           2.122222          813.9814
1                    I              10           2.360000          927.1390
2                    I              10          -0.140000         1069.1507
3                    I              10           1.620000         1088.3394
4                    I               9           1.233333          996.1348

   aromaticity  isoelectric_point  instability  charge_at_pH7
0     0.000000           5.570017    11.422222      -0.204125
1     0.000000           5.570017    11.280000      -0.204125
2     0.100000           5.972266    61.830000      -1.038557
3     0.100000           5.880358   -16.470000      -0.204004
4     0.111111           4.050028    20.855556      -2.210095
```

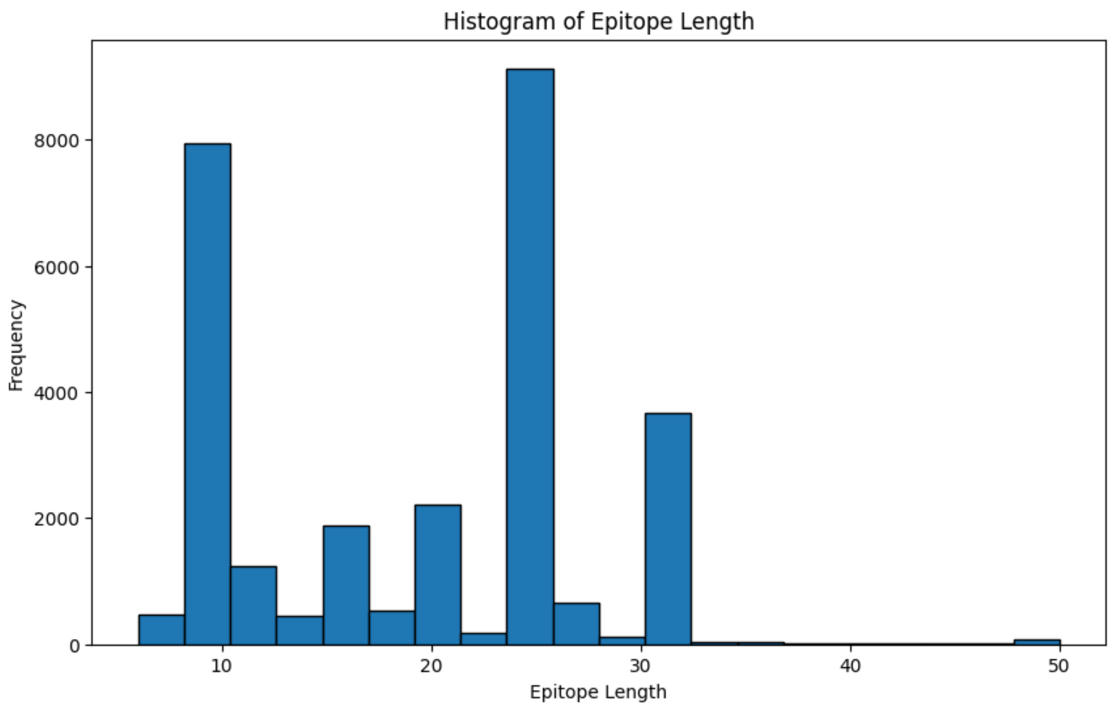
[21]:
```
epitopes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28681 entries, 0 to 28680
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   epitope_objecttype     28681 non-null  object
 1   epitope_name           28681 non-null  object
 2   fullsequence           7164 non-null   object
 3   mhcrestriction_name    17613 non-null  object
 4   mhcrestriction_class   17613 non-null  object
 5   epitope_length         28681 non-null  int64
 6   epitope_avg_hydro      28681 non-null  float64
 7   molecular_weight       28623 non-null  float64
 8   aromaticity            28681 non-null  float64
 9   isoelectric_point      28681 non-null  float64
 10  instability            28623 non-null  float64
 11  charge_at_pH7          28681 non-null  float64
dtypes: float64(6), int64(1), object(5)
memory usage: 2.6+ MB
```

0.4.2 Properties of Epitopes

Length

[23]:
```python
# hist of epitope length
plt.figure(figsize=(10, 6))
plt.hist(epitopes['epitope_length'], bins=20, edgecolor='black')
plt.xlabel('Epitope Length')
plt.ylabel('Frequency')
plt.title('Histogram of Epitope Length')
plt.show()
```

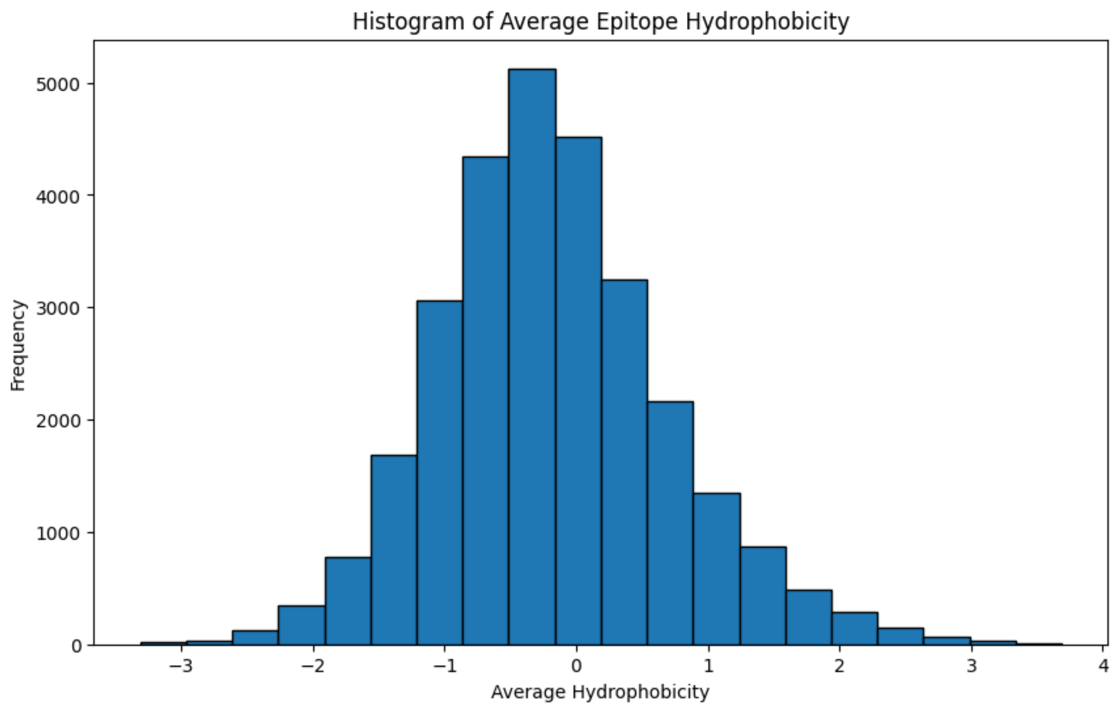


[24]:
```
epitopes['epitope_length'].describe()
```

[24]:
```
count    28681.000000
mean        19.389422
std          8.255925
min          6.000000
25%         10.000000
50%         20.000000
75%         25.000000
max         50.000000
Name: epitope_length, dtype: float64
```

Hydrophobicity

[25]:
```
# histogram of average hydrophobicity
plt.figure(figsize=(10, 6))
plt.hist(epitopes['epitope_avg_hydro'], bins=20, edgecolor='black')
plt.xlabel('Average Hydrophobicity')
plt.ylabel('Frequency')
plt.title('Histogram of Average Epitope Hydrophobicity')
plt.show()
```



[26]:
```
epitopes['epitope_avg_hydro'].describe()
```

[26]:
```
count    28681.000000
mean        -0.178410
std          0.883064
min         -3.312000
25%         -0.762500
50%         -0.240000
75%          0.333333
max          3.688889
Name: epitope_avg_hydro, dtype: float64
```

Composition

[27]:
```
# plot the composition of the epitopes, sort by the composition of the amino␣
↪acids
# Calculate mean composition and sort

'''
mean_composition = epitope_composition_df.mean().sort_values(ascending=False)

# Plot the sorted composition
plt.figure(figsize=(10, 6))
plt.bar(mean_composition.index, mean_composition.values)
plt.xlabel('Amino Acid')
plt.ylabel('Composition')
plt.title('Composition of Epitopes')
plt.show()

'''
```

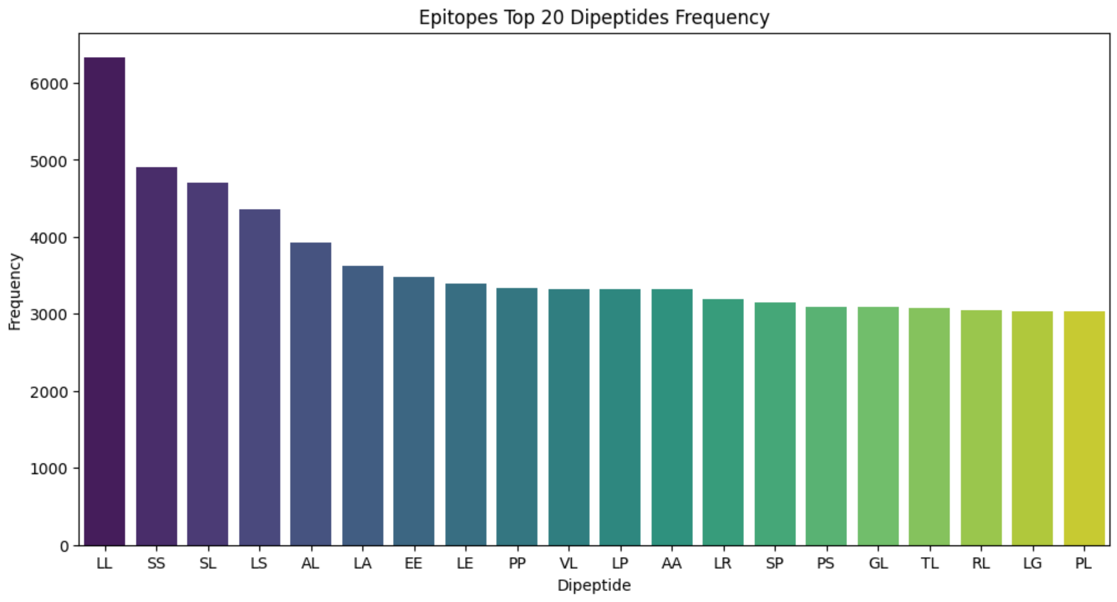
[27]:
```
"\nmean_composition =
epitope_composition_df.mean().sort_values(ascending=False)\n\n# Plot the sorted
composition\nplt.figure(figsize=(10, 6))\nplt.bar(mean_composition.index,
mean_composition.values)\nplt.xlabel('Amino
Acid')\nplt.ylabel('Composition')\nplt.title('Composition of
Epitopes')\nplt.show()\n\n"
```

n-gram frequency analysis

[28]:
```
def ngram_frequency(peptides, n=2):
    ngrams = []
    for peptide in peptides:
        if len(peptide) < n:
            continue
        for i in range(len(peptide) - n + 1):
            ngram = peptide[i:i+n]
            ngrams.append(ngram)
    return Counter(ngrams)

dipeptide_freq = ngram_frequency(epitopes['epitope_name'], n=2)

df_ngram = pd.DataFrame(dipeptide_freq.items(), columns=['ngram', 'count'])
df_ngram = df_ngram.sort_values('count', ascending=False)

top_n = 20
top_ngram = df_ngram.head(top_n)

plt.figure(figsize=(12, 6))
sns.barplot(x='ngram', y='count', data=top_ngram, palette="viridis")
plt.title(f"Epitopes Top {top_n} Dipeptides Frequency")
plt.xlabel("Dipeptide")
plt.ylabel("Frequency")
plt.show()
```

```
/var/folders/5j/4p7c5_1x2fg18bk0nf74_hg40000gn/T/ipykernel_31212/733366050.py:20
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

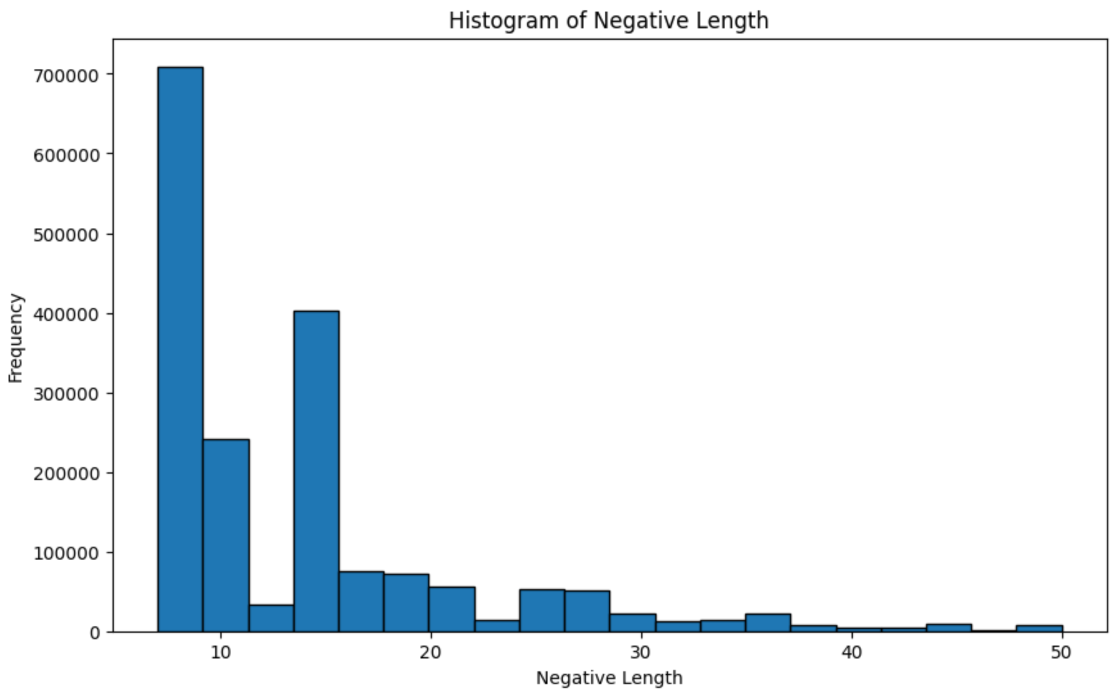
```
sns.barplot(x='ngram', y='count', data=top_ngram, palette="viridis")
```



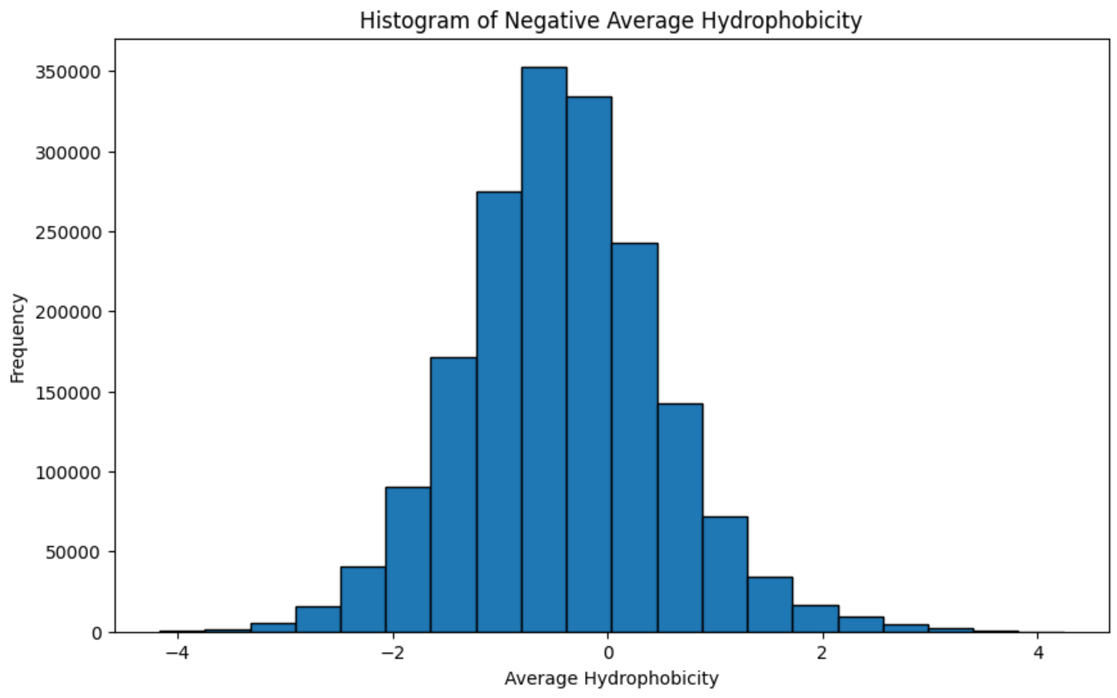
MHC Binding Affinity

0.4.3 Properties of negative samples

```
[29]: # hist of negative length
plt.figure(figsize=(10, 6))
plt.hist(negatives['peptide_length'], bins=20, edgecolor='black')
plt.xlabel('Negative Length')
plt.ylabel('Frequency')
plt.title('Histogram of Negative Length')
plt.show()
```



[30]:
```
# histogram of average hydrophobicity
plt.figure(figsize=(10, 6))
plt.hist(negatives['peptide_avg_hydro'], bins=20, edgecolor='black')
plt.xlabel('Average Hydrophobicity')
plt.ylabel('Frequency')
plt.title('Histogram of Negative Average Hydrophobicity')
plt.show()
```



[31]:
```
negatives['peptide_avg_hydro'].mean()
```

[31]:
```
-0.4169864724628861
```

[32]:
```
# plot the composition of the negatives, sort by the composition of the amino␣
↪acids
# Calculate mean composition and sort

'''
mean_composition = negatives_composition_df.mean().sort_values(ascending=False)

# Plot the sorted composition
plt.figure(figsize=(10, 6))
plt.bar(mean_composition.index, mean_composition.values)
plt.xlabel('Amino Acid')
plt.ylabel('Composition')
plt.title('Composition of Negative Samples')
plt.show()
'''
```

[32]:
```
"\nmean_composition =
negatives_composition_df.mean().sort_values(ascending=False)\n\n# Plot the
sorted composition\nplt.figure(figsize=(10, 6))\nplt.bar(mean_composition.index,
mean_composition.values)\nplt.xlabel('Amino
Acid')\nplt.ylabel('Composition')\nplt.title('Composition of Negative
Samples')\nplt.show()\n"
```

0.5 Modeling

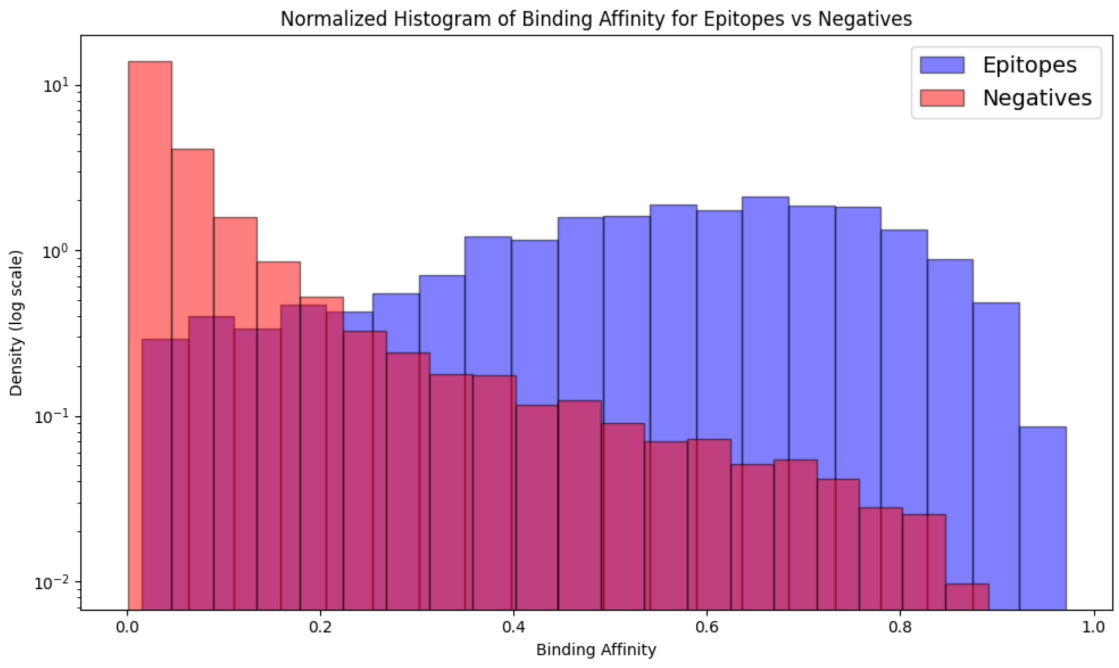
0.5.1 Data Preprocessing

[33]:
```
epitopes = pd.read_csv("data/ninemer_epitopes.csv")
epitopes = epitopes.drop(columns=['epitope_objecttype', 'fullsequence',␣
 ↪'mhcrestriction_name', 'mhcrestriction_class', 'epitope_length'])
epitopes = epitopes.rename(columns={'epitope_name': 'peptide',␣
 ↪'epitope_avg_hydro': 'peptide_avg_hydro'})
epitopes_BA_pred = pd.read_csv("data/ninemer_epitopes_BA_pred.csv")
epitopes_composition = epitopes.apply(lambda row:␣
 ↪count_amino_acids(row['peptide']), axis=1).apply(pd.Series)

negatives = pd.read_csv("data/ninemer_negatives_trimmed.csv")
negatives = negatives.drop(columns=['mhc', 'peptide_length'])
negatives = negatives.rename(columns={'peptide': 'peptide'})
negatives = negatives.drop_duplicates(subset=['peptide'])
negatives_BA_pred = pd.read_csv("data/ninemer_negatives_trimmed_BA_pred.csv")
negatives_BA_pred = negatives_BA_pred.drop_duplicates(subset=['peptide'])
negatives_composition = negatives.apply(lambda row:␣
 ↪count_amino_acids(row['peptide']), axis=1).apply(pd.Series)
```

[34]:
```
# Merge the 'Score_BA' column from epitopes_BA_pred into the epitopes dataframe
epitopes = pd.merge(epitopes, epitopes_BA_pred[['peptide', 'Score_BA',␣
 ↪'ic50']], on='peptide', how='left')
#epitopes = pd.merge(epitopes, epitopes_composition, on='peptide', how='left')

negatives = pd.merge(negatives, negatives_BA_pred[['peptide', 'Score_BA',␣
 ↪'ic50']], on='peptide', how='left')
#negatives = pd.merge(negatives, negatives_composition, on='peptide',␣
 ↪how='left')
```

[35]:
```
# plot Score_BA for epitopes and negatives overlaid on the same plot
plt.figure(figsize=(10, 6))

# Use density instead of raw counts to normalize the histograms
plt.hist(epitopes['Score_BA'], bins=20, alpha=0.5, color='blue',␣
 ↪edgecolor='black',
         label='Epitopes', density=True)
plt.hist(negatives['Score_BA'], bins=20, alpha=0.5, color='red',␣
 ↪edgecolor='black',
         label='Negatives', density=True)

# Alternative approach: use log scale for y-axis
```

```
plt.yscale('log')

plt.xlabel('Binding Affinity')
plt.ylabel('Density (log scale)')
plt.title('Normalized Histogram of Binding Affinity for Epitopes vs Negatives')
plt.legend(prop={'size': 14})  # Increased legend font size
plt.tight_layout()
plt.show()
```



[ ]:
```

```

[36]:
```
# Add label column to epitopes dataframe (positive class = 1)
epitopes['label'] = 1

# Add label column to negatives dataframe (negative class = 0)
negatives['label'] = 0

# Combine the positive and negative examples
combined_data = pd.concat([epitopes, negatives], ignore_index=True)

# Shuffle the combined dataset
combined_data = combined_data.sample(frac=1, random_state=42).
 ↪reset_index(drop=True)

# Define features and target
```

```python
X = combined_data.drop(columns=['peptide', 'label'])
y = combined_data['label']

# Identify numerical columns to scale (exclude one-hot encoded amino acid␣
 ↪columns)
numerical_cols = ['peptide_avg_hydro', 'molecular_weight', 'aromaticity',␣
 ↪'isoelectric_point', 'instability','Score_BA', 'charge_at_pH7']
amino_acid_cols = [col for col in X.columns if col not in numerical_cols]

# Split the data into training and testing sets (80% train, 20% test)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale numerical features using StandardScaler
scaler = StandardScaler()
X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])

# Print the shapes to verify the split
print(f"Training set: {X_train.shape[0]} samples")
print(f"Testing set: {X_test.shape[0]} samples")
print(f"Positive samples in training: {sum(y_train == 1)}")
print(f"Negative samples in training: {sum(y_train == 0)}")
print(f"Positive samples in testing: {sum(y_test == 1)}")
print(f"Negative samples in testing: {sum(y_test == 0)}")
print(f"Scaled numerical features: {numerical_cols}")
```

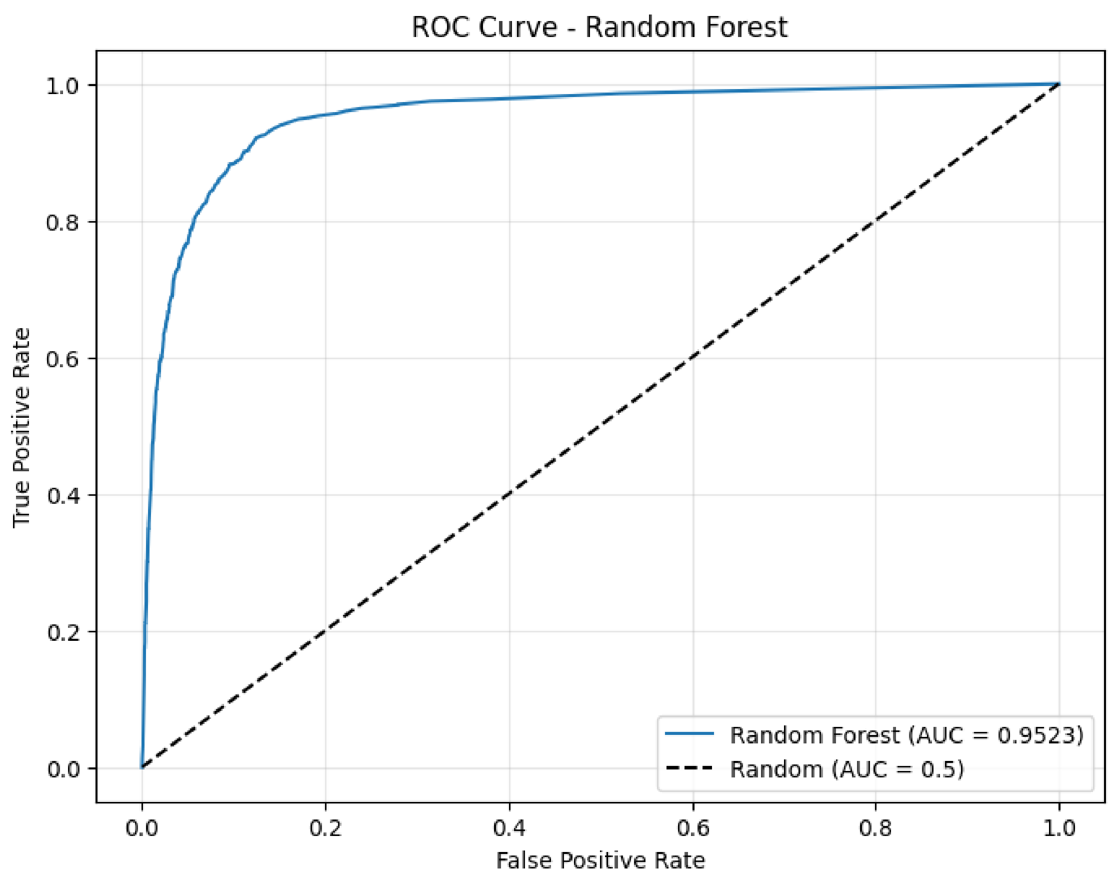
```
Training set: 20502 samples
Testing set: 5126 samples
Positive samples in training: 4236
Negative samples in training: 16266
Positive samples in testing: 1059
Negative samples in testing: 4067
Scaled numerical features: ['peptide_avg_hydro', 'molecular_weight',
'aromaticity', 'isoelectric_point', 'instability', 'Score_BA', 'charge_at_pH7']
```

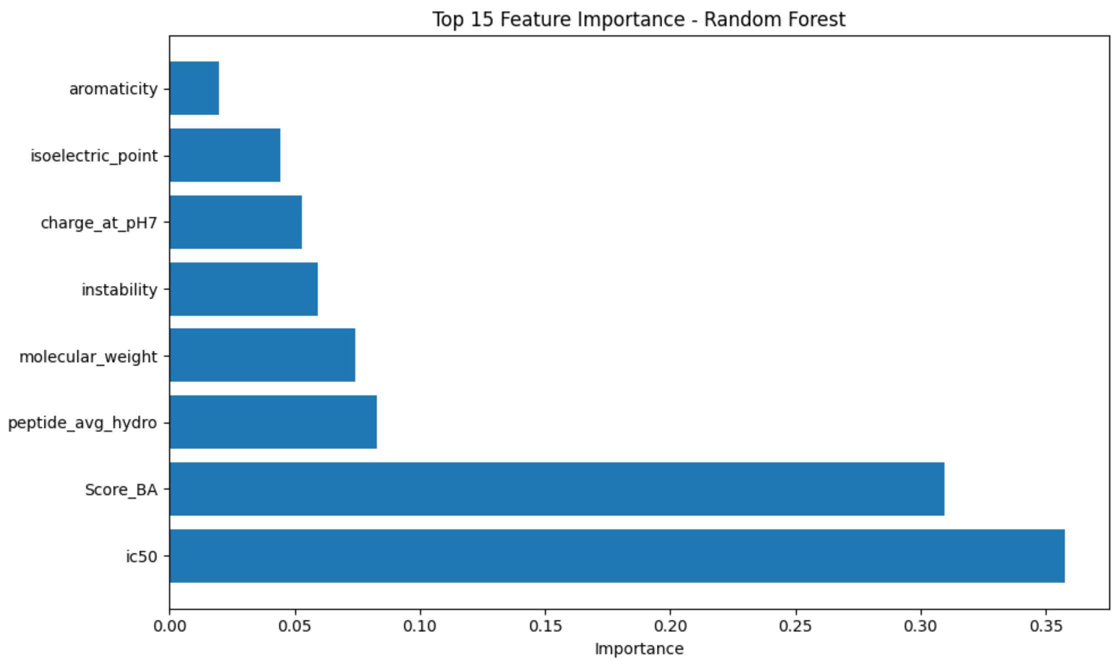
[37]:
```python
# drop the Score_BA column
#X_train = X_train.drop(columns=['Score_BA'])
#X_test = X_test.drop(columns=['Score_BA'])
```

[38]:
```python
# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(
    n_estimators=100,  # Number of trees
    max_depth=None,     # Maximum depth of trees
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]  # Probability estimates␣
 ↪for positive class

# Evaluate the model
print("Random Forest Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"\nROC AUC Score: {roc_auc:.4f}")

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Random Forest (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.5)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': rf_model.feature_importances_
})
```

```
feature_importance = feature_importance.sort_values('Importance',␣
  ↪ascending=False)

# Plot top 15 features
plt.figure(figsize=(10, 6))
top_features = feature_importance.head(15)
plt.barh(np.arange(len(top_features)), top_features['Importance'],␣
  ↪align='center')
plt.yticks(np.arange(len(top_features)), top_features['Feature'])
plt.xlabel('Importance')
plt.title('Top 15 Feature Importance - Random Forest')
plt.tight_layout()
plt.show()
```

```
Random Forest Model Evaluation:
Accuracy: 0.9138

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      4067
           1       0.82      0.75      0.78      1059

    accuracy                           0.91      5126
   macro avg       0.88      0.85      0.86      5126
weighted avg       0.91      0.91      0.91      5126


Confusion Matrix:
[[3893  174]
 [ 268  791]]

ROC AUC Score: 0.9523
```

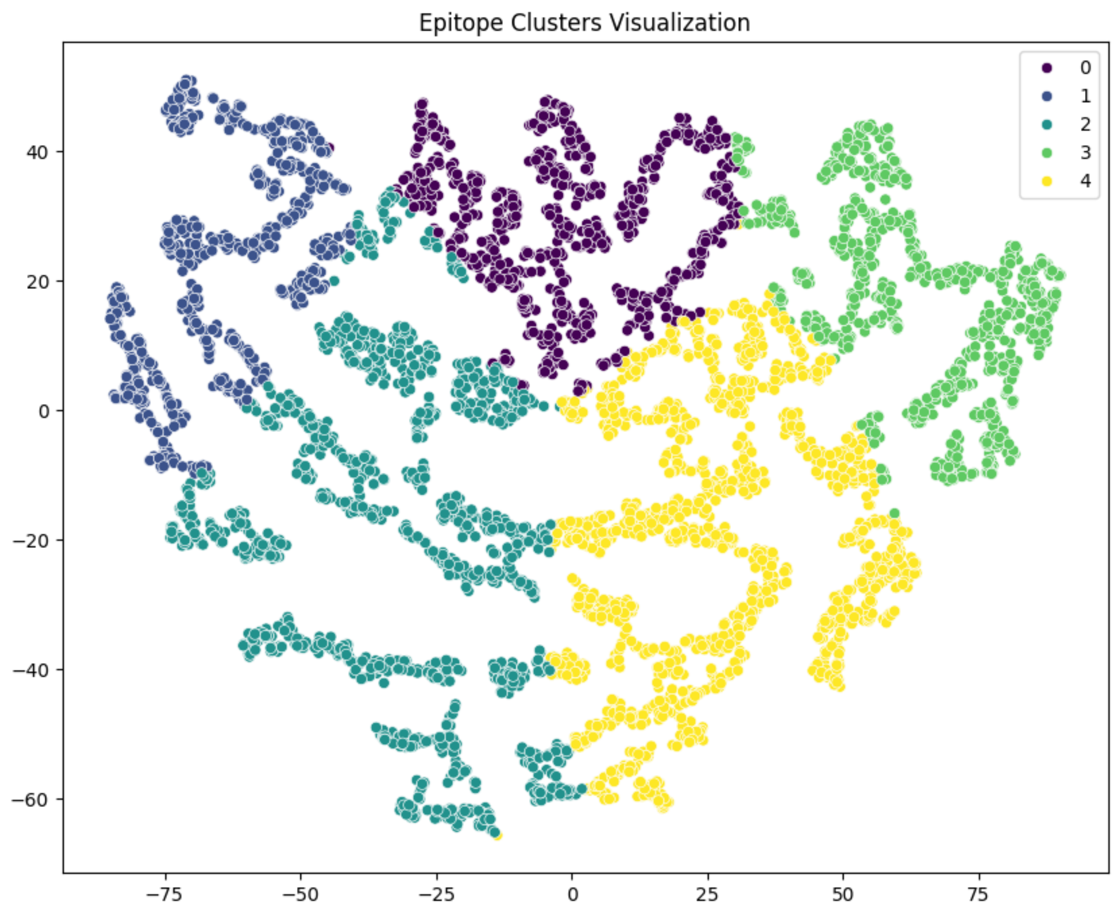

ROC Curve - Random Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Random Forest (AUC = 0.9523)
Random (AUC = 0.5)


Top 15 Feature Importance - Random Forest
aromaticity
isoelectric_point
charge_at_pH7
instability
molecular_weight
peptide_avg_hydro
Score_BA
ic50
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Importance

0.5.2 Clustering

[39]:
```python
# Example clustering approach
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer

# Create feature matrix (using your existing features)
X = pd.concat([epitopes[['peptide_avg_hydro', 'molecular_weight', 'aromaticity',
                         'isoelectric_point', 'instability', 'charge_at_pH7',␣
 ↪'Score_BA']],
               # Add amino acid composition features
               pd.get_dummies(epitopes['peptide'].apply(lambda x: ''.join(x)),␣
 ↪prefix='pos')], axis=1)

# Handle missing values
print("Number of NaN values in dataset:", X.isna().sum().sum())
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Option 1: K-means clustering
kmeans = KMeans(n_clusters=5, random_state=42)  # Adjust number of clusters
clusters = kmeans.fit_predict(X_imputed)
epitopes['cluster'] = clusters

# Option 2: Hierarchical clustering
# hclust = AgglomerativeClustering(n_clusters=5)
# clusters = hclust.fit_predict(X_imputed)

# Visualize with t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_imputed)

plt.figure(figsize=(10, 8))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=clusters, palette='viridis')
plt.title('Epitope Clusters Visualization')
plt.show()

# Analyze cluster characteristics
for cluster_id in range(5):
    cluster_peptides = epitopes[epitopes['cluster'] == cluster_id]
    print(f"Cluster {cluster_id}: {len(cluster_peptides)} peptides")
    print(f"Average binding score: {cluster_peptides['Score_BA'].mean():.2f}")
    print(f"Average hydrophobicity: {cluster_peptides['peptide_avg_hydro'].
↪mean():.2f}")

    # Find sequence motifs in cluster
    motif_analysis = pd.DataFrame()
    for i in range(9):  # For 9-mer peptides
        aa_counts = cluster_peptides['peptide'].str[i].
↪value_counts(normalize=True)
        motif_analysis[f'Position_{i+1}'] = aa_counts
    print("Top amino acids at each position:")
    for col in motif_analysis.columns:
        top_aas = motif_analysis[col].nlargest(3)
        print(f"{col}: {', '.join([f'{aa}({freq:.2f})' for aa, freq in top_aas.
↪items()])}")
    print("\n")
```

```
Number of NaN values in dataset: 937
```



```
Cluster 0: 827 peptides
```

```
Average binding score: 0.55
Average hydrophobicity: 0.02
Top amino acids at each position:
Position_1: S(0.12), L(0.11), A(0.07)
Position_2: L(0.22), P(0.13), S(0.09)
Position_3: S(0.12), L(0.10), A(0.08)
Position_4: S(0.13), E(0.13), P(0.13)
Position_5: S(0.10), L(0.10), R(0.07)
Position_6: S(0.12), L(0.12), P(0.09)
Position_7: P(0.13), S(0.13), L(0.09)
Position_8: S(0.12), P(0.12), L(0.09)
Position_9: L(0.31), V(0.16), I(0.10)


Cluster 1: 732 peptides
Average binding score: 0.53
Average hydrophobicity: 0.75
Top amino acids at each position:
Position_1: A(0.20), G(0.14), S(0.14)
Position_2: L(0.31), A(0.13), P(0.13)
Position_3: A(0.16), G(0.11), S(0.11)
Position_4: G(0.17), P(0.16), A(0.15)
Position_5: G(0.19), A(0.16), V(0.11)
Position_6: G(0.16), S(0.14), L(0.12)
Position_7: A(0.13), P(0.12), S(0.11)
Position_8: A(0.18), S(0.16), G(0.12)
Position_9: L(0.29), V(0.25), A(0.14)


Cluster 2: 1419 peptides
Average binding score: 0.56
Average hydrophobicity: 0.57
Top amino acids at each position:
Position_1: L(0.10), A(0.10), S(0.09)
Position_2: L(0.32), V(0.10), T(0.08)
Position_3: L(0.14), A(0.09), S(0.09)
Position_4: S(0.09), G(0.09), L(0.09)
Position_5: L(0.11), G(0.11), A(0.09)
Position_6: L(0.14), V(0.09), S(0.09)
Position_7: L(0.14), V(0.11), A(0.09)
Position_8: L(0.11), A(0.10), S(0.10)
Position_9: L(0.28), V(0.20), K(0.12)


Cluster 3: 863 peptides
Average binding score: 0.57
Average hydrophobicity: -0.31
Top amino acids at each position:
```

```
Position_1: R(0.14), F(0.12), Y(0.12)
Position_2: L(0.19), Y(0.15), R(0.10)
Position_3: Y(0.11), F(0.11), L(0.10)
Position_4: E(0.11), R(0.10), L(0.08)
Position_5: R(0.14), F(0.11), L(0.09)
Position_6: L(0.12), F(0.10), R(0.08)
Position_7: L(0.12), F(0.09), R(0.08)
Position_8: L(0.10), E(0.09), R(0.09)
Position_9: L(0.26), F(0.18), Y(0.12)


Cluster 4: 1454 peptides
Average binding score: 0.58
Average hydrophobicity: 0.19
Top amino acids at each position:
Position_1: F(0.12), K(0.11), R(0.10)
Position_2: L(0.27), V(0.09), Y(0.08)
Position_3: L(0.13), F(0.08), D(0.07)
Position_4: E(0.11), D(0.08), L(0.08)
Position_5: L(0.11), F(0.08), V(0.08)
Position_6: L(0.15), V(0.09), I(0.09)
Position_7: L(0.14), F(0.09), V(0.06)
Position_8: L(0.13), S(0.08), F(0.08)
Position_9: L(0.28), V(0.15), F(0.11)
```

0.5.3 New Model