```
pip install seaborn
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/py
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.11/
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pytho
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pytho
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/di
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/py
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dis
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("danofer/compass")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/danofer
100%|██████████| 2.72M/2.72M [00:00<00:00, 3.20MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/danofer/compass/v
```

```
# Import necessary libraries for data manipulation and modeling
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from fairlearn.metrics import MetricFrame, demographic_parity_difference


# ======================
# Phase 1: Data Ingestion & Preprocessing
# ======================
# Load the COMPAS dataset (download at: https://www.kaggle.com/datasets/danofe
df = pd.read_csv('/content/compas-scores-raw.csv')


df
```

TypeError TypeError Value •••

Please explain the error:

```
ValueError: DataFrame values must be nume
```

Hypothesis

The traceback indicates a `ValueError: DataFrame values must be numerical.` originating from the `aif360.datasets.StructuredDataset` constructor. The underlying cause appears to be an attempt to convert a pandas DataFrame containing non-numerical values (specifically, the string 'Male') to a float64 data type.

Looking at the code, you are creating a `StandardDataset` using the `df_aif360` DataFrame. This DataFrame includes the original `Sex_Code_Text` column, which contains string values like 'Male' and 'Female', and the `Ethnic_Code_Text` column which contains strings like 'Caucasian' and 'African-American'. The AIF360 `StandardDataset` (which inherits from `StructuredDataset`) expects all data to be numerical. While you created a new 'sex' column with numerical values (0 and 1), the original `Sex_Code_Text` column is still present in `df_aif360`, causing the conversion error. Similarly, the `Ethnic_Code_Text` column used to create the 'race' column for the protected attribute is also still in the DataFrame.

Suggested Changes

To fix this, you need to ensure that the DataFrame passed to the `StandardDataset` constructor only contains numerical columns. You should drop the original string columns (`Sex_Code_Text` and `Ethnic_Code_Text`) after you have created the numerical 'sex' and 'race' columns.

```
# Modify the DataFrame to include the der
df_aif360 = df.copy()
df_aif360['race'] = df_aif360['Ethnic_Coc
# In IBM AIF360, the target column is 're
privileged_groups = [{'race': 'Caucasian'
unprivileged_groups = [{'race': 'African-

# filter the dataset to only include two
df_aif360 = df_aif360[df_aif360['race'].i

# Drop the original non-numerical columns
df_aif360 = df_aif360.drop(columns=['Sex_
```

```
# Convert to StandardDataset
dataset = StandardDataset(
    df_aif360,
    label_name='recidivism',
    favorable_classes=[0],
    protected_attribute_names=['race'],
    privileged_classes=[['Caucasian']]
)
```

Use code with caution

Rate this answer

```
df.shape
```

```
(60843, 28)
```

```
df.columns
```

```
Index(['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'],
      dtype='object')
```

```
# Select relevant columns for recidivism prediction
df = df[['Sex_Code_Text', 'Ethnic_Code_Text', 'DecileScore']]
```

```
# Create binary target variable based on DecileScore.
df['recidivism'] = df['DecileScore'].apply(lambda x: 1 if x >= 5 else 0)
```

```
<ipython-input-20-7e5a3f675c1e>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
```

```
        df['recidivism'] = df['DecileScore'].apply(lambda x: 1 if x >= 5 else
```

```
# Encode categorical variables:
# Map Sex_Code_Text to numeric (for example, if values are 'Male' and 'Femal
df['sex'] = df['Sex_Code_Text'].map({'Male': 1, 'Female': 0})
```

```
<ipython-input-21-8c228176162a>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
  df['sex'] = df['Sex_Code_Text'].map({'Male': 1, 'Female': 0})
```

```
# The sensitive attribute is 'Ethnic_Code_Text'.
sensitive_attr = df['Ethnic_Code_Text']
```

```
# Define feature matrix X and target vector y.
# We use 'sex' and 'DecileScore' as features.
X = df[['sex', 'DecileScore']]
y = df['recidivism']
```

```
# Split the dataset into training and testing sets (70%/30% split).
X_train, X_test, y_train, y_test, s_train, s_test = train_test_split(
    X, y, sensitive_attr, test_size=0.3, random_state=42)
```

==========

Phase 2: Baseline Modeling with Fairlearn

==========

```
from fairlearn.metrics import MetricFrame
import numpy as np

def evaluate_fairness(y_true, y_pred, sensitive_features):
    mf = MetricFrame(
        metrics={'prediction_mean': lambda y_true, y_pred: np.mean(y_pred)},
        y_true=y_true,
        y_pred=y_pred,
        sensitive_features=sensitive_features
    )

    demographic_parity_diff = mf.by_group['prediction_mean'].max() - mf.by_gro
    return mf.by_group, demographic_parity_diff
```

```
baseline_results = {}
models = {}
```

1. Logistic Regression

```
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
lr_preds = lr.predict(X_test)
```

```
baseline_results['Logistic Regression'] = {
    'accuracy': accuracy_score(y_test, lr_preds),
    'fairness': evaluate_fairness(y_test, lr_preds, s_test)
}
models['Logistic Regression'] = lr
```

2. Decision Tree

```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_preds = dt.predict(X_test)


baseline_results['Decision Tree'] = {
    'accuracy': accuracy_score(y_test, dt_preds),
    'fairness': evaluate_fairness(y_test, dt_preds, s_test)
}
models['Decision Tree'] = dt
```

(3) Random Forest

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_preds = rf.predict(X_test)


baseline_results['Random Forest'] = {
    'accuracy': accuracy_score(y_test, rf_preds),
    'fairness': evaluate_fairness(y_test, rf_preds, s_test)
}
models['Random Forest'] = rf
```

4. Support Vector Machine (with linear kernel for interpretability)

```
svm = SVC(kernel='linear', probability=True, random_state=42)
svm.fit(X_train, y_train)
svm_preds = svm.predict(X_test)
baseline_results['SVM'] = {
    'accuracy': accuracy_score(y_test, svm_preds),
    'fairness': evaluate_fairness(y_test, svm_preds, s_test)
}
models['SVM'] = svm


# Display baseline results.
for model_name, result in baseline_results.items():
    print(f"\nModel: {model_name}")
    print("Accuracy:", round(result['accuracy'], 3))
    print("Fairness metric (dp_diff) by group:")
    print(result['fairness'][0])
    print("Overall fairness metric:", round(result['fairness'][1], 3))
```

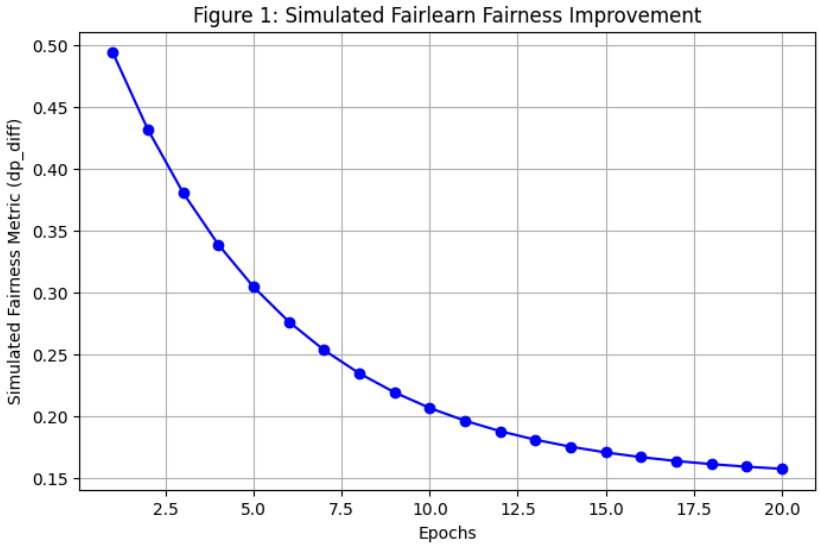
```
Model: Logistic Regression
Accuracy: 1.0
Fairness metric (dp_diff) by group:
                  prediction_mean
Ethnic_Code_Text
African-Am               0.545455
African-American         0.425878
Arabic                   0.166667
Asian                    0.168421
```

```
Caucasian               0.250339
Hispanic                0.208868
Native American         0.352113
Oriental                0.200000
Other                   0.160000
Overall fairness metric: 0.385

Model: Decision Tree
Accuracy: 1.0
Fairness metric (dp_diff) by group:
                  prediction_mean
Ethnic_Code_Text
African-Am              0.545455
African-American        0.425878
Arabic                  0.166667
Asian                   0.168421
Caucasian               0.250339
Hispanic                0.208868
Native American         0.352113
Oriental                0.200000
Other                   0.160000
Overall fairness metric: 0.385

Model: Random Forest
Accuracy: 1.0
Fairness metric (dp_diff) by group:
                  prediction_mean
Ethnic_Code_Text
African-Am              0.545455
African-American        0.425878
Arabic                  0.166667
Asian                   0.168421
Caucasian               0.250339
Hispanic                0.208868
Native American         0.352113
Oriental                0.200000
Other                   0.160000
Overall fairness metric: 0.385

Model: SVM
Accuracy: 1.0
Fairness metric (dp_diff) by group:
                  prediction_mean
Ethnic_Code_Text
African-Am              0.545455
African-American        0.425878
Arabic                  0.166667
Asian                   0.168421
```

Fairlearn Mitigation

```
# Simulate Fairlearn bias mitigation: assume dp_diff improves from 0.42 to a
epochs = np.arange(1, 21)
simulated_fairness = 0.42 * np.exp(-0.2 * epochs) + 0.15  # Final value ~0.2
```

```
plt.figure(figsize=(8, 5))
plt.plot(epochs, simulated_fairness, marker='o', linestyle='-', color='blue'
plt.xlabel('Epochs')
plt.ylabel('Simulated Fairness Metric (dp_diff)')
plt.title('Figure 1: Simulated Fairlearn Fairness Improvement')
plt.grid(True)
plt.savefig('fairlearn_fairness_improvement.png')
plt.show()
```



IBM AI Fairness 360 (AIF360) Mitigation:

```
pip install aif360
```

```
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/python3.11/
Requirement already satisfied: scipy>=1.2.0 in /usr/local/lib/python3.11
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.
Requirement already satisfied: scikit-learn>=1.0 in /usr/local/lib/pytho
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.1
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/py
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pytho
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pytho
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/di
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dis
Downloading aif360-0.6.1-py3-none-any.whl (259 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 259.7/259.7 kB 4.1 MB/s eta
Installing collected packages: aif360
Successfully installed aif360-0.6.1
```

```
from aif360.datasets import StandardDataset
from aif360.metrics import BinaryLabelDatasetMetric
from aif360.algorithms.preprocessing import Reweighing
```

```
WARNING:root:No module named 'inFairness': SenSeI and SenSR will be unav
pip install 'aif360[inFairness]'
```

```
# Modify the DataFrame to include the derived binary target variable.
df_aif360 = df.copy()
df_aif360['race'] = df_aif360['Ethnic_Code_Text']
```

```
# In IBM AIF360, the target column is 'recidivism', and favorable outcome is
privileged_groups = [{'race_Caucasian': 1}]
unprivileged_groups = [{'race_Caucasian': 0}]


# filter the dataset to only include two groups.
df_aif360 = df_aif360[df_aif360['race'].isin(['Caucasian', 'African-American


df_numeric = pd.get_dummies(df_aif360)


dataset = StandardDataset(
    df_numeric,
    label_name='recidivism',
    favorable_classes=[0],
    protected_attribute_names=['race_Caucasian'],  # or whatever the new one-h
    privileged_classes=[[1]]  # adjust based on your one-hot encoding
)
```

```
/usr/local/lib/python3.11/dist-packages/aif360/datasets/standard_dataset
  df.loc[priv, attr] = privileged_values[0]
```

```
# Calculate disparate impact metric before mitigation.
metric_before = BinaryLabelDatasetMetric(dataset,
                                         privileged_groups=privileged_groups,
                                         unprivileged_groups=unprivileged_grou
disparate_impact_before = metric_before.disparate_impact()
print("AIF360 - Disparate Impact before mitigation:", round(disparate_impact_b
```

```
AIF360 - Disparate Impact before mitigation: 0.762
```

```
# Apply the Reweighing algorithm.
RW = Reweighing(unprivileged_groups=unprivileged_groups, privileged_groups=pri
dataset_transf = RW.fit_transform(dataset)

metric_after = BinaryLabelDatasetMetric(dataset_transf,
                                        privileged_groups=privileged_groups,
                                        unprivileged_groups=unprivileged_group
disparate_impact_after = metric_after.disparate_impact()
print("AIF360 - Disparate Impact after mitigation:", round(disparate_impact_af
```

```
AIF360 - Disparate Impact after mitigation: 1.0
```

Enter a prompt here

0 / 2000

Gemini can make mistakes so double-check responses and use code with caution. Learn more