Documentation and Reporting: Hypothesis not accepted

```
import random

# Sample data
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 balanced data
X_balanced = [[entry['age'], entry['iq']] for entry in df]
y_balanced = [entry['group'] for entry in df]

# Split balanced data into training and test sets
split_ratio = 0.3  # 30% test set
split_index = int(len(X_balanced) * (1 - split_ratio))
X_train_balanced = X_balanced[:split_index]
X_test_balanced = X_balanced[split_index:]
y_train_balanced = y_balanced[:split_index]
y_test_balanced = y_balanced[split_index:]
```

```
# Improved RandomForestClassifier with randomness
class RandomForestClassifier:
    def __init__(self, n_trees=10, random_state=None):
        self.n_trees = n_trees
        self.random_state = random_state
        if random_state is not None:
            random.seed(random_state)
        self.trees = []

    def fit(self, X, y):
        for _ in range(self.n_trees):
            # Randomly sample from the data with replacement (bootstrap sample)
            indices = random.choices(range(len(X)), k=len(X))
            X_sample = [X[i] for i in indices]
            y_sample = [y[i] for i in indices]
            # Randomly pick features to use in this tree (feature bagging)
            feature_indices = random.sample(range(len(X[0])), k=random.randint(1,
len(X[0])))
            self.trees.append((X_sample, y_sample, feature_indices))

    def predict(self, X):
        predictions = []
        for x in X:
            votes = {}
            for X_sample, y_sample, feature_indices in self.trees:
                for idx, point in enumerate(X_sample):
                    # Compare the selected features for prediction
                    if all(x[f] == point[f] for f in feature_indices):
                        label = y_sample[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_balanced = RandomForestClassifier(random_state=42)
clf_balanced.fit(X_train_balanced, y_train_balanced)

# Make predictions on balanced data
y_pred_balanced = clf_balanced.predict(X_test_balanced)
```

```
# Calculate accuracy by gender after balancing
gender_perf_after = {}
for g in set(gender[split_index:]):
    group_indices = [i for i in range(len(X_test_balanced)) if gender[split_index:][i] == g]
    correct_predictions = sum(1 for i in group_indices if y_test_balanced[i] == y_pred_balanced[i])
    gender_perf_after[g] = correct_predictions / len(group_indices) if group_indices else 0

# Visualization using ASCII output
print("\nAfter Balancing:")
for g, acc in gender_perf_after.items():
    print(f"{g}: {'*' * int(acc * 50)} ({acc:.2f})")
```