In [10]:

```
import random
import math

import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.tree import DecisionTreeClassifier


import seaborn as sns
from scipy.stats import pearsonr
from itertools import cycle
```

Step 1:

Which dataset did you select?

We chose the Student Performance Data Set (Kaggle Description: This data approach student achievement in secondary education of two Portuguese schools) from

https://www.kaggle.com/larsen0966/student-performance-data-set?select=student-por.csv (https://www.kaggle.com/larsen0966/student-performance-data-set?select=student-por.csv)

Which regulated domain does your dataset belong to?

The dataset belongs to the Education domain.

How many observations are in the dataset?

There are 649 observations.

How many variables in the dataset?

There are 33 variables in the dataset.

Which variables did you select as your dependent variables?

We selected the grades related to Portuguese: G1 - first-period grade (numeric: from 0 to 20) G2 - second-period grade (numeric: from 0 to 20) G3 - final grade (numeric: from 0 to 20, output target)

How many and which variables in the dataset are associated with a legally recognized protected class?

There are two variables associated with legally recognized protected classes: age and sex.

Which legal precedence/law (as discussed in the lectures) does each protected class fall under?

Age: Age Discrimination in Employment Act of 1967 Sex: Equal Pay Act of 1963; Civil Rights Act of 1964, 1991

In [4]:

```python

def STEP1():
    df = pd.read_csv('dataset/student-por.csv', delimiter=',')
    print(df.head(5))
    print("Number of records port: ", len(df))
    r, c = df.shape
    print("rows port: ", r)
    print("columns port: ", c)
    print()
    for col in df.columns:
        print(col)
STEP1()
```

```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher
...
1     GP   F   17       U     GT3       T     1     1  at_home     other
...
2     GP   F   15       U     LE3       T     1     1  at_home     other
...
3     GP   F   15       U     GT3       T     4     2   health  services
...
4     GP   F   16       U     GT3       T     3     3    other     other
...

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        4   0  11  11
1      5        3      3     1     1      3        2   9  11  11
2      4        3      2     2     3      3        6  12  13  12
3      3        2      2     1     1      5        0  14  14  14
4      4        3      2     1     2      5        0  11  13  13

[5 rows x 33 columns]
Number of records port:  649
rows port:  649
columns port:  33

school
sex
age
address
famsize
Pstatus
Medu
Fedu
Mjob
Fjob
reason
guardian
traveltime
studytime
failures
schoolsup
famsup
paid
activities
```

```
nursery
higher
internet
romantic
famrel
freetime
goout
Dalc
Walc
health
absences
G1
G2
G3
```

Step 2:

2.1:

Protected Class	Variable	Raw Values	Subsets
Gender	sex	F, M	Female, Male
Age	age	15, 16, 17, 18, 19, 20, 21, 22	[15-17] [18-22]

2.2:

Min Grade = 0
Avg Grade = 11
Max Grade = 19

bins = [0, 11, 19]
labels = [0, 1]

df['G1'] = pd.cut(df.G1, bins=bins, labels=labels, include_lowest=True)
df['G2'] = pd.cut(df.G2, bins=bins, labels=labels, include_lowest=True)
df['G3'] = pd.cut(df.G3, bins=bins, labels=labels, include_lowest=True)

Dependent Variables	Raw Categories	Discrete Categories
G1	0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	[0-11] represented as 0 [12-19] represented as 1
G2	0, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	[0-11] represented as 0 [12-19] represented as 1
G3	0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	[0-11] represented as 0 [12-19] represented as 1

Age	Bin	G1	G2	G3
15-17	0	227	225	205
15-17	1	241	243	263
18-22	0	116	106	96
18-22	1	65	75	85

Age	Bin	G1	G2	G3
15-17	0	227	225	205
15-17	1	241	243	263
18-22	0	116	106	96
18-22	1	65	75	85

2.3:

Age Group	Gender	Total
15-17	Female	275
15-17	Male	193
18-22	Female	108
18-22	Male	73

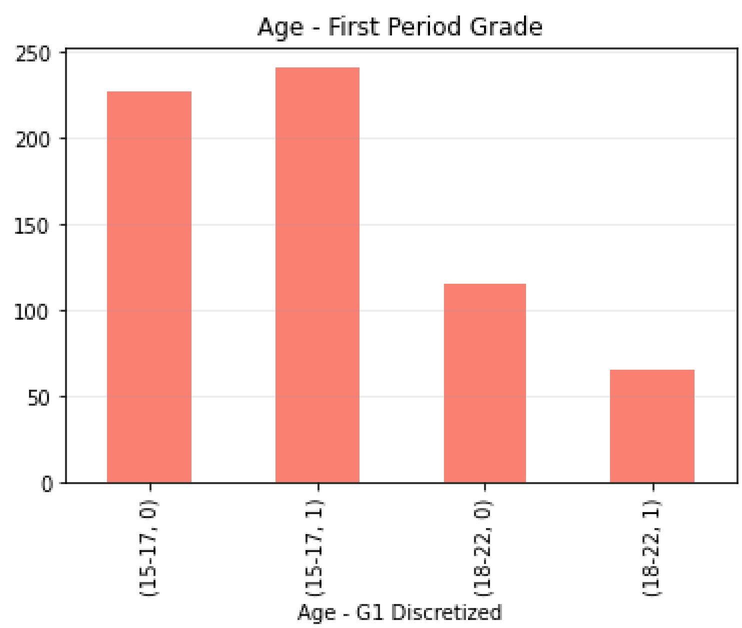
In [5]:

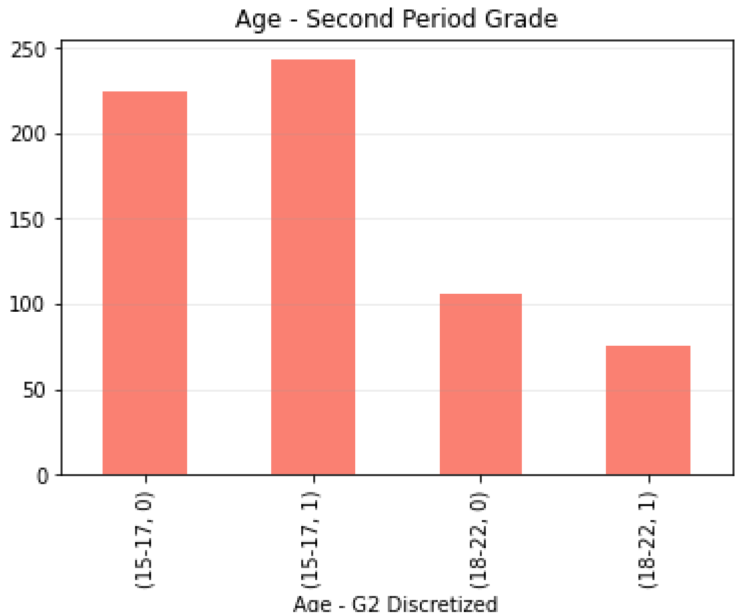
```

def STEP2():
    df = pd.read_csv('dataset/student-por.csv', delimiter=',')

    df.loc[df['sex'] == "M", ['sex']] = 'Male'
    df.loc[df['sex'] == "F", ['sex']] = 'Female'

    df.loc[df['age'].between(15, 17, inclusive=True), ['age_group']] =
    df.loc[df['age'].between(18, 22, inclusive=True), ['age_group']] =

    age_group_sex_freq = df.groupby(['age_group', 'sex']).size()
    age_group_sex_freq.to_csv('out/age_group_sex_freq.csv')

    age_unique = []
    g1_unique = []
    g2_unique = []
    g3_unique = []

    for x in df['age']:
        if x not in age_unique:
            age_unique.append(x)

    for x in df['G1']:
        if x not in g1_unique:
            g1_unique.append(x)

    for x in df['G2']:
        if x not in g2_unique:
            g2_unique.append(x)

    for x in df['G3']:
        if x not in g3_unique:
            g3_unique.append(x)

    age_unique.sort()
    g1_unique.sort()
    g2_unique.sort()
    g3_unique.sort()

    print("Age: ", age_unique)
    print("G1: ", g1_unique)
    print("G2: ", g2_unique)
    print("G3: ", g3_unique)

    print("G1 Max: ", df['G1'].max())
    print("G1 Min: ", df['G1'].min())
    print("G1 Avg: ", df['G1'].mean())

    print("G2 Max: ", df['G2'].max())
    print("G2 Min: ", df['G2'].min())
    print("G2 Avg: ", df['G2'].mean())

    print("G3 Max: ", df['G3'].max())
    print("G3 Min: ", df['G3'].min())
    print("G3 Avg: ", df['G3'].mean())

```

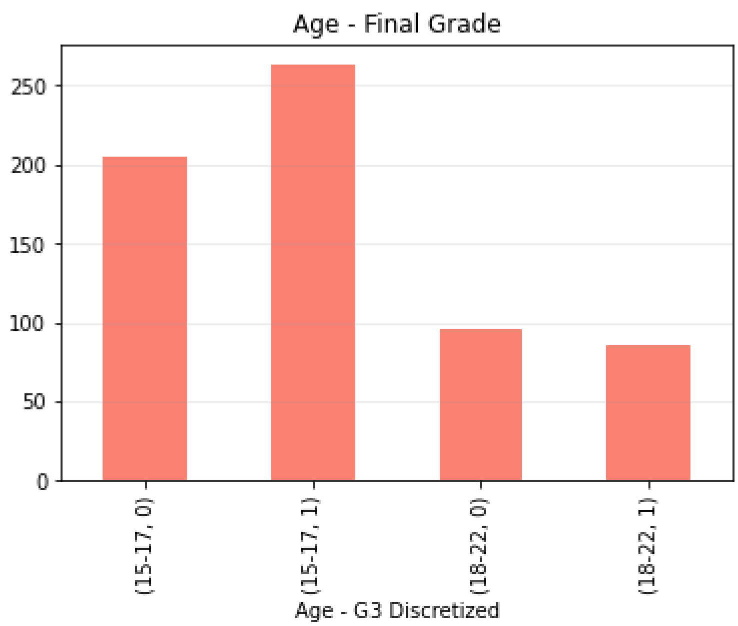
```
    bins = [0, 11, 19]
    labels = [0, 1]

    df['G1'] = pd.cut(df.G1, bins=bins, labels=labels, include_lowest=
    df['G2'] = pd.cut(df.G2, bins=bins, labels=labels, include_lowest=
    df['G3'] = pd.cut(df.G3, bins=bins, labels=labels, include_lowest=

    age_g1_freq = df.groupby(['age_group', 'G1']).size()
    age_g1_freq.to_csv('out/age_g1_freq.csv')

    age_g2_freq = df.groupby(['age_group', 'G2']).size()
    age_g2_freq.to_csv('out/age_g2_freq.csv')

    age_g3_freq = df.groupby(['age_group', 'G3']).size()
    age_g3_freq.to_csv('out/age_g3_freq.csv')

    sex_g1_freq = df.groupby(['sex', 'G1']).size()
    sex_g1_freq.to_csv('out/sex_g1_freq.csv')

    sex_g2_freq = df.groupby(['sex', 'G2']).size()
    sex_g2_freq.to_csv('out/sex_g2_freq.csv')

    sex_g3_freq = df.groupby(['sex', 'G3']).size()
    sex_g3_freq.to_csv('out/sex_g3_freq.csv')

    age_g1_freq.plot.bar(stacked=False, color="salmon")
    plt.title('Age - First Period Grade')
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.xlabel('Age - G1 Discretized')
    plt.savefig('out/age_g1_freq.png', bbox_inches='tight')
    plt.show()

    age_g2_freq.plot.bar(stacked=False, color="salmon")
    plt.title('Age - Second Period Grade')
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.xlabel('Age - G2 Discretized')
    plt.savefig('out/age_g2_freq.png', bbox_inches='tight')
    plt.show()

    age_g3_freq.plot.bar(stacked=False, color="salmon")
    plt.title('Age - Final Grade')
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.xlabel('Age - G3 Discretized')
    plt.savefig('out/age_g3_freq.png', bbox_inches='tight')
    plt.show()

    sex_g1_freq.plot.bar(stacked=False, color="lightblue")
    plt.title('Sex - First Period Grade')
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.xlabel('Sex - G1 Discretized')
    plt.savefig('out/sex_g1_freq.png', bbox_inches='tight')
    plt.show()

    sex_g2_freq.plot.bar(stacked=False, color="lightblue")
    plt.title('Sex - Second Period Grade')
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.xlabel('Sex - G2 Discretized')
```

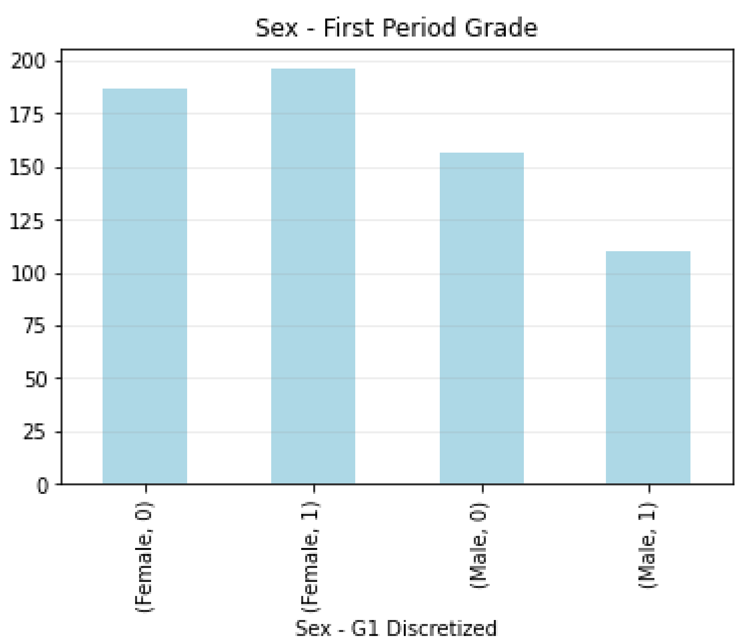
```
    plt.savefig('out/sex_g2_freq.png', bbox_inches='tight')
    plt.show()

    sex_g3_freq.plot.bar(stacked=False, color="lightblue")
    plt.title('Sex - Final Grade')
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.xlabel('Sex - G3 Discretized')
    plt.savefig('out/sex_g3_freq.png', bbox_inches='tight')
    plt.show()

STEP2()
```

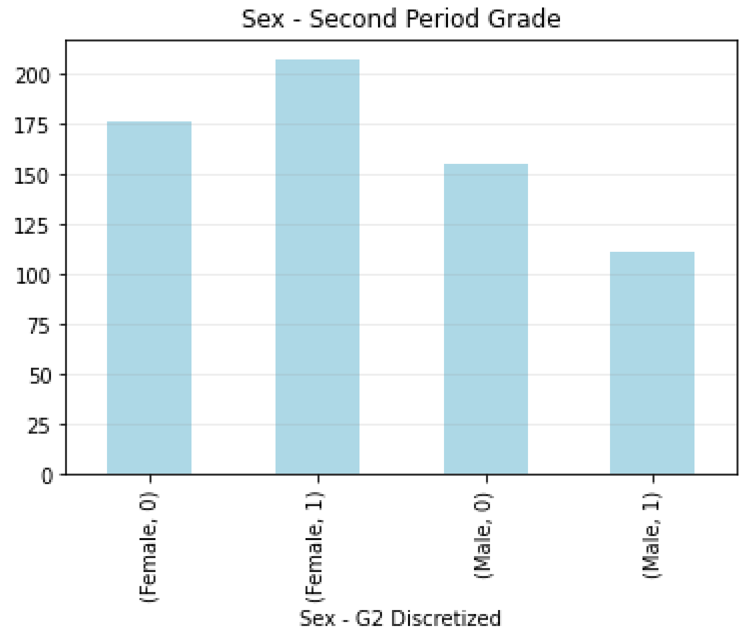
```
Age:  [15, 16, 17, 18, 19, 20, 21, 22]
G1:  [0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
G2:  [0, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
G3:  [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
G1 Max:  19
G1 Min:  0
G1 Avg:  11.399075500770415
G2 Max:  19
G2 Min:  0
G2 Avg:  11.570107858243452
G3 Max:  19
G3 Min:  0
G3 Avg:  11.906009244992296
```

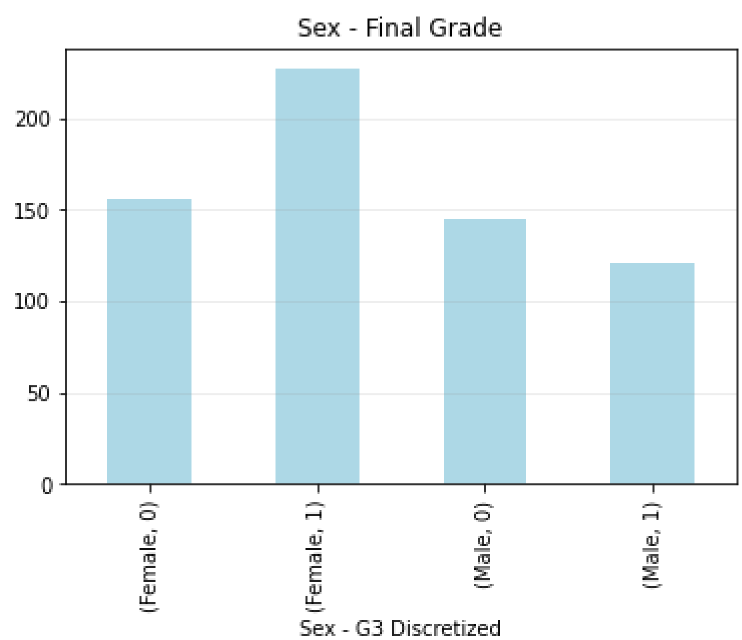



Age - Second Period Grade
0
50
100
150
200
250
(15-17, 0)
(15-17, 1)
(18-22, 0)
(18-22, 1)
Age - G2 Discretized


Age - Final Grade
0
50
100
150
200
250
(15-17, 0)
(15-17, 1)
(18-22, 0)
(18-22, 1)
Age - G3 Discretized


Sex - First Period Grade
0
25
50
75
100
125
150
175
200
(Female, 0)
(Female, 1)
(Male, 0)
(Male, 1)
Sex - G1 Discretized





Step 3:

Protected Class	Variable	Privileged Group	Unprivileged Group
Gender	sex	Female	Male

Protected Class	Variable	Privileged Group	Unprivileged Group
Age	age	[15-17]	[18-22]

The fairness metrics selected are: 1. Statistical Parity Difference 2. Disparate Impact The threshold chosen to calculate the fairness metric is the grade value of 12 (maximum grade value is 19).

Fairness Metrics Calculated:

Dependent Variable	Protected Class Variable	Statistical Parity Difference	Disparate Impact
G1	sex	-0.0982155	0.8080788
G1	age	-0.1558412	0.6973705
G2	sex	-0.1231767	0.7720932
G2	age	-0.1048661	0.7980356
G3	sex	-0.1378020	0.7674969
G3	age	-0.0923525	0.8356616

Bias Mitigation Strategy selected: **Reweighting.**
The weights were calculated for each Dependent Variable and Protected Class Variable Combination, and used these weights to calculate the new fairness metrics. (Different weights were used for each row, actual weight values can be found in the out folder after running the application).

Fairness Metrics Calculated after Reweighting:

Dependent Variable	Protected Class Variable	Statistical Parity Difference	Disparate Impact
G1	sex	0	1.0000000000000002
G1	age	0	1.0000000000000002
G2	sex	0	1
G2	age	0	1
G3	sex	0	1
G3	age	0	1.0000000000000002

In [13]:

```python

def STEP3():
    df = pd.read_csv('dataset/student-por.csv', delimiter=',')

    df.loc[df['sex'] == "M", ['sex']] = 'Male'
    df.loc[df['sex'] == "F", ['sex']] = 'Female'

    df.loc[df['age'].between(15, 17, inclusive=True), ['age_group']] =
    df.loc[df['age'].between(18, 22, inclusive=True), ['age_group']] =

    bins = [0, 11, 19]
    labels = [0, 1]

    df['G1'] = pd.cut(df.G1, bins=bins, labels=labels, include_lowest=
    df['G2'] = pd.cut(df.G2, bins=bins, labels=labels, include_lowest=
    df['G3'] = pd.cut(df.G3, bins=bins, labels=labels, include_lowest=

    age_g1_freq = df.groupby(['age_group', 'G1']).size()
    age_g2_freq = df.groupby(['age_group', 'G2']).size()
    age_g3_freq = df.groupby(['age_group', 'G3']).size()
    sex_g1_freq = df.groupby(['sex', 'G1']).size()
    sex_g2_freq = df.groupby(['sex', 'G2']).size()
    sex_g3_freq = df.groupby(['sex', 'G3']).size()

    up_sex = 'Male'
    up_age = '18-22'
    p_sex = 'Female'
    p_age = '15-17'

    # statistical parity difference G1 and age
    spd_g1_age = (age_g1_freq[up_age][1] / (age_g1_freq[up_age][1] + a
            age_g1_freq[p_age][1] / (age_g1_freq[p_age][1] + age_g1_fr

    # statistical parity difference G2 and age
    spd_g2_age = (age_g2_freq[up_age][1] / (age_g2_freq[up_age][1] + a
            age_g2_freq[p_age][1] / (age_g2_freq[p_age][1] + age_g2_fr

    # statistical parity difference G3 and age
    spd_g3_age = (age_g3_freq[up_age][1] / (age_g3_freq[up_age][1] + a
            age_g3_freq[p_age][1] / (age_g3_freq[p_age][1] + age_g3_fr

    # statistical parity difference G1 and sex
    spd_g1_sex = (sex_g1_freq[up_sex][1] / (sex_g1_freq[up_sex][1] + s
            sex_g1_freq[p_sex][1] / (sex_g1_freq[p_sex][1] + sex_g1_fr

    # statistical parity difference G1 and sex
    spd_g2_sex = (sex_g2_freq[up_sex][1] / (sex_g2_freq[up_sex][1] + s
            sex_g2_freq[p_sex][1] / (sex_g2_freq[p_sex][1] + sex_g2_fr

    # statistical parity difference G1 and sex
    spd_g3_sex = (sex_g3_freq[up_sex][1] / (sex_g3_freq[up_sex][1] + s
            sex_g3_freq[p_sex][1] / (sex_g3_freq[p_sex][1] + sex_g3_fr

    spd_data = [['G1', 'Age', spd_g1_age], ['G2', 'Age', spd_g2_age],
                ['G1', 'Sex', spd_g1_sex], ['G2', 'Sex', spd_g2_sex],

```

```
    pd.DataFrame(spd_data,
                 columns=['Dependent Variable', 'Protected Class Varial
        'out/spd.csv', index=False)

    # disparate impact G1 and age
    di_g1_age = (age_g1_freq[up_age][1] / (age_g1_freq[up_age][1] + age
            age_g1_freq[p_age][1] / (age_g1_freq[p_age][1] + age_g1_fre

    # disparate impact G2 and age
    di_g2_age = (age_g2_freq[up_age][1] / (age_g2_freq[up_age][1] + age
            age_g2_freq[p_age][1] / (age_g2_freq[p_age][1] + age_g2_fre

    # disparate impact G3 and age
    di_g3_age = (age_g3_freq[up_age][1] / (age_g3_freq[up_age][1] + age
            age_g3_freq[p_age][1] / (age_g3_freq[p_age][1] + age_g3_fre

    # disparate impact G1 and sex
    di_g1_sex = (sex_g1_freq[up_sex][1] / (sex_g1_freq[up_sex][1] + se
            sex_g1_freq[p_sex][1] / (sex_g1_freq[p_sex][1] + sex_g1_fre

    # disparate impact G2 and sex
    di_g2_sex = (sex_g2_freq[up_sex][1] / (sex_g2_freq[up_sex][1] + se
            sex_g2_freq[p_sex][1] / (sex_g2_freq[p_sex][1] + sex_g2_fre

    # disparate impact G3 and sex
    di_g3_sex = (sex_g3_freq[up_sex][1] / (sex_g3_freq[up_sex][1] + se
            sex_g3_freq[p_sex][1] / (sex_g3_freq[p_sex][1] + sex_g3_fre

    di_data = [['G1', 'Age', di_g1_age], ['G2', 'Age', di_g2_age], ['G
               ['G1', 'Sex', di_g1_sex], ['G2', 'Sex', di_g2_sex], ['G

    pd.DataFrame(di_data,
                 columns=['Dependent Variable', 'Protected Class Varial
        'out/di.csv', index=False)

    # re-weighting

    # weights for G1 and age
    w_pp_age_g1 = (age_g1_freq[p_age].sum() * (age_g1_freq[p_age][1] +
            age_g1_freq.values.sum() * age_g1_freq[p_age][1])
    w_pu_age_g1 = (age_g1_freq[up_age].sum() * (age_g1_freq[p_age][1] -
            age_g1_freq.values.sum() * age_g1_freq[up_age][1])
    w_np_age_g1 = (age_g1_freq[p_age].sum() * (age_g1_freq[p_age][0] +
            age_g1_freq.values.sum() * age_g1_freq[p_age][0])
    w_nu_age_g1 = (age_g1_freq[up_age].sum() * (age_g1_freq[p_age][0] -
            age_g1_freq.values.sum() * age_g1_freq[up_age][0])

    weights = [[w_pp_age_g1, w_pu_age_g1, w_np_age_g1, w_nu_age_g1]]
    pd.DataFrame(weights,
                 columns=['Positive outcome - Privileged Group', 'Posi
                          'Negative outcome - Privileged Group', 'Nega
        'out/weights_g1_age.csv', index=False)

    # applying the weights to calculate spd and di
    w_spd_g1_age = (w_pu_age_g1 * age_g1_freq[up_age][1] / (
            w_pu_age_g1 * age_g1_freq[up_age][1] + w_nu_age_g1 * age_g
                               w_pp_age_g1 * age_g1_freq[p_age][1] / (
```

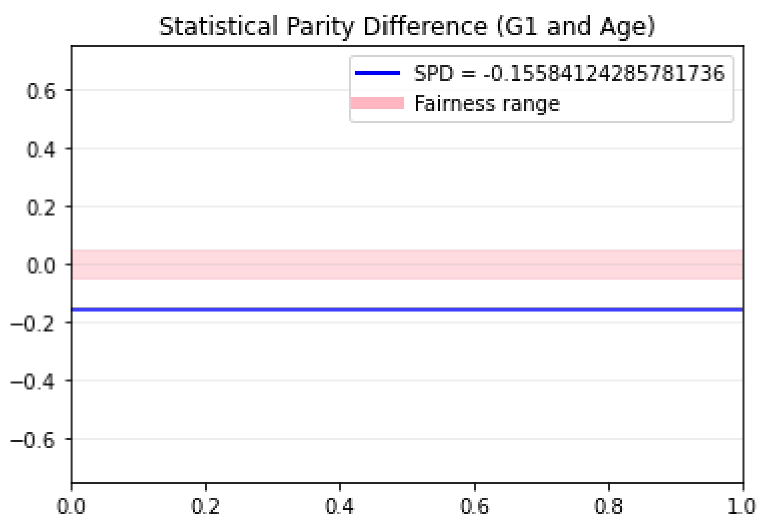
```
                            w_pp_age_g1 * age_g1_freq[p_age][1] + w_np_a
    w_di_g1_age = (w_pu_age_g1 * age_g1_freq[up_age][1] / (
            w_pu_age_g1 * age_g1_freq[up_age][1] + w_nu_age_g1 * age_g
                           w_pp_age_g1 * age_g1_freq[p_age][1] / (
                           w_pp_age_g1 * age_g1_freq[p_age][1] + w_np_ag

    # weights for G2 and age
    w_pp_age_g2 = (age_g2_freq[p_age].sum() * (age_g2_freq[p_age][1] +
            age_g2_freq.values.sum() * age_g2_freq[p_age][1])
    w_pu_age_g2 = (age_g2_freq[up_age].sum() * (age_g2_freq[p_age][1] +
            age_g2_freq.values.sum() * age_g2_freq[up_age][1])
    w_np_age_g2 = (age_g2_freq[p_age].sum() * (age_g2_freq[p_age][0] +
            age_g2_freq.values.sum() * age_g2_freq[p_age][0])
    w_nu_age_g2 = (age_g2_freq[up_age].sum() * (age_g2_freq[p_age][0] +
            age_g2_freq.values.sum() * age_g2_freq[up_age][0])

    weights = [[w_pp_age_g2, w_pu_age_g2, w_np_age_g2, w_nu_age_g2]]
    pd.DataFrame(weights,
                 columns=['Positive outcome - Privileged Group', 'Posi
                          'Negative outcome - Privileged Group', 'Nega
        'out/weights_g2_age.csv', index=False)

    # applying the weights to calculate spd and di
    w_spd_g2_age = (w_pu_age_g2 * age_g2_freq[up_age][1] / (
            w_pu_age_g2 * age_g2_freq[up_age][1] + w_nu_age_g2 * age_g
                            w_pp_age_g2 * age_g2_freq[p_age][1] / (
                            w_pp_age_g2 * age_g2_freq[p_age][1] + w_np_a
    w_di_g2_age = (w_pu_age_g2 * age_g2_freq[up_age][1] / (
            w_pu_age_g2 * age_g2_freq[up_age][1] + w_nu_age_g2 * age_g
                           w_pp_age_g2 * age_g2_freq[p_age][1] / (
                           w_pp_age_g2 * age_g2_freq[p_age][1] + w_np_ag

    # weights for G3 # and age
    w_pp_age_g3 = (age_g3_freq[p_age].sum() * (age_g3_freq[p_age][1] +
            age_g3_freq.values.sum() * age_g3_freq[p_age][1])
    w_pu_age_g3 = (age_g3_freq[up_age].sum() * (age_g3_freq[p_age][1] +
            age_g3_freq.values.sum() * age_g3_freq[up_age][1])
    w_np_age_g3 = (age_g3_freq[p_age].sum() * (age_g3_freq[p_age][0] +
            age_g3_freq.values.sum() * age_g3_freq[p_age][0])
    w_nu_age_g3 = (age_g3_freq[up_age].sum() * (age_g3_freq[p_age][0] +
            age_g3_freq.values.sum() * age_g3_freq[up_age][0])

    weights = [[w_pp_age_g3, w_pu_age_g3, w_np_age_g3, w_nu_age_g3]]
    pd.DataFrame(weights,
                 columns=['Positive outcome - Privileged Group', 'Posi
                          'Negative outcome - Privileged Group', 'Nega
        'out/weights_g3_age.csv', index=False)

    # applying the weights to calculate spd and di
    w_spd_g3_age = (w_pu_age_g3 * age_g3_freq[up_age][1] / (
            w_pu_age_g3 * age_g3_freq[up_age][1] + w_nu_age_g3 * age_g
                            w_pp_age_g3 * age_g3_freq[p_age][1] / (
                            w_pp_age_g3 * age_g3_freq[p_age][1] + w_np_a
    w_di_g3_age = (w_pu_age_g3 * age_g3_freq[up_age][1] / (
            w_pu_age_g3 * age_g3_freq[up_age][1] + w_nu_age_g3 * age_g
                           w_pp_age_g3 * age_g3_freq[p_age][1] / (
                           w_pp_age_g3 * age_g3_freq[p_age][1] + w_np_ag
```

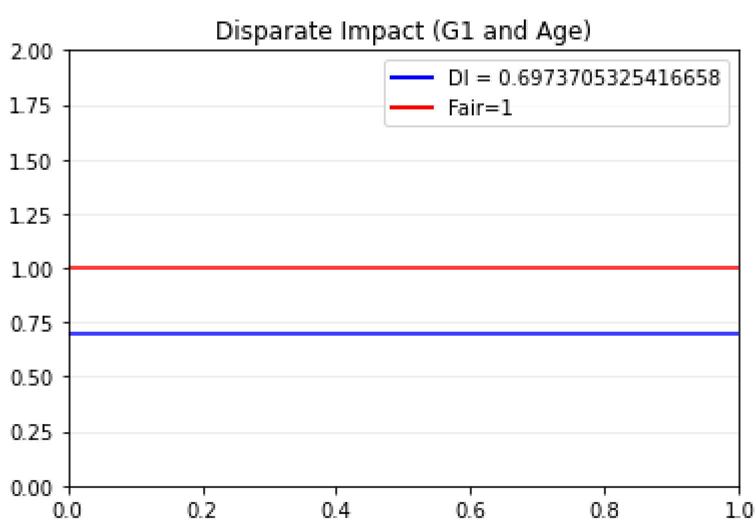
```

    # weights for G1 and sex
    w_pp_sex_g1 = (sex_g1_freq[p_sex].sum() * (sex_g1_freq[p_sex][1] +
            sex_g1_freq.values.sum() * sex_g1_freq[p_sex][1])
    w_pu_sex_g1 = (sex_g1_freq[up_sex].sum() * (sex_g1_freq[p_sex][1]
            sex_g1_freq.values.sum() * sex_g1_freq[up_sex][1])
    w_np_sex_g1 = (sex_g1_freq[p_sex].sum() * (sex_g1_freq[p_sex][0] +
            sex_g1_freq.values.sum() * sex_g1_freq[p_sex][0])
    w_nu_sex_g1 = (sex_g1_freq[up_sex].sum() * (sex_g1_freq[p_sex][0]
            sex_g1_freq.values.sum() * sex_g1_freq[up_sex][0])

    weights = [[w_pp_sex_g1, w_pu_sex_g1, w_np_sex_g1, w_nu_sex_g1]]
    pd.DataFrame(weights,
                 columns=['Positive outcome - Privileged Group', 'Posi
                          'Negative outcome - Privileged Group', 'Nega
        'out/weights_g1_sex.csv', index=False)

    # applying the weights to calculate spd and di
    w_spd_g1_sex = (w_pu_sex_g1 * sex_g1_freq[up_sex][1] / (
            w_pu_sex_g1 * sex_g1_freq[up_sex][1] + w_nu_sex_g1 * sex_g
                           w_pp_sex_g1 * sex_g1_freq[p_sex][1] / (
                           w_pp_sex_g1 * sex_g1_freq[p_sex][1] + w_np_s
    w_di_g1_sex = (w_pu_sex_g1 * sex_g1_freq[up_sex][1] / (
            w_pu_sex_g1 * sex_g1_freq[up_sex][1] + w_nu_sex_g1 * sex_g
                          w_pp_sex_g1 * sex_g1_freq[p_sex][1] / (
                          w_pp_sex_g1 * sex_g1_freq[p_sex][1] + w_np_se

    # weights for G2 and sex
    w_pp_sex_g2 = (sex_g2_freq[p_sex].sum() * (sex_g2_freq[p_sex][1] +
            sex_g2_freq.values.sum() * sex_g2_freq[p_sex][1])
    w_pu_sex_g2 = (sex_g2_freq[up_sex].sum() * (sex_g2_freq[p_sex][1]
            sex_g2_freq.values.sum() * sex_g2_freq[up_sex][1])
    w_np_sex_g2 = (sex_g2_freq[p_sex].sum() * (sex_g2_freq[p_sex][0] +
            sex_g2_freq.values.sum() * sex_g2_freq[p_sex][0])
    w_nu_sex_g2 = (sex_g2_freq[up_sex].sum() * (sex_g2_freq[p_sex][0]
            sex_g2_freq.values.sum() * sex_g2_freq[up_sex][0])

    weights = [[w_pp_sex_g2, w_pu_sex_g2, w_np_sex_g2, w_nu_sex_g2]]
    pd.DataFrame(weights,
                 columns=['Positive outcome - Privileged Group', 'Posi
                          'Negative outcome - Privileged Group', 'Nega
        'out/weights_g2_sex.csv', index=False)

    # applying the weights to calculate spd and di
    w_spd_g2_sex = (w_pu_sex_g2 * sex_g2_freq[up_sex][1] / (
            w_pu_sex_g2 * sex_g2_freq[up_sex][1] + w_nu_sex_g2 * sex_g
                           w_pp_sex_g2 * sex_g2_freq[p_sex][1] / (
                           w_pp_sex_g2 * sex_g2_freq[p_sex][1] + w_np_s
    w_di_g2_sex = (w_pu_sex_g2 * sex_g2_freq[up_sex][1] / (
            w_pu_sex_g2 * sex_g2_freq[up_sex][1] + w_nu_sex_g2 * sex_g
                          w_pp_sex_g2 * sex_g2_freq[p_sex][1] / (
                          w_pp_sex_g2 * sex_g2_freq[p_sex][1] + w_np_se

    # weights for G3 and sex
    w_pp_sex_g3 = (sex_g3_freq[p_sex].sum() * (sex_g3_freq[p_sex][1] +
            sex_g3_freq.values.sum() * sex_g3_freq[p_sex][1])
    w_pu_sex_g3 = (sex_g3_freq[up_sex].sum() * (sex_g3_freq[p_sex][1]
```

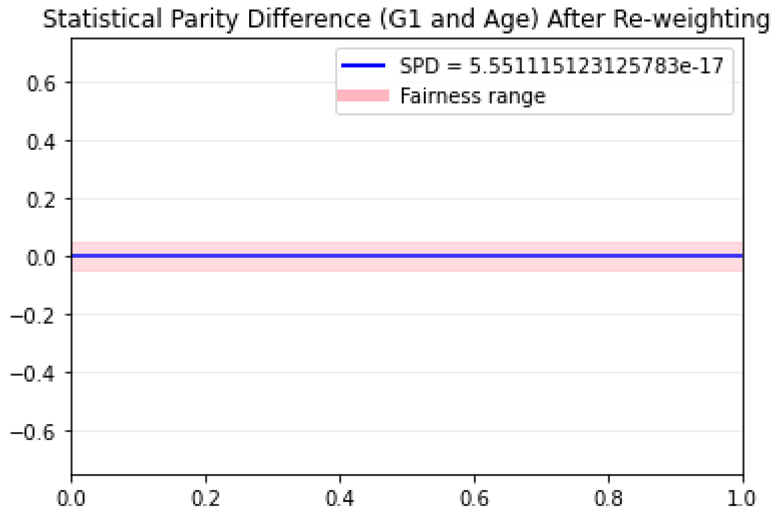
```
            sex_g3_freq.values.sum() * sex_g3_freq[up_sex][1])
    w_np_sex_g3 = (sex_g3_freq[p_sex].sum() * (sex_g3_freq[p_sex][0] +
            sex_g3_freq.values.sum() * sex_g3_freq[p_sex][0])
    w_nu_sex_g3 = (sex_g3_freq[up_sex].sum() * (sex_g3_freq[p_sex][0] 
            sex_g3_freq.values.sum() * sex_g3_freq[up_sex][0])

    weights = [[w_pp_sex_g3, w_pu_sex_g3, w_np_sex_g3, w_nu_sex_g3]]
    pd.DataFrame(weights,
                 columns=['Positive outcome - Privileged Group', 'Posi
                          'Negative outcome - Privileged Group', 'Nega
        'out/weights_g3_sex.csv', index=False)

    # applying the weights to calculate spd and di
    w_spd_g3_sex = (w_pu_sex_g3 * sex_g3_freq[up_sex][1] / (
            w_pu_sex_g3 * sex_g3_freq[up_sex][1] + w_nu_sex_g3 * sex_g
                          w_pp_sex_g3 * sex_g3_freq[p_sex][1] / (
                          w_pp_sex_g3 * sex_g3_freq[p_sex][1] + w_np_
    w_di_g3_sex = (w_pu_sex_g3 * sex_g3_freq[up_sex][1] / (
            w_pu_sex_g3 * sex_g3_freq[up_sex][1] + w_nu_sex_g3 * sex_g
                         w_pp_sex_g3 * sex_g3_freq[p_sex][1] / (
                         w_pp_sex_g3 * sex_g3_freq[p_sex][1] + w_np_se

    w_spd_data = [['G1', 'Age', w_spd_g1_age], ['G2', 'Age', w_spd_g2_
                  ['G1', 'Sex', w_spd_g1_sex], ['G2', 'Sex', w_spd_g2_

    pd.DataFrame(w_spd_data,
                 columns=['Dependent Variable', 'Protected Class Varia
        'out/w_spd.csv', index=False)

    w_di_data = [['G1', 'Age', w_di_g1_age], ['G2', 'Age', w_di_g2_age
                 ['G1', 'Sex', w_di_g1_sex], ['G2', 'Sex', w_di_g2_sex

    pd.DataFrame(w_di_data,
                 columns=['Dependent Variable', 'Protected Class Varial
        'out/w_di.csv', index=False)

    # apply weights to original data set TODO clean this up
    weighted_df = pd.read_csv('dataset/student-por.csv', delimiter=','
    weighted_df['G1_Weighted'] = weighted_df['G1'] + 0.0
    weighted_df.loc[weighted_df['sex'] == "M", ['sex']] = 'Male'
    weighted_df.loc[weighted_df['sex'] == "F", ['sex']] = 'Female'
    weighted_df.loc[weighted_df['age'].between(15, 17, inclusive=True)
    weighted_df.loc[weighted_df['age'].between(18, 22, inclusive=True)
    weighted_df['G1_PassFail'] = pd.cut(weighted_df.G1, bins=bins, labe
    weighted_df['G2_PassFail'] = pd.cut(weighted_df.G2, bins=bins, labe
    # weighted_df['G3_PassFail'] = pd.cut(weighted_df.G3, bins=bins, la

    # add weight columns as floats
    weighted_df['G1_Weighted'] = weighted_df['G1'] + 0.0
    weighted_df['G2_Weighted'] = weighted_df['G2'] + 0.0
    # weighted_df['G3_Weighted'] = weighted_df['G3'] + 0.0

    # outcome: // PO PG == female 15-17 e.g. row 2
    p_pos = (weighted_df['sex'] == p_sex) & (weighted_df['age_group'] 
    weighted_df.loc[p_pos, 'G1_Weighted'] = weighted_df.loc[p_pos, 'G1_
    p_pos = (weighted_df['sex'] == p_sex) & (weighted_df['age_group'] 
    weighted_df.loc[p_pos, 'G2_Weighted'] = weighted_df.loc[p_pos, 'G2_
```

```
    # p_pos = (weighted_df['sex'] == p_sex) & (weighted_df['age_group'
    # weighted_df.loc[p_pos, 'G3_Weighted'] = weighted_df.loc[p_pos, '(

    # outcome: // PO UG == male 18-22 e.g. row 229
    up_pos = (weighted_df['sex'] == up_sex) & (weighted_df['age_group'
    weighted_df.loc[up_pos, 'G1_Weighted'] = weighted_df.loc[up_pos, '(
    up_pos = (weighted_df['sex'] == up_sex) & (weighted_df['age_group'
    weighted_df.loc[up_pos, 'G2_Weighted'] = weighted_df.loc[up_pos, '(
    # up_pos = (weighted_df['sex'] == up_sex) & (weighted_df['age_grou
    # weighted_df.loc[up_pos, 'G3_Weighted'] = weighted_df.loc[up_pos,

    # outcome: // NO PG == female 15-17 e.g. row
    p_neg = (weighted_df['sex'] == p_sex) & (weighted_df['age_group'] =
    weighted_df.loc[p_neg, 'G1_Weighted'] = weighted_df.loc[p_neg, 'G1_
    p_neg = (weighted_df['sex'] == p_sex) & (weighted_df['age_group'] =
    weighted_df.loc[p_neg, 'G2_Weighted'] = weighted_df.loc[p_neg, 'G2_
    # p_neg = (weighted_df['sex'] == p_sex) & (weighted_df['age_group'
    # weighted_df.loc[p_neg, 'G3_Weighted'] = weighted_df.loc[p_neg, '(

    # outcome: // NO UG == male 18-22 e.g. row 165
    up_neg = (weighted_df['sex'] == up_sex) & (weighted_df['age_group'
    weighted_df.loc[up_neg, 'G1_Weighted'] = weighted_df.loc[up_neg, '(
    up_neg = (weighted_df['sex'] == up_sex) & (weighted_df['age_group'
    weighted_df.loc[up_neg, 'G2_Weighted'] = weighted_df.loc[up_neg, '(
    # up_neg = (weighted_df['sex'] == up_sex) & (weighted_df['age_grou
    # weighted_df.loc[up_neg, 'G3_Weighted'] = weighted_df.loc[up_neg,

    weighted_df.to_csv('out/weighted.csv', index=False)

    for i in range(0, 6):
        spd_i = spd_data[i]
        plt.axhline(y=spd_i[2], color='blue')
        plt.axhspan(-0.05, 0.05, alpha=0.5, color='#ffb6c1')
        axes = plt.gca()
        axes.set_ylim([-0.75, 0.75])
        plt.grid(True, axis='y', alpha=0.2, color='#999999')
        plt.title('Statistical Parity Difference (' + spd_i[0] + ' and
        plt.legend([Line2D([0], [0], color='blue', lw=2), Line2D([0],
                    ['SPD = ' + str(spd_i[2]), 'Fairness range'])
        plt.savefig('out/spd_' + spd_i[0] + '_' + spd_i[1] + '.png')
        plt.show()

        di_i = di_data[i]
        plt.axhline(y=di_i[2], color='blue')
        plt.axhline(y=1, color='red')
        axes = plt.gca()
        axes.set_ylim([0, 2])
        plt.grid(True, axis='y', alpha=0.2, color='#999999')
        plt.title('Disparate Impact (' + di_i[0] + ' and ' + di_i[1] +
        plt.legend([Line2D([0], [0], color='blue', lw=2), Line2D([0],
                    ['DI = ' + str(di_i[2]), 'Fair=1'])
        plt.savefig('out/di_' + di_i[0] + '_' + di_i[1] + '.png')
        plt.show()

        w_spd_i = w_spd_data[i]
        plt.axhline(y=w_spd_i[2], color='blue')
        plt.axhspan(-0.05, 0.05, alpha=0.5, color='#ffb6c1')
```

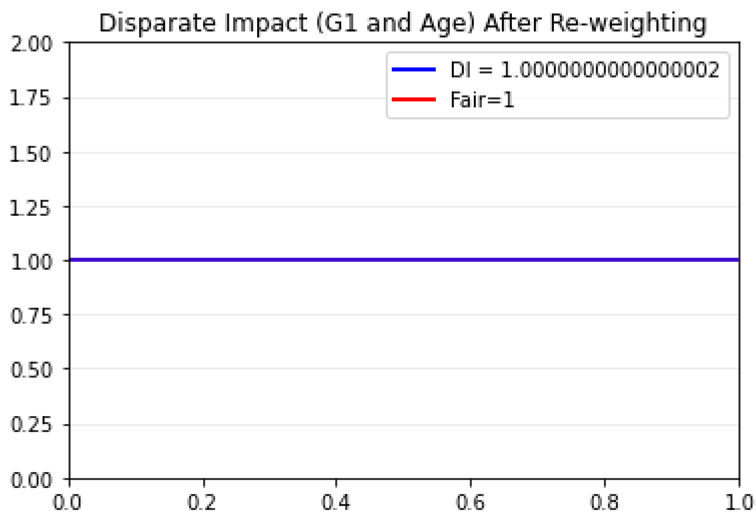
```
        axes = plt.gca()
        axes.set_ylim([-0.75, 0.75])
        plt.grid(True, axis='y', alpha=0.2, color='#999999')
        plt.title('Statistical Parity Difference (' + w_spd_i[0] + ' ar
        plt.legend([Line2D([0], [0], color='blue', lw=2), Line2D([0],
                   ['SPD = ' + str(w_spd_i[2]), 'Fairness range'])
        plt.savefig('out/w_spd_' + w_spd_i[0] + '_' + w_spd_i[1] + '.pı
        plt.show()

        w_di_i = w_di_data[i]
        plt.axhline(y=1, color='red')
        plt.axhline(y=w_di_i[2], color='blue')
        axes = plt.gca()
        axes.set_ylim([0, 2])
        plt.grid(True, axis='y', alpha=0.2, color='#999999')
        plt.legend([Line2D([0], [0], color='blue', lw=2), Line2D([0],
                   ['DI = ' + str(w_di_i[2]), 'Fair=1'])
        plt.title('Disparate Impact (' + w_di_i[0] + ' and ' + w_di_i[
        plt.savefig('out/w_di_' + w_di_i[0] + '_' + w_di_i[1] + '.png'
        plt.show()

STEP3()
```



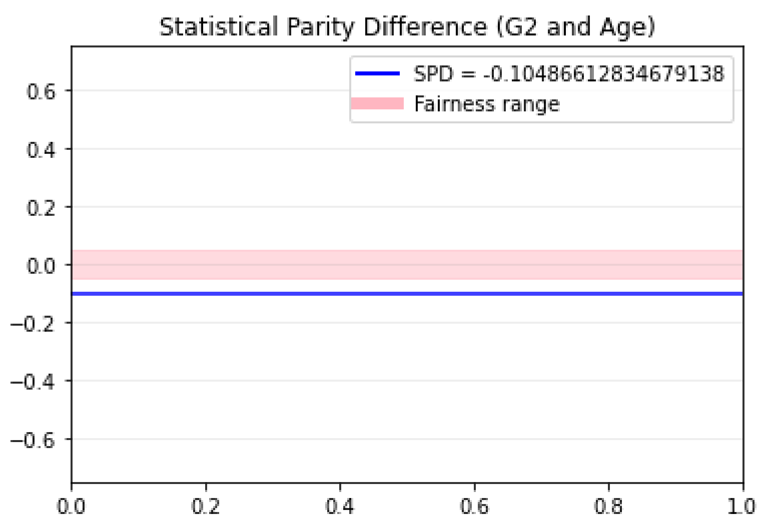




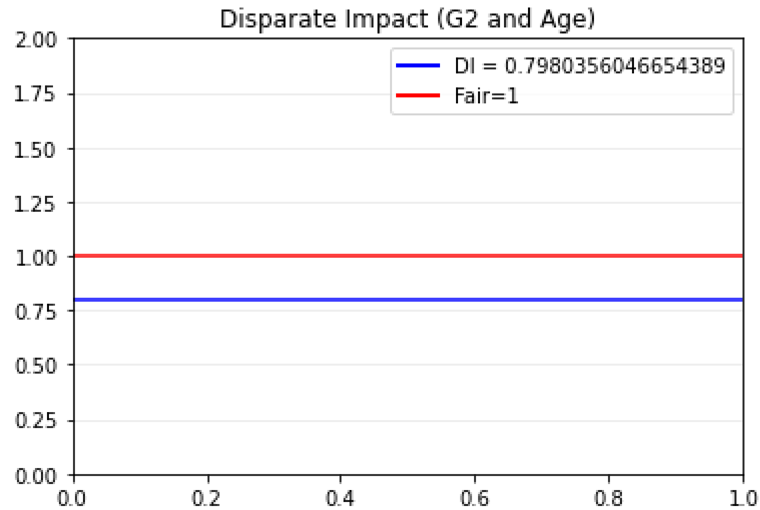
Statistical Parity Difference (G1 and Age) After Re-weighting
SPD = 5.551115123125783e-17
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0

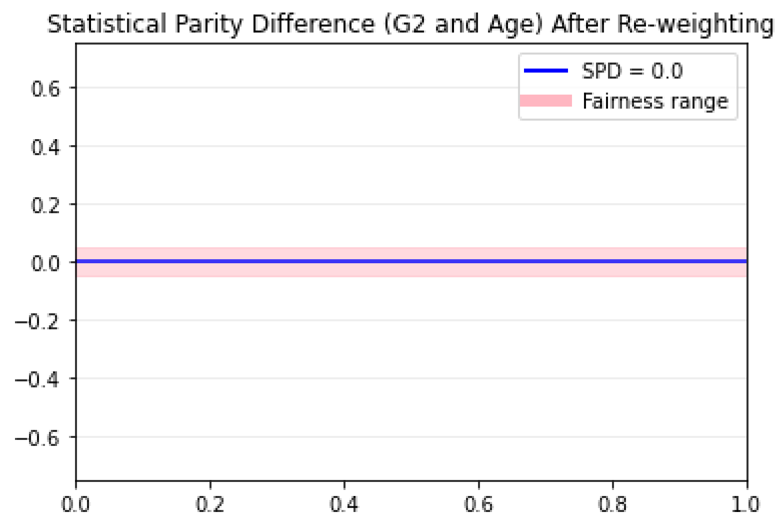


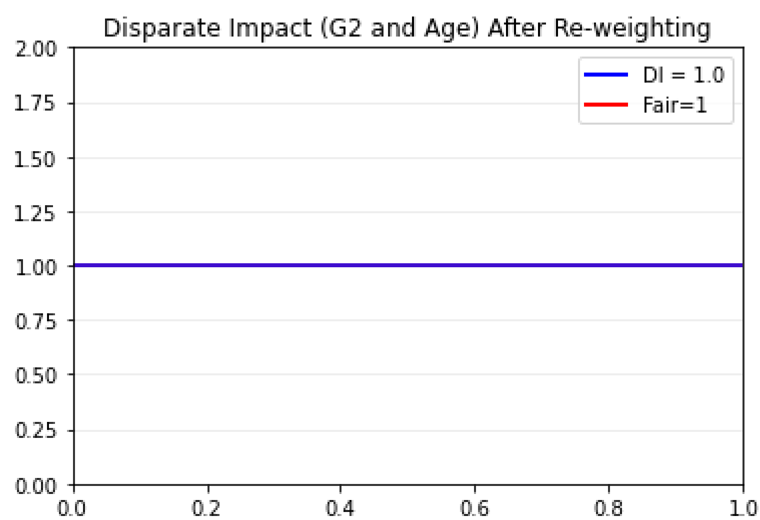
Disparate Impact (G1 and Age) After Re-weighting
DI = 1.0000000000000002
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

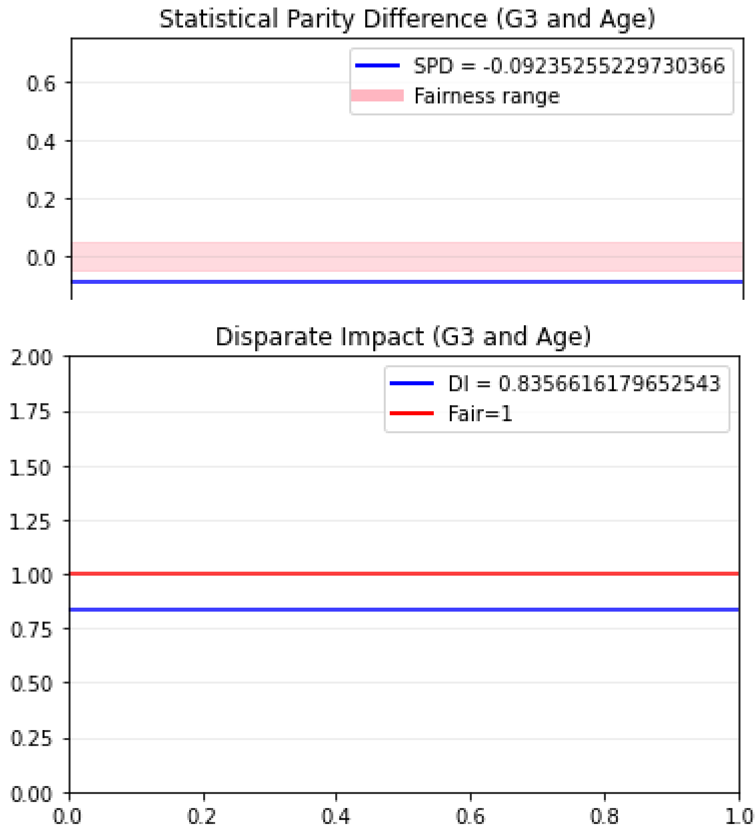


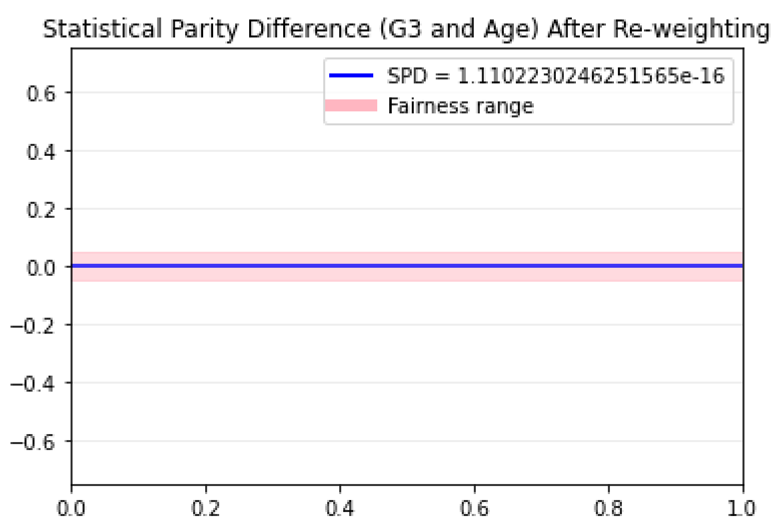
Statistical Parity Difference (G2 and Age)
SPD = -0.10486612834679138
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0


Disparate Impact (G2 and Age)
DI = 0.7980356046654389
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

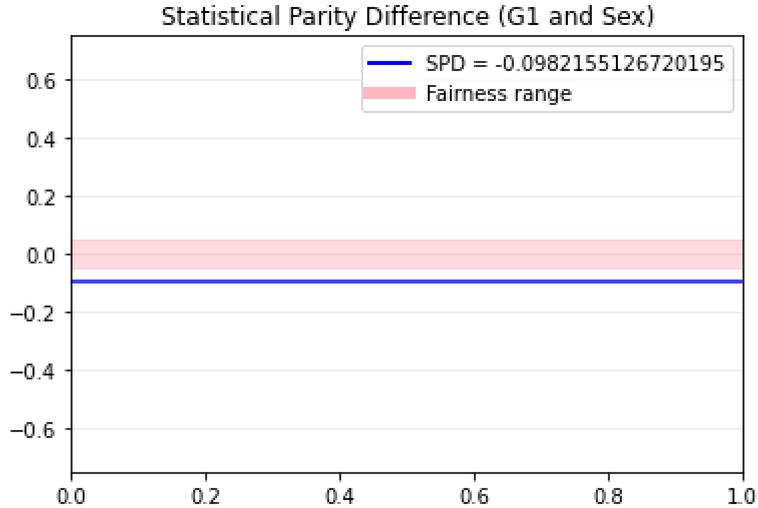

Statistical Parity Difference (G2 and Age) After Re-weighting
SPD = 0.0
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0

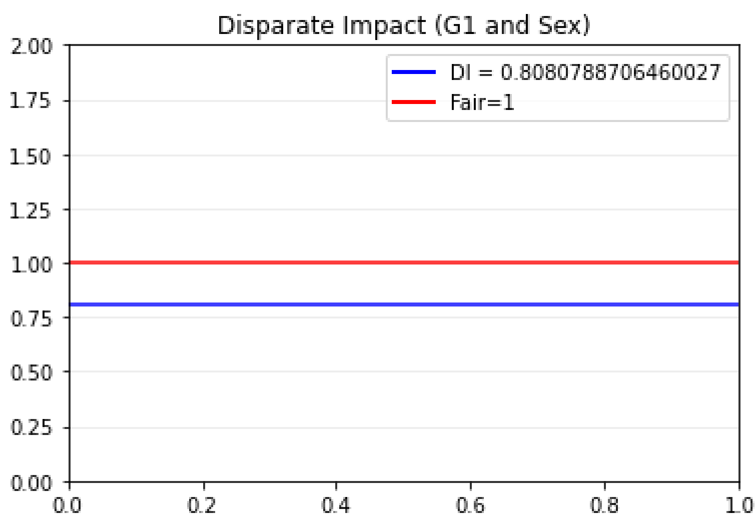

Disparate Impact (G2 and Age) After Re-weighting
DI = 1.0
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

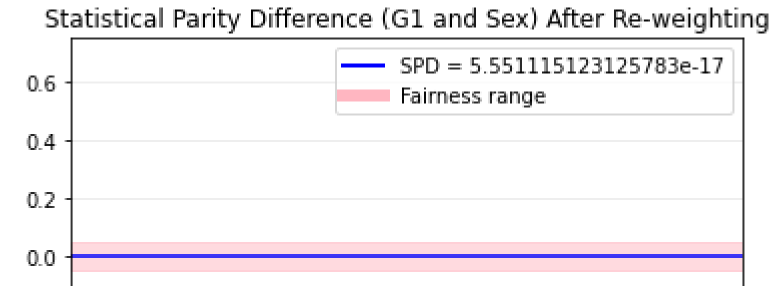

Statistical Parity Difference (G3 and Age)
SPD = -0.09235255229730366
Fairness range
0.6
0.4
0.2
0.0
Disparate Impact (G3 and Age)
DI = 0.8356616179652543
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

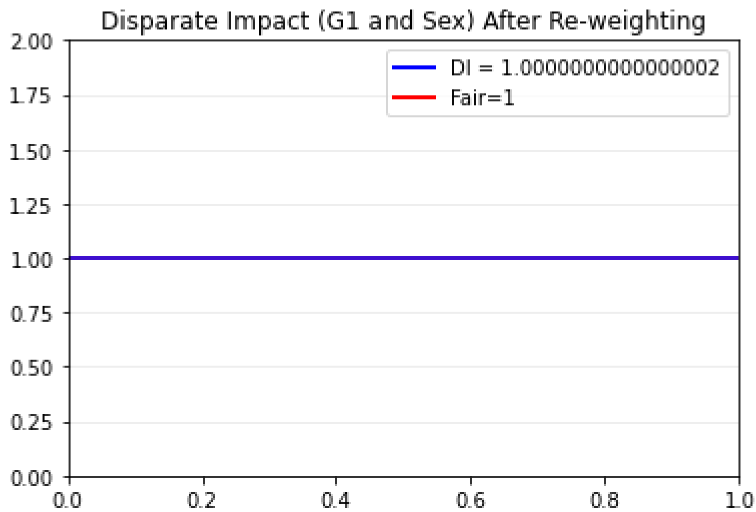

Statistical Parity Difference (G3 and Age) After Re-weighting
SPD = 1.1102230246251565e-16
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0

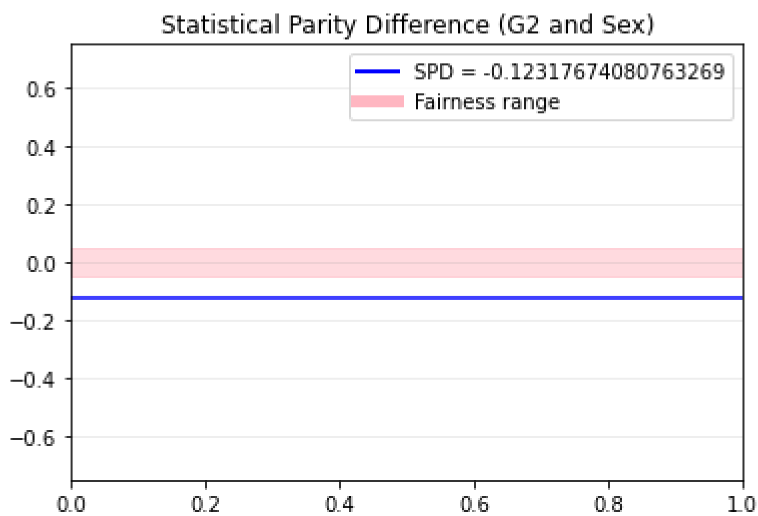

Disparate Impact (G3 and Age) After Re-weighting
DI = 1.0000000000000002
Fair=1

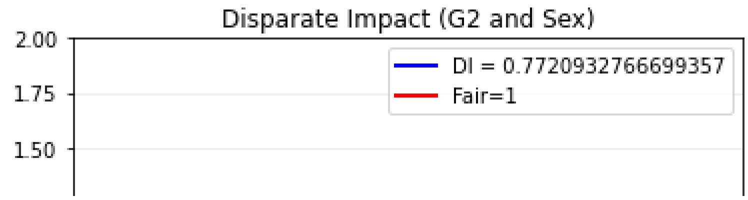

Statistical Parity Difference (G1 and Sex)
SPD = -0.0982155126720195
Fairness range

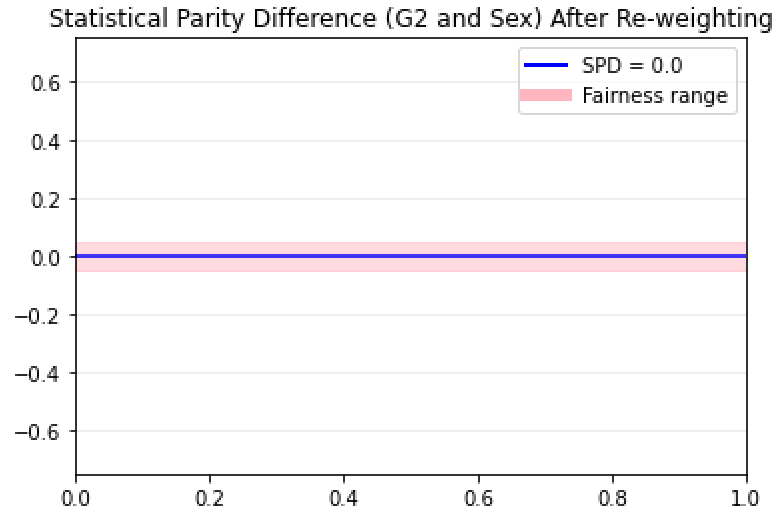

Disparate Impact (G1 and Sex)
DI = 0.8080788706460027
Fair=1

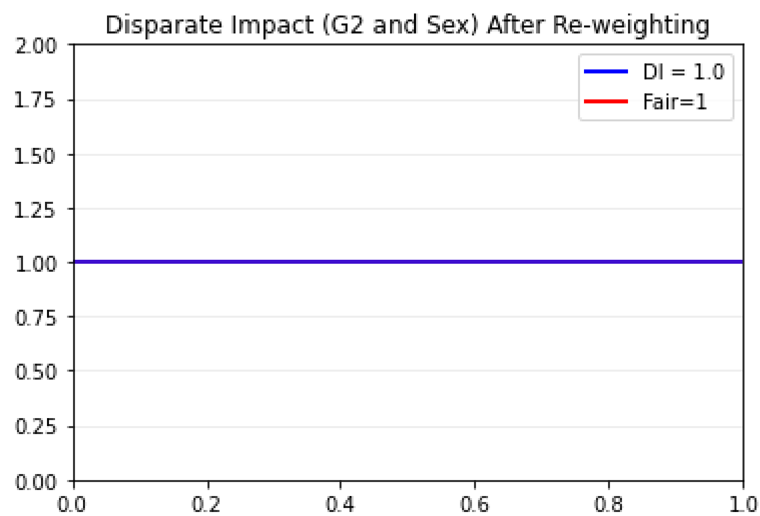

Statistical Parity Difference (G1 and Sex) After Re-weighting
SPD = 5.551115123125783e-17
Fairness range
0.6
0.4
0.2
0.0

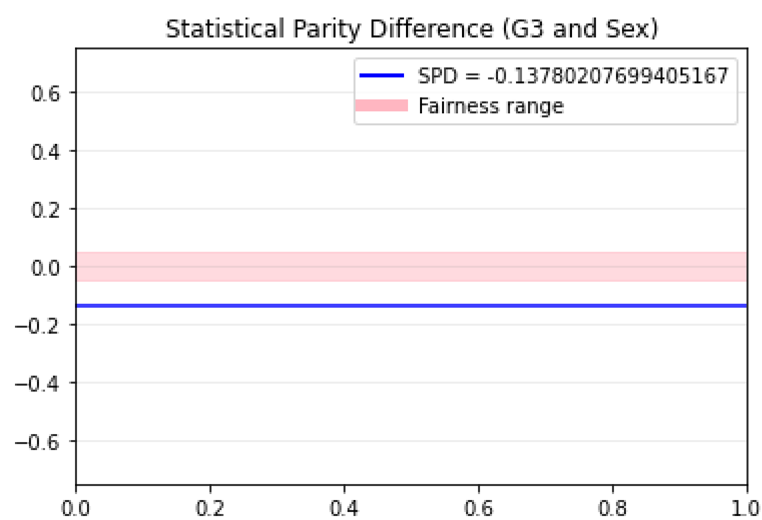

Disparate Impact (G1 and Sex) After Re-weighting
DI = 1.0000000000000002
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

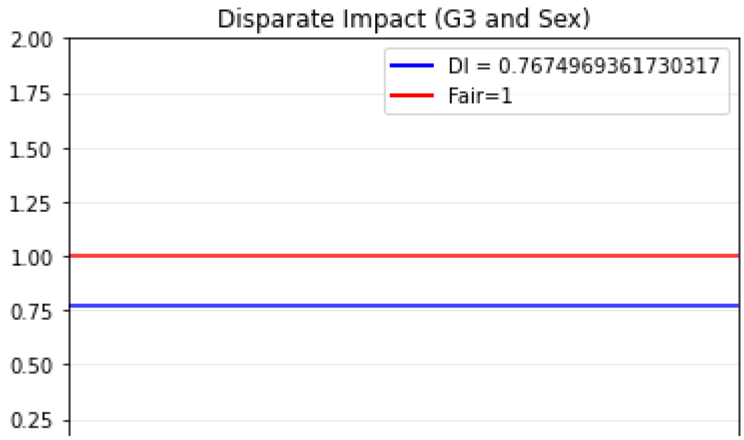

Statistical Parity Difference (G2 and Sex)
SPD = -0.12317674080763269
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0

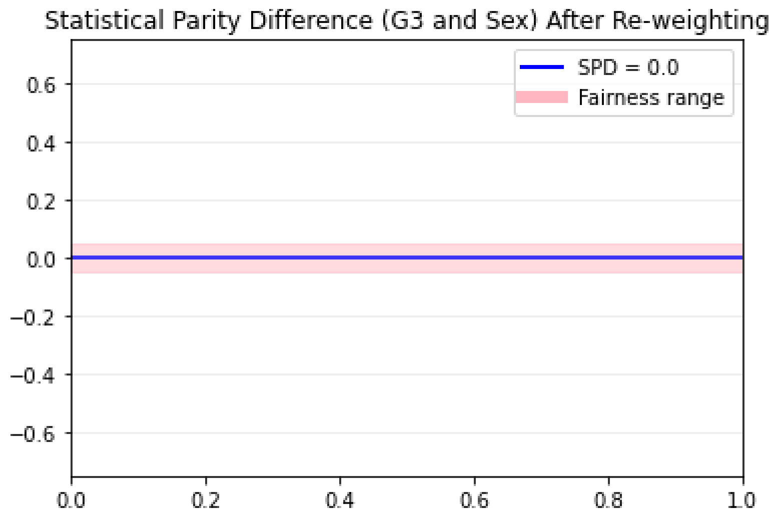

Disparate Impact (G2 and Sex)
DI = 0.7720932766699357
Fair=1
2.00
1.75
1.50

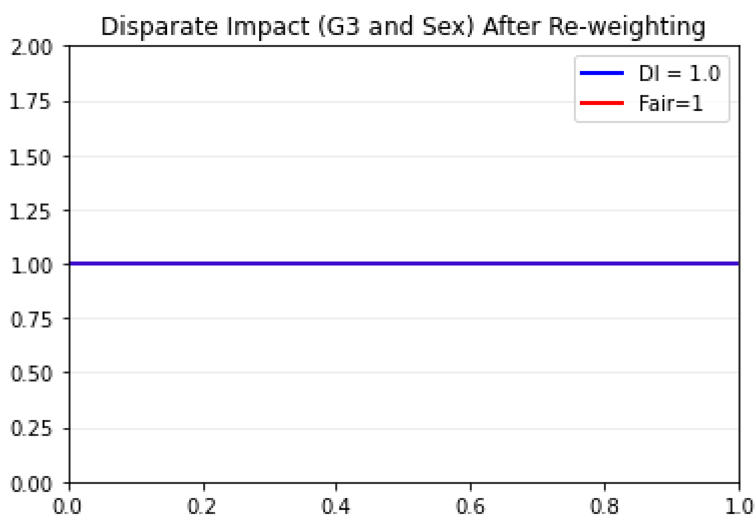

Statistical Parity Difference (G2 and Sex) After Re-weighting
SPD = 0.0
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0


Disparate Impact (G2 and Sex) After Re-weighting
DI = 1.0
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0


Statistical Parity Difference (G3 and Sex)
SPD = -0.13780207699405167
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0


Disparate Impact (G3 and Sex)
DI = 0.7674969361730317
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25


Statistical Parity Difference (G3 and Sex) After Re-weighting
SPD = 0.0
Fairness range
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0


Disparate Impact (G3 and Sex) After Re-weighting
DI = 1.0
Fair=1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

Step 4:

Option A: Independent Variable: 0.024299344 Privileged Group: Female Unprivileged Group: Male

Independent	Metric	Original	Transformed	Difference
Sex	Statistical Parity Difference	-0.119759532	-0.151282051	0.03152252
Sex	Disparate Impact	0.774299344	0.75	0.024299344

Was there a positive change, negative change, or no change on that fairness metric after transforming the dataset (from Step 3.4)?

After transforming the dataset, there was a slight negative change for both Statistical Parity Differnce and Disparate Impact.

Was there a positive change, negative change, or no change on that fairness metric after training the classifier - with respect to the original testing dataset and the transformed testing dataset?

In our opinion, there was no change on the fairness metric after training the classifier. If trained on the transformed data, the testing results would result in the same or similar metrics as the training transformed data itself.

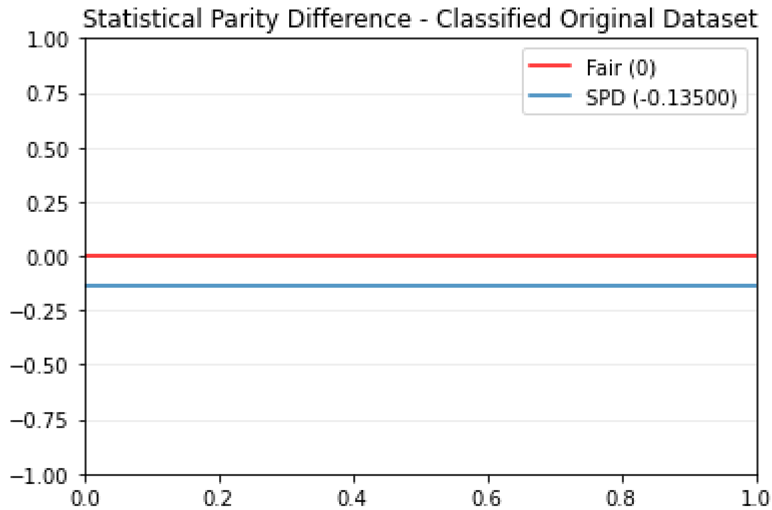
In [11]:

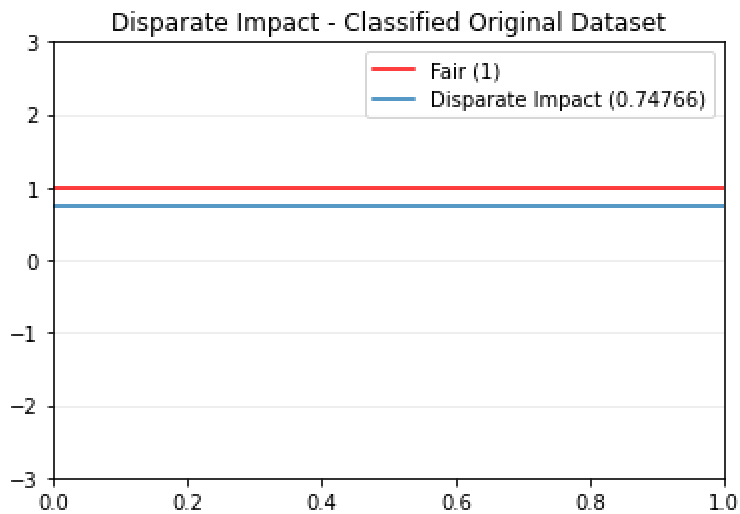
```

def STEP4():
    raw_df = pd.read_csv(r"dataset/student-por.csv", delimiter=',')
    df = raw_df.replace({'sex': {'M': 0, 'F': 1}})

    df_num_rows = int(len(df))

    # split data into train and test sets
    shuffled = df.sample(frac=1)
    df_train = shuffled.iloc[:int(df_num_rows / 2)]
    df_test = shuffled.iloc[int(df_num_rows / 2):]

    # classify data Y in this case is G3. Features are G1 and G2,

    features = ['sex', 'G1', 'G2']
    y_feature = 'G3'

    # create training data
    df_train_x = df_train[features]
    df_train_y = df_train[y_feature]

    # create and train classifier
    orig_classifier = DecisionTreeClassifier()
    orig_classifier.fit(df_train_x, df_train_y)

    # run classifier against the test set
    df_test_x = df_test[features]
    predicted = orig_classifier.predict(df_test_x)

    df_test_x['G3_Predicted'] = predicted
    df_test_x['G3_Actual'] = df_test[y_feature]

    # create a classifier for the data from Step 3.3
    weighted_raw_df = pd.read_csv(r"out/weighted.csv", delimiter=',')
    weighted_df = weighted_raw_df.replace({'sex': {'Male': 0, 'Female'

    weighted_df_num_rows = int(len(weighted_df))

    # split data into train and test sets
    weighted_shuffled = weighted_df.sample(frac=1)
    weighted_df_train = weighted_shuffled.iloc[:int(weighted_df_num_ro
    weighted_df_test = weighted_shuffled.iloc[int(weighted_df_num_rows

    # create training data
    features = ['sex', 'G1_Weighted', 'G2_Weighted']
    y_feature = 'G3'
    weighted_df_train_x = weighted_df_train[features]
    weighted_df_train_y = weighted_df_train[y_feature]

    # create and train classifier
    weighted_classifier = DecisionTreeClassifier()
    weighted_classifier.fit(weighted_df_train_x, weighted_df_train_y)

    # run classifier against the test set
    weighted_df_test_x = weighted_df_test[features]
    weighted_predicted = weighted_classifier.predict(weighted_df_test_
```

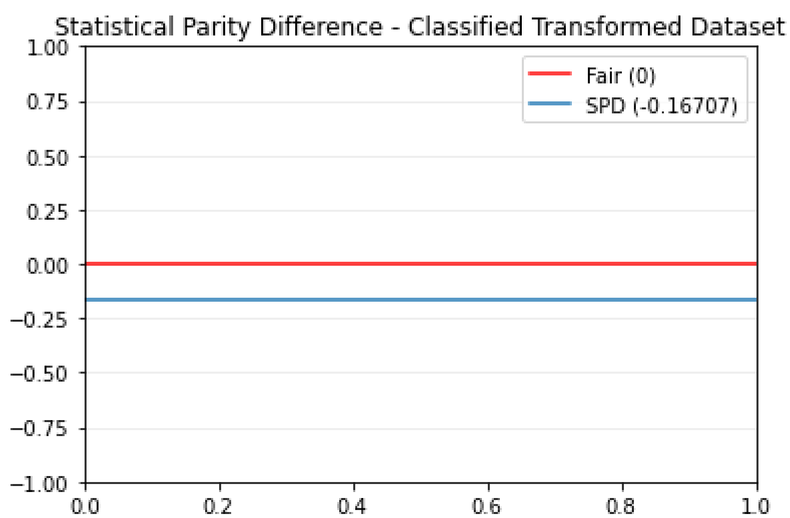
```python

    weighted_df_test_x['G3_Predicted'] = weighted_predicted
    weighted_df_test_x['G3_Actual'] = weighted_df_test[y_feature]

    df_test_x.to_csv('out/predicted_g3-unweighted.csv', index=False)
    weighted_df_test_x.to_csv('out/predicted_g3-weighted.csv', index=Fa

    # calculate fairness based on sex
    up_sex = 0  # male
    p_sex = 1  # female
    bins = [0, 11, 19]
    labels = [0, 1]

    # calculate fairness for original dataset
    df_test_x['G3_Predicted_Discrete'] = pd.cut(df_test_x['G3_Predicte
                                             include_lowest=True)
    sex_g3_freq_og = df_test_x.groupby(['sex', 'G3_Predicted_Discrete'

    # statistical parity difference G3 and sex
    spd_g3_sex_og = (sex_g3_freq_og[up_sex][1] / (sex_g3_freq_og[up_se
            sex_g3_freq_og[p_sex][1] / (sex_g3_freq_og[p_sex][1] + sex_

    # disparate impact G3 and sex
    di_g3_sex_og = (sex_g3_freq_og[up_sex][1] / (sex_g3_freq_og[up_sex
            sex_g3_freq_og[p_sex][1] / (sex_g3_freq_og[p_sex][1] + sex_

    # calculate fairness for transformed dataset
    weighted_df_test_x['G3_Predicted_Discrete'] = pd.cut(weighted_df_te
                                                      include_lowes
    sex_g3_freq_transformed = weighted_df_test_x.groupby(['sex', 'G3_P

    # spd of weighted for weighted dataset
    spd_g3_sex_transformed = (sex_g3_freq_transformed[up_sex][1] / (
            sex_g3_freq_transformed[up_sex][1] + sex_g3_freq_transform
                                     sex_g3_freq_transformed[p_sex][1]
                                     sex_g3_freq_transformed[p_sex][1]

    # disparate impact G3 and sex for weighted dataset
    di_g3_sex_transformed = (sex_g3_freq_transformed[up_sex][1] / (
            sex_g3_freq_transformed[up_sex][1] + sex_g3_freq_transform
                                    sex_g3_freq_transformed[p_sex][1] /
                                    sex_g3_freq_transformed[p_sex][1] +
    print("SPD original: {}".format(spd_g3_sex_og))
    print("DI original: {}".format(di_g3_sex_og))

    print("SPD weighted: {}".format(spd_g3_sex_transformed))
    print("DI weighted: {}".format(di_g3_sex_transformed))

    outcomes = pd.DataFrame(columns=['Independent Variable', 'Metric',
    outcomes.loc[len(outcomes)] = ['Sex', 'Statistical Parity Differenc
                                   spd_g3_sex_og - spd_g3_sex_transform
    outcomes.loc[len(outcomes)] = ['Sex', 'Disparate Impact', di_g3_se
                                   di_g3_sex_og - di_g3_sex_transforme

    outcomes.to_csv('out/fairness_metrics_classified.csv', index=False

    # Plots for step 5 #TODO Make these pretty
```

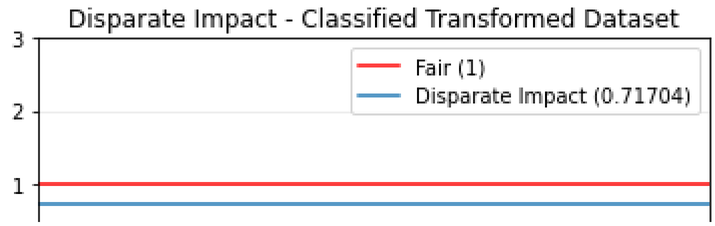
```
    plt.clf()
    plt.axhline(y=0, color='r', label='Fair (0)')
    plt.axhline(spd_g3_sex_og, label="SPD ({:.5f})".format(spd_g3_sex_
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.title("Statistical Parity Difference - Classified Original Data
    top = math.ceil(abs(spd_g3_sex_og) + .3)
    plt.ylim(-top, top)
    plt.savefig('out/spd_g3_sex_og.png', bbox_inches='tight')
    plt.legend()
    plt.show()

    plt.clf()
    plt.axhline(y=1, color='r', label='Fair (1)')
    plt.axhline(di_g3_sex_og, label="Disparate Impact ({:.5f})".format
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.title("Disparate Impact - Classified Original Dataset")
    top = math.ceil(abs(di_g3_sex_og) + .3 + 1)
    plt.ylim(-top, top)
    plt.legend()
    plt.savefig('out/di_g3_sex_og.png', bbox_inches='tight')
    plt.show()

    plt.clf()
    plt.axhline(y=0, color='r', label='Fair (0)')
    plt.axhline(spd_g3_sex_transformed, label="SPD ({:.5f})".format(sp
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.title("Statistical Parity Difference - Classified Transformed 
    top = math.ceil(abs(spd_g3_sex_transformed) + .3)
    plt.ylim(-top, top)
    plt.legend()
    plt.savefig('out/spd_g3_sex_transformed.png', bbox_inches='tight')
    plt.show()

    plt.clf()
    plt.axhline(y=1, color='r', label='Fair (1)')
    plt.axhline(di_g3_sex_transformed, label="Disparate Impact ({:.5f}
    plt.grid(True, axis='y', alpha=0.2, color='#999999')
    plt.title("Disparate Impact - Classified Transformed Dataset")
    top = math.ceil(abs(di_g3_sex_transformed) + .3 + 1)
    plt.ylim(-top, top)
    plt.legend()
    plt.savefig('out/di_g3_sex_transformed.png', bbox_inches='tight')
    plt.show()

STEP4()
```

```
SPD original: -0.135
DI original: 0.7476635514018691
SPD weighted: -0.16706786768131704
DI weighted: 0.7170382060893009
```


Statistical Parity Difference - Classified Original Dataset
Fair (0)
SPD (-0.13500)


Disparate Impact - Classified Original Dataset
Fair (1)
Disparate Impact (0.74766)


Statistical Parity Difference - Classified Transformed Dataset
Fair (0)
SPD (-0.16707)



Step 5:

Team Members:

- Alex Carmona
- Sarah Hernandez
- Neesha Sinha
- Yaima Valdivia

Questions, Answered:

1) Explain which fairness metric (if any) is best and provide a justification for your answer:

Both the Statistical Parity Difference and the Disparate Impact metrics, used here, successfully mitigated bias in our dataset, and both are used in industry to mitigate bias on a grander scale. The difference between these two measures is subtle. The Statistical Parity Difference measures, the statistical difference in outcomes between privileged and underprivileged groups, while the disparate impact measures the ratio of outcomes. While both serve useful functions, ultimatley I would pick th disparate impact measurement because it is 1) easier to explain, and 2) easier for the less statistical-minded to visualize. Because communication is key in any industry, I would pick the more communicabl metric over the lesser one.

Individual Responses:

Alex Carmona:

Did any of these approaches seem to work to mitigate bias (or increase fairness)? Explain your reasoning.
Yes, after applying the reweighting strategy, the bias seemed to be mitigated. Both SPD and DI were moved closer towards being fair.

Did any group receive a positive advantage?
Yes. The weights provided an advantage towards unprivileged groups.
Was any group disadvantaged by these approaches?
No. These improvements helped provide a positive outcome for unprivileged groups without making an impact on privileged groups.
What issues would arise if you used these methods to mitigate bias?
While using the weighted values to mitigate bias provides a “prettier” picture, it does not necessarily reflect reality. Additionally, the algorithm we used to create the weights is very rudimentary and would need to be revisited if we were to expand the data set.

Sarah Hernandez:

Did any of these approaches seem to work to mitigate bias (or increase fairness)? Explain your reasoning.
Yes! Our reweighting strategy proved to be most effctive. For both our metrics, Statisical Parity Difference and Disparate Impact, reweighting brought their values to their statisstical ideal (0 and 1, respectively!).
Did any group receive a positive advantage?
Yes! Reweighting resulted in a more fair distribution of outcomes, resulting in a positive advantage for our underprivileeged groupss.
Was any group disadvantaged by these approaches?
For this particular reweighting, no group was disadvantaged by the result.
What issues would arise if you used these methods to mitigate bias?
For one, applying our reweighting algorithm to another set would not prove useful, as it is specifically designed for this dataset alone. Additionally, while our reweighting method only serves to increase fairness across multiple metrics, the broader public may not see it as such, and could result in public outcry.

Neesha Sinha:

Did any of these approaches seem to work to mitigate bias (or increase fairness)? Explain your reasoning.
Reweighting strategies did work to mitigate bias. After applying different weights to each population, the fairness metrics we considered (SPD and DI) both changed to match the fair threshold (SPD = 0 and DI = 1).
Did any group receive a positive advantage?
Since the reweighting technique used different weights for each dependent variable and protected class variable combination, each of the six groups would be deemed "fair" once the appropriate weights were applied.
Was any group disadvantaged by these approaches?
Similar to the above answer, since we used different weights for each combination, each group was fair, and no one was disadvantaged since each combination was weighed separately.
What issues would arise if you used these methods to mitigate bias?
If we use reweighting to mitigate bias, it results in an inaccurate representation of the actual data. Since different populations would be weighed differently, we're essentially changing the data set to portray the populations on an equal footing.

Yaima Valdivia:

Did any of these approaches seem to work to mitigate bias (or increase fairness)? Explain your reasoning.
Reweighting did work to mitigate bias. The fairness metrics used achieved the fair threshold (Statistical Parity Difference = 0, Disparate Impact = 1) after the approach.
Did any group receive a positive advantage?
Both groups received a positive benefit after applying the custom reweighting techniques.
Was any group disadvantaged by these approaches?
There was no introduced disadvantage after reweighting the sets.
What issues would arise if you used these methods to mitigate bias?
The reweighting technique is ideal when we cannot change the values. Because the approach was customized to match the existing dataset, the same method could introduce biases in anything different.

In [ ]:

```
 
```