Bias Detection

```
import random

# Sample data with 77 entries
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]  # 'None' is a placeholder for missing data

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']

# Ensure all lists have the same length
if not (len(age) == len(iq) == len(group) == len(gender)):
    raise ValueError("All lists must have the same length.")

# Check if IQ has None and handle missing IQ values by replacing them with the mean
iq_values = [value for value in iq if value is not None]  # Exclude None values
iq_mean = sum(iq_values) / len(iq_values)

# Replace None values in the IQ list
iq = [value if value is not None else iq_mean for value in iq]

# Creating a DataFrame-like structure
df = []
for i in range(len(age)):
    df.append({
        'age': age[i],
        'iq': iq[i],
        'group': group[i],
        'gender': gender[i]
    })

# Creating target variable
for row in df:
    row['target'] = 1 if row['group'] == 'AVH+' else 0  # Binary classification

# Shuffle the data
random.seed(42)
random.shuffle(df)

# Splitting the data into training and test sets
split_index = int(0.7 * len(df))  # 70% for training
```

```
train_data = df[:split_index]
test_data = df[split_index:]

# Manual implementation of a basic decision rule
def predict(row):
    # Simple rule: if age > 40 and iq > 100, predict AVH+
    return 1 if (row['age'] > 40 and row['iq'] > 100) else 0

# Making predictions
y_pred = []
y_true = [row['target'] for row in test_data]

for row in test_data:
    y_pred.append(predict(row))

# Analyzing performance by gender
df_test = test_data.copy()
for i in range(len(df_test)):
    df_test[i]['actual'] = y_true[i]
    df_test[i]['predicted'] = y_pred[i]

# Adding gender to the test data
for i in range(len(df_test)):
    df_test[i]['gender'] = df[i + split_index]['gender']

# Analyze performance by gender
gender_performance = {}
for row in df_test:
    gender = row['gender']
    if gender not in gender_performance:
        gender_performance[gender] = {'TP': 0, 'TN': 0, 'FP': 0, 'FN': 0}

    if row['actual'] == 1 and row['predicted'] == 1:
        gender_performance[gender]['TP'] += 1  # True Positive
    elif row['actual'] == 0 and row['predicted'] == 0:
        gender_performance[gender]['TN'] += 1  # True Negative
    elif row['actual'] == 0 and row['predicted'] == 1:
        gender_performance[gender]['FP'] += 1  # False Positive
    elif row['actual'] == 1 and row['predicted'] == 0:
        gender_performance[gender]['FN'] += 1  # False Negative
```

```
# Print performance metrics by gender
for gender, metrics in gender_performance.items():
    print(f"Performance for {gender}:")
    print(f"TP: {metrics['TP']}, TN: {metrics['TN']}, FP: {metrics['FP']}, FN: {metrics['FN']}")

# Manual upsampling of the minority class (female) to match the size of the majority class (male)
df_majority = [row for row in df if row['gender'] == 'male']
df_minority = [row for row in df if row['gender'] == 'female']

# Upsample the minority class (female)
df_minority_upsampled = []
while len(df_minority_upsampled) < len(df_majority):
    df_minority_upsampled.append(random.choice(df_minority))

# Combine the upsampled minority with the majority class
df_balanced = df_majority + df_minority_upsampled

# Verify new gender distribution
gender_distribution = {'male': 0, 'female': 0}
for row in df_balanced:
    gender_distribution[row['gender']] += 1

print(f"New Gender Distribution:\nMale: {gender_distribution['male']}, Female: {gender_distribution['female']}")
```