fairlearn_decisiontree_age_gender

December 1, 2023

0.0.1 1. Data Visualization

[ ]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
df = pd.read_csv('../data/heart.csv')
print(df.shape)
df.head(10)
```

```
(918, 12)
```

[ ]:
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122
5   39   M           NAP        120          339          0     Normal    170
6   45   F           ATA        130          237          0     Normal    170
7   54   M           ATA        110          208          0     Normal    142
8   37   M           ASY        140          207          0     Normal    130
9   48   F           ATA        120          284          0     Normal    120

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
5              N      0.0       Up             0
6              N      0.0       Up             0
7              N      0.0       Up             0
8              Y      1.5     Flat             1
9              N      0.0       Up             0
```

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

```
[ ]: df.describe().T
```

[ ]:

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

RestingBP and Cholesterol have min 0 Change values from 0 to null for those two columns #Need to discuss how to handle them

```
[ ]: df['RestingBP'] = df['RestingBP'].replace(0, np.nan)
     df['Cholesterol'] = df['Cholesterol'].replace(0, np.nan)
     df.isnull().sum() / df.shape[0]
```

```
[ ]: Age              0.000000
     Sex              0.000000
     ChestPainType    0.000000
     RestingBP        0.001089
     Cholesterol      0.187364
     FastingBS        0.000000
     RestingECG       0.000000
     MaxHR            0.000000
```

```
ExerciseAngina    0.000000
Oldpeak           0.000000
ST_Slope          0.000000
HeartDisease      0.000000
dtype: float64
```

[ ]:
```
df['HeartDisease'].value_counts()
```

[ ]:
```
1    508
0    410
Name: HeartDisease, dtype: int64
```

[ ]:
```
fig = px.pie(df, names='HeartDisease', title='Percentage for each outcome')
fig.update_layout(width=500, height=300)
fig.show()
```

[ ]:
```
numerical= df.drop(['HeartDisease'], axis=1).select_dtypes('number').columns

categorical = df.select_dtypes('object').columns

print(f'Numerical Columns:  {df[numerical].columns}\n')
print(f'Categorical Columns: {df[categorical].columns}')
```

```
Numerical Columns:  Index(['Age', 'RestingBP', 'Cholesterol', 'FastingBS',
'MaxHR', 'Oldpeak'], dtype='object')

Categorical Columns: Index(['Sex', 'ChestPainType', 'RestingECG',
'ExerciseAngina', 'ST_Slope'], dtype='object')
```

[ ]:
```
df.groupby(['HeartDisease', 'Sex'])[['Age']].describe().T
```

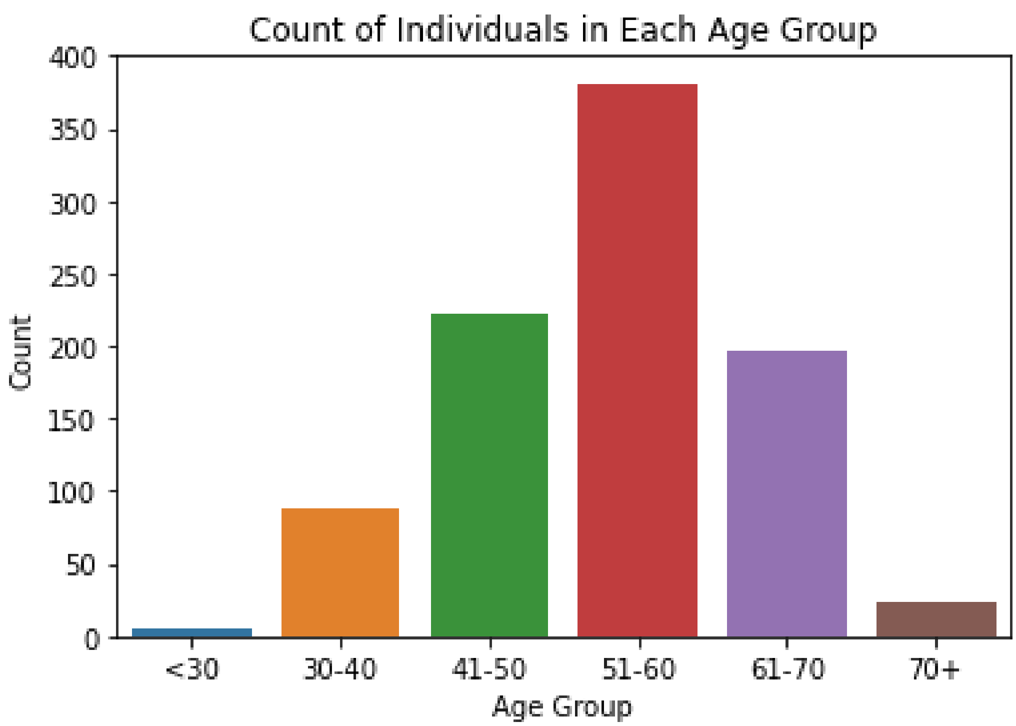
[ ]:

HeartDisease		0		1	
Sex		F	M	F	M
Age	count	143.000000	267.000000	50.000000	458.000000
	mean	51.202797	50.202247	56.180000	55.868996
	std	9.627981	9.344911	8.220656	8.788562
	min	30.000000	28.000000	33.000000	31.000000
	25%	44.000000	42.000000	51.000000	51.000000
	50%	51.000000	51.000000	58.000000	57.000000
	75%	57.000000	57.000000	62.000000	62.000000
	max	76.000000	75.000000	73.000000	77.000000

[ ]:
```
fig = px.box(df, y="Age", x="HeartDisease", points="all",  color="Sex",
             title="Distribution of age of database participants (stratified by␣
 ↪outcome)")
fig.show()
```

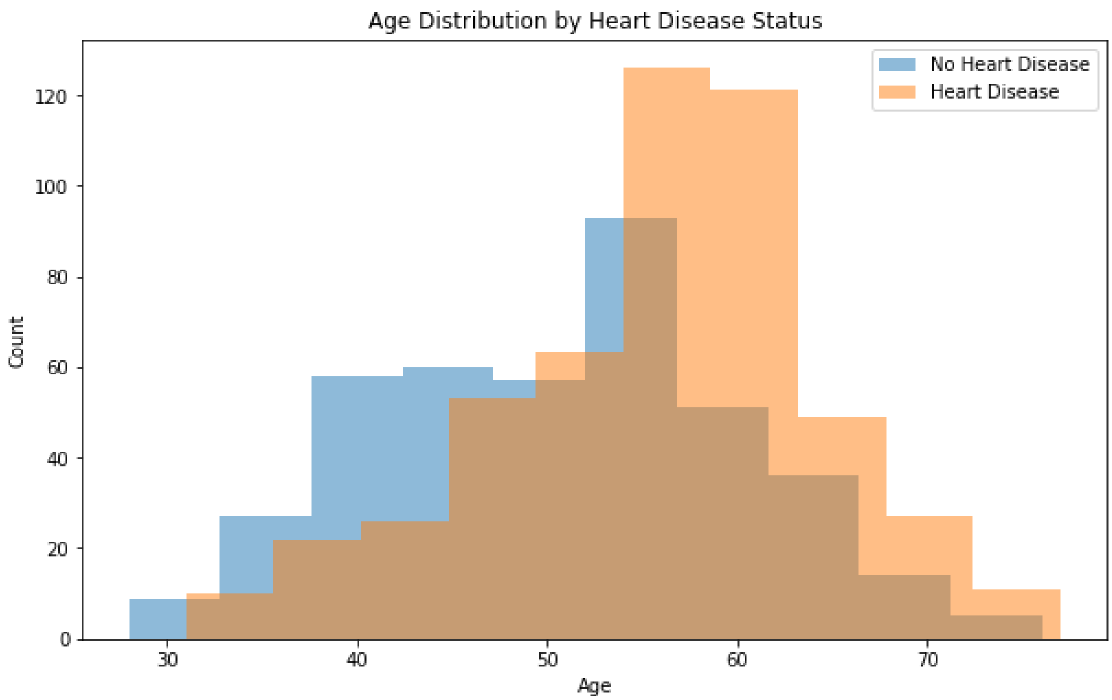
[ ]:
```python
fig = px.box(df, y="Age", x="HeartDisease", points="all",
             title="Distribution of age of database participants (stratified by␣
 ↪outcome)")

fig.show()
```

[ ]:
```python
unique_ages = df['Age'].unique()

bins = [0, 30, 40, 50, 60, 70, float('inf')]
labels = ['<30', '30-40', '41-50', '51-60', '61-70', '70+']

df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=True)

sns.countplot(x='AgeGroup', data=df)
plt.title("Count of Individuals in Each Age Group")
plt.xlabel("Age Group")
plt.ylabel("Count")
plt.show()
```



[ ]:
```python
df_no_disease = df[df['HeartDisease'] == 0]
df_with_disease = df[df['HeartDisease'] == 1]

# Plotting
```

```python
plt.figure(figsize=(10, 6))
plt.hist(df_no_disease['Age'], alpha=0.5, label='No Heart Disease')
plt.hist(df_with_disease['Age'], alpha=0.5, label='Heart Disease')
plt.title('Age Distribution by Heart Disease Status')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend()
plt.show()
```



[ ]:
```python
df_copy = df.copy()
df_copy['c'] = 1
df_copy.groupby(['HeartDisease', 'Sex']).agg('count')[['c']]
```

[ ]:
```
                    c
HeartDisease Sex
0            F    143
             M    267
1            F     50
             M    458
```

[ ]:
```python
fig = px.histogram(df_copy, x="Sex", y="c",
             color='HeartDisease', barmode='group',
             title="Distribution of outcome of database participants␣
 ↪(stratified by sex)")
```

```
fig.update_layout(width=700, height=300)

fig.show()
```

0.0.2 2. Data cleaning and pre-processing

2.1 Data cleaning

[ ]:
```
X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']

sex = df.pop("Sex") # Pop function drops and assigns at the same time
df.head(10)
```

[ ]:

	Age	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	\
0	40	ATA	140.0	289.0	0	Normal	172	
1	49	NAP	160.0	180.0	0	Normal	156	
2	37	ATA	130.0	283.0	0	ST	98	
3	48	ASY	138.0	214.0	0	Normal	108	
4	54	NAP	150.0	195.0	0	Normal	122	
5	39	NAP	120.0	339.0	0	Normal	170	
6	45	ATA	130.0	237.0	0	Normal	170	
7	54	ATA	110.0	208.0	0	Normal	142	
8	37	ASY	140.0	207.0	0	Normal	130	
9	48	ATA	120.0	284.0	0	Normal	120	

	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease	AgeGroup
0	N	0.0	Up	0	30-40
1	N	1.0	Flat	1	41-50
2	N	0.0	Up	0	30-40
3	Y	1.5	Flat	1	41-50
4	N	0.0	Up	0	51-60
5	N	0.0	Up	0	30-40
6	N	0.0	Up	0	41-50
7	N	0.0	Up	0	51-60
8	Y	1.5	Flat	1	30-40
9	N	0.0	Up	0	41-50

[ ]:
```
numeric_features = ["Age", "RestingBP", "Cholesterol", "Oldpeak", "RestingBP",␣
↪"MaxHR", "Oldpeak"]
X[numeric_features][:5]
```

[ ]:

	Age	RestingBP	Cholesterol	Oldpeak	RestingBP	MaxHR	Oldpeak
0	40	140.0	289.0	0.0	140.0	172	0.0
1	49	160.0	180.0	1.0	160.0	156	1.0
2	37	130.0	283.0	0.0	130.0	98	0.0
3	48	138.0	214.0	1.5	138.0	108	1.5
4	54	150.0	195.0	0.0	150.0	122	0.0

[ ]:
```
data = X[numeric_features]
X_num = pd.DataFrame(data, columns=numeric_features)
X_num[:5]
```

[ ]:

	Age	RestingBP	Cholesterol	Oldpeak	RestingBP	MaxHR	Oldpeak
0	40	140.0	289.0	0.0	140.0	172	0.0
1	49	160.0	180.0	1.0	160.0	156	1.0
2	37	130.0	283.0	0.0	130.0	98	0.0
3	48	138.0	214.0	1.5	138.0	108	1.5
4	54	150.0	195.0	0.0	150.0	122	0.0

[ ]:
```
categorical_features = ["ChestPainType", "RestingECG", "ST_Slope"]
X[categorical_features][:5]
```

[ ]:

	ChestPainType	RestingECG	ST_Slope
0	ATA	Normal	Up
1	NAP	Normal	Flat
2	ATA	ST	Up
3	ASY	Normal	Flat
4	NAP	Normal	Up

[ ]:
```
binary_features = ["Sex", "FastingBS", "ExerciseAngina"]
X[binary_features][:5]
```

[ ]:

	Sex	FastingBS	ExerciseAngina
0	M	0	N
1	F	0	N
2	M	0	N
3	F	0	Y
4	M	0	N

[ ]:
```
from sklearn.preprocessing import OrdinalEncoder

binary_transformer = OrdinalEncoder()

data = binary_transformer.fit_transform(X[binary_features])
X_bin = pd.DataFrame(data, columns=binary_features)
X_bin[:5]
```

[ ]:

	Sex	FastingBS	ExerciseAngina
0	1.0	0.0	0.0
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	0.0	0.0	1.0
4	1.0	0.0	0.0

```
[ ]: from sklearn.preprocessing import OneHotEncoder

categorical_transformer = OneHotEncoder(handle_unknown="ignore")
data = categorical_transformer.fit_transform(X[categorical_features]).toarray()
cols = categorical_transformer.get_feature_names_out()
X_cat = pd.DataFrame(data, columns=cols)
X_cat[:5]
```

```
[ ]:    ChestPainType_ASY  ChestPainType_ATA  ChestPainType_NAP  ChestPainType_TA  \
0                0.0                1.0                0.0               0.0
1                0.0                0.0                1.0               0.0
2                0.0                1.0                0.0               0.0
3                1.0                0.0                0.0               0.0
4                0.0                0.0                1.0               0.0

   RestingECG_LVH  RestingECG_Normal  RestingECG_ST  ST_Slope_Down  \
0             0.0                1.0            0.0            0.0
1             0.0                1.0            0.0            0.0
2             0.0                0.0            1.0            0.0
3             0.0                1.0            0.0            0.0
4             0.0                1.0            0.0            0.0

   ST_Slope_Flat  ST_Slope_Up
0            0.0          1.0
1            1.0          0.0
2            0.0          1.0
3            1.0          0.0
4            0.0          1.0
```

```
[ ]: X_data = pd.concat([X_cat, X_bin, X_num], axis = 1)
X_data[:5]
```

```
[ ]:    ChestPainType_ASY  ChestPainType_ATA  ChestPainType_NAP  ChestPainType_TA  \
0                0.0                1.0                0.0               0.0
1                0.0                0.0                1.0               0.0
2                0.0                1.0                0.0               0.0
3                1.0                0.0                0.0               0.0
4                0.0                0.0                1.0               0.0

   RestingECG_LVH  RestingECG_Normal  RestingECG_ST  ST_Slope_Down  \
0             0.0                1.0            0.0            0.0
1             0.0                1.0            0.0            0.0
2             0.0                0.0            1.0            0.0
3             0.0                1.0            0.0            0.0
4             0.0                1.0            0.0            0.0

   ST_Slope_Flat  ST_Slope_Up  Sex  FastingBS  ExerciseAngina  Age  RestingBP  \
```

```
0             0.0           1.0  1.0        0.0              0.0   40      140.0
1             1.0           0.0  0.0        0.0              0.0   49      160.0
2             0.0           1.0  1.0        0.0              0.0   37      130.0
3             1.0           0.0  0.0        0.0              1.0   48      138.0
4             0.0           1.0  1.0        0.0              0.0   54      150.0

   Cholesterol  Oldpeak  RestingBP  MaxHR  Oldpeak
0        289.0      0.0      140.0    172      0.0
1        180.0      1.0      160.0    156      1.0
2        283.0      0.0      130.0     98      0.0
3        214.0      1.5      138.0    108      1.5
4        195.0      0.0      150.0    122      0.0
```

2.1 Feature Scaling

0.0.3 3. Train/Test Data Split

[ ]:
```python
from sklearn.model_selection import train_test_split

# sensitive_attributes = X_data[['Sex', 'AgeGroup']]
sensitive_attributes = X_data['Sex']

X_train, X_test, y_train, y_test, sa_train, sa_test = train_test_split(X_data,␣
↪y, sensitive_attributes, test_size=0.2, random_state=42)
```

0.0.4 4. Model Training

Decision Tree Classifier

[ ]:
```python
from  sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

cf = DecisionTreeClassifier(min_samples_leaf=10, max_depth=4) #parameters have␣
↪not been tuned
cf.fit(X_train, y_train)

cv_scores = cross_val_score(cf, X_train, y_train, cv=5)

# Output the cross-validation accuracy scores
print("Average Cross-Validation Accuracy:", cv_scores.mean())

y_train_pred = cf.predict(X_train)
y_test_pred = cf.predict(X_test)
```

```
Average Cross-Validation Accuracy: 0.8337713167458765
```

Fairlearn's MetricFrame is used to compute these metrics across different groups. It takes the true labels (y_test), the model predictions (y_pred), and the sensitive attributes (sa_test).

Overall Metrics: mf.overall provides the metrics computed over the entire dataset.

Metrics by Group: mf.by_group shows the performance of each metric for each subgroup defined by the sensitive attributes.

Disparities: Optionally, you can compute disparities in metrics between groups, which can help in identifying which groups are being treated unfairly by the model.

The selection rate is the rate at which positive outcomes (e.g., predicting heart disease) are assigned. It is particularly useful in fairness analysis to understand how often each group (defined by sensitive attributes like age and gender) is receiving a positive prediction.

[ ]:
```
from fairlearn.metrics import MetricFrame
from fairlearn.metrics import selection_rate
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

metrics = {
    'accuracy': accuracy_score,
    'precision': precision_score,
    'recall': recall_score,
    'f1': f1_score,
    'selection_rate': selection_rate
}

mf = MetricFrame(metrics, y_true=y_test, y_pred=y_test_pred,␣
↪sensitive_features=sa_test)

# Output overall metrics and metrics by group
print("Overall Metrics:\n", mf.overall)
print("\nMetrics by Group:\n", mf.by_group)

# Optionally, compute and print disparities
disparities = mf.by_group / mf.overall
print("\nDisparities:\n", disparities)
```

```
Overall Metrics:
 accuracy          0.815217
precision         0.892473
recall            0.775701
f1                0.830000
selection_rate    0.505435
dtype: float64

Metrics by Group:
      accuracy  precision    recall        f1  selection_rate
Sex
0.0  0.851852   0.750000  0.750000  0.750000        0.296296
1.0  0.808917   0.905882  0.777778  0.836957        0.541401
```

```
Disparities:
      accuracy  precision    recall        f1  selection_rate
Sex
0.0   1.044938   0.840361  0.966867  0.903614        0.586221
1.0   0.992272   1.015025  1.002677  1.008381        1.071160
```

```
/home/phway/anaconda3/lib/python3.9/site-
packages/fairlearn/metrics/_metric_frame.py:77: FutureWarning:

You have provided 'metrics' as positional arguments. Please pass them as keyword
arguments. From version 0.10.0 passing them as positional arguments will result
in an error.
```

Overall Model Performance Metrics

- Accuracy (85.87%): The model correctly predicts heart disease status in about 86 out of 100 cases.
- Precision (88.57%): When the model predicts heart disease, there's about an 89% chance that the patient actually has heart disease.
- Recall (86.92%): The model correctly identifies about 87% of all actual cases of heart disease.
- F1 Score (87.74%): Indicates a good balance between precision and recall.
- Selection Rate (57.07%): About 57% of the predictions made by the model are positive for heart disease.

Metrics by Gender Group

Female (Sex = 0.0)

Accuracy: 74.07% (lower than the overall accuracy) Precision: 55.56% (significantly lower than the overall precision) Recall: 62.50% (lower than the overall recall) F1 Score: 58.82% (lower than the overall F1 score)

Male (Sex = 1.0)

Accuracy: 87.90% (slightly higher than the overall accuracy) Precision: 91.67% (higher than the overall precision) Recall: 88.89% (slightly higher than the overall recall) F1 Score: 90.26% (higher than the overall F1 score)

Disparities

Female (Sex = 0.0)

All metrics are below 1, indicating underperformance for females across all measured aspects.

Male (Sex = 1.0)

All metrics exceed 1, indicating that the model performs better for males across all measures.

Demographic Parity: It measures the ML model's ability to make prediction such that they are independent of the influence by sensitive groups.

Equalized odds: It also ensures that ML model's predictions are independent of sensitive groups. It's more strict than Demographic parity by ensuring all groups in the dataset have same true positive rates and false positive rates.

```
[ ]: from fairlearn.metrics import demographic_parity_difference,␣
     ↪equalized_odds_difference, demographic_parity_ratio, equalized_odds_ratio

     dpd = demographic_parity_difference(y_test, y_test_pred, sensitive_features =␣
     ↪sa_test)
     eod = equalized_odds_difference(y_test, y_test_pred, sensitive_features =␣
     ↪sa_test)
     dpr = demographic_parity_ratio(y_test, y_test_pred, sensitive_features =␣
     ↪sa_test)
     eqr = equalized_odds_ratio(y_test, y_test_pred, sensitive_features=sa_test)

     print("Demographic parity difference: {}".format(round(dpd, 2)))
     print("Equalized odds difference: {}".format(round(eod, 2)))
     print("Demographic parity ratio: {}".format(round(dpr, 2)))
     print(f'Value of equal odds ratio: {round(eqr, 2)}')
```

```
Demographic parity difference: 0.25
Equalized odds difference: 0.03
Demographic parity ratio: 0.55
Value of equal odds ratio: 0.76
```

1) Demographic difference in parity: if the absolute value is less than 0.1, then the model can be considered fair.
2) Balanced odds difference: if the absolute value is less than 0.1, then the model can be considered fair.
3) The difference is in equal opportunities: if the absolute value is less than 0.1, then the model can be considered fair.
4) Demographic parity coefficient: the fairness of this indicator ranges from 0.8 to 1.25.

In our case, the difference in demographic parity (0.21), demographic parity ratio (0.64)indicate unfairness, while the difference in equalization of chances (0.05) does not indicate unfairness.

Bias mitigation technique using Fairlearn's ExponentiatedGradient with the Demographic Parity constraint.

```
[ ]: from fairlearn.reductions import ExponentiatedGradient, DemographicParity,␣
     ↪EqualizedOdds

     classifier = DecisionTreeClassifier(min_samples_leaf=10, max_depth=4)
     dp = DemographicParity()
     dt_classifier_reduction = ExponentiatedGradient(classifier, dp)

     dt_classifier_reduction.fit(X_train, y_train, sensitive_features=sa_train)
```

```
prediction_dp = dt_classifier_reduction.predict(X_test)
```

Bias mitigation technique using Fairlearn's ExponentiatedGradient with the EqualizedOdds constraint.

```
[ ]: from fairlearn.reductions import ExponentiatedGradient, EqualizedOdds
from sklearn.tree import DecisionTreeClassifier

# Use EqualizedOdds instead of DemographicParity
eo = EqualizedOdds()

classifier = DecisionTreeClassifier(min_samples_leaf=10, max_depth=4)

# Apply ExponentiatedGradient with EqualizedOdds
eg_classifier_eo = ExponentiatedGradient(classifier, eo)
eg_classifier_eo.fit(X_train, y_train, sensitive_features=sa_train)
prediction_eo = eg_classifier_eo.predict(X_test)
```

Fairness Metrics after bias mitigation

```
[ ]: from fairlearn.metrics import MetricFrame
from fairlearn.metrics import selection_rate
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

metrics = {
    'accuracy': accuracy_score,
    'precision': precision_score,
    'recall': recall_score,
    'f1': f1_score,
    'selection_rate': selection_rate
}

mf = MetricFrame(metrics, y_true=y_test, y_pred=prediction_dp,␣
↪sensitive_features=sa_test)


print("Fairness metrics by demographic parity\n")
# Output overall metrics and metrics by group
print("Overall Metrics:\n", mf.overall)
print("\nMetrics by Group:\n", mf.by_group)

# Optionally, compute and print disparities
disparities = mf.by_group / mf.overall
print("\nDisparities:\n", disparities)
```

```
Fairness metrics by demographic parity

Overall Metrics:
 accuracy          0.777174
precision         0.789474
recall            0.841121
f1                0.814480
selection_rate    0.619565
dtype: float64

Metrics by Group:
      accuracy  precision    recall        f1  selection_rate
Sex
0.0   0.444444   0.347826  1.000000  0.516129        0.851852
1.0   0.834395   0.901099  0.828283  0.863158        0.579618

Disparities:
      accuracy  precision    recall        f1  selection_rate
Sex
0.0   0.571873   0.440580  1.188889  0.633692        1.374919
1.0   1.073627   1.141392  0.984736  1.059766        0.935524
```

```
/home/phway/anaconda3/lib/python3.9/site-
packages/fairlearn/metrics/_metric_frame.py:77: FutureWarning:

You have provided 'metrics' as positional arguments. Please pass them as keyword
arguments. From version 0.10.0 passing them as positional arguments will result
in an error.
```

[ ]:
```python
from fairlearn.metrics import MetricFrame
from fairlearn.metrics import selection_rate
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

metrics = {
    'accuracy': accuracy_score,
    'precision': precision_score,
    'recall': recall_score,
    'f1': f1_score,
    'selection_rate': selection_rate
}

mf = MetricFrame(metrics, y_true=y_test, y_pred=prediction_eo,␣
 ↪sensitive_features=sa_test)


print("Fairness metrics by equalized odds\n")
```

```python
# Output overall metrics and metrics by group
print("Overall Metrics:\n", mf.overall)
print("\nMetrics by Group:\n", mf.by_group)

# Optionally, compute and print disparities
disparities = mf.by_group / mf.overall
print("\nDisparities:\n", disparities)
```

```
Fairness metrics by equalized odds

Overall Metrics:
 accuracy          0.842391
precision         0.897959
recall            0.822430
f1                0.858537
selection_rate    0.532609
dtype: float64

Metrics by Group:
      accuracy  precision    recall        f1  selection_rate
Sex
0.0  0.703704   0.500000  0.750000  0.600000        0.444444
1.0  0.866242   0.953488  0.828283  0.886486        0.547771

Disparities:
      accuracy  precision    recall        f1  selection_rate
Sex
0.0  0.835364   0.556818  0.911932  0.698864        0.834467
1.0  1.028313   1.061839  1.007117  1.032555        1.028467
```

```
/home/phway/anaconda3/lib/python3.9/site-
packages/fairlearn/metrics/_metric_frame.py:77: FutureWarning:

You have provided 'metrics' as positional arguments. Please pass them as keyword
arguments. From version 0.10.0 passing them as positional arguments will result
in an error.
```

[ ]:
```python
dpd = demographic_parity_difference(y_test, prediction_dp, sensitive_features =␣
 ↪sa_test)
eod = equalized_odds_difference(y_test, prediction_dp, sensitive_features =␣
 ↪sa_test)
dpr = demographic_parity_ratio(y_test, prediction_dp, sensitive_features =␣
 ↪sa_test)

print("Demographic parity difference: {}".format(round(dpd, 2)))
print("Equalized odds difference: {}".format(round(eod, 2)))
```

```
print("Demographic parity ratio: {}".format(round(dpr, 2)))
```

```
Demographic parity difference: 0.27
Equalized odds difference: 0.63
Demographic parity ratio: 0.68
```

```
[ ]: dpd = demographic_parity_difference(y_test, prediction_eo, sensitive_features =␣
     ↪sa_test)
     eod = equalized_odds_difference(y_test, prediction_eo, sensitive_features =␣
     ↪sa_test)
     dpr = demographic_parity_ratio(y_test, prediction_eo, sensitive_features =␣
     ↪sa_test)

     print("Demographic parity difference: {}".format(round(dpd, 2)))
     print("Equalized odds difference: {}".format(round(eod, 2)))
     print("Demographic parity ratio: {}".format(round(dpr, 2)))
```

```
Demographic parity difference: 0.1
Equalized odds difference: 0.25
Demographic parity ratio: 0.81
```

1) Demographic difference in parity: if the absolute value is less than 0.1, then the model can be considered fair.
2) Balanced odds difference: if the absolute value is less than 0.1, then the model can be considered fair.
3) The difference is in equal opportunities: if the absolute value is less than 0.1, then the model can be considered fair.
4) Demographic parity coefficient: the fairness of this indicator ranges from 0.8 to 1.25.

In our case, the difference in demographic parity (0.01), demographic parity ratio (0.99) does not indicate unfairness, while the difference in equalization of chances (0.34) indicates unfairness.

Impact of Mitigation: The application of the Demographic Parity constraint has resulted in a more balanced selection rate between genders. However, this has come with a trade-off in terms of accuracy and precision, especially for females.

Trade-off Between Fairness and Performance: The mitigation strategy seems to have prioritized demographic parity (equal selection rates), leading to a decrease in other performance metrics for females. This is a common challenge in fairness interventions, where improving fairness in one aspect can lead to decreases in performance metrics.

Fairness Considerations: Despite the trade-offs, the model is now less biased in terms of selection rate, which was one of the initial fairness concerns.

Improvement in Fairness: The intervention based on Equalized Odds has led to a reduction in both demographic and equalized odds disparities, indicating a more fair model.