Evaluation and Validation

```
import random

# Sample data with 77 entries for each group
age = [47, 36, 43, 25, 52, 23, 38, 22, 57, 36,
       32, 51, 21, 26, 21, 20, 51, 31, 60, 53,
       42, 58, 33, 64, 53, 38, 31, 43, 38, 45,
       43, 55, 41, 51, 48, 42, 45, 41, 50, 61,
       43, 58, 47, 37, 53, 48, 35, 44, 37, 40,
       32, 25, 32, 52, 19, 34, 60, 44, 52, 24,
       48, 44, 66, 64, 20, 42, 32, 39, 41, 34,
       45, 29, 27, 30, 28, 31, 33]

iq = [81, 104, 108, 106, 102, 104, 100, 98, 112, 71,
      102, 93, 87, 100, 100, 110, 110, 101, 102, 108,
      108, 98, 104, 114, 106, 87, 106, 109, 108, 85,
      104, 102, 112, 100, 106, 100, 95, 89, 108, 83,
      106, 104, 114, 98, 93, 93, 100, 110, 104, 85,
      102, 102, 106, 106, 98, 116, 97, 104, 89, 110,
      102, 93, 98, 97, 105, 110, 95, 99, 96, 103,
      100, 104, 101, 105, 99, 107, None]

group = ['HC', 'AVH-', 'AVH+', 'HC', 'AVH-', 'AVH+',
         'AVH-', 'HC', 'AVH+', 'HC', 'AVH-', 'AVH-',
         'HC', 'AVH+', 'AVH-', 'AVH-', 'AVH+', 'HC',
         'AVH-', 'AVH-', 'HC', 'AVH-', 'AVH+', 'HC',
         'AVH-', 'AVH+', 'HC', 'AVH-', 'HC', 'AVH+',
         'AVH-', 'AVH+', 'HC', 'AVH-', 'AVH+', 'AVH+',
         'HC', 'AVH-', 'HC', 'AVH+', 'HC', 'AVH-',
         'AVH+', 'AVH+', 'HC', 'AVH-', 'HC', 'AVH+',
         'AVH+', 'AVH+', 'HC', 'HC', 'AVH+', 'AVH+',
         'AVH-', 'HC', 'AVH-', 'HC', 'AVH+', 'AVH-',
         'HC', 'AVH+', 'AVH+', 'AVH+', 'AVH-', 'AVH-',
         'AVH-', 'HC', 'AVH+', 'AVH-', 'HC', 'HC', 'AVH-', 'AVH+', 'HC', 'AVH-', 'AVH+']

gender = ['male', 'female', 'female', 'male', 'female', 'female',
          'male', 'male', 'female', 'male', 'female', 'female',
          'male', 'female', 'female', 'female', 'male', 'female',
          'female', 'male', 'male', 'female', 'female', 'male',
```

```
        'male', 'female', 'male', 'female', 'male', 'female',
        'female', 'female', 'male', 'female', 'female', 'female',
        'male', 'male', 'male', 'female', 'male', 'female',
        'female', 'female', 'male', 'female', 'male', 'female',
        'female', 'female', 'male', 'female', 'female', 'male',
        'female', 'female', 'male', 'male', 'male', 'male',
        'female', 'female', 'female', 'male', 'female', 'female',
        'male', 'female', 'male', 'female', 'female', 'female',
        'male', 'male', 'male', 'male', 'female']

# Create a list of dictionaries to represent the data
df = []
for i in range(len(age)):
    df.append({
        'age': age[i],
        'iq': iq[i],
        'group': group[i],
        'gender': gender[i]
    })

# Fill missing IQ values with the mean
iq_values = [value for value in iq if value is not None]
mean_iq = sum(iq_values) / len(iq_values)
for entry in df:
    if entry['iq'] is None:
        entry['iq'] = mean_iq

# Prepare data
X = [[entry['age'], entry['iq']] for entry in df]
y = [entry['group'] for entry in df]  # Target variable

# Split data into training and test sets
split_ratio = 0.3  # 30% test set
split_index = int(len(X) * (1 - split_ratio))
X_train = X[:split_index]
X_test = X[split_index:]
y_train = y[:split_index]
y_test = y[split_index:]

# Define a simple Random Forest Classifier
```

```
class RandomForestClassifier:
    def __init__(self, random_state=None):
        self.random_state = random_state
        self.trees = []

    def fit(self, X, y):
        self.trees.append((X, y))  # In practice, would create multiple decision trees

    def predict(self, X):
        predictions = []
        for x in X:
            votes = {}
            for tree_data in self.trees:
                for idx, point in enumerate(tree_data[0]):
                    if point == x:
                        label = tree_data[1][idx]
                        votes[label] = votes.get(label, 0) + 1
            predicted_label = max(votes, key=votes.get) if votes else None
            predictions.append(predicted_label)
        return predictions

# Create and train the classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate model performance
accuracy = sum(1 for true, pred in zip(y_test, y_pred) if true == pred) / len(y_test)
confusion_matrix = {}
for true, pred in zip(y_test, y_pred):
    if true not in confusion_matrix:
        confusion_matrix[true] = {}
    if pred not in confusion_matrix[true]:
        confusion_matrix[true][pred] = 0
    confusion_matrix[true][pred] += 1

# Output the results
print(f'Accuracy (Natural Distribution): {accuracy:.2f}')
```

```
print(f'Confusion Matrix (Natural Distribution):\n{confusion_matrix}')
```