

# ULL In-processing Mitigation

January 13, 2025

## 1 Install

```
[ ]: !pip install aif360  
      !pip install fairlearn
```

Collecting aif360

Downloading aif360-0.6.1-py3-none-any.whl.metadata (5.0 kB)

Requirement already satisfied: numpy>=1.16 in /usr/local/lib/python3.10/dist-packages (from aif360) (1.26.4)

Requirement already satisfied: scipy>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from aif360) (1.13.1)

Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.10/dist-packages (from aif360) (2.2.2)

Requirement already satisfied: scikit-learn>=1.0 in /usr/local/lib/python3.10/dist-packages (from aif360) (1.6.0)

Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from aif360) (3.8.0)

Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->aif360) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->aif360) (2024.2)

Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->aif360) (2024.2)

Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0->aif360) (1.4.2)

Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0->aif360) (3.5.0)

Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->aif360) (1.3.1)

Requirement already satisfied: cycycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->aif360) (0.12.1)

Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->aif360) (4.55.3)

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->aif360) (1.4.7)

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->aif360) (24.2)

Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-

```

packages (from matplotlib->aif360) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->aif360) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas>=0.24.0->aif360) (1.17.0)
Downloading aif360-0.6.1-py3-none-any.whl (259 kB)
      0.0/259.7 kB
? eta -:--:--
      259.7/259.7 kB
15.4 MB/s eta 0:00:00
Installing collected packages: aif360
Successfully installed aif360-0.6.1
Collecting fairlearn
  Downloading fairlearn-0.12.0-py3-none-any.whl.metadata (7.0 kB)
Requirement already satisfied: numpy>=1.24.4 in /usr/local/lib/python3.10/dist-
packages (from fairlearn) (1.26.4)
Requirement already satisfied: pandas>=2.0.3 in /usr/local/lib/python3.10/dist-
packages (from fairlearn) (2.2.2)
Requirement already satisfied: scikit-learn>=1.2.1 in
/usr/local/lib/python3.10/dist-packages (from fairlearn) (1.6.0)
Requirement already satisfied: scipy>=1.9.3 in /usr/local/lib/python3.10/dist-
packages (from fairlearn) (1.13.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0.3->fairlearn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=2.0.3->fairlearn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=2.0.3->fairlearn) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=1.2.1->fairlearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2.1->fairlearn)
(3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas>=2.0.3->fairlearn) (1.17.0)
Downloading fairlearn-0.12.0-py3-none-any.whl (240 kB)
      240.0/240.0 kB
15.5 MB/s eta 0:00:00
Installing collected packages: fairlearn
Successfully installed fairlearn-0.12.0

```

## 2 Imports

```
[ ]: import os
import copy
import math
import json
```

```

import collections

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_validate,
↳StratifiedKfold
from sklearn.metrics import get_scorer
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

from aif360.algorithms.inprocessing import MetaFairClassifier,
↳PrejudiceRemover, GerryFairClassifier
from fairlearn import metrics

```

```

/usr/local/lib/python3.10/dist-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.17.3 and <1.25.0 is required for this version of SciPy
(detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
WARNING:root:No module named 'inFairness': SenSeI and SenSR will be unavailable.
To install, run:
pip install 'aif360[inFairness]'

```

### 3 Settings

```
[ ]: final_df_filename = "final_df_v1"
```

```
[ ]: sensitive_feature = "s_gender"
      favorable_sensitive_label = "MALE"
      unfavorable_sensitive_label = "FEMALE"

      class_feature = "level_MAT"
      favorable_class_label = "PASSED"
      unfavorable_class_label = "NOT PASSED"

      perf_metrics = ["balanced_accuracy", "precision", "recall", "roc_auc", "f1"]
      fair_metrics = ["demographic_parity_ratio", "equalized_odds_ratio"]

```

## 4 Utils

```
[ ]: from sklearn.base import BaseEstimator, ClassifierMixin
      from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
      from aif360.datasets import BinaryLabelDataset
      from aif360.metrics import BinaryLabelDatasetMetric

[ ]: class AIF360SklearnWrapper(BaseEstimator, ClassifierMixin):
      def __init__(
          self,
          aif360_model,
          sensitive_feature,
          favorable_class_label,
          unfavorable_class_label,
          favorable_sens_group,
      ):
          self.aif360_model = aif360_model

          self.sensitive_feature = sensitive_feature
          self.favorable_sens_group = favorable_sens_group

          self.class_feature = "label"
          self.favorable_class_label = favorable_class_label
          self.unfavorable_class_label = unfavorable_class_label

          self.ordinal_encoder = None

      def _encode_features(self, df, fit):

          df_encoded = df.copy()

          # Encode class
          if self.class_feature in df_encoded.columns:
              df_encoded[self.class_feature] = df_encoded[
                  self.class_feature
              ].apply(lambda x: 1 if x == self.favorable_class_label else 0)
          else:
              df_encoded[self.class_feature] = 0 # or any constant value

          # Encode sensitive feature
          if self.sensitive_feature in df_encoded.
↪select_dtypes(include=['object', 'category']).columns:
              df_encoded[self.sensitive_feature] = df_encoded[
                  self.sensitive_feature
              ].apply(lambda x: 1 if x == self.favorable_sens_group else 0)
```

```

        # # Encode other columns
        # categorical_cols = df_encoded.select_dtypes(include=['object',
↪ 'category']).columns
        # numeric_cols = df_encoded.select_dtypes(exclude=['object',
↪ 'category']).columns

        # if fit:
        #     self.ordinal_encoder = OrdinalEncoder()
        #     self.ordinal_encoder.fit(df_encoded[categorical_cols])

        # if len(categorical_cols) > 0:
        #     df_encoded[categorical_cols] = self.ordinal_encoder.
↪ transform(df_encoded[categorical_cols])
        # if len(numeric_cols) > 0:
        #     df_encoded[numeric_cols] = df_encoded[numeric_cols]

        # # Check missing values
        # print(df_encoded.isna().sum())
        # df_encoded = df_encoded.set_index(self.sensitive_feature)
        # print(df_encoded.isna().sum())

    return df_encoded

def fit(self, X, y, **kwargs):
    if not self.sensitive_feature:
        raise ValueError("Protected attribute name must be specified.")

    df_encoded = self._encode_features(df=X.assign(label=y), fit=True)
    # print(df_encoded)
    dataset = BinaryLabelDataset(
        df=df_encoded,
        label_names=[self.class_feature],
        # sensitive_features=[self.sensitive_feature],
        protected_attribute_names=[self.sensitive_feature],
        favorable_label=1,
        unfavorable_label=0
    )
    self.aif360_model = self.aif360_model.fit(dataset, **kwargs)
    return self

def predict(self, X, decode=False):
    X_encoded = self._encode_features(df=X, fit=False)
    dataset = BinaryLabelDataset(
        df=X_encoded,
        label_names=[self.class_feature],
        # sensitive_features=[self.sensitive_feature],
        protected_attribute_names=[self.sensitive_feature],

```

```

        favorable_label=1,
        unfavorable_label=0
    )
    predictions = self.aif360_model.predict(dataset).
    ↪convert_to_dataframe()[0][self.class_feature]
    predictions = predictions.apply(lambda x: self.favorable_class_label if
    ↪x == 1 else self.unfavorable_class_label) if decode else predictions
    return predictions

def predict_proba(self, X):
    if hasattr(self.aif360_model, "predict_proba"):
        X_encoded = self._encode_features(df=X, fit=False)
        dataset = BinaryLabelDataset(
            df=X_encoded,
            label_names=[self.class_feature],
            # sensitive_features=[self.sensitive_feature],
            protected_attribute_names=[self.sensitive_feature],
            favorable_label=1,
            unfavorable_label=0
        )
        probabilities = self.aif360_model.predict_proba(dataset)
        return probabilities
    else:
        raise NotImplementedError("The wrapped model does not support
    ↪probability prediction.")

def score(self, X, y):
    y_pred = self.predict(X)
    return (y_pred == y).mean()

```

```

[ ]: def encode_feature(df, feature, favorable_label):

    df_encoded = df.copy()

    # Encode class
    if feature in df_encoded.columns:
        df_encoded[feature] = df_encoded[
            feature
        ].apply(lambda x: 1 if x == favorable_label else 0)
    return df_encoded

```

```

[ ]: def decode_feature(df, feature, favorable_label, unfavorable_label):

    df_decoded = df.copy()

    # Encode class
    if feature in df_decoded.columns:

```

```

df_decoded[feature] = df_decoded[
    feature
].apply(lambda x: favorable_label if x == 1 else unfavorable_label)
return df_decoded

```

```

[ ]: def compute_fair_metric(
    fair_metric_name, sensitive_feature, X, y_true, y_pred
):
    # metrics_module = __import__("metrics")
    metrics_module = globals()["metrics"]
    fair_metric_scorer = getattr(metrics_module, fair_metric_name)

    X_sensitive = encode_feature(X, sensitive_feature,
    ↪favorable_label=favorable_sensitive_label)
    X_sensitive = X_sensitive[sensitive_feature]

    return fair_metric_scorer(
        y_true=y_true,
        y_pred=y_pred,
        sensitive_features=X_sensitive,
    )

```

## 5 Src

```

[ ]: final_df = pd.read_csv(f"{final_df_filename}.csv").set_index("id_questionnaire")
final_df

```

```

[ ]:
s_frequency_of_work_with_teachers \
id_questionnaire
1          0.733333
3          0.833333
5          0.566667
6          0.533333
8          0.633333
...          ...
20421      0.566667
20422      0.700000
20423      0.400000
20424      0.500000
20427      0.666667

s_frequency_of_evaluations  s_frequency_of_internet_usage \
id_questionnaire
1          0.740741          0.066667
3          0.750000          0.666667

```

5	0.888889	0.133333
6	0.851852	0.133333
8	0.703704	0.266667
...	...	...
20421	0.777778	0.600000
20422	0.518519	0.266667
20423	0.555556	0.400000
20424	0.555556	0.200000
20427	0.740741	0.133333

	s_frequency_of_materials_in_class	f_ESCS \
id_questionnaire		
1	0.380952	0.235340
3	1.000000	0.261451
5	0.666667	3.151773
6	0.380952	-1.627731
8	0.666667	-0.358498
...	...	...
20421	0.476190	-0.764891
20422	0.380952	-0.048524
20423	0.571429	0.072922
20424	0.523810	2.755881
20427	0.476190	1.372627

	s_extent_of_classmates_affinity \
id_questionnaire	
1	0.714286
3	0.833333
5	0.952381
6	0.761905
8	0.857143
...	...
20421	0.904762
20422	0.952381
20423	0.809524
20424	0.904762
20427	0.857143

	s_extent_of_teacher_performance	s_extent_of_class_env \
id_questionnaire		
1	0.833333	0.791667
3	0.800000	0.833333
5	1.000000	1.000000
6	0.866667	1.000000
8	0.966667	0.833333
...	...	...
20421	1.000000	1.000000

20422	0.888889	0.916667
20423	1.000000	0.666667
20424	0.766667	0.791667
20427	0.900000	0.916667

	f_mother_age	s_frequency_of_computer_usage \
id_questionnaire		
1	43.0	0.111111
3	45.0	0.666667
5	39.0	0.000000
6	25.0	0.222222
8	37.0	0.333333
...	...	...
20421	41.0	0.111111
20422	36.0	0.222222
20423	46.0	0.111111
20424	41.0	0.111111
20427	37.0	0.000000

	f_number_of_homework_hours_a_week \
id_questionnaire	
1	14.0
3	9.0
5	6.0
6	5.0
8	10.0
...	...
20421	14.0
20422	2.0
20423	8.0
20424	2.0
20427	1.0

	s_extent_of_school_satisfaction	f_mother_education_level \
id_questionnaire		
1	0.944444	7.0
3	0.888889	3.0
5	1.000000	9.0
6	1.000000	3.0
8	0.833333	7.0
...	...	...
20421	1.000000	5.0
20422	0.944444	4.0
20423	1.000000	3.0
20424	0.833333	9.0
20427	1.000000	5.0

	f_extent_of_family_satisfaction	f_number_of_tech_at_home \
id_questionnaire		
1	0.700000	3.0
3	0.291667	4.0
5	0.833333	7.0
6	0.833333	3.0
8	1.000000	3.0
...	...	...
20421	1.000000	4.0
20422	0.708333	5.0
20423	0.833333	10.0
20424	0.666667	10.0
20427	0.291667	5.0

	f_frequency_of_books_at_home \
id_questionnaire	
1	1.000000
3	0.444444
5	0.555556
6	0.333333
8	0.555556
...	...
20421	0.222222
20422	0.555556
20423	0.444444
20424	0.555556
20427	0.444444

	f_frequency_of_parent_involved_in_school_activities \
id_questionnaire	
1	0.333333
3	0.166667
5	0.250000
6	0.083333
8	0.333333
...	...
20421	0.333333
20422	0.000000
20423	0.000000
20424	0.416667
20427	0.000000

	s_gender	level_MAT
id_questionnaire		
1	FEMALE	3.0
3	FEMALE	1.0
5	FEMALE	4.0

6	FEMALE	1.0
8	MALE	2.0
...	...	...
20421	MALE	2.0
20422	FEMALE	3.0
20423	MALE	2.0
20424	MALE	3.0
20427	FEMALE	2.0

[10735 rows x 19 columns]

```
[ ]: get_features = lambda curr_df: [col for col in curr_df.columns if not col.
    ↳startswith("level_")]
```

```
X, y = final_df[get_features(final_df)], final_df[class_feature]
```

```
y = y.apply(lambda elem: favorable_class_label if elem >= 3.5 else_
    ↳unfavorable_class_label)
```

```
[ ]: skf = StratifiedKFold(n_splits=5)
```

```
# Prepare list for 1) and 2)
```

```
df_records = [] # list of dicts; one dict per fold-metric
```

```
# Prepare dict for 3)
```

```
# Initialize a structure to hold lists of metric results for each group.
```

```
# groupwise_results[group][metric] = list of fold results
```

```
groups_in_data = X[sensitive_feature].unique().tolist()
```

```
groupwise_results = {
```

```
    group: {m: [] for m in perf_metrics} # We typically do group-wise for perf_
    ↳metrics
```

```
    for group in groups_in_data
```

```
}
```

```
# Prepare lists for 4)
```

```
all_labels = [] # final labels (either y or y_pred)
```

```
all_groups = [] # group membership (e.g., "MALE"/"FEMALE")
```

```
all_folds = [] # which fold
```

```
# Prepare dict for 5)
```

```
groupwise_confusions = {grp: [] for grp in groups_in_data}
```

```
# Evaluation loop
```

```
for fold_idx, (train_index, test_index) in enumerate(skf.split(X, y)):
```

```
    X_train = X.iloc[train_index, :]
```

```

X_test = X.iloc[test_index, :]
y_train = y.iloc[train_index]
y_test = y.iloc[test_index]

mitigator = MetaFairClassifier(sensitive_attr=sensitive_feature)
# mitigator = GerryFairClassifier()
# mitigator = PrejudiceRemover(sensitive_attr=sensitive_feature,
↪class_attr="label")
wrapper = AIF360SklernWrapper(
    aif360_model=mitigator,
    sensitive_feature=sensitive_feature,
    favorable_sens_group=favorable_sensitive_label,
    favorable_class_label=favorable_class_label,
    unfavorable_class_label=unfavorable_class_label,
)

# Fit the model
wrapper.fit(X_train, y_train)

# Predict on test
y_pred = wrapper.predict(X_test, decode=True)
y_pred.index = X_test.index

# Encode predictions and ground truth
y_pred_encoded = y_pred.apply(lambda x: 1 if x == favorable_class_label
↪else 0)
y_pred_encoded.index = X_test.index # align them explicitly if needed
y_test_encoded = y_test.apply(lambda x: 1 if x == favorable_class_label
↪else 0)
y_test_encoded.index = X_test.index # Should already match because we
↪didn't reset

# 1) Overall performance metrics
for pm in perf_metrics:
    pm_value = get_scorer(pm)._score_func(y_test_encoded, y_pred_encoded)
    df_records.append({
        "fold": fold_idx,
        "metric_type": "performance", # helps us separate them later
        "metric": pm,
        "value": pm_value
    })

# 2) Overall fairness metrics
for fm in fair_metrics:
    fm_value = compute_fair_metric(
        fair_metric_name=fm,

```

```

        sensitive_feature=sensitive_feature,
        X=X_test,
        y_true=y_test_encoded,
        y_pred=y_pred_encoded
    )
    df_records.append({
        "fold": fold_idx,
        "metric_type": "fairness",
        "metric": fm,
        "value": fm_value
    })

# 3) Groupwise performance
for grp in groups_in_data:
    # Create a boolean mask to filter X_test to just this group
    group_mask = (X_test[sensitive_feature] == grp)

    # Slice the true/pred labels for this group
    y_true_group = y_test_encoded[group_mask]
    y_pred_group = y_pred_encoded[group_mask]

    # Compute each performance metric
    for pm in perf_metrics:
        val = get_scorer(pm)._score_func(y_true_group, y_pred_group)
        groupwise_results[grp][pm].append(val)

# 4) Create a DataFrame for plotting barcharts
for idx in X_test.index:
    group_label = X_test.loc[idx, sensitive_feature]

    label_value = y_test[idx]

    all_labels.append(label_value)
    all_groups.append(group_label)
    all_folds.append(fold_idx)

# 5) Confusion Matrix
for grp in groups_in_data:
    group_mask = (X_test[sensitive_feature] == grp)
    y_true_grp = y_test[group_mask]
    y_pred_grp = y_pred[group_mask]

    cm = confusion_matrix(y_true_grp, y_pred_grp,
↪ labels=[unfavorable_class_label, favorable_class_label])
    # cm is a 2x2 array: [[TN, FP], [FN, TP]]

```

```

        groupwise_confusions[grp].append(cm)

df_dist = pd.DataFrame({
    "fold": all_folds,
    "group": all_groups,
    "label": all_labels
})

```

```

[ ]: df_metrics = pd.DataFrame(df_records)

df_metrics["metric"] = df_metrics["metric"].apply(lambda x: x.replace("_", " ").
↳title())

df_perf = df_metrics[df_metrics["metric_type"] == "performance"]

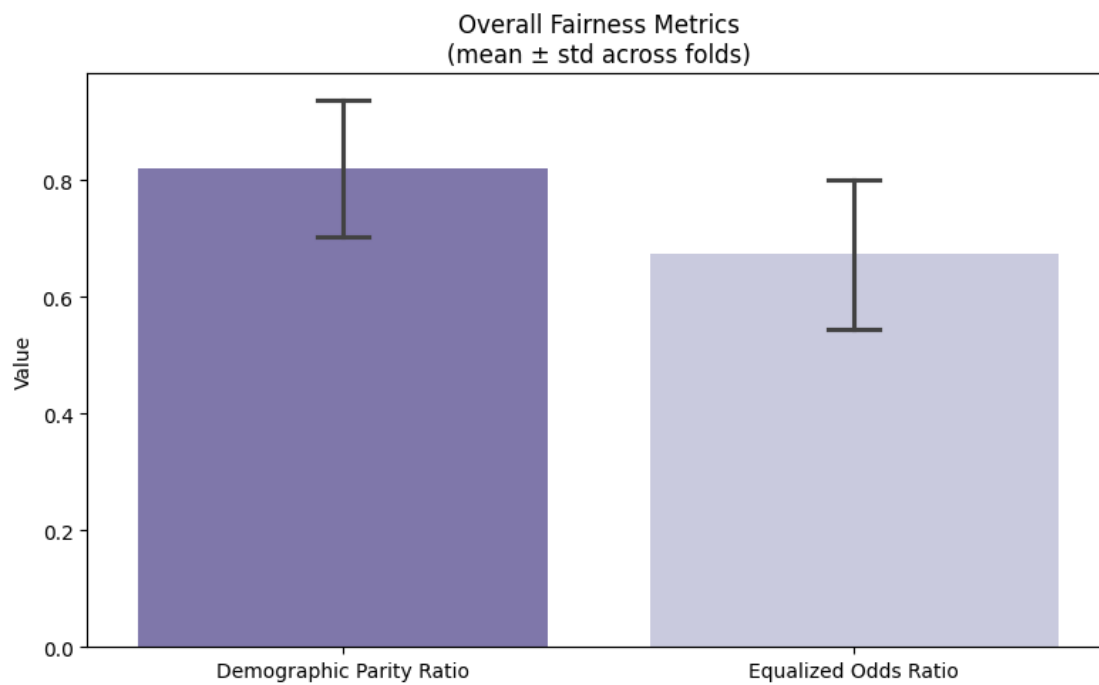
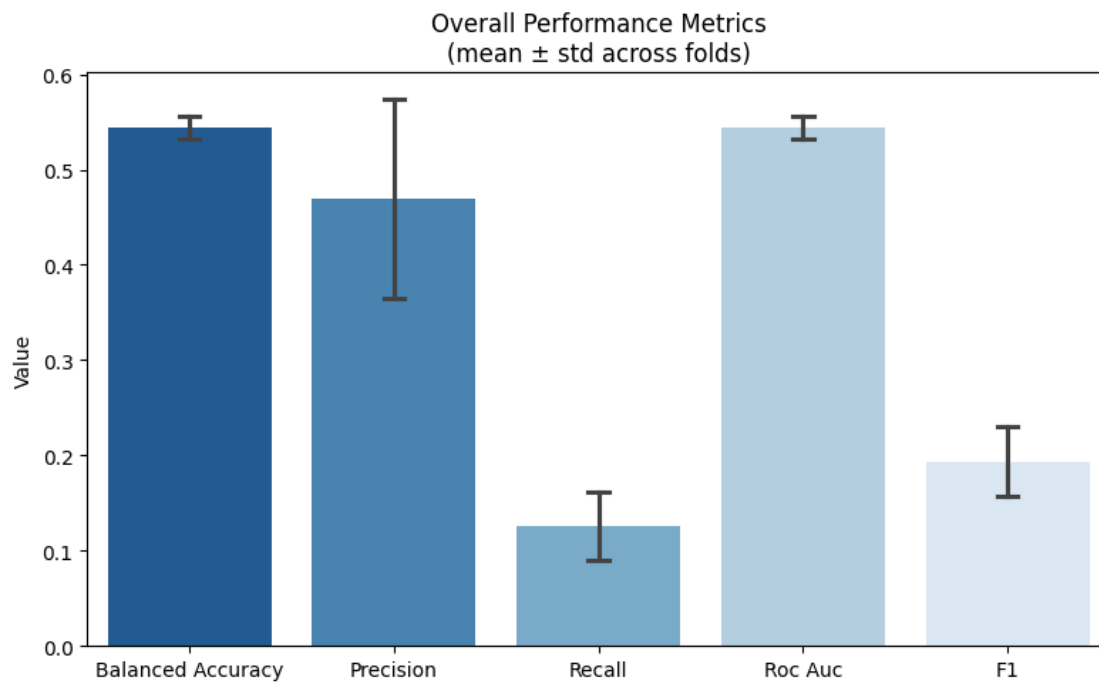
plt.figure(figsize=(8, 5))
sns.barplot(
    data=df_perf,
    x="metric",
    y="value",
    hue="metric",
    errorbar="sd",    # show mean ± 1 std (Seaborn 0.12+). For older Seaborn,↳
↳use ci="sd".
    capsize=.1,
    palette="Blues_r"
)
plt.title("Overall Performance Metrics\n(mean ± std across folds)")
plt.xlabel("")
plt.ylabel("Value")
# plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

df_fair = df_metrics[df_metrics["metric_type"] == "fairness"]

plt.figure(figsize=(8, 5))
sns.barplot(
    data=df_fair,
    x="metric",
    y="value",
    hue="metric",
    errorbar="sd",
    capsize=.1,
    palette="Purples_r"
)
plt.title("Overall Fairness Metrics\n(mean ± std across folds)")
plt.xlabel("")

```

```
plt.ylabel("Value")
# plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



```

[ ]: data_for_plot = []

for group, metrics_dict in groupwise_results.items():
    for metric, values_list in metrics_dict.items():
        # values_list is something like [0.75, 0.78, 0.73, 0.76, 0.74] (one for
        ↪ each fold)
        for val in values_list:
            data_for_plot.append({
                "group": group,
                "metric": metric.replace("_", " ").title(),
                "value": val
            })

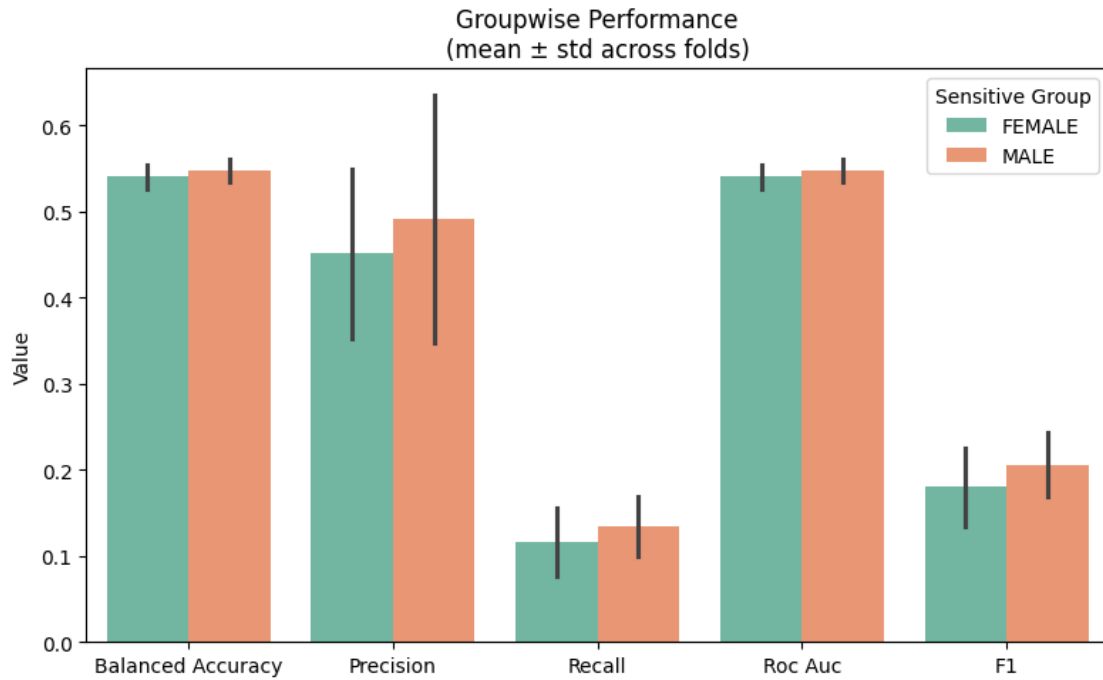
data_for_plot = pd.DataFrame(data_for_plot)

plt.figure(figsize=(8, 5))

sns.barplot(
    data=data_for_plot,
    x="metric",
    y="value",
    hue="group",
    errorbar="sd",      # to display ± 1 std
    palette="Set2"
)

plt.title("Groupwise Performance\n(mean ± std across folds)")
plt.xlabel("")
plt.ylabel("Value")
plt.legend(title="Sensitive Group")
# plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

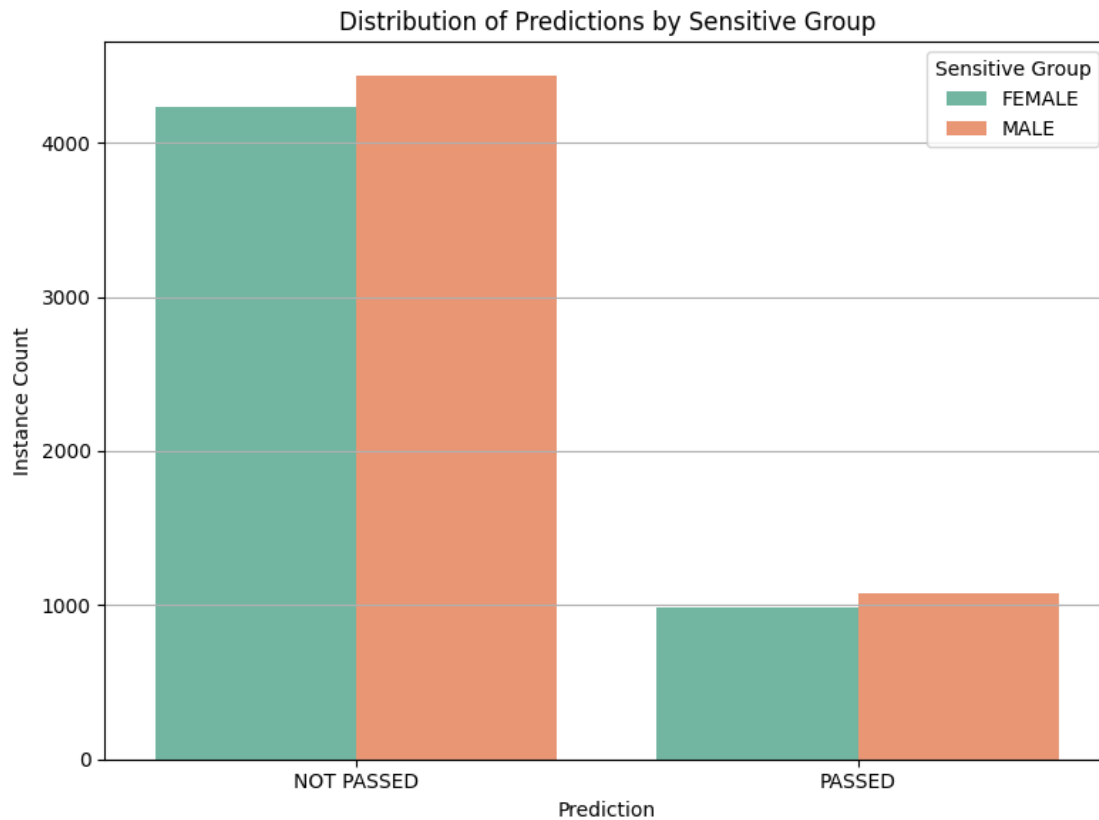
```



```
[ ]: plt.figure(figsize=(8, 6))

sns.countplot(
    data=df_dist,
    x="label",
    hue="group",
    palette="Set2"
)

plt.title("Distribution of Predictions by Sensitive Group")
plt.xlabel("Prediction")
plt.ylabel("Instance Count")
plt.legend(title="Sensitive Group")
plt.grid(axis="y")
plt.tight_layout()
plt.show()
```



```
[ ]: for grp, cm_list in groupwise_confusions.items():
    # Sum all the confusion matrices across folds
    sum_cm = np.sum(cm_list, axis=0)

    # Alternatively, you could compute the average: np.mean(cm_list, axis=0)

    disp = ConfusionMatrixDisplay(confusion_matrix=sum_cm,
    ↪display_labels=[unfavorable_class_label, favorable_class_label])
    disp.plot(values_format="d", cmap="Blues")
    plt.title(f"Confusion Matrix (Sum of folds) - Group: {grp}")
    plt.show()
```

