In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

Split the data in to training and test sets.

In [ ]:
```
X_all = pd.read_csv('Dataset/train_X.csv')
y_all = pd.read_csv('Dataset/train_y.csv')
```

Here we define the one-hot encoder, which converts the string of sequence of amino acids to a set of values, either one or zero. Each value represents the existance of a certain amino acid on a specific position.

In [3]:
```
def one_hot_encode(df):
    enc = OneHotEncoder(handle_unknown='ignore')
    X = np.array(X_all['ConstructedAASeq_cln'].apply(lambda x: list(x)).to_list())
    enc.fit(X)
    return enc, enc.transform(df['ConstructedAASeq_cln'].apply(lambda x: list(x)).to_list()).toarray()
```

We also load all the CSV files in the descriptors. These files describe some properties of each amino acid, and all data are numeric.

In [4]:
```
# Load descriptors
dpps = pd.read_csv('Dataset/descriptors/DPPS.csv', header=2).drop('AA_3', axis=1)
df_combined = dpps
for file_name in ['Physical', 'Physical', 'ST-scale', 'T-scale', 'VHSE-scale', 'Z-scale']:
    df = pd.read_csv(f'Dataset/descriptors/{file_name}.csv', header=2).drop('AA_3', axis=1)
    df_combined = df_combined.merge(df, on='AA_1', how='inner')
df_combined = df_combined.set_index('AA_1')
```

After combining the tables, we can see there are 38 columns.

In [5]:
```
df_combined.head()
```

Out[5]:

AA_1	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	...	VHSE2	VHSE3	VHSE4	VHSE5	VHSE6	VHSE7	VHSE8	Z(1)	Z(2)	Z(3)
A	-1.02	-2.88	-0.56	0.36	-6.15	-1.68	0.04	-2.51	-1.94	-0.01	...	-1.11	-1.35	-0.92	0.02	-0.91	0.36	-0.48	0.07	-1.73	0.09
R	1.99	4.13	-4.41	-1.02	4.78	3.04	-9.06	6.71	4.41	0.07	...	1.45	1.24	1.27	1.55	1.47	1.30	0.83	2.88	2.52	-3.44
N	-2.19	1.86	0.38	-0.13	-2.30	1.41	-5.71	-1.11	1.73	-0.19	...	0.00	-0.37	0.69	-0.55	0.85	0.73	-0.80	3.22	1.45	0.84
D	-6.60	3.32	1.61	0.36	-3.25	1.95	-7.36	0.14	1.24	-0.15	...	0.67	-0.41	-0.01	-2.68	1.31	0.03	0.56	3.64	1.13	2.36
C	0.21	1.12	3.42	-0.68	-2.27	-1.22	3.11	-2.98	-1.70	1.57	...	-1.67	-0.46	-0.21	0.00	1.20	-1.61	-0.19	0.71	-0.97	4.13

5 rows × 38 columns

Then we encode the data, and train a baseline logistic regression model with this encoding. We can see the accuracy is 0.9006, which is good.

In [ ]:
```
one_hot_encoder, one_hot_encoded = one_hot_encode(X_all)
brightnesses = y_all['Brightness_Class']
X_train_oh, X_test_oh, y_train, y_test = train_test_split(one_hot_encoded, brightnesses, test_size=0.2, random_state=5)

# Baseline One-Hot Encoding Logistic Regression Model
# Train the model (~30s)
logreg = LogisticRegression(max_iter=1000, random_state=5)
logreg.fit(X_train_oh, y_train.values)

# Predict
y_pred = logreg.predict(X_test_oh)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.9006
```

In the following codes, we will create a dataframe to examine the coeficients of the logistic regression model, for each amino acid-posiiton combination.

In [ ]:
```
variant_nums = []
for category in one_hot_encoder.categories_:
    variant_nums.append(len(category))

positions = []
i = 0
for n in variant_nums:
    for _ in range(n):
        positions.append(i)
    i += 1

one_hot_features = pd.DataFrame()
one_hot_features['AA'] = [i for s in one_hot_encoder.categories_ for i in s]
one_hot_features['Position'] = positions
one_hot_features['Coef'] = logreg.coef_[0]

one_hot_features
```

Out[ ]:

	AA	Position	Coef
0	S	0	0.034112
1	E	1	-0.214272
2	K	1	0.331179
3	M	1	0.513529
4	N	1	-0.137291
...	...	...	...
1925	H	235	-0.265211
1926	N	235	0.165153
1927	S	235	0.370086
1928	Y	235	0.031901
1929	K	236	0.034112

1930 rows × 3 columns

We aim to keep only important positions, namely the positions that include some amino acids that has a coeficient greater than 3 or less than -3.

In [17]:
```
important_positions = one_hot_features[abs(one_hot_features['Coef']) > 3]['Position'].values
unimportant_indexs = one_hot_features[one_hot_features['Position'].apply(lambda x: x not in important_positions)].index
one_hot_unimportant = one_hot_encoded[:, unimportant_indexs]
```

We will split the original dataset for future use.

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X_all, brightnesses, test_size=0.2, random_state=5)
```

Using the descriptor files, we take all the features and only on important positions. For each of them, we add 38 features with the value of the given amino acid on that position.

In [ ]:
```
def create_des_features(df, im_pos):
    matrixs = []
    for p in im_pos:
        matrix = []
        for aa in df['ConstructedAASeq_cln'].apply(lambda x: x[p]).to_list():
            matrix.append(df_combined.loc[aa].values)
        matrixs.append(np.array(matrix))

    return np.hstack(matrixs)

# ~8s
X_train_des = create_des_features(X_train, important_positions)
X_test_des = create_des_features(X_test, important_positions)
```

Split the data of all unimportant positions for future use.

In [21]:
```
X_train_oh, X_test_oh, y_train, y_test = train_test_split(one_hot_unimportant, brightnesses, test_size=0.2, random_state=5)
```

We combine the two parts we have now: the important positions which replaced by the 38 features in the descriptors, and the unimportant positions which still use the one-hot encoding construction.

In [23]:
```
# Scale and combine the one-hot on unimportant positions and descripters on important positions
def combine_and_scale(oh, des):
    scaled_numeric_features = np.hstack((oh, des))
    scaler = StandardScaler()
    return scaler.fit_transform(scaled_numeric_features)

X_train_oh_des = combine_and_scale(X_train_oh, X_train_des)
X_test_oh_des = combine_and_scale(X_test_oh, X_test_des)
```

Train the model, which gives 0.8969 accuracy, which is slightly lower, but may be more general for unseen data (which has been proved to be true given the result on Kaggle).

In [24]:
```
# Train the model
logreg = LogisticRegression(max_iter=1000, random_state=5)
logreg.fit(X_train_oh_des, y_train.values)

# Predict
y_pred = logreg.predict(X_test_oh_des)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.8969
```

Finally we train the model for prediction.

In [ ]:
```
# Final model
X_des = create_des_features(X_all, important_positions)
X_oh_des = combine_and_scale(one_hot_encoded, X_des)

final_model = LogisticRegression(max_iter=1000, random_state=5)
final_model.fit(X_oh_des, brightnesses.values)
```

Out[ ]:
```
LogisticRegression(max_iter=1000, random_state=5)
```

Write the prediction to file.

In [36]:
```
# Make File to Submit

def write_ans(predictions):
    out = pd.read_csv('Dataset/y_sample_submission.csv')
    out['Brightness_Class'] = predictions
    out.to_csv('Dataset/predictions.csv', index=False)

X_to_predict = pd.read_csv('Dataset/test_X.csv')
predictions = final_model.predict(
    combine_and_scale(one_hot_encode(X_to_predict)[1], create_des_features(X_to_predict, important_positions))
)
write_ans(predictions)
```