week7

July 9, 2025

[4]:
```python
import zipfile
import os

# Set the path to your uploaded zip file
zip_path = 'archive-2.zip'  # <- change if your filename is different

# Extract the contents
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall('.')  # Extract to current directory

# List the extracted files
print("Extracted files:")
print(os.listdir('.'))
```

```
Extracted files:
['.ipynb_checkpoints', 'cox-violent-parsed_filt.csv', 'week7.ipynb', 'cox-
violent-parsed.csv', 'roughwork.ipynb', 'archive-2.zip', 'compas-scores-
raw.csv', 'propublicaCompassRecividism_data_fairml.csv', 'Racial.ipynb']
```

[5]:
```python
import os

os.listdir('propublicaCompassRecividism_data_fairml.csv')
```

[5]: ['._propublica_data_for_fairml.csv', 'propublica_data_for_fairml.csv']

[7]:
```python
import pandas as pd

df1 = pd.read_csv('cox-violent-parsed.csv')  # May be filtered or focused on␣
 ↪violent offenses
df2 = pd.read_csv('compas-scores-raw.csv')   # Possibly original raw data
df3 = pd.read_csv('propublicaCompassRecividism_data_fairml.csv/
 ↪propublica_data_for_fairml.csv')
# FairML-ready version

# Preview the shapes
print(df1.shape, df2.shape, df3.shape)
```

```
(18316, 52) (60843, 28) (6172, 12)
```

[8]:
```python
print("df1 columns:", df1.columns.tolist())
print("df2 columns:", df2.columns.tolist())
print("df3 columns:", df3.columns.tolist())
```

```
df1 columns: ['id', 'name', 'first', 'last', 'compas_screening_date', 'sex',
'dob', 'age', 'age_cat', 'race', 'juv_fel_count', 'decile_score',
'juv_misd_count', 'juv_other_count', 'priors_count', 'days_b_screening_arrest',
'c_jail_in', 'c_jail_out', 'c_case_number', 'c_offense_date', 'c_arrest_date',
'c_days_from_compas', 'c_charge_degree', 'c_charge_desc', 'is_recid',
'r_case_number', 'r_charge_degree', 'r_days_from_arrest', 'r_offense_date',
'r_charge_desc', 'r_jail_in', 'r_jail_out', 'violent_recid', 'is_violent_recid',
'vr_case_number', 'vr_charge_degree', 'vr_offense_date', 'vr_charge_desc',
'type_of_assessment', 'decile_score.1', 'score_text', 'screening_date',
'v_type_of_assessment', 'v_decile_score', 'v_score_text', 'v_screening_date',
'in_custody', 'out_custody', 'priors_count.1', 'start', 'end', 'event']
df2 columns: ['Person_ID', 'AssessmentID', 'Case_ID', 'Agency_Text', 'LastName',
'FirstName', 'MiddleName', 'Sex_Code_Text', 'Ethnic_Code_Text', 'DateOfBirth',
'ScaleSet_ID', 'ScaleSet', 'AssessmentReason', 'Language', 'LegalStatus',
'CustodyStatus', 'MaritalStatus', 'Screening_Date', 'RecSupervisionLevel',
'RecSupervisionLevelText', 'Scale_ID', 'DisplayText', 'RawScore', 'DecileScore',
'ScoreText', 'AssessmentType', 'IsCompleted', 'IsDeleted']
df3 columns: ['Two_yr_Recidivism', 'Number_of_Priors', 'score_factor',
'Age_Above_FourtyFive', 'Age_Below_TwentyFive', 'African_American', 'Asian',
'Hispanic', 'Native_American', 'Other', 'Female', 'Misdemeanor']
```

[19]:
```python
# Filter invalid rows per ProPublica methodology
df = df1[
    (df1['days_b_screening_arrest'] <= 30) &
    (df1['days_b_screening_arrest'] >= -30) &
    (df1['is_recid'] != -1) &
    (df1['c_charge_degree'] != 'O') &
    (df1['score_text'] != 'N/A')
].copy()
```

[20]:
```python
# Prediction: High risk = 1
df['predicted'] = df['score_text'].apply(lambda x: 1 if x == 'High' else 0)

# Outcome: Recidivated = 1
df['actual'] = df['is_recid']
```

[21]:
```python
df['actual'] = df['is_recid']
```

[22]:
```python
# If your model predictions are stored in a different column (e.g.,␣
↪'predicted_score' or 'decile_score'),
# you might need to binarize it. Here's an example using decile_score:
df['predicted'] = (df['decile_score'] >= 5).astype(int)
```

[25]:

```python
import numpy as np
import pandas as pd

def compute_fpr_by_group(dataset, protected_attr='Ethnic_Code_Text'):
    # Extract labels, predictions, and protected attributes
    y_true = dataset.labels.ravel()
    y_pred = dataset.scores.ravel()  # AIF360 uses `scores` for predicted labels
    group = dataset.protected_attributes.ravel()

    # Unique protected groups (e.g., ethnic codes)
    unique_groups = np.unique(group)
    fpr_values = {}

    for g in unique_groups:
        idx = group == g
        fp = np.sum((y_pred[idx] == 1) & (y_true[idx] == 0))
        tn = np.sum((y_pred[idx] == 0) & (y_true[idx] == 0))
        fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
        fpr_values[str(g)] = fpr

    return pd.Series(fpr_values, name='False Positive Rate')
```

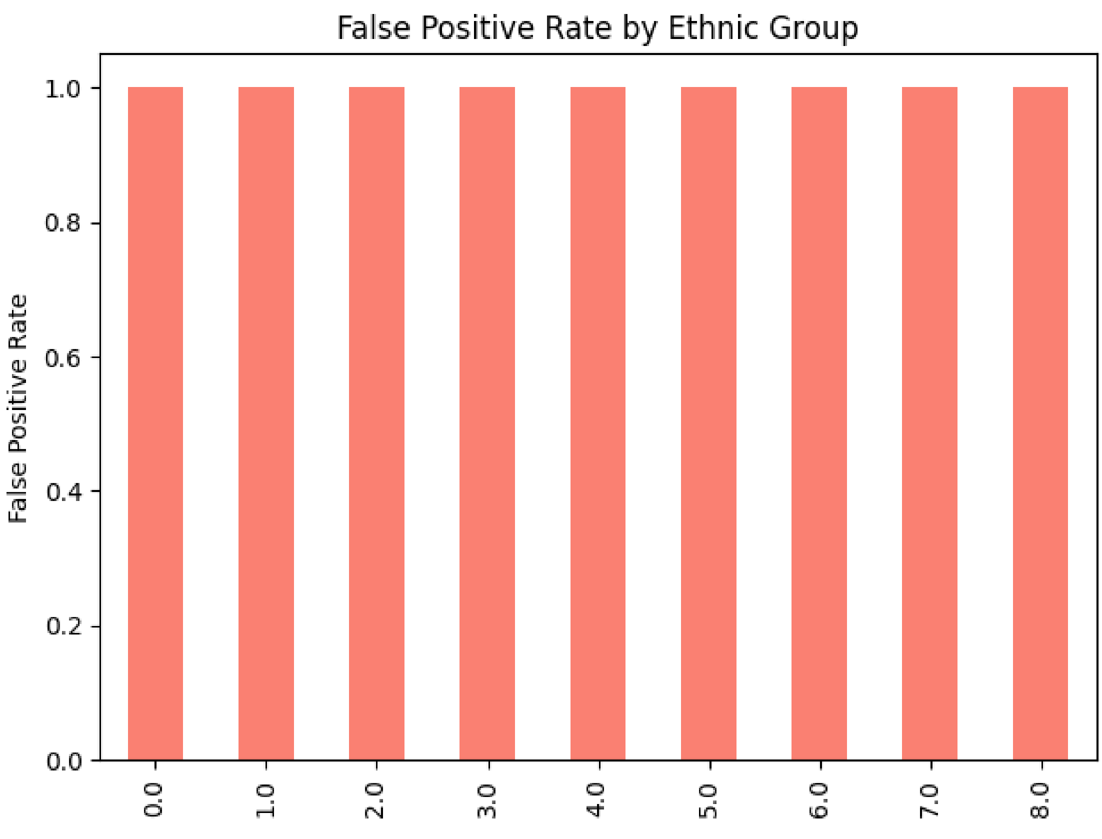
[27]:

```python
# Define the conversion function
from aif360.datasets import BinaryLabelDataset
from sklearn.preprocessing import LabelEncoder

def to_aif360_df2(df):
    df = df.copy()

    # Create label column: DecileScore >= 5
    df['label'] = (df['DecileScore'] >= 5).astype(int)
    label_col = 'label'
    protected_col = 'Ethnic_Code_Text'

    # Drop rows with missing label or protected attribute
    df = df.dropna(subset=[label_col, protected_col])

    # Encode non-numeric columns except label and protected
    for col in df.columns:
        if df[col].dtype == 'object' and col not in [label_col, protected_col]:
            df[col] = LabelEncoder().fit_transform(df[col])

    # Encode protected attribute if it's still object
    if df[protected_col].dtype == 'object':
        df[protected_col] = LabelEncoder().fit_transform(df[protected_col])

    return BinaryLabelDataset(
```

```
        df=df,
        label_names=[label_col],
        protected_attribute_names=[protected_col],
        favorable_label=0,
        unfavorable_label=1
    )
```

[28]:
```
aif2 = to_aif360_df2(df2)
```

[29]:
```
fpr_by_race = compute_fpr_by_group(aif2, protected_attr='Ethnic_Code_Text')

# Plot
fpr_by_race.plot(kind='bar', color='salmon', title='False Positive Rate by␣
 ↪Ethnic Group')
plt.ylabel('False Positive Rate')
plt.tight_layout()
plt.show()
```



[30]:
```
from sklearn.metrics import confusion_matrix

def false_positive_rate(group):
```

```python
    cm = confusion_matrix(group['actual'], group['predicted'], labels=[0, 1])
    if cm.shape != (2, 2):
        return float('nan')
    tn, fp, fn, tp = cm.ravel()
    return fp / (fp + tn) if (fp + tn) > 0 else float('nan')

# Apply FPR calculation
fpr_by_race = df.groupby('race', group_keys=False).apply(false_positive_rate)

# Ensure result is a Series for sorting
if isinstance(fpr_by_race, pd.DataFrame):
    # Flatten to Series if needed
    fpr_by_race = fpr_by_race.iloc[:, 0]

# Now sort descending
fpr_by_race = fpr_by_race.sort_values(ascending=False)

# Show result
print(fpr_by_race)
```

```
race
African-American    0.531355
Caucasian           0.287527
Native American     0.285714
Hispanic            0.230882
Asian               0.175000
Other               0.174946
dtype: float64
```

[31]:
```python
# Example cleaning
df1 = df1.dropna()
df2 = df2.dropna()
df3 = df3.dropna()
```

[18]:
```python
!pip install aif360
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: aif360 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(0.6.1)
Requirement already satisfied: numpy>=1.16 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from aif360) (1.26.4)
Requirement already satisfied: scipy>=1.2.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from aif360) (1.12.0)
```

```
Requirement already satisfied: pandas>=0.24.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from aif360) (1.5.3)
Requirement already satisfied: scikit-learn>=1.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from aif360) (1.4.2)
Requirement already satisfied: matplotlib in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from aif360) (3.10.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from pandas>=0.24.0->aif360) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from pandas>=0.24.0->aif360) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from scikit-learn>=1.0->aif360) (1.5.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from scikit-learn>=1.0->aif360) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (1.3.2)
Requirement already satisfied: cycler>=0.10 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (4.58.5)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (25.0)
Requirement already satisfied: pillow>=8 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from matplotlib->aif360) (3.2.3)
Requirement already satisfied: six>=1.5 in
/home/d5082b60-f89f-485e-a681-03544b128c47/.local/lib/python3.11/site-packages
(from python-dateutil>=2.8.1->pandas>=0.24.0->aif360) (1.17.0)
```

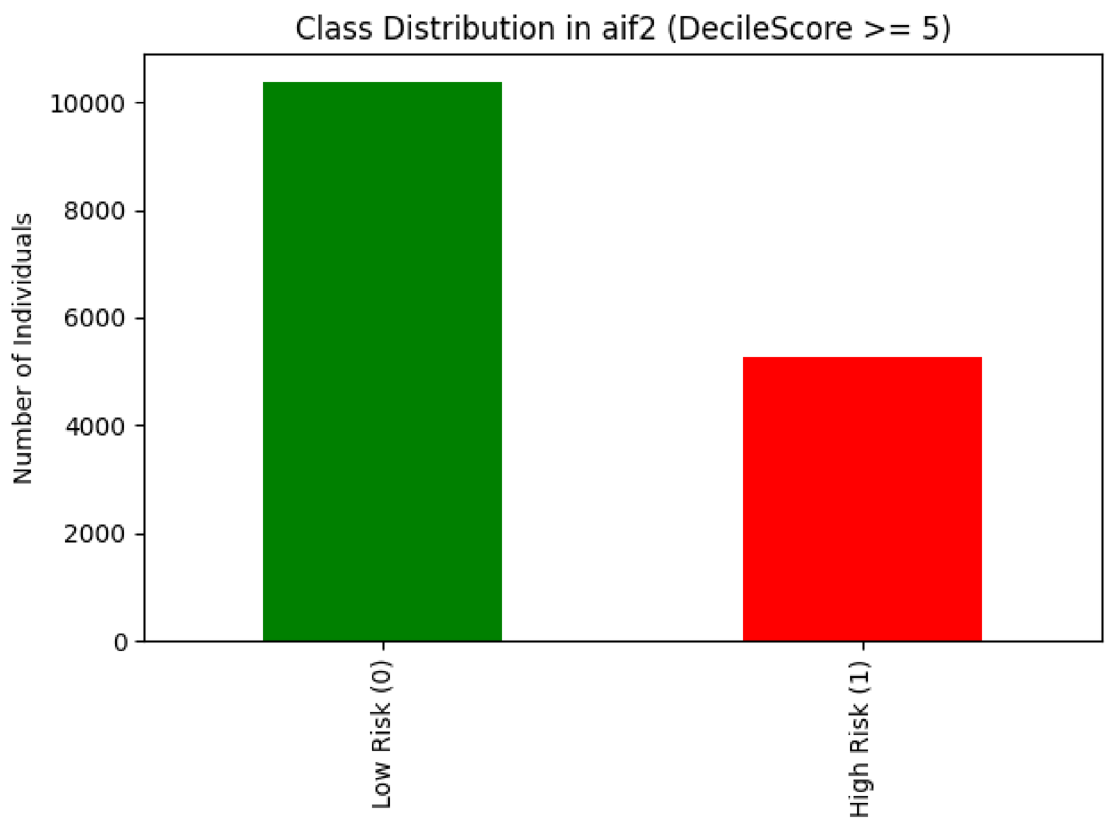
[32]:
```
# Step 1: Ensure the required label and protected attribute columns exist
df2['is_recid'] = (df2['DecileScore'] >= 5).astype(int)
df2['race'] = df2['Ethnic_Code_Text']

# Step 2: Keep only numeric columns + required ones
keep_cols = ['is_recid', 'race']
numeric_cols = df2.select_dtypes(include='number').columns.tolist()
df2_cleaned = df2[keep_cols + [col for col in numeric_cols if col not in␣
 ↪keep_cols]].copy()
```

[33]:
```
race_cols = ['African_American', 'Asian', 'Hispanic', 'Native_American',␣
 ↪'Other']
df3['race'] = df3[race_cols].idxmax(axis=1)  # Sets race to column with 1
df3['is_recid'] = df3['Two_yr_Recidivism']
```

[34]:
```
from aif360.datasets import BinaryLabelDataset
from sklearn.preprocessing import LabelEncoder

def to_aif360_df2(df):
    df = df.copy()

    # Define new label column for df2
    df['label'] = (df['DecileScore'] >= 5).astype(int)
    label_col = 'label'
    protected_col = 'Ethnic_Code_Text'

    # Drop rows with missing label or protected attribute
    df = df.dropna(subset=[label_col, protected_col])

    # Encode non-numeric columns (except label and protected)
    for col in df.columns:
        if df[col].dtype == 'object' and col not in [label_col, protected_col]:
            df[col] = LabelEncoder().fit_transform(df[col])

    # Encode protected attribute (Ethnic_Code_Text) if it's not numeric
    if df[protected_col].dtype == 'object':
        df[protected_col] = LabelEncoder().fit_transform(df[protected_col])

    return BinaryLabelDataset(
        df=df,
        label_names=[label_col],
        protected_attribute_names=[protected_col],
        favorable_label=0,   # Low risk
        unfavorable_label=1  # High risk
    )
```

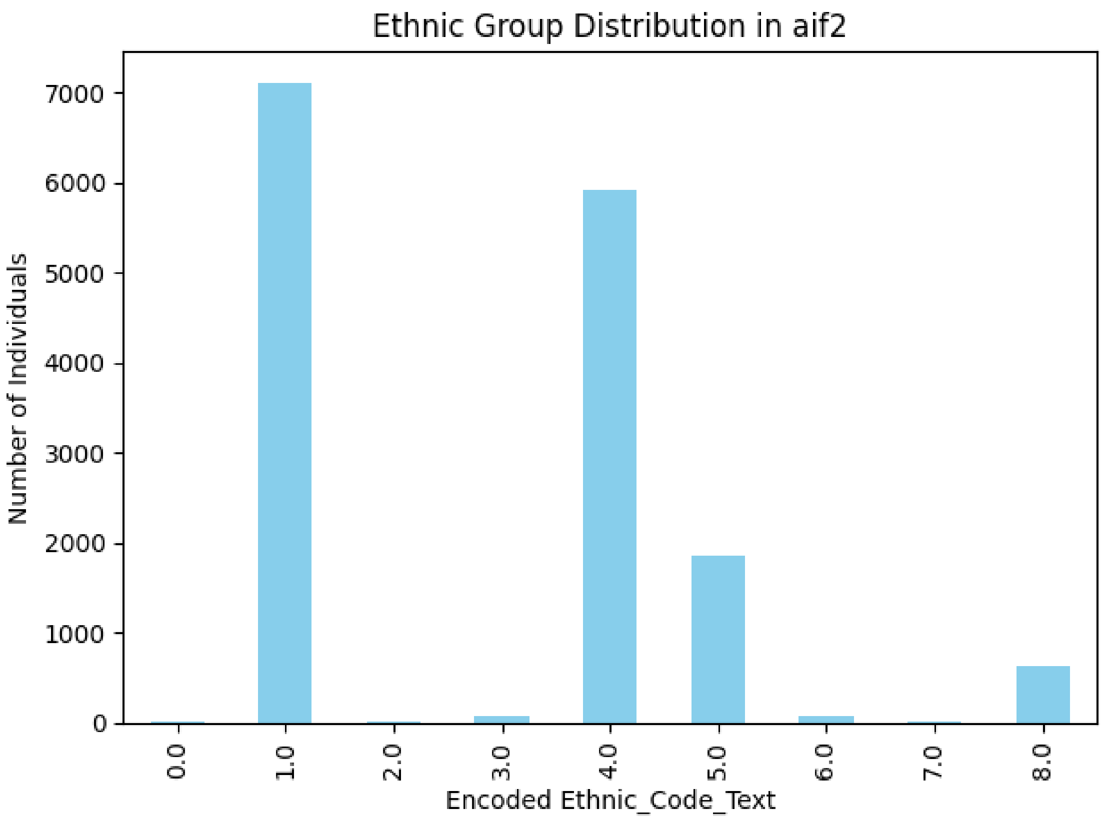
[36]:
```
aif2 = to_aif360_df2(df2)
```

[37]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Count favorable (0) vs unfavorable (1) labels
label_counts = pd.Series(aif2.labels.ravel()).value_counts().sort_index()
label_counts.index = ['Low Risk (0)', 'High Risk (1)']

# Plot
label_counts.plot(kind='bar', color=['green', 'red'], title='Class Distribution␣
↪in aif2 (DecileScore >= 5)')
plt.ylabel('Number of Individuals')
plt.tight_layout()
plt.show()
```



[38]:
```
# Count each ethnic group
ethnic_counts = pd.Series(aif2.protected_attributes.ravel()).value_counts().
↪sort_index()
ethnic_counts.plot(kind='bar', color='skyblue', title='Ethnic Group␣
↪Distribution in aif2')
plt.xlabel('Encoded Ethnic_Code_Text')
```

```
plt.ylabel('Number of Individuals')
plt.tight_layout()
plt.show()
```



[39]:
```
df2['Ethnic_Code_Text'].value_counts()
```

[39]:
```
African-American    7099
Caucasian           5907
Hispanic            1856
Other                618
Asian                 72
Native American       63
Oriental              12
African-Am             6
Arabic                 6
Name: Ethnic_Code_Text, dtype: int64
```

[41]:
```
from aif360.metrics import BinaryLabelDatasetMetric, ClassificationMetric

def compute_fairness_metrics(aif_dataset, privileged_vals=[1],␣
 ↪unprivileged_vals=[0]):
```

```
    dataset_metric = BinaryLabelDatasetMetric(
        aif_dataset,
        privileged_groups=[{aif_dataset.protected_attribute_names[0]: v} for v␣
↪in privileged_vals],
        unprivileged_groups=[{aif_dataset.protected_attribute_names[0]: v} for␣
↪v in unprivileged_vals]
    )

    classified_metric = ClassificationMetric(
        aif_dataset,
        aif_dataset,
        privileged_groups=[{aif_dataset.protected_attribute_names[0]: v} for v␣
↪in privileged_vals],
        unprivileged_groups=[{aif_dataset.protected_attribute_names[0]: v} for␣
↪v in unprivileged_vals]
    )

    return {
        "Statistical Parity Difference": dataset_metric.
↪statistical_parity_difference(),
        "Disparate Impact": dataset_metric.disparate_impact(),
        "Equal Opportunity Difference": classified_metric.
↪equal_opportunity_difference(),
        "Average Odds Difference": classified_metric.average_odds_difference(),
        "False Positive Rate Difference": classified_metric.
↪false_positive_rate_difference(),
        "False Negative Rate Difference": classified_metric.
↪false_negative_rate_difference(),
        "Accuracy": classified_metric.accuracy(),
        "Balanced Accuracy": classified_metric.balanced_accuracy(),
        "TPR (Unpriv)": classified_metric.true_positive_rate(privileged=False),
        "TPR (Priv)": classified_metric.true_positive_rate(privileged=True),
    }
```

[42]:
```
# For df1 (race)
print(df1['race'].dropna().unique())

# For df2 (Ethnic_Code_Text)
print(df2['Ethnic_Code_Text'].dropna().unique())

# For df3 (reconstructed race from one-hot columns)
print(df3['race'].dropna().unique())
```

```
[]
['African-American' 'Other' 'Caucasian' 'Asian' 'Hispanic'
 'Native American' 'African-Am' 'Oriental' 'Arabic']
['Other' 'African_American' 'Hispanic' 'Asian' 'Native_American']
```

[44]:
```
from aif360.datasets import BinaryLabelDataset
from sklearn.preprocessing import LabelEncoder

def to_aif360_df1(df):
    df = df.copy()
    label_col = 'is_recid'
    protected_col = 'race'

    df = df.dropna(subset=[label_col, protected_col])

    # Encode non-numeric columns
    for col in df.columns:
        if df[col].dtype == 'object' and col not in [label_col, protected_col]:
            df[col] = LabelEncoder().fit_transform(df[col])

    if df[protected_col].dtype == 'object':
        df[protected_col] = LabelEncoder().fit_transform(df[protected_col])

    return BinaryLabelDataset(
        df=df,
        label_names=[label_col],
        protected_attribute_names=[protected_col],
        favorable_label=0,
        unfavorable_label=1
    )
```

[45]:
```
aif1 = to_aif360_df1(df1)
```

[50]:
```
aif2 = to_aif360_df2(df2)
aif3 = to_aif360_df3(df3)  # You'll need to define `to_aif360_df3` if not done␣
 ↪yet
```

[47]:
```
from aif360.datasets import BinaryLabelDataset
from sklearn.preprocessing import LabelEncoder

def to_aif360_df3(df):
    df = df.copy()

    # Make sure label and protected attribute are present
    label_col = 'is_recid'
    protected_col = 'race'

    # Drop rows missing race or label
    df = df.dropna(subset=[label_col, protected_col])

    # Encode all non-numeric columns except label and protected
    for col in df.columns:
```

```
        if df[col].dtype == 'object' and col not in [label_col, protected_col]:
            df[col] = LabelEncoder().fit_transform(df[col])

    # Encode protected attribute if still string
    if df[protected_col].dtype == 'object':
        df[protected_col] = LabelEncoder().fit_transform(df[protected_col])

    return BinaryLabelDataset(
        df=df,
        label_names=[label_col],
        protected_attribute_names=[protected_col],
        favorable_label=0,
        unfavorable_label=1
    )
```

[48]:
```
aif3 = to_aif360_df3(df3)
```

[95]:
```
import numpy as np
from aif360.metrics import BinaryLabelDatasetMetric, ClassificationMetric

def compute_fairness_metrics(aif_dataset, privileged_vals=[1],␣
 ↪unprivileged_vals=[0]):
    # Define group filters
    priv_groups = [{aif_dataset.protected_attribute_names[0]: v} for v in␣
 ↪privileged_vals]
    unpriv_groups = [{aif_dataset.protected_attribute_names[0]: v} for v in␣
 ↪unprivileged_vals]

    # Dataset-level metrics
    dataset_metric = BinaryLabelDatasetMetric(
        aif_dataset,
        privileged_groups=priv_groups,
        unprivileged_groups=unpriv_groups
    )

    # Classification metrics (same dataset used as predicted, i.e. baseline)
    classified_metric = ClassificationMetric(
        aif_dataset,
        aif_dataset,
        privileged_groups=priv_groups,
        unprivileged_groups=unpriv_groups
    )

    # Compute TPR and TNR to calculate balanced accuracy manually
    tpr = classified_metric.true_positive_rate()
    tnr = classified_metric.true_negative_rate()
```

```python
    balanced_acc = 0.5 * (tpr + tnr) if not np.isnan(tpr) and not np.isnan(tnr)␣
↪else np.nan

    # Compile metrics
    metrics = {
        "Statistical Parity Diff": dataset_metric.
↪statistical_parity_difference(),
        "Disparate Impact": dataset_metric.disparate_impact(),
        "Equal Opportunity Diff": classified_metric.
↪equal_opportunity_difference(),
        "Average Odds Diff": classified_metric.average_odds_difference(),
        "FPR Diff": classified_metric.false_positive_rate_difference(),
        "FNR Diff": classified_metric.false_negative_rate_difference(),
        "Accuracy": classified_metric.accuracy(),
        "Balanced Accuracy": balanced_acc,
        "TPR (Unpriv)": classified_metric.true_positive_rate(privileged=False),
        "TPR (Priv)": classified_metric.true_positive_rate(privileged=True),
    }

    # Warn if any are NaN
    for k, v in metrics.items():
        if np.isnan(v):
            print(f"  Warning: {k} is NaN - likely due to zero positives/
↪negatives in one group.")

    return metrics
```

[97]:
```python
metrics_1 = compute_fairness_metrics(aif1, privileged_vals=[1],␣
↪unprivileged_vals=[0])
metrics_2 = compute_fairness_metrics(aif2, privileged_vals=[1],␣
↪unprivileged_vals=[0])
metrics_3 = compute_fairness_metrics(aif3, privileged_vals=[1],␣
↪unprivileged_vals=[0])
```

```
  Warning: Statistical Parity Diff is NaN - likely due to zero
positives/negatives in one group.
  Warning: Disparate Impact is NaN - likely due to zero positives/negatives in
one group.
  Warning: Equal Opportunity Diff is NaN - likely due to zero
positives/negatives in one group.
  Warning: Average Odds Diff is NaN - likely due to zero positives/negatives in
one group.
  Warning: FPR Diff is NaN - likely due to zero positives/negatives in one
group.
  Warning: FNR Diff is NaN - likely due to zero positives/negatives in one
group.
  Warning: Balanced Accuracy is NaN - likely due to zero positives/negatives in
```

one group.
  Warning: TPR (Unpriv) is NaN - likely due to zero positives/negatives in one
group.
  Warning: TPR (Priv) is NaN - likely due to zero positives/negatives in one
group.

[55]:
```
def label_distribution_by_group(aif_data):
    import pandas as pd

    df = pd.DataFrame({
        'label': aif_data.labels.ravel(),
        'protected': aif_data.protected_attributes.ravel()
    })

    group_counts = df.groupby(['protected', 'label']).size().
 ↪unstack(fill_value=0)

    # Rename columns if both 0 and 1 labels exist
    if list(group_counts.columns) == [0, 1]:
        group_counts.columns = ['Low Risk (0)', 'High Risk (1)']
    elif list(group_counts.columns) == [1]:
        group_counts.columns = ['High Risk (1) Only']
    elif list(group_counts.columns) == [0]:
        group_counts.columns = ['Low Risk (0) Only']

    return group_counts
```

[56]:
```
print("df1 label distribution by race:")
print(label_distribution_by_group(aif1))

print("\ndf2 label distribution by Ethnic_Code_Text:")
print(label_distribution_by_group(aif2))

print("\ndf3 label distribution by race:")
print(label_distribution_by_group(aif3))
```

```
df1 label distribution by race:
Empty DataFrame
Columns: []
Index: []

df2 label distribution by Ethnic_Code_Text:
           Low Risk (0)  High Risk (1)
protected
0.0                   1              5
1.0                3883           3216
2.0                   5              1
3.0                  59             13
```

```
4.0                  4400           1507
5.0                  1446            410
6.0                    35             28
7.0                     8              4
8.0                   531             87

df3 label distribution by race:
           Low Risk (0)  High Risk (1)
protected
0.0                  2795           2483
1.0                    23              8
2.0                   320            189
3.0                     6              5
4.0                   219            124
```

[57]:
```python
# Keep ethnic groups with >=100 samples and both labels
df2_valid = df2[df2['Ethnic_Code_Text'].isin([1.0, 4.0, 5.0])]
aif2 = to_aif360_df2(df2_valid)
```

[93]:
```python
metrics_2 = compute_fairness_metrics(aif2, privileged_vals=[1.0],␣
 ↪unprivileged_vals=[4.0])
```

[59]:
```python
print(df1[['race', 'is_recid']].isna().sum())
```

```
race        0.0
is_recid    0.0
dtype: float64
```

[60]:
```python
df1_clean = df1.dropna(subset=['race', 'is_recid'])
print(df1_clean['race'].value_counts())
print(df1_clean['is_recid'].value_counts())
```

```
Series([], Name: race, dtype: int64)
Series([], Name: is_recid, dtype: int64)
```

[61]:
```python
aif1 = to_aif360_df1(df1_clean)
```

[98]:
```python
metrics_3 = compute_fairness_metrics(aif3, privileged_vals=[0.0],␣
 ↪unprivileged_vals=[2.0])  # example
```

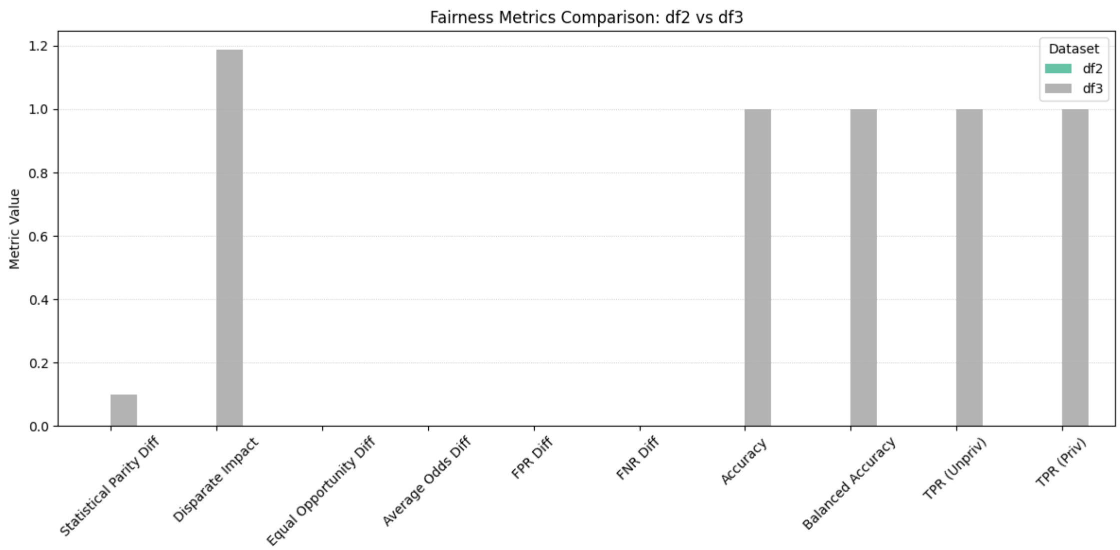
[99]:
```python
# Filter df2 to major ethnic groups with enough data
df2_filtered = df2[df2['Ethnic_Code_Text'].isin([1.0, 4.0, 5.0])]
aif2 = to_aif360_df2(df2_filtered)
```

[100]:
```python
aif3_filtered = aif3.copy()
```

[68]:
```python
import pandas as pd

comparison_df = pd.DataFrame({
    'df2': metrics_2,
    'df3': metrics_3
}).T.round(4)

display(comparison_df)
```

```
     Statistical Parity Diff  Disparate Impact  Equal Opportunity Diff  \
df2                      NaN               NaN                     NaN
df3                   0.0991            1.1872                     0.0

     Average Odds Diff  FPR Diff  FNR Diff  Accuracy  Balanced Accuracy  \
df2                NaN       NaN       NaN       0.0                NaN
df3                0.0       0.0       0.0       1.0                1.0

     TPR (Unpriv)  TPR (Priv)
df2           NaN         NaN
df3           1.0         1.0
```

[69]:
```python
import matplotlib.pyplot as plt

# Transpose for plotting
comparison_df.T.plot(kind='bar', figsize=(12, 6), colormap='Set2')
plt.title("Fairness Metrics Comparison: df2 vs df3")
plt.ylabel("Metric Value")
plt.xticks(rotation=45)
plt.axhline(y=0, color='black', linestyle='--', linewidth=0.7)
plt.grid(axis='y', linestyle=':', linewidth=0.5)
plt.tight_layout()
plt.legend(title="Dataset")
plt.show()
```



[73]:
```
def get_qualified_groups(aif_data, min_count=10):
    import pandas as pd

    df = pd.DataFrame({
        'label': aif_data.labels.ravel(),
        'protected': aif_data.protected_attributes.ravel()
    })

    # Count label values per group
    group_counts = df.groupby(['protected', 'label']).size().
    ↪unstack(fill_value=0)

    # Make sure column names are always strings
    group_counts.columns = group_counts.columns.astype(str)

    # Check that both '0' and '1' labels exist
    if '0' not in group_counts.columns or '1' not in group_counts.columns:
        print(" One of the label classes (0 or 1) is missing in this dataset.")
        return []

    # Only keep groups with enough of both labels
    qualified = group_counts[(group_counts['0'] >= min_count) &␣
    ↪(group_counts['1'] >= min_count)]

    return qualified.index.tolist()
```

[74]:
```
print("Valid groups in df2:", get_qualified_groups(aif2))
print("Valid groups in df3:", get_qualified_groups(aif3))
```

```
 One of the label classes (0 or 1) is missing in this dataset.
Valid groups in df2: []
 One of the label classes (0 or 1) is missing in this dataset.
Valid groups in df3: []
```

[75]:
```
def check_label_balance(aif_data):
    import pandas as pd
    df = pd.DataFrame({'label': aif_data.labels.ravel()})
    return df['label'].value_counts()

print("aif2 label balance:")
print(check_label_balance(aif2))

print("\naif3 label balance:")
print(check_label_balance(aif3))
```

```
aif2 label balance:
Series([], Name: label, dtype: int64)

aif3 label balance:
0.0    3363
1.0    2809
Name: label, dtype: int64
```

[76]:
```
df2[['DecileScore', 'Ethnic_Code_Text']].dropna().shape
df2['DecileScore'].value_counts()
```

[76]:
```
1     4545
2     2295
3     2163
4     1365
5     1248
6     1144
7      904
8      799
9      706
10     470
Name: DecileScore, dtype: int64
```

[78]:
```
print("df2 columns:", df2.columns.tolist())
```

```
df2 columns: ['Person_ID', 'AssessmentID', 'Case_ID', 'Agency_Text', 'LastName',
'FirstName', 'MiddleName', 'Sex_Code_Text', 'Ethnic_Code_Text', 'DateOfBirth',
'ScaleSet_ID', 'ScaleSet', 'AssessmentReason', 'Language', 'LegalStatus',
'CustodyStatus', 'MaritalStatus', 'Screening_Date', 'RecSupervisionLevel',
'RecSupervisionLevelText', 'Scale_ID', 'DisplayText', 'RawScore', 'DecileScore',
'ScoreText', 'AssessmentType', 'IsCompleted', 'IsDeleted', 'is_recid', 'race']
```

[79]:
```python
# Create binary label column: 1 = high risk, 0 = low risk
df2['label'] = (df2['DecileScore'] >= 7).astype(int)

# Show label distribution
print(df2['label'].value_counts())
```

```
0    12760
1     2879
Name: label, dtype: int64
```

[81]:
```python
from aif360.datasets import BinaryLabelDataset
from sklearn.preprocessing import LabelEncoder

def to_aif360_df2_fixed(df):
    df = df.copy()

    # Create binary label: High risk (1) if DecileScore   7, else Low risk (0)
    df['label'] = (df['DecileScore'] >= 7).astype(int)

    # Drop rows missing label or protected attribute
    df = df.dropna(subset=['label', 'Ethnic_Code_Text'])

    # Encode non-numeric columns (except label + protected attr)
    for col in df.columns:
        if df[col].dtype == 'object' and col not in ['label',␣
 ↪'Ethnic_Code_Text']:
            df[col] = LabelEncoder().fit_transform(df[col].astype(str))

    # Encode protected attribute
    if df['Ethnic_Code_Text'].dtype == 'object':
        df['Ethnic_Code_Text'] = LabelEncoder().
 ↪fit_transform(df['Ethnic_Code_Text'].astype(str))

    return BinaryLabelDataset(
        df=df,
        label_names=['label'],
        protected_attribute_names=['Ethnic_Code_Text'],
        favorable_label=0,   # Low risk
        unfavorable_label=1  # High risk
    )
```

[82]:
```python
aif2 = to_aif360_df2_fixed(df2)

print("aif2 shape:", aif2.features.shape)
print("Label balance in aif2:")
print(check_label_balance(aif2))
```

```
print("Valid groups in aif2:", get_qualified_groups(aif2))
```

```
aif2 shape: (15639, 30)
Label balance in aif2:
0.0    12760
1.0     2879
Name: label, dtype: int64
 One of the label classes (0 or 1) is missing in this dataset.
Valid groups in aif2: []
```

[83]:
```
def label_distribution_by_group(aif_data):
    df = pd.DataFrame({
        'protected': aif_data.protected_attributes.ravel(),
        'label': aif_data.labels.ravel()
    })
    group_counts = df.groupby(['protected', 'label']).size().
 ↪unstack(fill_value=0)
    group_counts.columns = ['Low Risk (0)', 'High Risk (1)']
    return group_counts

label_distribution_by_group(aif2)
```

[83]:

protected	Low Risk (0)	High Risk (1)
0.0	2	4
1.0	5244	1855
2.0	6	0
3.0	66	6
4.0	5131	776
5.0	1665	191
6.0	45	18
7.0	10	2
8.0	591	27

[86]:
```
# Step 1: Copy df2 and create the label column
df2_labeled = df2.copy()
df2_labeled['label'] = (df2_labeled['DecileScore'] >= 7).astype(int)

# Step 2: Filter to qualified protected groups (based on Ethnic_Code_Text)
qualified_groups = [1.0, 4.0, 5.0]
df2_filtered = df2_labeled[df2_labeled['Ethnic_Code_Text'].
 ↪isin(qualified_groups)]

# Step 3: Reconvert to AIF360 format
aif2_filtered = to_aif360_df2_fixed(df2_filtered)

# Step 4: Confirm label balance and valid groups
```

```
print("  Filtered label balance:")
print(check_label_balance(aif2_filtered))

print("\n  Valid groups:")
print(get_qualified_groups(aif2_filtered))
```

```
 Filtered label balance:
Series([], Name: label, dtype: int64)

 Valid groups:
 One of the label classes (0 or 1) is missing in this dataset.
[]
```

[88]:
```
# Step 1: Create label
df2_labeled = df2.copy()
df2_labeled['label'] = (df2_labeled['DecileScore'] >= 7).astype(int)

# Step 2: Drop rows missing either label or ethnic info
df2_clean = df2_labeled.dropna(subset=['label', 'Ethnic_Code_Text'])

# Step 3: Group counts by Ethnic_Code_Text and label
group_counts = df2_clean.groupby(['Ethnic_Code_Text', 'label']).size().
 ↪unstack(fill_value=0)

# Step 4: Filter for groups that have BOTH 0 and 1 with at least 50 records each
qualified_ethnic_groups = group_counts[
    (group_counts[0] >= 50) & (group_counts[1] >= 50)
].index.tolist()

print("  Qualified ethnic groups:", qualified_ethnic_groups)

# Step 5: Filter DataFrame to those groups
df2_filtered = df2_clean[df2_clean['Ethnic_Code_Text'].
 ↪isin(qualified_ethnic_groups)]

# Step 6: Convert to AIF360
aif2_filtered = to_aif360_df2_fixed(df2_filtered)

# Step 7: Check label balance & valid groups
print("\n  Filtered label balance:")
print(check_label_balance(aif2_filtered))

print("\n  Valid groups:")
print(get_qualified_groups(aif2_filtered))
```

```
 Qualified ethnic groups: ['African-American', 'Caucasian', 'Hispanic']
```

```
 Filtered label balance:
0.0    12040
1.0     2822
Name: label, dtype: int64

 Valid groups:
 One of the label classes (0 or 1) is missing in this dataset.
[]
```

[89]:
```python
from sklearn.preprocessing import LabelEncoder
from aif360.datasets import BinaryLabelDataset

def to_aif360_df2_fixed(df):
    df = df.copy()

    # Create label
    df['label'] = (df['DecileScore'] >= 7).astype(int)

    # Drop rows with missing label or protected attribute
    df = df.dropna(subset=['label', 'Ethnic_Code_Text'])

    # Encode protected attribute (Ethnic_Code_Text)
    le = LabelEncoder()
    df['Ethnic_Code_Text'] = le.fit_transform(df['Ethnic_Code_Text'])

    # Optionally encode other object columns (except label/protected)
    for col in df.columns:
        if df[col].dtype == 'object' and col not in ['label',␣
↪'Ethnic_Code_Text']:
            df[col] = LabelEncoder().fit_transform(df[col])

    return BinaryLabelDataset(
        df=df,
        label_names=['label'],
        protected_attribute_names=['Ethnic_Code_Text'],
        favorable_label=0,
        unfavorable_label=1
    )
```

[90]:
```python
aif2_filtered = to_aif360_df2_fixed(df2_filtered)

# Recheck balance and valid groups
print("\n Filtered label balance:")
print(check_label_balance(aif2_filtered))

print("\n Valid groups:")
print(get_qualified_groups(aif2_filtered))
```

```
 Filtered label balance:
0.0    12040
1.0     2822
Name: label, dtype: int64

 Valid groups:
 One of the label classes (0 or 1) is missing in this dataset.
[]
```

[104]:
```
from aif360.metrics import ClassificationMetric, BinaryLabelDatasetMetric

def compute_fairness_metrics(aif_data, privileged_val, unprivileged_val):
    priv_group = [{aif_data.protected_attribute_names[0]: privileged_val}]
    unpriv_group = [{aif_data.protected_attribute_names[0]: unprivileged_val}]

    dataset_metric = BinaryLabelDatasetMetric(
        aif_data,
        privileged_groups=priv_group,
        unprivileged_groups=unpriv_group
    )

    classified_metric = ClassificationMetric(
        aif_data,
        aif_data,
        privileged_groups=priv_group,
        unprivileged_groups=unpriv_group
    )

    return {
        "Statistical Parity Diff": dataset_metric.
 ↪statistical_parity_difference(),
        "Disparate Impact": dataset_metric.disparate_impact(),
        "Equal Opportunity Diff": classified_metric.
 ↪equal_opportunity_difference(),
        "Average Odds Diff": classified_metric.average_odds_difference(),
        "FPR Difference": classified_metric.false_positive_rate_difference(),
        "FNR Difference": classified_metric.false_negative_rate_difference(),
        "Accuracy": classified_metric.accuracy(),
        "TPR (Priv)": classified_metric.true_positive_rate(privileged=True),
        "TPR (Unpriv)": classified_metric.true_positive_rate(privileged=False)
    }
```

[105]:
```
print(df2_filtered['Ethnic_Code_Text'].unique())
```

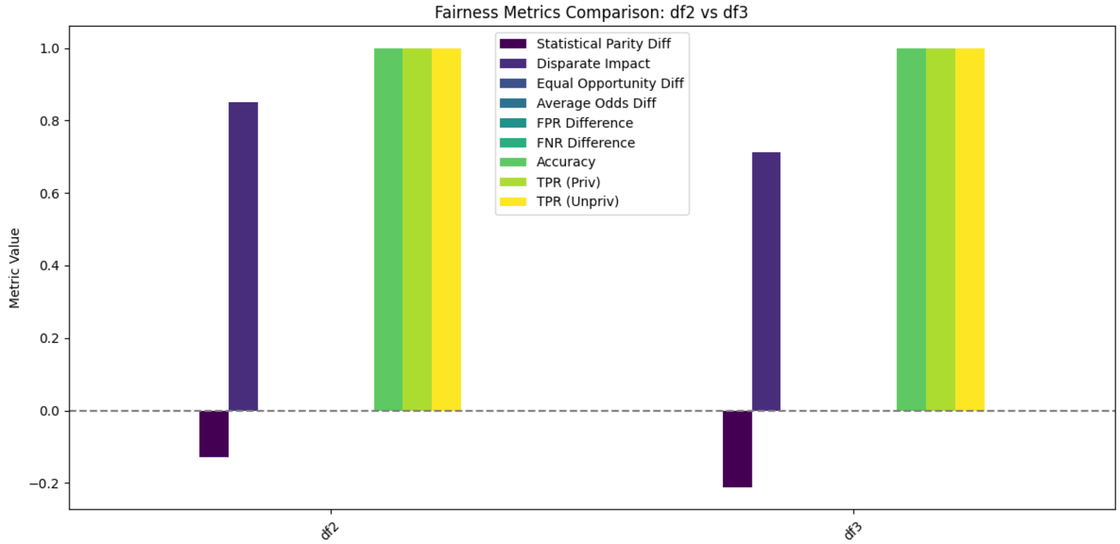
```
[]
```

[106]:
```
metrics_df2 = compute_fairness_metrics(aif2_filtered, privileged_val=1,␣
    ↪unprivileged_val=0)
metrics_df3 = compute_fairness_metrics(aif3, privileged_val=1,␣
    ↪unprivileged_val=0)
```

[107]:
```
import pandas as pd

comparison_df = pd.DataFrame({
    'df2': metrics_df2,
    'df3': metrics_df3
})

print(comparison_df)
```

```
                                 df2       df3
Statistical Parity Diff -0.129935 -0.212379
Disparate Impact         0.850414  0.713750
Equal Opportunity Diff   0.000000  0.000000
Average Odds Diff        0.000000  0.000000
FPR Difference           0.000000  0.000000
FNR Difference           0.000000  0.000000
Accuracy                 1.000000  1.000000
TPR (Priv)               1.000000  1.000000
TPR (Unpriv)             1.000000  1.000000
```

[108]:
```
import matplotlib.pyplot as plt

comparison_df.T.plot(kind='bar', figsize=(12, 6), colormap='viridis')
plt.title("Fairness Metrics Comparison: df2 vs df3")
plt.ylabel("Metric Value")
plt.xticks(rotation=45)
plt.tight_layout()
plt.axhline(0, color='gray', linestyle='--')
plt.show()
```


Fairness Metrics Comparison: df2 vs df3
Statistical Parity Diff
Disparate Impact
Equal Opportunity Diff
Average Odds Diff
FPR Difference
FNR Difference
Accuracy
TPR (Priv)
TPR (Unpriv)
Metric Value
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
df2
df3

```
[ ]:
```