```python
import math

# Calculate the weighted entropy of attribute SHOTS
enone = 0
eall = -8/14 * math.log(8/14, 2) - 6/14 * math.log(6/14, 2)
SHOTS = 14/20 * eall + 6/20 * enone
print("shots = ", SHOTS)

# Calculate the weighted entropy of attribute HAIR
elong = -6/8 * math.log(6/8, 2) - 2/8 * math.log(2/8, 2)
eshort = -2/12 * math.log(2/12, 2) - 10/12 * math.log(10/12, 2)
HAIR = 8/20 * elong + 12/20 * eshort
print("hair = ", HAIR)

# I am leaving this here even though it is not used
# Calculate the weighted entropy of attribute HEIGHT
# etiny = -3/4 * math.log(3/4, 2) - 1/4 * math.log(1/4, 2)
# emedium = -4/5 * math.log(4/5, 2) - 1/5 * math.log(1/5, 2)
# ehuge = -10/11 * math.log(10/11, 2) - 1/11 * math.log(1/11, 2)
# SIZE = 4/20 * etiny + 5/20 * emedium + 11/20 * ehuge
# print("size = ", SIZE)


'''
------------------ File Output ------------------
shots =  0.6896596952239761
hair =  0.7145247027726656

Shots has the smallest weighted entropy, so it is the best
attribute to split on.
--------------------------------------------------
'''
```

```python
import pandas as pd
import numpy as np
import math

df = pd.read_csv('HW4 - Sheet1.csv')
df = df.drop(columns=['SIZE'])
#print(df)
'''
Calculate the GINI index for the initial dataset

LaTeX formula for Gini Index
Gini(S) = 1 - \sum_{i=1}^{n} p_i^2

'''
def calculate_gini(data):
    total_samples = len(data)
    if total_samples == 0:
        return 0

    counts = data['TAKE-HOME'].value_counts()
    p0 = counts.get('no', 0) / total_samples
    p1 = counts.get('yes', 0) / total_samples
    gini = 1 - (p0**2 + p1**2)
    return gini

initial_gini = calculate_gini(df)

attributes = df.columns[:-1]  # Exclude the 'TAKE-HOME' column
best_attribute = None
best_gini_reduction = 0

'''
NOTICE -- We don't have to do every permutation of attributes because
           we are only looking at SHOTS and HAIR

This does

Î´Gini_A(S) = Gini(S) - Gini_A(S)

Gini_A(S) = Sj/S * Gini(Sj) + Sk/S * Gini(Sk)
'''
#print("att", attributes)
all_gini = {}
for attribute in attributes:
    unique_values = df[attribute].unique()
    giniA = 0

    for value in unique_values:
        subset = df[df[attribute] == value]
        #print(subset)
        #print('\n')
        weight = len(subset) / len(df)
        giniJ = calculate_gini(subset)
        giniA += weight * giniJ

    gini_reduction = initial_gini - giniA
    all_gini[attribute] = round(gini_reduction,3)
    if gini_reduction > best_gini_reduction:
        best_gini_reduction = gini_reduction
        best_attribute = attribute

print("Best Attribute to Split:", best_attribute)
print("GINI Reduction:", round(best_gini_reduction,3))
print("All GINI Reductions:", all_gini)


'''
```

```
File Output

Best Attribute to Split: HAIR
GINI Reduction: 0.163
All GINI Reductions: {'SHOTS': 0.137, 'HAIR': 0.163}

Considering the HAIR attribute has the highest score, we will split on that.
'''
```

Q1aiii

Information gain prefers splitting things into distinct categories based on attribute values while the GINI method likes to group multiple attributes together to try and minimize the probablility of misclassifying an instance.

```python
import pandas as pd
import numpy as np
import math

df = pd.read_csv('HW4 - Sheet1.csv')
#print(df)
'''
Calculate the GINI index for the initial dataset

LaTeX formula for Gini Index
Gini(S) = 1 - \sum_{i=1}^{n} p_i^2

'''
def calculate_gini(data):
    total_samples = len(data)
    if total_samples == 0:
        return 0

    counts = data['TAKE-HOME'].value_counts()
    p0 = counts.get('no', 0) / total_samples
    p1 = counts.get('yes', 0) / total_samples
    gini = 1 - (p0**2 + p1**2)
    return gini

initial_gini = calculate_gini(df)

attributes = df.columns[:-1]  # Exclude the 'TAKE-HOME' column
best_attribute = None
best_gini_reduction = 0

'''
NOTICE -- We don't have to do every permutation of attributes because
           we are only looking at SHOTS and HAIR

This does

Î´Gini_A(S) = Gini(S) - Gini_A(S)

Gini_A(S) = Sj/S * Gini(Sj) + Sk/S * Gini(Sk)
'''
#print("att", attributes)
all_gini = {}
for attribute in attributes:
    unique_values = df[attribute].unique()
    giniA = 0

    for value in unique_values:
        subset = df[df[attribute] == value]
        #print(subset)
        #print('\n')
        weight = len(subset) / len(df)
        giniJ = calculate_gini(subset)
        giniA += weight * giniJ

    gini_reduction = initial_gini - giniA
    all_gini[attribute] = round(gini_reduction,3)
    if gini_reduction > best_gini_reduction:
        best_gini_reduction = gini_reduction
        best_attribute = attribute

print("Best Attribute to Split:", best_attribute)
print("GINI Reduction:", round(best_gini_reduction,3))
print("All GINI Reductions:", all_gini)


'''
-------------------- File Output --------------------
```

```
Best Attribute to Split: SIZE
GINI Reduction: 0.247
All GINI Reductions: {'SHOTS': 0.137, 'HAIR': 0.163, 'SIZE': 0.247}

Considering the SIZE attribute has the highest score, we will split on that.
'''
```

```python
import math

'''
For several years, the University of Delaware has had difficulty predicting which high school
students
would accept Delaware’s offer of admission. Alice has developed a new model for predicting
whether a high
school senior, who has been offered admission to the University, will actually enroll. The
model is based
on state of residency, student’s high school GPA, the student’s SAT scores, the student’s
major, etc. Alice
wants to sell her model to the Admissions Office. She tells them that she has tested her model
on 8000 high
school seniors from previous years who were offered admission to the University and that her
model made a
correct prediction for 7000 of them. The Admissions Office would very much benefit from Alice’s
model if
it truly gives good predictions. With 95% confidence, what can Alice tell the Admissions Office
is the true
range of success of her model?
'''

# 95% confidence means Z = 1.96
n = 8000
p = 7000/8000
z = 1.96

# Calculate the confidence interval
ci = z * math.sqrt((p * (1-p)) / n)
lower_bound = p - ci
upper_bound = p + ci
print(f"95% confidence interval: ({round(lower_bound,3)} to {round(upper_bound,3)})")
print("Lower Bound: ", lower_bound)
print("Upper Bound: ", upper_bound)

'''
------------------ File Output ------------------
95% confidence interval: (0.868 to 0.882)
Lower Bound:  0.867752802265703
Upper Bound:  0.882247197734297
--------------------------------------------------

We can say with 95% confidence that the true accuracy
of Alice's model is between 86.8% and 88.2%.
'''
```

```python
import pandas as pd
import numpy as np
import math

df = pd.read_csv('HW4 - Sheet2.csv')

def calculate_entropy(data):
    class_counts = data['DECISION'].value_counts()
    probabilities = class_counts / len(data)
    entropy = -sum(probabilities * np.log2(probabilities))

    return entropy

def calculate_weighted_entropy(data, split_point):
    left = data[data['LENGTH'] <= split_point]
    right = data[data['LENGTH'] > split_point]

    # Calculate entropy for each subset
    entropy1 = calculate_entropy(left)
    entropy2 = calculate_entropy(right)
    print(f"({split_point}) Left side Entropy: {entropy1}")
    print(f"({split_point}) Right side Entropy: {entropy2}")


    weighted_entropy = (len(left) / len(data)) * entropy1 + (len(right) / len(data)) *
entropy2
    print(f"({split_point}) Weighted Entropy: {weighted_entropy}")

    return weighted_entropy


split_point_118 = 118
split_point_147 = 147

print("Calculations:")
weighted_entropy_118 = calculate_weighted_entropy(df, split_point_118)
print('')
weighted_entropy_147 = calculate_weighted_entropy(df, split_point_147)

print("\n")
print("Summary:")
print(f'Weighted Entropy at split point 118: {weighted_entropy_118}')
print(f'Weighted Entropy at split point 147: {weighted_entropy_147}')

better = "118" if  weighted_entropy_118 < weighted_entropy_147 else "147"
print(f'The better split point is {better}')


'''
------------------------ File Output ------------------------
Calculations:
(118) Left side Entropy: 1.584962500721156
(118) Right side Entropy: 1.1897319168207328
(118) Weighted Entropy: 1.2436269964435178

(147) Left side Entropy: 1.3787834934861753
(147) Right side Entropy: 0.9055872616982603
(147) Weighted Entropy: 1.056149699085324


Summary:
Weighted Entropy at split point 118: 1.2436269964435178
Weighted Entropy at split point 147: 1.056149699085324
The better split point is 147
------------------------------------------------------------

We choose 147 as our split point because it has a lower weighted
```

```
entropy than 118.

'''
```

```python
import pandas as pd
from scipy.stats import chi2_contingency, chi2

# Create a Pandas DataFrame with your data
data = pd.read_csv('HW4 - Sheet2.csv')
# categorize LENGTH
data = data[data['LENGTH'] <= 175]
data = data[data['LENGTH'] >= 112]
#print(data)

C = [
    [4, 0, 1],
    [0, 1, 7]
]

row_totals = []
col_totals = []
N = len(data)

# Calculate the row totals
for i in C:
    row_totals.append(sum(i))

# Calculate the column totals
for i in range(len(C[0])):
    total = 0
    for j in range(len(C)):
        total += C[j][i]
    col_totals.append(total)

E = [[0, 0, 0],
     [0, 0, 0]]

for i in range(len(row_totals)):
    for j in range(len(col_totals)):
        E[i][j] = (row_totals[i] * col_totals[j]) / N

print("Expected frequencies:")
for i in E:
    print(i)

# Calculate the chi-squared statistic
chi2 = 0
for i in range(len(C)):
    for j in range(len(C[0])):
        chi2 += ((C[i][j] - E[i][j])**2) / E[i][j]

print(f"Chi-Squared Statistic: {chi2} > 4.605")

'''
------------------------ File Output ------------------------
Expected frequencies:
[1.5384615384615385, 0.38461538461538464, 3.076923076923077]
[2.4615384615384617, 0.6153846153846154, 4.923076923076923]

Chi-Squared Statistic: 9.303125 > 4.605
------------------------------------------------------------

We reject the null hypothesis and we do NOT merge

'''
```

```python
import pandas as pd
from scipy.stats import chi2_contingency, chi2

# Create a Pandas DataFrame with your data
df = pd.read_csv('HW4 - Sheet2.csv')
# categorize LENGTH
df = df[df['LENGTH'] >= 147]

C = [
    [0, 1, 7],
    [1, 1, 5]
]

row_totals = []
col_totals = []
N = len(df)

# Calculate the row totals
for i in C:
    row_totals.append(sum(i))

# Calculate the column totals
for i in range(len(C[0])):
    total = 0
    for j in range(len(C)):
        total += C[j][i]
    col_totals.append(total)

E = [[0, 0, 0],
     [0, 0, 0]]

for i in range(len(row_totals)):
    for j in range(len(col_totals)):
        E[i][j] = (row_totals[i] * col_totals[j]) / N
print("Expected frequencies:")
for i in E:
    print(i)

# Calculate the chi-squared statistic
chi2 = 0
for i in range(len(C)):
    for j in range(len(C[0])):
        chi2 += ((C[i][j] - E[i][j])**2) / E[i][j]

print(f"Chi-Squared Statistic: {chi2} < 4.605")
'''
------------------------ File Output ------------------------
Expected frequencies:
[0.5333333333333333, 1.0666666666666667, 6.4]
[0.4666666666666667, 0.9333333333333333, 5.6]

Chi-Squared Statistic: 1.2723214285714284 < 4.605
------------------------------------------------------------

We do NOT reject the null hypothesis and we DO merge

'''
```